문제 data처리 명령어의 3가지 유형

다음 3가지 명령의 기계어를 작성하고 기능을 설명하라.

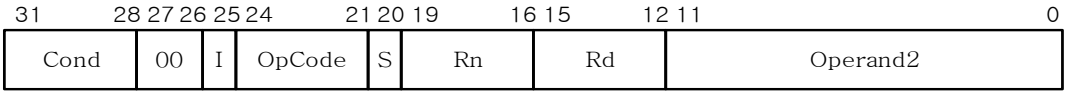
(단, r3에는 mov r3, #3, 명령으로 3 이 저장되어 있고, lsl은 logical shift left명령어이다.)

① add r0, r1, r2, lsl r3

② add r0, r1, r2, lsl #16

③ add r0, r2, #0x9000 0000

<풀이> data처리 명령의 format은 아래와 같다. 상세한 것은 다음 페이지(A)에 있음



cond : condition code summary의 맨 마지막 suffix가 AL, flags무시, alaway에 해당하므로 1110 이 된다

opcode : ARM data 처리 명령표 페이지(B)의 opcode항을 보면 add는 0100 이 된다.

I : 그림 (A)를 보면 문제의 1,2의 경우는 그 값이 0 이고 3의 경우는 1 이 된다 (Immediate난이다)

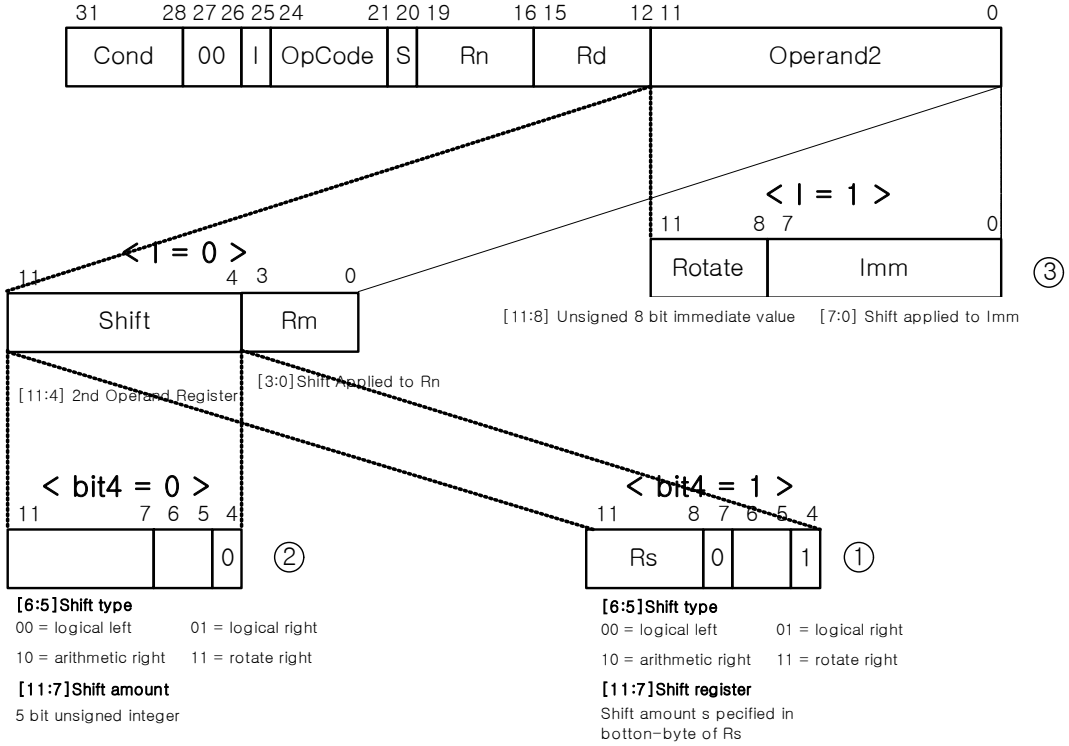
Rn, Rd : ①, ②, ③ 에 따라 상응하는 레지스터를 보여준다.

S : set condition set 신호로 명령 수행결과 CPSR에 설정 여부를 결정한다.

ARM condition codes.

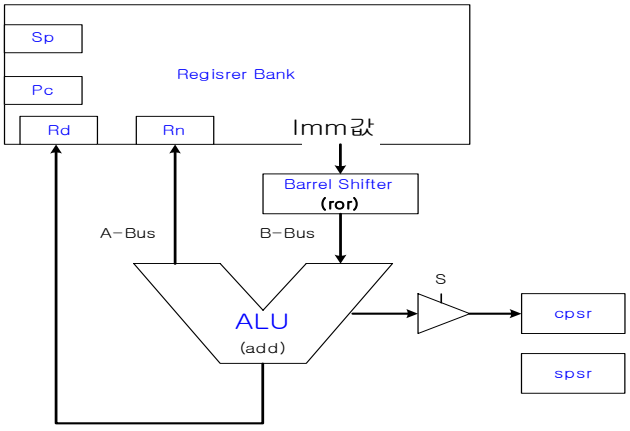
Opcode [31 : 28]	Memonic extension	Interpretation	Status flag for execution
0000	EQ	Equal / equal zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned high or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z clear
1010	GE	Signed greater than or equal	N equal to V
1011	LT	Signed less than	N is not equals V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equa	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never(do not use!)	none

그림(A) Intruction Set- Data Processing inst.



참고 Data처리 명령의 세가지 종류

Data처리 명령은 그림 (A)에서와 같이 크게 다음 3가지로 분류 할 수 있다.



Data처리시 B-bus에 들어가는 데이터를 구성하는 형식에 따라 다음과 같이 3가지로 분류한다.

① 둘(rm, rs)다 레지스터를 참조하여 B.S응 작동시켜 만든다

보기 : Add r0, r1, r2, ,lsl r3

② 하나(rm)은 레지스터를 참조하고 다른 하나는 즉치를 써서 B.S를 작동시킨다.

보기 : Add r0, r1, r2, lsl #16

③ 둘(rm, rs)다 즉치를 쓴다

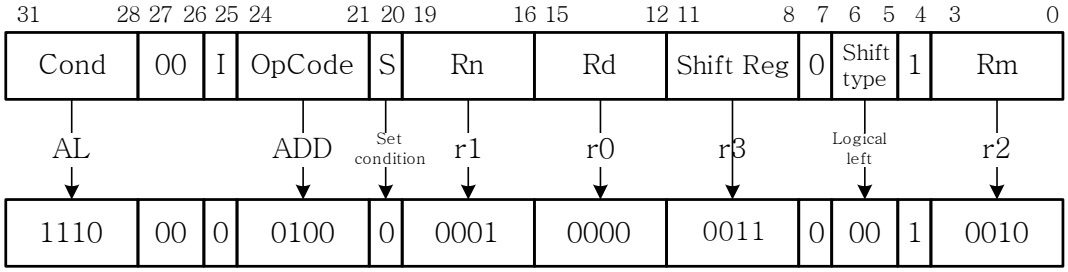
보기 : Add r0, r2, 0x9000 0000

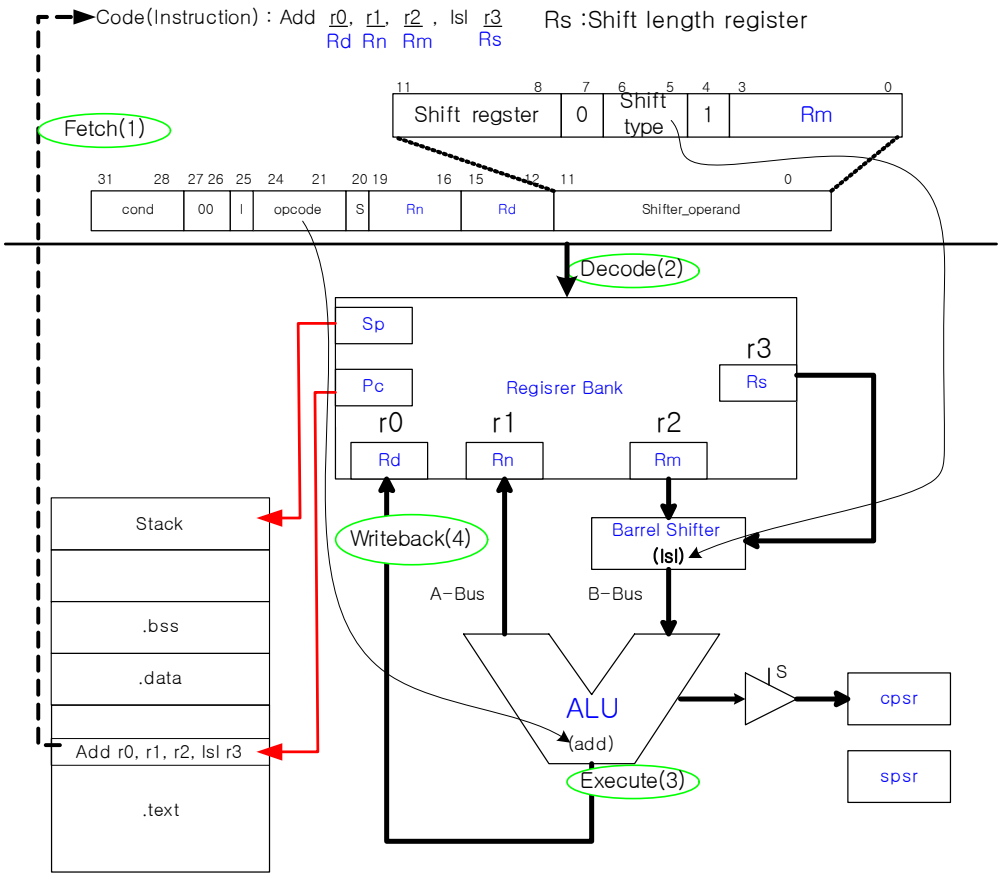
표(B) ARM data processing instructions.

Opcode [24 : 21]	Memonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalance	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := Not Op2

①의 경우

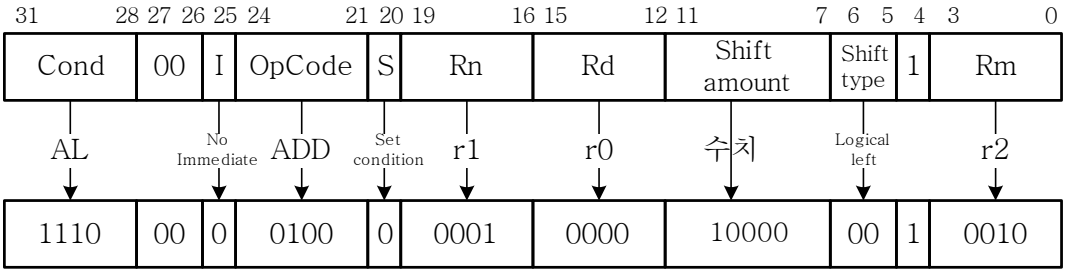
Add r0, r1, r2, lsl r3

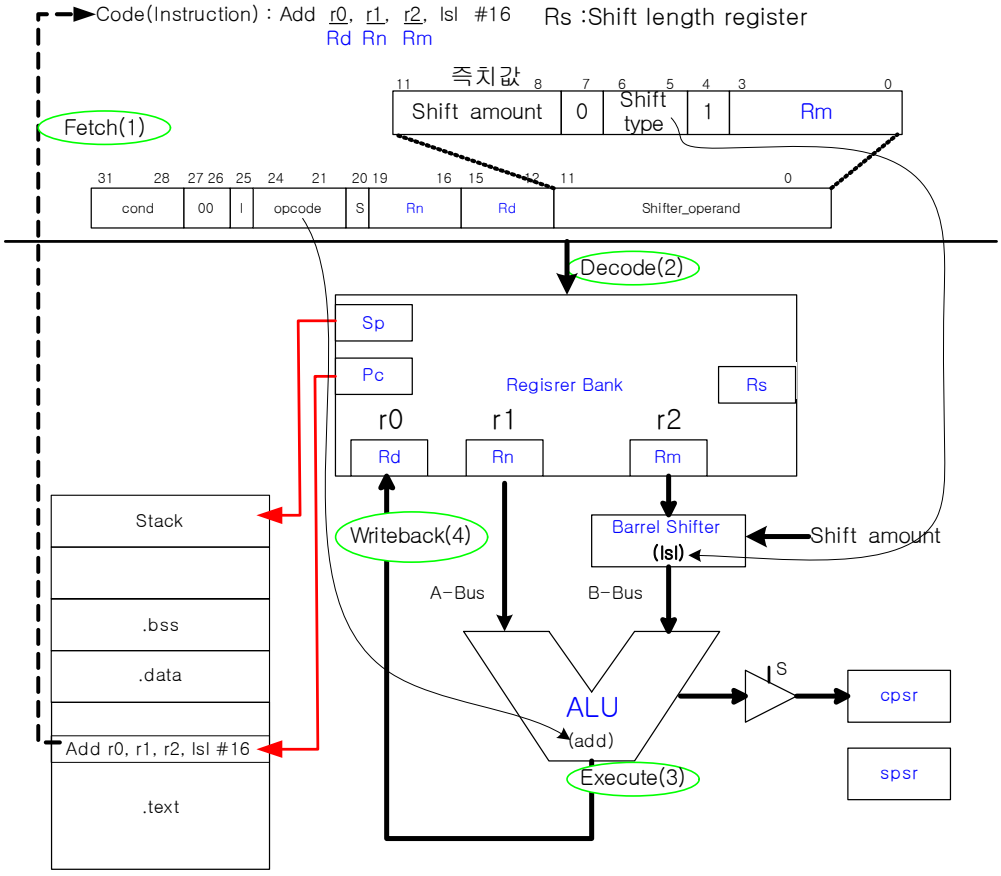




② 의 경우

Add r0, r1, r2, lsl #16

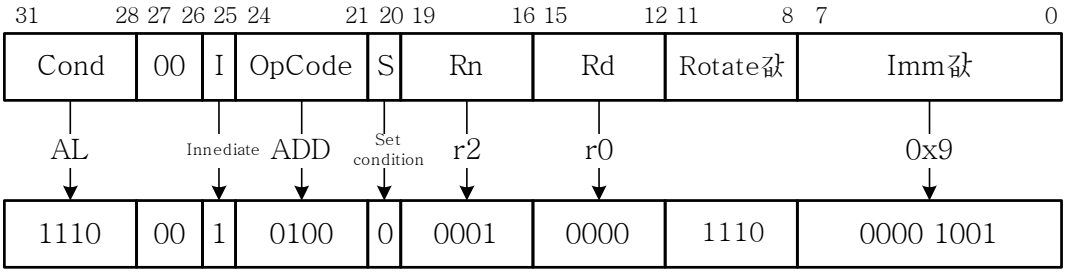




③의 경우

Add r0, r2, #0x9000 0000

B-bus의 값 생성시 둘다 즉치로 하는 경우

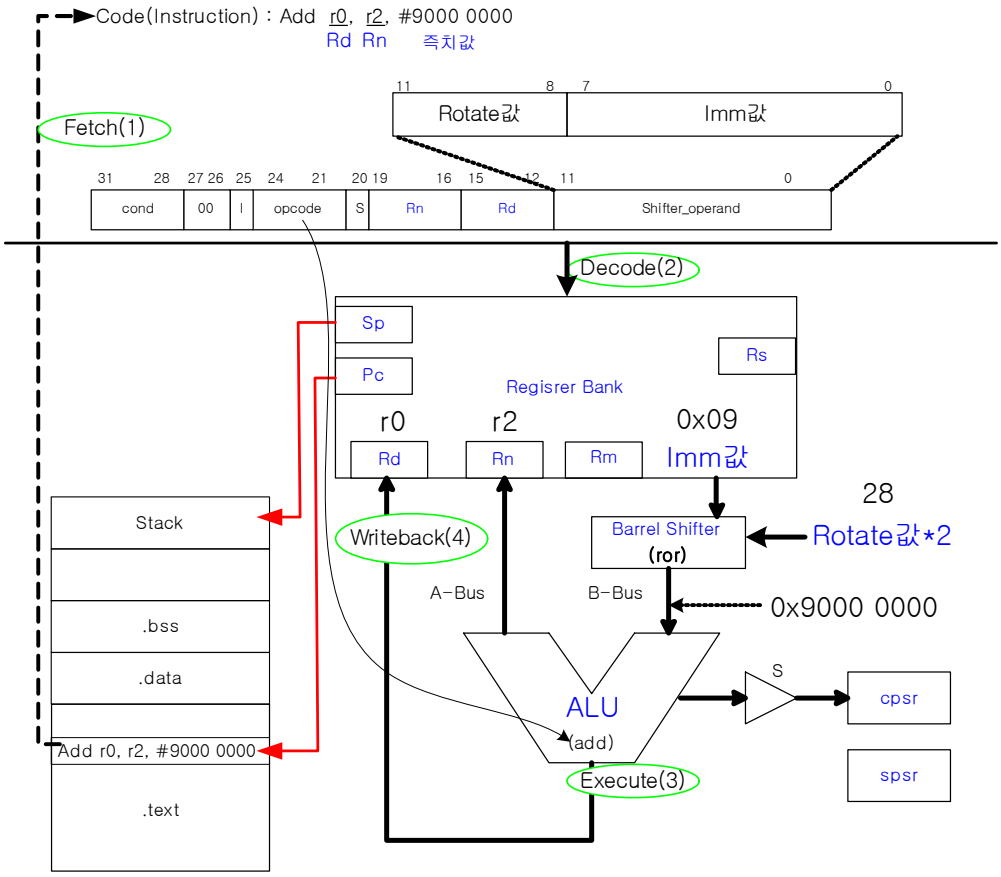


즉치값 0x9000 0000 (0b1001 0000 0000 0000 0000 0000 0000 0000)

0x9라는 값은 1001 이다 이것의 8bit 형태로 표시하면 0000 1001 이다. 그래서 Imm값은 0000 1001 이 된다. 이 0000 1001 을 28번 오른쪽으로 rotate해야 한다.

그런데, ③의 명령중 어떤 Rotate 값을 지정해 주게 되면 B-bus에 만들어 지는 오퍼랜드값은 Imm값을 Rotate값 * 2 만큼 오른쪽으로 회전한 결과값이 된다.

<ARM Archi.Referencr Manual 5.2.3참조>



따라서 Rotate값은 28/2가 된다. 즉 1110 == 14가 된다/

결론 각각의 기계어는 아래와 같다.

① Add r0, r1, r2, lsl r3

1110 0000 1000 0001 0000 0011 0001 0010

② Add r0, r1, r2, lsl #16

1110 0000 1000 0001 0000 1000 0000 0010

③ Add r0, r2, #0x9000 0000

1110 0010 1000 0010 0000 1110 0000 1001

명령어 각각의 기능 해설

① Add r0, r1, r2, lsl r3 -> r0 = r1 + r2 ×

×

ARM instruction에서 메모리 번지 지정방식에는 다음 두가지 방식이 있다.

1. pre-indexing addressing mode

   str r3, [r0, #4]……base register 불변

   str r3, [r0], #4……base register 갱신

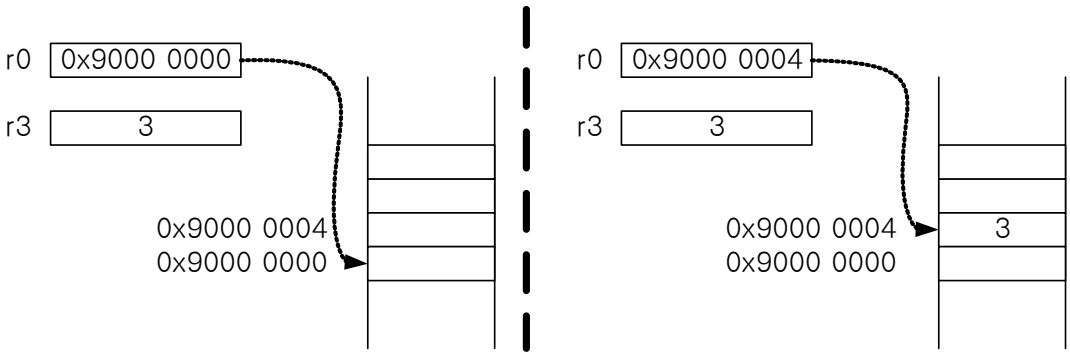
   메모리 주소 = base address + offset (r0 + 4)

2. post-indexing address mode

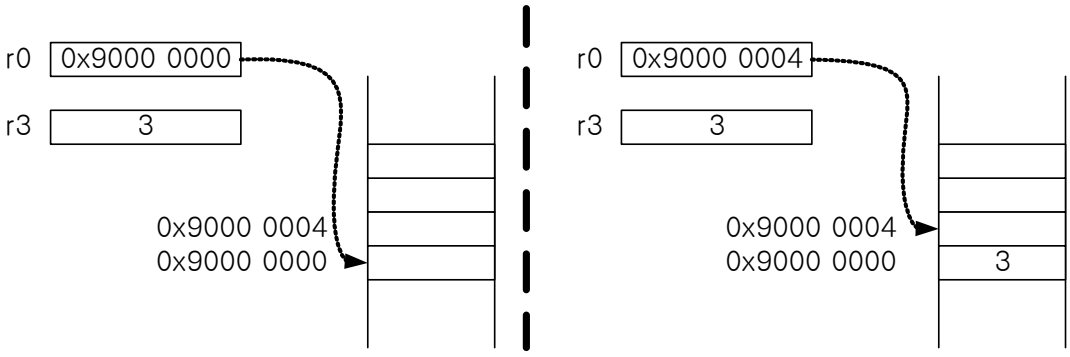
   str r3, [r0], #4

   메모리 주소 = base register값

   저장(store)후 base register값 = base register값 + offset

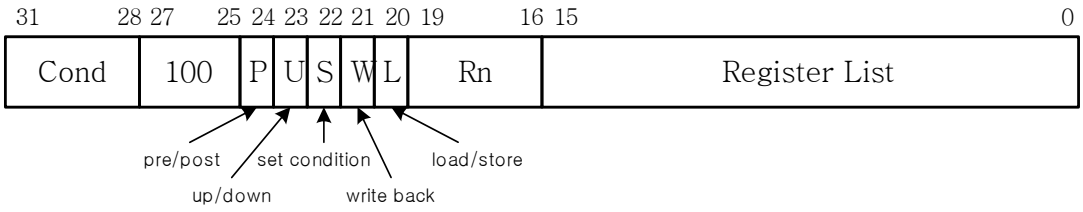


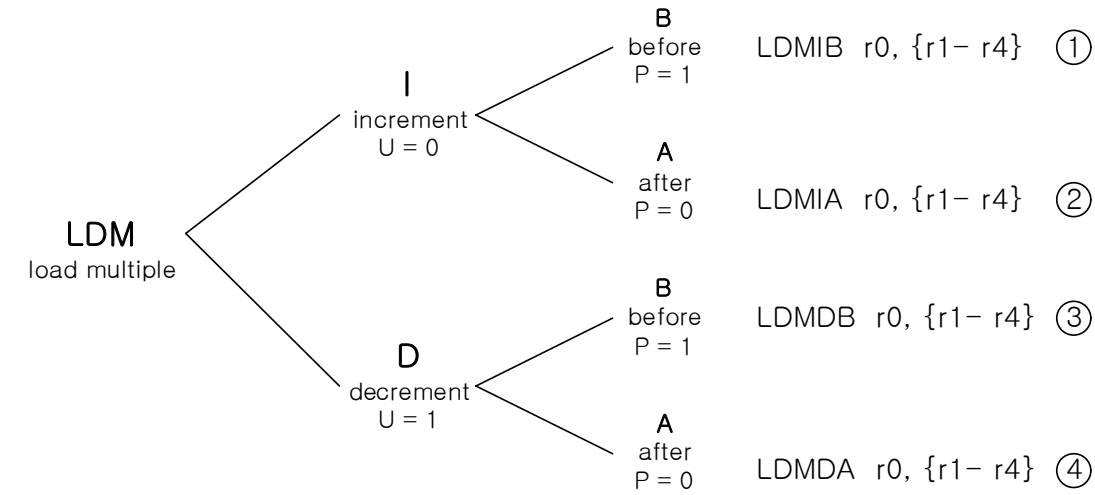
str r3, [r0, #4]! 실행 전후

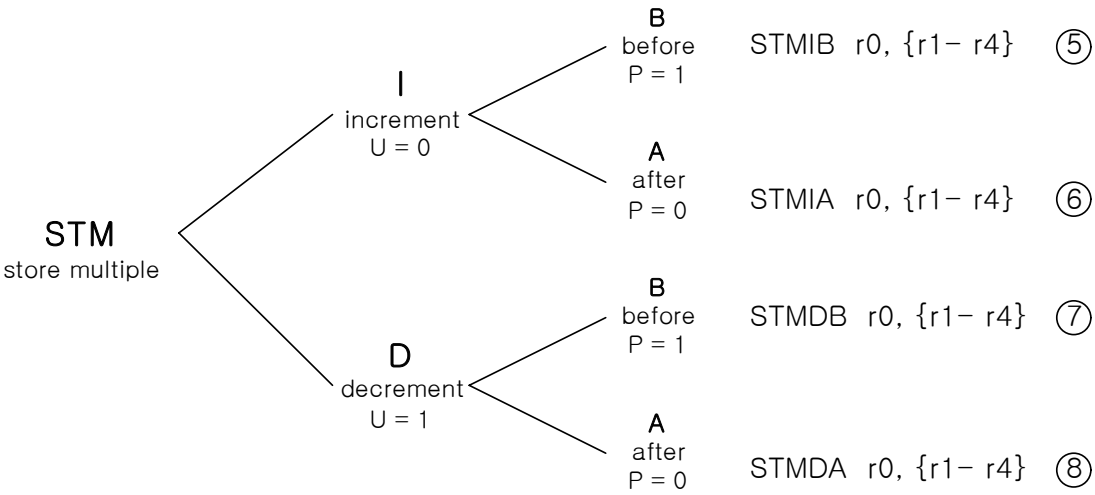


str r3, [r0], #4 실행 전후

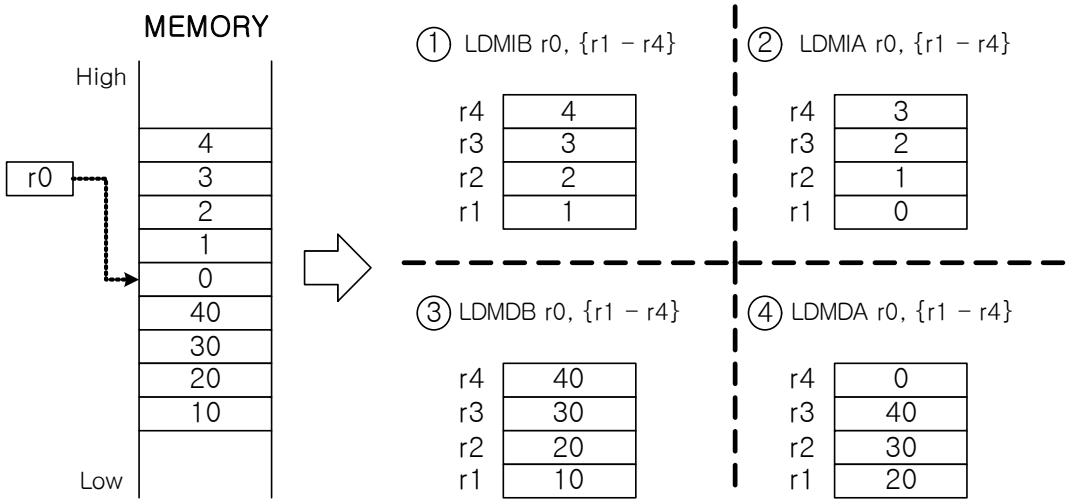
Data Transfer Instruction

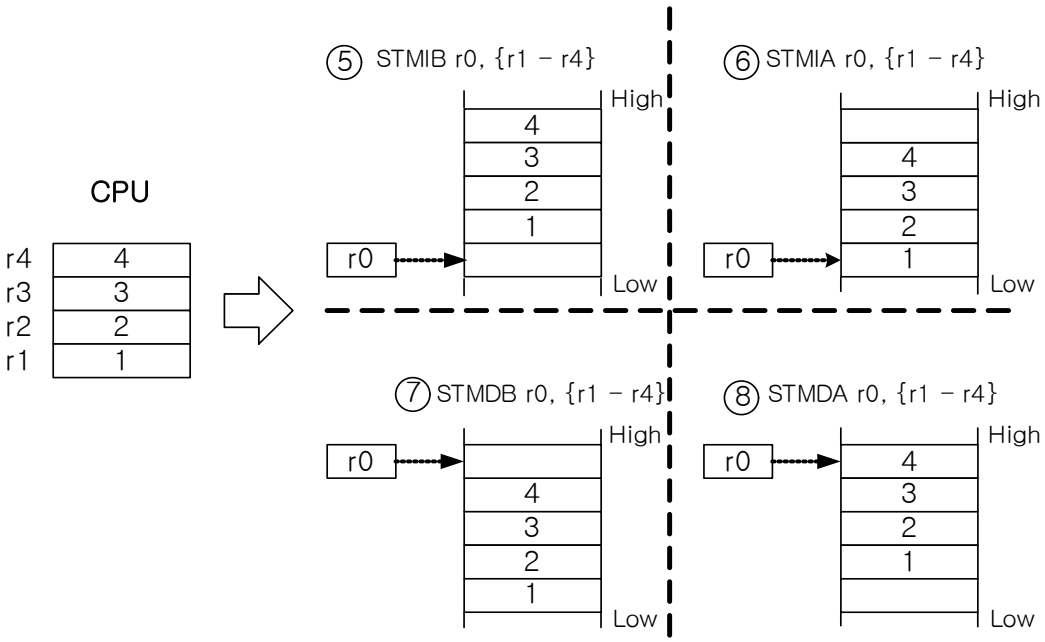




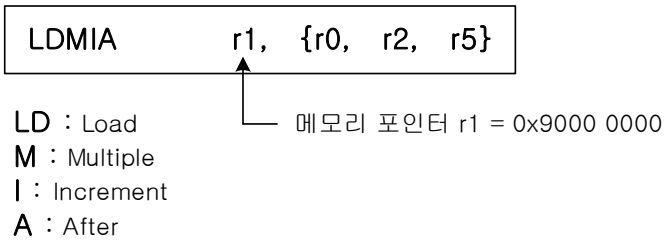


LDM 명령들

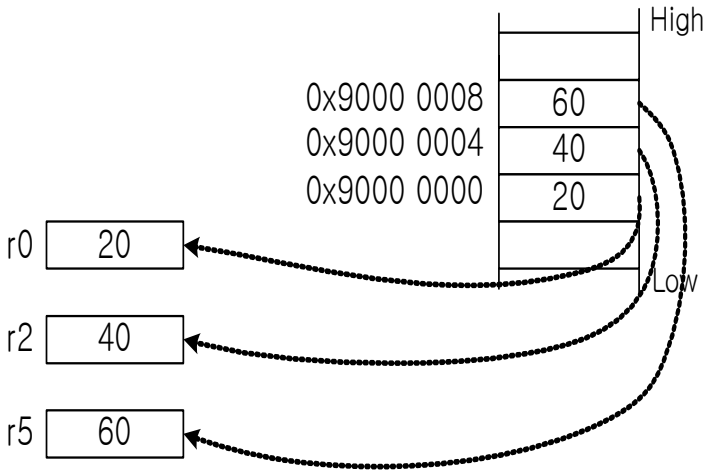

MEMORY
High
Low
r0
4
3
2
1
0
40
30
20
10
① LDMIB r0, {r1 - r4}
r4 4
r3 3
r2 2
r1 1
② LDMIA r0, {r1 - r4}
r4 3
r3 2
r2 1
r1 0
③ LDMDB r0, {r1 - r4}
r4 40
r3 30
r2 20
r1 10
④ LDMDA r0, {r1 - r4}
r4 0
r3 40
r2 30
r1 20


CPU
r4 4
r3 3
r2 2
r1 1
⑤ STMIB r0, {r1 - r4}
⑥ STMIA r0, {r1 - r4}
⑦ STMDB r0, {r1 - r4}
⑧ STMDA r0, {r1 - r4}
r0
High
Low

명령의 보기



$r0 := mem_{32}[r1]$, $r2 := mem_{32}[r1+4]$, $r5 := mem_{32}[r1+8]$



The mapping between the stack and block copy views of the load and store multiple instruction

		Ascending		Descending	
		Full	Empty	Full	Empty
Increment	Before	STMIB STMFA			LDMIB LDMED
	After		STMIA STMEA	LDMIA LDMFD	
Decrement	Before		LDMDB LDMEA	STMDB STMFD	
	After	LDMDA LDMFA			STMDA STMED