Лекція СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ ТА ФОРМАТИ КОМАНД

В командах ARM використовуються наступні види адресацій:

–неявна;

–регістрова;

–непряма;

–безпосередня;

–преіндексна з регістровим зміщенням без оновлення бази;

– преіндексна з масштабованим регістровим зміщенням без оновлення бази;

–преіндексна з безпосереднім зміщенням без оновлення бази;

–преіндексна з регістровим зміщенням та оновленням бази;

– преіндексна з масштабованим регістровим зміщенням та оновленням бази;

–преіндексна з безпосереднім зміщенням та оновленням бази;

–постіндексна з регістровим зміщенням;

–постіндексна з масштабованим регістровим зміщенням;

–постіндексна з безпосереднім зміщенням.

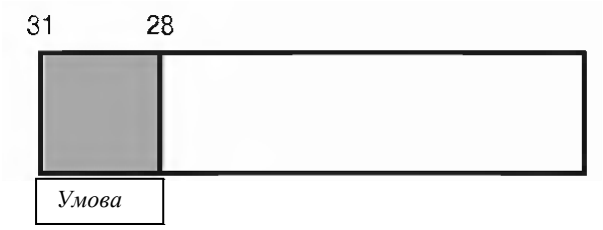
Всі команди ARM можна розбити на 6 основних груп: команди обробки даних, команди розгалуження, команди передачі даних, команди передачі блоків даних, команди множення, команда програмного переривання і команди роботи з співпроцесами.

1 Префікси команд

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

`

А в наборі команд ARM старші 4 біти коду команди завжди порівнюються з прапорцями умов в регістрі CPSR (рисунок 1).



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 1 – Положення бітів порівняння в команді ARM

Якщо їх значення не співпадають, команда не виконується і проходить через конвеєр як команда NOP (немає операції).

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 M0V або ADD, можна додати будь–який з шістнадцяти префіксів, що визначають тестовий стан прапорців умов (таблиця 1).

Таблиця 1 – 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

`

Продовження таблиці 1

КОД	Префікс	Прапорці	Значення
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
```

виконає 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 виконання традиційних команд умовного переходу і пов'язаного з цим повторним заповненням буфера.

Встановлення прапорців CPSR не є обов'язковою, і управляється бітом S інструкції.

`

2 Формати команд обробки операндів

На рисунку 2 наведено машинний код (формат) команд обробки операндів.

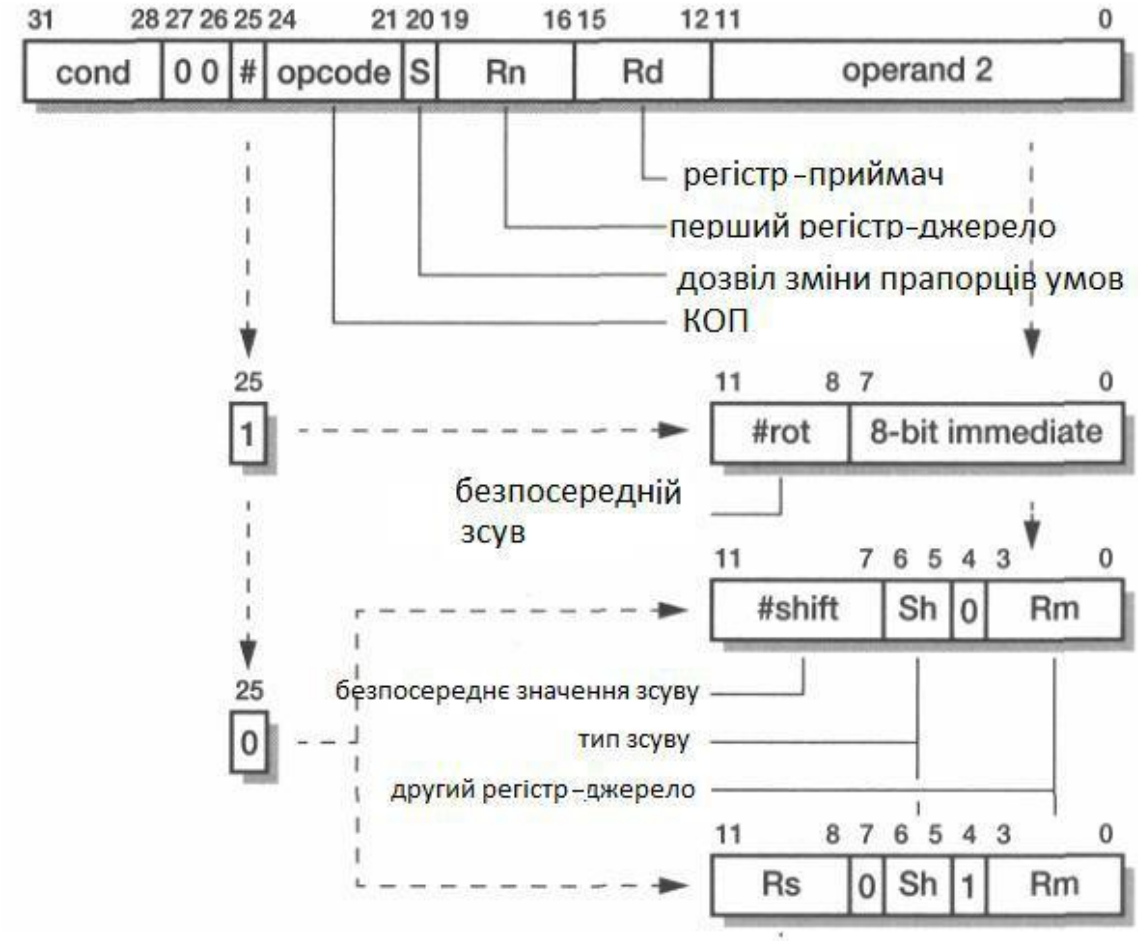


Рисунок 2 – Формат команд обробки операндів

Нижче наведено пояснення окремих полів формату команд, наведеного на рисунку 4.2:

– cond (біти 31…28) формує компілятор в залежності від префіксу команди (таблиця 3.3), якщо останній використовується;

–розряди 27, 26 дорівнюють нулю;

– # (25 р) залежить від того, де знаходиться операнд, який може зсуватися: якщо 25 р = 1, зсувається безпосередній 8–розрядний операнд (8–bit immediate) на величину зсуву, яка визначається полем #rot

`

(11 р…8 р); якщо 25 р = 0, зсувається вміст регістра Rm на величину зсуву, яка визначається полем #shift (11 р…7 р), або полем Rs (11 р…8 р), яке визначає один з регістрів, в яких знаходиться величина зсуву;

–opcode (24 р…21 р) являє собою код операції команди (КОП), який визначається мнемокодом команди;

– S (20 р) дозволяє/забороняє зміну прапорців умов за результатом виконання команди, наприклад:

AND R1, #0x0 ; не змінюватиме прапорці;

ANDS R1, #0x0; встановить прапорець Z в одиницю.

В інструкціях CMP, CMN, TST та TEQ цей розряд завжди = 1;

–Rn (19 р…16 р) визначає 1–й операнд–джерело (один з регістрів основного регістрового файлу);

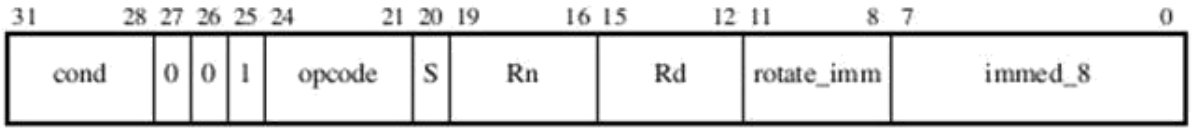
–Rd (15 р…12 р) визначає регістр–приймач (один з регістрів основного регістрового файлу);

– operand 2 (11 р…0 р) визначає 2–й операнд, вміст якого може зсуватися: #rot – число: 0…15, яке визначає величину зсуву 2–го 8–бітного безпосереднього операнда (8–bit immediate); #shift – визначає величину зсуву: (0…31), а поле Sh (6 р, 5 р) – визначає тип зсуву: Sh = 00 – арифметичний/логічний вліво; 01 логічний вправо; 10– арифметичний вправо; 11 – циклічний вправо; 4 р = 0, що визначає, що зсувається в залежності від поля Sh операнд Rm на величину #shift; Rm визначає другий операнд (один з регістрів основного регістрового файлу); якщо 4 р = 1, то поле Rs (11 р…8 р) визначає один з регістрів основного регістрового файлу, в якому записано величину зсуву, при цьому 7 р = 0, а 6 р, 5 р виконують функцію поля Sh, яка описана вище.

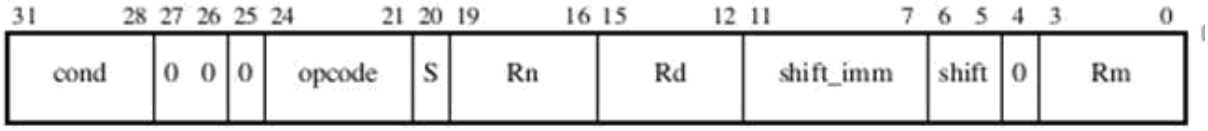
На рисунку 3 наведено приклади трьох форматів команд, які відповідають рисунку 2.

`

32-bit immediate 32-біти (безпосередній операнд)



Безпосередній зсув



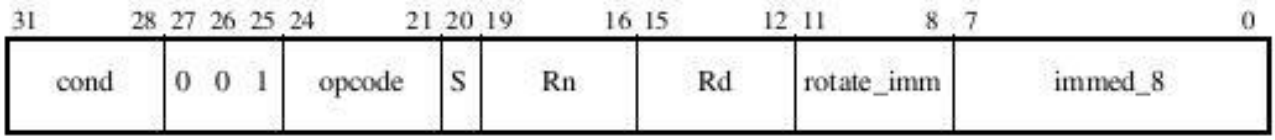
Регістровий зсув

Рисунок . – Види форматів команд при обробці операндів

Нижче наведено приклади мнемокодів та форматів команд, які відповідають рисунку 3.

2.1 Безпосередній операнд

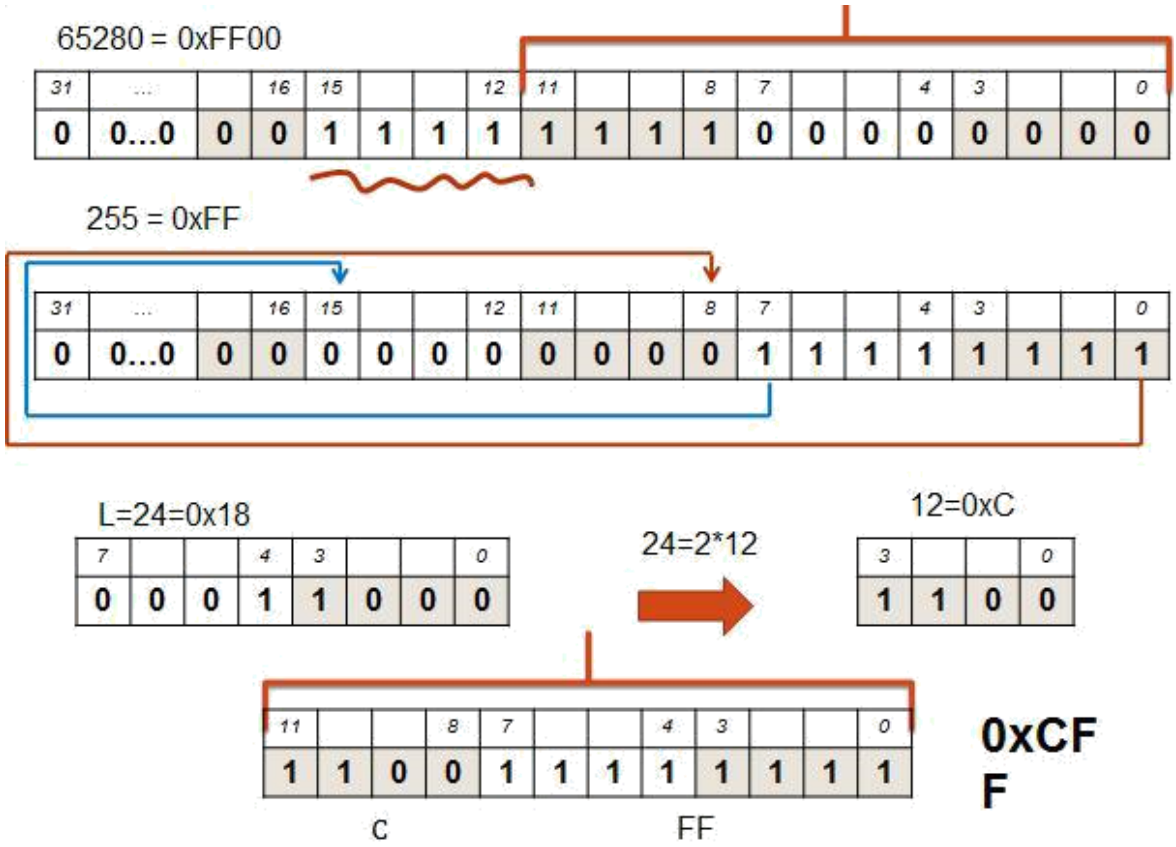
Операндом в даному випадку являється 32–бітне число. При цьому, дозволені лише ті числа. які можуть бути утворені циклічним зсувом вправо 8–бітного значення (immed – 8) на парну кількість позицій.



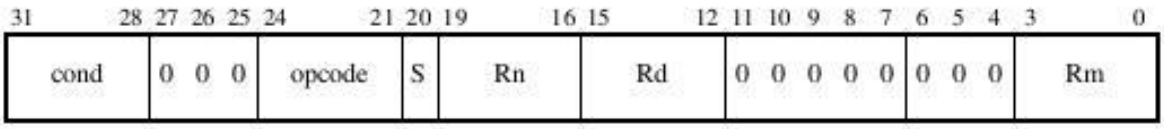
ADD R1, R2, #0xFF00 ; R1=R2+ 65280
E2821CF

1110 0010 1000 0010 0001 1100 1111 1111

`



2.2 Регістровий операнд



MOV R2, R0 ; *R2=R0*

MOV R2, R1 ; *R2=R1*

ADD R4, R3, R2 ; *R4=R3+R2*

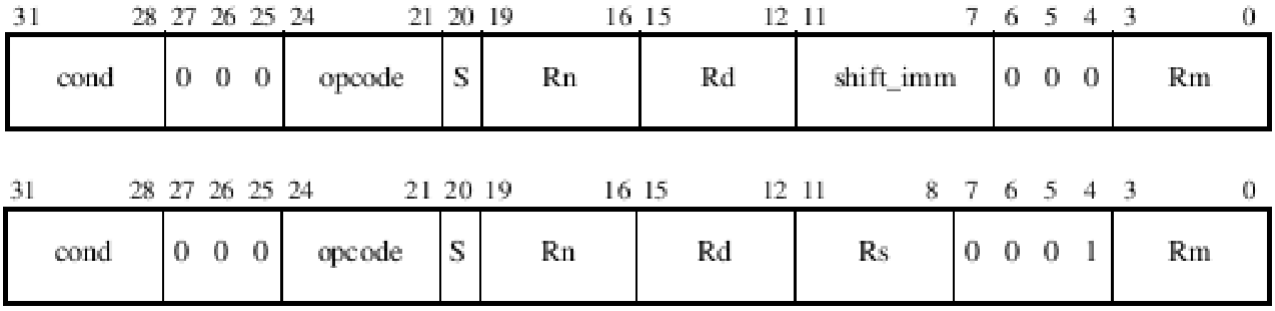
E1A02000 = 1110 0001 1010 0000 0010 0000 0000 0000

E1A02001 = 1110 0001 1010 0000 0010 0000 0000 0001

E0834002 = 1110 0000 1000 0011 0100 0000 0000 0010

`

2.3 Логічний зсув регістра вліво



<Rm>, LSL #<shift_imm>;

<Rm>, LSL <Rs>;

<Rm> вказує регістр, чиє значення буде

зсунуто; LSL вказує логічний зсув вліво;

<shift_imm> вказує зсув. Значення у діапазоні

0..31; <Rs> – регістр, що містить значення зсуву;

ADD R9, R4, R5, LSL #3 ; R9 = R4 + R5 x 8

ADD R9, R4, R5, LSL R1 ; R9 = R4 + R5 x R1

2.4 Логічний зсув регістра вправо

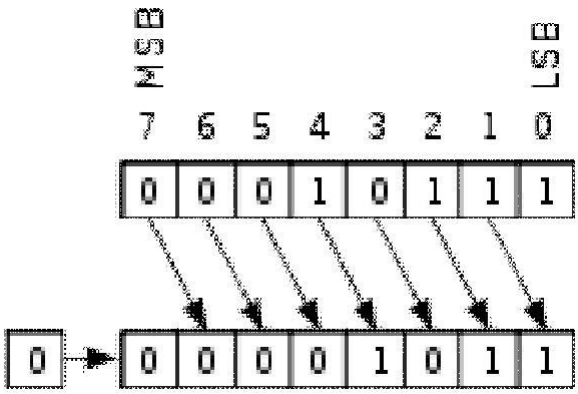
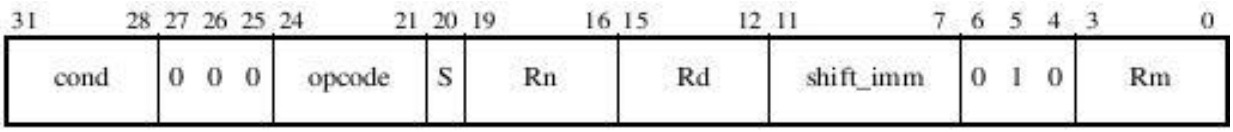
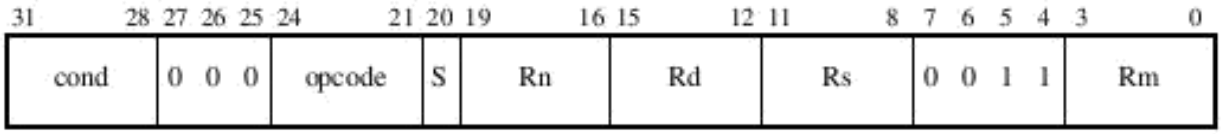


Рисунок 3 – Логічний зсув вправо



`



<Rm>, LSR #<shift_imm>

<Rm>, LSR <Rs>

<*Rm*> вказує регістр, чиє значення буде зсунуто

<*shift_imm* > вказує зсув (1..32, при значенні 32 : shift_imm = 0.)

<*Rs*> регістр, що містить значення зсуву

ADD R9, R3, R5, LSR #3; R9=R3+R5/8

ADD R9, R3, R5, LSR R1; R9=R3+R5/R1

2.5 Арифметичний зсув регістра вправо

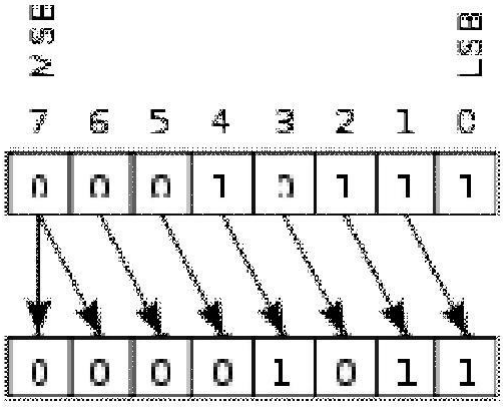
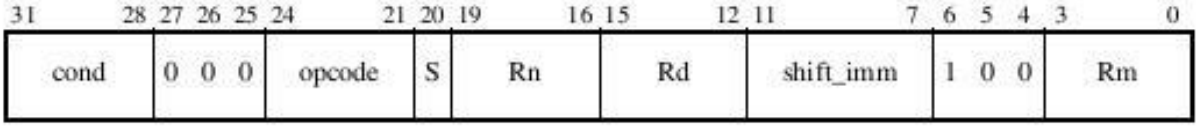


Рисунок 4 – Арифметичний зсув вправо



`

<Rm>, ASR #<shift_imm>

<Rm>, ASR <Rs>

<*Rm*> вказує регістр, чиє значення буде зсунуто

<*shift_imm* > вказує зсув (1..32, при значенні 32 : shift_imm = 0.)

<*Rs*> регістр, що містить значення зсуву

ADD R9, R2, R5, ASR #3; R9=R2+R5/8

ADD R9, R2, R5, ASR R1; R9=R2+R5/R1

2.6 Циклічний зсув вправо

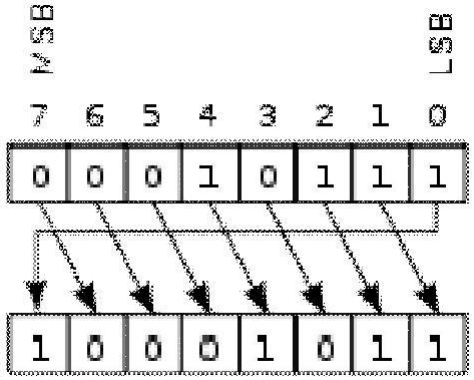
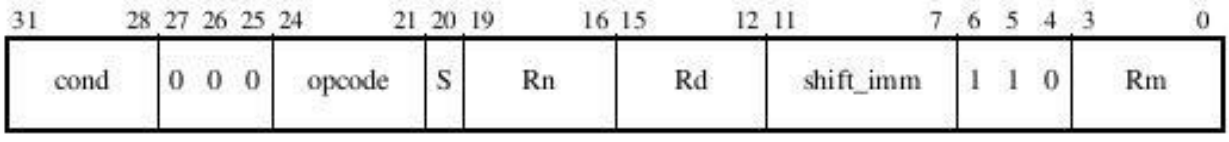
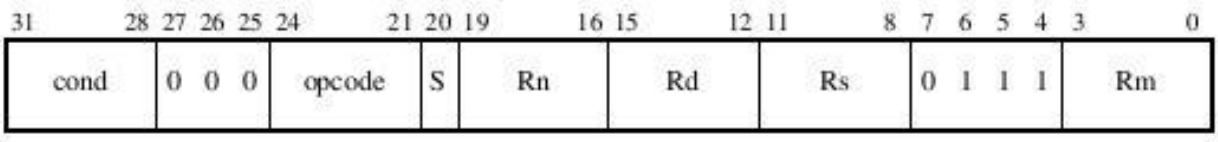


Рисунок 5 – Циклічний зсув вправо





`

<Rm>, ROR #<shift_imm>

<Rm>, ROR <Rs>

<*Rm*> вказує регістр, чиє значення буде зсунуто

<*shift_imm* > вказує зсув (1..32, при значенні 0: RRX.)

<*Rs*> регістр, що містить значення зсуву

MOV R2, R2, ROR #5 ; R2 = R2 ROR #5

MOV R2, R2, ROR R1 ;R2=R2RORR1

2.7 Циклічний зсув вправо (розширений)

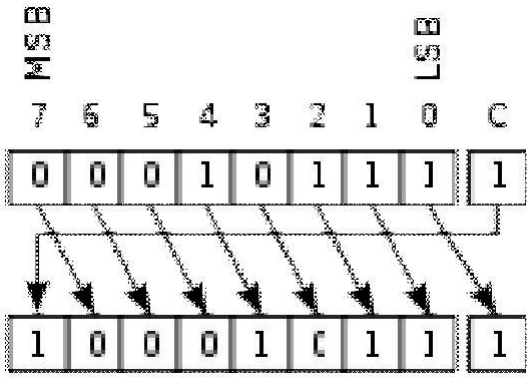
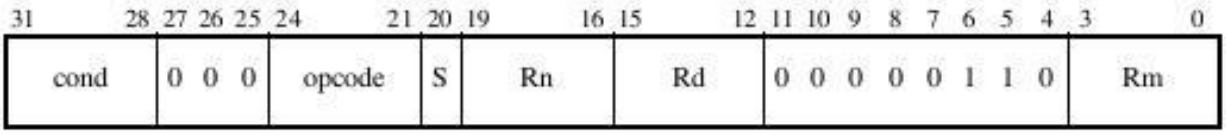


Рисунок 6 – Циклічний зсув вправо (розширений)



<Rm>, RRX

<*Rm*> вказує регістр, чиє значення буде зсунуто

MOVS R2,R2,RRX ; змінює папорці CPSR

MOV R2,R2,RRX ; не змінює прапорці CPSR

`

Приклади команд зі зсувом:

ADD R9, R4, R5, LSL #3 ;

E0849185 = 1110 0000 1000 0100 1001 0001 1000 0101

ADD R9, R3, R5, LSR R1;

E0839135 = 1110 0000 1000 0011 1001 0001 0011 0101

ADD R9, R2, R5, ASR #3;

E08291C5 = 1110 0000 1000 0010 1001 0001 1100 0101

MOV R2, R2, ROR R1

E1A02172 = 1110 0001 1010 0000 0010 0001 0111 0010

MOVS R2,R2, RRX

E1B02062 = 1110 0001 1011 0000 0010 0000 0110 0010

3 Формати команд завантаження/збереження

Команди завантаження (load) та зберігання (store) мають декілька форматів.

3.1 Команди завантаження/ збереження слова або байта без знаку

На рисунку 7 наведено формати команд завантаження та зберігання слова або байта без знаку.

Нижче наведено пояснення окремих полів формату команд, наведених на рисунку 7:

–cond (біти 31…28) формує компілятор в залежності від префіксу команди (таблиця 1), якщо останній використовується;

–27 p = 0; 26 p = 1;

–# (25 p) залежить від того, де знаходиться зміщення (offset_12), яке може використовуватись командою: якщо 25 p = 0, то в якості зміщення використовується 12–розрядне число; якщо 25 p = 1, то зміщення знаходиться в одному з регістрів основного регістрового файлу – Rm;

`

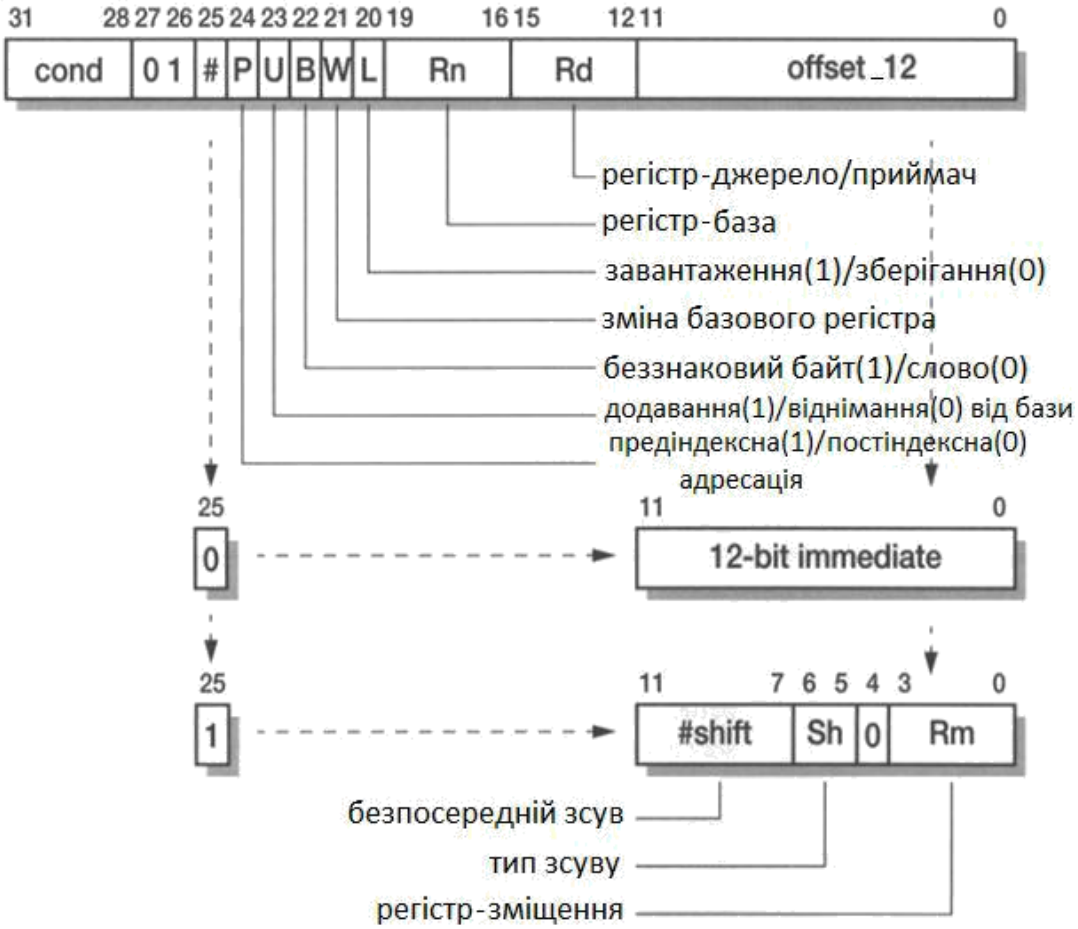


Рисунок 7 – Формат команд завантаження/збереження слова або байта без знаку

–24 р (P) залежить від виду адресації, яка використовується: преіндексна (P = 1), або постіндексна (P = 0);

–23 р (U) визначає, яка операція виконується над базою: додавання (U = 1), або віднімання (U = 0);

–22 р (B) визначає вид операнда: байт без знаку (B=1), або слово (B=0);

–21 р (W) приймає значення в залежності від значення 24 розряду (P): P=0, W=0, якщо виконуються команди LDR, LDRB, STR або STRB та здійснюється нормальний доступ до пам’яті; P=0, W=1, якщо виконуються інструкції LDRBT, LDRT, STRBT, STRT та здійснюється непривіле-

`

йований доступ до пам’яті (режим користувача); P=1, W=0, якщо базовий регістр не оновлюється (адресація зі зміщенням); P=1, W=1, якщо розрахована адреса пам’яті записується назад у базовий регістр (преіндексна адресація);

–20 р (L) визначає вид операції: завантаження (L=1), або зберігання (L=0);

–19 р…16 р (Rn) визначає один з регістрів основного регістрового файлу, який виступає в якості бази;

–15 р … 12 р (Rd) визначає один із регістрів основного регістрового файлу в якості джерела (при збереженні) або приймача (при завантаженні);

–11 р … 0 р (offset_12) визначає, де знаходиться зміщення: якщо 25р (#)=0, то в якості зміщення використовується 12–розрядне число; якщо 25р (#)=1, то в якості зміщення використовується один із регістрів основного регістрового файлу – R_m. Регістр R_m називають індексом, значення R_m також може зсувається на величину, яка визначається полем #shift. Вид зсуву визначається бітами 6, 5 (Sh): Sh=00 –логічний (арифметичний) вліво; 01 – логічний вправо; 10 – арифметичний вправо; 11 – циклічний вправо; 4 р = 0; 3 р … 0 р (R_m) визначає один із регістрів основного регістрового файлу – індекс.

На рисунку 4.8 наведено приклади трьох форматів команд, які відповідають рисунку 7.

У наведених нижче прикладах при виконанні деяких команд змінюється вміст регістра, який називають базою. Таку зміну називають індексуванням. Значення, на яке змінюється база, називають зміщенням. Останнє може зберігатися у регістрі (регістрове зміщення), або задаватися безпосереднім значенням (безпосереднє зміщення). Регістрове зміщення може масштабуватися (зсуватися вліво або вправо на відповідну величину). Адресацію, коли база змінюється до виконання команди, називають преіндексною, а якщо база

`

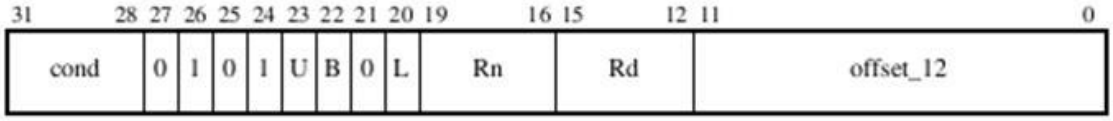
змінюється після виконання команди, називають постіндексною. Преіндексна адресація буває двох типів: без зберігання зміненої бази та зі зберіганням змінної бази.

Рисунок 8 – Види форматів команд завантаження/збереження слова, або байта без знаку

При преіндексній адресації без зберігання зміненої бази до виконання команди до бази додається зміщення, потім виконується команда, але після цього у базі зберігається значення, яке було до виконання команди. При преіндексній адресації зі зберіганням змінної бази до виконання команди до бази додається зміщення, потім виконується команда, після цього у базі зберігається нове значення. При постіндексній адресації база змінюється після виконання команди.

Нижче наведено приклади деяких команд розглянутого формату.

3.1.1 Безпосереднє зміщення (зсув) з преіндексною адресацією без зміни бази



Rn – база;

offset_12 – 12–ти розрядне зміщення;

`

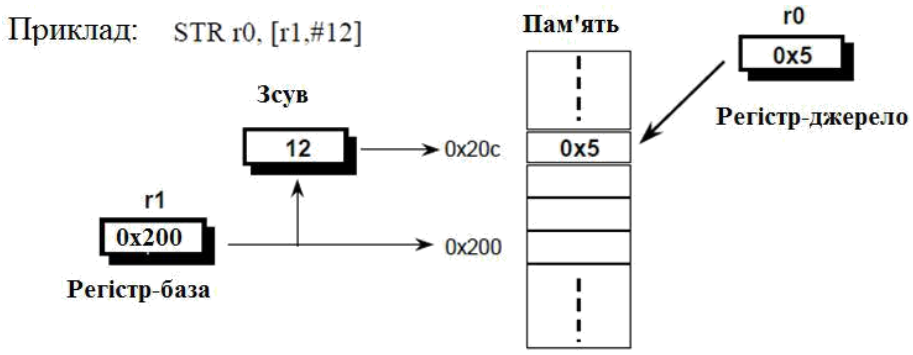


Рисунок 9 – Безпосереднє зміщення (зсув) з преіндексною адресацією без зміни бази

E581000C = 1110 0101 1000 0001 0000 0000 0000 1100

3.1.2 Безпосереднє зміщення (зсув) з преіндексною адресацією зі зміною бази

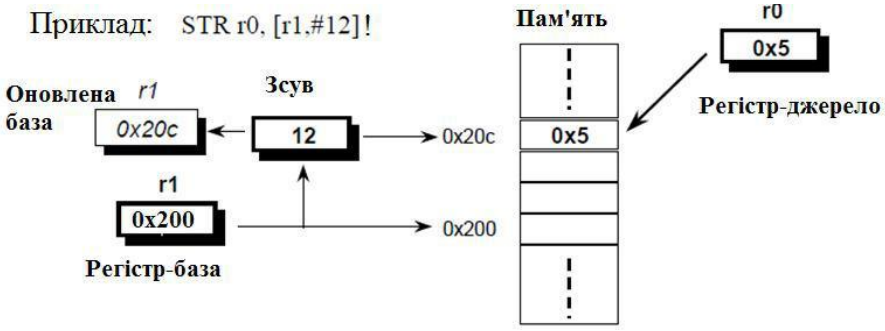
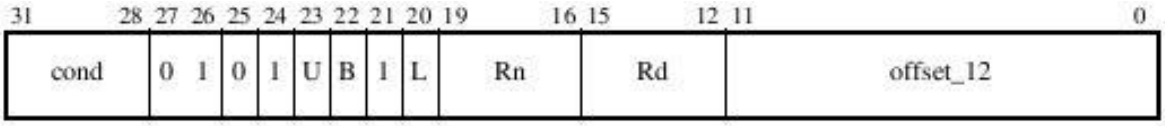


Рисунок 10 – Безпосередня преіндексна



E5A1000C = 1110 0101 1010 0001 0000 0000 0000 1100

`

3.1.3 Безпосередня постіндексна адресація

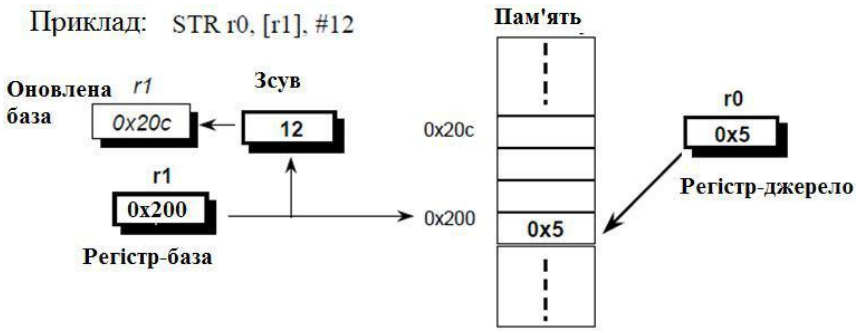
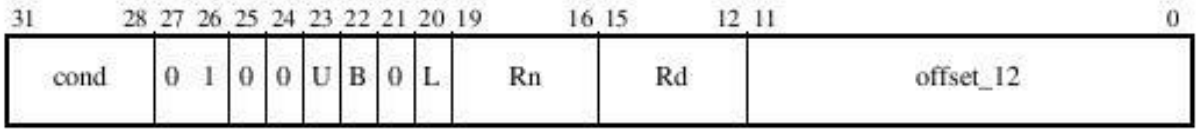


Рисунок 11 – Безпосередня постіндексна



E481000C = 1110 0100 1000 0001 0000 0000 0000 1100

3.1.4 Регістрове зміщення (зсув)з преіндексною адресацією без зміни бази

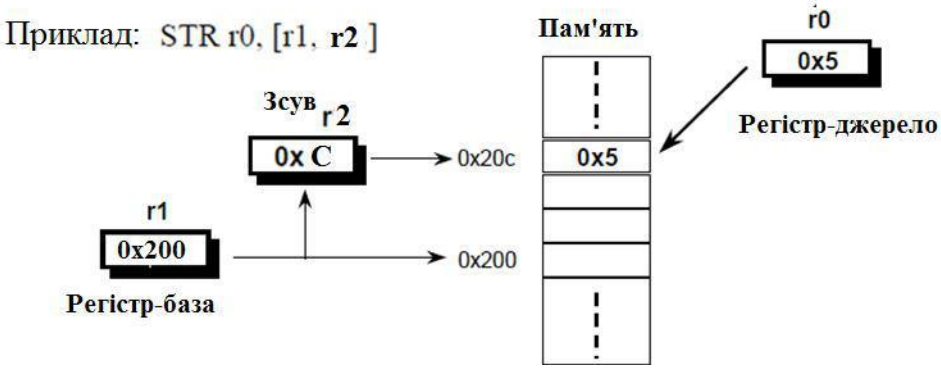
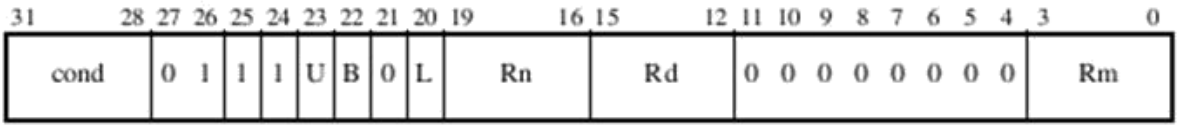


Рисунок 12 – Регістрове зміщення (зсув) з преіндексною адресацією без зміни бази

STR R0, [R1,R2]

`



3.1.5 Регістрове зміщення (зсув) з преіндексною адресацією та зміною бази

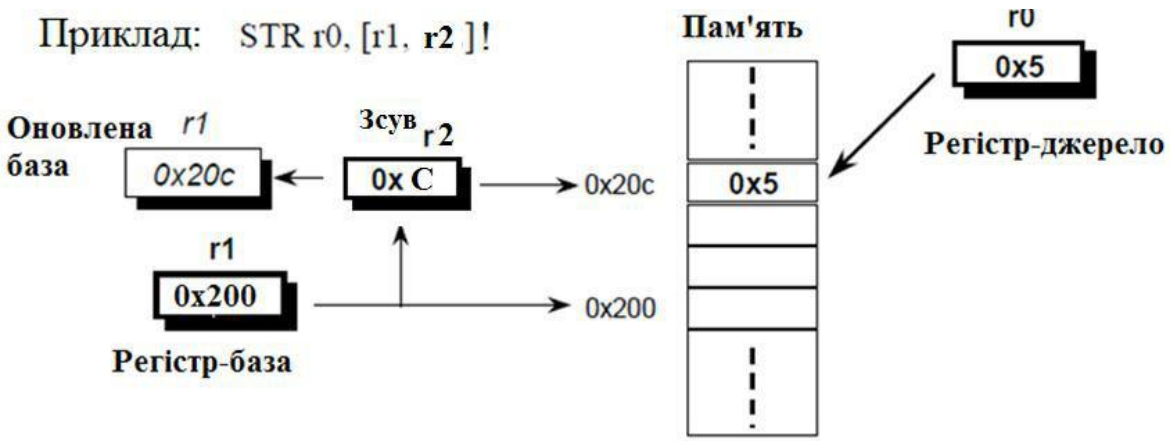
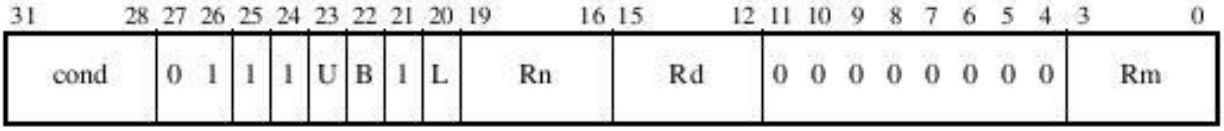


Рисунок 13 – Преіндексна регістрова адресація

STR R0, [R1,R2]!



3.1.6 Регістрове зміщення (зсув) з постіндексною адресацією

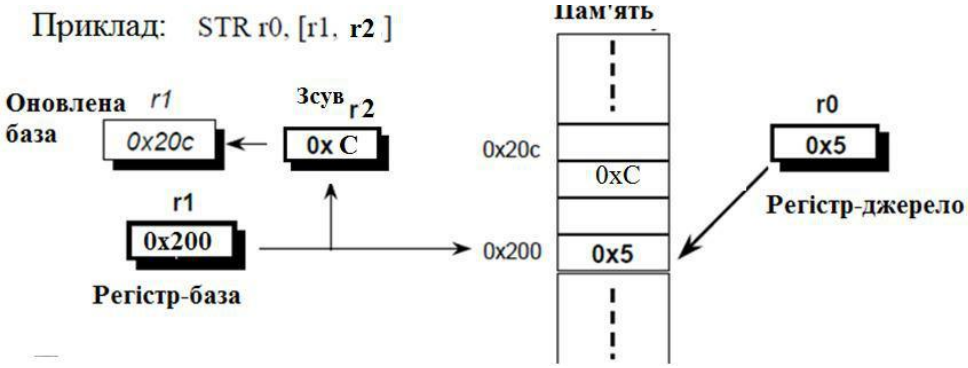
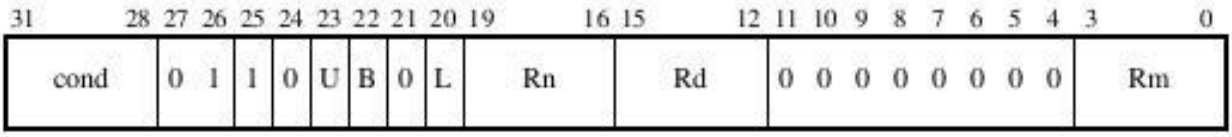


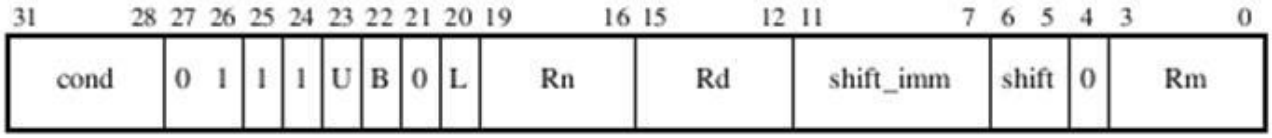
Рисунок 14 – Регістрове зміщення (зсув) з постіндексною адресацією

`

STR R0, [R1], R2



3.1.7 Регістрове зміщення із зсувом та прединдексною адресаці-єю з масштабованим регістровим зміщенням без оновлення бази



Варіанти адресації:

[<Rn>, +/–<Rm>, LSL #<shift_imm>];

[<Rn>, +/–<Rm>, LSR #<shift_imm>];

[<Rn>, +/–<Rm>, ASR #<shift_imm>];

[<Rn>, +/–<Rm>, ROR #<shift_imm>];

[<Rn>, +/–<Rm>, RRX];

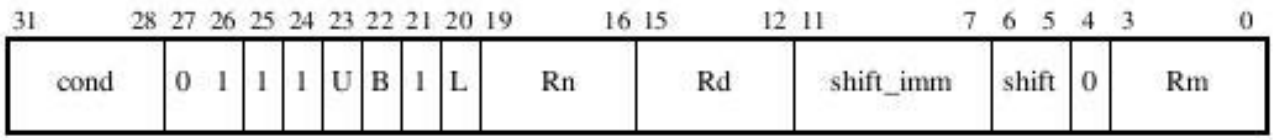
<Rn> – база;

<Rm> – індекс;

<shift_imm> – зсув;

STR R0, [R1, R2, LSL #5]; M(R1 + R2x32) = R0,

3.1.8 Регістрове зміщення зі зсувом та преіндексною адресацією з масштабованим регістровим зміщенням та оновленням бази

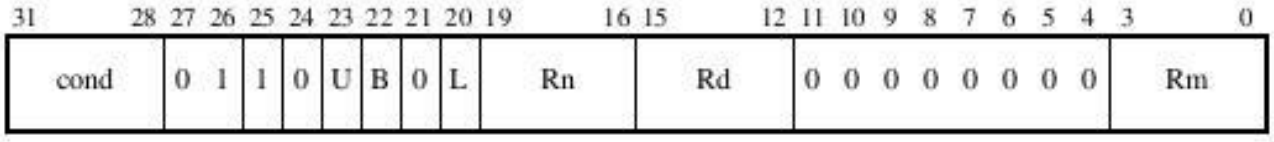


LDR R0, [R1, R2, LSL #2]! ; R0 = M(R1 + R2x4); R1 = R1 + R2x4.

`

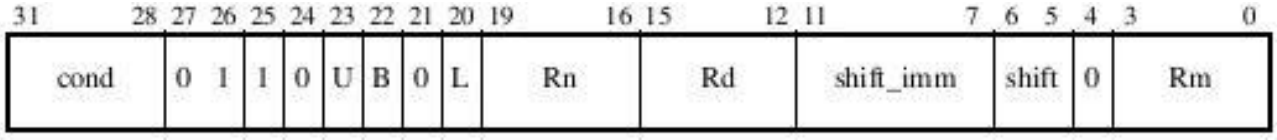
Може використовуватись, наприклад, для доступу до елементів масиву із розміром значення більшим ніж один елемент з попереднім оновленням покажчика елемента.

3.1.9 Регістрове зміщення з постіндексною адресацією



LDR R1, [R2], R3; R1 = M(R2) ; R2= R2 + R3.

3.1.10 Регістрове зміщення зі зсувом та постіндексною адресацією



LDR R0, [R1], R2, LSL #2 ; R0 = M(R1); R1 =R1+ R2x4.

Може використовуватись, наприклад, для доступу до елементів масиву із розміром більшим ніж один елемент з наступним оновленням покажчика елемента.

3.2 Команди завантаження/збереження половини слова, або подвійного слова та завантаження байта зі знаком

На рисунку 15 наведено формат команд завантаження/збереження половини слова, або подвійного слова та завантаження байта зі знаком.

`

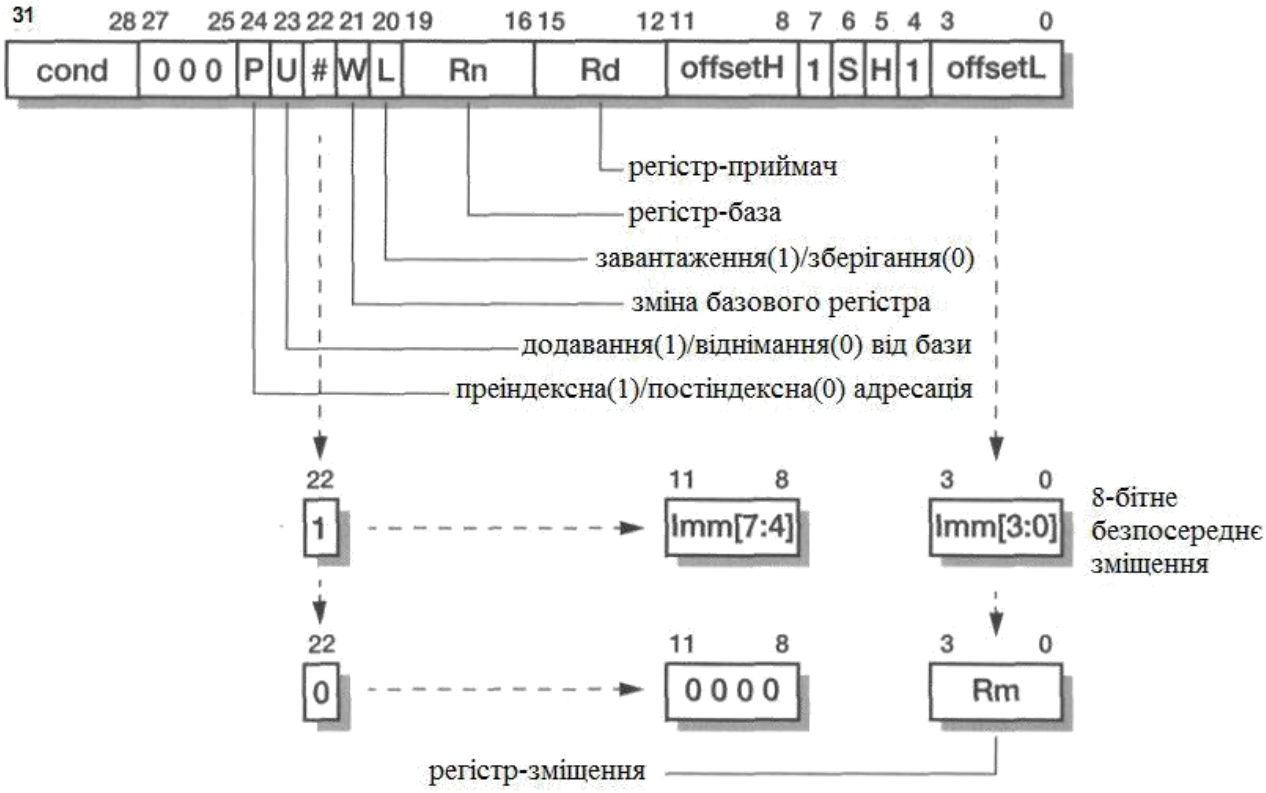


Рисунок 15 – Завантаження/збереження половини слова, або подвійного слова та завантаження байта зі знаком

Пояснення більшості бітів співпадає з рисунком 7. Нижче наведено тільки відмінності:

– 22р (#) визначає, де знаходиться зміщення відносно бази при формуванні адреси пам’яті: якщо 22р=1, то в якості зміщення використовується 8–розрядне число (розряди 11 … 8 (imm[7:4]) та 3 … 0(imm[3:0]); якщо 22р=0, то зміщення знаходиться в одному з регістрів основного регістрового файлу ;

– 21р (W) приймає значення в залежності від значення 24–ого розряду (P): P=0, W=1 при преіндексній адресації (базовий регістр оновлюється); W=0 при адресації зі зміщенням (базовий регістр не оновлюється);

– 6р (S) та 7р (H) змінюється в залежності від виконуваної команди (таблиця.2).

`

Таблиця 2 – Зміни бітів L,S та H в залежності від виконуваної команди

L	S	H	Результат
0	0	1	Зберегти половину слова
0	1	0	Завантажити подвійне слово
0	1	1	Зберегти подвійне слово
1	0	1	Завантажити половину слова без знаку
1	1	0	Завантажити байт зі знаком
1	1	1	Завантажити половину слова зі знаком

LDR|STR{<cond>}H|SH|SB|D <Rd>, <addressing_mode>

SH – Signed Halfword – половина слова зі знаком (лише LDR)

H – unsigned Halfword – половина слова без знаку

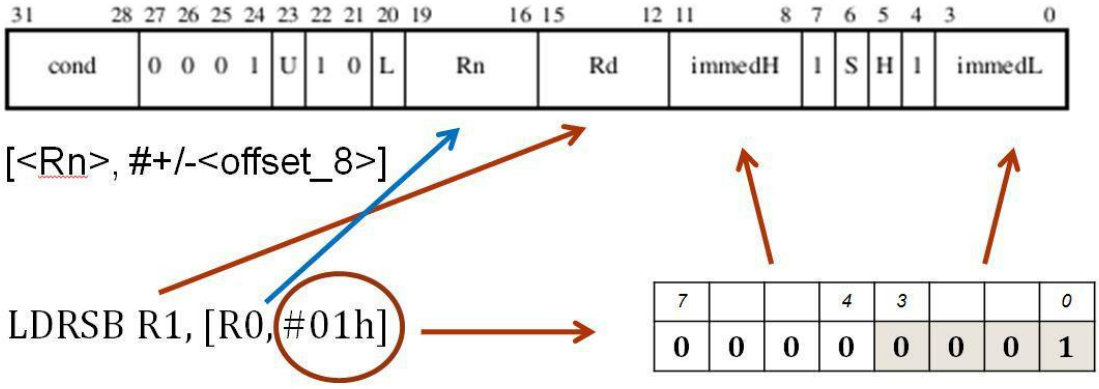
SB – Signed Byte – байт зі знаком (лише LDR)

D – Doubleword – подвійне слово.

Нижче наведено приклади деяких команд розглянутого формату.

`

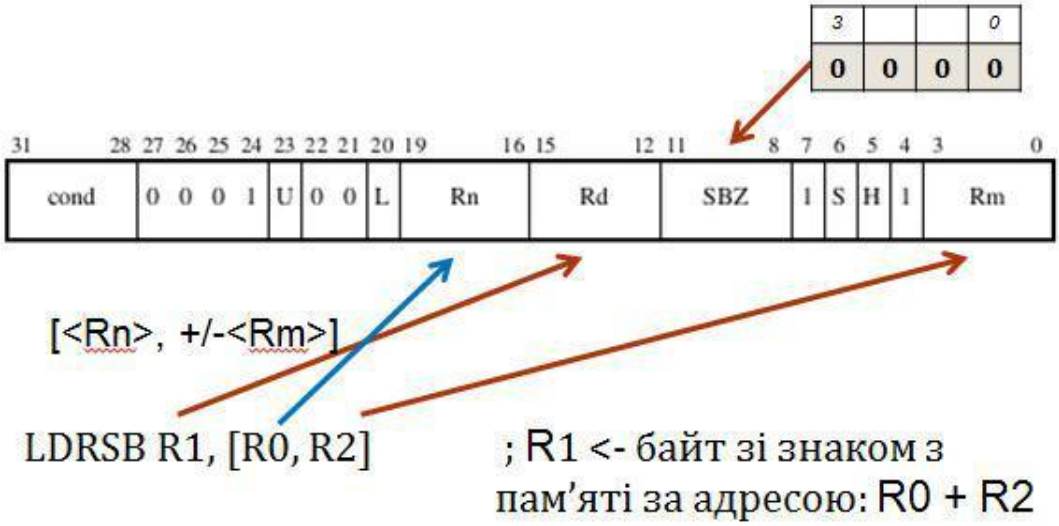
3.2.1 Преіндексна адресація з безпосереднім зміщенням без оновлення бази



; R1 <- байт зі знаком з пам'яті за адресою: R0 + 01h

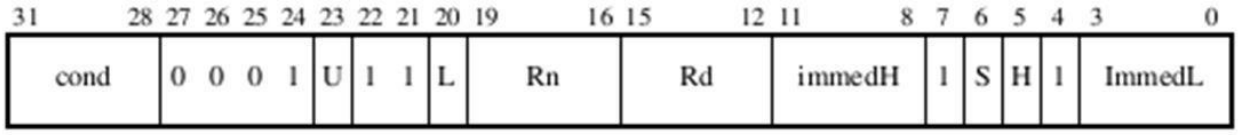
E1D010D1 = 1110 0001 1101 0000 0001 0000 1101 0001

3.2.2 Преіндексна адресація з регістровим зміщенням без оновлення бази



`

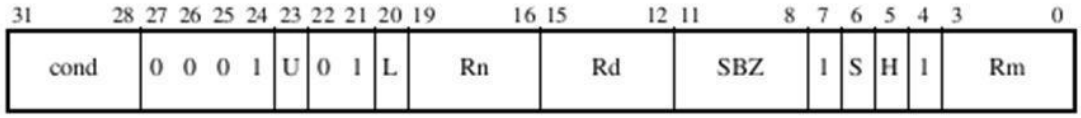
3.2.3 Преіндексна адресація з безпосереднім зміщенням та оновленням бази




```
LDRSB R0, [R1, #4]!        ;  R1 <- R1 + 4,
                           ; R0 <- байт зі знаком з
                           ; пам'яті за адресою R1 + 4
```

E1F100D4 = 1110 0001 1111 0001 0000 0000 1101 0100

3.2.4 Преіндексна адресація з регістровим зміщенням та оновленням бази

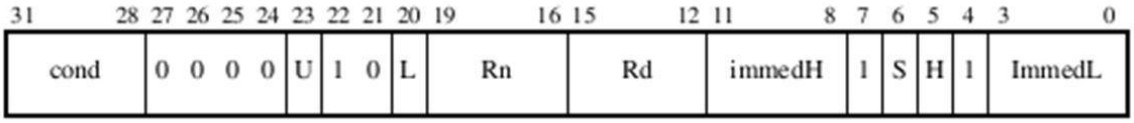



```
LDRSB R0, [R1, R2]!        ; R1 <- R1 + R2
                           ; R0 <- байт зі знаком з
                          ;пам'яті з адресою R1 + R2;
```

E1B100D2 = 1110 0001 1011 0001 0000 0000 1101 0010

`

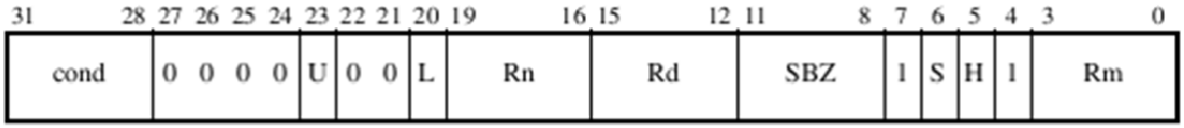
3.2.5 Постіндексна адресація з безпосереднім зміщенням




```
LDRSB R0, [R1], #2          ; R0 <- байт зі знаком з
                            ;пам'яті з адресою в R1
                            ; R1 <- R1 + 2
```

E0D100D2 = 1110 0000 1101 0001 0000 0000 1101 0010

3.2.6 Постіндексна адресація з регістровим зміщенням




```
LDRSB R0, [R1], R2          ; R0 <- байт зі знаком з
                            ;пам'яті з адресою в R1
                            ; R1 <- R1 + R2
```

E09100D2 = 1110 0000 1001 0001 0000 0000 1101 0010