제4장 THUMB 명령어

THUMB 명령어 포맷

THUMB 명령어는 ARM 명령어(32비트 포맷)의 16비트 버전이다. ARM 명령어가 16비트로 감소된 THUMB 명령어는 ARM 명령어보다 융통성이 있다. THUMB 명령어는 ARM920T 코어 안의 THUMB decompressor에 의해서 ARM 명령어로 decompress 된다.
THUMB 명령어가 ARM 명령어를 압축할 때, THUMB 명령어는 16비트 포맷의 명령어를 가지며 몇 가지 제한이 생긴다. 16비트 포맷에 의한 제한은 THUMB 명령어를 이용하는데 완전히 통지된다.

포맷에 대한 요약

THUMB 명령어는 그림 4-1에 나타나 있다.

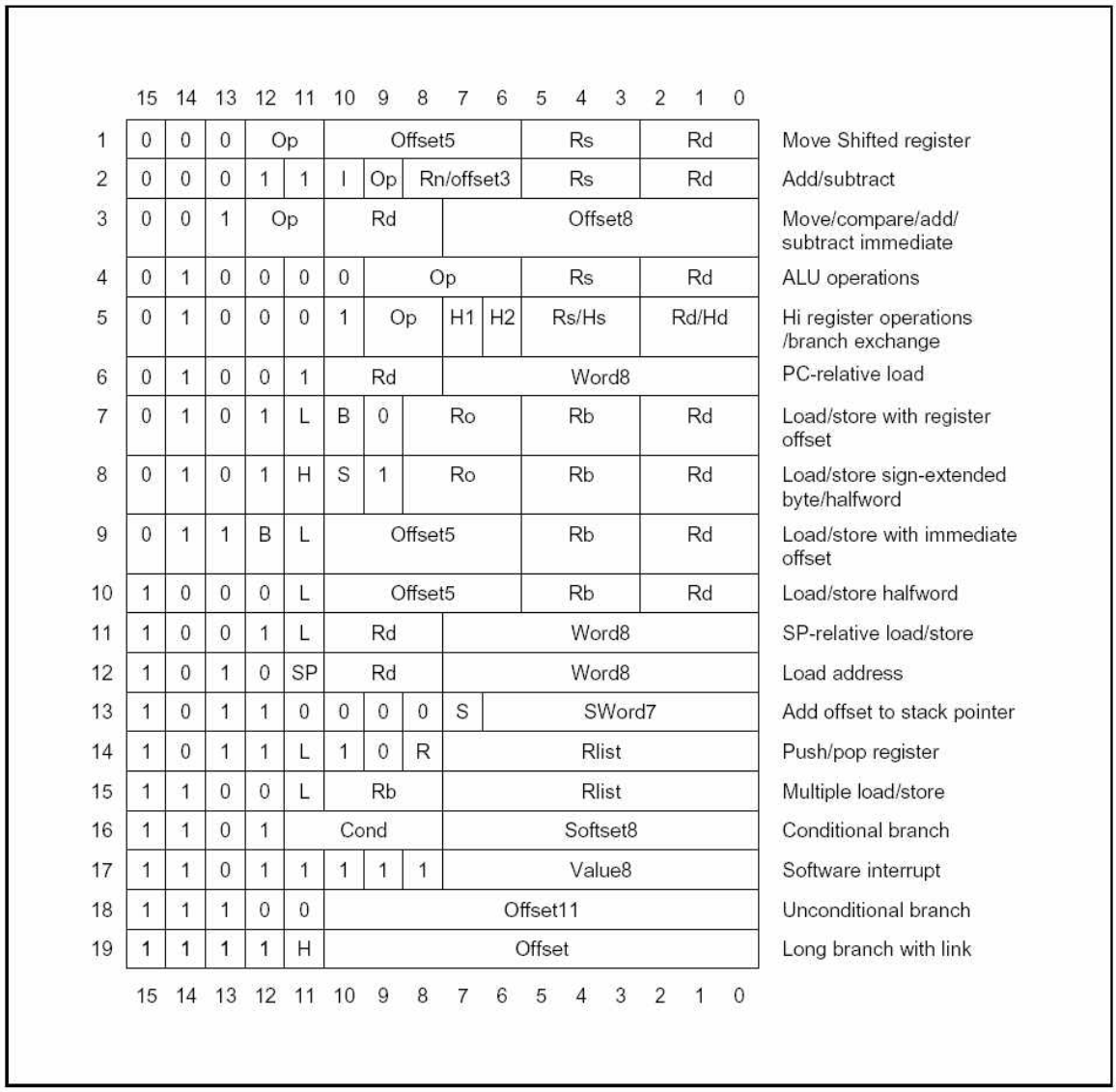

그림 4-1. THUMB 명령어 포맷

OPCODE 요약

표 4-1에서 THUMB 명령어를 요약하였다. 특별한 명령어에 대한 정보는 right-most column에 나타난 부분을 참조하시오.

표 4-1. THUMB 명령어 설정 opcode

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	Y	–	Y
ADD	Add	Y	–	Y (1)
AND	AND	Y	–	Y
ASR	Arithmetic Shift Right	Y	–	Y
B	Unconditional branch	Y	–	–
Bxx	Conditional branch	Y	–	–
BIC	Bit Clear	Y	–	Y
BL	Branch and Link	–	–	–
BX	Branch and Exchange	Y	Y	–
CMN	Compare Negative	Y	–	Y
CMP	Compare	Y	Y	Y
EOR	EOR	Y	–	Y
LDMIA	Load multiple	Y	–	–
LDR	Load word	Y	–	–
LDRB	Load byte	Y	–	–
LDRH	Load halfword	Y	–	–
LSL	Logical Shift Left	Y	–	Y
LDSB	Load sign-extended byte	Y	–	–
LDSH	Load sign-extended halfword	Y	–	–
LSR	Logical Shift Right	Y	–	Y
MOV	Move register	Y	Y	Y (2)
MUL	Multiply	Y	–	Y
MVN	Move Negative register	Y	–	Y

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
NEG	Negate	Y	–	Y
ORR	OR	Y	–	Y
POP	Pop register	Y	–	–
PUSH	Push register	Y	–	–
ROR	Rotate Right	Y	–	Y
SBC	Subtract with Carry	Y	–	Y
STMIA	Store Multiple	Y	–	–
STR	Store word	Y	–	–
STRB	Store byte	Y	–	–
STRH	Store halfword	Y	–	–
SWI	Software Interrupt	–	–	–
SUB	Subtract	Y	–	Y
TST	Test bits	Y	–	Y

NOTES :

1. condition code는 이 명령어의 포맷 5, 12, 13의 영향을 받지 않는다.
2. condition code는 이 명령어의 포맷 5 버전의 영향을 받지 않는다.

포맷1 : Move Shift Register

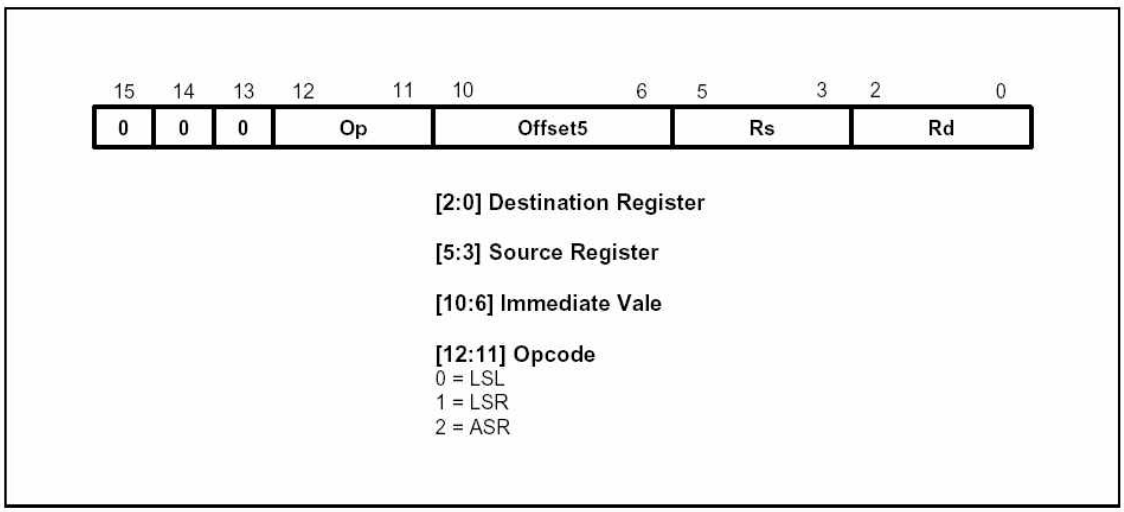

그림 4-2. 포맷 1

동작

이 명령어는 Lo 레지스터 사이의 쉬프트 된 값을 이동한다. THUMB 어셈블러 문법은 표 4-2에 나타나 있다.

NOTE

이 그룹 안의 모든 명령어들은 CPSR condition code를 설정한다.

표 4-2. 포맷 1 명령어에 대한 요약

OP	THUMB Assembler	ARM Equipment	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

명령어 사이클 시간

이 포맷의 모든 명령어들은 표 4-2에 나타난 ARM명령어와 같다. THUMB 명령어에 대한 명령어 사이클 시간은 ARM 명령어의 시간과 같다.

예제

```
LSR     R2, R5, #27     ; Logical shift right the contents
                        ; of R5 by 27 and store the result in R2.
                        ; Set condition codes on the result.
```

포맷 2 : ADD/SUBTRACT

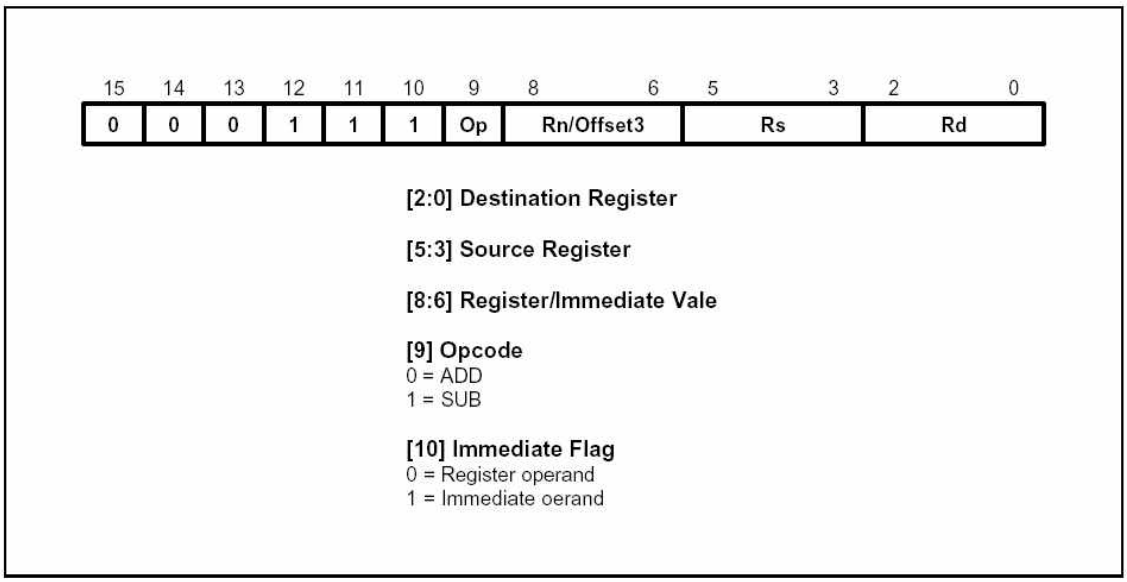

그림 4-3. 포맷 2

동작

이 명령어는 Lo 레지스터의 내용이나 3비트 immediate 값을 Lo 레지스터에 더하거나 뺀다. THUMB 어셈블러 문법이 표 4-3에 나타나 있다.

이 그룹의 모든 명령어들은 CPSR condition code를 선택한다.

표 4-3. 포맷 2 명령어에 대한 요약

OP	I	THUMB Assembler	ARM Equipment	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-3에 나타난 ARM 명령어와 같다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 같다.

예제

```
ADD     R0, R3, R4      ;  R0 := R3 + R4 and set condition codes on the result.
SUB     R6, R2, #6      ;  R6 := R2 - 6 and set condition codes.
```

포맷 3 : MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE

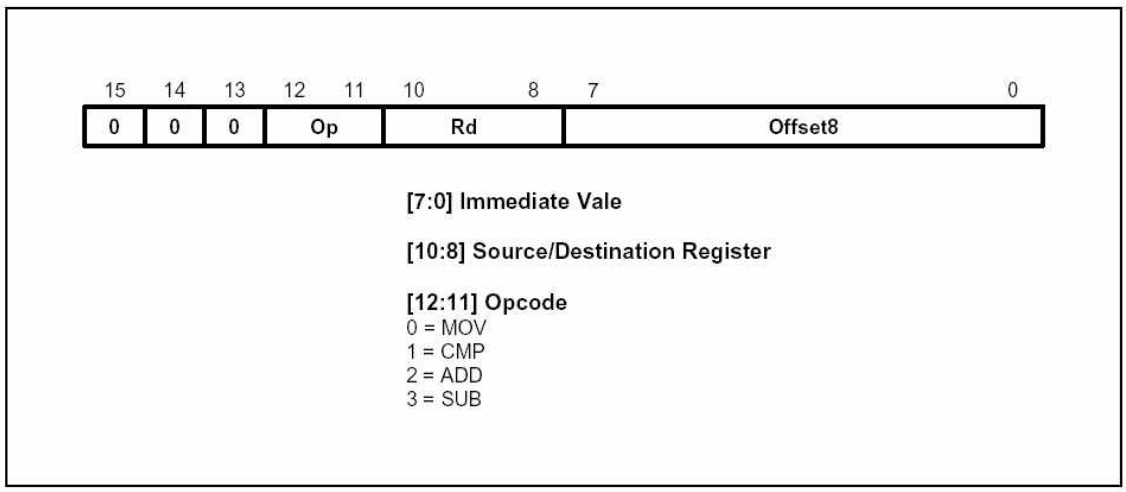

그림 4-4. 포맷 3

동작

이 그룹 안의 명령어는 Lo 레지스터와 8비트 immediate 값 사이에서 수행된다. THUMB 어셈블러 문법이 표 4-4에 나타나 있다.

NOTE

이 그룹 안의 모든 명령어는 CPSR condition code를 설정한다.

표 4-4. 포맷 3 명령어에 대한 요약

OP	THUMB Assembler	ARM Equipment	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

명령어 사이클 시간

이 포맷의 명령어는 표 4-4에 나타난 ARM 명령어와 같다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 같다.

예제

```
MOV     R0, #128        ;  R0 := 128 and set condition codes
CMP     R2, #62         ;  Set condition codes on R2 - 62
ADD     R1, #255        ;  R1 := R1 + 255 and set condition codes
SUB     R6, #145        ;  R6 := R6 - 145 and set condition codes
```

포맷 4 : ALU operation

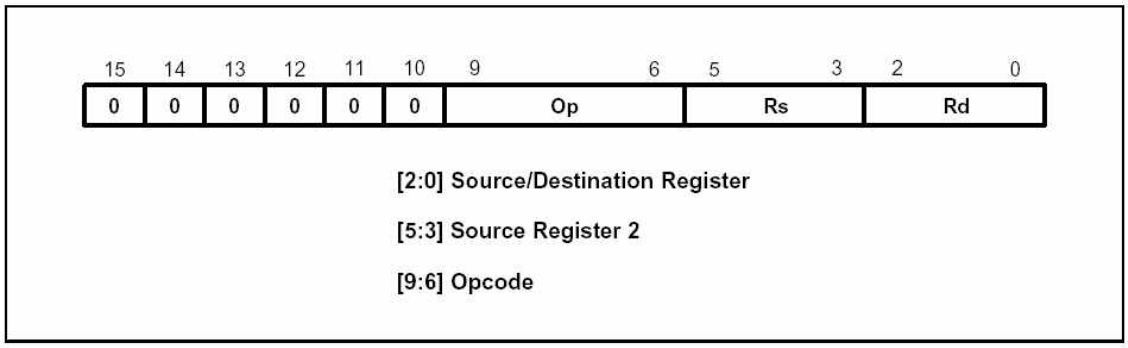

그림 4-5. 포맷 4

동작

아래의 명령어는 Lo 레지스터에서 ALU 동작을 수행한다.

NOTE

이 그룹 안의 모든 명령어는 CPSR condition code를 설정한다.

표 4-5. 포맷 4 명령어에 대한 요약

OP	THUMB Assembler	ARM Equipment	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = - Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-5에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
EOR     R3, R4      ; R3 := R3 EOR R4 and set condition codes
ROR     R1, R0      ; Rotate Right R1 by the value in R0, store
                    ; the result in R1 and set condition codes
NEG     R5, R3      ; Subtract the contents of R3 from zero,
                    ; Store the result in R5. Set condition codes ie R5 = - R3
CMP     R2, R6      ; Set the condition codes on the result of R2 - R6
MUL     R0, R7      ; R0 := R7 * R0 and set condition codes
```

포맷 5 : HI-Register Operations/Branch Exchange

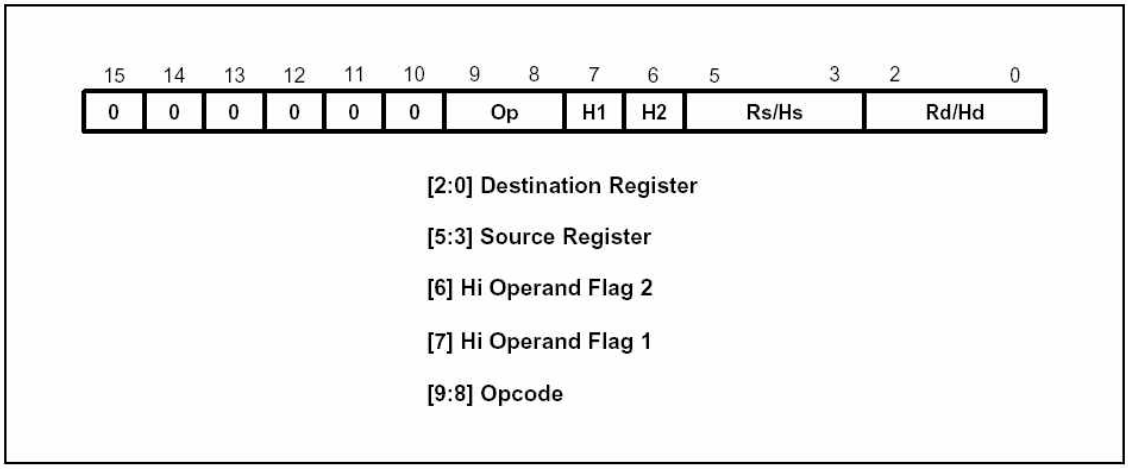

그림 4-6. 포맷 5

동작

이 그룹에는 4가지의 명령어가 있다. 처음 3개는 Lo와 Hi 레지스터 사이에서 수행되는 ADD, CMP와 MOV 동작을 나타낸다. 4번째 BX는 프로세서 상태를 전환하는데 사용되는 Branch를 나타낸다. THUMB 어셈블러 문법은 표 4-6에 나타나 있다.

NOTES

이 그룹에서 CMP(Op=01)는 CPSR condition code를 설정한다.
Op=00(ADD)와 Op=10(MOV)에 대해서 H1=0, H2=0의 동작이 정의되지 않았기 때문에 사용되어서는 안된다.

표 4-6. 포맷 5의 명령어에 대한 요약

Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-6에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

BX 명령어

BX는 Lo나 Hi 레지스터에 규정된 시작 어드레스의 루틴으로 Branch를 수행한다.

어드레스의 비트0은 루틴에 대한 엔트리에서 프로세서의 상태를 결정한다:

비트0=0　　프로세서가 ARM 상태로 진입한다.
비트0=1　　프로세서가 THUMB 상태로 진입한다.

NOTE

이 명령어에 대해서 H1=1의 동작은 정의되지 않았으며, 사용되어서는 안된다.

예제

Hi-Register Operations

```
ADD     PC, R5          ; PC := PC + R5 but don't set the condition codes.
CMP     R4, R12         ; Set the condition codes on the result of R4 - R12.
MOV     R15, R14        ; Move R14 (LR) into R15 (PC)
                        ; but don't set the condition codes,
                        ; eg. return from subroutine.
```

Branch and Exchange

```
                        ; Switch from THUMB to ARM state.
ADR     R1,outofTHUMB   ; Load address of outofTHUMB into R1.
MOV     R11,R1
BX      R11             ; Transfer the contents of R11 into the PC.
                        ; Bit 0 of R11 determines whether
                        ; ARM or THUMB state is entered, ie. ARM state here.
•
•
ALIGN
CODE32
outofTHUMB              ; Now processing ARM instructions...
```

R15를 오퍼랜드로 사용하기

R15가 오퍼랜드로 사용되면, 이 값은 비트0이 클리어 되는 명령어+4의어드레스가 된다. non-워드 정렬 어드레스에서 THUMB 상태의 BX PC를 수행하면 예측할 수 없는 결과를 가져온다.

포맷 6 : PC-Relative Load

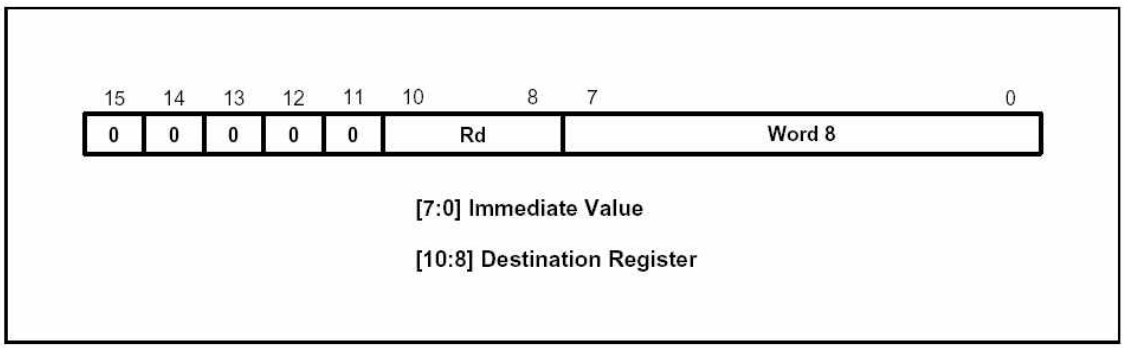

그림 4-7. 포맷 6

동작

이 명령어는 PC에서 10비트 immediate 옵셋으로 정의된 어드레스에서 워드를 호출한다. THUMB 어셈블러 문법은 아래와 같다.

표 4-7. PC-관련 호출 명령어의 요약

THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE : #lmm으로 규정되는 값은 full 10비트 어드레스이지만, 항상 워드-정렬(즉 비트 1:0이 0으로 설정된다.)이 되어야 하는데 이유는 어셈블러가 field 워드 8안에 #lmm >> 2이기 때문이다. PC의 값은 명령어의 어드레스 보다 큰 4바이트가 되지만, PC의 비트 1은 워드 정렬을 위해서 0으로 된다.

명령어 사이클 시간

이 포맷의 모든 명령어는 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
LDR R3,[PC,#844]        ; Load into R3 the word found at the
                        ; address formed by adding 844 to PC.
                        ; bit[1] of PC is forced to zero.
                        ; Note that the THUMB opcode will contain
                        ; 211 as the Word8 value.
```

포맷 7 : LOAD/STORE with Register Offset

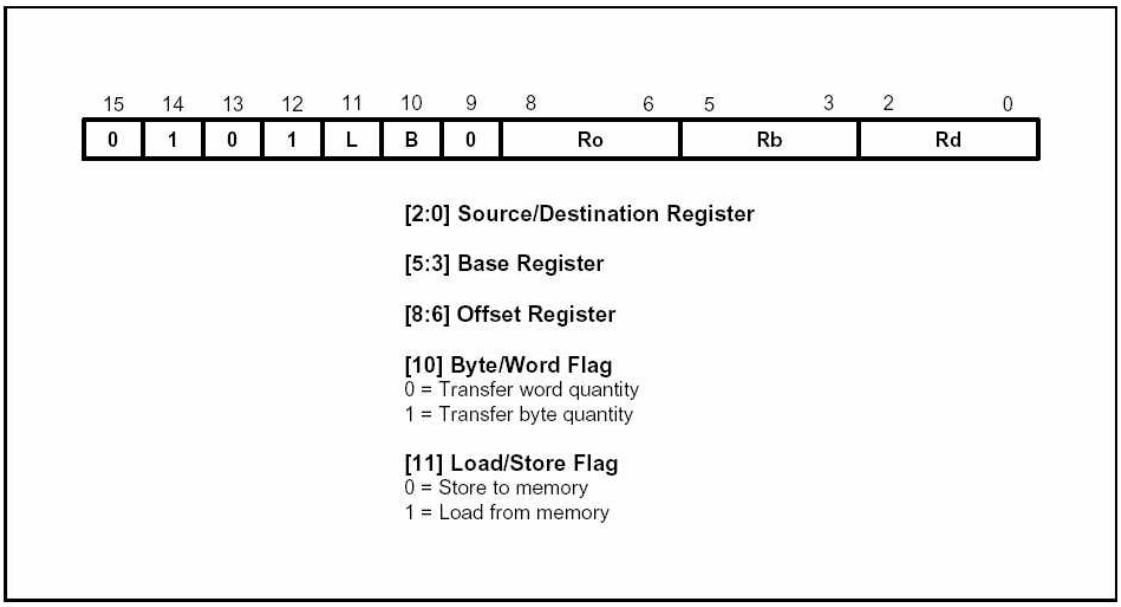

그림 4-8. 포맷 7

동작

이 명령어는 레지스터와 메모리 사이에 바이트나 워드 값을 전송한다. 메모리 어드레스는 0에서 7까지의 범위에 있는 옵셋 레지스터를 사용하는 pre-index된다. THUMB 어셈블러 문법은 표 4-8에 나타나 있다.

표 4-8. 포맷 7 명령어의 요약

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-8에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
STR     R3, [R2,R6]     ; Store word in R3 at the address
                        ; formed by adding R6 to R2.
LDRB    R2, [R0,R7]     ; Load into R2 the byte found at
                        ; the address formed by adding R7 to R0.
```

포맷 8 :LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD

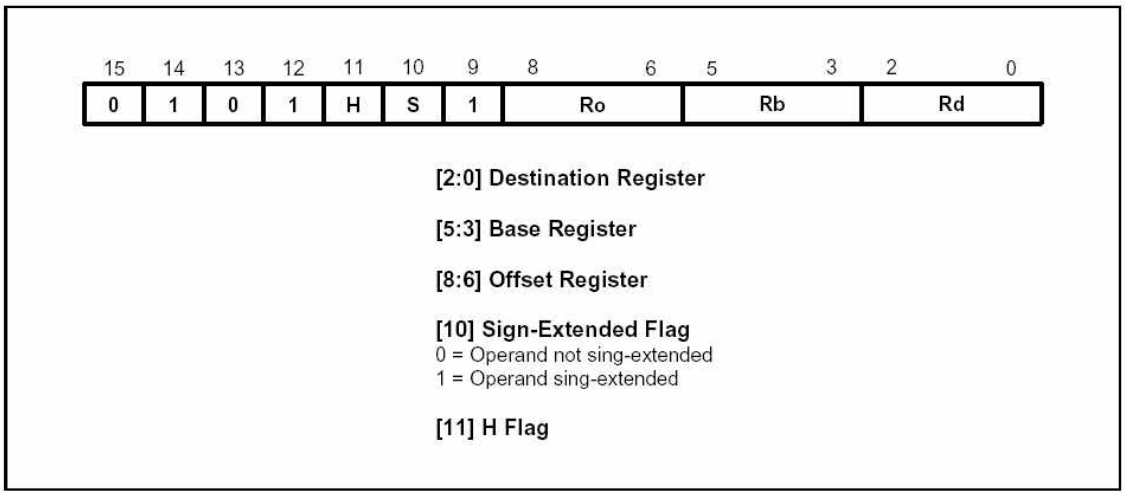

그림 4-9. 포맷 8

동작

이 명령어는 옵션으로 sign-extended 바이트나 halfwordfm fghcnf하고 halfword를 저장한다. THUMB 어셈블러 문법은 아래와 같다.

표 4-9. 포맷 8 명령어에 대한 요약

L	B	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-9에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
STRH    R4, [R3, R0]    ; Store the lower 16 bits of R4 at the
                        ; address formed by adding R0 to R3.
LDSB    R2, [R7, R1]    ; Load into R2 the sign extended byte
                        ; found at the address formed by adding R1 to R7.
LDSH    R3, [R4, R2]    ; Load into R3 the sign extended halfword
                        ; found at the address formed by adding R2 to R4.
```

포맷 9 : LOAD/STORE with IMMEDIATE OFFSET

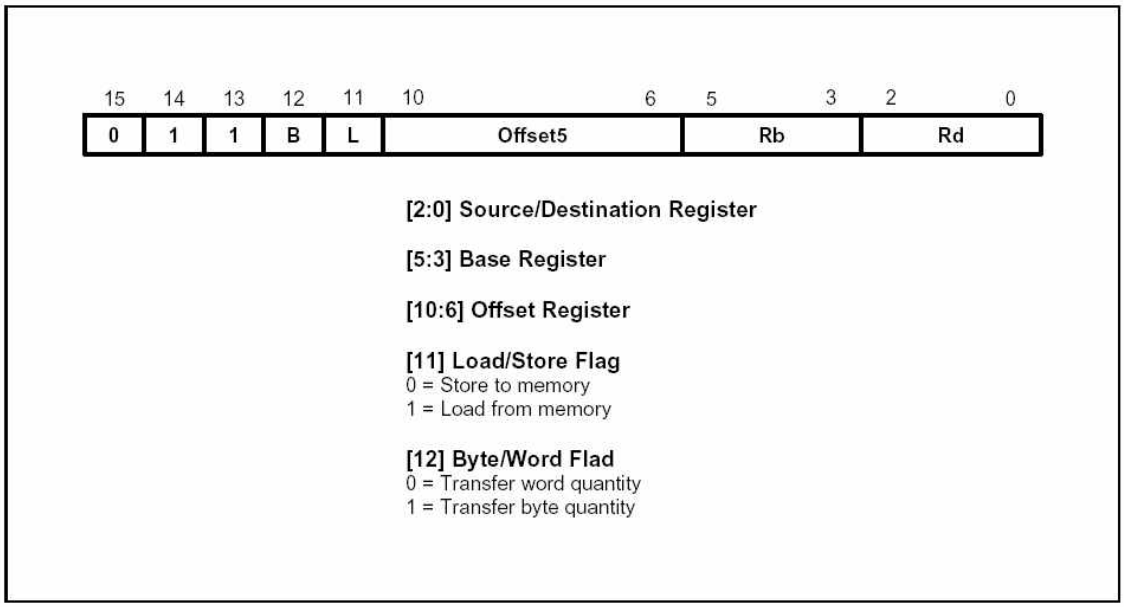

그림 4-10. 포맷 9

동작

이 명령어는 immediate 5나 7비트 옵셋을 사용해서 레지스터와 메모리 사이의 바이트나 워드 값을 전송한다. THUMB 어셈블러 문법은 표 4-10에 나타나 있다.

표 4-10. 포맷 9 명령어에 대한 요약

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE : 워드 접근(B=0)에 대해서, #Imm에 의한 값은 full 7비트 어드레스이지만, 어셈블러가 Offset5 field의 #Imm >>2에 놓이기 때문에 워드-정렬이 되어야 한다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-10에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
LDR     R2, [R5,#116]   ; Load into R2 the word found at the
                        ; address formed by adding 116 to R5.
                        ; Note that the THUMB opcode will
                        ; contain 29 as the Offset5 value.
STRB    R1, [R0,#13]    ; Store the lower 8 bits of R1 at the
                        ; address formed by adding 13 to R0.
                        ; Note that the THUMB opcode will
                        ; contain 13 as the Offset5 value.
```

포맷 10: LOAD/STORE HALFWORD

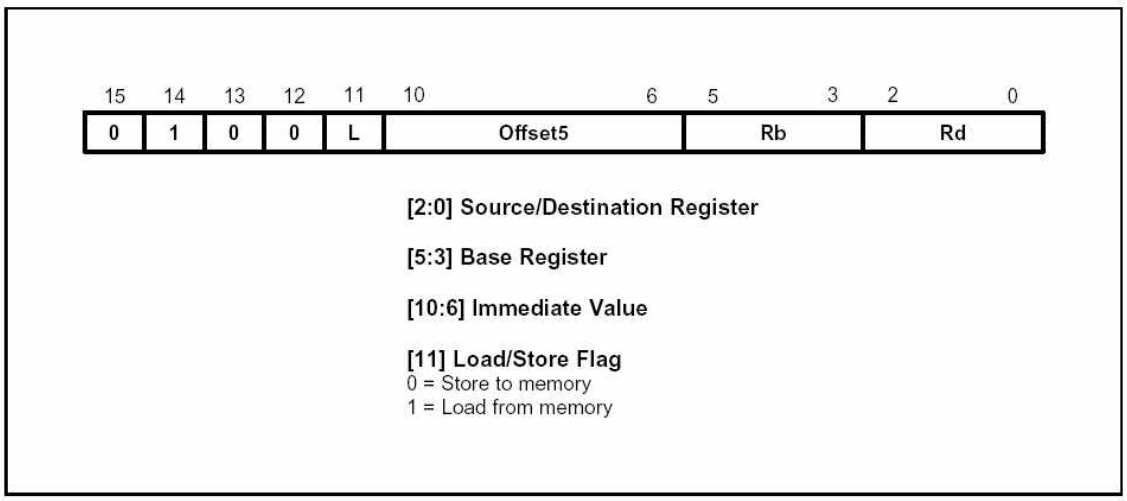

그림 4-11. 포맷 10

동작

이 명령어는 Lo 레지스터와 메모리 사이의 halfword 값을 전송한다. 어드레스는 pre-index 이며, 6비트 immediate 값을 이용한다. THUMB 어셈블러 문법은 표 4-11에 나타나 있다.

표 4-11. Halfword 데이터 전송 명령어

L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0 - 15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE : #Imm은 full 6비트 어드레스이지만 어셈블러가 Offset5 field에 #Imm>> 1로 놓이기 때문에 halfword-정렬(비트 0이 0으로 설정)이어야 한다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-11에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
STRH    R6, [R1, #56]   ; Store the lower 16 bits of R4 at the address formed by
                        ; adding 56 R1. Note that the THUMB opcode will contain
                        ; 28 as the Offset5 value.
LDRH    R4, [R7, #4]    ; Load into R4 the halfword found at the address formed by
                        ; adding 4 to R7. Note that the THUMB opcode will contain
                        ; 2 as the Offset5 value.
```

포맷 11 : SP-Relative LOAD/STORE

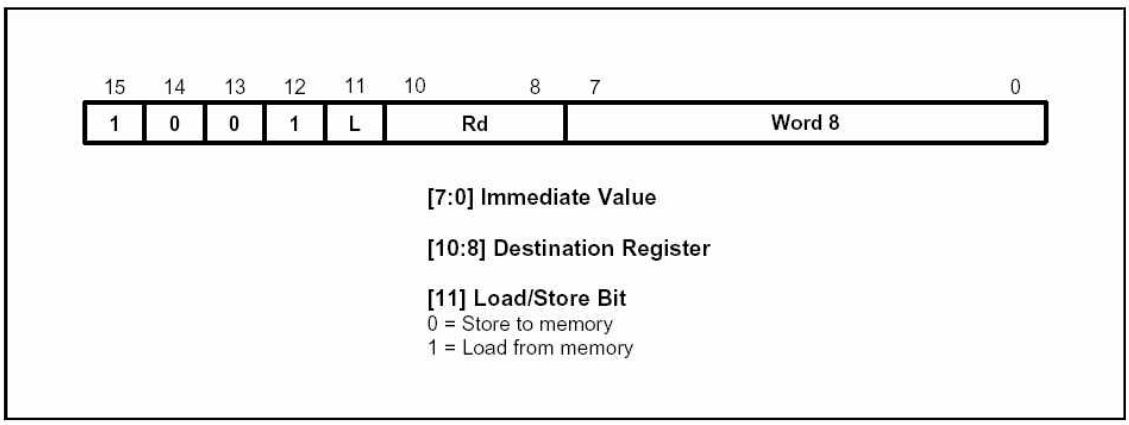

그림 4-12. 포맷 11

동작

이 명령러는 SP-relative 로드나 저장을 수행한다. THUMB 어셈블러 문법은 아래의 표에 나타나 있다.

표 4-12. SP-Relative Load/Store 명령어

L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE : #Imm에 공급되는 옵셋은 full 10비트 어드레스이며, 어셈블러가 Word8 field에서 #Imm >> 2이기 때문에 항상 워드-정렬이어야 한다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-12에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
STR        R4, [SP,#492]        ; Store the contents of R4 at the address
                                ; formed by adding 492 to SP (R13).
                                ; Note that the THUMB opcode will contain
                                ; 123 as the Word8 value.
```

포맷 12 : LOAD Address

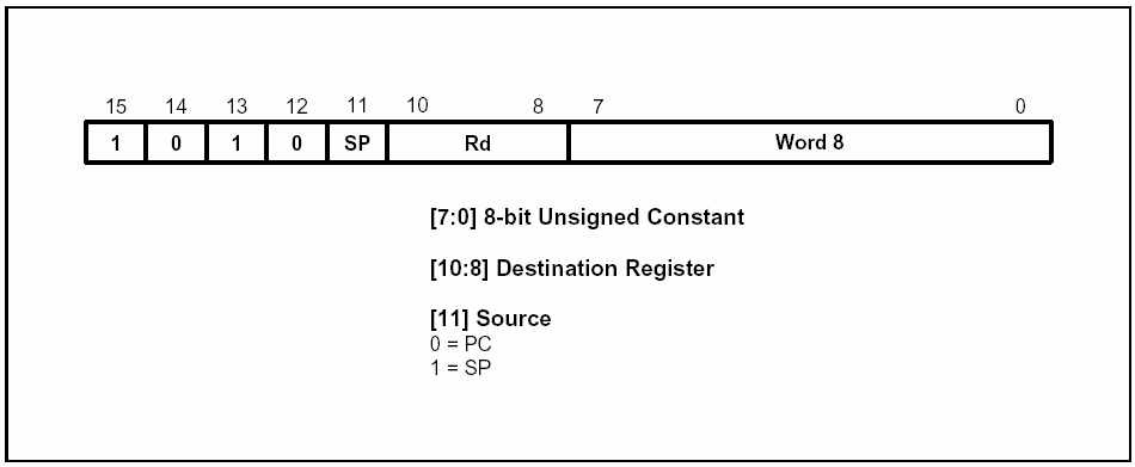

그림 4-13. 포맷 12

동작

이 명령어는 PC나 SP에 10비트 상수를 더해서 어드레스를 계산하며 레지스터에 어드레스 결과를 로드 한다. THUMB 어셈블러 문법은 아래의 표에 나타나 있다.

표 4-13. Load Address

L	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE : #Imm에 의한 값은 full 10비트 값이며, 어셈블러가 field Word 8의 #Imm >> 2이기 때문에 워드-정렬이어야 한다.

여기서 PC는 소스 레지스터로 사용되며, PC의 비트 1은 항상 0으로 읽힌다. PC의 값은 비트 1이 0으로 되기 전에 명령어의 어드레스 보다 4바이트가 크다.

CPSR condition code는 이 명령어의 영향을 받지 않는다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-13에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령

어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
ADD     R2, PC, #572    ; R2 := PC + 572, but don't set the
                        ; condition codes. bit[1] of PC is forced to zero.
                        ; Note that the THUMB opcode will
                        ; contain 143 as the Word8 value.
ADD     R6, SP, #212    ; R6 := SP (R13) + 212, but don't
                        ; set the condition codes.
                        ; Note that the THUMB opcode will
                        ; contain 53 as the Word 8 value.
```

포맷 13 : ADD OFFSET to STACK POINTER

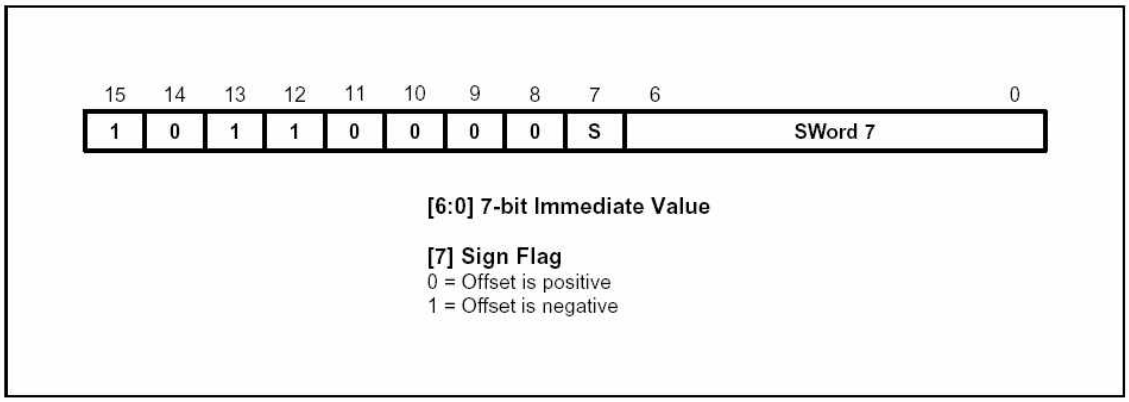

그림 4-14. 포맷 13

동작

이 명령어는 스택 포인터에 9비트 부호있는 상수를 더한다. 아래의 표는 THUMB 어셈블러 문법을 나타내고 있다.

표 4-14. ADD SP 명령어

L	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, # -Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE : #Imm에 의한 옵셋은 -/+ 508까지 이지만, 어셈블러가 #Imm을 field SWord7에 놓이기 전에 8비트 sign + magnitude number로 변환하기 때문에 워드-정렬이어야 한다. condition code는 이 명령어로 설정되지 않는다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-14에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
ADD     SP, #268        ; SP (R13) := SP + 268, but don't set the condition codes.
                        ; Note that the THUMB opcode will
                        ; contain 67 as the Word7 value and S=0.
ADD     SP, #-104       ; SP (R13) := SP - 104, but don't set the condition codes.
                        ; Note that the THUMB opcode will contain
                        ; 26 as the Word7 value and S=1.
```

포맷 14: PUSH/POP Register

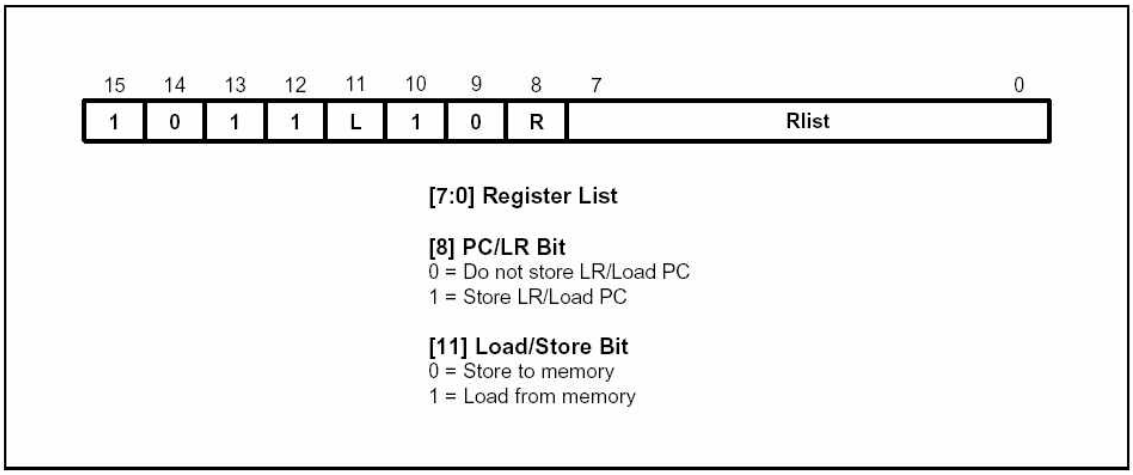

그림 4-15. 포맷 14

동작

이 명령어는 레지스터 0 - 7을 허가하며 스택에 LR을 push하고, PC는 스택에서 pop을 한다. THUMB 어셈블러 문법이 표 4-15에 나타나 있다.

NOTE

스택은 항상 full descend로 가정된다.

표 4-15. PUSH와 POP 명령어

L	B	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-15에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령

어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
PUSH    {R0-R4,LR}      ; Store R0,R1,R2,R3,R4 and R14 (LR) at
                        ; the stack pointed to by R13 (SP) and update R13.
                        ; Useful at start of a sub-routine to
                        ; save workspace and return address.
POP     {R2,R6,PC}      ; Load R2,R6 and R15 (PC) from the stack
                        ; pointed to by R13 (SP) and update R13.
                        ; Useful to restore workspace and return from sub-routine.
```

포맷 15 : MULTIPLE LOAD/STORE

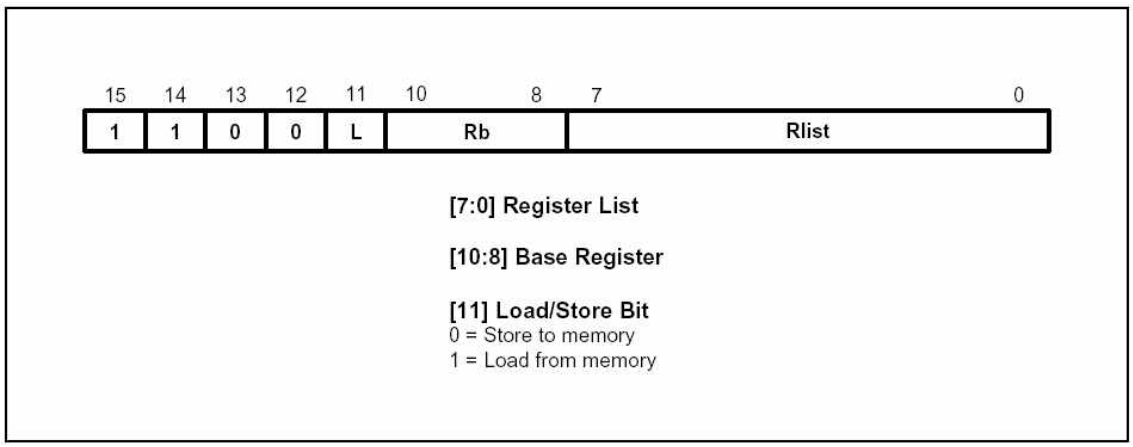

그림 4-16. 포맷 15

동작

이 명령어는 Lo 레지스터를 여러번 호출하고 저장한다. THUMB 어셈블러 문법은 아래의 표에 나타나 있다.

표 4-16. Multiple Load/Store 명령어

L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-16에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
STMIA     R0!, {R3-R7}        ; Store the contents of registers R3-R7
                              ; starting at the address specified in
                              ; R0, incrementing the addresses for each word.
                              ; Write back the updated value of R0.
```

포맷 16 : Conditional Branch

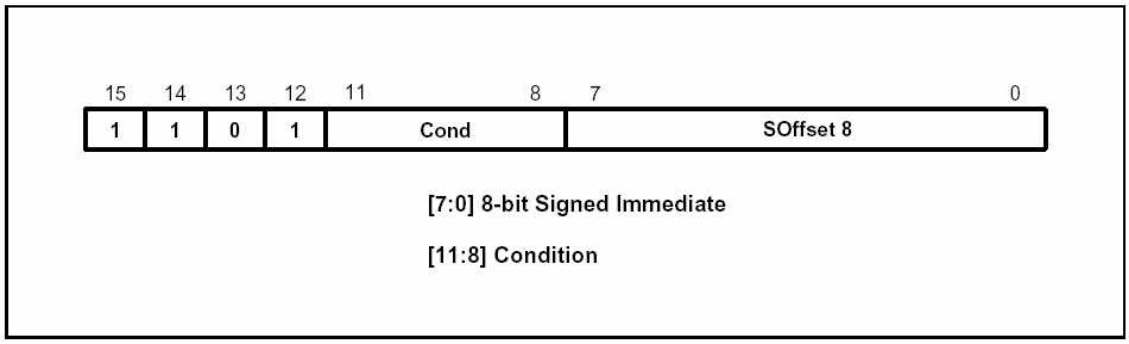

그림 4-17. 포맷 16

동작

이 명령어는 CPSR condition code의 상태에 따라서 conditional Branch를 수행한다. branch 옵셋은 현재의 명령어 앞에 PC를 1워드로 만드는 prefetch 동작을 수행해야 한다.

THUMB 어셈블러 문법은 아래의 표에 나타나 있다.

표 4-17. Conditional Branch 명령어

L	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)

L	THUMB assembler	ARM equivalent	Action
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES :

1. 라벨이 full 9비트 2의 보수 어드레스를 규정하는 동안에, 항상 halfword-정렬되어야 하며, 어셈블러는 실제로 field SOffset8의 label >>1에 놓여야 한다.
2. Cond = 1110은 정의되지 않으면, 사용되지 않아야 한다.
   Cond = 1111은 SWI 명령어를 만든다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 3-1에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
        CMP R0, #45          ; Branch to over-if R0 > 45.
        BGT over             ; Note that the THUMB opcode will contain
        •                    ; the number of halfwords to offset.
        •
        •
over    •                    ; Must be halfword aligned.
```

포맷 17 : 소프트웨어 인터럽트

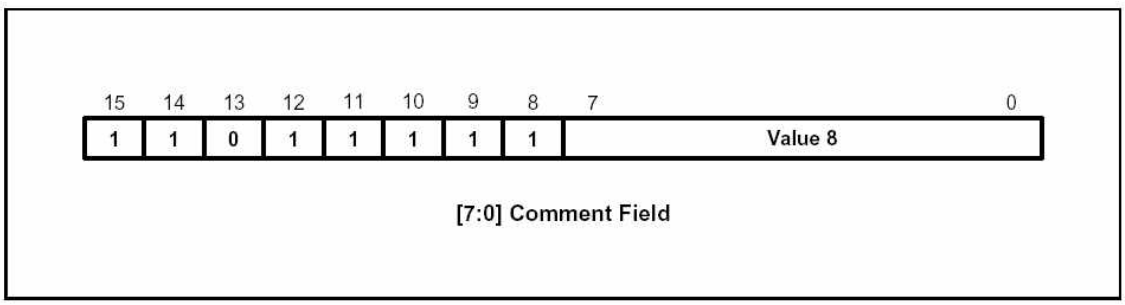

그림 4-18. 포맷 17

동작

SWI 명령어는 소프트웨어 인터럽트를 수행한다. SWI를 취하면, 프로세서는 ARM 상태로 전환되며, 슈퍼바이저 모드(SVC)로 진입한다.

이 명령어에 대한 THUMB 어셈블러 문법이 아래에 나타나 있다.

표 4-18. SWI 명령어

THUMB assembler	ARM equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE : Value 8이 SWI 핸들러에 의해서 사용된다; 프로세서에 의해서 무시된다.

명령어 사이클 시간

이 포맷의 모든 명령어는 표 4-18에 나타나 것처럼 ARM 명령어와 동일하다. THUMB 명령어에 대한 사이클 시간은 ARM 명령어와 동일하다.

예제

```
SWI 18                  ; Take the software interrupt exception.
                        ; Enter Supervisor mode with 18 as the
                        ; requested SWI number.
```

포맷 18 : Unconditional Branch

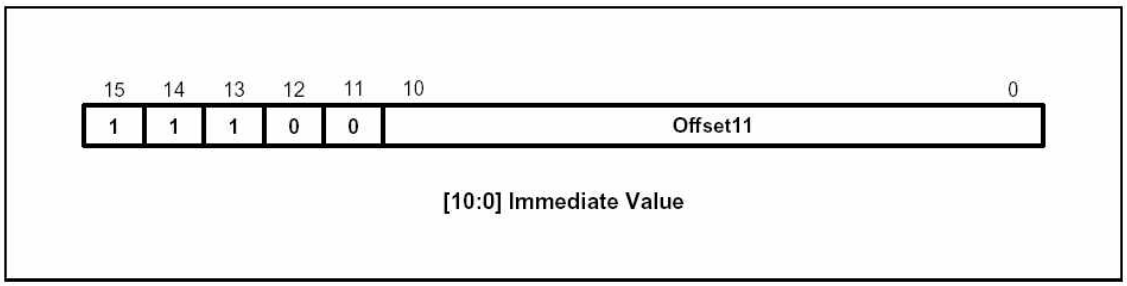

그림 4-19. 포맷 18

동작

이 명령어는 PC-relative Branch를 수행한다. THUMB 어셈블러 문법이 아래에 나타나 있다. Branch 옵셋은 현재의 명령어 이전에 PC를 1워드로 만드는 prefetch 동작을 취해야 한다.

표 4-19. Branch 명령어의 요약

THUMB assembler	ARM equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE : 라벨에 의한 어드레스는 full 12비트 2의 보스 어드레스이지만, 어셈블러가 Offset11 field의 label >> 1이기 때문에 항상 halfword 정렬이어야 한다.

예제

```
here    B here          ;  Branch onto itself. Assembles to 0xE7FE.
                        ;  (Note effect of PC offset).
        B jimmy         ;  Branch to 'jimmy'.
        •               ;  Note that the THUMB opcode will contain the number of
        •
        •               ;  halfwords to offset.
jimmy   •               ;  Must be halfword aligned.
```

포맷 19 : Long Branch with Link

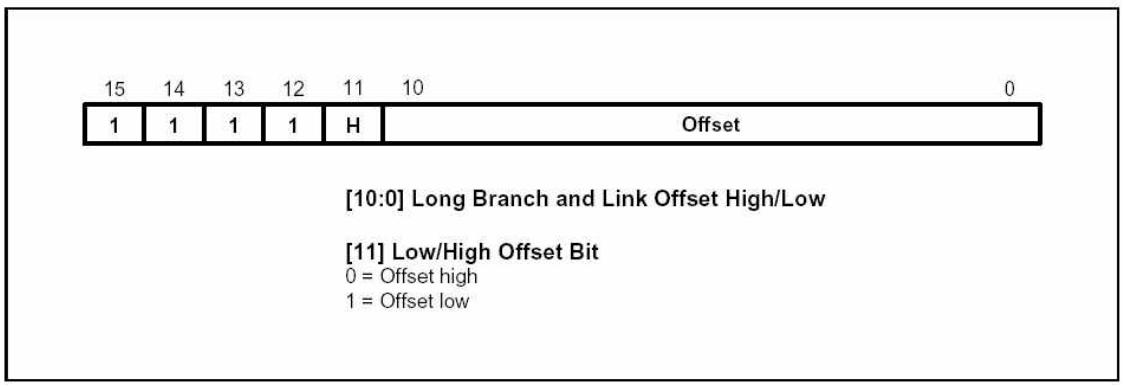

그림 4-20. 포맷 19

동작

이 포맷은 link를 갖는 long branch를 정의한다.

어셈블러는 2개의 11비트 halve에 label을 정의해서 23비트 2의 보수 half-word 옵셋을 split하며, 비트 0을 무시하고, 2개의 THUMB 명령어를 만든다.

명령어 1(H=0)

처음 명령어 Offset field는 타겟 어드레스의 상위 11비트를 포함한다. 12비트 만큼 왼쪽으로 쉬프트 되며 현재의 PC 어드레스에 더해진다. 어드레스의 결과는 LR에 놓인다.

명령어 2(H=1)

두 번째 명령어 Offset field는 타겟 어드레스의 11비트를 나타내는 하위 half를 포함한다. 이것은 왼쪽으로 1비트가 쉬프트 되며 LR에 더해진다. full 23비트 어드레스를 포함하는 LR은 PC에 놓이게 되며, BL을 따르는 명령어의 어드레스는 LR에 놓이고 lR의 비트 0이 설정된다.

Branch Offset는 현재의 명령어 전에 PC를 1워드로 만드는 prefetch 동작을 수행해야 한다.

명령어 사이클 시간

이 명령어 포맷은 ARM 명령어와 같지 않다.

표 4-20. BL 명령어

L	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12
1			temp := next instruction address PC := LR + OffsetLow << 1 LR := temp | 1

예제

```
        BL faraway      ; Unconditionally Branch to 'faraway'
next    •               ; and place following instruction
        •               ; address, ie "next", in R14,the Link
                        ; register and set bit 0 of LR high.
                        ; Note that the THUMB opcodes will
                        ; contain the number of halfwords to offset.
faraway •               ; Must be Half-word aligned.
        •
```

명령어 설정 예제

아래의 예제는 THUMB 명령어가 작고 효율적인 코드를 발생하는데 사용되는 예를 보여주고 있다. 각 예제는 ARM을 비교할 수 있도록 나타내고 있다.

Shift와 Add를 이용한 상수에 의한 곱셈

아래의 예는 ARM 등가에 따른 1,2, 3 THUMB 명령어를 사용해서 다양한 상수를 곱하는 코드를 나타낸다. 다른 상수는 4 이상의 명령어를 이용하는 것 보다는 built-in MUL 명령어를 사용하는 것이 낫다.

```
                Thumb       ARM

1. Multiplication by 2^n (1,2,4,8,...)

                LSL         Ra, Rb, LSL #n          ;  MOV Ra, Rb, LSL #n

2. Multiplication by 2^n+1 (3,5,9,17,...)

                LSL         Rt, Rb, #n              ;  ADD Ra, Rb, Rb, LSL #n
                ADD         Ra, Rt, Rb

3. Multiplication by 2^n-1 (3,7,15,...)

                LSL         Rt, Rb, #n              ;  RSB Ra, Rb, Rb, LSL #n
                SUB         Ra, Rt, Rb

4. Multiplication by -2^n (-2, -4, -8, ...)

                LSL         Ra, Rb, #n              ;  MOV Ra, Rb, LSL #n
                MVN         Ra, Ra                  ;  RSB Ra, Ra, #0

5. Multiplication by -2^n-1 (-3, -7, -15, ...)

                LSL         Rt, Rb, #n              ;  SUB Ra, Rb, Rb, LSL #n
                SUB         Ra, Rb, Rt

Multiplication by any C = {2^n+1, 2^n-1, -2^n or -2^n-1} * 2^n
Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional
constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62 .....

                (2..5)                              ;  (2..5)
                LSL         Ra, Ra, #n              ;  MOV Ra, Ra, LSL #n
```

일반의 부호있는 나누기

이 예제는 THUMB와 ARM코드에서 일반의 부호있는 나누기와 나머지 루틴을 나타내고 있다.

Thumb code

```
;signed_divide                          ;  Signed divide of R1 by R0: returns quotient in R0,
                                        ;  remainder in R1

;Get abs value of R0 into R3
            ASR     R2, R0, #31         ;  Get 0 or -1 in R2 depending on sign of R0
            EOR     R0, R2              ;  EOR with -1 (0×FFFFFFFF) if negative
            SUB     R3, R0, R2          ;  and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
            BEQ     divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
            ASR     R0, R1, #31         ;  Get 0 or -1 in R3 depending on sign of R1
            EOR     R1, R0              ;  EOR with -1 (0×FFFFFFFF) if negative
            SUB     R1, R0              ;  and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
            PUSH     {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
dividend ; right by 1 and stop as soon as shifted value becomes >.
            LSR     R0, R1, #1
            MOV     R2, R3
            B       %FT0
just_l      LSL     R2, #1
0           CMP     R2, R0
            BLS     just_l
            MOV     R0, #0              ;  Set accumulator to 0
            B       %FT0                ;  Branch into division loop

div_l       LSR     R2, #1
0           CMP     R1, R2              ;  Test subtract
            BCC     %FT0
            SUB     R1, R2              ;  If successful do a real subtract
0           ADC     R0, R0              ;  Shift result and add 1 if subtract succeeded

            CMP     R2, R3              ;  Terminate when R2 == R3 (ie we have just
            BNE     div_l               ;  tested subtracting the 'ones' value).


Now fix up the signs of the quotient (R0) and remainder (R1)
            POP     {R2, R3}            ;  Get dividend/divisor signs back
            EOR     R3, R2              ;  Result sign
            EOR     R0, R3              ;  Negate if result sign = - 1
            SUB     R0, R3
            EOR     R1, R2              ;  Negate remainder if dividend sign = - 1
            SUB     R1, R2
            MOV     pc, lr
```

ARM Code

```
signed_divide                                   ;  Effectively zero a4 as top bit will be shifted out later
            ANDS        a4, a1, #&80000000
            RSBMI       a1, a1, #0
            EORS        ip, a4, a2, ASR #32
;ip bit 31 = sign of result
;ip bit 30 = sign of a2
            RSBCS        a2, a2, #0

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)
            MOVS        a3, a1
            BEQ         divide_by_zero

just_l                                          ;  Justification stage shifts 1 bit at a time
            CMP         a3, a2, LSR #1
            MOVLS       a3, a3, LSL #1          ;  NB: LSL #1 is always OK if LS succeeds
            BLO         s_loop

div_l
            CMP         a2, a3
            ADC         a4, a4, a4
            SUBCS       a2, a2, a3
            TEQ         a3, a1
            MOVNE       a3, a3, LSR #1
            BNE         s_loop2
            MOV         a1, a4
            MOVS        ip, ip, ASL #1
            RSBCS       a1, a1, #0
            RSBMI       a2, a2, #0
            MOV         pc, lr
```

상수로 나누기

상수로 나누기는 shift, add, subtract와 같은 고정된 짧은 시퀀스로 수행이 가능하다. 여기서는 THUMB과 ARM 코드에서 ARM Cookbook의 알고리즘에 기반하는 10 루틴으로 나누는 예를 나타내고 있다.

Thumb Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        MOV     a2, a1
        LSR     a3, a1, #2
        SUB     a1, a3
        LSR     a3, a1, #4
        ADD     a1, a3
        LSR     a3, a1, #8
        ADD     a1, a3
        LSR     a3, a1, #16
        ADD     a1, a3
        LSR     a1, #3
        ASL     a3, a1, #2
        ADD     a3, a1
        ASL     a3, #1
        SUB     a2, a3
        CMP     a2, #10
        BLT     %FT0
        ADD      a1, #1
        SUB     a2, #10
0
        MOV      pc, lr
```

ARM Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        SUB     a2, a1, #10
        SUB     a1, a1, a1, lsr #2
        ADD     a1, a1, a1, lsr #4
        ADD     a1, a1, a1, lsr #8
        ADD     a1, a1, a1, lsr #16
        MOV     a1, a1, lsr #3
        ADD     a3, a1, a1, asl #2
        SUBS    a2, a2, a3, asl #1
        ADDPL   a1, a1, #1
        ADDMI   a2, a2, #10
        MOV     pc, lr
```