Computer Architecture ECS-355

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

(Week 12) Lecture 23-24

Learning objectives:

- Review of the last lecture
- Types of addressing modes
- Types of instructions
- Data Transfer & Manipulation
- Program control instructions

Resources: Beside these lecture handouts, this lesson will draw from the following

Text Book: Computer System Architecture by Morris Mano (3rd Edition) and **Reference book: Computer Architecture, by William Stallings (4th Edition)**.

Lecture:

Introduction

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.

2. To reduce the number of bits in the addressing field of the instruction.

The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time. To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the computer. The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Execute the instruction.

Program counter (PC)

There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction, and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence. In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction. Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.

An example of an instruction format with a distinct addressing mode field is shown in Fig. below. The operation code specifies the operation to be performed. The mode field is used to locate the operands needed for the operation. There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register. Moreover, as discussed in the preceding section, the instruction may have more than one address field, and each address field may be associated with its own particular addressing mode. Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

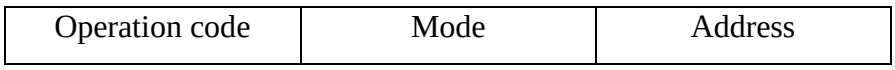


Figure: Instruction format with mode field.

Types of Addressing Modes

Implied Mode

In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

- Address of the operands are specified implicitly in the definition of the instruction
- No need to specify address in the instruction

Immediate Mode

In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value.

- Instead of specifying the address of the operand, operand itself is specified
- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

Register Mode

In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

- Address specified in the instruction is the register address
- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction

- Faster to acquire an operand than the memory addressing

Register Indirect Mode

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

- Instruction specifies a register which contains the memory address of the operand
- Saving instruction bits since register address is shorter than the memory address
- Slower to acquire an operand than both the register addressing or memory addressing

Autoincrement or Autodecrement Mode

This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access.

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 automatically.

Direct Address Mode

In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.

- Instruction specifies the memory address which can be used directly to the physical memory

- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory space

Indirect Addressing Mode

In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

- The address field of an instruction specifies the address of a memory location that contains the address of the operand
- When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory access

Relative Addressing Modes

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

- The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a designated register to calculate the address of the operand
- Address field of the instruction is short
- Large physical memory can be accessed with a small number of address bits

Three different Relative Addressing Modes depending on R;

- PC Relative Addressing Mode(R = PC)
    - EA = PC + IR(address)

- Indexed Addressing Mode(R = IX, where IX: Index Register)
    - EA = IX + IR(address)
- Base Register Addressing Mode(R = BAR, where BAR: Base Address Register)
    - EA = BAR + IR(address)

Instructions

Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks. The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields. The actual operations available in the instruction set are not very different from one computer to another. It so happens that the binary code assignments in the operation code field is different in different computers, even for the same operation. It may also happen that the symbolic name given to instructions in the assembly language notation is different in different computers, even for the same instruction. Nevertheless, there is a set of basic operations that most, if not all, computers include in their instruction repertoire. The basic set of operations available in a typical computer is the subject covered in this and the next section. Most computer instructions can be classified into three categories:

1. Data transfer instructions

2. Data manipulation instructions

3. Program control instructions

Data transfer instructions cause transfer of data from one location to another without changing the binary information content. Data manipulation instructions are those that perform arithmetic, logic, and shift operations. Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer. The instruction set of a particular computer determines the register transfer operations and control decisions that are available to the user.

Data Transfer Instructions

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

Data transfer instructions move data from one place in the computer to another without changing the data content. The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves. Table 8-5 gives a list of eight data transfer instructions used in many computers. Accompanying each instruction is a mnemonic symbol. It must be realized that different computers use different mnemonics for the same instruction name. The load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator. The store instruction designates a transfer from a processor register into memory. The move instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words. The exchange instruction swaps information between two registers or a register and a memory word. The input and output instructions transfer data among processor registers and input or output terminals. The push and pop instructions transfer data between processor registers and a memory stack.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instruction

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic instructions

2. Logical and bit manipulation instructions

3. Shift instructions

It must be realized, however, that each instruction when executed in the computer must go through the fetch phase to read its binary code value from memory. The operands must also be brought into processor registers according to the rules of the instruction addressing mode. The last step is to execute the instruction in the processor.

Arithmetic Instructions

The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions. The multiplication and division must then be generated by means of software subroutines. The four basic arithmetic operations are sufficient for formulating solutions to scientific problems when expressed in terms of numerical analysis methods.

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit Manipulation Instructions

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary-coded information. The logical instructions consider each bit of the operand separately and treat it as a Boolean variable. By proper application of the logical instructions it is possible to change bit values, to clear a group of bits, or to insert new bit values into operands stored in registers or memory words.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

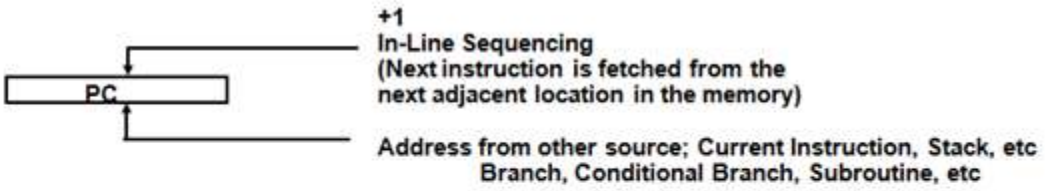
Instructions to shift the content of an operand are quite useful and are often provided in several variations. Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used. Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations. In either case the shift may be to the right or to the left.

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control Instructions

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed. Each time an instruction is fetched from memory, the program counter is incremented so that it contains the address of the next instruction in sequence. After the execution of a data transfer or data manipulation instruction,

control returns to the fetch cycle with the program counter containing the address of the instruction next in sequence. On the other hand, a program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered. In other words, program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data-processing operations. The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution. This is an important feature in digital computers, as it provides control over the flow of program execution and a capability for branching to different program segments.



Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test(by AND)	TST

* CMP and TST instructions do not retain their results of operations(- and AND, respectively). They only set or clear certain Flags.

Conditional Branch Instructions

Conditional Branch Instructions gives a list of the most common branch instructions. Each mnemonic is constructed with the letter B (for branch) and an abbreviation of the condition name. When the opposite condition state is used, the letter N (for no) is inserted to define the 0 state. Thus BC is Branch on Carry, and BNC is Branch on No Carry. If the stated condition is true, program control is transferred to the address specified by the instruction. If not, control continues with the instruction that follows. The conditional instructions can be associated also with the jump, skip, call, or return type of program control instructions.

Following table contains the list of conditional branch instructions.

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
	Signed compare conditions (A - B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Subroutine call and return

A subroutine is a self-contained sequence of instructions that performs a given computational task. During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program. Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a branch is made back to the main program. The instruction that transfers program control to a subroutine is known by different names. The most common names used are call subroutine, jump to subroutine, branch to subroutine, or branch and save address. A call subroutine instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two operations: (1) the address of the next instruction available in the program counter (the return address) is stored in a temporary location so the subroutine knows where to return, and (2) control is transferred to the beginning of the subroutine. The last instruction of every subroutine, commonly called return from subroutine, transfers the return address from the temporary location into the program counter. This results in

a transfer of program control to the instruction whose address was originally stored in the temporary location.

SUBROUTINE CALL

Call subroutine
Jump to subroutine
Branch to subroutine
Branch and save return address

Two Most Important Operations are Implied;

*** Branch to the beginning of the Subroutine**
- Same as the Branch or Conditional Branch

*** Save the Return Address to get the address of the location in the Calling Program upon exit from the Subroutine**
- Locations for storing Return Address
• Fixed Location in the subroutine(Memory)
• Fixed Location in memory
• In a processor Register
• In a memory stack
- most efficient way

CALL
SP ← SP - 1
M[SP] ← PC
PC ← EA

RTN
PC ← M[SP]
SP ← SP + 1