НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Бібліотека ідентифікції процесора для платформи .NET

Виконав : студент __4__ курсу, групи ІО-83
(шифр групи)
_____Гогсадзе Микола Георгійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент, Каплунов А. В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Гогсадзе Миколи Георгійовича

1. Тема проєкту Бібліотека ідентифікції процесора для платформи .NET
керівник проєкту Каплунов Артем Володимирович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2022 року №1139-с

2. Термін здачі студентом закінченого проєкту *6 червня 2022 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз існуючих архітектур.

Розділ 2. Аналіз технологій для реалізації проекту.

Розділ 3. Розробка бібліотеки ідентифікації процесора.

Розділ 4. Тестування та інструкція з роботи системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «30» серпня 2021 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2022-25.03.2022*	
4.	*Розробка структур окремих підсистем*	*25.03.2022-5.04.2022*	
5.	*Програмна реалізація системи*	*5.04.2022-15.04.2022*	
6.	*Оформлення пояснювальної записки*	*15.04.2022-20.05.2022*	
7.	*Захист програмного продукту*	*25.04.2022*	
8.	*Передзахист*	*15.05.2022*	
9.	*Захист*	*22.06.2022*	

Студент-дипломник _____ Микола ГОГСАДЗЕ
(підпис)

Керівник проєкту _____ Артем КАПЛУНОВ
(підпис)

АНОТАЦІЯ

У даній дипломній роботі розроблено бібліотеку для ідентифікації процесора для платформи .NET з використанням популярних технологій та методів розробки програмного забезпечення.

У результаті є розроблене універсальне програмне забезпечення, що містить детальну інформацію про функції процесору під час виконання. Бібліотека не споживає багато оперативної пам’яті та підходить для реалізації основних функцій і роботи в ізольованих середовищах. Прототип отриманої платформи, відповідає усім необхідним вимогам та дає можливість організувати ефективну взаємодію із розробниками.

ANNOTATION

In this thesis, a library was developed to identify the processor for the .NET platform using popular technologies and software development methods.

As a result, universal software has been developed that contains detailed information about the functions of the process at runtime. The library does not consume much RAM and is suitable for basic functions and work in isolated environments. The prototype of the received platform meets all necessary requirements and gives the chance to organize effective interaction with developers.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467400.002 ТЗ***	Бібліотека ідентифікації	3		
			процесора для платформи			
			.NET			
			Технічне завдання			
	А4	***ІАЛЦ.467400.003 ПЗ***	Бібліотека ідентифікації	81		
			процесора для платформи			
			.NET			
			Пояснювальна записка			
	А4	***ІАЛЦ.467400.004 Д1***	Бібліотека ідентифікації	1		
			процесора для платформи			
			.NET			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.467400.005 Д2***	Бібліотека ідентифікації	1		
			процесора для платформи			
			.NET			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467400.006 Д3***	Бібліотека ідентифікації	1		
			процесора для платформи			
			.NET			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467400.007 Д4***	Бібліотека ідентифікації	16		
			процесора для платформи			
			.NET			
			Текст програмного коду			

					# *ІАЛЦ.467400.001 ОА*		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Гогсадзе М.Г.			Бібліотека ідентифікації процесора для платформи .NET Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Каплунов А.В.					
						НТУУ "КПІ" ФІОТ ІО-83	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Бібліотека ідентифікації процесора для платформи .NET*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дана дипломна робота поширюється на бібліотеку для ідентифікації процесору архітектур X86, AMD64, AARCH32, AARCH64 для платформи .NET. Область застосування: розширення існуючих можливостей з отримання детальної інформації про процесор під час середовища виконання.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної роботи є завдання на виконання роботи освітньої програми “Комп'ютерні системи та мережі”, спеціальності за номером 123 “Комп'ютерна інженерія”, що є затверджено кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної проектної роботи є розроблене універсальне програмне забезпечення, що містить інформацію про функції процесору під час виконання. Бібліотека не споживає багато оперативної пам’яті та підходить для реалізації основних функцій і роботи в ізольованих середовищах.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є технічна документація, публікації в мережі Інтернет, відкриті джерела вихідного коду та науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Здійснення підтримки процесорів архітектур X86, AMD64, AARCH32, AARCH64.
- Універсальність бібліотеки, можливість запуску на різних операційних системах та середовищах .NET Core .NET Framework.
- Швидкість, простота та робота в ізольованих середовищах.

5.2. Вимоги до програмного забезпечення

- Операційні системи: macOS, Microsoft Windows 7/8/10/11, Linux, Android та IOS.
- Мова програмування C# будь-якої версії, .NET Framework або .NET Core SDK.
- Інтегроване середовище розробки Rider JetBrains або Visual Studio

5.3. Вимоги до апаратної частини

- Будь-яка з перечислених архітектур процесорів: X86, AMD64, AARCH32, AARCH64.
- Персональний комп’ютер з процесором Intel Core I5.
- Вільний простір жорсткого диску не менше 100МБ.
- Одноплатний комп’ютер Raspberry Pi випуску 3 та вище.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2021
Вивчення необхідної літератури та документації	15.12.2021-15.03.2022
Аналіз, складання та узгодження поставленого технічного завдання	15.03.2022-25.03.2022
Оформлення вступної частини роботи, вивчення існуючих рішень	25.03.2022-5.04.2022
Вивчення необхідних технологій та підходів розробки	5.04.2022-15.04.2022
Реалізація програмної частини	15.04.2022-20.05.2022
Оформлення пояснювальної записки	25.05.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Бібліотека ідентифікації процесора для платформи .NET*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ISA	(Instruction Set Architecture) Архітектура набору команд
SIMD	(Single Instruction Multiple Data) Одна інструкція багато даних
SSE	(Streaming SIMD Extension) Набір інструкцій потокового SIMD розширення
RID	(NET Runtime Identifier) Ідентифікатор .NET середовище виконання
AVX	(Advanced Vector Extension) Вдосконалене розширення вектору
AES	(Advanced Encryption Standard) Симетричний алгоритм блочного шифрування
SDK	(Software Development Kit) Комплект для розробки програмного забезпечення
FMA	(Fused Multiply-Add) Змішане множення додавання
ОС	Операційна система

ВСТУП

“Напиши один раз, запускай всюди” цю заповідь почали говорити ще з 1990 року при розробці мови програмування Java. Ви можете написати код програми на одній платформі та запустити його на будь-якому процесорі, що підтримує віртуальну машину Java. Але для розробників, яким потрібно збільшити оптимізацію кожного біту їхньої програми це є не достатньо. З самого початку ери обчислювальної техніки, розробники, що орієнтуються на продуктивність їхніх програм використовували точну інформацію про апаратне забезпечення, щоб максимально якісно налаштувати свій код.

Для прикладу, представимо, що ви працюєте над розробкою програмного забезпечення для якого швидкість та ефективність є визначальним, можливо новий відеокодек чи бібліотека для обробки за допомогою штучного інтелекту. Існують окремі інструкції, які значно покращать продуктивність, такі як злите множення та додавання, а також цілі набори інструкцій, як SSE2 та AVX, що можуть підвищити швидкість критичних частин вашої програми.

І ось виникає проблема: немає способу знати апріорі, які інструкції підтримує ваш процесор. Визначення виробника процесора є недостатнім. Наприклад, архітектура Intel Haswell підтримує набір інструкцій AVX2, а Sandy Bridge — ні. Деякі розробники вдаються до відчайдушних заходів, таких як читання ‘proc/cpuinfo’ для ідентифікації центрального процесору, а потім перегляд жорстко закодованих відображень ідентифікаторів ЦП з інструкціями.

Отже, беручи до уваги вище зазначену інформацію, у даному дипломному проекті було поставлено задачу, а саме: розробка бібліотеки для ідентифікації процесора для платформи .NET. Система cpu_features.NET являє собою невелику, швидку та просту бібліотеку з відкритим кодом, щоб повідомляти про функції процесору під час виконання. Наразі бібліотека підтримує архітектури процесорів x86, ARM/AArch64.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ АРХІТЕКТУР

1.1 Огляд архітектур центрального процесору

В даному розділі здійснемо короткий огляд існуючих архітектур від постачальників на ринку.

1.1.1 Основні поняття

Для початку хочу звернути увагу на звичному і не зовсім дивовижному для всіх факті: одна і та ж сама програма для персонального комп'ютера без будь-яких модифікацій та перетворень двійкового коду здатна запускатися на системах, які влаштовані усередині по-різному. Адже у центрального процесору від різних виробників та різних поколінь внутрішня будова, як правило, не повторюється: розміри транзисторів інші, частота варіюється, організація пам'яті зроблена по-різному, конвеєр містить у собі різну кількість стадій. Проте, той же самий двійковий код практично у більшості випадків завжди працює. Це можливо через утворені раніше домовленості від розробників апаратури та програмного забезпечення, які було закріплено у певній формі специфікації для набору інструкцій, скорочено ISA (instruction set architecture). Незмінність вище зазначеного інтерфейсу — це у більшості випадків є запорукою для комерційних досягнень у мікропроцесорній промисловості.

І все ж таки, практично кожна архітектура центрального процесору, яку йменують "старою" за період своєї еволюції постійно вдосконалюється різноманітними розширеннями ISA. Саме вони приносять у розробку програмного забезпечення зниження енергоспоживання, поліпшення швидкості обчислень, прискорення шифрування, засоби для забезпечення безпеки, віртуалізації, обробки сигналів тощо. Проте, виникає досить

суперечлива ситуація: із одного боку, існує потреба у певній стабільності, а з іншого - постійні поліпшення від конкуруючих між собою виробників.

Розбиратися у наведеній вище проблемі часто доводиться розробникам програмного забезпечення, якщо не прикладним, то спеціалістам у областях операційних систем та компіляторів. Їхні програми повинні «вміти» розрізняти особливості тієї апаратури, на якій вони запущені, і вибирати гілки виконання, що оптимально задіяні для виявлення розширення.

У специфікаціях на архітектури процесорів для цього зазвичай описуються засоби ідентифікації у вигляді спеціалізованих регістрів та інструкцій. Тому у даній дипломній роботі йде аналіз відмінностей, що існують у тому, яку інформацію, наскільки докладно та в якому форматі різні виробники доносять до системних розробників програмного забезпечення.

1.2 Порівняння різних архітектур

У даній дипломній роботі пропоную короткий огляд засобів ідентифікації низки популярних сімей центральних процесорів. Його було підготовлено за допомогою вивчення доступної в Інтернеті документації, власних експериментів там, де вдалося отримати необхідний процесор, вивчення вихідних джерел архітектурно-залежних частин ядер відкритих операційних систем, таких як Linux і FreeBSD, а також завдяки обговоренням з друзями, колегами зайнятими розробкою системного програмного забезпечення під дані архітектури. У кожному випадку відбуватиметься аналіз повної ємності архітектурного стану в бітах.

1.2.1 MIPS32

Архітектура процесору MIPS32 [1, 2] (нині належить відомій компанії Imagination Technologies) зберігає інформацію про ідентифікацію в регістрі PRid – п’ятнадцятий регістр нульового співпроцесора [3]. У ньому розміщено всього 32 біти.

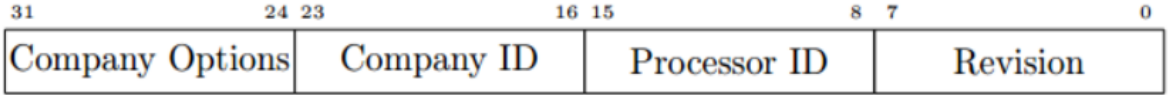


Рисунок 1.1 - Схема процесору MIPS32

Окрім того, для опису різних можливостей модуля роботи із числами з плаваючою комою, скорочено FPU, був задіяний Floating Point Implementation Register, скорочено FIR. Він здійснює лише операцію читання та містить ширину 32 біти [4].

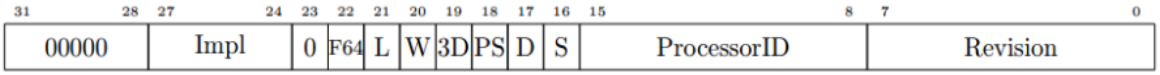


Рисунок 1.2 - Схема модуля роботи FIR

1.2.2 ARM

Ця відома та популярна RISC архітектура, яка не підтримує для розробників програмного забезпечення зручних засобів для ідентифікації наявних розширень. Саме такий висновок можливо здійснити із тієї інформації, що уніфікація коду підтримки архітектури ARM у операційній системі Linux була розроблена лише під час релізу версії 3.7. Даний факт досить вражає, адже враховуючи, що ARM – це архітектура, з якої випуск процесорів здійснює багато відомих виробників.

32-розрядний регістр cpuid [5] зі складу System Control Coprocessor обумовлює декілька властивостей, а саме охоплює код виробника та ревізію.

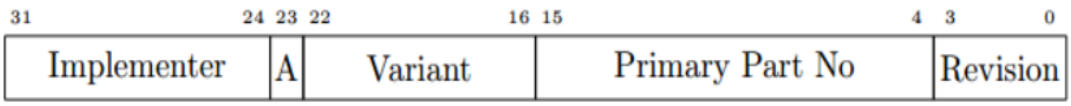


Рисунок 1.3 - Схема модуля роботи System Control Coprocessor

Приклади значень даного регістру при використанні різних продуктів наступні: Intel (XScale) PXA272 - 0x69054117, Qualcomm MSM7200A - 0x4117b362.

Набагато більше інформації доступні для отримання через Debug-Регістри, проте для таких операцій необхідні привілегії ядра процесора. Теоретично у вісімнадцяти регістрах по 32 біти кожного ядра можливо здійснити кодування 576 біт.

```
Vendor: ARM
Main ID Register                              0x410fb767
Cache Type Register                           0x1d152152
TCM Status Register                                  0
TLB Type Register                                0x800
Processor Feature Register 0                     0x111
Processor Feature Register 1                      0x11
Debug Feature Register 0                          0x33
Auxiliary Feature Register 0                         0
Memory Model Feature Register 0                0x1130003
Memory Model Feature Register 1               0x10030302
Memory Model Feature Register 2                0x1222100
Memory Model Feature Register 3                      0
Instruction Set Attributes Register 0           0x140011
Instruction Set Attributes Register 1          0x12002111
Instruction Set Attributes Register 2          0x11231121
Instruction Set Attributes Register 3           0x1102131
Instruction Set Attributes Register 4              0x1141
Instruction Set Attributes Register 5                   0
```

Рисунок - 1.4 Інформація про Debug регістри ядра процесора

1.2.3 IBM System

IBM System [6], має свої початки від мейнфреймів, проте достатньо сильно відрізняється у приналежності від більш звичних для більшості систем, що містять походження від більш застарілих версій. Але й на мейнфреймах необхідно було мати вміння розпізнавати розширення, яких з 1960-х років

могло налічує чимало. Для цього в архітектурі системи присутні інструкції STSI та STIDP.

STSI використовується задля отримання точної інформації про модель та ідентифікатор ємності основної конфігурації.

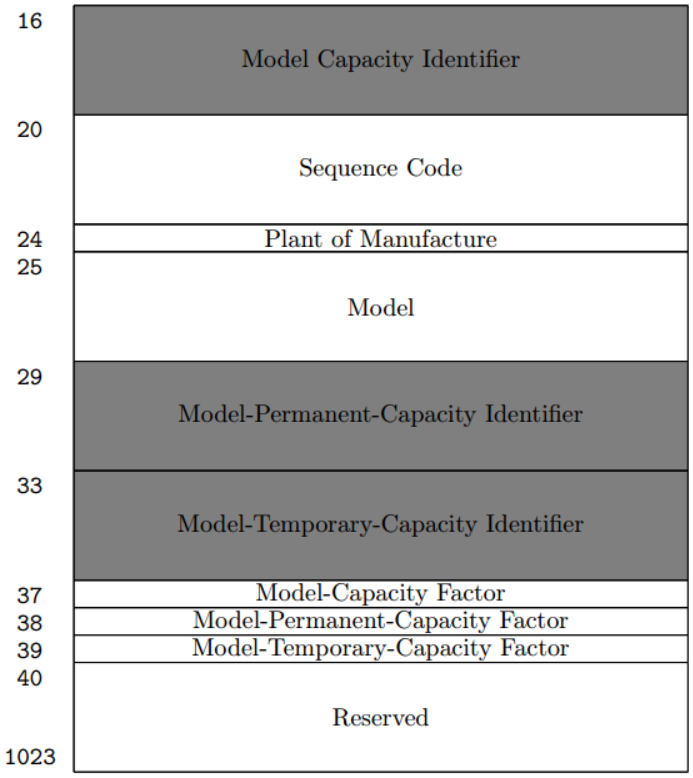


Рисунок 1.5 - Схема про модель та ідентифікатор ємності конфігурації

У сорока чотирьох регістрах по 32 біти можливо помістити близько 1,4 кбіт даних. STIDP здійснює повідомлення про тип процесора, ідентифікатор логічної партиції та серійний номер, які можуть використовуватися задля операції віртуалізації.

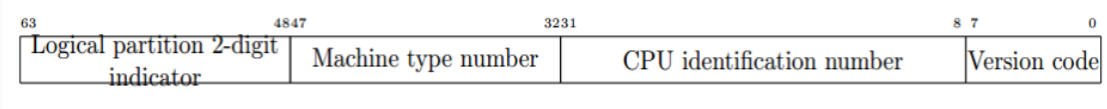


Рисунок 1.6 - Схема структури роботи STIDP

1.2.4 PowerPC

Єдиний у системі PowerPC [7] регістр PVR містить ширину 32 біти та інформацію лише про ревізію та ідентифікатор постачальника.

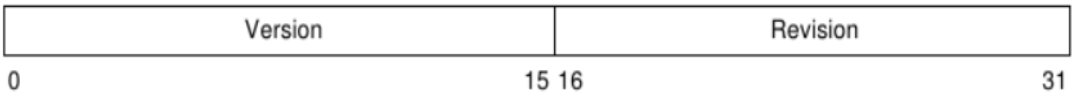


Рисунок 1.7 - Схема структури роботи PowerPC

Однак, для розширення специфікації ISA передбачений механізм APU (application processor units) - окремих "пристроїв", які здійснюють декодування та виконання опціональних команд.

1.2.5 SPARC

Стандарт SPARC версії 9 [8] від самої основи не був прив'язаний до певного виробника і саме тому його специфікації виглядають досить універсальними. Порівняно велика кількість пунктів у документації є спеціально позначена як «залежна від реалізації». Для ідентифікації процесора було введено 64-бітний регістр VER:

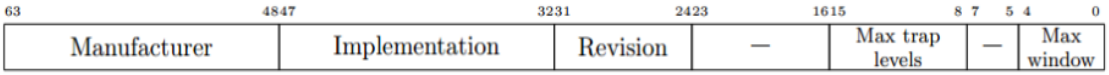


Рисунок 1.8 - Схема структури роботи SPARC

1.2.6 Intel IA-64 (Itanium)

Intel IA-64, дуже відомий на ринку процесорів як Itanium, був спроектований після серії Intel 80x86 та містить призначення для заміни останнього. Не дивно, що саме набір cpuid регістрів, які мали використання для ідентифікації архітектури IA-64 [9], віддалено нагадує певну структуру для здійснення виведення інструкції CPUID на архітектурі Intel IA-32.

На зараз архітектури IA-64 пропонують користувачам до п'яти регістрів cpuid. У разі, якщо необхідно здійснити розширення в майбутньому, біти від 0

до 7 у складі набору регістрів cpuid зберігають повноцінну кількість таких регістрів (тобто максимальна їх кількість може містити 256 біт).

Вміст регістрів cpuid для процесора Intel Itanium 9100, був отриманий за допомогою системи ggg-cpuid [10].

Leaf	Value
0	0x49656e69756e6547
0x1	0x6c65746e
0x2	0
0x3	0x20010104
0x4	0x5

Рисунок 1.9 - Вміст регістрів cpuid для процесора

1.2.7 Intel IA-32 и Intel 64

Усім користувачам знайомі модернізовані персональні комп'ютери, які містять походження від системи IBM PC, а також використовують архітектуру Intel IA-32. Починаючи з версії процесору Intel Pentium (і його клонів), нащадки даної архітектури містять у собі інструкцію CPUID. 64-бітове розширення, на разі відоме як система Intel64 [11] або AMD64 [12], не здійснило внесок у принципі зміни у її роботоздатність.

Інструкція CPUID приймає у собі на вхід два 32-бітних значення в регістри EAX та ECX, які називаються лист і підлист (з англійської leaf і subleaf), а також містять у собі результати в чотирьох 32-бітних регістрах: EAX, EBX, ECX і EDX. Варто зазначити, що в 64-бітному режимі все одно використовуються лише 32 біти всіх регістрів.

Теоретично лист і підлист здійснюють кодування 64 біт ключа, а сам висновок містить у собі 128 біт даних. На щастя, далеко не всі можливі комбінації на даний момент є доступними. На жаль, комбінації листів із підлистами також мають досить специфічну логіку. Із моменту здійснення

введення команди, обсяг виведення системи CPUID (тобто число максимально допустимих комбінацій листів та підлистів) було розширено у велику кількість разів.

Вивід системи CPUID для процесору Intel CORE i5-8250U CPU 1.60GHz, отриманий за допомогою cpuid [13].

```
0x00000000 0x00: eax=0x00000016 ebx=0x756e6547 ecx=0x6c65746e edx=0x49656e69
0x00000001 0x00: eax=0x000806ea ebx=0x02100800 ecx=0xfedaf387 edx=0xbfebfbff
0x00000002 0x00: eax=0x76036301 ebx=0x00f0b5ff ecx=0x00000000 edx=0x00c30000
0x00000003 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000004 0x00: eax=0x1c004121 ebx=0x01c0003f ecx=0x0000003f edx=0x00000000
0x00000004 0x01: eax=0x1c004122 ebx=0x01c0003f ecx=0x0000003f edx=0x00000000
0x00000004 0x02: eax=0x1c004143 ebx=0x00c0003f ecx=0x000003ff edx=0x00000000
0x00000004 0x03: eax=0x1c03c163 ebx=0x02c0003f ecx=0x00001fff edx=0x00000002
0x00000005 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000006 0x00: eax=0x000007f3 ebx=0x00000002 ecx=0x00000009 edx=0x00000000
0x00000007 0x00: eax=0x00000000 ebx=0x009c67ab ecx=0x00000000 edx=0xbc000400
0x00000008 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000009 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x0000000a 0x00: eax=0x07300404 ebx=0x00000000 ecx=0x00000000 edx=0x00000603
0x0000000b 0x00: eax=0x00000001 ebx=0x00000002 ecx=0x00000100 edx=0x00000002
0x0000000b 0x01: eax=0x00000004 ebx=0x00000008 ecx=0x00000201 edx=0x00000002
0x0000000c 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x0000000d 0x00: eax=0x0000001f ebx=0x00000440 ecx=0x00000440 edx=0x00000000
0x0000000d 0x01: eax=0x0000000f ebx=0x000003c0 ecx=0x00000000 edx=0x00000000
0x0000000d 0x02: eax=0x00000100 ebx=0x00000240 ecx=0x00000000 edx=0x00000000
0x0000000d 0x03: eax=0x00000040 ebx=0x000003c0 ecx=0x00000000 edx=0x00000000
0x0000000d 0x04: eax=0x00000040 ebx=0x00000400 ecx=0x00000000 edx=0x00000000
0x0000000e 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x0000000f 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000010 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000011 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000012 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000013 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
0x00000014 0x00: eax=0x00000001 ebx=0x0000000f ecx=0x00000003 edx=0x00000000
0x00000014 0x01: eax=0x02490002 ebx=0x003f3fff ecx=0x00000000 edx=0x00000000
0x00000015 0x00: eax=0x00000002 ebx=0x00000096 ecx=0x00000000 edx=0x00000000
0x00000016 0x00: eax=0x00000000 ebx=0x00000000 ecx=0x00000000 edx=0x00000000
```

Рисунок 1.10 - Вивід системи CPUID для процесору

1.3 Порівняння засобів ідентифікації

Усі досліджені системи дозволяють виявнити щонайменше два параметри: постачальника та певний номер “ревізії” для процесору. Дійсно, це є мінімально допустимим набором для здійснення ідентифікації. Якщо зберігати у програмному застосунку таблицю для відповідності параметрів «постачальник» → «продукт» → «список розширень», дана схема надасть можливість однозначно ідентифікувати властивості певної плати, де було здійснено запуск.

Не можна стверджувати, що цей мінімалізм є зручним. По-перше, необхідно якось сформувати, опрацювати та зберігати вище зазначену таблицю, а для цього прийдеться розглянути досить об’ємну кількість документації та здійснити певні експериментальні дії. По-друге, очевидна крайня негнучкість розробленого коду стосовно використання майбутніх систем. Досить новому постачальнику здійснити вихід на світовий ринок або вже присутньому розробити сумісний із центральним процесор із новим номером ревізії, і тоді наступний додаток не зможе чітко ідентифікувати його без здійснення оновлення таблиці із майбутньою перекомпіляцією. Використання бітових масок для операції кодування наявних розширень надає можливість "старому" коду чітко ігнорувати нові розширення, але дізнаватись популярні та відомі, що є досить корисними для зворотньої сумісності.

Що дійсно відрізняється у всіх наявних на світовому ринку архітектурах, так це обсяг даних, які надаються для операції ідентифікації. У наведеному графіку зібрано сукупно наближені значення ємності пов'язаного з цим архітектурного стану в бітових одиницях для усіх описаних вище центральних процесорів.

1.4 Аналіз відомих програмних продуктів

На ринку відома достатня кількість прикладів програмного забезпечення, яке створене для ідентифікації процесору. У кожного з них є свої особливості використання, переваги та недоліки. Дуже важливо, щоб здійснити ідентифікацію можливо було на різних платформах та отримати деталізовану інформацію. Для аналізу відомих на ринку програмних продуктів було обрано деякі приклади серед програмного забезпечення. Отже, стратегія з аналізу доволі різних підходів щодо вирішення однієї і тієї ж задачі — дозволяє достатньо змістовно та ефективно оцінити як і переваги так і недоліки серед кожного підходу та здійснити виведення оптимальної ідеї для розв’язання необхідного питання, яке буде опиратися на отримані нижче результати.

1.4.1 CPU INFOrmation library

CPUInfo — це відкрита бібліотека для виявлення важливої для операцій оптимізації продуктивності чіткої інформації про центральний процесор [14].

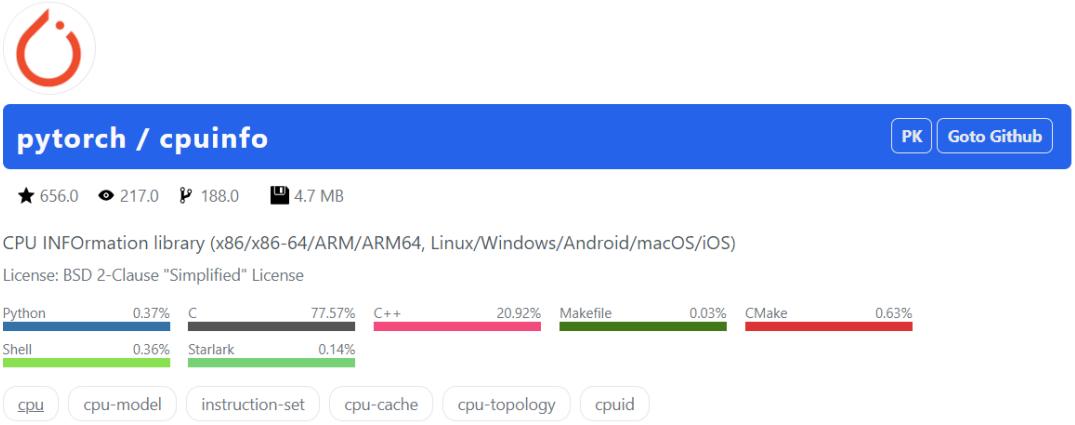


Рисунок 1.11 - Застосунок із документацією для CPUInfo

Розглянемо детальніше приклади використання бібліотеки:

Ім'я процесора модуля:

```
cpuinfo_initialize();
printf("Running on %s CPU\n", cpuinfo_get_package(0)->name);
```

Рисунок 1.12 - Лістинг для отримання імені процесору

Визначення, чи є цільовою 32-розрядна чи 64-розрядна система ARM:

```
#if CPUINFO_ARCH_ARM || CPUINFO_ARCH_ARM64
    /* 32-bit ARM-specific code here */
#endif
```

Рисунок 1.13 - Лістинг для отримання системи ARM

Перевірка, чи підтримує центральний процесор x86 AVX:

```
cpuinfo_initialize();
if (cpuinfo_has_arm_neon()) {
    neon_implementation(arguments);
}
```

Рисунок 1.14 - Лістинг для перевірки підтримки процесору ARM NEON

Перевірка, чи працює потік на ядрі Cortex-A53:

```
cpuinfo_initialize();
switch (cpuinfo_get_current_core()->uarch) {
    case cpuinfo_uarch_cortex_a53:
        cortex_a53_implementation(arguments);
        break;
    default:
        generic_implementation(arguments);
        break;
}
```

Рисунок 1.15 - Лістинг для перевірки потоку на ядрі Cortex-A53

Основні переваги:

- Підтримка використання на різних платформах:
  - Linux, Windows, macOS, Android, та iOS операційних системах
  - x86, x86-64, ARM, та ARM64 архітектурах
- Сучасний C/C++ інтерфейс:
  - Потокобезпечний
  - Немає виділення пам'яті після ініціалізації
  - Немає викидання помилок
- Виявлення підтримуваних наборів інструкцій, аж до розширень AVX512 (x86) та ARMv8.3
- Виявлення назви процесора, скорочено SoC та основної інформації:
  - Назва процесора (SoC)
  - Постачальник і мікроархітектура для кожного ядра центрального процесора
  - ID (MIDR на ARM, CPUID листа першого значення EAX на x86) для кожного ядра центрального процесора

- Виявлення інформації кешу:
    - Тип кешу (інструкції, дані, уніфікація), розмір та розмір рядка
    - Асоціативність кешу
    - Ядра та логічні процесори (тобто гіперпотоки), які спільно використовують кеш
- Виявлення інформації про топологію (відносно між якими логічними процесорами, ядрами та пакетами процесорів)
- Добре протестований програмний код:
    - понад шістдесят фіктивних тестів на основі даних із реальних пристроїв
    - Включає обхідні шляхи для поширених помилок в апаратному забезпеченні та ядрах ОС
    - Підтримує системи з гетерогенними ядрами, такими як big.LITTLE та Max.Med.Min
- Дозволена ліцензія з відкритим вихідним кодом (Simplified BSD)

Основні недоліки:

- Відсутня підтримка архітектури MIPS для андроїд середовища;
- Відсутня підтримка PowerPC64 для операційної системи LINUX;
- Неможливість визначати частоту центрального процесору:
- Відсутня підтримка резервного буферу транслювання, а саме ідентифікація кількості записів, асоціативності, покритих типів сторінок(інструкції, дані) та покритих розмірів сторінок;
- Відсутнє визначення кешу для використання CPUID листків 0x80000005-0x80000006 (AMD x86/x86-64);

1.4.2 cpu_features

Багатоплатформова бібліотека мови програмування C для отримання функцій центрального процесору (наприклад, усіх доступних інструкцій) під час виконання. Коли цей проект був створений, бачення полягало в тому, що його можна запускати на вбудованих пристроях з невеликим обсягом пам'яті, тому використовується C99 стандарт та бітові поля для зменшення споживання пам'яті [15].

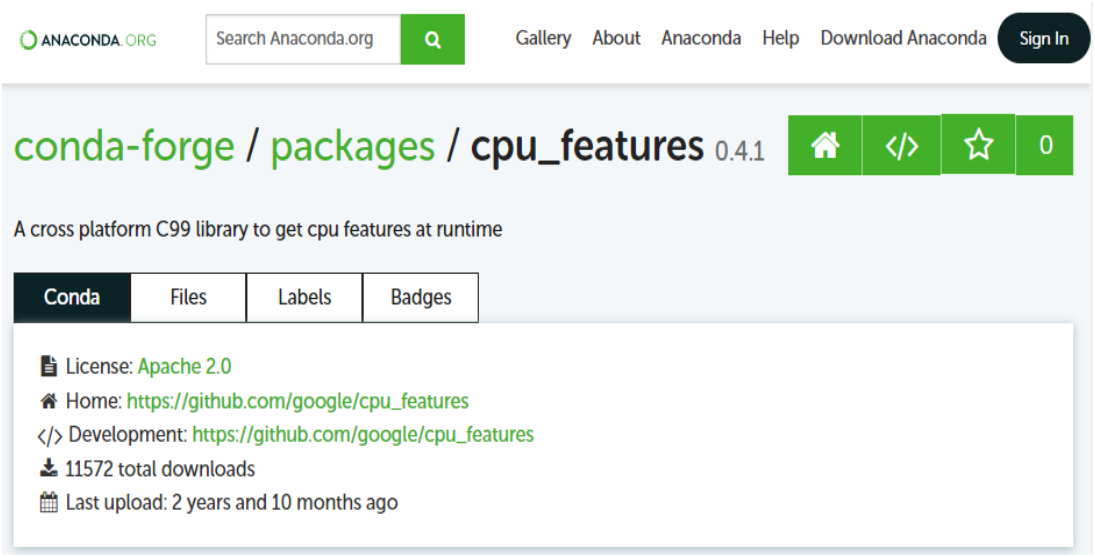


Рисунок 1.16 - Застосунок із документацією для cpu_features

Приклади використання бібліотеки:

Перевірка функцій під час виконання. Нижче наведено простий приклад, який виконує кодовий шлях, якщо центральний процесор підтримує як AES, так і набори інструкцій SSE4.2.

```
#include "cpuinfo_x86.h"

// For C++, add `using namespace cpu_features;`
static const X86Features features = GetX86Info().features;

void Compute(void) {
  if (features.aes && features.sse4_2) {
    // Run optimized code.
  } else {
    // Run standard code.
  }
}
```

Рисунок 1.17 - Лістинг для підтримки процесору AES

Кешування для більш швидкої оцінки складних перевірок. Якщо розробник бажає, то може прочитати всі функції відразу в глобальну змінну, а потім викликати конкретні функції, які необхідні. Нижче наведено зберігання всіх функції ARM, а потім перевірка, чи підтримуються AES та NEON [16].

```
#include <stdbool.h>
#include "cpuinfo_arm.h"

// For C++, add `using namespace cpu_features;`
static const ArmFeatures features = GetArmInfo().features;
static const bool has_aes_and_neon = features.aes && features.neon;

// use has_aes_and_neon.
```

Рисунок 1.18 - Лістинг зберігання всіх функції ARM

Перевірка прапорів часу компіляції. Наступний код визначає, чи було вказано компілятору використовувати набір інструкцій AVX (наприклад, команда g++ -mavx), і відповідно встановлює прапор has_avx.

```
#include <stdbool.h>
#include "cpuinfo_x86.h"

// For C++, add `using namespace cpu_features;`
static const X86Features features = GetX86Info().features;
static const bool has_avx = CPU_FEATURES_COMPILED_X86_AVX || features.avx;

// use has_avx.
```

Рисунок 1.19 - Лістинг перевірки використання набору інструкцій AVX

Відмова від поганих апаратних реалізацій на основі мікроархітектури. На x86 перше втілення функції в мікроархітектурі може бути не найефективнішим (наприклад, AVX на Sandy Bridge). Бібліотека надає функцію для отримання базової мікроархітектури, для того щоб вирішити, чи використовувати її.

```
#include <stdbool.h>
#include "cpuinfo_x86.h"

// For C++, add `using namespace cpu_features;`
static const X86Info info = GetX86Info();
static const X86Microarchitecture uarch = GetX86Microarchitecture(&info);
static const bool has_fast_avx = info.features.avx && uarch != INTEL_SNB;

// use has_fast_avx.
```

Рисунок 1.20 - Лістинг отримання базової мікроархітектури

Основні переваги:

- досить простий у використанні. Можна переконатися із наведених вище фрагментів прикладів.
- розширюваний, тобто легко додати відсутні функції або архітектури.
- сумісний зі старими версіями компіляторів та доступний на багатьох різноманітних архітектурах, тому його можна широко використовувати. Для того щоб гарантувати, що cpu_features працює на якомога більшій кількості платформ, його впровадили у дуже портативну версію мови програмування C: C99 стандарт.
- сумісний із “пісочницею”, тобто бібліотека використовує різноманітні стратегії, для того щоб впоратися з “пісочними” середовищами або коли функціонал ідентифікації процесору

недоступний. Це дуже корисно під час виконання необхідних інтеграційних тестів у герметичних середовищах.

- потокобезпечний, без виділення пам’яті та без винятків. cpu_features чудово підходить для реалізації основних функцій libc, таких як malloc, memcpy і memcmp.
- містить модульне тестування.

Основні недоліки:

- у бібліотеці існує потенційна проблема з бітовими полями, тому не надається жодних гарантій стабільності ABI.
- категорично не рекомендується використовування спільних бібліотек.
- відсутня підтримка систем архітектур AARCH64 для операційної системи macOS.

1.5 Аналіз вимог до програмного забезпечення

У бібліотеці передбачаються наступні архітектури: X86, AMD64, ARM, AARCH64. У користувача для даних архітектур є можливість отримувати ідентифікацію за допомогою таких функцій як CpuInfoAarch64, CpuInfoArm, CpuInfoX86. Споживач може запускати код тільки під архітектуру свого процесору, на якому він працює та створювати збірку версій для інших архітектур.

Дана бібліотека написана на .NET Standard 1.1, тому у користувача є можливість здійснювати запуск на різних середовищах .NET Core, .NET Framework. На .NET Framework можна лише запускати код під операційну систему Windows та X86/AMD64 архітектуру, для .NET Core можливо використовувати Linux, Windows, macOS, FreeBSD, та архітектури X86, AMD64, ARM, AARCH64.

У споживача бібліотеки під X86/AMD64 архітектуру повинен бути процесор, який може використовувати інструкцію cpuid, оскільки .NET платформа не підтримує процесори десятилітньої давності, тому немає сенсу робити ідентифікацію для даного випадку.

1.6 Розробка функціональних вимог

Для кожного проекту, не лише комерційного, необхідно достатньо уважно проаналізувати основні пріоритети серед усіх вимог, для того щоб він приносив найкращу користь для суспільства та водночас найбільший прибуток для компанії та розглядався досить успішним. Невикористання розумного підходу для оцінки критеріїв із чіткого формулювання пріоритетів заданих вимог є однією із тих причин, яка призводить до провалів більшості проектів. Саме тому, утворимо таблицю, яка складається з необхідних функціональних вимог для отримання бажаних результатів виходячи із заданих технічних можливостей.

Таблиця 1.1 – Таблиця умовних скорочень

Пріоритет	Скорочення	Опис
Обов'язковий	О	Означає, що вимога є обов’язковою для виконання, через те що проект буде зупинено на невизначений період, якщо її не вдасться реалізувати.
Високий	В	Якщо дані бажані вимоги не будуть реалізовані, це матиме достатньо значний вплив на проект, але він не буде зупинений.
Низький	Н	Якщо ці додаткові вимоги не будуть реалізовані, то вплив на проект буде досить низьким та незначним.

Таблиця 1.2 – Опис функціональних вимог до дипломного проекту

Ідентифікатор	Опис функціоналу	Пріоритет
1.1	Інтегрування мови програмування C із бібліотеки cpu_features у платформу .NET	O
1.2	Автоматизація процесів збірки бібліотеки cpu_features.	B
1.3	Покриття модульним тестуванням, інтеграції та тести на різних типах архітектур та операційних системах.	B
1.4	Публікування nuget пакету під різні архітектури та операційні системи.	H
1.5	Здатність виконання програмного коду на різних середовищах виконання платформи .NET, а саме середовищ .NET CORE та .NET Framework.	O
1.6	Створення технічної документації для зручного використання бібліотеки різними розробниками програмного забезпечення.	O

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі дипломної роботи було проаналізовано та розглянуто питання аукціонів у нашому житті, їхня класифікація, особливості та приклади програмного забезпечення, що використовується на сучасному ринку та призначення яких стосується даного дипломного проекту.

Згідно із наведених результатів аналізу можна зробити висновок, що більшість наявних аукціонних систем попри успішне використання у суспільстві не містить достатньої підтримки з боку адміністраторів. Тому головною метою розробки дипломного проекту було впровадження такої системи, яка містила б у собі найкращі дизайнерські та функціональні рішення, а також мінімізувала недоліки, які були знайдені у наведених вище прикладах.

З огляду на те, що необхідно створити онлайн аукціон із заданим графічним інтерфейсом для системи та здатністю до легкого розповсюдження, було прийняте рішення, що одним із найкращих засобів її реалізації є онлайн платформа у мережі Інтернет, технології розробки програмного забезпечення якої будуть детально розглянуті у наступному розділі.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Вибір необхідних технологій для розробки бібліотеки

Для створення бібліотеки було обрано C мова програмування для портування бібліотеки cpu_features та реалізації, також використовується C# яка викликає C функціонал і присутній функціонал тестування бібліотеки під різні архітектури та операційні системи для різних середовищ виконання.NET .

Rider – інтегроване середовище розробки для програмування на мові C#, дана середовище розробки надає можливість аналізу коду, влаштований інструмент рефакторингу, інструмент для аналізу покриття та запуску тестів.

У функціоналі портування використовуються різні компілятори тому в даній бакалаврської роботі будуть розглянуті для даній бібліотеці наступні C компілятори: GCC, MSVC, Clang.

2.1.1 Clang

Clang — це потужний та сучасний компілятор із відкритим для розробників вихідним кодом для різновидів мов програмування C, який бажає стати найкращим у розділі реалізації даних мов. Clang оснований на оптимізаторі та генераторі для коду LLVM, який дозволяє йому здійснювати забезпечення відмінної оптимізації та підтримки генерації коду для широко спектру цілей.

Clang було розроблено задля сприяння сімейства мов програмування мови C, яке включає у собі C, Objective-C, C++ та Objective-C++, а також величезну кількість діалектів із них.

Окрім того до цих основних мов та їхніх різновидів, Clang піддержує різноманітний спектр для мовних розширень. Дані розширення надаються задля зворотньої сумісності із GCC, Microsoft та рештою популярних компіляторів, а

також для поліпшення заданої функціональності за підтримки спеціальних особливостей та функцій технології Clang. Драйвер та мовні функції Clang умисно розроблені, так щоб їх можна було максимально зв’язати із компілятором GNU GCC, наскільки це програмно можливо, що у свою чергу полегшить міграцію із GCC на Clang. У великій кількості випадків програма “просто працює”. Clang також поширює альтернативний драйвер під назвою clang-cl, який сукупний із компілятором мови Visual C++, а саме cl.exe. Більше того до особливостей технології, Clang містить у собі ряд функцій, які можуть залежати від того, для якого типу архітектури процесора або операційної системи здійснюється компіляція.

2.1.2 Архітектурні особливості та обмеження Clang

Підтримка архітектури X86, як 32-розрядної, так і 64-розрядної, вважається достатньо надійною в Darwin (операційна система macOS), Linux, FreeBSD та Dragonfly BSD: вона була перевірена компіляцією для багатьох великих C, C++, Objective-C і Objective-C++ проектів.

З недоліків є те що у Clang компілятора не є повна сумісність з MSVC, наприклад на x86_64-mingw32 передача i128 типу (за значенням) несумісна з конвенцією викликів Microsoft x64, або що для операційної Windows ще не має повної інтеграції для розробки драйверів.

Визначено декілька рівнів мікроархітектури, визначених x86-64 psABI. Вони є кумулятивними в тому сенсі, що функції попередніх рівнів неявно включені в наступні рівні.

- -march=x86-64: CMOV, CMPXCHG8B, FPU, FXSR, MMX, FXSR, SCE, SSE, SSE2
- -march=x86-64-v2: CMPXCHG16B, LAHF-SAHF, POPCNT, SSE3, SSE4.1, SSE4.2, SSSE3
- -march=x86-64-v3: AVX, AVX2, BMI1, BMI2, F16C, FMA, LZCNT, MOVBE, XSAVE

•-march=x86-64-v4: AVX512F, AVX512BW, AVX512CD, AVX512DQ, AVX512VL

Підтримка ARM, зокрема версій ARMv6 та ARMv7, вважається надійною в операційній системі Darwin, а саме для iOS: вона була перевірена для здійснення операції правильної компіляції для багатьох розгорнутих кодових баз мов програмування C, C++, Objective-C та Objective-C++. Clang містить підтримку лише обмеженої кількості системних архітектур ARM. До прикладу, Clang ще не повністю здійснює підтримку версії ARMv5.

Підтримка PowerPC, особливо PowerPC64, вважається достатньо стабільною у операційних системах Linux та FreeBSD: вона була протестована для операції правильної компіляції у багатьох великих проектах, написаних мовою C і C++. У PowerPC, а саме 32-розрядної версії, все ще відсутні окремі функції, наприклад, PIC-код для платформ ELF [17].

2.1.3 GCC

Колекція компіляторів GNU містить інтерфейси для відомих мов програмування C, C++, Objective-C, Fortran, Ada, Go та D, а також певні бібліотеки для таких специфічних мов як libstdc++. Спочатку GCC був розроблений як компілятор задля операційної системи GNU. Система GNU була розроблена як безоплатне програмне забезпечення, тобто безкоштовне у тому сенсі, що воно містить повагу до свободи користувача.

2.1.4 MSVC

Microsoft Visual C++, скорочено MSVC – це відомий компілятор, який запускається лише на операційній системі, яка містить підтримку Microsoft Visual Studio для Windows.

2.1.5 CMake

CMake — це широко розширювана система із відкритим для розробників вихідним програмним кодом, яка здатна керувати процесом збирання в

операційній системі та вільним від компілятора способом. На відміну від більшості кросплатформних існуючих систем, CMake було розроблено саме для випадків використання у сполученні із рідним середовищем збирання. Прості файли у системі конфігурації, які розміщені в кожному вихідному переліку даних, так звані файли CMakeLists.txt, використовуються задля створення шаблонних файлів збирання, наприклад, make-файлів у системі Unix та проектів чи робочих просторів у системі Windows MSVC, що мають можливість використовуватися звичайним способом. CMake здатний генерувати рідне середовище збирання, яке буде компілювати вихідний шматок програми, створювати бібліотеки, генерувати обгортки та виконувані файли у вільних комбінаціях. CMake також має підтримку збирання на місці та поза місцем, тому може підтримувати декілька збирань із одного джерела. CMake окрім того підтримує статичні та динамічні збирання бібліотек. Ще однією досить приємною особливістю CMake є те, що він може створювати файли кешу, який є призначеним задля використання разом із графічним редактором. Наприклад, коли CMake починає запускатися, він знаходить файли, бібліотеки та виконувані файли і може побачити додаткові директиви збірки. Дана інформація збирається в кеші, який можна змінити користувачем перед створенням рідних файлів збирання [18].

CMake розроблено задля повної підтримки таких складних ієрархій каталогів та застосунків, які залежать від декількох типів бібліотек. Наприклад, CMake має підтримку проектів, які складаються із декількох наборів інструментів, тобто бібліотек, де кожен набір із інструментів може містити у собі кілька заданих каталогів, а програма у свою чергу, залежить від наборів існуючих інструментів та додаткового коду. CMake також має можливість обробляти такі ситуації, коли необхідно створити виконувані файли, для того щоб згенерувати програмний код, який згодом компілюється та зв’язується у кінцевому застосунку. Оскільки CMake містить відкритий вихідний програмний код та має простий, легко розширюваний дизайн, CMake можна

удосконалити за потреби для здійснення певної підтримки нових функцій. Використання CMake дуже просте. Процес збірки керується створенням одного чи декількох файлів CMakeLists.txt у кожному каталозі, включаючи підкаталоги, що у сукупності разом складають проект. Кожен CMakeLists.txt складається із однієї чи декількох команд. Кожна команда містить певну задану форму COMMAND (args…), де COMMAND — це назва зазначеної команди, а args — список доступних аргументів, який необхідно розділити пробілами. CMake надає велику кількість попередньо визначених команд, але якщо розробнику необхідно, то він має можливість додати свої власні команди для зручної роботи. Крім того, досвідчений користувач може розширити інші генератори із make-файлів для певної низки компіляторів чи операційних систем. Хоча Unix і MSVC++ наразі містять підтримку від постачальника, інші розробники додають іншу підтримку для компілятора чи операційної системи. Розробник можете вивчити наведену на порталі сторінку прикладів, щоб побачити більшу кількість деталей.

2.1.6 clang-tidy

Він є одною із частин низки інструментів clang, який транслює код C та C++ для бекенда LLVM. clang ще не досить відомий у системах HPC, як сімейства компіляторів GNU і Intel, але для налаштування він здатен бути цінним та корисним інструментом.

Він є одною із частин низки інструментів clang, який транслює код C та C++ для бекенда LLVM. clang ще не досить відомий у системах HPC, як сімейства компіляторів GNU і Intel, але для налаштування він здатен бути цінним та корисним інструментом.

Достатньо ретельні попередження clang-tidy, які генеруються clang і clang++. Якщо потрібно дізнатися всі попередження, про які можуть попередити компілятори, користувач може використовувати єдиний прапорець -Weverything. Однак не рекомендується використовувати це у виробництві,

оскільки користувач може здобути попередження, які несумісні з іншими параметрами командного рядка. Ціль -Weverything використовується лише для розробників компіляторів.

clang-tidy є небезкорисний інструмент, певна доля набору інструментів clang, який може визначити додаткові проблеми, про які компілятори не попередять.

Одним з недоліків є те, що для використання clang-tidy потрібно створювати компіляційну базу даних. Це є зручним, якщо CMake є інструментом для створення.

2.1.7 clang-format

clang-format містить підтримку двох способів для надання параметрів користувацького стилю: необхідно розробнику безпосередньо вказати конфігурацію стилю в опції командного рядку -style= чи використати команду -style=file та помістити конфігурацію стилей у файл .clang-format або _clang-format у каталозі заданого проекту.

При використанні -style=file, clang-format для кожного вхідного файлу буде здійснювати пошук файлів .clang-format, який розташований у найближчому батьківському каталозі вхідного файлу. Якщо використовується стандартне введення, то пошук здійснюється із поточного каталогу.

При використанні -style=file:<format_file_path>, clang-format для кожного вхідного файлу буде використовувати файл формату, який розташований за адресою <format_file_path>. Шлях може бути абсолютним чи відносним до робочого каталогу [19].

2.1.8 Bazel

Bazel є інструментом для збирання та тестування з відкритим програмним кодом, схожий до CMake, Maven, Make та Gradle. Bazel використовує мову складання високого рівня, доступним людині. Bazel піддержує проекти деякими

мовами та створює певні результати для декількох платформ. Bazel підтримує великі кодові проекти в декількох сховищах і велику кількість розробників.

Bazel пропонує наступні переваги:

• Мова складання високого рівня. Bazel використовує абстрактну, зрозумілу для людини мову, щоб описати властивості збірки вашого проекту на високому семантичному рівні. На відміну від інших інструментів, Bazel працює на основі концепцій бібліотек, двійкових файлів, сценаріїв і наборів даних, захищаючи вас від складності написання окремих викликів таких інструментів, як компілятори та компоновщики.

• Bazel швидкий і надійний. Bazel кешує всю раніше виконану роботу та відстежує зміни як вмісту файлу, так і команд збірки. Таким чином, Базель знає, коли щось потрібно перебудувати, і перебудовує лише це. Щоб ще більше прискорити збірку, розробник може налаштувати проект так, щоб він збирався паралельно та поступово.

• Bazel є мультиплатформенним. Bazel працює підтримує Linux, macOS та Windows операційні системи. Bazel здатен створювати двійкові файли та пакети для розгортання для кількох платформ, включаючи настільні, серверні та мобільні, з одного проекту.

• Bazel ваги. Bazel зберігає гнучкість під час обробки збірок із понад 100 тис. вихідних файлів. Bazel працює з кількома сховищами та базами користувачів у десятках тисяч.

• Bazel розширюється. Підтримується багато мов, і користувач може розширити Bazel для підтримки будь-якої іншої мови або фреймворку.

Для того щоб створити або протестувати проект за допомогою Bazel, розробники зазвичай роблять наступне:

• Налаштувати Bazel. Завантажити та встановити Bazel.

• Налаштувать робочу область проекту, яка являє собою каталог, де Bazel шукає вхідні дані збірки та файли BUILD і де зберігає вихідні дані збірки.

• Написати файл BUILD, який вказує Bazel, що і як побудувати.

• Розробник пише свій файл BUILD, оголошуючи цілі збірки, використовуючи Starlark, мову, специфічну для домену.

• Ціль збірки визначає набір вхідних артефактів, які буде створено Bazel, а також їх залежності, правило збірки, яке Bazel використовуватиме для його створення, а також параметри, які налаштовують правило збірки.

• Правило збірки визначає інструменти збірки, які використовуватиме Bazel, такі як компілятори та компонувальники, а також їх конфігурації. Bazel постачається з низкою правил збірки, що охоплюють найпоширеніші типи артефактів на підтримуваних мовах на підтримуваних платформах.

• Запустити Bazel з командного рядка. Bazel розміщує ваші результати в робочому просторі.

• Окрім побудови, розробник може використовувати Bazel для запуску тестів і запитів до збірки, щоб відстежувати залежності у вашому коді.

Під час виконання збірки або тесту Bazel виконує наступне:

• Завантажує файли BUILD, що відповідають цілі.

• Аналізує вхідні дані та їх залежності, застосовує вказані правила збірки та створює графік дій.

• Виконує дії побудови на входах, поки не будуть створені остаточні вихідні дані збірки.

Оскільки вся попередня робота по збірці кешується, Bazel може ідентифікувати та повторно використовувати кешовані артефакти і лише перебудувати або перевіряти те, що було змінено. Для подальшого забезпечення коректності розробник може налаштувати Bazel на герметичний

запуск збірок і тестів за допомогою пісочниці, мінімізуючи перекіс і максимізуючи відтворюваність [20].

Графік дій представляє артефакти побудови, зв’язки між ними та дії збірки, які виконуватиме Bazel. Завдяки цьому графіку Bazel може відстежувати зміни вмісту файлу, а також зміни дій, таких як команди збірки чи тестування, і знати, які роботи зі збирання були виконані раніше. Графік також дозволяє легко відстежувати залежності у вашому коді.

2.1.9 .NET Standard

Коли Microsoft вперше випустила .NET Core 1.0, він містив лише невеликий вибір API, доступних на той час у .NET Framework. Прагнучи спростити написання бібліотек, які можна було б використовувати як у .NET Framework, так і в .NET Core (без необхідності багатоцільового використання), вони представили концепцію .NET Standard.

На відміну від .NET Core і .NET Framework, які є платформами, які розробник може завантажити та запустити, .NET Standard — це лише визначення інтерфейсу. Кожна версія .NET Standard містить список API, які платформа повинна підтримувати, щоб реалізувати цю версію .NET Standard.

Кожна версія .NET Standard є суворим наднабором попередніх версій, тому всі API з попередніх версій доступні в пізніших версіях. Аналогічно, якщо платформа реалізує, скажімо, .NET Standard 1.4, то за визначенням вона також реалізує .NET Standard 1.0-1.3:

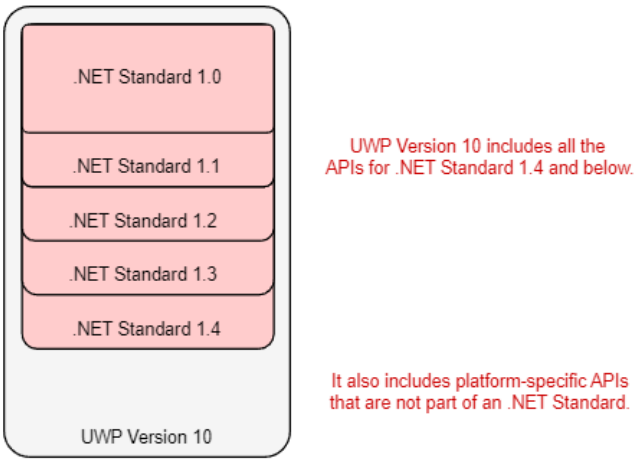


Рисунок 2.1 - Схема підтримок версій .NET Standard

.NET Standard 2.0 був випущений разом із .NET Core 2.0, і представляє багато нових API в порівнянні з попередніми версіями. .NET Standard був сильно змодельований на поверхні .NET Framework 4.6.1. Результатом було те, що розробник повинен мати можливість написати бібліотеку, націлену на .NET Standard 2.0, і вона повинна працювати на .NET Framework 4.6.1+ і .NET Core 2.0+.

2.1.10 Мета .NET Standard

Коли .NET Framework був випущений, він служив загальною платформою розробки додатків для настільних комп'ютерів Windows і серверних середовищ. Після цього для мобільних платформ була представлена нова версія .NET Framework під назвою .NET Compact Framework. Аналогічно, впровадження Silverlight, платформ додатків Windows 8 тощо додало свої власні особливості .Net Framework та середовища виконання. Таким чином, було багато вертикалей для платформ .NET з подібними бібліотеками базового класу, які розвивалися окремо і поставлялися окремими командами в Microsoft. Для розробників для спільного використання бібліотечного коду або компонента між цими платформами потрібно повторно реалізувати та створити його для певної платформи, хоча вони надавали ту саму функцію. Для

вирішення цієї проблеми було введено новий спосіб компіляції бібліотек, який називається Portable Class Library [21].

2.1.11 Портативна бібіліотека класів

Для вирішення цієї проблеми спільного використання коду введені портативні бібліотеки класів. Це проект бібліотеки для повторного використання, де розробник вказує платформи .NET, на які користувач планує орієнтуватися під час створення бібліотеки. Цей проект дозволяє розробникам створювати портативні бібліотеки, які надають лише ті бібліотеки базового класу (або API), які є спільними для вибраних платформ. Інструменти Visual Studio керують цим за розробників. Наприклад, якщо розробник вибере цільові платформи як .NET Framework і Silverlight, бібліотека зможе використовувати лише ті API, які зазвичай доступні на обох цих платформах.

Інструменти Visual Studio керують цим за розробників. Наприклад, якщо розробник вибере цільові платформи як .NET Framework і Silverlight, бібліотека зможе використовувати лише ті API, які зазвичай доступні на обох цих платформах.

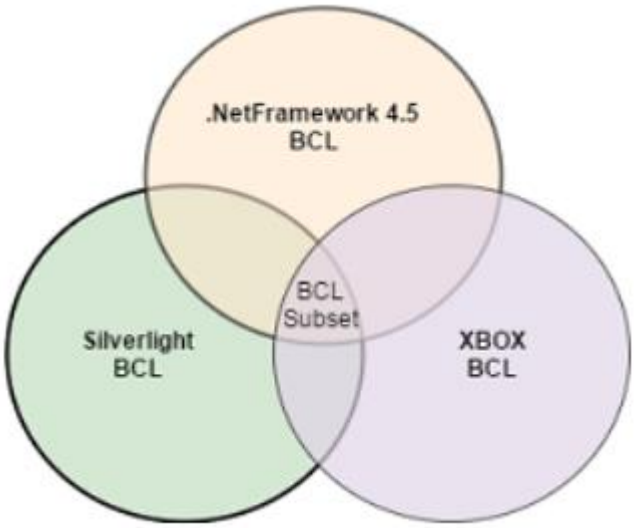


Рисунок 2.2 - Діаграма Венна

Припускаючи, що розробник створює проект переносної бібліотеки класів, націлений на платформи .NET 4.5, Silverlight і Xbox, проект бібліотеки може використовувати лише загальні бібліотеки (підмножина BCL на діаграмі Венна), які отримані перетином усіх платформ BCL. Отже, чим більше платформ, тим менше поверхня (кількість) загального API.

Для цього у Visual Studio 2012 є тип проекту під назвою Portable Class Library project. Вказано нижче:

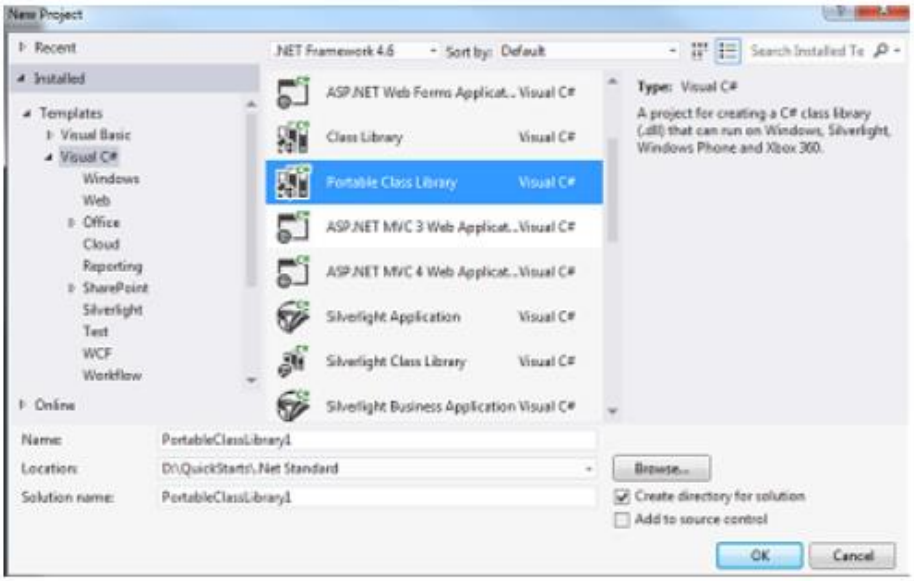


Рисунок. 2.3 - Створення проекту типу Portable Class Library

Для конфігурації портивної бібліотеки необхідно задати відповідно параметри:

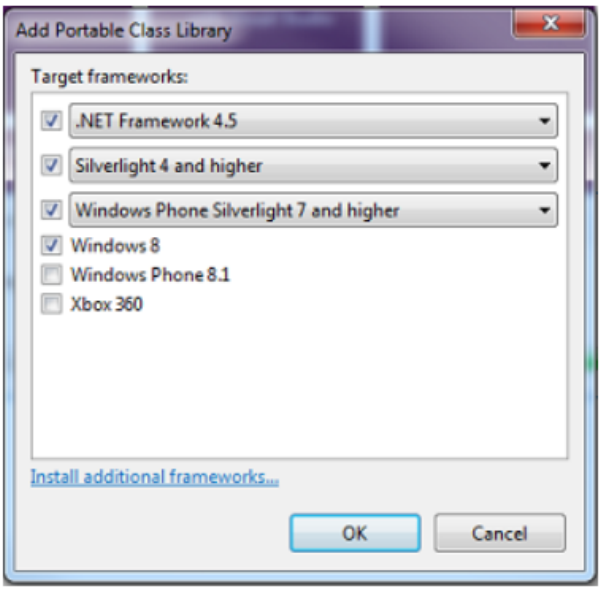


Рисунок 2.4 - Конфігурація параметрів для бібліотеки

Наведений вище вибір створить клас портативної бібліотеки, який дозволить використовувати лише API, які є загальними для .NET Framework 4.5.Silverlight 4, Windows Phone Silverlight 7 та Windows 8.

Хоча портативні бібліотеки дозволяли обмін кодом між кількома платформами, вони все одно мали недоліки, наприклад, додавання підтримки нової платформи вимагає перекомпіляції з новим набором цільових платформ і видалення посилань API, які не є частиною нового перетину бібліотек базового класу. Portable Class Libraries також не має жодного контролю над цільовими платформами, які все ще розвивалися окремо.

2.1.12 Вступ .NET Standard

Розробники .NET Standard вирішили все це по-іншому, надав специфікацію API, яку всі платформи повинні реалізувати, щоб залишатися сумісним з .NET Standard. Це об’єднало бібліотеки базового класу різних платформ .NET і проклало шлях для спільного використання бібліотек, а також привело до централізації еволюції BCL, як показано нижче.

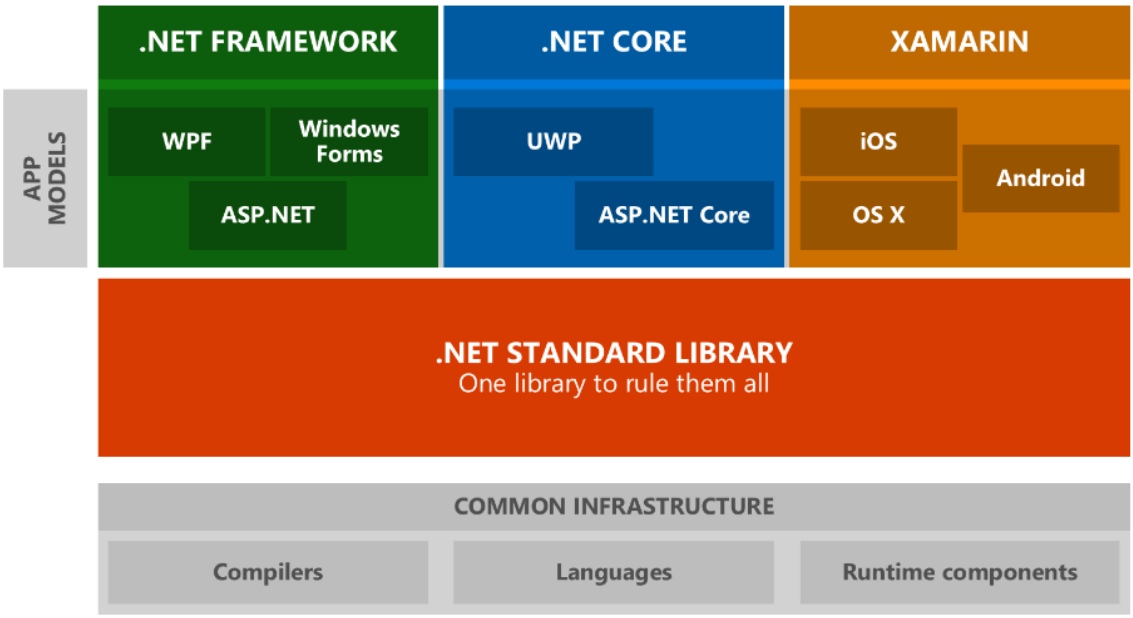


Рисунок 2.5 - Схема сумісності бібліотек .NET

Версія .NET Standard подібна до версій фреймворку. Поточна версія .NET Standard — 1.6, а .NET Core 1.0, 1.1 підтримує .NET Standard 1.6. Чим вище номер версії, тим вище кількість API, які він підтримує. Наразі .NET Core є свого роду еталонною реалізацією для специфікації .NET Standard. Нова версія .NET Standard супроводжується випуском .NET Core. Майбутня версія .NET Standard — 2.0, а майбутня версія .NET Core, ймовірно, підтримуватиме 2.0. Перегляньте наведену нижче матрицю .NET Standard та підтримуваних платформ, щоб зрозуміти, яка версія платформ реалізує яку версію .NET

Standard.

1.0 1.1 1.2 1.3 1.4 1.5 1.6 **2.0** 2.1

.NET Standard 2.0 has 32,638 of the 37,118 available APIs.

.NET implementation	Version support
.NET and .NET Core	2.0, 2.1, 2.2, 3.0, 3.1, 5.0, 6.0
.NET Framework [1]	4.6.1 [2], 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8
Mono	5.4, 6.4
Xamarin.iOS	10.14, 12.16
Xamarin.Mac	3.8, 5.16
Xamarin.Android	8.0, 10.0
Universal Windows Platform	10.0.16299, TBD
Unity	2018.1

Рисунок 2.6 - Діаграма підтримуваних платформ для .NET Standard

Наведена вище матриця підтримується та оновлюється в репозиторії .NET Standard GitHub тут.

Наприклад, щоб розробити бібліотеку, яка підтримує .NET Framework 4.5.1 і .NET Core 1.0, нам потрібно орієнтуватися на .NET Standard 1.2, тобто на найнижчу версію .NET Standard, яку реалізують 2 фреймворки.

Тепер у пакетах NuGet є інформація про підтримку версії .NET Standard на Nuget.org. Наприклад, на даний момент JSON.Net підтримує .NET Standard,

Наразі фреймворк .NET Core має дуже обмежену поверхню API в порівнянні з повним .NET Framework. Отже, дуже очікується випуск .NET Standard 2.0, який додасть майже всі API, які повністю підтримує .NET Framework, до .NET Core.

2.1.13 Частини .NET Standard

.NET Standard має лише бібліотеки базових класів, які можна спільно використовувати між платформами, за винятком кількох випадків, коли виникне виняток NotImplemented. API для конкретної моделі програми, які зазвичай називають Framework Class Libraries (FCL), як-от WebForms, Windows Forms, WPF, WCF тощо, і API для ОС, як-от Registry, AppDomain тощо, виходять із специфікації .NET Standard, щоб загальний BCL залишався платформою агностик.

2.1.14 Xamarin

Xamarin - це фреймворк для кросплатформної розробки мобільних програм (iOS, Android, Windows Phone) з використанням мови C#. Ідея дуже проста. Ви пишете код своєю улюбленою мовою, із застосуванням всіх звичних для розробника мовних фіч нібито LINQ, лямбда-виразів, Generic та async. При цьому ви маєте повний доступ до всіх можливостей SDK платформи і рідного механізму створення UI, отримуючи на виході програму, яка, строго кажучи, нічим не відрізняється від нативних і (принаймні, запевняє), не поступається їм у продуктивності.

Фреймворк складається з кількох основних частин:

- Xamarin.IOS – бібліотека класів для C#, що надає розробнику доступ до iOS SDK;
- Xamarin.Android - бібліотека класів для C #, що надає розробнику доступ до Android SDK;
- Компілятори для iOS та Android;
- IDE Xamarin Studio;
- Плагін для Visual Studio.

Певний час тому досить широку популярність здобули ряд фреймворків (наприклад PhoneGap), які пропонують розробку кросплатформових мобільних додатків на HTML5 з використанням JavaScript. Ідея полягає в тому, що програма розробляється як звичайний сайт для мобільних пристроїв з

використанням відповідних js-бібліотек, наприклад, Jquery Mobile. Потім все це упаковується в якийсь контейнер, який для користувача виглядає як нативна програма. Мінуси цих кадрів очевидні: по-перше, ви не маєте доступу до нативних елементів UI. Тобто навіть якщо ви хочете використовувати стандартну кнопку "Назад" для iPhone, ви повинні її намалювати та зверстати. По-друге, ви отримуєте урізаний та узагальнений API для роботи з платформою. Таким чином, ті чи інші фічі, притаманні якійсь окремій платформі, будуть вам недоступні. Ну і третє і найважливіше – така програма фізично запускається всередині браузера телефону (точніше всередині контролю WebView). Не потрібно розписувати довго, що це означає: низька продуктивність (особливо «хороший» WebView на старих версіях Android) і величезні проблеми з відображенням (ну, панове, це браузер). Хоча, звичайно, у певних випадках ці фреймворки можуть бути дуже доречними.

Xamarin заснований на open-source реалізації платформи .NET - Mono. Ця реалізація включає власний компілятор C#, середовище виконання, а також основні .NET бібліотеки. Мета проекту – дозволити запускати програми, написані на C#, на операційних системах, відмінних від Windows – Unix-системах, Mac OS та інших. Важливо, що розробкою Xamarin займаються самі люди, що й розробкою Mono. І це не Microsoft з усіма витікаючими плюсами та мінусами.

З точки зору виконання програм між iOS і Android є одна ключова відмінність - спосіб їхньої попередньої компіляції. Як відомо, для виконання програм в Android використовується віртуальна Java-машина Dalvik. Нативні програми, які пишуться на Java, компілюються в якийсь проміжний байт-код, який інтерпретується Dalvik`ом команди процесора в момент виконання програми (тобто аналогічно тому, як працює CLR в .NET). Це так звана Just-in-time компіляція (компіляція на льоту). У iOS використовується інша модель компіляції - Ahead-of-Time (компіляція перед виконанням). Xamarin враховує цю різницю, надаючи окремі компілятори для кожної з цих платформ, які

дозволяють на виході отримувати справжні, нативні програми, які виконуються поза контекстом браузера і можуть використовувати всі апаратні та програмні ресурси платформи.

2.1.15 Документація та ком'юніті

Xamarin має відмінну документацію, що містить докладні посібники, сніпети, а також велику базу прикладів. Документація безпосередньо по всіх класах бібліотек Monotouch та Monodroid є частиною загальної документації Mono. Але, на жаль, цього все одно недостатньо, щоб покрити весь низку питань, які можуть виникати в процесі розробки. У Xamarin існує ком'юніті розробників, яке сконцентовано на офіційному форумі та на StackOverlow. Активністю та ініціативністю людей у ком'юніті похвалитися не велика. У цьому плані неоціненну допомогу надала приватна технічні підтримка з інженерами електронною поштою, доступна в business-ліцензії. Відповідають, як правило, протягом кількох годин і не стандартними відписками «спробуйте вимкнути та включити», а справді розбираються в проблемі та допомагають її вирішити. Слід розуміти, що база питань і відповідей, накопичена для нативної розробки набагато ширше, ніж для Xamarin, тому, як би розобнику не хотілось, користовучу все ж таки доведеться розібратися в специфічному синтаксисі objective-c, щоб розуміти приклади коду на тому ж StackOverflow. Крім того, це відкриє вам доступ до прочитання та розуміння офіційної документації для платформи, що на певному етапі може стати дуже важливим. З іншого боку, в цьому є й позитивний момент: отримавши такий базис, вам буде простіше перейти до нативної розробки за необхідності.

2.1.16 Xamarin Studio

Xamarin Studio - кросплатформова IDE, яка працює як на Mac OS X, так і на Windows. На вигляд ця вона виглядає дуже простою і доброзичливою, проте

за зовнішньою простою ховається досить потужний інструмент, який включив у себе безліч фіч, звичних нам у Visual Studio і Resharper:

Приємне підсвічування синтаксису;

- Автодоповнення коду (включаючи можливість одночасного імпорту namespaces);
- Зручний універсальний пошук за назвами файлів, типами, членами класів тощо;
- Розвинені можливості навігації за проектом: швидкий перехід до опису класу, перехід до базового класу, список місць використання класу тощо;
- Різні механізми рефакторингу та швидка підказка (як alt+Enter у Resharper);
- Досить розвинені механізми дебага, включаючи стеження, перегляд поточного значення змінної при наведенні, візуалізацію потоків та аналог Immediate window VS;
- Вбудована інтеграція із системами контролю версій: SVN, Git та TFS (для TFS, щоправда, потрібні сторонні утиліти);

Xamarin пропонує можливість вести розробку в Visual Studio після встановлення спеціального плагіна, який доступний у business-ліцензії (на момент виходу статті – 999 $), але є місяць тріалу. Плюси очевидні: ви стаєте розробником мобільних програм, не змінюючи місця дислокації, і можете використовувати всю важку артилерію в особі Resharper та інших ваших улюблених плагінів. Після встановлення плагіна для Visual Studio вам потрібно налаштувати з'єднання з Mac, яке буде використано при запуску проекту на виконання. Тобто. після запуску, програма автоматично пересилається на Mac, де компілюється і завантажується або на симулятор або на пристрій, при цьому процес процес налагодження, розстановка брейкпоінтів і т.д. буде відбуватися у Visual Studio.

Варіантів роботи у Visual Studio кілька. Або Розробнику слідує використовувати віртуальну машину всередині Mac (наприклад Parallels), куди потрібно завантажувати Windows і Visual Studio, або використовувати дві різні фізичні машини, причому використовувати один Mac для кількох PC-розробників важко, т.к. налагодження вимагає маніпуляцій із симулятором. І останній варіант – використовувати віртуальну машину з Mac OS X (так званий hackintosh). Цілком собі життєздатний варіант, хоч і є деякі обмеження. Наприклад, в Xcode доведеться переміщатися Storyboard тільки з використанням смуг прокручування, т.к. windows не дуже схожа на справжню мишу від Mac з усіма витікаючими.

ВИСНОВОК ДО РОЗДІЛУ 2

В розділі 2 було проаналізовано та розглянуто ряд технологій для безпосередньої реалізації проекту мовою С. Мову програмування С та, вчасності бібліотеку cpu_features було обрано з декількох причин, однією з яких була умова запуску вбудованих пристроїв з мінімальним обсягом використованої оперативної пям'яті при виконанні.

В даному дипломному проекті було використано три С-компілятори:

- GCC
- MSVC
- Clang

Даний набір компіляторів дозволяє використовувати функціонал бібліотеки максимально гнучко, не зважаючи на платформу. GCC або Gnu Compiler Collection представляє собою сімейство компіляторів з відкритим програмним кодом для багатьох мов програмування (C, C++, Ada, Go, Objective-C тощо) та э стандартним компілятором серед Unix-систем. MSVC є компілятором від Microsoft, що забезпечує роботу під Windows ОС. І, Clang є кросплатформеним компілятором загального призначення, який має велику підтримку та кількість діалектів, що підтримуються на сьогодні.

Під час компіляції важливо правильно збудувати проект, вказавши усі залежності та вихідні файли для зборки. На даному етапі використовується система зборки. Вибір системи зборки пав на найпопулярнішу утиліту з відкритим програний кодом CMake. Дана система представляється у вигляді конфігураційних файлів, що задають ієрархію каталогів та залежності між ними. Для уніфікації усіх файлів з вихідним програмним кодом використовується утиліта clang-format та clang-tidy в якості аналізатора та форматування коду.

РОЗДІЛ 3
РОЗРОБКА БІБЛІОТЕКИ ІДЕНТИФІКАЦІЇ ПРОЦЕСОРА

3.1 Проектування загальної структури бібліотеки

Структура проекту забезпечує офіційне середовище для використання менеджерами проекту для впливу на членів команди, щоб вони зробили все можливе у виконанні своїх обов'язків і завдань. Цю структуру слід спланувати, щоб допомогти побудувати співпрацю між членами команди економічно ефективним способом з мінімальним перекриттям і дублюванням зусиль.

Однією з головних переваг цієї структури є те, що вона дозволяє команді бути в курсі щоденних завдань і спілкуватися, оскільки вони належать до однієї організаційної команди та мають одного адміністратора чи менеджера. Ще одна головна перевага полягає в тому, що група чи команда володіють потужним почуттям ідентичності, оскільки всі вони зосереджені на досягненні однієї мети та успішному виконанні проекту.

З точки зору управління, ця структура дозволяє організації краще використовувати ресурси для проекту, оскільки нагляд зосереджений на одній команді. Менша ймовірність дублювання діяльності та ресурсів.

Список папок bin/include/lib/doc/src/build/data/config і так далі до нескінченного списку є набором речей з різних місць. Оскільки розробники можуть вільно розгортати свої власні застосунки, це дійсно залежить від типу проекту, мови, платформи чи IDE, з якими вони працюють.

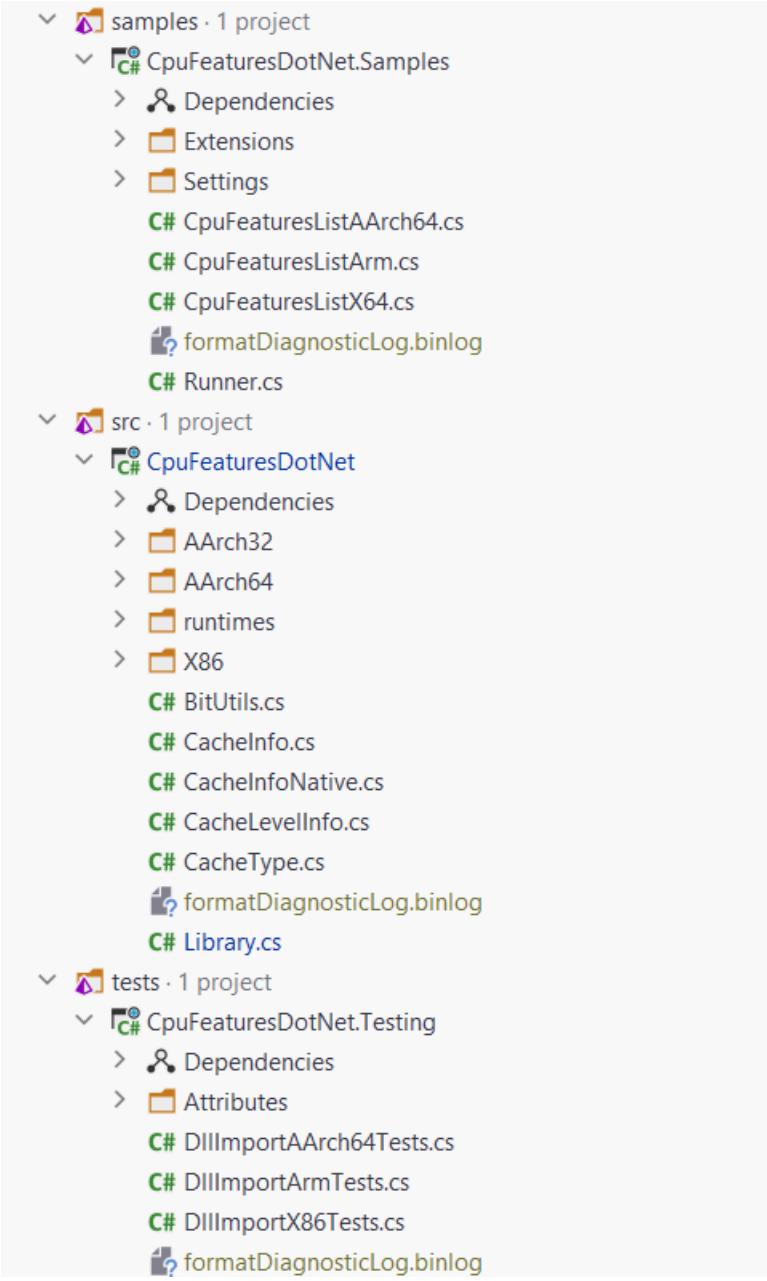


Рисунок 3.1 - Структура бібліотеки cpu_features.NET

Зокрема, GNU Linux, який створив структури для своїх пакетів[22] і має багато бюрократії щодо того, що в них має бути. Це, у свою чергу, вплинуло на інших, і деякі з них просто мали сенс, тому люди не відставали від цього. Примітно, що розробники Microsoft C++ не дотримуються цього, зазвичай ви бачите .sln (файли рішень, створені Visual Studio) тощо, однак якщо автори також розробляють для Linux, ви можете побачити деякі запозичення від

способу організації Linux через їхні звички та знайомство з цією структурою папок. Що стосується папок «include», «bin» і «lib», така організація також, як правило, справедлива для DLL, і вся річ «lib» в основному запозичена.

Загалом, що стосується папок проекту, це виглядає так:

- bin/ папка, яка містить скомпільований файл .DLL або (іноді) .lib або виконувану програму або файл .exe — «двійковий»
- include/ папка, яка містить загальнодоступний .h (заголовні файли) для бібліотеки, яка буде включена в іншу програму, необхідну для використання бібліотеки або DLL
- lib/ папка, яка містить статичні файли lib (попередньо скомпільований код) і є частиною бібліотеки
- doc/ всі ці чудові посібники, які програмісти ненавидять писати, або створені з результатів вихідної утиліти документації, як-от Doxygen, чи файлу README, чи файлів .man для команди Linux «man», або автономного веб-сайту, щоб ви могли бачити .html файли
- build/ папка, яка іноді використовується для зберігання сценаріїв збірки, напівкомплектованого коду, інших речей, пов'язаних з процесом компіляції, зазвичай вони генеруються утилітою або компілятором, ви можете побачити тут .objs, але можете ні. Існують інші назви цієї папки на кожній платформі, і іноді ви можете знайти онлайн-документацію з описом її структури, як для Android
- src/ тут знаходиться вихідний код, люди редагують цей матеріал, і він може мати підпапки
- data/ деякі програми постачаються з «зразками даних» або «тестовими даними», які зазвичай знаходяться тут і доступні програмі в папці bin/
- examples/ деякі бібліотеки постачаються з набором програм, які тестують різні «одиниці» «системи», а також демонструють використання

- contrib/ це раніше означало «фрагменти, внесені користувачем» або вихідний код, наданий первинному автору або авторам для використання або для розширення проекту з відкритим кодом (або навіть із закритим вихідним кодом), хоча ідея «внесок» досягла популярності Пік був давним-давно, в той час, коли не у всіх була утиліта керування версіями джерел, як-от git

У C++ також є різні типи файлів, а також файли, пов'язані з компілятором, із певними розширеннями (хоча ідея «розширення» файлу з часом розмивалася, на даний момент ми говоримо про MIME/типи) у будь-якому конкретному випадку. демонструвати:

.cpp і .h поєднуються, .h є «заголовком» — розроблений таким чином, щоб дозволити програмістам випускати libs та .h файли без вихідного коду, створювати власні або закриті бібліотеки, які можна поширювати без вихідного коду.

C++ більше не вимагає від розробника використання .cpp, тож якщо користувач бачить .hpp, в основному бачить комбінацію файлів .cpp і .h в одному файлі. Зазвичай те, що з'являється в .cpp, знаходиться в кінці файлу .hpp, а частина .h – на початку файлу .hpp. Як правило, .hpp є чисто C++, тому що він не допускає функцій у стилі C:

- Makefile — цей файл, хоча він може бути написаний від руки, зазвичай генерується за допомогою autoconf і може бути пов'язаний з командою «configure» — цілим світом і сам по собі і не пов'язаний з Microsoft Visual Studio — використовується для «побудови» програми (автоматизація компіляції та зв'язування), як правило, з використанням компіляторів, таких як gpp, g++ або gcc (всі, по суті, той самий компілятор) і пов'язаний компілятор GNU. У Microsoft Visual Studio є інший світ і інший компілятор/компонувальник/набір параметрів, про які варто турбуватися. У наступні роки розробники

Linux створили версії autoconf, які також генерують сценарії збірки Visual Studio.

- Користувач може побачити інші речі, але це основні докази з вихідного коду. Існує файл .suo, який є необов'язковим для Visual Studio (він містить базу даних intellisense). І в Linux, і в Windows є ряд файлів .obj (компільований вихідний код, який ще не пов'язано), є .make і .bat і .sh і .doc і .html і купа інших специфічних файлів на різних платформах, наприклад у Windows користувач може побачити файл «.pdb», який є інформацією про символ налагодження для програми — а в Linux зазвичай є сценарії, які не мають конкретного розширення файлу, або вони мають розширення .sh — і якщо розробник бачить інші файли, це можуть бути файли конфігурації, або вони можуть бути документацією або тестовими даними. Інші файли можуть бути іншими.
- Як правило, файл .pro означає, що програма написана на QT, а QT має власні структури папок, які також можна налаштувати. Файл .pro – це файл, який користувач завантажує в QT Creator. Користувач може побачити інші файли, як-от «.qmake», «.ui», «.qc» тощо, усі для QT, яка зараз належить через придбання Nokia, яка придбала The Qt Company, Microsoft.

3.2 Marshaling

Маршалінг взаємодії керує тим, як дані передаються в аргументах методу і повертаються значення між керованою і некерованою пам'яттю під час викликів. Маршалінг взаємодії — це діяльність під час виконання, яку виконує служба маршалінгу загальномовного середовища виконання.

Більшість типів даних мають спільні представлення як в керованій, так і в некерованій пам’яті. Розпорядник взаємодії обробляє ці типи за вас. Інші типи можуть бути неоднозначними або взагалі не представлені в керованій пам'яті.

Рисунок 3.2 - Приклад маршалінгу у бібліотеці для виклику С коду для отримання інформації архітектури X86

Неоднозначний тип може мати або кілька некерованих уявлень, які співвідносяться з одним керованим типом, або відсутню інформацію про тип, наприклад розмір масиву. Для неоднозначних типів маршаллер забезпечує представлення за замовчуванням та альтернативні уявлення, де існує декілька представлень. Розробник може надати розпорядку чіткі інструкції щодо того, як маршалювати неоднозначний тип.

Середовище виконання загальної мови надає два механізми взаємодії з некерованим кодом:

- Виклик платформи, який дозволяє керованому коду викликати функції, експортовані з некерованої бібліотеки.
- Взаємодія COM, яка дозволяє керованому коду взаємодіяти з об’єктами моделі компонентних об’єктів (COM) через інтерфейси.

І виклик платформи, і взаємодія COM використовують маршалінг взаємодії для точного переміщення аргументів методу між викликом і викликаним і назад, якщо потрібно. Виклик методу платформи переходить від керованого до некерованого коду, за винятком випадків, коли задіяні функції зворотного виклику. Незважаючи на те, що виклики виклику платформи можуть надходити лише від керованого коду до некерованого, дані можуть надходити в обох напрямках як вхідні або вихідні параметри. Виклики методів взаємодії COM можуть надходити в будь-якому напрямку.

3.2.2 Marshaling Strings

Викликання платформи копіює параметри рядка, перетворюючи їх із формату .NET Framework (Unicode) у некерований формат (ANSI), якщо потрібно. Оскільки керовані рядки є незмінними, виклик платформи не копіює їх назад із некерованої пам’яті в керовану пам’ять, коли функція повертається.

3.2.3 Marshaling Classes, Structures та Unions

Класи та структури подібні в .NET Framework. Обидва можуть мати поля, властивості та події. Вони також можуть мати статичні та нестатичні методи. Одна помітна відмінність полягає в тому, що структури є типами значень, а класи - типами посилання.

У наступній таблиці наведено перелік параметрів сортування для класів, структур і об’єднань; описує їх використання; і надає посилання на відповідний зразок виклику платформи.

Таблиця 3.1 – Опис функціональних вимог до дипломного проекту

Тип	Опис
Клас за значенням.	Передає клас із цілими членами як параметр In/Out, як керований випадок.
Структура за значенням.	Передає структури як параметри In.

Продовження таблиці 3.1

Тип	Опис
Конструкція з вкладеними структурами (сплощена).	Передає структури як вхідні/вихідні параметри.
Структура з вказівником на іншу структуру.	Передає клас, який представляє структуру з вкладеними структурами в некерованій функції. У керованому прототипі структура сплощена до однієї великої конструкції.
Масив структур із цілими числами за значенням.	Передає структуру, яка містить покажчик на другу структуру як член.
Масив структур із цілими числами та рядками за посиланням.	Передає масив структур, які містять цілі числа та рядки як вихідний параметр. Викликана функція виділяє пам'ять для масиву.
Союзи з типами цінностей.	Передає об’єднання з типами значень (цілочисельні та подвійні).
Союзи зі змішаними типами.	Передає об’єднання зі змішаними типами (цілочисельним і рядковим).
Структура з макетом для платформи.	Передає тип з визначеннями рідної упаковки.
Нульові значення в структурі.	Передає клас із цілими членами як параметр In/Out, як керований випадок.

Цей зразок демонструє, як передати структуру, яка вказує на другу структуру, передати структуру з вбудованою структурою та передати структуру з вбудованим масивом.

Зразок Structs використовує такі некеровані функції, показані з їх оригінальним оголошенням функції:

Цей зразок демонструє, як передати структуру, яка вказує на другу структуру, передати структуру з вбудованою структурою та передати структуру з вбудованим масивом. Зразок Structs використовує такі некеровані функції, показані з їх оригінальним оголошенням функції:

- TestStructInStruct експортовано з PinvokeLib.dll:

```
C++

int TestStructInStruct(MYPERSON2* pPerson2);
```

Рисунок 3.3 - Оголошення функції TestStructInStruct

- TestStructInStruct3 експортовано з PinvokeLib.dll:

```
C++

void TestStructInStruct3(MYPERSON3 person3);
```

Рисунок 3.4 - Оголошення функції TestStructInStruc3

- TestArrayInStruct експортовано з PinvokeLib.dll:

```
C++

void TestArrayInStruct(MYARRAYSTRUCT* pStruct);
```

Рисунок 3.5 - Оголошення функції TestArrayInStruct

PinvokeLib.dll — це користувацька некерована бібліотека, яка містить реалізації для раніше перерахованих функцій і чотирьох структур: MYPERSON, MYPERSON2, MYPERSON3 і MYARRAYSTRUCT.

```
C++

typedef struct _MYPERSON
{
   char* first;
   char* last;
} MYPERSON, *LP_MYPERSON;

typedef struct _MYPERSON2
{
   MYPERSON* person;
   int age;
} MYPERSON2, *LP_MYPERSON2;

typedef struct _MYPERSON3
{
   MYPERSON person;
   int age;
} MYPERSON3;

typedef struct _MYARRAYSTRUCT
{
   bool flag;
   int vals[ 3 ];
} MYARRAYSTRUCT;
```

Рисунок 3.5 - Оголошення структур

Керовані структури MyPerson, MyPerson2, MyPerson3 і MyArrayStruct мають такі характеристики:

- MyPerson містить лише члени рядка. Поле CharSet встановлює рядки у формат ANSI при передачі некерованій функції.
- MyPerson2 містить IntPtr до структури MyPerson. Тип IntPtr замінює вихідний покажчик на некеровану структуру, оскільки програми .NET Framework не використовують покажчики, якщо код не позначено як небезпечний.
- MyPerson3 містить MyPerson як вбудовану структуру. Структуру, вбудовану в іншу структуру, можна вирівняти, помістивши елементи вбудованої структури безпосередньо в основну структуру, або її можна залишити як вбудовану структуру, як це зроблено в цьому зразку.
- MyArrayStruct містить масив цілих чисел. Атрибут MarshalAsAttribute встановлює значення перерахування UnmanagedType на ByValArray, яке використовується для вказівки кількості елементів у масиві.

До всіх структур у цьому зразку застосовується атрибут StructLayoutAttribute, щоб забезпечити послідовне розташування елементів у пам’яті в тому порядку, в якому вони з’являються.

Клас NativeMethods містить керовані прототипи для методів TestStructInStruct, TestStructInStruct3 і TestArrayInStruct, викликаних класом App. Кожен прототип оголошує один параметр, а саме:

- TestStructInStruct оголошує посилання на тип MyPerson2 як свій параметр.
- TestStructInStruct3 оголошує тип MyPerson3 як свій параметр і передає параметр за значенням.
- TestArrayInStruct оголошує посилання на тип MyArrayStruct як свій параметр.

Структури як аргументи методам передаються за значенням, якщо параметр не містить ключове слово ref (ByRef у Visual Basic). Наприклад, метод TestStructInStruct передає посилання (значення адреси) на об’єкт типу MyPerson2 в некерований код. Щоб маніпулювати структурою, на яку вказує MyPerson2, зразок створює буфер заданого розміру та повертає його адресу шляхом поєднання методів Marshal.AllocCoTaskMem і Marshal.SizeOf. Далі зразок копіює вміст керованої структури в некерований буфер. Нарешті, у зразку використовується метод Marshal.PtrToStructure для маршування даних з некерованого буфера в керований об’єкт і метод Marshal.FreeCoTaskMem для звільнення некерованого блоку пам’яті.

3.3 Створення нативних пакетів

Власний пакет містить власні двійкові файли замість керованих збірок, що дозволяє використовувати його в проектах C++ (або подібних). (Див. власні пакети C++ у розділі Споживання.)

Щоб бути витратним матеріалом у проекті C++, пакет повинен бути націлений на рідну структуру. Наразі з цією структурою не пов’язано жодних версій, оскільки NuGet однаково обробляє всі проекти C++.

Нативні пакети NuGet, орієнтовані на власний, надають файли в папках \build, \content і \tools; \lib у цьому випадку не використовується (NuGet не може безпосередньо додавати посилання на проект C++). Пакет також може включати файли цілей і реквізитів у \build, які NuGet автоматично імпортує в проекти, які споживають пакет. Ці файли мають називатися так само, як ідентифікатор пакета з розширеннями .targets та/або .props. Наприклад, пакет Microsoft.Web.WebView2 містить файл Microsoft.Web.WebView2.targets у своїй папці \build.

```
#
#  CpuFeaturesDotNet Runtime Package
#

set(CPU_FEATURES_DOTNET_RUNTIME_NUGET ${CPU_FEATURES_DOTNET_NUGET_NAME}.runtime.${CPU_FEATURES_DOTNET_RID})
set(CPU_FEATURES_DOTNET_RUNTIME ${CPU_FEATURES_DOTNET_NAME}.runtime.${CPU_FEATURES_DOTNET_RID})
set(CPU_FEATURES_DOTNET_RUNTIME_PROJECT ${CPU_FEATURES_DOTNET_RUNTIME}.csproj)

configure_file(
        ../cmake/templates/${CPU_FEATURES_DOTNET_NAME}.runtime.csproj.in
        ${CMAKE_CURRENT_SOURCE_DIR}/CpuFeaturesDotNet/${CPU_FEATURES_DOTNET_RUNTIME_PROJECT}
        @ONLY)

add_custom_command(
        OUTPUT ${PROJECT_BINARY_DIR}/dotnet/${CPU_FEATURES_DOTNET_RUNTIME}/timestamp
        COMMAND ${DOTNET_CLI} build -c Release ${CPU_FEATURES_DOTNET_RUNTIME_PROJECT}
        COMMAND ${DOTNET_CLI} pack -c Release ${CPU_FEATURES_DOTNET_RUNTIME_PROJECT} -p:PackageId=${CPU_FEATURES_DOTNET_RUNTIME_NUGET}
        COMMAND ${CMAKE_COMMAND} -E make_directory ${PROJECT_BINARY_DIR}/dotnet/${CPU_FEATURES_DOTNET_RUNTIME}
        COMMAND ${CMAKE_COMMAND} -E touch ${PROJECT_BINARY_DIR}/dotnet/${CPU_FEATURES_DOTNET_RUNTIME}/timestamp
        DEPENDS
            ${CMAKE_CURRENT_SOURCE_DIR}/CpuFeaturesDotNet/${CPU_FEATURES_DOTNET_RUNTIME_PROJECT}
            ${CPU_FEATURES_DOTNET}
        BYPRODUCTS
            ${CMAKE_CURRENT_SOURCE_DIR}/CpuFeaturesDotNet/${CPU_FEATURES_DOTNET_RUNTIME_PROJECT}/bin
            ${CMAKE_CURRENT_SOURCE_DIR}/CpuFeaturesDotNet/${CPU_FEATURES_DOTNET_RUNTIME_PROJECT}/obj
        COMMENT "Generate ${CPU_FEATURES_DOTNET_RUNTIME} package (${PROJECT_BINARY_DIR}/dotnet/${CPU_FEATURES_DOTNET_RUNTIME}/timestamp)"
        WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}/CpuFeaturesDotNet
)
```

Рисунок 3.6 - Приклад створення NuGet пакету для бібліотеки

Папку \build можна використовувати для всіх пакетів NuGet, а не тільки для рідних пакетів. Папка \build поважає цільові рамки, як і папки \content, \lib та \tools. Це означає, що ви можете створити папку \build\net40 і папку \build\net45, і NuGet імпортує відповідні файли реквізитів та цільових файлів у проект. Використання сценаріїв PowerShell для імпорту цілей MSBuild не потрібне.

3.4 Каталог .NET RID

RID – скорочення від Runtime Identifier. Значення RID використовуються для визначення цільових платформ, на яких працює програма. Вони використовуються пакетами .NET для представлення ресурсів, що стосуються платформи, у пакетах NuGet. Наступні значення є прикладами RID: linux-x64, ubuntu.14.04-x64, win7-x64 або osx.10.12-x64. Для пакетів із рідними залежностями RID визначає, на яких платформах пакет можна відновити.

Один RID можна встановити в елементі <RuntimeIdentifier> файлу вашого проекту. Кілька RID можна визначити як список, розділений крапкою з комою в елементі <RuntimeIdentifiers> файлу проекту. Вони також використовуються за допомогою параметра --runtime з такими командами .NET CLI:

- dotnet build
- dotnet clean
- dotnet pack
- dotnet publish
- dotnet restore
- dotnet run
- dotnet store

RID, які представляють конкретні операційні системи, зазвичай мають такий шаблон: [os].[version]-[architecture]-[additional qualifiers], де:

- [os] — це прізвисько операційної/платформної системи. Наприклад, ubuntu.
- [version] — це версія операційної системи у вигляді номера версії, розділеного крапкою (.). Наприклад, 15.10. Версія не повинна бути маркетинговими версіями, оскільки вони часто представляють кілька дискретних версій операційної системи з різною площею API платформи.

- [architecture] — це архітектура процесора. Наприклад: x86, x64, arm або arm64.
- [additional qualifiers] додатково диференціюють різні платформи. Наприклад: aot.

3.5 RID граф

Граф RID або резервний графік виконання — це список RID, сумісні один з одним. RID визначені в пакеті Microsoft.NETCore.Platforms. Ви можете побачити список підтримуваних RID і графік RID у файлі runtime.json, який розташований у сховищі dotnet/runtime. У цьому файлі ви можете побачити, що всі RID, крім базового, містять оператор «#import». Ці твердження вказують на сумісні RID.

Коли NuGet відновлює пакети, він намагається знайти точну відповідність для вказаного середовища виконання. Якщо точного збігу не знайдено, NuGet повертається до графіка, доки не знайде найближчу сумісну систему відповідно до графіка RID.

Наступний приклад є фактичним записом для osx.10.12-x64 RID:

```JSON
"osx.10.12-x64": {
    "#import": [ "osx.10.12", "osx.10.11-x64" ]
}
```

Рисунок 3.7 - Приклад RID для osx.10.12-x64

Наведений вище RID визначає, що osx.10.12-x64 імпортує osx.10.11-x64. Отже, коли NuGet відновлює пакети, він намагається знайти точну відповідність для osx.10.12-x64 в пакеті. Якщо NuGet не може знайти певну середовище виконання, він може відновити пакунки, які вказують, наприклад, середовище виконання osx.10.11-x64.

У наступному прикладі показано трохи більший графік RID, також визначений у файлі runtime.json:

```
win7-x64    win7-x86
   |   \   /    |
   |   win7     |
   |     |      |
win-x64  |  win-x86
      \  |  /
        win
         |
        any
```

Рисунок 3.8 - Графік RID для Windows операційної системи

Усі RID в кінцевому підсумку відображаються назад до кореневого будь-якого RID. Є деякі міркування щодо RID, які ви повинні мати на увазі під час роботи з ними:

- Не намагайтеся розібрати RID, щоб отримати складові частини.
- Не створюйте RID програмно.
- Використовуйте RID, які вже визначені для платформи.
- RID мають бути конкретними, тому не припускайте нічого з фактичного значення RID.

3.6 Використання RID

Щоб мати можливість використовувати RID, ви повинні знати, які RID існують. Нові значення регулярно додаються на платформу. Щоб отримати останню повну версію, перегляньте файл runtime.json у сховищі dotnet/runtime.

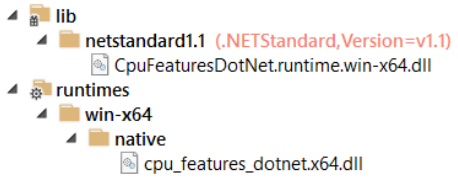


Рисунок 3.9 - Структура NuGet пакету із використанням технології RID для системи Windows-X64

RID, які не прив’язані до певної версії або дистрибутива ОС, є кращим вибором, особливо якщо ви маєте справу з кількома дистрибутивами Linux, оскільки більшість RID дистрибутивів зіставляються з RID, що не стосуються конкретного дистрибутива.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі було здійснено детальний огляд щодо реалізації бібліотеки для ідентифікації процесору. Було підібрано відповідну для бібліотек структуру проекту, з точки зору керування, вона дозволяє здійснювати кращу організацію використання ресурсів задля проекту, меншої ймовірності дублювання діяльності та ресурсів.

Розглянуто детальне застосування маршалінгу у проекті. Маршалінг взаємодії здійснює керування тим, як отримані дані передаються в аргументах методів та повертають задані значення між управляючою і неуправляючою пам'яттю під час операції викликів. Маршалінг взаємодії це така діяльність під час виконання, яку виконує служба маршалінгу загальномовного середовища виконання.

Також описано застосування Runtime Identifier, скорочено RID – потужний інструмент для визначення цільових платформ, на яких працює програма. Він використовуються пакетами платформи .NET задля представлення відповідних ресурсів, які стосуються платформи, у пакетах системи NuGet.

У результаті завдяки застосуванню прогресивних та потужних технологій отримано бібліотеку, що є гнучкою до застосування, а також простою у масштабуванні на ринку. При цьому, отримана система повністю відповідає вимогам, які були описані у першому та другому розділах.

РОЗДІЛ 4
ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНСТРУКЦІЯ ПО РОБОТІ З НИМ

4.1 Перевірка роботи програмного коду на різних архітектурах процесора та операційних системах

Тестування у даній дипломній роботі для різних архітектур процесорів, середовищ виконань та операційних систем використовується у двох етапах. Перший етап тестування це перевірка, що код самої cpu_features бібліотеки ідентифікує інформацію процесора для певної архітектури правильно. Оскільки є безліч різних мікроархітектур і неможливо мати всі процесори різних архітектур та мікроархітектур для тестування, тому було використано сайт де розміщено cpuid дамп майже для всіх мікроархітектур процесорів X86 та інформації процесору від відомих виробників. Дана інформація використовується в програмі модульного тестування як фейкові данні процесору, що порівнюються з очікуваним результатом.

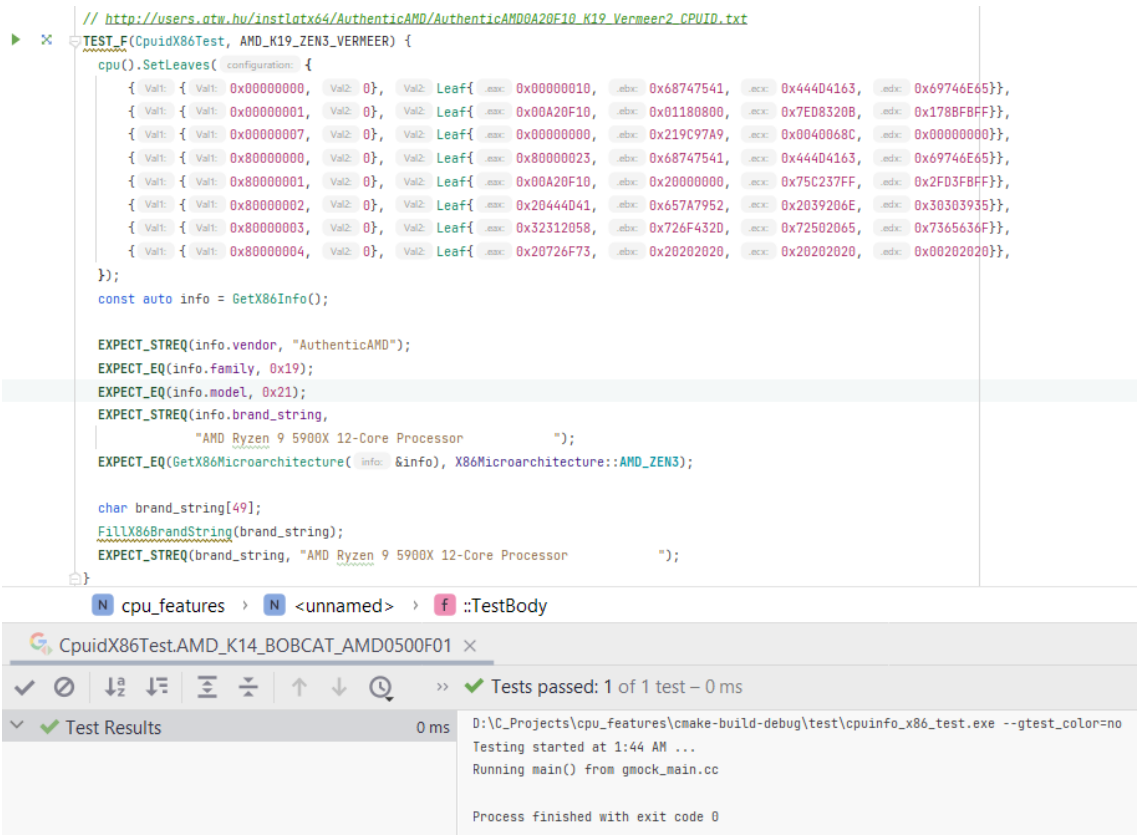


Рисунок 4.1 – Результат успішного тестування архітектури X86

Вище наведено приклад тесту, де зображено, що процесор з архітектурою налаштовано листи для регістрів EAX, EBX, ECX, EDX для архітектури X86 та мікроархітектурою AMD K19 (ZEN3) з кодовим ім’ям VERMEER співпадає з очікуваними результатом, а саме те що вендор процесору AuthenticAMD, сімейство 0x19 та модель 0x21, що відповідає сімейству та кодовому імені у тесті.

Рисунок 4.2 – Інформація процесора AMD K19

Другий етап тестування - це тестування на певній операційній системі з певною архітектурою та середовищем виконання .NET.

```
Mykola Hohsadze +2
public class DllImportX86Tests
{
    [FactX64]
    Nikolay Hohsadze +1
    public void DllImportX86_GetX86Info_Success()
    {
        X86Info.GetX86Info();
    }

    [FactX64]
    Mykola Hohsadze
    public void DllImportX86_GetX86Microarchitecture_Success()
    {
        var x86Info = X86Info.GetX86Info();
        X86Info.GetX86Microarchitecture(x86Info);
    }

    [FactX64]
    Mykola Hohsadze
    public void DllImportX86_GetX86CacheInfo_Success()
    {
        CacheInfo.GetX86CacheInfo();
    }

    [FactX64]
    Mykola Hohsadze
    public void DllImportX86_GetX86BrandString_Success()
    {
        X86Info.GetX86BrandString();
    }

    [FactX64]
    Mykola Hohsadze +2
    public void DllImportX86_GetAarch64Info_Throws()
    {
        Assert.Throws<PlatformNotSupportedException>(testCode: () => Aarch64Info.GetAarch64Info());
    }
}
```

Рисунок 4.3 - Програмний код другого етапу тестування для X86

На рисунку 4.3 наведено тести для архітектури X86, які перевіряють, що код C# правильно викликає нативну бібліотеку cpu_features, та те що при іншій архітектурі буде відпрацьовувати виключення.

На рисунку 4.4 відображено те саме, що й на рисунку 4.3 тільки для архітектури AARCH64.

```
namespace CpuFeaturesDotNet.Testing
{
    Nikolay Hohsadze +1
    public class DllImportAArch64Tests
    {
        [FactAArch64]
        Mykola Hohsadze +1
        public void DllImportAArch64_GetAarch64Info_Success()
        {
            Aarch64Info.GetAarch64Info();
        }

        [FactAArch64]
        Mykola Hohsadze +1
        public void DllImportAArch64_GetX86Info_Throws()
        {
            Assert.Throws<PlatformNotSupportedException>( testCode: () => X86Info.GetX86Info());
        }
    }
}
```

Рисунок 4.4 - Програмний код другого етапу тестування для AARCH64

Для запуску тестів на різних архітектур та операційних системах було використано сервіси GithubActions та Drone CI. Github Actions виключно для операційних систем Windows, macOS та Linux з архітектурою X64, X86 та Drone CI для архітектур ARM, AARCH64.

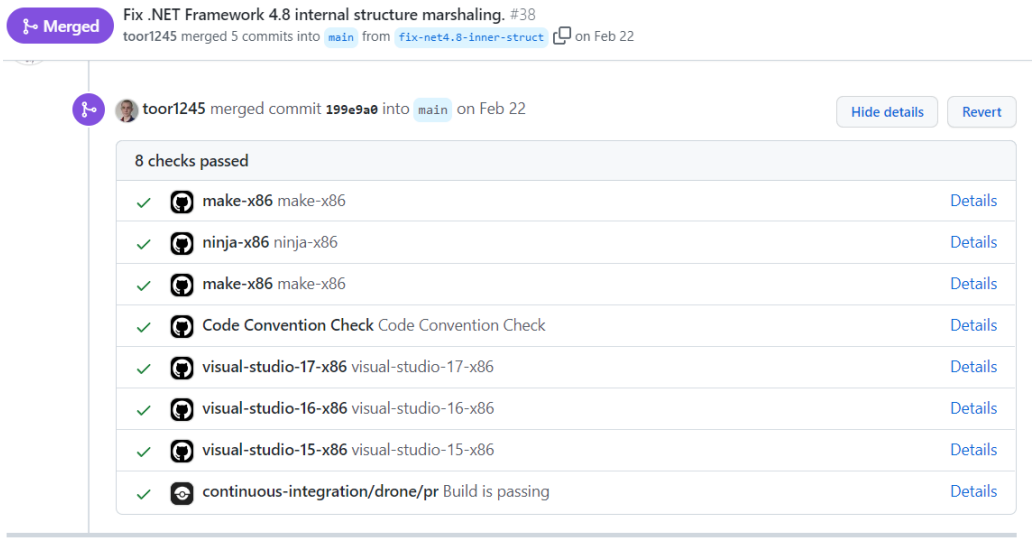


Рисунок 4.5 - Тестування програмного коду за допомогою сервісу Gihub Actions

На рисунку 4.6 наведено тестування та результат вдалого тесту для архітектури AARCH64.

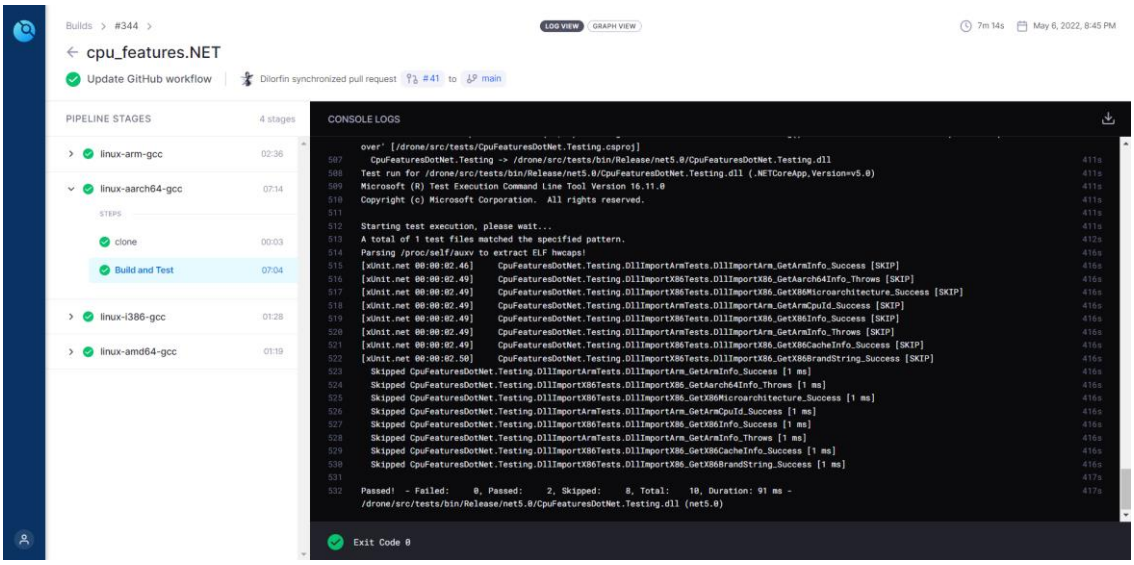


Рисунок 4.6 - Тестування програмного коду за допомогою сервісу Drone CI

Усі тести під інші архітектури програма пропускає і тільки виконує тести під дану архітекутру.

4.2 Інструкція використання бібліотеки

Нижче наведено приклад, використання перевірок осибливостей процесору під час виконання, де програмний код виконує перевірку, чи центральний процесор підтримує набори інструкцій SSE4A

```
namespace CpuFeaturesDotNet.Samples
{
    public class Program
    {
        public static void Main(string[] args)
        {
            X86Info info = X86Info.GetX86Info();
            X86Microarchitecture uarch = X86Info.GetX86Microarchitecture(info);
            bool hasFastAvx = info.Features.IsSupportedAVX && uarch != X86Microarchitecture.INTEL_SANDYBRIDGE;
        }
    }
}
```

Рисунок 4.7 - Приклад використання коду для перевірки наявності SSE4A

Розробник може прочитати всі функції відразу в глобальну змінну, а потім використовувати конкретні функції з цієї глобальної змінної.

```
using System;
using CpuFeaturesDotNet.X86;

namespace CpuFeaturesDotNet.Samples
{
    public class Program
    {
        public static X86Info X86Info = X86Info.GetX86Info();

        public static void Main(string[] args)
        {
            X86Microarchitecture uarch = X86Info.GetX86Microarchitecture(info);
            bool hasFastAvx = info.Features.IsSupportedAVX && uarch != X86Microarchitecture.INTEL_SANDYBRIDGE
        }
    }
}
```

Рисунок 4.8 – Приклад зчитування інформації та її кешування

```
    "IsSupportedCX16": true,
    "IsSupportedSHA": false,
    "IsSupportedDPOPCNT": true,
    "IsSupportedDMOVBE": true,
    "IsSupportedDRDRND": true,
    "IsSupportedDCA": false,
    "IsSupportedSS": true,
    "IsSupportedADX": true
  },
  "Family": 6,
  "Model": 142,
  "Stepping": 10,
  "Vendor": "GenuineIntel",
  "BrandString": "Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz"
}
{
  "Size": 4,
  "Levels": [
    {
      "Level": 1,
      "CacheType": "CPU_FEATURE_CACHE_DATA",
      "CacheSize": 32768,
      "Ways": 8,
```


Рисунок 4.9 – Результат ідентифікації процесору

У наведеному вище рисунку зображено результат виконання ідентифікації процесору Intel(R) Core(TM) i5-8250U

ВИСНОВОК ДО РОЗДІЛУ 4

У розглянутому вище розділі було розібрано основні можливості реалізованої бібліотеки, перевірена працездатність системи, сумісність із різними версіями постачальників. Також було продемонстровано особливості застосування бібліотеки до процесорів під час виконання, де програмний код виконує перевірку, чи центральний процесор підтримує задані набори інструкцій.

Здійснено декілька етапів тестування, перший етап - це перевірка, що програмний код самої cpu_features бібліотеки належним чином ідентифікує інформацію процесора для певної архітектури. Другий етап - це тестування на певній операційній системі з чітко визначеною архітектурою та середовищем виконання .NET. Вважаю, що моя робота у повній мірі відповідає заявленим цілям. Безперечно, є широкий спектр можливостей для подальшого поліпшення та розвитку системи.

На мою думку, поставлені у ході проектування вимоги до бакалаврської роботи були успішно реалізовані.

ВИСНОВКИ

Основна мета даної роботи – реалізація бібліотеки для ідентифікації процесора для платформи .NET, яка б дозволила іншим розробникам отримувати детальну інформацію про тип процесора та його сумісність з іншими системами.

У першому розділі дипломної роботи було проаналізовано та розглянуто питання аукціонів у нашому житті, їхня класифікація, особливості та приклади програмного забезпечення, що використовується на сучасному ринку та призначення яких стосується даного дипломного проекту.

У другому розділі було проаналізовано та розглянуто ряд технологій для безпосередньої реалізації проекту мовою C. Мову програмування C та, вчасності бібліотеку cpu_features було обрано з декількох причин, однією з яких була умова запуску вбудованих пристроїв з мінімальним обсягом використованої оперативної пам’яті при виконанні.

У третьому розділі було здійснено детальний огляд щодо реалізації бібліотеки для ідентифікації процесору. Було підібрано відповідну для бібліотек структуру проекту, з точки зору керування, вона дозволяє здійснювати кращу організацію використання ресурсів задля проекту, меншої ймовірності дублювання діяльності та ресурсів.

У розглянутому вище розділі було розібрано основні можливості реалізованої бібліотеки, перевірена працездатність системи, сумісність із різними версіями постачальників. Також було продемонстровано особливості застосування бібліотеки до процесорів під час виконання, де програмний код виконує перевірку, чи центральний процесор підтримує задані набори інструкцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Is Global Optimization? [Електронний ресурс] // MathWorks – Режим доступу до ресурсу: https://www.mathworks.com/help/gads/what-is-global-optimization.html.
2. Christodoulos A. F. State of the Art in Global Optimization / A. F. Christodoulos, M. P. Panos // Computational Methods and Applications / A. F. Christodoulos, M. P. Panos. – Dordrecht, Netherlands: Springer US, 1996. – С. 139–180.
3. Little J.D.C., Murty K.G., Sweeney D.W., and Karel C. / An algorithm for the traveling salesman problem. Operations Research, 1963. С. 972-989.
4. Yang Xin-She. Firefly Algorithm, Stochastic Test Functions and Design optimization // International Journal of Bio-Inspired Computation. 2010. V. 2. N 2. P. 78—84.
5. Mehrabiana A.R., Lucase C. A novel numerical optimization algorithm inspired from weed colonization // Ecological informatics. 2006. No 1. P. 355—366.
6. Yang X.-S., Deb S. Cuckoo search via Levy flights // Proc. of the world Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, USA, pp. 210—214.
7. Tuba M., Subotic M., Stanarevic N. Modified cuckoo search algorithm for unconstrained optimization problems // Proc. of the 5th European Computing Conference (ECC'I1), Paris, France, April 28—30, 2011. pp. 263—268.
8. Mucherino A., Seref O. Monkey Search: A Novel Meta-Heuristic Search for Global Optimization // Data Mining, System Analysis and Optimization in Biomedicine, AIP Conference Proceedings. 2007. P. 162—173.
9. A tour of the C# language [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

10. PYPL PopularitY of Programming Language [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://pypl.github.io/PYPL.html.
11. TIOBE Index for April 2021 [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.
12. MATT WATSON. What is C# used for? [Електронний ресурс] / MATT WATSON // STACKIFY PRODUCT & COMPANY UPDATES. – 2020. – Режим доступу до ресурсу: https://stackify.com/what-is-c-used-for/.
13. Top 10 Most Important Features Of C# [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/top-10-most-important-features-of-C-Sharp-programming/.
14. Advantages of C# language. [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://blogs.brainsmiths.com/post/2019/12/10/advantages-of-c-language.aspx.
15. C Sharp – Features, Advantages and Disadvantages [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.urbannaturale.com/c-sharp-features-advantages-and-disadvantages/.
16. Java vs C# - 10 Key Differences between Java and C# [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.guru99.com/java-vs-c-sharp-key-difference.html.
17. What is .NET Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework.
18. What is .NET Framework? Explain Architecture & Components [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/net-framework.html.
19. Desktop Guide (Windows Forms .NET) [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0.

20. Getting started with .NET Charts [Електронний ресурс] – Режим доступу до ресурсу: https://www.i-programmer.info/programming/uiux/2756-getting-started-with-net-charts.html.
21. A flexible charting library for .NET [Електронний ресурс]. – 2007. – Режим доступу до ресурсу: https://www.codeproject.com/Articles/5431/A-flexible-charting-library-for-NET.
22. NET regular expressions [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions.
23. Adam Hughes. Microsoft SQL Server [Електронний ресурс] / Adam Hughes, Craig Stedman. – 2019. – Режим доступу до ресурсу: https://searchdatamanagement.techtarget.com/definition/SQL-Server.
24. Use the MSTest framework in unit tests [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/test/using-microsoft-visualstudio-testtools-unittesting-members-in-unit-tests?view=vs-2019.

ДОДАТОК 1

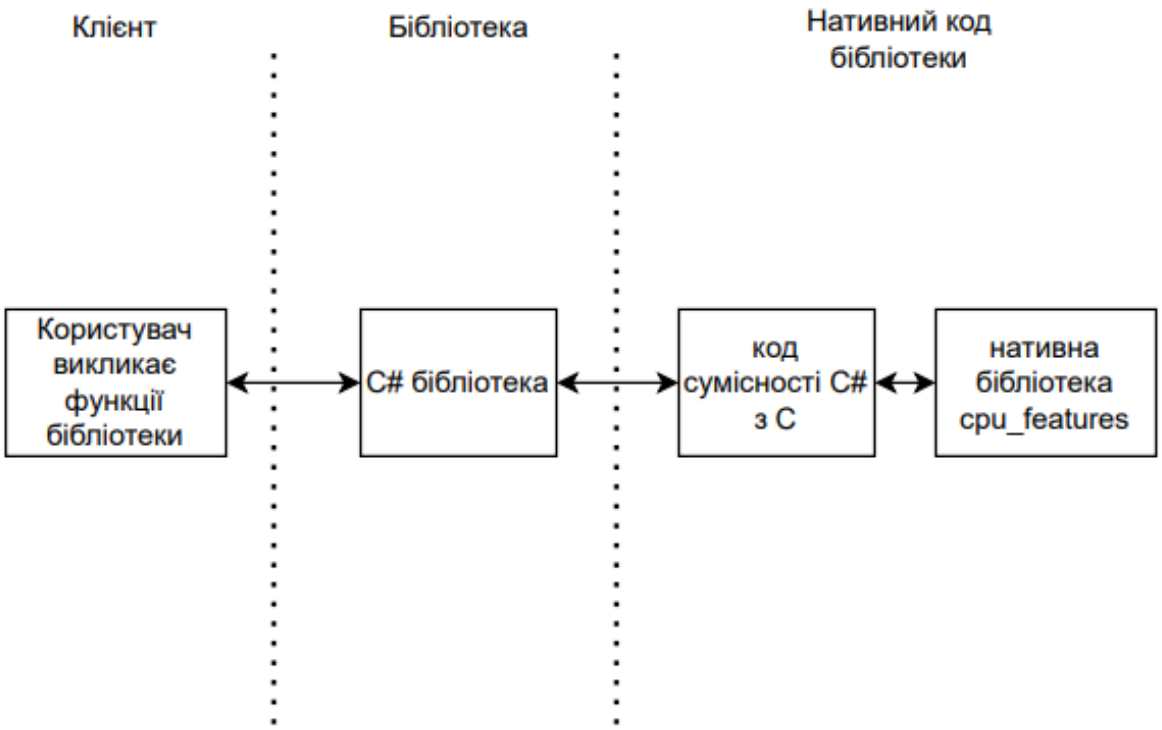
Бібліотека ідентифікації процесора для платформи .NET

Структурна схема системи

ІАЛЦ.467400.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467400.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Гогсадзе М. Г.			**Бібліотека ідентифікації процесора для платформи .NET**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Каплунов А. В.					
					Структурна схема системи	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-83	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 2

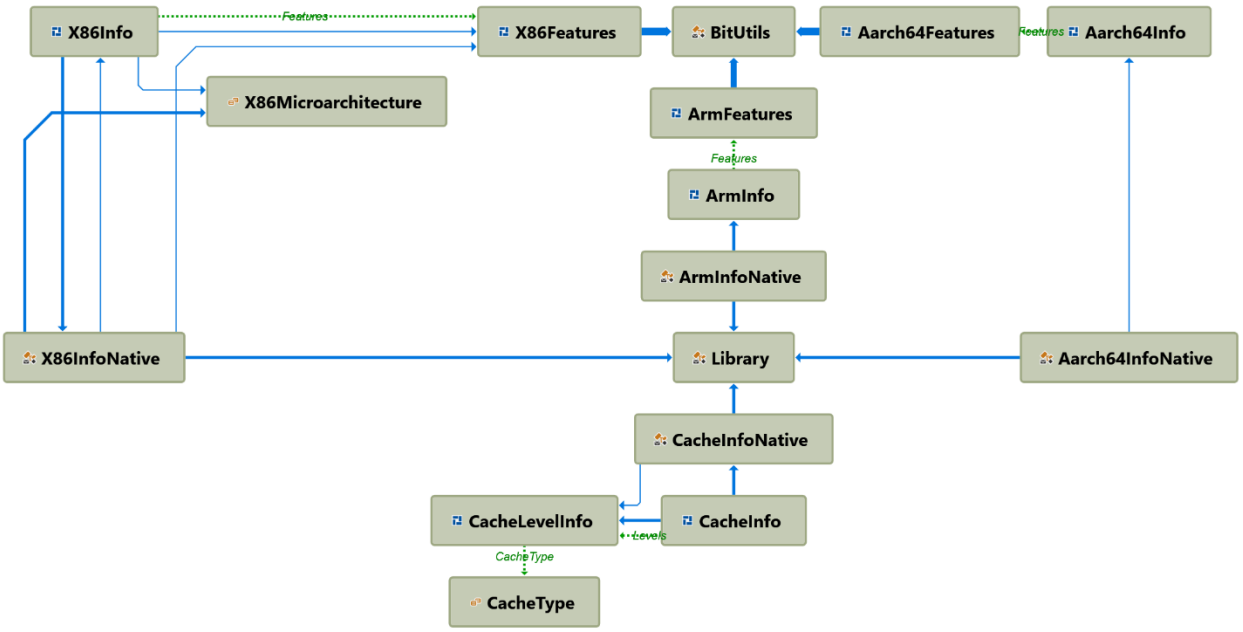
Бібліотека ідентифікації процесора для платформи .NET

Функціональна схема (діаграма класів)

ІАЛЦ.467400.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467400.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Гогсадзе М. Г.			**Бібліотека ідентифікації процесора для платформи .NET**	Літ. / Аркуш / Аркушів: 1 / 1
Перевірив		Каплунов А.В.				
Н. Контр.		Сімоненко В. П.			Функціональна схема (діаграма класів)	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-83
Затвердив						

ДОДАТОК 3

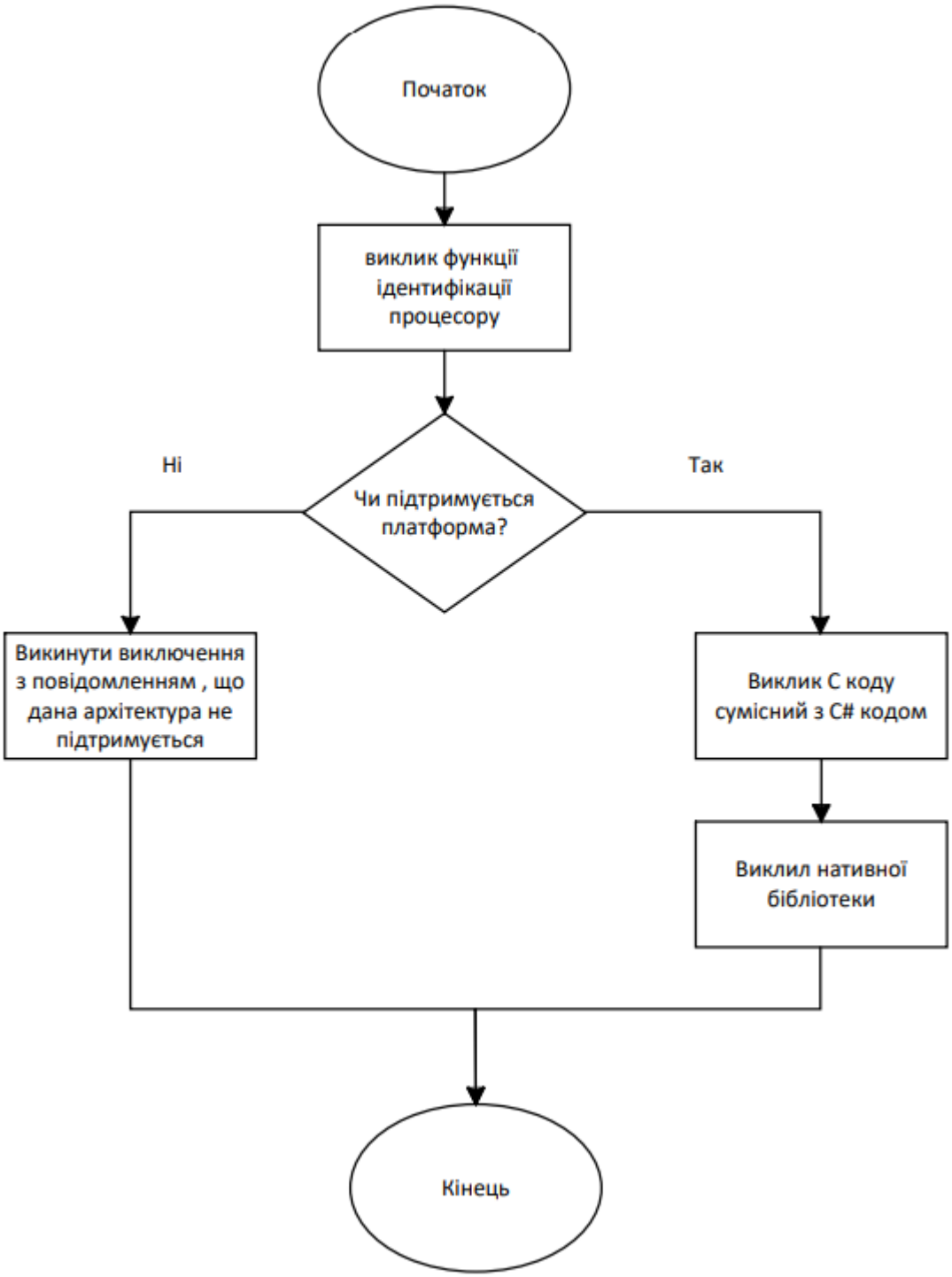
Бібліотека ідентифікації процесора для платформи .NET

Алгоритм дій програмного забезпечення

ІАЛЦ.467400.006 Д3

Аркушів 1

Київ 2022 р



					ІАЛЦ.467400.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Гогсадзе М. Г.			**Бібліотека ідентифікації процесора для платформи .NET** Алгоритм дій програмного забезпечення	Літ. / Аркуш / Аркушів: 1 / 1
Перевірив		Каплунов А. В.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-83
Затвердив						

ДОДАТОК 4

Бібліотека ідентифікації процесора для платформи .NET

Текст програмного коду

ІАЛЦ.467400.007 Д4

Аркушів 16

Київ 2022 р

```
using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.AArch32
{
    [StructLayout(LayoutKind.Sequential)]
    public readonly struct ArmFeatures
    {
        private readonly int featuresRaw1;

        public bool IsSupportedSWP => BitUtils.IsBitSet((ulong)featuresRaw1, 0);
        public bool IsSupportedHALF => BitUtils.IsBitSet((ulong)featuresRaw1, 1);
        public bool IsSupportedTHUMB => BitUtils.IsBitSet((ulong)featuresRaw1, 2);
        public bool IsSupported_26BIT => BitUtils.IsBitSet((ulong)featuresRaw1, 3);

        public bool IsSupportedFASTMULT => BitUtils.IsBitSet((ulong)featuresRaw1, 4);
        public bool IsSupportedFPA => BitUtils.IsBitSet((ulong)featuresRaw1, 5);
        public bool IsSupportedVFP => BitUtils.IsBitSet((ulong)featuresRaw1, 6);
        public bool IsSupportedEDSP => BitUtils.IsBitSet((ulong)featuresRaw1, 7);

        public bool IsSupportedJAVA => BitUtils.IsBitSet((ulong)featuresRaw1, 8);
        public bool IsSupportedIWMMXT => BitUtils.IsBitSet((ulong)featuresRaw1, 9);
        public bool IsSupportedCRUNCH => BitUtils.IsBitSet((ulong)featuresRaw1, 10);
        public bool IsSupportedTHUMBEE => BitUtils.IsBitSet((ulong)featuresRaw1, 11);
        public bool IsSupportedNEON => BitUtils.IsBitSet((ulong)featuresRaw1, 12);
        public bool IsSupportedVFPV3 => BitUtils.IsBitSet((ulong)featuresRaw1, 13);
        public bool IsSupportedVFPV3D16 => BitUtils.IsBitSet((ulong)featuresRaw1, 14);
        public bool IsSupportedTLS => BitUtils.IsBitSet((ulong)featuresRaw1, 15);
        public bool IsSupportedVFPV4 => BitUtils.IsBitSet((ulong)featuresRaw1, 16);
        public bool IsSupportedIDIVA => BitUtils.IsBitSet((ulong)featuresRaw1, 17);
        public bool IsSupportedIDIVT => BitUtils.IsBitSet((ulong)featuresRaw1, 18);
        public bool IsSupportedVFPD32 => BitUtils.IsBitSet((ulong)featuresRaw1, 19);
        public bool IsSupportedLPAE => BitUtils.IsBitSet((ulong)featuresRaw1, 20);

        public bool IsSupportedEVTSTRM => BitUtils.IsBitSet((ulong)featuresRaw1, 21);
        public bool IsSupportedAES => BitUtils.IsBitSet((ulong)featuresRaw1, 22);
        public bool IsSupportedPMULL => BitUtils.IsBitSet((ulong)featuresRaw1, 23);
        public bool IsSupportedSHA1 => BitUtils.IsBitSet((ulong)featuresRaw1, 24);
        public bool IsSupportedSHA2 => BitUtils.IsBitSet((ulong)featuresRaw1, 25);
        public bool IsSupportedCRC32 => BitUtils.IsBitSet((ulong)featuresRaw1, 26);

        internal ArmFeatures(int featuresRaw1)
        {
            this.featuresRaw1 = featuresRaw1;
```

```
        }
    }
}

using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.AArch32
{
    [StructLayout(LayoutKind.Sequential)]
    public readonly struct ArmInfo
    {
        public readonly ArmFeatures Features;
        public readonly int Implementer;
        public readonly int Architecture;
        public readonly int Variant;
        public readonly int Part;
        public readonly int Revision;

#if (ARM32 && OS_LINUX)
        /// <summary>
        /// Calls cpuid and returns an initialized Arm info.
        /// </summary>
        public static ArmInfo GetArmInfo() => ArmInfoNative._GetArmInfo();

        /// <summary>
        /// Compute CpuId from ArmInfo.
        /// </summary>
        public static uint GetArmCpuId(ArmInfo info) => ArmInfoNative._GetArmCpuId(in info);
#else
        /// <summary>
        /// Calls cpuid and returns an initialized ArmInfo.
        /// </summary>
        public       static       ArmInfo       GetArmInfo()       =>       throw       new
System.PlatformNotSupportedException();

        /// <summary>
        /// Compute CpuId from ArmInfo.
        /// </summary>
        public     static     uint     GetArmCpuId(ArmInfo     info)     =>     throw     new
System.PlatformNotSupportedException();
#endif
    }
}
```

```
using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.AArch32
{
    internal static class ArmInfoNative
    {
        [DllImport(Library.NATIVE_LIBRARY, EntryPoint = "GetArmInfoPort")]
        public static extern ArmInfo _GetArmInfo();

        [DllImport(Library.NATIVE_LIBRARY, EntryPoint = "GetArmCpuIdPort")]
        public static extern uint _GetArmCpuId(in ArmInfo info);
    }
}

using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.AArch64
{
    [StructLayout(LayoutKind.Sequential)]
    public readonly struct Aarch64Features
    {
        private readonly int featuresRaw1;
        private readonly int featuresRaw2;

        public bool IsSupportedFP => BitUtils.IsBitSet((ulong)featuresRaw1, 0);
        public bool IsSupportedASIMD => BitUtils.IsBitSet((ulong)featuresRaw1, 1);
        public bool IsSupportedEVTSTRM => BitUtils.IsBitSet((ulong)featuresRaw1, 2);
        public bool IsSupportedAES => BitUtils.IsBitSet((ulong)featuresRaw1, 3);
        public bool IsSupportedPMULL => BitUtils.IsBitSet((ulong)featuresRaw1, 4);
        public bool IsSupportedSHA1 => BitUtils.IsBitSet((ulong)featuresRaw1, 5);
        public bool IsSupportedSHA2 => BitUtils.IsBitSet((ulong)featuresRaw1, 6);
        public bool IsSupportedCRC32 => BitUtils.IsBitSet((ulong)featuresRaw1, 7);
        public bool IsSupportedATOMICS => BitUtils.IsBitSet((ulong)featuresRaw1, 8);
        public bool IsSupportedFPHP => BitUtils.IsBitSet((ulong)featuresRaw1, 9);
        public bool IsSupportedASIMDHP => BitUtils.IsBitSet((ulong)featuresRaw1, 10);
        public bool IsSupportedCPUID => BitUtils.IsBitSet((ulong)featuresRaw1, 11);
        public bool IsSupportedASIMDRDM => BitUtils.IsBitSet((ulong)featuresRaw1, 12);
        public bool IsSupportedJSCVT => BitUtils.IsBitSet((ulong)featuresRaw1, 13);
        public bool IsSupportedFCMA => BitUtils.IsBitSet((ulong)featuresRaw1, 14);
        public bool IsSupportedLRCPC => BitUtils.IsBitSet((ulong)featuresRaw1, 15);
        public bool IsSupportedDCPOP => BitUtils.IsBitSet((ulong)featuresRaw1, 16);
        public bool IsSupportedSHA3 => BitUtils.IsBitSet((ulong)featuresRaw1, 17);
        public bool IsSupportedSM3 => BitUtils.IsBitSet((ulong)featuresRaw1, 18);
```

```
        public bool IsSupportedSM4 => BitUtils.IsBitSet((ulong)featuresRaw1, 19);
        public bool IsSupportedASIMDDP => BitUtils.IsBitSet((ulong)featuresRaw1, 20);
        public bool IsSupportedSHA512 => BitUtils.IsBitSet((ulong)featuresRaw1, 21);
        public bool IsSupportedSVE => BitUtils.IsBitSet((ulong)featuresRaw1, 22);
        public bool IsSupportedASIMDFHM => BitUtils.IsBitSet((ulong)featuresRaw1, 23);
        public bool IsSupportedDIT => BitUtils.IsBitSet((ulong)featuresRaw1, 24);
        public bool IsSupportedUSCAT => BitUtils.IsBitSet((ulong)featuresRaw1, 25);
        public bool IsSupportedILRCPC => BitUtils.IsBitSet((ulong)featuresRaw1, 26);
        public bool IsSupportedFLAGM => BitUtils.IsBitSet((ulong)featuresRaw1, 27);
        public bool IsSupportedSSBS => BitUtils.IsBitSet((ulong)featuresRaw1, 28);
        public bool IsSupportedSB => BitUtils.IsBitSet((ulong)featuresRaw1, 29);
        public bool IsSupportedPACA => BitUtils.IsBitSet((ulong)featuresRaw1, 30);
        public bool IsSupportedPACG => BitUtils.IsBitSet((ulong)featuresRaw1, 31);
        public bool IsSupportedDCPODP => BitUtils.IsBitSet((ulong)featuresRaw2, 0);
        public bool IsSupportedSVE2 => BitUtils.IsBitSet((ulong)featuresRaw2, 1);
        public bool IsSupportedSVEAES => BitUtils.IsBitSet((ulong)featuresRaw2, 2);
        public bool IsSupportedSVEPMULL => BitUtils.IsBitSet((ulong)featuresRaw2, 3);
        public bool IsSupportedSVEBITPERM => BitUtils.IsBitSet((ulong)featuresRaw2, 4);
        public bool IsSupportedSVESHA3 => BitUtils.IsBitSet((ulong)featuresRaw2, 5);
        public bool IsSupportedSVESM4 => BitUtils.IsBitSet((ulong)featuresRaw2, 6);
        public bool IsSupportedFLAGM2 => BitUtils.IsBitSet((ulong)featuresRaw2, 7);
        public bool IsSupportedFRINT => BitUtils.IsBitSet((ulong)featuresRaw2, 8);
        public bool IsSupportedSVEI8MM => BitUtils.IsBitSet((ulong)featuresRaw2, 9);
        public bool IsSupportedSVEF32MM => BitUtils.IsBitSet((ulong)featuresRaw2, 10);
        public bool IsSupportedSVEF64MM => BitUtils.IsBitSet((ulong)featuresRaw2, 11);
        public bool IsSupportedSVEBF16 => BitUtils.IsBitSet((ulong)featuresRaw2, 12);
        public bool IsSupportedI8MM => BitUtils.IsBitSet((ulong)featuresRaw2, 13);
        public bool IsSupportedBF16 => BitUtils.IsBitSet((ulong)featuresRaw2, 14);
        public bool IsSupportedDGH => BitUtils.IsBitSet((ulong)featuresRaw2, 15);
        public bool IsSupportedRNG => BitUtils.IsBitSet((ulong)featuresRaw2, 16);
        public bool IsSupportedBTI => BitUtils.IsBitSet((ulong)featuresRaw2, 17);
        public bool IsSupportedMTE => BitUtils.IsBitSet((ulong)featuresRaw2, 18);

        internal Aarch64Features(int featuresRaw1, int featuresRaw2)
        {
            this.featuresRaw1 = featuresRaw1;
            this.featuresRaw2 = featuresRaw2;
        }
    }
}
```

```
namespace CpuFeaturesDotNet.AArch64
{
    [StructLayout(LayoutKind.Sequential)]
    public readonly struct Aarch64Info
    {
        public readonly Aarch64Features Features;
        public readonly int Implementer;
        public readonly int Variant;
        public readonly int Part;
        public readonly int Revision;

#if (ARM64 && OS_LINUX)
        /// <summary>
        /// Calls cpuid and returns an initialized AArch64 info.
        /// </summary>
        public static Aarch64Info GetAarch64Info() => Aarch64InfoNative._GetAarch64Info();
#else
        /// <summary>
        /// Calls cpuid and returns an initialized AArch64 info.
        /// </summary>
        public      static      Aarch64Info      GetAarch64Info()      =>      throw      new
System.PlatformNotSupportedException();
#endif
    }
}

using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.AArch64
{
    internal static class Aarch64InfoNative
    {
        [DllImport(Library.NATIVE_LIBRARY, EntryPoint = "GetAarch64InfoPort")]
        public static extern Aarch64Info _GetAarch64Info();
    }
}

using System.Runtime.InteropServices;

namespace CpuFeaturesDotNet.X86
{
    [StructLayout(LayoutKind.Sequential)]
    public readonly struct X86Features
```

```
{
    private readonly int featuresRaw1;
    private readonly int featuresRaw2;

    public bool IsSupportedFPU => BitUtils.IsBitSet((ulong)featuresRaw1, 0);
    public bool IsSupportedTSC => BitUtils.IsBitSet((ulong)featuresRaw1, 1);
    public bool IsSupportedCX8 => BitUtils.IsBitSet((ulong)featuresRaw1, 2);
    public bool IsSupportedCLFSH => BitUtils.IsBitSet((ulong)featuresRaw1, 3);
    public bool IsSupportedMMX => BitUtils.IsBitSet((ulong)featuresRaw1, 4);
    public bool IsSupportedAES => BitUtils.IsBitSet((ulong)featuresRaw1, 5);
    public bool IsSupportedERMS => BitUtils.IsBitSet((ulong)featuresRaw1, 6);
    public bool IsSupportedF16C => BitUtils.IsBitSet((ulong)featuresRaw1, 7);
    public bool IsSupportedFMA4 => BitUtils.IsBitSet((ulong)featuresRaw1, 8);
    public bool IsSupportedFMA3 => BitUtils.IsBitSet((ulong)featuresRaw1, 9);
    public bool IsSupportedVAES => BitUtils.IsBitSet((ulong)featuresRaw1, 10);
    public bool IsSupportedVPCLMULQDQ => BitUtils.IsBitSet((ulong)featuresRaw1, 11);
    public bool IsSupportedBMI1 => BitUtils.IsBitSet((ulong)featuresRaw1, 12);
    public bool IsSupportedHLE => BitUtils.IsBitSet((ulong)featuresRaw1, 13);
    public bool IsSupportedBMI2 => BitUtils.IsBitSet((ulong)featuresRaw1, 14);
    public bool IsSupportedRTM => BitUtils.IsBitSet((ulong)featuresRaw1, 15);
    public bool IsSupportedRDSEED => BitUtils.IsBitSet((ulong)featuresRaw1, 16);
    public bool IsSupportedCLFLUSHOPT => BitUtils.IsBitSet((ulong)featuresRaw1, 17);
    public bool IsSupportedCLWB => BitUtils.IsBitSet((ulong)featuresRaw1, 18);

    public bool IsSupportedSSE => BitUtils.IsBitSet((ulong)featuresRaw1, 19);
    public bool IsSupportedSSE2 => BitUtils.IsBitSet((ulong)featuresRaw1, 20);
    public bool IsSupportedSSE3 => BitUtils.IsBitSet((ulong)featuresRaw1, 21);
    public bool IsSupportedSSSE3 => BitUtils.IsBitSet((ulong)featuresRaw1, 22);
    public bool IsSupportedSSE4_1 => BitUtils.IsBitSet((ulong)featuresRaw1, 23);
    public bool IsSupportedSSE4_2 => BitUtils.IsBitSet((ulong)featuresRaw1, 24);
    public bool IsSupportedSSE4A => BitUtils.IsBitSet((ulong)featuresRaw1, 25);

    public bool IsSupportedAVX => BitUtils.IsBitSet((ulong)featuresRaw1, 26);
    public bool IsSupportedAVX2 => BitUtils.IsBitSet((ulong)featuresRaw1, 27);

    public bool IsSupportedAVX512F => BitUtils.IsBitSet((ulong)featuresRaw1, 28);
    public bool IsSupportedAVX512CD => BitUtils.IsBitSet((ulong)featuresRaw1, 29);
    public bool IsSupportedAVX512ER => BitUtils.IsBitSet((ulong)featuresRaw1, 30);
    public bool IsSupportedAVX512PF => BitUtils.IsBitSet((ulong)featuresRaw1, 31);
    public bool IsSupportedAVX512BW => BitUtils.IsBitSet((ulong)featuresRaw2, 0);
    public bool IsSupportedAVX512DQ => BitUtils.IsBitSet((ulong)featuresRaw2, 1);
    public bool IsSupportedAVX512VL => BitUtils.IsBitSet((ulong)featuresRaw2, 2);
    public bool IsSupportedAVX512IFMA => BitUtils.IsBitSet((ulong)featuresRaw2, 3);
```

```
        public bool IsSupportedAVX512VBMI => BitUtils.IsBitSet((ulong)featuresRaw2, 4);
        public bool IsSupportedAVX512VBMI2 => BitUtils.IsBitSet((ulong)featuresRaw2, 5);
        public bool IsSupportedAVX512VNNI => BitUtils.IsBitSet((ulong)featuresRaw2, 6);
        public bool IsSupportedAVX512BITALG => BitUtils.IsBitSet((ulong)featuresRaw2, 7);
        public bool IsSupportedAVX512VPOPCNTDQ => BitUtils.IsBitSet((ulong)featuresRaw2, 8);
        public bool IsSupportedAVX512_4VNNIW => BitUtils.IsBitSet((ulong)featuresRaw2, 9);
        public bool IsSupportedAVX512_4VBMI2 => BitUtils.IsBitSet((ulong)featuresRaw2, 10);
        public  bool  IsSupportedAVX512_SECOND_FMA  =>  BitUtils.IsBitSet((ulong)featuresRaw2,
11);
        public bool IsSupportedAVX512_4FMAPS => BitUtils.IsBitSet((ulong)featuresRaw2, 12);
        public bool IsSupportedAVX512_BF16 => BitUtils.IsBitSet((ulong)featuresRaw2, 13);
        public bool IsSupportedAVX512_VP2INTERSECT => BitUtils.IsBitSet((ulong)featuresRaw2,
14);

        public bool IsSupportedAMX_BF16 => BitUtils.IsBitSet((ulong)featuresRaw2, 15);
        public bool IsSupportedAMX_TILE => BitUtils.IsBitSet((ulong)featuresRaw2, 16);
        public bool IsSupportedAMX_INT8 => BitUtils.IsBitSet((ulong)featuresRaw2, 17);

        public bool IsSupportedPCLMULQDQ => BitUtils.IsBitSet((ulong)featuresRaw2, 18);
        public bool IsSupportedSMX => BitUtils.IsBitSet((ulong)featuresRaw2, 19);
        public bool IsSupportedSGX => BitUtils.IsBitSet((ulong)featuresRaw2, 20);
        public bool IsSupportedCX16 => BitUtils.IsBitSet((ulong)featuresRaw2, 21);
        public bool IsSupportedSHA => BitUtils.IsBitSet((ulong)featuresRaw2, 22);
        public bool IsSupportedDPOPCNT => BitUtils.IsBitSet((ulong)featuresRaw2, 23);
        public bool IsSupportedDMOVBE => BitUtils.IsBitSet((ulong)featuresRaw2, 24);
        public bool IsSupportedDRDRND => BitUtils.IsBitSet((ulong)featuresRaw2, 25);

        public bool IsSupportedDCA => BitUtils.IsBitSet((ulong)featuresRaw2, 26);
        public bool IsSupportedSS => BitUtils.IsBitSet((ulong)featuresRaw2, 27);
        public bool IsSupportedADX => BitUtils.IsBitSet((ulong)featuresRaw2, 28);

        internal X86Features(int featuresRaw1, int featuresRaw2)
        {
            this.featuresRaw1 = featuresRaw1;
            this.featuresRaw2 = featuresRaw2;
        }
    }
}
```