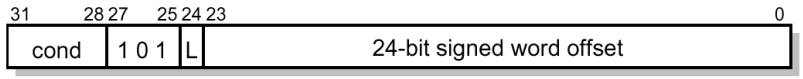
Lecture 8 ARM Instruction Set Architecture

- In this lecture, we will consider some aspects of ARM instruction set architecture (ISA) in some details.
- We shall consider the format of some instruction codes and their relationship with the assembly instruction.
- We starts with a simple Branch and Branch with Link instruction:



- Note that the top 4 bits [31:28] are always used to specify the conditions under which the instruction is executed.
- The L-bit (bit 24) is set if it is a branch with link instruction.
  - BL is jump to subroutine instruction - r14 <- return address
- 24-bit signed offset specifies destination of branch in 2's complement form. It is shifted left by 2 bits to form a word offset.
- The range of branch is +/- 32 Mbytes.

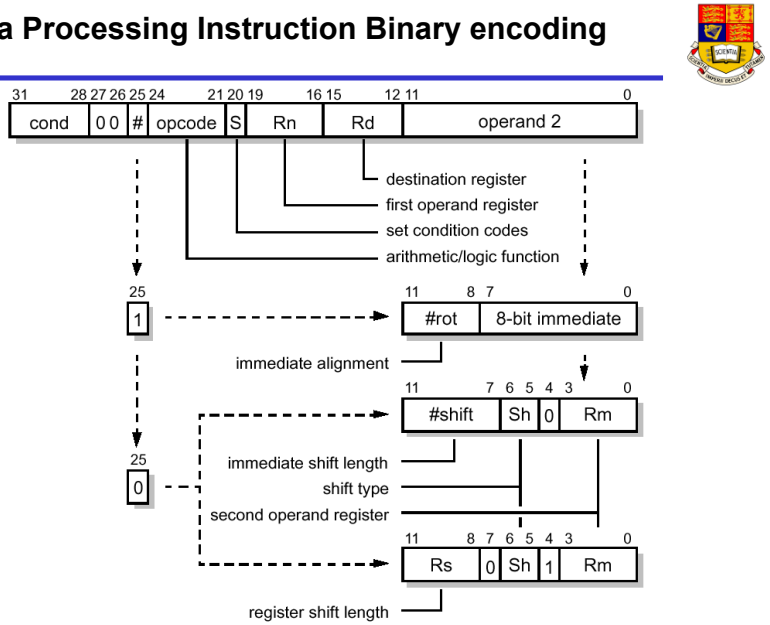
ARM condition codes fields

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher 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 set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

Data Processing Instructions

- Uses 3-address format: first operand - always register; second operand - register/shifted register/immediate value; result - always a register
- For second register operand, it can be logical/arithmetic/rotate. This is specified in "shift-type"
- How much to shift by is either a constant #shift or a register
- For immediate value second operand, only rotation is possible
- S-bit controls condition code update
  - N flag - set if result is negative (N equals bit 31 of result)
  - Z flag - set if result is zero
  - C flag - set if there is a carry-out from ALU during arithmetic operations, or set by shifter
  - V flag - set in an arithmetic operation if there is an overflow from bit 30 to bit 31. It is significant only when operands are viewed as 2's complement signed values

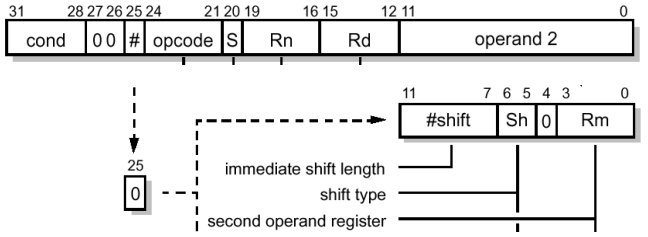
Data Processing Instruction Binary encoding



ARM data processing instructions

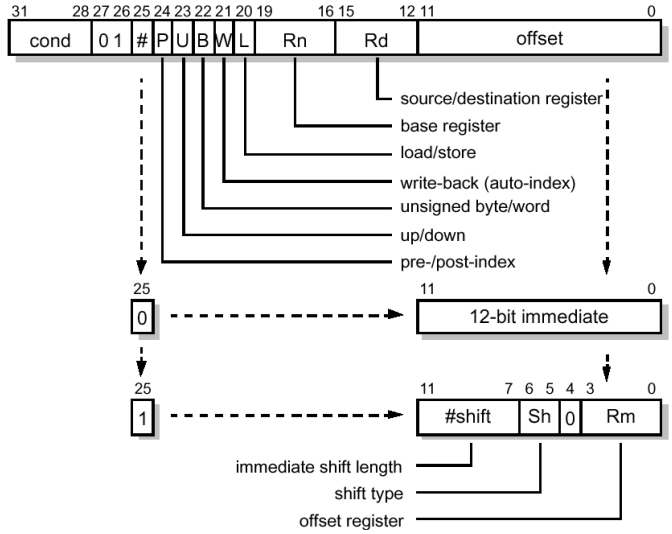
Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract 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 equivalence	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

Example of data processing instructions



- ADD r5, r1, r3 **E081 5003**
- ADDNE r0, r0, r0, LSL #2 **1090 0100**

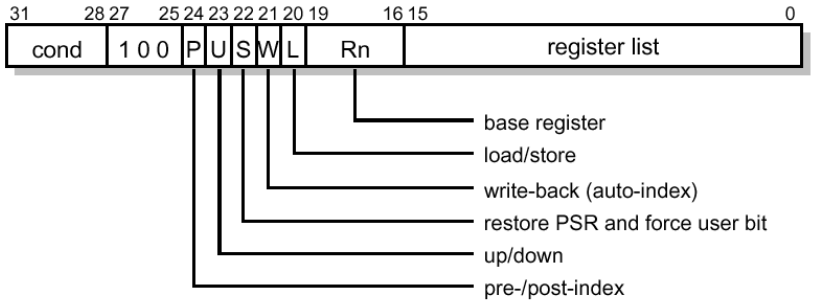
Data Transfer Instructions (LDR/STR)



Data Transfer instructions

- P = 1 means pre-indexed, i.e. modify the address BEFORE use
- P = 0 means post-indexed, i.e. modify the address AFTER use
- B = 1 selects unsigned byte transfer (default is word transfer)
- <offset> may be #+/- 12-bit immediate value (i.e. constant)
- <offset> may also be +/- register
- write-back (or "!") = 1 if the base register is updated
- All the shift parameters are the same as before

Multiple register transfer instructions



- STMIA r13!, {r0-r2, r14} **E8AD 4007**

Multiply Instructions

- ARM has a number of multiply instructions
  - Produce product of two 32-bit binary numbers held in registers.
  - Results of 32-bit*32-bit is 64 bits. Some ARM processors stores the entire 64-bit results in registers. Other ARM processors only stores the LOWER 32-bit products.
  - Multiply-Accumulate instruction also add product to accumulator value to form a running total.

Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit result)	Rd := (Rm * Rs) [31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd := (Rm * Rs + Rn) [31:0]
100	UMULL	Unsigned multiply long	RdHi:RdLo := Rm * Rs
101	UMLAL	Unsigned multiply-accumulate long	RdHi:RdLo += Rm * Rs
110	SMULL	Signed multiply long	RdHi:RdLo := Rm * Rs
111	SMLAL	Signed multiply-accumulate long	RdHi:RdLo += Rm * Rs

Example of using ARM Multiplier

- This calculates a scalar product of two vectors, 20 long.
- r8 and r9 points two the two vectors
- r11 is the loop counter
- r10 stores results

```
        MOV   r11, #20        ; initialize loop counter
        MOV   r10, #0         ; initialize total
LOOP    LDR   r0, [r8], #4    ; get first component
        LDR   r1, [r9], #4    ;   …. and second
        MLA   r10, r0, r1, r10 ; accumulate product
        SUBS  r11, r11, #1    ; decrement loop counter
        BNE   LOOP
```


Multiply with constants

- When multiplying by a constant value, it is possible to replace the general multiply with a fixed sequence of adds and subtracts that have the same effect.
- For instance, multiply by 5 could be achieved using a single instruction:

```
ADD  Rd, Rm, Rm, LSL #2        ; Rd = Rm + (Rm * 4) = Rm * 5
```

- This is obviously better than the MUL version:

```
MOV Rs, #5
MUL Rd, Rm, Rs
```

- What constant multiplication is this?

```
ADD   r0, r0, r0, LSL #2       ; r0' := 5 x r0
RSB   r0, r0, r0, LSL #3       ; r0'' := 7 x r0'
```