

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

А.Є.Коваленко

Комп'ютерна схемотехніка і архітектура комп'ютерів.

Підготовка та оформлення курсових робіт

**Затверджено Вченою радою НТУУ «КПІ» як навчально-методичний
посібник для студентів, які навчаються за спеціальністю
«Комп'ютерні науки та інформаційні технології»**

Київ
НТУУ «КПІ»
2016

УДК 681.3.06

Гриф надано Вченою радою НТУУ «КПІ»
(протокол № 7 від 06.06.2016 р.)

Рецензенти: О.Ф.Волошин, д-р техн. наук, проф.,
Київський національний університет ім. Тараса Шевченка
Аль-Аммори Алі, д-р техн. наук, проф.,
Національний транспортний університет

Відповідальний
редактор

В.Д.Романенко, д-р.техн.наук, проф.
Національний технічний університет України
«Київський політехнічний інститут»

Коваленко А.Є. Комп'ютерна схемотехніка і архітектура комп'ютерів. Підготовка та оформлення курсових робіт : навч.-метод. посібник для студентів, які навчаються за спеціальністю «Комп'ютерні науки та інформаційні технології» [Електронне видання] / А.Є.Коваленко.- К.: НТУУ «КПІ», 2016.-472 с. Бібліогр. : с.467-468.

Системно викладено основні питання розділів дисципліни «Комп'ютерна схемотехніка і архітектура комп'ютерів», необхідні для підготовки, оформлення і захисту курсової роботи.

Навчально-методичний посібник містять пояснення щодо вимог, структури курсової роботи, основні положення теоретичного курсу, які можуть бути предметом розгляду за темою курсової роботи, теоретичний матеріал та приклади реалізацій по окремих аспектах проектування цифрових пристроїв, можливий порядок самостійної роботи студента над курсовою роботою з курсу «Комп'ютерна схемотехніка і архітектури комп'ютерів», окремі форми і приклади оформлення курсової роботи, приблизний перелік тем курсових робіт, контрольні питання, етапність і критерії оцінювання якості виконання курсової роботи по етапах. Наведено необхідний матеріал існуючих стандартів і нормативних документів з оформлення курсових робіт згідно з існуючими вимогами.

Призначено для студентів, які навчаються за спеціальністю «Комп'ютерні науки та інформаційні технології»

Навчально-методичний посібник може бути корисними для студентів вищих навчальних закладів, викладачів.

УДК 681.3.06
За редакцією автора
©**А.Є.Коваленко, 2016**

ЗМІСТ

ВСТУП	11
1 ПОРЯДОК ВИКОНАННЯ СТУДЕНТОМ КУРСОВОЇ РОБОТИ	12
1.1 Мета виконання курсової роботи	12
1.2 Графік виконання курсової роботи	13
1.3 Перелік тем курсових робіт	13
1.4 Рейтингова система оцінювання успішності навчання. виконання курсової роботи.....	15
1.5 Вимоги до структури курсової роботи	17
2 СТРУКТУРНА ОРГАНІЗАЦІЯ КОМП'ЮТЕРІВ	19
2.1 Структура та функціонування комп'ютера за фон-Нейманом.	19
2.2 Комп'ютери не фон-неймановського типу	21
2.3 Рівні подання комп'ютера.....	24
2.4. Апаратний рівень подання комп'ютера.....	27
2.5 Структура комп'ютера з відкритою архітектурою	29
2.5.1 Склад і призначення основних компонентів	29
2.5.2 Шинна організація комп'ютера	31
2.5.3 Багатошинна організація комп'ютерів	32
3 КОМБІНАЦІЙНІ СХЕМИ	34
3.1 Дешифратори і шифратори.....	34
3.1.1 Дешифратори	34
3.1.2. Шифратори	37
3.2 Мультиплексори і демультиплексори	37
3.2.1. Мультиплексори	37
3.2.2 Демультиплексори.....	42
3.2.3 Каскадування мультиплексорів	44
3.2.5 Мультиплексування шин	46
3.2.6 Застосування демультиплексорів.....	46
3.2.7 Каскадування демультиплексорів.....	49
3.2.8 Демультиплексування шин	50
3.3 Синтез комбінаційних схем	52
3.3.1 Реалізація перемикальних функцій з використанням типових комбінаційних схем	52
3.3.2 Застосування мультиплексорів для побудовування комбінаційних схем.....	53
3.3.3 Реалізація логічних функцій на дешифраторах	54
3.4 Суматори.....	57
3.4.1 Класифікація суматорів	58
3.4.2 Однорозрядні суматори	62

3.4.3 Багаторозрядні суматори	62
3.4.4 Послідовний багаторозрядний суматор	65
3.4.5 Паралельні багаторозрядні суматори	66
3.4.6 Двійково-десяткові суматори.....	67
3.5 Схеми арифметико-логічних пристроїв	71
3.6 Спеціалізовані комбінаційні схеми	73
3.6.1 Схеми порівняння	73
3.6.1.1 Загальна характеристика схем порівняння	73
3.6.1.2 Схеми порівняння слів з константою	74
3.6.1.3 Схеми порівняння багаторозрядних слів.....	76
3.6.1.4 Групові схеми порівняння багаторозрядних слів.....	77
3.6.1.5 Схеми порівняння двох слів «на більше».....	79
3.6.1.6 Групові багаторозрядні схеми порівняння «на більше»	81
3.6.2 Схеми контролю	82
3.6.2.1 Схеми контролю операцій.....	82
3.6.2.2 Схеми контролю парності	84
4 ПОСЛІДОВНІСНІ СХЕМИ	87
4.1 Тригери	87
4.1.1 Класифікація тригерів	87
4.1.2 Таблиця переходів і логічні рівняння RS-тригера.....	90
4.1.3 Асинхронний RS-тригер на елементах I-HE	92
4.1.5 Синхронні RS-тригери	93
4.1.6 Двоступеневі RS-тригери	96
4.1.7 Тригери типу JK	97
4.1.8 T-тригери.....	99
4.1.9 D-тригери	101
4.1.10 DV-тригер	102
4.1.11 Інші типи тригерів	103
4.2 Регістри	104
4.2.1 Основні операції регістра	104
4.2.2 Склад і типи регістрів	105
4.2.3 Регістри паралельного типу	105
4.2.4 Регістри послідовного типу.....	110
4.2.5 Перетворення послідовного коду у паралельний	111
4.2.6 Генератори псевдовипадкових чисел на регістрах.....	112
4.2.7 Синтез регістрів на асинхронних тригерах	115
4.3 Лічильники	123
4.3.1 Класифікація лічильників.....	123
4.3.2 Структури лічильників.....	123
4.3.3 Типи міжрозрядних переносів в лічильниках	126
4.3.4 Синтез лічильників з паралельним переносом	127
4.3.5 Синтез лічильників з наскрізним переносом	128

4.3.6 Лічильники з груповим переносом	129	6.10 Адресація байтів і слів	183
5 СХЕМИ ПАМ'ЯТІ	130	6.11 Регістри процесора	185
5.1 Принципи функціонування запам'ятовувальних пристроїв	130	7 ПРИНЦИПИ ПРОЕКТУВАННЯ СХЕМ	190
5.1.1 Функція пам'яті	130	7.1 Методи аналізу і синтезу цифрових пристроїв	190
5.1.2 Класифікація пам'яті	130	7.1.1 Аналіз і синтез комбінаційних схем	190
5.1.3 Основні параметри пам'яті	132	7.1.2 Метод Квайна – Мак-Класки мінімізації функції	190
5.2 Мікросхеми пам'яті	134	7.1.3 Алгоритм Квайна – Мак-Класки	191
5.2.1 Вхідні та вихідні сигнали мікросхеми пам'яті	134	7.2 Аналіз і синтез послідовнісних схем	195
5.2.2 Часові характеристики мікросхем пам'яті	135	7.2.1 Аналіз тригерних схем	195
5.2.3 Класифікація і способи доступу до даних у напівпровідниковій пам'яті	136	7.2.2 Аналіз схем із застосуванням скінченних автоматів	198
5.2.4 Загальна характеристика кеш-пам'яті	139	7.3 Класифікація методів і засобів проектування цифрових пристроїв	200
5.2.5 Загальна характеристика постійної пам'яті	141	7.3.1 Класифікація цифрових інтегральних схем	200
5.2.6 Загальна характеристика флеш-пам'яті	141	7.3.2 Класифікація засобів и проектування	205
5.2.7 Характеристика статичних запам'ятовуючих пристроїв	143	8 МОВИ ПРОЕКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ	206
5.2.8 Принцип побудови динамічного запам'ятовуючого елемента	145	8.1 Засоби опису цифрових пристроїв	206
5.2.9 Структури запам'ятовувальних пристроїв	146	8.2 Етапи проектування пристроїв з використанням систем САПР	209
5.2.9.1 Структура 2D статичних ОЗП	146	8.3 Мова проектування VHDL	211
5.2.9.2 Структура 3D запам'ятовувальних пристроїв	147	8.3.1 Загальні поняття	211
5.2.9.3 Структура 2DM статичних ЗП	149	8.3.2 Опис проекту мовою VHDL	213
5.2.9.4 Пам'ять з послідовним доступом	151	8.3.3 Приклади реалізації цифрових пристроїв мовою VHDL	214
6 МІКРОПРОЦЕСОРИ	156	8.3.4 Опис типових вузлів	215
6.1 Класифікація мікропроцесорів	156	8.3.5 Проектування мовою VHDL з використанням бібліотек	217
6.1.1 Загальні поняття	156	9 АРХІТЕКТУРА СИСТЕМИ КОМАНД	219
6.1.2 Види мікропроцесорів	156	9.1 Основні поняття	219
6.2 Будова процесорів комп'ютера	159	9.2 Класифікація архітектур системи команд	219
6.3 Мікропроцесорні засоби	163	9.2.1 Стекові архітектури	220
6.4 Архітектури мікропроцесорів	164	9.2.2 Акумуляторні архітектури	221
6.5 Режими обміну даними у комп'ютері з використанням процесорів	168	9.2.3 Архітектури з індексними регістрами	223
6.5.1 Синхронний і асинхронний обмін	168	9.2.4 Архітектури «регістр-пам'ять» і «регістр-регістр»	223
6.5.2 Обмін в режимі переривання	170	9.2.5 Архітектури «пам'ять-пам'ять»	226
6.5.3 Прямий доступ до пам'яті	171	9.3 Адресність команд	226
6.5.4 Канальний обмін даними	172	9.4 Особливості побудови систем команд	227
6.6 Виконання команд процесором	173	9.4.1 Класифікація команд	227
6.7 Адресація операндів	174	9.4.2 Системи команд CISC	228
6.8 Методи адресації	174	9.4.3 Системи команд RISC	229
6.8.1 Безпосередня адресація	175	9.4.4 Системи команд VLIW	230
6.8.2 Пряма адресація	175	9.4.5 Системи команд SSE	231
6.8.3 Регістрова адресація	176	9.4.5.1 Технологія SSE	231
6.8.4 Автоінкрементна і та автодекрементна адресація	177	9.4.5.2 Набір команд SSE2	232
6.8.5 Індексна адресація	178	9.4.5.3 Набір команд SSE3	232
6.9 Сегментація пам'яті	178	9.4.5.4 Набір команд SSE4	233
		9.4.5.5 Набір команд SSE5	234

9.4.6 Системи команд AVX	234	11.7.2 Операції та засоби реалізації потокових шифрів	305
9.5 Програмні моделі систем команд CPU	238	11.7.3 Реалізація нелінійних функцій	305
9.5.1 Класифікація мікропроцесорів за призначенням	238	11.7.4 Алгоритм RC4	306
9.5.2 Мікроархітектура мікропроцесора.....	240	11.7.5 Алгоритми VEST	310
9.5.3 Параметри процесорів.....	245	11.7.6 Порівняння потокових алгоритмів.....	313
10 МОВИ АСЕМБЛЕРА.....	247	11.7.7 Алгоритм A5.....	313
10.1 Загальні поняття.....	247	12 ОСНОВИ ПОБУДОВИ ВІДМОВОСТІЙКИХ СИСТЕМ.....	316
10.2 Переваги і недоліки мов асемблера.....	249	12.1 Поняття відмовостійкості.....	316
10.3 Директиви асемблера.....	250	12.2 Типи відмов розподілених систем.....	317
10.4 Синтаксис асемблера.....	250	12.3 Способи забезпечення відмовостійкості.....	318
10.5 Типові набори команд асемблера.....	253	12.4 Групові процеси	320
11 ЗАСТОСУВАННЯ СХЕМНО-ПРОГРАМНИХ РІШЕНЬ АРХІТЕКТУР		12.4.1 Відмовостійкість процесів.....	320
КОМП'ЮТЕРІВ.....	256	12.4.2 Надійний зв'язок клієнт-сервер.....	323
11.1 Застосування схемних рішень для криптографічних перетворень.....	256	12.4.3 Надійна групова розсилка.....	326
11.1.1 Переваги апаратного шифрування.....	256	12.4.4 Надійна групова доставка.....	333
11.1.2 Види пристроїв апаратного шифрування	257	12.4.5 Розподілене підтвердження.....	336
11.1.3 Додаткові можливості апаратних шифраторів	258	12.5 Відновлення процесів.....	339
11.1.4 Апаратно-програмні елементи шифрування.....	260	12.6 Моделі системного діагностування	346
11.2 Криптосистеми шифрування.....	262	12.7 Інтегрована модель процесів діагностування.....	350
11.2.1 Характеристика симетричних криптосистем.....	263	12.8 Структурно-апаратні методи забезпечення відмовостійкості.....	353
11.2.2 Симетричні шифри.....	265	12.9 Завадостійкі коди	356
11.2.3 Перетворення у симетричних алгоритмах шифрування	266	12.9.1 Лінійні блокові коди.....	356
11.2.4 Мережі Фейстеля	267	12.9.2 Ітеративний код.....	357
11.3 Алгоритм DES	268	12.9.3 Виявлення помилок на основі кодових синдромів	359
11.3.1 Загальна характеристика	268	12.10 Завадостійке кодування на основі поліноміальних кодів	363
11.3.2 Схема шифрування DES.....	269	12.10.1 Поліноміальне кодування інформації циклічних кодів.....	363
11.3.3 Схема утворення функції f.....	271	12.10.2 Схемна реалізація	365
11.3.4 Генерування ключів	277	13 КОМП'ЮТЕРИ ПАРАЛЕЛЬНОЇ ДІЇ.....	368
11.3.5 Дешифрування тексту в DES.....	279	13.1 Таксономія Флінна.....	368
11.3.6 Режими використання DES	281	13.2 Класифікація апаратних засобів PIC.....	376
11.3.6.1 Режим електронної кодової книги ECB.....	281	13.3 Моделі узгодженості паралельних комп'ютерів	380
11.3.6.2 Режим зчеплення блоків CBC.....	281	13.3.1 Логічні годинники за алгоритмом Лампорта	381
11.3.6.3 Режим зворотного зв'язку за шифротекстом CFB.....	282	13.3.2 Глобальний стан.....	383
11.3.6.4 Режим зворотного зв'язку за виходом OFB	283	13.4 Архітектури паралельних комп'ютерів.....	384
11.4 Алгоритм Triple DES	284	13.4.1 Класифікації паралельних комп'ютерів.....	384
11.5 Алгоритм AES	286	13.4.2 Масово-паралельна архітектура	386
11.6 Алгоритм IDEA	295	13.4.3 Суперкомп'ютери.....	390
11.6.1 Загальна характеристика	295	13.4.4 Суперкомп'ютери Cray.....	392
11.6.2 Схема шифрування	295	13.4.5 Суперкомп'ютер Cray X1	397
11.6.3 Апаратні реалізації алгоритму IDEA.....	302	13.4.6 Архітектура NUMA	401
11.7 Потокові симетричні алгоритми шифрування	303	13.4.7 Архітектура ccNUMA	402
11.7.1 Принципи роботи потокових криптосистем	303		

13.4.7 Архітектури кластерних систем.....	404	16.3.2 Нумерація ілюстрацій.....	439
13.4.7.1 Класифікація кластерних систем.....	404	16.4 Таблиці.....	439
13.4.7.2 Архітектура кластера.....	404	16.4.1 Розміщення та оформлення простих таблиць.....	439
13.4.7.3 Кластери розподілу навантаження.....	407	16.4.2 Розміщення та оформлення таблиць, розділених на частини.....	440
13.4.7.4 Обчислювальні кластери.....	407	16.4.2 Нумерація таблиць.....	440
13.4.7.5 Системи розподілених обчислень grid.....	408	16.5 Посилання.....	440
13.4.7.6 Кластер серверів, організованих програмно.....	409	16.6 Переліки.....	441
13.4.7.7 Особливості реалізації кластерів.....	409	16.7 Примітки.....	441
13.4.7.8 Кластер Beowulf.....	412	16.8 Додатки.....	442
13.5 Системи PVP.....	414	16.9 Посилання на першоджерела.....	443
13.6 Суперкомп'ютер NEC Earth Simulator.....	415	16.9.1 Посилання на загальний перелік джерел.....	443
13.7 Система SP2.....	418	16.9.2 Посилання на джерела у межах сторінки.....	443
13.8 Системи NUMA-Q.....	421	16.10 Виноски.....	444
13.9 Масивно паралельні системи Cray/SGI Origin.....	423	16.11 Запитання і завдання для самоконтролю.....	444
13.10 Архітектура SOMA.....	425	17 ОФОРМЛЕННЯ БІБЛІОГРАФІЧНИХ ОПИСІВ.....	446
14 ОСОБЛИВОСТІ ПІДГОТОВКИ СКЛАДОВИХ ЧАСТИН КУРСОВОЇ РОБОТИ.....	426	17.1 Области опису документів.....	446
14.1 Загальні вимоги до структури і змісту курсової роботи.....	426	17.2 Опис елементів бібліографічного опису.....	447
14.2 Складання плану роботи.....	427	17.2.1 Правила подання елементів бібліографічного опису.....	451
14.3 Підготовка першого розділу основної частини роботи.....	428	17.2.1 Застосування літер і спеціальних знаків.....	451
14.4 Загальні виоги щодо оформлення курсової роботи.....	429	17.2.2 Посилання на авторів.....	451
14.4.1 Нормативна документація.....	429	17.2.3 Електронні видання.....	452
14.4.2 Загальні вимоги до структури і оформлення роботи.....	430	17.2.4 Паралельні назви і відомості до назв.....	453
14.4.3 Структурні елементи вступної частини.....	431	17.2.5 Відомості про відповідальність.....	455
14.4.4 Обсяг курсової роботи.....	431	17.2.6 Область видання.....	456
15 ЗМІСТ СТРУКТУРНИХ КОМПОНЕНТІВ КУРСОВОЇ РОБОТИ.....	432	17.2.7 Опис нормативних документів і стандартів.....	456
15.1 Титульний аркуш.....	432	17.2.8 Відомості про місце видання.....	457
15.2 Реферат.....	432	17.2.9 Відомості про видавця.....	458
15.3 Зміст.....	434	17.2.10 Опис фізичних характеристик і супроводжувачий матеріал.....	460
15.4 Перелік умовних позначень, символів, скорочень і термінів.....	434	17.2.11 Опис стандартних номерів.....	460
15.5 Розділи і підрозділи.....	434	17.3 Приклади повного оформлення бібліографічного опису.....	461
15.6 Пункти і підпункти.....	435	17.4 Запитання і завдання для самоконтролю.....	463
16 ОФОРМЛЕННЯ ГРАФІЧНО-ТЕКСТОВОЇ ІНФОРМАЦІЇ.....	436	18 ПІДГОТОВКА ДО ЗАХИСТУ ТА ЗАХИСТ КУРСОВОЇ РОБОТИ.....	465
16.1 Текст.....	436	18.1 Допуск до захисту.....	465
16.1.1 Розміщення та виправлення тексту.....	436	18.2 Підготовка доповіді для захисту курсової роботи.....	465
16.1.2 Нумерація і розмежування текстів.....	436	ВИКОРИСТАНА ЛІТЕРАТУРА.....	467
16.1.3 Нумерація сторінок.....	437	Додаток А.....	469
16.2 Формули та рівняння.....	437	Додаток Б.....	471
16.2.1 Розміщення формул і рівнянь.....	437	Додаток В.....	472
16.2.2 Нумерація формул та рівнянь.....	438		
16.3 Ілюстрації.....	438		
16.3.1 Розміщення ілюстрацій.....	438		

ВСТУП

Виконання курсової роботи студентом відіграє значну роль в успішному засвоєнні теоретичних знань і в отриманні необхідних вмінь їх застосування з дисципліни «Комп'ютерна схемотехніка і архітектура комп'ютерів».

Підготовка курсової роботи з дисципліни передбачає певний порядок самостійної роботи за загальними вимогами. Тому роботу студентів варто розглядати на різних етапах навчання: планування, виконання окремих розділів, підготовки до захисту і захисту.

Змістовне наповнення цих етапів включає ознайомлення студентів з можливим переліком варіантів тем курсових робіт, фактичних даних із комп'ютерної схемотехніки і архітектури комп'ютера, обмежений теоретичний матеріал для розв'язання задач аналізу та проектування, приклади типових навчальних задач та їх схемних реалізацій.

Використання базового теоретичного матеріалу даного навчального посібника за визначеними розділами призначено для прискорення процесу ознайомлення студентів з основними особливостями виконання курсової роботи за вибраною темою, для планування виконання курсової роботи, попереднього ознайомлення з лекційним матеріалом дисципліни, складання індивідуального графіку виконання курсової роботи.

Основними етапами роботи протягом семестру є:

- підготовка до першої атестації;
- підготовка до другої атестації;
- підготовка до захисту курсової роботи.
- захисту курсової роботи.

1 ПОРЯДОК ВИКОНАННЯ СТУДЕНТОМ КУРСОВОЇ РОБОТИ

1.1 Мета виконання курсової роботи

Метою виконання курсової роботи є формування у студентів здатностей:

- здатності до оцінювання апаратних рішень сучасних архітектур комп'ютерів для вибору раціональних рішень;
- здатності проводити аналіз архітектури команд комп'ютера для визначення ефективності керування процесами;
- здатності самотійно розв'язувати поточні задачі синтезу цифрових пристроїв.

При виконанні курсової роботи студенти мають продемонструвати такі результати навчання:

знання з принципів і структури побудови архітектур комп'ютерів; схемотехнічних рішень компонентів комп'ютерів; з особливостей організації виконання команд центральним процесорним елементом (CPU); з архітектури системи команд (ISA) процесорів; систем автоматизації подання, моделювання і проектування мікросхем;

уміння проводити інформаційний пошук з за вибраною темою курсової роботи; проводити відбір ефективних схемних рішень; застосовувати засоби опису і проектування цифрових пристроїв; виконувати аналіз поточного стану і тенденцій розвитку схемотехнічних рішень і визначати тенденції розвитку архітектур комп'ютерів;

досвід застосування новітніх рішень для розв'язання схемотехнічних задач побудови цифрових пристроїв, аналізу системних рішень архітектур комп'ютерів.

1.2 Графік виконання курсової роботи

Графік виконання курсової роботи поділяють на ряд етапів

Тиждень	Етап	СРС*
2	Отримання теми та завдання	2
3-5	Підбор та вивчення літератури	4
6-7	Підготовка розділу 1	4
8-10	Підготовка розділу 2	4
11-13	Підготовка розділу 3	4
14-15	Оформлення курсової роботи	4
16	Подання курсової роботи на перевірку	4
17	Захист курсової роботи	4

СРС* – самостійна робота студентів

1.3 Перелік тем курсових робіт

Тематика курсових робіт передбачає вибір студентом напряму досліджень і визначення теми досліджень з курсової роботи. Основними варіантами тем є такі.

1. Архітектури сучасних планшетних комп'ютерів.
2. Особливості застосування операційних систем у сучасних мобільних пристроях.
3. Архітектури систем команд багатоядерних CPU.
4. Реалізація систем команд багатоядерних CPU.
5. Системні шини сучасних ноутбуків.
6. Особливості схемотехнічних рішень мобільних комп'ютерів.
7. Архітектури сучасних смартфонів.
8. Сучасні засоби інтегрування мобільних пристроїв

9. Бездротові засоби зв'язку мобільних телефонів і смартфонів
10. Прогнозування розвитку схемо технічної бази мобільних комп'ютерів
11. Схемотехнічні рішення мобільних пристроїв хмарних обчислень
12. Апаратна і програмна база хмарних обчислень
13. Історія і тенденції розвитку систем команд ISA Intel
14. Історія і тенденції розвитку архітектур CPU Intel
15. Історія і тенденції розвитку систем команд ISA AMD
16. Історія і тенденції розвитку архітектур CPU AMD.
17. Прогнозування розвитку ринку схемотехнічних виробів сучасних комп'ютерів.
18. Прогнозування розвитку ринку сучасних мобільних пристроїв сучасних комп'ютерів.
19. Порівняльний аналіз використання операційних систем для архітектур мобільних комп'ютерних пристроїв.
20. Апаратно-програмні засоби вбудованих комп'ютерів.
21. Апаратно-програмна підтримка драйверів сучасних комп'ютерів.
22. Мобільні комп'ютерні системи хмарних обчислень
23. Порівняльний аналіз і тенденції розвитку багатоядерних CPU.
24. Сенсорні системи розподілених систем та їх схемні і програмні рішення.
25. Схемотехнічні рішення систем завадостійкого кодування даних сучасних комп'ютерів..
26. Схемотехніка генераторів псевдовипадкових чисел сучасних комп'ютерів.
27. Сучасні архітектури системних шин комп'ютерів.

28. Системи проектування на основі FPGA.

Примітка. Тема може бути запропонована студентом самостійно у межах основних розділів курсу «Комп'ютерна схемотехніка та архітектури комп'ютерів»».

За наявності власних пропозицій студента або нових даних у питаннях розробки і впровадження нових схемотехнічних рішень протягом перших 2-3 тижнів виконання курсової роботи можливою є модифікація теми.

1.4 Рейтингова система оцінювання успішності навчання. виконання курсової роботи

Рейтинг студента з кредитного модуля складається з балів, які він отримує за (додаток А):

а) розробки і виконання підготовки і оформлення курсової роботи протягом семестру. За виконання цієї роботи застосовують стартову складову (г 1);

б) захист курсової роботи. Він складає складову захисту курсової роботи (г 2):

Сума цих двох складових утворює сумарну рейтингову оцінку за 100 – бальною шкалою.

Стартова складова (г 1) складається з таких елементів:

своєчасне виконання графіку виконання курсової роботи – 5-3 балів;

сучасність та обґрунтування прийнятих рішень – 12-7 балів;

правильність застосування методів аналізу і розрахунку – 10-6 балів;

якість оформлення, виконання вимог нормативних документів – 6-4 балів;

якість графічного матеріалу і дотримання вимог ДСТУ – 7-4 балів;
Складова захисту курсової роботи (г 2) складається з таких елементів:

ступінь володіння матеріалом – 10-6 балів;

повнота аналізу можливих варіантів – 15-9 балів;

ступінь обґрунтування прийнятих рішень – 20-12 балів;

вміння захищати свою думку – 15-9 балів;

Перша (стартова) характеризує роботу студента над курсовою роботою та якість її оформлення. Друга складова характеризує якість захисту курсової роботи. Розмір першої складової дорівнює 40 балів, а розмір другої складової – 60 балів.

До загального рейтингу можуть додаватись бали, отримані за необов'язкові складові. До необов'язкових складових віднесено:

участь у модернізації робіт з дисципліни;

доповіді на наукових студентських семінарах, конференціях, якщо робота мала відношення до курсової роботи.

За їх виконання студент може отримати до 10 заохочувальних балів (у межах максимального числа 10 заохочувальних балів на повний рейтинг 100 балів).

За отриманими рейтинговими балами визначають оцінку за курсову роботу згідно із таблицею

Бали $R = r_1 + r_2$	ECTS оцінка	Оцінка за курсову роботу
95-100	A	відмінно добре
85-94	B	
75-84	C	задовільно
65-74	D	
60-64	E	
Менше 60	Fх	Не зараховано
Курсова робота не допущена до захисту	F	не допущено

1.5 Вимоги до структури курсової роботи

Порядок розміщення матеріалу курсової роботи:

Титульна сторінка курсової роботи (додаток Б

Завдання на виконання курсової роботи (додаток В)

Реферат українською мовою

Реферат англійською мовою

Зміст

Перелік скорочень, позначень

Вступ

Постановка задачі

Розділ 1 Огляд першоджерел (літератури) сучасного рівня з предмету досліджень (не менше 75% за останні 5 років). Висновки

Розділ 2 Аналіз схемо технічних і програмних рішень. Висновки

Розділ 3 Аналіз тенденцій розвитку, побудова моделей. Висновки

Розділ 4 Експериментальне дослідження, програмування. Висновки

Висновки до роботи

Література (приблизно 10-15 назв джерел).

Додаток А Слайди ілюстративного матеріалу до курсової роботи (в межах 5-15 слайдів).

Додаток Б Програми і/або командні файли з коментарями, алгоритми за темою досліджень тощо

Додатки В, Г,

Змістовне наповнення окремих частин роботи залежить від характеру курсової роботи. Загальний обсяг роботи зазвичай повинен бути у межах 30-40 сторінок.

Курсова робота повинна бути оформлена згідно з діючими стандартами і нормативними документами [6-7]. У першу чергу це стосується текстової частини та оформлення бібліографічних посилань.

Додаткову інформацію щодо виконання курсової роботи можна отримати із джерел [1-15], а також інших джерел, зокрема з Інтернет.

За індивідуальними завданнями студент повинен уміти застосовувати програмні засоби проектування і моделювання цифрових пристроїв, операційні системи, зокрема, із залученням програмування в оболонках операційних систем і використанням стандартних пакетів.

Організація курсової роботи студентів спирається на плануванні виконання необхідних дій підготовки до практичних занять, контрольних робіт протягом семестру і підготовки до захисту курсової роботи.

Значне місце у курсовій роботі займають питання практичного засвоєння матеріалу дисципліни і подальшого використання отриманих знань для розв'язання задач за темою роботи.

2 СТРУКТУРНА ОРГАНІЗАЦІЯ КОМП'ЮТЕРІВ

2.1 Структура та функціонування комп'ютера за фон-Нейманом.

2.1.1 Архітектура і принципи побудови машин фон-Неймана.

Переважаюча частина сучасних обчислювальних машин, зокрема персональних комп'ютерів (ПК), ноутбуків, є комп'ютерами фон-неймановського (принстонського) типу, які базуються на таких принципах.

Основними блоками машини є блок керування (control unit), арифметико-логічний пристрій (АЛП) (ALU - arithmetic and logic unit), запам'ятовувальний пристрій (ЗП) (memory) і пристрій введення-виведення (ПВВ). Пристрій ЗП зазвичай створюють на основі оперативної основної пам'яті (main memory) і кеш-пам'яті.

Програми і дані зберігаються в одній і тій же пам'яті; концепція програми, що зберігається (stored program) є основною.

Пристрій керування і арифметичний пристрій зазвичай об'єднуються у центральний процесор CPU (central processing unit), який може включати внутрішню пам'ять у вигляді кеш-пам'яті і різноманітних регістрів. Центральний процесор CPU визначає виконання інструкцій (команд, дій, елементарних операцій) шляхом зчитування команд з оперативної пам'яті. Сучасні CPU за архітектурою дозволяють працювати з пристроями ПВВ і забезпечувати їх прямий доступ до пам'яті.

Програма для фон-неймановської машини складається з набору команд, які перевіряються одна за іншою. Адреса чергової комірки пам'яті, з якої повинна вибиратись наступна вказується лічильником команд, який знаходиться у пристрої керування.

Сучасні комп'ютери мають можливість одночасного, зокрема конвейсного, виконання кількох команд.

Дані, з якими працює програма, можуть включати в себе змінні. Змінні можуть бути поіменовані. До значень цих змінних можна звертатись у процесі виконання програми або міняти ці значення під час виконання програми з використанням наданих імен.

Узагальнена будова комп'ютера за фон-Нейманом показано на рис. 1.1. Класичні багатопроцесорні і багатомашинні архітектури є фон-неймановськими машинами (системами), оскільки в цих архітектурах застосовуються різноманітні методи ефективної взаємодії фон-неймановських машин. Переважаюча кількість персональних комп'ютерів також побудовані за фон-неймановською архітектурою.

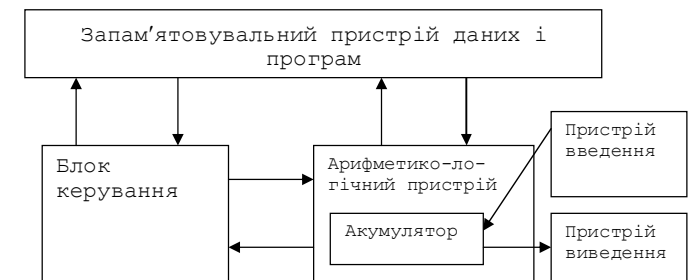


Рисунок 2.1 - Структура комп'ютера фон-Неймана

2.2 Комп'ютери не фон-неймановського типу

У машинах не фон-неймановської архітектури модель обчислень радикально відрізняється від класичної фон-неймановської машини. Основними такими відмінностями можуть бути:

- відсутність послідовної передачі керування, зокрема відсутність лічильника команд.
- відсутня концепція змінної, тобто без використання поіменованих областей (комірок) пам'яті.

Прикладом таких машин є потокова машина (dataflow machine), у яких є потік значень даних від операцій, що генерують ці значення, до операцій, що використовують («споживають») ці значення у якості операндів. Останні операції починають виконуватись (ініціюються) безпосередньо за наявності (отриманні) всіх операндів, що забезпечує високий рівень паралелізму.

Іншим прикладом не фон-неймановської архітектури є Гарвардська архітектура (рис. 2.2).

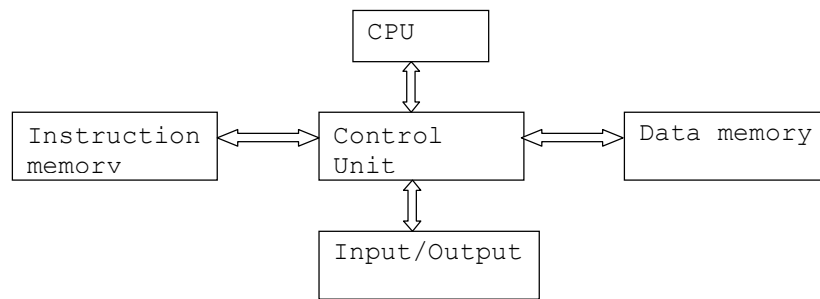


Рисунок 2.2 – Узагальнена Гарвардська архітектура

Особливістю є роздільне розміщення даних і програм на відокремлених пристроях пам'яті – даних і програм (Data memory,

Instruction memory). Інструкцію процесор CPU отримує з Instruction memory, а дані (зокрема операнди) – з Data memory. Процесор CPU може отримати доступ до пристроїв введення-виведення (Input/Output).

Пам'яті Data memory, Instruction memory мають свої власні характеристики системи адресації. Наприклад, Instruction memory може мати лише режим доступу Read, а Data memory режими Read/Write. Їх можна виконувати як постійну пам'ять (Instruction memory) або оперативну (Data memory). Системи адресації відокремлені, а тому вони можуть мати різну розрядність шин адрес і даних, тактову частоту, часову діаграму доступу тощо, залежно від потреб і елементної бази.

За Гарвардською архітектурою будують як комп'ютери, так і окремі архітектури процесорів CPU, мікроконтролерів (наприклад, CPI), процесорів сигналів (зокрема, TMS320).

Один з варіантів її реалізації є структура комп'ютера на рис. 2.3. В комп'ютері виконується відокремлений (зокрема, за адресним простором) доступ до даних і програм. Основна перевага такої архітектури – спрощена структура комп'ютера, оскільки здійснюється доступ лише до однієї загальної пам'яті і виключається можливість перерозподілу пам'яті між програмою і даними. Розміщення стеку спрощує доступ до пам'яті.

Додатковими перевагами є можливість паралельного виконання команд за рахунок суміщення виклику і дешифрації команд, виконання виклику даних для процесора, зокрема, з використанням регістрової пам'яті введення і виведення даних.

Стек містить послідовність команд програми. Дешифратор команд дозволяє визначити послідовності інструкцій, які виконує процесор за командами програми.

Розділеність адресних просторів команд і даних дозволяє захищати програми, розміщуючи їх у постійній пам'яті. При цьому на апаратному

рівні шини адресації даних і програм можуть мати різні параметри (ширину, розрядність, швидкодію тощо).

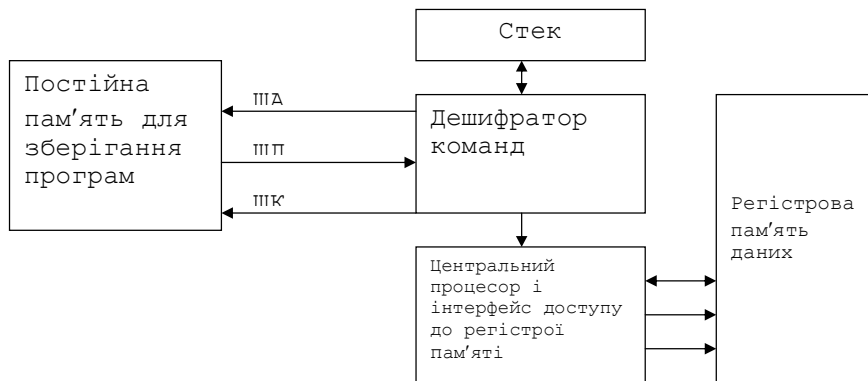


Рисунок 2.3 - Організація комп'ютера з Гарвардською архітектурою

Найбільшого поширення пристрої з Гарвардською архітектурою набули як спеціалізовані обчислювачі чи пристрої обробки даних.

Недослідком гарвардської архітектури у порівнянні з фон-неймановською є складність реалізації.

У модифікованих Гарвардських архітектурах (Modified Harvard Architecture) припускається можливість доступу до Instruction memory як до пам'яті даних. У більш складних застосуваннях ієрархії пам'яті процесор CPU може мати відокремлені кеш-пам'ять для даних і кеш-пам'ять для інструкцій. При цьому можуть виникати проблеми когентності кешів, тобто цілісності і сумісності даних, які вони зберігають.

2.3 Рівні подання комп'ютера

Подання комп'ютера визначається метою відображення характерних особливостей комп'ютера з певних сторін. Наслідком є цілий ряд абстракцій, прийнятих до такого подання. Сучасним підходом до подання структурної організації комп'ютера є багаторівнева комп'ютерна організація, яка передбачає розгляд комп'ютера як сукупність віртуальних машин з різними системами команд.

Для багатоядерних процесорів CPU для комп'ютерів з одним процесором таке подання показано на рис. 2.4. Нижнім рівнем подання є віртуальна машина M0i на основі команд одного з ядер і CPU. Система команд визначається машинною мовою L0i, яка закладається при створенні CPU на апаратному рівні.

Наступні рівні віртуальних машин визначаються застосуванням додаткових засобів трансляції і інтерпретації системи команд, більш зручної користувачу. Така реалізація віртуальних машин може мати апаратну підтримку.

Первага такого підходу полягає у можливості використання користувачем лише віртуальної машини певного рівня. Для створення більш ефективних програм виникає потреба у використанні кількох віртуальних машин, починаючи з певного рівня, до нижнього.

Переважаюча кількість сучасних комп'ютерів мають два і більше віртуальних рівнів. На рис. 2.5 показано рівневу організацію комп'ютера з шістьма рівнями.

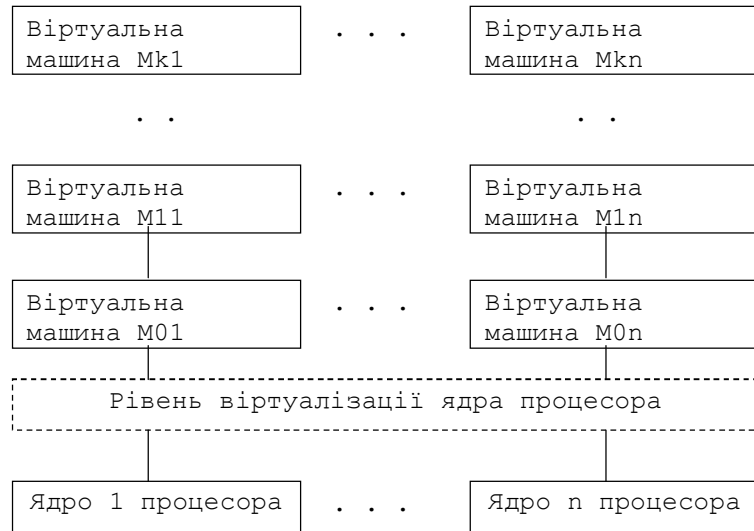


Рисунок 2.4 - Багаторівнева багатоядерна організація комп'ютера

Цифровий логічний рівень підтримується апаратним забезпеченням з використанням фізичних елементів, які виконують логічні булеві операції (AND, OR, NOT, XOR) як логічні перемикальні елементи (логічні вентиля – gates). Вентилі зазвичай реалізують на основі транзисторів. Сукупність вентилів дозволяє реалізувати довільні булеві (перемикальні) функції. Система команд цифрового логічного рівня визначається на мікроархітектурному рівні.

На мікроархітектурному рівні використовують регістри локальної пам'яті певної розрядності (наприклад, 8, 16, 32, 64) і арифметико-логічний пристрій ALU (arithmetical logical unit), який виконує прості арифметичні операції.

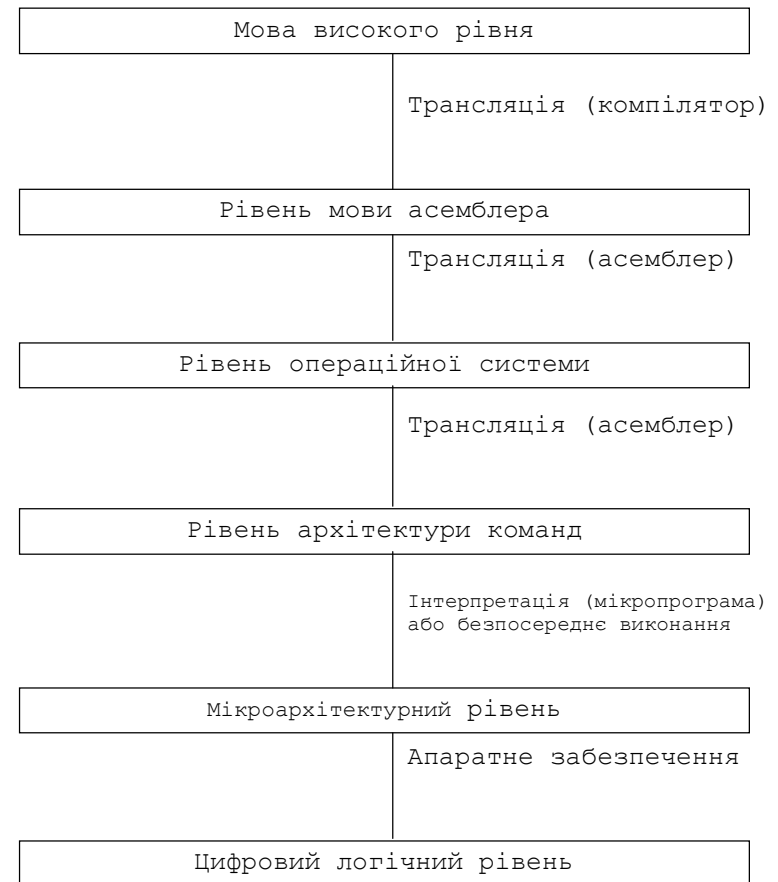


Рисунок 2.5 - Багаторівнева багатоядерна організація комп'ютера

Приклад тракту пересилання даних під час виконання операцій додавання значень даних регістрів А, В у CPU для фон-неймановського комп'ютера показано на рис. 2.6.

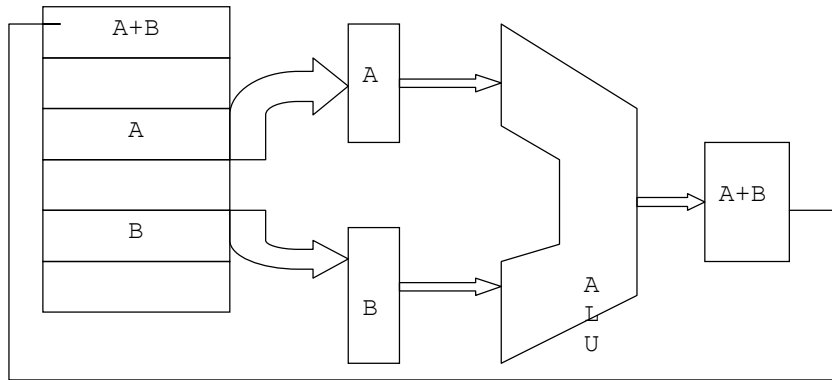


Рисунок 2.6 - Пересилання даних при додаванні

2.4. Апаратний рівень подання комп'ютера

Комп'ютер являє собою складну систему пов'язаних між собою елементів, які утворюють структуру комп'ютера. Подання структури може проводитись на таких рівнях:

- електричних схем, для яких елементами є транзистори, діоди, резистори тощо;

- логічних схем, коли у якості неподільних елементів використовують логічні і запам'ятовувальні елементи;

- операційних схем, які являють сукупність пристроїв і зв'язків між ними для реалізації мікрооперацій. Під мікрооперацією розуміють елементарний акт передавання або перетворення інформації, який ініціюється одним керуючим сигналом;

- структурних схем, кожна з яких виконує комплекс функцій передавання і перетворення інформації (процесор, накопичувачі на магнітних дисках, мережні карти, мікропроцесор, монітор тощо).

Логічні схеми мають умовні позначення. Системи позначень можуть дещо розрізнятись для одних і тих логічних функцій. Найбільш поширеними з таких елементів є схеми логічних елементів І, АБО, НІ, тригери, суматори та напівсуматори.

Операційну схему можна поділити на дві частини: операційну і керуючу. Операційна частина складається із взаємопов'язаних пристроїв і дозволяє виконувати сукупність мікрооперацій над словами. Керуюча частина працює як мікропрограмний автомат, який на основі послідовності вхідних і внутрішніх інформаційних сигналів виробляє послідовність керуючих сигналів для виконання перетворення інформації. Порядок функціонування операційної частини описується мікропрограмами, на основі яких синтезується структура керуючої частини операційної схеми.

2.5 Структура комп'ютера з відкритою архітектурою

2.5.1 Склад і призначення основних компонентів

Переважає кількість сучасних ПК наслідує відкриту архітектуру побудови комп'ютера, запропоновану компанією IBM. Відкритість передбачає можливість підключення нових додаткових пристроїв, зокрема системних пристроїв і плат розширення, а також просте вбудовування програм користувача на різних рівнях програмного забезпечення комп'ютера. Основні компоненти комп'ютера з відкритою архітектурою показано на рис. 2.7.

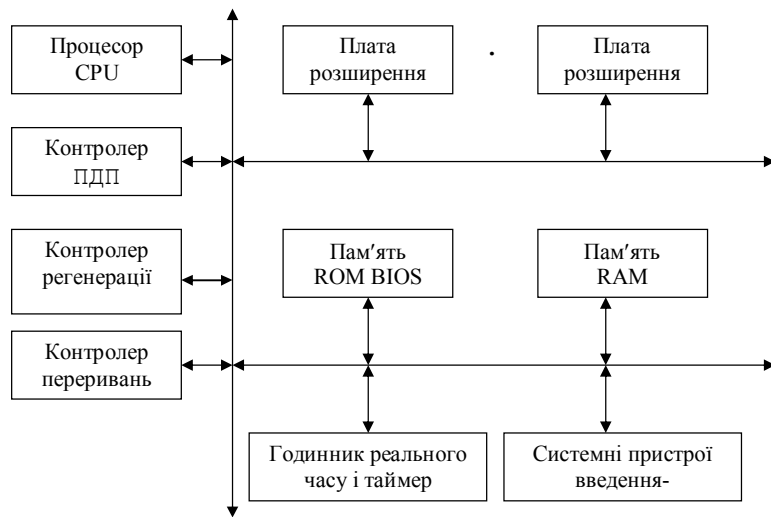


Рисунок 2.7 - Архітектура комп'ютера з відкритою архітектурою.

Двонаправленими стрілками позначені системні шини (даних, адрес, керування) системної магістралі, через які відбувається обмін даними між окремими складовими комп'ютера.

Постійна пам'ять ROM BIOS містить програму початкового завантаження, опис конфігурації системи, драйвери для взаємодії з системними пристроями.

Центральний процесор CPU (Central Processing Unit) зазвичай має швидко внутрішню оперативну пам'ять – кеш-пам'ять, яка може включати кілька рівнів (наприклад, L1, L2, L3).

Контролер переривань перетворює переривання системної магістралі (шини) у апаратні переривання центрального процесора CPU і визначає адреси переривання, за якими починається обробка цих переривань у разі їх виникнення.

Контролер прямого доступу до пам'яті дозволяє розвантажити процесор від пересилання даних між зовнішніми пристроями введення-виведення і оперативною пам'яттю RAM (Random Access Memory). Оперативна пам'ять RAM є системною і призначена для поточного зберігання програм і даних, які реалізують процеси обробки даних.

Контролер регенерації виконує оновлення у динамічній оперативній пам'яті DRAM (Dynamic RAM). Пам'ять DRAM є реалізацією RAM і має своїми перевагами великий об'єм при малих розмірах а також невелике енергоспоживання.

Годинник часу комп'ютера веде облік часу (дати і поточного часу), який використовується при створенні і поновленні файлів синхронізації процесів тощо.

Плати розширення дозволяють доповнювати комп'ютер новими пристроями. Для встановлення плат розширення на материнській платі

(motherboard) комп'ютера передбачені спеціальні місця для рознімів – слоти. Слоти мають підключення у межах системних магістралей, що дозволяє використовувати відповідні плати, зокрема одразу після підключення плат за технологією “підключи та грай” – PnP (Plug and Play), широке застосування якої розпочалося з 1997 р.

Однією з важливих вимог удосконалення архітектури комп'ютера є підвищення продуктивності обміну даними між компонентами комп'ютера. Це забезпечується переважно удосконаленням типів системних магістралей, швидкості обміну між окремими пристроями і використанням кількох системних шин.

2.5.2 Шинна організація комп'ютера

Шина VLB (VESA Local Bus), або VL-BUS є 32-розрядним стандартом локальної шини, розроблений асоціацією VESA у 1992 р.

Шина AGP (Accelerated Graphics Port) – прискорений графічний порт є 32-розрядною специфікацією шини графічного інтерфейсу обміну даними для мікросхем корпорації Intel реалізації тривимірної графіки. Шина AGP ґрунтується на PCI, має тактову частоту 133 МГц і розроблено у 1997 р. Існують версії AGP 1x, AGP 2x, AGP 4x, AGP 8x, які мають швидкості передавання даних відповідно 266, 533, 1066, 2128 Мбайт/с. Остання версія спрямована на зменшення енергоспоживання графічних адаптерів.

Шина ISA (Industry Standard Architecture) є 8-розрядною шиною розширення, яка для 16-розрядних ПК отримала назву розширеної ISA – EISA (Expanded ISA) або AT-bus з такою частотою 8 МГц.

32-розрядна шина PCI призначена для з'єднання периферійних компонентів (диски, відеоадаптер тощо) з центральним процесором, являє альтернативу шин VLB, EISA і розрахована на тактову частоту пересилання даних 132 Мбайт/с. Шина розроблена у 1993 р., а з 1995 р. стала промисловим стандартом. Існують ряд удосконалень цієї шини, зокрема PCI Express (3GIO) з швидкістю передавання до 250 Мбайт/с і PCI-X, яка є 64-розрядною, сумісною з PCI і має тактову частоту 133 МГц і пропускну здатність до 1,06 Гбайт/с. Специфікація PCI-X 2.0 має тактові частоти 266 і 533 МГц.

2.5.3 Багатошинна організація комп'ютерів

У процесі удосконалення архітектури було розроблено кілька основних типів системних шин: PC XT-Bus, ISA (Industry Standard Architecture), EISA (Enhanced ISA), VLB (VESA Local Bus) (з 1989 р.), PCI (Peripheral Component Interconnect Bus) (з 1999 р.). Системна шина PCI зараз є найбільш поширеною.

Оскільки значна частина даних пересилається між основною пам'яттю і процесором, то застосовують додатково високошвидкісні локальну шину і шину пам'яті за структурою з трьома шинами, як показано на рис. 2.8.

Локальна шина забезпечує зв'язок між швидкою кеш-пам'яттю і процесором; шина пам'яті забезпечує зв'язок постійної пам'яті, системної пам'яті і контролера системної шини. До системної шини підключена решта компонентів комп'ютера. Склад ліній системних шин (адресних ліній, ліній даних, керуючих сигналів) можуть суттєво розрізнятися.

3 КОМБІНАЦІЙНІ СХЕМИ

3.1 Дешифратори і шифратори

3.1.1 Дешифратори.

Вузлом (операційною схемою) комп'ютера називають обладнання, за допомогою якого виконується одна або декілька мікрооперацій.

Повним дешифратором називається КС, яка має n входів і 2^n виходів, і реалізує 2^n функцій виду:

$$D_j(x) = \begin{cases} 1, & j = x, j = 0, 1, \dots, 2^n - 1 \\ 0, & j \neq x, x = \sum_{i=1}^n x_i 2^{i-1} \end{cases}$$

Кожному n – розрядному вхідному слову відповідає одиничний сигнал на одній з вихідних 2^n шин і нулі на решті виходів. Кожну функцію D_j можна подати у вигляді:

$$\begin{aligned} D_0 &= \bar{x}_n \bar{x}_{n-1} \dots \bar{x}_2 \bar{x}_1 \\ D_1 &= \bar{x}_n \bar{x}_{n-1} \dots \bar{x}_2 x_1 \\ &\dots \\ D_{2^n-1} &= x_n x_{n-1} \dots x_2 x_1 \end{aligned}$$

Система функцій є сукупністю всіх можливих конституент одиниці, які можна утворити від n змінних.

Наприклад, дешифратор у випадку $n=3$ позначають, як показано на рис. 3.1а, а функціонує за таблицею істинності на рис. 3.1б:

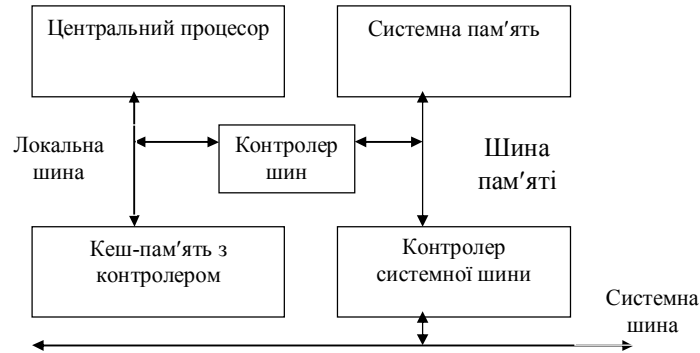


Рисунок 2.8 - Трьохшинна організація з'єднання компонентів комп'ютера

Якщо в архітектурі комп'ютера застосовують різні типи симених шин (наприклад, ISA, PCI), то для кожної з них використовують свої контролери системної шини. Приклад такої архітектури показано на рис. 2.9.

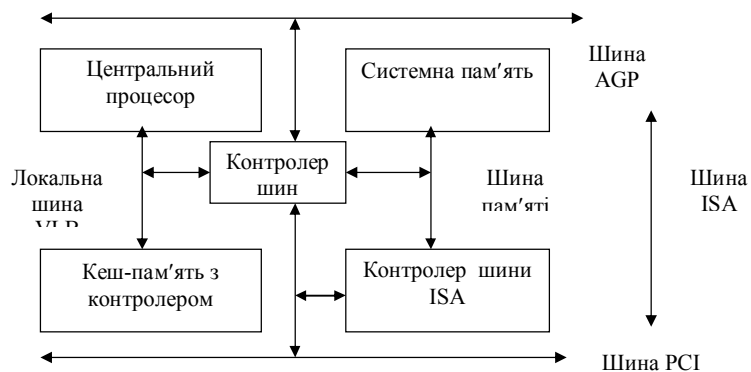
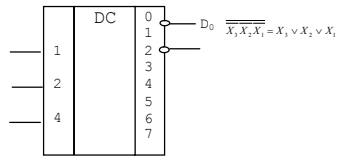


Рисунок 2.9 - Трьохшинна організація з'єднання компонентів комп'ютера



а)

X1	X2	X3	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

б)

Рисунок 3.1 - Дешифратор

Дешифратор це перетворювач двійкового позиційного коду в 2^n код.

Бувають дешифратори з інверсними виходами (рис. 3.2).

X3	X2	X1	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Рисунок 3.2 – Дешифратор з інверсними виходами

Дешифратори можуть мати додаткові входи управління.

Відомі два способи реалізації дешифраторів $D_j(x)$: лінійний і каскадний. У лінійному дешифраторі кожен з функцій $D_j(x)$ реалізують окремо на n - входівому елементі (у випадку дешифратора з прямими виходами або дешифратора з інверсними виходами). Якщо для побудови дешифратора необхідно використовувати І-НЕ, то функції $D_j(x)$ необхідно подавати у такій операторній формі:

$$D_j(x) = \overline{\tilde{x}_n \tilde{x}_{n-1} \dots \tilde{x}_2 \tilde{x}_1}$$

Якщо дешифратор необхідно реалізувати на АБО-НЕ, то $D_j(x)$ подають у такій операторній формі:

$$D_j(x) = \tilde{x}_n \vee \tilde{x}_{n-1} \vee \dots \vee \tilde{x}_2 \vee \tilde{x}_1$$

Каскадний спосіб побудови дешифраторів використовують тоді, коли n (кількість входів) велике і перевищує кількість входів елементів. Дешифроване слово розбивається на підслова. Для кожного слова формують всі константи одиниці шляхом побудови лінійного дешифратора. У кожному наступному каскаді утворюється кон'юнкція конститuent одиниці попереднього каскаду. Каскадний метод побудови дешифраторів найбільш поширений.

На практиці часто використовують неповні дешифратори, тобто такі, у яких кількість виходів менше 2^n (частковий дешифратор). Прикладом є двійково-десятковий дешифратор, у якому кожний десятковий розряд подають тетрадою (чотири розряди) двійкових розрядів.

3.1.2. Шифратори

Шифратором називають схему для перетворення унітарного m -розрядного коду у k -розрядний двійковий позиційний код. У загальному випадку: $m > k$, $k = \lfloor \log_2 m \rfloor$. Шифратор виконує операцію, обернену до функції дешифратора.

Шифратор перетворює одиночний сигнал на одному з m входів у двійковий номер цього входу. Наприклад, для $m=8$ на рис. 3.3 наведено таблицю істинності шифратора.

Входи								Виходи		
7	6	5	4	3	2	1	0	Y3	Y2	Y1
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Рисунок 3.3 – Таблиця істинності для шифратора

3.2 Мультиплектори і демультиплектори

3.2.1. Мультиплектори

Мультиплексором називається функціональний вузол комп'ютера, призначений для почергової комутації (перемикання) інформації від одного з n входів на загальний вихід. Номер конкретної вхідної лінії, що підключається до виходу в кожний такт машинного часу, визначається

адресним кодом A_0, A_1, \dots, A_{m-1} . Зв'язок між числом інформаційних n і адресних m входів визначається співвідношенням $n=2^m$. Таким чином, мультиплексор реалізує керовану передачу даних від кількох вхідних ліній в одну вихідну.

Умовне графічне позначення мультиплексорів показано на рис. 3.4. Функція мультиплексорів записується буквами MUX (multiplexor). Мультиплектори застосовують для таких операцій: комутації окремих ліній і груп ліній (шин); перетворення паралельного коду в послідовний; реалізації логічних функцій; побудови схем порівняння, генераторів кодів.

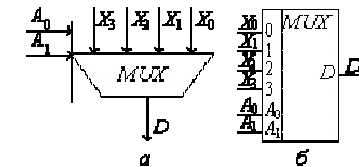


Рисунок 3.4 – Мультиплексор

Мультиплексор символічно часто позначають: “ $n-1$ ”. Логіка роботи чотиривходового мультиплексора наведена на рис. 3.5, де A_0, A_1 – адресний код; F_0, F_1, F_2, F_3 – виходи внутрішнього дешифратора; X_0, X_1, X_2, X_3 – вхідна інформація; D – загальний інформаційний вихід.

На основі рис. 3.5 вираз для вихідної функції D можна представити з використанням виходів F_0-F_3 внутрішнього дешифратора у вигляді:

$$D = F_0 X_0 \vee F_1 X_1 \vee F_2 X_2 \vee F_3 X_3,$$

A1	A0	F0	F1	F2	F3	D
0	0	1	0	0	0	F0X0
0	1	0	1	0	0	F1X1
1	0	0	0	1	0	F2X2
1	1	0	0	0	1	F3X3

Рисунок 3.5 – Таблиця істинності для мультиплексора

З використанням мінтермів адресного коду (рис. 3.6)

$$D = \overline{A_1} \overline{A_0} X_0 \vee \overline{A_1} A_0 X_1 \vee A_1 \overline{A_0} X_2 \vee A_1 A_0 X_3.$$

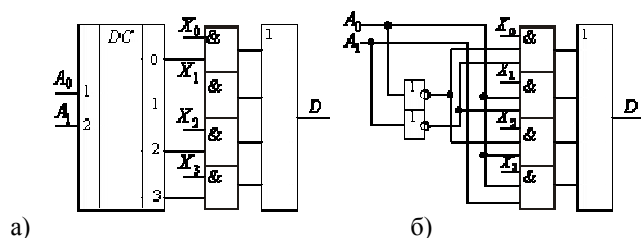


Рисунок 3.6 - Схеми мультиплексорів: а) – з внутрішнім дешифратором; б) – з адресними мінтермами

Мультиплексори відносять до пристроїв комутування цифрової інформації. Вони здійснюють комутацію одного з кількох інформаційних входів x_i до одного виходу y .

Схеми мультиплексорів мають кілька інформаційних входів, адресні входи, вхід дозволу мультиплексування (стробуючий вхід) та один вихід.

Кожному з інформаційних входів мультиплексора відповідає номер, який називається адресою, двійкове число якого подається до адресних входів.

Часто мультиплексори мають входи керування для можливості їх подальшого об'єднання. На рис. 3.7, 3.8 показано приклад мультиплексора 4-1 на основі дешифратора.

Адресний дешифратор D1, перетворює двійковий код у десятковий для керування роботою мультиплексора. В залежності від комбінації стану адресних входів a_1 та a_2 на одному з чотирьох виходів дешифратора з'являється одиничний потенціал, який дає дозвіл на спрацьовування відповідної схеми I ($D_2 \dots D_5$). Наприклад, при адресному числі 01, коли $a_1=1$ та $a_2=0$, на виході 1 дешифратора D1 установлюється рівень логічної одиниці, а на всіх останніх — нульовий. Тому логічний елемент D3 має дозвіл на спрацьовування.

Якщо при цьому на інформаційному вході x_1 діє логічна одиниця, то на виході D_3 установлюється 1, а при $x_1=0$ на виході логічного елемента D_3 буде теж нульовий потенціал. При цьому, незалежно від стану інформаційних входів x_0, x_2, x_3 , на виході логічного елемента АБО D_6 інформація повторює стан x_1 . Якщо активізований вхід дозволу $E=1$, то на виході мультиплексора у з'являється 1 або 0 в залежності від значення x_1 .

Функціонування мультиплексора описується таблицею істинності рис. 3.7. При нульовому керуючому сигналі $E=0$ зв'язок між інформаційними входами x_i та виходом y відсутній. Тому незалежно (позначка «X») від стану адресних входів a_1 та a_2 вихід y нульовий. При $E=1$ на вихід передається логічний рівень того з інформаційних входів x_i номер якого i заданий на адресних входах.

Адресні входи		Керуючий вхід E	Вихід y
a1	a2		
X	X	0	0
0	0	1	x0
1	0	1	x1
0	1	1	x2
1	1	1	x3

Рисунок 3.7 – Таблиця істинності для мультиплексора 4-1

Логічний вигляд мультиплексора 4-1 має вигляд (рис. 3.8)

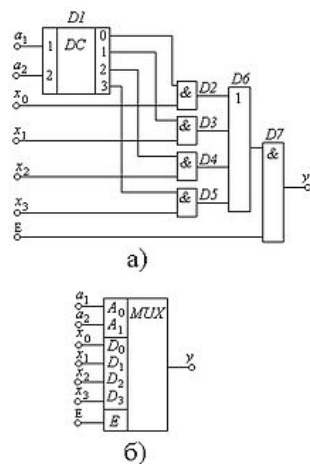


Рисунок 3.8 – Мультиплексор 4-1; а — схема; б — умовне позначення

Мультиплексори мають різне число входів, починаючи з 2. Деякі мультиплексори мають комплементарний вихід (прямий у інверсний).

При комутації багаторозрядних слів використовують кілька мультиплексорів, виходи яких з'єднуються за схемою АБО. Для цієї мети випускаються кілька однотипних мультиплексорів в одному корпусі.

3.2.2 Демультимплексори

Демультимплексором називають функціональний вузол комп'ютера, призначений для комутації (перемикання) сигналу з одного інформаційного входу на один з n інформаційних виходів.

Демультимплексор відноситься до пристроїв комутування цифрової інформації. Він здійснює комутацію одного інформаційного входу до одного з декількох виходів, адреса якого задана. Демультимплексор має один інформаційний вхід, декілька виходів та адресні входи (рис. 3.9).

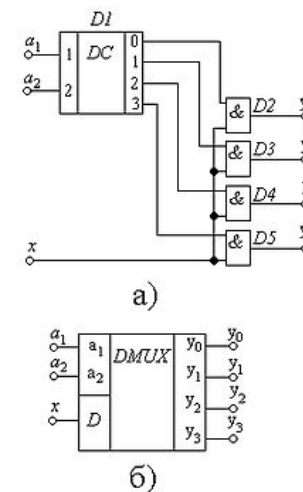


Рисунок 3.9 – Демультимплексор 1-4 на базі дешифратора D1 та логічних елементів 2-1 D2...D5 (без входу дозволу); а - схема; б - умовне позначення

Таким чином, на приймальному кінці мультимплексованої магістралі потрібно виконати зворотню операцію - демультимплексування.

Демультимплексор можна реалізувати на дешифраторі з n-входами, в якому вхід дозволу E використовується як інформаційний. Якщо для побудови схеми демультимплексора використати дешифратор без входу дозволу E, то необхідно мати m двовхідних логічних елементів 2-І.

Входи дешифратора a1, a2 є адресними. Тому в залежності від адресного числа лише на одному з виходів дешифратора з'являється логічна одиниця, яка дає дозвіл до спрацювання лише одного з чотирьох кон'юкторів D2...D5. На другі входи кожного кон'юктора надходить шина сигналу x.

Вхідна інформація відтворюється на виході одного з чотирьох логічних елементів D2...D5, який одержав дозвіл по другому адресному входу.

Можна виконати синхронний демультимплексор, якщо використовувати три-входові логічні елементи 3І і на третій вхід подати синхросигнал або сигнал дозволу від зовнішнього джерела.

Функціонування демультимплексора 1-4 подають таблицею істинності (рис. 3.10)

Адресні входи		Виходи			
a ₁	a ₂	Y ₀	Y ₁	Y ₂	Y ₃
0	0	x	0	0	0
1	0	0	x	0	0
0	1	0	0	x	0
1	1	0	0	0	x

Рисунок 3.10 – Таблиця істинності демультимплексора

3.2.3 Каскадування мультимплексорів

В інтегральному виконанні мультимплексори випускають на чотири, вісім або шістнадцять входів. Каскадування дозволяє реалізувати комутацію довільного числа вхідних ліній на базі серійних мікросхем мультимплексорів меншої розрядності.

Приклад побудови схеми мультимплексора на 16 входів на основі типових чотиривходових мультимплексорів показаний на рис. 3.11.

Молодші розряди адреси A1, A0 підключаються до адресних входів усіх мультимплексорів першого рівня, на виходах яких виробляються такі функції: $D_0 = F_0 X_0 \vee F_1 X_1 \vee F_2 X_2 \vee F_3 X_3$; $D_1 = F_0 X_4 \vee F_1 X_5 \vee F_2 X_6 \vee F_3 X_7$; $D_2 = F_0 X_8 \vee F_1 X_9 \vee F_2 X_{10} \vee F_3 X_{11}$; $D_3 = F_0 X_{12} \vee F_1 X_{13} \vee F_2 X_{14} \vee F_3 X_{15}$, де F0 – F3 – виходи внутрішніх дешифраторів: $F_0 = \overline{A_1} \overline{A_0}$; $F_1 = \overline{A_1} A_0$; $F_2 = A_1 \overline{A_0}$; $F_3 = A_1 A_0$; X15 – X0 – вхідні змінні.

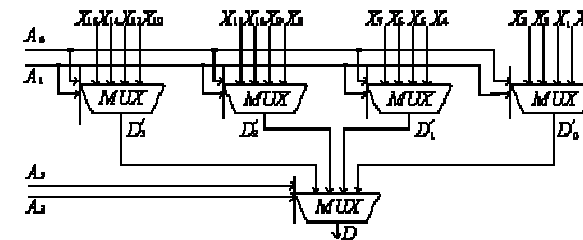


Рисунок 3.11 – Каскадування мультимплексорів

3.2.4 Реалізація логічних функцій мультимплексорами

За допомогою мультимплексорів реалізуються логічні функції з числом змінних m, що дорівнює розрядності адресного коду. Функція, що

виконується, має бути представлена в ДДНФ. При цьому змінні поступають на адресні входи, а інформаційні входи використовуються як настроювальні – на них подаються константи нуля і одиниці залежно від функції, яка реалізується. Вихідна функція триадресного мультиплексора на вісім входів описується рівнянням:

$$D(A) = \bar{A}_2 \bar{A}_1 \bar{A}_0 X_0 \vee \bar{A}_2 \bar{A}_1 A_0 X_1 \vee \bar{A}_2 A_1 \bar{A}_0 X_2 \vee \bar{A}_2 A_1 A_0 X_3 \vee A_2 \bar{A}_1 \bar{A}_0 X_4 \vee A_2 \bar{A}_1 A_0 X_5 \vee A_2 A_1 \bar{A}_0 X_6 \vee A_2 A_1 A_0 X_7.$$

Якщо потрібно отримати логічну функцію з десятковими еквівалентами мінтермів 1, 3, 5 і 7, то на парні входи X₀, X₂, X₄ і X₆ необхідно подати константу “0”, а на непарні X₁, X₃, X₅ і X₇ – константу “1”. У результаті отримаємо (рис. 3.12):

$$D(A) = \bar{A}_2 \bar{A}_1 A_0 \vee \bar{A}_2 A_1 A_0 \vee A_2 \bar{A}_1 A_0 \vee A_2 A_1 A_0.$$

За допомогою додаткових логічних перетворень можна реалізувати логічні функції з числом змінних m+1, тобто на одиницю більше розрядності адресного коду мультиплексора.

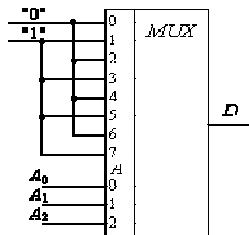


Рисунок 3.12 – Логічна функція на мультиплексорі

3.2.5 Мультиплексування шин

Мультиплексування шин – це по чергове перемикання шин (груп ліній) від кількох джерел інформації до одного приймача. Такі мікрооперації реалізуються схемами на основі мультиплексорів одиночних ліній. При виборі кількості й типу мультиплексорів враховують: число комутуваних шин дорівнює 2^m, де m – довжина адресного коду; i-й номер входу всіх мультиплексорів служить для підключення розрядів певної однієї шини.

Схема мультиплексора чотирьох X(n), Y(n), Z(n) і S(n) шин показана на рис. 3.13. Для її побудови потрібно n двоадресних чотиривходових мультиплексорів, де n – довільна розрядність шин, що комутуються.

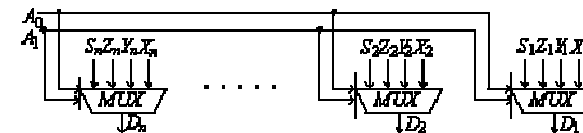


Рисунок 3.13 – Мультиплексор шин

3.2.6 Застосування демультимплексорів

Демультимплексором називається функціональний вузол комп'ютера, призначений для комутації (перемикання) сигналу з одного інформаційного входу D на один з n інформаційних виходів. Номер виходу, на який в кожний такт машинного часу передається значення вхідного сигналу, визначається адресним кодом A₀, A₁, A₂, ..., A_{m-1}. Адресні входи m інформаційні виходи n пов'язані співвідношенням n=2^m або m=

$\log_2 n$. Мультиплексори і демультимплексори також називають “селектори” даних.

В умовних графічних позначеннях (рис. 3.14) функція демультимплексора позначається буквами DMX.

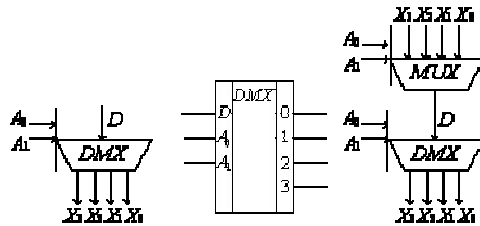


Рисунок 3.14– Умовні графічні позначення демультимплексорів: а – на функціональних схемах; б – на принципових схемах; в – типове з’єднання з мультиплексором

Демультимплексори використовують для таких операцій: комутації як окремих ліній, так і багаторозрядних шин; перетворення послідовного коду в паралельний; реалізації логічних функцій тощо. Демультимплексори часто позначають: “ $1 \rightarrow n$ ”. Логіка роботи двоадресного демультимплексора на мові мікрооперацій наведена на рис. 3.15, де D – інформаційний вхід; F0, F1, F2 і F3 – виходи внутрішнього дешифратора адреси.

A1	A0	F0	F1	F2	F3	X0	X1	X2	X3
0	0	1	0	0	0	F0D	–	–	–
0	1	0	1	0	0	–	F1D	–	–
1	0	0	0	1	0	–	–	F2D	–
1	1	0	0	0	1	–	–	–	F3D

Рисунок 3.15 – Логіка роботи двоадресного демультимплексора

За даними рис. 3.15 записуємо систему рівнянь для інформаційних виходів:

$$X0 = F0D = \bar{A}_1 \bar{A}_0 D; \quad X1 = F1D = \bar{A}_1 A_0 D;$$

$$X2 = F2D = A_1 \bar{A}_0 D; \quad X3 = F3D = A_1 A_0 D;$$

На основі рівнянь побудовані схеми демультимплексорів із внутрішнім дешифратором (рис. 3.16, а) і з поєднанням адресних і вхідних змінних на тривходових елементах І (рис. 3.16, б).

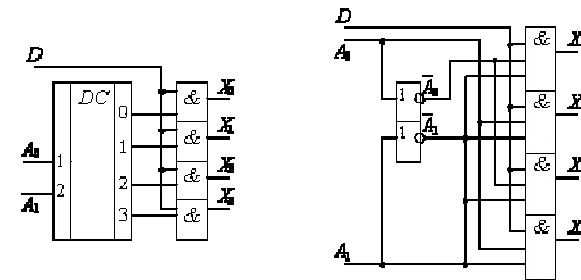


Рисунок 3.16– Схема демультимплексорів: а – з внутрішнім дешифратором; б – з поєднанням адресних і вхідних змінних

Схема демультимплексора з поєднанням адресних і вхідних змінних забезпечує високу швидкість, проте вимагає застосування логічних елементів з більшим числом входів.

3.2.7 Каскадування демультиплексорів

Каскадування дозволяє реалізувати комутацію одного вхідного сигналу на довільне число вихідних ліній на базі серійних мікросхем меншої розрядності. Нехай потрібно реалізувати демультиплексування вхідного сигналу на n вихідних ліній, що визначаються m -розрядним адресним кодом, на базі типових мікросхем меншої розмірності виду “ $1 \rightarrow n$ ”. Для цього потрібно використати $L = n/n_1$ типових демультиплексорів з числом адресних входів $m_1 = \log_2 n_1$ кожен.

Число старших адресних розрядів, що дорівнює різниці $m - m_1$, використовується додатковим “ведучим” демультиплексором, який розташовується у першому рівні схеми каскадування. Ведучий демультиплексор визначає почергове увімкнення одного з L демультиплексорів мікросхем другого рівня. Каскадування демультиплексорів виду “ $1 \rightarrow 4$ ” для реалізації комутатора “ $1 \rightarrow 16$ ” показано на рис. 3.17.

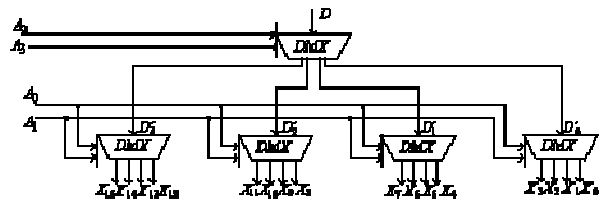


Рисунок 3.17 – Каскадування демультиплексорів

Нехай для схеми (рис. 3.17) адресний код $A_3A_2A_1A_0 = 1010$ і значення вхідного сигналу $D=1$. Тоді на виході ведучого демультиплексора $D_3 = A_3 \bar{A}_2 D = 1$, а на інших виходах встановлюються нульові значення.

Одиничне значення сигналу D_3 передається на вихід X_{10} веденого демультиплексора згідно зі співвідношенням $X_{10} = A_1 \bar{A}_0 D_3 = 1$.

Демультиплексори не випускають як курсові вироби на інтегральних мікросхемах. Функцію демультиплексора зазвичай реалізують на дешифраторах, що мають входи стробування (дешифратори-демультиплексори).

3.2.8 Демультиплексування шин

Під демультиплексуванням шин розуміється почергове перемикання груп ліній від одного джерела інформації до багатьох приймачів. Такі мікрооперації реалізуються зазвичай на основі демультиплексорів одиночних ліній. При виборі кількості і типу демультиплексора враховують: число шин, які комутуються, дорівнює 2^m , де m – довжина адресного коду; кількість демультиплексорів, які використовуються, визначається розрядністю n шин, які демультиплексуються; адресні входи всіх мультиплексорів паралельно об'єднуються. Схема мультиплексора вхідної шини $D(n)$ на чотири вхідні шини $X(n)$, $Y(n)$, $Z(n)$ і $S(n)$ показана на рис. 3.18.

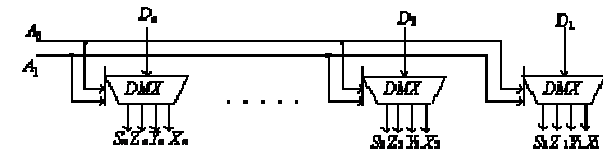


Рисунок 3.18 – Демультиплексор шин

Типове включення мультиплексорів і демультимплексорів для комутації вхідних і вихідних шин n -розрядних регістрів А, В, С і D показано на рис. 3.19. У АЛП така комутація забезпечує використання як першого операнда суматора вміст будь-якого регістра і запис результату операції в будь-який регістр, вказаний мікропрограмою команди, що виконується.

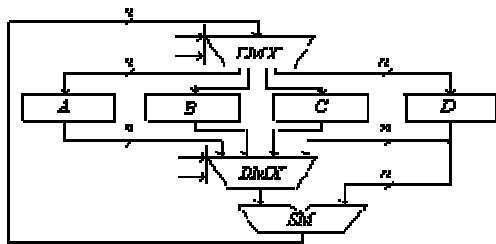


Рисунок 3.19 – Типова комутація вхідних і вихідних шин регістрів АЛП

3.3 Синтез комбінаційних схем

Синтез схем передбачає розробку функціональних схемних рішень, моделювання цих рішень для виявлення можливих недоліків при реалізації та експлуатації (моделювання роботи на основі часових діаграм, вибору різних рівнів параметрів сигналів для заданої технології виготовлення, вплив технологічних помилок на якість функціонування схем, аналіз характеристик функціонування з врахуванням відмов тощо), та подальше виготовлення дослідних зразків схем. Фізичне моделювання отриманих схем дає підстави для подальшого виготовлення партій мікросхем.

Сучасні підходи до проектування цифрових пристроїв передбачає застосування готових схем, які виробляє промисловість, створення нових схемних рішень на основі готових схем, застосування засобів автоматизації процесів проектування. Створення схем на основі існуючих передбачає використання типових схемних рішень. Автоматизація створення та моделювання схем включає вибір мови автоматизації проектування. Розглянемо деякі приклади цих підходів.

3.3.1 Реалізація перемикальних функцій з використанням типових комбінаційних схем

Для реалізації перемикальних функцій з використанням типових комбінаційних схем застосовують типові мікросхеми малої і середньої ступені інтеграції: мультиплексори, дешифратори, суматори тощо.

Мультиплексори дозволяють передавати сигнал одного з інформаційних входів на вихід схеми за значенням адресних входів. На інформаційних входах формують константи нуля або одиниці заданої

логічної функції, а за значеннями змінних, яким відповідають адресні входи мультиплексора відповідне значення функції передають на вихід.

Демультимплексори забезпечують передавання єдиного інформаційного входу на один з виходів, адреса якого визначається адресними входами схеми.

У поєднанні мультиплексори і демультимплексори застосовують для комутації сигналів $N \times N$, тобто довільний з N вхід на один з N виходів.

Дешифратори дозволяють реалізувати конституенти одиниці за заданими значеннями вхідних адрес.

3.3.2 Застосування мультиплексорів для побудовання комбінаційних схем

Використання мультиплексора дозволяє зменшити складність КС (кількість умовних корпусів мікросхем). Застосування мультиплексора для синтезу КС ґрунтується на тому, що будь-яку булеву функцію можна розкласти за n змінними.

Наприклад, розкладання функції від n аргументів за аргументами X_1, X_2 дасть такий вираз:

$$F(X_1, \dots, X_n) = \overline{X_1} X_2 F(0, 0, X_3, \dots, X_n) \vee \overline{X_1} X_2 F(0, 1, X_3, \dots, X_n) \vee X_1 \overline{X_2} F(1, 0, X_3, \dots, X_n) \vee X_1 X_2 F(1, 1, X_3, \dots, X_n) = \overline{X_1} \overline{X_2} F_0(X_3, \dots, X_n) \vee \overline{X_1} X_2 F_1(X_3, \dots, X_n) \vee X_1 \overline{X_2} F_2(X_3, \dots, X_n) \vee X_1 X_2 F_3(X_3, \dots, X_n)$$

де F_0, \dots, F_3 – залежать від $n-2$ –х аргументів.

Такому розкладенню функції відповідає КС:

За адресними входами X_1, X_2 визначають одну з функцій F_0, F_1, F_2, F_3 на виході мультиплексора.

Приклад Реалізувати функцію $y = \overline{a} * \overline{c} + a * \overline{c}$ на мультиплексорі.

ДДНФ функції $y = \overline{a} * \overline{c} + a * \overline{c} = V(0, 2)$. Тому на інформаційні входи 0, 2 треба подати значення логічної одиниці (F_0, F_2), а на решту входів (F_1, F_3) – значення логічного нуля. Оскільки вхід «земля» зазвичай відповідає нулю (генератор нуля), то значення одиничного сигналу отримують з використанням інвертора (генератор одиниці).

В результаті отримаємо схемну реалізацію, подану на рис. 3.20.

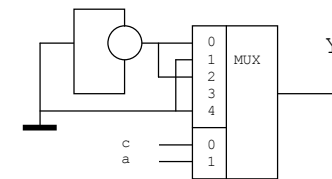


Рисунок 3.20 – Схемна реалізація на мультиплексорі

3.3.3 Реалізація логічних функцій на дешифраторах

Дешифратори з прямими виходами реалізують 1-конституенти (мін терми) вхідних змінних. Тому логічну функцію, подану у ДДНФ, можна реалізувати шляхом підключення відповідних виходів до входів схеми АБО.

Дешифратори з інвертованими виходами реалізують 0-конституенти (макстерми) вхідних змінних. Тому логічну функцію, подану у ДДНФ, можна реалізувати шляхом підключення відповідних виходів до входів схеми АБО, у ДКНФ – із застосуванням схем І.

Дешифратори із стробуючими входами дозволяють розширити кількість вхідних змінних дешифратора, тобто побудувати дешифратори з більшою кількістю адресних входів.

Приклад Реалізувати функцію $y = \bar{a} * \bar{c} + a * \bar{b} * c$ на дешифраторі з прямими виходами.

ДДНФ функції $y = \bar{a} * \bar{b} * \bar{c} + \bar{a} * b * \bar{c} + a * \bar{b} * c = V(0,2,5)$.

Отримаємо схемну реалізацію, подану на рис. 3.21.

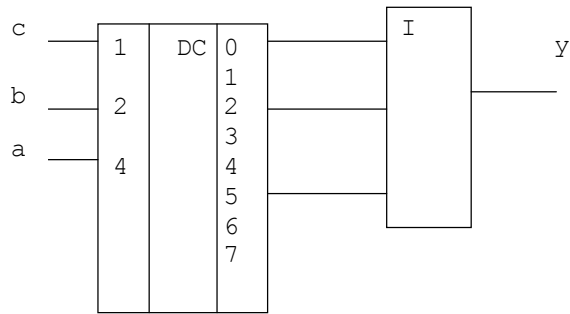
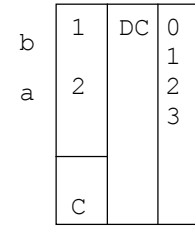


Рисунок 3.21 – Схемна реалізація на дешифраторі

Приклад На дешифраторі з двома адресними входами і із прямим стробуючим входом побудувати дешифратор з трьома входами.

Вихідний дешифратор (рис. 3.22а) має такі позначення таблицю істинності (рис. 3.22б).



а)

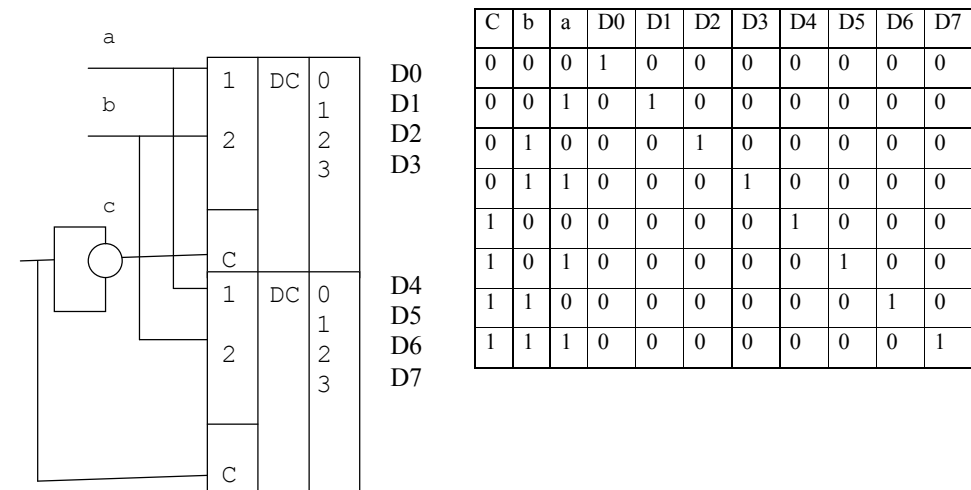
C	a	b	D0	D1	D2	D3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

б)

Рисунок 3.22 – Схемна реалізація дешифратора на 2 входи

Адресний вхід а є старшим, а в – молодшим.

Тому схемна реалізація передбачає використання входу С у якості старшого адресного входу і має такий вигляд (рис. 3.23).



D0
D1
D2
D3

D4
D5
D6
D7

C	b	a	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Рисунок 3.23 – Схемна реалізація дешифратора на 3 входи

3.4 Суматори

Суматором називають функціональний вузол, призначений для додавання двох чисел. Операцію віднімання замінюють додаванням чисел в оберненому або доповняльному кодах. Операції множення та ділення зводять до реалізації багаторазового додавання та зсування.

Суматори є основою побудови арифметико-логічних пристроїв, мікропроцесорів, спеціалізованих обчислювальних пристроїв, зокрема пристроїв порівняння, множення, ділення.

Існує велике різноманіття суматорів, які використовують у цифрових пристроях.

Для виконання арифметичних і логічних операцій застосовують відповідні мікросхеми суматорів і арифметико-логічних пристроїв для виконання цих операцій з врахуванням системи числення і форми подання даних. Такі мікросхеми зазвичай мають у своєму складі суматори і схеми вибору операндів, інвертування, додавання одиниці тощо. Арифметико-логічний пристрій є основою побудови мікропроцесорів.

Суматором (adder, summer) називають операційний вузол, призначений для виконання операції додавання двох чисел. Для позиційних систем числення суму чисел подають як

$$Z = X + Y$$

$$X = \sum_{i=1}^n x_i \cdot k^{i-1}$$
$$Y = \sum_{i=1}^n y_i \cdot k^{i-1}$$

де k – основа системи числення (двійкова, трійкова, тощо);

x_i, y_i – значення i -го розряду чисел X, Y .

Зазвичай функцію суматора на схемах позначають буквами SM або Σ . Суматор складається з окремих схем, які називаються однорозрядними сумматорами. Однорозрядні сумматори виконують дії з додавання значень однойменних розрядів двох чисел (операндів).

Суматори є основою побудови арифметико-логічних пристроїв, мікропроцесорів, CPU, спеціалізованих обчислювальних пристроїв.

3.4.1 Класифікація суматорів

Суматори класифікуються за такими ознаками: способом додавання (паралельні, послідовні та паралельно-послідовні); за кількістю входів (напівсуматори, однорозрядні та багаторозрядні суматори); організацією зберігання результату додавання (комбінаційні, накопичувальні, комбіновані); організацією перенесення між розрядами (з послідовним, наскрізним, паралельним або комбінованим перенесеннями); системою числення (позиційні двійкові, двійково-десяткові, трійкові, та непозиційні, наприклад, у системі залишкових класів); розрядністю (довжиною) операндів (зазвичай, 8-, 16-, 32-, 64-розрядні); способом подання від'ємних чисел (в оберненому або доповняльному кодах, а також в їхніх модифікаціях); за часом додавання (синхронні, асинхронні).

Класифікацію суматорів подано на рис. 3.24.

За способом виконання операції додавання розрізняють комбінаційні і накопичувальні суматори. У комбінаційних суматорах додавання виконуються одночасно у часі для всіх розрядів з одержанням результату по всіх розрядах. У накопичувальних суматорах додавання по розрядах розділено у часі: Спочатку виконується додавання для перших

(молодшиз) розрядів, з отриманням суми і сигналу переносу по розряду, який використовуються далі для додавання у наступних (старших) розрядах і т.і.

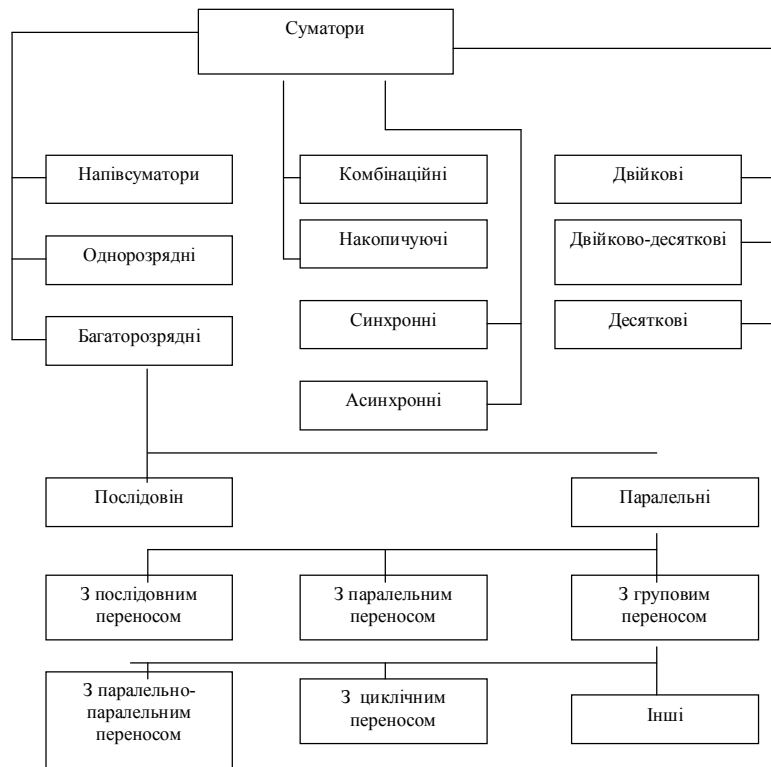


Рисунок 3.24- Класифікація суматорів

За кількістю входів розрізняють:

напівсуматори, тобто однорозрядні суматори ($n=1$) з двома вхідними розрядами відповідно для x_i , u_i ;

повні однорозрядні суматори, тобто однорозрядні суматори ($n=1$) з двома вхідними розрядами відповідно для x_i , u_i і входом переносу з молодшого розряду;

багаторозрядні суматори, призначені для додавання багаторозрядних чисел ($n>1$).

За способом подання і оброблення чисел багаторозрядні суматори поділяють на:

послідовні, для яких додавання виконується послідовно розряд за розрядом, тобто спочатку для $i=1$, потім для $i=2$ з врахуванням переносу з попереднього (молодшого) розряду;

паралельні, в яких всі розряди чисел подаються паралельно.

Серед паралельних багаторозрядних суматорів розрізняють суматори:

з послідовним переносом, у яких додавання в кожному окремому розряді виконується додаванням розрядів чисел x_i , u_i ; і переносу з $i-1$ -го розряду;

з паралельним переносом, у яких додавання відбувається паралельно в кожному окремому розряді додаванням розрядів чисел x_i , u_i ; і переносу, отриманого у результаті аналізу переносів всіх попередніх $i-1$ -го розрядів; в таких суматорах застосовують спеціальні додаткові схеми формування вхідного сигналу переносу по кожному розряду;

з груповим переносом, у яких паралельний сигнал переносу формується по окремих групах з послідовних розрядів, з використанням цих переносів, як у суматорах з послідовним переносом.

Суматори з паралельним переносом поділяють на суматори з паралельно-паралельним переносом, у яких групові переноси формуються паралельно по групах і використовуються між групами як паралельні, і

послідовно-паралельні з циклічним переносом, для яких переноси у групах формуються послідовно, а між групами паралельно.

За системою числення розрізняють двійкові і десяткові, а також комбіновані двійково-десяткові, для яких кожен цифру розряду подають у двійковій формі тетрадою (чотири двійковими розрядами).

За часом виконання додавання розрізняють синхронні і асинхронні суматори. У синхронних суматорах визначається постійний час (такт) виконання операції додавання, незалежно від значення доданків. В асинхронних суматорах крім сигналу переносу формується сигнал відсутності переносу, і тому час додавання може залежити від числових значень доданків.

Суматори з постійним інтервалом часу для додавання називаються синхронними. Суматори, в яких інтервал часу для додавання визначається моментом фактичного закінчення операції, називаються асинхронними.

В асинхронних суматорах є спеціальні схеми, які визначають фактичний момент закінчення додавання і повідомляють про це в пристрій керування. На практиці переважно використовуються синхронні суматори.

Суматори характеризуються такими параметрами: швидкістю – часом виконання операції додавання t_a , який відраховується від початку подачі операндів до одержання результату; часто швидкість характеризується кількістю додавання в секунду $F_a=1/t_a$, тут мають на увазі операції типу реєстр–реєстр (тобто числа зберігаються в реєстрах АЛП); апаратними витратами: вартість однорозрядної схеми додавання визначається загальним числом логічних входів використаних елементів; вартість багаторозрядного суматора визначається загальною кількістю використаних мікросхем; споживаною потужністю суматора.

3.4.2 Однорозрядні суматори

Однорозрядним суматором називається логічна схема, яка виконує додавання значень i -х розрядів X_i та Y_i двійкових чисел з урахуванням перенесення Z_i з молодшого сусіднього розряду та виробляє на виходах функції результат S_i і перенесення P_i в старший сусідній розряд.

На основі однорозрядних схем додавання на три входи та два виходи будуються багаторозрядні суматори будь-якого типу. Алгоритм роботи однорозрядного суматора відображається таблицею істинності (табл. 10.1). На основі таблиці рис. 3.25 записують систему логічних функцій для результату S_i та перенесення P_i у ДДНФ:

X_i	Y_i	Z_i	S_i	P_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рисунок 3.25– Таблиця суми і переносу однорозрядного суматора

$$S_i = X_i \bar{Y}_i \bar{Z}_i \vee X_i Y_i Z_i \vee X_i \bar{Y}_i Z_i \vee X_i Y_i \bar{Z}_i;$$

$$P_i = \bar{X}_i Y_i Z_i \vee X_i \bar{Y}_i Z_i \vee X_i Y_i \bar{Z}_i \vee X_i Y_i Z_i;$$

3.4.3 Багаторозрядні суматори

При комбiнаційних суматорiв враховують такі чинники: схема має характеризуватися регулярністю (подiбністю) структури та мінімальною вартістю, тобто мати по можливості найменше число логічних входiв всiх елементiв;

З метою підвищення швидкодiї багаторозрядного суматора потрібен мінімальний час одержання функції перенесення $t_{\Gamma} = k t_p$, де k – число послiдовно увiмкнених елементiв вiд входiв до виходiв P_i або \bar{P}_i ; t_p – середня затримка розповсюдження сигналу одним логічним елементом в обраній серiї iнтегральних мiкросхем; параметр k часто називають каскадністю схем.

Таким чином, для мінімiзацiї часу одержання перенесення необхідно зменшити каскадність схеми та використати iнтегральні мiкросхеми з малим часом затримки розповсюдження сигналу.

Для схем однорозрядних суматорiв на основi рiвнянь можуть вироблятися як прямі P_i , так й iнверсні значення функції перенесення. Така органiзацiя перенесень називається парафазною.

Для побудови схеми однорозрядного суматора на унiверсальних логічних елементах I-HE (рис. 3.26 а) застосовують перетворення на основi правил подвійної iнверсiї та де Моргана до такого вигляду:

$$S_i = \overline{\overline{X_i Y_i Z_i} \overline{X_i Y_i Z_i} \overline{X_i Y_i Z_i} \overline{X_i Y_i Z_i}}; \quad P_i = \overline{\overline{X_i Y_i} \overline{X_i Z_i} \overline{Y_i Z_i}}.$$

При засосуванні функції «Виключальне АБО» (XOR):

$$S_i = (X_i \oplus Y_i)Z_i \vee (\overline{X_i} \oplus \overline{Y_i})Z_i = X_i \oplus Y \oplus Z_i;$$

$$P_i = X_i Y_i \vee (\overline{X_i} Y_i \vee X_i \overline{Y_i})Z_i = X_i Y_i \vee (X_i \oplus Y_i)Z_i.$$

Схему однорозрядного суматора на елементах «виключальне АБО» показано на рис. 2.28 б .

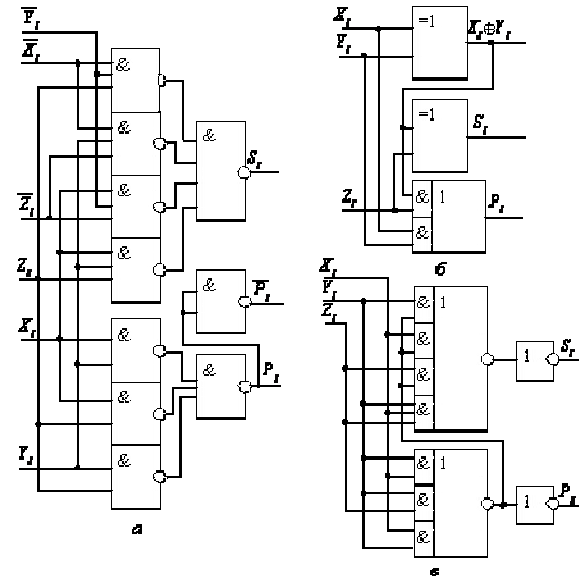


Рисунок 3.26 – Схеми однорозрядних суматорiв: а – на елементах I-HE ; б – на елементах «виключальне АБО»; в – з використанням власного перенесення

Напiвсуматором називають логічну схему, яка виконує додавання значень i -х розрядiв X_i i Y_i двiйкових чисел X i Y та реалiзує на виходi значення результату M_i i перенесення в старший сусiдній розряд R_i (:

$$M_i = \overline{X_i} Y_i \vee X_i \overline{Y_i} = X_i \oplus Y_i; \quad R_i = X_i Y_i.$$

Напівсуматор виконує лише частину завдання додавання в i -му розряді, оскільки не враховує перенесення з сусіднього молодшого розряду. Схему напівсуматора показано на рис. 2.29 а, б.

Схема однорозрядного суматора може бути побудована на основі двох напівсуматорів і додаткового логічного елемента ЧИ, як показано на рис 3.27. в.

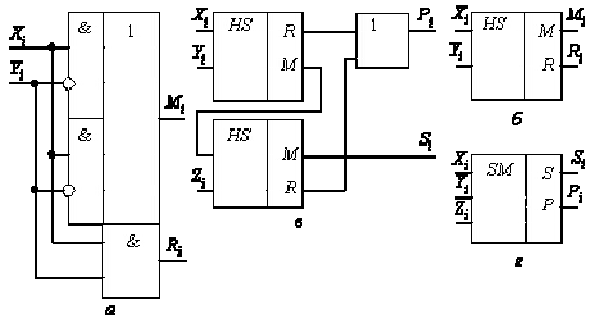


Рисунок 3.27– Схеми суматорів: а, б – напівсуматор і його умовне позначення; в, г – однорозрядний суматор і його умовне позначення

3.4.4 Послідовний багаторозрядний суматор

Послідовний двійковий багаторозрядний суматор містить: n -розрядні зсуваючі регістри операндів X і Y , регістр результату S , однорозрядний суматор SM і двоступеневий D -тригер для запам'ятовування перенесення. Усі регістри забезпечують одночасне зсування праворуч, у бік молодших розрядів (рис. 3.28).

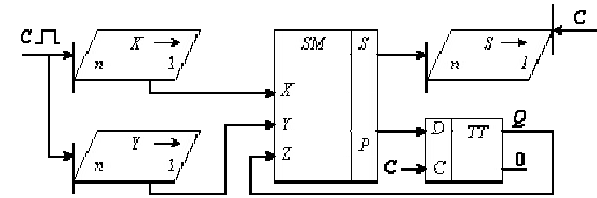


Рисунок 3.28– Схема послідовного багаторозрядного суматора

У послідовному суматорі попарна подача значень розрядів X_i і Y_i починається з молодших розрядів. Утворюються значення суми S_i і перенесення P_i , які записуються відповідно в регістр результату та в тригер запам'ятовування перенесення на один такт T_c . Послідовне додавання виконується за стільки тактів, скільки розрядів у числі. Тому час додавання t_s визначається співвідношенням: $t_s = nT_c$, де T_c – тривалість машинного такту.

Послідовний суматор потребує мінімальних апаратних витрат, однак тривалість операції додавання пропорційна розрядності операндів. Тому послідовний суматор можна використовувати у відносно повільнодіючих цифрових пристроях.

3.4.5 Паралельні багаторозрядні суматори

Паралельний багаторозрядний суматор містить n однорозрядних суматорів (схем додавання), наприклад, чотири, як показано на рис. 3.29.

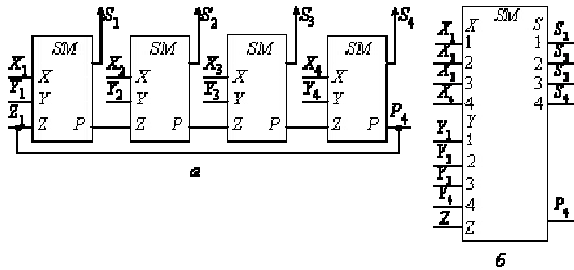


Рисунок 3.29– Паралельний чотирирозрядний суматор: а – схема; б – умовне позначення

Значення всіх розрядів двох чисел X та Y надходять на входи відповідних однорозрядних суматорів паралельно (одночасно). В паралельних суматорах з послідовним перенесенням значення сигналу перенесення P_i передається від розряду до розряду послідовно в часі (асинхронно).

3.4.6 Двійково-десяткові суматори

Двійково-десяткові суматори використовують для обробки масивів десяткової інформації за порівняно простими алгоритмами, оскільки при цьому вилучаються витрати часу на переведення чисел з десяткової системи числення в двійкову і навпаки. Кожну десяткову цифр X_i кодують двійковим кодом прямого заміщення “8421” (двійковою тетрадою), тобто $X_i = X_{i4}X_{i3}X_{i2}X_{i1}$ і $Y_i = Y_{i4}Y_{i3}Y_{i2}Y_{i1}$.

Наприклад $X_i = 7_{10} = 0111_{2-10}$, $Y_i = 9_{10} = 1001_{2-10}$; для дворозрядних десяткових чисел: $X_i X_{i-1} = 16_{10} = 00010110_{2-10}$; $Y_i Y_{i-1} = 28_{10} = 00101000_{2-10}$. Один розряд двійково-десятькового суматора (декада) містить чотирирозрядний суматор SM1 для одержання попередньої суми в тетраді,

чотирирозрядний суматор SM2 для корекції результату та логічний елемент І ЧИ для вироблення ознак корекції, як показано на рис. 3.30.

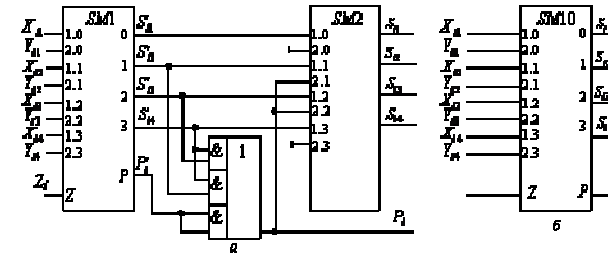


Рисунок 3.30 – Однорозрядний двійково-десятьковий суматор: а – схема; б – умовне позначення

Декада працює таким чином. Двійкові тетради десяткових цифр $X_i = X_{i4}X_{i3}X_{i2}X_{i1}$ і $Y_i = Y_{i4}Y_{i3}Y_{i2}Y_{i1}$ разом із перенесенням поступають на входи суматора SM1 і на його виходах утворюється попередня сума $S_i^* = S_{i4}S_{i3}S_{i2}S_{i1}$, де S_i^* – десятковий еквівалент тетради (рис. 3.31).

При цьому можливі три випадки: 1) для значення $0 \leq S_i^* \leq 9$ корекція не потрібна; 2) для значень $10 \leq S_i^* \leq 15$ потрібно відняти з попередньої суми число 10 і здійснити перенесення в старшу сусідню декаду; віднімання числа 10 в доповняльному коді відповідає додаванню за допомогою суматора SM2 до попереднього результату числа шість, тобто плюс 0110₂; ознакою такої корекції є одиничне значення функції корекції суми та перенесення

$$F^*_i = S^*_i4 S^*_i3 \vee S^*_i4 S^*_i2,$$

яке реалізується елементом І-АБО; 3) для значень $16 \leq S_i^* \leq 19$ на виході суматора SM1 виникає перенесення P_i^* з вагою 16₁₀.

До корекції					Після корекції					Примітка
P	S4	S3	S2	S1	P?	S?4	S?3	S?2	S?1	
0	0	0	0	0	0	0	0	0	0	Корекція не потрібна
0	0	0	0	1	0	0	0	0	1	
0	0	0	1	0	0	0	0	1	0	
...	
0	1	0	0	1	0	1	0	0	1	Корекція потрібна: мінус 10 і перенесення в старшу декаду
1	0	0	0	0	0	1	0	1	0	
1	0	0	0	1	0	1	0	1	1	
...	
1	0	0	0	0	0	1	1	1	0	
1	0	1	1	0	1	0	0	0	0	Корекція потрібна: плюс 6
1	0	1	1	1	1	0	0	0	1	
1	1	0	0	0	1	0	0	1	0	
1	1	0	0	1	1	0	0	1	1	

Рисунок 3.31 – Корекція десяткових цифр

Однак у старшій декаді його значення сприймається як 10, тому потрібно додати до попереднього результату за допомогою суматора число шість, тобто 0110_2 . З урахуванням наведеного вище рівняння F^*i функцію корекції результату та перенесення можна записати у вигляді: $P_i = P^*i$ в $F_i = P^*i$ в $S^*i_4 S^*i_3$ в $S^*i_4 S^*i_2$. Таким чином, в усіх випадках, коли $P_i = 1$, до попередньої суми додається число плюс 0110_2 і формується перенесення у старший розряд.

Схема чотирирозрядного двійково-десятькового суматора з послідовним перенесенням в тетрадах і між декадами показана на рис. 3.32.

Для двійково-десятькових суматорів можна використовувати групові структури прискорених перенесень.

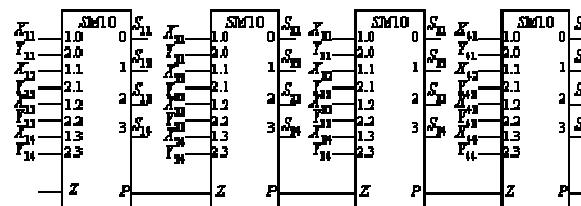


Рисунок 3.32 – Схема чотирирозрядного двійково-десятькового суматора.

Операція віднімання в двійково-десятьковому суматорі замінюється додаванням операндів у оберненому або доповнювальному кодах. Обернений код від'ємних десяткових чисел одержують заміною кожної цифри її доповненням до дев'яти.

Схема одного десяткового суматора з перетворювачами прямого коду операндів і результату в обернений код показана на рис. 3.33.

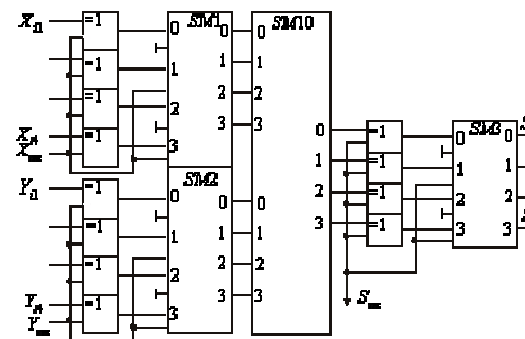


Рисунок 3.33 – Схема одного розряду десяткового суматора з перетворювачами прямого коду в обернений

Значення від'ємних чисел при $X_{zn}=1$, $Y_{zn}=1$, $S_{zn}=1$ інвертується схемою “виключальне АБО”; при цьому утворюється двійковий код тетради з надлишком шість. Корекцію результату виконують суматорами SM1, SM2 і SM3, в яких віднімання замінюється додаванням двійкової тетради з оберненим кодом числа шість, тобто плюс 1010_2 .

3.5 Схеми арифметико-логічних пристроїв

Мікросхеми арифметико-логічних пристроїв (ALU) призначені для виконання арифметичних та порозрядних логічних мікрооперацій залежно від вхідних сигналів налагодження.

Наприклад, мікросхема ALU в серіях ТТЛШ (рис. 3.34) включає: · інформаційні входи для подання двох чотирирозрядних операндів X і Y; · входи налаштування E3–E0 для задання номера однієї з мікрооперацій; · вхід M для задання типу мікрооперації: M=0 – арифметичні, M=1 – логічні; · вхід перенесення \bar{c}_1 , необхідний тільки при виконанні арифметичних мікрооперацій; · виходи: результату мікрооперації S4–S1, послідовного перенесення L, генерації G, транзиту H, а також вихід з відкритим колектором від внутрішнього компаратора для вироблення ознаки рівності операндів $F_{A=B}$

Перелік арифметичних і логічних операцій, які виконують ALU, наведений на рис. 3.34.

При виконанні логічних операцій перенесення між розрядами не використовується. Арифметичні операції реалізуються з урахуванням перенесень і позик. В арифметичні операції включені фрагменти логічних дій.

Наприклад, запис $(X \vee Y) + XY$ означає, що спочатку виконується операція інверсії (\bar{Y}), потім – логічного додавання ($X \vee Y$) та логічного множення ($X \cdot \bar{Y}$), а потім одержані таким чином два числа додаються арифметично з урахуванням перенесень.

E3	E2	E1	E0	Логіка M=1	Арифметика M=0
0	0	0	0	\bar{Y}	X
0	0	0	1	$\overline{X \vee Y}$	$X \vee Y$
0	0	1	0	$\bar{X}Y$	$X \vee Y$
0	0	1	1	0	-1
0	1	0	0	$\bar{X}\bar{Y}$	$X + X\bar{Y}$
0	1	0	1	\bar{Y}	$(X \vee Y) + X\bar{Y}$
0	1	1	0	$X \oplus Y$	$X - Y - 1$
0	1	1	1	$X\bar{Y}$	$X\bar{Y} - 1$
1	0	0	0	$\bar{X} \vee Y$	$X + X\bar{Y}$
1	0	0	1	$\overline{X \oplus Y}$	$X + Y$
1	0	1	0	Y	$(X + Y) + X\bar{Y}$
1	0	1	1	$X \cdot Y$	$X\bar{Y} - 1$
1	1	0	0	1	$X + Y$
1	1	0	1	$X \vee \bar{Y}$	$(X \vee Y) + X$
1	1	1	0	$X \vee Y$	$(X \vee \bar{Y}) + X$
1	1	1	1	X	$X - 1$

Рисунок 3.34 – Функції ALU

Мікросхема ALU виконує операцію арифметичного додавання двох чотирирозрядних операндів X і Y, якщо на входи керування подані сигнали E3E2E1E0=1001 та M=0. В цьому випадку мікросхема ALU виконує функцію суматора.

3.6 Спеціалізовані комбінаційні схеми

Апаратна реалізація архітектур комп'ютерів потребує застосування значної кількості схем, які забезпечують сумісне функціонування значної кількості компонентів керування і передавання даних (компараторів даних, схем контролю, визначення пріоритету для групи сигналів тощо) з врахуванням можливого впливу завад і відмов фізичних елементів.

3.6.1 Схеми порівняння

Схемою порівняння (компаратором) називають функціональний вузол комп'ютера, призначений для вироблення ознак відношень між двійковими словами (числами). Ознаки відношень записуються у вигляді: $F_i = A * K$ або $F_i, A * K$ або $F A * K$; $F_i = A * B$ або $P_i, A * B$ або $F A * B$, де A і B – двійкові або двійково-десяткові числа; K – двійкова константа; i – номер відношення (часто пропускається); $*$ – операція відношення вигляду $=, \neq, <, >, \leq, \geq$ і т. ін.; F_i – функція, що задає результат відношення: лог.1 – якщо відношення виконується, тобто істинне, і лог.0 – якщо відношення не виконується, тобто помилкове.

3.6.1.1 Загальна характеристика схем порівняння

Функція компаратора на схемах часто позначають символами COMP (comparator) або символами $=$.

Основними відношеннями вважаються: «рівне» $F_{A=B}$, «більше» $F_{A>B}$ і «менше» $F_{A<B}$. Часто схеми, що реалізують відношення $F_{A>B}$ і «менше» $F_{A<B}$, називають схемами порівняння «на більше» або «на менше». На їх основі можна отримати інші схеми, наприклад

$$F_{A \neq B} = \text{INV}(F_{A=B}),$$

$$F_{A \leq B} = \text{INV}(F_{A > B}),$$

$$F_{A \leq B} = F_{A=B} + F_{A < B}.$$

де INV, + - операції інвертування (заперечення) і диз'юнкції відповідно.

Схеми порівняння часто використовують для формування сигналів ознак відношення, логічних умов у мікропрограмних автоматах, у командах передачі керування, у пристроях контролю і діагностики.

Після виконання машинної команди комп'ютера автоматично формуються ознаки результатів операції. Ці ознаки, які називаються прапорами (прапорцями), вміщуються в спеціальний регістр прапорців. До прапорців зазвичай відносять ознаки нульового результату, переповнення розрядної сітки, знак результату, наявність перенесень із старшого розряду суматора, парне або непарне число одиниць в результаті тощо.

3.6.1.2 Схеми порівняння слів з константою

Нехай заданим є двійкове слово B , а слово A задано двійковими розрядами $A=A_2A_1A_0$. Необхідно отримати вектор прапорців для ознак відношень $F_1F_2F_3$ двійкового слова A з наступними заданими константами B :

$$F_1 := F_{A=B} (B=000);$$

$$F_2 := F_{A=B} (B=111)$$

$$F_3 := F_{A \leq B} (B=011).$$

3.6.1.3 Схеми порівняння багаторозрядних слів

На рис. 3.35 подано таблицю істинності значень ознак відношень F1, F2, F3 слова A з константами. В результаті отримуємо функції:

$$F1 = \text{INV}(A_2) * \text{INV}(A_1) * \text{INV}(A_0),$$

$$F2 = A_2 * A_1 * A_0,$$

$$F3 = \text{INV}(A_2)$$

A2	A1	A0	F1	F2	F3
0	0	0	1	0	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	0

Рисунок 3.35 – Функції порівняння слова з константою

Схема порівняння за цими функціями подано на рис. 3.36.

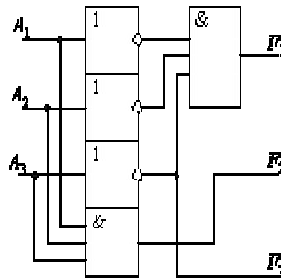


Рисунок 3.36 – Схема порівняння слова з константою

Двійкові n-розрядні слова рівні, коли одночасно попарно рівні їх розряди, тобто $A_i = B_i$ або $A(n) = B(n)$, $i = 1, 2, \dots, n$.

На рис. 3.37 подано функцію порівняння r_i одного розряду і слів A, B

Ai	Bi	ri
0	0	1
0	1	0
1	0	0
1	1	1

Рисунок 3.37 – Функція порівняння розряду і слів A, B

Функцію r_i -х розрядів A і B можна подати як

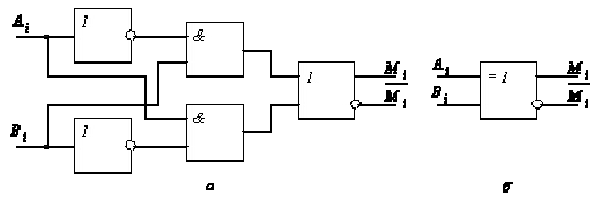
$$r_i = \overline{A_i} \overline{B_i} \vee A_i B_i = \overline{A_i \oplus B_i} = \overline{M_i}$$

де M_i – функція додавання по модулю 2.

Схемну реалізацію функції r_i подано на рис. 3.38.

Ознака рівності двох n-розрядних слів $P_{A=B}$ визначається логічним добутком порозрядних умов r_i :

$$P_{A=B} = r_n r_{n-1} \dots r_1 = \overline{M_n} \cdot \overline{M_{n-1}} \dots \overline{M_1}$$



а – схема; б – умовне позначення

Рисунок 3.38 – Схемна реалізація функції порівняння

Схему порівняння двох чотирирозрядних слів А і В згідно з цим виразом показано на рис. 3.39. Схема включає чотири логічних елементи «виключальне АБО» і один кон'юнктор.

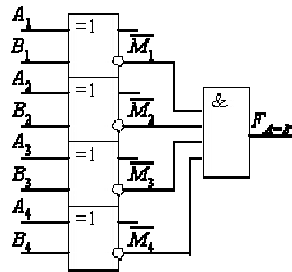
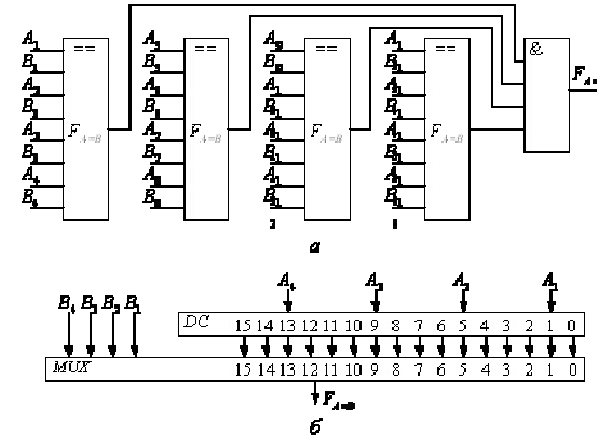


Рисунок 3.39 – Схема порівняння двох чотирирозрядних слів А і В

3.6.1.4 Групові схеми порівняння багаторозрядних слів

Схеми порівняння малої розрядності можна використовувати як групи для побудови схеми порівняння великої розрядності. Наприклад, на першому рівні порівняння застосовують чотирирозрядні групи порівняння рис. 3.39. На другому рівні реалізують загальну ознаку (прапорець) логічним множенням групових ознак.

Зокрема, для розрядності слів $n = 16$ отримаємо чотири групові ознаки порівняння: $F_{A=B}^{1,4}$; $F_{A=B}^{5,8}$; $F_{A=B}^{9,12}$; $F_{A=B}^{13,16}$, де верхні індекси означають номери розрядів у групах. Тоді ознака порівняння двох 16-розрядних слів запишеться у вигляді: $F_{A=B} = F_{A=B}^{1,4} \cdot F_{A=B}^{5,8} \cdot F_{A=B}^{9,12} \cdot F_{A=B}^{13,16}$. Схема порівняння двох 16-розрядних слів показана на рис. 3.40 а.



а – групова структура; б – на основі дешифратора і мультиплексора

Рисунок 3.40– Схема порівняння двох слів на рівність

Існують також інші схемні рішення. Схема порівняння двох чотирирозрядних чисел А і В на основі дешифратора і мультиплексора показана на рис. 3.40 б.

Дешифратор DC виробляє одиничне значення сигналу на тому виході, номер якого також визначається десятковим еквівалентом вхідного коду. Наприклад, при $A_4 A_3 A_2 A_1 = 0111$ логічна одиниця з'явиться на виході з номером сім.

Мультиплексор MUX підключає до виходу той вхід з DC, номер якого також визначається десятковим еквівалентом вхідної комбінації. Якщо $V_4V_3V_2V_1 = 0111$, то дозволяється проходження на вихід сигналу із сьомого входу. Таким чином, якщо слова A і B рівні, то формується прапор $F_{A=B} = 1$. Для решти значень B $F_{A=B} = 0$.

3.6.1.5 Схеми порівняння двох слів «на більше»

Схема порівняння двох слів A і B «на більше» за абсолютним значенням виробляє ознаку $F_{A>B}$. Для аналізу нерівності слів A і B виконують порівняння розрядів послідовно у напрямку від старших розрядів до молодших. Молодші розряди включаються в аналіз у тому випадку, коли старші розряди рівні (еквівалентні). Для отримання ознаки $F_{A>B}$ будують диз'юнктивну суму порозрядних умов.

Для кожної пари розрядів A і B формують сигнал порівняння C_i для ознака $A_i > B_i$; та сигнал рівності r_i до аналізу молодших розрядів обох слів (рис. 3.41).

A_i	B_i	C_i	r_i
0	0	0	1
0	1	0	0
1	0	1	0
1	1	0	1

Рисунок 3.41 – Порівняння по розряду

Звідси отримаємо вираз для C_i

$$C_i = A_i \bar{B}_i; r_i = \bar{A}_i \bar{B}_i \vee A_i B_i = \bar{A}_i \oplus \bar{B}_i = M_i.$$

Тому ознаку $F_{A>B}$ визначають як :

$$F_{A>B} = C_n \vee r_n C_{n-1} \vee \dots \vee r_{n-1} \dots r_2 C_1.$$

Наприклад, для порівняння двох чотирирозрядних слів «на більше» ознаку отримуємо вираз

$$F_{A>B} = C_4 \vee r_4 C_3 \vee r_4 r_3 C_2 \vee r_4 r_3 r_2 C_1 = A_4 \bar{B}_4 \vee \bar{M}_4 A_3 \bar{B}_3 \vee \bar{M}_4 \bar{M}_3 A_2 \bar{B}_2 \vee \bar{M}_4 \bar{M}_3 \bar{M}_2 A_1 \bar{B}_1.$$

Схему порівняння «на більше» для двох чотирирозрядних слів A і B показано на рис. 3.42.

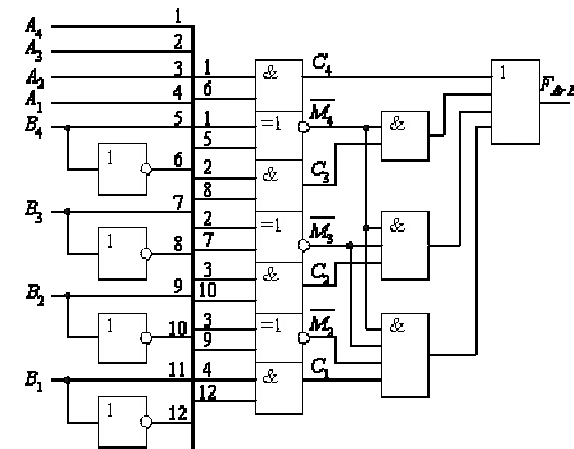


Рисунок 3.42 – Схема порівняння двох слів «на більше»

3.6.1.6 Групові багаторозрядні схеми порівняння «на більше»

При реалізації схем порівняння багаторозрядних слів «на більше» можна розбивати слова на групи, наприклад, з чотирьох розрядів. Кожна група виробляє свою ознаку нерівності $F_{iA>B}$ і умову рівності розрядів чисел.

Наприклад, для $n = 16$ маємо чотири групи, які об'єднуються згідно із співвідношенням

$$F_{A>B}^4 = F_{A>B}^4 \vee M_{16}^4 M_{15}^4 M_{14}^4 M_{13}^4 \vee M_{12}^4 M_{11}^4 M_{10}^4 M_9^4 \vee M_8^4 M_7^4 M_6^4 M_5^4 \vee M_4^4 M_3^4 M_2^4 M_1^4$$

де $F_{A>B}^4$ – прапор порівняння «на більше» в найстаршій групі з розрядами $A_{16} - A_{13}, B_{16} - B_{13}$;

$M^4 = M_{16} M_{15} M_{14} M_{13}$ – умова для підключення до аналізу сусідньої молодшої групи;

$F_{A>B}^3$ – прапор порівняння «на більше» у групі з розрядами $A_{12} - A_9, B_2 - B_9$;

$M^3 = M_{12} M_{11} M_{10} M_9$ – умова аналізу молодшої групи;

$F_{A>B}^2$ – прапор порівняння «на більше» у групі з розрядами $A_8 - A_5, B_8 - B_5$;

$M^2 = M_8 M_7 M_6 M_5$ – умова підключення молодшої групи;

$F_{A>B}^1$ – прапор порівняння «на більше» у групі з розрядами $A_4 - A_1, B_4 - B_1$.

Схему порівняння «на більше» двох 16-розрядних слів A і B показано на рис. 3.43.

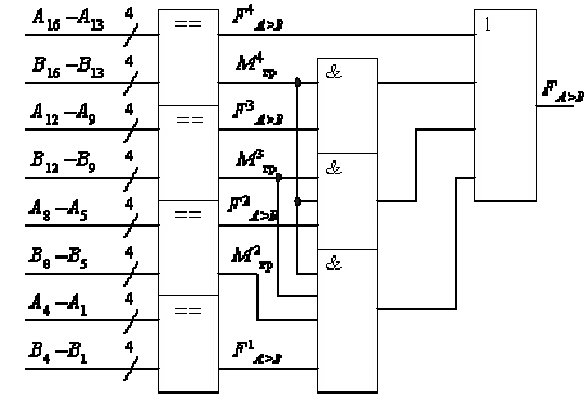


Рисунок 3.43 – Схема порівняння «на більше» двох 16-розрядних слів

3.6.2 Схеми контролю

Контроль (виявлення) і корекція (виправлення) результатів виконання операцій є важливою передумовою надійної роботи комп'ютера. Зазвичай застосовують програмний (обчислення за різними алгоритмами з одержанням однакового результату) або апаратний контроль (шляхом додаткової апаратури).

3.6.2.1 Схеми контролю операцій

До апаратних методів відносяться дублювання операцій з відновленням вхідних сигналів. Зокрема, на рис. 3.44 а подано апаратний метод дублювання операції додавання з допомогою двох (SM), які отримують на входи доданки $A(n)$ і $B(n)$. Результати $S1(n)$ і $S2(n)$ порівнюють за допомогою схеми порівняння. Якщо обидва результати

рівні, то на виході схеми порівняння значення ознаки $F_{S1=S2} = 1$ і помилок немає. При нульовому значенні ознаки F_{S1} результат операції є недовірним, оскільки отримано різні результати.

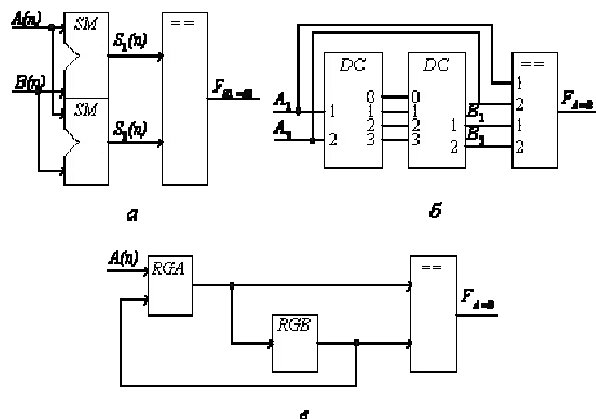


Рисунок 3.44 – Застосування схеми порівняння для контролю операцій

Схема контролю методом відновлення вхідних сигналів показана на рис. 3.44 б. Дворозрядне слово A_2A_1 декодується і значення унітарного коду з виходів дешифратора поступає на входи шифратора. При правильній роботі дешифратора і шифратора вхідний код A_2A_1 має збігатися з вихідним кодом шифратора B_2B_1 . При цьому на виході схеми порівняння встановиться одиничне значення ознаки $F_{A=B}$.

При передачі інформації з одного регістра в інший контроль правильності пересилки може здійснюватися порозрядним порівнянням вмісту цих двох регістрів. На рис. 3.44 в показано один з варіантів контролю пересилок слів між регістрами. Після передачі інформації з

регістра А в регістр В (або навпаки) проводиться порівняння їх вмісту. Якщо значення двох слів збігаються, то значення ознаки рівності набуває одиничного значення, інакше – виробляється сигнал помилки.

3.6.2.2 Схеми контролю парності

У комп'ютерах широко використовують контроль парності (за паритетом) для виявлення одиничної помилки в одному двійковому розряді (наприклад, втрата або поява зайвої одиниці). Помилка призводить до зміни парності кількості одиниць (за парністю або непарністю).

При контролі за парністю значення контрольного розряду $F_{КП}$ вибирається таким, щоб загальне число одиниць у байті й контрольному біті було парним

$$F_{КП} = A_1 \oplus A_2 \oplus A_3 \oplus A_4 \oplus A_5 \oplus A_6 \oplus A_7 \oplus A_8.$$

Внаслідок операції додавання за модулем два значень розрядів байта з парним числом одиниць одержуємо значення контрольного байта $F_{КП} = 0$. При додаванні за модулем два значень розрядів байта з непарним числом одиниць значення контрольного байта $F_{КП} = 1$.

При контролі за непарністю значення контрольного біта $F_{КП}$ визначають як

$$F_{КП} = \overline{A_1 \oplus A_2 \oplus A_3 \oplus A_4 \oplus A_5 \oplus A_6 \oplus A_7 \oplus A_8} = \overline{F_{КП}}$$

Схеми контролю непарності використовують частіше ніж схеми контролю за парністю. Приклад реалізації схеми контролю за парністю

наведено на рис. 3.11. Такі схеми часто називають схемами згортки, схемами контролю за модулем два, схемами контролю за паритетом.

Для контролю пам'яті при записуванні байта в пам'ять комп'ютера одночасно формується (генерується) значення його контрольного розряду. При зчитуванні байта, що зберігається у пам'яті, здійснюється додавання за модулем два значень його розрядів спільно з контрольним бітом згідно з визначеним способом контролю парності або непарності.

Для визначення ознаки контролю вводять керувальний сигнал V , який разом з сигналом M надходить на входи схеми «виключальне АБО» в четвертому рівні; на прямому й інверсному виходах цього рівня формуються пряме й інверсне значення контрольного розряду: $F = M \oplus V$; $\bar{F} = \overline{M \oplus V}$. Режим роботи схеми контролю, показаної на рис. 3.45 а, наведено на рис. 3.46.

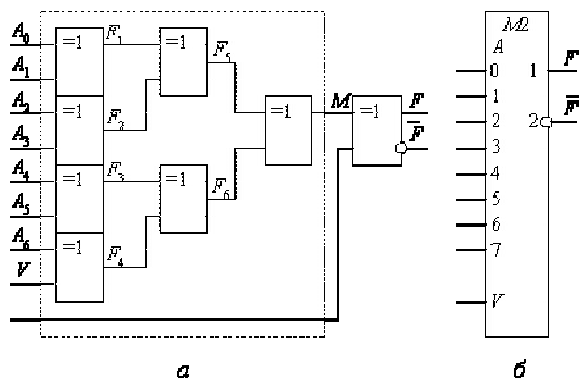


Рисунок 3.45 – Схеми контролю за парністю: а – ступінчатє включення елемента «виключальне АБО»; б – умовне позначення

Входи A8 – A1		V	F	\bar{F}
На входах:	Парне число одиниць	0	0	1
	Непарне число одиниць	0	1	0
На входах:	Парне число одиниць	1	1	0
	Непарне число одиниць	1	0	1

Рисунок 3.46 – Режим роботи схеми контролю

Із рис. 3.46 видно, що при $V = 0$ на виході F генерується значення контрольного розряду для контролю парності, при $V = 1$ - непарності.

Приклад схеми контролю непарності пересилок байта від джерела інформації (ДІ) до приймача інформації (ПІ) показана на рис. 3.47.

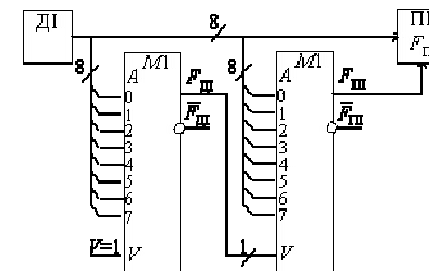


Рисунок 3.47 – Контроль пересилок байта

Схема контролю з боку джерела інформації виступає як генератор значення контрольного розряду непарності $F_{ДІ}$. Схема контролю з боку приймача інформації забезпечує додавання за модулем два значень розрядів визначеного байта спільно з визначеним контрольним бітом непарності. Прийом інформації можливий тільки при виконанні умови непарності $F_{ПІ} = 1$ з боку приймача.

4 ПОСЛІДОВНІСНІ СХЕМИ

У послідовнісних схемах, на відміну від комбінаційних схем, стани сигналів на виходах схеми залежить не тільки від стану сигналів на входах схеми, але й від внутрішнього стану схеми. Існує значна кількість послідовнісних схем. Серед типових з них можна виділити тригери, регістри, лічильники, послідовні суматори. Послідовнісні схеми є основою побудови схем керування, мікропроцесорів, мікропрограмних автоматів.

4.1 Тригери

Тригер – це запам'ятовувальний елемент з двома стійкими станами, зміна яких відбувається під дією вхідних сигналів. На основі тригерів будують типові функціональні вузли комп'ютерів – регістри, лічильники, накопичувальні суматори, а також мікропрограмні автомати.

4.1.1 Класифікація тригерів

Усі різновиди тригерів являють собою елементарний автомат, який вміщує власне елемент пам'яті (ЕП) та схему керування (СхК), яка утворює вхідну логіку (рис.4.1).

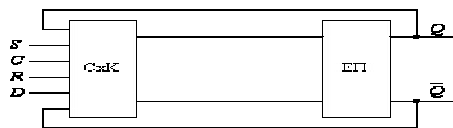


Рисунок 4.1 – Структура тригера

Стан тригера визначається сигналами на прямому Q інверсному \bar{Q} виходах. При позитивному кодуванні сигналів схеми тригера високий рівень напруги на прямому виході відображає значення логічної одиниці (стан $Q = 1$), а низький рівень – значення логічного нуля (стан $Q = 0$). Сигнали на входах тригера часто позначають латинськими буквами R, S, T, C, V тощо, які визначають тип тригера.

Тригери класифікують за ознаками: логікою функціонування (RS, JK, D, T тощо); способом записування інформації (асинхронні й синхронні); моментом реакції на тактовий сигнал (статичні, динамічні); кількістю тактів синхронізації (одно-, дво- і тритактові); кількістю ступенів (одно- або двоступеневі тригери); складом логічних елементів (тригери на елементах І-НЕ, АБО-НЕ тощо.). Розрізняють тригери: з роздільною установкою станів “0” і “1” (RS-тригери); з одним інформаційним входом (D-тригери); з лічильним входом (T-тригери); універсальні з роздільною установкою станів “0” і “1” (JK-тригери); комбіновані (RST-, RSJK-тригери); із складною вхідною логікою. Деякі з цих тригерів подано на рис. 4.2.

Входи тригерів розділяються на інформаційні (R, S, T тощо) та керувальні (C, V). Інформаційні (логічні) входи призначені для приймання сигналів інформації, яка запам'ятовується. Керувальні входи призначені для керування записуванням інформації. У тригерах може бути два види керувальних сигналів: сигнал синхронізації (тактовий сигнал) C, який надходить до C-входу (тактового входу) і визначає момент виконання операції запису чи зміни стану; і дозвільний сигнал V, який надходить до V-входу і часто визначає режим підключення схеми тригера.

За способом записування (приймання) інформації розрізняють асинхронні й синхронні (тактовні) тригери. Тригери, які не мають С-входу, називають асинхронними (рис.4.2, а і б). В асинхронних тригерах записування інформації відбувається в будь-який момент часу при надходженні сигналів до інформаційних входів. Тригери, які мають С-вхід, називаються синхронними. У синхронному тригері записування інформації можливе при збігу сигналів на інформаційному й синхронному входах.

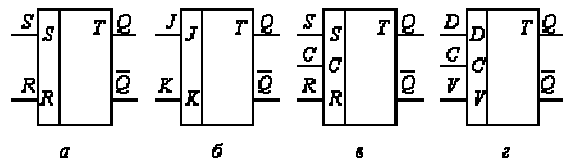


Рисунок 4.2 – Умовні позначення тригерів: а, б – асинхронних; в, г – синхронних

До V-входів тригера надходять сигнали, які дозволяють ($V = 1$) або забороняють ($V = 0$) записування інформації. У синхронних тригерах з V-входом записування інформації можливе при збігу сигналів на інформаційному, С- і V- виходах (рис.4.2, г).

Залежно від кількості тактових сигналів, необхідних для формування нового стану, розрізняють однотокові, двотокові та багатотокові тригери.

За способом керування записуванням (моментом реакції на тактовий сигнал) виділяють синхронні тригери зі статичним (за рівнем), динамічним (за фронтами) та двоступеневим керуванням.

В асинхронних тригерах записування нуля і одиниці можливе у будь-який момент часу. При цьому вхідний інформаційний сигнал одночасно є й керувальним.

У синхронних тригерах з керуванням за рівнем записування інформації можливе тільки впродовж тривалості тактового сигналу. При цьому тактові сигнали можуть бути прямими (змінюватися від нуля до одиниці) або інверсними (змінюватися від одиниці до нуля) (рис.4.3, а і б).

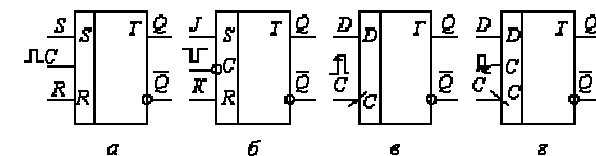


Рисунок 4.3 – Керувальні входи тригера: а – прямий статичний; б – інверсний статичний; в – прямий динамічний; г – інверсний динамічний

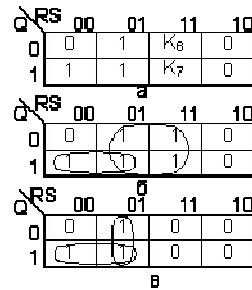
При керуванні фронтами дозвіл на записування інформації дається тільки в момент перепаду тактового сигналу від нуля до одиниці (прямий динамічний вхід) або від одиниці до нуля (інверсний динамічний вхід). В інші моменти часу тригер не реагує на вхідні інформаційні сигнали незалежно від рівня тактового імпульсу (рис.4.3, в і г).

4.1.2 Таблиця переходів і логічні рівняння RS-тригера

Назва “RS-тригер” утворена від перших літер слів RESET (скидання, відключення) і SET (установлення) (рис. 4.4). Стани K6, K7 тригера є невизначеними, оскільки непритустими є вхідні сигнали Rt і St, які одночасно мають значення одиниці (заборонена комбінація сигналів).

R_t	S_t	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	K_6
1	1	1	K_7

Таблиця переходів RS-тригера



Карти Карно для RS-тригерів

Рисунок 4.4 – Таблиця переходів і карти Карно для RS-тригерів

Таблиці переходів відповідає карта Карно (рис.4.4 а), де значення функції Q_{t+1} для мінтермів R_t, S_t, \bar{Q}_t , і R_t, S_t, Q_t замінено невизначеними значеннями K_6 і K_7 . Після їх довизначення і врахувавши, що комбінації вхідних сигналів $R_t S_t = 1$ не існує, отримаємо карти Карно для $K_6 = K_7 = 1$ (рис.4.4 б) і $K_6 = K_7 = 0$ (рис.4.4 в). Звідси отримуємо логічні рівняння асинхронного RS-тригера:

$$K_6 = K_7 = 1, \quad Q_{t+1} = S_t \vee \bar{R}_t \bar{Q}_t, \quad (4.1)$$

$$K_6 = K_7 = 0, \quad Q_{t+1} = \bar{R}_t (S_t \vee Q_t). \quad (4.2)$$

Логічні вирази визначають новий стан тригера Q_{t+1} залежно від старого стану Q_t та вхідних сигналів R_t і S_t .

4.1.3 Асинхронний RS-тригер на елементах І-НЕ .

Вираз (4.1) зручно реалізувати на елементах І-НЕ після перетворень:

$$Q_{t+1} = \overline{S_t \vee \bar{R}_t} \cdot \overline{\bar{S}_t \cdot \bar{R}_t} \cdot Q_t. \quad (4.3)$$

Із аналізу діаграм роботи RS-тригера випливає, що елементи І-НЕ в схемі перемикаються послідовно. Є інтервал часу, коли на обох виходах встановлюються однакові сигнали $Q = 1$ і $\bar{Q} = 1$ (рис. 4.5, в, заштриховані області) – явище “ризик”.

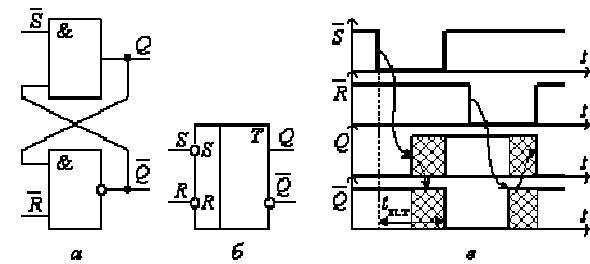


Рисунок 4.5 – Асинхронний RS-тригер на елементах І-НЕ: а – схема; б – умовне позначення; в – часові діаграми

4.1.4 Асинхронний RS-тригер на елементах АБО - НЕ.

Вираз (4.2) можна перетворити до зручної реалізації на елементах АБО НЕ:

$$Q_{n+1} = \overline{R(S \vee Q)} \vee \overline{R \vee (S \vee Q)} \quad (4.4)$$

Схему асинхронного RS-тригера на двох елементах АБО - НЕ з логічними зв'язками на основі виразу (4.4) подано на рис.4.6 а.

Інтервал часу, коли на обох виходах установлюються однакові сигнали $Q = 0$ і $\overline{Q} = 0$, визначає нестабільний інтервал ("ризик") (рис.4.6 в).

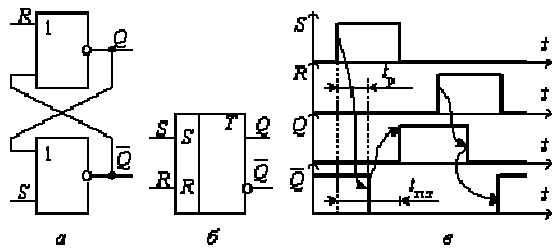


Рисунок 4.6 – Асинхронний RS-тригер на елементах АБО-НЕ: а – схема; б – умовне позначення; в – часові діаграми

4.1.5 Синхронні RS-тригери

Асинхронні RS- тригери можна перетворити на синхронні, включивши сигнали синхронізації С.

Для побудови синхронного RS-тригера на елементах І - НЕ треба замінити в логічному виразі (4.3) змінні S і R на синхронізовані сигнали CS і CR, де С – синхросигнал:

$$Q_{n+1} = \overline{CS} \overline{CR} Q \quad (4.5)$$

Схему синхронного RS-тригера на чотирьох елементах І - НЕ з логічними зв'язками на основі виразу (4.5) подано на рис.4.7 а. Елементи D1 і D2 утворюють схему керування з прямими входами, а елементи D3 і D4 утворюють асинхронний RS-тригер.

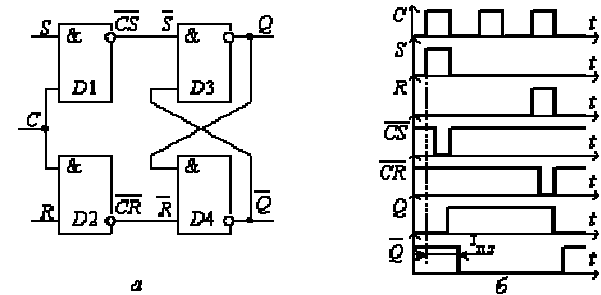


Рисунок 4.7 – Синхронний RS-тригер на елементах І - НЕ : а – схема; б – часові діаграми

За умови $CS = 1$ на виході елемента D1 встановлюється логічний нуль і тригер перемикається у стан $Q=1$. При значенні сигналів $CR = 1$ на виході елемента D2 встановлюється логічний нуль і тригер перемикається в стан $Q=0$.

Комбінація вхідних сигналів $CSR = 1$ є забороненою, оскільки призводить до невизначеного стану тригера. Із часової діаграми (рис.4.7 б) випливає, що час перемикання тригера $t_{п.т} = 3t_p$, а тривалість синхросигналу (з урахуванням запасу на одну затримку) визначається з

умови $t_c = 4t_p$. Максимальна і робоча частоти перемикання тригера відповідно дорівнюють: $f_{max} = 1/3t_p$ і $f_p = 1/4t_p$.

Для побудови синхронного RS-тригера на елементах АБО - НЕ слід замінити в логічному виразі (4.4) змінні S і R на \overline{CS} і \overline{CR} :

$$Q_{n+1} = \overline{\overline{CR} \vee (\overline{CS} \vee Q)} \cdot \overline{\overline{C} \vee \overline{R} \vee (\overline{C} \vee \overline{S} \vee Q)} \quad (4.6)$$

Схема синхронного RS-тригера на чотирьох елементах АБО-НЕ з логічними зв'язками на основі виразу (4.6) показана на рис.4.8. Елементи D1 і D2 складають схему керування з інверсними входами, а елементи D3 і D4 утворюють фіксатор (асинхронний RS-тригер).

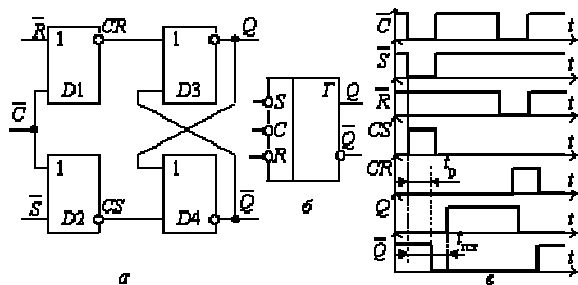


Рисунок 4.8 – Синхронний RS-тригер на елементах АБО-НЕ: а – схема; б – умовне позначення; в – часові діаграми

При значенні сигналів $\overline{C} = 0$ і $\overline{S} = 0$ на виході елемента D2 встановлюється логічна одиниця (тобто $CS = 1$) і тригер переключиться в стан $Q=1$. При значенні сигналів $\overline{C} = 0$ і $\overline{R} = 0$ на виході елемента D1 встановлюється логічна одиниця (тобто $CR = 1$) і тригер переключиться в

стан $Q=0$. Комбінація сигналів $\overline{C} = \overline{S} = \overline{R} = 0$ заборонена, тому що призводить до невизначеного стану тригера.

4.1.6 Двоступеневі RS-тригери

Двоступеневі тригери будують за способом “M–S” (Master-Slave) і забезпечують поєднання двох процесів – одночасного записування нової інформації та зчитування старої. Під час дії синхроімпульсу C перший ступінь “M” (Master – основний) приймає нову вхідну інформацію, а другий ступінь “S” (Slave – допоміжний) в цей же час передає у зовнішні схеми стару інформацію. Після закінчення синхроімпульсу C інформація з першого ступеня переписується у другий ступінь.

При однофазному (однотактному) обміні інформацією зв'язок між ступенями реалізується за допомогою інвертора (рис.4.9 а), заборонених зв'язків (рис.4.9 б) або різнополярного керування (рис.4.9 в). При двотактному обміні зв'язок між ступенями забезпечується двома серіями синхросигналів – C1 і C2 (рис.4.9 г).

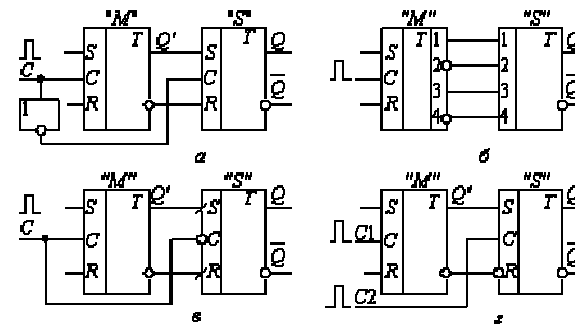


Рисунок 4.9 – Організація зв'язку між ступенями тригера: а – з інвертором; б – із забороняючими зв'язками; в – з різнополярним керуванням; г – з двофазним обміном

4.1.7 Тригери типу JK

Тригером типу JK називають запам'ятовувальний елемент з двома станами інформаційними входами J (аналог S) і K (аналог R). JK -тригер описують таблицею переходів рис. 4.10.. Тригер є подібним до RS-тригера, але за умови збігу сигналів JK = 1 він працює як однорозрядний двійковий лічильник, тобто переключасться у стан, протилежний початковому стану.

Тому тригер типу JK є універсальним, оскільки може виконувати функції RS-тригера (при роздільному надходженні сигналів J і K), T-тригера (при одночасній подачі сигналів J і K), D-тригера (при подачі сигналу від входу J через інвертор на вхід K).

З карти Карно даного тригера (рис.4.10) отримуємо рівняння

$$Q_{n+1} = \overline{K} \cdot Q \vee J \cdot \overline{Q} \quad (4.7)$$

Для побудови одноступеневого синхронного JK-тригера на елементах І-НЕ потрібно замінити в рівнянні (4.7) змінні K і J на сигнали СК і JK. Після перетворення на основі правил подвійної інверсії та правил де Моргана:

$$Q_{n+1} = \overline{\overline{C} \cdot \overline{K} \cdot Q \vee \overline{C} \cdot J \cdot \overline{Q}} = \overline{\overline{C} \cdot \overline{K} \cdot Q} \cdot \overline{\overline{C} \cdot J \cdot \overline{Q}} \quad (4.8)$$

Таблиця 3.2

K _t	J _t	Q _t	Q _{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

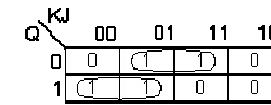


Рис.2.11. Карта Карно для JK-тригера

Рисунок 4.10 – Таблица переходів одноступеневого JK-тригера

Схему JK-тригера з логічними зв'язками на основі рівняння (4.8) показано на рис.4.11.

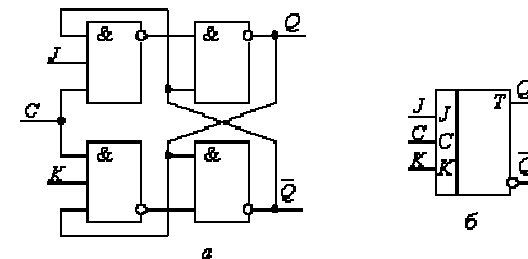


Рисунок 4.11 – Одноступеневий JK-тригер: а – схема; б – умовне позначення

Двоступеневий синхронний JK-тригер на елементах І-НЕ подано на рис.4.12. Нова інформація знімається з виходів Q основного M- ступеня, а стара – з виходів Q* допоміжного S-ступеня.

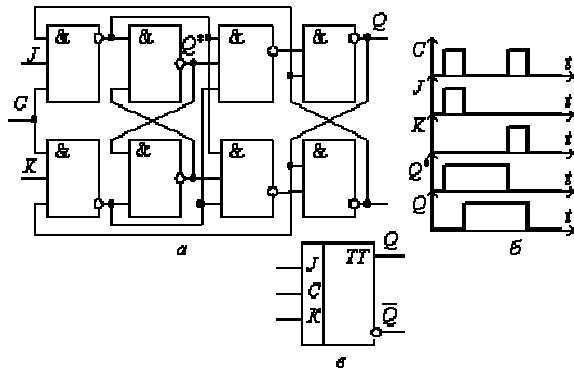


Рисунок 4.12 – Двоступеневий JK-тригер на елементах І-НЕ: а – схема; б – часові діаграми; в – умовне графічне позначення

Із часових діаграм (рис.4.12 б) випливає, що при застосуванні JK-тригера для зберігання інформації сигнали на входи J і K подають почергово; а при збігу сигналів на входах J і K реалізують лічильний тригер. Час перемикання JK-тригера визначається сумою затримок першого і другого ступенів.

4.1.8 T-тригери

Асинхронний тригер типу T є запам'ятовувальним елементом з двома стійкими станами та одним інформаційним T-входом. Стан T-тригера змінюється на протилежний після кожного надходження лічильного сигналу на T-вхід.

Функціонування асинхронного T - тригера можна подати таблицею переходів (рис. 4.13) і описати логічним рівнянням

$$Q_{n+1} = \bar{T} \cdot Q_n \vee T \cdot \bar{Q}_n \quad (4.9)$$

Звідси отримусмо реалізацію T-тригера на елементах І - НЕ перетворення рівняння (4.9):

$$Q_{n+1} = \overline{\overline{\bar{T} \cdot Q_n} \vee \overline{T \cdot \bar{Q}_n}} \quad (4.10)$$

Для виключення інверсії сигналу T у рівнянні (4.10) використовують тотожність $\bar{T} \cdot Q_n = (\bar{T} \cdot \bar{Q}_n) \cdot Q_n$. Перемикання тригера визначається сумісною дією лічильних сигналів T і зворотного зв'язку виходів Q і \bar{Q} .

Затримка вихідного сигналу може здійснюватися лінією затримки (в імпульсно-потенціальній системі елементів) чи додатковим тригером (в потенціальній системі елементів).

Схему одноступеневого асинхронного T-тригера на елементах І-НЕ з логічними зв'язками відповідно до рівняння (4.10) показано на рис. 4.14. Сигнали з виходів елементів D1 і D2 затримуються на час Δt , що дорівнює тривалості лічильного сигналу на T-вході.

T _t	Q _t	Q _{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

Рисунок 4.13 – Таблиця переходів T-тригера

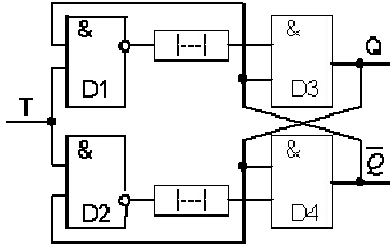


Рисунок 4.14 – Схема Т-тригера

4.1.9 D-тригери

Синхронний тригер типу D є запам'ятовувальним елементом з двома стійкими станами і одним інформаційним (Date) D-входом. Функціонування D-тригера описують логічним рівнянням

$$Q_{t+1} = C_t D_t.$$

Після перемикання стан D-тригера визначає значення сигналу на D-вході в тактові моменти часу за синхросигналом C_t . Тому D-тригери часто називають тригерами затримки (від слова Delay – затримка).

Схему D-тригера можна побудувати на основі синхронного RS-тригера, якщо сигнал по входу S одночасно подавати через інвертор на вхід R (рис. 4.15 а). Перетворимо рівняння (4.5) замінивши сигнал S на D і сигнал R на \bar{D} :

$$Q_{t+1} = \overline{\overline{C} \cdot \overline{S} \cdot \overline{C} \cdot \overline{R}} \cdot \overline{\overline{C} \cdot \overline{D} \cdot \overline{C} \cdot \overline{D}} \cdot Q_t \quad (4.11)$$

Схема D-тригера на елементах І-НЕ з логічними зв'язками згідно з рівнянням (4.11) показана на рис. 4.15 б. D-тригер “слідкує” за зміною сигналу на D-вході під час дії синхросигналу C і зберігає ту інформацію, яка була в момент його закінчення. RS-тригери такої властивості не мають і тому вони менше завадостійкі порівняно з D-тригерами.

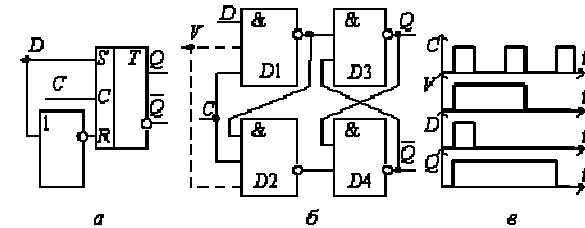


Рисунок 4.15 – D-тригер: а – на основі RS-тригера; б – на елементах І-НЕ; в – часові діаграми роботи

4.1.10 DV-тригер

Для затримки інформації в D-тригері на довільне число тактів використовують дозвільний V-вхід, як показано штриховою лінією на рис. 4.15 б.

Якщо $V = 1$, то DV-тригер функціонує як звичайний тригер затримки. За умови $V = 0$, то робота схеми за входами блокується і DV-тригер зберігає попередню інформацію.

Схема двоступеневого одноктактного DV-тригера на елементах І-НЕ із забороняючими зв'язками між ступенями показана на рис. 4.16.

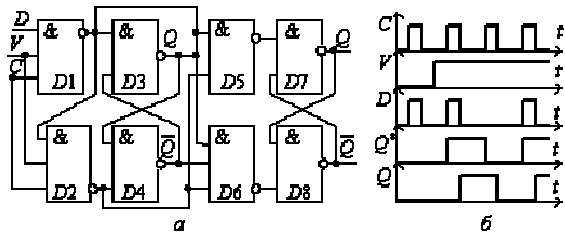


Рисунок 4.16 – Двоступеневий DV-тригер: а – схема; б – часова діаграма

4.1.11 Інші типи тригерів

Існує досить багато інших типів тригерів, які розрізняються складом інформаційних і керувальних входів, а також режимами роботи. Зокрема, у S – тригерах і R – тригерах тригер RS довізначають для станів K6, K7 (рис. 4.4) відповідно як для сигналів SET і RESET.

У реальних мікросхемах тригерів застосовують кілька тактових і дозвільних входів, на яких реалізують певну логіку вибору різних режимів роботи. Це розширює функціональність тригера.

4.2 Регістри

Регістр являє собою пристрій з набором двійкових ланок (тригерів з керуючими елементами), головним призначенням якого є зберігання інформації у вигляді багаторозрядних двійкових чисел (двійкового коду). На відміну від пристроїв довготривалої пам'яті в регістрах інформація запам'ятовується короткочасно, тобто на період одного або кількох циклів роботи всієї системи.

4.2.1 Основні операції регістра

Регістри призначені для запису, зберігання і читання одного двійкового числа або іншої кодової комбінації. Крім цих основних операцій регістри виконують додаткові операції: інвертування коду, скидання в нульовий стан, перетворення послідовного коду в паралельний і навпаки.

В загальному випадку регістри забезпечують виконання наступних мікрооперацій:

- установка регістра в нуль(скидання, гасіння);
- прийом слова з іншого регістра, лічильника тощо;
- передача слів на інший регістр, лічильник тощо;
- перетворення кодів збережених слів в інверсні коди;
- зсув слова вліво або вправо на необхідне число розрядів;
- перетворення послідовного коду в паралельний і навпаки.

4.2.2 Склад і типи регістрів

Схеми конкретних регістрів можуть реалізувати лише деякі з перелічених мікрооперацій.

Запам'ятовуючі елементи регістру за заданою кількістю розрядів двійкового числа виготовляють на основі RS-, D-, JK-тригерів. Для допоміжних операцій (введення до регістру або виведення з нього числа, яке зберігається, перетворення коду двійкового числа, зсуву числа на певне число розрядів вліво або вправо) застосовують комбінаційні схеми на основі логічних елементів.

Залежно від способу запису інформації регістри поділяють на 2 типи

- 1) регістри паралельного типу (без зсуву);
- 2) регістри послідовного типу (зі зсувом);
- 3) комбіновані регістри.

4.2.3 Регістри паралельного типу

В паралельних регістрах без зсуву інформація (двійкові числа - слова) записуються одночасно до всіх розрядів (паралельний код). Паралельним регістром називають такий регістр, що реалізує всі перераховані мікрооперації крім зсуву й перетворення послідовного коду в паралельний і навпаки. Якщо в паралельному регістрі на вхід кожного каскаду інформація надходить по двох каналах у парафазному коді, то такий регістр називають парафазним.

При наявності тільки одного каналу (прямого або інверсного) надходження інформації в кожному розряді регістра називають однофазним. Загальна схема паралельного регістру зображена на рис. 4.17.

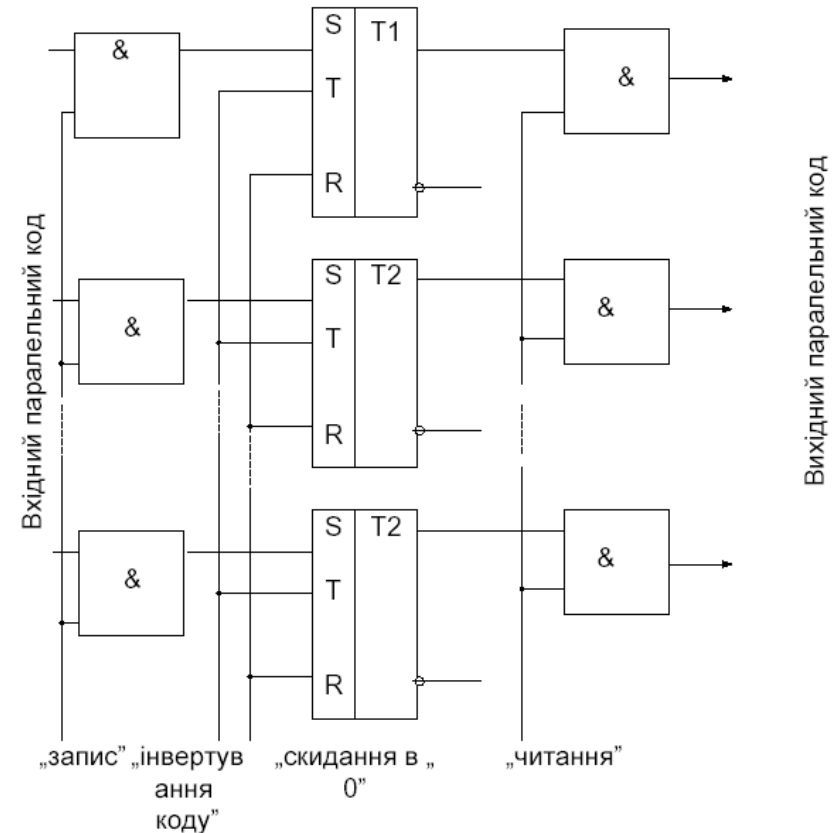


Рисунок 4.17 - Загальна схема паралельного регістру

Схема паралельного однофазного регістра, що виконує перші дві мікрооперації з наведеного вище списку, показана на рис. 4.18. Тут у тих розрядах, де $X_i = 1$ відбудеться установка тригерів в одиничний стан. Там де $X_i = 0$, стан тригерів не змінюється.

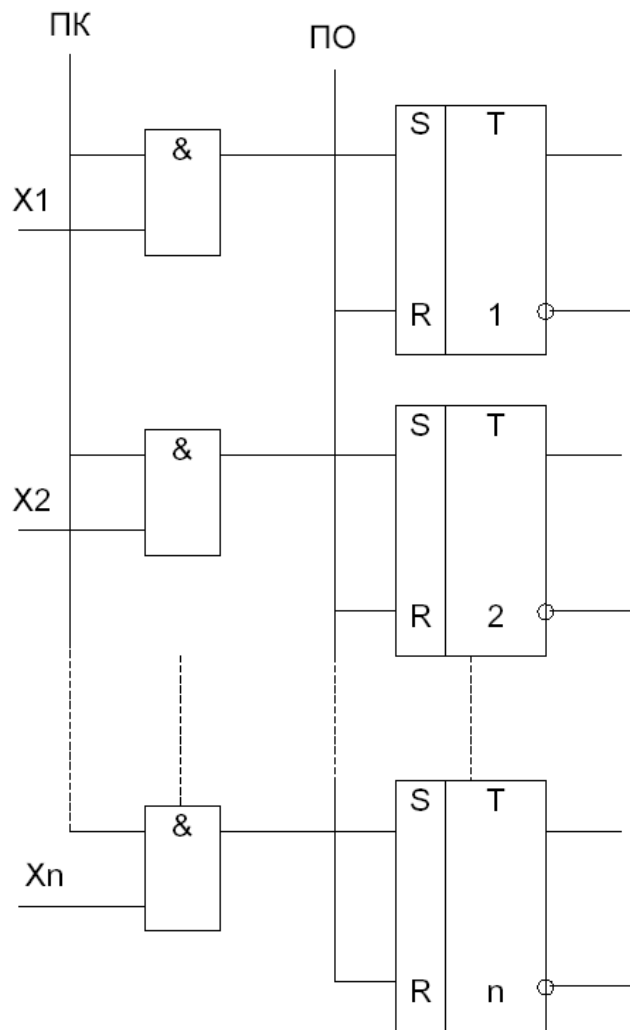


Рисунок 4.18 - Схема паралельного однофазного регістра на RS-тригерах

Видача інформації з регістра може відбуватися в прямому, інверсному й парафазному кодах. Схема видачі інформації в прямому й інверсному кодах показана на рис. 4.19.

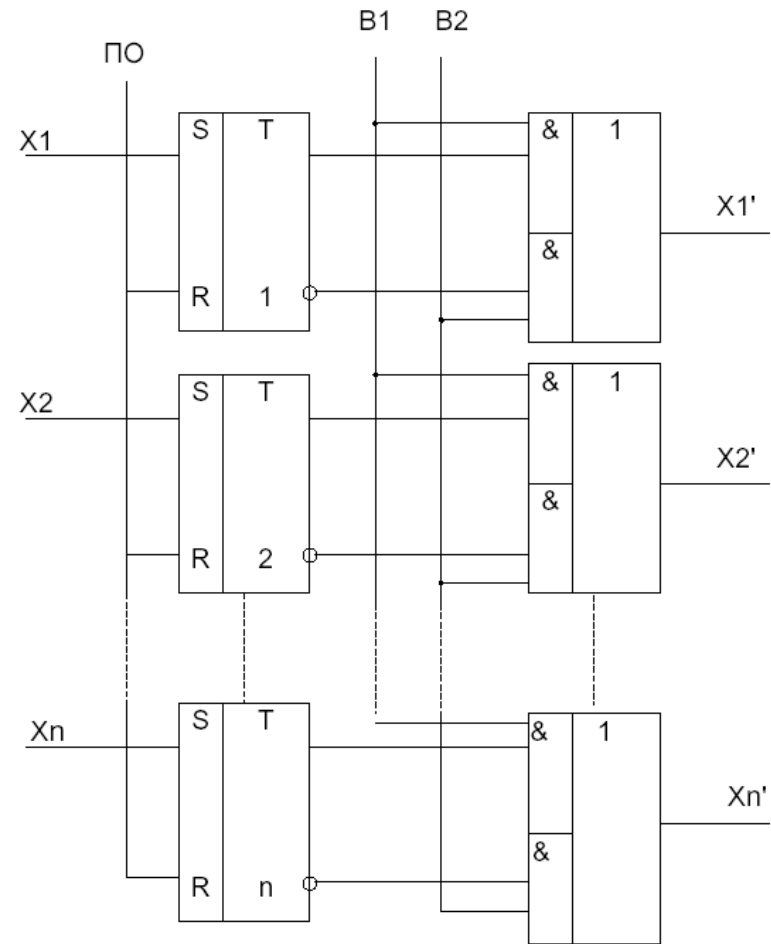


Рисунок 4.19 - Схема видачі інформації в прямому й інверсному кодах

Тут В1 - сигнал видачі прямого коду, В2 - сигнал видачі інверсного коду. Одночасна поява сигналів В1 й В2 заборонена.

Схема однофазного паралельного регістра на тактованих D-тригерах показана на рис. 4.20.

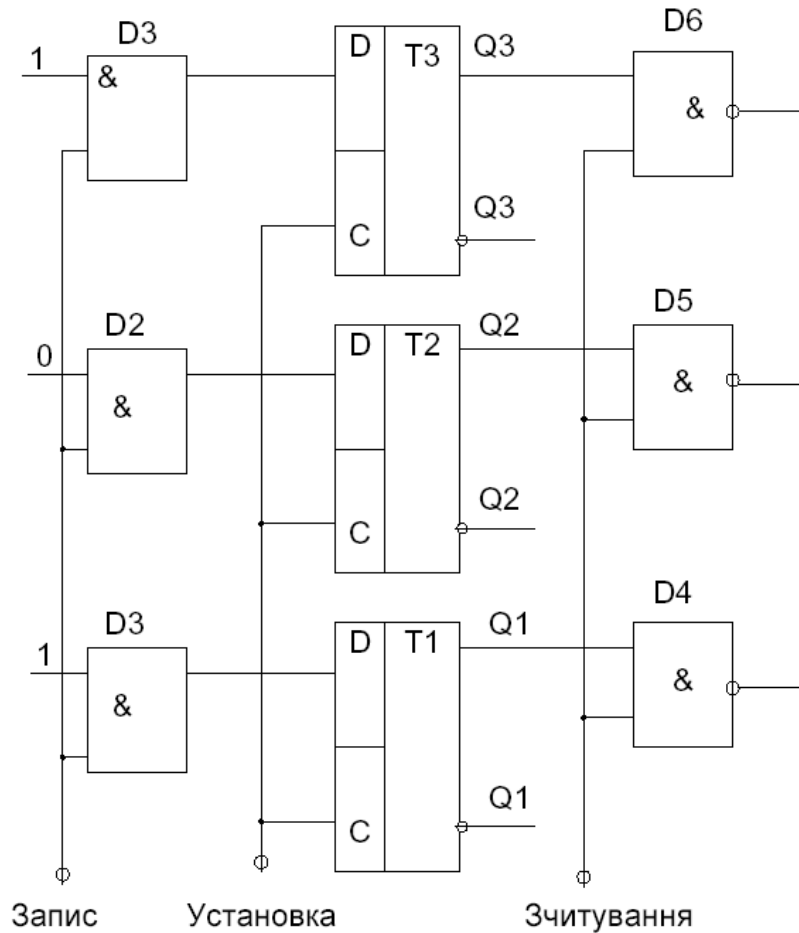


Рисунок 4.20 - Схема однофазного паралельного регістра на тактованих D-тригерах

Кількість тригерів відповідає числу розрядів регістра. Кожний тригер зберігає код одного розряду числа, що запам'ятовується. В розглянутому випадку регістр зберігає код трирозрядного двійкового числа. Крім тригерів, у схемі регістра є схема вводу (запису) інформації на логічних елементах I (мікросхеми D1-D3) і схема виводу інформації однофазним способом в прямому коді на логічних елементах I-II (мікросхеми D4-D6).

4.2.4 Регістри послідовного типу

Загальна структурна схема регістру послідовного типу (із зсувом) зображена на рис. 4.21

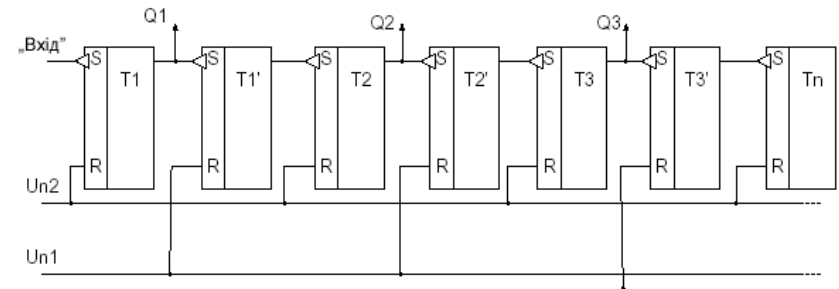


Рисунок 4.21- Загальна структурна схема регістру послідовного типу

4.2.6 Генератори псевдовипадкових чисел на регістрах

4.2.5 Перетворення послідовного коду у паралельний

Схема чотирьохрозрядного регістра зсуву вправо на JK-тригерах, яка забезпечує перетворення кодів, показана на рис. 4.22 а. Старший розряд регістра за допомогою інвертора на К-вході працює в режимі D-тригера.

Нехай від накопичувачів на магнітних дисках або стрічках на вхід регістра по лінії D надходить послідовний код слова $A=1101$ у напрямку від молодших розрядів до старших.

Значення розрядів слова надходить одночасно із синхроімпульсами, які забезпечують як приймання коду в старший розряд, так і одночасний зсув вмісту регістра вправо (рис. 4.22 б)

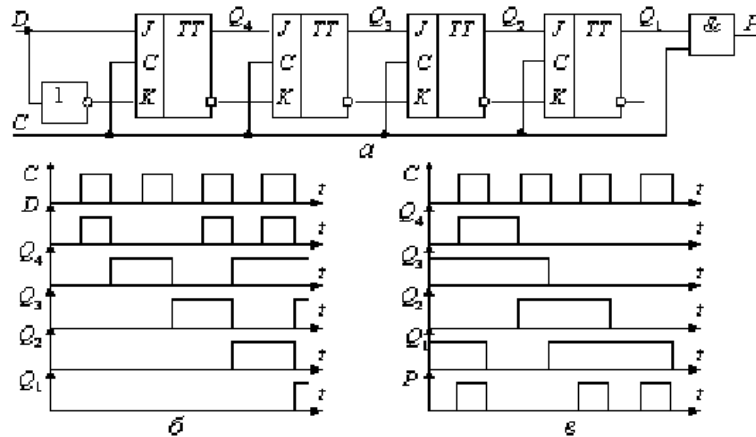


Рисунок 4.22 - Регістр зсуву: а) . схема ; б) . перетворення послідовного коду в паралельний; в) . перетворення паралельного коду у послідовний

Перетворення паралельного коду у послідовний на вихід Р (рис. 4.22 в) виконують зсувом вмісту регістра вправо за тактовим сигналом С.

Генератори псевдовипадкових чисел дозволяють відтворити довільну послідовність чисел заданої розрядності. Ця послідовність чисел може бути задана графом переходів, який визначає стани розрядів регістра.

Розглянемо синтез генератора псевдовипадкових чисел на регістрі. Нехай необхідно створити генератор послідовності значень $0 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 2 \rightarrow 4$.

Для реалізації такого пристрою достатньо використати трьохрозрядний регістр. Граф можливих переходів станів регістра за станами тригерів Q1, Q2, Q3 наведено на рис. 4.23:

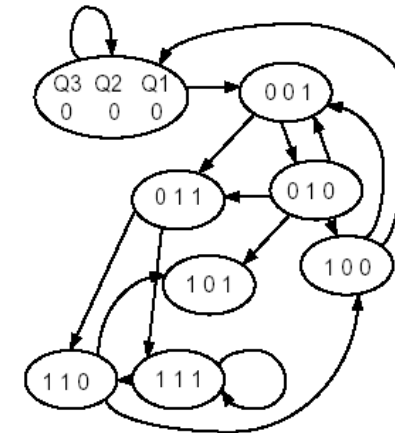


Рисунок 4.23 - Граф переходів

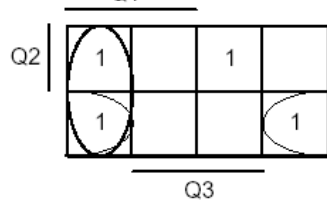
Для побудови генератора необхідно побудувати таблицю переходів для станів регістра (рис. 4.24).

Q3	Q2	Q1	Q3 ^{t+1}	Q2 ^{t+1}	Q1 ^{t+1}	Перехід
0	0	0	0	0	1	↑
0	0	1	0	1	1	1
0	1	1	1	1	1	1
1	1	1	1	1	0	↓
1	1	0	1	0	1	↑
1	0	1	0	1	0	↓
0	1	0	1	0	0	0
1	0	0	0	0	0	0

Рисунок 4.24 – Таблиця переходів для станів регістра

Вихідна функція регістра (рис. 4.25) дозволяє побудувати генератор псевдовипадкових чисел (рис. 4.26)

$$F = \overline{Q1}Q2Q3 \vee Q1\overline{Q2}Q3 \vee Q1Q2\overline{Q3} \vee \overline{Q1}Q2Q3 ,$$



$$D = Q1\overline{Q3} \vee \overline{Q2}Q3 \vee \overline{Q1}Q2Q3 ,$$

Рисунок 4.25 – Таблиця переходів для станів регістра

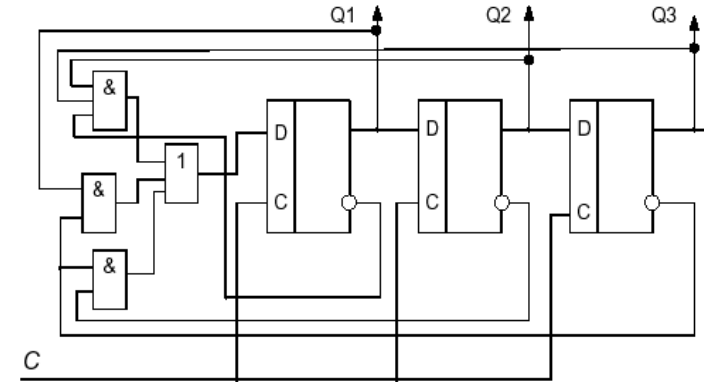


Рисунок 4.26 – Генератор псевдовипадкових чисел на базі регістра

4.2.7 Синтез регістрів на асинхронних тригерах

Синтез регістрів полягає у визначенні переліку мікрооперацій (МО), які він реалізує, визначенні типів тригерів для синтезу і побудові функцій збудження тригерів для реалізації мікрооперацій.

Часто для проектування регістрів використовують асинхронні тригери, зокрема RS-тригери і D-тригери, подані на рис. 4.27.

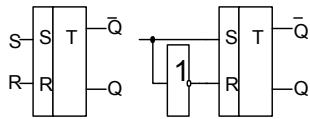


Рисунок 4.27 – RS-тригер і D-тригер

Методика регістру полягає у виконанні наступних кроків.

1. Побудувати таблицю функцій збудження заданого типу тригера.
2. Побудувати повну таблицю переходів і-ого розряду регістра при виконанні заданої операції.
3. Заповнити значення функції збудження та подати їх у відповідній операторній формі згідно із заданим елементним базисом.
4. За отриманою формою побудувати схему регістра.

Розглянемо деякі приклади синтезу регістрів для заданих мікрооперацій.

Задача. Побудувати чотирьохрозрядний регістр на асинхронних тригерах RS, для виконання на ньому мікрооперації кон'юнкції AND (&). Комбінаційну схему (КС) побудувати з використанням елементів 2 АБО-НЕ.

$$\text{AND} : Q_i(t+1) = Q_i(t) \& X_i$$

Побудуємо функцію збудження RS – тригера (рис. 4.28)

Q (t)	Q (t+1)	Fr	Fs
0	0	*	0
0	1	0	1
1	0	1	0
1	1	0	*

Рисунок 4.28 – Функція збудження RS-тригера

При Yand=0 МО не виконується, тому $Q_i(t+1) = Q_i(t)$

При Yand=1 $Q_i(t+1) = Q_i(t) \& X_i(t)$

Будуємо повну таблицю переходів і-ого розряду регістра (рис. 4.29)

Yand	$X_i(t)$	$Q_i(t)$	$Q_i(t+1)$	fr_i	fs_i
0	0	0	0	*	0
0	0	1	1	0	*
0	1	0	0	*	0
0	1	1	1	0	*
1	0	0	0	*	0
1	0	1	0	1	0
1	1	0	0	*	0
1	1	1	1	0	*

Рисунок 4.29 – Таблиця переходів і-ого розряду регістра

Функції збудження F_R , F_S тригерів подано на рис. 4.30.

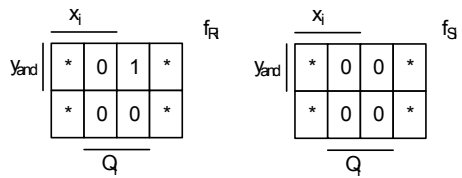


Рисунок 4.30 – Функції збудження f_R , f_S тригера

Мінімізація функцій f_R , f_S

$$f_{R_i} = y_{AND} \overline{x_i} = \overline{y_{AND} \vee x_i} \quad f_{S_i} = "0"$$

Дозволяє отримати схемне рішення, подане на рис. 4.31

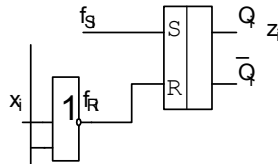


Рисунок 4.31 - Схема одного розряду регістра

На основі цього рішення будемо схему регістра (рис. 4.32)

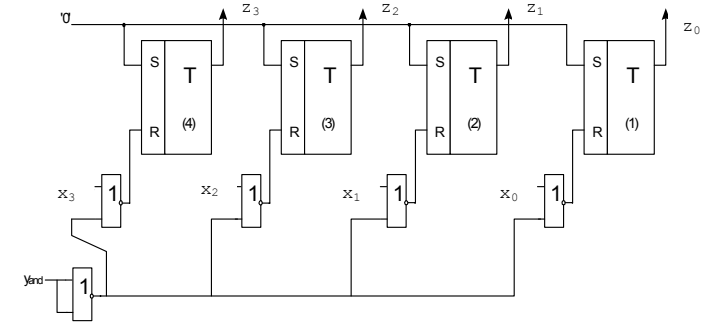


Рисунок 4.32 - Схема повного регістру

Позначення отриманого рішення регістра на схемах подано на рис.4.33

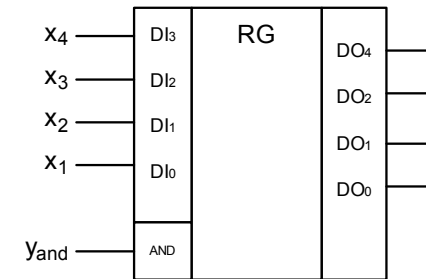


Рисунок 4.33 - Схема повного регістру

Узагальненням функції збудження тригера i -ого розряду є диз'юнкція однойменних функцій збудження, що відповідають окремим мікроопераціям.

Задача На асинхронних D-тригерах побудувати чотирьохрозрядний регістр для виконання на ньому операцій WRITE, READ, OR. Комбінаційну схему побудувати на елементах І-НЕ.

Розв'язання. Сигналами вибору мікрооперацій є Y_w, Y_r, Y_{or}
Будуємо таблицю функції збудження D-тригера (рис. 4.34)

$Q(t)$	$Q(t+1)$	f_d
0	0	0
0	1	1
1	0	0
1	1	1

Рисунок 4.34 - Функція збудження D-тригера

Складаємо повну таблицю переходів для виконання МО WRITE (рис. 4.35)

$$Q_i(t+1) = X_i(t)$$

Y_w	$X_i(t)$	$Q_i(t)$	$Q_i(t+1)$	f_{d_i}
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

Рисунок 4.35 – Таблиця переходів і-го розряду регістра

Якщо сигнал $Y_w=0$, то МО не виконується. Будуємо карту Карно КС для функції збудження (рис. 4.36).

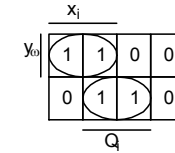


Рисунок 4.36 - Функція збудження тригера

Мінімізацією функції збудження

$$f_{D_i}(y_w) = y_w x_i \vee \overline{y_w} Q_i = \overline{\overline{y_w} x_i \overline{Q_i}}$$

Для мікрооперації OR будуємо повну таблицю переходів виконання МО OR (рис. 4.37)

$$Q_i(t+1) = Q_i(t) \vee X_i(t)$$

Y_{or}	$X_i(t)$	$Q_i(t)$	$Q_i(t+1)$	f_{D_i}
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Рисунок 4.37 – Таблиця переходів і-го розряду регістра

При $Y_{or} = 0$ мікрооперація не виконується. Будуємо карту Карно КС для функції збудження (рис. 4.38)

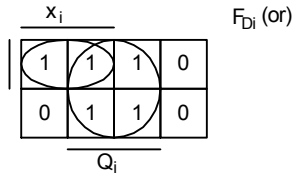


Рисунок 4.38 - Функція збудження тригера

Виконуємо мінімізацію функції f_{di} для МО OR.

$$f_{di}(or) = y_{or} x_i \vee Q_i = \overline{y_{or} x_i} \cdot \overline{Q_i}$$

Загальна функція збудження тригера і-ого розряду дорівнює

$$\begin{aligned} f_{di} &= f_{di}(y_w) \vee f_{di}(y_{or}) = \\ &= \overline{y_w x_i} \cdot \overline{y_w Q_i} \vee \overline{y_{or} x_i} \cdot \overline{Q_i} \\ &= \overline{y_w x_i} \cdot \overline{y_w Q_i} \cdot \overline{y_{or} x_i} \cdot \overline{Q_i} \end{aligned}$$

Звідси отримуємо схему одного розряду (рис. 4.39)

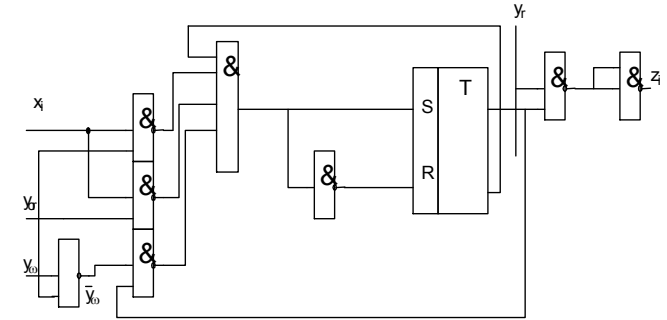


Рисунок 4.39 – Схема одного розряду регістра

Умовне графічне позначення регістру на функціональній схемі дано на рис. 4.40

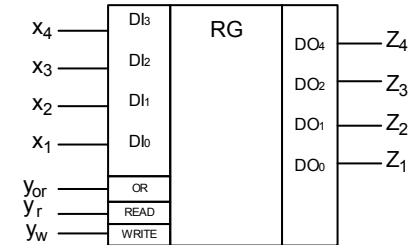


Рисунок 4.40 - Схема повного регістру

Для даної схеми у фіксований момент часу можна виконувати лише одну мікрооперацію.

4.3 Лічильники

Лічильником називають пристрій, призначений для виконання мікрооперації лічби та зберігання слів. Кількість дозволених станів лічильника називається його періодом або модулем.

4.3.1 Класифікація лічильників

За характером МО лічби лічильники поділяють на інкрементні, декрементні та реверсивні. У момент надходження чергового сигналу інкрементний лічильник збільшує свій стан на 1, а декрементний зменшує на 1.

Реверсивний лічильник може виконувати МО inc та dec залежно від сигналу управління.

Залежно від основи системи числення, в якій виконується МО лічби, лічильники бувають: двійкові, двійково-десяткові та 2-5. Лічильники бувають синхронні та асинхронні. Лічильник складається з тригерів.

4.3.2 Структури лічильників

Узагальнена структура синхронного лічильника подано на рис. 4.41.

Призначення сигналів:

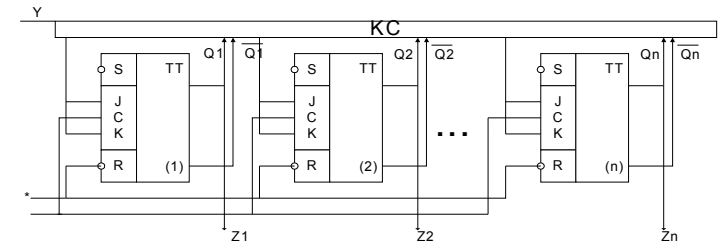
y - сигнал МО

y=1 – режим інкремента inc

y=0 – режим декремента dec

x - лічильний сигнал

S, R - асинхронне встановлення в 0 або 1



z_1, \dots, z_n - стан лічильника

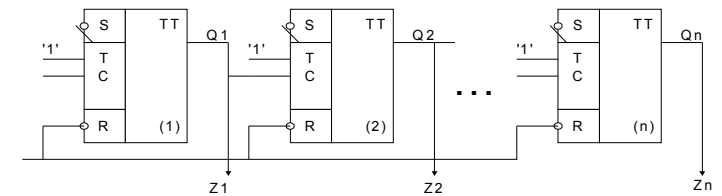
f_i - функція збудження (переносу) i-го розряду

Рисунок 4.41 - Узагальнена структура синхронного лічильника

У синхронних лічильниках перемикання всіх тригерів відбувається одночасно під дією спільного синхросигналу. В асинхронних тригери перемикаються неодноразомно.

Асинхронний лічильник можна побудувати на основі синхронних тригерів. В асинхронних лічильниках має місце послідовний перенос з одного розряду в інший.

Функціональну схему n-розрядного інкрементного асинхронного лічильника з послідовним переносом подано на рис. 4.42.



0, 1, 2, $2^n - 1$ - період лічильника

Рисунок 4.42 - Інкрементний асинхронний лічильник

Часову діаграму інкрементного лічильника подано на рис. 4.43. Окремі розряди лічильника можуть виконувати функцію подільовача.. частоти на 2, 4, 8, 16.

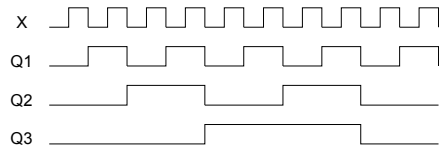


Рисунок 4.43- Часова діаграма інкрементного лічильника

Декрементний асинхронний лічильник показано на рис. 4.44. Часова діаграма – на рис. 4.45.

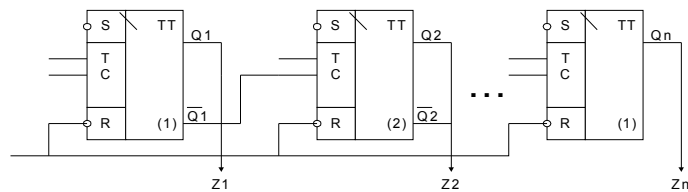


Рисунок 4.44 - Декрементний асинхронний лічильник

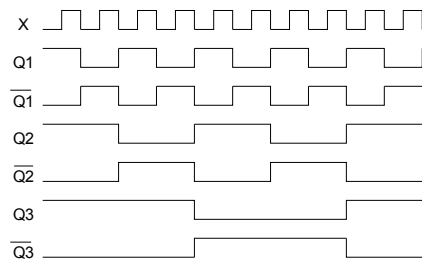


Рисунок 4.45- Часова діаграма декрементного лічильника

У випадку декрементного лічильника перемикання тригера у першому розряді змінюється з кожним надходженням сигналу, а в решті тригерів тільки в тому випадку, коли всі тригери молодших розрядів встановлено в 0.

4.3.3 Типи міжрозрядних переносів в лічильниках

За способом організації переносу між розрядами лічильники поділяються на такі:

- з послідовним переносом;
- з наскрізним переносом;
- з паралельним переносом;
- з груповим переносом.

У лічильниках з послідовним переносом перенос у сусідній старший розряд формується тільки в момент перемикання тригера в попередньому розряді. Такі лічильники є асинхронними.

У лічильника з паралельним переносом аргументи функції переносу для кожного розряду залежить від сигналів на виходах всіх попередніх розрядів. Переноси для всіх розрядів лічильника формуються одночасно.

Для інкрементного лічильника функція переносу i -го розряду має такий вигляд:

$$f_i = Q_1 Q_2 \dots Q_{i-1}, \quad i \neq 1$$

$$f_1 = 1$$

Для декрементного лічильника:

$$f_i = \bar{Q}_1 \bar{Q}_2 \dots \bar{Q}_{i-1}, \quad i \neq 1$$

$$f_1 = 1$$

Для реверсивного лічильника:

$$f_i = Q_1 Q_2 \dots Q_{i-1} y \vee \bar{Q}_1 \bar{Q}_2 \dots \bar{Q}_{i-1} \bar{y}$$

$$f_1 = 1$$

$$y = 1 - \text{inc}$$

$$y = 0 - \text{dec}$$

4.3.4 Синтез лічильників з паралельним переносом

Розглянемо побудову інкрементного лічильника з паралельним переносом з періодом (модулем) 16.

Функції переносу будуть виглядати так:

$$f_1 = 1$$

$$f_2 = Q_1$$

$$f_3 = Q_1 Q_2$$

$$f_4 = Q_1 Q_2 Q_3$$

Схемна реалізація показана на рис. 4.46.

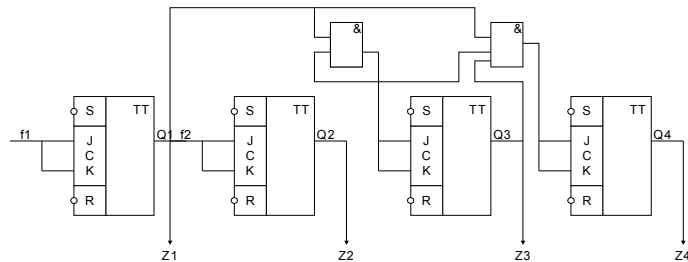


Рисунок 4.46 - Схемна реалізація інкрементного лічильника з паралельним переносом

Перевага паралельного переносу - висока швидкість. Недолік - значні апаратні витрати. Для побудови декрементного лічильника на входи елементів «i» треба подавати інверсний сигнал \bar{Q}_i

4.3.5 Синтез лічильників з наскрізним переносом

Наскрізний перенос є альтернативою послідовному та паралельному переносам. Схему переносу організують таким чином, щоб функція збудження (переносу) i-1 го розряду була аргументом для i-го розряду лічильника. Сигнали переносу для кожного розряду лічильника формують почергово, починаючи з молодших розрядів.

Функція переносу для інкрементного лічильника:

$$f_i = f_{i-1} Q_{i-1}, \quad i \neq 1$$

$$f_1 = 1$$

Функція переносу для декрементного лічильника:

$$f_i = f_{i-1} \bar{Q}_{i-1}, \quad i \neq 1$$

$$f_1 = 1$$

Функція переносу для реверсивного лічильника:

$$f_1 = 1$$

$$f_2 = f_2' \vee f_2'', \text{ де } f_2' = y Q_1, \quad f_2'' = \bar{y} \bar{Q}_1$$

$$f_3 = f_3' \vee f_3'', \text{ де } f_3' = f_2' Q_2, \quad f_3'' = f_2'' \bar{Q}_2$$

$$f_i = f_i' \vee f_i'', \text{ де } f_i' = f_{i-1}' Q_{i-1}, \quad f_i'' = f_{i-1}'' \bar{Q}_{i-1}$$

$$i = 3, 4, \dots, n$$

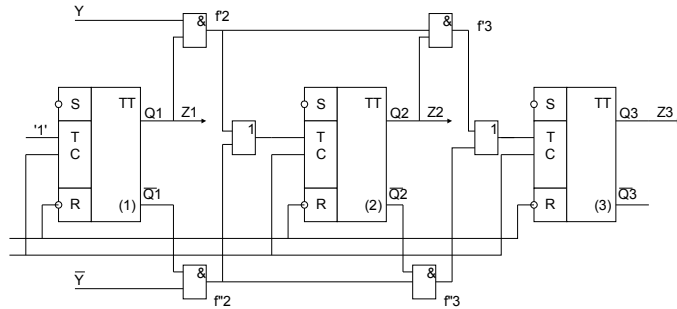


Рисунок 4.47 - Схемна реалізація лічильників з наскрізним переносом

Якщо n велике, то наскрізний перенос потребує менше витрат ніж паралельний перенос, але порівняно з паралельним переносом - менша швидкодія.

4.3.6 Лічильники з груповим переносом

Груповий перенос застосовується якщо n велике. У цьому випадку розряди лічильника розбивають на групи. У межах однієї групи організують паралельний перенос, а між групами послідовний або наскрізний.

Основним часовим параметром для лічильників є максимальна частота (Фліч) надходження лічильних сигналів. Цей параметр характеризує мінімально допустимий період надходження лічильних сигналів при якому лічильник працює без збою.

5 СХЕМИ ПАМ'ЯТІ

5.1 Принципи функціонування запам'ятовувальних пристроїв

5.1.1 Функція пам'яті

Пам'яттю комп'ютера називають сукупність різних пристроїв, призначених для приймання, зберігання і видачі двійкової інформації. Окремий пристрій називають запам'ятовуючим пристроєм (ЗП) або просто пам'яттю. Термін "запам'ятовуючий пристрій" вживають тоді, коли треба підкреслити принцип його побудови: на магнітному осерді, напівпровідниках тощо. Термін "пам'ять" застосовують, коли вказують на функцію, яку вона виконує: основну, постійну тощо.

Пам'ять комп'ютера функціонує під керуванням операційної системи, яка розміщує масиви інформації в пам'яті, забезпечує їхній захист від несанкціонованого доступу та виконує інші функції. Продуктивність і обчислювальні можливості комп'ютера значною мірою визначаються складом і характеристиками ЗП, які застосовуються.

5.1.2 Класифікація пам'яті

Пам'ять сучасних комп'ютерів класифікують за функціональним призначенням, видом носія інформації, способом організації доступу до інформації. За функціональним призначенням пам'ять комп'ютерів поділяється на дві основні групи: зовнішню і внутрішню.

Зовнішні ЗП призначені для тривалого зберігання великих масивів інформації з ємністю до гігабайта і більше та малою швидкодією. Зовнішня пам'ять містить в собі накопичувачі на магнітних стрічках, дисках, барабанах та оптичних дисках.

Внутрішні ЗП призначені для зберігання програм і даних, які виконуються в поточний момент часу. До внутрішньої пам'яті відносять: надоперативні (регістрові) ЗП, які використовують реєстри загального призначення процесора; вони мають невелику інформаційну ємність і швидкодію роботи процесора; кеш–пам'ять, яка служить для зберігання копій інформації, що використовуються в поточних операціях обміну.

Висока швидкодія кеш–пам'яті підвищує продуктивність комп'ютера. Оперативні ЗП характеризуються високою швидкодією інформаційною ємністю до сотень мегабайт; гігабайт. Оперативна пам'ять (ОП) комп'ютерів перших поколінь будувалася на магнітних осердях. Зараз ОП реалізується на напівпровідникових великих інтегральних схемах (ВІС) ЗП. У процесі роботи інформація із зовнішньої пам'яті при необхідності переписується в оперативний ЗП (ОЗП).

Постійні ЗП для зберігання програм, що не міняються, будують на напівпровідникових ВІС. У постійну пам'ять інформація записується заздалегідь і її можна тільки прочитувати.

Оперативні й постійні ЗП утворюють основну пам'ять комп'ютера; спеціалізовані види пам'яті – багатопортові, асоціативні, відеопам'ять тощо.

За фізичним принципом побудови пам'ять комп'ютера буває: магнітна (на осерді та плівках, на циліндричних і плоских магнітних доменах); ультразвукова (магніострикційна, електрострикційна); сегнетоелектрична і голографічна (лазерна), на основі надпровідності; напівпровідникова на ВІС і НВІС, ультра-ВІС.

Напівпровідникові ВІС ЗП в свою чергу характеризуються: технологією виготовлення: на біполярних транзисторах (ТТЛШ, ЕЗЛ, І2Л), на МОН-структурах (р-МОН, п-МОН, КМОН). Серед новітніх розробок

слід відмітити ЗП, де використані планарні технології на основі арсеніду галію.

ЗП за способом зберігання інформації поділяють на статичні та динамічні. У статичних ЗП елементом пам'яті є тригер, а у динамічних елемент пам'яті будують на конденсаторі і МОН-транзисторах.

За енергозалежністю: розрізняють енергозалежні ВІС ЗП, в яких при відключенні джерела живлення інформація, яка зберігається, руйнується (що справедливо в цей час для більшості напівпровідникових мікросхем пам'яті), і енергонезалежні (зазвичай на сегнетоелектриках), в яких інформація зберігається.

За структурною організацією ВІС ЗП символічно подається у вигляді матриці $N \times m$, де N – кількість адресних одиниць інформації, що зберігаються; m – розрядність. Організацію у вигляді $N \times 1$ називають однорозрядною, а $N \times m$ – словниковою.

Елементний базис пам'яті сучасних комп'ютерів складають мікросхеми різного ступеня інтеграції. Основою будь-якого ЗП є елемент пам'яті (ЕП) статичного або динамічного типу, призначений для записування, зберігання і зчитування одного біта інформації – цифри 0 або 1. Сукупність ЕП, які утворюють p -розрядне слово, називають коміркою пам'яті (КП). Множина КП утворює запам'ятовуючий масив, який називається матрицею M елементів пам'яті.

5.1.3 Основні параметри пам'яті

Основними операціями в пам'яті є записування і зчитування певної одиниці інформації, наприклад, байта. Ці операції називаються також зверненням до пам'яті. Пам'ять характеризується інформаційною ємністю,

фізичним об'ємом, питомою ємністю і вартістю, шириною виборки, споживаною потужністю і швидкодією.

Інформаційна ємність E являє собою максимальний об'єм даних, який може одночасно зберігатися в пам'яті. Ємність визначають у бітах, байтах ($8 \text{ біт} = 1 \text{ байт}$), кілобайтах ($2^{10} \text{ байт} = 1 \text{ Кбайт}$), мегабайтах ($2^{10} \text{ Кбайт} = 1 \text{ Мбайт}$) і гігабайтах ($2^{10} \text{ Мбайт} = 1 \text{ Гбайт}$) (при цьому потрібно врахувати, що $2^{10} = 1024$). Питома ємність визначається відношенням інформаційної ємності ЗП до його інформаційної ємності.

Питома вартість – це відношення вартості ЗП до його інформаційної ємності.

Ширина виборки подається числом розрядів, які записуються в ЗП або зчитуються з нього за одне звернення.

Споживану потужність задають або для усього ЗП, або на зберігання одного біту інформації. Основними вимогами до пам'яті є максимально велика інформаційна ємність, висока швидкодія (малий час звернення t_{zv} менший за 10 нс), мінімальна споживана потужність (менша за 1 мкВт на 1 біт інформації, яка зберігається).

У наш час жоден з видів ЗП не задовольняє цих вимог повною мірою. Тому в пам'яті використовуються різні види ЗП, які розрізняються принципами побудови і своїми характеристиками. Швидкодія ЗП вимірюється часом записування і зчитування та тривалістю відповідних ім циклів.

Час записування t_{WR} – це інтервал між моментами появи керуючого сигналу записування і установленням КП в стан, який задають вхідні сигнали.

Час зчитування – це інтервал між моментами появи керуючого сигналу читання t_{RD} і даних на виході пам'яті. Мінімумально допустимий

інтервал між послідовними читаннями t_{CYR} і записуваннями t_{CYW} створює відповідний цикл.

Тривалість циклів може перевищувати час читання чи записування, оскільки після цих операцій необхідна додаткова затримка для встановлення початкового стану пам'яті. Як тривалість циклу звернення до пам'яті беруть величину $t_{CY} = \max(t_{CYW}, t_{CYR})$.

5.2 Мікросхеми пам'яті

5.2.1 Вхідні та вихідні сигнали мікросхеми пам'яті

Мікросхеми ОП мають типові виводи, на яких діють визначені адресні, інформаційні та керуючі сигнали (рис. 5.1, а).

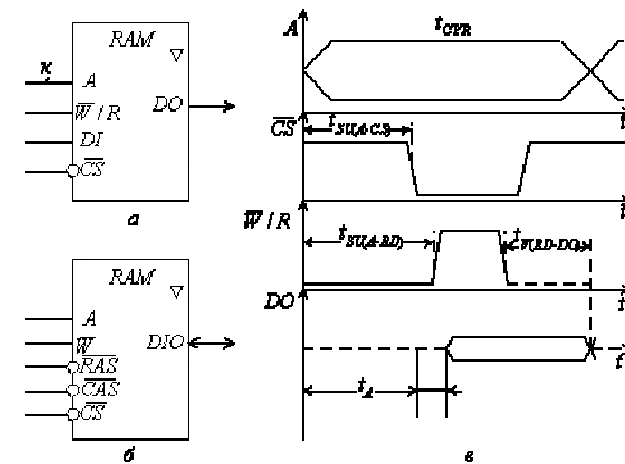


Рисунок 5.1 - Мікросхеми ОП: а, б – умовні графічні позначення; в – часові діаграми сигналів

Призначення виводів мікросхеми ОП і сигналів на них є такі:

A (Address) – входи адреси, розрядність к якій визначається співвідношенням

$$k = \log_2 N,$$

де $N=2^k$ – максимально можливе число даних (біт, байт, слів), що зберігаються в пам'яті і адресуються як єдине ціле;

DI (Data Input) – шина вхідних даних; DO (Data Out) – шина вихідних даних;

\overline{W}/R (Write/Read) – сигнал записування даних при $\overline{W}/R=0$ або зчитування при $\overline{W}/R=1$;

\overline{CS} (Chip Select) або \overline{CE} (Chip Enable) – сигнал дозволу при $\overline{CS}(\overline{CE}) = 0$ чи заборони, якщо $\overline{CS}(\overline{CE}) = 1$, роботи даної мікросхеми.

Особливістю роботи динамічних ЗП є мультиплексування адресних шин ША. Адреса, наприклад, $A=A15, A14, \dots, A0$ ділиться на старшу напівадресу $A_x=A15, A14, \dots, A8$ і молодшу $A_y=A7, A6, \dots, A0$.

Напівадреси подаються на одні й ті ж входи адреси мікросхеми пам'яті. Надходження напівадреси A_x супроводжується сигналом \overline{RAS} (Row Address Strobe), а напівадреси A_y – сигналом \overline{CAS} (Column Address Strobe). Такий спосіб адресації зменшує число виводів корпусу ІМС. Часто виводи DI і DO об'єднуються у спільний вивод DIO.

5.2.2 Часові характеристики мікросхем пам'яті

Вимоги до взаємного часового положення двох сигналів (A–B) задають такими параметрами: часом попереднього установлення $t_{SU}(A–B)$ сигналу A відносно сигналу B, тобто інтервалом між початками обох

сигналів; часом утримання $t_H(A–B)$ – інтервалом часу між початком сигналу A і закінченням сигналу B; часом зберігання $t_V(A–B)$ – інтервалом між закінченнями сигналів A і B.

Тривалість сигналів позначається як t_W (Width – ширина). Для ЗП характерна така послідовність сигналів у часі (рис. 5.1, в): спочатку адреса, потім виборка мікросхеми \overline{CS} , потім строб записування–читання \overline{W}/R .

Індексом A (Access) позначають інтервали часу від появи керуючого сигналу до появи даних на виході (рис.5.1, в).

5.2.3 Класифікація і способи доступу до даних у напівпровідниковій пам'яті

У напівпровідникових ЗП виділяють адресні, послідовні й асоціативні способи доступу до даних (рис. 5.2).

При адресному доступі адресний код указує номер комірки пам'яті, з якою має проводитися обмін. Усі комірки в момент звернення рівнодоступні.

До адресних ЗП відносяться: RAM (Random Access Memory), українські синоніми: ОЗП (оперативний ЗП) або ЗПДВ (ЗП з довільною вибіркою); ROM (Read Only Memory), український термін – ПЗП (постійні ЗП).

Оперативні ЗП зберігають дані, необхідні при виконанні поточної програми; вони можуть бути змінені в будь-який момент часу. Оперативні ЗП в більшості є енергозалежні.

У постійних ЗП вміст комірок або взагалі не змінюється, або змінюється лише в спеціальних режимах. Запам'ятовуючі пристрої RAM поділяються на статичні SRAM (Static RAM) і динамічні DRAM (Dynamic RAM).

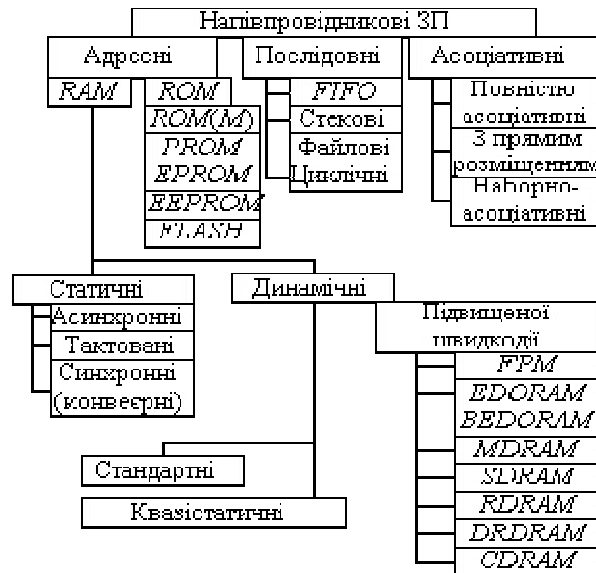


Рисунок 5.2 - Класифікація напівпровідникових ЗП

У статичних RAM елементами пам'яті є тригери. Вони зберігають свій стан, поки схема має напругу живлення і нові дані не записуються.

У динамічних RAM дані зберігаються у вигляді зарядів конденсаторів, створюваних компонентами МОН-транзисторів. Саморозряд конденсаторів веде до руйнування даних, тому вони періодично (кожні 2–30 мс) мають регенеруватися. Але щільність упакування динамічних ЕП перевищує в 4–5 разів такий же показник для статичних RAM. Регенерація даних здійснюється за допомогою спеціальних контролерів.

Розроблені також DRAM із внутрішніми схемами регенерації; такі ЗП називаються квазістатичними. Статичні ОЗП поділяють на такі типи:

асинхронні – керуючі сигнали можна задавати як імпульсами, так і рівнями; тактовані – в них деякі сигнали мають бути обов'язково імпульсами, наприклад, сигнал дозволу роботи \overline{CS} ; синхронні, в яких організований конвеєрний канал передачі даних, що синхронізується від тактової системи процесора.

Динамічні ЗП характеризуються найбільшою інформаційною ємністю і невисокою вартістю, тому вони використовуються як основна пам'ять комп'ютерів.

Статичні ЗП в 4–5 разів дорожчі динамічних і приблизно у стільки ж разів менша їхня інформаційна ємність. Їхнім достоїнством є висока швидкодія, а типовою областю застосування – схеми кеш-пам'яті.

Постійна пам'ять типу ROM(M) програмується при виготовленні за допомогою масок, тому її називають ПЗП масочним. В подальших різновидах ROM у позначеннях є буква P (від Programmable). Це – пам'ять, що одноразово програмується користувачем – PROM (в українській термінології ППЗП – програмовані ПЗП) та багаторазово програмується – EPROM, EEPROM. Пам'ять типу Flash по ЕП подібна до EEPROM (інакше E2PROM), але має структурні й технологічні особливості, які дозволяють виділити її в окремий тип.

У ЗП з послідовним доступом дані, що записуються, створюють чергу. Зчитування виконується слово за словом в порядку записів або навпаки. Прямий порядок зчитування використовується в буферах FIFO з дисципліною “перший прийшов – перший вийшов (First In – First Out)”, а також у файлових і циклічних ЗП. Різниця між пам'яттю FIFO і файловим ЗП полягає в тому, що у FIFO записування у пустий буфер зразу доступне для читання (тобто поступає в кінець ланцюга моделі ЗП).

У файлових ЗП дані поступають у початок ланцюга і з'являються на виході після деякого числа звертань, яке дорівнює числу елементів у ланцюзі.

У циклічних ЗП слова доступні одне за одним з постійним періодом, який визначається ємністю пам'яті. До них відноситься відеопам'ять (VRAM). Зчитування в оберненому порядку властиве стековим ЗП з дисципліною "останній прийшов – першим вийшов". Такі ЗП називаються буферами LIFO (Last In – First Out). Час доступу до конкретної одиниці інформації, що зберігається в послідовних ЗП, є випадковою величиною. В найгіршому випадку для такого доступу треба переглянути весь об'єм інформації, що зберігається у цій пам'яті.

Асоціативний доступ реалізує пошук інформації за деякою ознакою, а не за адресою. В найбільш повній версії всі слова, які зберігаються в пам'яті, можуть одночасно перевірятися на відповідність ознаці, наприклад, на збіг визначених полів слів – тегів (від tag). Ознаку задає вхідне слово (тегова адреса). На вихід передаються слова (можливо, жожного, або кілька слів), які задовольняють ознаці. Дисципліна виведення слів, якщо тегу задовольняє кілька слів, та дисципліна записування нових даних можуть бути різними. Основна область використання асоціативної пам'яті в комп'ютерах – кешування даних.

5.2.4 Загальна характеристика кеш-пам'яті

Кеш-пам'ять (від Cache – схованка) є засіб копіювання і зберігання блоків даних основної пам'яті типу DRAM в процесі виконання програми. Кеш-пам'ять часто побудована на швидкодіючих тригерних елементах пам'яті (ЕП і має невелику ємність у порівнянні з основною динамічною

пам'яттю. Кеш зберігає обмежене число даних і тегів. Тег містить інформацію про фізичну адресу і стан даних.

При кожному зверненні до основної пам'яті спеціальний контролер перевіряє за тегом наявність цієї копії в кеші. Якщо вона є, то виробляється сигнал Hit (кеш-попадання) і звернення відбувається до кеш-пам'яті (рис. 5.3).

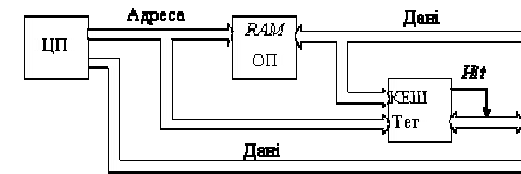


Рисунок 5.3 - . Структура кеш-пам'яті

Якщо копії немає (кеш-промах), то сигнал Hit не виробляється і виконується читання з ОП та одночасне розміщення зчитаних даних в кеші. Обмін з ОП може відбуватися двома способами: перший: звернення до ОП поєднується з одночасним пошуком інформації в тегу. Звернення при попаданні до ОП анулюється; другий: звертання до ОП проводиться тільки після виявлення кеш-промаху.

У сучасних комп'ютерах кеш будують за дворівневою схемою: первинний кеш (L1 Cache) має об'єм десятки Кбайт і вбудовується в процесор. Для підвищення продуктивності часто використовуються роздільні кеші для команд і даних (Гарвардська архітектура); вторинний кеш (L2 Cache), зазвичай встановлюють на системній платі, він має об'єм декілька Мбайт. Більшість прикладних програм має циклічний характер і багаторазово використовує одні й ті самі дані, тому наявність кеша

зменшує кількість звернень до відносно повільної ОП. Тим самим збільшується продуктивність обробки даних.

5.2.5 Загальна характеристика постійної пам'яті

Постійна пам'ять призначена для збереження програм, констант, табличних функцій іншої інформації, яка записується заздалегідь і не змінюється в процесі поточної роботи комп'ютера. Вона застосовується також у перетворювачах кодів, знакогенераторах, у мікропрограмних пристроях керування. Загальним для всіх мікросхем постійної пам'яті є енергонезалежність, словникова організація і використання режиму зчитування як основного.

Мікросхеми постійної пам'яті розділяються на такі групи: ПЗП або ROM (Read Only Memory) – програмуються одноразово заводом-виготовлячем, часто називаються масочними; ППЗП або PROM (Programmable ROM) – програмуються одноразово електричним способом користувачем; • РПЗП-УФ або EPROM (Erasable PROM) – програмуються багаторазово (репрограмуються) з ультрафіолетовим стиранням і електричним записуванням; • РПЗП-ЕС або EEPROM (Electrical EPROM) – програмуються і стираються багаторазово електричним способом.

5.2.6 Загальна характеристика флеш-пам'яті

Флеш-пам'ять (Flash Memory) використовує ЕП на транзисторах ЛІЗМОН з електричним стиранням і записуванням інформації. Вона відноситься до постійної пам'яті типу EEPROM, але ряд архітектурних і функціональних особливостей дозволили виділити флеш-пам'ять в окремий клас.

Флеш-пам'ять використовує поряд з традиційними адресними і керуючими сигналами спеціальні команди. Інформація у мікросхемах флеш-пам'яті записується і зберігається в блоках визначеного розміру, іноді – призначення. При цьому стирання інформації здійснюється або для всієї пам'яті разом, або для великих блоків; це спрощує схеми ЕП.

Флеш-пам'ять переважає EEPROM у тому, що не вимагає спеціальної апаратури для записування чи стирання даних. Розрізняють такі види флеш-пам'яті:

файлова флеш-пам'ять (Flash File) – масив ЕП розділений на блоки однакового розміру (симетрична архітектура);

флеш-пам'ять з несиметричною архітектурою (Boot Block) – масив ЕП розділений на блоки різного розміру; один з блоків має апаратні засоби для захисту інформації в ньому;

флеш-пам'ять з можливістю стирання тільки всього масиву ЕП (Bulk Erase);

флеш-пам'ять з можливістю записування інформації за різних напруг програмування (Start Voltage);

пам'ять з використанням нових ЕП з чотирма станами, які зберігають по два біти (Strata Flash).

Файлова флеш-пам'ять орієнтована на заміну жорстких магнітних дисків. Такі ЗП в сотні разів зменшують споживану потужність, збільшують механічну міцність та надійність, зменшують їхні розміри і масу та на декілька порядків підвищують швидкодію при читанні даних.

Мікросхеми, які замінюють магнітні диски, мають ідентичні блоки та розвинені засоби обміну інформацією. Мікросхеми файлової флеш-пам'яті фірми Intel мають інформаційну ємність 4 – 32 Мбіт, час доступу – 70 – 150 нс.

Мікросхеми Boot Block використовують однобайтову або перемикальну одно- чи двобайтову організацію і складаються з декількох блоків різного розміру. Один з блоків має додаткові апаратні засоби захисту від зміни даних; він призначений для зберігання дуже важливою інформації, яка не змінюється при модифікації даних в інших блоках. Мікросхеми Boot Block призначені для зберігання компонентів системного програмного забезпечення. Привілейований блок містить програму-завантажувач, яка записує з диска необхідні дані для ініціалізації пристроїв комп'ютера.

Мікросхема Boot Block типу 28F00BX/N, яка часто застосовується для зберігання програм базової системи введення-виведення (BIOS) в ПЕОМ, має час доступу 75–150 нс, гарантується 105 циклів стирання – програмування.

Мікросхема 28F00BX містить: основний блок об'ємом 112 Кбайт; два блоки параметрів ємністю по 4 Кбайт кожний; блок-завантажувач об'ємом 4 Кбайт, стирання і програмування якого можливе тільки за особливих умов. Основний блок і блоки параметрів захисту – рівноправні.

Виділення невеликих блоків параметрів дозволяє зберігати в них інформацію, яка часто змінюється. Мікросхеми Bulk Erase мають однобайтову організацію ємністю 32–256 Кбайт, час доступу 65–200 нс і являють собою єдиний масив, який стирається відразу.

5.2.7 Характеристика статичних запам'ятовуючих пристроїв

У статичних ЗП функцію запам'ятовування біту інформації виконують тригери. Вони реалізуються за будь-якою схемотехнікою – ТТЛШ, І2Л, ЕЗП, п-МОН, КМОН іншими. Найбільш інтенсивно розвиваються ОЗП на КМОН-структурах, які при зменшенні роздільної

здатності до 0,2 мкм набувають високої швидкодії. При цьому вони зберігають свої традиційні переваги – велику інформаційну ємність та дуже мале енергоспоживання – до долей мікрвольта на один ЕП.

Статичні ОЗП (SRAM) зазвичай мають структуру 2DM, а при невеликій інформаційній ємності будуються за структурою 2D. Вони широко використовуються в кеш-пам'яті, яка повинна мати максимально можливу швидкодію. Для побудови ЕП статичних ЗП широко використовують RS-тригери за схемотехнікою КМОН. Типова схема такого RS-тригера містить: власне тригер на транзисторах VT1 і VT2 (п-тип) та на навантажувальних транзисторах VT3 і VT4 (р-тип); ключі вибірки на транзисторах VT5 і VT6 (рис. 5.4).

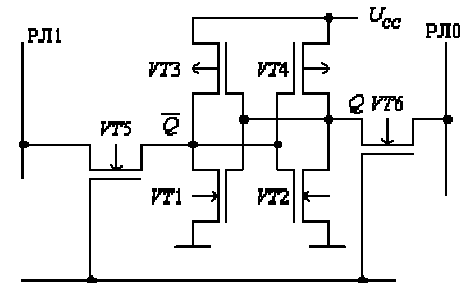


Рисунок 5.4 - Схема RS-тригера на КМОН структурах

З прямим Q інверсним \bar{Q} виходами тригера через ключі вибірки пов'язані розрядні лінії записування–зчитування RЛ0 та RЛ1. В режимі зберігання транзистори VT5 і VT6 закриті. Перед записуванням даних на розрядні лінії подається високий рівень напруги. При виборі даного тригера (лінія вибору ЛВ=1) для записування одиниці встановлюють RЛ1=0, RЛ0=1, а при записуванні нуля – навпаки.

Під час зчитування даних двонаправлені ключі VT5 і VT6 відкриваються, якщо сигнал ЛВі=1. Паразитна ємність стоку закритого транзистора в тригері зменшується на мале значення ΔU , а ємність стоку відкритого транзистора збільшується на таке саме значення. При цьому стан тригера не змінюється. Зміна сигналів на лініях P30 і P31 подається на входи диференційного підсилювача, який формує значення лог.0 або лог.1.

5.2.8 Принцип побудови динамічного запам'ятовуючого елемента

У динамічній пам'яті типу DRAM інформація зберігається у вигляді зарядів на дуже малій ємності $C_3=0,01...0,05$ пФ, яка створена між стоком і підкладкою МОН-транзистора (рис. 5.5).

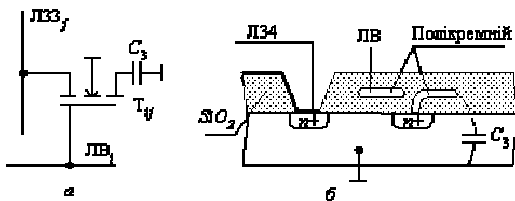


Рисунок . 5.5 - Динамічний одностранзисторний ЕП: а – схема; б – топологія

Стік транзистора не має зовнішнього виводу. Для записування інформації на лінію вибірки ЛВі подається високий рівень напруги, яка відкриває транзистор T_{ij} . Створюється провідний канал, і рівень напруги на розрядній лінії записування–зчитування Л33j визначає стан конденсатора C_3 : заряджений при високому рівні (стан “1”) та розряджений при низькому (стан “0”)

5.2.9 Структури запам'ятовувальних пристроїв.

5.2.9.1 Структура 2D статичних ОЗП

Організація ОЗП у вигляді структури 2D являє собою матрицю запам'ятовуючих елементів (ЗЕ) розмірністю $M=k \times m$, де M – інформаційна ємність ЗП, k - кількість слів пам'яті, m – розрядність слів. Загальну структура 2D подано на рис. 5.6.

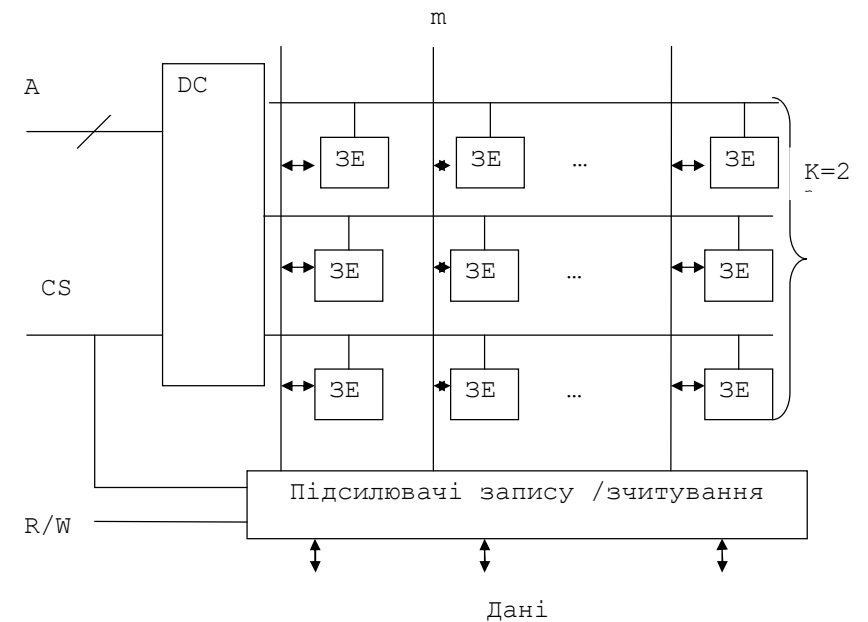


Рисунок 5.6 - Структура 2D RAM

Дешифратор DC за наявності сигналу вибору мікросхеми CS формує одиничний сигнал на одному з виходів, що дозволяє вибрати одне з m - розрядних слів матриці запам'ятовувальних елементів ЗЕ. Кількість таких слів становить $K = 2^n$.

За наявності сигналу R входу R/W слово з m вхідних сигналів Дані підсилювача запису/зчитування подають на елементи ЗЕ, активізовані вихідним сигналом дешифратора DC, і записуються у відповідні ЗЕ.

За наявності сигналу W входу R/W слово з m сигналів елементів ЗЕ, активізованих вихідним сигналом дешифратора DC, передають підсилювачу запису/зчитування і видаються як підсилені вихідні сигнали на виході цього підсилювача.

Пристрої за структурою 2D використовують у RAM малої ємності. Недоліком структури є ускладнення дешифратора при значному збільшенні кількості адресних входів n .

5.2.9.2 Структура 3D запам'ятовувальних пристроїв

Пристрої з такою структурою дозволяють спростити дешифратор адресації до ЗЕ. Дешифратор розбивають на два дешифратора: DC_x, DC_y. На перетині вихідних сигналів цих дешифраторів активізується доступ до відповідного елемента ЗЕ.

Функціональне рішення для ROM такої 3D організації пам'яті показано на рис. 5.7. Дешифратори мають однакову кількість вхідних адресних входів $n/2$. Тому на виході кожного дешифратора отримуємо кількість виходів $2^{n/2} + 2^{n/2} = 2^{n/2+1}$ замість кількості 2^n . Наприклад, при $n=10$ у першому випадку отримаємо 64 виходи, а для структури 2D кількість виходів становить 1024.

Для отримання багаторозрядних ЗП виходи дешифраторів підключають до однобітових запам'ятовувальних матриць, як показано на рис. 5.7.

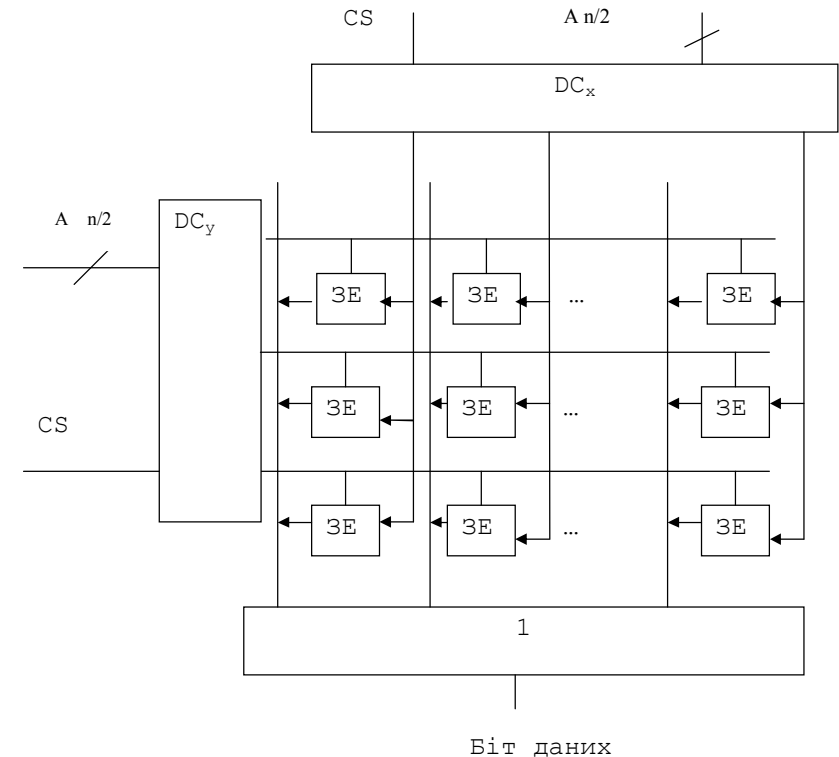


Рисунок 5.7 - Структура 3D для одно розрядної організації

Багаторозрядну організацію ЗП на основі структури 3D утворюють шляхом багатократного дублювання структур рис. 5.7 зі збереженням спільних шин адресації, як показано на рис. 5.8.

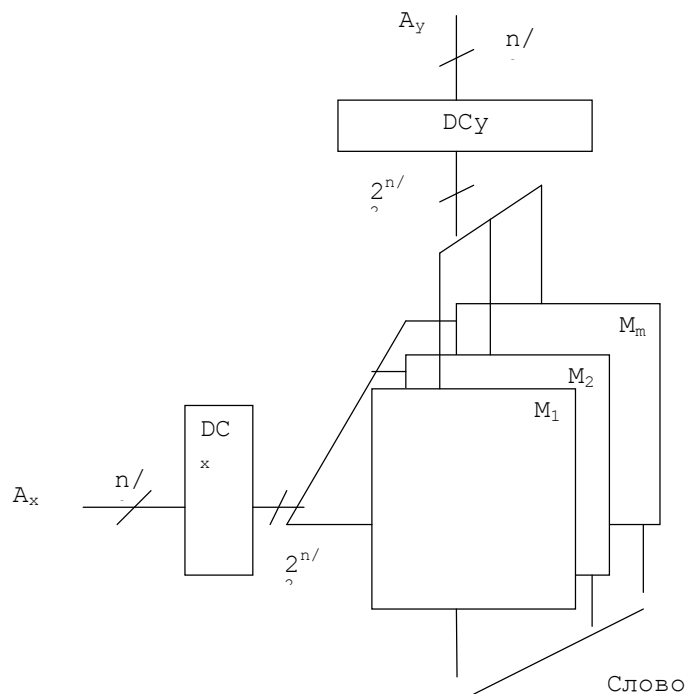


Рисунок 5.8 - Структура 3D для багаторозрядної організації

5.2.9.3 Структура 2DM статичних ЗП

У структурах 2DM на відміну від 2D довжина рядка може бути набагато довшою за довжина слів, які зберігаються у ROM пам'яті (рис. 5.9). Дешифратор DC_x за адресою $A_1 = A_{n-1} \dots A_k$ активізує одне з 2^{n-k} слів, кожне з яких має довжину $m2^k$, з матриці ЗП розмірністю $2^{n-k} \times m2^k$.

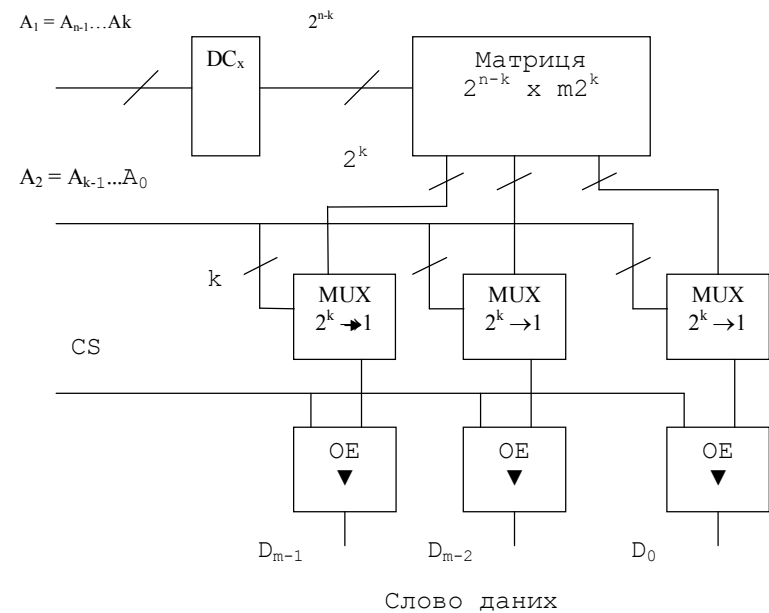


Рисунок 5.9 - Структура 2DM для багаторозрядної організації ROM

На другому рівні дешифрування за допомогою адреси $A_2 = A_{k-1} \dots A_0$ зі слова довжиною $m2^k$ виділяють за допомогою низки мультиплексорів одно бітові значення. А за допомогою сигналу вибору CS і буферів OE утворюють вихідне слово довжиною m .

Узагальнену структуру 2DM для RAM показано на рис. 5.10. Сигнал керування R/W визначає режим зчитування або запису для блока слів матриці M. А самі дані довжиною m під час зчитування або запису проходять через буфер даних BD.

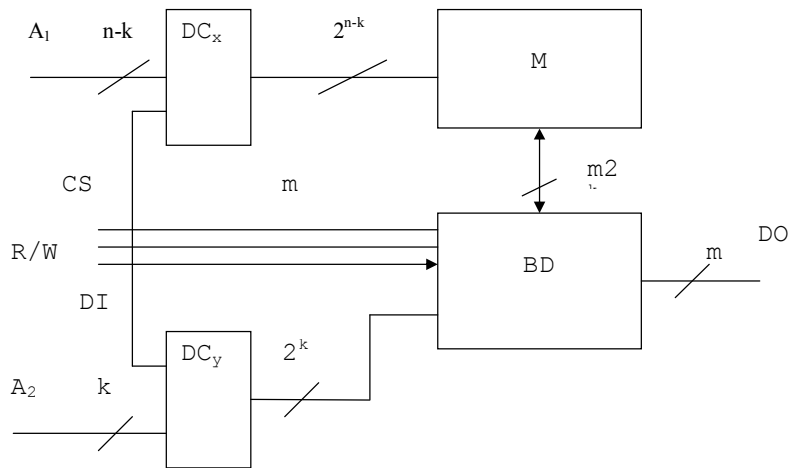


Рисунок 5.10 - Структура 2DM для багаторозрядної організації RAM

5.2.9.4 Пам'ять з послідовним доступом

Пам'ять з послідовним доступом передбачає просування по ланці елементів пам'яті, подібно до регістру зсуву. До основних типів такої пам'яті належать відеопам'ять, буфер FIFO, кеш-пам'ять.

Відеопам'ять (рис. 5.11) передбачає виведення на екран послідовності кодів у порядку сканування екрану. Коди містять параметри кольору, яскравості окремих пікселів. Кадр монітора являє собою послідовність слів, які відповідають окремим пікселям. Зазвичай довжина слів становить від 8 до 24 бітів.

Під час зчитування вибирається нижній канал мультиплектора MX і інформація переписується з регістру у наступний регістр RG. В даних існує спеціальний код синхросигналу (показано сигнал для коду кадрового синхросигналу ККС), поява якого (за допомогою схеми порівняння – компаратора =) синхронізує запуск розгортки монітора.

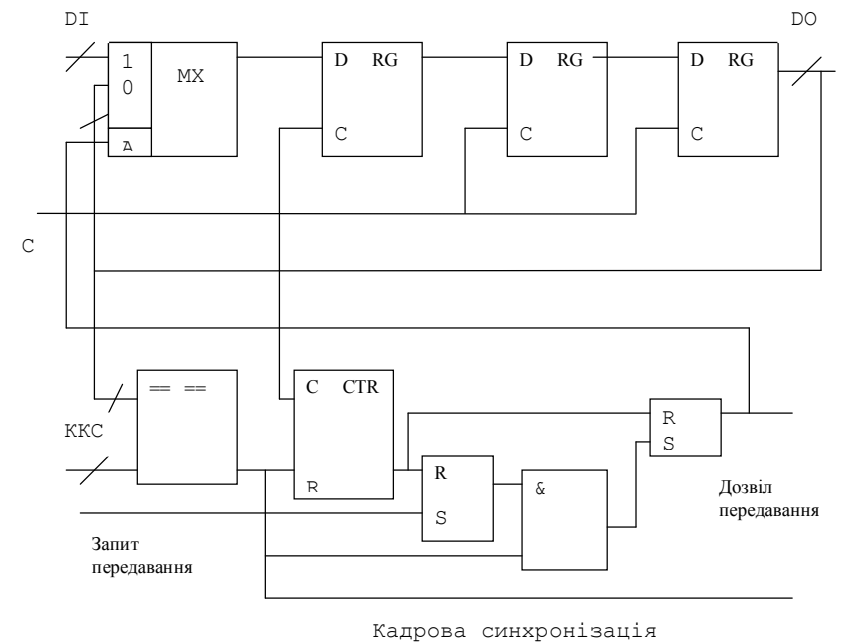


Рисунок 5.11 - Структура відеопам'яті

Пакетний запис виконують за даними на верхньому каналі мультиплектора MX для входу DI після появи сигналу запиту передавання в момент проходження кадрового синхросигналу. Формується сигнал передавання на запис з DI у регістри RG з подальшим зсувом даних у регістрах.

Після отримання повного кадра лічильник CTR, довжина якого становить один кадр, переповнюється. Сигнал переповнення пристрій переходить у режим циклічного перезапису.

Буфер FIFO (рис. 5.12) дозволяє дані у порядку вибрки слів. Інтервали між словами можуть бути різними. Моменти запису визначають за зовнішнім сигналом.

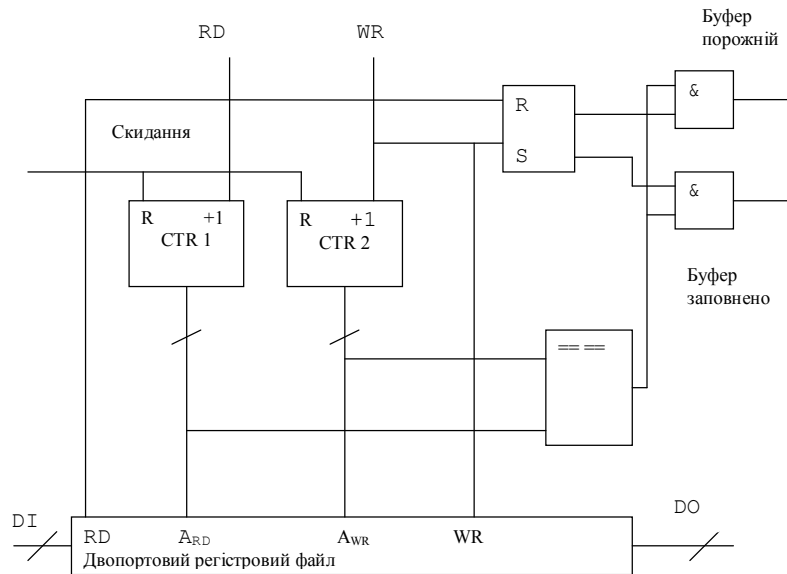


Рисунок 5.12 - Буфер FIFO

Перед початком роботи лічильники CTR 1 (читання), CTR 2 (запису) скидають. Під час запису і зчитування зміст відповідних лічильників збільшується на одиницю. Для однакових значень лічильників формується сигнал про порожній буфер. А при переповненні буфера запису

формується сигнал заповнення буфера. В цьому випадку приймання даних припиняється. Якщо буфер порожній, то припиняють зчитування.

Кеш- пам'ять (рис. 5.13) забезпечує збереження копії частини оперативної пам'яті і більшу продуктивність, оскільки зазвичай реалізують на тригерних елементах.

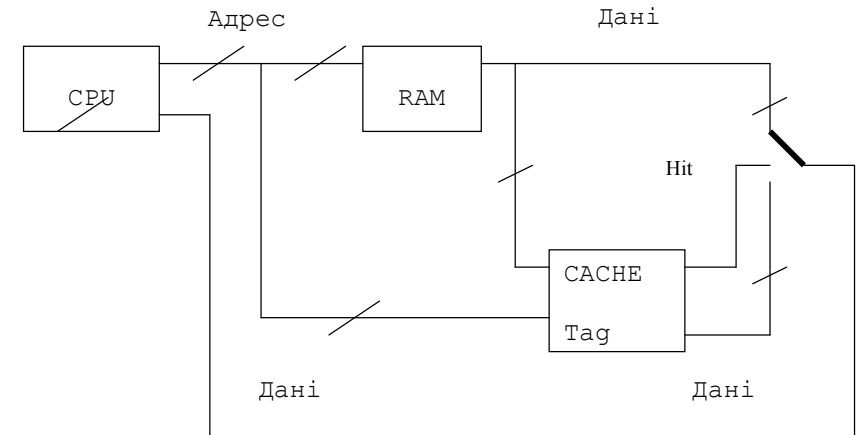


Рисунок 5.13 - Структура кешованої пам'яті

За наявності при зчитуванні даних копії даних у кеші, дані зчитують з кешу за відповідним сигналом Hit. Дані надходять на спільну шину даних. Інакше, за відсутності сигналу Hit, дані зчитують з основної пам'яті RAM і заносять у кеш. Перевага полягає у частому застосуванні даних кеша, які повторно використовують.

Дані, які вміщують у кеш супроводжують відповідним додатковими даними – тегом. Тег містить дані про комірки основної RAM, які

зберігають у кеші. Це дозволяє будувати на основі кешу асоціативну пам'ять.

У повністю асоціативній кеш-пам'яті (рис. 5.14) FASM (Fully Associated Cache Memory) поле Тег містить повну фізичну адресу інформації RAM. За наявності даних, які шукають, за значенням тега, формується сигнал Hit. При зчитуванні даних і їх наявності у кеші сигнал Hit =1. А при відсутності Hit =0. В останньому випадку у вільну комірку тег-кеш вміщують дані з RAM. Ці процедури виконують з використанням спеціального контролера.

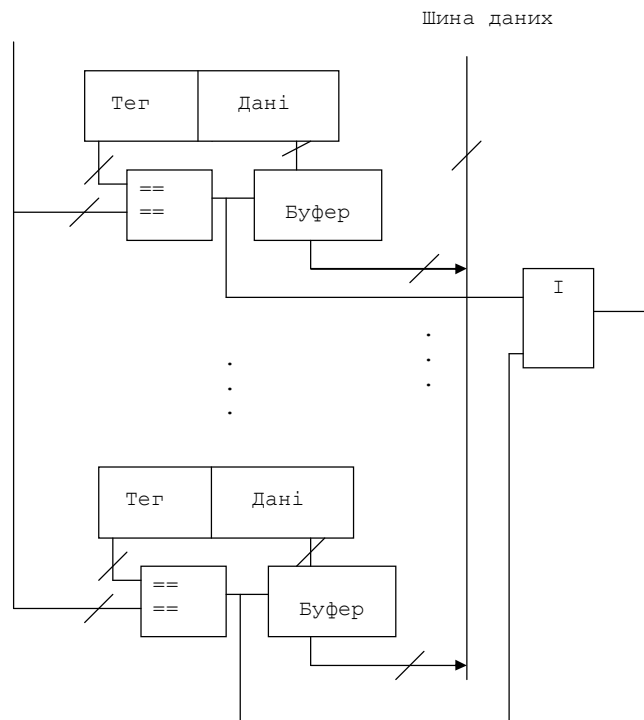


Рисунок 5.14 - Структура асоціативної кеш-пам'яті

6 МІКРОПРОЦЕСОРИ

6.1 Класифікація мікропроцесорів.

6.1.1 Загальні поняття

Мікропроцесор – це програмно керований пристрій, який здійснює обробку даних, а також керування нею і побудований на одній або кількох великих інтегральних схемах. Мікропроцесор включає арифметико-логічний пристрій (АЛП), який використовується для реалізації великої кількості команд мікропроцесора. Кількість команд сучасних мікропроцесорів може становити кілька сотен.

Основними передумовами створення і подальшого бурхливого розвитку мікропроцесорів є:

1. Ефективне багаторазове використання апаратних засобів (зокрема, АЛП) для реалізації великої кількості команд;
2. Універсальність мікропроцесорів, оскільки мікропроцесор може виконувати будь – яку систему команд, за умови, що деякі з них можуть реалізовувати досить складні функції обробки даних;
3. Пристосованість до реалізації за технологіями кристал метал-окисел напівпровідник (КМОП), що дозволяє створювати мікросхеми з кількістю елементів до десятків мільйонів логічних елементів при незначних енергетичних витратах для роботи цих елементів;
4. Малий час розробки різноманітних пристроїв обробки даних і керування на мікропроцесорах;
5. Незначні витрати для модернізації цих пристроїв шляхом зміни програми їх роботи.

6.1.2 Види мікропроцесорів

Мікропроцесори класифікують за

1. Областю використання (мікроконтролери, універсальні мікропроцесори, сигнальні мікропроцесори);
2. За принципом будови внутрішньої структурою (Гарвардська архітектура і архітектура фон- Неймана);
3. За системою команд (акумуляторні мікропроцесори і мікропроцесори з регістрами загального призначення);

Мікропроцесори з регістрами загального призначення можуть виконувати операції над довільними комірками пам'яті. Залежно від типу операцій розрізняють одно адресні, двоадресні і трьохадресні мікропроцесори.

В акумуляторних мікропроцесорах операції виконуються тільки для однієї комірки пам'яті – акумулятора. На практиці мікропроцесори виконують як акумуляторні, так і операції з використанням регістрів.

У Гарвардській архітектурі є два типи пам'яті: пам'ять програм і пам'ять даних. Це дозволяє обмежити виконання операції запису у пам'ять програм. Додатково виділяють окремі шини обміну даних для пам'яті програм і пам'яті даних. Гарвардська архітектура застосовується у сучасних мікропроцесорах підвищеної надійності і швидкодії.

В архітектурі фон-Неймана пам'ять програм і даних сумішені, що дозволяє використовувати одні і ті ж ділянки пам'яті для програм і даних. Це зумовлює гнучко модернізувати роботу пристрою, але ускладнює керування і знижує надійність.

У персональних комп'ютерах найбільшого поширення набули мікропроцесори, які називають центральними процесора CPU (Central Processing Unit) і включають:

1. Арифметико-логічний пристрій
2. Блок керування

3. Блок пам'яті, який включає, зокрема регістрову пам'ять і кеш-пам'ять

4. Пристрій введення-виведення.

За законом Гордона Мура кількість транзисторів в інтегральних схемах подвоюється кожні 18 місяців. Тому з моменту створення першого мікропроцесора у 1971 році, з'явилась значна кількість мікропроцесорів.

Мікропроцесори до 2002 року іноді класифікують за поколіннями. Основні параметри деяких з цих мікропроцесорів наведені у таблиці 6.1.

Таблиця 6.1 – Покоління мікропроцесорів

Поко- ління	Рік випуску	Тип	Розрядність шини дані/адреса	Ємність кеш- пам'яті, Кбайт	Частота сист. шини, МГц	Частота процесора, МГц
1	1978	I8086	16/16	-		
2	1982	I80286	16/16	-		
3	1985- 1988	I80386DX I80386SX	32/32 16/32	- 8	16-133	16-133
4	1994	I80486DX4	32/32	8+8	25-40	75-120
5	1997	Pentium MMX	64/32	16+16	66	168-333
6	1999	Pentium III	64/36	16+16	100	450-1200
7	2002	AMD Athlon XP	64/36	64+64	333	2700-2800
	2002	Pentium IV	64/36	12+8	533	2200-3060

Кількість транзисторів в Pentium IV складає 42-55 млн. CPU встановлюють у різних різних типів: Slot (Athlon, Celeron інші) і Socket (Pentium IV, Athlon XP).

У сучасних CPU кількість транзисторів досягає до 600-1500 млн.

6.2 Будова процесорів комп'ютера

Універсальні комп'ютери поділяються на три функціонально зв'язані апаратні частини: процесор, пам'ять і периферійні пристрої. Процесор – це основна функціональна частина комп'ютера, яка інтерпретує й виконує команди, тобто безпосередньо реалізує програмно-керований процес обробки даних.

Процесор складається з пристрою керування, арифметико-логічного пристрою (АЛП) та блоку інтерфейсу (БІФ) для з'єднання із зовнішнім середовищем – пам'яттю, периферійними пристроями (рис. 6.1). Оброблення даних здійснюється в АЛП, який містить арифметико-логічний блок (АЛБ), блок регістрів загального призначення (РЗП), блок контролю (БК) і місцевий блок керування при децентралізованому керуванні.

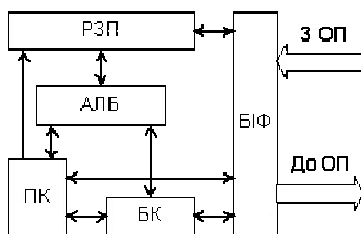


Рисунок 6.1 – Будова процесора

Процесор, який виконує в обчислювальній системі основні функції, називають центральним (ЦП). Спеціалізований процесор, призначений для керування зовнішніми пристроями (накопичувачами, дисплеями, принтерами тощо.) називають контролером.

Процесор характеризується архітектурою, до якої відносять: список арифметико-логічних операцій (система команд); типи і формати команд і даних; організацію адресного простору пам'яті і периферійних пристроїв; способи адресації команд і даних; функції складових частин і структуру зв'язків з іншими пристроями машин та режими роботи.

Арифметико-логічний блок має універсальний двійковий комбінаційний суматор, двійково-десятковий суматор або схему десяткової корекції, регістри для тимчасового зберігання двох операндів і результату операцій та регістр прапорців. Для підвищення продуктивності в АЛП можуть включати спеціалізовані вузли-зсувачі, помножувачі, схеми прискореного переносу інших пристроїв. Ряд процесорів мають по два АЛП. Розрядність АЛП визначає розрядність всього процесора. В РЗП зберігаються початкові дані, проміжні та кінцеві результати, адреси даних, константи, які необхідні в процесі виконання команди.

Всі операції в АЛП реалізуються як просторово-часові послідовності мікрооперацій над двійковими словами, кожна з яких є сукупністю булевих операцій над бітами слів. В АЛП реалізуються такі типові мікрооперації: передачі слів між регістрами та регістрами і пам'яттю; додавання двох слів, декремент (мінус 1) або інкремент (плюс 1) слова; арифметичні, логічні та циклічні зсуви вправо чи вліво; порозрядні логічні операції ЧИ, І, виключальне ЧИ та порівняння операндів; перетворення кодів слів – інверсія, доповнення, розширення тощо.

Пристрій керування (ПК) керує процесом оброблення даних, забезпечує основні режими роботи (початкових установлень, очікування,

переривання, прямого доступу до пам'яті, діагностики і контролю) та взаємодію всіх пристроїв комп'ютера. Для виконання цих функцій ПК має в своєму складі регістр і дешифратор команд, програмний лічильник для задання адреси наступної команди, блок керування та схеми синхронізації, діагностики й контролю. До складу процесора можуть входити спеціальні системні засоби (служба часу, засоби міжпроцесорного зв'язку, пульт керування тощо.).

Пристрій керування послідовно зчитує код команди з пам'яті і розміщує його в регістр команд (інструкцій). Блок керування дешифрує команду і формує послідовності керуючих сигналів. Для виконання однієї мікрооперації в АЛП необхідний один керуючий сигнал.

В одному машинному такті реалізується сукупність мікрооперацій – мікрокоманда. Множина мікрокоманд створює мікропрограму команди. Кожна команда має свою мікропрограму, час виконання якої називається командним циклом.

Розрізняють апаратні, мікропрограмні та комбіновані блоки керування. Апаратні блоки керування побудовані на основі схемної логіки, а мікропрограмні – програмовної логіки (мають пам'ять мікропрограм). Комбіновані блоки керування використовують обидва способи їхньої реалізації.

При централізованому керуванні один ПК керує процесом оброблення команд і даних у всій машині. При децентралізованому керуванні ПК формує основні керуючі сигнали, а опрацюванням даних керує місцевий блок керування, розміщений в АЛП. Всі команди в комп'ютері реалізуються на основі принципу мікропрограмного керування, тобто виконання мікропрограм.

Мікросхема, яка виконує функції мікропроцесора або його частини, називається мікропроцесорною. Сукупність мікропроцесорних інших

мікросхем, які сумісні за конструктивно-технологічним виконанням і призначені для спільного використання, називається мікропроцесорним комплектом (МПК).

До характеристик мікропроцесорних інтегральних мікросхем відносять: розміри кристала і кількість транзисторів у ньому, тип корпусу і кількість виводів.

Найбільш важливими статичними і динамічними електричними параметрами мікропроцесорів як мікроелектронних виробів є:

- кількість джерел живлення та їхня напруга;
- струм і потужність споживання; кількість серій синхроімпульсів, їхні частота і амплітуда;
- рівні логічних сигналів; вхідна і вихідна ємності, навантажувальна здатність;
- час затримки розповсюдження сигналів, число операцій в секунду над операндами, які зберігаються в регістрах-акумуляторах.

Для виробництва мікропроцесорів використовують схемотехнічні технології: ТТЛШ, ЕЗЛ, І2Л, n-МОН, р-МОН, КМОН тощо.

Залежно від режиму роботи розрізняють такі процесори:

однопрограмні (виконують одну програму);

багатопрограмні (мають засоби для одночасного виконання кількох програм);

мультипроцесори (системи, в яких одночасно можуть бути активними декілька процесорів);

конвеєрні (команди виконуються послідовно рядом пристроїв, причому різні пристрої можуть одночасно обробляти відповідні частини декількох команд);

матричні (мають спеціальну архітектуру, розраховану на оброблення числових масивів);

співпроцесори (арифметичні розширювачі), призначені для розширення списку команд ЦП; самостійно не використовуються;

периферійні, які виконують функції введення-виведення інформації (асоціативні процесори, в яких характер обробки даних визначається змістом самих даних).

За видом оброблюваної інформації розрізняють цифрові (звичайні) та аналогові мікропроцесори. В аналогових мікропроцесорах на вході використовують аналого-цифровий перетворювач (АЦП) для перетворення аналогових величин в цифровий код, а на виході – схеми цифро-аналогового перетворювача (ЦАП), які перетворюють цифрові дані в аналогові.

6.3 Мікропроцесорні засоби

Основою мікропроцесорних засобів є мікропроцесорні комплекти МПК і базові кристали, ВІС пам'яті (рис. 6.2).

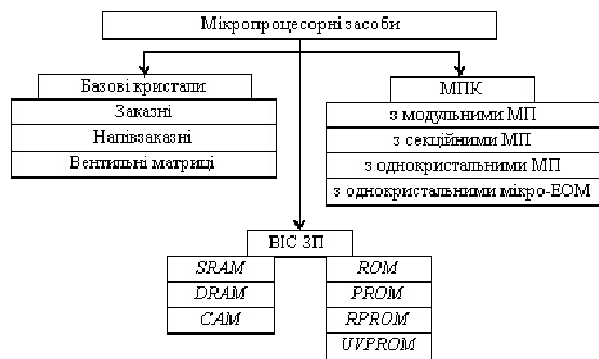


Рисунок 6.2 - Склад мікропроцесорних засобів

Конструктивно-технологічний розвиток мікропроцесорів відбувається в таких напрямках. Використовують нові технології, наприклад, БіКМОН, в яких комбінуються біполярні транзистори (для збільшення швидкості) та КМОН структури (для зменшення споживаної потужності та підвищення щільності компоновки). В перших мікропроцесорах відстань між сусідніми лініями дорівнювала 10 мкм, то в останніх виробках вона дорівнює 22 нм.

Зростає рівень інтеграції: від 2800 транзисторів в чипах перших мікропроцесорів до 10–12 млн і більше в останніх виробках типу Itanium. За законом Мура характеристики мікросхем мають поліпшуватися в два рази кожні 18 місяців при збереженні вартості.

У сучасних CPU на чипі може розташовуватись більше 1,2 млрд транзисторів, які мають працювати на частоті 2700 МГц. Збільшується розрядність оброблюваних даних: від чотирьох: у перших мікропроцесорах до 64 для Itanium. Прискорюється зміна поколінь мікропроцесорів за рахунок збільшення щільності розміщення елементів на еристалі (від 180 нм до 8-12 нм), та удосконалення архітектур, зокрема з використанням кешів, збільшення кількості і розрядності регістрів, багатоядерності тощо.

6.4 Архітектури мікропроцесорів

Архітектуру мікропроцесора характеризують: список команд та їхні формати; способи адресації; розрядність і ємність адресованої пам'яті; структура регістрів та їхні функції тощо. Історично першими склалися такі основні архітектури мікропроцесорів: з акумулятором, з РЗП, зі стековою організацією та комбіновані (рис.6.3).

Структура всіх мікропроцесорів містить такі вузли і блоки, об'єднані спільною внутрішньою системною шиною даних: АЛБ; ПК; ІР – реєстр команд (інструкцій); FL – реєстр ознак (прапорців); EAR – реєстр виконавчої адреси; БІФ – блок інтерфейсу з вихідними шинами адреси, даних і керування.

В архітектурі мікропроцесора з акумуляторами (рис. 6.3, а) додатково використовують індексний реєстр X і показчик стека SP; реєстр R виконує функції акумулятора А.

При виконанні арифметико-логічних операцій перший операнд попередньо розміщується в акумуляторі, а другий надходить з ОП безпосередньо на вхід АЛБ.

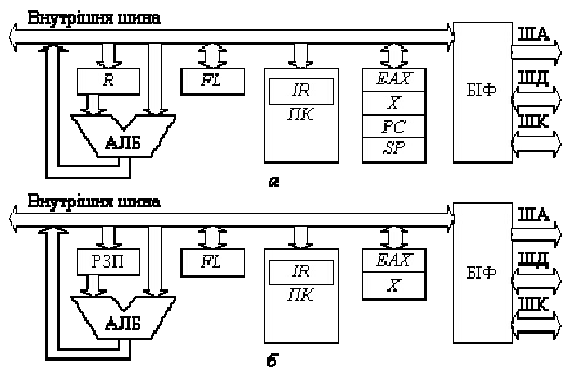


Рисунок 6.3 - Структура мікропроцесорів: а – з акумулятором і стеком; б – з РЗП

Результат операції розміщується в акумуляторі, а в реєстр FL автоматично записуються ознаки операції. В архітектурі мікропроцесора з РЗП (рис. 6.3, б) зазвичай використовують від восьми до 16 реєстрів, кожний з яких може виконувати функції акумулятора. Збільшення числа

РЗП значно зменшує кількість звернень до ОП, що підвищує продуктивність комп'ютера.

Але збільшення кількості реєстрів призводить до втрат машинного часу у випадках переривання програм, оскільки при цьому необхідно зберегти зміст РЗП в пам'яті (зазвичай у стеку), а потім відновити його після обробки переривання (збереження і відновлення контексту процесів). Тому при виборі кількості РЗП враховують цей фактор.

У мікропроцесорі зі стековою архітектурою (рис. 6.3, а) відсутній акумулятор і РЗП, а R виконує функції реєстра тимчасового зберігання даних ОП на час виконання операції.

Читання даних і їхнє записування у стекову пам'ять здійснюють за допомогою показчика стека SP. Усі операції з даними виконують відповідно до польського запису: операнди розміщують у стеку в послідовності виконання над ними дій. У стек послідовно завантажують числа a, b і c (рис. 6.4).

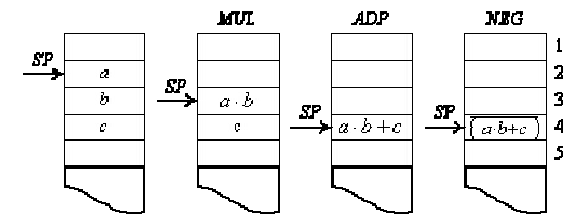


Рисунок 6.4 - Ілюстрація роботи стека

За командою множення MUL одержуємо добуток $a \cdot b$, який записується в адресі b. За командою додавання ADD маємо $a \cdot b + c$, який записується в адресі c. Після команди інвертування NEG в комірці за адресою c записується обернений код результату $\overline{a \cdot b + c}$.

В мікропроцесорах з комбінованою архітектурою (рис. 6.5) об'єднують значною мірою властивості архітектур з акумулятором, РЗП і стеком. До них відносять однокристальний мікропроцесор 8080 (1974 р., фірма Intel), а також модель Z80 фірми Zilog.

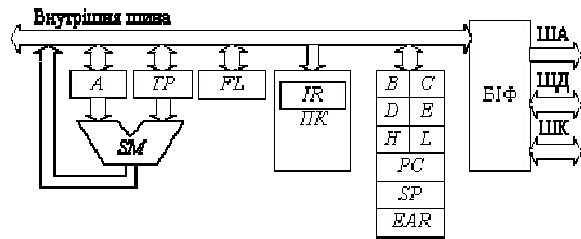


Рисунок 6.5 - Структура мікропроцесора з комбінованою архітектурою (близькою до моделі 8085А)

Комбінована структура містить: складний комбінований суматор SM (в ньому виконується більшість арифметичних і логічних операцій), регістри A і TP; разом вони створюють АЛБ; блок РЗП, регістри B, C, D, E, H, L, програмний лічильник PC, покажчик стека SP, регістр виконавчої адреси EAR; блок ІФ з ША, ШД і ШК; пристрій керування ПК з регістром команд IR.

Сукупність обчислювальних засобів, куди входять один або декілька мікропроцесорів та напівпровідникова пам'ять і засоби інтерфейса, називається мікропроцесорною системою (МПС).

6.5 Режими обміну даними у комп'ютері з використанням процесорів

6.5.1 Синхронний і асинхронний обмін

Процесори забезпечують програмно керований обмін інформацією між ядром машини і периферією. Реалізують такі види програмно керованого обміну інформацією: синхронний і асинхронний обміни в послідовних і паралельних двійкових кодах. Обмін виконується за схемою рис. 6.6, а з перериванням програми за запитом ПП.

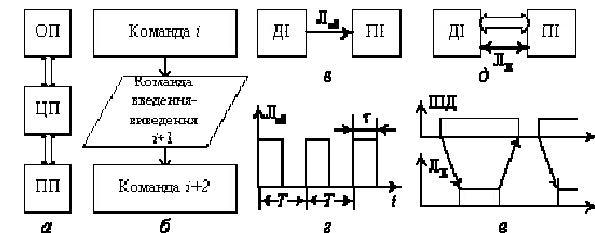


Рисунок 6.6 - Синхронний обмін: а – схема, б – алгоритм; в,г – послідовний обмін без стробування; д,е – паралельний обмін із стробуванням

При синхронній передачі джерело інформації ДІ завжди готове до обміну відповідно до алгоритму (рис. 6.6, б). Джерело інформації виставляє і утримує значення даних на лінії послідовного обміну $L_{об}$ протягом часу t , який складається з тривалості затримки розповсюдження сигналу на лінії, його розпізнавання і фіксації в регістрі приймача ПІ (рис. 6.6, в,г).

При синхронному паралельному обміні часто використовують сигнал квитування, який передається і приймається по окремій лінії L_K і визначає інтервал часу надійного приймання даних приймачем (рис. 6.6, д,е).

Асинхронний обмін виконується при готовності (Ready) зовнішнього пристрою до обміну даними відповідно до алгоритму (рис. 6.7, а). При асинхронній передачі паралельного коду по ШД використовують метод квитування, в якому поєднуються спільна дія сигналу стробування ЛС від джерела ДІ та сигналу підтвердження приймання ЛП від приймача ПІ (рис. 6.7, б,в).

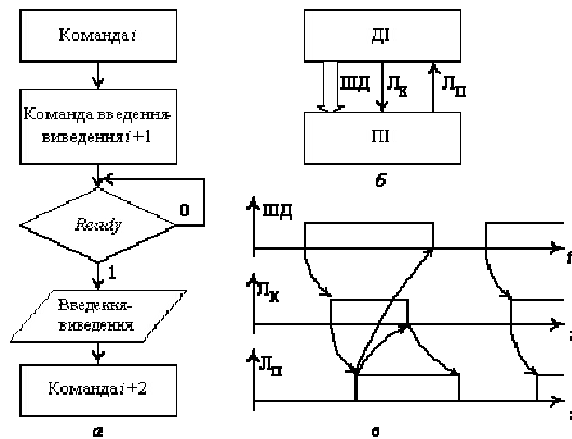


Рисунок 6.7 - Асинхронний обмін: а – алгоритм; б – схема; в – часові діаграми роботи з квитуванням

Основним недоліком синхронного і асинхронного обміну є значне завантаження процесора операціями введення–виведення, що призводить до суттєвого зменшення продуктивності комп'ютера. Тому синхронний і

асинхронний обміни використовують при передачі одиночних байтів чи слів.

6.5.2 Обмін в режимі переривання

Обмін в режимі переривання здійснюють апаратно за ініціативою зовнішнього пристрою чи програмно – командою переривання INT. Процесор, одержавши апаратний запит на переривання, закінчує поточну команду, пересилає в ОП зміст своїх регістрів і переходить на підпрограму обслуговування переривання. Після її закінчення процесор відновлює зміст своїх регістрів і продовжує виконання перерваної програми (рис. 6.8, а).

Апаратно режим переривання забезпечується контролером переривань (КПР), до якого підключаються ПП. Вихід INT контролера подається на відповідний вхід процесора ЦП, а на ШД пересилається початкова адреса підпрограми обслуговування (рис. 6.8, б). При програмному перериванні адреса підпрограми подається в самій команді переривання INT.

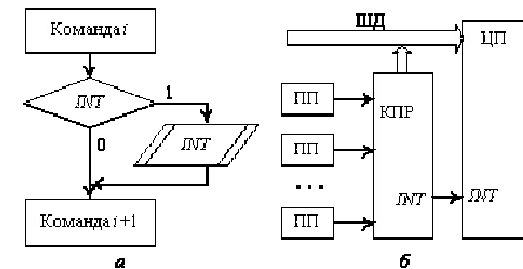


Рисунок 6.8 - Обмін за перериванням: а – алгоритм; б – схема підключення контролера КПР

6.5.3 Прямий доступ до пам'яті

Прямий доступ до пам'яті (ПДП) використовують для швидкого обміну масивами інформації між основною пам'яттю і периферією. При цьому процесор (мікропроцесор) звільняється від безпосереднього керування операціями введення–виведення. ПДП реалізується відповідно до алгоритму (рис.6.9, а).

В міні- і мікрокомп'ютерах прямим доступом керує контролер прямого доступу до пам'яті (КПДП). Перед початком обміну процесор пересилає в КПДП таку інформацію (програмування контролера): початкову адресу області пам'яті, яка бере участь у обміні; напрямом операції обміну – введення чи виведення; кількість байтів, які підлягають передачі.

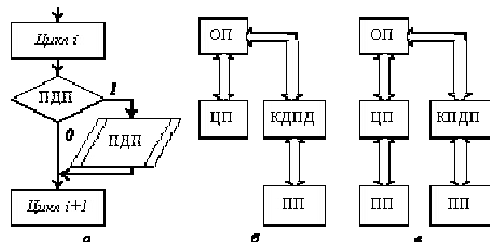


Рисунок 6.9 - Режим ПДП: а – алгоритм; б,в – схеми підключення контролерів КПДП

Особливість режиму ПДП – обмін даними може бути між машинними циклами в команді (“Захват циклу”), а також після закінчення

команди. Контролер ПДП керує обміном даними між ОП і ПП без участі процесора (рис. 6.9, б).

При необхідності в комп'ютері використовують програмно-керований обмін окремими байтами (він не вимагає програмування КПДП) і обмін масивами у режимі ПДП (рис. 6.9, в).

6.5.4 Канальний обмін даними

В універсальних комп'ютерах обмін інформацією між ОП і ПП забезпечують спеціальні пристрої - КВВ або просто канали (рис. 6.10).

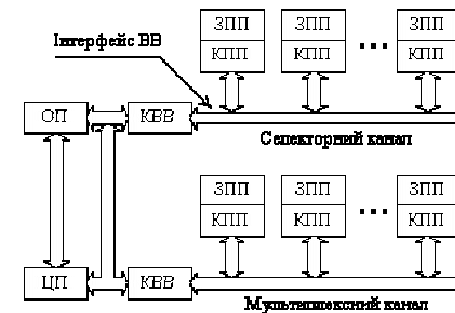


Рисунок 6.10 - Канальна структура комп'ютера

В каналах використовують два режими обміну інформацією: монопольний та розподілу в часі.

Монопольний режим реалізують селекторним каналом, а розподілу в часі - мультиплексним каналом. Засоби каналу, які призначені для обслуговування одного ПП, називаються підканалом.

Селекторні канали обслуговують швидкодіючі пристрої, в першу чергу ЗЗП (накопичувачі на дисках і магнітних стрічках). Селекторний

канал має один підканал. Після встановлення зв'язку він монополярно обслуговує тільки один ПП: інші пристрої чекають закінчення даної програми обміну.

У малих комп'ютерах використовують один селекторний канал, у великих - до шести. Мультиплексний канал (один в машині) паралельно обслуговує сотні повільно діючих ПП в режимі розподілу часу (клавіатура, принтери, перфатори тощо.).

6.6 Виконання команд процесором

Основна функція будь-якого процесора, заради якої він і створюється - це виконання команд. Система команд визначає логіку роботи процесора і його реакцію на ті або інші комбінації зовнішніх подій.

Найкомпактніші і швидкі програми і підпрограми створюються на мові Асемблер, яка є символьним записом цифрових кодів машинної мови, кодів команд процесора.. Часто застосовують мови програмування високого рівня, окрема C, C++ з включенням фрагментів на мові Асемблер.

Знання системи команд і мови Асемблер дозволяє підвищити ефективність деяких найважливіших частин програмного забезпечення будь-якої мікропроцесорної системи. А можливість включати окремі фрагменти на мові Асемблера в програму на мові програмування високого рівня дозволяє поєднати переваги цих мов.

Кожна команда, що вибирається (читається) з пам'яті процесором, визначає алгоритм поведінки процесора на найближчі декілька тактів. Код команди (код операції, інструкції) визначає операцію, яку належить виконати процесору і з якими операндами (тобто кодами даних), де узяти початкову інформацію для виконання команди і куди помістити результат.

Код команди (код операції) може займати від одного до декількох байт. Процесор дізнається про те, скільки байт команди йому треба читати з першого прочитаного ним байта або слова. В процесорі код команди розшифровується і перетворюється в набір мікрооперацій, виконуваних окремими вузлами процесора.

6.7 Адресація операндів

Багато команд процесора працює з кодами даних (операндами). Одні команди вимагають вхідних операндів (одного або двох), інші видають вихідні операнди (частіше один операнд). Вхідні операнди називають операндами-джерелами, а вихідні називають операндами-приймачами. Вхідні і вихідні коди операндів визначають місце розташування даних.

Вони можуть знаходитися у внутрішніх регістрах процесора (найзручніший і швидкий варіант), у системній пам'яті (найпоширеніший варіант). Нарешті, вони можуть знаходитися в пристроях вводу/виводу.

Визначення місця положення операндів проводиться кодом команди. Існують різні методи, за допомогою яких код команди може визначити, звідки брати вхідний операнд і куди надсилати вихідний операнд. Ці методи називаються методами адресації. Ефективність вибраних методів адресації багато в чому визначає ефективність роботи всього процесора в цілому.

6.8 Методи адресації

Кількість методів адресації в різних процесорах може бути від 4 до 16. Розглянемо декілька типових методів адресації операндів, що використовуються зараз в більшості мікропроцесорів.

6.8.1 Безпосередня адресація

Безпосередня адресація (Рис. 6.11) припускає, що операнд (вхідний) знаходиться в пам'яті безпосередньо за кодом команди. Операнд зазвичай є константою, яку треба кудись переслати, до чогось додати тощо.

Наприклад, команда може полягати в тому, щоб додати число 6 до вмісту якогось внутрішнього реєстра процесора. Це число 6 розташовуватиметься в пам'яті, усередині програми в адресі, наступній за кодом даної команди складання.



Рисунок 6.11 - Безпосередня адресація.

6.8.2 Пряма адресація

Пряма (або абсолютна) адресація (Рис. 6.12) припускає, що операнд (вхідний або вихідний) знаходиться в пам'яті за адресою, код якого знаходиться усередині програми зразу ж за кодом команди.

Наприклад, команда може полягати в тому, щоб очистити (зробити нульовим) вміст елемента пам'яті з адресою 1000000. Код цієї адреси

1000000 розташовуватиметься в пам'яті, усередині програми в наступній адресі за кодом даної команди очищення.



Рисунок 6.12 - Пряма адресація.

6.8.3 Регістрова адресація

Регістрова адресація (Рис. 6.13) припускає, що операнд (вхідний або вихідний) знаходиться у внутрішньому реєстрі процесора. Наприклад, команда може полягати в тому, щоб переслати число з нульового реєстра в перший. Номери обох реєстрів (0 і 1) визначатимуться кодом команди пересилки.

Непрямо-регістрова (вона ж непряма) адресація припускає, що у внутрішньому реєстрі процесора знаходиться не сам операнд, а його адреса в пам'яті (Рис. 6.14). Наприклад, команда може полягати в тому, щоб очистити елемент пам'яті з адресою, що знаходиться в нульовому реєстрі. Номер цього реєстра (0) визначатиметься кодом команди очищення.



Рисунок 6.13 - Регістрова адресація

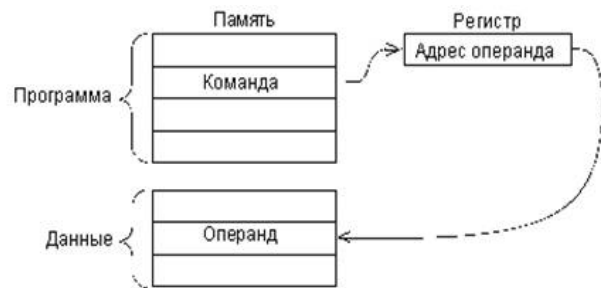


Рисунок 6.14 - Непряма адресація.

6.8.4 Автоінкрементна і та автодекрементна адресація

Автоінкрементна адресація дуже близька до непрямой адресації, але відрізняється від неї тим, що після виконання команди вміст регістра, що використовується, збільшується на одиницю або на два. Цей метод адресації дуже зручний, наприклад, при послідовній обробці кодів з масиву даних, що знаходиться в пам'яті. Після обробки коду адреса в регістрі посилає на наступний код з масиву. При використанні непрямой

адресації в даному випадку довелося б збільшувати вміст цього регістра окремою командою.

Автодекрементна адресація працює схоже на автоінкрементну, але тільки вміст вибраного регістра зменшується на одиницю або на два перед виконанням команди. Ця адресація також зручна при обробці масивів даних. Сумісне використання автоінкрементної і автодекрементної адресації дозволяє організувати пам'ять стекового типу.

6.8.5 Індексна адресація

Поширеними є індексні методи, які припускають для обчислення адреси операнда надбавку до вмісту регістра заданої константи (індексу). Код цієї константи розташовується в пам'яті безпосередньо за кодом команди.

Вибір того або іншого методу адресації в значній мірі визначає час виконання команди. Найшвидша адресація - це регістрова, оскільки вона не вимагає додаткових циклів обміну по магістралі. Якщо ж адресація вимагає звернення до пам'яті, то час виконання команди збільшуватиметься за рахунок тривалості необхідних циклів звернення до пам'яті. Чим більше внутрішніх регістрів у процесора, тим частіше і вільніше можна застосовувати регістрову адресацію, і тим швидше працюватиме система в цілому.

6.9 .Сегментація пам'яті

Сегментація пам'яті застосовується у деяких процесорах, наприклад в процесорах IBM PC-сумісних персональних комп'ютерів.

В процесорі Intel 8086 сегментація пам'яті організована таким чином. Вся пам'ять системи подається не у вигляді безперервного простору, а у вигляді кількох областей - сегментів заданого розміру (по 64 Кбайта), положення яких в просторі пам'яті можна змінювати програмним шляхом. Для зберігання кодів адрес пам'яті використовуються не окремі регістри, а пари регістрів:

сегментний регістр, який визначає адресу початку сегменту (тобто положення сегменту в пам'яті);

регістр покажчика (регістр зсуву), який визначає положення робочої адреси усередині сегменту.

Фізична 20-розрядна адреса пам'яті, що виставляється на зовнішню шину адреси, утворюється так, як показано на рисунку 6.15, тобто шляхом складання зсуву і адреси сегменту із зсувом на 4 біти. Положення цієї адреси в пам'яті показано на рисунку .

Сегмент може починатися тільки на 16-байтній межі пам'яті (оскільки адреса початку сегменту, по суті, має чотири молодші нульові розряди, як видно з рисунок 6.16), тобто з адреси, кратної 16. Ці допустимі межі сегментів називаються межами параграфів.

Введення сегментації, перш за все, пов'язано з тим, що внутрішні регістри процесора 16-розрядні, а фізична адреса пам'яті 20-розрядна (16-розрядна адреса дозволяє використовувати пам'ять тільки в 64 Кбайт, що явно недостатньо).

В процесорі MC68000 фірми Motorola внутрішні регістри 32-розрядні, тому проблеми сегментації пам'яті не виникає.

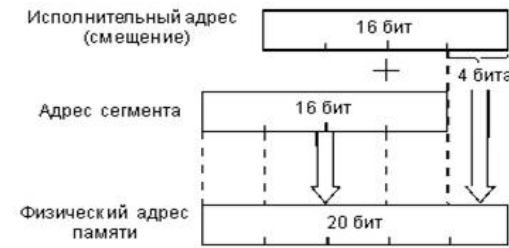


Рисунок 6.15 - Формування фізичної адреси пам'яті з адреси сегменту і зсуву.



Рисунок 6.16 - Фізична адреса в сегменті (всі коди - шістнадцяткові).

Застосовуються і складніші методи сегментації пам'яті. Наприклад, в процесорі Intel 80286 в так званому захищеному режимі адреса пам'яті обчислюється відповідно до рис. 6.17. В сегментному регістрі в даному випадку зберігається не базова (початкова) адреса сегментів, а коди селекторів, що визначають адреси в пам'яті, по яких зберігаються

дескриптори (тобто описувачі) сегментів. Область пам'яті з дескрипторами називається таблицею дескрипторів. Кожний дескриптор сегменту містить базову адресу сегменту, розмір сегменту (від 1 до 64 Кбайт) і його атрибуту. Базова адреса сегменту має розрядність 24 біт, що забезпечує адресацію 16 Мбайт фізичної пам'яті.

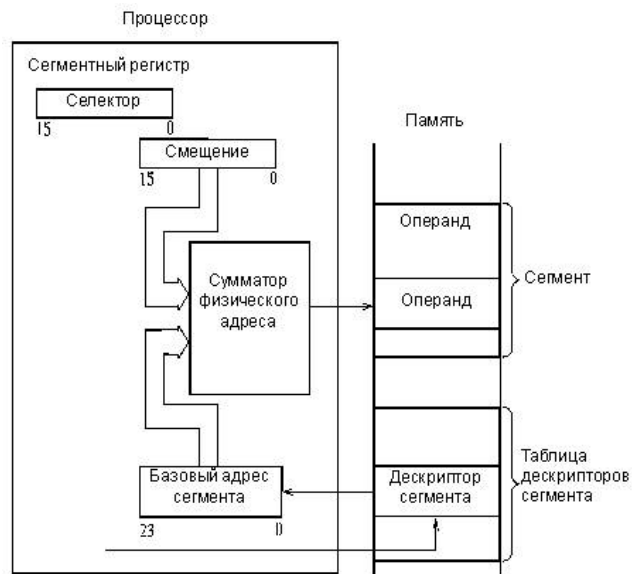


Рисунок 6.17 - Адресація пам'яті в захищеному режимі процесора Intel 80286

Таким чином, на суматор, що обчислює фізичну адресу пам'яті, подається не вміст сегментного регістра, як у попередньому випадку, а базова адреса сегменту з таблиці дескрипторів.

Ще складніший метод адресації пам'яті з сегментацією використаний в процесорі Intel 80386 і в більш пізніх моделях процесорів фірми Intel. Цей метод ілюструється рис. 6.18. Адреса пам'яті (фізична адреса) обчислюється в три етапи. Спочатку обчислюється так звана ефективна адреса (32-розрядний) шляхом підсумовування трьох компонентів: бази, індексу і зсуву (Base, Index, Displacement), причому можливе множення індексу на масштаб (Scale).

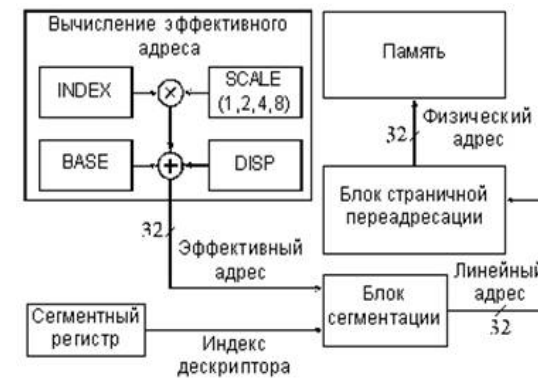


Рисунок 6.18 - Формування фізичної адреси пам'яті процесора 80386 в захищеному режимі.

Ці компоненти мають наступний зміст:

зсув є 8-, 16- або 32-розрядним числом, включеним у команду;

база, яка являє собою вміст базового регістра процесора. Зазвичай воно використовується для вказівки на початок деякого масиву;

індекс, який відображає вміст індексного реєстра процесора. Зазвичай воно використовується для вибору одного з елементів масиву;

масштаб, який є множником (він може бути рівний 1, 2, 4 або 8), вказаний в коді команди, на який, перед додаванням з іншими компонентами, домножається індекс. Він використовується для вказівки розміру елемента масиву.

Потім спеціальний блок сегментації обчислює 32-розрядну лінійну адресу, яка є сумою базової адреси сегменту з сегментним реєстром з ефективною адресою.

Нарешті, фізична 32-бітова адреса пам'яті утворюється шляхом перетворення лінійної адреси блоком сторінкової переадресації, який здійснює переклад лінійної адреси у фізичний сторінками по 4 Кбайта.

У будь-якому випадку сегментація дозволяє виділити в пам'яті один або кілька сегментів для даних і один або кілька сегментів для програм. Перехід від одного сегменту до іншого зводиться всього лише до зміни вмісту сегментного реєстра. Іноді це дуже зручно, але для програміста працювати з сегментованою пам'яттю зазвичай складніше, ніж з безперервною, несегментованою пам'яттю, оскільки доводиться стежити за межами сегментів, за їх описом, перемиканням тощо.

6.10 Адресація байтів і слів

Багато процесорів, що мають розрядність 16 або 32, здатні адресувати не тільки ціле слово в пам'яті (16-розрядне або 32-розрядне), але і окремі байти. Кожному байту в кожному слові при цьому відводиться своя адреса. Так, у разі 16-розрядних процесорів всі слова в пам'яті (16-розрядні) мають парні адреси. А байти, що входять в ці слова, можуть мати

як парні адреси, так і непарні. Наприклад, нехай 16-розрядний елемент пам'яті має адресу 23420, і в ній зберігається код 2A5E (рис. 6.19).

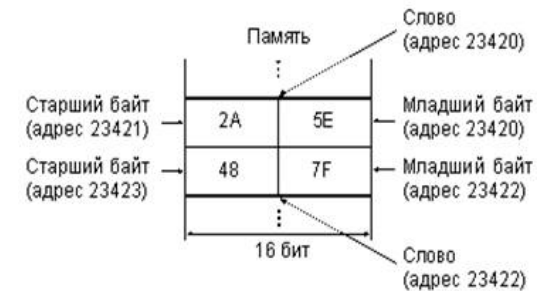


Рисунок 6.19 - Адресація слів і байтів.

При зверненні до цілого слова (з вмістом 2A5E) процесор виставляє адресу 23420. При зверненні до молодшого байта цієї комірки (з вмістом 5E) процесор виставляє ту ж саму адресу 23420, але використовує команду, що адресує байт, а не слово. При зверненні до старшого байта цієї комірки (з вмістом 2A) процесор виставляє адресу 23421 і використовує команду, що адресує байт.

Наступний по порядку 16-розрядний елемент пам'яті з вмістом 487F матиме адресу 23422, тобто знову ж таки парний. Її байти матимуть адреси 23422 і 23423.

Для розрізнення байтових і словних циклів обміну на магістралі в шині управління передбачається спеціальний сигнал байтового обміну. Для роботи з байтами в систему команд процесора вводяться спеціальні команди або передбачаються методи байтової адресації.

6.11 Регістри процесора

Внутрішні регістри процесора є надоперативною пам'яттю невеликого розміру, яка призначена для тимчасового зберігання службової інформації або даних. Кількість регістрів в різних процесорах може бути від 6-8 до декількох десятків.

Регістри можуть бути універсальними і спеціалізованими. Спеціалізовані регістри, які присутні в більшості процесорів. Зокрема, це регістр-лічильник команд, регістр стану (PSW), регістр покажчика стека. Решта регістрів процесора може бути як універсальними, так і спеціалізованими.

Наприклад, в 16-розрядному процесорі T-11 фірми DEC було 8 регістрів загального призначення і один регістр стану. Всі регістри мали по 16 розрядів. З регістрів загального призначення один відводився під лічильник команд, інший - під покажчик стека. Вся решта регістрів загального призначення повністю взаємозамінна, тобто мають універсальне призначення, можуть берегти як дані, так і адреси (показчики), індекси тощо. Максимально допустимий об'єм пам'яті для даного процесора складав 64 Кбайт (адреса пам'яті 16-розрядна).

В 16-розрядному процесорі MC68000 фірми Motorola було 19 регістрів: 16-розрядний регістр стану, 32-розрядний регістр лічильника команд, 9 регістрів адреси (32-розрядних) і 8 регістрів даних (32-розрядних). Два регістри адреси відведено під покажчики стека. Максимально допустимий об'єм пам'яті, що адресується, 16 Мбайт (зовнішня шина адреси 24-розрядна). Всі 8 регістрів даних взаємозамінні. 7 регістрів адреси - теж взаємозамінні.

В 16-розрядному процесорі Intel 8086, який став базовим в лінії процесорів, що використовуються в персональних комп'ютерах,

реалізований принципово інший підхід. Кожний регістр цього процесора має своє особливе призначення, і замінювати один одного регістри можуть тільки частково або ж не можуть взагалі.

Процесор 8086 має 14 регістрів розрядністю по 16 біт. З них чотири регістри (AX, BX, CX, DX) - це регістри даних, кожний з яких крім зберігання операндів і результатів операцій має ще і своє специфічне призначення:

регістр AX - множення, розподіл, обмін з пристроями вводу/виводу (команди введення і висновку);

регістр BX - базовий регістр в обчисленнях адреси;

регістр CX - лічильник циклів;

регістр DX - визначення адреси вводу/виводу.

Для регістрів даних існує можливість роздільного використання обох байтів. Наприклад, для регістра AX вони мають позначення AL - молодший байт і AH - старший байт. Наступні чотири внутрішні регістри процесора - це сегментні регістри, кожний з яких визначає положення одного з робочих сегментів (рис. 6.20):

регістр CS (Code Segment) відповідає сегменту команд, виконуваних в даний момент;

регістр DS (Data Segment) відповідає сегменту даних, з якими працює процесор;

регістр ES (Extra Segment) відповідає додатковому сегменту даних;

регістр SS (Stack Segment) відповідає сегменту стека.



Рисунок 6.20 - Сегменты команд, данных і стека в пам'яті.

Всі ці сегменти можуть перекриватися для оптимального використання простору пам'яті. Наприклад, якщо програма займає тільки частину сегменту, то сегмент даних може починатися відразу після завершення роботи програми (з точністю 16 байт), а не після закінчення всього сегменту програми.

Наступні п'ять реєстрів процесора (SP - Stack Pointer, BP - Base Pointer, SI - Source Index, DI - Destination Index, IP -Instruction Pointer) служать покажчиками (тобто визначають зсув в межах сегменту). Наприклад, лічильник команд процесора утворюється парою реєстрів CS і IP, а покажчик стека - парою реєстрів SP і SS.

Реєстри SI, DI використовують в рядкових операціях, тобто при послідовній обробці декількох елементів пам'яті однією командою.

Реєстр FLAGS є реєстром стану процесора (PSW). З його 16 розрядів використовуються тільки дев'ять (рис. 6.21): CF (Carry Flag) - прапор перенесення при арифметичних операціях, PF (Parity Flag) - прапор

парності результату, AF (Auxiliary Flag) - прапор додаткового перенесення, ZF (Zero Flag) - прапор нульового результату, SF (Sign Flag) - прапор знака (співпадає із старшим бітом результату), TF (Trap Flag) - прапор покрокового режиму (використовується при відладці), IF #@: - прапор дозволу апаратних переривань, DF #@; - прапор напряму при рядкових операціях, #@

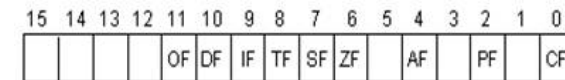


Рисунок 6.21 - Реєстр стану процесора 8086

Біти реєстра стану встановлюються або очищаються залежно від результату виконання попередньої команди і використовуються деякими командами процесора. Біти реєстра стану можуть також встановлюватися і очищатися спеціальними командами процесора.

В багатьох процесорах виділяють спеціальний реєстр, так званий акумулятор (тобто накопичувач). Зазвичай тільки цей реєстр-акумулятор може брати участь у всіх операціях, тільки через нього може проводитися взаємодія з пристроями вводу/виводу.

Іноді в цей реєстр вміщують результат виконаної команди (в цьому випадку говорять навіть про «акумуляторну» архітектуру процесора). Наприклад, в процесорі 8086 реєстр даних AX можна вважати своєрідним акумулятором, оскільки саме він обов'язково бере участь в командах множення і ділення, а також тільки через нього можна пересилати дані в пристрій вводу/виводу і з пристрою вводу/виводу.

Виділення спеціального регістра-акумулятора спрощує структуру процесора і прискорює пересилки кодів усередині процесора, але в деяких випадках уповільнює роботу системи в цілому, оскільки весь потік інформації повинен пройти через один регістр-акумулятор. У разі, коли кілька регістрів процесора повністю взаємозамінні, таких проблем не виникає.

7 ПРИНЦИПИ ПРОЕКТУВАННЯ СХЕМ

Автоматизація проектування цифрових пристроїв і мікросхем сприяла стандартизації мов проектування і уніфікації опису, моделювання схем, розробці загальних принципів проектування. У розділі міститься інформація для курсових робіт над створенням опису архітектури і поведінки пристроїв з врахуванням особливостей подання даних. Наведені приклади можуть бути основою розв'язання задач для самостійного опрацювання і контрольних робіт.

7.1 Методи аналізу і синтезу цифрових пристроїв

7.1.1 Аналіз і синтез комбінаційних схем

Методи синтезу комбінаційних схем є основою проектування цифрових пристроїв. Методи ґрунтуються на теорії перемикальних (булевих) функцій і передбачають застосування типових комбінаційних схем (дешифраторами, шифраторами, мультиплексорами, демультиплексорами, суматорами тощо), а наведені приклади відповідають завданням та практичним задачам контрольних робіт та екзамену.

7.1.2 Метод Квайна – Мак-Класки мінімізації функції

Метод дозволяє отримати покриття функції і ґрунтується на застосуванні відношень склеювання і поглинання.

$$\begin{aligned}x_1x_2 \vee x_1\overline{x_2} &= x_1 & x_1 \vee x_1x_2 &= x_1 \\(x_1 \vee x_2)(x_1 \vee \overline{x_2}) &= x_1 & (x_1 \vee x_2)x_1 &= x_1\end{aligned}$$

Вихідною формою функції є ДДНФ.

Етапи мінімізації:

- 1) записати функцію у вигляді ДДНФ
- 2) застосувати співвідношення склеювання послідовно, попарно для всіх конституент одиниці, а потім до імплікант (n-1)-го рангу, (n-2)-го рангу тощо. поки можливе одержання нових імплікант
- 3) виконати всі можливі поглинання, внаслідок чого визначаються всі прості імпліканти (диз'юнкція простих імплікант дасть скорочену ДНФ)
- 4) побудувати матрицю покриття (імплікантну матрицю Квайна) і знайти тупиковий ДНФ. Вибрати мінімальну ДНФ з числа тупикових.

7.1.3 Алгоритм Квайна – Мак-Класки

Для машинної реалізації метода застосовують алгоритм пошуку мінімізованих виконують у такій послідовності.

- 1) Знаходять покриття $\Pi(z)$ у такому порядку:
 - а) сформувати кубічний комплекс функції;
 - б) в кожному i -му кубічному комплексі відзначають куби (прості імпліканти), що не утворили жодного з кубів $i+1$ –го кубічного комплексу.Отримані прості імпліканти утворюють покриття функції.
- 2) Отримують таблицю покриттів Квайна, у якій рядками є прості імпліканти, а стовпчики відповідають 0-кубам функції (1-конституентам). На перетині i –го рядка і j – го стовпчика відмічають покриття конституенти простою імплікантою
- 3) Визначають покриття мінімальної вартості. Для цього

а) виділяють ядро Квайна. Якщо 0-куб покривається лише однією простою імплікантою, то ця імпліканта є істотною і входить до ядра Квайна, а отже і до покриття мінімальної вартості;

б) з таблиці Квайна викреслюють рядки з істотними простими імплікантами і стовпчики, покриті цими простими імплікантами (елементами ядра Квайна). Якщо в отриманій скороченій таблиці є істотні прості імпліканти, то їх включають до ядра Квайна і знов модифікують таблицю Квайна, і так далі;

в) стискають отриману функцію по стовпчиках, викреслюючи стовпчики, які поглинаються іншими за простими імплікантами;

г) стискають таблицю по рядках, для чого викреслюють рядки, що цілком входять в інші рядки;

д) після виконання всіх стискань по стовпчиках і рядках та включення істотних простих імплікант в ядро Квайна, отримують циклічну таблицю Квайна, прості імпліканти якої можуть входити до покриття мінімальної вартості.

4) Отримують всі можливі тупикові форми покриття функції на основі ядра Квайна і циклічної таблиці Квайна. Для цього у форму покриття функції включають всі імпліканти ядра і набір простих імплікант циклічної таблиці, які забезпечують покриття всіх 0-кубів циклічної таблиці.

5) З тупикових форм покриттів функції вибирають покриття диз'юнктивної форми мінімальної вартості. Таких форм може бути декілька.

Приклад. Мінімізувати функцію чотирьох змінних $Z(x) = \vee (0, 1, 2, 4, 5, 7, 8, 10, 12, 14, 15)$.

Згідно з алгоритмом виконуємо послідовність перетворень.

1. Формуємо кубічний комплекс функції $K(Z)$. Для зручності розділяємо конституенти одиниці на групи з однаковою кількістю одиниць у кожній групі. Формування комплексу $K_0(Z)$ показано у табл.7.1.

Таблиця 7.1 - Кубічний комплекс $K_0(Z)$

Номер 0-куба	0-куби за кількістю одиниць у поданні 1-конституент				
	0	1	2	3	4
1	0000	0001	0101	0111	1111
2		0010	1010	1110	
3		0100	1100		
4		1000			

Формування комплексу $K_1(Z)$ показано у табл. 7.2.

Таблиця 7.2 - Формування комплексу $K_1(Z)$

Номер 1-куба	1-куби за кількістю одиниць у поданні конституент			
	0 (0-1)	1 (1-2)	2 (2-3)	3 (3-4)
1	000-	0-01	01-1 *	-111 *
2	00-0	-010	1-10	111- *
3	0-00	010-	11-0	
4	-000	-100		
5		1-00		
6		10-0		

Формування комплексу $K_2(Z)$ показано у табл.7.3.

Таблиця 7.3 - Формування комплексу $K_2(Z)$

Номер 2-куба	2-куби за кількістю одиниць у поданні конституент		
	0 ((0-1)-(1-2))	1 ((1-2)-(2-3))	2 ((2-3)-(3-4))
1	0-0- *	1- -0 *	
2	-0-0 *		
3	0-0- *		
4	--00 *		
5			
6			

2. Куби, які не утворили куби вищого рангу, є простими імплікантами і формують покриття функції

$$\Pi(Z)=(01-1, -111, 111-, 0-0-, --00, -0-0, 1- - 0)$$

3. На основі покриття $\Pi(Z)$ будемо таблицю покриттів Квайна (табл. 7.4).

Прості імпліканти 0-0-, -0-0 є істотними і входять до ядра Квайна, оскільки лише вони покривають відповідно 0-куби 0001 (1) і 0010 (2). Тому модифікуємо таблицю з виключенням стовпчиків 0-кубів 0, 1, 2, 4, 5, 8, 10. В результаті отримуємо скорочену циклічну таблицю Квайна (табл. 7.5).

Таблиця 7.4 -Таблиця Квайна

Прості імпліканти	0-куби										
	0	1	2	4	5	7	8	10	12	14	15
01-1					*	*					
-111						*					*
111-										*	*
0-0-	*	*		*	*						
-- 00	*			*			*		*		
-0-0	*		*				*	*			
1--0							*	*	*	*	

Таблиця 7.5 - Скорочена таблиця Квайна

Прості імпліканти				
	7	12	14	15
01-1	*			
-111	*			*
111-			*	*
-- 00		*		
1--0		*	*	

Після поглинання імплікант отримуємо таблицю 6 покриттів Квайна

Таблиця 7.6 - Таблиця покриттів Квайна

Прості імпліканти				
	7	12	14	15
-111	*			*
1--0		*	*	

Тобто отримана функція є єдиною

$$Y = (0-0-, -0-0, -111, 1—0) = \bar{x}_3 \bar{x}_1 + \bar{x}_2 \bar{x}_0 + x_2 x_1 x_0 + x_3 x_0$$

7.2 Аналіз і синтез послідовнісних схем

Методи синтезу послідовнісних схем є основою проектування цифрових пристроїв із складною поведінкою за наявності в них елементів пам'яті (зокрема, тригерів, регістрів, статичних і динамічних елементів та схем пам'яті).

Канонічні методи синтезу передбачають виділення комбінаційної і керувальної частини пристрою з подальшим їх об'єднанням. Керувальна частина описується кількома способами, зокрема абстрактними (Мілі, Мура) і структурними автоматами. Оскільки найпростішими послідовнісними схемами є тригери, то багато задач аналізу і синтезу орієнтовані на опис конкретних типів тригерів.

7.2.1 Аналіз тригерних схем

Розглянемо побудову графа переходів асинхронного RS-тригера на елементах І-НЕ. Обґрунтувати позначення RS-тригера на схемі.

За результатами аналізу поведінки тригера таблиця 7.7 переходів RS-тригера на елементах І-НЕ має такий вигляд.

Таблиця 7.7 – Таблиця переходів RS-тригера на елементах І-НЕ

$f_1(t)$	$f_2(t)$	Q(t)	Q(t+1)
0	0	0	-
0	1	0	1
1	0	0	0
1	1	0	0
0	0	1	-
0	1	1	1
1	0	1	0
1	1	1	1

Рискою позначено невизначений стан тригера, тобто неприпустиме значення сигналів $f_1 = 0, f_2 = 0$.

Інвертування вхідних сигналів f_1, f_2 приводить до таблиці переходів зазвичайго асинхронного RS-тригера.

Для отримання функції збудження тригера необхідно визначити значення сигналів f_1, f_2 , за яких здійснюється перехід зі стану Q(t) у стан Q(t+1). Отримана таблиця 7.8 функції збудження RS-тригера на елементах І-НЕ має такий вигляд.

Таблиця 7.8 – Таблиця функції збудження RS-тригера

Q(t)	Q(t+1)	f_1	f_2
0	0	1	*
0	1	0	1
1	0	1	0
1	1	*	1

Зірочка позначає довільний стан (0 або 1) відповідного сигналу.

Звідси отримуємо граф переходів зі стану Q(t) у стан Q(t+1) RS-тригера на елементах І-НЕ, показаний на рис. 7.1 б.

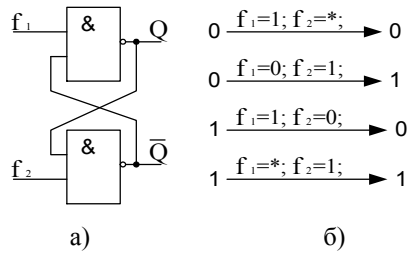


Рисунок 7.1 – Схема і граф переходів RS-тригера

Після інвертування вхідних сигналів f_1 , f_2 таблиці переходів асинхронного RS-тригера на елементах І-НЕ отримаємо таку таблицю переходів (x' позначено інвертоване значення x).

Таблиця 7.9 відповідає таблиці переходів звичайного асинхронного RS-тригера за умови, що сигнали f_1 , f_2 відповідають сигналам R, S. Тому на схемах асинхронний RS-тригер на елементах І-НЕ позначають з інвертованими вхідними сигналами R, S, як показано на рис. 7.2.

Таблиця 7.9 – Таблиця переходів асинхронного RS-тригера

$f_1(t)$	$f_2(t)$	Q(t)	Q(t+1)
1	1	0	-
1	0	0	1
0	1	0	0
0	0	0	0
1	1	1	-
1	0	1	1
0	1	1	0
0	0	1	1

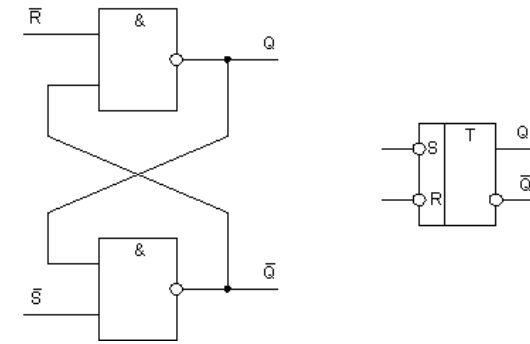


Рисунок 7.2 – Схемна реалізація і позначення асинхронного RS-тригера на елементах І-НЕ

7.2.2 Аналіз схем із застосуванням скінченних автоматів

Аналіз і синтез послідовнісних схем часто виконують із застосуванням скінченних автоматів.

Виконаємо кодування входів, виходів, станів асинхронного RS-тригера і подамо його автоматом Мура у вигляді таблиць переходів, виходів, графа.

Таблиця переходів RS-тригера має такий вигляд (таблиця 7.10).

Таблиця 7.10 – Таблиця переходів асинхронного RS-тригера

R	S	Q(t)	Q(t+1)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	-
0	0	1	1
0	1	1	1
1	0	1	0
1	1	1	1

Формуємо алфавіти автомата Мура. Кодуємо припустимі набори вхідних сигналів RS, внутрішніх станів Q та вихідного сигналу Y, який асоціюється з виходом Q тригера, буквами алфавітів X, Z, Y абстрактного автомата Мура.

R	S	Алфавіт
0	0	X ₀
0	1	X ₁
1	0	X ₂

Q	Алфавіт
0	Z ₀
1	Z ₁

Y	Алфавіт
0	Y ₀
1	Y ₁

На основі таблиці переходів тригера отримаємо таблицю функції переходів і виходів автомата Мура (таблиця 7.11).

Таблиця 7.11 – Таблиця функції переходів і виходів автомата Мура

X	Z, Y	
	Z ₀ , Y ₀	Z ₁ , Y ₁
X ₀	Z ₀	Z ₁
X ₁	Z ₁	Z ₁
X ₂	Z ₀	Z ₀

Отримаємо граф переходів (рис. 7.3).

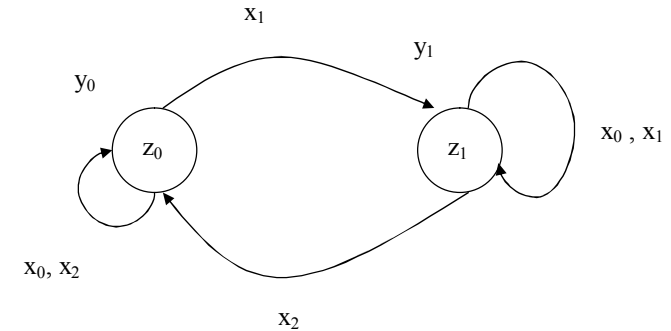


Рисунок 7.3 – Граф переходів тригера RS

Аналіз і синтез мікропрогамних автоматів більш складних послідовнісних пристроїв потребує складних моделей, які будують з використанням спеціальних програмних засобів.

7.3 Класифікація методів і засобів проектування цифрових пристроїв

7.3.1 Класифікація цифрових інтегральних схем

Основними підходами проектування цифрових пристроїв (ЦП) є розробка цих пристроїв на основі існуючих мікросхем або створення нових мікросхем, які відповідають особливостям роботи пристроя. Перший підхід передбачає створення невеликої кількості екземплярів (до кількох десятків або сотен) пристроїв з можливою подальшою модифікацією або обслуговуванням (заміна непрацездатних компонентів, незначна перебудова структури або заміна окремих частин більш сучасними за технічними характеристикам у процесі експлуатації).

Другий підхід є характерним для пристроїв, призначених для виконання типових операцій і значній кількості випуску таких пристроїв.

Так, наприклад, економічно обгрунтовані пробні партії пристроїв в цьому випадку оцінюються тисячами штук.

Вартість проектування цифрових пристроїв на основі типових схем, зокрема великих інтегральних схем (ВІС) і спеціалізованих пристроїв обробки даних, в значній мірі залежить від вартості відповідних комплектуючих на ринку і може складати незначну частину вартості самого пристрою. При проектуванні ВІС масового виробництва затрачаються великі кошти. Наприклад, вартість проектування першого 32-розрядного мікропроцесора склало 140 млн. дол., а ОЗП об'ємом 1 Мбіт склало 395 млн. дол. На рис. 7.4 наведено класифікацію цифрових ІС за методами проектування.

Спеціалізовані схеми передбачають налагодження (напівзамовні схеми) або виготовлення в умовах виробництва ІС (замовні схеми). У напівзамовних користувач сам має можливість міняти логіку роботи схеми, наприклад через програмування функцій. Зокрема, для схеми програмованої логіки МРГА (Mask Programmable Gate Arrays) із матриці програмованих логічних блоків, між рядками і стовпчиками встановлюються програмовані зв'язки з використанням малої кількості фотошаблонів для виготовлення, що дозволяє знизити витрати.

Стандартні схеми передбачають виготовлення ІС великими партіями. Вони включають виготовлення схем великої і малої степені інтеграції (стандартних мікропроцесорів, ОЗП, мікроконтролерів), а також створення схем на основі програмування за технологіями програмованої логіки. До схем із програмованою логікою належать схеми PLD, FPGA, CPLD та їх комбінації.

Програмовані логічні пристрої PLD (Programmable Logic Device) об'єднують схеми програмованої матричної логіки PAL (Programmable Array Logic) і програмованих логічних матриць PLA (Programmable Logic

Array). Схеми PAL реалізують систему перемикальних функцій, представлених у ДНФ, кожна з яких складається із невеликої кількості кон'юнктив. Схеми PLA являють собою реалізацію деякої ДНФ.

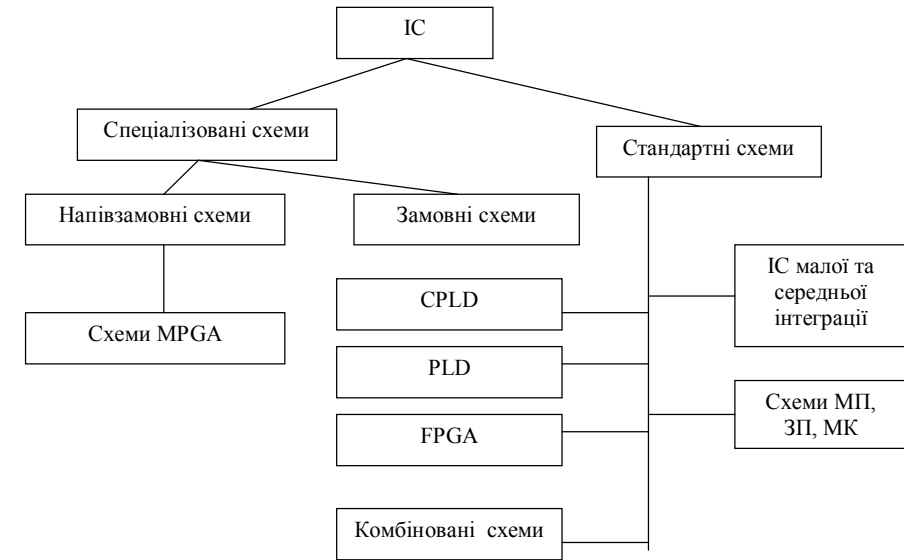


Рисунок 7.4 - Класифікація інтегральних мікросхем за методами проектування

Схеми CPLD (Complex PLD) являють собою структуру із сукупності PAL і перепрограмованої логіки GAL (Generic Array Logic) на основі PLD, які об'єднані матрицею програмованих з'єднань.

Програмована вентильна матриця FPGA (Field Programmable Gate Array) передбачає одноразове або багаторазове перепрограмування і є альтернативою (з 1984 року) схемам PLD та ASIC (Application Specific Integrated Circuits).

Виготовлення стандартних ІС передбачає використання бібліотек стандартних елементів, що спрощує процес проектування, особливо із застосуванням систем автоматизованого проектування (САПР). На рис. 7.5 представлено співвідношення між різними типами ІС за ступенем інтеграції та об'ємами партій схем, які виготовляються.

Застосування програмованої логіки дозволяє значно спростити алгоритми проектування цифрових пристроїв. На рис. 7.6 показана узагальнена послідовність автоматизованого проектування цифрових пристроїв із застосуванням САПР.

Концептуальний синтез дозволяє визначити основні вимоги до функціонування пристроя, множину вхідних і вихідних сигналів, характер і взаємозв'язок складових частин проекту пристрою.

Введення даних у САПР виконується з використанням певних автоматизованих засобів, які зазвичай є складовими мов проектування пристроїв. На основі введених даних виконується компіляція проекту пристроя з отриманням відповідних варіантів реалізації проекту у вибраній елементній базі. Це дозволяє проводити подальший аналіз функціональних і часових параметрів проекту пристрою з метою визначення відповідності заданим вимогам до пристрою і узгодженості різних режимів його використання.

Функціональне і часове моделювання пристрою передбачає перевірку на спроможність виконувати ним заданих функцій з врахуванням часових обмежень роботи окремих його складових елементів і режимів застосування. Виявлені у процесі моделювання помилки і неузгодженості виправляються з подальшим аналізом результату моделювання нового варіанту пристроя. Тобто процес автоматизації проектування являє собою ітеративний процес, який передбачає багаторазове повернення до реалізацій проекту.

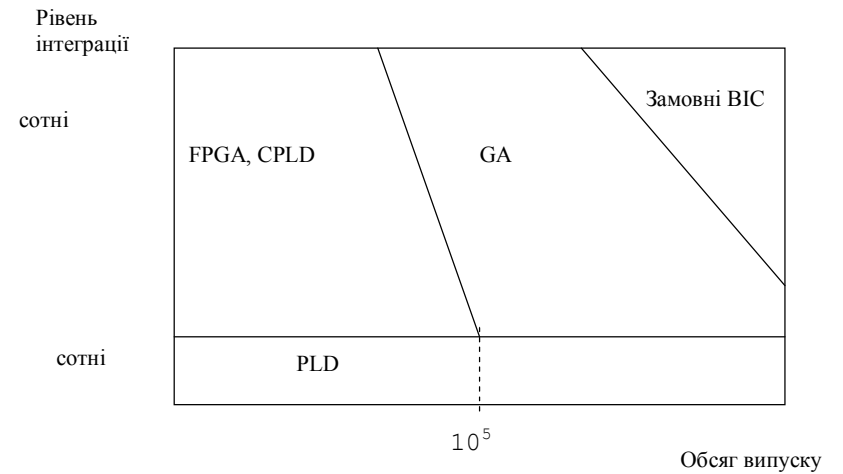


Рисунок 7.5 - Области областей доцільного виготовлення схем

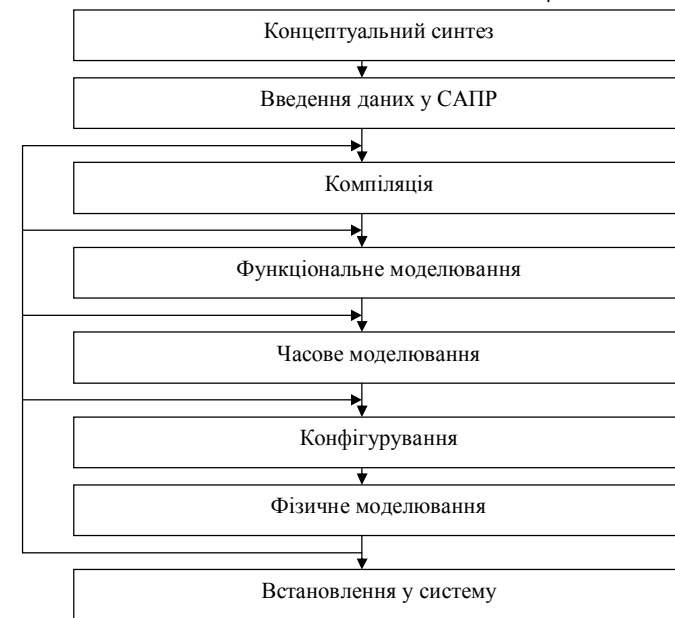


Рисунок 7.6 - Узагальнена послідовність проектування пристроїв

7.3.2 Класифікація засобів и проектування

Сучасні методи проектування ЦП ґрунтуються на комплексному застосуванні засобів мов опису і аналізу проектів пристроїв з метою подальшого прискореного випуску і продажу виробленої продукції. Значне місце у створенні проектів ЦП обіймають САПР із використанням спеціалізованих мов проектування.

Найбільш поширеними універсальними засобами опису ЦП є графічний, текстовий опис і опис у вигляді часових діаграм. Останній засіб зазвичай використовують як додатковий засіб у складі мов автоматизації проектування.

Графічне подання проекту пристроя виконується у заданому базисі бібліотечних елементів. Перевагами графічного подання є традиційність, наочність і звичність для розробників апаратури.

8 МОВИ ПРОЕКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ

8.1 Засоби опису цифрових пристроїв

Сучасні мови опису апаратури HDL (Hardware Description Languages) передбачають опис його поведінки і структури пристроя у текстовому вигляді. Перевагами такого опису є зручність отримання документації на пристрій, компактність і зручність при автоматизації перетворень проекту, зокрема початкової генерації проекту, зміні міжблокових з'єднань тощо. Загальна структура засобів проектування ЦП представлено на рис. 8.1.

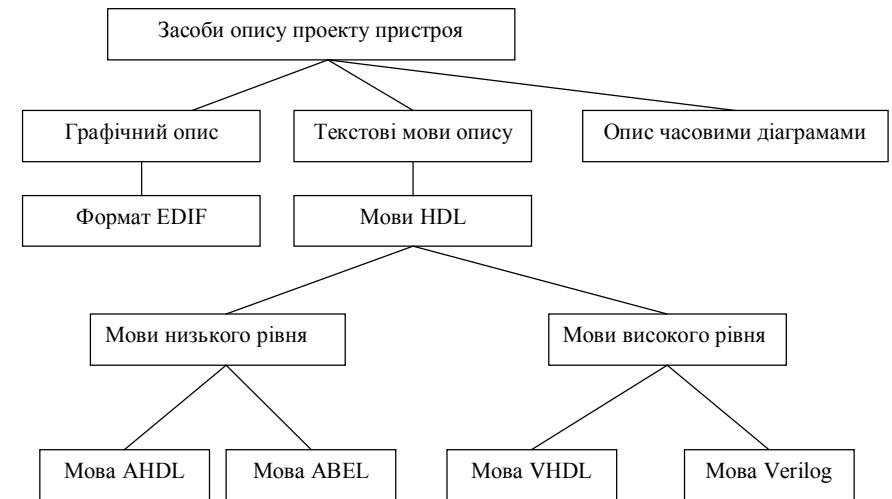


Рисунок. 8.1 - Засоби опису цифрових пристроїв

Мови низького рівня є спеціалізованими і ґрунтуються на певній апаратній платформі окремих виробників. Вони забезпечують отримання більш високих параметрів проекту пристроя. Обмеженням цих мов є

орієнтація при проектуванні на використання певних апаратних рішень, зокрема певного виробника мікросхем

Мови високого рівня є більш універсальними і не пов'язані тісно з певною апаратною платформою. Характерним для них є можливість описувати паралельну дію окремих частин пристрою за умови загального керування. Поведінку пристрою описують алгоритмом у послідовній формі, тобто через послідовність операторів присвоєння і прийняття рішень.

Допоміжними засобами аналізу проектів пристроїв у мовах проектування є використання окремих засобів представлення поведінки у вигляді часових діаграм.

Формат передавання інформації про проект EDIF (Electronic Design Interchange Format) є складовою частиною набору стандартів електронного обміну даними EDI (Electronic Design Interchange) для пересилання документів телекомунікаційними мережами.

Мова AHDL (Altera HDL) є мовою компанії Altera Corporation для програмування своїх пристроїв CPLD і FPGA. Він є подібним до мов VHDL, Verilog за складом операцій моделювання і підтримується компіляторами Altera's Quartus та Max+.

Мова VHDL (Very high speed integrated circuit (VHSIC) є HDL – мовою опису надшвидкісних схем з високим рівнем інтеграції. Мову розроблено у 1980-х роках МО США (стандарт IEEE 1076) вона є подібною за синтаксисом мові ADA. Мова дозволяє описувати подію, структуру системи і декомпозувати систему на підсистеми, моделювати поведінку функціонування системи. Існує кілька версій мови, зокрема VLSI, VHDL-T

Мова Verilog компанії Cadence Design Systems призначена для опису апаратури, ґрунтується на мові C і, частково, Паскалі Вона призначена для опису пристроя на рівні компонентів і плат, а також системи в цілому.

Основою цих мов служать кремнієві компілятори (silicon compilers), вхідні дані для яких являють собою опис електронних схем у САПР з використанням змінних, які задають сигнали або групи сигналів мікросхем. Результатом роботи компілятора є топологія мікросхеми, яка реалізує задану логіку, та супроводжуюча документація.

Поширені мови проектування цифрових пристроїв наведено у таблиці 8.1.

Таблиця 8.1 - Мови проектування цифрових пристроїв

Мова проектування	Назва	Примітка
ABEL	Expression Language	Advanced Boolean Expression Language
AHDL	Altera HDL	a proprietary language from Altera
AHPL	A Hardware language	A Hardware Programing language
Bluespec		HDL with a Verilog syntax
CUPL	Compiler for Program.Logic	language from Logical Devices, Inc.
HJJ	Hardware Join Java	based on Join Java
JHDL		based on Java
Lava		based on Haskell
Lola		a simple language used for teaching
M		A HDL from Mentor Graphics
MyHDL		based on Python
PALASM		for PAL devices
RHDL	Ruby HDL	based on the Ruby programming language
SystemVerilog		a superset of Verilog to address system-level design and verification
Verilog		most widely-used HDL
VHDL	VHSIC HDL	most widely-used HDL

8.2 Етапи проектування пристроїв з використанням систем САПР

У процесі проектування пристроя його зазвичай поділяють на операційний блок (ОБ) і блок керування (БК), який зручно задавати у вигляді автомату керування. ОБ виконує перетворення даних і складається із стандартних частин: регістрів, суматорів, лічильників, мультиплексорів, демультимплексорів, дешифраторів тощо. БК подає на входи ОБ керувальні сигнали, які забезпечують сумісну роботу елементів ОБ у заданій послідовності. Для складних пристроїв можливим є виділення кількох частин ОБ-БК, які об'єднують між собою.

Основними етапами проектування є такі.

1. Складання змістовної граф-схеми алгоритму або функціональної блок- схеми пристроя.

На цьому етапі виконують декомпозицію задачі на підзадачі, вибір основних функціональних блоків і їх зв'язків між собою. Етап ґрунтується на виконанні вимог технічного завдання до пристроя.

2. Розробка загальної структури операційного блока.

Визначається ієрархія елементів певного рівня та їх зв'язки між собою.

3. Опис роботи автомату керування.

На етапі визначають функціонування пристрою із використанням вибраних методів опису роботи автомата. Сучасним є застосування на цьому етапі переходу від словесного опису пристрою до графічного подання з використанням графічних редакторів. Опис пристрою проводиться у вигляді граф-схем переходів (діаграми станів пристрою), яка задає поведінку автомата. Зокрема, поширеним є застосування програми Foundation фірми Xilinx та програм типу StateCAD Version 3.2

пакету Workview Office фірми Viewlogic. Остання, наприклад, дозволяє виконувати такі операції:

- малювати граф переходів з позначенням станів, умов і пріоритетів переходів, сигналів, що формуються;

- перевіряти коректність складання графу переходів;

- компілювати проект;

- моделювати поведінку автомата в інтерактивному режимі або у режимі компіляції.

Результати роботи програми подаються у формах інших мов, зокрема VHDL, Verilog, ABEL, AHDL.

4. Компіляція проекту пристроя.

Компіляція проекту включає перевірку проекту на неузгодженість роботи, наявність помилок і виконується як послідовність підетапів: збирання бази даних проекту, контролю з'єднань, логічної мінімізації проекту, формування файлів конфігурування програмованих мікросхем.

5. Тестування проекту.

Етап дозволяє виконувати моделювання роботи пристроя і виявити помилки функціонування.

6. Визначення часових характеристик пристроя.

Етап дозволяє визначити часові характеристики, зокрема:

- мінімальні і максимальні затримки між вхідними і вихідними сигналами;

- максимальну продуктивність пристроїв;

- час встановлення і затримки сигналів, які гарантують надійну роботу пристроя.

7. Організація випробування та натурних експериментів над діючими пристроями (перевірка роботи пристроя у різних функціональних, температурних режимах).

8.3 Мова проектування VHDL

8.3.1 Загальні поняття.

Мову проектування VHDL створено у 1985 р. (перша версія) за замовленням міністерства оборони США як уніфікований засіб опису (стандарт) цифрових систем. Подальший розвиток призвів до застосування VHDL не тільки як засобу опису, а й проектування і моделювання функціонування пристроїв.

Найвідоміші версії мови VHDL наведені у таблиці 8.2.

Таблиця 8.2 – Версії мови VHDL

Версія	Рік створення	Стандарт	Примітка
VHDL-87	1987	IEEE 1076.1	стандарт
VHDL-93	1993	IEEE 1076.2	стандарт
VHDL-2000	2000	IEEE 1076.3	стандарт Захищені дані
VHDL-2002	2002		
VHDL-2006	2004	Draft 3.0	

Версії мови стали стандартом (IEEE 1076.1, IEEE 1076.2 IEEE 1076.3) і використовуються у інших засобах автоматизованого проектування пристроїв. Зокрема, стандарт IEEE 1076.1, відомий як VHDL-AMS припускає застосування аналогових і змішаних компонентів.

VHDL є проблемно-орієнтованою мовою і призначена для проектування різних рівнів ієрархічного подання цифрового пристрою – від типових вентилів до системного подання. Мова дозволяє описувати структуру і/або поведінку пристроїв і може застосовуватись для синтезу і/або моделювання цифрових систем та їх складових частин. Важливо, що

отримані VHDL – моделі відображаються у програмовані логічні пристрої, зокрема PLD, FPGA.

Синтаксичні конструкції мови VHDL містять загальноалгоритмічну і проблемно-орієнтовану складові. Загальноалгоритмічна містить оператори дії (присвоєння :=, умови IF, вибору CASE, циклу LOOP) і типи даних: числові, логічні, символні, агреговані (масиви, записи, файли). Проект пристрою або його складова частина подаються об'єктами проекту – Entity з інтерфейсом Entity Declaration і описом архітектурного тіла Architecture body. Об'єкт Entity має ім'я і інтерфейс входів і виходів, а архітектура включає опис структури або поведінки об'єкта.

Сучасні версії VHDL підтримують дев'ятизначну логіку (U, X, 0, 1, Z, W, H, L, -) замість бітової логіки (0, 1), що значно розширює можливості моделювання, аналізу і синтезу цифрових пристроїв.

Структурний опис передбачає подання опису типів компонентів і їх інтерфейсу (для виводів); зв'язків компонентів між собою, тобто відображає схемний варіант розв'язання задачі побудови пристрою. Переважно цей опис застосовують для проектування критичних за часом фрагментів проекту пристроя, зокрема структур з паралельним переносом у лічильниках і суматорах.

Поведінковий опис визначає функції для реалізації, які надає компілятор, але не визначає схемний спосіб їх реалізації. Це забезпечує компілятору свободу вибору схемних рішень, які можуть поступатись більш ефективним рішенням кваліфікованих спеціалістів з проектування. Перевагами опису поведінки є компактність, наочність подання функціонування пристрою і простота проектування пристрою або його фрагментів. Тому, зокрема, поведінковий опис варто застосовувати для схем з послідовним переносом.

У сучасних САПР зазвичай застосовують комбіновані підходи на основі сумісного використання структурних і поведінкових описів фрагментів пристроїв. Для критичних частин пристрою застосовують структурний опис, а для решти частини – поведінковий.

Об'єкт проекту може мати ієрархічну будову. Це передбачає, що подання об'єкту верхнього рівня містить звернення до компонентів нижнього рівня.

8.3.2 Опис проекту мовою VHDL

Опис проекту складається з таких частин:

- посилань на бібліотеку функціональних елементів (Library Declaration), на яких виконується подальше автоматизоване проектування;
- опис об'єктів (Entity Declaration);
- опис архітектури (Architecture Declaration), яка подає структуру і/або поведінку пристрою, що проектується.

Проблемно-орієнтовані засоби включають:

- засоби опису ієрархії проекту для подання структури і/або поведінки окремих об'єктів проекту;
- засоби подання і опису паралелізму для виконання дій і операторів;
- визначення сигналу (Signal) для фізичних об'єктів з відповідними часовими вимірами своїх значень і засобами роботи з ними.

Архітектурне тіло включає опис структури, поведінки об'єкта або їх комбінацію. Зв'язок між об'єктами виконується з використанням спеціальних синтаксичних конструкцій. Основними з таких конструкцій є:

- інтерфейс структурної компоненти (component . . . port);
- зв'язок компонентів між собою (port map, generate map);
- створення фрагмента структури (for . . . generate, if . . . generate);

- конкретизація структури (for . . . use).

Способи опису поведінки включають стиль опису (програмування).

Основними стилями є:

- послідовний стиль, за яким перетворення потоку вхідних даних у потік вихідних даних виконується на основі лише послідовних операторів;
- паралельний стиль, який визначає поведінку на рівні паралельно виконуваних процесів;
- потоковий стиль, за яким опис задано у вигляді послідовності паралельних операторів мови VHDL;
- автоматний спосіб опису автоматами Мілі і Мура.

Опис сигналів має певні особливості. Зокрема, використовується поняття призначення значення сигналу (\leq), яке, на відміну від оператора присвоєння ($=$) значення змінній, передбачає затримку змін стану сигналу до моменту підготовки результату перетворень у всіх ініційованих процесах. Це дозволяє врахувати затримки проходження сигналів у реальних схемних реалізаціях елементів.

8.3.3 Приклади реалізації цифрових пристроїв мовою VHDL

Розглянемо приклади реалізації етапів проектування пристроїв з використанням мови VHDL.

Приклад. Описати мовою VHDL поведінку елемента для реалізації функції $z=(a + b)*c$.

Надаємо назву f3 проекту і описуємо входи і виходи.

```
ENTITY f3 IS -- entity declaration
PORT (a, b, c: IN BIT; -- port statement
      Z: OUT BIT);
END f3;
```

Архітектура пристрою для реалізації функції має такий вигляд.

```
ARCHITECTURE one OF f3 IS      - - architecture "one" of
entity f3
BEGIN
  orand3: PROCESS
  BEGIN
    IF (c='1') THEN z<= a OR b
      ELSE z <= '0';
    END IF;
    WAIT ON a, b, c;
    END PROCESS;
END;
```

Приклад. Описати архітектуру лічильника із тактуючим входом clock і скидання reset.

```
sync_count: PROCESS
BEGIN
  WAIT UNTIL clock='1';
  IF (reset = '1' ) THEN count <= "00000000";
    ELSE count <= count + '1';
  END IF;
END PROCESS
```

8.3.4 Опис типових вузлів

До типових вузлів відносяться комбінаційні схеми, регістрові схеми і цифрові автомати. Комбінаційні схеми описують за допомогою:

- арифметичних і логічних виразів;
- умовних (IF) або селективних (CASE) операторів призначення значень сигналу.

Входи схеми подають сигналами або змінними, які мають певний тип, зокрема

- bit- бітовий;
- bit_vector – бітовий вектор;

- std_logic - стандартний логічний;
- std_logic_vector – векторний стандартний логічний.

У регістрових схемах застосовують процесне або блокове подання. У процесному поданні в операторі PROCESS зазвичай використовують оператори IF, CASE, а у блочному – оператори умовного призначення значення сигналу (<=... WHEN).

Приклад . Описати D-тригер як блочне подання.

```
ENTITY d_ff IS - - entity declaration
PORT (d, c , r: IN BIT; - - port statement
q: INOUT BIT);
END d_ff;
ARCHITECTURE one OF d_ff IS      - - architecture "one" of
entity f_ff
BEGIN
  beh_tr: BLOCK (c = '1' OR r = '1');
  BEGIN
    q <= GUARDED '0' WHEN r = '1'
      ELSE d WHEN c = '1';
    ELSE q;
  END BLOCK beh_tr;
END one;
```

Для регістрових схем враховують: наявність синхронізації вхідних сигналів; тип синхронізації (асинхронна, потенційне або динамічне керування); способи визначення вхідних, вихідних сигналів і внутрішніх станів.

У автоматному поданні пристрою переважно застосовують процесну форму опису поведінки пристрою. Архітектура може включати від одного до чотирьох процесів. Окремий процес вводять для опису процедур тактування і початкового встановлення автомата. Для опису альтернативних варіантів формування переходів можна застосовувати оператор IF.

8.3.5 Проектування мовою VHDL з використанням бібліотек

У процесі проектування часто застосовуються стандартні бібліотеки.

Приклад. Синтезувати схему AND на основі бібліотечної.

```
-- (this is a VHDL comment)
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;
-- this is the entity
entity ANDGATE is
  port (
    IN1 : in std_logic;
    IN2 : in std_logic;
    OUT1: out std_logic);
end ANDGATE;
architecture RTL of ANDGATE is
begin
  OUT1 <= IN1 and IN2;
end RTL;
```

Приклад. Синтезувати мультиплексор з двома інформаційними входами A, B, виходом X і селектором S.

```
-- template 1:
X <= A when S = '1' else B;
-- template 2:
with S select X <= A when '1' else B;
-- template 3:
process(A,B,S)
begin
  case S is
    when '1'    => X <= A;
    when others => X <= B;
  end case;
end process;
-- template 4:
process(A,B,S)
begin
  if S = '1' then
    X <= A;
  else
    X <= B;
  end if;
```

```
end process;
```

У проектуванні значної кількості схем застосовують послідовні секції, які містять оператор встановлення сигналу process.

Приклад. Створити елемент пам'яті, стан якого міняється при зростанні сигналу.

```
-- latch template 1:
Q <= D when Enable = '1' else Q;
-- latch template 2:
process(D,Enable)
begin
  if Enable = '1' then
    Q <= D;
  end if;
end process;
```

Приклад . Створити асинхронний тригер SR

```
-- SR-latch template 1:
Q <= '1' when S = '1' else
  '0' when R = '1' else Q;
-- SR-latch template 2:
process(S,R)
begin
  if S = '1' then
    Q <= '1';
  elsif R = '1' then
    Q <= '0';
  end if;
end process;
Template 2 has an implicit "else Q <= Q;" which may be
explicitly added if desired.
-- This one is a RS-latch (i.e. reset dominates)
process(S,R)
begin
  if R = '1' then
    Q <= '0';
  elsif S = '1' then
    Q <= '1';
  end if;
end process;
```

9 АРХІТЕКТУРА СИСТЕМИ КОМАНД

9.1 Основні поняття

Архітектура системи команд ISA (instruction set architecture) є складовою частиною архітектури комп'ютера, яка включає інформацію про:

- набір машинних команд (перелік та семантику операцій, які здатна виконувати обчислювальна машина);
- доступні регістри (внутрішні комірки пам'яті центрального процесора CPU (central processing unit), їх функціональне призначення, розрядність, кількість тощо);
- розрядність та формати операндів;
- способи адресації пам'яті (методи доступу до операндів, які зберігаються в пам'яті);
- особливості обробки виняткових ситуацій;
- обробку переривань.

9.2 Класифікація архітектур системи команд

Залежно від організації взаємодії між обчислювальним пристроєм, регістрами й пам'яттю, виділяють наступні класи архітектур: стекові архітектури; акумуляторні архітектури; архітектури «регістр-пам'ять» і «регістр-регістр»; архітектури «пам'ять-пам'ять».

9.2.1 Стекові архітектури

У стекових архітектурах (рис. 9.1) набір регістрів (або відповідна область пам'яті, де відбувається безпосередня обробка даних) організований у стек (stack) або магазин. Для керування регістровим стеком вводяться дві спеціальні команди: PUSH і POP («заштовхнути» і «виштовхнути»). З їхньою допомогою організується взаємодія з основною пам'яттю.

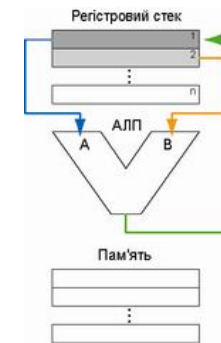


Рисунок 9.1 - Стекова архітектура

Команда PUSH розміщує операнд на вершину стека. Команда POP повертає значення с вершини та розміщує на вершину наступний елемент. В якості поточних операндів в стековій машині завжди мають дві верхніх позиції стека. Тому в обчислювальних командах навіть немає необхідності вказувати адреси операндів. Тому систему команд стекових машин вважають нульадресною.

Результат операції записується знову в вершину стека на місце першого операнда, а на місце другого піднімається наступний і так до спустошення стека.

Стекові архітектури прості в реалізації. Багато перших ЕОМ були побудовані саме в такий спосіб. Стековими були й перші калькулятори.

Однак продуктивність таких архітектур невисока, а підвищувати її можна лише за рахунок зменшення часу циклу. Однак є галузі, де продуктивність не є головним показником. Тому стекові архітектури є досить перспективними для застосування в промислових системах керування та широкому класі вбудованих систем, де важливі простота й дешевина реалізації разом з прийнятною швидкістю обробки. Стекові машини пристосовані для забезпечення режиму реального часу реакції на переривання, що має першорядну важливість у системах керування.

9.2.2 Акумуляторні архітектури

В акумуляторних архітектурах (рис. 9.2) «уявним» залишається тільки один операнд, що перебуває в спеціальному регістрі- акумуляторі. Другий операнд береться безпосередньо з пам'яті або з тимчасового сховища, в яке завантажуються попередньо.

Розвитком цього принципу можна вважати архітектури з наборами спеціалізованих регістрів, в яких деякі регістри були закріплені за певними операціями, або існували якісь інші обмеження на їхнє використання.

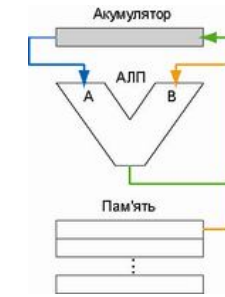


Рисунок 9.2 - Акумуляторна архітектура

Для завантаження операндів необхідно явно вказувати адресу відповідної комірки пам'яті. Таким був набір команд обчислювальної машини Baby (1948), розробленої в Манчестерському університеті, яка вважається першою ЕОМ, в якій достатньо повно був реалізований принцип збереження програми в пам'яті. Цей набір складався з наступних команд:

```
[000]JMP S : CI = S
[010]LDN S : A = -S
[011]SUB S : A = A - S
[011]CMP : If A < 0, CI = CI + 1
[100]JMP S : CI = CI+S
[110]STO S : S = A
[111]HLT : Закінчити програму
```

Тут А — регістр акумулятора, S — комірка пам'яті, представлена своєю адресою.

Звичайна програма складання двох чисел в акумуляторі в цій архітектурі системи команд виглядає так:

```
LDN X1 ; A = -x1
SUB X2 ; A = A-y = -x-y = -(x+y)
STO Y ; Y = -(x+y)
LDN Y ; A = -(-(x+y)) = x+y
```

Подібною була також система команд першої вітчизняної ЕОМ МЕСМ (1948 р., Київ), створена групою інженерів під керівництвом академіка С. О. Лебедева.

9.2.3 Архітектури з індексними регістрами

Індексні регістри вперше застосовані у комерційних ЕОМ Ferranti Mark-1 (1952). Можна було налаштувати цю машину так, що перед виконанням команди, інформація одного з таких регістрів сумувалась безпосередньо з самою командою, або деякими її складовими, тобто був реалізований принцип програми, яка самомодифікується в процесі виконання. Таким чином, для організації прямої адресації переходів, достатньо було завантажити в один з індексних регістрів потрібну адресу, залишивши відповідне поле команди JMP нульовим.

Акумуляторні архітектури були досить популярні в перших ЕОМ через свою простоту. Спеціалізовані регістри були популярними в архітектурах 1970-х років, зокрема, застосовувалися в ІВМ S/360 і в мікропроцесорах серії x86, а сьогодні одержали свій розвиток в архітектурах цифрових сигнальних процесорів (digital signal processor, DSP).

9.2.4 Архітектури «регістр-пам'ять» і «регістр-регістр»

В командах архітектур типу «регістр-пам'ять» (рис. 9.3 а) допускається в команді вказувати в якості місцезнаходження одного з операндів комірки пам'яті.

В архітектурах «регістр-регістр» (рис. 9.3 в) завантаження операндів з пам'яті й збереження результатів виконуються спеціальними командами, і безпосередньо в обчисленнях допускається використання тільки операндів, які знаходяться в регістрах процесора. Останній підхід покладений в основу архітектур зі скороченим набором команд (RISC).

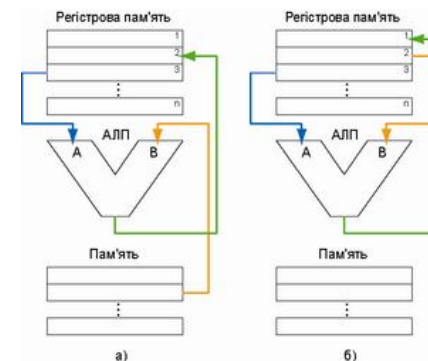


Рисунок 9.3 - Архітектура «регістр-пам'ять» (а) та «регістр-регістр» (б)

Архітектури «регістр-пам'ять» і «регістр-регістр» відносяться до розряду регістрових архітектур. В них як основне сховище операндів використовується набір регістрів загального призначення або спеціалізованих регістрів.

Використання регістрів веде до істотного підвищення продуктивності: по-перше регістрова пам'ять швидше оперативної, по-друге набір регістрів дає більше можливостей програмісту й компілятору для організації складних обчислень (на відміну, наприклад, від стекових архітектур, де порядок обчислення визначається однозначно й не може бути змінений). Це полегшує реалізацію мікроархітектурних методів

підвищення продуктивності обчислень, таких як конвеєризація або сполучення операцій.

Планування використання регістрів в процесі обчислень дозволяє скоротити відсоток звертань до повільної (у порівнянні з регістрами) оперативної пам'яті за рахунок повторного використання результатів обчислень і зберігання в регістрах часто використовуваних змінних. У зв'язку з останньою обставиною зростає роль архітектур з регістрами загального призначення, де всі регістри мають однакову функціональність. Це дає компілятору максимум можливостей щодо оптимального їх розподілу в процесі обчислень. Оптимізувати використання регістрової пам'яті на архітектурах зі спеціалізованими регістрами значно важче через наявні обмеження на використання тих або інших регістрів.

Кількість регістрів в регістрових архітектурах залежить від конкретних завдань, адже в одному випадку необхідно оперувати великими масивами даних і постійно довантажувати операнди в регістри, в іншому всі обчислення локалізовані й при наявності достатньої кількості вільних регістрів, можна вмістити там всі використовувані змінні.

Архітектури з регістрами загального призначення (РЗП) вперше з'явилися в машині Pegasus (1956) фірми Ferranti. Кожний з таких РЗП за функціональністю був ідентичний акумулятору перших машин, і це істотно полегшувало процес програмування й прискорювало швидкодію, адже тепер не потрібно було зберігати проміжні результати в повільній пам'яті й увесь час підзавантажувати звідти потрібні змінні. Їх можна було розміщувати в регістрах.

Це вдосконалення позначилось й на системі команд. В перших машинах вони були однооперандними (тобто в команді вказувався лише один операнд, а другий завжди знаходився в регістрі-акумуляторі). Надалі одноадресні команди були змінені на двоадресні та триадресні.

9.2.5 Архітектури «пам'ять-пам'ять»

Архітектури «пам'ять-пам'ять» дозволяють працювати безпосередньо з оперативною пам'яттю. Характерними є серія комп'ютерів VAX фірми Digital Equipment Corporation.

9.3 Адресність команд

Адресність команд визначається кількістю операндів, які може обробляти команда. Виділяють одноадресні, двоадресні, триадресні команди (рис. 9.4).

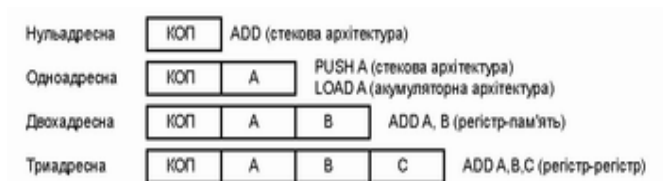


Рисунок 9.4 - Адресність команд

Стекова архітектура може зватися нульадресною або нульоперандною. В ній і вхідні операнди, і адреса для запису результату визначені за замовчуванням, тому в команді нічого вказувати не потрібно.

Акумуляторні архітектури є однооперандними.

Архітектури типу «регістр-пам'ять» часто мають двооперандний формат, оскільки в команді вказується місцезнаходження обох джерел операції, а результат за замовчуванням записується на місце першого, яке

при необхідності повторного використання доведеться заново завантажувати.

Архітектури «регістр-регістр» мають найгнучкіший триоперандний формат з явною вказаними як операндами джерел, так і регістра, в якому буде збережено результат. В модулях векторної обробки (наприклад, обробки мультимедійних даних) сучасних процесорів загального призначення та процесорах цифрової обробки сигналів можна знайти й спеціалізовані чотириоперандні команди.

9.4 Особливості побудови систем команд

9.4.1 Класифікація команд

Всю номенклатуру операцій (команд, інструкцій), реалізованих в сучасних комп'ютерах, можна розбити на наступні функціональні групи:

арифметичні й логічні команди для цілочисельних арифметичних і логічних обчислень (додавання, віднімання, множення, зсуви й логічні операції, порівнювання тощо);

команди пересилання даних для завантаження/збереження в RISC-процесорах, команди переміщення даних (типу mov) в архітектурах, що допускають пряму роботу з пам'яттю;

управління для умовної або безумовної зміни ходу виконання програми, обробки виняткових ситуацій;

системної або привілейовані команди управління віртуальною пам'яттю, переключенням контексту, привілейовані команди операційної системи;

арифметичні з плаваючою комою для арифметичних операції із числами у форматі плаваючої коми;

команди десяткової арифметики;

робота з рядками (strings) для пошуку в рядку, пересилання рядка, порівняння рядків;

графічні, які зазвичай являють собою векторні операції, що прискорюють обробку графічних сцен.

9.4.2 Системи команд CISC

В середині 1960-х років з'явився клас універсальних машин, які нараховували більше десятка способів адресації, не меншу кількість регістрів різної функціональності й більше сотні операцій на всі випадки життя, включаючи найекзотичніші. Тому згодом такі архітектури системи команд одержали назву повних CISC (complete instruction set computer). В деяких комп'ютерах система команд нагадувала мови високого рівня.

Все це повинно було полегшити й прискорити написання програм і зменшити складність компіляторів, скоротити час налагодження, а також обсяги використовуваної пам'яті, яка тоді коштувала дорого, а чим більше складних дій вдавалося реалізувати в компактних машинних командах (тобто чим вищою була їх семантика), тим менше в пам'яті займала програма.

Однак «повні» системи команд виявлялися вкрай надлишковими. CISC характеризується порівняно великою кількістю команд. Команди мають різний час виконання та займають різну кількість пам'яті, що призводить до абсурду команду переходу на x команд вперед (кількість кроків в пам'яті для переходів компілятор розраховує після основної компіляції програми). Продуктивність програми сильно залежить від умінь та знань програміста.

9.4.3 Системи команд RISC

До середини 1970-х років зменшилась кількість програм на машинних мовах, оскільки якість оптимізуючих компіляторів стала цілком прийнятною, а писати на мовах високого рівня легше й швидше ніж в машинних командах. Стало актуальним питання сумісності та міжплатформеної сумісності програмного забезпечення. До того ж, складні набори команд, реалізовані за допомогою повільних мікропрограм, обмежували можливості для подальшого росту швидкодії за рахунок мікроархітектурних оптимізацій.

Архітектури із скороченим набором команд RISC (reduced instruction set computer) створювалися із прицілом на максимальну реалізацію можливостей компіляторів і заздалегідь проектувалися з урахуванням можливої оптимізації процесу обчислень на рівні мікроархітектури.

Наприклад, формат типової триадресної команди на рис. 9.5 мають команди RISC-процесорів, які безпосередньо оперують з регістрами.



Рисунок 9.5 - Триадресна команда

RISC характеризується порівняно невеликою кількістю команд (40 — 1024 для різних архітектур). Але головними ознаками є фіксований розмір команди, за винятком команд переходу, які можуть міститися в 2—3 командних словах (містять адресу).

Також ключовою характеристикою є фіксований час виконання команд. Кожна команда виконується, наприклад за один машинний такт (AVR до 20МГц) чи за 4 машинних такти (PIC до 80МГц). Продуктивність програм суттєво залежить від якості компілятора (при використанні асемблера та машинних команд більше залежить від програміста).

9.4.4 Системи команд VLIW

Сучасною проблемою є складність сучасних процесорів. Мікроархітектурні методи динамічного планування, які перевпорядковують команди та виявляють прихований паралелізм, дають істотний приріст в продуктивності, але вимагають створення винятково складних апаратних структур, які надто складно проектувати й втілювати.

Тому зараз на перший план виходять системи з довгим командним словом VLIW (very long instruction word). В командах VLIW більша частина механізму сполучення операцій виконується на програмному рівні, а процесору дається вже готовий «план дій». Апаратна структура виходить простішою, що робить такі системи досить перспективними, незважаючи на безліч складностей, пов'язаних з їхнім використанням.

VLIW характеризується записом команд в пам'ять фіксованими «пачками», наприклад по чотири команди, які виконуватимуться одночасно на різних арифметико-логічних пристроях. Характерним є надлишок необхідної програмної пам'яті для низки команд, оскільки компілятору не вдається розпаралелити всі команди.

VLIW архітектури з'явилися в 1990 -х роках. Її особливістю є використання дуже довгих команд (до 128 біт і більше), окремі поля яких містять коди, що забезпечують виконання різних операцій. Таким чином,

одна команда викликає виконання відразу декількох операцій паралельно в різних операційних пристроях, що входять в структуру мікропроцесора.

9.4.5 Системи команд SSE

9.4.5.1 Технологія SSE

Технологія SSE передбачає використання набору команд SSE (Streaming SIMD Extensions - потокове SIMD-розширення процесора) компанії Intel для CPU, починаючи з процесора Pentium III.

Технологія дозволяє сумістити виконання мультимедійних команд MMX з командами копроцесора та виконувати команди роботи з дійсними числами.

SSE включає в архітектуру процесора вісім 128-бітових регістрів (xmm0 до xmm7), кожен з яких трактується, як послідовність 4 значень із рухомою крапкою одиначної точності. SSE містить набір інструкцій, які виконують операції зі скалярними і упакованими типами даних.

Наприклад, асемблерна вставка у програмі на мові ANSI C++ `__asm` для роботи з SSE для перемноження пакетів рухомих крапок має вигляд

```
float a[4] = { 300.0, 4.0, 4.0, 12.0 };
float b[4] = { 1.5, 2.5, 3.5, 4.5 };

__asm {
movups xmm0, a ; // помістити 4 змінні з рухомою
крапкою із a в регістр xmm0
movups xmm1, b ; // помістити 4 змінні з рухомою
крапкою із b в регістр xmm1

mulps xmm1, xmm0 ; // перемножити пакети рухомих
крапок: xmm1=xmm1*xmm0
; // xmm10 = xmm10*xmm00
; // xmm11 = xmm11*xmm01
```

```
; // xmm12 = xmm12*xmm02
; // xmm13 = xmm13*xmm03
```

```
movups a, xmm1 ; // вивантажити результати із
регістра xmm1 по адресам a
};
```

9.4.5.2 Набір команд SSE2

В командах SSE2 (розширення 2 для команд SSE), запроваджених у CPU Pentium 4, використовують вісім 128-бітних регістра (xmm0 до xmm7), що увійшли до архітектури x86 з вводом розширення SSE. Кожний регістр подає два значення з плаваючою точкою подвійної точності. Команди SSE2 включають набір інструкцій для виконання операцій зі скалярними і упакованими типами даних. Також SSE2 містить інструкції для потокової обробки цілочислових даних в тих же 128-бітних xmm регістрах, що робить це розширення більш прийнятним для цілочислових обрахунків, ніж використання набору інструкцій MMX, що з'явилися набагато раніше. Перевага у швидкості обчислень досягається в тому випадку виконання однієї послідовності дій над різними даними.

9.4.5.3 Набір команд SSE3

Потокове SIMD-розширення процесора SSE3 (Streaming SIMD Extensions 3), відоме як PNI (Prescott New Instruction) є набір інструкцій, розроблених Intel у 2004 році для ядра Prescott процесора Pentium 4. Подібну реалізацію у 2005 році представила компанія AMD для процесорів Athlon 64.

Набір SSE3 містить 13 інструкцій: FISTTP (x87), MOVSLDUP (SSE), MOVSHDUP (SSE), MOVDDUP (SSE2), LDDQU (SSE/SSE2), ADDSUBPD

(SSE), ADDSUBPD (SSE2), HADDPS (SSE), HSUBPS (SSE), HADDPD (SSE2), HSUBPD (SSE2), MONITOR (аналога у реалізації SSE3 від AMD немає), MWAIT (також відсутній у реалізації SSE3 від AMD).

Основними з них є:

ADDSUBPD (Add Subtract Packed Double).

ADDSUBPS (Add Subtract Packed Single).

HADDPD (Horizontal Add Packed Double).

HADDPS (Horizontal Add Packed Single).

HSUBPD (Horizontal Subtract Packed Double).

HSUBPS (Horizontal Subtract Packed Single).

FISTTP — перетворення дійсного числа в ціле з округленням в меншу сторону.

LDDQU — завантаження 128-біт не вирівняних даних із пам'яті в регістр xmm, з попередженням перетину границі рядку кеша.

До типових CPU компанії AMD з командами SSE3 належать: Athlon 64; Athlon 64 X2; Athlon 64 FX; Opteron ; Sempron; Phenom; Phenom II; Athlon II; Turion 64; Turion 64 X2; Turion X2; Turion X2 Ultra; Turion II X2 Mobile; Turion II X2 Ultra; APU FX Series

До типових CPU компанії Intel належать: Celeron D; Celeron (starting with Core microarchitecture); Pentium 4 (since Prescott); Pentium D; Pentium Extreme Edition ; Pentium Dual-Core; Pentium (starting with Core microarchitecture); Core; Xeon (since Nocona); Atom.

9.4.5.4 Набір команд SSE4

Набір команд SSE4 мікроархітектури Intel Core реалізовано у CPU серії Penryn у 2006 році. SSE4 состоит из 54 инструкций, 47 из них относят к SSE4.1 (они есть в процессорах Penryn). Полный набор команд

(SSE4.1 и SSE4.2, то есть 47 + оставшиеся 7 команд) доступен в процессорах Intel с микроархитектурой Nehalem, которые были выпущены в середине ноября 2008 года и более поздних редакциях. Ни одна из SSE4 инструкций не работает с 64-х битными mmx регистрами (только со 128-ми битными xmm0-15).

Компилятор языка Си от Intel начиная с версии 10 генерирует инструкции SSE4 при задании опции -QxS. Компилятор Sun Studio от Sun Microsystems с версии 12 update 1 генерирует инструкции SSE4 с помощью опций -xarch=sse4_1 (SSE4.1) и -xarch=sse4_2 (SSE4.2)[2]. Компилятор GCC поддерживает SSE4.1 и SSE4.2 с версии 4.3[3], опции -msse4.1 и -msse4.2, или -msse4, включающая оба варианта.

9.4.5.5 Набір команд SSE5

Потокове SIMD-розширення процесора SSE5 (Streaming SIMD Extensions 5) запропоновано AMD у 2007 році як 128-bit SSE – команди ядра для архітектур AMD64. Але у 2009 році AMD замінила SSE5 трьома множинами розширення команд XOP, FMA4, CVT16, які є сумісними з набором команд AVX компанії Intel. Ці групи команд реалізовані у ядрі процесора Bulldozer у 2011 році за технологією 32 нм.

9.4.6 Системи команд AVX

Системи команд AVX (Advanced Vector Extensions) – це розширення системи команд x86 для мікропроцесорів Intel і AMD, запропоноване Intel в березні 2008. AVX надає різні поліпшення, нові інструкції і нову схему кодування машинних кодів.

За новою схемою кодування інструкцій VEX ширина векторних регістрів SIMD збільшується з 128 (XMM) до 256 біт (регістри YMM0 - YMM15). Існуючі 128-бітові SSE інструкції використовують молодшу половину нових YMM регістрів, не змінюючи старшу частину. Для роботи з YMM регістрами додані нові 256-бітові AVX інструкції.

У майбутньому можливе розширення векторних регістрів SIMD до 512 або 1024 біт. Наприклад, процесори з архітектурою Lagabee вже мають векторні регістри (ZMM) шириною в 512 біт, і використовують для роботи з ними SIMD команди з MVEX і VEX префіксами, але при цьому вони не підтримують AVX.

Неруйнуючі операції. Набір AVX інструкцій використовує трьохоперандний синтаксис. Наприклад, замість $a = a + b$ можна використовувати $c = a + b$, при цьому регістр a залишається незмінним. У випадках, коли значення a використовується далі в обчисленнях, це підвищує продуктивність, оскільки позбавляє від необхідності зберігати перед обчисленням і відновлювати після обчислення регістр, що містив a , з іншого регістру або пам'яті.

Для більшості нових інструкцій відсутні вимоги до вирівнювання операндів в пам'яті. Однак, рекомендується стежити за вирівнюванням розмір операнда, щоб уникнути значного зниження продуктивності.

Набір інструкцій AVX містить в собі аналоги 128-бітних SSE інструкцій для дійсних чисел. При цьому, на відміну від оригіналів, збереження 128-бітного результату буде обнуляти старшу половину YMM регістру. 128-бітові AVX інструкції зберігають інші переваги AVX, такі як нова схема кодування, трьохоперандний синтаксис і невіривнений доступ до пам'яті. Рекомендується відмовитися від старих SSE інструкцій на користь нових 128-бітних AVX інструкцій, навіть якщо достатньо двох операндів.

Нова схема кодування інструкцій VEX використовує VEX префікс. На даний момент існують два VEX префікса, довжиною 2 і 3 байти. Для 2-х байтного VEX префікса перший байт дорівнює $0xC5$, для 3-х байтного $0xC4$.

У 64-бітному режимі перший байт VEX префікса унікальний. У 32-бітному режимі виникає конфлікт з інструкціями LES і LDS, який дозволяється старшим бітом другого байта, він має значення тільки в 64-бітному режимі, через невідтримувані форми інструкцій LES і LDS.

Довжина існуючих AVX інструкцій, разом з VEX префіксом, не перевищує 11 байт. У наступних версіях очікується поява більш довгих інструкцій. Нові інструкції AVX наведені у таблиці 9.1.

Також в специфікації AVX описана група інструкцій PCLMUL (Parallel Carry-Less Multiplication, Parallel CLMUL)

PCLMULLQLQDQ xmmreg,xmmrm [rm: 66 0f 3a 44 /r 00]

PCLMULHQLQDQ xmmreg,xmmrm [rm: 66 0f 3a 44 /r 01]

PCLMULLQHQQDQ xmmreg,xmmrm [rm: 66 0f 3a 44 /r 02]

PCLMULHQHQQDQ xmmreg,xmmrm [rm: 66 0f 3a 44 /r 03]

PCLMULQDQ xmmreg,xmmrm,imm [rmi: 66 0f 3a 44 /r ib]

Інструкції AVX підходять для інтенсивних обчислень з плаваючою комою в мультимедіа програмах та наукових завданнях. Там, де можлива більш висока ступінь паралелізму, збільшує продуктивність з дійсними числами.

Підтримка в операційних системах.

Використання YMM регістрів вимагає підтримки з боку операційної системи. Наступні системи підтримують регістри YMM:

Linux: з версії ядра 2.6.30, [5] released on June 9, 2009.

Windows 7: підтримка додана в Service Pack 1

Windows Server 2008 R2: підтримка додана в Service Pack 1

Таблиця 9.1 - - Нові інструкції AVX

Інструкція	Опис
VBROADCASTSS, VBROADCASTSD, VBROADCASTF128	Копіює 32-х, 64-х або 128-ми бітний операнд з пам'яті в усі елементи векторного реєстра XMM або YMM.
VINSERTF128	Заміщає молодшу або старшу половину 256-ти бітного реєстра YMM значенням 128-ми бітного операнда. Інша частина реєстра-одержувача не змінюється.
VEXTRACTF128	Витягує молодшу або старшу половину 256-ти бітного реєстра YMM і копіює в 128-ми бітний операнд-призначення.
VMASKMOVPS, VMASKMOVPD	Умовно зчитує будь-яку кількість елементів з векторного операнда з пам'яті в реєстр-одержувач, залишаючи інші елементи неліченими і обнуляючи відповідні їм елементи реєстра-одержувача. Також може умовно записувати будь-яку кількість елементів з векторного реєстра в векторний операнд в пам'яті, залишаючи інші елементи операнда пам'яті незміненими
VPERMILPS, VPERMILPD	Переставляє 32-х або 64-х бітові елементи вектора згідно операнду-селектору (з пам'яті або з реєстра).
VPERM2F128	Переставляє 4 128-ми бітних елемента двох 256-ти бітних реєстрів в 256-ти бітний операнд-призначення з використанням безпосередньої константи (imm) в якості селектора.
VZEROALL	Обнуляє всі YMM реєстри і позначає їх як невикористовувані. Використовується при перемиканні між 128-ми бітним режимом і 256-ти бітовим.
VZERoupper	Обнуляє старші половини всіх реєстрів YMM. Використовується при перемиканні між 128-ми бітним режимом і 256-ти бітовим.

9.5 Програмні моделі систем команд CPU

Мікропроцесор, або центральний процесор CPU (Central Processing Unit) виконує обчислення і обробку даних (за винятком деяких математичних операцій, здійснюваних в комп'ютерах, що мають співпроцесор).

Мікропроцесор - це центральний блок комп'ютера, призначений для управління роботою всіх інших блоків і виконання арифметичних і логічних операцій над інформацією.

Мікропроцесор виконує такі основні функції:

читання і дешифрування команд з основної пам'яті;

читання даних з основної пам'яті і реєстрів адаптерів зовнішніх пристроїв;

прийом та обробку запитів і команд від адаптерів на обслуговування зовнішніх пристроїв;

обробку даних і їх запис в основну пам'ять і реєстри адаптерів зовнішніх пристроїв;

вироблення керуючих сигналів для всіх інших вузлів і блоків комп'ютеру.

У багатьох сучасних комп'ютерах використовують CPU, сумісні з сімейством мікросхем Intel, компанії Intel, а також компаніями AMD, Cyrix, IDT, Rise Technologies.

9.5.1 Класифікація мікропроцесорів за призначенням

Мікропроцесори поділяють на окремі класи відповідно до їх архітектури, структури і функціонального призначення (рис. 9.6).

Мікропроцесори загального призначення призначені для вирішення широкого кола завдань обробки різноманітної інформації. Їх основною областю використання є персональні комп'ютери, робочі станції, сервери інші цифрові системи масового застосування.

Спеціалізовані мікропроцесори орієнтовані на вирішення специфічних завдань управління різними об'єктами. Вони містять додаткові мікросхеми (інтерфейсні), які забезпечують спеціалізоване використання. Мають особливу конструкцію, підвищену надійність.

Мікроконтролери є спеціалізованими мікропроцесорами, які орієнтовані на реалізацію пристроїв керування, вбудованих у різноманітну апаратуру. Характерною особливістю структури мікроконтролерів є розміщення на одному кристалі з центральним процесором внутрішньої пам'яті і великого набору периферійних пристроїв.

Цифрові процесори сигналів (ЦПС) являть собою клас спеціалізованих мікропроцесорів, орієнтованих на цифрову обробку вхідних аналогових сигналів. Специфічною особливістю алгоритмів обробки аналогових сигналів є необхідність послідовного виконання ряду команд множення-підсумовування з накопиченням проміжного результату в регістрі-акумуляторі. Тому архітектура ЦПС орієнтована на реалізацію швидкого виконання операцій такого роду. Набір команд цих процесорів містить спеціальні команди MAC (Multiplication with Accumulation), які реалізують ці операції.



Рисунок 9.6 – Класифікація мікропроцесорів за функціональним призначенням

9.5.2 Мікроархітектура мікропроцесора

Мікроархітектура мікропроцесора - це апаратна організація і логічна структура мікропроцесора, реєстри, керуючі схеми, арифметико-логічні пристрої, запам'ятовуючі пристрої і пристрої, які зв'язують їхні інформаційні магістралі. Макроархітектура мікропроцесора є: система команд, типи оброблюваних даних, режими адресації і принципи роботи мікропроцесора.

При описі архітектури та функціонування процесора зазвичай використовується його подання у вигляді сукупності програмно-доступних реєстрів, що утворюють реєстрову або програмну модель. У цих реєстрах містяться оброблювані дані (операнди) і керуюча інформація. Відповідно, в реєстрову модель входить група реєстрів загального призначення, службові реєстри для зберігання операндів, і група службових реєстрів, що забезпечують управління виконанням

програми і режимом роботи процесора, організацію звернення до пам'яті (захист пам'яті, сегментна і сторінкова організація тощо).

Регістри загального призначення утворюють внутрішню реєстрову пам'ять процесора. Склад і кількість службових реєстрів визначається архітектурою мікропроцесора. Зазвичай в їх склад входять:

Програмний лічильник PC (або CS + IP в архітектурі мікропроцесорів Intel);

Регістр стану SR (або EFLAGS);

Регістри управління режимом роботи процесора CR (Control Register);

Регістри, що реалізують сегментну і сторінкову організацію пам'яті;

Регістри, що забезпечують налагодження програм і тестування процесора.

Склад пристроїв і блоків, що входять в структуру мікропроцесора, і реалізуються механізми їх взаємодії визначаються функціональним призначенням і областю застосування мікропроцесора.

Архітектура та структура мікропроцесора тісно взаємопов'язані. Реалізація тих чи інших архітектурних особливостей вимагає введення в структуру мікропроцесора необхідних апаратних засобів (пристроїв і блоків) і забезпечення відповідних механізмів їх спільного функціонування.

У сучасних мікропроцесорах реалізуються наступні варіанти архітектур:

мікропроцесори типу CISC з повним набором системи команд;

мікропроцесори типу RISC з усіченим набором системи команд;

мікропроцесори типу VLIW з надвеликим командним словом;

мікропроцесори типу MISC з мінімальним набором системи команд і

вельми високою швидкодією тощо.

3 CISC архітектура CPU реалізована для багатьох типів мікропроцесорів. Вони виконують великий набір різноформатних команд з використанням численних способів адресації. Вони виконують більше 200 команд різного ступеня складності, які мають розмір від 1 до 15 байт і забезпечують більше 10 різних способів адресації. Таке велике різноманіття виконуваних команд і способів адресації дозволяє програмісту реалізувати найбільш ефективні алгоритми вирішення різних завдань.

При цьому суттєво ускладнюється структура мікропроцесора, особливо його пристрій управління, що приводить до збільшення розмірів і вартості кристалу, зменшенню продуктивності. У той же час багато команд і способів адресації використовуються досить рідко.

Аналіз кодів програм, які генеруються компіляторами мов вищого рівня, показує, що компілятори з усієї системи команд CPU використовують тільки обмежений набір простих команд. Це команди типу "регістр-регістр", "регістр-пам'ять".

В RISC архітектурах відрізняється використовують обмежений набір команд фіксованого формату. Сучасні RISC - процесори зазвичай реалізують близько 100 команд, що мають фіксований формат довжиною 4 байта.

При цьому значно скорочується число використовуваних способів адресації. Зазвичай в RISC - процесорах всі команди обробки даних виконуються тільки з реєстровою або безпосередньою адресацією. Для зменшення кількості звертань до пам'яті RISC-процесори мають збільшений об'єм внутрішніх реєстрів - від 32 до декількох сотень, тоді як в CISC-процесорах кількість реєстрів загального призначення переважно становить 8-16.

В CPU з архітектурою Фон – Неймана наявність загальної пам'яті дозволяє оперативно перерозподіляти її обсяг для зберігання окремих масивів команд, даних і реалізації стека в залежності від розв'язуваних завдань. Використання спільної шини для передачі команд і даних значно спрощує відладку, тестування і поточний контроль функціонування системи, збільшує її надійність.

Основний недолік - необхідність послідовної вибірки команд і даних по спільній системній шині. При цьому шина стає вузьким місцем, яке обмежує продуктивність системи. Постійно зростаючі вимоги до продуктивності мікропроцесорних систем викликали в останні роки більш широке застосування Гарвардської архітектури при створенні багатьох типів сучасних мікропроцесорів.

Гарвардська архітектура характеризується фізичним поділом пам'яті команд (програм) і пам'яті даних. У її оригінальному варіанті використовувався також окремий стек для зберігання вмісту програмного лічильника, який забезпечував можливості виконання вкладених підпрограм.

Кожна пам'ять з'єднується з процесором окремою шиною, що дозволяє одночасно з читанням/записом даних при виконанні поточної команди робити вибірку і декодування наступної команди. Завдяки такому поділу потоків команд і даних і поєднанню операцій їх вибірки реалізується більш висока продуктивність, ніж з архітектурою Фон – Неймана

Недоліки Гарвардської архітектури пов'язані з необхідністю більшого числа шин, а також з фіксованим об'ємом пам'яті, виділеної для команд і даних, призначення якої не може оперативно перерозподілятися. Тому потрібно використовувати пам'ять більшого об'єму, коефіцієнт використання якої при вирішенні різноманітних задач виявляється

нижчим, ніж в системах з Принстонською архітектурою. Проте розвиток мікроелектроніки дозволив в значній мірі подолати вказані недоліки.

Тому Гарвардська архітектура широко застосовується у внутрішній структурі сучасних мікропроцесорів, де використовується окрема кеш-пам'ять для зберігання команд і даних. Разом з цим, у зовнішній структурі більшості мікропроцесорних систем реалізуються принципи з архітектурою Фон – Неймана. Гарвардська архітектура отримала також широке застосування в мікроконтролерах та цифрових сигнальних процесорах.

До мікропроцесору і системної шині поряд з типовими зовнішніми пристроями можуть бути підключені і додаткові плати з інтегральними мікросхемами, що розширюють і поліпшують функціональні можливості мікропроцесора. До них відносяться математичний співпроцесор, контролер прямого доступу до пам'яті, співпроцесор вводу/виводу, контролер переривань тощо.

Математичний співпроцесор використовують для прискорення виконання операцій над двійковими числами з плаваючою комою, над кодованими десятковими числами, для обчислення тригонометричних функцій. Математичний співпроцесор має свою систему команд і працює паралельно з основним мікропроцесором, але під управлінням останнього. В результаті відбувається прискорення виконання операцій в десятки разів.

Моделі мікропроцесора, починаючи з МП 80486 DX, включають математичний співпроцесор в свою структуру.

Співпроцесор вводу/виводу за рахунок паралельної роботи з мікропроцесором значно прискорює виконання процедур вводу/виводу при обслуговуванні декількох зовнішніх пристроїв, звільняє мікропроцесор від обробки процедур введення/виведення, в тому числі реалізує режим прямого доступу до пам'яті.

9.5.3 Параметри процесорів

Робота процесора полягає і в послідовному виконанні команд з оперативної пам'яті, і в швидкості виконання команд. Чим швидше процесор виконує команди, тим вища продуктивність комп'ютера в цілому. Швидкість роботи процесора залежить від декількох параметрів.

Швидкодія процесора вимірюється в мегагерцах (МГц). Тактова частота визначає максимальний час виконання перемикачів між елементами ЕОМ.

Реальна частота роботи ядра процесора може становити 1,5 – 4 ГГц. Тактова частота визначається множенням частоти зовнішньої шини процесора на коефіцієнт множення. Зовнішня шина використовується для обміну даними з іншими пристроями і може мати позначення FSB (Front Side Bus). Наприклад, для процесора Intel Core 2DUO E6600 частота FSB - 266,6 МГц, множник - 9, в результаті тактова частота буде рівна 2400 МГц.

Розрядність процесора визначає максимальну кількість двійкових розрядів, які можуть бути оброблені одночасно. У процесор входить три важливих пристрої, основною характеристикою яких є розрядність:

- шина вводу і виводу даних;
- внутрішні регістри;
- шина адреси пам'яті.

Коли говорять про шину процесора, найчастіше мають на увазі шину даних, яка являє набір з'єднань (або виводів) для передачі або прийому даних. Чим більше сигналів одночасно надходить на шину, тим більше даних передається по ній за певний інтервал часу і тим швидше вона працює.

Шина адреси передає адреси комірок пам'яті, в яку або з якої пересилаються дані. Збільшення кількості розрядів, які використовуються для формування адреси, дозволяє збільшити кількість адресованих комірок. Розрядність шини адреси визначає максимальний обсяг пам'яті, що адресується процесором.

Розрядність внутрішніх регістрів визначає кількість розрядів оброблюваних процесором даних, а також характеристики програмного забезпечення і команд.

10 МОВИ АСЕМБЛЕРА

10.1 Загальні поняття

Мова асемблера — мова програмування низького рівня, яка використовується для програмування комп'ютерів, мікропроцесорів, мікроконтролерів інших мікросхем. Мова програмування низького рівня. Ближча до специфіки роботи самого процесора, для якого вона написана. Вважають, що мови низького рівня складніші й потребують більш вузької спеціалізації програміста, оскільки програма написана на асемблері для одного типу процесорів виявиться не завжди придатною для роботи з іншими процесорами. З іншого боку якісні програми, написані на асемблері компактні та швидкі.

Асемблер як система програмування включає мову асемблера та транслятор з цієї мови.

Транслятор з автокоду Асемблер переводить початкову програму, написану на автокодї, в переміщену програму на мові машинній. Оскільки асемблер здійснює трансляцію на мову завантажувача, при завантаженні програми необхідна налаштування умовних адрес, тобто адрес, значення яких залежать від розташування даної програми в пам'яті ЕОМ і від її зв'язків з іншими незалежно трансльованими програмами.

Найбільш поширеними є транслятори MASM, GAS, AS, fasm, NASM, RosASM, TASM, Yasm, HNASM.

У простому випадку асемблер переводить одне речення початкової програми в один об'єкт (команду, константу) модуля завантаження. При цьому взаємне розташування об'єктів в модулі завантаження і, зрештою, в пам'яті машини визначається порядком речень в початковій програмі на автокодї і повністю залежить від програміста.

Асемблер виконує і допоміжні функції, зокрема, підготовку до друку документів необхідної форми, реєстрація зв'язків даної програми з іншими програмами тощо. Для цього в автокодах передбачають команди асемблера, які не породжують об'єктів в робочій програмі і призначені тільки для визначення допоміжних дій транслятора асемблера.

Трансляція зазвичай вимагає двох переглядів початкової програми: при першому перегляді здійснюється розподіл пам'яті і надання значень символічним іменам; при другому — формується робоча програма у вигляді модуля завантаження. В процесі трансляції асемблер проводить повний синтаксичний контроль початкової програми, забезпечуючи достатньо точну діагностику помилок за місцем і характером.

Розширення можливостей автокодів досягається за рахунок використання макрокоманд, що будуються за правилами, близькими до правил написання команд автокоду. Макрокоманди описують складніші функції, для реалізації яких потрібна група звичайних команд. В цьому випадку перед трансляцією проводиться заміна макрокоманд макророзширеннями — послідовностями команд на базовій мові відповідно до макроозначень. У останніх задається прототип макрокоманди із структурою списку параметрів і процедура генерування макророзширення.

Транслятор, що виконує функції макрогенератора і асемблера, називається макроасемблером. При трансляції з мов високого рівня асемблер часто використовують для виконання завершальної фази трансляції.

10.2 Переваги і недоліки мов асемблера

Команди мови асемблера відповідають машинним кодам відповідного мікропроцесора чи мікроконтролера. Фактично, мова асемблера являє собою зручнішу символічну форму запису машинних команд. Як наслідок, програми написані для одного типу процесорів, на іншому не будуть функціонувати.

Мова асемблера також містить засоби для створення міток та переходів, що необхідно для створення циклів та розгалужень. Можуть бути наявні засоби для створення макросів, процедур.

Кожне сімейство (модельний ряд) мікропроцесорів має свій набір команд і, відповідно, свій набір інструкцій на мові асемблера.

Перевагами і недоліками є:

мінімальна кількість надлишкового коду (використання меншої кількості команд та звернень в пам'ять). Як наслідок — велика швидкість і менший розмір програми;

великі обсяги коду, велике число додаткових дрібних завдань;

погана читабельність коду, труднощі підтримки (налагодження, додавання можливостей);

труднощі реалізації парадигм програмування та будь-яких інших скільки-небудь складних конвенцій, складність спільної розробки;

менша кількість доступних бібліотек, їх несумісність;

безпосередній доступ до апаратури (портам введення-виведення, особливим регістрам процесора тощо);

максимальна «підгонка» для потрібної платформи (використання спеціальних інструкцій, технічних особливостей «заліза»);

не мобільність переносу коду на інші платформи (крім двійково сумісних).

10.3 Директиви асемблера

Крім інструкцій, програма може містити директиви: команди, що не переводяться безпосередньо в машинні інструкції, а керують роботою компілятора. Набір і синтаксис їх значно різняться і залежать не від апаратної платформи, а від використовуваного компілятора (породжуючи діалекти мов в межах одного сімейства архітектур).

В якості набору директив можна виділити:

визначення даних (констант і змінних);

управління організацією програми в пам'яті і параметрами вихідного файлу;

визначення режиму роботи компілятора;

всілякі абстракції (тобто елементи мов високого рівня) — від оформлення процедур і функцій (для спрощення реалізації парадигми процедурного програмування) до умовних конструкцій і циклів (для парадигми структурного програмування);

макроси;

10.4 Синтаксис асемблера

Розрізняють Intel-синтаксис і AT&T-синтаксис для програм на асемблер. Intel-синтаксис є одним з найпоширеніших, використовується для IBM-сумісних комп'ютерів. AT&T-синтаксис відрізняється від Intel-синтаксису наявністю суфіксів до мнемонік та префіксів до операндів.

Наприклад, для Intel-синтаксису

Вивести слова на екран Hello, World! монітора для версії Intel x86

(IA32)

```
mov ax,cs
mov ds,ax mov ah,9
mov dx, offset Hello
int 21h xor ax,ax
int 21h

Hello:

db "Hello World!",13,10,"$"
```

Приклад для NASM Linux, використовується Intel синтаксис:

- nasm -f elf -o hello.o hello.asm
- ld -o hello hello.o

```
SECTION .data
msg db "Hello, world!",0xa
len equ $ - msg
SECTION .text
global _start
_start:                ; Точка входа в
программу              mov eax, 4      ;
'write' системный вызов mov ebx, 1
mov ecx, msg           ; Указатель на данные
mov edx, len           ; Количество данных
int 0x80               ;Вызов ядра
mov eax, 1             ; '_exit' системный вызов
mov ebx, 0             ; Возвращаем 0 (все хорошо)
int 0x80               ; Вызов ядра
```

Приклад для версії Apple II:

```
*****
* HELLO WORLD FOR 6502 APPLE ][ *
*****
STROUT EQU $DB3A
```

```
LDY #>HELLO
LDA #<HELLO
JMP STROUT
```

```
HELLO ASC "HELLO WORLD!",00
```

Приклад для NASM Linux я FreeBSD/x86

```
SECTION .data
msg: db "Hello, world",10
len: equ $-msg

SECTION .text
global _start

syscall: int 0x80
ret

_start: push len
push msg
push 1 ; stdout
mov eax, 4 ; write(2)
call syscall
add esp, 3*4

push 0
mov eax, 1 ; exit(2)
call syscall
```

Програми на асемблер легко вбудовують в програмні коди мов вищого рівня. Наприклад

```
main ()
{
int a = 1; // оголошуємо змінну a і кладемо туди значення 1
int b = 2; // оголошуємо змінну b і кладемо туди значення 2
int c; // оголошуємо змінну c, але не ініціалізуємо її
// Початок асемблерної вставки
```

```

__asm{
mov eax, a // завантажуюмо значення змінної a в регістр EAX
mov ebx, b // завантажуюмо значення змінної b в регістр EBX
add eax, ebx // додаємо EAX з EBX, записуючи результат в EAX
mov c, eax // завантажуюмо значення EAX у змінну c
}
// Кінець асемблерної вставки
// Виводимо вміст c на екран
// За допомогою звичної функції printf
printf ("a + b =% x +% x =% x \ n", a, b, c);
}

```

10.5 Типові набори команд асемблера

Типовими командами мови асемблера є (для Intel-синтаксиса архітектури x86):

Команди пересилання даних (mov тощо.)

Арифметичні команди (add, sub, imul)

Логічні і побітові операції (or, and, xor, shr)

Команды управління виконанням програм (jmp, loop, ret и др.)

Команди виклику переривань: int

Команди введення і виведення (in, out)

Для мікроконтролерів характерними є команди переходу за умовами і перевірки також:

cjnc — перейти, якщо не рівно;

djnz — декремент, якщо не нульовий результат і перейти;

cfsneq — порівнянн; у випадку не порівняння, пропустити наступну

команду.

Програмування мовою асемблер

Приклад Роглянемо текст програми на асемблер.

```
#this is in a file first.s
```

```
.globl main
```

```
main:
```

```
movl $20,%eax
```

```
movl $10,%ebx
```

```
ret
```

Директива .globl робить доступним линкеру програми C. За його відсутності то за командою

```
% gcc first.s
```

```
/usr/lib/crt1.o: In function `'_start':
```

```
: undefined reference to `main'
```

```
collect2: ld returned 1 exit status
```

Команда `movl $20, %eax` завантажує у регістри двійкове значення 20 `%eax`. За допомогою опції компілятора C `-S`, генерують код асемблера для вихідного тексту програми.

Наприклад, для програми `simple.c`

```
int main(){
int x=10,y=15;
return 0;
}
```

За допомогою команди

```
_ gcc -S simple.c
```

отримаємо код на асемблері `file simple.s`

```
.globl main
```

```
.type main, @function
```

```

main:
pushq %rbp
movq %rsp, %rbp
movl $10, -8(%rbp)
movl $15, -4(%rbp)
movl $0, %eax
leave
ret

```

Тут
movl \$10, -4(%ebp)
вказує на завантаження числа 10 за адресою ebp-4.
Іншим прикладом є перетворення з програми на C у код асемблера

```

#include <stdio.h>
int main(){
printf("hello world\n");
return 0;
}
// Assembly code
.LC0:
.string "hello world"
.text
.globl main
.type main, @function
main:
pushq %rbp
movq %rsp, %rbp
movl $.LC0, %edi
call printf
movl $0, %eax
leave
ret

```

11 ЗАСТОСУВАННЯ СХЕМНО-ПРОГРАМНИХ РІШЕНЬ АРХІТЕКТУР КОМП'ЮТЕРІВ

11.1 Застосування схемних рішень для криптографічних перетворень

Шифрування даних здійснюють з використанням трьох видів засобів шифрування/дешифрування: апаратних, програмно-апаратних і програмних. Основна відмінність полягає не тільки в способі реалізації шифрування і ступеня надійності захисту даних, а й в ціні, що часто стає для користувачів визначальним фактором. Найдешевшими є програмні пристрої шифрування, дорожчими є програмно-апаратні засоби, найдорожчими - апаратні. Незважаючи на те, що ціна апаратних засобів шифрування істотно вище програмних, їх якість із захисту інформації значно вища.

11.1.1 Переваги апаратного шифрування

Велика кількість засобів шифрування даних створюється у вигляді спеціалізованих фізичних пристроїв. Програмні шифратори, зазвичай, дешевше апаратних і в ряді випадків здатні забезпечити більшу швидкість обробки інформації.

Основними перевагами апаратних шифраторів є: апаратний генератор випадкових чисел створює дійсно випадкові числа для формування надійних ключів шифрування та електронного цифрового підпису; апаратна реалізація криптоалгоритму гарантує його цілісність; шифрування і зберігання ключів здійснюються в самій платі шифратора, а не в оперативній пам'яті комп'ютера; завантаження ключів в шифрувальний пристрій здійснюється з електронних ключів Touch Memory (i-Button) і смарт-карт безпосередньо, а не через системну шину комп'ютера і

ОЗП, що виключає можливість перехоплення ключів; за допомогою апаратних шифраторів можна реалізувати системи розмежування доступу до комп'ютера та захисту інформації від несанкціонованого доступу; застосування спеціалізованого процесора для виконання всіх обчислень розвантажує центральний процесор комп'ютера; можна встановлювати кілька апаратних шифраторів на одному комп'ютері, що ще більше підвищує швидкість обробки інформації (ця перевага притаманна шифратору для шин PCI); застосування парафазних шин при створенні шифропроцесора виключає загрозу читання ключової інформації по коливаннях електромагнітного випромінювання, що виникають при шифруванні даних, в ланцюгах «земля – джерело живлення» пристрою; шифрування даних проходить швидше ніж в програмно-апаратних засобах шифрування.

При установці на комп'ютер спеціалізованого шифрувального обладнання буде виникати менше проблем, ніж при додаванні в системне програмне забезпечення функцій шифрування даних. У самому кращому випадку шифрування повинно проводитися так, щоб користувач не помічав його. Щоб зробити це за допомогою програмних засобів, вони повинні бути приховані досить глибоко в операційній системі. Виконати цю операцію безболісно з налагодженою операційною системою дуже непросто. Але під'єднати шифрувальне пристрій до персонального комп'ютера або до модему зможе і не професіонал.

11.1.2 Види пристроїв апаратного шифрування

Сучасний ринок пропонує три різновиди апаратних засобів шифрування інформації потенційним покупцям: блоки шифрування в каналах зв'язку; самодостатні шифрувальні модулі (вони самостійно

виконують всю роботу з ключами); шифрувальні плати розширення для установки в персональні комп'ютери

Майже всі пристрої перших двох типів є вузько спеціалізовані. Тому потрібно досконало досліджувати обмеження, які при установці ці пристрої накладають на відповідні пристрої, прикладне програмне забезпечення та операційні системи до прийняття остаточного рішення про їх закупівлю. Інакше можна дарма витратити гроші, не наблизившись до бажаної мети. Однак іноді компанії, які продають комунікаційне обладнання, пропонуть встановлені пристрої апаратного шифрування, що іноді полегшує вибір.

Плати розширення для персональних комп'ютерів є більш універсальним засобом апаратного шифрування. Зазвичай, їх дуже легко налаштувати так, щоб вони шифрували всю інформацію, яка записується на жорсткий диск або пересилається в порти і дисководи. Часто захист від електромагнітного випромінювання в платах розширення для апаратного шифрування відсутня, оскільки безглуздо захищати ці плати, якщо весь комп'ютер не захищається аналогічним чином.

11.1.3 Додаткові можливості апаратних шифраторів

Використання цілої плати розширення тільки для апаратного шифрування занадто дорого. Тому, крім функцій шифрування, виробники намагаються додати в свої пристрої різноманітні додаткові можливості. Наприклад, додають генератор випадкових чисел. Він необхідний в основному для генерації криптографічних ключів. Значна кількість алгоритмів шифрування застосовують їх і для інших цілей. Зокрема, алгоритм електронного підпису ГОСТ Р 34.10 – 2001 при обчисленні підпису використовують щоразу нове випадкове число.

Часто засоби шифрування застосовують при довіреному завантаженні для контролу входу на комп'ютер. Кожен раз, коли користувач включає персональний комп'ютер, пристрій буде вимагати від нього введення персональної інформації (наприклад, вставити дискету з ключами). Тільки якщо пристрій розпізнає надані ключі і вважатиме їх «своїми», завантаження буде продовжена. Інакше користувач буде змушений розбирати комп'ютер і виймати звідти плату шифратора, щоб включити комп'ютер (проте, як відомо, інформація на жорсткому диску також може бути зашифрована).

Додатковими можливостями апаратних шифраторів є контроль цілісності файлів операційної системи. Зловмисник не може у відсутність користувача що-небудь змінити в операційній системі. Шифратор зберігає у своїй пам'яті перелік всіх важливих файлів із заздалегідь підрахованими для кожного контрольними сумами (або хеш-значеннями), і комп'ютер буде блокований, якщо при черговому завантаженні не співпаде контрольна сума хоча б одного з файлів.

Пристроєм криптографічного захисту даних називають плату розширення з усіма перерахованими вище можливостями. Пристрій апаратного шифрування, який контролює вхід на персональний комп'ютер і перевіряє цілісність всіх файлів операційної системи, часто називають «електронним замком». Для нього необхідно програмне забезпечення - утиліта, яка генерує ключі для користувачів і зберігає їх список для розпізнавання «свій / чужий». Крім цього, необхідна програма для вибору важливих файлів і підрахунку їх контрольних сум. Доступ до цих додатків зазвичай є тільки у адміністратора з безпеки. Він повинен заздалегідь конфігурувати всі пристрої для користувачів. Якщо з'являться проблеми, то він повинен розібратися в їх причинах.

11.1.4 Апаратно-програмні елементи шифрування

Блокові потокові шифри реалізуються по-різному. Поточкові шифри, що розшифровують дані за одним бітом, не дуже підходять для програмних реалізацій. Блокові шифри легше реалізовувати програмно, тому що вони дозволяють уникнути трудомних маніпуляцій з бітами. оперують зручними для комп'ютера блоками даних. З іншого боку, потокові шифри більше підходять для апаратної реалізації.

Більшість реальних поточкових шифрів заснована на регістрах зсуву зі зворотним зв'язком. Регістр зсуву застосовують для генерації ключової послідовності. Регістр зсуву зі зворотним зв'язком складається із двох частин: регістру зсуву і функції зворотнього зв'язку (рис. 11.1).

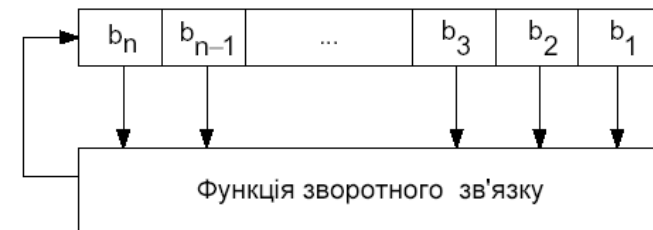


Рисунок 11.1 -. Регістр зсуву зі зворотним зв'язком

Регістр зсуву становить послідовність бітів, кількість яких визначається довжиною зсуву регістру. Якщо довжина рівна n -бітам, то регістр називається n -бітовим регістром зсуву. Щоразу, коли потрібно витягти біт, всі біти регістру зсуву зсуваються вправо на 1 позицію. Новий крайній лівий біт і функцією всіх інших бітів регістру. На виході регістру виявляється один, зазвичай молодший значущий біт. Періодом регістру

називається довжина одержуваної послідовності до початку і. повторення. Найпростішим видом регістру зсуву зі зворотним зв'язком і регістр зсуву з лінійним зворотним зв'язком (РЗЛЗЗ) (рис. 11.2). четвертого б.т.в. b_n b_1 n b

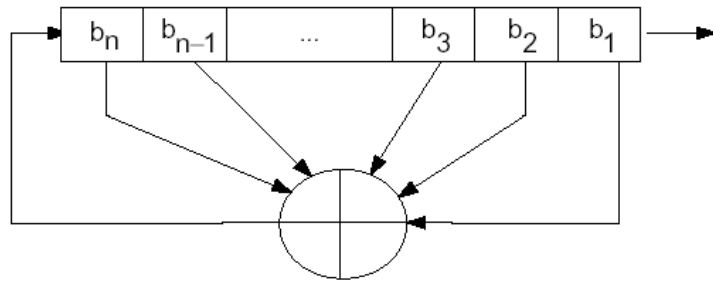


Рисунок 11.2 -..Регістр зсуву з лінійним зворотним зв'язком

Зворотний зв'язок становить XOR деяких бітів регістру; ці біти називаються відповідною послідовністю. На рис. 11.3 показаний 4-бітовий РЗЛЗЗ з відводом від першого четвертого бітів.

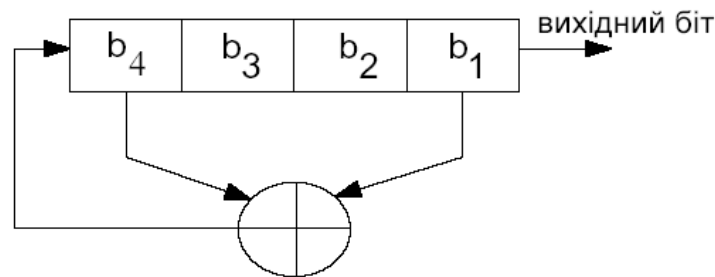


Рисунок 11.3- 4-бітовий РЗЛЗЗ

Випадкові числа, які генеруються з використанням РЗЛЗЗ, формують підряд біти послідовності з сильнокорельованими властивостями і для деяких типів додатків зовсім не є випадковими. Незважаючи на це, РЗЛЗЗ часто використовуються при розробці алгоритмів шифрування.

11.2 Криптосистеми шифрування.

Шифрування тексту M за допомогою перетворення $C = E_{K_1}(M)$ дозволяє отримати зашифроване повідомлення C , де K_1 - ключ шифрування.

Дешифрування (розшифрування) $M' = D_{K_2}(C)$ дозволяє відновити текст M з криптографічного повідомлення C за допомогою ключа K_2 .

Розрізняють симетричні криптосистеми з одним ключем для шифрування и дешифрування ($K_1=K_2$) і асиметричні системи з двома ключами K_1 , K_2 , окремо для шифрування і дешифрування. У асиметричних системах наявність і навіть знання загальнодоступного ключа (public key) не дозволяє визначити секретний ключ. Для використання механізмів криптографічного закриття інформації в локальній обчислювальній мережі необхідна організація спеціальної служби генерації ключів і їх розподіл між її абонентами.

На рис. 11.4 показано симетричну криптосистему з одним ключем.

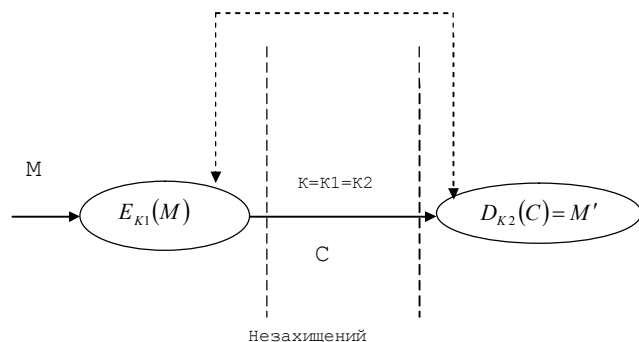


Рисунок 11.4 - Симетрична система з одним ключем

Конфіденційність передавання електронних документів у комп'ютерних системах для симетричних систем забезпечується конфіденційністю ключа шифрування.

11.2.1 Характеристика симетричних криптосистем

Основними прикладами симетричних криптосистем є DES, IDEA, ГОСТ 28147-89. Основою цих алгоритмів часто слугують мережі (сітки) Фейстеля. Мережа Фейстеля – це схема (метод) зворотних перетворень тексту, за яких обчислення однієї з частин тексту накладається на інші місця тексту.

Тривалий час використовувався симетричний алгоритм шифрування на основі стандарту DES (Data Encryption Standard), який є федеральним стандартом США. DES розроблено фірмою IBM та рекомендовано для використання Агентством національної безпеки США. Алгоритм

криптографічного захисту відомий і опублікований. Він характеризується такими властивостями:

- високим рівнем захисту даних проти дешифрування і можливої модифікації даних;

- простотою розуміння;

- високим ступенем складності, яка робить його розкриття дорожчим від отриманого прибутку;

- методом захисту, який базується на ключі і не залежить від секретності механізму алгоритму;

- економічністю в реалізації і швидкодії.

Разом з тим DES має ряд недоліків, зокрема:

- малий розмір ключа (56 біт), що потребує для розкриття близько 7×10^{16} операцій. На даний час апаратури, здатної виконати такі обсяги обчислень немає, але сумісне використання великої кількості комп'ютерів дозволяє їх виконати у прийнятний час;

- окремі блоки, що містять однакові дані будуть виглядати однаково, що є погано з точки зору криптографії.

Тому з 2001р. алгоритм DES замінено на алгоритм шифрування AES, але його окремі модифікації продовжують застосовуватись у менш відповідальних випадках.

Більш захищеним, у порівнянні з DES, є стандарт шифрування даних ГОСТ 28147-89. Він є єдиним алгоритмом криптографічного перетворення даних для великих інформаційних систем і не накладає обмежень на ступінь секретності інформації. Цей стандарт має переваги алгоритму DES і в той же час позбавлений від його недоліків (зокрема, за довжиною ключа). Крім того, в стандарт закладено метод, що дозволяє зафіксувати невиявлену випадкову чи навмисну модифікацію зашифрованої

інформації. Недоліком алгоритму шифрування за стандартом ГОСТ 28147-89 є складність його програмної реалізації.

11.2.2 Симетричні шифри

Основними видами симетричних шифрів є поточкові (stream cipher) і блокові (block cipher) шифри. У поточковому шифрі кожний символ відкритого тексту перетворюють у символ шифрованого тексту, залежно від ключа та його розташування у потоці відкритого тексту. Тому поточні шифри стають нестійкими за умови пропуску окремих символів шифротексту.

За способами синхронізації розрізняють синхронні і само синхронізовані поточкові шифри. Для реалізації цих шифрів часто застосовують лінійні регістри зсуву (LFSR— Linear Feedback Shift Registers)

Вхідними даними блокового шифру є блок розміром n і ключ довжиною k . На основі перетворень на виході отримують n – бітний зашифрований блок (рис. 11.5).



Рисунок 11.5 – Формування зашифрованого блоку

Блочні шифри передбачають багатократне застосування до блоків базових перетворень: складного перетворення для локальної частини блока; простого перетворення між частинами блока. На початку всі вхідні

дані, подані у двійковій формі, розбивають на блоки сталої довжини n . Серед поширених підходів до створення симетричних блокових шифрів є застосування мереж Фейстеля.

Найвідоміші алгоритми симетричного шифрування подано у табл. 11.1.

Таблиця 11.1 – Характеристика алгоритмів симетричного шифрування

Алгоритм	Рік	Кількість раундів	Довжина ключа, біт	Розмір блока, біт	Кількість підблоків
Blowfish	1993	16	до 448	64	2
Camellia	2000	18/24	128/192/256	128	2
Cartman-II	2008	64/36/16	512/384/256	128	4
CAST-128	1996	12/16	40-128	64	2
CAST-256	1998	12×4=48	128/192/256	128	2
CIPHERUNICORN-A	2000	16	128/192/256	128	2
CIPHERUNICORN-E	1998	16	128	64	2
CLEFIA	2007	16	128/192/256	128	16
DEAL	1998	6 (8)	(128/192) 256	128	2
DES	1977	16	56	64	2
DFC	1998	8	128/192/256	128	?
FEAL	1987	4-32	64	64	2
GTEA	2009	32-128	256-4096	128	4
ГОСТ 28147-89	1989	32/16	256	64	2
IDEA	1991	8+1	128	64	4
KASUMI	1999	8	128	64	2
Khufu	1990	16-32/64	512	64	2
LOK197	1997	16	128/192/256	128	2
Lucifer	1971	16	48/64/128	48/32/128	2
MacGuffin	1994	32	128	64	4
MAGENTA	1998	6/8	128/192/256	128	2
MARS	1998	32	128—1248	128	2
Mercy	2000	6	128	4096	?
MISTY1	1995	4×n(8)	128	64	4
Raiden	2006	16	128	64	2
RC2	1987	16+2	8-128	64	4
RC5	1994	1-255(12)	0-2040(128)	32/64/128	2

11.2.3 Перетворення у симетричних алгоритмах шифрування

Основними принципами при створенні шифрів є застосування операції розсіювання і перемішування. Розсіювання дозволяє розподілити вплив одного знаку відкритого тексту на багато знаків шифротексту.

Перемішування полягає у використанні шифрованих перетворень, які ускладнюють відновлення взаємозв'язку статистичних властивостей відкритого і шифрованого текстів. В складених шифрах застосовують послідовність простих шифрів.

При перестановці перемішують символи відкритого тексту, а саме перемішування визначається секретним ключем. В підстановці кожний символ відкритого тексту замінюється символом цього ж алфавіту.

В сучасних блокових шифрах довжина блоків зазвичай складає 64 біта, тобто блок може мати 2^{64} значень.

11.2.4 Мережі Фейстеля

Перетворення у мережах Фейстеля виконують над блоками $L_i R_i$, $i=1, 2, 3, \dots$, які являють собою ліву (L) і праву (R) половини регістру зсуву. У блочних шифрах застосовують пряме (рис.11.6) і зворотне (рис. 11.7) перетворення Фейстеля. Зокрема, у алгоритмі DES $i=1, 2, \dots, 16$.

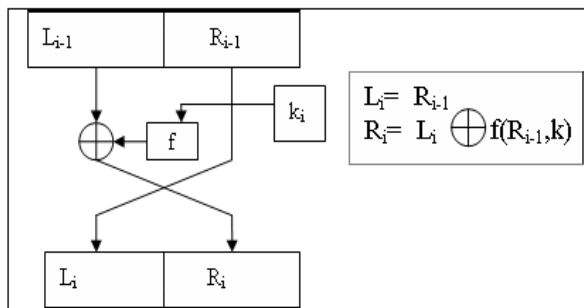


Рисунок . 11.6 - Пряме перетворення мережею Фейстеля

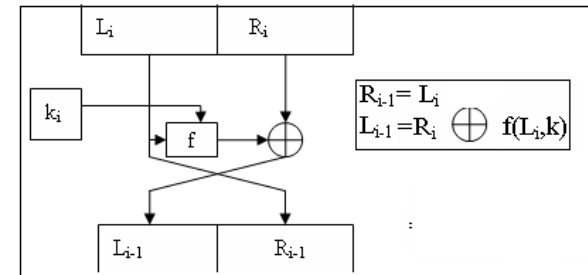


Рисунок . 11.7 - Зворотне перетворення мережею Фейстеля

11.3 Алгоритм DES

11.3.1 Загальна характеристика

Алгоритм DES (Data Encryption Standard)— симетричний алгоритм шифрування, було створено у компанії IBM і затверджено у якості офіційного стандарту у 1977 р. Основою побудови алгоритму був шифр Lucifer.

В алгоритмі застосовують комбінацію нелінійних (S-блоки) та лінійних (перестановки E, IP, IP-1) перетворень. Застосування DES передбачає кілька режимів:

- електронної кодової книги (ECB — Electronic Code Book);
- зчеплення блоків (CBC — Cipher Block Chaining);
- зворотного зв'язку за шифротекстом (CFB — Cipher Feed Back);
- зворотного зв'язку за виходом (OFB — Output Feed Back).

Алгоритм DES має довжину блоків 64 біти і довжину ключа 56, що стало приводом для критики, оскільки не в повній мірі задовольняло вимогам. У 2002 році цей алгоритм замінено на алгоритм AES, але він продовжує застосовуватись у цілому ряді програмних продуктів.

У DES застосовують пряме (для шифрування) і зворотне (для дешифрування) перетворення мережею Фейстеля, як показано на рис. 2.,3.

11.3.2 Схема шифрування DES

Схему шифрування DES подано на рис. 11.8.

На схемі позначення L_i і R_i відповідають лівій і правій половині 64-бітового блока L_iR_i , k_i — 48 бітові ключі, f — функція шифрування, IP — початкова перестановка, IP^{-1} - кінцева перестановка.

Відкритий блок тексту T з 64 біт перетворюється за IP згідно із табл.11.2

Таблиця 11.2 - Початкова перестановка IP

8	0	2	4	6	8	0		0	2	4	6	8	0	2		
2	4	6	8	0	2	4		4	6	8	0	2	4	6		
7	9	1	3	5	7			9	1	3	5	7	9	1		
1	3	5	7	9	1	3		3	5	7	9	1	3	5		

Зокрема, перші три біти результату перетворення блока $IP(T)$ за початковою перестановкою IP являють собою біти 58, 50, 42 вихідного блоку T , а чотири останні є бітами 31, 23, 15, 7 блоку тексту T .

Наступні шістнадцять циклів перестановки 64-бітних блоків виконуються з використанням перетворень Фейстеля за схемою рис. 11.8 у такій послідовності.

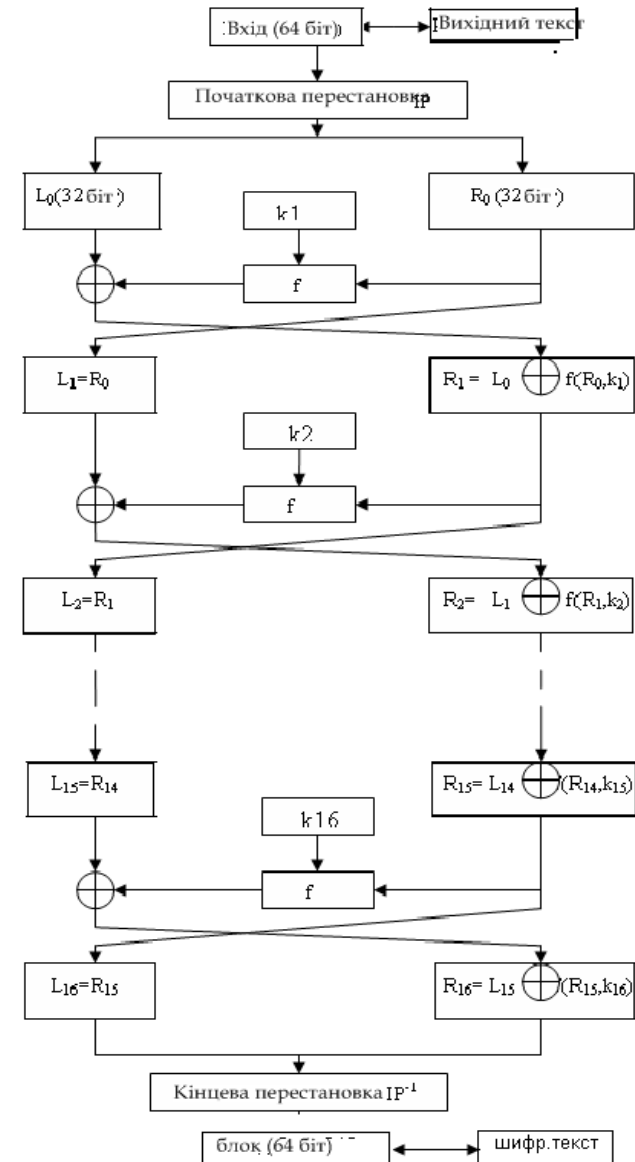


Рисунок. 11.8 - Схема шифрування DES

Блок IP(T) розбивають на дві частини L0,R0, де L0,R0 є відповідно 32 старших і 32 молодших бітів блоку T0 IP(T)= L0R0

Для поточного $T_i - 1 = L_i - 1R_i - 1$ результату ітерації (i-1), результат i-ої ітерації $T_i = L_iR_i$ визначають за співвідношеннями

$$L_i = R_{i-1} - 1$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$

Ліва половина L_i становить праву половину попереднього вектора $L_i - 1R_i - 1$. Значення правої половини R_i визначають побітовим додаванням $L_i - 1$ та значення функції $f(R_i - 1, k_i)$ за модулем 2. У перетворенні Фейстеля функція f (функція Фейстеля) відіграє роль шифрування.

Аргументами f є 32 бітовий вектор $R_i - 1$ і 48 бітовий ключ k_i , який отримують перетворенням 56 - бітового вихідного ключа k . Для обчислення f застосовують функцію розширення E , перетворення S і перестановки P .

11.3.3 Схема утворення функції f

Загальну схему утворення функції f подано на рис. 11.9.

Функція E розширює 32 - бітовий вектор $R_i - 1$ до 48 - бітового вектора $E(R_i-1)$ згідно з табл. 11.9.

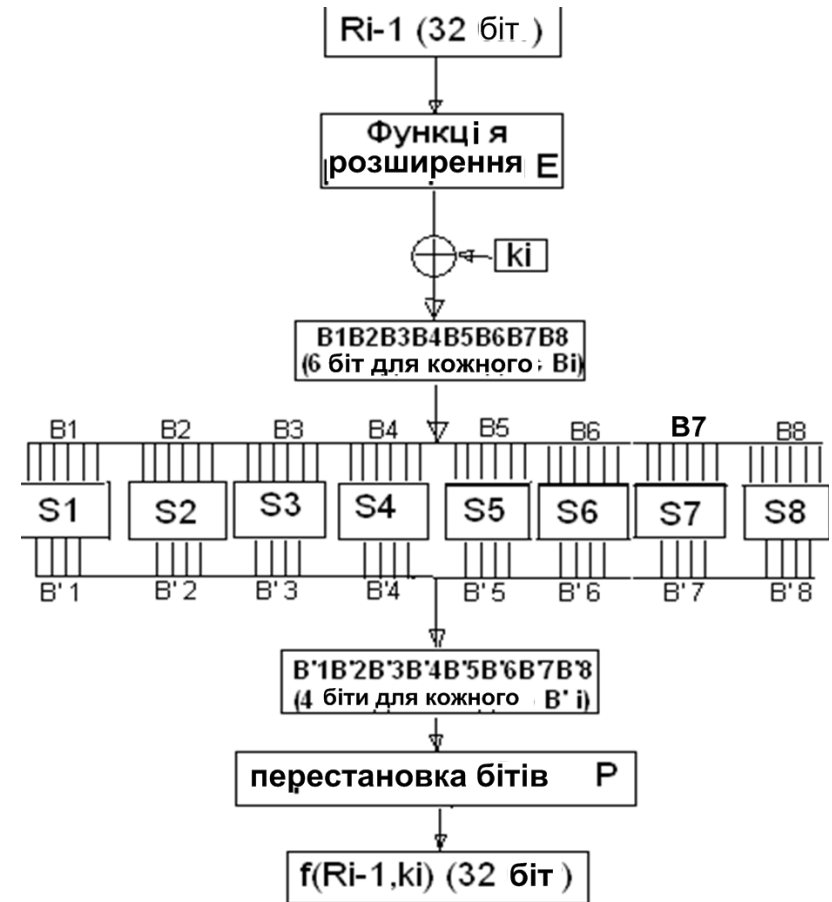


Рисунок 11.9 - Схема утворення функції f

Таблиця.11.3 - Функція розширення E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Перші три біти $E(R_i - 1)$ є бітами відповідно 32, 1, 2 вектора $R_i - 1$, а останні три біти $E(R_i - 1)$ є бітами відповідно 31, 32, 1 вектора $R_i - 1$. Біти 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, 29, 32 дублюються.

Отриманий після перестановки код $E(R_i - 1)$ додають за модулем 2 до ключа k_i (операція XOR) і розбивають на вісім блоків B_1, B_2, \dots, B_8 : $E(R_i - 1) = B_1 B_2 \dots B_8$

Кожний 6-бітовий блок B_i трансформують у 4-бітовий блок B'_j за допомогою перетворень S_j . $j=1, \dots, 16$. Функції перетворення подано у табл.11.4.

Таблиця 11.4 - Перетворення S_j . $j=1, \dots, 16$.

Номер рядка	Номер стовпчика																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S_1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S_2
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S_3
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S_4
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	

3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S ₅
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	S ₆
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S ₇
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	S ₈
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Перетворення виконуються у такий спосіб. На вхід матриці S_j надходить блок $V_j = b_1b_2b_3b_4b_5b_6$, з яких число b_1b_6 вказує номер рядка матриці, а число $b_2b_3b_4b_5$ вказує номер стовпчика.

Наприклад, для $V_j = 100110$ отримуємо рядок $b_1b_6 = 102 = 2$ і стовпчик з номером $b_2b_3b_4b_5 = 00112 = 3$ матриці S_j . Зокрема, для $j=1$ вибирають елемент 8. Сукупність блоків V_1, V_2, \dots, V_8 забезпечують вибір чотирьохбітового елемента з кожної матриці S_1, S_2, \dots, S_8 .

Нехай потрібним є отримання для $V_3 = 101111$ значення V'_3 . Перший і останній розряди визначають рядок $112 = 3$ таблиці S_3 , а $01112 = 7$ сьомий стовпчик, тобто $V'_3 = 7 = 01112$.

Перестановку P функції $f(R_i - 1, k_i)$ отримують для 32-бітового блоку $V'_1V'_2 \dots V'_8$ згідно з табл. 11.5.

Таблиця 11.5 - Перестановка P

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

$$f(R_i - 1, k_i) = P(V'_1V'_2 \dots V'_8)$$

Перші чотири біта результуючого вектора після дії функції f є біти 16, 7, 20, 21 вектора $V'_1V'_2 \dots V'_8$.

11.3.4 Генерування ключів

Генерування ключів k_i виконується за початковим заданим 64-бітовим ключем k . Початковий 64-розрядний код ділять на байти. Вісім бітів з позицій 8, 16, 24, 32, 40, 48, 56, 64 додають до ключа k так, щоб кожний байт мав непарну кількість одиниць. Це використовується для подальшого контролю і виявлення похибок передавання ключів. Далі виконують перестановку бітів, крім бітів 8, 16, 24, 32, 40, 48, 56, 64, згідно з табл. 11.6.

Таблиця 11.6 - Перестановка біт ключа

57	49	41	33	25	17	9	1	58	50	42	34	26	18	C_0
10	2	59	51	43	35	27	19	11	3	60	52	44	36	
63	55	47	39	31	23	15	7	62	54	46	38	30	22	D_0
14	6	61	53	45	37	29	21	13	5	28	20	12	4	

Перестановка визначена блоками C_0 і D_0 по 28 біт. Перші три біти C_0 є біти 57, 49, 41 розширеного ключа, а перші три біта D_0 є біти 63, 55, 47. Значення $C_i D_i$ отримують з $C_{i-1} D_{i-1}$ одним або двома циклічними зсувами згідно з табл. 11.7.

Таблиця 11.7 - Перестановки $C_i D_i$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Кількість зсувів	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Ключ k_i , $i=1, \dots, 16$ складається з 48 біт, вибраних з бітів вектора $C_i D_i$ (56 біт) згідно з табл. 8. Зокрема, першим і другим є біти 14, 17 вектора $C_i D_i$.

Таблиця 11.8 - Формування ключа k_i з $C_i D_i$.

14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

Кінцева перестановка $P-1$ діє на $T16$ згідно з табл. 11.9.

Таблиця 11.9 - Перестановка IP-1

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

11.3.5 Дешифрування тексту в DES

Схему дешифрування показано на рис. 11.10. Розшифрування відбувається у зворотному порядку відносно шифруванню. Для перетворення застосовують зворотні перетворення Фейстеля (рис. 11.6):

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f(L_i, k_i)$$

Ключі k_i , функція f , перестановки IP та IP-1 визначаються як і у випадку шифрування.

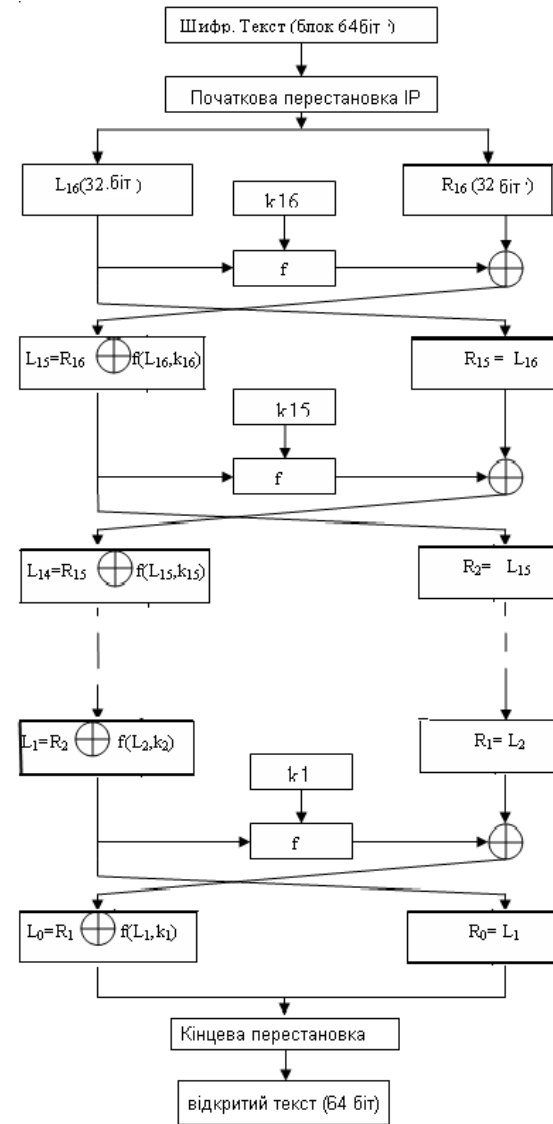


Рисунок 11.10 - Схема дешифрування блоку тексту

11.3.6 Режими використання DES

11.3.6.1 Режим електронної кодової книги ECB

Режим передбачає розбивку тексту на окремі блоки, кожний з яких шифрується незалежно від інших, як показано на рис. 11.11.

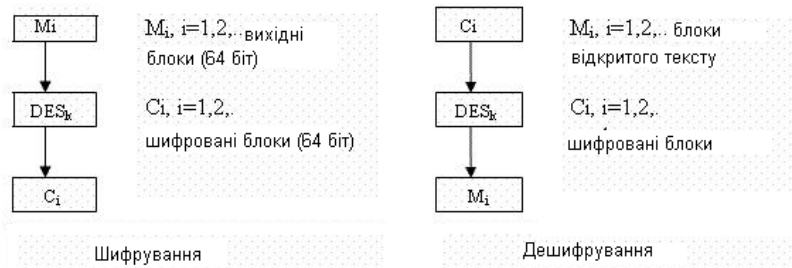


Рисунок 11.11 - Шифрування і дешифрування тексту у режимі ECB

Такий режим може незалежно застосовуватись до окремих частин тексту, або до тексту в цілому.

11.3.6.2 Режим зчеплення блоків CBC

Режим передбачає розбивку тексту на окремі блоки, кожний з яких за модулем 2 додається до попереднього зашифрованого блоку, як показано на рис. 11.12.

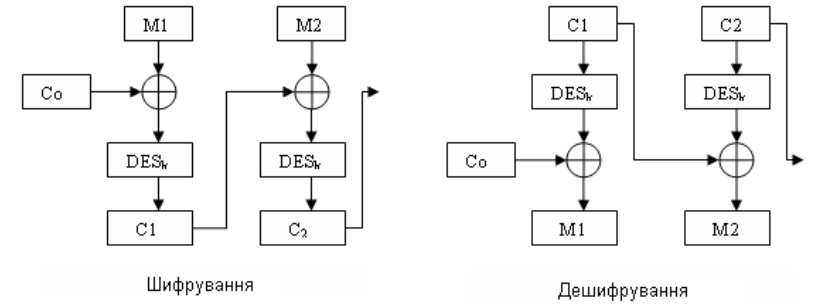


Рисунок 11.12 - Шифрування і дешифрування тексту у режимі CBC

Під час шифрування кожний зашифрований попередній блок $C_i, i > 0$ додається до наступного відкритого блоку M_{i+1} , а початковий вектор C_0 зберігають у секреті.

Дешифрування виконується у зворотному порядку.

11.3.6.3 Режим зворотного зв'язку за шифротекстом CFB

Режим передбачає розбивку тексту на окремі блоки, кожний з яких додають до «гами» зашифрованого попереднього блоку, як показано на рис. 11.13.

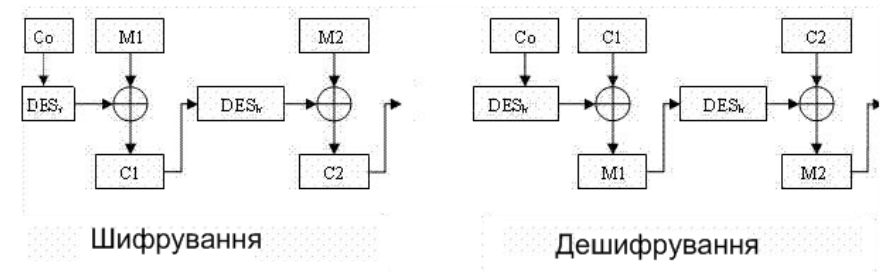


Рисунок 11.13 - Шифрування і дешифрування тексту у режимі CFB

Блочну гаму Z_0, Z_1, \dots , ...отримують за шифруванням $Z_i = \text{DES}_k(C_i - 1)$, $C_i = M_i \oplus Z_i$, а початковий вектор C_0 зберігають у секреті.

11.3.6.4 Режим зворотного зв'язку за виходом OFB

Режим передбачає розбивку тексту на окремі блоки, кожний з яких додають до «гами» багаторазово зашифрованого «секрету», як показано на рис. 11.14.

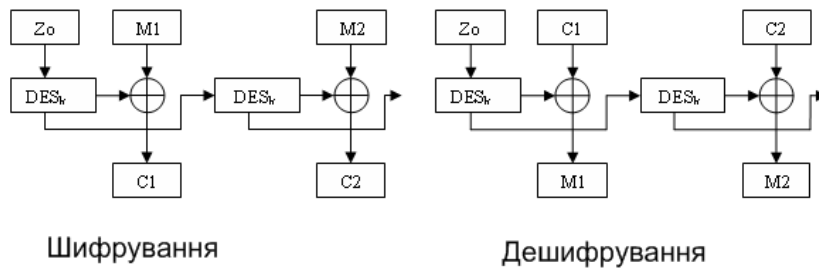


Рисунок 11.14 - Шифрування і дешифрування тексту у режимі OFB

Блочну гаму Z_0, Z_1, \dots утворюють за шифруванням $Z_i = \text{DES}_k(Z_{i-1})$, $C_i = M_i \oplus Z_i$, а початковий вектор C_0 зберігають у секреті.

11.4 Алгоритм Triple DES

Алгоритм Triple DES (3DES або TDES) ґрунтується на алгоритмі DES і опубліковано у 1978р. Алгоритм передбачає застосування блоків тексту з 64 біт і розрядністю ключа 112 біт для модифікації 2TDES або 168 біт для 3TDES. При розробці застосовані мережі Фейстеля з використанням 48 раундів DES-еквівалентних раундів перетворень.

Швидкість роботи 3DES нижча за DES, але криптостійкість набагато вища. Зокрема, час на криптоаналіз 3DES у може бути мільярд більше, ніж час для розкриття DES.

Загальну схему алгоритма 3DES подано на рис. 11.15.

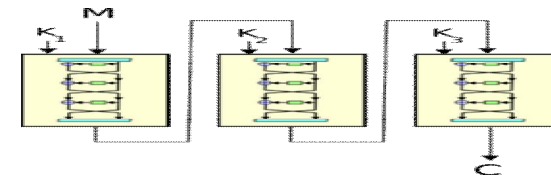


Рисунок 11.15 - Схема алгоритма 3DES

Для найпростішого випадку шифрування, відомому під назвою EEE, виконують перетворення за виразом

$$\text{DES}(k_3; \text{DES}(k_2; \text{DES}(k_1; M)))$$

де k_1, k_2, k_3 — ключі для кожного DES-кроку;

M — вхідні дані, які шифрують.

Існує три типи алгоритмів 3DES:

DES-EEE3, за яким шифрування виконують тричі із застосуванням різних ключів;

DES-EDE3, за яким виконують операції шифрування-дешифрування-шифрування трьома різними ключами;

DES-EEE2 та DES-EDE2, для яких виконують операції, як у першому і другому випадках, але на першому і третьому кроці використовують однакові ключі.

Найпоширенішим є алгоритм DES-EDE3, за яким шифрування виконують за виразом

$$C = E_{k_3}(E_{k_2}^{-1}(E_{k_1}(P)))$$

а дешифрування за виразом

$$P = E_{k_1}^{-1}(E_{k_2}(E_{k_3}^{-1}(C)))$$

У процесі реалізації алгоритму 3DES ключі можуть вибрані у комбінаціях:

k_1, k_2, k_3 незалежні;

k_1, k_2 незалежні, а $k_1 = k_3$

$k_1 = k_2 = k_3$

Ефективна довжина ключа 3DES враховує повторюваність ключів. Зокрема для трьох різних ключів враховують те, що у DES 64-бітовий ключ ділять на вісім байтів, в кожному з яких використовують лише 7 біт. Тобто загальна довжина ключа становить $56 \times 3 = 168$ бітів. Але у результаті атак «зустріч посередині» (meet-in-the-middle) ефективна криптостійкість

становить лише 112 біт. Для DES-EDE, де $k_1 = k_3$, ефективна довжина ключа становить лише 80 біт.

Для успішної атаки на 3DES потрібно біля 232 біт відкритого тексту, 2113 кроків, 290 циклів шифрувань DES та 288

Алгоритми 3DES DES-EEE3, DES-EDE3 застосовують в Internet, зокрема у PGP та S/mime. 3DES використовують також у керуванні ключами у стандартах ANSI X9.17 и ISO 8732 в у PEM (Privacy Enhanced Mail). У порівнянні з алгоритмом AES програмна реалізація 3DES повільніша у 6 разів. Тому часто 3DES реалізують апаратно.

11.5 Алгоритм AES

Стандарт симетричного блочного шифрування AES (Advanced Encryption Standard), оснований на алгоритмі блочного шифрування під назвою Rijndael, прийнято у якості стандарту США (FIPS 197) у 2002 р. замість алгоритму DES, який мав ряд недоліків. Основними недоліками DES є мала довжина ключа (56 біт), орієнтація на апаратну реалізацію, невисока швидкодія програмної реалізації.

Довжина шифрключа K може бути 128, 192, 256 біт, розмір блоку від 128 до 256 біт з кроком 32 біти. Довжина блоку вхідних даних $input$ і стану $State$ постійна і становить 128 біт.

Псевдокод шифрування алгоритму AES показано на рис. 11.16.

Значення $State$ є проміжним результатом шифрування і являє масив байтів з чотирьою рядків і Nb 32-бітових стовпчиків слів. Для AES $Nb = 4$.

Cipher Key є секретний ключ, за яким за допомогою процедури Key Expansion отримують набір ключів для раундів Round Keys. Cipher Key подають як масив з чотирьох рядків і Nk стовпчиків.

Для позначення у байтах використовують $N_b = 4$ для input та State, $N_k = 4, 6, 8$ для різної довжини ключів Cipher Key.

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end

```

Рисунок 11.16 - Псевдокод шифрування алгоритму AES

Спочатку input копіюють у масив State за правилом $s[r,c] = in[r + 4c]$, $0 \leq r < 4$ $0 \leq c < N_b$. До State застосовують процедуру AddRoundKey() і далі процедуру трансформації (раунд) 10, 12, 14 разів залежно від довжини ключа. Значення State копіюють в output $out[r + 4c] = s[r,c]$, $0 \leq r < 4$ $0 \leq c < N_b$.

Псевдокод дешифрування алгоритму AES показано на рис. 11.17.

Додаткові процедури, які використовують у алгоритмі, передбачають такі перетворення. AddRoundKey() виконує операцію XOR значення ключа раунда Round Key з State. Довжина RoundKey співпадає з розміром State. Наприклад, при $N_b = 4$ довжина RoundKey становить 128 біт, тобто 16 байт. Round Keys отримують з Cipher Key з використанням процедури Key Expansion.

```

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)
        InvSubBytes(state)
        InvAddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)
    end for

```

```

InvShiftRows(state)
InvSubBytes(state)
InvAddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

```

```

out = state
end

```

Рисунок 11.17 - Псевдокод дешифрування алгоритму AES

На рис. 11.18 подано схему перетворень у AddRoundKey().

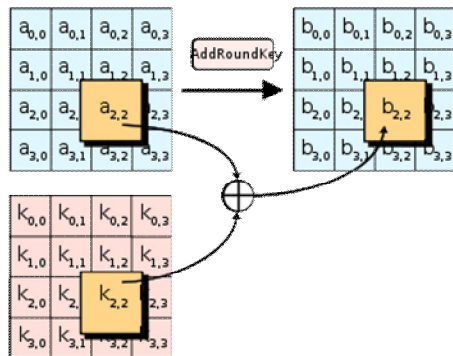


Рисунок 11.18 - Процедура AddRoundKey

У процедурі AddRoundKey RoundKey кожного раунда об'днують з State. Значення RoundKey отримують з CipherKey з використанням процедури KeyExpansion.

Процедура SubBytes() трансформує State із застосуванням нелінійної таблиці заміщення байтів S-box до кожного байта State.

У ShiftRows() виконують циклічне зміщення останніх трьох рядків State на різну кількість біт, як показано на рис. 11.19.

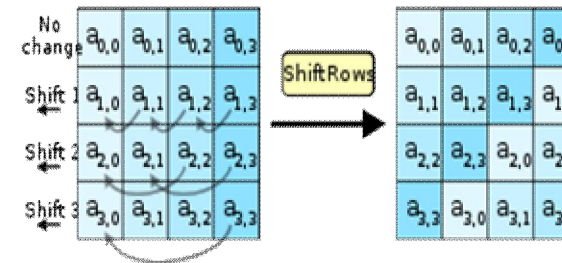


Рисунок 11.19 - Процедура ShiftRows

Байти у кожному рядку state циклічно зсувають управо на r байтів. Зсув залежить від номера рядка. Наприклад, для нульового рядка $r = 0$, для першого рядка $r = 1$.

Для алгоритму Rijndael зміщення рядків для 128 і 192 бітових рядків одноковий. Для блоку розміром 256 біт рядок 2, 3 і 4 зміщуються на 1, 3, 4 байти відповідно.

Для MixColumns() змішують всі стовпчики State для отримання нових. На рис. 11.20.16 показано схему перетворень даної процедури.

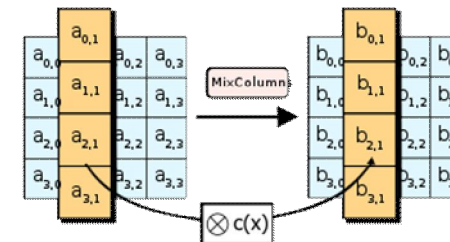


Рисунок 11.20 - Процедура MixColumns

У процедурі MixColumns кожний стовпчик стану перемножують з фіксованим багаточленом $c(x)$

$$c(x) = 3x^3 + x^2 + x + 2$$

Тобто у MixColumns чотири байти кожного стовпчика State змішують з використанням лінійної трансформації. Кожний стовпчик у MixColumns трактують як поліном четвертої ступені. Над цими поліномами виконують множення у GF(28) за модулем $x^4 + 1$ на багаточлен. Разом з ShiftRows MixColumns вносить дифузію у шифр.

У процедурі SubBytes кожний байт у state замінюють відповідним елементом у фіксованій 8-бітій таблиці заміни S; $b_{ij} = S(a_{ij})$, як показано на рис. 11.21.

Кожний вхідний байт нелінійно міняють на байт на основі таблиці заміни S-box. Побудова S-box включає отримання зворотного числа в GF {28} і застосування до кожного байта b з S-box операції

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i,$$

де $0 \leq i < 8$,

b_i є і-ий біт b, c_i — і-ий байт $c = \{63\}$ або $\{01100011\}$.

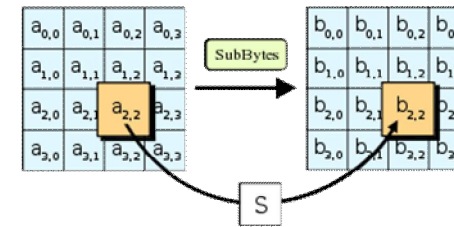


Рисунок 11.21- Процедура SubBytes

Це забезпечує захист від атак, які використовують прості алгебраїчні властивості. Утворення нового значення b показано на рис. 11.22.

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Рисунок 11.22 - Утворення значення b

Процедура KeyExpansion() на основі Cipher Key, K дозволяє отримати ключі для всіх раундів. Процедура отримує $Nb * (Nr + 1)$ слів. На початку потрібно набір Nb слів, а для кожного з Nr раундів необхідно Nb ключових наборів даних. Отриманий масив ключів позначено $w[i]$, $0 \leq i < Nb * (Nr + 1)$. Алгоритм KeyExpansion() подано на рис. 11.23.

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)


```

begin
  word temp
  i = 0;

  while ( i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while

  i = Nk

  while ( i < Nb * (Nr+1))
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end

Рисунок 11.23 - Псевдокод процедури KeyExpansion

```

Функція SubWord() застосовує S-бок для чотирьохбайтового вхідного слова по байтово. Слово $[a_0, a_1, a_2, a_3]$ у RotWord() циклічно переставляють з отриманням $[a_1, a_2, a_3, a_0]$. Масив слів, зі словом постійним для даного

раунда, $Rcon[i]$, містить значення $[x_{i-1}, 00, 00, 00]$, де $x = \{02\}$, а x_{i-1} є ступенем x у $GF\{28\}$ ($i \geq 1$).

Перші N_k слів заповнені Cipher Key. У кожне наступне слово $w[i]$ записують $w[i-1]$ XOR $w[i-N_k]$. Для слів, позиція яких кратна N_k , перед виконанням операції XOR до $w[i-1]$ застосовують перетворення, за якою виконують XOR с константою раунда $Rcon[i]$. Перетворення полягає у циклічному зсуві байтів у слові RotWord(), після чого виконують процедуру SubWord(). SubWord() подібна до SubBytes(), але для вхідних і вихідних слів розміром у слово.

Процедура KeyExpansion() для 256 бітового Cipher Key дещо відмінна від довжини 128, 192. При $N_k = 8$ і $i - 4$ кратних N_k процедуру SubWord() застосовують до $w[i-1]$.

У процесі дешифрування використовують похідні, по відношенню до попередніх, процедури:

InvMixColumns() - перетворення, обернене до MixColumns();

InvShiftRows() – перетворення, обернене по відношенню до ShiftRows()

InvSubBytes() – перетворення, обернене до SubBytes()

Функція RotWord() у процедурі Key Expansion виконує циклічну перестановку чотирьохбайтового слова.

11.6 Алгоритм IDEA

11.6.1 Загальна характеристика

Міжнародний алгоритм шифрування даних IDEA (International Data Encryption Algorithm) опубліковано у 1991 р. Алгоритм має довжину ключа 128 біт, довжину блоку 64 біт і ґрунтується на мережах Фейстеля і передбачає виконання 8.5 раундів перетворень.

Алгоритм IDEA запатентовано у ряді країн, зокрема в Австрії, Франції, Німеччині, Італії, Нідерландах, Іспанії, Швеції, Швейцарії, Англії, США з терміном дії до 2010—2011 рр. Але є можливість застосовувати цей алгоритм для некомерційного використання. У 2005 р. заявлено про новий шифр IDEA NXT, який успадковує IDEA.

11.6.2 Схема шифрування

Незашифрований 64-бітовий блок ділять на чотири 16-бітових підблоки D_1, D_2, D_3, D_4 над якими виконують операції XOR, множення за модулем $2^{16} + 1$ із застосуванням числа 216 у якості нуля, додавання за модулем 216, як показано на рис. 11.24.

Операції є несумісними між собою, оскільки мають такі властивості: жодні дві з них не відповідають дистрибутивному закону, тобто

$$a * (b + c) \neq (a * b) + (a * c)$$

і асоціативному закону, тобто

$$a + (b \oplus c) \neq (a + b) \oplus c$$

Такі властивості ускладнюють криптоаналіз і підвищують захищеність алгоритму у порівнянні з алгоритмом DES.

Для дешифрування використовують той же алгоритм.

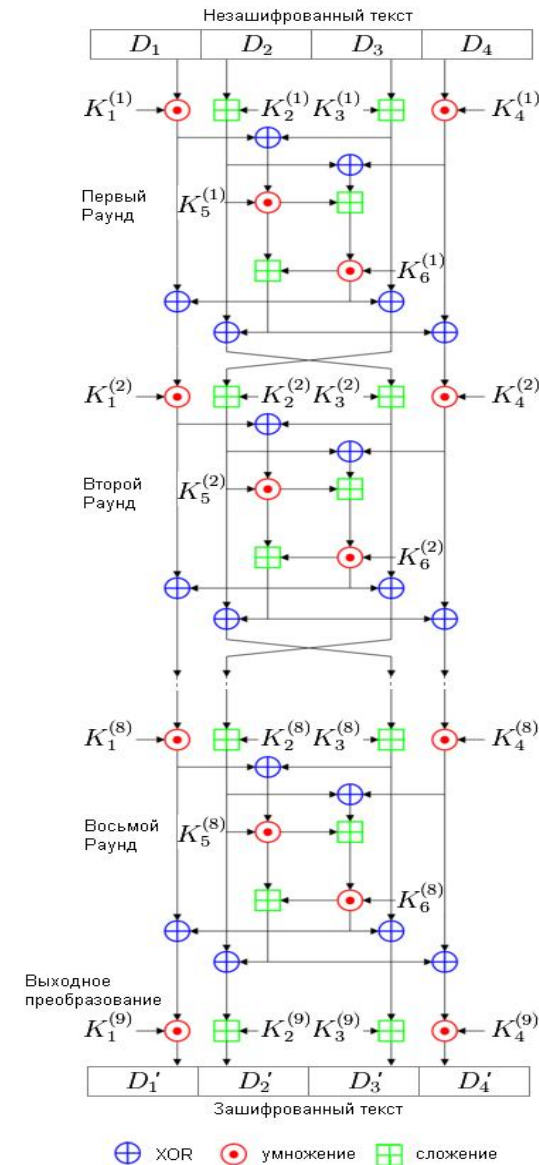


Рисунок 11.24 - Схема шифрування IDEA

Для восьми раундів шифрування з 128-бітового ключа генерують по шість 16-бітових підключів по кожному раунду, а для вихідного перетворення генерують чотири 16-бітових підключі. Генерування цих 52 підключів відбувається у такому порядку:

1. Вихідний 128-бітовий ключ розбивають на вісім 16-бітових блоки, які утворюють перші вісім підключів

$$(K_1^{(1)} K_2^{(1)} K_3^{(1)} K_4^{(1)} K_5^{(1)} K_6^{(1)} K_1^{(2)} K_2^{(2)})$$

2. Поточний 128-бітовий ключ циклічно зсувають на 25 біт, а з утвореного ключа формують наступні вісім 16-бітових підключів

$$(K_3^{(2)} K_4^{(2)} K_5^{(2)} K_6^{(2)} K_1^{(3)} K_2^{(3)} K_3^{(3)} K_4^{(3)})$$

3. Повторюють циклічний зсув і формування підключів до завершення генерування всіх 52 16-бітових підключів тобто отримання таблиці підключів табл. 11.10.

Таблиця 11.10 - Таблиця раундових підключів шифрування IDEA

Номер раунда	Підключі
1	$K_1^{(1)} K_2^{(1)} K_3^{(1)} K_4^{(1)} K_5^{(1)} K_6^{(1)}$
2	$K_1^{(2)} K_2^{(2)} K_3^{(2)} K_4^{(2)} K_5^{(2)} K_6^{(2)}$
3	$K_1^{(3)} K_2^{(3)} K_3^{(3)} K_4^{(3)} K_5^{(3)} K_6^{(3)}$
4	$K_1^{(4)} K_2^{(4)} K_3^{(4)} K_4^{(4)} K_5^{(4)} K_6^{(4)}$
5	$K_1^{(5)} K_2^{(5)} K_3^{(5)} K_4^{(5)} K_5^{(5)} K_6^{(5)}$
6	$K_1^{(6)} K_2^{(6)} K_3^{(6)} K_4^{(6)} K_5^{(6)} K_6^{(6)}$
7	$K_1^{(7)} K_2^{(7)} K_3^{(7)} K_4^{(7)} K_5^{(7)} K_6^{(7)}$
8	$K_1^{(8)} K_2^{(8)} K_3^{(8)} K_4^{(8)} K_5^{(8)} K_6^{(8)}$
Перетворення на виході	$K_1^{(9)} K_2^{(9)} K_3^{(9)} K_4^{(9)}$

Послідовність виконання перетворень шифрування така.

64-бітовий блок відкритого тексту розбивають на 16-бітові підблоки

$$(D_1^{(0)}, D_2^{(0)}, D_3^{(0)}, D_4^{(0)})$$

3. Для кожного раунда $i, i = 1 \dots 8$ обчислюють значення

$$A^{(i)} = D_1^{(i-1)} * K_1^{(i)}$$

$$B^{(i)} = D_2^{(i-1)} + K_2^{(i)}$$

$$C^{(i)} = D_3^{(i-1)} + K_3^{(i)}$$

$$D^{(i)} = D_4^{(i-1)} * K_4^{(i)}$$

$$\begin{aligned}
 E^{(i)} &= A^{(i)} \oplus C^{(i)} \\
 F^{(i)} &= B^{(i)} \oplus D^{(i)} \\
 D_1^{(i)} &= A^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \\
 D_2^{(i)} &= C^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \\
 D_3^{(i)} &= B^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})
 \end{aligned}$$

4. Для отриманих у восьмому раунді чотирьох під блоків

$$(D_1^{(8)}, D_2^{(8)}, D_3^{(8)}, D_4^{(8)})$$

виконують вихідне перетворення

$$\begin{aligned}
 D_1^{(9)} &= D_1^{(8)} * K_1^{(9)} \\
 D_2^{(9)} &= D_3^{(8)} + K_2^{(9)} \\
 D_3^{(9)} &= D_2^{(8)} + K_3^{(9)} \\
 D_4^{(9)} &= D_4^{(8)} * K_4^{(9)}
 \end{aligned}$$

Зашифрованим буде текст

$$(D_1^{(9)}, D_2^{(9)}, D_3^{(9)}, D_4^{(9)})$$

Схема дешифрування

Для дешифрування використовують інші підключі, наведені у табл.

11.11.

Таблиця 11.11 - Таблиця раундових підключів дешифрування IDEA

Номер раунда	Підключі
1	$1/K_1^{(9)} - K_2^{(9)} - K_3^{(9)} 1/K_4^{(9)} K_5^{(8)} K_6^{(8)}$
2	$1/K_1^{(8)} - K_3^{(8)} - K_2^{(8)} 1/K_4^{(8)} K_5^{(7)} K_6^{(7)}$
3	$1/K_1^{(7)} - K_3^{(7)} - K_2^{(7)} 1/K_4^{(7)} K_5^{(6)} K_6^{(6)}$
4	$1/K_1^{(6)} - K_3^{(6)} - K_2^{(6)} 1/K_4^{(6)} K_5^{(5)} K_6^{(5)}$
5	$1/K_1^{(5)} - K_3^{(5)} - K_2^{(5)} 1/K_4^{(5)} K_5^{(4)} K_6^{(4)}$
6	$1/K_1^{(4)} - K_3^{(4)} - K_2^{(4)} 1/K_4^{(4)} K_5^{(3)} K_6^{(3)}$
7	$1/K_1^{(3)} - K_3^{(3)} - K_2^{(3)} 1/K_4^{(3)} K_5^{(2)} K_6^{(2)}$
8	$1/K_1^{(2)} - K_3^{(2)} - K_2^{(2)} 1/K_4^{(2)} K_5^{(1)} K_6^{(1)}$
Перетворення на виході	$1/K_1^{(1)} - K_2^{(1)} - K_3^{(1)} 1/K_4^{(1)}$

Перший і четвертий підключі і-го раунда отримують з першого і четвертого підключів (10-і)-го раунда шифрування з використанням мультиплікативної інверсії. Мультиплікативну інверсію $1/K$ підключа K визначають з виразу

$$(1/K) * K = 1 \text{ mod } (2^{16} + 1)$$

Оскільки $2^{16} + 1$ є простим числом, то кожне число, не рівне нулю, має унікальну мультиплікативну інверсію за модулем $2^{16} + 1$.

Для раунда 1 і 9 другий і третій підключі отримують з другого і третього підключів раундів 9 і 1 шифрування адитивною інверсією. Адитивна інверсія $-K$ підключа K визначена за виразом

$$-K + K = 0 \text{ mod } (2^{16}).$$

Для раундів з 2 по 8 другий і третій підключі отримують з третього і другого підключів для раундів від 8 до 2 шифрування адитивною інверсією. Останні два підключі i -го раунда дорівнюють останнім двом підключам $(9-i)$ -го раунда шифрування.

Для шифрування повідомлень довжиною більше 64 біт алгоритм IDEA використовують у таких режимах:

електронної книги кодів ECB (Electronic Code Book);

блочного передавання зашифрованого тексту CBC (Cipher Block Chaining);

шифрування із зворотним зв'язком CFB (Cipher Feedback);

зворотного зв'язку виведення OFB (Output Feedback)

Алгоритм застосовують також для обчислення коду автентифікації повідомлення MAC (Message Authentication Code), хеш-значень, для розподілення ключів.

Основними областями застосування алгоритму IDEA є:

- шифрування аудіо і відео даних для систем кабельного телебачення, відоконференцій, дистанційного навчання, VoIP;
- захист комерційної і фінансової інформації з кон'юнктурними коливаннями;
- захист ліній зв'язку через модем, маршрутизатор, для систем ATM і GSM технологій;

- смарт-карти;
- загальнодоступні пакети конфіденційної електронної пошти PGP v2.0 и OpenPGP;
- мережі VPN;
- жорсткі диски і великі файлові системи

11.6.3 Апаратні реалізації алгоритму IDEA

Апаратна реалізація у порівнянні з програмною припускає паралельну реалізацію окремих перетворень і малі енерговитрати. Основні характеристики програмних і апаратних реалізацій подано у табл. 11.12.

Таблиця 11.12 – Реалізації алгоритму IDEA

Рік	Реалізація	Швидкість шифрування (Мб/сек)
1998	програмна	23,53
2000	програмна	44
1992	ASIC 1,5 мкм КМОП	44
1994	ASIC 1,2 мкм КМОП	177
1995	ASIC 0,8 мкм КМОП	355
1998	ASIC 0,7 мкм КМОП	424
1998	4 x XC4020XL	528
1999	ASIC 0,25 мкм КМОП	720

11.7 Поточкові симетричні алгоритми шифрування

11.7.1 Принципи роботи поточкових криптосистем

Основою шифрування у поточкових криптосистемах є комбінування відкритого тексту m з псевдовипадковими значеннями секретного поточкового ключа (keystream) k . У такому комбінуванні часто застосовують операцію XOR, а шифр називають бінарним адитивним поточковим шифром (binary additive stream cipher).

Наприклад, шифрування біту m_i відкритого тексту бітом k_i секретного ключа дозволяє отримати біт повідомлення c_i за виразом

$$c_i = m_i \oplus k_i$$

Дешифрування виконують за виразом

$$m_i * = c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i$$

Біти ключа шифрування і дешифрування повинні співпадати. Тому поточкові шифри чутливі до пропуску бітів ключа і повідомлення.

Зазвичай шифрування виконують по бітах або по байтах. Криптостійкість поточкових шифрів залежить від алгоритмів формування псевдовипадкових ключів.

Поточкові шифри дозволяють реалізувати ідею одноразових блокнотів (one-time pad – OTP), відомих також як шифри Вернама (Vernam cipher), які неможливо взламати.

Перевагами поточкових алгоритмів шифрування у порівнянні з блоковими є менші апаратні витрати і більша швидкодія. Недоліками є

більша чутливість до вибору ключів і умов шифрування, зокрема до довжини ключів, відсутності повторів відкритого тексту. Формування псевдовипадкових чисел на основі початкових заданих чисел (зазвичай, невеликого розміру, наприклад, 128 біт), полегшують умови здійснення криптоаналітичних атак.

У синхронних поточкових шифрах формування розрядів секретного ключа не залежить від попередніх розрядів відкритого тексту (при шифруванні) або шифрованого тексту (у процесі дешифрування). Самосинхронізовані поточкові шифри передбачають залежність наступних розрядів від шифрування попередніх розрядів.

Синхронні шифри передбачають узгодженість переданих біт з отриманими шифрованими бітами на приймальній стороні. Якщо у процесі передавання деякі біти можуть втрачатись (наприклад, внаслідок активних атак), то необхідно застосовувати спеціальні методи і алгоритмічні рішення відновлення зашифрованих даних на приймальній стороні. Здійснення активних атак на зашифровані дані дозволяють здійснювати передбачувані зміни у розшифрованому тексті, що надає додаткові можливості здійснення цілеспрямованого зменшення обчислень при розкритті шифру. Так, зміна у розряді шифротексту призводить до зміни одного розряду у розшифрованому відкритому тексті.

Симетричні поточкові криптосистеми, у яких для обчислення ключів використовують попередні N розрядів шифротексту називають самосинхронізованими поточковими шифрами (self-synchronizing stream ciphers), асинхронними поточковими шифрами (asynchronous stream ciphers) або шифротекстовими ключами (ciphertext autokey, скорочено СТАК). Отримувач шифротексту після отримання N розрядів за допомогою генератора потоку ключа відновлює пропущені розряди. До таких шифрів

відносять блокові шифри у режимі зворотного шифрування (block cipher in cipher-feedback mode CFB).

11.7.2 Операції та засоби реалізації поточкових шифрів

Використання лінійних регістрів зсуву із зворотним зв'язком

Регістри із зворотним зв'язком LFSR часто застосовують у поточкових шифросистемах. На рис. 11.25 показано узагальнену схему регістру зсуву.

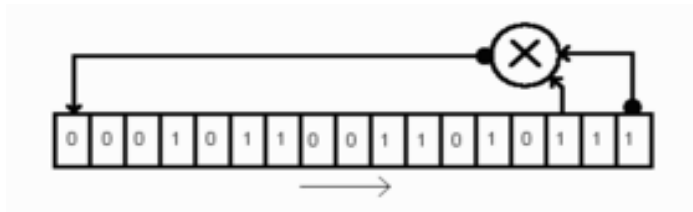


Рисунок 11.25 - Узагальнена схема зсуву

Над частиною розрядів виконують операцію (зазвичай XOR, OR, AND), а результат її виконання використовують у якості молодшого розряду регістру. Можливими є і більш складні схеми формування сигналів зворотного зв'язку.

11.7.3 Реалізація нелінійних функцій

Для створення нелінійних функцій у поточкових алгоритмах шифрування застосовують кілька регістрів LFSR, як показано на рис. 11.26.

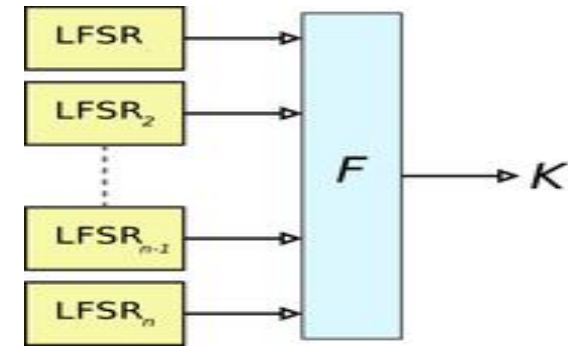


Рисунок 11.26 - Реалізація нелінійних функцій на основі регістрів LFSR

Булеву функцію F задано на n змінних, які отримують на виходах регістрів LFSR. Частина з регістрів може використовуватись із зворотними зв'язками. Утворена схема комбінаційного генератора може бути чутливою до кореляційних атак.

11.7.4 Алгоритм RC4

Шифр RC4 (Ron's Code або Rivest's Cipher) розроблено компанією RSA Security Inc. і для його використання потрібна ліцензія. Іноді його використовують під назвою ARCFOUR, ARC4 (Alledged RC4)) або алгоритму ключового розкладу (Key-Scheduling Algorithm - KSA. Алгоритм застосовують у протоколах комп'ютерних мереж, зокрема у протоколах SSL и TLS, а також в алгоритмі бездротових мереж WEP, WPA і для шифрування паролів у Windows NT.

Алгоритм побудовано на на застосуванні генератора псевдовипадкових чисел із нормальним розподілом, параметризованим

ключем. Довжина ключа зазвичай складає від 5 до 64 байт, а максимальне значення 256 байт.

До переваг шифру відносять простоту його реалізації, швидкодію і можливість працювати з ключами змінної довжини. Типова реалізація передбачає виконання 19 машинних команд на кожний байт тексту. В США рекомендовано застосовувати ключ довжиною 128 біт, для закордонних відділень 56 біт і 40 біт для експортованих шифрів.

Схему генератора потоку ключа алгоритму подано на рис. 11.27.

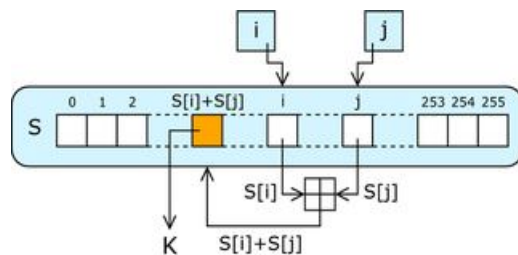


Рисунок 11.27 - Генератор ключового потоку K алгоритму RC4

Алгоритм визначено розміром блока n (на рис.4.3 n = 8). Внутрішній стан RC4 визначено масивом S (S-box), який є перестановкою всіх слів розміром 2n слів, і двома лічильниками i, j. Генератор ключового потоку кожний раз переставляє слова в S, і вибирає наступне відмінне від попередніх значення.

У процесі шифрування функція генерування ключового потоку F генерує послідовність бітів, які об'єднують з бітами відкритого тексту за модулем два. Для дешифрування виконують регенерацію потоку ключа і об'єднують із зашифрованим текстом операцією за модулем два. Для створення початкового стану ключового потоку використовують функцію ініціалізації з ключем змінної довжини.

Ініціалізацію починають із заповнення масиву S для ключа Key довжиною l байт за алгоритмом

```
for i = 0 to 2n - 1
  S[i] = i
```

Для отриманого масиву виконують скремблювання у такій послідовності

```
j = 0
for i = 0 to 2n - 1
  j = (j + S[i] + Key[i mod l]) mod 2n
  Перестановка (S[i], S[j])
```

Далі виконують ініціалізацію для масиву S

```
i = 0
j = 0
Цикл генерації
i = (i + 1) mod 2n
j = (j + S[i]) mod 2n
Перестановка (S[i], S[j])
Результат: K = S[(S[i] + S[j]) mod 2n]
```

Приклад програмної реалізації алгоритму RC4 на мові C для l=1 показано на рис. 11.28.

```
unsigned char S[256];
unsigned int I, j;
```



```

/* ключовий розклад */
void rc4_init(unsigned char *key, unsigned int key_length) {
    for (I = 0; I < 256; i++)
        S[i] = I;

    for (I = j = 0; I < 256; i++) {
        unsigned char temp;

        j = (j + key[I % key_length] + S[i]) & 255;
        temp = S[i];
        S[i] = S[j];
        S[j] = temp;
    }

    I = j = 0;
}

/* Виведення псевдовипадкового байта */
unsigned char rc4_output() {
    unsigned char temp;

    I = (I + 1) & 255;
    j = (j + S[i]) & 255;

    temp = S[j];
    S[j] = S[i];
    S[i] = temp;
}

```

Рисунок 11.28 - Програмна реалізація алгоритму RC4

Алгоритм є дуже вразливим до маніпуляції бітами. Тому рядом компаній (зокрема Microsoft) його вважають застарілим. Так у .NET Framework від Microsoft його реалізації немає.

Проведені криптоаналітичні дослідження підтвердили, що перший байт ключового потоку корельовано з першими трьома байтами ключа. Доведені також (у 2007 році) корельованість перестановки і ключа.

Відомими атаками є атаки Флурера, Мантина та Шаміра (2001 р.), які призвели до взламу шифрування WEP у бездротових мережах стандарту IEEE 802.11 і зумовили перехід до розробки нового стандарту безпечних мереж WPA.

Криптосистема стає нечутливою до атаки при відкиданні n байт ключового потоку (рекомендовано n=768, але існують консервативні оцінки n=3072).

Алгоритм RC4 не застосовує LFSR і його зручно реалізовувати програмно (до 8-16 машинних команд на байт).

11.7.5 Алгоритми VEST

Алгоритми шифрування VEST (Very Efficient Substitution Transposition) є сім'єю шифрів (VEST-4, VEST-8, VEST-16, VEST-32), орієнтованих переважно на апаратну реалізацію. Ефективна програмна реалізація цих алгоритмів невідома.

Перші офіційні повідомлення про VEST були подані у 2005 р. Довжина ключа може бути довільною, а розрядність стану становить від 256 біт (для шифру VEST-4) до 768 біт (для шифру VEST-32). Захищеність забезпечується для довжини повідомлень 80-256 біт.

В основі структур алгоритмів цих шифрів є застосування перетворень за допомогою нелінійних регістрів NLSFR (nonlinear feedback shift register), перетворень мереж підстановок і перестановок SPN (substitution-permutation network) і збалансованих Т-функцій. Основні характеристики шифрів VEST наведені у табл. 11.13.

Таблиця 11.13 - Порівняльні характеристики алгоритмів VEST

Шифр	VEST-4	VEST-8	VEST-16	VEST-32	AES-128
Розрядність на виході, біт на один такт	4	8	16	32	128
Проголошена захищеність, біт	80	128	160	256	128
Рекомендована довжина ключа, біт	160	256	320	512	128
Рекомендована довжина хешу, біт	160	256	320	512	
Довжина лічильника, біт	163	163	171	171	
Основна (core) розрядність, біт	83	211	331	587	
Розрядність стану (State), біт	256	384	512	768	128

Збалансовані Т-функції описують за допомогою бієктивного NLSFR з паралельним зворотнім зв'язком (NLPFSR), мережами SPN і нелінійними лічильниками RNS.

Алгоритм VEST включає чотири компоненти: нелінійний лічильник, лінійний перетворювач (linear counter diffusor), бієктивний нелінійний акумулятор і лінійний вихідний перетворювач (linear output combiner). На рис. 11.29 подано формування значень акумулятора для VEST-4.

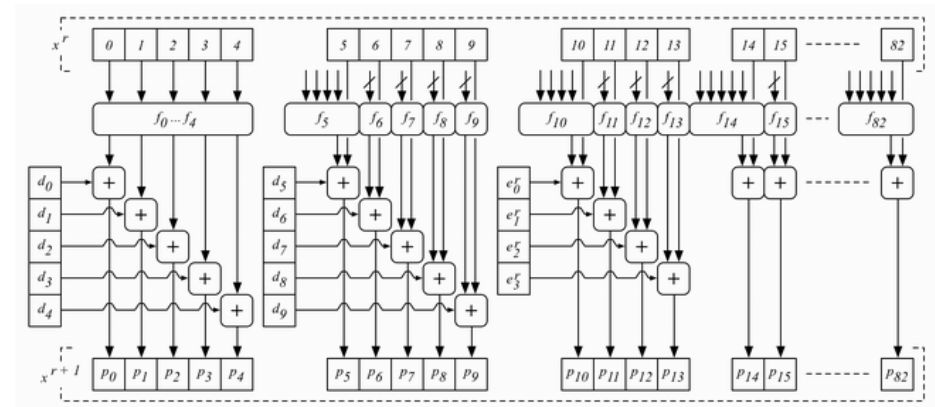


Рисунок 11.29 - Схема формування значень акумулятора для VEST-4

На вході задано біти $d_0 - d_9$. Біти $p_0 - p_4$ стану акумулятора перетворюють за допомогою блоку підстановок 5×5 і лінійно комбінують з першими вхідними бітами на кожному раунді. вхідними бітами на кожному раунді. Наступні п'ять бітів акумулятора лінійно комбінують з наступними вхідними бітами і нелінійними значеннями функції для чотирьох молодших біт акумулятора.

У режимі шифрування біти зворотного зв'язку шифрованого тексту лінійно включають у акумулятор у вигляді розрядів $e_0 - e_3$, з об'єднанням із чотирма молодшими бітами акумулятора.

Використання молодших бітів у якості вхідних даних функції є характерним для Т-функцій і являють підстановку, яка супроводжується подальшою псевдовипадковою перестановкою всіх бітів стану.

Лічильник RNS складається із шістнадцяти NLFSR. Акумулятор приймає 10 розрядів і перетворює їх у вихідні біти.

Складна структура акумулятора спричиняє складність програмної реалізації алгоритму VEST. Тому алгоритми реалізують переважно апаратно.

Деякі з характеристик апаратної реалізації подано у табл. 11.14.

Таблиця 11.14 - Характеристики апаратних реалізацій VEST

Реалізація	Тактова частота	VEST-4	VEST-8	VEST-16	VEST-32
Апаратна	250 MHz	~1 Gbit/s	~2 Gbit/s	~4 Gbit/s	~8 Gbit/s
Програмна	250 MHz	< 1.0 Mbit/s	< 0.8 Mbit/s	< 1.1 Mbit/s	< 1.3 Mbit/s
Оцінка виграшу		> 1000 x	> 2300 x	> 3500 x	> 6000 x

Подальший розвиток реалізацій на схемах з технологіями алгоритму VEST-32 показує можливість отримання 256-бітової реалізації на швидкості шифрування 10 Гб/с для 180 нм BIC технологій на основі схем ASIC з використанням 45 К вентилів і 110 нм технологій з швидкістю 20 Гб/с..

11.7.6 Порівняння потокових алгоритмів

У табл.5 подано основні дані про поширені потокові алгоритми. Найбільш вживаним є алгоритм RC4. Відомими застосуваннями є алгоритми A5/1, A5/2, Chameleon, FISH, Helix, ISAAC, MUGI, Panama, Phelix, Pike, SEAL, SOBER, SOBER-128 and WAKE.

11.7.7 Алгоритм A5

Алгоритм призначено для забезпечення захисту голосового зв'язку у стільникових телефонних системах на основі GSM. Алгоритм A5/1 на

момент створення (1987 р.) було розраховано на території Європи, а A5/2 (1989 р.) – за межами Європи.

Узагальнену структуру алгоритму подано на рис. 11.30.

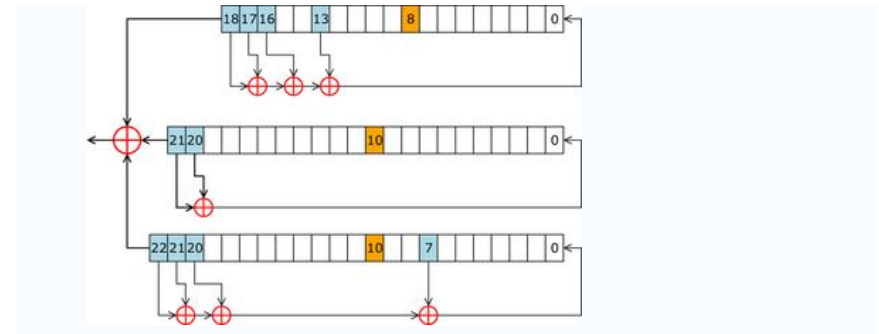


Рисунок 11.30 - Структура алгоритму A5

Алгоритм передбачає використання трьох регістрів LFSR. Регістр синхронізують за умови співпадіння тактового біту (виділено) з одним або двома тактовими бітами інших двох регістрів.

У GSM передавання даних здійснюється пакетами (bursts), які надсилаються кожні 4.615 мс і містять 114 біт для даних. В A5/1 для 114 біт створюють ключовий потік (keystream), який об'єднують через операцію XOR з 114 бітами даних. Ініціалізацію ключа виконують 64-бітовим ключем разом з 22-бітовим номером кадра. Але оскільки 10 біт фіксуються на значенні нуль, то ефективна довжина ключа становить лише 54 біти.

Реалізація зворотного зв'язку у регістрах виконують згідно з характеристичними поліномами, поданими у табл. 11.15.

Таблиця 11.15 - Характеристика LFSR

Номер LFSR	Довжина, в біт	Характеристичний поліном	Тактовий біт
1	19	$x^{18} + x^{17} + x^{16} + x^{13} + 1$	8
2	22	$x^{21} + x^{20} + 1$	10
3	23	$x^{22} + x^{21} + x^{20} + x^7 + 1$	10

Тактування роботи тригера виконується за мажоритарним принципом за співпаданням двох або трьох тактових бітів регістрів. Тому на кожному кроці два або три регістри синхронізують і імовірність синхронізації становить 3 / 4.

Початково тригери встановленні у нульове значення. Тоді за 64 такта ключ з 64 біт змішують з даними за схемою: для $0 \leq i < 64$ значення біту і додають до найменш важливого біту least significant bit (LSB) з використанням операції XOR:

$$R[0] = R[0] \oplus K[i]$$

Недоліками алгоритму є можливість здійснення пасивних атак за відомими відкритими текстами. За деякими оцінками складність здійснення атак становить 238 – 248.

Існують розробки по створенню апаратних реалізацій A5 на схемах FPGA. Останніми роками продовжуються роботи по розробці активних і пасивних крипто аналітичних атак для різних реалізацій цього алгоритму.

12 ОСНОВИ ПОБУДОВИ ВІДМОВОСТІЙКИХ СИСТЕМ

12.1 Поняття відмовостійкості

Спроможність системи надавати послуги за наявності відмов називають відмовостійкістю (fault tolerance). Розподілена система відмовляє (fail), якщо вона не в змозі виконувати свою роботу. Помилкою (error) називають стан системи, який може призвести до непрацездатності. Помилкою є також отримання пошкоджених пакетів даних каналами зв'язку. Причиною помилок може бути відмова (fault), зокрема пошкодження ліній комунікації.

Відмови поділяють на:

Перехідні (transient faults), тобто одноразові;

Перемежовані (intermittent faults), які виникають час від часу;

Постійні (permanent faults), які зберігаються постійно.

Відмовостійкість тісно пов'язана з поняттям надійних систем (Dependable System). Надійність визначає сукупність вимог до розподілених систем, включаючи:

доступність (availability), тобто готовність до роботи;

безвідмовність (reliability), тобто спроможність системи працювати без відмов;

безпеку (safety), тобто спроможність системи при відмовах не приходити до катастрофічного стану (ситуації);

ремонтно здатність (maintainability), яка визначає складність усунення відмов або відхилень у роботі розподіленої системи.

Інформаційну структуру багатьох сучасних комп'ютерних систем часто зручно подавати як розподілену інформаційну систему.

12.2 Типи відмов розподілених систем

Існує кілька класифікацій розподілених інформаційних систем (РІС) за відмовами, які можуть призвести до непрацездатності системи. За умови розгляду РІС як взаємодію серверів і клієнтів, набуло поширення виділення таких типів відмов і обставин їх виникнення:

поламки (crash failure), коли сервер раптово припиняє працювати і виконувати свою роботу;

пропуск даних (omission failure) при неправильному реагуванні на запити. Розрізняють: пропуск прийому (receive omission), наприклад, при неотриманні запиту, зокрема внаслідок того, що на сервері не запущено відповідний процес; пропуск передачі (send omission), коли сервер виконує свою роботу, але не в змозі надіслати відповідь, зокрема, при переповненні буфера. Існує багато інших типів пропусків даних, зокрема, зумовлених помилкою в програмі;

помилки синхронізації (timing failure), які виникають при очікуванні відповіді більше ніж зазначено, або у невизначений час, зокрема, помилка працездатності (performance failure), зумовлена занадто пізнім отриманням даних при передачі мультимедійної інформації;

помилки відповіді (response failure), які полягають у невірних відповідях сервера. При помилках значення (value failure) сервер повертає невірну відповідь на запит (наприклад, для пошукової машини). При помилках стану (state transition failure) сервер відхиляється від вірного потоку керування. Зокрема, він не в змозі розпізнати отримане повідомлення;

довільні помилки (arbitrary failure) або візантійські помилки (Byzantine failures), при яких сервер відправляє випадкове повідомлення у

випадкові моменти часу. Наслідком є отримання клієнтом невірних відповідей, за умови, що клієнт сприймає їх як вірні;

Відомі також помилки аварійних зупинок (fail-stop failures). В системах зупинки без повідомлення помилки (fail-silent systems) за помилку може сприйматись дуже повільна робота сервера. Сервер може висилати випадкові повідомлення, які сприймаються іншими процесами як інформаційне «сміття». Тому вони не призводять до помилки і вважаються безпечними (fail-safe).

12.3 Способи забезпечення відмовостійкості

Відмовостійка РІС повинна бути спроможна до маскування факту помилок від інших процесів. Основний метод такого маскування – використання надлишковості (redundancy). Розрізняють наступні типи введення надлишковості:

інформаційна, яка забезпечується застосуванням завадостійких кодів;

часова, що полягає у повторному виконанні дій, зокрема, повторна транзакція. Часова надлишковість важлива для прохідних і переміжючих відмовах;

фізична, яка реалізується на апаратному або програмному рівні, наприклад, із застосуванням різноманітних методів резервування.

Для забезпечення відмовостійкості досить популярним є потрійне резервування (Triple Modular Redundancy), приклад якого наведено на рис.

12.1

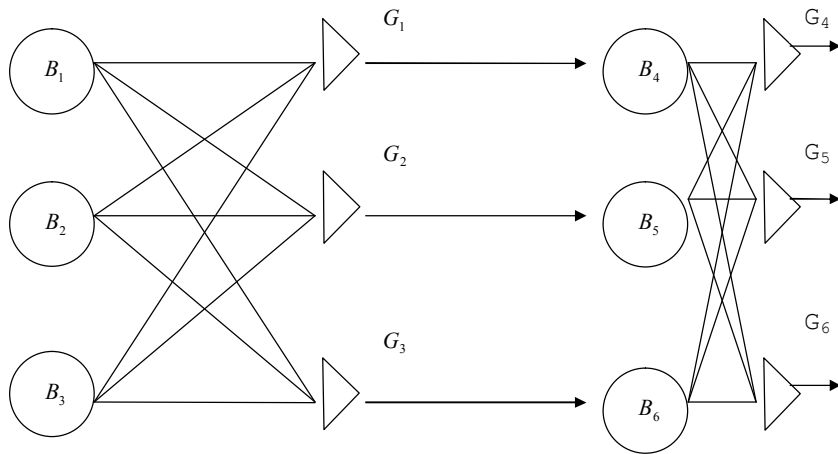


Рисунок 12.1 - Triple Modular Redundancy

Тут $B_1 \div B_6$ - функціональні модулі, а $G_1 - G_6$ - схеми голосування «2 з 3». При відмові одного з елементів B_1, B_2, B_3 , або B_4, B_5, B_6 система продовжує працювати з використанням схеми голосування два з трьох. Відмова G_1, G_2, G_3 сприймається як відмова B_1, B_2, B_3 . Такі схеми потребують занадто великих витрат апаратури або резервування програмних ресурсів.

Схеми голосування можуть бути реалізовані, наприклад, за допомогою схеми на рис 12.2 а з таблицею істинності 12.2 в

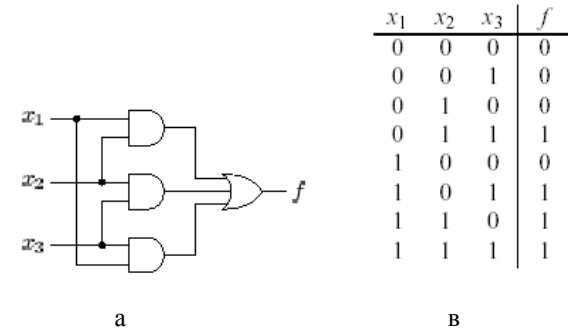


Рисунок 12.2 – Схема голосування

12.4 Групові процеси

12.4.1 Відмовостійкість процесів.

Процеси РІС є основними складовими реалізації послуг системи. Надлишковість забезпечується створенням груп ідентичних процесів. При отриманні повідомлення групою, воно передається всім членам групи. Групи можуть бути динамічними і статичними. Для процесу, який направляє повідомлення у вигляді запиту процесу-виконавцю (наприклад, серверу), реалізується хоча б один з процесів даної групи.

Структурна організація групи процесів може бути довільною. Найбільшого поширення набули однорангові групи з довільними зв'язками між процесами групи, і ієрархічні групи, в яких один з процесів виділяється як координатор інших. Тобто в однорангових групах отримання відповіді на запит відбувається колективно, а в ієрархічній – координатор здійснює вибір процесів-виконавців (рис. 12.3).

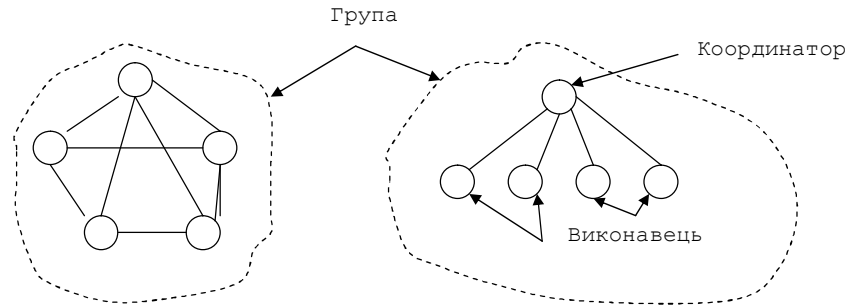


Рисунок 12.3 – Групи з різними типами зв’язків між процесами

Однорангові системи більш складні за будовою і більш ефективні, оскільки втрата одного чи декількох процесів дозволяє продовжити обробку повідомлення.

Поширеними методами знищення і створення груп є створення сервера груп (group server) і розподіленого керування членства процесів розподіленої групи. Сервер груп централізовано отримує запити до груп і реалізує базу даних груп системи і членства в них окремих процесів (рис. 12.4).

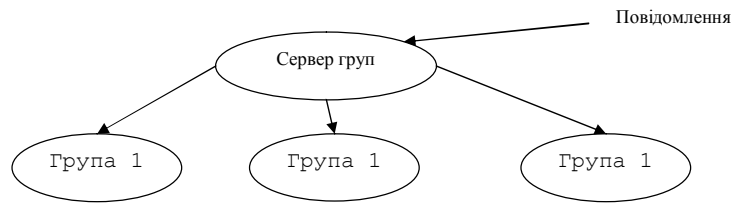


Рисунок 12.4 – Розподілене керування членства процесів

Метод розподіленого керування членством доцільно застосовувати при групових розсилках повідомлення. Зовнішній учасник групи взаємодії може надіслати повідомлення всім членам групи і об’явити намір вийти в групи. Вихід з групи здійснюється надсиланням повідомлення «продавання» членам групи. Аварійний вихід члена визначається за результатами звертання до нього інших членів групи. За відсутності відгуку при звертанні здійснюється примусове вилучення з групи.

Проблеми виникають при одночасній відмові кількох членів, що призводить до непрацездатності групи. Реалізація повторної зборки групи з працездатних процесів здійснюється за спеціальними протоколами.

Механізм маскуванню помилок здійснюється реплікацією процесів і організацією груп. Залежно від моделі відмови, потрібна різна кількість процесів для маскуванню. Так, при наявності вірних K процесів, $K+1$ -ий процес забезпечує роботу системи для найпростіших моделей. Для візантійських помилок необхідно $2K+1$ реплікованих процесів.

Реальна взаємодія процесів в групах здійснюється за рахунок застосування певних узгоджень між процесами при прийнятті рішень. Алгоритми розподіленого узгодження припускають різні передумови роботи системи: надійність каналів передачі повідомлень між процесами, механізм обробки повідомлень, визначення достовірності відповідей, способи визначення непрацездатності процесів і їх вилучення тощо. Прості моделі не відповідають реальним системам, а більш складні моделі потребують більш складних алгоритмічних рішень.

Узгодження у розподілених системах неможливе при наявності дефектного процесу, який припиняє свою роботу без сповіщення. Це характерно для дуже повільних процесів, які неможливо відрізнити від працездатних. Визначення конкретного розв’язку в кожному випадку потребує глибоких теоретичних досліджень

12.4.2 Надійний зв'язок клієнт-сервер

Надійний зв'язок забезпечується різними рівнями взаємодії. Можна виділити надійний зв'язок нижнього рівня, зокрема, з використанням TCP-протоколів, та верхнього рівня з застосуванням RPC та RMI.

Взаємодію з RPC можна поділити за помилками на п'ять класів:

- неспроможність клієнта виявити сервер;
- втрата повідомлення з запитом від клієнта до сервера;
- поламка сервера після отримання запиту;
- втрата зворотного повідомлення від сервера до клієнта;
- поламка клієнта після отримання відповіді.

Забезпечення відмовостійкості для кожного з цих класів має свої особливості. Неспроможність клієнта виявити сервер може бути наслідком відключення сервера або зміни сервером версії інтерфейсу і відповідно скелетону. Розв'язком проблеми є примусити спричинити похибку виключення (exception). Наприклад, в Java використовують обробник сигналів (подій), який спричиняє появу цього сигналу та його обробку. Це порушує прозорість за рахунок процедур виключення.

Втрата повідомлення з запитом від клієнта до сервера розв'язується шляхом застосування таймера. При переповненні таймера і неотриманні підтвердження повідомлення надсилається повторно. Повторна посилка дозволяє нормально продовжити виконання процесу.

Поламка сервера після отримання запиту призводить до двох ситуацій: поламка до відправлення зворотного повідомлення; поламка до початку обробки, як показано на рис. 12.5.

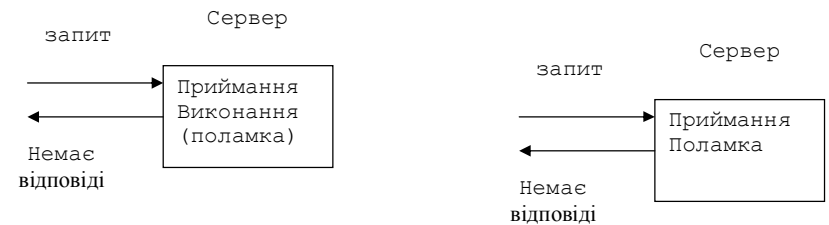


Рисунок 12.5 – Поламка сервера після отримання запиту процесів

При нормальному виконанні за запитом сервер видає необхідну відповідь (рис. 12.6)

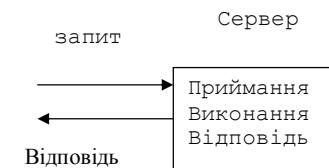


Рисунок 12.6 – Нормальне виконання за запитом

Нормальна дія при поламці в двох визначених ситуаціях дещо відрізняється. В першій ситуації клієнту треба надіслати повідомлення, а у другій ситуації потрібно надіслати запит про повторну передачу повідомлення. Але операційна система може сприймати ці дві ситуації однаково, а саме, як переповнення таймеру.

Основними методами вирішення цієї проблеми є:

очікування перевантаження сервера або спроба зв'язатись з іншим сервером та повторення операції («семантика мінімум один раз» at least once semantics);

відмова від подальших спроб і повернення повідомлення про помилку («семантика максимум один раз» - at most once semantics) – що гарантує не більше одного виклику RPC;

клієнт не отримує жодної допомоги і жодних гарантій. Виклик RPC або не виконується, або викликається довільну кількість разів.

Привабливою виглядає «Семантика в точності один раз», але при цьому виникає складність її реалізації. Таким чином, поламка сервера кардинально міняє природу RPC і проводить розподіл між однопроцесорними і розподіленими системами.

Втрата зворотного повідомлення від сервера до клієнта призводить до невпевненості клієнта у причинах відсутності відповіді. Частина операцій може, наприклад, повторюватись довільну кількість разів («ідемпотентний запит»).

Поламка клієнта після отримання відповіді. Поламка клієнта веде до ситуації «сироти» (orphans), яка породжує кілька проблем, зокрема, втрату процесорного часу, блокування файлів або інше зв'язування ресурсів. Розв'язками є:

знищення сиріт (extermination), яке полягає запису перед RPC, у спеціальному журналі. Недоліком є надлишковість об'єму інформації, можливість породження внучатих сиріт (grandorphans), вплив розподілу мережі на фрагменти, зокрема, внаслідок поламок шлюзів;

реінкарнація (reincarnation) без запису на диск. Час розбивається на епохи з послідовною нумерацією. При перевантаженні клієнт відправляє широкомовні повідомлення всім машинам з повідомленням своєї епохи. При надходженні на сервер всі віддалені обчислення за запитом клієнта припиняються;

варіант м'якої реінкарнації (gentle reincarnation). При надходженні повідомлення про зміну епох, при наявності на машині віддалених

обчислень, робиться спроба знайти їх власника. Обчислення припиняються в разі відсутності власника;

Вичерпання часу (expiration), за яким кожному виклику RPC надається стандартний час роботи T. По завершенню часу робиться запит (оренда) наступного терміну. Значення T вибирається достатнім для вмирання сиріт, що призводить до непередбачених наслідків.

12.4.3 Надійна групова розсилка.

Під надійною групою розсилкою (reliable multicasting) розуміють гарантовану доставку повідомлення всім членам групи процесів, яким воно направлено. Виділяють надійну розсилку в умовах надійного зв'язку з хибним функціонуванням, і розсилку при наявності надійного зв'язку з коректно працюючими процесами.

Можливо також наявність ненадійного зв'язку для базового системи зв'язку, що може вести до втрат повідомлень. Проста надійна розсилка в цій ситуації досягається нумерацією повідомлень і аналізом відправником відповідей (позитивних або негативних) про отримання повідомлень. Ненадходження його відправнику призводить до повторної розсилки повідомлення.

При великій кількості отримувачів повідомлень і використанні підтверджень від отримувачів може виникнути ефект зворотного удару (feedback implosion), який полягає у критичному, для відправника, повідомлення потоку підтверджень. Цей потік може бути зменшено за рахунок формування отримувачем повідомлень лише негативних підтверджень, тобто повідомлень з певними номерами. Але у відправника залишається проблема букерування старих повідомлень, що може спричинити перевантаження буфера.

Існує кілька схем розв'язку проблеми надійної розсилки, серед яких виділяють дві взаємо протилежні (полярні):

неієрархічне управління зворотнім зв'язком;

ієрархічне управління зворотнім зв'язком

При неієрархічному управлінні використовується модель пригнічення відгуків (feedback suppression), яка лежить в основі масштабованої розсилки SRM (Scalable Reliable Multicasting). В SRM повертається лише негативне підтвердження, яке розсилається відправнику і решті членів групи, яка отримують повідомлення. В результаті вони відмовляються від надсилання своїх негативних підтверджень, що призводить до покращення масштабованості розподіленої системи.

Проблемами при застосуванні SRM є можливість одночасного надсилання негативних підтверджень кількома отримувачами і проблема переривання роботи процесів, які успішно отримали повідомлення.

Ієрархічне управління зворотнім зв'язком передбачає розбивку групи на підгрупи з власними координаторами, як показано на рис. 12.7.

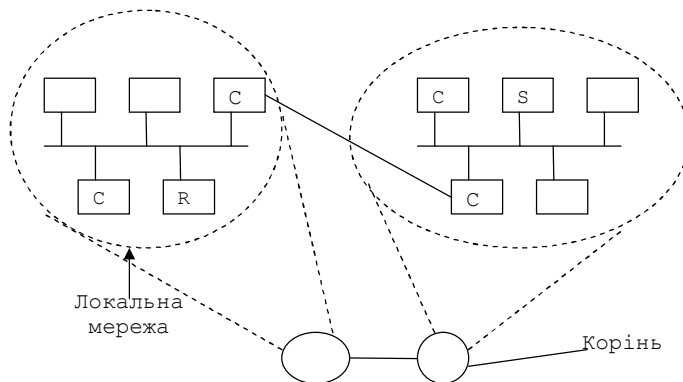


Рисунок 12.7 – Ієрархічне управління зворотнім зв'язком

Тут С – координатор, S – відправник повідомлення, R – отримувач повідомлення.

Локальні координатори відповідають за обробку запитів на повторну передачу. Ці запити відправляються отримувачами підгрупи локальної мережі. Координатор підтримує буфер повідомлень. Якщо сам координатор пропускає повідомлення, то він звертається до координатора батьківської підгрупи.

В схемі з підтвердженням локальний координатор направляє підтвердження батьківському координатору. При отриманні підтвердження повідомлення всіма членами підгрупи і їх нащадками координатор вилучає повідомлення з власного буфера. Існує цілий ряд проблем побудови дерев підгруп, зокрема на основі маршрутизаторів, які доповнюються функціями групової розсилки. Доставка повідомлень при груповій розсилці в певному порядку за умови «всім процесам групи або нікому» при можливій наявності помилок в процесах отримало назву проблема атомарної групової розсилки AMP (atomic multicast problem).

Прикладом застосування є репліковані БД, які є надбудовою над розподіленою системою. Кожній репліці відповідає процес БД. Зміни надсилаються всім членам групи реплік, і далі виконуються локально. При поламці репліка AMP передбачає відмову у розсилці повідомлень групі. Репліки групи можуть вилучити поламани репліку на основі нового узгодження про членство, забезпечуючи подальшу вірну роботу. Подальше входження репліки до групи встановлюється входженням її з встановленням стану, який відповідає поточним станам реплік групи.

Таким чином, надійна групова розсилка може бути визначена в термінах груп процесів і зміни членства в групі, та описана в вигляді моделі доставки і прийому повідомлень. Проходження повідомлення показано на рис. 12.8.

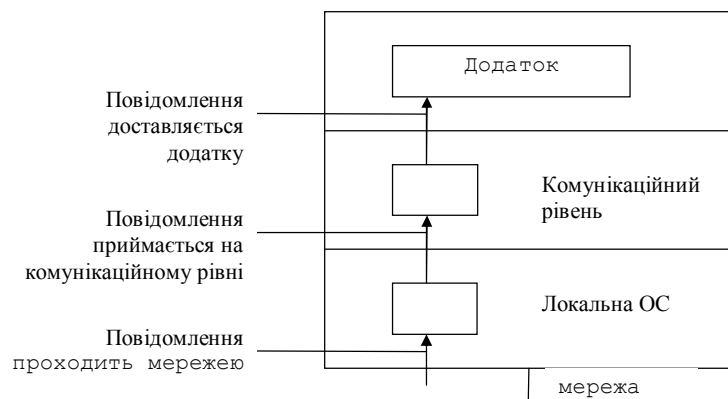


Рисунок 12.8 – Проходження повідомлення

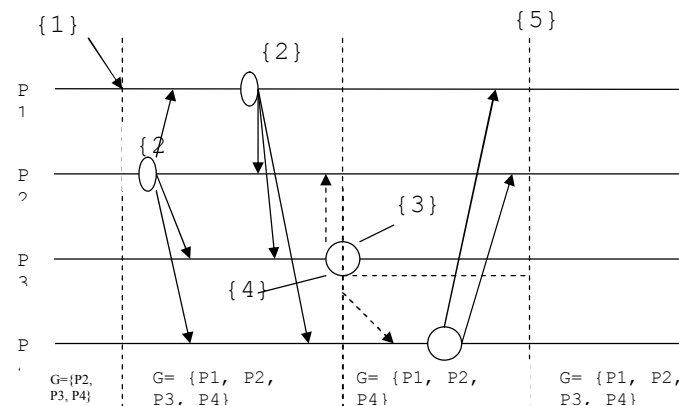
Розподілені системи включають комунікаційний рівень з локальним буфером для зберігання повідомлень. При атомарній груповій розсилці повідомлення m членам групи асоціюється зі списком процесів - отримувачів. Список розсилки відповідає поданню групи (group view), тобто групи процесів, з якими працює відправник в момент розсилки повідомлення m . Процеси групи G мають однакове подання і згодні з розв'язком за AMP.

При виході з групи міняється подання і членам групи надсилається повідомлення vc про входження або вихід процесу з групи G . Таким чином, в системі знаходяться повідомлення m і vc і треба гарантувати, що m або буде доставлено процесам з G до отримання ними vc , або взагалі не доставлено.

Доставка m припускає відмову у разі поламки відправника m і отримання членами групи G повідомлення про аварійне завершення. Повідомлення m ігнорується у разі поламки процесу до його відправки.

При відмові відправника у ході розсилки повідомлення або приймається або відмовляється всіма процесами, що залишилися. Надійна групова розсилка з цими властивостями називається віртуально-синхронною (virtually synchronous).

Приклад розсилки повідомлень процесами P1, P2, P3, P4 показано на рис. 12.9.



- де : {1} – процес P1 приєднується до групи;
- {2} – надійна групова розсилка кількох повідомлень;
- {3} – відмова процесу P3;
- {4} – часткова групова розсилка від процесу P3 анулюється;
- {5} – відновлення роботи процесу P3 і приєднання до групи.

Рисунок 12.9 – розсилки повідомлень процесами P1, P2, P3, P4

В момент 1 до групи надсилається процес P1. До моменту 3 здійснюється надійна розсилка двох повідомлень. В момент 3 відбувається поламка P3 і повідомлення, які він надсилає, анулюються. До моменту 5 на основі нового узгодження відбувається робота групи з процесів P1, P2, P4.

Після приєднання P3 в момент 5, P3 приймає участь у груповій розсилці повідомлень.

Принцип віртуальної синхронної розсилки у проміжках між змінами подання, яке утворюють бар'єри, неподоланні для розсилки. Це дозволяє розробнику додатку вважати, що групові розсилки відбуваються в різні епохи.

При групових розсилках важливим є порядок їх слідування. Можна виділити чотири основних варіанти:

Надійна неупорядкована групова розсилка (reliable unordered multicast).

Надійна групова розсилка у порядку FIFO.

Надійна причинно-упорядкована групова розсилка

Повністю упорядкована групова розсилка.

Перший варіант є віртуальною синхронною груповою розсилкою, яка не дає ніяких гарантій порядку приходу повідомлень до різних процесів. Нехай примітиви відправки і отримання повідомлень процесами P1, P2, P3 мають вигляд(рис. 12.10):

Процес	P1	Процес P2	Процес P3
Відправка m1		Отримання m1	Отримання m2
Відправка m2		Отримання m2	Отримання m1

Рисунок 12.10 – Примітиви відправки і отримання повідомлень

Процеси не ламаються, а комунікаційний рівень передає відповідному процесу повідомлення у визначеному порядку.

У другому варіанті комунікаційний рівень повинен доставляти повідомлення в порядку їх відправки (reliable FIFO – ordered multicast)

комунікацій рівень повинен доставляти повідомлення у порядку їх відправки. Наприклад, у порядку, показаному на рис. 12.11.

Процес	P1	Процес P2	Процес P3	Процес P4
Відправка m1		Отримання m1	Отримання m3	Відправка m3
Відправка m2		Отримання m2	Отримання m1	Відправка m4
		Отримання m3	Отримання m2	
		Отримання m4	Отримання m4	

Рисунок 12.11 – Примітиви відправки і отримання повідомлень

Комунікаційний рівень P2 після отримання m2 очікує m1, і доставляє його перед m2, а m3 перед m4.

Однак обмежень на доставку повідомлень, відправлених різними відправниками немає. Тобто якщо процес P2 приймає m1 перед m3, то ці повідомлення можуть бути йому доставлені йому саме в такому порядку. Але процес P3 отримує m3 перед m1. Повідомлення m3, m4 від іншого відправника теж упорядковуються.

При третьому варіанту упорядкованої розсилки (reliable causally-ordered multicast) повідомлення доставляється в порядку потенційного причинного зв'язку між ними. Наприклад, якщо m1 причинно передує m2, то порядок їх доставки буде m1, m2, незалежно від того, належать вони одному чи різним відправникам.

Варіант повністю упорядкованої групової розсилки (total-ordered multicast) передбачає, що незалежно від того, як упорядкована доставка повідомлення (причинно, за FIFO, або не упорядковано), додатково вимагається, щоб повідомлення доставлялось всім членам групи в однаковому порядку. Для попереднього прикладу P2, P3 може спочатку отримати m3, а потім m1. Процес P3 порушує упорядкованість, оскільки отримує m3, а потім m1.

Віртуальна синхронна надійна групова розсилка, яка підтримує повністю упорядковану доставку, називається атомарною груповою розсилкою (atomic multicasting).

12.4.4 Надійна групова доставка

Прикладом реалізації надійної групової доставки є система Isis, яка базується на використанні в мережі TCP механізмів наскрізного зв'язку. Відмова відправника повідомлення m може бути лише до його відправки. Повідомлення отримуються комунікаційним рівнем за FIFO на основі з'єднання за протоколом TCP.

Кожний процес групи G зберігає m до отримання його всіма членами групи, а саме m називають стійким повідомленням (stable).

Нехай при поточному поданні G_i необхідно встановити нове подання G_{i+1} , яке відрізняється від G_i мінімум одним процесом. Деякий процес P дізнається про зміну подання на основі повідомлення, можливо від процесу, який входить або покидає групу, або від процесу, який виявив відмову одного з процесів групи, що має бути вилученим з групи.

Наприклад, процес 4 повідомив, що процес 7 відмовив (рис. 12.12).

Нехай процес 6 дозволяє змінити подання G_i на G_{i+1} і надсилає нестійкі повідомлення \square (помічає їх як стійкі), і повідомлення узгодження (закреслений прямокутник) на всі члени групи, крім процесу 7 (рис. 12.13).

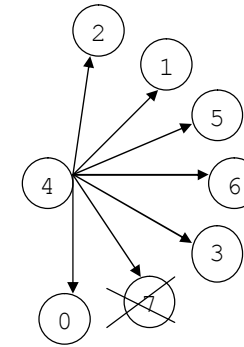


Рисунок 12.12 – Надсилання і отримання повідомлень за умови відмови процесу 7

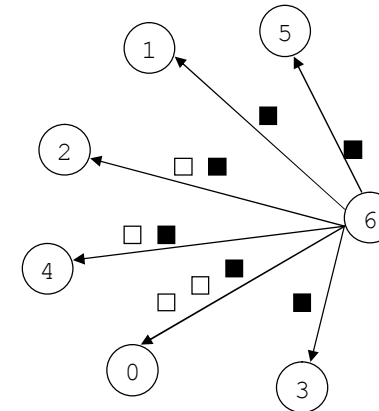


Рисунок 12.13 – Надсилання і отримання повідомлень процесом 6

Після отримання процесом P повідомлення узгодження (flush message) для подання G_{i+1} від іншого процесу, він може встановити нове подання на основі отриманих повідомлень узгодження (рис. 12.14).

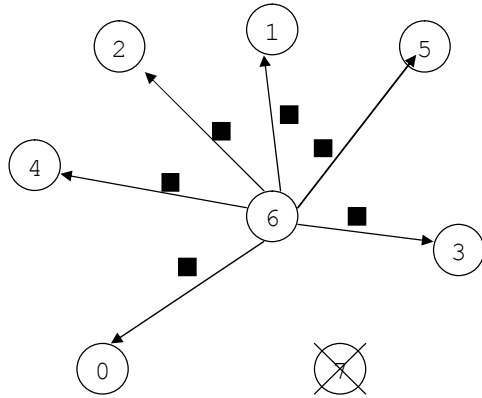


Рисунок 12.14 – Встановлення нового подання процесом 6

Процес Q, отримавши повідомлення m відповідно до подання G_i , доставляє його з врахуванням порядку слідування. При його повторі, він пропускає m .

При отриманні повідомлення про зміну подання G_{i+1} процес Q також пересилає свої нестійкі повідомлення з подальшим пересиланням повідомлення узгодження членам G_{i+1} .

Недоліком цього протоколу є його неспроможність боротися зі збоями процесів в момент об'яви нової зміни преставлення. Тобто пропущення, що до встановлення нового подання G_{i+1} кожним членом G_{i+1} жоден процес в G_{i+1} не відмовить. Ця проблема розв'язується шляхом об'яви про зміну подань для довільного подання G_{i+k} ще до того, як попередні зміни будуть встановлені всіма процесами.

12.4.5 Розподілене підтвердження.

Розподілене підтвердження (Distributed commit) являє собою більш загальну задачу у порівнянні з атомарною груповою розсилкою і включає виконання операцій з одним членом групи або всією групою.

При надійній груповій доставці ця операція являє собою доставку повідомлення. При розподілених транзакціях операцією є підтвердження транзакції на одному з сайтів, задіяних для транзакції. Використовуються схеми розподіленого підтвердження: одно, двох та трифазні. В цих схемах застосовують спеціальні процеси-координатори. Прості схеми, відомі як протокол однофазного підтвердження (one-phase commit protocol), координатор повідомляє процесам-учасникам про їх стан локально для локального здійснення операції.

В протоколі двофазного підтвердження 2PC (Two-phase commit protocol) для розподіленої транзакції подання координатора як скінченого автомата показано на рис. 12.15.

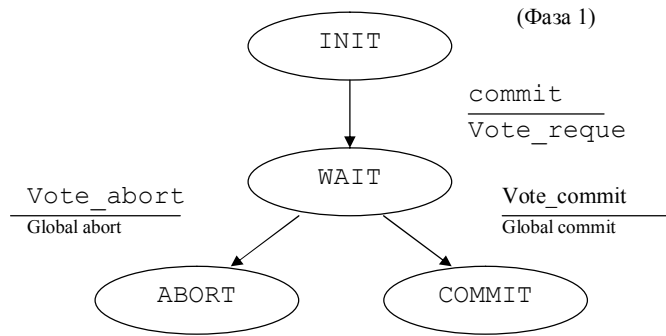


Рисунок 12.15 – Протокол двофазного підтвердження

Вершини відповідають станам координатора, а дуги- переходу зі стану в стан. Чисельник вказує очікуване повідомлення, а знаменник – повідомлення що надсилається.

Скінчений автомат для учасника показано на рис. 12.16.

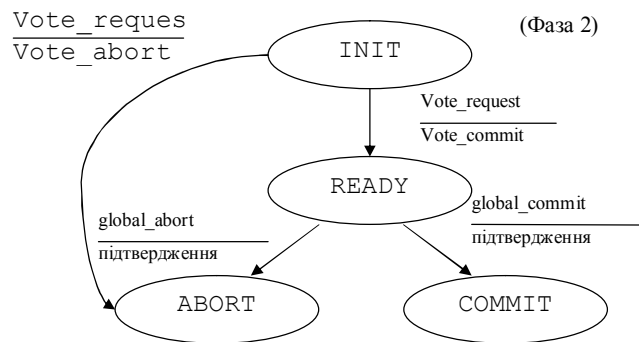


Рисунок 12.16 – Скінчений автомат для учасника

За відсутності помилок перша фаза (голосування) полягає у виконанні двох кроків:

координатор розсилає всім учасникам повідомлення `vote_request`.

учасник відповідає на `vote_request` повідомленням `vote_commit` (тобто готовність локально підтверджувати свою частину розподіленої транзакції), або повідомленням `vote_abort` - аборт в протилежному разі.

Друга фаза включає два кроки:

координатор збирає відповіді учасників. При підтвердженні транзакції всіма учасниками, координатор починає виконувати відповідні дії і надсилає всім учасникам повідомлення `Global_Commit`, а при перериванні транзакції принаймні одним учасником розсилає повідомлення `Global_Abort`

Якщо учасник проголосував за підтвердження і отримав повідомлення `global_commit`, то він локально підтвердив транзакцію. Інакше, при отриманні `global_abort` він локально перериває транзакцію.

При наявності помилок координатора або учасника мають місце стани, у яких ці процеси блокуються і очікують приходу повідомлення (для учасника `INIT`, `READY`, а для координатора `WAIT`). Тому при відмові процесу виникає проблема нескінченного очікування, яке можна подолати, застосовуючи механізм тайм-ауту (`time out`). Координатор в стані `WAIT` очікує приходу голосів всіх учасників. Учасник в стані `INIT` очікує повідомлення `void_request`, а в `READY` - `global_abort` або `global_commit` від координатора. Але цей розв'язок не є ефективним.

Кращим розв'язком цієї проблеми є дозвіл на контактування учасника `P` з учасником `Q` з виконанням відповідних дій залежно від поточного стану `Q`. Наприклад, при перебуванні `P` в стані `READY`, на основі контактів з `Q`, можуть виконуватися такі дії (рис. 12.17).

Стан учасника Q	Дії учасника P	Можливі причини
COMMIT	Перейти в стан COMMIT	P не отримав GLOBAL_COMMIT
ABORT	Перейти в стан ABORT	Q не отримав vote_request
INIT	Перейти в стан ABORT	Q не отримав vote_request
READY	Зв'язатися з іншим учасником	Блокування, до отримання повідомлення global_abort або global_commit від координатора

Рисунок 12.17 – Скінчений автомат для учасника

Проблеми виникають, коли учасник відмовляє у стані READY. Тоді після відновлення він не може визначити, на основі власної інформації, завершувати чи переривати транзакцію.

У координатора є два критичних стани. Коли він розпочинає протокол 2PC, він повинен зберегти дані про те, що знаходяться у початковому стані WAIT, щоб після відновлення повторно розіслати vote_request всім учасникам. Він також повинен зберегти рішення, прийняте в ході другої фази, для повторної розсилки після відновлення.

Для збереження повідомлень учасник і координатор можуть застосувати спеціальний журнал повідомлень перед їх розсилкою.

12.5 Відновлення процесів

Відновлення після відмов частини системи полягає у заміні хибного стану системи вірним станом без похибок. Використовують два основних способи відновлення після помилок:

зворотне виправлення (backward recovery), яке полягає у поверненні до попереднього стану, яке визначається створеною контрольною точкою (checkpoint);

пряме виправлення (forward recovery), за яким робиться спроба перевести систему у новий працездатний стан, що вимагає попередніх знань про характер помилок.

Зворотне виправлення більш поширене, але вимагає витрат у продуктивності роботи системи, і залежить від конкретного додатку, що зумовлює неможливість реалізації у деяких випадках (наприклад, у незворотних банківських процесах в UNIX –системах).

Тому перед порядок зі створенням контрольних точок застосовують протоколювання повідомлень (message logging), за яким повідомлення перед відправкою записують у журнал (при протоколюванні відправником у журнал sender logging, а отримувачем у журнал receiver-based logging). Тоді у процесі, що приймає, при виникненні відмови виконується відновлення останньої контрольної точки і повторно запускаються всі прийняті повідомлення, що краще відповідає відновленню реального стану обробки даних. Тому на практиці застосовують комбінацію невеликої кількості контрольних точок, з протоколюванням повідомлень.

Для зберігання інформації при відновленні використовують сховища трьох категорій: ОЗП, дискові сховища і стійкі сховища (stable storage), спеціально розроблені для виживання у довільних ситуаціях. В останніх застосовують точні копії дисків, що дозволяє визначити і виправити такі помилки. Такі пристрої доцільно застосовувати для збереження транзакцій.

При створенні контрольних точок записуються несуперечливі глобальні стани або розподілені знімки (distributed snapshot).

Кожний процес час від часу записує свій стан в локальне стійке сховище. Відновлення стану системи виконується з цих локальних станів, зазвичай у вигляді останнього розподіленого знімку, який називають границею відновлення (recovery line). Приклад відновлення показано на рис. 12.18.

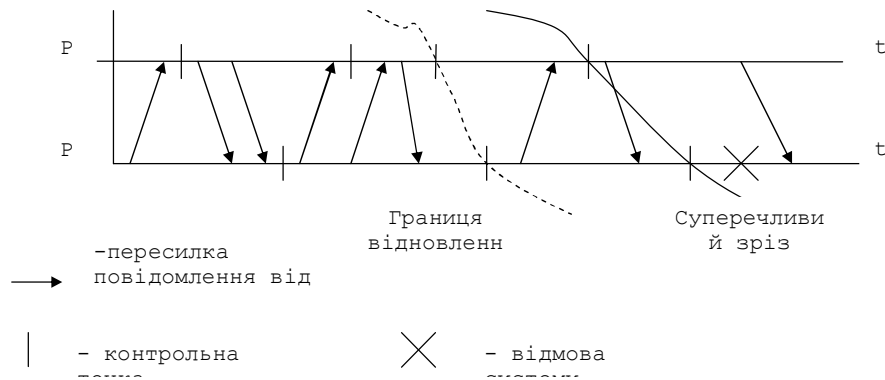


Рисунок 12.18 – Відновлення стану системи

Розрізняють незалежне і координоване створення контрольних точок. При незалежному створенні (independent checkpointing) локальні стани процесів фіксуються без врахування границі відновлення, що може призвести до ефекту доміно при відновленні (domino effect). Цей ефект полягає у послідовному відкаті по контрольних точках процесів, зумовлений відсутністю границі відновлення (рис. 12.19).

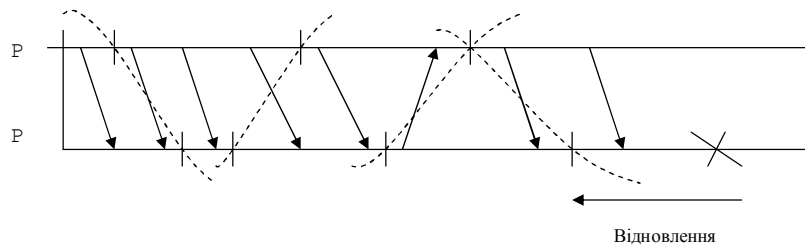


Рисунок 12.19 – Відкат по контрольних точках процесів

Іншим недоліком є проблема «збирання сміття» зі сховища контрольних точок, що вимагає визначення границь відновлення.

Для створення несуперечливих глобальних станів застосовують такий механізм. Позначимо контрольну точку з номером m процесу P_i через $CP[i](m)$, і через $INT[i](m)$ інтервал часу між контрольними точками $CP[i](m-1)$ і $CP[i](m)$.

Відправку в інтервалі $INT[i](m)$ процес P_i супроводжує парою (i, m) . При отриманні процесом P_j в інтервалі $INT[j](n)$ повідомлення $i (i, m)$ від P_i , процес P_j записує залежність $INT[i](m) \rightarrow INT[j](n)$ при створенні контрольної точки $CP[j](n)$.

При відкаті процесу P_i в контрольній точці $CP[i](m-1)$ необхідно переконавшись у несуперечливості глобального стану для процесів, що отримали від нього повідомлення в інтервалі $INT[i](m)$.

Наприклад, процес P_j як мінімум повинен відкотитись до контрольної точки $CP[j](n-1)$. Якщо $CP[j](n-1)$ не призводить до несуперечливого стану, треба продовжити відновлення до контрольної точки $CP[j](n-2)$, і т.і.

При координованому створенні контрольної точки (coordinated checkpointing) процеси синхронізують у такий спосіб, щоб запис їх станів в локальне стабільне сховище відбувалось одночасно, за умови глобальної несуперечливості стану. Для цього застосовують спеціальний алгоритм розподіленого знімку стану.

Найпростішим є використання двохфазного протоколу блокування. Спочатку координатор розсилає процесам повідомлення CHECKPOINTING_REQUEST. При отриманні процесом цього повідомлення, він створює локальну контрольну точку і починає упорядковувати в чергу всі подальші повідомлення які приходять до нього від працюючого додатку, а відправляє координатору підтвердження

контрольної точки. Координатор, отримавши повідомлення від всіх процесів, розсилає повідомлення CHECKPOINT_DONE, яке дозволяє продовжити виконання блокованим процесом.

Удосконаленням цього алгоритму є групова розсилка запиту на створення контрольних точок тільки тим процесам, які залежить від координатора. Процес залежить від координатора, якщо він отримує повідомлення, яке прямо або опосередковано пов'язано з повідомленням, яке відправлено координатором з моменту створення попередньої контрольної точки. Це приводить до поняття інкрементного знімку стану (incremental snapshot).

Для отримання цього знімку координатор розсилає запит на створення контрольних точок тільки тим процесам, яким він відправляє повідомлення з моменту створення останньої контрольної точки. Процес P, отримавши цей запит, розсилає запит всім іншим процесам, до яких він з моменту створення контрольної точки сам надсилав повідомлення і т.і.

Після визначення всіх процесів, виконується друга групова розсилка для завершення створення контрольних точок і запуску виконання процесів місця, де вони були зупинені.

Для зменшення витрат при створенні контрольних точок застосовують різні технології. Поширеною є технологія протоколювання повідомлень (message logging), яка полягає у простому відтворенні відправки повідомлень, їх отриманні і обробці після створення контрольної точки.

Цей підхід працює при виконанні умов кусочно-детермінованої моделі (piecewise deterministic model), згідно з якою виконання кожного процесу поділяється на послідовність інтервалів, у які відбуваються події (за Лампортом). Інтервал починається з не детермінованої події, наприклад, з моменту отримання повідомлення, і завершується перед

наступною не детермінованою подією. Не детерміновані події записуються у журнал.

Існує багато схем протоколювання, які розрізняються, в основному, роботою з процесами – сиротами (orphas process). Це процес, стан якого протирічить стану іншого процесу, який спочатку відмовив, а потім відновився. Приклад такого процесу показано на рис. 12.20.

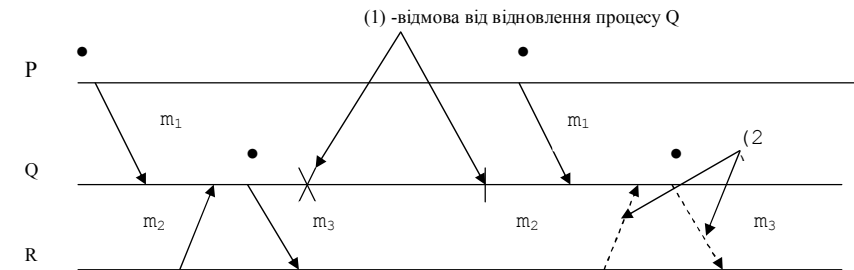


Рисунок 12.20 – Відкат по контрольних точках процесів

Крапками позначено протоколювані повідомлення. У випадку (2) оскільки не вдається відновити m2, то не відновлюється і повідомлення m3. Приклад показує, що невірне відтворення подій приводить до появи процесів-сиріт.

На початку процес Q отримує від процесів P і R повідомлення m1, m2, і направляє повідомлення m3 процесу R. Оскільки m2 не записано до журналу, то при відновленні буде задіяно лише m1, відсутність m2 призводить до заборони пересилання m3. Сирота R має m3, яке не може використати, і стан його протирічить стану процесу Q.

Для уникнення таких ситуацій і застосовують різні схеми протоколювання. Для їх характеристики введено позначення і визначені поняття. Кожне повідомлення m має заголовок, який містить інформацію,

необхідну для повторної передачі цього повідомлення і його обробки. Повідомлення називають стабільним (stable), якщо його не можна втратити, наприклад, записавши у сховище. Від приходу повідомлення m залежить виконання процесів $DEP(m)$. Якщо m' причинно залежить від m , тобто $m \rightarrow m' \rightarrow Q$ і доставляється Q , то Q входить до $DEP(m)$. Повідомлення може надходити від іншого, порівняно з m , процесу.

Набір $COPY(m)$ містить процеси, які мають копію m і можуть бути відправленими, але не зберігатись у надійному локальному сховищі. Після отримання Q повідомлення m , процес Q стає членом $COPY(m)$.

Процес Q є сирота після відмови кількох процесів, якщо існує повідомлення m , за яким Q належить $DEP(m)$, а всі процеси з $COPY(m)$ відмовили.

Для уникнення процесів-сиріт необхідно, щоб в $DEP(m)$ не було виживши процесів, якщо відмовляють всі процеси в $COPY(m)$. (тобто $Q(t)$ належить до перетину множин $DEP(m)$, $COPY(m)$) Для цього кожен процес, що залежить від m , повинен зберігати його копію.

Існує два основних підходи розв'язання цієї задачі:

1) протоколи песимістичного протоколювання (pessimistic logging protocols), за якими нестійке повідомлення надходить хоча б до одного залежного від нього процесу;

2) протоколи оптимістичного протоколювання (optimistic logging protocols), у яких робота починається після відмови процесу. Наприклад, для m при відмові всіх процесів з $COPY(m)$ процес-сирота з $DEP(m)$ відкатується у стан, коли він перестає входити до $DEP(m)$.

У песимістичному протоколі найгіршим є відмова процесу P до запису в нього повідомлення m , що забороняє передачу ним власних повідомлень, поки m не буде записано у стале сховище, що гарантує повторну передачу. Це дозволяє уникнути появи процесів - сиріт.

12.6 Моделі системного діагностування

Відмовостійкість системи є її здатність виконувати свої функції за наявності відмов деяких елементів. У відмовостійкій системі відмова одного або кількох елементів призводить лише до часткового зниження якості роботи. Відмовостійкість забезпечується застосуванням методів і засобів введення надлишковості (наприклад використанням методів резервування, паралельного обчислення тощо).

Найраціональніше використання апаратних та програмних засобів у створенні відмовостійких систем досягається за рахунок застосування методів і засобів системного діагностування. Під час розв'язання задач системного діагностування виникає дві проблеми:

побудова моделей взаємодії елементів, що призводить до одержання результатів контролю;

одержання та обробка результатів контролю з метою діагностування працездатності елементів.

Системне діагностування ґрунтується на аналізі синдрому системи, що одержують як результат взаємного контролю пар елементів системи. Результатом є визначення технічного стану системи з врахуванням моделі діагностування.

Процес системного діагностування (СД) визначається як множина взаємодій пар елементів a, b з одержанням певних результатів контролю c . Позначимо множини станів працездатності a, b, c через X, Y, C :

$X = \{ a, \bar{a} \} = \{ b, \bar{b} \}$ $C = \{ c, \bar{c} \}$. Базову множину S_0 станів СД (БСД) означимо як декартовий добуток $S_0 = X \times Y$. Сукупність підмножин Z , для яких виконується умова

$$Z \subset SO, Z \neq SO, Z \neq \emptyset$$

назвемо початковою множиною станів (ПМС) і позначимо через SU.

Пару (Z1, Z2), що задовольняє умовам

$$Z1, Z2 \in SU, Z1 \neq Z2, Z1 \not\subset Z2, Z2 \not\subset Z1$$

і відповідає станам С, будемо називати моделлю взаємодії під час системного діагностування (МД). Така формалізація дає можливість розглядати процеси системного діагностування як пошук станів множин X, Y працездатності елементів a,b, тобто технічного стану системи за результатами контролю (синдрому) системи.

Двомісною логічною функцією F(x,y), істинне значення якої набувається для підмножин Z, можна описати одержані внаслідок взаємодії пар елементів результати контролю. Кількість таких функцій - 14.

Позначимо модель взаємодії елементів(MBE) DM

$$DM = (M1, M2), |M1|=k1, |M2|=k2$$

де M1, M2 - припустимий набір стану працездатності взаємодіючих елементів за двома станами результатів контролю.

Моделі MBE можна разбити на класи за потужністю k підмножин визначення функцій F(x,y). З класом MSD(k1,k2) співвідноситься множина MBE з кількістю k=k1 і k= k2 відповідно для першого і другого результатів контролю, тобто

$$MSD(k1,k2) = \{ (Fi, Fj) \mid Fi \in F^{k1}, Fj \in F^{k2}, Fi \neq Fj \},$$

$$\text{де } F^k = \{ F(x,y) \mid \forall (x,y) \in Z, |Z|=k : F(x,y)=1 \}.$$

За умови $k1 \leq k2$ кількість моделей $N(MSD(k1,k2))$ відповідного класу $N(MSD(k1,k2)) = |MSD(k1,k2)|$ становить

Позначення	Кількість
N(MSD(1,1))	6.
N(MSD(1,2))	24.
N(MSD(1,3))	16
N(MSD(2,2))	15.
N(MSD(2,3))	24.
N(MSD(3,3))	6.

На основі цих даних можна оцінити кількість моделей за деякими умовами. Так, зокрема для MSD0, $MSD0 = \cup MSD(k1,k2)$, для $k1+k2 \geq 4$, тобто $MSD0 = MSD(1,3) \cup MSD(2,2) \cup MSD(2,3) \cup MSD(3,3)$

$$|MSD0| = 61$$

Значне зменшення кількості моделей досягається під час введення обмежень, пов'язаних з властивостями процесів системного діагностування. Важливими є властивості взаємодії, за яких забезпечується непоглинання множин станів Z за різними результатами контролю і повноті множини станів елементів.

При розробці засобів системного діагностування враховують властивості процесів діагностування та програмно-апаратних засобів для їх реалізації. Властивість непоглинання за результатами контролю задовольняє умови:

якщо $(Fi, Fj) \in MSD0$, то

$$Zi = \{ (x,y) \mid Fi(x,y) = 1 \} \quad Zj = \{ (x,y) \mid Fj(x,y) = 1 \}$$

і не має місця жодне з співвідношень $Zi \subset Zj, Zj \subset Zi$

Властивість повноти припустимих станів елементів задовольняє умови:

$$\forall F_i, F_j, F_i \neq F_j, \quad Z_i \cup Z_j = S_0$$

На основі цих властивостей узагальнений процес проектування засобів системного діагностування полягає у виконанні таких кроків:

- 1) визначити структуру зв'язків елементів, що виникають у процесі системного діагностування;
- 2) визначити процедури діагностування для одержання результатів контролю і побудувати модель МВЕ;
- 3) на основі властивостей процесу системного діагностування визначити клас моделей системного діагностування і модель МСД;
- 4) проаналізувати одержане рішення і у разі потреби, повторно виконати визначені кроки..

Отже, процес проектування ітеративний. Аналіз діагностовної системи на основі визначених моделей діагностування полягає у дослідженні поведінки елементів системи в разі взаємного контролю та переходу від одержаного синдрому системи (набору результатів контролю) до технічного стану системи. Поточні технічні стани системи шукають за результатами контролю, одержуваними для множин зв'язків пар елементів у процесі експлуатації системи. Відповідно до реалізації процесів системного діагностування слід задати модель діагностування для кожного зв'язку, структуру таких зв'язків та результати контролю.

Структуру системи задають у вигляді діагностичного графу (ДГ) $G = G(V, E)$, де V є множиною елементів системи $v_i, i = 1, \dots, n, v_i \in V$, а E - множиною спрямованих зв'язків (v_i, v_j) взаємодії між ними в процесі діагностування, $v_i, i = 1, \dots, n, v_i \in V, n = |V|, m = |E|$. Для кожного з зв'язків можна одержати результат контролю $a_{i,j}$.

Синдром системи А

$$A = \{ a_{ij} \mid \exists (v_i, v_j) \in E, v_i, v_j \in V \}$$

і підсиндроми

$$A_l \subseteq A, \quad l = 1, \dots, k$$

являють собою вихідні дані для знаходження технічного стану системи за визначеною МВЕ за допомогою алгоритму діагностування.

12.7 Інтегрована модель процесів діагностування

Найраціональніше використання апаратних та програмних засобів під час створення відмовостійких комп'ютерних мережних систем досягається за рахунок застосування методів і засобів системного діагностування. Традиційний підхід до системного діагностування ґрунтується на використанні синдрому системи, одержуваного як наслідок взаємного контролю елементів. Процес системного діагностування (СД) визначається як множина взаємодій пар елементів a, b з одержанням певних результатів контролю.

Для множини станів працездатності елементів системи (робочих станцій, маршрутизаторів, концентраторів тощо) a, b і стану взаємодії c через X, Y, C позначають стани працездатності цих елементів $X = \{ a, \bar{a} \} = \{ b, \bar{b} \} C = \{ c, \bar{c} \}$. Базова множина S_0 станів СД (БСД) є декартовий добуток $S_0 = X \times Y$. Сукупність підмножин Z , для яких виконується $Z \subseteq S_0, Z \neq S_0, Z \neq \emptyset$, називають початковою множиною станів (ПМС). Її можна позначити SU . Пара множин (Z_1, Z_2) , що задовольняє умовам $Z_1, Z_2 \in SU, Z_1 \neq Z_2, Z_1 \not\subseteq Z_2, Z_2 \not\subseteq Z_1$ і відповідає парі станам (c, \bar{c}) , називають моделлю діагностування (МД). Тоді процеси системного діагностування можна розглядати як пошук станів множин працездатності

елементів системи, тобто технічного стану системи за результатами контролю (синдрому).

Аналіз системи на основі моделей діагностування полягає у дослідженні поведінки елементів системи за умови взаємного контролю та переходу від одержаного синдрому системи до технічного стану. Поточні технічні стани системи шукають за результатами контролю, які одержують для множин зв'язків пар елементів у процесі експлуатації системи. Відповідно до реалізації процесів системного діагностування потрібно задати модель діагностування для кожного зв'язку, структуру таких зв'язків та результати контролю.

Структуру зв'язків задають у вигляді діагностичного графу (ДГ) $G = G(V, E)$, де V є множиною елементів системи $v_i, i = 1, \dots, n, v_i \in V$, а E - множиною спрямованих зв'язків (v_i, v_j) , за якими здійснюється взаємодія між елементами у процесі діагностування $v, v_j, I_j = 1, \dots, n, v, v_j \in V, n = |V|$ з множиною E спрямованих зв'язків $(v_i, v_j), m = |E|$. Для кожного з них можна одержати результат контролю a_{ij} . Синдром системи $A = \{ a_{ij} | \exists (v_i, v_j) \in E, v_i, v_j \in V \}$ і підсиндроми $A_l \subseteq A, l = 1, \dots, k$ являють собою вихідні дані для знаходження технічного стану системи (ТСС). Діагностичний граф G з означеною для нього МД, заданими для всіх зв'язків E , являє собою структуру взаємоконтролю (СВК) $SID, SID = (G, DM)$, для якої реалізується той чи інший алгоритм системного діагностування.

Помічений ДГ (ПДГ) $GA = (V, E, A)$ з означеною для нього моделлю МД являє деяку реалізацію СВК системи (PCBK). Множина допустимих PCBK для заданого ДГ є областю визначення СВК (МОСВК). Фіксовані стани працездатності елементів за певної МД, що не суперечать

синдрому, називають допустимим набором станів елементів (ДНС). Для ПДГ GA та відомої МД можна знайти множину ДНС $SC, SC = \{ SE_j \}, |SC| \geq 1$, де елемент $SE_i \in$ множиною V_g працездатних та множиною V_f непрацездатних елементів $v_i, i = 1, \dots, n, v_i \in V$, сумісних з симптомами синдрому A .

Інтегрована метамодель процесу системного діагностування на основі теорії взаємодіючих процесів Ч.Хоара має вигляд: $SDP = (PT \text{ op1}) (PA \text{ op2}) (PC \text{ op3}) (PE \text{ op4})$, де PT - процес тестування системи для одержання ПДГ (або частини ПДГ); PA - процес аналізу ДГ; PC - процес отримання моделі зв'язків елементів системи; PE - процес керування на основі результатів діагностування та подальшого корегування моделей взаємодії та структури СД; op_i - операції входження відповідного процесу до загального процесу СД.

Операції входження процесу до SDP відображають взаємовідносини між процесами, які зумовлені властивостями об'єкта діагностування. Зокрема, можливе послідовне виконання цих процесів, що характерно, наприклад для малопотужних засобів діагностування з централізованим керуванням.

Процес тестування системи PT описує особливості одержання результатів контролю a_{ij} з урахуванням особливостей взаємодії елементів системи під час СД. Так, для маршрутизаторів результатами можуть служити дані підтвердження таблиць маршрутизації суміжних маршрутизаторів. Процес аналізу діагностичного графа PA дає змогу зробити висновки про працездатність елементів системи за результатами контролю з урахуванням моделі взаємодії елементів. Він може виконуватись як процес моніторингу комп'ютерної мережі. Процес отримання моделі зв'язків PC дає змогу використовувати структури зв'язків

елементів системи та властивості їх взаємодії під час СД. Зокрема, можуть застосовуватись дані мережеметрії. Процес керування на основі результатів діагностування РЕ дає змогу коригувати структуру діагностування та моделі взаємодії на підставі попередніх результатів СД.

12.8 Структурно-апаратні методи забезпечення відмовостійкості

Для забезпечення відмовостійкості систем широко застосвують різномітні методи активного і пасивного резервування у поєднанні з засобами і методами діагностування. Схема активного резервування передбачає паралельну роботу кількох компонентів системи, зокрема як показано на рис. 12.21. За наявності однакового результату модулів M1, M2 вихід Out сприймається як достовірний за наявності сигналу Agree на виході компаратора C.

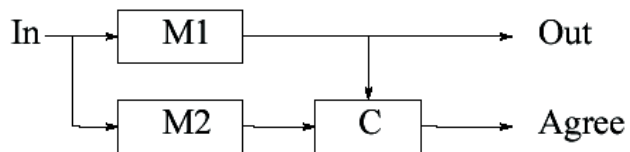


Рисунок 12.21 – Схема активного резервування

Практична реалізація такої схеми показано на рис. 12.22.

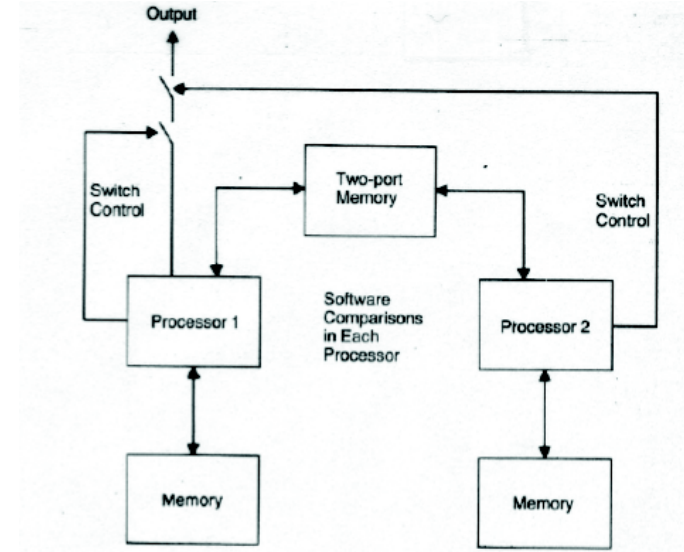


Рисунок 12.22 – Схема активного резервування на основі двох процесорів

Порівняння даних від процесорів може бути реалізовано з використанням спеціального програмного забезпечення.

Комбіновані структурні методи, які поєднують активне резервування і схеми голосування дозволяють

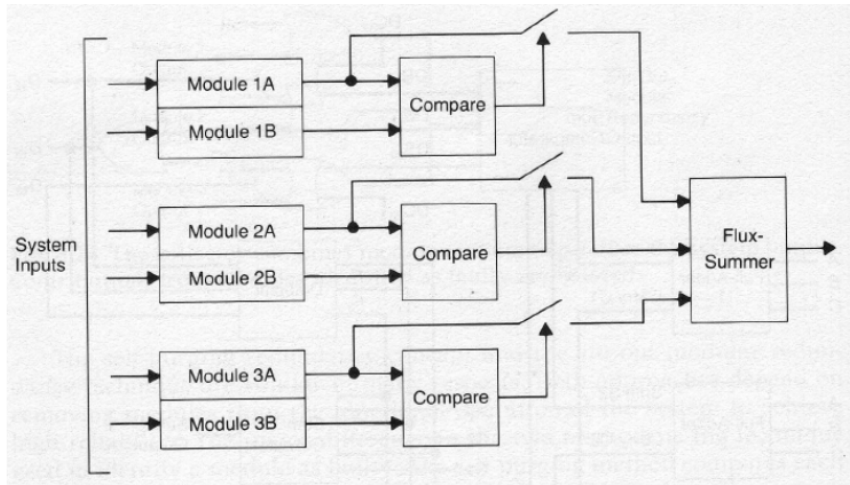


Рисунок 12.23 – Комбінована структура

12.9 Завадостійкі коди

12.9.1 Лінійні блокові коди

Кодер для блокових кодів перетворює інформаційні блоки завдовжки k символів у повідомлення з n символів, з яких $r=n-k$ є надлишковими (перевірними, контрольними). Перевірні символи забезпечують виявлення й виправлення помилок, спричинених завадами у каналі зв'язку. Кожний блок з n символів залежить лише від відповідного k -символьного інформаційного блоку і не залежить від інших блоків.

Кодер для згорткових кодів перетворює послідовність з k інформаційних символів у блок з n кодових символів, де $n > k$. Кодовий n -символьний блок залежить не тільки від k -символьного інформаційного блоку, наявного на вході у поточний момент часу, але і від попередніх m блоків повідомлення. Згорткові коди часто використовують для побітового передавання даних каналами.

Блоковий код завдовжки n символів, що складається з 2^k кодових слів, називають лінійним (k, n) -кодом за умови, що всі 2^k кодових слів утворюють k -вимірний підпростір векторного простору n -послідовностей двійкового поля $GF(2)$, тобто порозрядна сума за модулем 2 ($\text{mod } 2$) двох кодових слів також є кодовим словом даного коду. Систематичний код має інформаційну частину з k символів і надлишкову (перевірну) частину з $n-k$ символів постійної довжини. Найпростішим лінійним блоковим кодом є $(n-1, n)$ - код з контролем парності з використанням перетворення виду $E(m_1, \dots, m_k) = (m_1, \dots, m_k, m_{k+1})$, де m_{k+1} є символом парності або непарності інформаційних розрядів m_1, \dots, m_k .

12.9.2 Ітеративний код

Ітеративний код є найпростішим кодом з виправленням помилок. Повідомлення m розміщують у вигляді матриці з додаванням до кожного рядка і стовпця контрольних символів перевірки на парність. За умови виникнення однієї помилки перевірка на парність у відповідному рядку і стовпці не виконуватиметься, а координати помилки однозначно визначаються номерами стовпця і рядка, в яких не виконується перевірка на парність.

Приклад. Для кодування послідовності з $k=12$ інформаційних елементів застосовується ітеративний метод. Записати твірну матрицю еквівалентного лінійного блокового коду. Закодувати повідомлення (110111000110). Виправити помилку в прийнятій послідовності (1101110001110001111).

Для розв'язання задачі інформаційну послідовність розмістимо у вигляді матриці розмірністю 3×4 . Вихідна матриця кодування для інформаційної послідовності $(m_1, m_2, \dots, m_{12})$ має вигляд

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 \\ m_5 & m_6 & m_7 & m_8 \\ m_9 & m_{10} & m_{11} & m_{12} \end{pmatrix} \Rightarrow \begin{pmatrix} m_1 & m_2 & m_3 & m_4 & r_1 \\ m_5 & m_6 & m_7 & m_8 & r_2 \\ m_9 & m_{10} & m_{11} & m_{12} & r_3 \\ r_4 & r_5 & r_6 & r_7 & r_8 \end{pmatrix},$$

де r_1, r_2, \dots, r_8 – перевірні елементи.

Тоді послідовність, що зберігається або передається по каналу зв'язку, є: $u = (m_1, m_2, m_3, m_4, r_1, m_5, m_6, m_7, m_8, r_2, m_9, m_{10}, m_{11}, m_{12}, r_3, r_4, r_5, r_6, r_7, r_8)$.

З вихідної матриці отримуємо систему перевірних рівнянь, що визначає правила знаходження перевірних елементів:

$$\begin{cases} r_1 = m_1 + m_2 + m_3 + m_4, \\ r_2 = m_5 + m_6 + m_7 + m_8, \\ r_3 = m_9 + m_{10} + m_{11} + m_{12}, \\ r_4 = m_1 + m_5 + m_9, \\ r_5 = m_2 + m_6 + m_{10}, \\ r_6 = m_3 + m_7 + m_{11}, \\ r_7 = m_4 + m_8 + m_{12}, \\ r_8 = m_1 + m_2 + \dots + m_{12}. \end{cases}$$

Система перевірних рівнянь визначає правила знаходження перевірних символів залежно від інформаційних

Для системи перевірних рівнянь знаходимо твірну матрицю $G_{12 \times 20}$ еквівалентного лінійного блокового коду, яку можна легко привести до канонічного вигляду, розмістивши стовпці, що відповідають перевірним елементам у правій частині твірної матриці, а одиничну підматрицю, що визначає інформаційну частину кодового слова – у лівій:

$$G_{12 \times 20} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

$m_1, m_2, m_3, m_4, r_1, m_5, m_6, m_7, m_8, r_2, m_9, m_{10}, m_{11}, m_{12}, r_3, r_4, r_5, r_6, r_7, r_8$

Якщо на приймачній стороні прийнято послідовність (11011110001110001111), то можна виявити і виправити помилку. Декодуємо послідовність за ітеративним методом.

Запишемо прийняте слово у вигляді матриці (у даному випадку розмірністю 4×5) і виконаємо перевірку на парність за її кожним рядком і стовпчиком. У разі відсутності помилки контрольні елементи парності за рядками і стовпчиками матриці мають значення 0. Невиконання контролю парності у стовпчику і рядку однозначно визначить координати помилкового елемента – це дозволить його виправити.

$$\left| \begin{array}{ccccc|c} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right| \Rightarrow \left| \begin{array}{ccccc|c} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right|.$$

Помилковим є елемент $m[3,1]$, який виправляємо.

$$\left| \begin{array}{ccccc|c} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right| \Rightarrow \left| \begin{array}{ccccc|c} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right|.$$

Отже, виправлена комбінація така:

$$y=(11011110000110001111).$$

12.9.3 Виявлення помилок на основі кодових синдромів

Твірна матриця G розміром $k \times n$

$$G_{k \times n} = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & \dots & 0 & p_{11} & p_{12} & \dots & p_{1,n-k} \\ 0 & 1 & 0 & \dots & 0 & p_{21} & p_{22} & \dots & p_{2,n-k} \\ 0 & 0 & 1 & \dots & 0 & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & p_{k1} & p_{k2} & \dots & p_{k,n-k} \end{array} \right).$$

одинична підматриця I_{kk}
перевірні підматриця $P_{k \times (n-k)}$

лінійного блокового систематичного (k, n) - коду визначає кожне кодове слово як лінійну комбінацією рядків матриці G , а кожен лінійну комбінація рядків G - кодовим словом.

Для блока повідомлення $m=(m_1, m_2, \dots, m_k)$ кодове слово є послідовність $u=m \times G$, де для $i=1, 2, \dots, k$ $u_i = m_i$; для $i=k+1, \dots, n$ $u_i = m_1 p_{1i} + m_2 p_{2i} + \dots + m_k p_{ki}$; $i=1, 2, \dots, n-k$ - номер стовпчика перевірної частини $P_{k \times (n-k)}$ твірної матриці $G_{k \times n}$.

Перевірні матриця $H_{(n-k) \times n}$ розмірності $(n-k) \times n$ має властивість: для кодового слова u $u \times H^T = 0$, тобто ортогональна будь-якій кодівій послідовності даного коду і має структуру

$$H_{(n-k) \times n} = \left(\begin{array}{cccc|cccc} p_{11} & p_{12} & \dots & p_{1k} & 1 & 0 & 0 & \dots & 0 \\ p_{21} & p_{22} & \dots & p_{2k} & 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ p_{1,n-k} & p_{2,n-k} & \dots & p_{k,n-k} & 0 & 0 & 0 & \dots & 1 \end{array} \right),$$

$P_{(n-k) \times k}^T$
 $I_{(n-k) \times (n-k)}$

де $P_{(n-k) \times k}^T$ - транспонована перевірна підматриця твірної матриці $G_{k \times n}$; $I_{(n-k) \times (n-k)}$ - одинична підматриця.

За допомогою перевірної матриці можна визначити, чи є прийнята послідовність кодовим словом даного коду. Наприклад, для матриці кода (4, 7)

$$\underline{H}_{3 \times 7} = \left(\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array} \right).$$

і послідовності $y=(1011001)$ отримаємо

$$y \times H^T = (1011001) \times \left(\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array} \right)^T = (1 \ 0 \ 1) \neq 0.$$

Тобто $y=(1011001)$ не є кодовим словом даного коду.

Перевірні матриці використовують для перевірки наявності помилок $e=(e_1, e_2, \dots, e_n)$ у повідомленнях $u=(u_1, u_2, \dots, u_n)$, що надходять до каналу зв'язку. На виході каналу зв'язку з шумами отримують послідовність $y = u+e$, де u - передане кодове слово; e - вектор помилок у каналі. Вектор помилок $e=(e_1, e_2, \dots, e_n)$, є двійковою послідовністю завдовжки n з одиницями у тих позиціях, де виникли помилки.

Наприклад, вектор помилок $e=(0001000)$ означає однократну помилку у четвертому біті, $e=(1100000)$ - двократну помилку у першому і другому бітах. Для $u=(0001000)$, $e=(0001000)$, $y=(0000000)$.

Для перевірки наявності помилок у прийнятій послідовності y , декодер обчислює кодовий синдром S як $(n-k)$ - послідовність

$$S = (S_1, S_2, \dots, S_{n-k}) = y \times H^T,$$

де y - прийнята кодована послідовність;

H^T - транспонована перевірна матриця коду.

Послідовність y є кодовим словом при $S=(0 \ 0 \ \dots \ 0)$, і не є кодовим словом за умови $S \neq 0$.

Наприклад, для $y = (y_1, y_2, \dots, y_7)$ синдром

$$S = y \times H_{3 \times 7}^T = (y_1, y_2, \dots, y_7) \times \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}^T = ((y_1 + y_3 + y_4 + y_5), (y_2 + y_2 + y_3 + y_6), (y_2 + y_3 + y_4 + y_7)).$$

Декодер не виявляє помилки, для яких синдром

$$S = y \times H^T = 0.$$

Кодовий синдром залежить лише від вектора помилок і не залежить від переданого слова, оскільки

$$S = y \times H^T = (u + e) \times H^T = u \times H^T + e \times H^T = 0 + e \times H^T = e \times H^T,$$

За наявним вектором e можна відновити кодове слово: $u^* = y + e$.

Наприклад для $e_4 = (0001000)$

$$e_4 \times H^T = (0001000) \times \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}^T = (1 \ 0 \ 1) - помилка у 4-му біті$$

Помилку можна вилучити за $u^* = y + e$.

Існує однозначна відповідність між координатами поодиноких помилок і їх синдромами, тобто, знаючи синдром, можна визначити позицію коду, в якій виникла помилка.

12.10 Завадостійке кодування на основі поліноміальних кодів

12.10.1 Поліноміальне кодування інформації циклічних кодів

Лінійний блоковий (k,n) -код можна подати коефіцієнтами полінома

$$u(x) = u_0 + u_1 x + u_2 x^2 + \dots + u_{n-1} x^{n-1},$$

Поліноміальним кодом називають множину всіх многочленів степені не більше $n-1$, що мають спільний множник – деякий фіксований многочлен $g(x)$ степеня $r=n-k$. Цей многочлен $g(x)$ називають твірним многочленом коду. Поліноміальний код з твірним многочленом $g(x)$ кодує повідомлення $m(x)$ поліномом

$$u(x) = m(x) \cdot g(x) = u_0 + u_1 x + u_2 x^2 + \dots + u_{n-1} x^{n-1},$$

або кодовим словом з коефіцієнтів цього многочлена $u = (u_0, u_1, \dots, u_{n-1})$.

Матриця $G_{k \times n}$ поліноміального коду з твірним многочленом $g(x)$ степеня $r=n-k$ має вигляд

$$G_{k \times n} = \begin{pmatrix} g_0 & g_1 & g_2 & \dots & g_r & \dots & 0 \\ 0 & g_0 & g_1 & \dots & g_{r-1} & g_r & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & g_r & \dots & \dots \end{pmatrix},$$

Ненульові елементи в i -му рядку є послідовністю коефіцієнтів твірного многочлена, розташованих з j -го по $(j+r)$ -й стовпець.

Вектор помилок $e = e_0, \dots, e_{n-1}$ є не визначеним у тому і лише у тому випадку, коли його многочлен $e(x) = e_0 + e_1 x + \dots + e_{n-1} x^{n-1}$ ділиться на твірний поліном коду $g(x)$ без остачі. Тобто прийнята послідовність $c(x) = m(x)g(x) + e(x)$ ділиться на $g(x)$ без остачі тоді і тільки тоді, коли $e(x)$ ділиться на $g(x)$ без остачі.

Тому помилка, многочлен якої не ділиться на $g(x)$ можна визначити, а помилка, многочлен якої ділиться на $g(x)$, не можна визначити. Як наслідок, виявлення помилки поліноміальним кодом з твірним поліномом $g(x)$ може бути здійснене за допомогою ділення многочленів. Якщо залишок від ділення многочлена прийнятої послідовності на твірний поліном $g(x)$ ненульовий, то при передачі відбулося спотворення даних.

До простих і поширених поліноміальних кодів відносять циклічні коди. Лінійний блоковий (k, n) - код називають циклічним, якщо в результаті циклічного зсуву кодового слова $u = (u_0, u_1, \dots, u_{n-1})$ отримують нове кодове слово $v = (u_{n-1}, u_0, u_1, \dots, u_{n-2})$, яке також належить даному коду.

Особливості застосування циклічних поліноміальних кодів визначають такі їх властивості: кожний ненульовий поліном повинен мати степінь в межах від $(n-k)$ до $n-1$; існує лише один кодовий поліном $g(x) = 1 + g_1 x + g_2 x^2 + \dots + g_{n-k-1} x^{n-k-1} + x^{n-k}$ степені $(n-k)$, що є дільником кожного кодового полінома $u(x) = m(x) \cdot g(x)$ і який називають твірним поліномом коду.

Кодове слово циклічного коду складається з перевірної частини з $(n-k)$ перевірних символів і інформаційної частини m завдовжки k символів. Перевірні символи є коефіцієнтами полінома $p(x)$ – остачі від ділення кодового слова $u(x) = m(x) \cdot x^{n-k}$ на твірний поліном $g(x)$.

12.10.2 Схемна реалізація

Для твірного полінома $1 + X^3 + X^4$ з кодом 1101 схемну реалізацію кодування на основі регістрів зсуву подано на рис. 12.24. Схеми XOR відповідають одиничному значенню коефіцієнта члена полінома, а затримки D дозволяють визначити його положення. Дані інформаційного повідомлення, яке кодується, надходять на вхід Data, а результат кодування виводять на вихід Encoded Output/

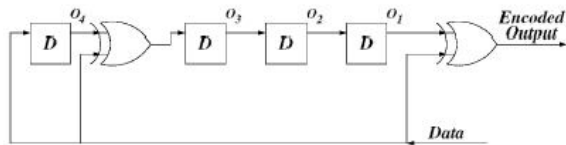


Рисунок 12.24 - Схемна реалізація кодування на основі твірної полінома

Наприклад, для розряду 5 множення поліномів при формуванні закодованого повідомлення відповідає результату обчислення за модулем 2 суми проміжних результатів, поданих на рис. 12.25.

$$\begin{array}{r}
 10001100101 \\
 \times \quad 11001 \\
 \hline
 10001100101 \\
 00000000000 \\
 00000000000 \\
 10001100101 \\
 10001100101 \\
 \hline
 110000100011101
 \end{array}$$

Рисунок 12.25 – Обчислення поліному закодованого повідомлення на основі твірної поліному 1101

Для послідовності інформаційного повідомлення 10001100101 отримаємо повідомлення 110000100011101 (рис. 12.26). Тут значення i_3 відповідає входу члена Q_3 полінома.

shift clock	input data	Q_4	i_3	Q_3, Q_2, Q_1	encoded output
1	1	0	1	000	1
2	0	1	1	100	0
3	1	0	1	110	1
4	0	1	1	111	1
5	0	0	0	111	1
6	1	0	1	011	0
7	1	1	0	101	0
8	0	1	1	010	0
9	0	0	0	101	1
10	0	0	0	010	0
11	1	0	1	001	0
12	0	1	1	100	0
13	0	0	0	110	0
14	0	0	0	011	1
15	0	0	0	001	1

Рисунок 12.26 – Таблиця обчислення повідомлення

Декодування виконують як ділення на твірний поліном. На рис.12.27 подано результат декодування для повідомлення без помилки і з помилкою.

На рис.12.28 подано схему декодування повідомлення. За співвідношеннями, а на рис. 12.29 таблицю декодування.

$$\begin{aligned}
 E(X) &= D(X) \cdot G(X) = D(X)(1 + X^3 + X^4) \\
 &= D(X) + D(X)(X^3 + X^4)
 \end{aligned}$$

$$D(X) = E(X) - D(X)(X^3 + X^4) = E(X) + D(X)(X^3 + X^4)$$

13 КОМП'ЮТЕРИ ПАРАЛЕЛЬНОЇ ДІЇ

13.1 Таксономія Флінна

Таксономія (класифікація) Флінна (M. Flynn) — це загальна класифікація архітектур EOM за ознаками наявності паралелізму в потоках команд (інструкцій) і даних. Розмаїтість архітектур EOM в цій таксономії Флінна зводиться до чотирьох основних класів: SISD, SIMD, MISD, MIMD.

Архітектура SISD (single instruction — single data) подає одиночний потік команд і даних. Паралелізм відсутній. До цієї категорії відносяться послідовні архітектури, у тому числі й фон-нейманівського типу (рис. 13.1). Через PU позначено процесорний елемент (одно або багатоядерний CPU).

```

110000100011101:11001 = 10001100101
11001
-----
10100
11001
-----
11010
11001
-----
11111
11001
-----
11001
11001
-----
00000

110000100011101:11001 = 10001100110
11001
-----
10100
11001
-----
11011
11001
-----
10111
11001
-----
11100
11001
-----
01011
    
```

Рисунок 12.27 – Декодування повідомлення

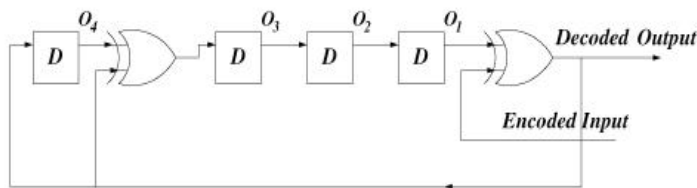


Рисунок 12.28 – Схема декодування повідомлення

shift clock	encoded input	i ₄	O ₄	i ₃	O ₃ , O ₂ , O ₁	decoder output
1	1	1	0	1	000	1
2	0	0	1	1	100	0
3	1	1	0	1	110	1
4	1	0	1	1	111	0
5	1	0	0	0	111	0
6	0	1	0	1	011	1
7	0	1	1	0	101	1
8	0	0	1	1	010	0
9	1	0	0	0	101	0
10	0	0	0	0	010	0
11	0	1	0	1	001	1
12	0	0	1	1	100	0
13	0	0	0	0	110	0
14	1	0	0	0	011	0
15	1	0	0	0	001	0

Рисунок 12.29 – Таблиця декодування повідомлення

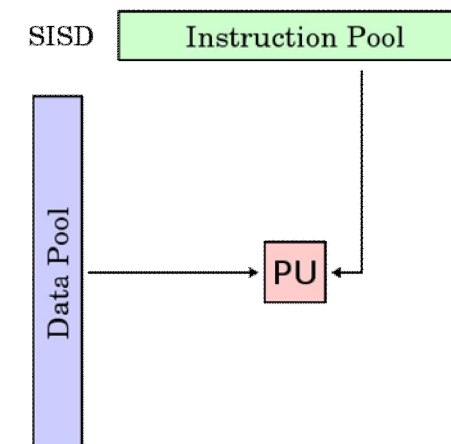


Рисунок 13.1 - Архітектура SISD

Архітектура SIMD (single instruction — multiple data) подає одиночний потік команд і декілька потоків даних (рис. 13.2). Паралелізм в таких архітектурах полягає в можливості одночасного виконання однієї й тої ж операції над декількома елементами даних. Досягається це централізованою видачею команд декільком обчислювальним пристроям спільним для них пристроєм управління.

Найпоширенішими представниками архітектур типу SIMD є так звані векторні EOM, оптимізовані для паралельного виконання однотипних операцій над елементами векторів і матриць.

Спеціалізовані векторні процесори іноді вбудовуються в комп'ютери загального призначення. Зокрема, в багатьох сучасних мікропроцесорах вбудовані обмежені можливості векторних обчислень для обробки даних мультимедіа.

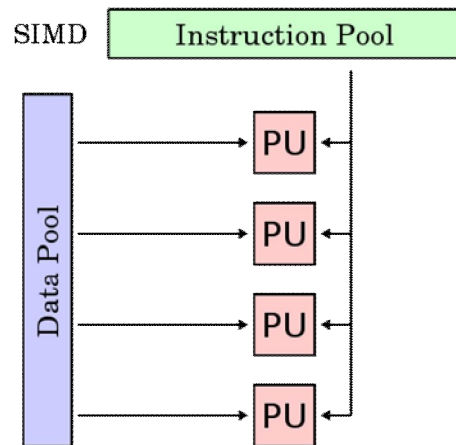


Рисунок 13.2 - Архітектура SIMD

Архітектура MISD (multiple instruction — single data) передбачає обробку кількох потоків команд для одиночного потоку даних (рис. 13.3).

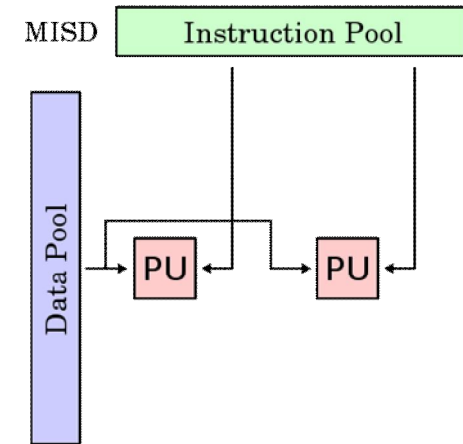


Рисунок 13.3 - Архітектура MISD

Не існує єдиної думки щодо того, які архітектури відносити до даного класу. Деякі включають у нього конвеєрні обчислювачі, в яких цілісна операція розбивається на послідовність простіших етапів з суміщенням різних етапів для різних порцій даних у часі. Таким чином, кожна порція даних від моменту завантаження з пам'яті до запису результату обробки проходить через кілька етапів обробки (стадій конвеєра). Подібні принципи покладені в основу систолічних архітектур, які також відносяться до цього класу.

Обчислення в MISD є одним з видів паралельної обчислювальної архітектури, де багато функціональних блоків виконують різні операції на тих же даних.

Конвеєрні архітектури належать до цього типу, хоча дані відрізняються після обробки на кожному етапі.. Відмовостійкі комп'ютери, що виконують одну і ту ж інструкції налішково для того, щоб виявити і маскувати помилки (реплікація завдань), можна також віднести до цього типу MISD.

Архітектури MIMD і SIMD комп'ютерів є придатними для методів паралельної обробки даних. Зокрема, вони дозволяють краще масштабувати і використовувати обчислювальні ресурси, ніж е робить MISD. Тим не менше, одним з яскравих прикладів комп'ютерів MISD в обчисленнях є управління польотом Space Shuttle.

Систолічні масиви також є прикладом MISD архітектури. У типовому систолічному масиві паралельні вхідні потоки даних проходять через мережу жорстко пов'язаних процесорів, що нагадують людський мозок.

Систолічні масиви часто жорстко пов'язані для конкретної операції, зокрема множення і накопичення для масової-паралельної інтеграції, згортки, кореляцій, матричного множення або сортування даних.

Наприклад, для обчислення полінома за правилом Горнера

$$y = (\dots (((a_n * x + a_{n-1}) * x + a_{n-2}) * x + a_{n-3}) * x + \dots + a_1) * x + a_0$$

лінійний систолічний масив, в якому процесори розташовані парами, кожний перемножає вхідні дані на x і передає результат вправо, наступний додає a_j і передає вправо.

Систолічний масив, як правило, складається з великої монолітної мережі примітивних обчислювальних вузлів, які можуть бути жорстко або програмно налаштовані для конкретного додатка. Вузли, як правило, фіксовані і ідентичні, а з'єднання програмується.

Більш загальні хвильові процесори (wavefront processors) хвильового фронту, навпаки, використовують складне і індивідуальне програмування вузлів, які можуть бути або не можуть бути монолітними, залежно від розміру масиву і конструктивних параметрів. Оскільки поширення хвиль, як даних через систолічного масиву нагадує пульс-судинної системи людини, ім'я систолічний був заповичений з медичної термінології.

Основною перевагою систолічного масивів, що всі дані операндів і часткові результати містяться в (проходять) через масив процесорів. Немає необхідності для доступу до зовнішніх шин, основної пам'яті або внутрішніх кешів протягом кожної операції, як у звичайних послідовних машинах.

Архітектури MIMD (multiple instructions — multiple data) передбачає одночасно кілька потоків команд і даних (рис. 13.4). Найпоширеніший на сьогоднішній день клас паралельних архітектур, в яких кожний процесор здатний працювати незалежно від інших над своїм завданням.

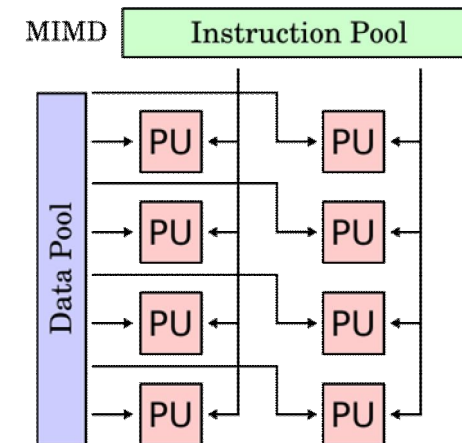


Рисунок 13.4 - Архітектура MIMD

Обчислення у MIMD є засобом для досягнення паралелізму . Машини з MIMD мають ряд процесорів , які функціонують асинхронно і незалежно один від одного. У будь-який час різні процесори можуть виконувати різні інструкції на різних частинах даних.

MIMD архітектури можуть використовуватись у ряді прикладних областей, таких як системи автоматизованого проектування та автоматизованого виробництва, моделювання та комунікаційні комутатори. Машини MIMD можуть використовувати розділену або розподілену пам'ять. Розділена пам'ять може мати шинну або ієрархічну організацію. Розподілена пам'ять може бути гіперкубом або мати меш-схему (mesh interconnection schemes) з'єднань.

Прикладом системи MIMD є Intel Xeon Phi з Larrabee мікроархітектурою. Процесори мають кілька ядер обробки (до 61 від 2015 року), які можуть виконувати різні інструкції за різними даними.

Останнім часом значна кількість створених паралельних комп'ютерів є системи з архітектурою MIMD. Зазвичай процесори всі пов'язані з "глобально доступною пам'яттю" через програмне або апаратне забезпечення.

З погляду програміста, цю модель пам'яті краще зрозуміти, ніж розподілену модель пам'яті. Недоліками є: масштабованість за межами тридцяти двох процесорів, а модель спільної пам'яті менш гнучка, ніж розподіленої модель пам'яті. Прикладами поширених систем пам'яті є UMA (Uniform Memory Access), COMA (Cache Only Memory Access) і NUMA (Non-Uniform Memory Access).

Машини MIMD із загальною пам'яттю процесори використовують спільну пам'ять. У простіших випадках всі процесори приєднані до шини, яка з'єднує їх до пам'яті.

В архітектурі SPMD (Single program, multiple data streams) з однією програмою (але у кількох точках) і багатьма потоками даних. SPMD є паралельною моделлю виконання програм і передбачає кратне кооперування процесів при виконанні процесів. Модель SPMD запропонована Frederica Darema у проекті RP3.

В архітектурі MPMD (Multiple programs, multiple data streams) передбачають одночасне виконання кількох програм (не менше двох) і для кількох потоків даних. Зазвичай одна з програм стає головною ("the explicit host/node programming model") або менеджером (the "Manager/Worker" strategy), які запускають програму надання даних іншим вершинам другої програми. Результати виконання вершин повертають менеджеру. Прикладами систем є ігровий консоль Sony PlayStation 3 з архітектурою процесора SPU/PPU. PlayStation 3 є гральна консоль сьомого покоління, третя в сімействі ігрових систем «PlayStation». Розроблена компанією Sony Computer Entertainment. Консоль є багатофункціональною: за її допомогою можна не тільки грати в ігри, а й слухати музику, дивитися фільми (на носіях Blu-ray Disc та DVD) і виходити в Інтернет.

Процесором PS3 є багатоядерний процесор Cell Broadband Engine, сумісно розроблений компаніями IBM, Sony та Toshiba Corporation. Cell складається з одного процесорного елемента на основі POWER-архітектури (PPE — POWER Processor Element) та восьми синергічних процесорних елементів (SPE — Synergistic Processor Element).

Також у приставці використовується графічний процесор RSX або «Синтезатор реальності», створений компаніями NVidia та SCEI. Відеочіп працює на частоті 550 МГц та має 256 МБ пам'яті, що дозволяє виводити на екран два якісних HDTV-потоки одночасно. Графічний процесор є гібридом на основі чіпсетів G70 та G80. Теоретичні потужності процесорів

PlayStation 3: центрального процесора - 218 G FLOPS, графічного процесора - 1,8 TFLOPS. [9]

Оскільки ігрова консоль Sony PS3 побудована на процесорі Cell та має можливість запуску операційних систем сімейства GNU/Linux, стало можливо використовувати приставку не за призначенням, а наявність порта GigabitEthernet дозволило об'єднувати декілька пристроїв у кластери, та будувати суперкомп'ютери. У вільному доступі є документ A Rough Guide to Scientific Computing On the PlayStation 3, який детально описує побудову суперкомп'ютера на базі PS3, написаний у Innovative Computing Laboratory, University of Tennessee Knoxville

Кластер із 200 приставок Sony PS3 використовувався для демонстрації вразливості сертифікатів, підписаних за допомогою алгоритму md5 на конференції Chaos Communication Congress наприкінці 2008 року

На приставці встановлена повноцінна операційна система, до якої через онлайн-сервіс можна завантажувати патчі та оновлення. До системи включений повноцінний веб-браузер.

Графічний інтерфейс PS3 під назвою XrossMediaBar (XMB) був розроблений на основі інтерфейсу портативної приставки PSP і є центром керування можливостями консолі. Програма дозволяє переглядати фотографії, програвати музику і фільми з жорсткого диску та Blu-Ray, а також запускати ігри (як з дисків. Зокрема 15.6 ГБ диск вміщує близько 9 годин відео високої чіткості, а 50 ГБ диск вміщує близько 23 годин відео стандартної чіткості з роздільною здатністю 1920x1080 точок. Розробники планують незабаром підвищити кількість реєструючих шарів до 8, тим самим збільшивши загальну ємність до 200 GB.

13.2 Класифікація апаратних засобів PIC

Єдиної класифікації PIC не існує. Визначальним для обробки даних в PIC є доступ процесора до пам'яті та кількість процесорів. У зв'язку з цим мультипроцесорні розподілені системи поділяють на (рис. 13.5):

- 1) мультипроцесорні PIC із загальним для процесорів модулями пам'яті;
- 2) мультикомп'ютерні PIC із локальною пам'яттю для кожного процесора.

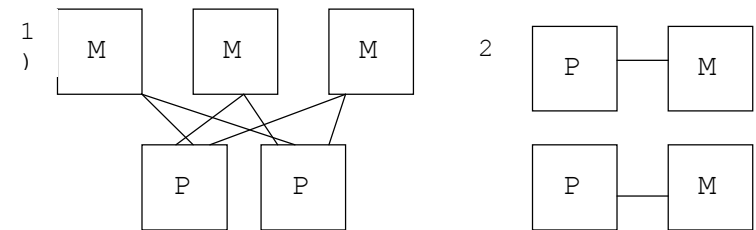


Рисунок 13.5 – Архітектура мультипроцесорних і мультикомп'ютерних PIC

За мережами зв'язку між процесорами і пам'яттю PIC поділяють на: системи з шинною архітектурою та з комутованою архітектурою.

Системи із шинною архітектурою (рис. 13.6) мають загальний доступ до єдиної шини, через яку здійснюється обмін даними.

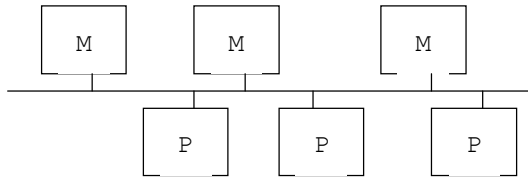


Рисунок 13.6 – Система із шинною архітектурою

У комутованих мультипроцесорних системах (рис. 13.7) використовуються комутатори чи комутаційне середовище, яке дозволяє підключатися процесорам до певних модулів пам'яті.

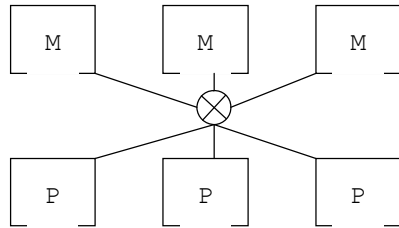


Рисунок 13.7 – Мультипроцесорна система із комутатором

Мультикомп'ютерні системи поділяються на мультикомп'ютерні ПІС із шинною організацією (архітектурою) (рис. 13.8)

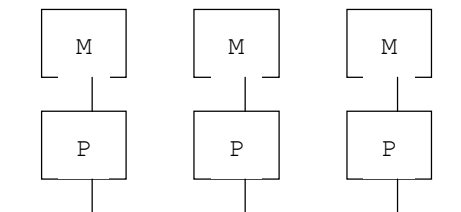


Рисунок 13.8 – Мультикомп'ютерна системи із шинною організацією

і з комутованою архітектурою (рис. 13.9).

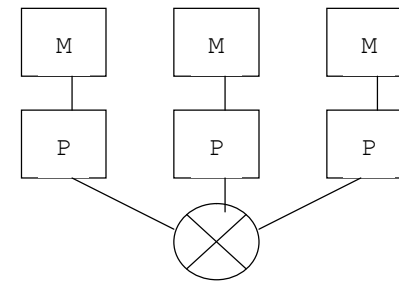


Рисунок 13.9 – Мультикомп'ютерна системи із комутатором

Для одночасного доступу до пам'яті використовують кілька підходів у мультипроцесорних системах:

1. застосування кешів (рис. 13.10)

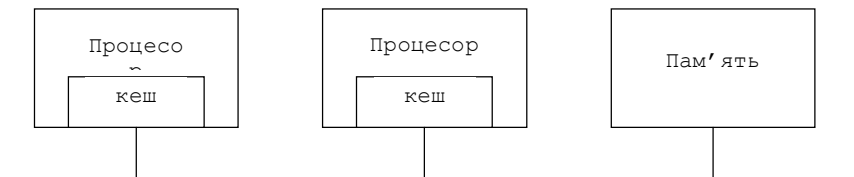


Рисунок 13.10 – Мультипроцесорна система із шиною і кешем

Під час роботи системи здійснюється реплікація даних у вигляді сторінок (4 або 8 Кб). Недоліками застосування кешів є:

- дублювання даних у пам'яті;
 - необхідність узгодженості даних для різних процесорів;
 - обмежена масштабованість.
2. застосування комутованих матриць (рис. 13.11)

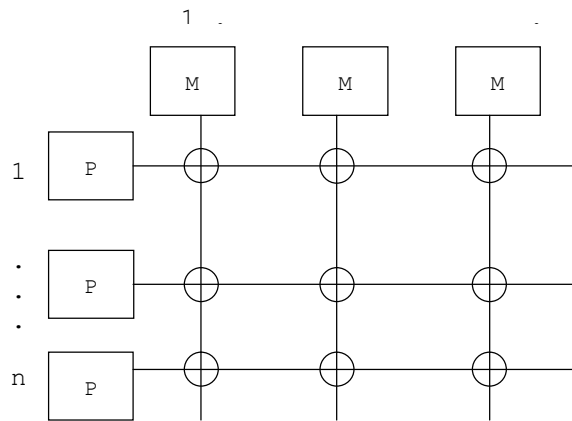


Рисунок 13.11 – Мультипроцесорна система з комутованою матрицею

Це потребує при кількості модулів або процесорів n^2 комутаторів.

Шинна архітектура таких систем зумовлює переповнення трафіку за рахунок взаємоблокувань при реалізації великої кількості звертань до шини.

У гомогенних мультикомп'ютерних системах застосовують регулярні структурні з'єднання.

1) Найбільш поширені – квадратні решітки (рис. 13.12)

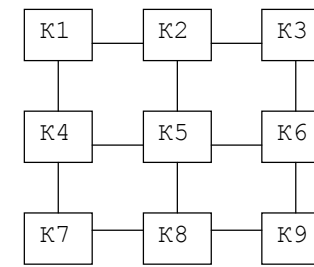


Рисунок 13.12 – Мультипроцесорна система з комутованою матрицею

2) Гіперкуб, який являє собою куб розміром n . В вузлах гіперкуба розміщують процесори.

У мультикомп'ютерних системах зв'язок між процесорами здійснюється шляхом маршрутизації повідомлень за допомогою спеціальних швидкісних мереж комутації. Розрізняють:

- процесори з масовим паралелізмом (із кількістю процесорів до 1000)
- кластер з n робочих станцій.

У гетерогенних мультикомп'ютерних системах зв'язок між комп'ютерами не утворює регулярної структури. В них можуть застосовуватись швидкісні мережі телекомунікації або інші мережні технології.

13.3 Моделі узгодженості паралельних комп'ютерів

Паралельні комп'ютери мають мати модель узгодженості (також відому як модель пам'яті). Модель узгодженості описує правила проведення різноманітних операцій з пам'яттю, та що ми отримуємо в результаті цих операцій.

Однією з перших моделей узгодженості була модель послідовної щільності Леслі Лампорта. Послідовна узгодженість — це властивість

паралельної програми давати такий самий результат що і її послідовний аналог. Конкретніше, програма послідовно узгоджена, якщо "... результат будь-якого запуску такий самий, як був би якщо операції на кожному окремому процесорі виконувались так, ніби вони виконуються в певній послідовності заданій програмою.

Транзакційна пам'ять це типовий приклад моделі узгодженості. Транзакційна пам'ять взяла у теорії баз даних принцип атомарної транзакції та застосувала його при доступі до пам'яті.

Математично, ці моделі можуть представлятись кількома способами. Починаючи з кінця 1970-х, були розроблені числення процесів, такі як Числення Систем, що Спілкуються та Послідовні Процеси, що Спілкуюються, щоб уможливити алгебраїчне обґрунтування систем складених з взаємодіючих компонентів. Новіші доповнення до сім'ї числень процесів, такі як π -числення, додали можливість обґрунтування динамічних топологій. Такі логіки як TLA+, та математичні моделі такі як траси та діаграми подій актора, також були розвинені щоб описати поведінку конкурентних систем.

13.3.1 Логічні годинники за алгоритмом Лампорта

Взаємодія процесів у розподіленій системі можлива лише за умови їх синхронізації в часі при пересиланні, отриманні і передачі інформації.

Глобальна синхронізація годинників машин дозволяє проводити упорядкування ресурсів системи (версій файлів, повідомлень тощо). Годинники комп'ютерів не можуть забезпечити глобальну синхронізацію, оскільки мають похибки відліку, пов'язані із застосуванням кварцевих годинників. З часом їх використання призводить до розсинхронізації (clock

skew). Тому час, який асоціюється з файлом, об'єктом повідомлення призводить до неправильної роботи програм, які з ними працюють.

В сучасних розподілених системах застосовують універсальний узгоджений час UTC (Universal Coordinated Time), який по суті замінив час за Грінвичем (Greenwich mean time). Відлік UTC ґрунтується на глобальному часі за атомним годинником TAI (International Atomic Time), який визначають як середній час тиків годинника на цезії -133 після 1.01.1959 р., поділений на число 9192631770.

В багатьох випадках при взаємодії процесів застосовують узгодження, не обов'язково точне, годинників за часом, який називають логічним часом (logical clock). Для синхронізації логічних годинників Лампорт визначив відношення $a \rightarrow b$ ("a відбувається раніше за b"). Відношення виконується, коли

1) a і b є подіями одного процесу, подія a відбувається раніше за подію b , і відношення $a \rightarrow b$ є істинним;

2) якщо a – подія відправки повідомлення одним процесом, а подія b – отримання його іншим, то відношення $a \rightarrow b$ є істинним. Повідомлення не можна отримати раніше, ніж воно було відправлено.

З транзитивності відношення випливає, що з відношень $a \rightarrow b$, $b \rightarrow c$, випливає відношення $a \rightarrow c$.

Дві події x , y паралельні (concurrent), якщо вони відбуваються у різних процесорах і відношення $x \rightarrow y$, $y \rightarrow x$ не є істинними.

Для виміру часу події x застосовують відліки часу $C(x)$. Для істинного $a \rightarrow b$ виконується $C(a) < C(b)$, а для різних процесів при відправці повідомлень (подія a) і отриманні повідомлення (подія b) треба встановити $C(a) < C(b)$. Час тільки збільшується, тобто корекція робиться лише в сторону збільшення.

За алгоритмом Лампорта кожне повідомлення містить час відправки за годинником відправника. При отриманні повідомлення годинник відправника є дійсним, інакше відбувається корегування його годинника на +1. Для того, щоб для різних подій $a, b, C(a) \neq C(b)$ застосовують додавання десяткового розряду, наприклад 40.1 і 40.2.

Для підтримки причинно-наслідкових зв'язків застосовують векторні відліки (відмітки) часу (vector time stamps), за якими кожному процесу P_i приписують вектор V_i з властивостями:

$V_i[i]$ – кількість подій, які відбулися з P_i до даного моменту часу;

2) якщо $V_i[j] = K$, то процес P_i знає, що з процесом P_j відбулося K подій.

13.3.2 Глобальний стан.

Глобальний стан (global state) розподіленої системи включає в себе локальний стан кожного процесу разом з повідомленнями, що перебувають у русі (тобто відправлені, але не доставлені). Локальний стан процесу залежить від його аналізу і використання. В якості глобального стану часто застосовують розподілений знімок (distributed snapshot), властивістю якого є відображення несуперечливого глобального стану.

Знімок містить записи відправки і отримання повідомлення, або лише відправки. Алгоритм побудови знімка може ініціюватись кількома процесами і тому можуть створюватись кілька знімків станів. Наприклад, процес P_i починає з запису власного локального стану і відправляє маркер кожним із своїх вихідних каналів. Після отримання маркера, процеси фіксують свої локальні стани і направляють процесу, що ініціював створення знімка.

13.4 Архітектури паралельних комп'ютерів

13.4.1 Класифікації паралельних комп'ютерів

Відповідно до класифікації Р.Хокні можна виділити найбільш поширені сьогодні класи високошвидкісних обчислювальних систем. Для зручності, ми їх класифікуємо згідно загальноприйнятих сьогодні у світовій комп'ютерній спільноті термінології.

Сьогодні для класифікації суперкомп'ютерних систем прийнято використовувати такі терміни як:

ccNUMA (cache coherent Non Uniform Memmory Access),

SMP (Symmetric Multinprocessing),

MPP (Massively Parallel Processing),

Cluster Systems, PVP (Parallel Vector Processing).

Ці загальноживані класи систем мають особливості, властиві різним класам систем за Р.Хокні. Причому всі системи, які потрапляють до списку найпотужніших суперкомп'ютерних систем світу мають властивості, як мережних систем, так і систем з перемикачем за класифікацією Р.Хокні.

Хоча зустрічаються й винятки. Наприклад, є кластерні системи, побудовані на однопроцесорних вузлах, отже вони є чисто мережними системами.

Для класифікації сучасних суперкомп'ютерів та їх співвідношення з класифікацією Р.Хокні можна ввести V-класифікацію (Рис. 13.13), та відобразимо на неї ряд найбільш поширених комп'ютерних обчислювальних систем, а саме (IBM SP2, NEC SX-6, HP Superdome, Cray X1, NOW Cluster)

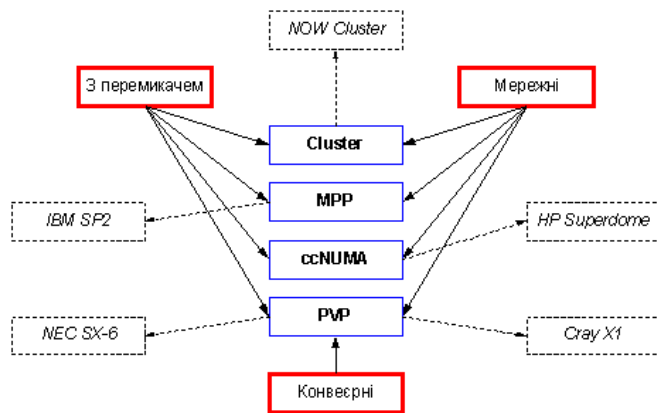


Рисунок 13.13 - V-класифікація.

На рисунку товстими лініями виділені паралелепіеди що класифікують системи за Р.Хокні, тонкими - сучасну поширену термінологію класифікації, а пунктиром - найбільш поширені сучасні обчислювальні системи.

До систем зі спільною пам'яттю відносяться паралельні системи SMP (Symmetric Multiprocessing) та NUMA (Non Uniform Memory Access). Перші мають кращу масштабованість задач на багатопроцесорних конфігураціях, другі краще масштабуються апаратно (тобто дозволяють будувати ефективні конфігурації з більшою кількістю процесорів).

Обчислювальні системи зі спільною пам'яттю найзручніші у програмуванні серед усіх паралельних систем. При програмуванні такої системи практично можна брати до уваги необхідність розподілення масивів даних, оскільки всі процесори мають доступ до спільних ресурсів пам'яті, портів вводу-виводу, тощо. У зв'язку з цим програми для таких системи легко проектуються та відлагоджуються. В той же час,

обчислювальні системи зі спільною пам'яттю мають дуже високу вартість і погано масштабуються за кількістю процесорів.

Ядро SMP системи складається з невеликої кількості (до 32) обчислювальних процесорів, які мають однорідний доступ до пам'яті (всі процесори на однаковій швидкості) та шин вводу-виводу даних. Доступ до пам'яті зазвичай здійснюється через спільну шину або матричний комутатор. Оскільки шинна конфігурація не ефективна для великої кількості активних елементів, то вона, зазвичай, використовується в системах з кількістю процесорів до 4-х. Сучасні процесори для підвищення ефективності роботи з локальною пам'яттю використовують кеш.

У багатопроцесорних системах зі спільною пам'яттю кеш пам'яті потрібно синхронізувати і це виливається в окрему проблему, яку потрібно вирішувати на апаратному рівні.

В SMP системах апаратно підтримується когерентність кеш-пам'яті. Слід зазначити, що такі системи, завдяки однорідному високошвидкісному доступу всіх процесорів до пам'яті та ресурсів вводу-виводу, найкраще підходять для задач типу OLTP (online transaction processing - обробка транзакцій в реальному часі) для яких багато користувачів мають доступ до однієї бази даних. Завдяки однорідності та, як результат, спрощених особливостей паралельного програмування, для SMP систем існують доволі ефективні засоби автоматичного розпаралелювання послідовних програм.

13.4.2 Масово-паралельна архітектура

Серед сучасних поширених обчислювальних систем з розподіленою пам'яттю виділяють MPP (Massively Parallel Processing - масивно паралельні системи) та кластерні системи. Принципової різниці між цими

класами немає, їх розділяє хіба що підхід до побудови обчислювальної системи.

MPP - це спеціально спроектовані "з нуля" системи з використанням високошвидкісних процесорів, спеціалізованих комунікаційних інтерфейсів та операційних систем.

В той же час кластерна обчислювальна система - це набір стандартних широко розповсюджених програмно-апаратних компонент, об'єднаних для вирішення спільних задач. У кластері в якості процесорних елементів використовуються стандартні однопроцесорні або SMP комп'ютери, а в якості міжвузлових комунікацій - стандартні мережні інтерфейси. Зрозуміло, що межа між обома типами досить умовна і тому кластерну обчислювальну систему часом називають дешевим варіантом MPP комп'ютера.

При побудові MPP систем виробники суперкомп'ютерів використовують надсучасні апаратні та програмні технології, а також різноманітні програмні засоби для збільшення швидкодії систем та зручності їх використання.

Так, наприклад, у систем ТЗЕ компанії Cray локальна пам'ять кожного процесорного елемента (містить один процесор, локальну пам'ять та комунікаційний інтерфейс) є частиною фізично розподіленої, але логічно розділеної пам'яті всього комп'ютера. Будь-який вузол (процесорний елемент) через свій комунікаційний інтерфейс може напряму звернутися до пам'яті будь-якого іншого вузла, не перериваючи його роботи. В тій же системі використовується апаратна підтримка бар'єрної синхронізації, що дозволяє уникнути значних накладних витрат на її програмну реалізацію.

Масово-паралельна архітектура (Massively Parallel Processor, MPP), також масивно-паралельна архітектура, масивно-паралельний процесор –

архітектура паралельної ЕОМ з розподіленими блоками обчислень, зокрема розподіленою пам'яттю, тобто з наявною в кожного з процесорів власної пам'яті. Тому ці архітектури також називають архітектурами з розподіленою пам'яттю (хоча, якщо бути точнішим, клас архітектур з розподіленою пам'яттю є підкласом масово-паралельних архітектур).

За рахунок цього система позбувається централізованої шини обміну даними між процесорами та проблем з масштабуванням, властивих симетрично-паралельним (SMP) системам з загальною для всіх процесорів пам'яттю.

Дані й команди можуть бути розподілені по локальних областях пам'яті процесорів, і для звертання одного процесора до пам'яті іншого застосовується звичайно спеціальний механізм обміну повідомленнями (message passing). Швидкість звертань до локальної й нелокальної пам'яті в таких системах істотно різниться, і конфігурації (топологія) зв'язків між обчислювальними блоками «процесор + локальна пам'ять» здобувають ключове значення, адже чим довший шлях від одного процесора до іншого, тим довше буде йти обробка запитів, тим більший відсоток процесорного часу система буде витратити на задачах обміну даними та міжпроцесорної комунікації.

З'єднання «кожний з кожним» застосовуються досить рідко через їхню дорожнечу й складність при великій кількості обчислювальних модулів. Замість цього застосовуються інші топології, що забезпечують найкоротші шляхи при мінімальних апаратних витратах (топології тора, n-мірні гіперкуби й сітки). Ціною гарної масштабованості таких систем є складність їхнього програмування через необхідність відстеження даних в локальних областях пам'яті, перекладання на програміста завдання по організації обміну інформацією між процесорами й т.ін.

Векторний процесор — процесор, в якому операндами деяких команд можуть слугувати впорядковані масиви даних — вектори. Відрізняється від скалярних процесорів, які можуть працювати лише з одним оператором в одиницю часу. Абсолютна більшість процесорів є скалярними або близькими до них. Векторні процесори були розповсюджені в галузі наукових обчислень, де вони були основою більшості суперкомп'ютерів починаючи з 1980х і до 1990х. Але різке збільшення продуктивності і активна розробка нових процесорів призвели до того, що векторні процесори були витіснені зі сфери повсякденних процесорів.

В більшості сучасних мікропроцесорів є векторні розширення (зокрема команд SSE), крім того сучасні відеокарти та фізичні прискорювачі можна розглядати як векторні співпроцесори.

Наприклад, є відмінності роботи векторного і скалярного процесора, при додаванні 10 чисел. При «звичайному» програмуванні використовується цикл, що бере пари чисел послідовно, і додає їх.

повторити цикл 10 разів

прочитати наступну інструкцію та декодувати

отримати перший доданок

отримати другий доданок

скласти

зберегти результат

кінець циклу

Для векторного процесора алгоритм буде значно відрізнятися:

отримати наступну інструкцію і декодувати

отримати 10 перших доданків

отримати 10 других доданків

додати

зберегти результат

Реалізація фірми Cray розширила можливості обчислень, що дозволило виконувати декілька різних операцій одразу. Наприклад, розглянемо код, що додає 2 набори чисел і помножує на третій, у Cray ці операції здійснились би так:

отримати наступну інструкцію і декодувати

отримати 10 чисел

отримати 10 чисел

отримати 10 чисел

додати і помножити їх

зберегти результат

Таким чином, математичні операції виконуються значно швидше, основним фактором, що обмежує стає час необхідний для добуття даних з пам'яті.

13.4.3 Суперкомп'ютери

Суперкомп'ютер (supercomputer) — загальний термін, який використовується для позначення класу існуючих найпотужніших комп'ютерних систем. Суперкомп'ютери, зазвичай, використовуються при вирішенні складних наукових інженерних задач, які вимагають виконання великої кількості математичних операцій та(чи) працюють з великими об'ємами даних.

Так уряд Японії на 2010 р. для створення національного суперкомп'ютера виділив 253 млн доларів бюджетних грошей. Компанія ІВМ наразі працює над створенням суперкомп'ютера Blue Waters, здатного досягти продуктивності 16 петафлопс.

В 2012 р. IBM випустили суперкомп'ютер Sequoia з потужністю 20 петафлопс. Цей комп'ютер використовується для моделювання випробувань ядерної зброї. Уже здійснено запуск суперкомп'ютерного кластера на базі IBM Blade Center з потужністю 10 терафлопс в Казахстансько-Британському технічному університеті, який в 12 редакції Top50 посів 19 місце. Серйозні заявки зробила Білорусія. В 2019 р. очікується поява суперкомп'ютера з продуктивністю, що буде вимірюватись уже в ексафлопсах (10^{18} операцій за секунду).

Останнім часом розвиток отримав Exascale computing, що позначає гіпотетичні суперкомп'ютери з продуктивністю порядку одного ексафлопса (exaFLOPS). Така продуктивність в тисячу разів вище, ніж у систем петафлопсного класу, що з'явилися в 2008 році.

Один ексафлопс дорівнює тисячі петафлопс, мільярду мільярдів (10^{18}) операцій над числами з плаваючою комою в секунду (звичайно враховуються операції над числами в 64 -біном форматі IEEE 754 double). Різні автори прогнозували наприкінці 2000-х років можливу побудову ексафлопсних систем не раніше ніж в 2018-2020 роках.

Станом на 2013 рік найбільш продуктивний суперкомп'ютер, Тяньхе-2 досяг рівня в 33 Пфлопс (55 теоретичних) при енергоспоживанні в 24 МВт.

У Європейському Союзі діє три проекти з розвитку апаратних і програмних технологій для ексафлопсних суперкомп'ютерів: CRESTA (Collaborative Research into Exascale Systemware, Tools and Applications), DEEP (Dynamical ExaScale Entry Platform), Mont-Blanc. У Японії інститут RIKEN (Advanced Institute for Computational Science) планував створення ексафлопсної системи в 2020 з енергоспоживанням не вище 30 МВт.

13.4.4 Суперкомп'ютери Cray

Суперкомп'ютери Cray -1 спроектовано Сеймуром Креєм і створено компанією Cray Research Inc. у 1976 році. Пікова продуктивність машини — 133 Мфлопс.

Для Cray-1 побудовано процесор, який швидко виконував і скалярні і векторні обчислення. Цього вдалося домогтися через створення так званих «векторних реєстрів» — модулів пам'яті невеликого обсягу, які розташовувалися близько до процесора і працювали дуже швидко (але коштували дуже дорого). Таким чином центральний процесор брав дані з реєстрів і записував дані теж в реєстри, реалізуючи новий принцип роботи з пам'яттю «реєстр-реєстр».

ОЗП (від 1 до 4 мегабайт), великий набір процесорних реєстрів, що складаються з групи векторних реєстрів по 64 елементи, блок скалярних реєстрів, блок адресних реєстрів. Кожна група реєстрів пов'язана зі своїм конвеєрним процесором (рис. 13.14)..

Дана система могла виконувати скалярні операції над векторними даними, над адресами, числами з плаваючою комою (порядок — 15, мантиса — 49). Швидкодія 180 млн операцій за секунду з плаваючою комою. У даній системі використовуються команди довжиною 16 або 32 розрядів. У коротких командах 7 розрядів виділяється під код операції, 3 адресних поля по 3 розряду, визначали номер реєстра для зберігання операндів. У довгих — 22 розряди для того, щоб можна було знайти операнд в загальному полі ОП. Один з реєстрів визначає довжину вектора, другий — реєстр маски.

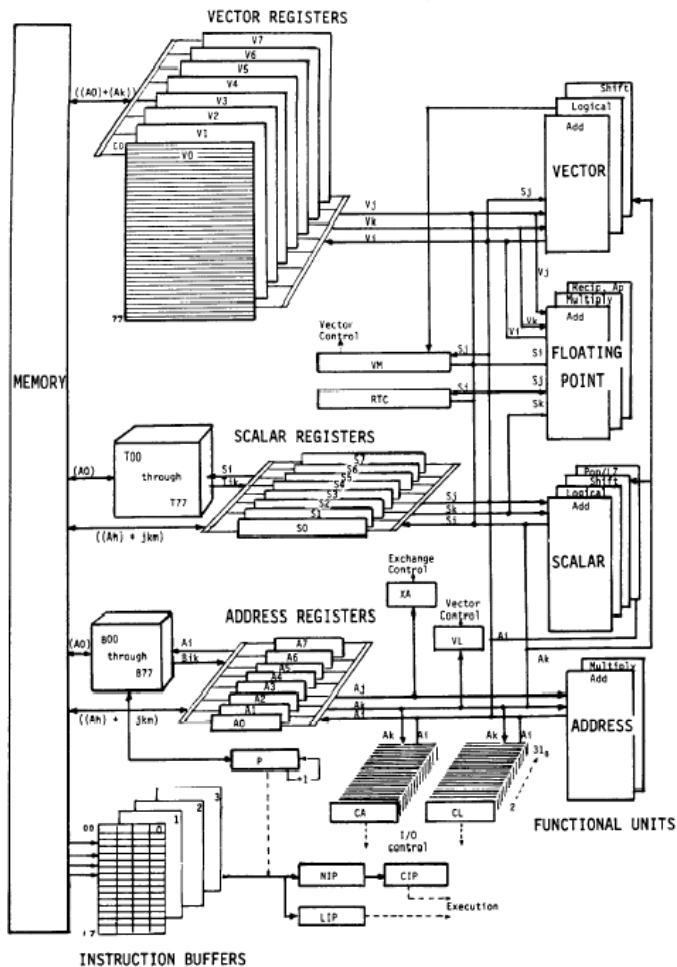


Рисунок 13.14 -The Cray-1 computation section

Центральний процесор Cray-1 складався з 500 друкованих плат, на кожній з яких з обох сторін розташовувалося по 144 мікросхеми. Всього виходило 144.000 мікросхем, які охолоджувалися фреоном. Для кращого охолодження і циркуляції фреону в охолоджувальній системі центральний

процесор був виконаний у стилі «вежі» з 12 колонами, складеними у формі дуги довжиною 270 градусів (у вигляді літери «С» — від «Cray», якщо дивитися зверху), а охолоджувальна система була розташована в підставі цієї вежі. Так був створений характерний, оригінальний і впізнаваний вигляд комп'ютера, що нагадував диван.

Машина пропонувалася в трьох модифікаціях: А, В і С, які відрізнялися один від одного тільки розмірами пам'яті: 1 мільйон слів, 500 тисяч слів і 250 тисяч слів відповідно. З цих модифікацій реально продавалися тільки Cray-1A і Cray-1B. На Cray-1C так і не знайшлося покупця, і відповідно не було побудовано жодного примірника цієї модифікації.

У 1984 Cray Research анонсувала поліпшену модель Cray X - MP з одним (Cray X - MP / 1), двома (Cray X - MP / 2) і 4 - ма процесорами (Cray X - MP / 4) з ОЗП 4 і 8 млн . слів. Найпродуктивніша система X - MP / 48 мала 4 процесора і пам'ять 8 млн слів. Її теоретична пікова продуктивність перевищувала 800 Мфлопс. Процесор цих моделей підтримував векторну адресацію ОЗП (читання / модифікація / и міг максимально адресувати до 16 млн. слів в ОЗП.

У 1978 році для Cray-1 був випущений перший стандартний пакет програмного забезпечення, що складався з трьох головних продуктів:

Операційна система (Cray Operating System, COS) (пізні машини працювали під UNICOS)

Мова асемблера Cray (Cray Assembly Language, CAL)[7]

Cray FORTRAN (CFT)[8], перший компілятор FORTRAN з автоматичною векторизацією

Конкурентами Cray - 1 на ринку суперкомп'ютерів виступали машини компанії CDC START - 100 и Cyber 76 , TI ASC компанії Texas

Instruments, ILLIAC IV компанії Burroughs , IBM 370/195 , STARAN. Було продано 61 машина Cray – 1.

Подальшим розвитком у 1979 рік було створення Cray-1S: оновленням система введення-виведення, збільшений діапазон пам'яті: від 1 до 4 мільйонів слів. У 1982 рік створено Cray-1M, де використано сучасні, менш дорогі компоненти, що дозволило при тій же продуктивності, що і Cray-1S, знизити вартість машини з 8-13.3 мільйона доларів до 4-7 мільйонів доларів

У 1982 році паралельною командою інженерів під керівництвом Стіва Чена на основі Cray-1 був створений багатопроцесорний комп'ютер Cray X-MP (1986 р.) з удосконаленою архітектурою (Extended Architecture или Cray X-MP/EA), с довжиною тактового циклу 8,5 нс (тактова частота 117 МГц) на чіпах матриць макрокомірок і масивах логічних вентилів. Було застосовано удосконалені 32-х бітові регістри А і В та 32-бітова адресна арифметика з адресацією до 2 млрд слів.

Максимальний об'єм ОЗП складав 64 млн слів, організованих у 64 банка. Підтримувалась 24-бітова адресація для сумісності з Cray. Досягнута пікова продуктивність 942 Мфлопс.

Архітектура Cray X-MP була удосконалена Supertek Computers на NBIC у мінікомп'ютері S-1. Надалі S-1 випускався під назвами Cray XMS, Cray Y-MP EL, Cray EL90, Cray J90

Команії Hitachi, Fujitsu и NEC створили сумісний з Cray X-MP паралельні суперкомп'ютери.

Векторний суперкомп'ютер Cray-2 випускався компанією Cray Research з 1985 року. Він був найпродуктивнішим комп'ютером свого часу, обігнавши за продуктивністю інший суперкомп'ютер, Cray X-MP. Пікова продуктивність Cray-2 становила 1,9 Гфлопс. Тільки в 1990 році цей рекорд був побитий суперкомп'ютером ETA-10G.

Розробка Cray-2 (1985 р.) показала тестах пікову продуктивність 1.9 Гфлопс, і стала найшвидшим в світі суперкомп'ютером, змістивши з п'єдесталу Cray X-MP, випущений в 1983 році. Цей титул Cray-2 утримував до 1990 року, коли компанія ETA (підрозділ компанії CDC) випустила суперкомп'ютер ETA-10G.

Cray-2 мав 2 або 4 векторних процесора, час такту 4.1 нс, об'єм пам'яті: 256 мільйонів 64-розрядних слів, UNIX-подібну операційну систему Unicos або Cray Operating System. Програмне забезпечення включало два компілятора мови Fortran: CFT2 і CFT77 з автоматичною векторизацією коду, компілятор мови C, макро-асемблер CAL, утиліти і бібліотеки для роботи з пристроями введення-виведення та організації виконання завдань. Енергоспоживання: 195 кВт. У 1985 році вартість Cray-2 становила 17.6 мільйона доларів США.

Але на на ринку всі її переваги в продуктивності виявлялися здебільшого за рахунок цієї швидкої і великої пам'яті. Завдяки властивостям пам'яті Cray-2 комп'ютерне моделювання змогло перейти від двомірних моделей і наближених тривимірних до точних тривимірних моделей.

Подальший розвиток архітектури Cray пішов по шляху систем MPP масивно паралельних архітектур з розподіленою пам'яттю таких як Cray T3D і Cray T3E. Сучасні машини Cray-T3D можуть налічувати від 32 до 2048 обчислювальних вузлів. Кожний вузол містить по два процесори Alpha 150MHz, кожен з 8 М слів ОЗП, та контролера мережних передач. ОЗП вузла утворює "розподілену спільну" пам'ять (DSM, distributed shared memory),

13.4.5 Суперкомп'ютер Cray X1

Cray X1 складається з процесорних вузлів, по чотири процесори в кожному. Процесори в X1 називаються - MSP (рис. 13.15). MSP (Multi-Streaming Processor)- багатопотоковий процесор.

Процесор складається з чотирьох скалярних елементів, кожен з яких має пару векторних конвесрів. Кожна скалярна секція може вибирати і виконувати по дві інструкції на такт. Скалярні секції працюють в процесорі на частоті 400MHz, а отже скалярна швидкодія MSP складає 3.2Gflops.

Векторні секції процесора працюють на частоті 800MHz. Оскільки на кожний скалярний у процесорі припадає два векторних, то не є складним поррахувати, що MSP має векторну швидкодію в 12.8Gflops при виконанні обчислень над 64 розрядними даними з плаваючою точкою, та 25.6Gflops при обробці 32 розрядних.

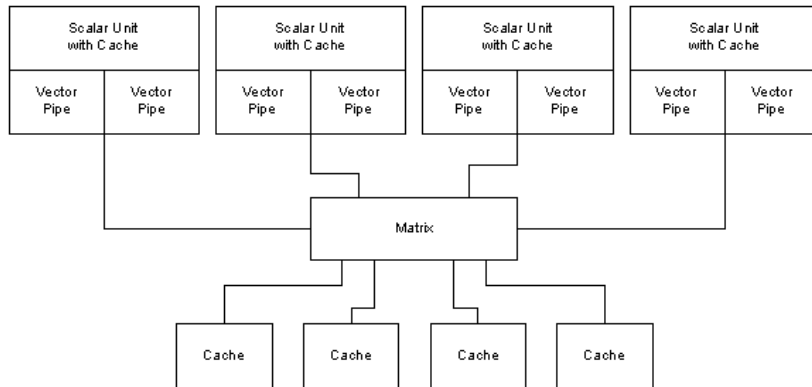


Рисунок 13.15 - Структурна схема MSP

Дизайн високошвидкісної підсистеми вводу-виводу даних та оперативної пам'яті забезпечує (Рисунок 6):

Пропускнну здатність процесорний елемент - кеш на швидкості 76GB/s (51.2GB/s загрузка та 25.6GB/s збереження).

Пікову пропускнну здатність локальної оперативної пам'яті 51GB/s на процесор (неперервна потокова пропускна здатність локальної оперативної пам'яті складає 38.4GB/s на процесор, інтегрально з усіх чотирьох кеш модулів процесора). У системі використовується 256Mbit RAMBUS пам'ять. Для досягнення надвисоких показників швидкодії в кожному вузлі використовуються 16 контролерів пам'яті.

Загальна пропускна здатність між процесором та міжвузловими комунікаціями, які забезпечують доступ до глобальної оперативної пам'яті, фізично розподіленої між вузлами, складає 25.6GB/s. Відповідно між процесорними вузлами (4-х процесорний) - 102GB/s.

Пропускнну здатність для підсистеми вводу-виводу 1.2GB/s на один канал, до 4.8GB/s на вузол і до 75GB/s на кабінет. Кожний канал є спільним ресурсом і доступний будь-якому процесору в межах усієї системи.

· Затримки при доступі до глобальної оперативної пам'яті для всіх вузлів системи знаходяться в рамках мікросекунди. Для системи на 512 процесорів (128 вузлів) затримка по доступу до глобальної оперативної пам'яті складає близько однієї мікросекунди (рис. 13.16).

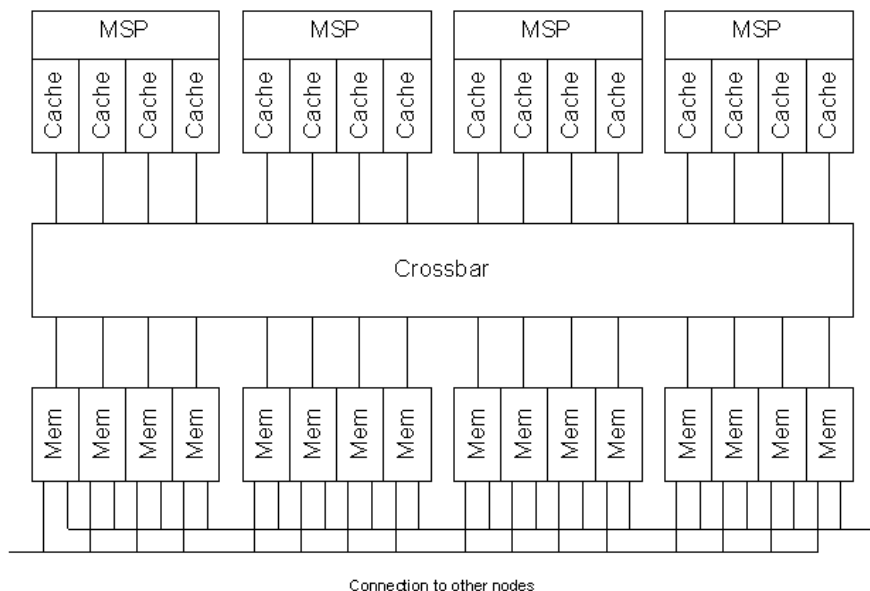


Рисунок 13.16 - Структурна схема вузла Cray X1

При побудові багатовузлових конфігурацій, більше 8-ми, в системах Cray X1 використовується топологія гіперкубу. Для системи на 512 процесорів (128 вузлів) використовується гіперкуб на 16-ть вершин, в кожній з яких розміщується 8 вузлів, з'єднаних двома маршрутизаторами, які забезпечують також зв'язок із зовнішніми щодо вершини маршрутизаторами (рис. 13.17).

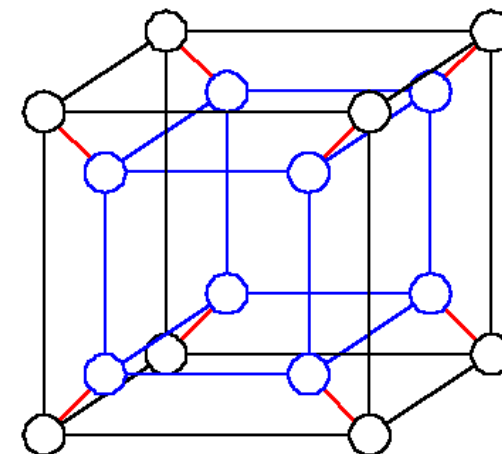


Рисунок 13.17 - Гіперкуб, спрощена структурна схема мережі Cray X1

Слід зауважити, що для ряду неспеціалізованих систем при виконанні задач суперкомп'ютерного класу спостерігається дуже низька ефективність використання процесорної потужності, а саме ефективність на межі 5% від пікової швидкодії. В системі Cray X1, дякуючи надзвичайно високій пропускній здатності оперативної пам'яті та підсистеми вводу-виводу, на аналогічних задачах, можна досягнути ефективності у 50%.

В Cray X1 використовується операційна система UNICOS/mp. Вона базується на UNIX System V і підтримує всі команди інтерфейси POSIX. Практично, це вдосконалена версія ОС UNICOS/mk, яка використовувалась у системах Cray T3E. Програмне забезпечення Cray X1 підтримує програмування в рамках моделі спільної та розподіленої пам'яті на C, C++, Fortran, SHMEM, Co-array Fortran, UPC, Parallel C та Open MP.

13.4.6 Архітектура NUMA

Для більшої масштабованості мультипроцесорів пристосована архітектура NUMA (NonUniform Memory Access - з неоднорідним доступом до пам'яті). Як і мультипроцесори UMA, вони забезпечують єдиний адресний простір для всіх процесорів, але, на відміну від машин UMA, доступ до локальних модулів пам'яті відбувається швидше, ніж до віддалених.

Машини NUMA мають три ключові характеристики, якими всі вони володіють і які в сукупності відрізняють їх від інших мультипроцесорів:

- існує один адресний простір, видимий для всіх процесорів;
- доступ до віддаленої пам'яті проводиться з використанням команд LOAD і STORE;
- доступ до віддаленої пам'яті відбувається повільніше, ніж доступ до локальної пам'яті.

Доступ процесора до власної локальної пам'яті проводиться безпосередньо, що набагато швидше, ніж доступ до віддаленої пам'яті через комутатор або мережу.

NUMA архітектура об'єднує в собі переваги систем зі спільною пам'яттю та високу масштабованість. Зазвичай, NUMA обчислювальні системи складаються з SMP вузлів, які з'єднані між собою за допомогою високошвидкісних комутаторів, що забезпечують можливість роботи всіх процесорів у системі в рамках одного адресного простору.

Безумовно, що через різномірність, доступ до пам'яті, яка знаходиться в локальному вузлі, значно швидший, ніж доступ до пам'яті яка знаходиться у віддаленому вузлі. Причому, у випадку використання

каскадного підключення вузлів один до одного через декілька комутаторів, доступ до пам'яті вузлів у рамках різних комутаторів буде різним.

Саме через те, що у рамках всієї обчислювальної системи з архітектурою NUMA підтримується спільний адресний простір (система зі спільною пам'яттю), програмування такої системи так само просте, як і програмування системи з архітектурою SMP. Але, оскільки існує неоднорідність у швидкодії доступу до локальної і віддаленої пам'яті, виникає проблема яку доводиться вирішувати як зі сторони проектувальників апаратних, так і програмних засобів.

Проектувальники апаратних засобів прагнуть досягнути мінімальних затримок на комутаторах і, як наслідок, - мінімальної неоднорідності при доступі процесорів до пам'яті локальних та віддалених вузлів. А програмісти, які проектують системні програмні засоби, намагаються їх спроектувати з урахуванням мінімізації доступу до пам'яті віддалених вузлів.

13.4.7 Архітектура ccNUMA

В архітектурі NUMA кеш-пам'ять процесорів може бути так само синхронізована як і в SMP системах, що забезпечує ефективну роботу системи в рамках одного адресного простору.

Така архітектура носить назву ccNUMA (cache coherent Non Uniform Memory Access). Когерентність кеш-пам'яті забезпечується спеціалізованими апаратними засобами. Але, нажаль, масштабованість таких систем обмежується як програмними так і апаратними особливостями, серед яких можливості операційної системи керувати великою кількістю процесорів, адресувати великий об'єм пам'яті, а також

обмеження у швидкодії апаратних ресурсів, які забезпечують когерентність кеш між вузлами.

Сьогодні максимальна кількість, яку підтримують системи з архітектурою ccNUMA, встановлена в системах SGI Origin2000 складає 256 процесорів.

Обчислювальні системи з розподіленою пам'яттю масштабуються набагато краще, ніж системи зі спільною пам'яттю, але програмувати їх складніше. Основною особливістю паралельних систем з розподіленою пам'яттю є те, що прямиий доступ до пам'яті інших вузлів неможливий або здійснюється спеціалізованими засобами.

Відповідно, обмежень на операційні системи по масштабуванню кількості вузлів не накладається. Ідея побудови таких систем дуже проста, обчислювальні вузли (зазвичай SMP або ccNUMA) зі своїми локальними процесорами, пам'яттю та копією операційної системи об'єднуються за допомогою комунікаційних інтерфейсів. Через комунікаційні інтерфейси здійснюється обмін в рамках всієї обчислювальної системи. Такий підхід дозволяє будувати системи, масштабованість яких за кількістю процесорів практично необмежена.

Категоричних вимог до однорідності вузлів та швидкодії міжпроцесорного обміну в системах з розподіленою пам'яттю немає. В залежності від задач та доступних ресурсів такі системи будуються з вузлів із різною кількістю процесорів, на різноманітних, часом неспеціалізованих операційних системах з різними засобами міжвузлового зв'язку. Дякуючи таким особливостям, параметр ціна/швидкодія у систем із розподіленою пам'яттю заздалегідь краще, ніж в систем з іншими архітектурами.

13.4.7 Архітектури кластерних систем

13.4.7.1 Класифікація кластерних систем

Кластер — це декілька незалежних обчислювальних машин, що використовуються спільно і працюють як одна система для вирішення тих чи інших задач, наприклад, для підвищення продуктивності, забезпечення надійності, спрощення адміністрування тощо. Обчислювальний кластер потрібен для збільшення швидкості обрахунків за допомогою паралельних обчислень.

Кластер — це різновид паралельної або розподіленої системи, яка: складається з декількох зв'язаних між собою комп'ютерів; використовується як єдиний, уніфікований комп'ютерний ресурс».

Зазвичай розрізняють наступні основні види кластерів:

відмовостійкі кластери (High-availability clusters, HA, кластери високої доступності);

кластери з балансуванням навантаження (Load balancing clusters);

обчислювальні кластери (High performance computing clusters);

grid-системи.

13.4.7.2 Архітектура кластера

Найпоширенішим є використання однорідних кластерів, тобто таких, де всі вузли абсолютно однакові по своїй архітектурі й продуктивності.

Для кожного кластера є виділений сервер — керуючий вузол (frontend). На цьому комп'ютері встановлене програмне забезпечення, яке активізує обчислювальні вузли при старті системи й управляє запуском програм на кластері.

Властиво обчислювальні процеси користувачів запускаються на обчислювальних вузлах, причому вони розподіляються так, що на кожний процесор доводиться не більш одного обчислювального процесу.

Користувачі мають домашні каталоги на сервері доступу — шлюзі (цей сервер забезпечує зв'язок кластера із зовнішнім світом через корпоративну ЛВС або Інтернет), безпосередній доступ користувачів на керуючий вузол виключається, а доступ на обчислювальні вузли кластера можливий (наприклад, для ручного керування компіляцією завдання).

Обчислювальний кластер, як правило, працює під керуванням однієї з різновидів ОС Unix — багатокористувацької багатозадачної мережевої операційної системи. Зокрема, в ІК НАН України кластери працюють під керуванням ОС Linux — вільно розповсюджуваного варіанта Unix.

Існує декілька способів зайняти обчислювальні потужності кластера:

- Запускання багатьох однопроцесорних завдань. Це може бути сприятливим варіантом, якщо потрібно провести багато незалежних обчислювальних експериментів з різними вхідними даними, причому час проведення кожного окремого розрахунків не має значення, а всі дані розміщуються в об'ємі пам'яті, доступному одному процесу.

- Запускати готові паралельні програми. Для деяких завдань доступні безкоштовні або комерційні паралельні програми, які при необхідності Ви можете використовувати на кластері. Як правило, для цього досить, щоб програма була доступна у вихідних текстах, реалізована з використанням інтерфейсу MPI на мовах C/C++ або Фортран.

- Викликати у своїх програмах паралельні бібліотеки. Для деяких областей, наприклад, лінійна алгебра, доступні бібліотеки, які дозволяють вирішувати широке коло стандартних підзадач з використанням можливостей паралельної обробки. Якщо звертання до таких підзадач становить більшу частину обчислювальних операцій

програми, то використання такої паралельної бібліотеки дозволить одержати паралельну програму практично без написання власного паралельного коду. Прикладом такої бібліотеки є SCALAPACK.

- Створювати власні паралельні програми. Це найбільш трудомісткий, але й найбільш універсальний спосіб. Існує кілька варіантів такої роботи, зокрема, вставляти паралельні конструкції в готові паралельні програми або створювати з «нуля» паралельну програму.

Паралельні програми на обчислювальному кластері працюють у моделі передачі повідомлень — message passing. Це значить, що, хоч програма складається з багатьох процесів, кожен з яких працює на своєму процесорі й має власний захищений адресний простір, ці процеси можуть обмінюватися повідомленнями за допомогою структур операційної системи, таких як семафори та спільно використовувана пам'ять. Тобто процес, який повинен одержати дані, викликає операцію receive (прийняти повідомлення), і вказує, від якого саме процесу він повинен одержати дані, а процес, який повинен передати дані іншому, викликає операцію send (послати повідомлення) і вказує, якому саме процесу потрібно передати ці дані.

Відмовостійкі кластери та системи взагалі будуються по трьом основним принципам:

- з холодним резервом або активний / пасивний. Активний вузол виконує запити, а пасивний чекає його відмови і включається в роботу, коли така відбудеться. Приклад — резервні мережеві з'єднання, зокрема, алгоритм зв'язуючого дерева. Наприклад зв'язку DRBD та HeartBeat.

- з гарячим резервом або активний / активний. Всі вузли виконують запити, в разі відмови одного навантаження перерозподіляється між рештою. Тобто кластер розподілення навантаження з підтримкою перерозподілу запитів при відмові. Прикладами є практично всі кластерні

технології, наприклад, Microsoft Cluster Server. OpenSource проект OpenMosix.

– з модульною надмірністю. Застосовується тільки у випадку, коли простій системи абсолютно неприпустимий. Всі вузли одночасно виконують один і той же запит (або частини його, але так, що результат досяжний і при відмові будь-якого вузла), з результатів береться будь-який. Необхідно гарантувати, що результати різних вузлів завжди будуть однакові (або відмінності гарантовано не вплинуть на подальшу

13.4.7.3 Кластери розподілу навантаження

Принцип їх дії будується на розподілі запитів через один або кілька вхідних вузлів, які перенаправляють їх на обробку в інші, обчислювальні вузли. Початкова мета такого кластера — продуктивність, однак, у них часто використовуються також і методи, що підвищують надійність. Подібні конструкції називаються серверними фермами. Програмне забезпечення (ПЗ) може бути як комерційним (OpenVMS, MOSIX, Platform LSF HPC, Solaris Cluster Moava Cluster Suite, Maui кластера Scheduler), так і безкоштовним (OpenMosix Sun Grid Engine, Linux Virtual Server).

13.4.7.4 Обчислювальні кластери

Кластери використовуються в обчислювальних цілях, зокрема в наукових дослідженнях. Для обчислювальних кластерів істотними показниками є висока продуктивність процесора в операціях над числами з плаваючою точкою (flops) і низька латентність об'єднує мережі, і менш істотними — швидкість операцій введення-виведення, яка більшою мірою важлива для баз даних та web-сервісів. Обчислювальні кластери

дозволяють зменшити час розрахунків, порівняно з одиночним комп'ютером, розбиваючи завдання на паралельно виконуваних гілках, які обмінюються даними по зв'язуючій мережі. Одна з типових конфігурацій — набір комп'ютерів, зібраних із загальнодоступних компонентів, з встановленою на них операційною системою Linux, і пов'язаних мережею Ethernet, Myrinet, InfiniBand або іншими відносно недорогими мережами. Таку систему прийнято називати кластером Beowulf.

Спеціально виділяють високопродуктивні кластери (Позначаються англ. аббревіатурою HPC Cluster' — High-performance computing cluster). Список найпотужніших високопродуктивних комп'ютерів (також може позначатися англ. аббревіатурою HPC) можна знайти в світовому рейтингу TOP500. У Росії ведеться рейтинг найпотужніших комп'ютерів СНДТОР50 Суперкомп'ютери.

13.4.7.5 Системи розподілених обчислень grid

Такі системи не прийнято вважати кластерами, але їх принципи в значній мірі схожі з кластерною технологією. Їх також називають grid-системами. Головна відмінність — низька доступність кожного вузла, тобто неможливість гарантувати його роботу в заданий момент часу (вузли підключаються і відключаються в процесі роботи), тому завдання повинне бути розбите на ряд незалежних один від одного процесів. Така система, на відміну від кластерів, не схожа на єдиний комп'ютер, а служить спрощеним засобом розподілу обчислень. Нестабільність конфігурації, в такому випадку, компенсується великим числом вузлів.

13.4.7.6 Кластер серверів, організованих програмно

Кластер серверів (в інформаційних технологіях) — група серверів, об'єднаних логічно, здатних обробляти ідентичні запити і використовуються як єдиний ресурс. Найчастіше сервери групуються за допомогою локальної мережі. Група серверів володіє більшою надійністю і більшою продуктивністю, ніж один сервер. Об'єднання серверів в один ресурс відбувається на рівні програмних протоколів.

На відміну від апаратного кластера комп'ютерів, кластери організовані програмно, вимагають:

Наявності спеціального програмного модуля (Cluster Manager), основною функцією якого є підтримка взаємодії між усіма серверами — членами кластеру:

Синхронізації даних між усіма серверами — членами кластеру;

Розподіл навантаження (клієнтських запитів) між серверами — членами кластеру;

Від уміння клієнтського програмного забезпечення розпізнавати сервер, що являє собою кластер серверів, і відповідним чином обробляти команди від Cluster Manager;

Якщо клієнтська програма не вміє розпізнавати кластер, вона буде працювати тільки з тим сервером, до якого звернулася спочатку, а при спробі Cluster Manager перерозподілити запит на інші сервери, клієнтська програма може взагалі позбутися доступу до цього сервера (результат залежить від конкретної реалізації кластера).

13.4.7.7 Особливості реалізації кластерів

Архітектура кластерних обчислювальних систем є досить простою і, зрозумілою, дякуючи наслідуванню традиційних комп'ютерних технологій.

Обчислювальна потужність кластерної система визначається швидкістю обчислювальних вузлів та міжвузлових комунікацій.

Вибір платформи для застосування в якості обчислювального вузла кластерної системи зазвичай обумовлюється такими факторами як:

максимальна швидкість процесорів, які підтримуються платформою
· максимальна кількість процесорів на платформі;

вага міжпроцесорних комунікацій в алгоритмі типових задач, на які орієнтована система;

доля вартості платформи у вартості системи, порівняно з вартістю комунікацій для міжвузлового зв'язку.

При виборі інтерфейсів міжвузлових комунікацій враховуються параметри пропускної здатності інтерфейсу, затримки передачі повідомлень (програмно-апаратна складова) та вплив інтерфейсу на завантаження центрального процесора задачами пересилки повідомлень. Так, наприклад, застосування у використаному інтерфейсі технологій прямого доступу до пам'яті дозволяє прикладній програмі безпосередньо передавати дані у пам'ять віддаленого вузла.

Розробники кластерних обчислювальних систем зазвичай використовують в якості комунікаційних інтерфейсів міжвузлового зв'язку такі технології як: Ethernet, Myrinet, SCI, Giganet та деякі інші. Причому однією з найсерйозніших проблем при виборі інтерфейсу є саме величина затримки, а не швидкості передачі, так як різниця швидкості доступу до локальної пам'яті та передачі пакета до іншого вузла залишається критичною величиною для багатьох задач.

Час затримки спеціалізованих інтерфейсів для міжвузлового зв'язку в кластерних системах на порядок, інколи на два, краще, ніж у традиційних мережних інтерфейсів, як то Fast Ethernet чи Gigabit Ethernet. Комунікаційні інтерфейси міжвузлового зв'язку зазвичай використовують

канальні технології, але при цьому мають гнучкість близьку до мережних інтерфейсів і достатню для організації високошвидкісних кластерних систем.

Незважаючи на такий відрив технологій, у кластерних системах часто використовують Ethernet інтерфейси, через їх надзвичайно низьку ціну та відкритість, порівняно зі спеціалізованими комунікаціями. Також традиційні мережні інтерфейси часто застосовуються для міжвузлових комунікацій, у випадку, коли кластерна система проектується для задач, які не вимагають частого міжпроцесорного обміну.

Проблематика побудови кластерних систем не закінчується на виборі інтеграції апаратного забезпечення. Перед початком її програмування та використання потрібно підготувати системну програмну складову для забезпечення функціонування у програмному середовищі відповідних прикладних задач.

Одним з найбільш відпрацьованих та популярних варіантів програмного забезпечення для підтримки моделей кластерингу з динамічним розподіленням ресурсів та передачею повідомлень між підзадачами є спільне використання пакетів MOSIX та MPI. MOSIX дозволяє динамічно розподіляти задачі по всіх вузлах кластеру прозоро для користувачів системи. Причому задачу не потрібно для цього переписувати, використовуються абсолютно звичайні класичні програмні пакети.

Для програмування кластеру в моделі передачі повідомлень найчастіше використовується MPI (Message Passing Interface). Це стандарт для побудови паралельних програм та обміну повідомленнями. Існують реалізації MPI для мов C/C++ і Fortran, як в безкоштовних так і в комерційних варіантах для більшості розповсюджених суперкомп'ютерних

платформ. В тому числі є багато варіантів для високошвидкісних обчислювальних кластерних систем.

13.4.7.8 Кластер Beowulf

Beowulf (Beowolf) - кластер, який складається з широко поширеного апаратного забезпечення, що працює під управлінням операційної системи, поширюваної з вихідними кодами (наприклад, GNU / Linux або FreeBSD).

Перший кластер з 16 процесорів Intel DX4, з'єднаних мережею Ethernet з дублюванням каналів. Ідея створення системи з готових стандартних компонентів для конкретних обчислювальних потреб швидко поширилася в NASA, а також в академічній та дослідницькій середовищі. Зусилля з розробки таких машин швидко вилилися в те, що зараз називається проєкт Beowulf.

Зараз такі машини зазвичай зазивають "Кластерні EOM класу Beowulf" (Beowulf Class Cluster Computers). Особливістю такого кластера також є масштабованість, тобто можливість збільшення кількості вузлів системи з пропорційним збільшенням продуктивності. Вузлами в кластері можуть служити будь-які серійно випускаються автономні комп'ютери, кількість яких може бути від 2 до 1024 і більше. Для розподілу обробки даних між вузлами зазвичай використовуються технології MPI або PVM.

Переваги Beowulf-систем:

- вартість системи набагато нижче вартості суперкомп'ютера;
- можливість збільшення продуктивності системи;
- можливість використання застарілих комп'ютерів, тим самим збільшується термін експлуатації комп'ютерів;
- широка поширеність, а значить і доступність, апаратного забезпечення.

Вузли кластера. Підходящим вибором в даний момент є системи на базі процесорів Intel: Pentium II або Pentium II Xeon - або однопроцесорні ПК, або SMP-сервера з невеликим числом процесорів (2-4, можливо до 6).

З деяких причин оптимальним вважається побудова кластерів на базі двопроцесорних систем, незважаючи на те, що в цьому випадку настройка кластера буде дещо складніше (головним чином тому, що доступні відносно недорогі материнські плати для 2 процесорів Pentium II).

Варто встановити на кожен вузол 64-128МВ оперативної пам'яті (для двопроцесорних систем 64-256МВ). Одну з машин слід виділити в якості центральної (головний) куди слід встановити досить великий жорсткий диск, можливо більш потужний процесор і більше пам'яті, ніж на інші (робочі) вузли. Має сенс забезпечити (захищену) зв'язок цієї машини із зовнішнім світом.

При комплектації робочих вузлів цілком можливо відмовитися від жорстких дисків - ці вузли будуть завантажувати ОС через мережу з центральною машиною, що, крім економії коштів, дозволяє конфігурувати ОС і все необхідне ПЗ тільки 1 раз (на центральній машині). Якщо ці вузли не будуть одночасно використовуватися в якості користувальницьких робочих місць, немає необхідності встановлювати на них відеокарти і монітори. Можлива установка вузлів в стійки (rackmounting), що дозволить зменшити місце, займане вузлами, але коштуватиме дещо дорожче.

Можлива організація кластерів на базі вже існуючих мереж робочих станцій, тобто робочі станції користувачів можуть використовуватися в якості вузлів кластера вночі і у вихідні дні. Системи такого типу іноді називають COW (Cluster of Workstations). Кількість вузлів слід вибирати виходячи з необхідних обчислювальних ресурсів і доступних фінансових коштів. Слід розуміти, що при великому числі вузлів доведеться також встановлювати більш складне і дороге мережеве обладнання.

Мережі. У найпростішому випадку використовується один сегмент Ethernet (10Mbit / sec на кручений парі). Однак дешевизна такої мережі, внаслідок колізій обертається великими накладними витратами на міжпроцесорні обміни; а хорошу продуктивність такого кластера слід очікувати тільки на завданнях з дуже простої паралельної структурою і при дуже рідкісних взаємодіях між процесами (наприклад, перебір варіантів).

13.5 Системи PVP

PVP (Parallel Vector Processing - паралельні векторно-конвеєрні) обчислювальні системи складаються з мережі подібних до SMP вузлів, тобто систем зі спільною пам'яттю, але, на відміну від класичних SMP систем, вони містять векторно-конвеєрні процесори. Мережа у надпотужних PVP систем зазвичай побудована на комутаторах, за своїми характеристиками не гірших за комутатори MPP систем, та має масу спеціалізованих особливостей, орієнтованих на ефективні паралельні обчислення.

Особливості архітектури векторних процесорів (внутрішній паралелізм потоків даних зі зміщенням у часі) дозволяють досягнути дуже високої швидкодії в рамках одного процесора при його правильному завантаженні. Історія векторно-конвейерних архітектур розпочалася в 1976 році з комп'ютера Cray-1. Ця архітектура була найбільш успішною у світі суперкомп'ютерних технологій до кінця минулого тисячоліття і залишається такою досі.

На початку 2002 року компанія NEC оголосила про створення найпотужнішого у світі суперкомп'ютера "the Earth Simulator", побудованого на векторних процесорах. Earth Simulator складається з 640 обчислювальних вузлів, кожен з яких містить 8 процесорів потужністю

8Gflops кожен. Пікова швидкодія системи складає 40TFlops, швидкодія на тестах LINPACK склала 35.6Tflops, а це 89% пікової швидкодії, що свідчить про високу ефективність системи. У найближчих конкурентів по списку top500 ці показники складають 68% та 69% відповідно, а у найближчих кластерної системи з комунікаціями на Mynet - 50%.

При проектуванні програмного забезпечення для векторно-конвеєрних обчислювальних систем потрібно враховувати їх специфічні особливості. Для ефективного завантаження таких процесорів в програмах виділяють фрагменти, які замінюються векторними командами (векторизація програм). Якщо ж такі ділянки в програмі виділити неможливо, або якщо їх небагато, то кажуть, що програма погано векторизується, і виконання її на векторному процесорі є неефективним.

13.6 Суперкомп'ютер NEC Earth Simulator

NEC Earth Simulator, класичний представник паралельних векторно-конвеєрних архітектур з розподіленою пам'яттю. Як вже зазначалось, Earth Simulator складається з 640 обчислювальних вузлів (PN - Processor Node) по 8 процесорів у кожному. Всього система складається з 5120 процесорів із загальним обсягом оперативної пам'яті 10ТВ. Пікова швидкодія системи складає 40TFlops, швидкодія на тестах LINPACK склала 35.6Tflops. Слід зауважити, що високої швидкодії на цій, як і на будь-якій іншій векторно-конвеєрній системі, можна досягнути лише на задачах, які добре векторизуються.

Кожен вузол системи побудований на базі процесорів, які використовуються в суперкомп'ютерах SX-6. Процесори SX-6 є цілими векторними процесорами, реалізованими на одному чіпі, дякуючи чому співвідношення ціна/швидкодія цих процесорів є дуже високим.

Міжвузлові комунікації забезпечуються через матричний комутатор IXS. Система вводу-виводу суперкомп'ютера дозволяє під'єднувати сучасні пристрої за інтерфейсами SCSI, Fibre Channel та HIPPI. Вона також містить вживані сьогодні мережні інтерфейси Ethernet (Рисунок 13.18).

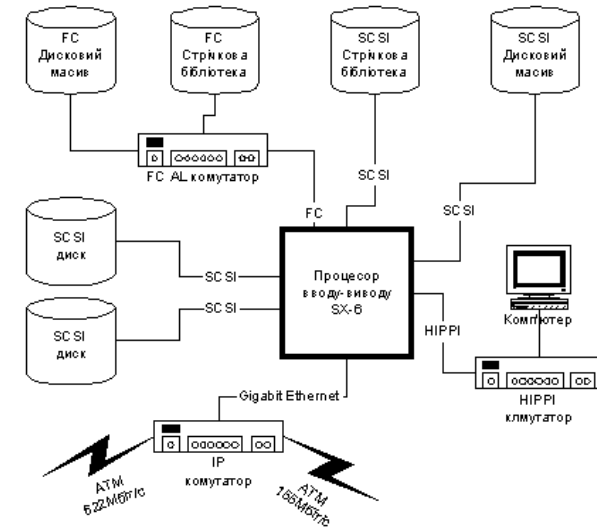


Рисунок 13.18- Структурна схема підключення периферійних пристроїв SX-6

Кожен обчислювальний вузол містить (рис. 13.19):

- 8 векторних обчислювальних процесорів (AP - arithmetic processor) потужністю 8Gflops кожен
- 16GB оперативної DDR SDRAM пам'яті, доступної напряму кожному процесору вузла через високошвидкісний матричний комутатор. Загальна пропускна здатність пам'яті вузла складає 256GB/s.
- пристрої віддаленого контролю (RCU - Remote Controll Unit)

· процесори вводу-виводу, які забезпечують швидкість передачі даних до 8GB/s.

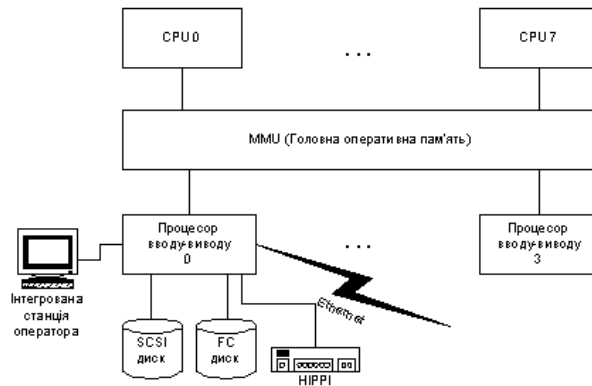


Рисунок 13.19. Структурна схема вузла SX-6

В комп'ютерах SX-6 орієнтована на суперкомп'ютерне використання операційна система типу UNIX - SUPER-UX. ОС відповідає UNIX System V стандарту та містить додаткові особливості й розширення для високошвидкісного вводу-виводу та підтримки багатовузлових конфігурацій (HPC кластерингу).

За твердженнями NEC всі програми написані для попереднього покоління суперкомп'ютерів SX-5, будуть без перекомпіляції працювати на SX-6 SUPER-UX. Ця ОС також підтримує POSIX інтерфейс як на рівні API так і на рівні командного інтерфейсу, що спрощує перенесення прикладного програмного забезпечення під SUPER-UX.

13.7 Система SP2

Система SP2 є класичною MPP системою з SMP вузлами на базі процесорів Power4 від IBM, з програмуванням в моделі з розподіленою пам'яттю та передачею повідомлень. Серед багатьох інших ці системи вирізняє їх універсальність та гнучкість архітектури, яка забезпечила їм популярність як серед наукових, так і промислових прикладних задач.

При проектуванні системи розробники з IBM ставили собі за мету побудувати систему, яка буде відповідати наступним принципам:

1. Високошвидкісна масштабована паралельна система повинна використовувати стандартні мікропроцесори, конструкції та операційні системи.
2. Для досягнення найкращого співвідношення ціна/швидкодія потрібно швидко виводити на ринок новітні технологічні досягнення.
3. Потрібні рівні затримки та часу доступу, а також пропускну здатності пам'яті можуть бути досягнуті лише за допомогою спеціалізованих мереж для обміну даними та комунікаційних підсистем.
4. Система повинна підтримувати середовище програмування та виконання програм, ідентичне стандартному відкритому середовищу ОС UNIX.
5. Для підтримки високошвидкісного обчислення вимогливих науково-технічних обчислень та обробки великих баз даних в системі слід реалізувати повний набір сервісних засобів, таких як: паралельний високошвидкісний зчитування та запис даних, високошвидкісні файлові системи, паралельні програмні бібліотеки.
6. Високого рівню готовності системи, побудованої на базі стандартних компонент, може бути досягнуто при відносно невеликих

додаткових затратах шляхом дублювання таких компонент, несправність яких може призвести до відмови всієї системи, а також шляхом забезпечення дуже швидкого відновлення системи при виникненні збоїв.

7. Необхідний рівень підтримки інтерфейсу з паралельною обчислювальною системою, як з єдиною машиною, може бути реалізований шляхом створення глобальних сервісних засобів по розподіленню ресурсів та команд, а також, створення засобів централізованого керування та адміністрування.

Закладені принципи у проектуванні системи призвели до побудови системи на основі масивно паралельної архітектури з розподіленою пам'яттю та високошвидкісною системою міжвузлових комунікацій з програмуванням в моделі передачі повідомлень.

У сучасних реалізаціях IBM SP2 в якості вузлів використовують Power4 сервери pSeries моделей 655 (4-8 процесорів) та 690 (8-32 процесора), а в якості міжвузлових комунікацій - високошвидкісні комутатори SP Switch 2.

Кожний обчислювальний вузол системи містить по чотири POWER4 multichip module (MCM). MCM являє собою обчислювальний процесор з одним або двома ядрами та кеш пам'яттю (Рисунок 8). Таким чином, вузли будуються з 4-х або 8-ми процесорних блоків.

Для задач, максимально вимогливих до пропускної здатності процесор-пам'ять, таких як обчислювальна гідродинаміка та структурний аналіз, краще використовувати моделі з одним процесором на кристалі. Більша пропускна здатність процесор-пам'ять у таких моделях досягається завдяки виділеній кеш пам'яті, в той час як для процесорів з подвійним ядром використовується спільний кеш другого рівня, що зменшує пропускну здатність в рамках одного процесора.

Потужність кожного процесора Power4+ частотою 1.5GHz складає 6Gflops, відповідно на один MCM з подвійним ядром - 12Gflops. Кеш пам'ять другого рівня об'ємом 1.5MB, інтегрована в кристал MCM і працює на частоті ядра. L3 кеш знаходиться між кристалом MCM та оперативною пам'яттю, її об'єм сягає 32MB на процесор. Швидкість обміну з кеш другого рівня для MCM з двома процесорами складає 124.8GB/s, для MCM з одним процесором - 83.2GB/s.

Таке порівняння складається на перший погляд не на користь однопроцесорного MCM (рис. 13.20), але, у зв'язку з тим, що вся пропускна здатність доступна одному процесору, і немає конкуренції за пропускну здатність L2 кеш, ми отримуємо значну перевагу для однопроцесорного кристала.

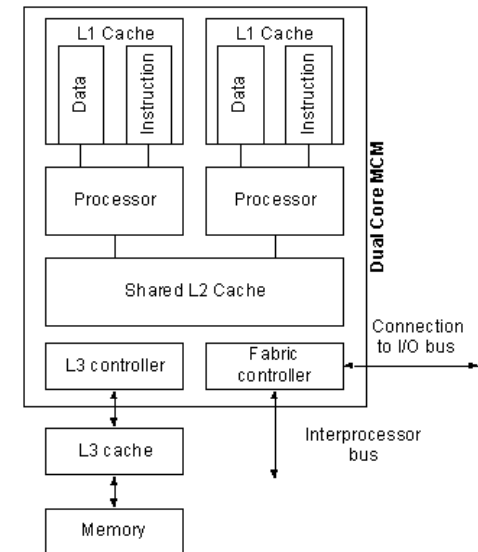


Рисунок 13.20 - Структура MCM з подвійним ядром* -

Структура MCM з одиночним ядром аналогічна до MCM з подвійним ядром, за виключенням відсутнього другого процесора та його

L1 кеш пам'яті для даних та команд. Відповідно, кеш пам'ять другого рівня стає не розподільною, а призначеною окремому процесору.

MCM в процесорному модулі зв'язані між собою через шинний пристрій з пропускнуою здатністю 51.2GB/s. Як для одно-, так і для двох'ядерних MCM, використовують однаковий чіпсет, і, - відповідно однаковий комутаційний пристрій (рис. 13.21).

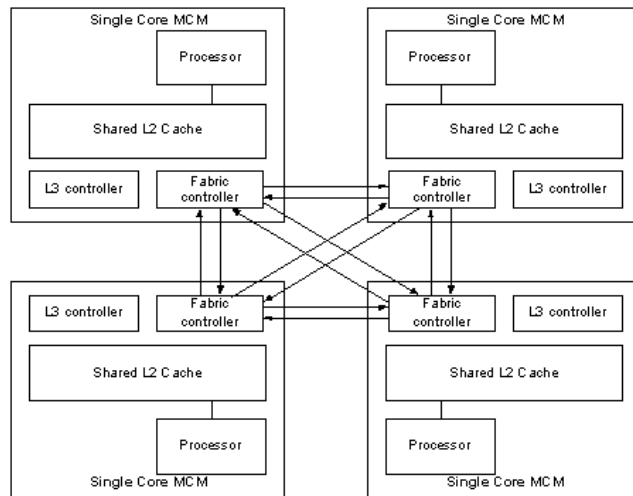


Рисунок 13.21 - Структурна схема процесорного блоку

13.8 Системи NUMA-Q

Багатопроесорні сервери IBM NUMA-Q складаються з окремих процесорних модулів. Кожен модуль має власну оперативну пам'ять і чотири процесори x86. Модулі називаються quad (четвірки) (Рисунок 6.4)

Четвірки сполучені високошвидкісними каналами IQ-Link з центральним комутатором. Заміна загальної шини на зіркоподібну топологію з центральним комутатором дозволяє вирішити проблеми арбітражу доступу до шини, зокрема, усунути затримки при запиті до арбітра шини і чеканні його відповіді запрошуючому пристрою. NUMA-системи фірми IBM можуть містити до 16 четвірок, тобто до 64 процесорів.

Архітектура дозволяє також включати в ці системи процесори з архітектурою, відмінною від x86, наприклад RS/6000 і System/390, дозволяючи, таким чином, створити в межах однієї машини гетерогенну мережу з надвисокошвидкісними каналами зв'язку.

При більшому числі модулів застосовуються ще складніші топології, наприклад гіперкубічна. У таких системах кожен вузол зазвичай також містить декілька процесорів і власну оперативну пам'ять (Рисунок 13.23).

При гіперкубічному з'єднанні, кількість вузлів N пропорційно міри двійки, а кожен вузол має $\log_2 N$ з'єднань з іншими вузлами. Кожен вузол здатний не лише обмінюватися повідомленнями з безпосередніми сусідами по топології, але і маршрутизувати повідомлення між вузлами, що не мають прямого з'єднання. Щонайдовша дорога між вузлами, що знаходяться в протилежних вершинах куба, має довжину $\log_2 N$ і не є єдиним (Рисунок 13.22). Завдяки множинності доріг, маршрутизатори можуть вибирати для кожного повідомлення найменш завантажену в даний момент дорогу або обходити вузли, що відмовили.

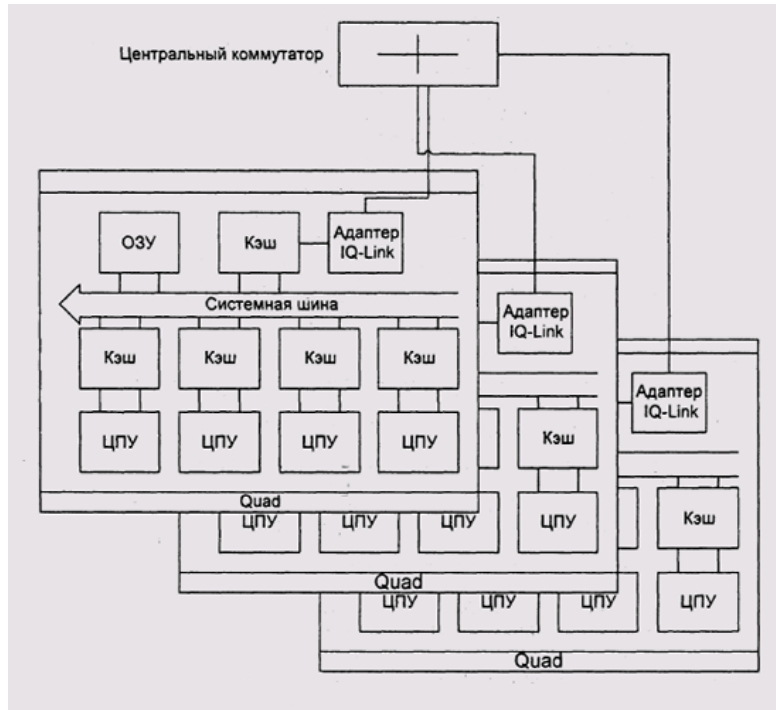


Рисунок 13.22 - . NUMA-Q з трьома чотирипроцесорними модулями

13.9 Масивно паралельні системи Cray/SGI Origin

Вузли суперкомп'ютерів сімейства Cray/SGI Origin сполучені в гіперкуб каналами з пропускною спроможністю 1 Гбайт/с. Адаптери з'єднань забезпечують не просто обмін даними, а прозорий (хоча і з падінням продуктивності) доступ процесорів кожного з вузлів до оперативної пам'яті інших вузлів і забезпечення когерентності процесорних кешів (рис. 13.23).

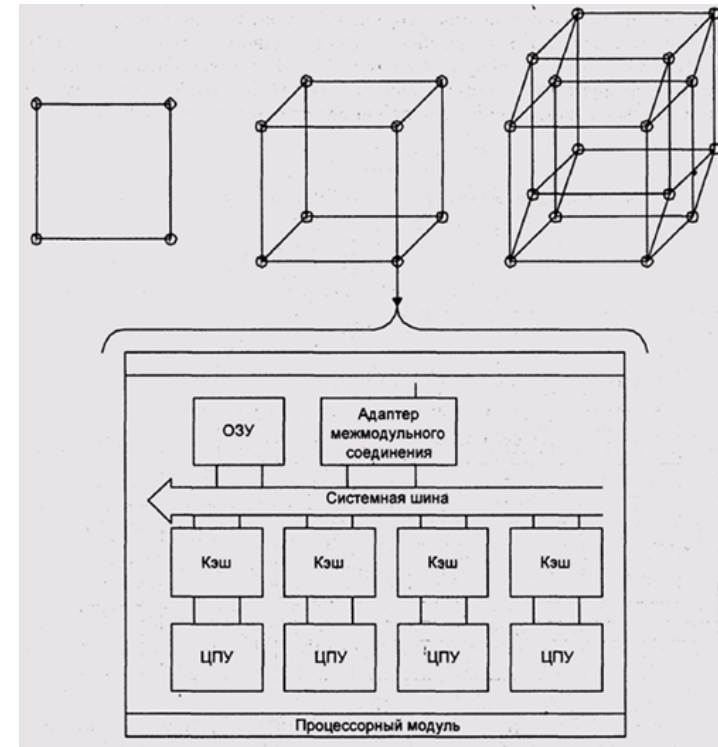


Рисунок 13.23 - Гіперкуби з 4, 8 і 16-у вершинами

Відмінність в швидкості доступу до локальної пам'яті процесорного модуля і інших модулів є проблемою, і при невдалому розподілі завантаження між модулями (такому, що міжмодульні звернення будуть часті) Приведе до значного зниження продуктивності системи. Відомо дві основні дороги пом'якшення цієї проблеми.

13.10 Архітектура COMA

COMA (Cache Only Memory Architecture) — архітектура пам'яті, при якій робота з нею відбувається як з кешем. Система переносить сторінки пам'яті, з якою даний процесорний модуль працює частіше за інших, в його локальну пам'ять (рис. 13.24).

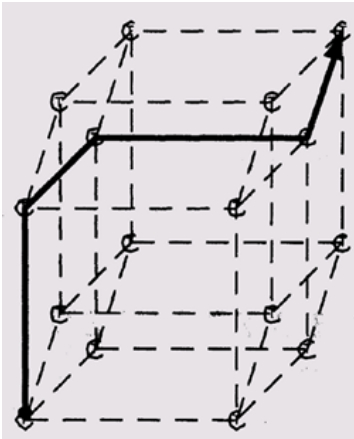


Рисунок 13.24 - Найдовша дорога в гіперкубі

CC-NUMA (Cache-Coherent Non-Uniform Memory Access неоднорідний доступ до пам'яті із забезпеченням когерентності кешів). У цій архітектурі адаптери міжмодульних з'єднань забезпечуються власною кеш-пам'яттю, яка використовується при зверненнях до ОЗП інших модулів. Основна діяльність центрального комутатора і каналів зв'язку полягає в підтримці когерентності цих кешів.

14 ОСОБЛИВОСТІ ПІДГОТОВКИ СКЛАДОВИХ ЧАСТИН КУРСОВОЇ РОБОТИ

14.1 Загальні вимоги до структури і змісту курсової роботи

Курсова робота за структурою і змістом повинна відповідати певним вимогам. Зазвичай курсова робота є самостійним закінченим науковим або технічним дослідженням задачі з схемотехніки або архітектури комп'ютера. Вона не обов'язково передбачає створення програмного продукту чи нового технічного рішення комп'ютерної системи. Зокрема, робота може бути спрямована на аналіз сучасного рівня або тенденцій розвитку конкретної області схемотехніки або архітектури комп'ютера, теоретичних чи прикладних розробки допомогою існуючих програмних інструментальних систем або власного програмного продукту.

Курсова робота складається з основної частини, обов'язкового графічного ілюстративного матеріалу (креслень, плакатів, які містять діаграми, графіки залежностей, таблиці, малюнки тощо). При захисті курсової роботи може використовуватись додатково демонстраційний матеріал в графічному, електронному (відеоматеріали, мультимедіа, презентації тощо) або натурному (моделі, макети, зразки пристроїв тощо) вигляді.

Наприклад, курсова робота може бути присвячена аналізу, моделюванню або прогнозуванню розвитку певних напрямів схемотехніки, удосконалення параметрів і характеристик пристроїв, теоретичному дослідженню методів оцінювання виробництва і збуту функціональних частин комп'ютерів або їх типів, методам аналізу і синтезу схем. При цьому можуть застосовуватись відомі програмні засоби Matlab, Statistica, Econometric Views (Eviews), VHDL тощо.

Тема роботи може бути чисто теоретичною і не потребувати використання стандартних програмних інструментальних засобів для її виконання. Наприклад, такою роботою може бути дослідження статистичних комерційних даних випуску певних схем, теоретична розробка нових схем або модифікації архітектур.

Умовно за тематикою курсові роботи із схмотехніки і архітектур комп'ютерів можна поділити на такі групи:

- спрямовані на поглиблений аналіз, моделювання цифрових схем, розробку нових схмотехнічних рішень;
- спрямовані на розробку нових математичних моделей, методів чи алгоритмів реалізації нових структур архітектур пристроїв та комп'ютерів;
- орієнтовані на моделювання та оцінювання параметрів і станів цифрових систем контролю і діагностування цифрових пристроїв;
- спрямовані на аналіз, вибір конкретних технічних рішень на основі фінансово-економічних та соціальних даних виробництва та ринку продажу вибраного класу пристроїв

Тематику курсової роботи визначає викладач з можливим корегуванням за пропозиціями студента.

14.2 Складання плану роботи

Попередній наблизений план (зміст) роботи необхідно написати відразу після узгодження теми курсової роботи. Наявність плану сприяє поглибленому розумінню головної мети та послідовності виконання курсової

роботи, значному прискоренню її виконання і підвищенню якості очікуваних результатів

14.3 Підготовка першого розділу основної частини роботи

Перший розділ (глава) курсової роботи дозволяє провести огляд існуючих схемних рішень, методів, алгоритмів їх проектування, визначити актуальність досліджень, інженерних рішень, проектних розробок тощо. Тому він визначає подальший напрямок подальших досліджень, вибір або створення цифрових пристроїв, математичних моделей, необхідність моделювання або проведення експериментальних досліджень тощо. Загальний обсяг першого розділу зазвичай не повинен перевищувати 20% основної частини курсової роботи.

При написанні огляду необхідно користуватись всією доступною спеціальною літературою, особливо літературою за останніх 10-15 років. Необхідно звернути особливу увагу на статті в журналах та препринти, які містять розробки за останні п'ять років (рекомендовано до 50 %) у вибраному напрямі, а також на надійні, достовірні джерела з мережі Інтернет.

Огляд повинен не тільки констатувати факт існування того чи іншого методу, але й містити його коротку його характеристику. Така характеристика має включати елементи:

- основні схемні рішення, співвідношення функціонування;
- клас задач, до розв'язання якого може бути застосований пристрій або клас комп'ютерів;
- відомі випадки схемних реалізацій та переваги їх застосування;
- недоліки та обмеження стосовно застосування;
- необхідно вказати також оцінити апаратну та програмну складність пристроя, порівняльний аналіз характеристик обчислювальних витрат.

Все це не вимагає багато часу у роботі, але сприяє отриманню чіткої стартової інформації стосовно напрямку, в якому необхідно рухатись. Критичний огляд повинен надати можливість автору курсової роботи визначити: необхідність подальших досліджень для досягнення поставленої мети: вдосконалення існуючих експериментальних рішень, моделей, методів чи алгоритму пристроя, створення нових розробок тощо. Крім того, коректно виконаний огляд сприяє розширенню загального світогляду автора та формуванню системного розуміння поставленої задачі; допомагає правильно сформулювати задачу і напрям подальших досліджень.

У більшості випадків для підтвердження достовірності отриманих результатів застосовують комп'ютерне моделювання. Обґрунтування дає можливість уникнути хибних шляхів та методів розв'язання поставленої задачі.

14.4 Загальні вимоги щодо оформлення курсової роботи

14.4.1 Нормативна документація

Нормативна документація визначає правила і загальні вимоги до оформлення завершених наукових і проектних робіт на основі існуючих стандартів і методичних рекомендацій НТУУ «КПІ» з метою поширення цих робіт для подальшого використання. Метою даної частини методичного посібника є подання основних з цих уніфікованих правил і вимог на основі діючих нормативних документів [1, 2, 3, 4]. Для більш повного ознайомлення з цими правилами і вимогами, а також для розв'язання менш типових ситуацій оформлення слід звертатись до самих документів [1, 2, 3, 4].

Порядок підготовки курсової роботи визначено нормативними документами НТУУ «КПІ» [1, 2]. Одним з етапів підготовки цих завершених робіт є їх оформлення згідно з існуючими стандартами.

Основним нормативним документом, який може бути застосований у процесі оформлення курсової роботи є стандарт України [3]. Бібліографічний опис використаних у роботі першоджерел повинен відповідати стандарту [4]. Характерні приклади застосування цього стандарту наведені у [5].

14.4.2 Загальні вимоги до структури і оформлення роботи

Курсова робота включає вступну частину, основну частину, додатки.

Вступна частина зазвичай містить такі структурні елементи: титульний аркуш; список авторів (для роботи кількох авторів); реферат; зміст; перелік умовних позначень, символів, одиниць, скорочень і термінів.

Основна частина зазвичай містить такі структурні елементи: вступ; суть роботи (зазвичай 3-5 розділів), з висновками до кожного розділу; висновки до роботи; рекомендації; перелік посилань (список використаних джерел, література). Обсяг основної частини роботи визначає характер магістерської дисертації.

Додатки містять ілюстративний матеріал і/або презентації, матеріали, які підтверджують окремі положення основної частини, порядок отримання результатів, результати експериментальних досліджень, протоколи випробувань тощо. Обсяг і форму подання додатків не обмежують.

Склад структурних елементів визначає структуру роботи і залежить від освітньо-кваліфікаційного рівня роботи - магістерська дисертація..

Зміст курсової роботи має відповідати її темі. Текст курсової роботи зазвичай подають державною або іноземною (іноземних студентів) мовою у друкованому вигляді на аркушах формату А4 шрифтом Times New Roman 14 пунктів, міжрядковий інтервал 1,5.

14.4.3 Структурні елементи вступної частини

Вступна частина містить:

титульний аркуш (Додаток Б);

завдання на курсову роботу (Додаток Б);

реферат українською іноземною мовами;

зміст;

перелік умовних позначень, символів, скорочень і термінів;

Основна частина містить:

розділи (глави), кількість яких зазвичай складає від трьох до п'яти;

висновки до кожного розділу;

загальні висновки до курсової роботи;

список використаних джерел

список джерел фактологічного матеріалу (за необхідності);

додатки (за необхідності, але з обов'язковим додатком презентації або з графічним пояснювальним матеріалом).

14.4.4 Обсяг курсової роботи

Орієнтовний обсяг курсової роботи складає до 30-50 сторінок (основна частина). У разі виконання курсової роботи кількома студентами комплексної теми можливо мати спільну частину курсової роботи, але наявність одноосібних частин є обов'язковою.

Курсову роботу необхідно оформлювати відповідно до стандарту [3] з неухильним дотриманням порядку подання окремих видів текстового матеріалу, таблиць, формул, ілюстрацій і списку використаних джерел.

15 ЗМІСТ СТРУКТУРНИХ КОМПОНЕНТІВ КУРСОВОЇ РОБОТИ

15.1 Титульний аркуш

Титульний аркуш править за основне джерело бібліографічної інформації, необхідної для оброблення та пошуку документа – курсової роботи.

15.2 Реферат

Реферат повинен містити [3]:

– відомості про обсяг роботи, кількість ілюстрацій, таблиць, додатків, кількість джерел згідно з переліком посилань (усі відомості наводять, включаючи дані додатків);

– текст реферату;

– перелік ключових слів, записаних великими літерами у називному відмінку через коми.

Слово **РЕФЕРАТ**, написане напівжирним шрифтом, розміщують по центру з нової сторінки. Ключові слова повинні відображати основний зміст, суть роботи і призначені для подальшого пошуку даної роботи зацікавленими особами у світі. Тому особливу увагу треба приділити правильному написанню теми роботи іноземною мовою.

Приклад оформлення реферату наведено у додатку Д.

Реферат курсової роботи обсягом до 0.5-1.0 сторінки українською іноземною мовами має містити:

- відомості про обсяг роботи, кількість ілюстрацій, таблиць, додатків, джерел за переліком посилань;

- текст реферату;

- ключові слова (від 5 до 15 слів).

Текст реферату повинен відображати зміст курсової роботи у такій послідовності:

– актуальність теми (сутність, стан розв'язування наукової проблеми, актуальність для розвитку відповідної галузі науки та виробництва, обґрунтування доцільності проведення досліджень);

– мета й завдання дослідження (запланований результат досліджень з виявлення нових фактів, висновків, рекомендацій, закономірностей, або уточнення відомих раніше, але недостатньо досліджених);

– об'єкт дослідження (процес, система, обладнання, пристрій, технологія, програмний продукт, інформаційна технологія, явище тощо, що породжує проблемну ситуацію і обране для дослідження);

– предмет дослідження (характеристики і властивості об'єкту, на які спрямовані дослідження);

– методи дослідження (з визначенням того, що саме досліджувалось тим чи іншим методом);

– наукова новизна одержаних результатів (анотація нових здобутків, одержаних студентом особисто);

– практичне значення одержаних результатів;

– апробація результатів курсової роботи (оприлюднення на семінарах, конференціях тощо);

– публікації (у збірниках наукових праць, матеріалах і тезах конференцій, патентах тощо).

Частини реферату, для яких відсутні дані, опускають.

15.3 Зміст

До змісту включають: перелік умовних позначень, символів, скорочень і термінів; вступ; перелік і назви всіх розділів, підрозділів, пунктів і підпунктів (якщо вони мають заголовки); висновки до розділів, висновки до курсової роботи; рекомендації; перелік посилань; назви додатків і номери сторінок, які містять початок матеріалу. У змісті можуть бути перелічені номери і назви ілюстрацій та таблиць із зазначенням сторінок, на яких вони вміщені.

15.4 Перелік умовних позначень, символів, скорочень і термінів

Перелік треба друкувати двома колонками, у яких слова за абеткою наводять, наприклад, скорочення, справа – їхню детальну розшифровку

Якщо умовні скорочення, спеціальні терміни, символи, позначення і таке інше зустрічаються менше трьох разів, перелік не складають, а їхню розшифровку наводять у тексті за першим згадуванням.

15.5 Розділи і підрозділи

Розділи і підрозділи повинні мати заголовки. Заголовки розділів розміщують посередині рядка і виділяють напівгрубим шрифтом. Заголовки підрозділів починають з абзацного відступу.

Кожний розділ починається із нової сторінки. Розміщення матеріалу роботи по розділах може бути доволі довільним, але з врахуванням змістовного навантаження по кожному розділу. Загальний обсяг першого розділу дисертації не повинен перевищувати 20% обсягу основної частини.

Наприкінці кожного розділу обов'язково формулюють висновки зі стислим викладенням основних наукових і практичних результатів розділу. Необхідно враховувати, що при цьому не слід переказувати те, що зроблено у розділі, а сформулювати те, що із результатів розділу випливає.

15.6 Пункти і підпункти

Пункти і підпункти можуть мати заголовки. Заголовки розміщують, починаючи з абзацного відступу. Абзацний відступ становить п'ять символів. Заголовки, що складаються з кількох речень розділяють крапками, але в кінці заголовку крапка не ставиться.

Відстань між заголовком і подальшим чи попереднім текстом повинен бути не менше, ніж два рядки. Відстань між рядками заголовку повинен бути тим же самим, що і в основному тексті, тобто 1,5 інтервали.

16 ОФОРМЛЕННЯ ГРАФІЧНО-ТЕКСТОВОЇ ІНФОРМАЦІЇ

16.1 Текст

16.1.1 Розміщення та виправлення тексту

Загальні вимоги до оформлення тексту, ілюстрацій, таблиць або їх сполучень визначає стандарт [3].

Текст роботи друкують, додержуючись таких розмірів берегів: верхній, лівий і нижній – не менше 20 мм, правий – не менше 10 мм.

Помилки, описки та графічні неточності допускається виправляти підчищенням або зафарбовуванням білою фарбою і нанесенням на тому ж місці або між рядками виправленого зображення машинописним способом або від руки.

Прізвища, назви установ, організацій, фірм інші власні імена наводять мовою оригіналу. Допускається транслітерувати власні назви і наводити назви організацій у перекладі на мову роботи, додаючи (при першій згадці) оригінальну назву.

16.1.2 Нумерація і розмежування текстів

Заголовки структурних елементів роботи і заголовки розділів розташовують посередині рядка, друкують великими літерами без крапки в кінці, не підкреслюють

Структурні елементи «РЕФЕРАТ», «ЗМІСТ», «ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ», «ПЕРЕДМОВА», «ВСТУП», «ВИСНОВКИ», «РЕКОМЕНДАЦІЇ», «ПЕРЕЛІК ПОСИЛАНЬ» не нумерують.

Розділи і підрозділи повинні мати заголовки і порядкову нумерацію арабськими цифрами без крапки в кінці. Пункти, підпункти можуть не мати заголовку і не відобразитись у змісті. Номер підрозділу складається з номера розділу і порядкового номера підрозділу. Пункти повинні мати порядкову нумерацію в межах кожного розділу або підрозділу, відокремлених крапкою, наприклад, 1.1, 1.3.1. Підпункти мають порядкову нумерацію у межах пунктів або розділу (при відсутності підрозділу), наприклад, 2.1.4, 3.1.2.1.

Підрозділи, пункти, підпункти починають з абзацного відступу і друкують маленькими літерами, крім першої великої. Абзацний відступ повинен дорівнювати п'яти знакам впродовж усього тексту роботи. Перенесення слів у заголовку розділів не допускається.

Відстань між заголовком і подальшим чи попереднім текстом повинен бути не менше ніж два рядки, а відстань між заголовками приймають такою ж, як у тексті.

16.1.3 Нумерація сторінок

Сторінки курсової роботи слід нумерувати арабськими цифрами, дотримуючись наскрізної нумерації. Номер сторінки проставляють у правому верхньому куті сторінки без крапки

16.2 Формули та рівняння

16.2.1 Розміщення формул і рівнянь

Формули та рівняння розташовують безпосередньо після тексту, в якому вони згадуються, посередині сторінки. Вище і нижче формули повинно залишатись не менше одного вільного рядка.

Пояснення значень символів і числових коефіцієнтів, що входять до формули або рівняння, наводять безпосередньо під формулою у тій послідовності, в якій вони наведені у формулі чи рівнянні.

Пояснення значення кожного символу та числового коефіцієнта слід давати з нового рядка. Перший рядок пояснення починають з абзацу словом «де» без двокрапки, а наступні рядки починають також з абзацу.

Переносити формули чи рівняння на наступний рядок допускається тільки на знаках виконуваних операцій, повторюючи знак операції на початку наступного рядка.

Формули, що йдуть одна за одною й не розділені текстом, відокремлюють комою [3].

16.2.2 Нумерація формул та рівнянь

Формули і рівняння нумерують арабськими цифрами порядковою нумерацією у межах розділу. Номер складається з номера розділу і порядкового номера формули або рівняння, відокремлених крапкою і проставляють у дужках. Наприклад, (2.3) – третя формула другого розділу.

Номер формули або рівняння розміщують у крайньому правому положенні на рядку на рівні формули або рівняння.

16.3 Ілюстрації

16.3.1 Розміщення ілюстрацій

Ілюстрації (креслення, рисунки, схеми, графіки, діаграми, фотознімки) розміщують безпосередньо після тексту, де вони згадуються, або на наступній сторінці [3].

Ілюстрації можуть мати назву та пояснювальні дані. Ілюстрацію позначають словом «Рисунок__» і розміщують по центру після пояснювальних даних (за їх наявності), наприклад «Рисунок 3.1 – Схема розміщення».

16.3.2 Нумерація ілюстрацій

Ілюстрації нумерують арабськими цифрами порядковою нумерацією у межах розділу. Номер складається з номера розділу і порядкового ілюстрації, відокремлених крапкою. Наприклад, «Рисунок 2.3» – третя ілюстрація другого розділу.

Номер і назву ілюстрації розміщують по центру після ілюстрації.

16.4 Таблиці

16.4.1 Розміщення та оформлення простих таблиць

Таблиці розміщують безпосередньо після тексту, де вони згадуються, або на наступній сторінці. Таблиця має номер і може мати назву, яку друкують малими літерами (крім першої великої).

Наприклад,

Таблиця 2.3 – Завантаженість процесора

Горизонтальні та вертикальні лінії, які розмежовують рядки таблиці, а також лінії справа, зліва і знизу, що обмежують таблицю, можна не проводити.

Підзаголовки таблиці пишуть у однині, починають з великої літери, крім випадків, коли вони складають одне речення з заголовком.

16.4.2 Розміщення та оформлення таблиць, розділених на частини

Якщо таблиця не вміщується у межі формати сторінки, то її розділяють на частини. Частини розміщують одна над однією, поруч, або на наступній сторінці, повторюючи у кожній частині таблиці її головку і боковик.

При поділі на частини допускається її головку або боковик замінювати відповідними номерами граф чи рядків, нумеруючи їх арабськими цифрами у першій частині таблиці.

Слово «Таблиця__» вказують зліва над першою частиною таблиці, а для решти частин пишуть «Продовження таблиці __»

16.4.2 Нумерація таблиць

Таблиці нумерують арабськими цифрами порядковою нумерацією у межах розділу. Номер складається з номера розділу і порядкового таблиці, відокремлених крапкою. Наприклад, «Таблиця 2.3» – третя таблиця другого розділу.

16.5 Посилання

Посилання в тексті роботи на джерела слід зазначати порядковим номером за переліком посилань, виділеним двома квадратними дужками, наприклад, «у роботах [1 – 7] ...».

При посиланнях на розділи, підрозділи, пункти, підпункти, ілюстрації, таблиці, формули, рівняння, додатки зазначають їх номери. При посиланнях слід писати [3]: «... у розділі 4...», «...дивись 2.1 ...», «...на рис. 1.3 ...» або «...на рисунку 1.3 ...», «...у таблиці 3.2 ...»,

«...за формулою (3.1) ...», «...у рівняннях (1.23)-(1.25) ...», «...у додатку Б...».

16.6 Переліки

Переліки можуть бути наведені всередині пунктів або підпунктів. Перед переліком ставлять двокрапку. Перед кожною позицією переліку слід ставити малу літеру української абетки з дужкою, або, не нумеруючи – дефіс (перший рівень деталізації).

Для подальшої деталізації переліку слід використовувати арабські цифри з дужкою (другий рівень деталізації).

Приклад

Клітини:

- а) форма і розмір клітин;
 - 1) частина клітини;
 - 2) неживі включення протопластів;
- б) живий склад клітин;
- в) утворення тканини.

Переліки першого рівня деталізації друкують малими літерами з абзацного відступу, другого рівня – з відступом відносно місця розташування переліків першого рівня.

16.7 Примітки

Примітки призначені для пояснення змісту тексту, таблиці, ілюстрації і розміщують після тексту, таблиці, ілюстрації. Одну примітку не нумерують, декілька приміток нумерують. Примітку розміщують з абзацного відступу.

Приклад

Примітка. _____

Приклад

Примітки:

1. _____

2. _____

16.8 Додатки

Додатки вміщують матеріал, який:

- є необхідним для повного відображення курсової роботи;
- не може бути включеним у основну частину, оскільки порушує обмеження за обсягом основної частини, міняє впорядковане й логічне уявлення про роботу;
- не може бути розміщений через великий обсяг або спосіб відтворення (наприклад, за форматом);
- є призначеним для фахівців даної галузі і не впливає на розуміння основної частини широким колом читачів.

Додатки не мають строгих обмежень оформлення, як для основної частини. Але нумерація розділів, підрозділів, пунктів, підпунктів, формул, таблиць, рисунків додатку починають з символічного позначення додатка. Наприклад, Рисунок А.1.2, Таблиця Б.4.2.

У додатки можуть бути включені:

- перелік плакатів або слайдів виступу при захисті курсової роботи;
- описи комп'ютерних програм;
- додаткові ілюстрації і таблиці;

- матеріали, які мають великий обсяг і не можуть бути включені до основної частини роботи (протоколи, фотографії, висновки комісії, тексти комп'ютерних програм, інструкції тощо);

- додатковий перелік джерел для фахівців;

- опис нової апаратури, схеми оброблення даних для проведення експериментів тощо.

Перед додатком, який має самостійне значення, вміщують аркуш з текстом ДОДАТОК Літера з назвою (за наявності).

Перед номером посилання або виноска вказують літеру додатка з розділовою крапкою між номером. Наприклад, у таблиці Б.4.

16.9 Посилання на першоджерела

16.9.1 Посилання на загальний перелік джерел

Посилання в тексті на джерела загального переліку джерел зазначають порядковим номером джерела у квадратних дужках і, за необхідності, з номером сторінок. Наприклад, [12], [4-7], [5, С. 21-32], [17, С.76].

16.9.2 Посилання на джерела у межах сторінки

Посилання виконують за джерелами у виносках на сторінці. Виноска містить нумерований бібліографічний опис джерел, а посилання вказує на цей номер. Наприклад, «загальний алгоритм оброблення даних наведено у [4]*». У бібліографічному описі джерела, наприклад, можуть бути вказані окремі сторінки, на яких міститься необхідний матеріал.

16.10 Виноски

Виноски на сторінці дозволяють наводити окремі дані, бібліографічний опис джерел, пояснення по окремих даних, наведених у таблицях, рисунках тощо.

Виноски позначають порядковими арабськими цифрами з дужкою в межах окремої сторінки. Номер виноска у тексті вказують після відповідних наведених даних (після слова, числа, символу, речення, таблиці тощо), до яких дають пояснення з виноски.

Текст виноски вміщують під таблицею або в кінці сторінки й відокремлюють від таблиці або основного тексту сторінки лінією довжиною 30-40 мм у лівій частині сторінки.

Текст виноски починають з абзацного відступу, друкують через один інтервал або з мінімальним міжрядковим інтервалом. Перед текстом виноски вміщують номер виноски.

Наприклад, у основному тексті «у роботі [7]* наведені дані експерименту». У виносці

*[7]

16.11 Запитання і завдання для самоконтролю

1. Якими є вимоги до оформлення тексту курсових робіт?
2. Що є структурними елементами дисертації і яким шрифтом їх оформляють ?

3. Як виконують нумерацію формул та рівнянь ?
4. Як виконують нумерацію таблиць ?
5. Як розділяють таблиці на частини ?
6. Як виконують нумерацію рисунків (ілюстрацій) ?
7. Як розділяють рисунки на частини ?
8. Якими є вимоги до оформлення нумерованих і нелюерованих переліків ?
9. Як оформляють примітки на сторінках ?
10. Який матеріал за обсягом і змістом вкочають у додатки ?
11. Як нумерують додатки ?
12. Як нумерують розділи, підрозділи, рисунки, таблиці, формули додатків ?
13. Як оформлюють примітки на сторінці ?
14. Як виконують виноски на сторінці ?
15. Як посилаються на загальний перелік джерел, на сторінки джерела ?

17 ОФОРМЛЕННЯ БІБЛІОГРАФІЧНИХ ОПИСІВ

17.1 Області опису документів

Бібліографічний опис документів у посиланнях в курсовій роботі виконують з використанням стандарту ДСТУ 7.1:2006 [4]. Стандарт описує універсальні правила, спільні для всіх видів опублікованих та неопублікованих документів на будь-яких носіях: книг, періодичних та продовжуючих видань, нотних, картографічних, аудіовізуальних, образотворчих, нормативних та технічних документів, депонованих рукописів, мікроформ, електронних ресурсів; складової частини документів; групи однорідних та різнорідних документів.

ДСТУ 7.1:2006 покликаний забезпечити впровадження сучасних автоматизованих технологій опрацювання документів, ведення інформаційних баз даних; ефективність пошуку та використання документів всіх видів та типів; результативний обмін бібліографічною інформацією між інформаційними службами, бібліотеками, видавцями та книготорговельними організаціями як в середині країни, так і за кордоном.

Всі види документів описуються із застосуванням восьми областей бібліографічного опису. До складу яких входять :

- область назви і відомостей про відповідальність;
- область видання;
- область специфічних відомостей;
- область віхідних відомостей;
- область фізичної характеристики;
- область серії;
- область приміток;

- область стандартного номера (чи його альтернативи) та умов доступності.

Кожна з цих областей складається з елементів, які поділяються на обов'язкові та факультативні. У бібліографічному описі можуть бути тільки обов'язкові чи обов'язкові та факультативні елементи. Обов'язкові елементи містять бібліографічні відомості, які забезпечують ідентифікацію документа, їх наводять у будь-якому описі. Необхідність застосування та набір факультативних елементів визначається окремо.

Приклади бібліографічного опису при написанні курсової роботи подано у додатку В для характерних випадків їх застосування.

17.2 Опис елементів бібліографічного опису

До обов'язкових елементів бібліографічного опису відносять такий перелік:

- перші відомості про відповідальність в усіх областях (області назви та відомостей про відповідальність, області видання, області серії);
- додаткові відомості про видання;
- ім'я видавця, розповсюджувача тощо;
- основна назва серії та підсерії;
- міжнародний стандартний номер серіального видання, що було надано серії чи підсерії (ISSN);
- номер випуску серії чи підсерії;

- окремі примітки в описі певних видів документів (в описі електронних ресурсів — примітки про джерело основної назви, примітки про системні вимоги).

Заголовок від опису відокремлюють крапкою. Області опису відокремлюють одна від одної крапкою і тире. При повторенні окремих областей повторюють крапку і тире, за винятком області серії (відомості про кожну серію беруться в окремі круглі дужки без знаку крапка і тире між ними).

Відомості, що запозичені не з приписного джерела інформації, наводять у квадратних дужках (уклад. В. Петренко [ін.] або [б. м.] (без місця) ін). Необхідно зазначити, що квадратні дужки застосовуються у межах однієї області. Якщо суміжні елементи відносяться до різних областей, то кожен елемент береться в окремі квадратні дужки.

Відповідно до стандарту ДСТУ 7.1:2006, для розрізнення приписної та граматичної пунктуації застосовують проміжок в один друкований знак до і після приписного знака. Проміжки між знаками та елементами опису є обов'язковими. Виняток — крапка та кома — проміжки залишають тільки після них. Знаки крапка з комою та три крапки до винятку не відносяться.

В області назви та відомостей про відповідальність використовують елемент — загальне позначення матеріалу. Це факультативний елемент, який доцільно зазначати в описі для інформаційних масивів, що вміщують відомості про документи різних видів.

Загальне позначення матеріалу приводиться після основної назви у квадратних дужках з прописної (великої) літери.

[Текст]

[Електронний ресурс]

[Відеозапис]

[Звукозапис]

[Ізоматеріал]

[Карти]

[Комплект]

[Кінофільм]

[Мікроформа]

[Мультимедія]

[Ноти]

[Рукопис]

Великі літери у бібліографічному описі застосовують відповідно до сучасних правил граматики тієї мови, на якій складений бібліографічний опис, незалежно від того, які букви спожиті в джерелі інформації. З великої літери починають перше слово кожної області, а також перше слово наступних елементів: загального позначення матеріалу ([Текст], [Електронний ресурс], [Карта] ін) і будь-яких заголовків у всіх областях опису. Решту всіх елементів записують з маленької букви. Зберігають великі і малі літери тільки в офіційних найменуваннях сучасних організацій і інших іменах власних.

Особливу увагу необхідно приділяти відомостям про відповідальність. Якщо опис доповнюється заголовком бібліографічного запису, ім'я особи в заголовку наводять у формалізованому вигляді: спочатку прізвище, потім ім'я (можливо ім'я та по батькові) або псевдонім. Тільки у відомостях про відповідальність є можливість зазначити, в якому вигляді особа, що несе інтелектуальну чи іншу відповідальність за документ, представлена в документі.

Перші слова, що відносяться до назви та відомостей про відповідальність записуються з малої літери, якщо вони не є власними назвами, першими словами назви чи цитатами.

Відомості про видання наводять у формулюванні та послідовності, зазначеній у джерелі інформації. Додаткові відомості про видання (виправлене, стереотипне, перероблене тощо) та перші відомості про відповідальність, що відносяться до конкретного зміненого видання твору, є обов'язковими елементами.

Область вихідних відомостей пов'язана з формою представлення відомостей про місце видання, ім'я видавця: їх слід наводити у формі та відмінку, зазначених у джерелі інформації, а не лише в називному відмінку. Відомості про видавця мають статус обов'язкового елемента.

Рік видання, навіть тоді, коли відомості про нього відсутні в документі, повинен бути встановлений хоча б приблизно. В таких випадках дату видання наводять у квадратних дужках разом зі знаком запитання, наприклад: [2007?]. Позначення "б. р." (без року) в описі не наводять.

В області фізичної характеристики використовують елемент — специфічне позначення матеріалу. Після відомостей про кількість фізичних одиниць зазначають позначення фізичного носія документа, наприклад:

1 електрон. опт. диск

1 папка (24 окр. арк.)

Обов'язковими елементами області серії є: основна назва серії та підсерії; Міжнародний стандартний номер серіального видання (ISSN); номер випуску серії та підсерії. Для декількох серій відомості про кожну серію беруться у круглі дужки та відокремлюються проміжком.

Область приміток в цілому факультативна, але під час опису деяких об'єктів, окремі примітки є обов'язковими: примітки про джерело основної назви, про системні вимоги в описі електронних ресурсів, відомості про депонування в описі депонованої наукової роботи.

17.2 Правила подання елементів бібліографічного опису

17.2.1 Застосування літер і спеціальних знаків

У списку джерел з маленької літери пишуть відомості, що відносяться до заголовку (підруч. для вузів, матеріали конф., тези, навчально-методичний посібник тощо), відомості про відповідальність (ред., упоряд., редкол. тощо.), наприклад:

Психологія : підруч. для вузів.

Психология : словарь / отв. ред. Гончарук П. В.

У стандарті [4] застосовують проміжок в один друкований знак до і після приписного знака: тире (–), скісна риска (/), дві скісні (//), двокрапка (:). Виняток – крапка (.) та кома (,) – проміжки залишають тільки після них.

17.2.2 Посилання на авторів

Запис реквізитів статті одного автора з періодичного друкованого видання матиме наступний вигляд:

Прізвище ініціали автора. Назва статті. / ім'я, по батькові автора або ініціали і прізвище автора // Назва журналу. – Рік. – № . – С. ?–?.

Наприклад:

Волинець І. М. Краєзнавчі матеріали зарубіжної літератури / І. М. Волинець // Світло. – 2002. – № 4. – С. 112–116.

Для запису статті двох і більше авторів використовують такий опис:

Прізвище ініціали першого автора. Назва статті. / ініціали прізвище першого автора, ініціали, прізвище другого автора // Назва журналу. – Рік. – № . – С. ?–?.

У заголовках творів одного, двох і трьох авторів зазначається ім'я першої особи, яке обов'язково повторюється у зоні відповідальності за видання у точності до форми запису на титульному аркуші.

Наприклад:

Коваленко А.Є. Розподілені інформаційні системи [Текст] : навч. посіб. / А.Є.Коваленко.- К. : НТУУ «КПІ», 2008. – 244с. – Бібліогр. : С. 231-232. – ISBN 978-966-622-280-3

Відомості, не зазначені на титулі, наводяться у квадратних дужках (крім зони ISBN), напр.: / [голов. ред. В. Пилипенко] [б. м.] (без місця)

17.2.3 Електронні видання

Оформлення статті з електронного видання має таку форму опису:

Прізвище ініціали автора. Назва статті [Електронний ресурс] / ім'я, по батькові автора або ініціали прізвище автора // Назва журналу. – Рік. – № . – Режим доступу: електронна адреса, за якою розміщена стаття <http://www...>

Наприклад:

Кабан Л. В. Оцінювання інноваційної діяльності загальноосвітніх навчальних закладів регіону [Електронний ресурс] / Лариса Василівна Кабан // Народна освіта. – 2007. – Випуск 1. – Режим доступу: <http://www.narodnaosvita.kiev.ua/vupysku\1\statti\2kaban\2kaban.htm>

Загальне позначення матеріалу наводять після основної назви з великої літери у квадратних дужках, наприклад:

Неопубліковані гуцульські п'єси [Текст]

Богородиця з дитям і похвалою [Образотворчий матеріал]

Антологія лемківської пісні [Ноти]

Житомирська область [Карти]

Україна 2004: події, документи, факти [Електронний ресурс]

Якщо твір розміщено на декількох носіях, що відносяться до різних категорій матеріалів, наводять загальне позначення матеріалів, прийнятих за основний об'єкт опису. Відомості про решту носіїв можна навести в області фізичної характеристики чи в області приміток (наприклад, при описі CD-ROMа і брошури до нього).

Якщо серед декількох об'єктів на різних носіях не можна вибрати основний об'єкт, наводять позначення "[Мультимедіа]" чи "[Комплект]", наприклад, інформація може бути розміщена на аудіокасеті, відеодиску та у вигляді короткого пояснювального тексту в брошурі.

В описі електронних ресурсів в області виду й обсягу ресурсу зазначають вид ресурсу і відомості про його обсяг, наприклад: Москва 2004 [Электронный ресурс] : електрон. бизнес-карта ; Санкт-Петербург, 2004 : електрон. бизнес-карта. — Электрон. граф. дан. и прогр. (560 Мб). — К. : Транснавіком, 2004. — 2 електрон. опт. диски (CD-ROM). — Систем. требования: Pentium-166 ; RAM 64 Mb ; SVGA 2Mb ; CD-ROM 4-x ; Windows 98, ME, 2000 XP. — Загл. с титул. экрана. — На контейнере номер: X35U6H-74AMKC-SCWC3T-CJUS24. — MCP4LZ-H85R5Z-FTG8MQ.

17.2.4 Паралельні назви і відомості до назв

В описі може бути наведено паралельну назву як еквівалент основної назви іншою мовою чи іншою графікою. Вона має ті самі форми і правила наведення, що й основна назва. Перед паралельною назвою ставлять знак рівності. Як паралельну назву можна навести також назву оригіналу, що вміщена у джерелі інформації іншою мовою, ніж основна назва.

Наприклад (з врахуванням вживання великої та малої літер):

Основна назва [Загальне позначення матеріалу] = Паралельна назва : відомості, які відносяться до назви / відомості про Авторство чи Відповідальність ; про інших Осіб. – Відомості про повторність видання / Відповідальність за видання. – Зона специфічних відомостей. – Місце видання : Вид-во, рік. – Фізична (кількісна) характеристика. – (Серія і підсерія ; №, т.). – Примітки (додаткова інформація від бібліографа, напр.: системні вимоги до електрон. ресурсів). – ISBN.

Перед відомостями, що відносяться до назви, ставлять знак "двокрапка". Відомості, що відносяться до назви, можуть бути різнорідними або однорідними.

Різнорідні відомості чи групи різнорідних відомостей, що відносяться до назви, наводять зі знаком "двокрапка", що передує їм. Однорідні відомості розділяють між собою тими самими розділовими знаками, які наявні у джерелі інформації, наприклад:

Україна — Німеччина : розвиток законодавства в рамках європейського права : матеріали семінару

Однак, якщо у джерелі інформації однорідні відомості, що відносяться до назви, розділені крапками, в описі їх наводять через кому. Це пов'язано з новими правилами вживання малих і великих букв. За відсутності розділових знаків у джерелі інформації ці відомості також розділяють комами, наприклад:

В країні журавлів : новели, етюди, оповід.

Редакторська справа : проблеми майстерності: зміст, форма, нюанси

У тому випадку, якщо у джерелі інформації уміщено більше однієї назви, одну з них, вибрану за визначеними правилами (тематичним, виділенням якимось чином, наведе ним першим) наводять в описі як основну назву, а другу — як відомості, що відносяться до назви. Другу

назву завжди наводять з великої літери (на відміну від інших відомостей, що відносяться до назви), і слова в ній не скорочують. наприклад:

За сестрою [Текст] : Козацька помста

Ясна зірниця світова [Ноти] : Богданів марш

Зоряниця [Образотворчий матеріал] : (Язицька Зірка—Чарівниця)

Відомості, необхідні для розкриття чи пояснення основної назви, жанру твору, виду документа тощо, можуть бути сформульовані на основі аналізу документа. В цих випадках їх беруть у квадратні дужки.

17.2.5 Відомості про відповідальність

Відомості про відповідальність уміщують інформацію про осіб та організації, що брали участь у створенні інтелектуального, художнього чи іншого змісту твору, що є об'єктом опису, наприклад:

Ф. Б. Керр ; пер. з англ. В. Немченко

В. Ф. Салабай, Н. М. Довганик, М. В. Борисенко, М. П. Чуб ; Київ. нац. екон. ун-т ім. В. Гетьмана.

В описі не зазначають підпорядкованість організації. Структурні підрозділи відділялись від назви основної організації комою, наприклад

НАН України, Ін-т історії України, НДІ козацтва ; редкол.: В. А. Смолій (відп. ред.).

Крім цього, відомості про відповідальність можуть уміщувати тільки слова і фрази, що несуть інформацію про виконану роботу, якщо у джерелі інформації нема осіб чи назв організацій та якщо їх не вдалося установити, наприклад: / відредаговано автором / ілюстровано групою художників
Якщо відомості про відповідальність наводять із одного джерела інформації, їх записують у тому порядку, як і в ньому, якщо відомості про відповідальність запозичені із різних джерел інформації, їх наводять у

логічному порядку: спочатку імена осіб чи назви організацій, що зробили най більший внесок в інтелектуальніший, художній чи інший зміст твору, потім відомості про інших осіб чи інші органи зації, наприклад:

В. В. Івата, С. А. Ткаченко, С. В. Шевчук ; ред. В. Д. Пантелєєв ; Нац. ун-т кораблебудування ім. адмірала Макарова Оксана Іваненко ; [упоряд. В. Татаринова] ; худож. О. Кошель

17.2.6 Область видання

Зміни, що стосуються області видання, полягають у тому, що відомості про нього наводять у формулюваннях і в послідовності, наявних у джерелі інформації. Додаткові відомості про видання (виправлене, доповнене, стереотипне, змінене тощо) і перші відомості про відповідальність, що відносяться до конкретного зміненого видання, є обов'язковими елементами, наприклад:

2-ге вид., переробл. та доповн.

Нове вид. / перегл. та анот. Сільвією Мезюр.

Після області видання в описі уміщується область, що має назву "область специфічних відомостей", яка застосовується при описі об'єктів, що є особливим типом публікації або розміщені на специфічних носіях. До них відносяться картографічні, нотні, серіальні документи; стандарти і технічні умови; патентні документи, мікроформи, якщо на них розміщені всі названі види документів, а також електронні ресурси.

17.2.7 Опис нормативних документів і стандартів

В описі нормативних документів із стандартизації (стандартів і технічних умов) в області специфічних відомостей вказують позначення

раніше чинного документа, дату набуття ним чинності, терміни дії об'єкта бібліографічного опису, наприклад:

Національна стандартизація. Правила розроблення, по будови, викладання, оформлення, ведення національних класифікаторів [Текст] : ДСТУ 1.10:2005. — На заміну ДСТУ 3456—96, КНД 50—028—94 ; чинний від 2006-01-01. — К. : Держспоживстандарт України, 2006. — IV, 16 с., включ. обкл. : табл. ; 29 см. — (Національний стандарт України).

В описі патентних документів в області специфічних відомостей зазначають реєстраційний номер заявки на патентний документ, дату її подання, дату публікації та відомості про офіційне видання, в якому оприлюднено відомості про патентний документ; так звані відомості про конвенційний пріоритет (дату подання заявки, номер і назву країни конвенційного пріоритету, які наводять у круглих дужках). Також в області можуть бути зазначені індекси національної патентної класифікації, наприклад:

Холодильник однокамерний без морозильної камери по побутовий [Текст] : патент 14709 : МКЗ 07-07 / Горін О. М., Волощенко О. В., Чуріл О. О. ; власник патенту Закрите АТ "Укр. наук.-дослід. і проект. ін-т побут. машинобудування". — № 200601173 ; заявл. 26.07.06 ; опубл. 10.08.07, Бюл. № 12 (кн. 2). — 2 с. : іл.

17.2.8 Відомості про місце видання

Відомості про місце видання, ім'я видавця потрібно наводити у формі й відмінкові, зазначених у джерелі інформації, а не тільки у називному відмінку.

Назву місця видання наводять у формі й відмінкові, зазначених у джерелі інформації, наприклад:

Львів

У Кіровограді

Якщо зазначено декілька місць видання, наводять назву, виділену поліграфічним способом чи вказану першою в джерелі інформації. Випущені відомості відзначають скороченням "[тощо., і ін.]". Можуть бути наведені назви другого й наступного місць видання, що відділяються одна від іншої крапкою з комою, наприклад:

К. [ін.]

К. ; Львів

17.2.9 Відомості про видавця

Відомості про видавця наводять так, як вони зазначені в приписаному джерелі інформації, зберігаючи слова або фрази, що вказують функції, виконувані особою чи організацією. Відомості про видавничі функції організації виражені словами "видавництво", "видавничий дім", "видавнича організація", "видавець" тощо випускають при наявності тематичної назви, але зберігають, якщо ім'я (найменування) видавця і ці слова граматично пов'язані.

Відомості про форму власності видавця, розповсюджувача тощо (АТ, ВАТ, ТОВ, ЗАТ, LTD, Inc тощо), зазвичай випускають, наприклад:

У джерелі інформації: В описі:

Видавництво "Дніпро" : Дніпро

ЗАТ "Броварська друкарня": Броварська друкарня

Видавничий дім "Всесвіт": Всесвіт

Видавництво ім. Олени Теліги : Вид-во ім. Олени Теліги

Ім'я (найменування) видавця тощо наводять у короткій формі, яка забезпечує його розуміння та ідентифікацію. Якщо воно увійшло до

попередньої області у повній формі, в цій області його можна скоротити до найкоротшої форми, аж до акронімної, наприклад:

Львів : Укр. акад. друкарства

Х. : РЦНГТ

Якщо видавець — фізична особа, в описі наводять його прізвище ініціали у формі й відмінку, зазначених у джерелі інформації, наприклад:

К. : Яновська М.А.

Львів : О.В. Богданова

Найменування видавничого філіалу наводять після імені (найменування) видання й відділяють комою, наприклад:

Донецьк : Поліпрес, дочір. п-во

У разі наявності декількох груп відомостей, що включають місце видання та ім'я (найменування) видавця, що відносяться до нього, їх зазначають послідовно й відділяють один від одного крапкою із комою, наприклад:

К. : Генеза ; Запоріжжя : Прем'єра

Якщо в документі відсутні відомості про видавця, що є обов'язковим елементом бібліографічного опису, наводять скорочення "[б. в.]", що також обов'язкове.

На доповнення до відомостей про видавця в описі може бути зазначений новий елемент області вихідних даних — відомості про функції розповсюджувача, дистриб'ютора тощо, що є факультативними. Пояснення названих функцій, якщо вони відсутні в попередніх відомостях області, наводять у квадратних дужках, наприклад:

К. : Арт-Пресс : Дніпро-Ант [розповсюджувач]

17.2.10 Опис фізичних характеристик і супроводжувачий матеріал

Як інші фізичні характеристики об'єкта опису можуть бути наведені відомості про ілюстрації, про матеріал, з якого виготовлено об'єкт опису. Перед відомостями ставлять двокрапку. Кожні наступні відомості відділяють від попередніх комою, наприклад:

XXI, 117, [3] арк. іл.

3 електрон. опт. диски (CD-ROM) : звук, колір

1 зв. диск (120 хв) : цифровий, стерео

У разі необхідності зазначають розміри об'єкта опису, перед якими ставлять знак крапка з комою.

Останнім елементом області є відомості про супроводжувальний матеріал, перед яким ставлять знак плюс. Арабськими цифрами зазначають кількість фізичних одиниць, назву супроводжувального матеріалу, а також відомості про його обсяг і розмір, наприклад:

56 с. : іл. + 1 електрон. опт. диск

2 електрон. опт. диска (CD-ROM) : 12 см + 1 брош. (5 с. ; 13 см)

17.2.11 Опис стандартних номерів

В області стандартного номера (чи його альтернативи) та умов доступності наводять Міжнародні стандартні номери, присвоєні об'єкту опису: Міжнародний стандартний номер книги (ISBN) чи Міжнародний стандартний номер серіального видання (ISSN), або ж будь-який інший міжнародний номер, присвоєний об'єкту опису в установленому порядку. Стандартні номери наводять з прийнятою аббревіатурою і приписаними проміжками й дефісами, наприклад:

ISBN 966-521-406-3

Коваленко А.Є. Операційні системи [Текст] : навч. посіб. / А.Є.Коваленко. — К. : НТУУ «КПІ», 2010. — 248с. — Бібліогр. : с. 248. — ISBN 978-966-622-321-3

17.3 Приклади повного оформлення бібліографічного опису

1. Коренівський Д. Г. Дестабілізуючий ефект параметричного білого шуму в неперервних та дискретних динамічних системах / Коренівський Д. Г. ; НАН України, Ін-т математики. — К. : Ін-т математики, 2006. — 111 с. — (Математика та її застосування) (Праці / Ін-т математики НАН України ; т. 59). — Бібліогр.: с. 97—106 (93 назви) та в підрядк. прим. — ISBN 966-02-3964-5.

2. Суберляк О. В. Технологія переробки полімерних та композиційних матеріалів : підруч. [для студ. вищ. навч. закл.] / О. В. Суберляк, П. І. Баштанник ; М-во освіти і науки України, Ін-т інновац. технологій і змісту освіти. — Львів : Растр-7, 2007. — 375 с. : іл., табл., портр. — Бібліогр.: с. 358—362. — ISBN 978-966-2004-01-4.

3. Межгосударственные стандарты : каталог : в 6 т. / [сост. Ковалева И. В., Рубцова Е. Ю. ; ред. Иванов В. Л.]. — Львов : НТЦ "Леонорм-Стандарт", 2005— . — (Серия "Нормативная база предприятия"). — ISBN 966-7961-45-1.- Т. 1. — 2005. — 277 с. — ISBN 966-7961-46-Х.

4. Бондаренко В. Г. Теорія ймовірностей і математична статистика. Ч.1 / В. Г. Бондаренко, І. Ю. Канівська, С. М. Парамонова ; Нац. техн. ун-т України "Київ. політехн. ін-т". — К. : НТУУ "КПІ", 2006. — 125 с. : іл.

5. Кібернетика в сучасних економічних процесах : зб. текстів виступів на республік. міжвуз. наук.-практ. конф. / Держкомстат України, Ін-т статистики, обліку та аудиту. — К. : ICOA, 2002. — 147 с. : іл., табл. — ISBN 966-8059-08-5.

6. Проблеми обчислювальної механіки і міцності конструкцій = Problems of mechanics and strength of structures : зб. наук. пр. / наук. ред. В. І. Моссаковський. — Дніпропетровськ : Навч. кн., 1999. — 215 с. : іл., табл. — Текст: укр., рос. — Бібліогр. в кінці ст. — ISBN 966-7056-81-3.

7. Українсько-німецький тематичний словник = Ukrainisch-deutsches thematisches Wörterbuch : [близько 15 000 термінів / уклад. Н. Яцко ін.]. — К. : Карпенко, 2007. — 219 с. — ISBN 966-8387-23-6.

8. Кримінально-процесуальний кодекс України : за станом на 1 груд. 2005 р. / Верховна Рада України. — Офіц. вид. — К. : Парлам. вид-во, 2006. — 207 с. — (Бібліотека офіційних видань). — ISBN 966-611-412-7.

9. Нгуен Ші Данг. Моделювання і прогнозування макроекономічних показників в системі підтримки прийняття рішень управління державними фінансами : автореф. дис. на здобуття наук. ступеня канд. техн. наук : спец. 05.13.06 "Автоматиз. системи упр. та прогрес. інформ. технології" / Нгуен Ші Данг ; Нац. техн. ун-т України "Харків. політехн. ін-т". — К., 2007. — 20 с. : іл., табл. — Бібліогр.: с. 17—18.

10. Козіна Ж. Л. Теоретичні основи і результати практичного застосування системного аналізу в наукових дослідженнях в області спортивних ігор / Ж. Л. Козіна // Теорія та методика фізичного виховання. — 2007. — № 6. — С. 15—18, 35—38. — Бібліогр.: с. 38.

11. Валькман Ю. Р. Моделирование НЕ-факторов — основа интеллектуализации компьютерных технологий / Ю. Р. Валькман, В. С.

Быков, А. Ю. Рыхальский // Системні дослідження інформаційні технології. — 2007. — № 1. — С. 39—61. — Библиогр.: с. 59—61.

12. Библиотека і доступність інформації у сучасному світі: електронні ресурси в науці, культурі та освіті [Електронний ресурс] : (підсумки 10-ї Міжнар. конф. "Крим-2003") / Л. Й. Костенко, А. О. Чекмарьов, А. Г. Бровкін, І. А. Павлуша // Библиотечний вісник — 2003. — № 4. — С. 43. — Режим доступу до журн.: <http://www.nbuv.gov.ua/articles/2003/03klinko.htm>. — Назва з екрану.

Елементи, виділені напівгрубим курсивом, є факультативними і вводяться в бібліографічний запис залежно від теми курсової роботи. Проміжки між знаками та елементами запису є обов'язковими і використовують для розрізнення знаків граматичної і приписаної пунктуації.

17.4 Запитання і завдання для самоконтролю

1. Що описує стандарт ДСТУ ГОСТ 7.1:2006 ?
2. Якими вживають елементи бібліографічного опису від одного до трьох авторів ? Навести приклад.
3. Як описують джерело для чотирьох авторів ? Навести приклад.
4. В чому особливість опису джерела за умови від п'яти і більшої кількості авторів ? Навести приклад.
5. У чому полягає особливість оформлення опису тексту ?
6. У чому полягає особливість оформлення опису електронного видання з Інтернет ?
7. Якими є особливості оформлення опису періодичних видань ?
8. Якими є особливості оформлення опису стандартів ?

9. Описати особливості оформлення опису джерела з кількох томів, частин.

10. Як описують стандартні номери ISBN, ISSN ?

18 ПІДГОТОВКА ДО ЗАХИСТУ ТА ЗАХИСТ КУРСОВОЇ РОБОТИ

18.1 Допуск до захисту

Допуск до захисту курсової роботи здійснюється керівником, який приймає позитивне рішення на підставі виконання основних вимог до змісту і оформлення студентом курсової роботи.

Курсова робота повинна бути підтверджена підписами студента, керівника роботи.

18.2 Підготовка доповіді для захисту курсової роботи

Для захисту курсової роботи бажаним є підготовка слайдів за темою роботи. При створенні слайдів треба уникати багатослівності. Якщо, крім формул, на слайдах наводять текстовий матеріал, то він повинен мати мінімальний об'єм. Доцільним є подання слайдів в одному з додатків курсової роботи, як ілюстративний матеріал.

Необхідно враховувати, що слайди створюються не тільки для спрощення викладення основних положень курсової роботи, але й для комплексного, швидкого ознайомлення з ними присутніх. Присутні повинні зрозуміти суть роботи та оцінити її результати за обмежений час (10-12 хвилин). Тому подання інформації щодо використаних (запропонованих) методів та отриманих нових результатів повинне бути максимально чітким і зрозумілим.

Структура доповіді на захисті повинна бути повністю узгоджена з послідовністю подання матеріалів на слайдах.

Оцінювання складової захисту курсової роботи (r_2) виконують за критеріями (додаток А):

- ступінь володіння матеріалом – 10-6 балів («відмінно»- 10-9 балів, «добре»-8-7 балів, «задовільно»-6 балів);
- повнота аналізу можливих варіантів – 15-9 балів («відмінно»- 15-13 балів, «добре»-12-11 балів, «задовільно»-10-9 балів);
- ступінь обґрунтування прийнятих рішень – 20-12 балів («відмінно»- 20-18 балів, «добре»-17-15 балів, «задовільно»-14-12 балів);
- вміння захищати свою думку – 15-9 балів («відмінно»- 15-13 балів, «добре»-12-11 балів, «задовільно»-10-9 балів).

ВИКОРИСТАНА ЛІТЕРАТУРИ

1. Таненбаум Э. Архитектура компьютера. 4-е изд. /Э.Таненбаум. – СПб.:Питер, 2003. – 704 с.: ил. – [Электронний ресурс].-Режим доступу: <http://login.kpi.ua>
2. Коваленко А.Є. Проектування цифрових пристроїв /Коваленко А.Є.- К.:ННК «ІПСА» НТУУ «КПІ», 2012.-54 с.
3. Коваленко А.Є. Розподілені інформаційні системи : навч. посібн / операційних систем : метод. вказівки із самост. роботи студентів з дисципліни Коваленко А.Є. – К.: НТУУ «КПІ», 2008-244с.
4. Коваленко А.Є. Операційні системи : навч. посібн / Коваленко А.Є. – К.: НТУУ «КПІ», 2010-248с.
5. Схемотехніка електронних систем. У 3 кн. Кн.. 2. Цифрова схемотехніка : підручник. / [В.І.Бойко, А.М.Гуржій, В.Я.Жуйков та ін.]- 2-ге вид. - К.: Вища школа, 2004.- 423 с. ISBN 966-642-200-6
6. Положення про державну атестацію студентів НТУУ «КПІ» / Уклад.: В. П. Головенкін, В. Ю. Угольніков. – К.: НТУУ «КПІ», 2013. – 98 с.
7. Приклади оформлення бібліографічного опису // Бюлетень ВАК України.- 2008.- №3.- С.9-13.
8. Схемотехніка електронних систем. У 3 кн. Кн.. 3. Мікропроцесори та мікроконтролери : підручник. / [В.І.Бойко, А.М.Гуржій, В.Я.Жуйков та ін.]- 2-ге вид. - К.: Вища школа, 2004.- 423 с.- ISBN 966-642-202-6.
9. ДСТУ 3008-95. Державний стандарт України. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – Чинний

від 01.01.96.- [Електронний ресурс].- Режим доступу: http://www.dnu.dp.ua/docs/ndc/standarts/DSTU_3008-95.pdf

10. ДСТУ ГОСТ 7.1:2006 Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання.- Чинний від 01.07.07. .- [Електронний ресурс].- Режим доступу: <http://www.zounb.zp.ua/resource/GOST/index.html>
11. Самофалов К.Г. Цифровые электронные вычислительные машины / Самофалов К.Г., Корнейчук В.И, Тарасенко В.П. –К.: Вища школа, 1983. – 455 с.
12. Лукашук Л.О. Схемотехніка логічних та послідовнісних схем / Лукашук Л.О.. –Львів, Львівська політехніка, 2004.- 116с.
13. Хорошевский В.Г. Архитектура вычислительных систем / Хорошевский В.Г. .-М.:МГТУ им. Баумана, 2008.-520 с.
14. Мараховський Л. Ф., Воеводін С. В., Міхно Н. Л., Шарапов О. Д. Комп'ютерна схемотехніка: практикум. Для бакалаврів спеціальності “Інтелектуальні системи прийняття рішень”: — К.: КНЕУ, 2007. — 279 с.
15. Угрюмов Е.П. Цифровая схемотехника. / Е.П.Угрюмов. – СПб.:БХВ-Петербург, 2001. – 528 с.: ил. ISBN 5-8206-0100-9.
16. Положення про державну атестацію студентів НТУУ «КПІ» / Уклад.: В. П. Головенкін, В. Ю. Угольніков. – К.: НТУУ «КПІ», 2013. – 98 с.
17. Комп'ютерна схемотехніка : лабор. практик. / уклад. : В.І. Дрововозов, С.В. Журавель, А.Б. Коцор – К. : НАУ, 2012. – 74 с.
18. Бабич М. П. Комп'ютерна схемотехніка [Текст] : Навч. посіб. для студ. вищ. навч. закл. / М. П. Бабич, І. А. Жуков ; Національний авіаційний ун-т. - К. : НАУ, 2002. - 508 с.

Додаток А

ПОЛОЖЕННЯ про рейтингову систему оцінки успішності студентів

з курсової роботи _____
(код та назва)

для спеціальності _____
(шифр та назва)

факультету системних досліджень Інституту прикладного системного аналізу

Рейтингова оцінка з курсової роботи має дві складові. Перша (стартова) характеризує роботу студента над курсовою роботою та якість її оформлення.

Друга складова характеризує якість захисту курсової роботи.

Розмір першої складової дорівнює 40 балів, а розмір другої складової – 60 балів.

Система рейтингових балів

1. Стартова складова (r_1):

- своєчасне виконання графіку виконання курсової роботи – 5-3 балів («відмінно»- 5 балів, «добре»-4бали, «задовільно»-3бали);
- сучасність та обґрунтування прийнятих рішень – 12-7 балів («відмінно»- 12-11 балів, «добре»-10-9 балів, «задовільно»-8-7 балів);
- правильність застосування методів аналізу і розрахунку – 10-6 балів («відмінно»- 10-9 балів, «добре»-8-7 балів, «задовільно»-6 балів);
- якість оформлення, виконання вимог нормативних документів – 6-4 балів («відмінно»- 6 балів, «добре»-5 балів, «задовільно»-4 бали);
- якість графічного матеріалу і дотримання вимог ДСТУ – 7-4 балів («відмінно»- 7-6 балів, «добре»-5 балів, «задовільно»-4 бали);

2. Складова захисту курсової роботи (r_2):

- ступінь володіння матеріалом – 10-6 балів («відмінно»- 10-9 балів, «добре»-8-7 балів, «задовільно»-6 балів);
- повнота аналізу можливих варіантів – 15-9 балів («відмінно»- 15-13 балів, «добре»-12-11 балів, «задовільно»-10-9 балів);

– ступінь обґрунтування прийнятих рішень – 20-12 балів («відмінно»- 20-18 балів, «добре»-17-15 балів, «задовільно»-14-12 балів);

– вміння захищати свою думку – 15-9 балів («відмінно»- 15-13 балів, «добре»-12-11 балів, «задовільно»-10-9 балів).

До загального рейтингу можуть додаватись бали, отримані за неонов'язкові складові. До неонов'язкових складових віднесено:

участь у модернізації робіт з дисципліни;

доповіді на наукових студентських семінарах, конференціях, якщо робота мала відношення до курсової роботи.

За їх виконання студент може отримати до 10 заохочувальних балів (у межах максимального числа 10 заохочувальних балів на повний рейтинг 100 балів).

За отриманими рейтинговими балами визначають оцінку за курсову роботу згідно із таблицею

Бали $R = r_1 + r_2$	ECTS оцінка	Залікова оцінка
95-100	A	відмінно
85-94	B	добре
75-84	C	
65-74	D	задовільно
60-64	E	
Менше 60	Fx	незадовільно
Курсова робота не допущена до захисту	F	не допущено

Додаток Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Навчально-науковий комплекс «Інститут прикладного системного аналізу»

Кафедра математичних методів системного аналізу

Курсова робота

з дисципліни «Комп'ютерна схемотехніка і архітектура комп'ютерів»

зі спеціальності _____

на тему: _____

Виконав (-ла): студент (-ка) _____ курсу, групи _____
(шифр групи)

_____ (прізвище, ім'я, по батькові)

_____ (підпис)

Керівник

доцент, к.т.н. Коваленко А.Є

_____ (підпис)

Київ – 20__ року

Додаток В

Національний технічний університет України
«Київський політехнічний інститут»

Навчально-науковий комплекс «Інститут прикладного системного аналізу»

Кафедра математичних методів системного аналізу

Освітньо-кваліфікаційний рівень: бакалавр

Напрямок підготовки:

Спеціальність «Системи штучного інтелекту»

ЗАВДАННЯ на курсову роботу студенту

_____ (прізвище, ім'я, по батькові)

1. Тема роботи _____

2. Строк подання студентом курсової роботи _____

3. Вихідні дані до роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) _____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): _____ слайдів, з них

1) _____

2) _____ 3) _____

Календарний план

№ з/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Примітка
	Отримання теми та завдання		
	Підбор та вивчення літератури		
	Підготовка розділів роботи		
	Оформлення курсової роботи		
	Подання курсової роботи на перевірку		
	Захист курсової роботи		

Студент

_____ (підпис)

_____ (ініціали, прізвище)

© А.С.Коваленко, 2016

Навчальне видання

Коваленко Анатолій Єпіфанович

**КОМП'ЮТЕРНА СХЕМОТЕХНІКА І АРХІТЕКТУРА
КОМП'ЮТЕРІВ. ПІДГОТОВКА ТА ОФОРМЛЕННЯ
КУРСОВИХ РОБІТ**

Навчально-методичний посібник для студентів, які навчаються за спеціальністю
«Комп'ютерні науки та інформаційні технології»

Редактор А.С.Коваленко
Комп'ютерна верстка А.С.Коваленко