Лекція 3ХАРАКТЕРИСТИКА ЯДРА ARM–МІКРОКОНТРОЛЕРІВ

1 Основні положення

Характеристику ядра ARM–мікроконтролерів розглянемо на прикладі сімейства LPC2300, побудованого на основі ЦПП ARM7. Щоб використовувати ці мікроконтролери, вам зовсім не потрібно бути експертом в області програмування процесора ARM7, оскільки турботу про більшість складних моментів бере на себе компілятор мови С. Тим не менш, щоб розробити надійний пристрій, ви повинні мати хоча б загальне уявлення про те, як працює ЦПП і які у нього є особливості.

У цьому розділі ми розглянемо основні характеристики ядра ARM7 разом з його моделлю програмування, а також коротко обговоримо набір команд, який використовується даним процесором. В результаті ви отримаєте всю необхідну інформацію про процесор, що є «серцем» сімейства LPC2300. Для більш поглибленого вивчення процесорів ARM рекомендуємо звернутися до книг, зазначених в списках літератури.

Головний принцип, що лежить в основі процесора ARM, це простота. Ядро ARM7 є RISC–машиною, яка передбачає використання невеликої кількості команд і відповідно складається з відносно невеликої кількості логічних елементів. Завдяки цьому процесор ARM7 ідеально підходить для використання у вбудованих системах. Він має високу продуктивність, низьке енергоспоживання і займає невелику частину загальної площі кристала.

3.2 Конвейер команд

Основний елемент ЦПП ARM7 це конвеєр команд, який використовується для обробки команд, які зчитуються з пам'яті програм. Конкретно, в ядрі ARM7 реалізовано триступеневий конвеєр (рисунок .1).

`

Рисунок 1 – Робота триступеневого конвеєра

Триступеневий конвеєр є найпростішою різновидністю конвеєрів і не схильний до виникнення різних небезпечних ситуацій, таких як «читання раніше запису», які зустрічаються в конвеєрах з великим числом ступенів.

Конвеєр має три апаратно–незалежні ступені, завдяки яким одночасно з виконанням однієї команди здійснюється декодування другої і вибірка третьої.

Він настільки ефективно прискорює проходження команд через ЦПП, що більшість команд ARM можуть виконуватися за один такт. Конвеєр найбільш ефективний при виконанні лінійного коду. При виявленні переходу конвеєр скидається, і для відновлення виконання програми з максимальною швидкістю він повинен спочатку заповнитися. Пізніше ми з вами побачимо, що набір команд процесора ARM має кілька цікавих особливостей, що дозволяють виключити з коду короткі переходи для поліпшення проходження коду по конвеєру. Оскільки конвеєр є складовою частиною ЦПП, він повністю прихований від програміста. Однак, важливо пам'ятати, що значення лічильника команд (Program Counter – PC) на 8 байт перевищує значення адреси поточної виконуваної команди. У зв'язку з цим необхідно обережно підходити до обчислення зсувів у разі відносної адресації з використанням лічильника команд.

`

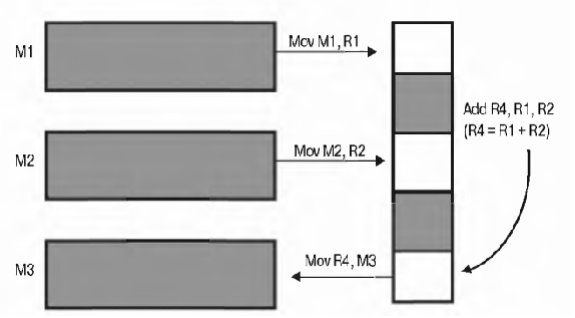
Наприклад, команда:

```
0x4000          LDR          PC, [PC, # 4]
```

завантажить в лічильник команд PC вміст, що знаходиться за адресою PC + 4. Оскільки PC випереджає поточну команду на 8 байт, в нього буде додано вміст за адресою 0x400C, а не 0x4004.

3 Регістри регістрового файлу

Процесор ARM7 має архітектуру «load–and–store» (завантаження – збереження), тому для виконання будь–якої команди обробки даних необхідно спочатку перенести ці дані з пам'яті в певні регістри, виконати команду обробки даних і потім записати отримані значення назад в пам’ять (рисунок 2).



ЦПП ARM7 має архітектуру «Load-and-store». Усі команди обробки даних можуть працювати тільки з основним регістровим файлом.

Рисунок 2 – Обробка даних

Основний регістровий файл складається з 16 призначених регістрів: R0 ... R15 (рисунок 3).

Кожен з цих регістрів є 32–бітовим. У вітчизняній літературі прийнято користуватися поняттями «розряд», «розрядний». Двійковий розряд – це біт. У даних лекціях ми будемо дотримуватися зарубіжної термінології («біт», «бітний»), що більш відповідає сучасній тенденції в цифровій техніці.

`

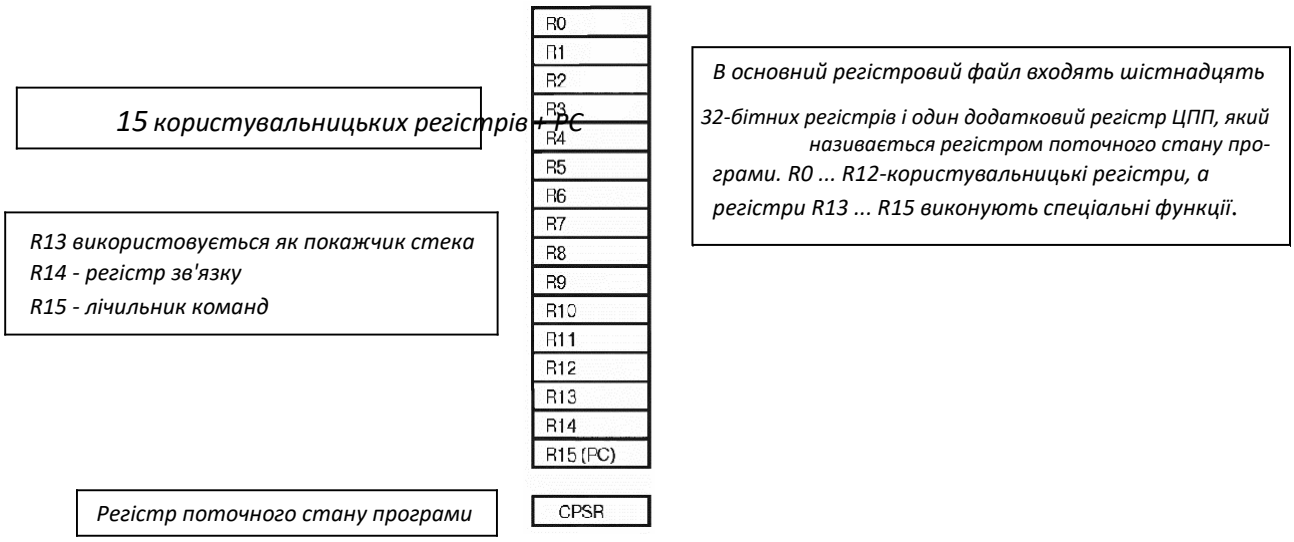


Рисунок 3 – Структура основного регістрового файлу

Регістри R0 ... R12 призначено виключно для потреб користувача і не виконують ніяких інших функцій, в той час як регістри R13 ... R15 мають додаткові функції. Регістр R13 використовується як покажчик стека (Stack Pointer – SP). Регістр R14 називається регістром зв'язку (Link Register – LR). При виклику підпрограми адреса повернення автоматично запам'ятовується в регістрі зв'язку, звідки потім зчитується при поверненні. Таке рішення дозволяє швидко переходити до «кінцевих» функцій (функції, які не викликають інших функцій) і повертатися з них. Якщо функція входить до складу «гілки», тобто викликає інші функції, вміст регістра зв'язку необхідно зберігати в стеку (R13). Нарешті, регістр R15 виконує функції лічильника команд (PC). Що цікаво, багато команд можуть працювати з регістрами R13 ... R15, як із звичайними користувацькими регістрами.

4. Регістр поточного стану програми

Поряд з банком регістрів у ЦПП є додатковий 32–бітний регістр, який називається регістром поточного стану програми (Current Program Status Register – CPSR). Регістр CPSR містить набір прапорців, які керують функціонуванням ЦПП ARM7 і відображають його стан (рисунок 4).

`

Рисунок 4 – Регістр поточного стану програми

У старших чотирьох бітах регістра CPSR зберігаються прапорці умов, значення яких визначаються процесором. Ці прапорці відображають результат виконання чергової команди обробки даних. Завдяки їм ви можете дізнатися, чи не було отримано в результаті виконання команди від'ємне або нульове значення, а також чи не відбулося перенесення або переповнення. Молодші вісім бітів регістра CPSR містять прапорці, значення яких прикладна програма може змінювати. Біти 7 і 8 є прапорцями I і F відповідно. Ці прапорці використовуються для дозволу і заборони двох ліній переривань, які є зовнішніми по відношенню до ЦПП ARM7.

Як ми з вами побачимо пізніше, усі периферійні модулі мікроконтролерів LPC2300 підключені до цих двох ліній переривань. При роботі з даними бітами потрібно дотримуватися обережності, оскільки для заборони будь–якого з джерел переривань у відповідний біт необхідно записати 1, а не 0, як можна було б припустити. П'ятий біт регістра є прапорцем THUMB.

ЦПП ARM7 підтримує два набори команд – 32–бітний набір команд ARM і 16–бітний набір команд THUMB. Відповідно прапорець T показує, який з наборів команд використовується. Врахуйте, що програма не повинна

`

намагатися безпосередньо встановлювати або скидати цей прапорець для перемикання між наборами команд. Коректний механізм зміни поточного набору команд ми розглянемо трохи пізніше. Останні п'ять молодших бітів регістра CPSR є прапорцями режиму. Процесор ARM7 підтримує 7 режимів роботи (рисунок 5).

ЦПП ARM7 має 6 різних робочих режимів, які використовуються для обробки виняткових ситуацій. Затінені регістри представляють собою дублюючі регістри, які «включаються» при зміні режиму роботи. Регістр SPSR використовується для збереження вмісту регістра CPSR при перемиканні режимів.

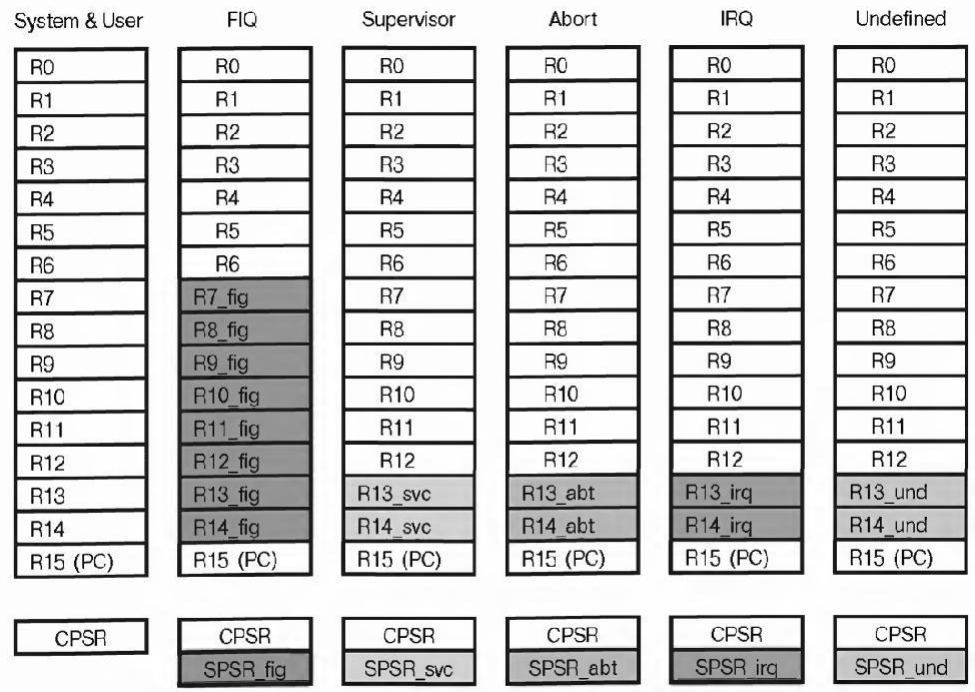


Рисунок 5 – Режими роботи процесора

Прикладні програми, як правило, виконуються в режимі User. У цьому режимі програма може звертатися до регістрів R0 ... R15 і CPSR. Однак при виникненні виняткових ситуацій (таких як переривання, помилка пам'яті, або виконання команди генерації програмного переривання) режим роботи процесора змінюється. При цьому регістри R0 ... R12 і R15 залишаються тими ж самими, а регістри R13 (SP) і R14 (LR) замінюються новою парою регістрів, унікальної для кожного режиму. Таким чином, кожен режим має власні регістр зв'язку та покажчик стека. Більш того, в режимі швидких переривань

`

(FIQ) дублюються і регістри R7 ... R12. Це дозволяє відразу приступити до обробки переривання FIQ, не витрачаючи час на збереження регістрів в стеку.

У кожному з режимів, за винятком режиму User, є додатковий регістр, який називається регістром збереженого стану програми (Saved Program Status Register – SPSR). Якщо в момент виникнення виняткової ситуації програма знаходилася в режимі User, відбувається зміна режиму і поточний зміст регістра CPSR зберігається в регістрі SPSR.

Після обробки виняткової ситуації (при поверненні з обробника) вміст регістра CPSR відновлюється з SPSR, забезпечуючи відновлення виконання прикладної програми.

5 Режими обробки виняткових ситуацій

При виникненні виняткової ситуації змінюється режим роботи ЦПП, і в PC завантажується адреса відповідного вектора переривання (таблиця 1). Таблиця векторів починається з нульової адреси. Першим в таблиці розташовано вектор скидання, а за ним інші вектори (по 4 байти на кожен).

При одночасному виникненні декількох виняткових ситуацій використовується метод пріоритетів. Пріоритети переривань наведено в таблиці 2.

Коли виникає виняткова ситуація, наприклад, переривання IRQ, процесор виконує наступні дії (рисунок 6). По–перше, адреса наступної виконуваної команди (PC + 4) зберігається в регістрі зв’язку.

Таблиця 1 – Адреси векторів переривань

Виняткова ситуація	**Режим**	**Адреса вектора**
Reset (скидання)	Supervisor	0x00000000
Undefined instruction (невизначена команда)	Undefined	0x00000004

Кожному режиму роботи відповідає свій вектор переривання. При зміні процесором режиму проводиться перехід з цього вектора. Зверніть увагу! Вектор за адресою 0x00000014 відсутній!

`

Продовження таблиці 1

Виняткова ситуація	**Режим**	**Адреса вектора**
SWI (програмне переривання)	Supervisor	0x00000008
Prefetch Abort (помилка звернення до пам'яті при вибірці команди)	Abort	0x0000000C
Data Abort (помилка звернення до пам'яті при доступі до даних)	Abort	0x00000010
IRQ (переривання)	IRQ	0x00000018
FIQ (швидке переривання)	FIQ	0x0000001C

Зауваження. *У таблиці векторів є «дірка», оскільки вектор з адресою 0x00000014 відсутній. Ця адреса використовувався в попередніх версіях процесорів ARM, а в процесорі ARM7 він збережений, щоб забезпечити програмну сумісність між різними архітектурами ARM. Однак, як ми побачимо пізніше, в мікроконтролерах сімейства LPC2300, ці чотири байти грають дуже важливу роль.*

Таблиця 2 – Пріоритети переривань

Пріоритет		**Виняткова ситуація**
Найвищий	1	Reset
	2	Data Abort
	3	FIQ
	4	IRQ
	5	Prefetch Abort
Найнижчий	6	Undefined instruction / SWI

Кожне джерело виняткової ситуації має фіксований пріоритет. Вбудовані периферійні пристрої обслуговуються перериваннями FIQ і IRQ. Пріоритети переривань від периферійних пристроїв можна призначати всередині цих груп.

Потім регістр CPSR копіюється в регістр SPSR кінцевого режиму (в нашому випадку SPSR irq). Після цього в PC заноситься адреса вектора переривання режиму виняткової ситуації. Для режиму IRQ це адреса – 0x00000018. У той самий час режим роботи процесора змінюється на IRQ, в результаті чого регістри R13 і R14 замінюються відповідними регістрами цього режиму.

`

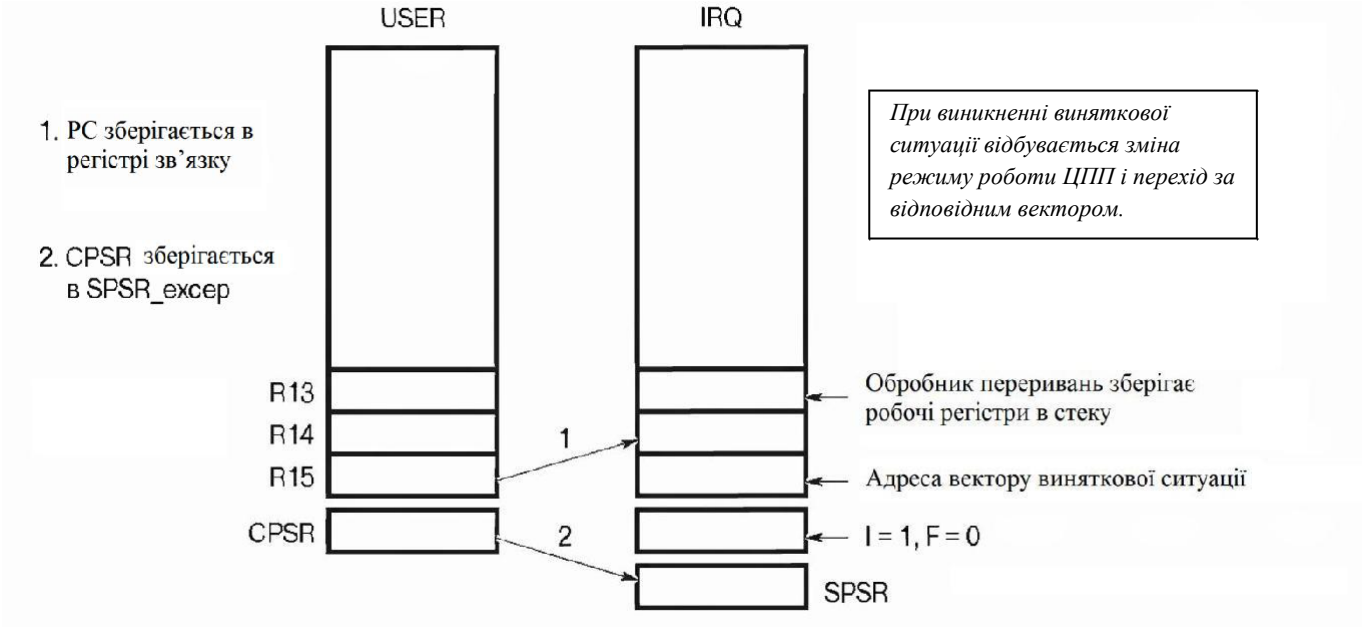


Рисунок 6 – Обробка виняткових ситуацій

При вході в режим IRQ встановлюється прапорець I регістра CPSR, що призводить до відключення лінії IRQ. Якщо потрібно використовувати вкладені переривання, то треба вручну дозволити переривання IRQ в програмі і занести вміст регістра зв'язку в стек, щоб зберегти вихідну адресу повернення.

З вектора переривання програма перейде до виконання підпрограми обробки переривань. Перше, що необхідно зробити в цій підпрограмі, – зберегти у стеку IRQ всі регістри з діапазону R0 ... R12, які будуть в ній використовуватися. Потім можна приступати до обробки виняткової ситуації.

Після завершення обробки виняткової ситуації необхідно повернутися в режим User і продовжити виконання програми з перерваного місця. Проте в наборі команд ARM відсутні команди типу «повернення» або «повернення з підпрограми», тому маніпуляції з лічильником команд PC необхідно здійснювати, використовуючи звичайні команди. Ситуація ускладнюється тим, що існує декілька різних варіантів повернення.

Для початку глянемо на команду SWI. При виконанні цієї команди адреса наступної виконуваної команди зберігається в регістрі зв'язку, після чо-

`

го проводиться обробка виняткової ситуації. Все, що потрібно зробити для повернення з виняткової ситуації, – це завантажити вміст регістра зв'язку назад в PC, і програма продовжить своє виконання.

Однак щоб ЦПП при цьому переключився назад у режим User, необхідно використовувати спеціальну команду пересилання MOVS (більш докладно ми розглянемо її трохи пізніше). Таким чином, команда повернення з програмного переривання буде наступною:

```
MOVS  R15,  R14          ; Скопіювати регістр зв'язку в PC і переключити режими.
```

А при виникненні виняткової ситуації за перериваннями FIQ і IRQ, поточна виконувана команда скидається і виконується перехід до обробника виняткової ситуації. При поверненні з виняткової ситуації в регістрі зв'язку знаходиться адреса відкинутої команди плюс 4. Щоб відновити виконання програми з потрібного місця, треба зменшити значення, що зберігається в регістрі зв'язку, на 4. В даному випадку для зменшення вмісту регістра зв'язку і збереження результату в PC ми використовуємо спеціальну команду віднімання, яка також відновлює і режим роботи ЦПП. Команда повернення з режимів FIQ, IRQ і Abort виглядає наступним чином:

```
SUBS  R15,  R14,  # 4          ; R15 <– R14 – #4.
```

У випадку, якщо відбулася помилка звернення до пам'яті, виняткова ситуація виникне через одну команду після тієї, виконання якої стало її причиною. В ідеалі, в цьому випадку треба перейти до підпрограми обробки переривання Data Abort, з'ясувати і усунути причину труднощів і знову спробувати виконати команду, що викликала виняткову ситуацію. Відповідно, треба «відмотати» PC назад на дві команди – відкинуту і ту, яка викликала виникнення виняткової ситуації. Іншими словами, потрібно відняти від регістра зв'язку число вісім і зберегти результат в PC. Таким чином, команда повернення з переривання Data Abort має вигляд:

`

```
SUBS  R15,  R14,  # 8                ; R15 <– R14 – #8.
```

При виконанні команди повернення модифікований вміст регістра зв'язку завантажується в лічильник команд, ЦПП перемикається назад у режим User, а вміст регістра SPSR переписується назад в CPSR. У разі виникнення виняткових ситуацій FIQ або IRQ додатково дозволяються відповідні переривання. В результаті всіх цих дій процесор виходить з привілейованого режиму і повертається до виконання користувальницької програми (рисунок 7).

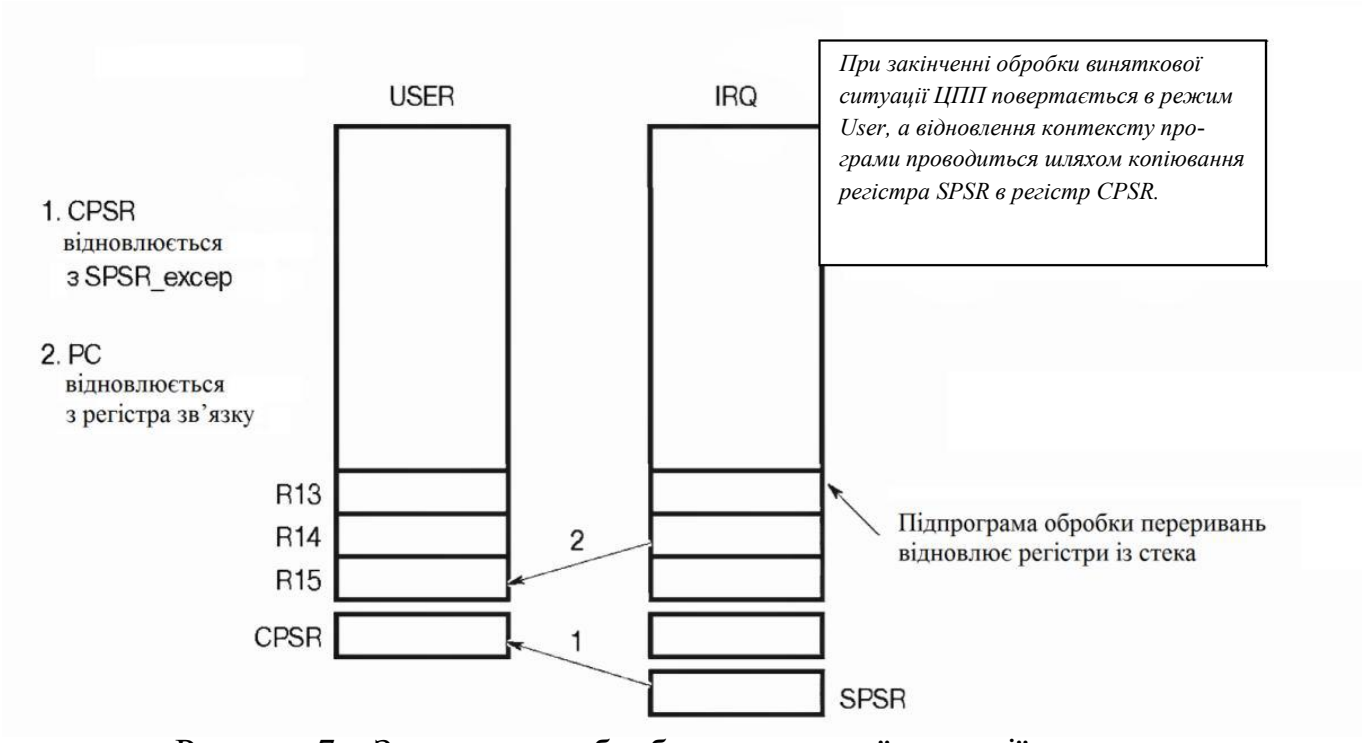


Рисунок 7 – Завершення обробки виняткової ситуації

6. Набір команд ARM7

Тепер, коли отримано загальне уявлення про ядро ARM, його моделі програмування і режими роботи, настав час познайомитися з його набором, або, якщо точніше, наборами команд. Оскільки сьогодні для програмування МК–в широко використовується мова Сі, немає необхідності бути експертом в області програмування на асемблері ARM7.

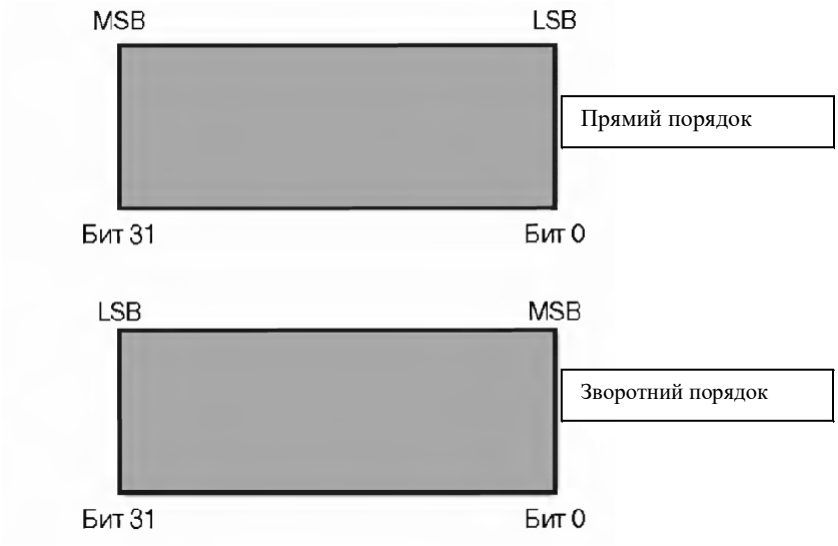
`

Однак, щоб розробляти дійсно ефективні програми, дуже важливо розуміти машинний код, який переховується за рядками програми на мові високого рівня. Перш ніж приступити до вивчення команд ARM7, необхідно зазначити, що насправді ЦПП ARM7 підтримує два набори команд: набір команд ARM з 32–бітними командами і набір команд THUMB з 16–бітними командами. Далі в посібнику слово ARM означатиме 32–бітний набір команд, а слово ARM7 – власне ЦПП.

Ядро ARM7 було розроблено таким чином, щоб його можна було використовувати як у якості процесора із зворотним порядком байтів (big–endian processor), так і в якості процесора з прямим порядком байтів (little–endian processor). У першому випадку старший біт (Most Significant Bit – MSB) 32–бітного слова розташовується на початку слова, а в другому випадку – в кінці (рисунок 8). В сімействі LPC2300 використовується тільки прямий порядок байтів (тобто MSB є бітом з самою старшою адресою), що значно полегшує роботу з процесором. Проте використовуваний вами компілятор для ARM7 повинен вміти компілювати код в обох форматах. У зв'язку з цим необхідно упевнитися, що формат слів задано правильно, інакше отриманий код буде «вивернуто навиворіт».

Одна з найбільш цікавих особливостей набору команд ARM полягає в тому, що кожна команда підтримує умовне виконання. У традиційних мікроконтролерах єдиними умовними командами є команди умовних переходів, і, можливо, ряд інших, таких як команди перевірки або зміни стану окремих бітів.

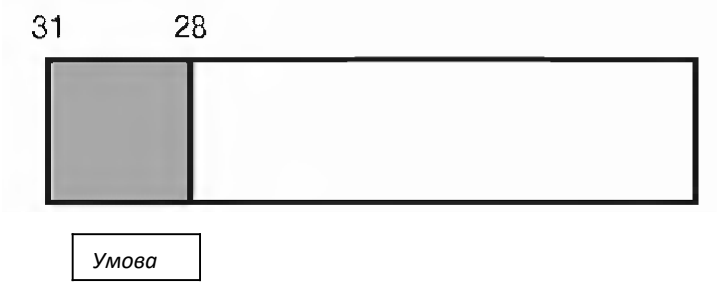
`



ЦПП ARM7 підтримує як прямий, так і зворотний порядок байтів. Тим не менш, в мікроконтролерах NXP використовується тільки прямий порядок байтів (little endian).

Рисунок 8 – Прямий і зворотний порядок байтів

В наборі команд ARM старші 4 біти коду команди завжди порівнюються з прапорцями умов в регістрі CPSR (рисунок 9). Якщо їх значення не співпадають, команда не виконується і проходить через конвеєр як команда NOP (немає операції).



Кожна команда ARM (32-бітна) є умовно виконуваної. Між 4 старшими бітами коду команди і прапорцями умов регістра CPSR проводиться операція «Логічне І». Якщо значення не співпадають, виконується команда NOP.

Рисунок 9 – Положення бітів порівняння в команді ARM

Таким чином, можна виконати будь–яку команду обробки даних, що змінює прапорці умов в регістрі CPSR. Потім, залежно від результату, наступна команда може бути виконана, а може і ні. До базових мнемонічних позначень команд асемблера, таким як MOV або ADD, можна додати будь–який з шістнадцяти префіксів, що визначають тестований стан прапорців умов (таблиця 3).

`

Таблиця 3 – Префікси команд

КОД	Префікс	Прапорці	Значення
0000	EQ	Z встановлений	Дорівнює
0001	NE	Z скинутий	Не дорівнює
0010	CS	C встановлений	Більше або дорівнює (беззнакове)
0011	CC	C скинутий	Менше (беззнакове)
0100	MI	N встановлений	Від’ємний результат
0101	PL	N скинутий	Додатний результат
0110	VS	V встановлений	Переповнення
0111	VC	V скинутий	Немає переповнення
1000	HI	C встановлений, Z скинутий	Більше (беззнакове)
1001	LS	C скинутий, Z встановлений	Менше або дорівнює (беззнакове)
1010	GE	N дорівнює V	Більше або дорівнює (беззнакове)
1011	LT	N не дорівнює V	Менше (знакове)
1100	GT	Z скинутий I (N = V)	Більше (знакове)
1101	LE	Z встановлений АБО (N < > V)	Менше або дорівнює (знакове)
1110	AL	(ігнорується)	Безумовне виконання

Наприклад, команда:

```
EQMOV    R1, # 0x00800000
```

виконує завантаження числа 0x00800000 в регістр R1 тільки в тому випадку, якщо результат виконання останньої команди обробки даних був «дорівнює» і відповідно встановлено прапорець Z регістра CPSR. Метою такого умовного виконання команд є забезпечення безперервності потоку команд через конвеєр, тому що при кожному виконанні команд переходу конвеєр скидається і на його повторне заповнення потрібен час, що різко знижує загальну продуктивність. На практиці існує певний поріг, при якому примусове «проштовхування» команд NOP через конвеєр виявляється ефективніше ви-

`

конання традиційних команд умовного переходу і пов'язаного з цим повторним заповненням буферу.

Зазначений поріг дорівнює трьом командам, тому короткий перехід, такий як:

```
        if ( x < 100 )
{               x++;            }
```

при використанні умовно–виконуваних команд ARM буде реалізовано більш ефективно.

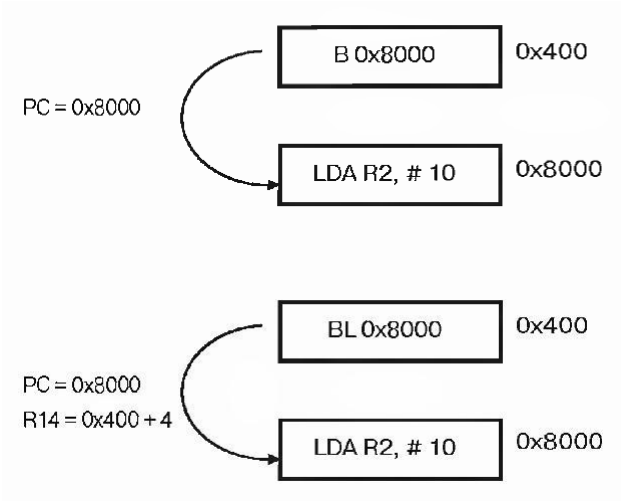
Всі команди ARM можна розбити на 6 основних груп: команди розгалуження, команди обробки даних, команди передачі даних, команди передачі блоків даних, команди множення і команда програмного переривання.

6.1 Команди розгалуження

Базова команда переходу (B), як випливає з її назви, дозволяє виконувати перехід в діапазоні до 32 Мбайт як вперед, так і назад. Модифікована версія команди, команда переходу зі збереженням адреси (BL), виконує ту ж операцію, проте при цьому зберігає в регістрі зв'язку поточне значення PC, яке збільшене на чотири (рисунок 10).

Таким чином, команда переходу зі збереженням адреси використовується як команди виклику підпрограм, що зберігає адресу повернення в регістрі зв'язку. Для повернення з підпрограм можна використовувати команду звичайного переходу, що виконує перехід за адресою, яка знаходиться в регістрі зв'язку.

`

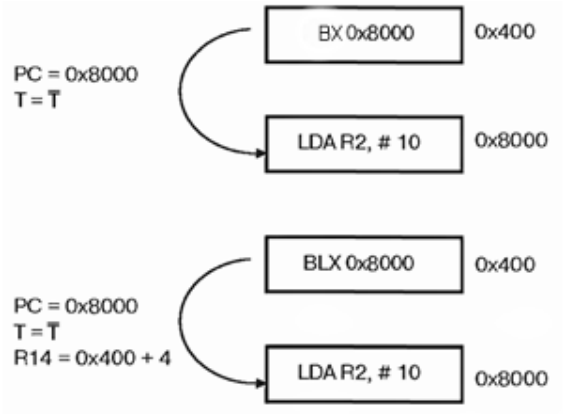


Команда переходу має кілька різновидів. Звичайна команда переходу (B) здійснює перехід за заданою адресою. Команда переходу зі збереженням адресу (BL) здійснює перехід за заданою адресою, зберігаючи при цьому адресу повернення в регістрі R14.

Рисунок 10 – Команди переходу B і BL

Використовуючи префікси умов, виконуються умовні переходи і умовні виклики підпрограм. Існує ще два різновиди команди переходу: «перехід зі зміною стану» (BX) і «перехід зі зміною стану та збереженням адреси» (BLX). Ці команди виконують ті ж операції, що і попередні команди, але при цьому ще й виконують переключення з набору команд ARM на THUMB і назад (рисунок 11).

Це єдиний спосіб, який трнба застосовувати для зміни використовуваного набору команд, так як безпосередні маніпуляції з прапорцем T регістра CPSR можуть привести до непередбачуваних результатів.



Команди переходу зі зміною стану (BX) і переходу зі зміною стану та збереженням адреси (BLX) виконують такі ж операції, як і звичайні команди переходу та переходу зі збереженням адреси (B і BL), додатково здійснюючи перемикання між наборами команд ARM і THUMB.

Рисунок 11 – Команди переходу BX і BLX

`

6.2 Команди обробки даних

Команди обробки даних наведено в таблиці 4, а узагальнений формат всіх команд обробки даних наведено на рисунку 12.

Таблиця 4 – Команди обробки даних

Мнемокод	Опис команди
AND	Логічне побітове « І »
EOR	Логічне побітове « виключне АБО »
SUB	Віднімання
RSB	Зворотне віднімання
ADD	Додавання
ADC	Додавання з урахуванням перенесення
SBC	Віднімання з запозиченням
RSC	Зворотне віднімання з запозиченням

Мнемокод	Опис команди
TST	Перевірка бітів
TEQ	Побітове порівняння
CMP	Порівняння
CMN	Порівняння з запереченням
ORR	Логічне побітове « АБО »
MOV	Пересилання
BIC	Скидання бітів (маскування)
MVN	Пересилання з інверсією

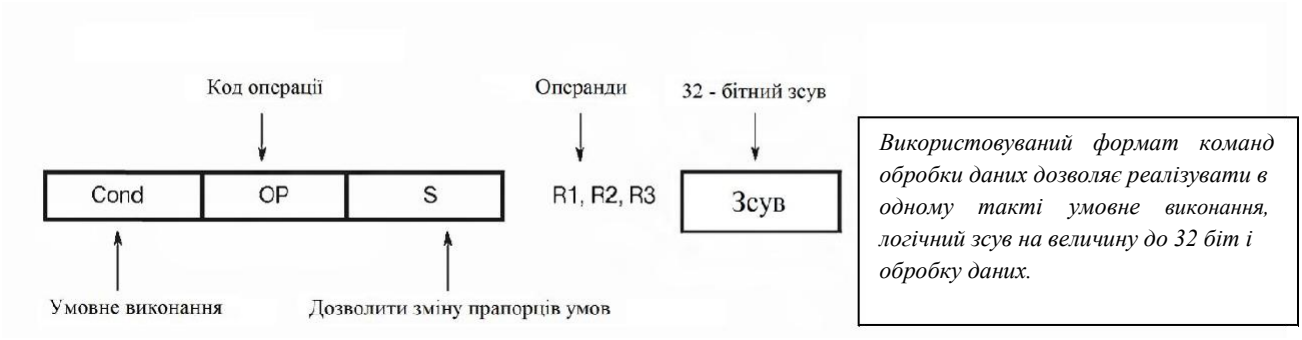


Рисунок 12 – Формат команд обробки даних

У кожній команді є регістр результату і два операнди. Перший операнд обов'язково повинен бути регістром,тоді як другий може бути як регістром, так і константою.

`

Крім цього, в ЦПП ARM7 є багаторегістровий пристрій циклічного зсуву (barrel shifter), що дозволяє при виконанні команди зсувати значення 2–го операнда на величину до 32 біт. Біт S використовується для керування прапорцями умов. Якщо цей біт встановлено, прапорці умов змінюються відповідно до результату виконання команди.

Якщо цей біт скинуто, стан прапорців умов не змінюється. Однак якщо при встановленому біті S в якості регістра результату вказаний лічильник команд (R15), проводиться копіювання вмісту регістра SPSR поточного режиму в регістр CPSR. Ця можливість використовується для відновлення PC і перемикання в початковий режим в кінці обробки виняткових ситуацій. Не намагайтеся виконати таку команду в режимі User, оскільки в цьому режимі відсутній регістр SPSR і відповідно результат виконання цієї команди неможливо передбачити.

Ці особливості надають нам багатий набір команд обробки даних (таблиця 4), який, з одного боку, дозволяє створювати дуже ефективні програ-ми, а з іншого – є джерелом «нічних кошмарів» для розробників компілято-рів.

Наприклад, в результаті компіляції виразу

```
if (Z == 1)
R1 = R2 + (R3 x 4);
```

може бути згенерована наступна команда:

```
EQADDS R1, R2, R3, LSL # 2.
```

6.3 Команди передачі даних

6.3.1 Команди завантаження/збереження

Наступну групу складають команди передачі даних (таблиця 5). ЦПП ARM7 підтримує команди завантаження / збереження, які дозволяють пере-

`

силати знакові і беззнакові числа різного розміру (слово, напівслово, байт) в / із заданого регістра.

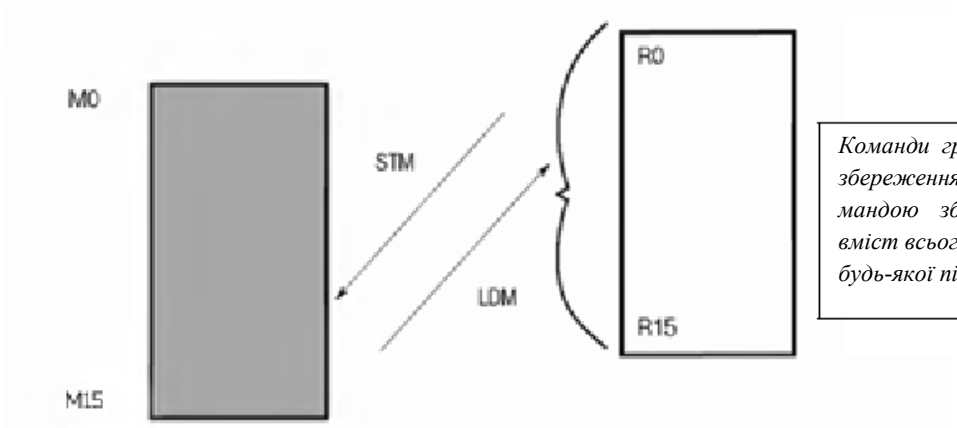
Таблиця 5 – Команди передачі даних

Мнемокод	Опис команди	Мнемокод	Опис команди
LDR	Завантажити слово	STR	Зберегти слово
LDRH	Завантажити напівслово	STRH	Зберегти напівслово
LDRSH	Завантажити напівслово зі знаком	STRSH	Зберегти напівслово зі знаком
LDRB	Завантажити байт	STRB	Зберегти байт
LDRSB	Завантажити байт зі знаком	STRSB	Зберегти байт зі знаком

Оскільки набір регістрів повністю незалежний, можна завантажувати 32–бітове значення безпосередньо в PC, здійснюючи, таким чином, перехід в межах всього адресного простору процесора. Якщо кінцева адреса лежить поза діапазоном команди переходу, можна просто завантажити збережену константу в лічильник команд.

3.2 Групове копіювання регістрів

Крім команд завантаження / збереження вмісту окремих регістрів, в наборі команд ARM є команди для завантаження (LDM) і збереження (STM) груп регістрів (рисунок 13). Таким чином, за допомогою однієї команди можна скопіювати в пам'ять весь блок регістрів або його частину, а за допомогою іншої – відновити його вміст.



Команди групового завантаження / збереження дозволяють однією командою зберігати / відновлювати вміст всього регістрового файлу або будь-якої підмножини регістрів.

Рисунок 13 – Команди LDM і STM

`

6.4 Команда обміну

У наборі команд ARM є також команда обміну (SWP), завдяки якій забезпечується підтримка семафорів реального часу. Ця неперервна команда здійснює одночасний обмін вмісту двох регістрів (рисунок 14). Завдяки такому рішенню запобігається переривання процесу обміну критичними даними при виникненні виняткової ситуації. У мові Сі ця команда безпосередньо недоступна та підтримується вбудованими функціями бібліотек компіляторів.

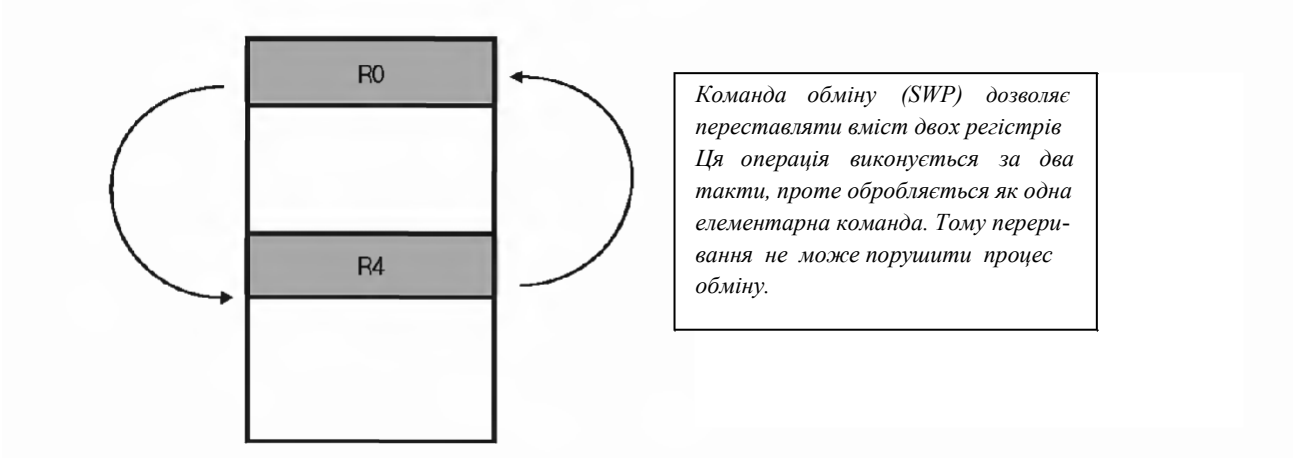


Рисунок 14 – Команди обміну

6.5 Команди зміни регістрів стану

Як вже було зазначено вище регістри CPSR і SPSR є регістрами ЦПП, однак не входять до складу основного банку регістрів. Безпосередньо звертатися до цих регістрів можуть тільки дві команди ARM – MSR і MRS. Зазначені команди забезпечують пересилання вмісту регістра CPSR або SPSR в / із заданого регістра (рисунок 15). Наприклад, щоб заборонити переривання IRQ, необхідно скопіювати вміст регістра CPSR в робочий регістр, встановити прапорець I (шляхом виконання операції «I» між цим регістром і числом 0x00000080) і завантажити отримане значення назад в регістр CPSR. Команди MSR і MRS доступні у всіх режимах процесора, за винятком режиму User.

`

Таким чином, тільки перебуваючи в привілейованому режимі, можна змінювати робочий режим процесора і дозволяти / забороняти переривання,.

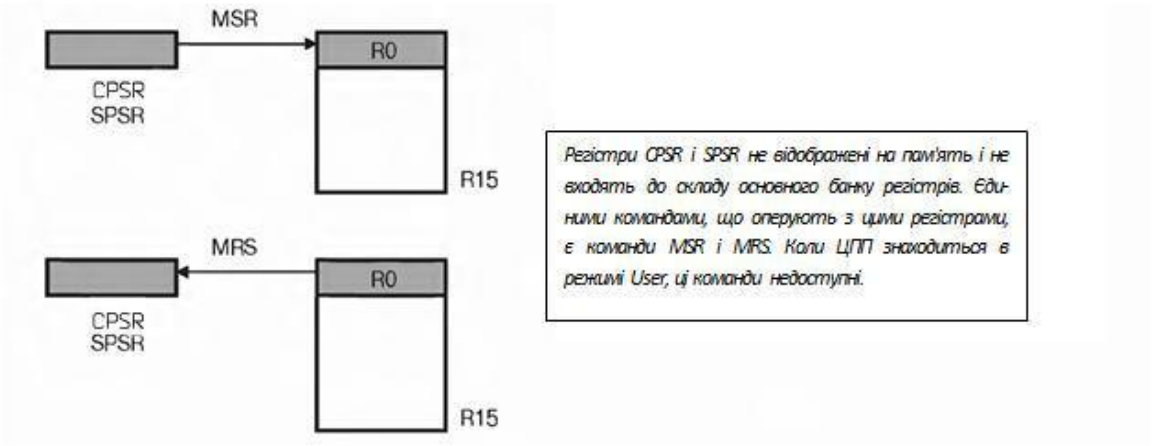


Рисунок 15 – Зміна регістрів стану

Після входу в режим User ви можете з нього вийти тільки при виникненні виняткової ситуації, скиданні, генерації переривань FIQ і IRQ або ж в результаті виконання команди SWI.

6.6 Команда програмного переривання

Команда програмного переривання SWI генерує виняткову ситуацію, в результаті чого процесор перемикається в режим Supervisor, а в лічильник команд заноситься значення 0x00000008. Як і всі інші команди ARM, команда SWI містить в чотирьох старших бітах прапорці умовного виконання, за якими розташовується код операції (рисунок 3.16). Інша частина слова команди залишається вільною. Однак у цих невикористовуваних бітах може зберігатися число. Дані біти можна перевіряти на початку підпрограми обробки переривання, щоб визначити, яку саме частину підпрограми слід виконувати. Таким чином, за допомогою команди SWI можна перемикатися в захищений режим для виконання привілейованих ділянок програми або обробки системних викликів.

`

Команда програмного переривання (SWI) перемикає ЦПП в режим Supervisor і переходить за адресою вектора SWI. Біти О...23 не задіяні, проте в них можна передавати значення, визначені користувачем.

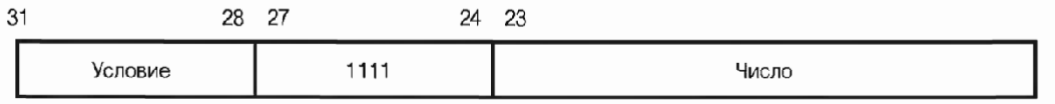


Рисунок 16 – Команда SWI

Після компіляції команди

SWI #3

в невикористовуваних бітах слова команди буде записано число 3. У підпрограмі обробки переривання SWI ми можемо перевірити значення слова команди наступним чином (текст написаний на псевдокоді):

```
switch ( * (R14 – 4) & OxOOFFFFFF)          / / Повернутися на 4 байти назад
 {                                          / / Замаскувати старші 8 бітів і
                                            / / Виконати перехід
                                            / / відповідно до результату
case (SWI–1)
        ...
```

Залежно від використовуваного компілятора доводиться реалізовувати таку перевірку самостійно або ж компілятор автоматично вставить необхідні команди в код програми.

6.7 Команди множення чисел

Поряд з багаторегістровим пристроєм циклічного зсуву в ядрі ARM7 є вбудований модуль помножувача / суматора (MAC). Модуль MAC підтримує множення чисел типу integer і long integer. Команди множення чисел типу integer виконують множення двох 32–бітних регістрів і поміщають результат в третій 32–бітний регістр. Команда множення з накопиченням виконує мно-

`

ження і додає результат до проміжної суми. Команди множення чисел типу long integer перемножують вміст двох 32–бітних регістрів і поміщають 64–бітний результат у два регістри. Аналогічно, є команда довгого множення з накопиченням (таблиця 6).

Таблиця 3.6 – Команди множення

Мнемокод	**Опис команди**	**Роздільна здатність**
MUL	Множення	32–бітний результат
MULA	Множення з накопиченням	32–бітний результат
UMULL	Беззнакове множення	32–бітний результат
UMLAL	Беззнакове множення з накопиченням	32–бітний результат
SMULL	Знакове множення	32–бітний результат
SMLAL	Знакове множення з накопиченням	32–бітний результат

7 Набір команд Thumb

Незважаючи на те, що ARM7 є 32–бітовим процесором, він підтримує ще один набір команд (16–бітний), який називається THUMB (рисунок 17). Насправді, цей набір команд є стислою формою набору команд ARM. За рахунок цього команди, збережені в 16–бітному форматі, розпаковуються в команди ARM, а потім виконуються. Хоча команди THUMB забезпечують меншу продуктивність в порівнянні з командами ARM, завдяки їм досягається більш висока щільність коду. Таким чином, для створення компактних програм, що розміщуються в невеликих однокристальних мікроконтролерах, код програм необхідно компілювати у вигляді сукупності функцій THUMB і ARM.

`

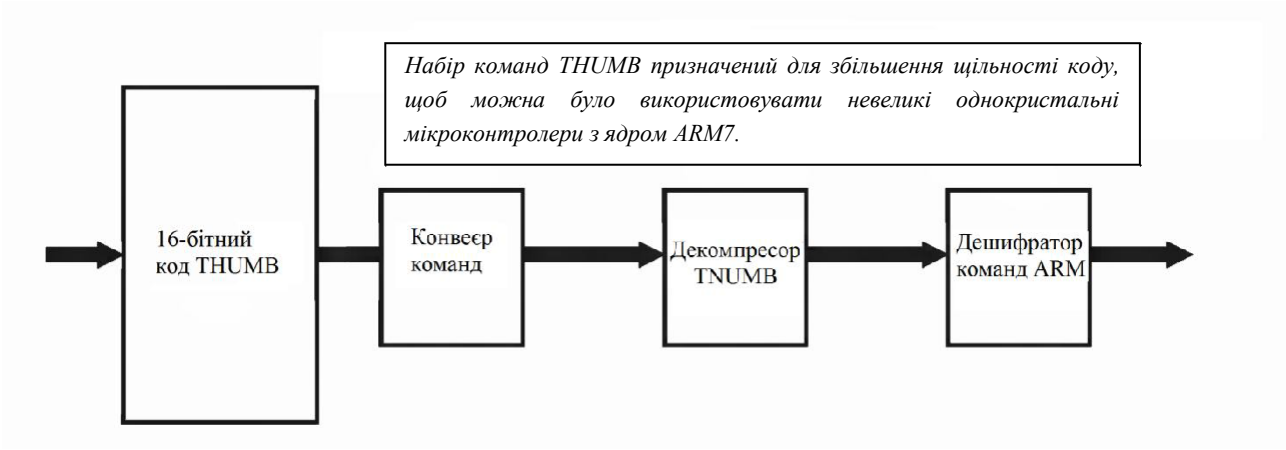


Рисунок 17 – Набір команд THUMB

Цей процес називається interworking і елементарно підтримується всіма компіляторами. При компіляції з використанням набору команд THUMB ви отримаєте 30–відсоткову економію пам'яті програм, у той час як цей же код, скомпільований з використанням команд ARM, буде виконуватися на 40% швидше.

Набір команд THUMB набагато більше схожий на набори команд традиційних мікроконтролерів. На відміну від команд ARM, команди THUMB не підтримують умовного виконання (за винятком команд умовних переходів). Команди обробки даних мають двоадресний формат, причому як регістр результату використовується один з регістрів–джерел:

Команда ARM	Команда THUMB	Дія
ADD R0, R0, R1	ADD R0, R1	R0 = R0 + R1

Команди THUMB не мають повного доступу до всіх регістрів регістрового файлу (рисунок 18). З регістрами R0 ... R7 (так званими молодшими регістрами) можуть працювати всі команди обробки даних. А до регістрів R8

.... R12 (старші регістри) можуть звертатися тільки 3 з них:

MOV, ADD, CMP.

`

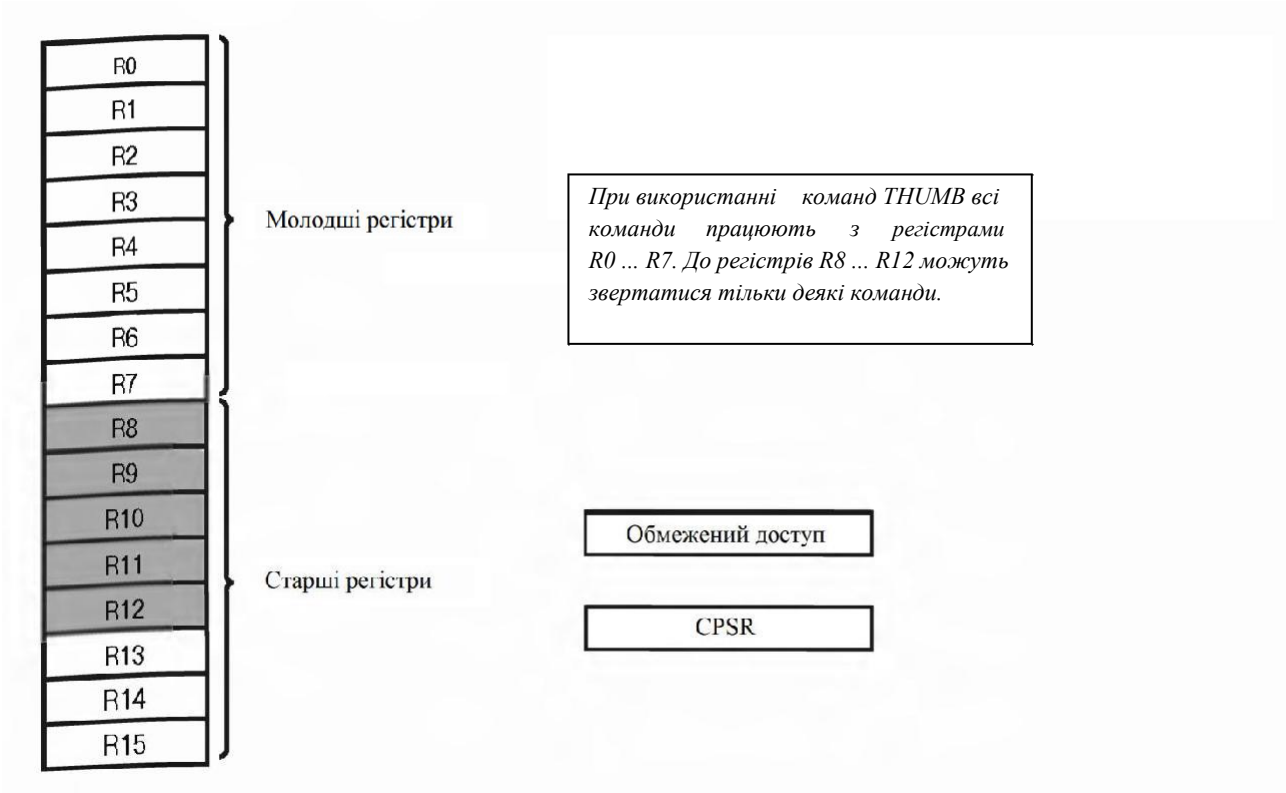


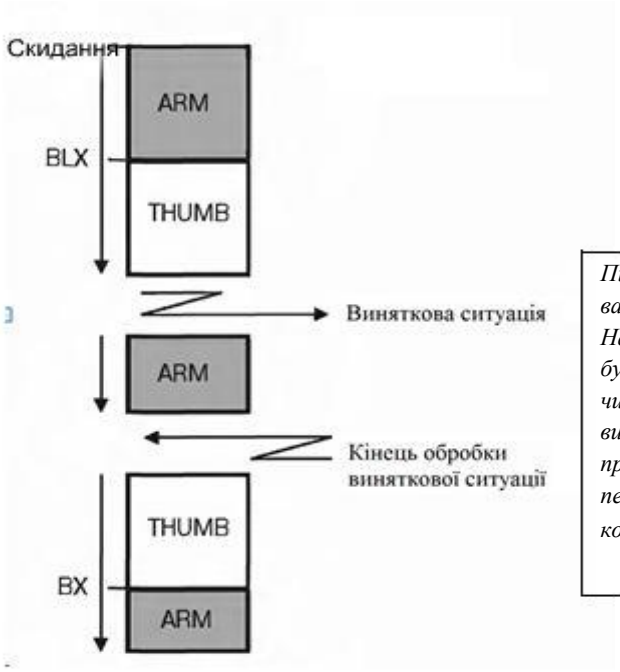
Рисунок 18 – Доступ до регістрового файлу

У наборі команд THUMB відсутні команди MSR і MRS, тому змінювати регістри CPSR і SPSR можна тільки за допомогою непрямої адресації. Якщо треба змінити будь–які користувацькі біти в регістрі CPSR, необхідно переключитися в режим ARM. Змінювати режими можна за допомогою команд BX і BLX (рисунок 19). Крім того, автоматичне перемикання в режим ARM відбувається при скиданні, або при переході до обробки виняткової ситуації.

У складі набору команд THUMB є більш звичні команди роботи зі стеком PUSH і POP (рисунок 20). За допомогою цих команд реалізується спадаючий стек, який жорстко прив'язаний до регістра R13.

І, нарешті, в наборі команд THUMB є команда SWI, що працює так само, як і аналогічна команда набору ARM, але містить тільки 8 невикори–стовуваних бітів, що обмежує максимальну кількість викликів SWI до 255.

`



Після скидання ARM7 буде виконувати команди ARM (32-бітні). Набір команд можна змінити в будь-який момент, використовуючи команду BX, BLX. При виникненні виняткової ситуації процесор автоматично перемикається на 32-бітний набір команд ARM.

Рисунок 19 – Перемикання в режим ARM

У *наборі команд THUMB є спеціальні команди PUSH і POP, які реалізують спадаючий стек, використовуючи в якості покажчика стека регістр R13.*

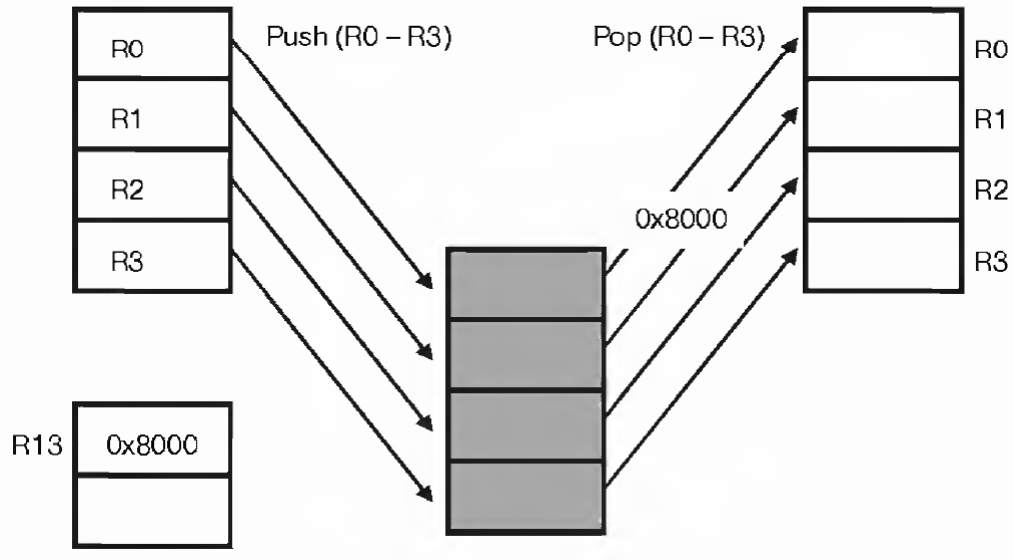


Рисунок 20 – Команди PUSH і POP

`

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1) Охарактеризуйте конвеєр команд ЦПП ARM7.
2) Яку має архітектуру процесор ARM7?
3) Перелічіть регістри ARM7 та опишіть структуру основного регістрового файлу.
4) Опишіть призначення та формат регістра поточного стану програми.
5) Які режими роботи підтримує процесор ARM7?
6) Опишіть відмінність режимів обробки виняткових ситуацій від основних режимів роботи процесора.
7) Чим відрізняються процесори із зворотним порядком байтів (big–endian processor) від процесорів з прямим порядком байтів (little–endian processor)?
8) Як команди процесорів ядра ARM7 підтримують умовне виконання?
9) Дайте пояснення префіксам команд (їх прапорці і значення): EQ; NE; CS; CC; VC; VS; MI; PL; HI; LS; GE; LT; GT; LE; AL.
10) Опишіть особливості виконання наступних груп команд: переходів; обробки даних; копіювання регістрів; копіювання груп регістрів; обміну; зміни регістрів стану; програмного переривання та множення..