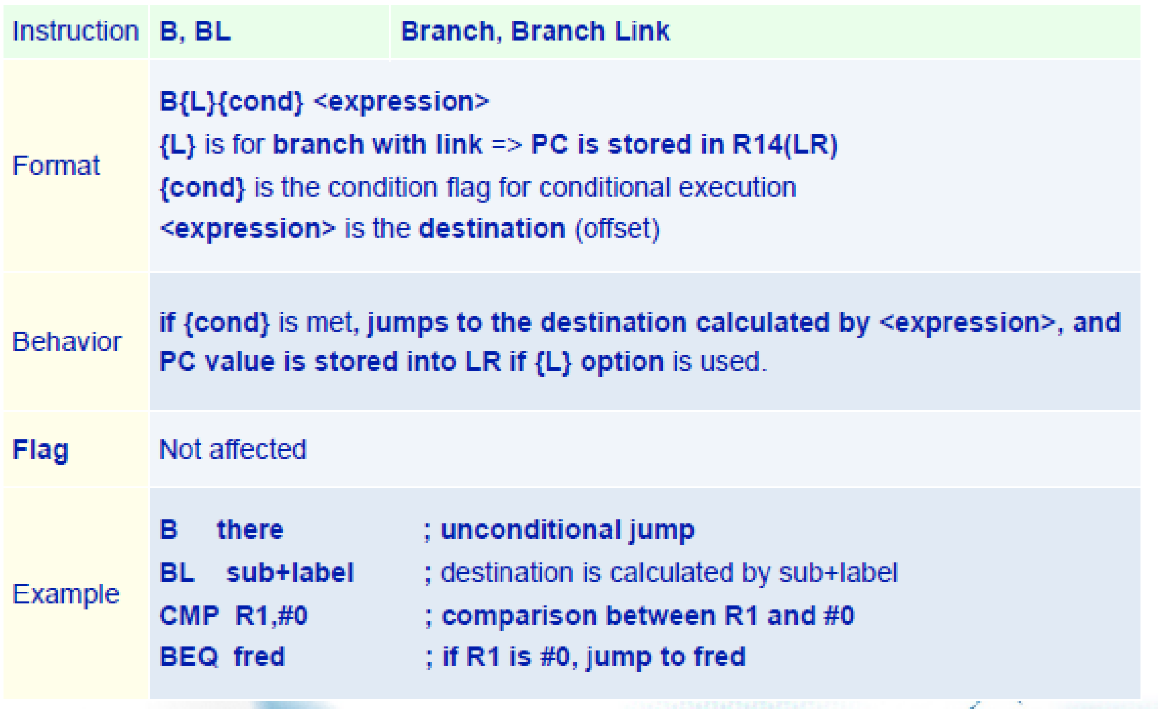
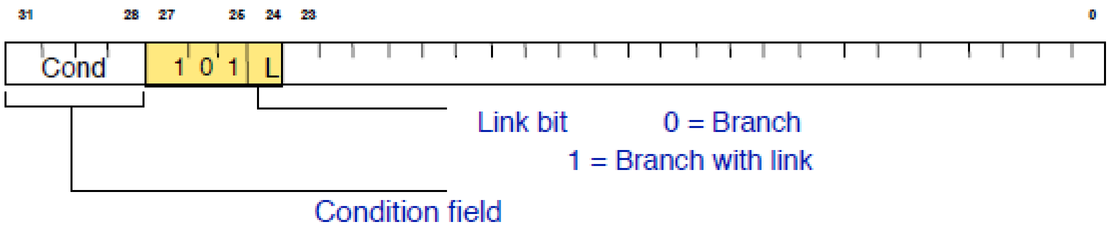
Branch Instructions



Instruction	B, BL	Branch, Branch Link
Format	B{L}{cond} <expression> {L} is for branch with link => PC is stored in R14(LR) {cond} is the condition flag for conditional execution <expression> is the destination (offset)	
Behavior	if {cond} is met, jumps to the destination calculated by <expression>, and PC value is stored into LR if {L} option is used.	
Flag	Not affected	
Example	B there BL sub+label CMP R1,#0 BEQ fred	; unconditional jump ; destination is calculated by sub+label ; comparison between R1 and #0 ; if R1 is #0, jump to fred

- BL : Branch with link(PC값을 LR에 저장하고 branch)
- cond : 조건부 실행
- expression : 코드의 위치를 변수같은 형태로 저장해놓은 것



- Destination address calculation
  - Bits [23:0] are used as offset using PC as the basis
  - Offset calculation
    - MSB is the sign bit (+ or -)
    - The remaining 23 bits are used as offset with 2 bits left shift operation
    - 16 MB의 range -> 부호비트 빼면 8 MB range
    - Word addressing의 관점으로 접근하면 32MB range(하위 2비트는 무조건 0인것을 이용)

실습

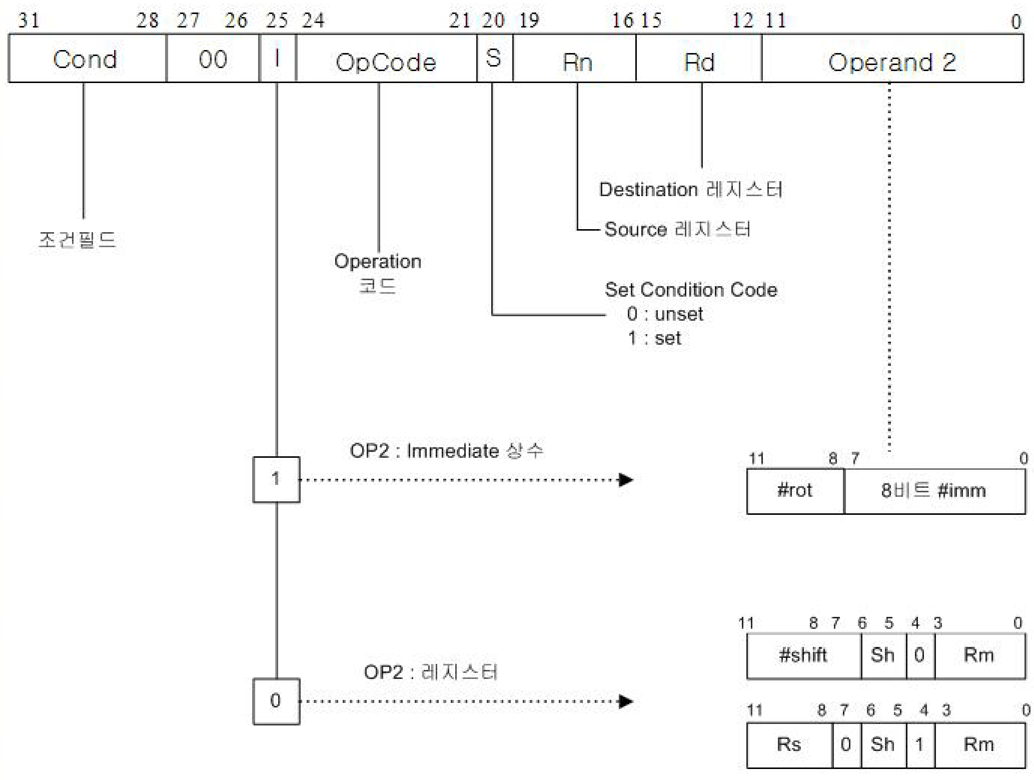
```
        mov     r0, #1      ; 0x00
        add     r1, r0, r0  ; 0x04
        bl      func        ; 0x08
        add     r1, r0, r0  ; 0x0C
        sub     r1, r0, r0  ; 0x10
loop    b       loop        ; 0x14
func    add     r1, r0, r0  ; 0x18
        mov     pc, lr      ; 0x1C
```

- 파이프라인 구조때문에 execution line 기준으로 fetch는 2라인 뒤, decode는 1라인 뒤에서 이루어지고 있다
- bl 실행시 돌아와서 다음으로 실행할 pc값을 lr에 저장
  - 브랜치 명령 실행시 파이프라인이 한번 깨짐

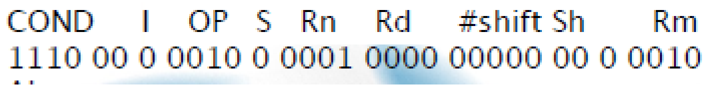
Subroutines

- Branch and Link (BL) instruction is used
  - Stores the next address into LR
  - LR <= PC - 4 : 파이프라이닝 때문에 현재 PC는 2칸앞에 있기때문에 -4 해주는 것
- Returning from subroutine
  - `MOV pc, lr`

Data Processing Instructions



- ADD r0, r1, r2


COND I OP S Rn Rd #shift Sh Rm
1110 00 0 0010 0 0001 0000 00000 00 0 0010

  - S : set condition code
  - Rn : src
  - Rd : dest

  - 11~0 비트 : i가 0이면 레지스터, i가 1이면 immediate
    - shift : 쉬프트해서 넣기
- AL
  - e0810002
  - 1110 0000 1000 0001 0000 0000 0000 0010

Arithmetic Operations

- Lists

opcode	inst	behavios	explanation
0100	ADD	Rd := Rn + Op2	Add
0101	ADC	Rd := Rn + Op2 + Carry	Add with Carry
0010	SUB	Rd := Rn – Op2	Subtract
0110	SBC	Rd := Rn – Op2 – 1 + Carry	Subtract with Carry
0011	RSB	Rd := Op2 - Rn	Reverse subtract
0111	RSC	Rd := Op2 – Rn – 1 + Carry	Reverse Subtract with Carry

- Examples
  - ADD r0, r1, r2
    - r0 = r1 + r2
  - SUBGT r3, r3, #1
    - r3 = r3 - 1 (이전 명령어의 결과가 Greater Than일시)
  - RSBLES r4, r5, #5
    - r4 = 5 - r5 (이전 명령어의 결과가 Less Equal일시), set condition code (S)

실습 : Carry 활용 64비트 연산

```
# 더하기
;        FF FFFFFFFF
;       +          1
;       -------------
;       100 00000000
ldr     r1, =0x000000ff  ; MSB of OP1
ldr     r0, =0xffffffff  ; LSB of OP1
ldr     r3, =0x00000000  ; MSB of OP2
ldr     r2, =0x00000001  ; LSB of OP2
ldr     r5, =0x0 ; MSB of destination register
ldr     r4, =0x0 ; LSB of destination register

adds    r4, r0, r2 ; s를 이용해 carry 저장
adc     r5, r1, r3 ; carry까지 포함한 연산

# 빼기
;       100 00000000
;       -           1
;       -------------
;       FF FFFFFFFF
ldr     r1, =0x00000100  ; MSB of OP1
ldr     r0, =0x0  ; LSB of OP1
ldr     r3, =0x00000000  ; MSB of OP2
ldr     r2, =0x00000001  ; LSB of OP2
ldr     r5, =0x0 ; MSB of destination register
ldr     r4, =0x0 ; LSB of destination register

subs    r4, r0, r2
sbc     r5, r1, r3
```

Logical Operations

- Lists

opcode	inst	behavior	explanation
0000	AND	Rd := Rn AND Op2	AND
1111	ORR	Rd := Rn OR Op2	OR
0001	EOR	Rd := Rn XOR Op2	Exclusive OR
1110	BIC	Rd := Rn AND (NOT Op2)	Bit Clear

  - 틀린부분있음 주의
- Examples
  - AND r0, r1, r2
  - BICEQ r2, r3, #7
    - BIC : AND의 반대개념이라고 보면됨
    - EQ
  - EORS r1, r3, r0
    - S

Comparison Operations

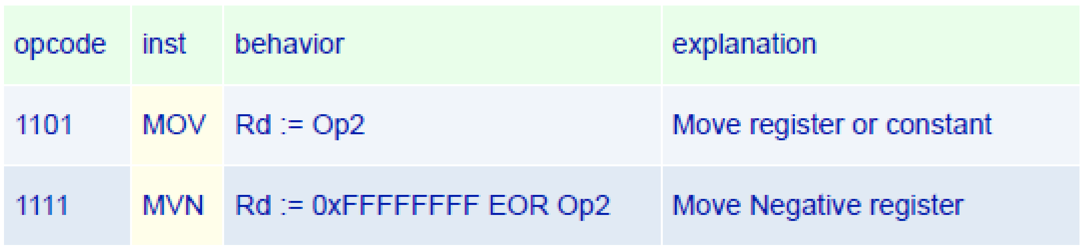
- No 'S' postfix to affect condition flags
- Lists

opcode	inst	behavior	expanation
1010	CMN	CPSR flags := Rn + Op2	Compare Negative
1011	CMP	CPSR flags := Rn-Op2	Compare
1000	TEQ	CPSR flags := Rn EOR Op2	Test bitwise equality
1001	TST	CPSR flags := Rn AND Op2	Test bits

- Examples
  - CMP r0, r1
  - TSTEQ r2, #5

Data Move Operations

- Lists



opcode	inst	behavior	explanation
1101	MOV	Rd := Op2	Move register or constant
1111	MVN	Rd := 0xFFFFFFFF EOR Op2	Move Negative register

- Examples
    - MOV r0, r1
        - r1 -> r0
    - MOVS r2, #10
        - 10 -> r2
    - MVNEQ r1, #0
        - if zero flag set then 0 -> r1
- ldr은 오래걸리는 연산, mov도 비슷한 기능 수행가능
    - 하지만, mov로 immediate값을 넣을려고 할 경우, 특정 조건을 만족하는 값들만 넣을 수 있음

Arithmetic Operation with Shift

- 값을 shift 해서 들어오게 할 수 있음
- Benefit
    - Arithmetic operation on the operands where shift operation to 2nd operand is applied in a single instruction
    - Barrel shifter is used
- Arithmetic operation with shift format
    - Immediate value based

      `ADD r5, r5, r3 LSL #3`

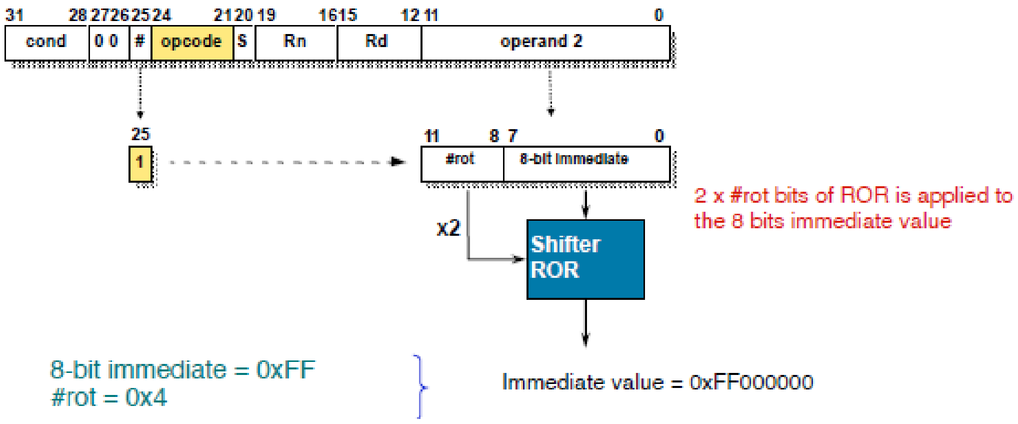
        - (LSL : Logical Shift Left)
    - Register value is used for shift bits

      `ADD r5, r5, r3 LSL r2`

- ASR(**Arithmetic** Shift Right) : The signed value of the contents of a register divided by a power of two. It copies the sign bit into vacated bit positions on the left
    - 대상을 숫자로 보고, **부호비트를 복사**해서 shift, 오른쪽으로shift할때만 중요
- LSL(=ASL) : The value of a register is **multiplied** by a power of two
    - 논리적 연산 : 새로 들어온 값은 무조건 0
- LSR : The unsigned value of a register is **divided** by a variable power of two
    - Both instructions(LSL, LSR) insert zeros into the vacated bit positions
- ROR(Rotate) : The value of the contents of a register rotated by a value
    - The bits that are **rotated off the right end** are inserted into the vacated bit **positions on the left**
    - ROL은 명령어들 표현을 위해 3비트를 쓰기 아까워서 안씀(어차피 ROR로 표현 가능)

Operand 2 and Immediate Value

- How we can use big immediate values in the limited space inside 32 bits instruction format?
    - A tricky method to represent immediate values using just 12 bits



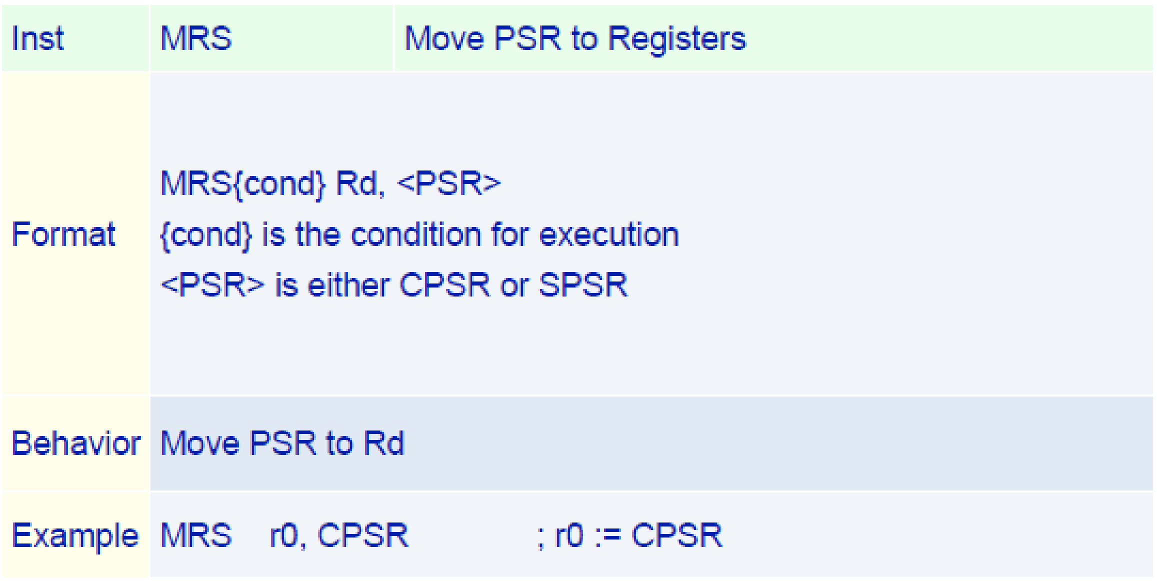
        - 8비트만 immediate value로 쓴다
        - 4비트는 #rot로 사용, #rot * 2 만큼 ROR한 value로 인식
            - 2의 배수 단위만 표현 가능

- Valid Immediate Values : 두칸단위(여덟비트)의 align 확인
    - `0x000000FF`
    - `0x00000FF0`
    - `0xFF000000`
    - `0xF000000F`
- Invalid Immediate Values
    - `0x000001FE1`
    - `0xF000F000`
    - `0x55550000`

PSR Transfer Instruction

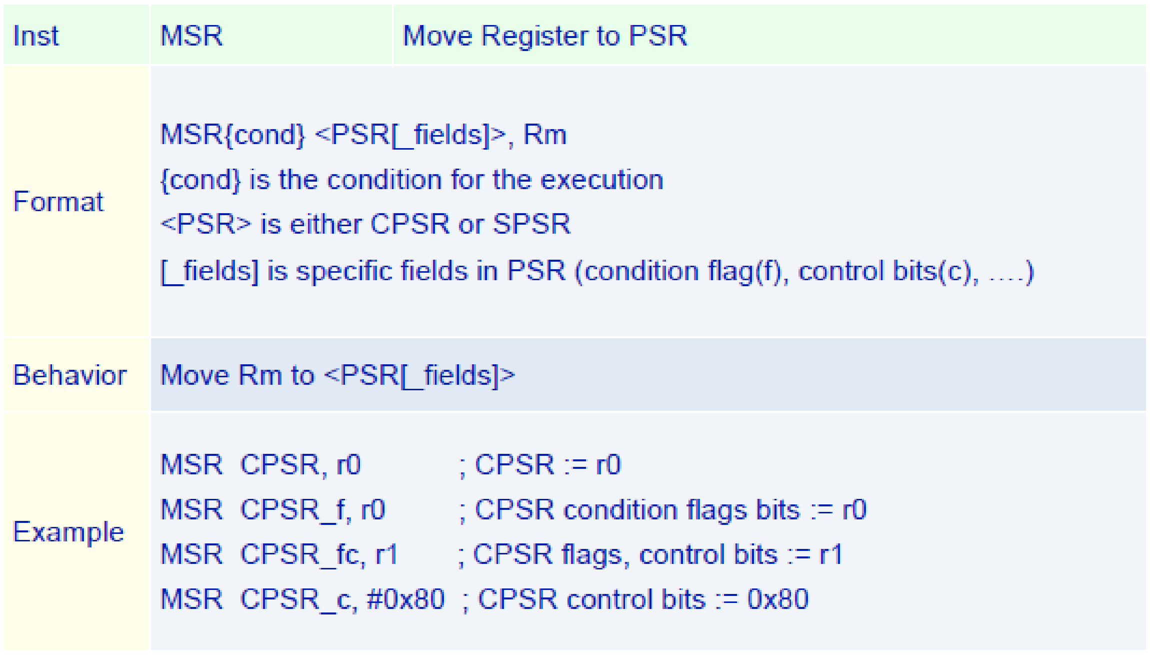
- PSR(Program Status Register) needs special treatment
    - The only possible way to access is through moving register contents between general purpose registers and PSRs
    - **MRS** : Move PSR to Register
    - **MSR** : Move Register to PSR
- CPSR(Current Program Status Register)
    - CPSR can be accessed at all the operation modes
    - Only **condition fields** can be modified at USER mode
- SPSR(Saved Program Status Register)
    - Each operation mode has a dedicated SPSR
        - user mode does not have SPSR

MRS



Inst	MRS	Move PSR to Registers
Format	MRS{cond} Rd, <PSR>	
	{cond} is the condition for execution	
	<PSR> is either CPSR or SPSR	
Behavior	Move PSR to Rd	
Example	MRS r0, CPSR ; r0 := CPSR	

MSR



Inst	MSR	Move Register to PSR
Format	MSR{cond} <PSR[_fields]>, Rm	
	{cond} is the condition for the execution	
	<PSR> is either CPSR or SPSR	
	[_fields] is specific fields in PSR (condition flag(f), control bits(c), ….)	
Behavior	Move Rm to <PSR[_fields]>	
Example	MSR CPSR, r0 ; CPSR := r0	
	MSR CPSR_f, r0 ; CPSR condition flags bits := r0	
	MSR CPSR_fc, r1 ; CPSR flags, control bits := r1	
	MSR CPSR_c, #0x80 ; CPSR control bits := 0x80	

- Controbl bits 중 i bit와 f bit가 중요(IRQ, FIQ 관련 비트)
- f(Condition flag bit), c(Control bits)