UNIT-3 : Types of Instructions

JNTUK
CSE
2-2
R16

Syllabus :

- Arithmetic and Logic Instructions
- Branch Instructions
- Input/Output Operations
- Addressing Modes

TextBook :

1. Computer Organization, Carl Hamacher, Zvonks Vranesic, Safea Zaky, 5th Edition, McGrawHill.

① Arithmetic and Logic Instructions :

(i) Arithmetic Instructions
(ii) Logic Instructions

(i) Arithmetic Instructions :

* ADD
* SUB
* MUL
* MLA

(*) ADD :

→ The basic Assembly Language expression for arithmetic instruction is

OPcode R_d, R_n, R_m

- where the operation specified by the OPcode is performed using the operands in general-purpose registers R_n and R_m.
- the result is placed in Register R_d

Example : ADD R_0, R_2, R_4 ($\because R_0 \leftarrow [R_2] + [R_4]$)

→ Instead of being contained in Register R_m, the second operand can be given in the immediate mode

Example : ADD $R_0, R_3, \#17$ ($\because R_0 \leftarrow [R_3] + 17$)

→ The second operand can be shifted (or) rotated before being used in the operation.

→ When a shift (or) rotation is required, the instruction is given as

ADD R0, R1, R5, LSL #4

- Here, the second operand, which is contained in Register R5, is shifted left 4 bit positions, and it is then added to the contents of Register R1
- the sum is placed in Register R0

(*) SUB :

SUB R0, R6, R5 (∵ R0 ← [R6] − [R5])

(*) MUL :

→ Two versions of a Multiply instruction are provided.

(I) The first version multiplies the contents of two registers and places the low-order 32-bits of the product in a third register
The high-order bits of the product, if there are any, are discarded.

Example :

MUL R0, R1, R2 (∵ R0 ← [R1] × [R2])

(II) The Second version specifies a fourth register whose contents are added to the product before storing the result in the destination register.

Example: MLA R0, R1, R2, R3 (∵ R0 ← [R1] × [R2] + [R3])

- This is called Multiply-Accumulate Operation
- It is used in numerical algorithms for digital signal processing.

(ii) LOGIC Instructions:

→ The different Logic Instructions are,

* AND
* OR
* XOR (exclusive OR)
* BIC (Bit-Clear)

→ Logic Instructions have the same format as the Arithmetic Instructions

AND R_d, R_n, R_m ($\because R_d \leftarrow [R_n] \wedge [R_m]$)

- which is a bitwise logical AND between the operands in Registers R_n and R_m

Example:

AND R_0, R_0, R_1 ($\because R_0 \leftarrow [R_0] \wedge [R_1]$)

- If register R_0 contains the hexadecimal pattern 02FA62CA and R_1 contains the pattern 0000FFFF, then the result 000062CA being placed in register R_0.

Examples:

(i) OR R_0, R_1, R_2 ($\because R_0 \leftarrow [R_1] \vee [R_2]$)

(ii) XOR R_0, R_1, R_2 ($\because R_0 \leftarrow [R_1] \oplus [R_2]$)

→ The BIC (Bit-Clear) instruction is closely related to the AND instruction
→ It complements each bit in operand R_m before ANDing them with the bits in register R_n

BIC R_0, R_0, R_1 ($\because R_0 \leftarrow [R_0] \wedge (\text{NOT}[R_1])$)

- If R_0 is 02FA62CA and R_1 is 0000FFFF, then the result is 02FA0000 being placed in R_0.

→ The MOVE NEGATION instruction, with the OP-code mnemonic MVN, Complements the bits of the source operand and places the result in R_d

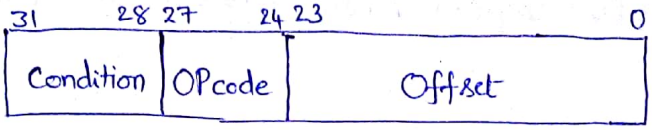
Example: MVN R_0, R_3

→ If the contents of R3 are the hexadecimal pattern 0F0F0F0F, then it places the result F0F0F0F0 in register R0

(2) Branch Instructions.

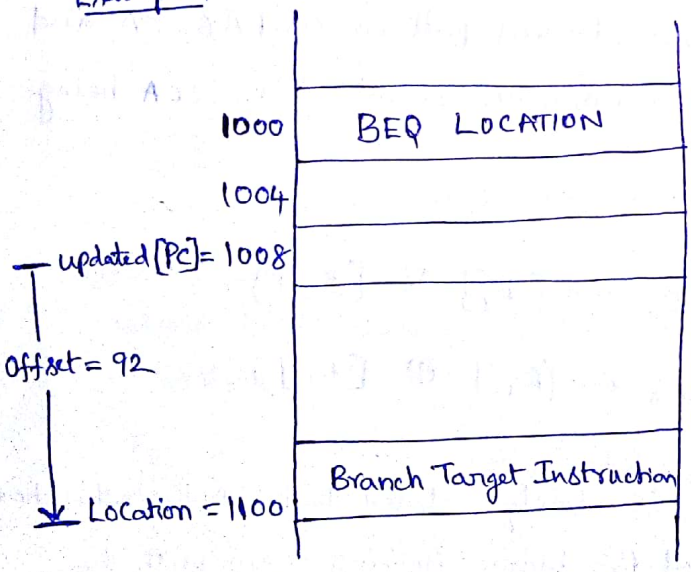
→ Conditional branch instructions contain a signed, 2's complement, 24-bit offset that is added to the updated contents of the program counter (PC) to generate the branch target address

→ The format for the branch instructions is depicted as,



Instruction Format

Example:



Determination of a Branch Target Instruction

→ The BEQ (Branch if Equal to 0) causes a branch if the Z flag is set to 1

→ The condition to be tested to determine whether (or) not branching should take place is specified in the high-order 4-bits, b_{31-28}, of the instruction word.

→ A Branch instruction is executed in the same way as any other ARM instruction, that is, it is executed only if the current state of the condition code flags corresponds to the condition specified in the condition field of the instruction

→ In Example, If the Branch instruction is at address location 1000, and the branch target address is 1100, then the offset has to be 92 because

the contents of the updated PC will be 1000+8 = 1008 when the address 1100 is computed.

(i) Setting Condition Codes:

→ Some instructions, such as COMPARE, given by

CMP Rn, Rm

- which performs the operation,

[Rn] - [Rm]

- have the sole purpose of setting the condition code flags based on the result of the subtract operation.

→ On the other hand, the arithmetic and Logic instructions affect the condition code flags only if explicitely specified to do so by a bit in the OP-code field.

→ This is indicated by appending the suffix S to the assembly language OP-code mnemonic

Example:

→ The instruction

ADDS R0, R1, R2

sets the condition code flags, but

ADD R0, R1, R2

does not.

③ I/O Operations:

→ The ARM architecture uses memory-mapped I/O

→ Reading a character from a keyboard (or) sending a character to a display can be done using program-controlled I/O

→ Suppose that bit 3 in each of the device status registers INSTATUS (Keyboard) and OUTSTATUS (display) contains the respective control flags SIN and SOUT

→ Also assume that the Keyboard DATAIN and display DATAOUT registers are located at addresses INSTATUS +4 and OUTSTATUS+4, immediately following the status register locations.

→ The read and write wait loops can then be implemented as,

Example:

- Assume that address INSTATUS has been located into Register R_1
- The instruction sequence given as,

```
READWAIT   LDR   R3, [R1]
           TST   R3, #8
           BEQ   READWAIT
           LDRB  R3, [R1, #4]
```

- It reads a character into register R3 when a key has been pressed on the keyboard
- The test (TST) instruction performs the bitwise logical AND operation on its two operands and sets the condition code flags based on the result.
- The immediate operand 8 has a single one in the bit 3 position.
- Therefore the result of the TST operation will be zero if bit 3 of INSTATUS is zero and will be nonzero if bit 3 is one, signifying that a character is available in DATAIN.
- The BEQ instruction branches back to READWAIT if the result is zero, looping until a key is pressed, which sets bit 3 of INSTATUS to One.

- Assuming that address OUTSTATUS has been loaded into register R2.

The instruction sequence is given as,

```
WRITEWAIT  LDR  R4, [R2]
           TST  R4, #8
           BEQ  WRITEWAIT
           STRB R3, [R2,#4]
```

- It sends the character in Register R3 to the DATAOUT register when the display is ready to receive it.

④ Addressing Modes:

→ The 68000 processor addressing modes is depicted in a table as,

Name	Assembler Syntax	Addressing Modes
Immediate	#Value	Operand = Value
Absolute Short	Value	EA = Sign Extended WValue
Absolute Long	Value	EA = Value
Register	Rn	EA = Rn (∴Operand = [Rn])
Register Indirect	(An)	EA = [An]
Indexed basic	WValue (An)	EA = WValue + [An]
Indexed Full	BValue (An, RK.S)	EA = BValue + [An] + [RK]
Relative basic	WValue (PC) (or) Label	EA = WValue + [PC]
Relative Full	BValue (PC, RK.S) (or) Label (RK)	EA = BValue + [PC] + [RK]
Auto increment	(An) +	EA = [An] Increment An ;
Autodecrement	−(An)	Decrement An EA = [An] ;

→ The 68000 processor have several addressing modes.

(1) Immediate Mode:

- The operand is contained in the instruction
- Four sizes of operands can be specified, Byte, Word, Long-word, OP-code word.
- The fourth size consists of very small numbers that can be

included directly in the OP-code word of some instructions.

(ii) Absolute Mode: (Direct Mode)

- The absolute address of an operand is given in the instruction, following the OP code.
- There are two versions of this mode - long and short.
- In the long mode, a 24-bit address is specified explicitly.
- In the short mode, a 16-bit value is given in the instruction to be used as the low-order 16-bits of an address.
- The sign bit of this value is extended to provide the high-order eight bits of the address.
- Since the sign bit is either 0 (or) 1, it follows that in the short mode only two pages of the addressable space can be accessed.
- These are the 0 page and the FF8 page, each containing 32K bytes.

(iii) Register Mode:

- The operand is in a processor register specified in the instruction.

(iv) Register Indirect mode:

- The effective address of the operand is in an address register specified in the instruction.

(v) Basic Index Mode:

- A 16-bit signed offset and an address register, An, are specified in the instruction
- The sum of this offset and the contents of An is the effective address of the operand.

(vi) Full Index Mode:

- An 8-bit signed offset, an address register An, and an index register R_K (either an address (or) a data register) are given in the instruction.
- The effective address of the operand is the sum of the offset and the contents of registers An and R_K
- Either all 32-bits (or) the sign-extended low-order 16 bits of R_K are used in the derivation of the address.

(vii) Basic Relative Mode:

→ This is same as the basic index mode except that the program Counter is used instead of an address register, An.

(viii) Full Relative Mode:

→ This is same as the full index mode except that the program Counter is used instead of an address register, An.

(ix) Autoincrement Mode:

- The effective address of the operand is in an address register, An, specified in the instruction
- After the operand is accessed, the contents of An are incremented by 1, 2, (or) 4, depending on whether a byte, a word, (or) a Long-word operand, respectively, is involved.

(x) Autodecrement Mode:

- The contents of an address register, An, Specified in the instruction are decremented by 1, 2, (or) 4, depending on whether a byte, a word, (or) a long-word operand, respectively, is involved.
- The effective address of the operand is the decremented contents of An.