UNIT 1 NOTES

Digital Computer:

The digital computer is a digital system that performs various computational tasks. Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.

A computer system is sometimes subdivided into two functional entities: hardware and software. The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device. Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks. A sequence of instructions for the computer is called a program.

The hardware of the computer is usually divided into three major parts, as shown in Fig. 1-1. The central processing unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions. The memory of a computer contains storage for instructions and data. It is called a random· access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time. The input and output processor (lOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world. The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

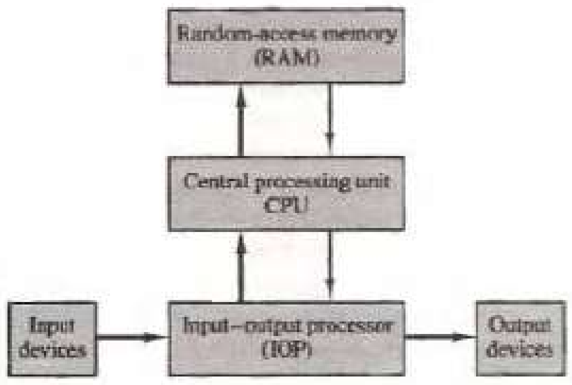

Random-access memory (RAM)
Central processing unit CPU
Input devices
Input–output processor (IOP)
Output devices

Figure 1-1 Block diagram of a digital computer.

Functional Unit of Digital Computer:

A computer consists of five functionally independent main parts: input, memory, arithmetic & logic, output, control unit.

The input unit accepts coded information from human operator from an electromechanical device such as key board or from other computers over digital communication lines.

The information is either stored in computer's memory for future reference or immediately used by ALU to perform desired operation.

The processing steps are determined by a program stored in memory.

Finally the results are sent back to the outside world through the output unit.

All of these operations are coordinated by the control unit.

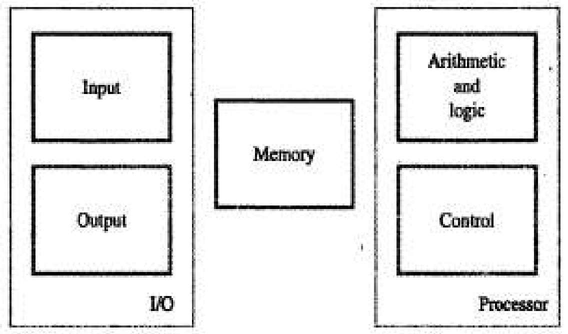


Figure 1.1 Basic functional units of a computer.

Computer Organization: Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organization structure to verify that the computer parts operate as intended.

Computer organization refers to the operational units and their interconnections that realize the architectural specifications.
Example: Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.
• Computer Organization:
– Design of the components and functional blocks using which computer systems are built.

– *Analogy*: civil engineer's task during building construction (cement, bricks, iron rods, and other building materials).

Computer Architecture: Computer architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.
Example: Architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

• Computer Architecture:
– How to integrate the components to build a computer system to achieve a desired level of performance.
– *Analogy*: architect's task during the planning of a building (overall layout, floorplan, etc.).

These two processor architectures can be classified by how they use memory.

Von-Neumann architecture:
In a Von-Neumann architecture, the same memory and bus are used to store both data and instructions that run the program. Since you cannot access program memory and data memory simultaneously, the Von Neumann architecture is susceptible to bottlenecks and system performance is affected.

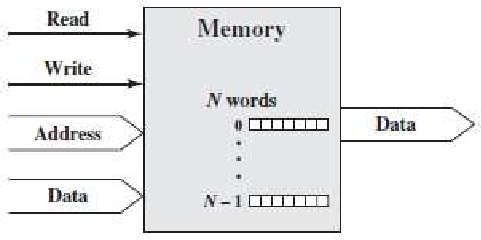
Harvard Architecture:

The Harvard architecture stores machine instructions and data in separate memory units that are connected by different busses. In this case, there are at least two memory address spaces to work with, so there is a memory register for machine instructions and another memory register for data. Computers designed with the Harvard architecture are able to run a program and access data independently and therefore simultaneously. Harvard architecture has a strict separation between data and code. Thus, Harvard architecture is more complicated but separate pipelines remove the bottleneck that Von Neumann creates.

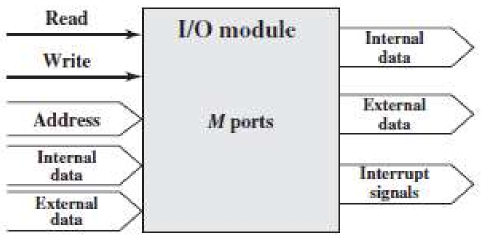
Interconnection Structure:

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules. The collection of paths connecting the various modules is called the interconnection structure.

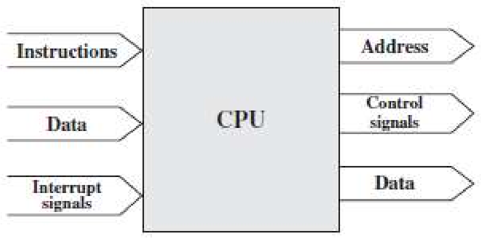
• **Memory:** Typically, a memory module will consist of N words of equal length. Each word is assigned a unique numerical address (0, 1, . . . , N – 1). A word of data can be read from or written into the memory. The nature of the operation is indicated by read and writes control signals. The location for the operation is specified by an address.



• **I/O module:** From an internal (to the computer system) point of view, I/O is functionally similar to memory. There are two operations, read and write. Further, an I/O module may control more than one external device. We can refer to each of the interfaces to an external device as a *port* and give each a unique address (e.g., 0, 1, . . . , M– 1). In addition, there are external data paths for the input and output of data with an external device. Finally, an I/O module may be able to send interrupt signals to the processor.



• **Processor:** The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system. It also receives interrupt signals.



The interconnection structure must support the following types of transfers:

• **Memory to processor:** The processor reads an instruction or a unit of data from memory.
• **Processor to memory:** The processor writes a unit of data to memory.

• **I/O to processor:**The processor reads data from an I/O device via an I/O module.
• **Processor to I/O:** The processor sends data to the I/O device.

• **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

Bus Interconnection:

A bus is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.

Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0. Over time, a sequence of binary digits can be transmitted across a single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor, memory, I/O) is called a *system bus.* The most common computer interconnection structures are based on the use of one or more system buses.

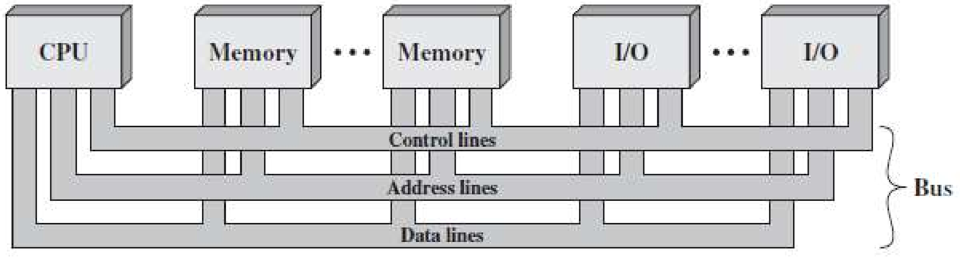
Bus Structure:

A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function.

The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the *data bus*. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system. Furthermore, the address lines are generally also used to address I/O ports. Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module. For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module

(module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).



The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include

- **Memory write:** Causes data on the bus to be written into the addressed location
- **Memory read:** Causes data from the addressed location to be placed on the bus
- **I/O write:** Causes data on the bus to be output to the addressed I/O port
- **I/O read:** Causes data from the addressed I/O port to be placed on the bus
- **Transfer ACK:** Indicates that data have been accepted from or placed on the bus
- **Bus request:** Indicates that a module needs to gain control of the bus
- **Bus grant:** Indicates that a requesting module has been granted control of the bus
- **Interrupt request:** Indicates that an interrupt is pending
- **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized
- **Clock:** Is used to synchronize operations
- **Reset:** Initializes all modules

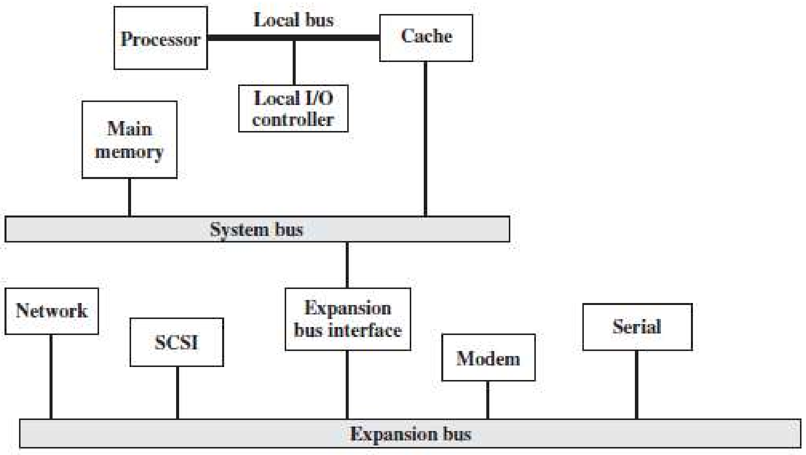
The operation of the bus is as follows. If one module wishes to send data to another, it must do two things: (1) obtain the use of the bus, and (2) transfer data via the bus. If one module wishes to request data from another module, it must (1) obtain the use of the bus, and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

Multiple-Bus Hierarchies:

If a great number of devices are connected to the bus, performance will suffer. There are two main causes:

1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.

2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.



(a) Traditional bus architecture

Bus Arbitration:

When the same set of address/data/control lines are shared by different units then the bus arbitration logic comes into play. Access to a bus is arbitrated by a bus master. Each node on a bus has a bus master which requests access to the bus, called a bus request, when then node requires to use the bus. This is a global request sent to all nodes on the bus. The node that currently has access to the bus responds with either a bus grant or a bus busy signal, which is also globally known to all bus masters.There are three schemes for bus arbitrations are:

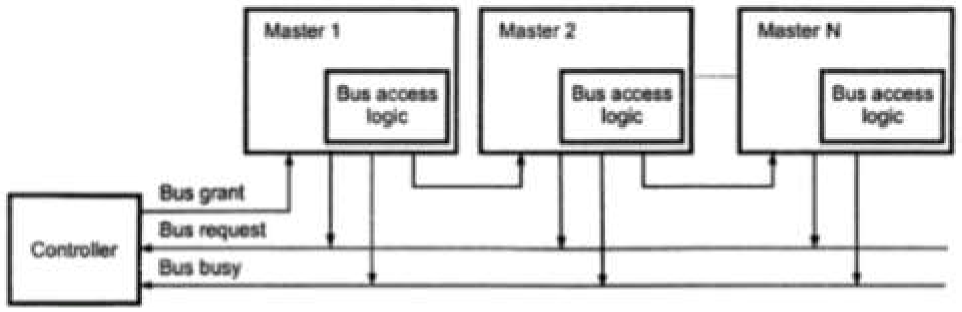
a. Daisy chaining

b. Polling method

c. Independent request

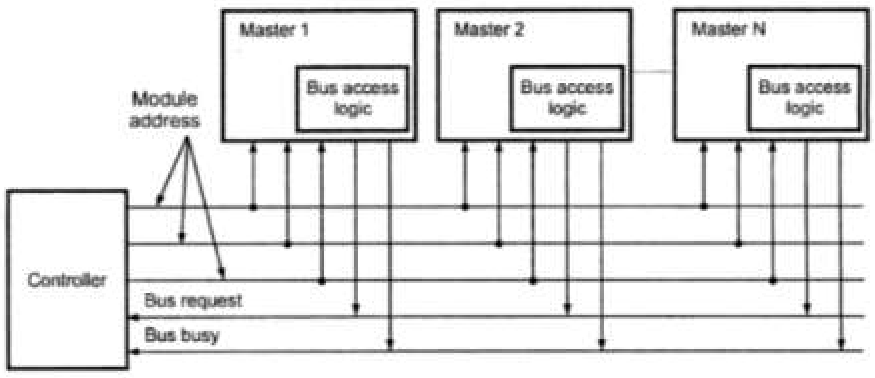
a) Daisy chaining

- The system connections for Daisy chaining method are shown in fig below.



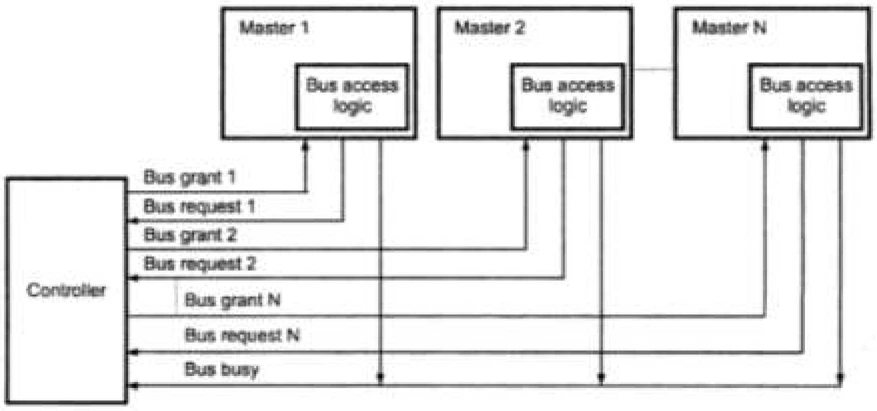
- It is simple and cheaper method. All masters make use of the same line for bus request.
- In response to the bus request the controller sends a bus grant if the bus is free.
- The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activities the busy line and gains control of the bus.
- Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access.

b) Polling method



- The system connections for polling method are shown in figure above.
- In this the controller is used to generate the addresses for the master. Number of address line required depends on the number of master connected in the system.
- For example, if there are 8 masters connected in the system, at least three address lines are required.
- In response to the bus request controller generates a sequence of master address. When the requesting master recognizes its address, it activated the busy line ad begins to use the bus.

c) Independent request



- The figure below shows the system connections for the independent request scheme.
- In this scheme each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.
- The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

Register Transfer Language

•The operations executed on data stored in registers are called micro-operations
•A micro-operation is an elementary operation performed on the information stored in one or more registers.
•Examples are shift, count, clear, and load

•The internal hardware organization of a digital computer is best defined by specifying
1. The set of registers it contains and their functions
2. The sequence of micro-operations performed on the binary information stored
3. The control that initiates the sequence of micro-operations

The symbolic notation used to describe the micro-operation transfers among registers is called a register transfer language.

Register Transfer

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

Four registers are essential to instruction execution:
• Program counter (PC): Contains the address of an instruction to be fetched
• Instruction register (IR): Contains the instruction most recently fetched

• Memory address register (MAR): Contains the address of a location in memory
• Memory buffer register (MBR): Contains a word of data to be written to memory or the word most recently read

For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register, and R 1 (for processor register). The individual flip-flops in an n-bit register are numbered in sequence from 0 through n - 1, starting from 0 in the rightmost position and increasing the numbers toward the left.

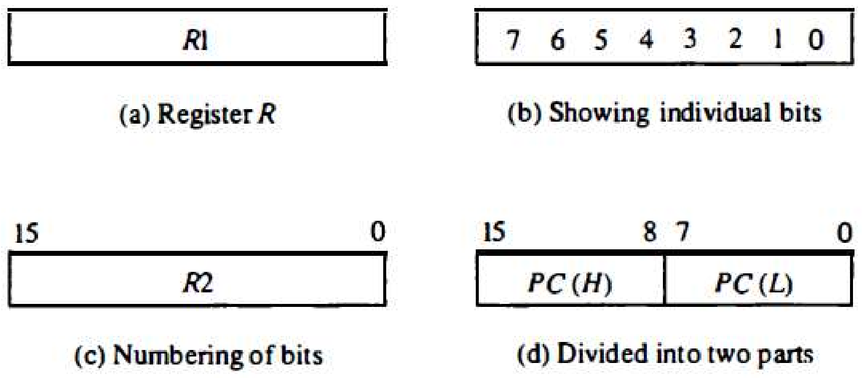
Figure 4-1 shows the representation of registers in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 4-1(a). The individual bits can be distinguished as in (b). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c). A 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC . The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.

Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement

$$R2 \leftarrow R1$$

denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R l. By definition, the content of the source register R1 does not change after the transfer.

Figure 4-1 Block diagram of register.



Control Function:

If we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.

If (P = 1) then (R2 <--R1)

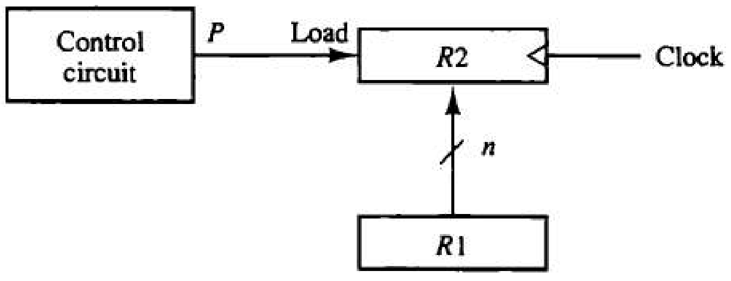
where P is a control signal generated in the control section. It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

P: R2 <--R1

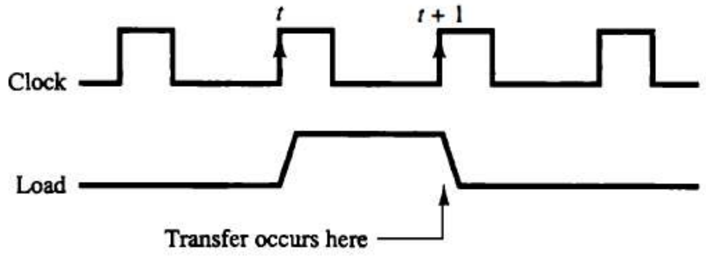
The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1.

Figure 4-2 shows the block diagram that depicts the transfer from R1 to R2. The n outputs of register R1 are connected to the n inputs of register R2. The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known. Register R2 has a load input that is activated by the control variable P. It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

Figure 4-2 Transfer from $R1$ to $R2$ when $P = 1$.



(a) Block diagram



(b) Timing diagram

In the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t. The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel. P may go back to 0 at time t + 1; otherwise, the transfer will occur with every clock pulse transition while P remains active.

Note that the clock is not included as a variable in the register transfer statements. It is assumed that all transfers occur during a clock edge transition. Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t + 1.

The basic symbols o f the register transfer notation are listed in Table 4-1 .Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time. The statement

$$T: R2 \leftarrow R1, R1 \leftarrow R2$$

denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1. This simultaneous operation is possible with registers that have edge-triggered flip-flops.

TABLE 4-1 Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1$, $R1 \leftarrow R2$

• Designate computer registers by capital letters to denote its function
• The register that holds an address for the memory unit is called MAR
• The program counter register is called PC
• IR is the instruction register and R1 is a processor register
• The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1

Bus and Memory Transfers:

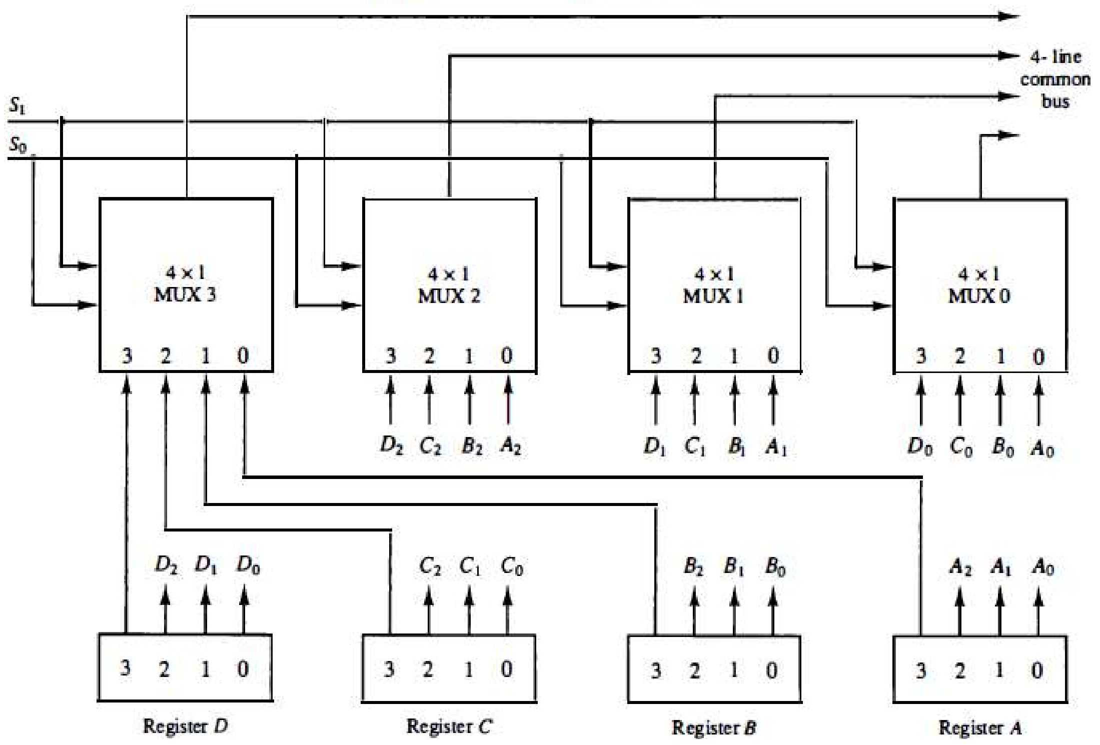
Bus Transfer:
Digital computer h a s many registers, and paths must b e provided to transfer information from one register to another. The number of wires will be excessive if separate lines are used between each register and all other registers in the system. A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

Common Bus System Using Multiplexer:

A common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in Fig. 4-3. Each register has four bits, numbered 0 through 3. The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A1.

The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

Figure 4-3 Bus system for four registers.



The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When S_1S_0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if S_1S_0 = 01, and so on. Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

TABLE 4-2 Function Table for Bus of Fig. 4-3

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n , the number of bits in each register. The size of each multiplexer must be k x 1 since it multiplexes k data lines. For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows:

$$BUS \leftarrow C, \quad R1 \leftarrow BUS$$

The content of register C is placed on the bus, and the content of the bus is loaded into register R 1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

$$R1 \leftarrow C$$

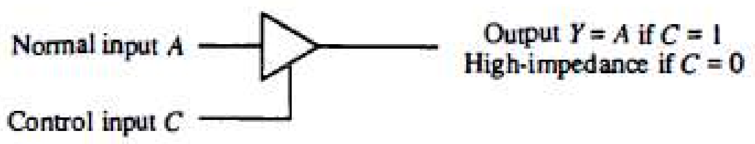
From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

Common Bus System Using Three-State Bus Buffers:

A bus system can be constructed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.

The graphic symbol of a three-state buffer gate is shown in Fig. 4-4. It is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input. When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

Figure 4-4 Graphic symbols for three-state buffer.



The construction of a bus system with three-state buffers is demonstrated in Fig. 4-5. The outputs of four buffers are connected together to form a single bus line. The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line. No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high-impedance state. One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram. When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled. When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

To construct a common bus for four registers of n bits each using three- state buffers, we need n circuits with four buffers in each as shown in Fig. 4-5. Each group of four buffers receives one significant bit from the four registers. Each common output produces one of the lines for the common bus for a total of n lines. Only one decoder is necessary to select between the four registers.

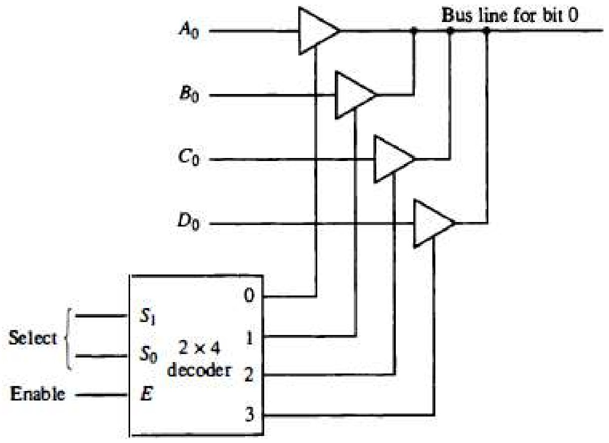


Figure 4-5 Bus line with three state-buffers.

Memory Transfer:

The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation. A memory word will be symbolized by the letter M.
Consider a memory unit that receives the address from a register, called the address register, symbolized by AR . The data are transferred to another register, called the data register, symbolized by DR . The read operation can be stated as follows:

$$\text{Read: } DR \leftarrow M[AR]$$

This causes a transfer of information into DR from the memory word M selected by the address in AR.

$$\text{Write: } M[AR] \leftarrow R1$$

This causes a transfer of information from R1 into the memory word M selected by the address in AR.

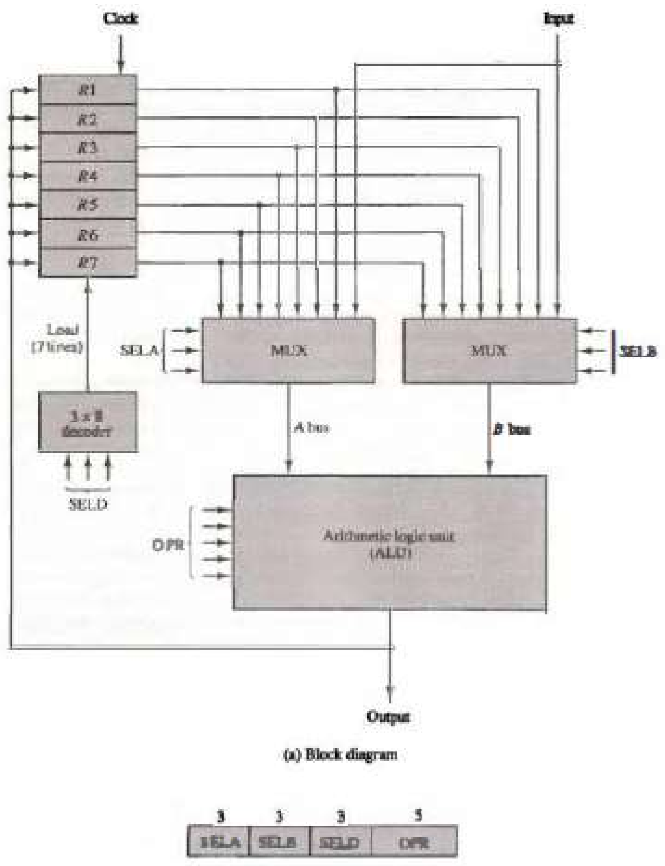
General Register Organization

A bus organization for seven CPU registers is shown in Fig. 8-2. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B . The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed. The result of the micro-operation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation

R 1 <--R2 + R3

the control must provide binary selection variables to the following selector inputs:
1. MUX A selector (SELA): to place the content of R2 into bus A .
2 . MUX B selector (SELB): to place the content o f R 3 into bus B .
3 . ALU operation selector (OPR): to provide the arithmetic addition
A + B .
4. Decoder destination selector (SELD): t o transfer the content o f the
output bus into R 1 .



(b) Control word

Figure 8-2 Register set with common ALU.

Control Word:

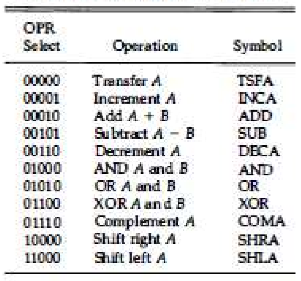
There are 14 binary selection inputs in the unit, and their combined value specifies a control word. The 14-bit control word is defined in Fig. 8-2(b). It consists of four fields. Three fields contain three bits each, and one field has five bits. The three bits of SELA select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU. The 14-bit control word when applied to the selection inputs specify a particular micro-operation.

TABLE 8-1 Encoding of Register Selection Fields

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

The encoding of the register selections is specified in Table 8-1. The 3-bit binary code listed in the first column of the table specifies the binary code for each of the three fields. The register selected by fields SELA, SELB, and SELD is the one whose decimal number is equivalent to the binary number in the code. When SELA or SELB is 000, the corresponding multiplexer selects the external input data. When SELD = 000, no destination register is selected but the contents of the output bus are available in the external output.

TABLE 8-2 Encoding of ALU Operations



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	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

Examples of Micro-operations

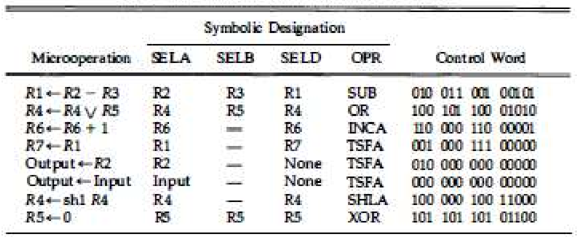
A control word of 14 bits is needed to specify a rnicrooperation in the CPU. The control word for a given microoperation can be derived from the selection variables. For example, the subtract rnicrooperation given by the statement

R 1 <- R 2 - R3

specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A - B. Thus the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding listed in Tables 8-1 and 8-2. The binary control word for the subtract micro-operation is 010 011 001 00101 and is obtained as follows:

Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

TABLE 8-3 Examples of Microoperations for the CPU



	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

A register can be cleared to 0 with an exclusive-OR operation. This is because $x \oplus x = 0$.

Stack Organization:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push (or push-down) because it can be thought of as the result of pushing a new item on top. The operation of deletion is called pop (or pop-up) because it can be thought of as the result of removing one item so that the stack pops up. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack:

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 8-3 shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word

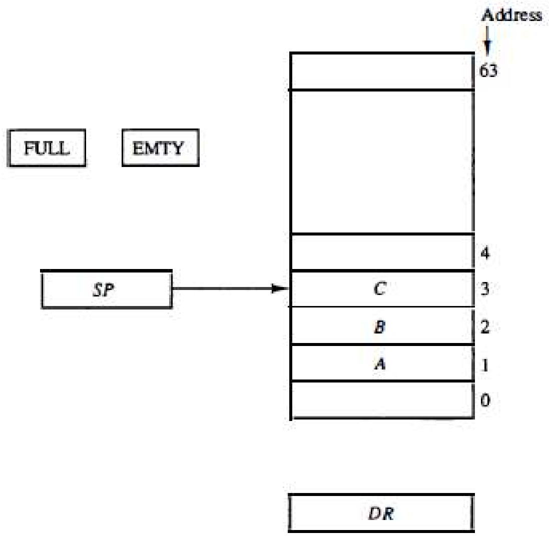


Figure 8-3 Block diagram of a 64-word stack.

at address 3 and decrementing the content of SP . Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place.

In a 64-word stack, the stack pointer contains 6 bits because 26 = 64. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 1 1 1 1 1 1 . The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack.

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of micro-operations;

SP ← SP + 1	**Increment stack pointer**
M [SP] ← DR	**Write item on top of the stack**
If (SP = 0) then (FULL ←1)	**Check if stack is full**
EMTY ←0	**Mark the stack not empty**

The stack pointer is incremented so that it points to the address of the next-higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M [SP] denotes the memory word specified by the address presently available in SP. The first item stored in the stack is at address L The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to L This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0. Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

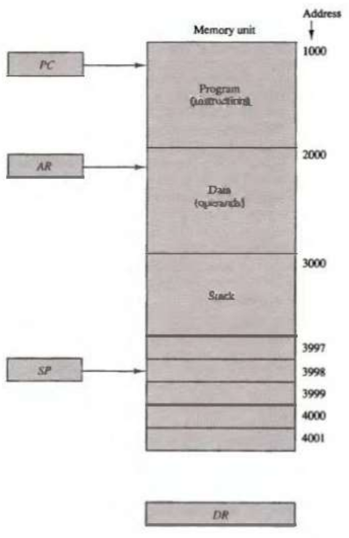
A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of micro-operations:

DR ←M [SP]	**Read item from the top of stack**
SP ←SP – 1	**Decrement stack pointer**
If (SP = 0) then (EMTY ←1)	**Check if stack is empty**
FULL ←0	**Mark the stack not full**

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1 This condition is reached if the item read was in location 1 Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63. In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1

Memory Stack:

A stack can exist as a stand-alone unit as in Fig. 8-3 or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Figure 8-4 shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.
In Fig. 8-4, the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000 , the second item is stored at address 3999, and the last address that can be used for the stack Is 3000. No provisions are available for stack limit checks.



We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.
Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack). The stack limits can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case). After a push operation, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register.
A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. Henceforth, SP is automatically decremented or incremented with every push or pop operation. The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

Shift Micro-operation:

Shift microoperations are used for serial transfer of data. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position. The information transferred through the serial input determines the type of shift. There are three types of shifts: logical, circular, and arithmetic.
Logical Shift:
A logical shift is one that transfers 0 through the serial input.The symbols shl and shr for logical shift-left and shift-right rnicrooperations.
For example:

$$R1 \leftarrow \text{shl } R1$$
$$R2 \leftarrow \text{shr } R2$$

are two microoperations that specify a 1-bit shift to the left of the content of register R 1 and a 1-bit shift to the right of the content of register R2. The register symbol must be the same on both sides of the arrow. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

Circular Shift:
The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input.

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Arithmetic Shift:

An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form.

Figure 4-1 1 shows a typical register of n bits. Bit R $_{n-1}$ in the leftmost position holds the sign bit. R_{n-2} is the most significant bit of the number and R_0 is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus R $_{n-1}$ remains the same; R_{n-2} receives the bit from R $_{n-1}$ and so on for the other bits in the register. The bit in R_0 is lost.

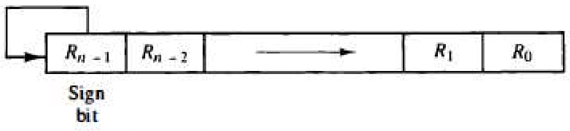


Figure 4-11 Arithmetic shift right.

The arithmetic shift-left inserts a 0 into R_0 and shifts all other bits to the left. The initial bit of R $_{n-1}$ is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R $_{n-1}$ changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R $_{n-1}$ is not equal to $R_{n-2.}$ An overflow flip-flop V, can be used to detect an arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

If $V_s = 0$, there is no overflow, but if $V_s = 1$, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

```
PUSH    A    TOS←A
PUSH    B    TOS←B
ADD          TOS←(A + B)
PUSH    C    TOS←C
PUSH    D    TOS←D
ADD          TOS←(C + D)
MUL          TOS←(C + D) * (A + B)
POP     X    M[X]←TOS
```

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

Question: Write a program to evaluate the arithmetic statement:

$$X = \frac{A - B + C*(D*E - F)}{G + H*K}$$

a. Using a general register computer with three address instructions.
b. Using a general register computer with two address instructions.
c. Using an accumulator type computer with one address instructions.
d. Using a stack organized computer with zero-address operation instructions.

Solution:

RPN: AB – C + DE * F – *GHK * + /

Addressing Modes:

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.

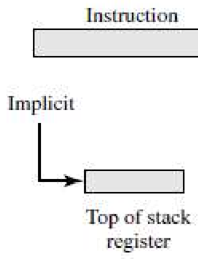
Figure 8-6 Instruction format with mode field.

Opcode	Mode	Address

There are two modes that need no address field at all. These are the implied and immediate 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.

Example:
Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.



(g) Stack

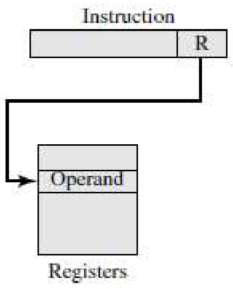
Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
The simplest form of addressing is immediate addressing, in which the operand value is present in the instruction

Operand = A

Instruction
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' registers. Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:

EA = R

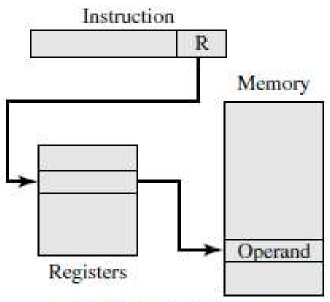


(d) Register

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.
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.

Just as register addressing is analogous to direct addressing, register indirect addressing is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location or a register. Thus, for register indirect address,

The advantages and limitations of register indirect addressing are basically the same as for indirect addressing. In both cases, the address space limitation (limited range of addresses) of the address field is overcome by having that field refer to a word-length location containing an address.



(e) Register indirect

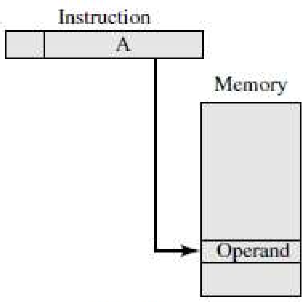
Autoincrement or Autodecrement Mode: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational type instruction.

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.

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:

$$\mathbf{EA = A}$$



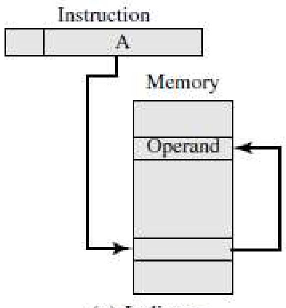
(b) Direct

Indirect Address 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.
With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand.This is known as *indirect addressing:*

$$EA = (A)$$

A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing:

$$EA = (\ldots.(A) \ldots..)$$



(c) Indirect

Displacement Addressing: A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use, but the basic mechanism is the same. This as *displacement addressing:*

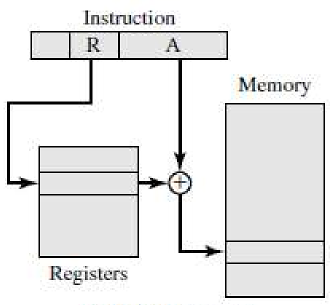
$$EA = A + (R)$$

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit.The value contained in one address field (value = A) is used directly.The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.
There are three of the most common uses of displacement addressing:
- Relative addressing
- Base-register addressing
- Indexing

A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following computation:

effective address = address part of instruction + content of CPU register

The CPU register used in the computation may be the program counter, an index register, or a base register.



(f) Displacement

Relative Address Mode: 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.
The address part of the instruction is usually a signed number (in 2' s complement representation) which can be either positive or negative. 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.

To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826. The effective address computation for the relative address mode is 826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction.

For relative addressing, also called PC-relative addressing, the implicitly referenced register is the program counter (PC). That is, the next instruction address is added to the address field to produce the EA.

Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
For indexing, the interpretation is typically the following: The address field references a main memory address, and the referenced register contains a positive displacement from that address.

Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
For base-register addressing, the interpretation is the following; The referenced register contains a main memory address, and the address field contains a displacement (usually an unsigned integer representation) from that address. The register reference may be explicit or implicit.

Example: The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500. The first word of the instruction specifies the operation code and mode, and the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of processor register R 1 is 400, and the content of an index register XR is 100. AC receives the operand after the instruction is executed. The figure lists a few pertinent addresses and shows the memory content at each of these addresses.

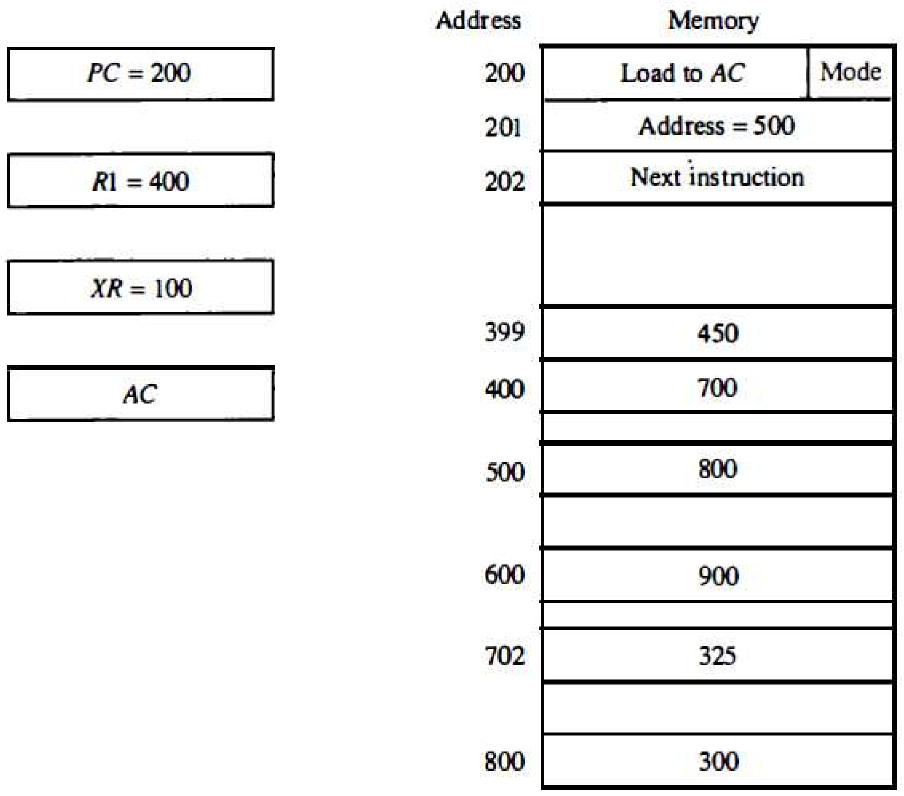


Figure 8-7 Numerical example for addressing modes.

The mode field of the instruction can specify any one of a number of modes. For each possible mode we calculate the effective address and the operand that must be loaded into AC .
In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC . (The effective address in this case is 201 .)

In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.

In the relative mode the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202.)

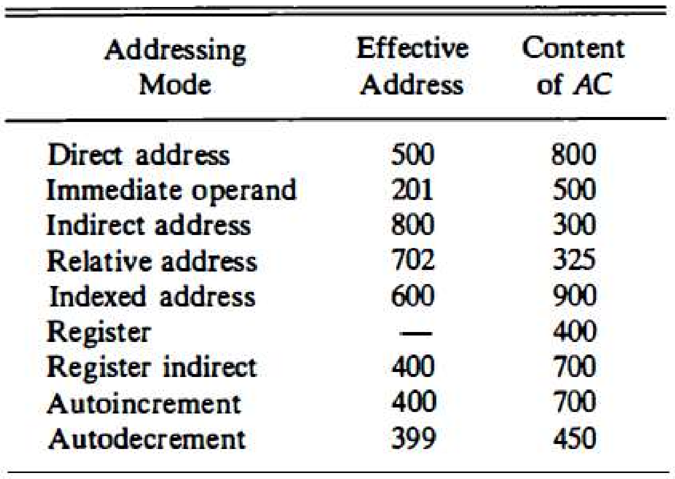
In the index mode the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900. In the register mode the operand is in R 1 and 400 is loaded into AC . (There is no effective address in this case.)

In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700.

The auto-increment mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction.

The auto-decrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450. the effective address and the operand loaded into AC for the nine addressing modes.

TABLE 8-4 Tabular List of Numerical Example



Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450