UNIT 3

Content

Objectives

After studying this unit, you will be able to:

- Explain the functions of CPU
- Explain the general register organization
- Describe stack organization
- Discuss instruction format
- Discuss addressing modes

Introduction

The Central Processing Unit (CPU) is the heart of a computer system. The CPU along with the memory and the I/O sub-systems develops a powerful computer system.

Companies such as AMD, IBM, Intel, Motorola, SGI, and Sun manufacture CPUs that are used in various kinds of computers such as desktops, mainframes, and supercomputers. A CPU comprises thin layers of thousands of transistors. Transistors are microscopic bits of material that block electricity at one voltage (non-conductor) and allow electricity to pass through them at different voltage (conductor). These tiny bits of materials are the semiconductors that take two electrical inputs and generate a different output when one or both inputs are switched on. As CPUs are small, they are also referred to as microprocessors.

An Overview of the CPU

Central Processing Unit (CPU) is the most important unit in a computer system. It is the component which controls all internal and external devices as well as performs arithmetic and logic operations to execute the set of instructions stored in the computer’s memory.

A CPU comprises three major components. They are:

- Register Set
- ALU
- Control Unit (CU)

Register Set

The register set differs from one system to another. The register set comprises many registers which include general purpose registers and special purpose registers. The general purpose registers do not perform any specific function. They store the temporary data that is required by a program. The special purpose registers perform specific functions for the CPU.

Example: Instruction Register (IR) is a special purpose register that stores the instruction that is currently being
executed.

ALU

The ALU performs all the arithmetic, logical, and shift operations by providing necessary circuitry that supports these computations.

Control Unit

The control unit fetches the instructions from the main memory, decodes the instructions, and then executes it. The control unit is discussed in detail in the units ahead.

As shown in figure the CPU consists of the register set, ALU, and CU.

The CPU interacts with the main memory and input/output devices. The CPU reads and writes data to and from the memory system and transfers data to and from the I/O devices.

A simple execution cycle in the CPU can be described as below:

1. The CPU fetches the instruction to be executed from the main memory and stores it in the Instruction Register (IR).
2. The instruction is decoded.
3. The operands are fetched from the memory system and stored in the CPU registers.
4. The instructions are then executed.
5. The results are transferred from the CPU registers to the memory system.

if there are more instructions to be executed, the execution cycle repeats. Any pending interrupts are also checked during the execution cycle.

Example: The interrupts such as I/O device request, arithmetic overflow, or pages is checked during the execution cycle.

The actions of the CPU are defined by the micro-orders issued by the control unit. The micro-orders are the control signals, which are sent over specified control lines.

Example: Suppose you need to execute an instruction that moves the contents of register A to register B. If both the registers are connected to data bus C, then the control unit issues a micro- order (control signal) to register A to place its contents on the data bus C. Another micro-order is sent to register B to read from data bus C. The control signals are activated either through hardwired control or microprogramming.

Figure : Components of CPU

Thus, CPU is the primary element of a computer system, which carries out each instruction of a program to perform basic arithmetical, logical, and input/output operations.

Figure 3.1 illustrates the components of the CPU.

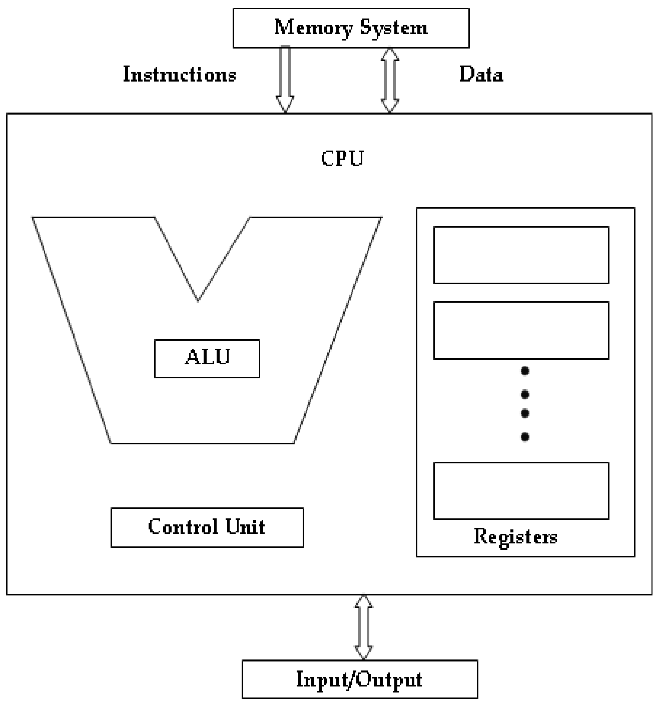


Figure : Components of CPU

General Register Organization

A group of flip-flops form a register. A register is a special high speed storage area in the CPU. They comprise combinational circuits that perform data processing. The data is always represented in a register before processing. The registers speed up the execution of programs.

Registers perform two important functions in the CPU operation. They are:

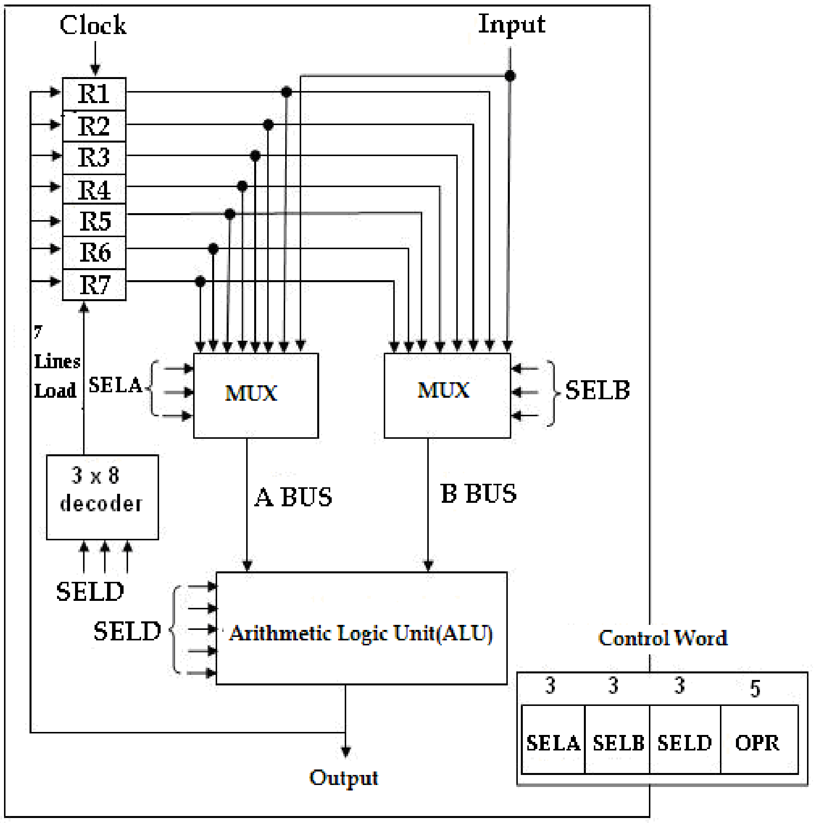
1.Providing a temporary storage area for data. This helps the currently executing programs to have a quick access to the data, if needed.

2.Storing the status of the CPU as well as information about the currently executing program.

Example: Address of the next program instruction, signals received from the external devices and error messages, and such other information is stored in the registers.

We know that referring to memory locations is considered difficult and time consuming. Hence,storing the pointers, return addresses, temporary results, and program counters into the register is more efficient than the memory. The number of registers varies from one computer system to another If a CPU contains a number of registers, then A Common bus is used to connect these registers. A general organization of seven CPU registers is shown in figure

Figure : General Organization of Registers



As observed in figure 3.2, the CPU bus system is operated by the control unit. The control unit directs the information flow through the ALU by selecting the function of the ALU as well as components of the system.

Consider R1+R2 + R3, the following are the functions performed within the CPU:

MUX A Selector (SELA): It is used to place R2 into bus A

MUX B Selector (SELB): It is used to place R3 into bus B

ALU Operation Selector (OPR): It selects the arithmetic addition (ADD)

Decoder Destination Selector (SELD): It transfers the result into R1.

The multiplexers of 3-state gates are implemented with the buses. The state of 14 binary selection inputs specifies the control word. The 14-bit control word specifies a micro-operation.

The encoding of register selection fields are specified in table 3.1.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Notes 3-state gates are the types of logic gates having three states of output: high (H), low (L) and high-impedance (Z).
Various micro-operations are performed by the ALU. Some of the operations performed by the ALU are listed in table 3.2.

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	ADD A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Comple ment A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Table 3.3: ALU Micro-Operations

Some of the ALU micro-operations are shown in the table 3.3:

Micro-operation	SELA	SELB	SE LD	OPR	Control Word			
R1 ° R2 – R3	R2	R3	R1	SUB	010	011	001	00101
R4 ° R4 R5	R4	R5	R4	OR	100	101	100	01010
R6 ° R6 + 1	R6	-	R6	INCA	110	000	110	00001
R7 ° R1	R7	-	R1	TSFA	001	000	111	00000
Output ° R2	R2	-	None	TSFA	010	000	000	00000
Output ° Input	Input	-	None	TSFA	000	000	000	00000
R4 ° shl R4	R4	-	R4	SHLA	100	000	100	11000
R5 ° 0	R5	R5	R5	XOR	101	101	101	01100

Types of Registers

There are many different types of register available in the market. Some of them are:

1. Data Register: It is used to store data.
2. Accumulator Register: It is considered as a special data register.
3. Address Register: It holds the memory address.
4. Index Register: It holds the index of the memory address.
5. Condition Register: It determines whether the instruction should be executed or not.
6. General Purpose Register: It stores data and addresses.
7. Special Purpose Register: It stores the status of the programs.
8. Floating Point Register: It is a kind of data register that stores the floating point numbers.
9. Constant Register: It stores read-only values.

General purpose registers are also called as processor registers. These processor registers provide the fastest means to access data.

Note: Processor register is mostly found at the top of the memory hierarchy.

Stack Organization

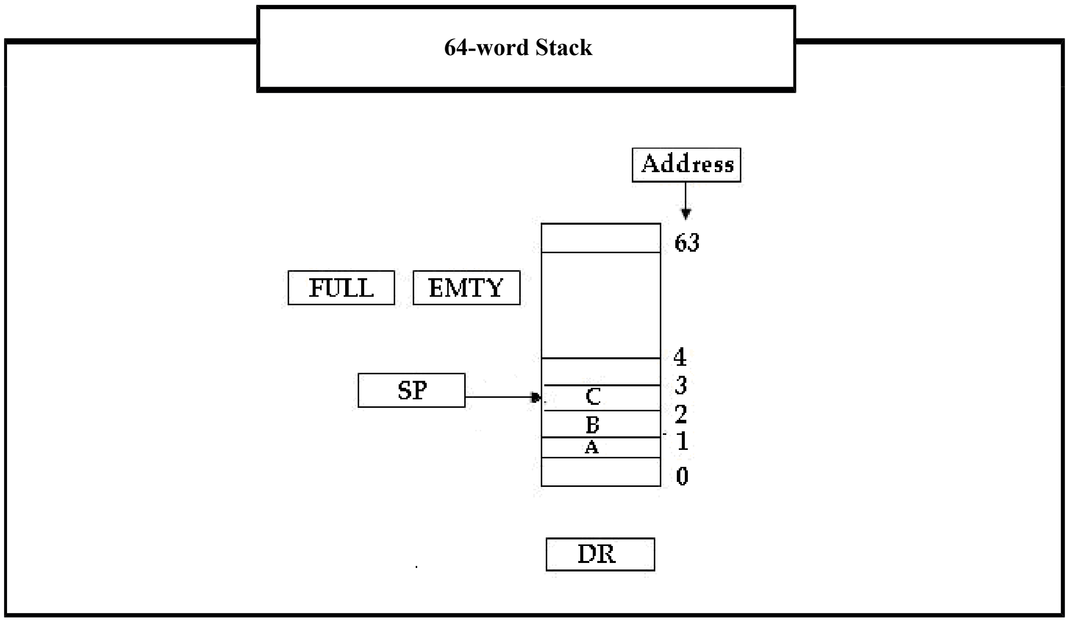
Stack, also called as Last In First Out (LIFO) list, is the most useful feature in the CPU. It stores information such that the item stored last is retrieved first. Stack is a memory unit with an address register. This register holds the address for the stack, which is called as Stack Pointer (SP). The stack pointer always holds the address of the item that is placed at the top of the stack.

You can insert an item into or delete an item from the stack. The insertion operation is called as push operation and the deletion operation is called as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register.

Register Stack

Stack can be organized as a collection of memory words or registers. Consider a 64-word register stack organized as shown in figure 3.3. The stack pointer register contains a binary number, which is the address of the item present at the top of the stack. The three items A, B, and C are placed in the stack. The item C is at the top of the stack and the stack pointer holds the address of C that is,

The top item is popped from the stack by reading memory word at address 3 and decrementing the stack pointer by 1. Now, B is at the top of the stack and the SP holds the address of B that is, 2. To insert a new word, the stack is pushed by incrementing the stack pointer by 1 and inserting a word in that incremented location.



Here, the stack pointer contains 6 bits, since 26 = 64, and the SP cannot exceed 63 (111111 in binary) because if 63 is incremented by 1, then the result is 0(111111 + 1= 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 then the result is 111111. Thus, when the stack is full, the one bit register 'FULL' is set to 1. If the stack is empty, then the one bit register 'EMTY' is set to 1. The data register DR holds the binary data which is written into or read out of the stack.

First the SP is set to 0, EMTY is set to 1, and FULL is set to 0. Now, as the stack is not full (FULL = 0), a new item is inserted using the push operation. The push operation is performed as below:

SP SP + 1, stack pointer is incremented

K[SP] DR, place an item on the top of the stack If (SP = 0) then (FULL 1), check if stack is full

EMPTY 0, if stack is full, then mark the stack as not empty

The stack pointer is incremented by 1 and the address of the next higher word is stored in the SP. The word from DR is inserted into the stack using the memory write operation. As per figure 5.3, the first item is stored at address 1 and the last item is stored at address 0. If the stack pointer is at 0, then the stack is full and 'FULL' is set to 1. This is the condition when the SP was in location 63 and after incrementing SP , the last item is stored at address 0. Once an item is stored at address 0, there are no more empty registers in the stack. The stack is full and the 'EMTY' is set to 0.

You can perform pop operation (deletion) only if the stack is not empty. To delete an item from the stack, the following micro-operations are performed.

DR K[SP], an item is read from the top of stack SP SP – 1, stack pointer is decremented

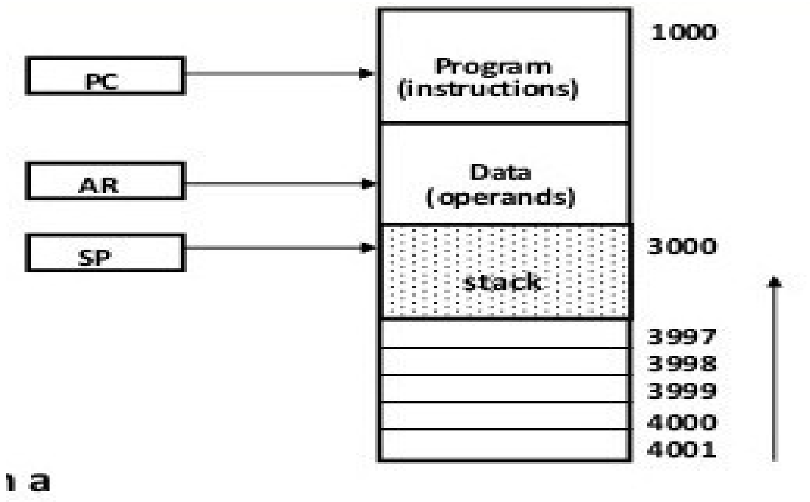
If (SP = 0) then (EMTY   1), check if stack is empty FULL 0, if stack empty, then mark the stack empty

The top item from the stack is read and sent to DR and then the stack pointer is decremented. If the stack pointer reaches 0, then the stack is empty and 'EMTY' is set to 1. This is the condition when the item in location 1 is read out and the SP is decremented by 1.

Memory Stack

Stack can be implemented in the CPU by allotting a portion of the computer memory to a stack operation and using a processor register as a stack pointer. In this case, it is implemented in a random access memory attached to the CPU.

In figure 3.4, a portion of the computer memory is divided into three segments: program, data and stack. The address of the next instruction in the program is stored in the pointer Program Counter (PC). The Address Register (AR) points to an array of the data. SP always holds the address of the item present at the top of the stack. The three registers that are connected to the common   bus are PC, AR, and SP. PC is used to read the instruction during fetch phase. An operand is read during execute phase using address register. An item is pushed into or popped from the stack using stack pointer. Figure 3.4 depicts the memory stack.



the SP points to an initial value '4001'. Here, the stack grows with decreasing addresses. The first item is stored at address 4000, the next item is stored at address 3999 and the last item is stored at address 3000.

As we already know, data register is used to read an item into or from the stack. You use push operation to insert a new item into the stack.

Notes To insert another item into the stack, the stack pointer is decremented by 1 so that it points at the address of the next location/word. A word from DR is inserted into the top of the stack using memory write operation.

To delete an item from the stack, you need to use the pop operation:

DR <- M[SP]

SP <- SP + 1

The top item is read into the DR and then the stack pointer is decremented to point to the next item in the stack.

Here, two processor registers are used to check the stack limits. One processor register holds the upper limit (3000) and the other holds the lower limit (4001). During push operation, the SP is compared with the upper limit to check if the stack is full. During pop operation, the SP is compared with the lower limit to check if the stack is empty.

An item in the stack is pushed or popped using two micro-operations. They are:

1. Accessing the memory through SP

2. Updating SP

A stack pointer is initially loaded with the bottom address of the stack in memory. Thereafter, SP is automatically incremented or decremented depending on the operation performed (push or pop). As the address is automatically updated in the stack pointer, the CPU can refer to the memory stack without specifying the address.

Instruction Format

An instruction consists of a combination of operation codes and operands that deal with the operation codes. Instruction format basically provides the layout of bits in an instruction. It includes fields such as opcode, operands, and addressing mode. The instruction length is usually kept in multiples of the character length, which is 8 bits. When the instruction length is fixed, a number of bits are allocated to opcode, operands, and addressing modes. The bits are distributed such that if more number of bits is allocated to the opcode field, then less number of bits are allocated to the operands and addressing. The task of allocating bits in the instruction can be simplified by considering the following factors:

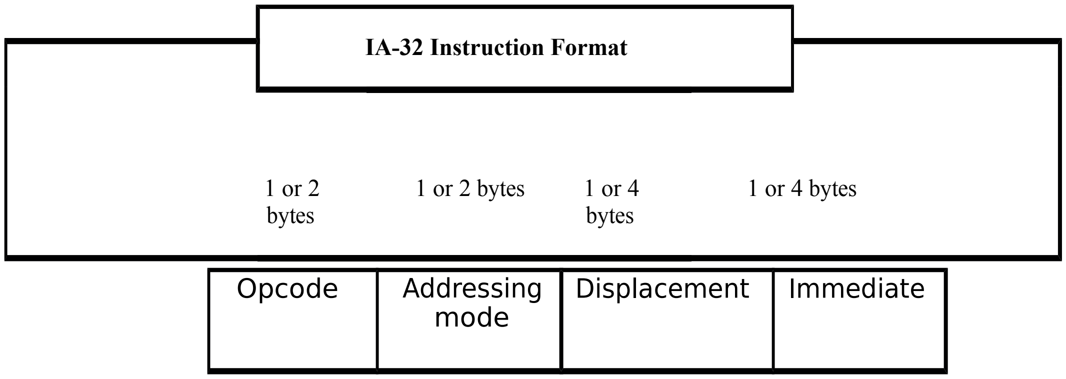
1. Number of addressing modes

2. Number of operands

3. Number of CPU registers

4. Number of register sets

Figure 3.5 shows the general IA-32 (Intel Architecture- 32 bits) instruction format. IA-32 is the instruction format that is used in Intel’s most successful microprocessors. This instruction format consists of four fields, namely opcode field, addressing mode field, displacement field, and immediate field.



As shown in figure the opcode field has 1 or 2 bytes. The addressing mode field also includes 1 or 2 bytes. In the addressing mode field, an instruction needs only one byte if it uses only one register to generate the effective address of an operand. The field that immediately follows the addressing mode field is the displacement field. If an effective address for a memory operand is calculated using the displacement value, then it uses either one or four bytes to encode. If an operand is an immediate value, then it is placed in the immediate field and it occupies either one or four bytes.

Instructions in a computer can be of different lengths with varying number of addresses. The number of address fields in the instruction format of a computer varies according to the organization of its registers. Based on the number of address fields the instruction can be classified as three address instructions, two address instructions, one address instruction, and zero address instruction.

Three Address Instructions

The general format of a three address instruction is represented as:

operation source 1, source 2, destination ADD A, B, C

where A, B and C are the three variables that are assigned to a distinct location in the memory. ‘ADD’ is the operation that is performed on the operands. ‘A’ and ‘B’ are the source operands and ‘C’ is the destination operand.

Here, bits are required to specify the three operands. n bit is required to specify one operand (one memory address). Similarly, 3n bits are required to specify three operands (three memory addresses). Bits are also required to specify the ADD operation.

Two Address Instructions

The general format of a two address instruction is represented as:

operation source, destination ADD A, B

where A and B are the two variables that are assigned to a distinct location in the memory. ‘ADD’ is the operation that is performed on the operands. This instruction adds the content of the variables A and B and stores the result in variable B. Here, ‘A’ is the source operand and ‘B’ is considered as both source and destination operands.

Here, bits are required to specify the two operands. n bit is required to specify one operand (one memory address). Similarly, 2n bits are required to specify two operands (two memory addresses). Bits are also required to specify the ADD operation.

One Address Instruction

The general format of one address instruction is represented as:

operation source

ADD A

where A is the variable that is assigned to a distinct location in the memory. ‘ADD’ is the operation that is performed on the operand A. This instruction adds the content of the variable A into the accumulator and stores the result in the accumulator by replacing the content of the accumulator.

Some more examples of one address instructions are:

LOAD A: The content of memory location A is stored in the accumulator. STORE B:

The content of accumulator is stored in the memory location B.

The operand in the instruction can either be the source or the destination, depending on the instruction.

Zero Address Instructions

The locations of the operands in zero address instructions are defined implicitly. These instructions store operands in a structure called pushdown stack.

We now know that one address instruction uses less number of bits, whereas three address instructions are uses more number of bits. Similarly, the three address instructions require more memory access when compared to one address instructions. Thus, three address instructions take more time to execute instructions when compared to one address instructions.

To reduce the execution time of the instructions, it is advised to refer the operands from the processor registers instead of referring the operands from the memory.

Summary

- A Central Processing unit (CPU) is the main unit of a computer system.
- There are three main components of CPU, namely register set, ALU, and control unit.
- Registers are the temporary storage, which are constructed from flip-flops. They store the status of the CPU.

- Stack is considered as a memory unit with an address register. It has a stack pointer, which always points at the top item in the stack.
- If stack is organized as a collection of registers, then stack is considered as register stack. If implemented in a random access memory attached to the CPU, then stack is considered as memory stack.
- Instruction format is the layout of bits in an instruction. An instruction includes fields such as opcode, operands, and addressing mode.

Addressing Modes

In almost all CPU designs, addressing modes are a part of the instruction set architecture. Various addressing modes are defined in a given instruction set architecture. These addressing modes describe the procedure by which language instructions in instruction set architecture identify the operands of each instruction. We can specify how to calculate the effective memory address of an operand by using addressing modes. This is done by using information held in registers and/or constants contained within a machine instruction or elsewhere.

RISC and CISC are the two most commonly used instruction sets, which are discussed in this unit.

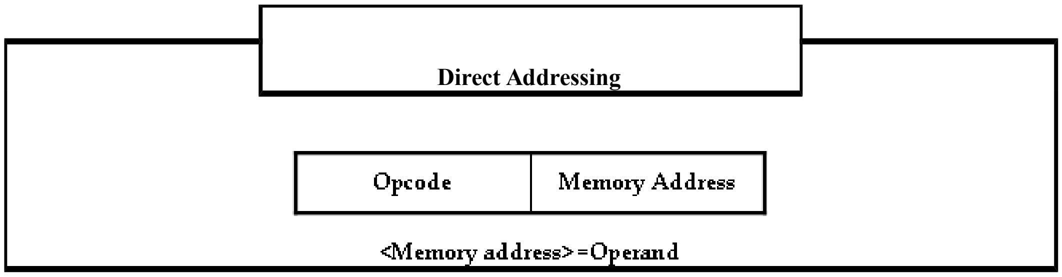
Need for Addressing Modes

The operands of the instructions can be located either in the main memory or the CPU registers. If the operand is placed in the main memory, then the instruction provides the location address in the operand field. Many methods are followed to specify the operand address. The different methods/modes for specifying the operand address in the instructions are known as addressing modes. The exact addressing mode used in the instruction can be specified to the control unit by using any of the following two methods:

1. The opcode explicitly specifies the addressing mode in the instruction.
2. A separate addressing mode field is specified in the instruction.

Some of the most common addressing modes used by the computers are:

1. Direct Addressing Mode
2. Indirect Addressing Mode
3. Register Addressing Mode
4. Immediate Addressing Mode
5. Index Addressing Mode
6. Direct Addressing Mode



In direct addressing mode, the operand address is explicitly specified in the instruction. This mode can be illustrated using the below assembly language statements:

LOAD R1, A   //The content of memory location A is loaded into register R1 MOV B, A   //The content of memory location A is moved to memory location B

JUMP A   //The program control is transferred to the instruction at memory location A

In direct addressing mode, the operand address is directly available in the instruction. Hence, eliminating the operand address calculation step the instruction cycle time decreases. However, the operand field in the instruction limits the number of bits for the operand address.

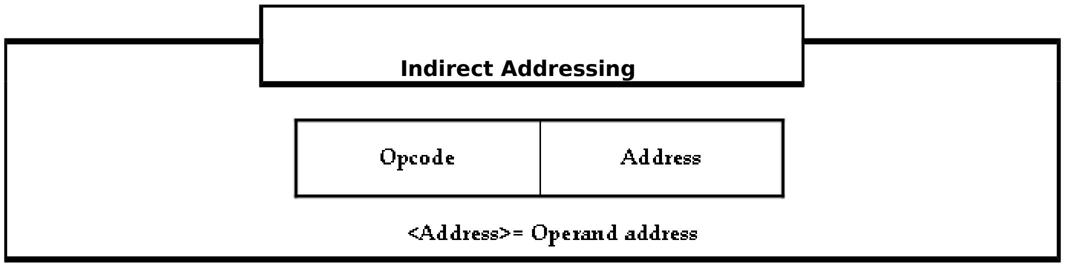
Indirect Addressing Mode

In indirect addressing mode, the address of the location ‘A’ contains address of another location ‘B’, which actually holds the operand. It is represented as below:

A = B, B = operand

Thus, ‘A’ is considered as the pointer. By modifying the content of location ‘A’, you can change the value of ‘B’, without changing the instruction. The below assembly language statement illustrates this mode.

MOVE (A), R1 //The content present in the address A is loaded into the register R1.The figure 8.3 illustrates indirect addressing mode.



The indirect addressing mode provides flexibility in programming. You can change the address during program run-time without altering the contents of the instruction. However, as there are two memory accesses even for the single level indirect addressing, the instruction cycle time increases.

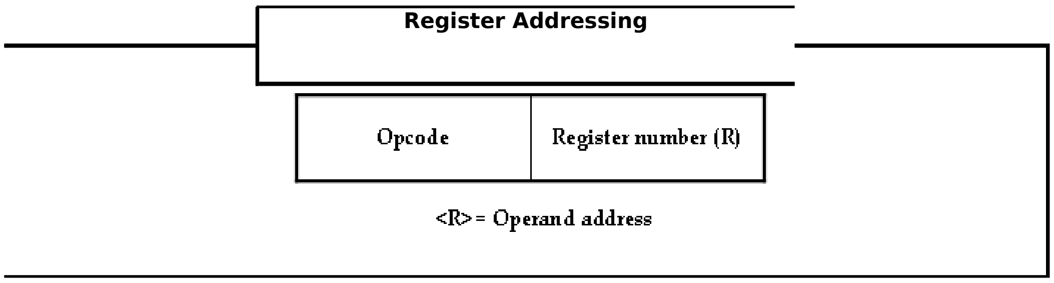
Register Addressing Mode

In register addressing mode, the register holds the operand. In the instruction, the register number that holds the operand is specified. The long programs find this mode useful as it helps to store the intermediate results in the registers. The following assembly language statements illustrate this mode:

ADD R1, R2 //The contents of the registers R1 and R2 are added and the result of the addition is stored in register R1.

STORE R1, M1 //The contents of the register R1 are stored in memory address M1.

Here, the first operand uses register addressing mode and the second operand uses direct addressing mode. The figure depicts the register addressing mode



The register addressing mode provides faster operand fetch without memory access. However, the number of registers is limited. Hence, the programmers must effectively utilize the registers.

Immediate Addressing Mode

In immediate addressing mode, the operand is a part of the instruction. Hence, memory reference is not required to retrieve the operand. This mode is used to define constants and set initial values to the variable. The below assembly language statements illustrate the immediate addressing mode:

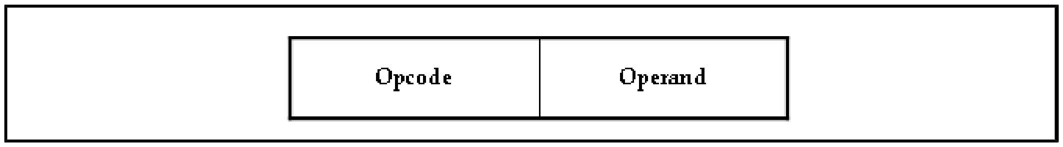
MOVE #14, R1 or MVI R1, 14 //The binary equivalent of 14 is loaded in the register R1

ADD #14, R1 //The binary equivalent of 14 and the contents of R1 are added and the result is stored in register R1

CMP #14, R1 //The binary equivalent if 14 is compared with the contents of R1.

The ‘#’ sign indicates that the constant following the sign is the immediate operand.

The figure 8.5 depicts the immediate addressing mode.



Once the instruction is fetched, the operand is also fetched in the instruction. This reduces the instruction cycle time. However, the value of the operand is limited because this mode is limited   to the size of the address field.

Index Addressing Mode

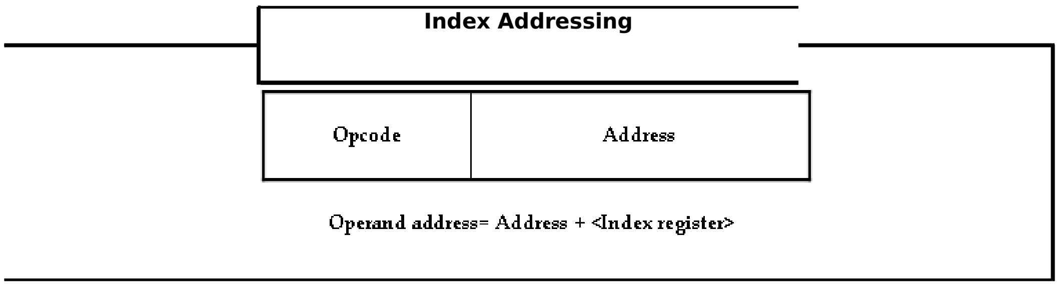
depicts index addressing mode.

The index addressing mode includes an index register which holds an offset/displacement. The effective address of the operand is obtained by adding the offset with the contents of the registers present in the instruction.

The start address of an array in the memory is obtained from the address field in the instruction. The difference between the start address and the operand address provides an index value for the operand. The index value is stored in the index register. The operands are stored in consecutive locations in the memory. Thus, by changing the value of the index or by incrementing the index value, you can access any operand in the array.

Some CPUs possess auto-indexing feature, which automatically increments the index registers when an instruction with index addressing is executed.

These addressing modes provide flexibility in writing efficient programs. Addressing modes help in reducing the instruction length by including a separate field for the address. This helps the programmers to handle complex tasks such as loop control, program relocations, and indexing of an array.


Index Addressing
Opcode
Address
Operand address= Address + <Index register>