5. ARM 명령어 세트

– 전체 요약

In Chapter 3 we looked at user-level ARM assembly language programming and got the general flavour of the ARM instruction set. In this chapter we will look in finer detail at the instruction set to see the full range of instructions that are available in the standard ARM instruction set.

Some ARM cores will also execute a compressed form of the instruction set where a subset of the full ARM instruction set is encoded into 16-bit instructions. These instructions are 'Thumb' instructions, and are discussed in Chapter 7. The only aspects of the Thumb architecture we will see in this chapter are the instructions available in the ARM instruction set which cause the processor to switch to executing Thumb instructions. Likewise, some ARM cores support instruction set extensions to enhance their signal processing capabilities, discussion of which is deferred to Section 8.9 on page 239.

As with any processor's full instruction set, the ARM instruction set has corners which conceal complex behaviour. Often these corners are not at all useful to programmers, in which case ARM Limited does not define the behaviour of the processor in the corner cases and the corresponding instructions should not be used. The fact that a particular implementation of the ARM behaves in a particular way in such a case should not be taken as meaning that future implementations will behave the same way. Programs should only use instructions with defined semantics!

Some ARM instructions are not available on all ARM chips; these will be highlighted as they arise.

5.1 서론

- 데이터 형태

▷ ARM은 기본적으로 6개의 데이터 형태를 지원함

8, 16, 32 비트 signed, unsigned byte, half-word, word (half-word와 word는 메모리에서 2-바이트, 4-바이트 경계로 align됨)

▷ ARM의 모든 명령어는 32 비트 워드, word-align 되어 있고 Thumb 명령어는 half-word이며 2 바이트 경계로 align 되어 있음

▷ ARM coprocessor는 부동 소수점과 같은 다른 데이터 형태를 지원

▷ 부동 소수점 coprocessor가 없는 경우 부동 소수점 형태는 기본 데이터 형태를 사용하는 소프트웨어에 의해 번역 (변환)됨

- 메모리 구조

▷ Little endian : 주소가 증가하는 방향으로 하위 바이트부터 차례로 저장

▷ Big endian : 주소가 증가하는 방향으로 상위 바이트부터 차례로 저장

▷ ARM은 기본적으로 little-endian이지만 어느 방식도 사용 가능

- 특권 모드 (privileged mode)

▷ 대부분의 프로그램은 사용자 모드에서 처리되지만 예외를 처리하거나 supervisor calls을 처리하기 위해 특별한 동작 모드를 제공

▷ 현재의 동작 모드는 CPSR 레지스터의 하위 5비트에 의해 설정 가능

CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

▷ 각 특권 모드가 시작될 때 사용자 모드가 정상적으로 다시 시작될 수 있도록 CPSR의 상태는 각 모드의 SPSR 레지스터에 저장되어야 함

5.2 예외

- 예외의 종류

▷ 예외는 인터럽트와 메모리 폴트와 같이 프로그램 실행 동안 예기치 못한 사건을 처리하기 위해 사용, ARM은 3가지 형태의 예외로 구분

1. Exceptions generated as the direct effect of executing an instruction.
   Software interrupts, undefined instructions (including coprocessor instructions where the requested coprocessor is absent) and prefetch aborts (instructions that are invalid due to a memory fault occurring during fetch) come under this heading.
2. Exceptions generated as a side-effect of an instruction.
   Data aborts (a memory fault during a load or store data access) are in this class.
3. Exceptions generated externally, unrelated to the instruction flow.
   Reset, IRQ and FIQ fall into this category.

- 예외 시작

▷ 예외가 발생하면 ARM은 현재의 명령어를 처리하고 예외 처리를 시작, 예외가 reset인 경우 현재의 명령어를 즉시 종료하고 예외 처리

▷ side-effect나 외부에서 발생하는 예외는 다음 명령어를 침해하여 예외 시퀀스가 처리되지만 direct-effect 예외는 발생하는 시퀀스에서 처리됨

▷ 예외가 발생할 때 프로세서의 동작 시퀀스

- It changes to the operating mode corresponding to the particular exception.
- It saves the address of the instruction following the exception entry instruction in r14 of the new mode.
- It saves the old value of the CPSR in the SPSR of the new mode.
- It disables IRQs by setting bit 7 of the CPSR and, if the exception is a fast interrupt, disables further fast interrupts by setting bit 6 of the CPSR.
- It forces the PC to begin executing at the relevant vector address given in Table 5.2.

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

▷ 각 벡터 어드레스에는 예외 처리 함수로 이동하는 branch 명령이 있음

▷ 각 특권 모드에는 두 개의 bank 레지스터 (r13_{mod}, r14_{mod})가 복귀 주소와 스택 포인터를 저장하기 위해 사용

▷ 스택 포인터는 예외 처리에서 사용되는 레지스터를 스택에 저장

▷ FIQ 모드에서는 보다 나은 성능을 위해 추가적인 레지스터(r8_fiq – r12_fiq)

를 제공, 사용자 레지스터를 저장할 필요가 없거나 최대한 줄일 수 있음

▷ Supervisor call이 supervisor call을 요청하는 경우 (re-entrant) SPSR 레지스터는 일반 레지스터에 저장되어야 함

- 예외 복귀

- Any modified user registers must be restored from the handler's stack.
- The CPSR must be restored from the appropriate SPSR.
- The PC must be changed back to the relevant instruction address in the user instruction stream.

▷ CPSR과 PC 복원은 동시에 실행되어야 함

CPSR이 먼저 복원되면 bank 레지스터 r14는 접근 불가능하고 PC이 먼저 복원되면 명령어 흐름이 바뀌어 CPSR 복원이 불가능

▷ 하나의 명령어로 두 단계가 동시에 실행되는 메커니즘을 제공

▷ 복귀 주소가 bank 레지스터 r14에 있는 경우의 복원

- To return from a SWI or undefined instruction trap use:

```
MOVS    pc, r14
```

- To return from an IRQ, FIQ or prefetch abort use:

```
SUBS    pc, r14, #4
```

- To return from a data abort to retry the data access use:

```
SUBS    pc, r14, #8
```

- IRQ and FIQ must return one instruction early in order to execute the instruction that was 'usurped' for the exception entry.
- Prefetch abort must return one instruction early to execute the instruction that had caused a memory fault when first requested.
- Data abort must return two instructions early to retry the data transfer instruction, which was the instruction *before* the one usurped for exception entry.

▷ 복귀 주소가 스택에 있는 경우 (re-entrant 동작)는 스택 어드레싱에 의한 다중 레지스터 이동 명령어를 사용하여 복원

```
LDMFD   r13!, {r0-r3, pc}^      ; restore and return
```

▷ PC이 복원되는 동시에 CPSR도 복원됨, 스택 복귀 메커니즘 사용을 위해 복귀 주소가 스택에 저장되기 전에 조정되어야 함

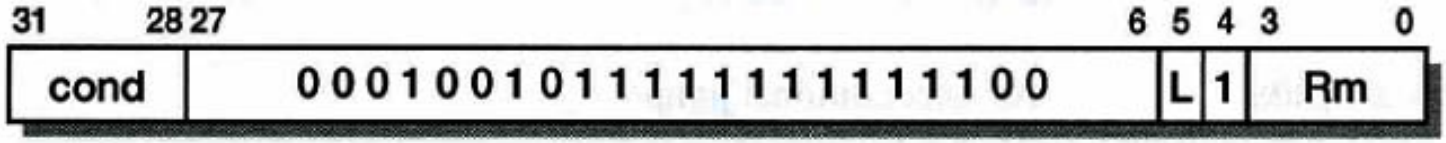
- 예외 우선 순위

▷ 여러 개의 예외가 동시에 발생하는 경우 우선 순서가 높은 예외부터 처리

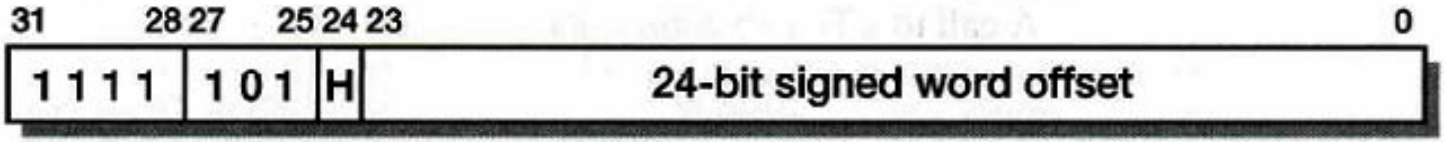
1. reset (highest priority);
2. data abort;
3. FIQ;
4. IRQ;
5. prefetch abort;
6. SWI, undefined instruction (including absent coprocessor). These are mutually exclusive instruction encodings and therefore cannot occur simultaneously.

5.5 Branch, Branch with Link and eXchange (BX, BLX)

(1) BX|BLX Rm



(2) BLX label



In the first format the branch target is specified in a register, Rm. Bit[0] of Rm is copied into the T bit in the CPSR and bits[31:1] are moved into the PC:

- If Rm[0] is 1, the processor switches to execute Thumb instructions and begins executing at the address in Rm aligned to a half-word boundary by clearing the bottom bit.
- If Rm[0] is 0, the processor continues executing ARM instructions and begins executing at the address in Rm aligned to a word boundary by clearing Rm[1].

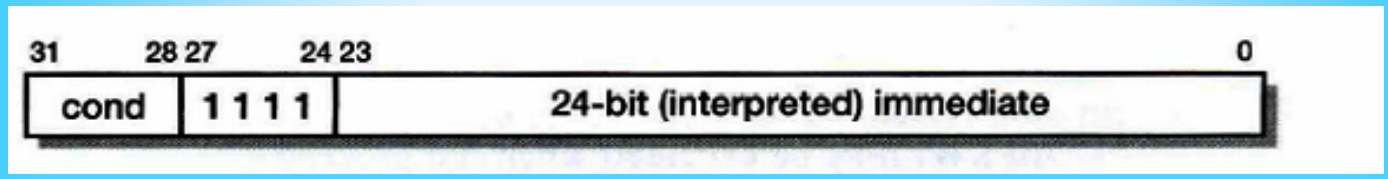
An unconditional jump:

```
        BX      r0      ; branch to address in r0,
                        ; enter Thumb state if r0[0] = 1
```

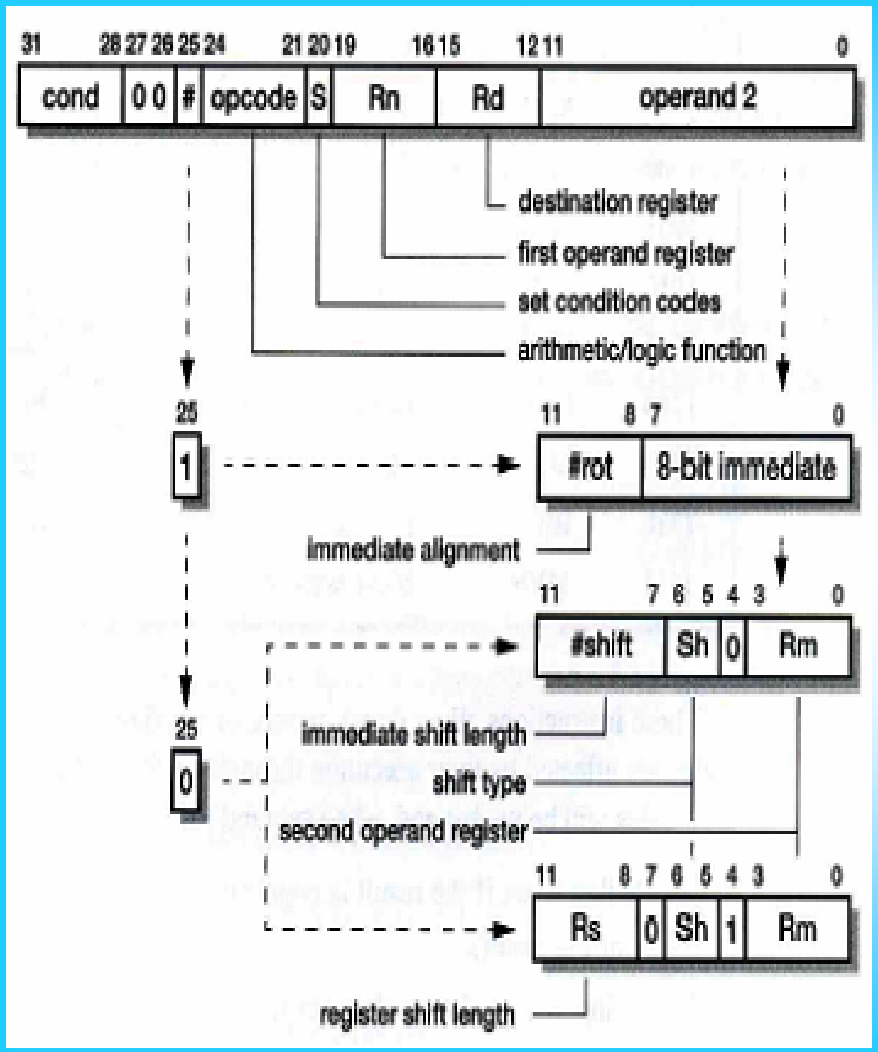
A call to a Thumb subroutine:

```
        CODE32          ; ARM code follows
        ..
        BLX     TSUB    ; call Thumb subroutine

        ..
        CODE16          ; start of Thumb code
TSUB    ..              ; Thumb subroutine
        BX      r14     ; return to ARM code
```

Software Interrupt (SWI)



5.7 데이터 처리 명령어 형식



Opcode [24:21]	Mnemonic	Meaning
0000	AND	Logical bit-wise AND
0001	EOR	Logical bit-wise exclusive OR
0010	SUB	Subtract
0011	RSB	Reverse subtract
0100	ADD	Add
0101	ADC	Add with carry
0110	SBC	Subtract with carry
0111	RSC	Reverse subtract with carry
1000	TST	Test
1001	TEQ	Test equivalence
1010	CMP	Compare
1011	CMN	Compare negated
1100	ORR	Logical bit-wise OR
1101	MOV	Move
1110	BIC	Bit clear
1111	MVN	Move negated

5.10 싱글 워드와 부호 없는 바이트 데이터 이동 명령어 형식

The pre-indexed form of the instruction:

```
LDR|STR{<cond>}{B} Rd, [Rn, <offset>]{!}
```

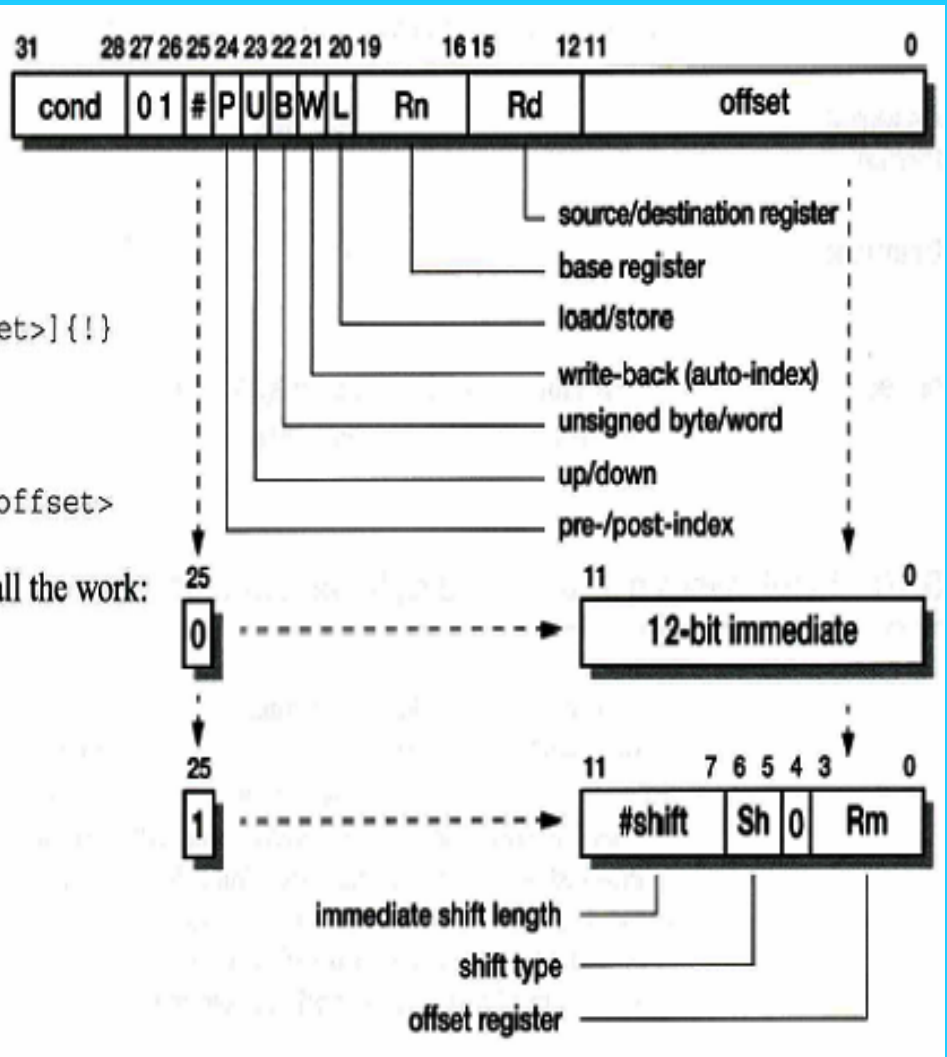
The post-indexed form:

```
LDR|STR{<cond>}{B}{T} Rd, [Rn], <offset>
```

A useful PC-relative form that leaves the assembler to do all the work:

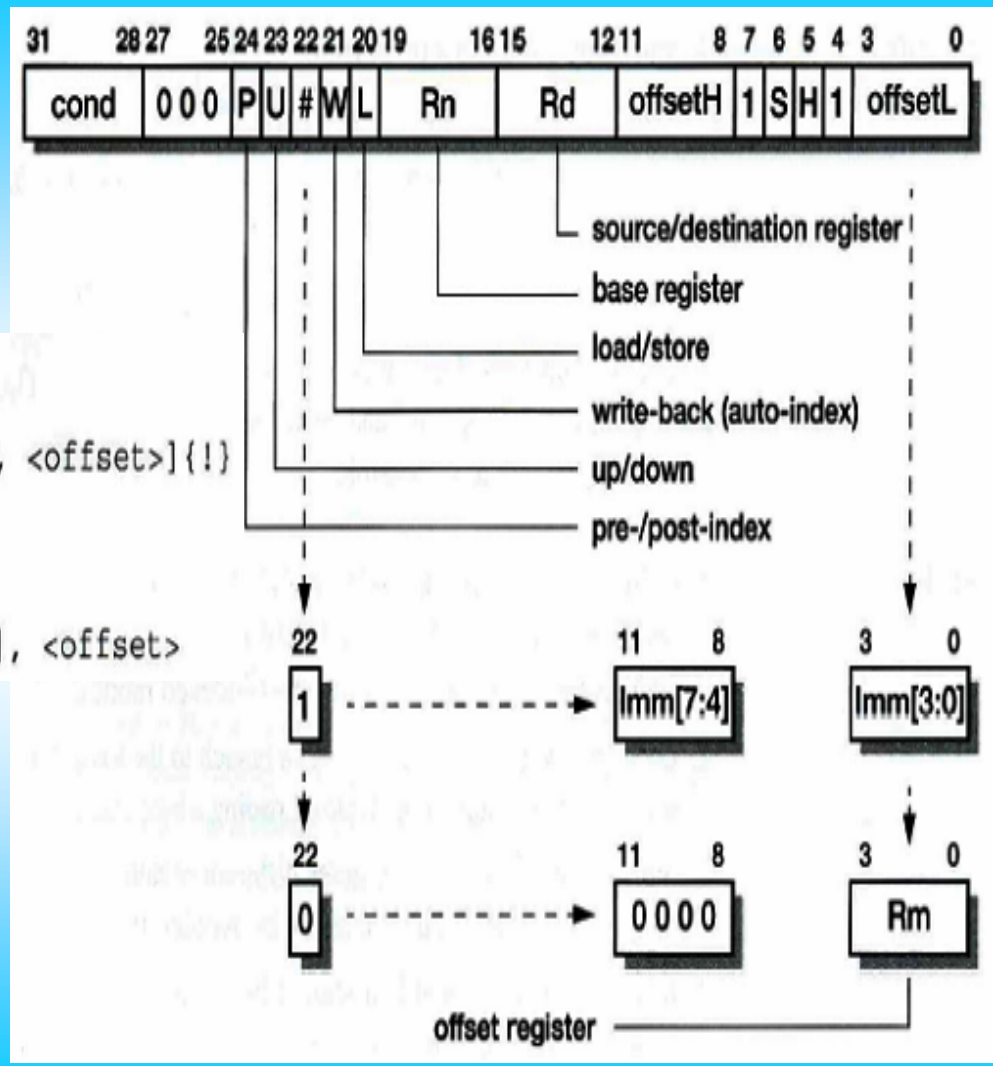
```
LDR|STR{<cond>}{B} Rd, LABEL
```

```
        LDR     r1, UARTADD
        STRB    r0, [r1]
        ..
UARTADD &       &1000000
```



5.11 하프 워드와 부호 있는 바이트 데이터 이동 명령어 형식

S	H	Data type
1	0	Signed byte
0	1	Unsigned half-word
1	1	Signed half-word



The pre-indexed form:

```
LDR|STR{<cond>}H|SH|SB Rd, [Rn, <offset>]{!}
```

The post-indexed form:

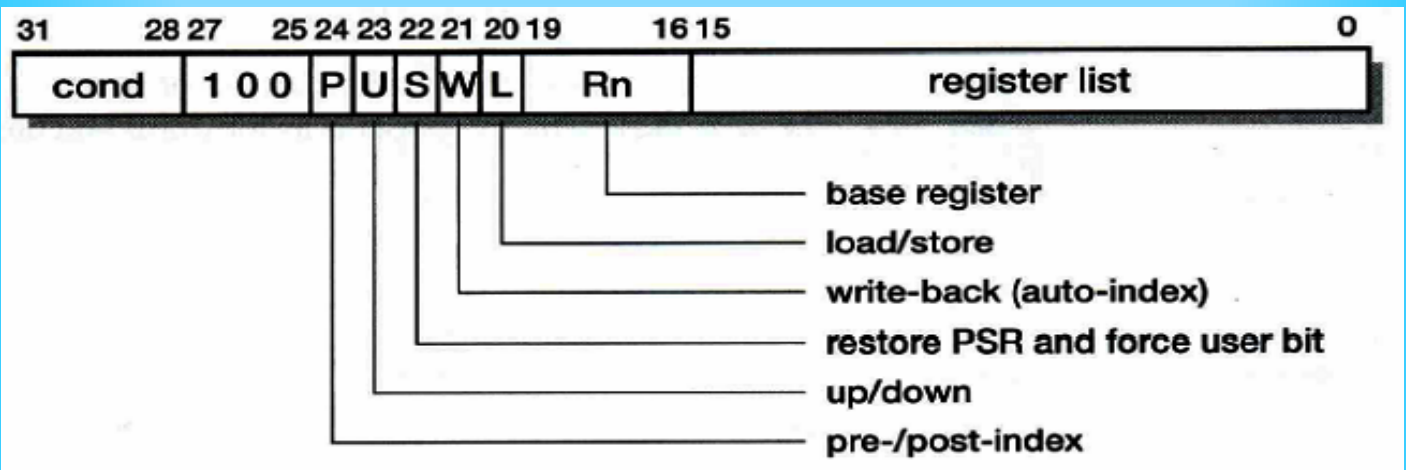
```
LDR|STR{<cond>}H|SH|SB Rd, [Rn], <offset>
```

```
        ADR     r1, ARRAY1
        ADR     r2, ARRAY2
        ADR     r3, ENDARR1
LOOP    LDRSH   r0, [r1], #2
        STR     r0, [r2], #4
        CMP     r1, r3
        BLT     LOOP
```

5.12 다중 레지스터 이동 명령어 형식

The normal form of the instruction is:

```
LDM|STM{<cond>}<add mode> Rn{!}, <registers>
```



In a non-user mode, the CPSR may be restored by:

```
LDM{<cond>}<add mode> Rn{!}, <registers + pc>^
```

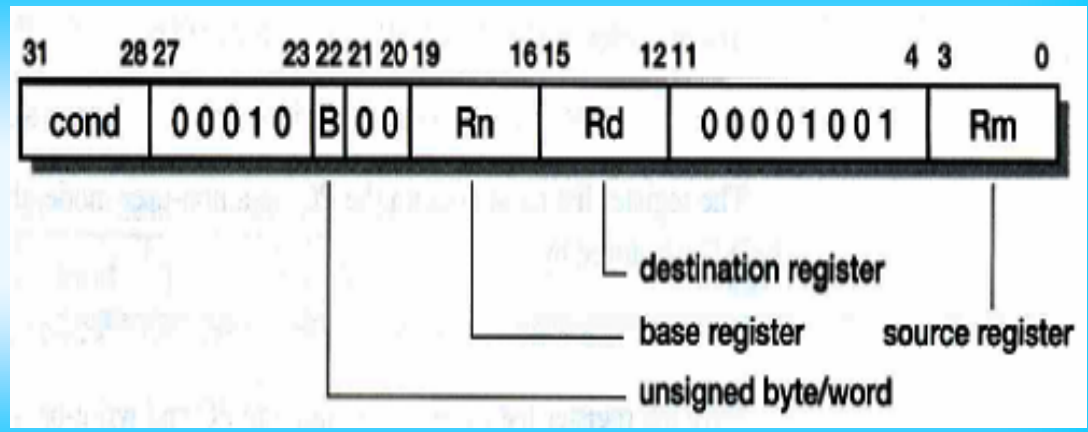
The register list must contain the PC. In a non-user mode, the user registers may be saved or restored by:

```
LDM|STM{<cond>}<add mode> Rn, <registers - pc>^
```

Here the register list must not contain the PC and write-back is not allowed.

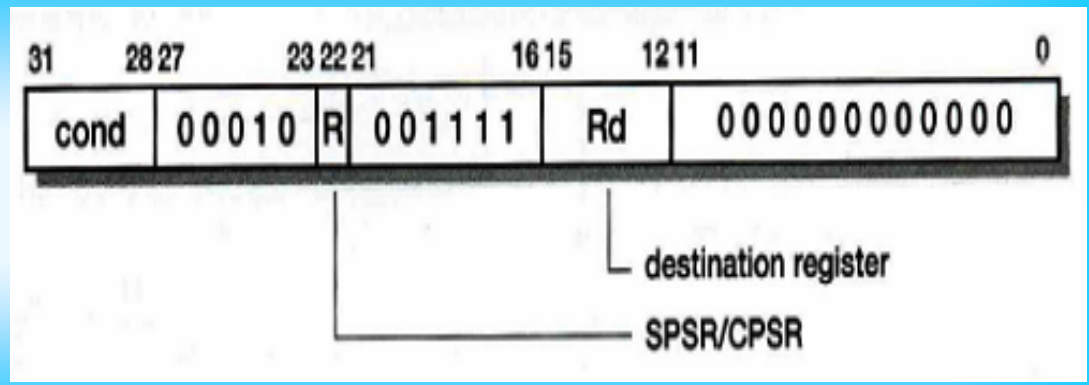
5.13 메모리와 레지스터 내용을 서로 교환하는 명령어 형식 (SWP)

```
SWP{<cond>}{B} Rd, Rm, [Rn]


ADR     r0, SEMAPHORE
SWPB    r1, r1, [r0]
```



5.14 상태 레지스터 내용을 일반 레지스터로 이동시키는 명령어 형식

```
MRS{<cond>} Rd, CPSR|SPSR


MRS     r0, CPSR
MRS     r3, SPSR
```

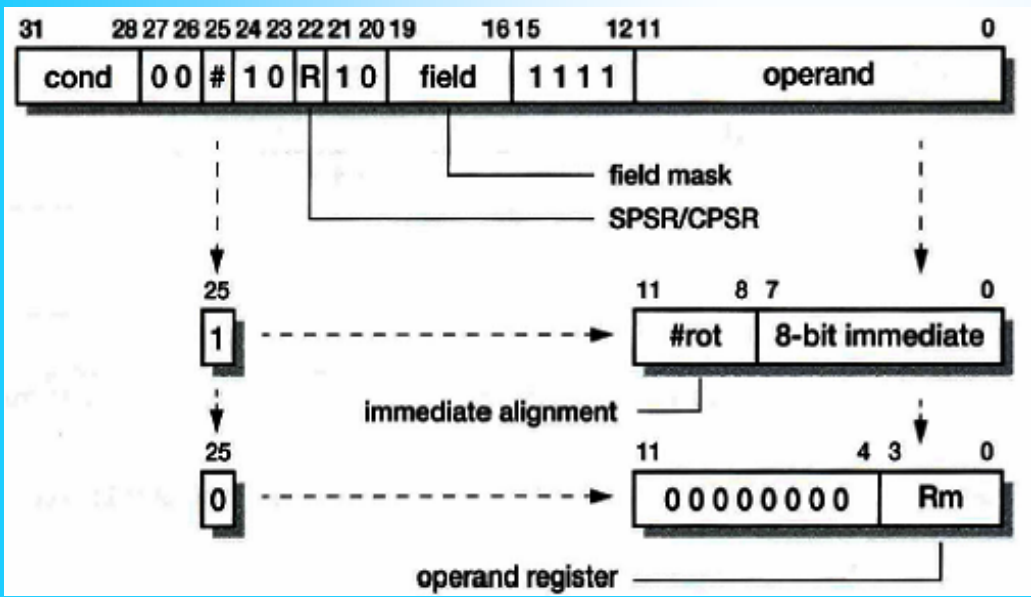


5.15 일반 레지스터 내용을 상태 레지스터로 이동시키는 명령어 형식

```
MSR{<cond>} CPSR_f|SPSR_f, #<32-bit immediate>
MSR{<cond>} CPSR_<field>|SPSR_<field>, Rm
```

where <field> is one of:

- c – the control field – PSR[7:0].
- x – the extension field – PSR[15:8] (unused on current ARMs).
- s – the status field – PSR[23:16] (unused on current ARMs).
- f – the flags field – PSR[31:24].




```
MSR     CPSR_f, #&f0000000;
```

```
MRS     r0, CPSR
ORR     r0, r0, #&20000000
MSR     CPSR_f, r0
```

To switch from supervisor mode into IRQ mode

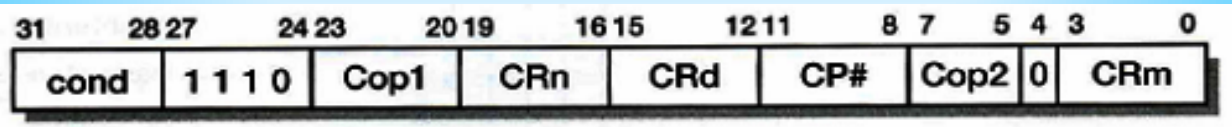
```
MRS     r0, CPSR
BIC     r0, r0, #&1f
ORR     r0, r0, #&12
MSR     CPSR_c, r0
```

5.16 Coprocessor 명령어

- ARM은 coprocessor를 추가하여 명령어 세트를 확장하는 메커니즘을 지원
  - ▷ 가장 일반적인 coprocessor는 캐쉬나 메모리 관리 장치 (MMU)와 같은 on-chip 함수들을 제어하기 위해 사용
  - ▷ 부동 소수점 연산이나 특별한 목적을 위해 coprocessor 사용 가능
- ARM coprocessor는 자체의 레지스터를 가지고 있고 ARM이 명령어 실행을 제어하기 때문에 coprocessor는 데이터 처리와 데이터 이동 명령어를 처리
- ARM coprocessor 주요 동작
  - ▷ 데이터 처리 : coprocessor 내부 레지스터의 내용을 이용하여 연산을 수행하고 결과를 coprocessor 내부 레지스터에 저장
  - ▷ 데이터 이동 : 메모리 접근을 위한 주소는 ARM이 발생하고 coprocessor는 데이터 이동의 수를 제어함 (최대 16 워드), 이동할 때 데이터 변환을 실행
  - ▷ 레지스터 이동 : ARM과 coprocessor 레지스터 사이 데이터를 이동함, coprocessor 비교 결과는 제어 흐름을 위해 ARM CPSR로 이동하여야 함

5.17 Coprocessor 데이터 처리 명령어 형식

```
CDP{<cond>} <CP#>, <Cop1>, CRd, CRn, CRm{, <Cop2>}
```




```
CDP     p2, 3, C0, C1, C2
CDPEQ   p3, 6, C1, C5, C7, 4
```

5.18 Coprocessor 데이터 이동 명령어 형식

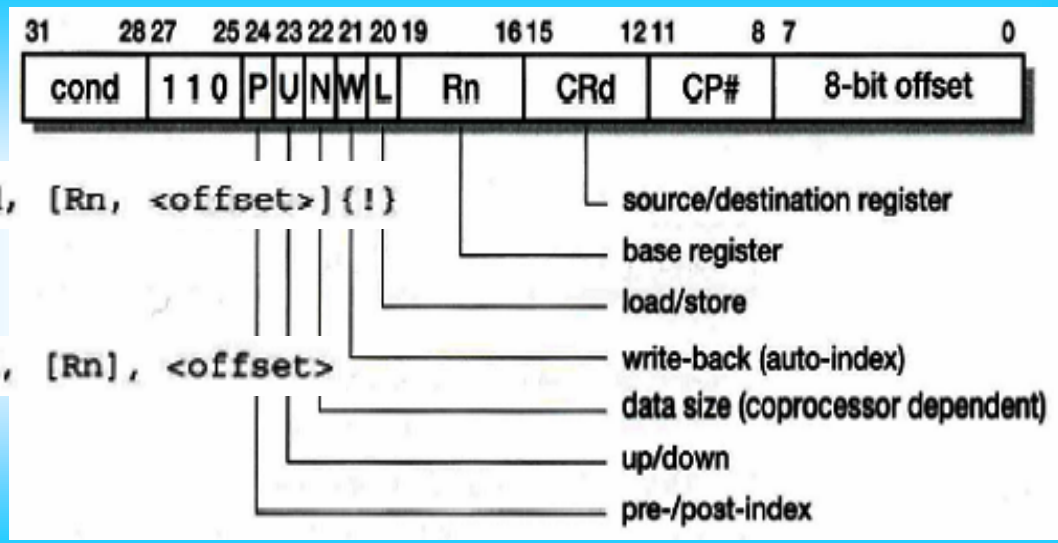
The pre-indexed form:

```
LDC|STC{<cond>}{L} <CP#>, CRd, [Rn, <offset>]{!}
```

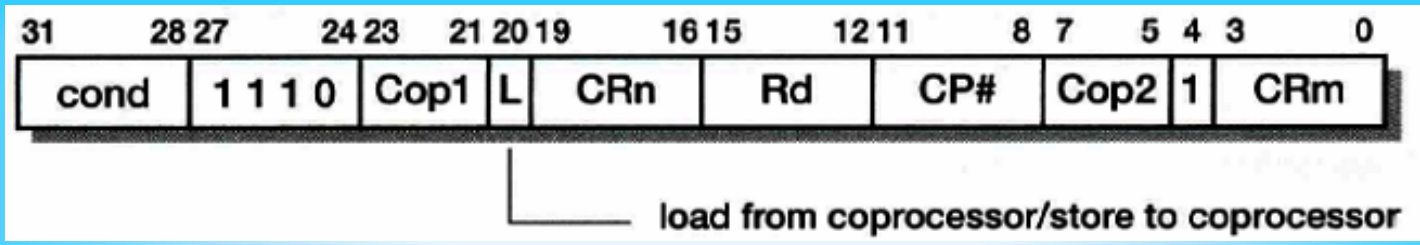
The post-indexed form:

```
LDC|STC{<cond>}{L} <CP#>, CRd, [Rn], <offset>
```

```
LDC     p6, C0, [r1]
STCEQL  p5, C1, [r0], #4
```



5.19 Coprocessor 레지스터 이동 명령어 형식



Move to ARM register from coprocessor:

```
MRC{<cond>} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}
```

Move to coprocessor from ARM register:

```
MCR{<cond>} <CP#>, <Cop1>, Rd, CRn, CRm{, <Cop2>}

MCR     p14, 3, r0, C1, C2
MRCCS   p2, 4, r3, C3, C4, 6
```

5.21 사용되지 않는 명령어 공간

- 사용되지 않는 명령어 코드 형식이나 정의되지 않은 opcode를 실행할 때 ARM은 undefined 명령어 트랩을 발생 (undefined 예외 모드 실행)

5.22 메모리 폴트(faults)

- 프로세서에 의한 메모리 접근이 정확하게 완료될 수 없는 경우 메모리 시스템은 메모리 폴트를 나타내는 신호를 프로세서에 보냄
- 프로세서는 메모리 폴트가 발생하면 abort 예외를 시작하고 시스템 소프트웨어로 문제 해결을 시도함
- 메모리 폴트가 발생하는 주요 원인
  - ▷ 페이지 부재 : 요청된 메모리 위치가 있는 페이지가 디스크에 있는 경우 시스템 소프트웨어는 요청된 페이지를 디스크로부터 메모리로 복사한 후에 폴트된 메모리 접근을 다시 시도
  - ▷ 페이지 보호 : 요청된 메모리 위치를 접근할 수 없는 경우, 단지 읽을 수만 있는 페이지에 쓰기 명령으로 내용을 변경하려고 할 때 폴트 발생
  - ▷ 메모리에서 에러가 검출된 경우, 에러 검출 블록만 있고 에러 정정 블록이 없는 경우는 더 이상 실행 불가능 (디스크를 사용하지 않고 메모리 용량이 작은 임베디드 시스템인 경우 메모리 폴트를 검출, 정정하는 블록이 없음)

- ARM은 명령어 fetch에서 발생한 메모리 폴트와 데이터 접근에서 발생하는 메모리 폴트를 구분하여 prefetch abort, data abort 예외를 처리함

5.23 ARM 구조 변형

- Version 3 : 32 비트 어드레싱, 독립적인 CPSR과 SPSR, coprocessor emulation 과 가상 메모리 지원을 위해 undefined와 abort 모드 추가, (ARM6, ARM600, ARM610, ARM7, ARM700, ARM710)
- Version 4 : signed and unsigned half-word와 signed byte 저장, 적재 명령어를 추가, 시스템 모드 도입 (StrongARM, ARM8, ARM810)
- Version 4T : 16 비트 Thumb 명령어 형식 도입 (ARM7TDMI, ARM720T, ARM740T, ARM9TDMI, ARM920T, ARM940T)
- Version 5T : BLX, CLZ, BRK 명령어를 지원
- Version 5TE : 신호처리 명령어 세트 확장 (ARM10TDMI, ARM1020E)