RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Computer Organization and Architecture**

Subject Code: **CS-404**

Semester: **4th**

UNIT – 1

Basic Structure of Computer:- Structure of Desktop Computers, CPU: General Register Organization- Memory Register, Instruction Register, Control Word, Stack Organization, Instruction Format, ALU, I/O System, bus, CPU and Memory Program Counter, Bus Structure, Register Transfer Language- Bus and Memory Transfer, addressing modes.

STRUCTURE OF DESKTOP COMPUTERS

The desktop computers are the computers which are usually found on a home or office desk. They consist of processing unit, storage unit, visual display and audio as output units, and keyboard and mouse as input units. Usually storage unit of such computer consists of hard disks, CD-ROMs, and diskettes. Desktop computers are basically digital computers. They consist of **fi**ve functionally independent units: input, memory, arithmetic and logic, output and control units. Memory unit is also known as storage unit, and arithmetic and logic unit (ALU) and control unit are combine as processing unit. Fig. shows these five functional

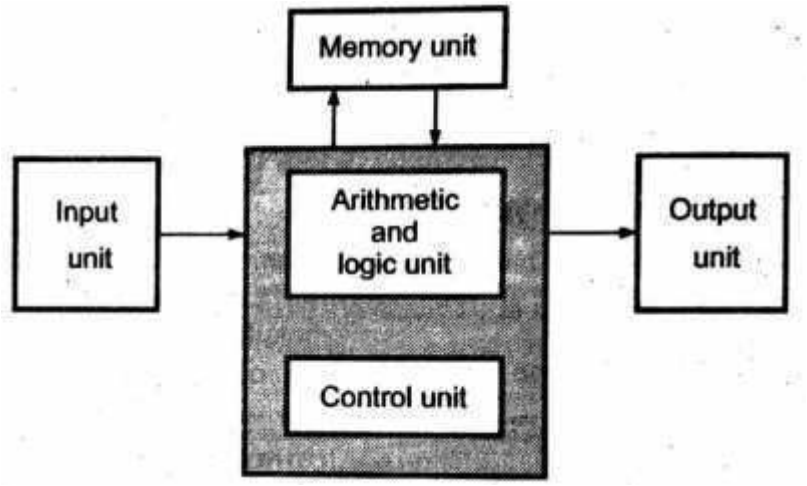


Figure 2. units of a computer.

Input Unit - The input unit accepts the digital information from user with the help of input devices such as keyboard, mouse, microphone etc. The information received from the input unit is either stored in the memory for later use or immediately used by the arithmetic and logic unit to perform the desired operations. The most commonly used input devices are keyboard and mouse, the keyboard is used for entering text and numeric information. On the other hand, mouse is used to position the screen cursor and thereby-enter the information by selecting option. Apart from keyboard and mouse there are many other input devices are available, which include joysticks, trackball, digitizers and scanners etc.

Memory Unit - The memory unit is used to store programs and data. Usually, two types of memory devices are used to form a memory unit: primary storage memory device and secondary storage memory device. The primary memory, commonly called main memory is a fast memory used for the storage of programs and active data. The main memory is a semiconductor memory. It consists of a large number of semiconductor storage cells, each capable of storing one bit of information. The main memory consists of only randomly accessed memories. These memories are fast but they are small in capacities and expensive. Therefore, the computer uses the secondary storage memories such as magnetic tapes, magnetic disks for the storage of large amount of data.

Arithmetic and Logic Unit - The Arithmetic and Logic Unit (ALU) is responsible for performing arithmetic operations such as add, subtract, division and multiplication and logical operations such

as ANDing, ORing, Inverting etc. To perform these operations, operands from the main memory are brought into the high speed storage elements called registers of the processor. Each register can store one word of data and they are used to store frequently used operands. The access times to registers are typically 5 to 10 times faster than access times to memory. After performing operation, the result is either stored in the register or memory location.

Output Unit - The output unit sends the processed results to the user using output devices such as video monitor, printer, plotter, etc. The video monitors display the output on the CRT screen whereas printers and plotters give the hard- copy output.

Control Unit - The control unit co—ordinates and controls the- activities amongst the functional. The basic function, of control unit is to fetch the instructions stored in the main memory, identify .the operations and the devices involved in it and accordingly generate control signals to execute the desired operations. The control unit uses control signals or timing signals to determine when a given action is to take place. It controls input and output operations, data transfers between the processor, memory and input/ output devices using timing signals.

CPU (CENTRAL PROCESSING UNIT)

The CPU is the brain of the Computer system. It works as an administrator of a system. All the operations within the system are supervised and controlled by CPU. It interprets and co-ordinates the instructions. The data and instructions are temporarily stored in its memory unit. After performing Operation, the result of operation can be stored in this memory unit.

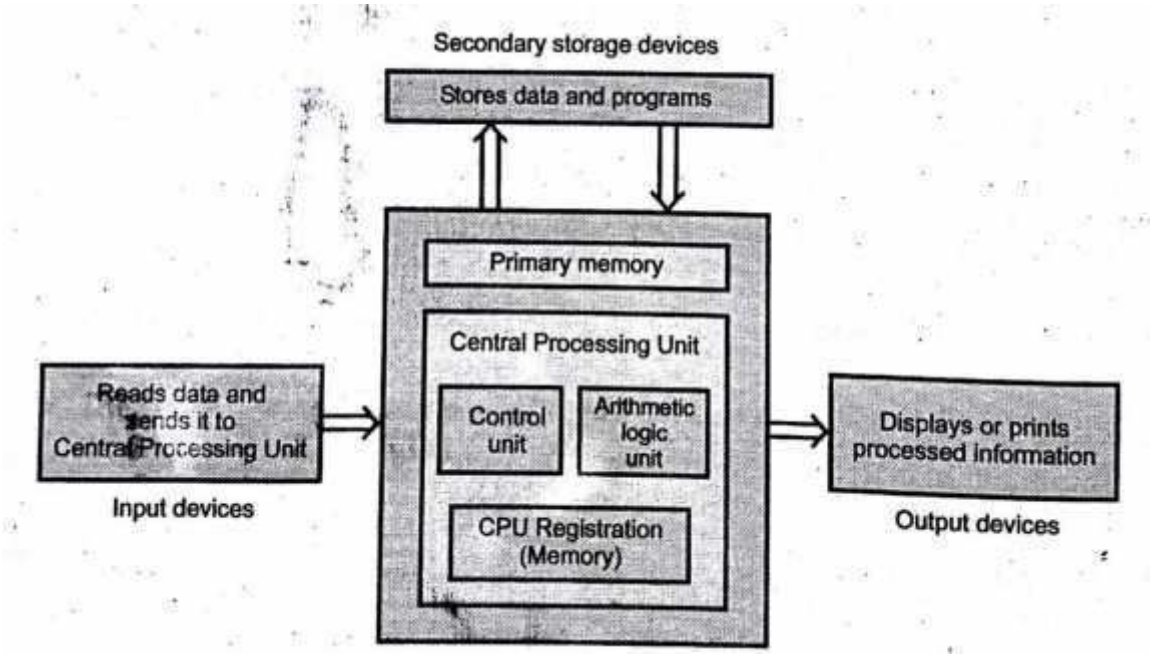


Figure 2 CPU and its interaction with other units

The results of operation are sent towards output unit for the user. Thus, CPU controls all internal and external devices, performs arithmetic and logical operations, controls the memory usage and control the sequence of operations. For performing all these operations, the CPU has three subunits.

- Arithmetic and Logic Unit (ALU)
- Control Unit
- Memory (CPU registers)

Arithmetic and Logic Unit (ALU) - ALU is a subunit of CPU. It performs arithmetic operations like addition, subtraction and logic operations like OR, AND, invert, exclusive-OR on binary words.

The data stored in memory unit is transferred to ALU. The ALU performs the operation, that is, the data is processed and the result is stored in internal memory unit of CPU. The result of final operation is transferred from memory unit to an output unit. Arithmetic and logic operations performed by ALU sets flags to represent certain conditions such as equal to condition, zero condition, greater than condition and so on.

Control Unit - The control unit controls all the operations which internally take place within the CPU and also the operations of CPU related to input/output devices. The control unit directs the overall functioning of a computer system. This unit also checks the correctness of sequence of operations. It fetches instructions in a program from the primary storage unit, interprets them and generates control signals to ensure correct execution of the program. The control signals generated by the control unit direct the overall functioning of the other units of the computer.

CPU Registers - Register is a group of flip-flops which can be used to store a word. It is a high speed temporary storage space for holding data, addresses and instructions during processing the instruction. Registers are not referenced by their addresses; they are directly accessed. To perform execution of instruction, the processor contains a number of registers used for temporary storage of data and some special function registers.

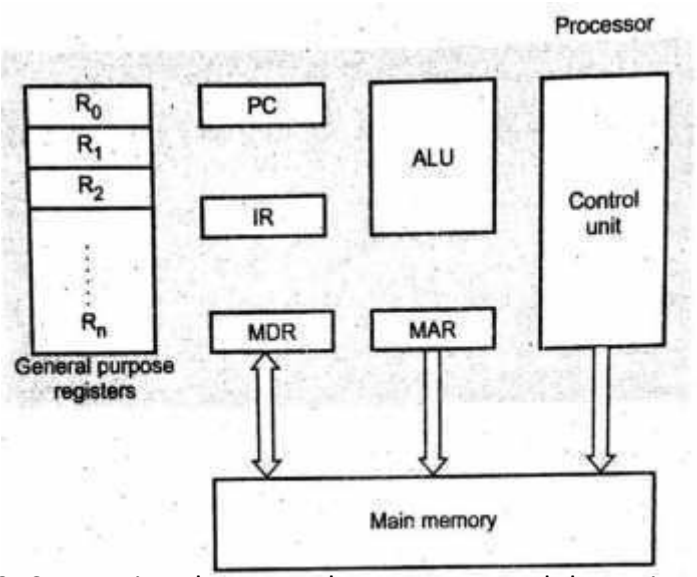


Figure 3. Connections between the processor and the main memory

The special function registers include Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR) and Memory Data Register (MDR).
Program Counter (PC):- A program is a series of instructions stored in the memory. These instructions tell the CPU exactly how to get the desired result. It is important that these instructions must be executed in a proper order to get. the correct result. The sequence of instruction execution is monitored by the program counter. It keeps track of which instruction is being executed and what the next instruction.

Instruction Register (IR):- It is used to hold the instruction that is currently being executed. The contents of IR are available to the control unit, which generate the timing signals that control the various processing elements involved in executing the instruction.

Memory Address Register [MAR] and Memory Data Register (MDR): - These registers are used to handle the data transfer between the main memory and the processor. The MAR holds the address of the main memory to or from which data is to be transferred. The MDR sometimes also called MBR (Memory Buffer Register) contains the data to be written into or read from the addressed word of the main memory.

General purpose registers - These are used to hold the operands for arithmetic and logic operations and/or used to store the result of the operation. Since the access time of these registers is lowest, these are used to store frequently used data. Arithmetic logic unit uses CPU registers to accept data for processing. After processing data ALU Stores result again in the CPU register. A'LU also stores the status of the result in. the CPU register. This result includes information whether result is positive/negative, zero, having even/odd parity, any carry/borrow resulted during arithmetic operation and so on. This register is commonly known as flag register or program status register. Supervisory control unit of a CPU uses flag register to execute conditional branch/jump instructions.

GENERAL REGISTER ORGANIZATION- MEMORY REGISTER, INSTRUCTION REGISTER

Bus organization

Fig. 4 shows the general bus organization for seven CPU registers. It shows that how registers are selected and how data flow between register and ALU take place. Decoder is used to select a particular register. 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 the input data for the particular bus. The A and B buses form the two inputs of an Arithmetic Logic Unit (ALU). The operation select lines decide the micro operation to be performed by ALU. The result of the micro- operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

The basic function performed by a computer is the execution of a program. The program, which is to be executed, is a set of instructions, which are stored in memory. The central processing unit (CPU) executes the instructions of the program to complete a task. The instruction execution takes place in the CPU registers. Let us, first discuss few typical registers, some of which are commonly available in of machines. These registers are:-

- Memory Address Register (MAR):- Connected to the address lines of the system bus. It specifies the address of memory location from which data or instruction is to be accessed (for read operation) or to which the data is to be stored (for write operation).
- Memory Data Register (MDR):- Connected to the data lines of the system bus. It specifies which data is to be accessed (for read operation) or to which data is to be stored (for write operation).
- Program Counter (PC):- Holds address of next instruction to be fetched, after the execution of an on-going instruction.
- Instruction Register (IR):- Here the instructions are loaded before their execution or holds last instruction fetched.
- Accumulator (ACC):- It holds the result generated by ALU.

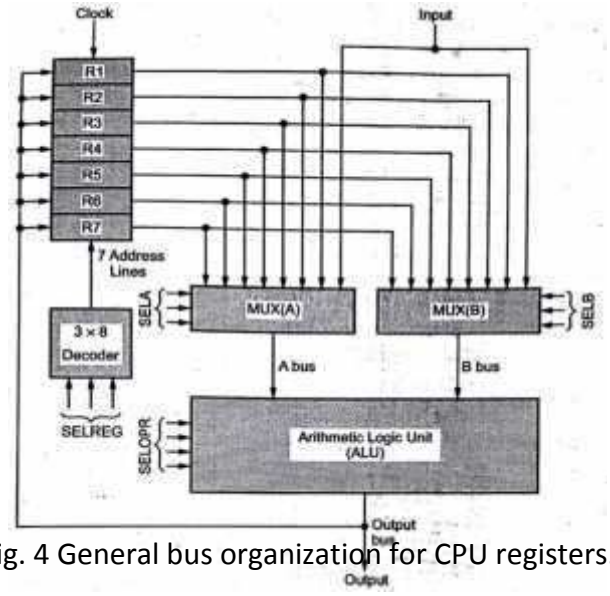


Fig. 4 General bus organization for CPU registers.

We designate computer registers by capital letters to denote the function of the register. For example, the register that holds an address for the memory unit is usually called a memory address register, represented by MAR. Other examples are PC (for program counter), IR (for instruction register) and R1 (for processor register). We show the individual flip-flops in an n-bit register by giving numbers them in sequence from 0 through n - 1, starting from 0 in the right most position and increasing the numbers toward the left. A 16-bit register is divided into two halves.

Low byte (Bits 0 through 7) is assigned the symbol L
High byte (Bits 8 through 15) is assigned the symbol H.

The name of a 16-bit register is PC. The symbol PC (L) represents the low order byte and PC (H) designates the high order byte. The statement

Table 2.1: List of Registers for the Basic Computer

Register Symbol	Number of bits	Register name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

R2 ?R1 refers the transfer of the content of register R1 into register R2.

It should be noted that the content of the source register R1 does not change after the transfer. In real applications, the transfer occurs 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 of the system. For convenience we 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 written as follows:

P: R2 ?R1

A read operation implies transfer of information to the outside environment from a memory word, whereas storage of information into the memory is defined as write operation. Symbolizing a memory word by the letter M, it is selected by the memory address during the transfer which is a specification for transfer operations. The address is specified by enclosing it in square brackets following the letter M. For example, the read operation for the transfer of a memory unit M from an address register AR to another data register DR can be illustrated as:

Read: DR ←M[AR]

The write operation transfer the contents of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address in the AR. The write operation can be stated symbolic as follows:

Write: M[AR] ← R1

This cause a transfer on information from R1 into the memory word M selected by the address in AR

CONCEPT OF CONTROL WORD

The combined value of a binary selection inputs specifies the control word. That Fig. 5 shows the control Word format. It consists of four **fields** SELA, SELB and SELREG contain three bits each and SELOPR field contains four bits. Thus the total bits in the control word are 13-bits.

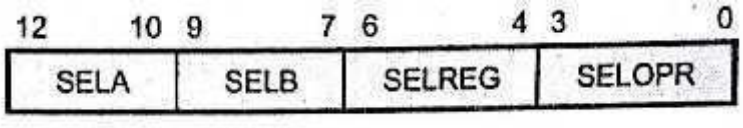


Figure 5 Format of control word

The three bits of SELA select a source registers of the A input of the ALU. The three bits of SELB select a source registers of the B input of the ALU. The three bits of SELREG select a destination register using the decoder. The four bits of SELOPR select the operation to be performed by ALU.

STACK ORGANIZATION

Data operated on by a program can be stored in the CPU registers and in the computer memory. It can be organized properly for easy access. For example, the data items of similar data type can be organized in the term of arrays and can be accessed using array pointers. Like array data structure, mast of the computers supports an important data structure known as a stack.

A stack is a list of data elements, usually words or bytes with the accessing restriction that the elements can be added or removed at one end of the list only. This end is called the top of the stack and the other end is called the bottom of the stack. As only one end is used for adding and removing data elements, the element stored last is the first element retrieved. Thus stack structure is also known as Last-In-First-Out (LIFO) list. The terms Push and Pop are used to describe placing a new data element on the stack and removing the top data element from the stack, respectively.

The stack in the digital computer is a part of register unit or memory unit with a register that holds the address for the stack. The part of register array or memory used for stack is called stack area and the register used to hold the address of stack is called stack pointer. The value in the stack pointer always points at the top data element in the stack.

1. Register Stack

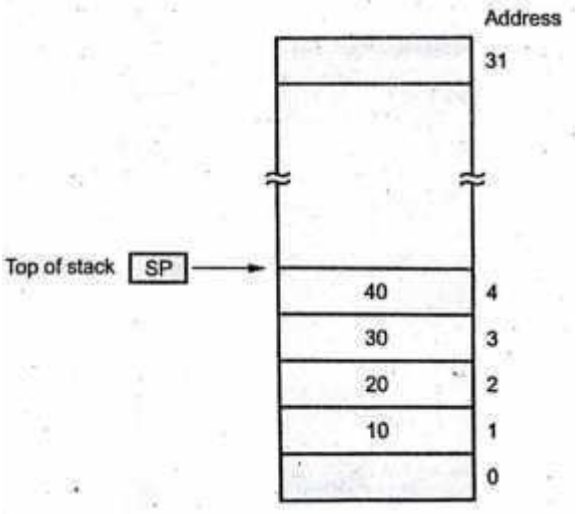


Figure 6. 32 word register stack

A stack can be placed in a portion of a memory unit or it can be organized as a collection of a finite number of CPU registers. The Figure 6 shows the organization of a 32-word register stack. The stack pointer holds the address of the register that is currently the top of stack. As shown in the Fig. 6 four data elements 10, 20, 30 and 40 are placed in the stack. The data element 40 is on the top of stack therefore, the content of SP is now 4.

To remove the top data element the stack is popped by reading the register at address 4 and decrementing the content of SP. The data element 30 is now on top of the stack since SP holds address 3. To insert a new data element, stack is pushed by incrementing SP and writing a data element in the incremented location in the stack. If SP reaches 31 and one more data element is pushed then SP is incremented to 0 and data element is stored in the register 0. Now there is no more empty registers in the stack and stack is said to be FULL. The Fig. 6 shows the operation of stack when specified sequence of instructions are executed.

2. Memory Stack

The operation of memory stack is exactly similar to the register stack. It is implemented using computer memory instead of CPU register array. The number of registers in the CPU is limited and it restricts the size of stack in the stack computer. But when stack is implemented using memory its size is extended upto the memory addressing capacity of the CPU. But computer memory is also shared by program and data, as shown in the Fig. 7 Hence a part of computer memory is used for the stack. The memory stack has an advantage of large size but the operation on it is slower than that of register stack. This is because register stack is internal to the CPU and does not heed any memory access.

In the stack organized computers instruction's operands are stored at the top of the stack, so data processing instructions do not need to contain addresses as they do in a conventional Von Neumann computer. The add operation A + B is specified for a stack computer by the following sequence of three instructions.

```
PUSH    A
PUSH    B
ADD
```

The first PUSH instruction loads A into top of stack (TOS). Execution of PUSH B causes A's location to become TOS + 1 and places B in the new TOS immediately above A. To execute ADD, the top two words of the stack are popped into the ALU where they are added, and the sum is pushed back into the stack. As shown by program sequence in the stack computer operator is specified after the operands i.e. AB + instead of A + B. This method of representing arithmetic expression is known as Reverse Polish notation. The stack computer evaluates arithmetic and other expressions using a reverse polish notation format.

In most conventional computers stack is used to implement subroutine call and return instructions. In some situations, it is better to execute part of a program that is not in sequence with the main program. For example, there may be a part of a program that must be repeated many times during the execution of the entire program. Rather than writing repeated part of the program again and again, the programmer' can write that part only once. This part is written separately. The part of the program which is written separately is called subroutine. When the subroutine is called by CALL instruction the current content of PC are pushed on the stack and after execution of subroutine program the content of PC are popped back into the PC to execute the next instruction after the CALL.

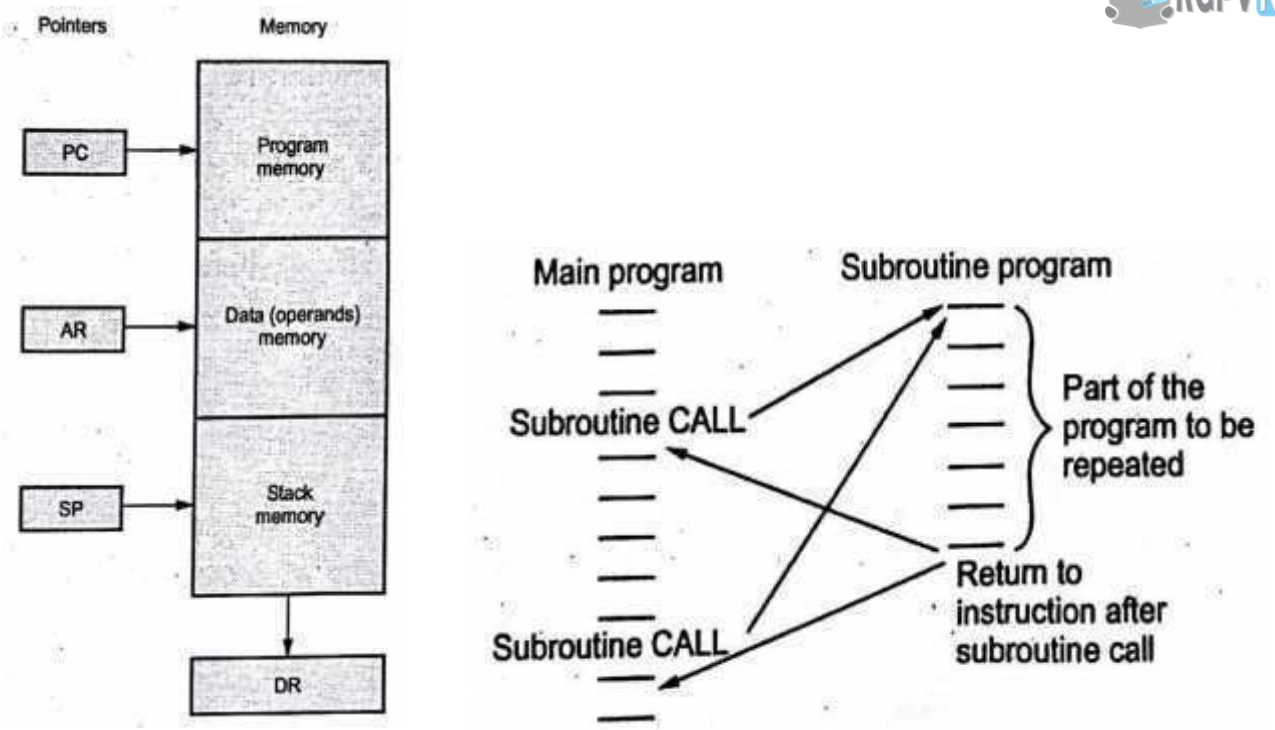


Figure 7 a) Sharing of computer memory by programs, data& stack area b) Execution of subroutine programs

Notations

Basically, there are three methods of representing arithmetic expressions. These are

1. A+B : Infix notation.
2. + AB : Prefix or polish notation.
3. AB + : Postfix or reverse polish notation

INSTRUCTION FORMAT

The internal construction of the CPU, including the processor registers available and their logical capabilities. Computer has a variety of instruction code formats, it is the control unit within the CPU that interprets each instruction code and provides the necessary control functions needed to process the instruction. The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. These information fields of instructions are called elements of instruction & most common fields found in instruction formats are:

Operation code: - The operation code field in the instruction specifies the operation to be performed. The operation is specified by binary code hence the name operation code or simply opcode.

Source / Destination operand: - The source/destination operand field directly specifies the source/destination operand for the instruction. Source operand address: - The operation specified by the instruction may require one or more operands. The source operand may be in the CPU register or in the memory. Many times the instruction specifies the address of the source operand so that operand(s) can be accessed and operated by the CPU according to the instruction.

Destination operand address: - The operation executed by the CPU may produce result. Most of the time the results are stored in one of the operand. Such operand is known as destination operand. The instruction which produce result specifies the destination operand address.

Next instruction address: - The next instruction address tells the CPU from where to fetch the next instruction after completion of execution of current instruction. For JUMP and BRANCH instructions the address of the next instruction is specified within the instruction. However, for other instructions, the next instruction to be fetched immediately follows the current instruction.

At times other special fields are also employed, sometimes employed as for example a field that gives the number of shifts in a shift-type instruction. The operation code field of an instruction is referred to a group of bits that define various processor operations, such as add, subtract, complement, and shift. A variety of alternatives for choosing the operands from the given address is specified by the bits that define the mode field of the instruction code. Execution of operations done by some data stored in memory or processor registers through specification received by computer instructions.

A memory address specifies operands residing in memory whereas those residing in processor registers are specified by a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R_0 through R_{15} will have a register address field of four bits. The internal organization of the registers of a computer determines the number of address fields in the instruction formats. Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization.

An example of an accumulator-type organization is the basic computer. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as:

ADD X

Where X is the address of the operand. The ADD instruction in this case results in the operation AC AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

2. General register organization.

The instruction format in this type of computer needs three register address fields. Thus, the instruction for an arithmetic addition may be written in an assembly language as:

ADD Rl, R2, R3

To denote the operation Rl R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same and one of the source registers. Thus, the instruction:

ADD Rl, R2

Would denote the operational R1 R1 + R2. Only register addresses for R1 and R2 need to be specified in this instruction.

3. Stack organization.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction:

MOV R1, R2

denotes the transfer R1 R2 (or R2 R1, depending on the particular computer). Thus, transfer-type instructions need two address fields to specify the source and the destination.

Usually only two or three address fields are employed general register-type computers in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized by:

ADD R1, X

would specify the operation R1 R1+M[X]. It has two address fields, one for register R1 and the other for the memory address X.

Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction:

PUSH X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction:

ADD

In a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. Since all operands are implied to be in the stack, the need for specifying operands with the address field does not arise. Most of the computers comprise one of the three types of organizations. we will evaluate the arithmetic statement

X = (A + B)*(C + D)

Three Address Instructions

The three address instruction can be represented symbolically as

ADD A, B, C

Where A, B, C are the variables. These variable names are assigned to distinct locations in the memory. In this instruction operands A and B are called source operands and operand C is called destination operand and ADD is the operation to be performed on the operands. Thus the general instruction format for three address instruction is

Operation Source 1, Source 2, Destination

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

ADD R1, A, B R1 M[A] + M[B] ADD R2, C, D R2 M[C] + M[D] MUL X, R1, R2 M[X] R1 * R2

It is assumed that the computer has two processor registers, Rl and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

The number of bits required to represent such instruction include :

1. Bits required to specify the three memory addresses of the three operands. If n-bits are required to specify one memory address, 3n bits are required to specify three memory addresses.
2. Bits required to specify the operation.

Two Address Instructions

The two address instruction can be represented symbolically as

ADD A, B

This instruction adds the contents of variables A and B and stores the sum in variable B destroying its previous contents. Here, operand A is source operand however operand B serves as both source and destination operand. The general instruction format for two address instruction is

Operation Source, Destination

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or memory word. The program to evaluate X = (A + B)* (C + D) is as follows:

MOV R1, A R1?M[A] ADD R1, B
R1?RI+M[B] MOV R2, C R2?M[C]
ADD R2, D R2?R2+M[D] MUL R1,
R2 R1?R1*R2
MOV X, R1 M[X]?R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation transferred.

To represent this instruction less number of bits are required as compared to three address instruction. The number of bits required to represent two address instructions include:

1. Bits required specifying the two memory addresses of the two operands, i. e. 2n bits.
2. Bits required specifying the operation.

One Address Instruction

The one address instruction can be represented symbolically as

ADD B

This instruction adds the contents of variable A into the processor register called accumulator and stores the sum back into the accumulator destroying the previous contents of the accumulator. In this instruction the second operand is assumed implicitly in a unique location accumulator. The general instruction format for one address instruction is

Operation Source
One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the

second register and assume that the AC contains the result of all operations. The program to evaluate X=(A+B)*(C+D) is:

LOAD A AC M[A] ADD B
AC AC+M[B] STORE T M[T] AC
LOAD C AC M[C] ADD D, A
AC AC+M[D] MUL T AC AC *
M[T] STORE X M[X] AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result

Few more examples of one address instructions are:

LOAD A: This instruction copies the contents of memory location A into the M accumulator.
STORE B: This instruction copies the contents of accumulator into memory location B.

Zero Address Instructions

In these instructions, the locations of all operands are defined implicitly. Such instructions are found in machines that store operands m a structure called a pushdown stack. A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

PUSH A TOS A PUSH B
TOS B ADD T TOS (A+B)
PUSH C TOS C PUSH D
TOS D ADD T TOS (C+D)
MUL T TOS (C+D)*(A+B)
POP T 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.

The instruction with only one address will require less number of bits to represent it, and instruction with three addresses will require more number of bits to represent it. Therefore, to access entire instruction from the memory, the instruction with three addresses requires more memory accesses while instruction with one address requires less memory accesses. The speed of instruction execution is mainly depend on how much memory accesses it requires for the execution. If memory accesses are more time is required to execute the instruction. Therefore, the execution time for three address instructions is more than the execution time for one address instructions.

To have a less execution time we have to use instructions with minimum memory accesses. For this instead of referring the operands from memory it is advised to refer operands from processor registers.

ALU

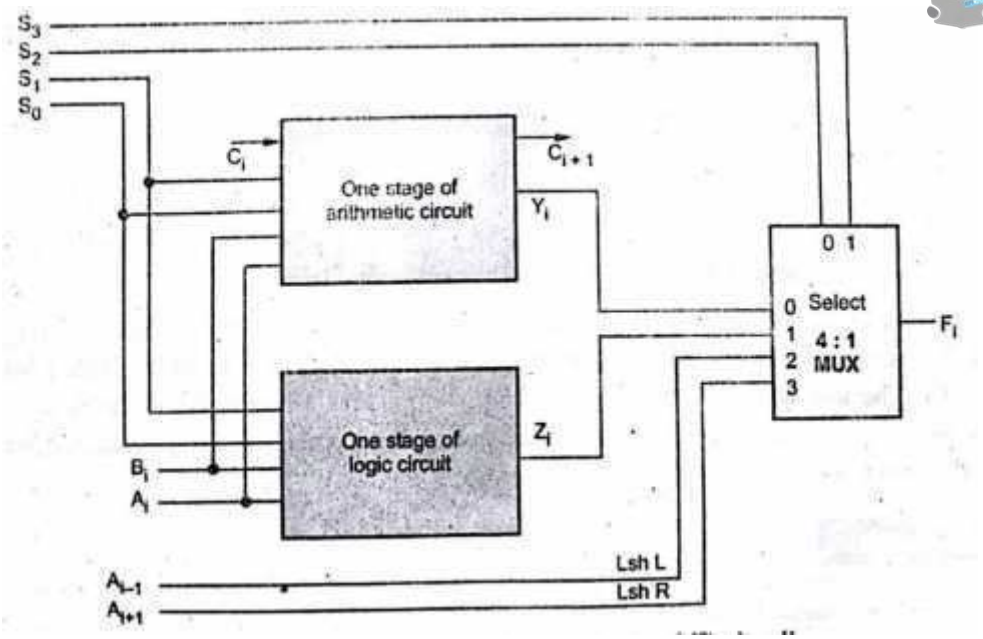


Figure 8. One stage of arithmetic logic shift circuit

The design of arithmetic and logic circuits for a simple ALU. By combining both Circuits with the help of multiplexer we can get the arithmetic and logic circuit, as shown in the Fig. 8. The Fig. 8 shows one stage of an arithmetic logic shift unit. The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. Inputs S_1 and S_0 are used to select a particular micro- operation. A 4:1 multiplexer at the output chooses between an arithmetic output in Yi and a logic output in Z*i*. The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for shift left operation and A_{i+1} for the right shift operation.

I/O SYSTEM BUS

The central processing unit memory unit and I/O unit are the hardware components/modules of the computer. They work together with communicating each other and have paths for connecting the modules together. The collection of paths connecting the various modules is called the interconnection structure. A group of wires called bus is used to provide necessary signals for communication between modules. A bus is a shared transmission medium, it must only be used by one device at a time and when used to connect major computer components (CPU, memory, I/O) is ' called a system bus.

The system bus is separated into three functional groups: data bus, address bus and control bus

Data lines (data bus) - Move data between system modules. The data bus lines are bidirectional CPU can read data on these lines from mentor or from a port, as well as send data out on these lines to a memory location or to a port. Width is a key factor. It determines number of bytes that can be transferred in one cycle and hence the overall system performance.

Address lines (address bus) - Designate source or destination of data on the data bus. It is an unidirectional bus. Width determines the maximum possible memory capacity of the system. It also used to address I/O ports. Typically: High-order bits select a particular module. Lower order bits select a memory location or I/0 port within the module.

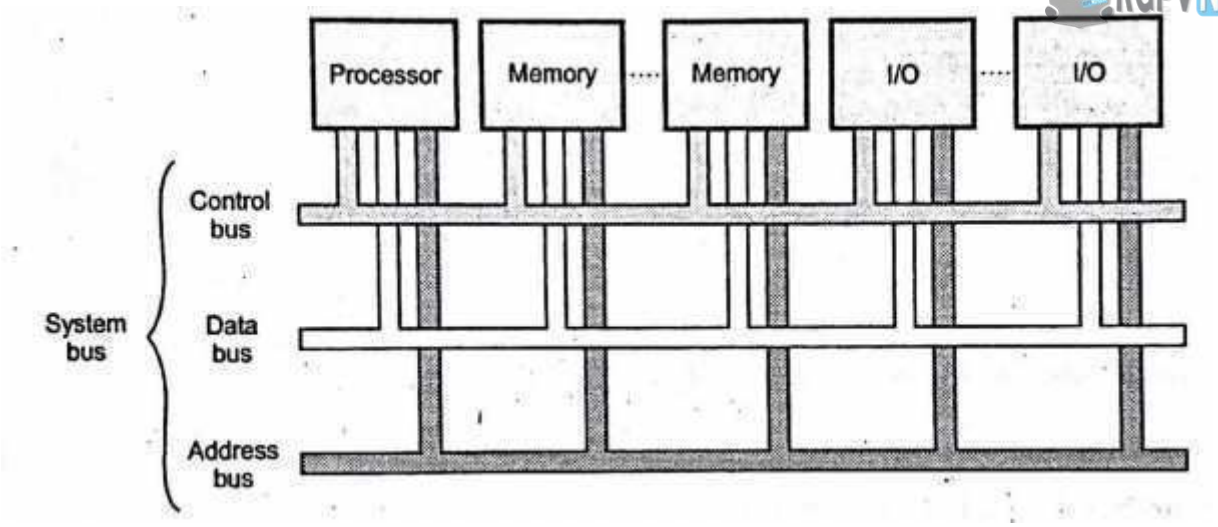


Figure 9. Bus Interconnection scheme

Control lines (Control bus) - Control access to use the data and address lines. Typical control lines include:

1. Memory Read and Memory write
2. I/O Read and [/0 Write
3. Transfer ACK
4. Bus Request and Bus Grant
5. Interrupt Request and Interrupt ACK
6. Clock & Reset

If one module wishes to send data to another, it must:

1. Obtain use of the bus
2. Transfer data via the bus

If one module wishes to request data from another, it must _:

1. Obtain use of the bus
2. Transfer a request to the other module over control and address lines
3. Wait for second module to send data

Connecting I/O Devices to CPU and Memory

Fig 10. Shows the bus interconnection scheme. It shows that how I/0 devices are connected to CPU and memory. I/0 devices are interfaced to CPU through I/O interface or I/O module. The I/O interface consists of circuit, which connect an I/O device to a CPU bus. On one side of the interface we have the bus signals for address, data and control. On the other side we have a data path with its associated controls to transfer data between the interface and the I/O device. Usually I/O interface or I/O module is capable of interfacing more than one external device.

Since data, address and control buses are connected in parallel to CPU, memory and I/O the I/O module is allowed to exchange data directly with memory without going through the processor, using Direct Memory-Access (DMA). The bus interconnection scheme shown in Fig. 10 supports following types of data transfers:

1. Memory to processor - Memory read operation
2. Process to memory – Memory write operation
3. Processor to I/O – I/O write operation
4. I/O to processor - I/O read operation
5. I/O to or from memory- DMA operation

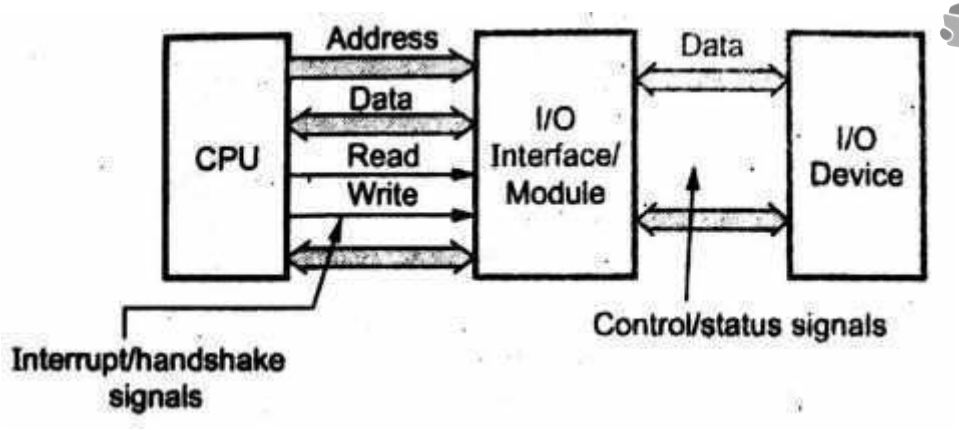


Figure 10. Connecting I/O device to the CPU

BUS STRUCTURE

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. A common bus system can be constructed using multiplexers. These multiplexers select the source register whose binary information is then placed on the bus. The system bus is a cable which carries data communication between the major components of the computer, including the microprocessor. Not all of the communication that uses the bus involves the CPU, although naturally the examples used in this tutorial will centre on such instances. The system bus consists of three different groups of wiring, called the data bus, control bus and address bus. These all have separate responsibilities and characteristics, which can be outlined as follows:

Control Bus

The control bus carries the signals relating to the control and co-ordination of the various activities across the computer, which can be sent from the control unit within the CPU. Different architectures result in differing number of lines of wire within the control bus, as each line is used to perform a specific task. For instance, different, specific lines are used for each of read, write and reset requests.

Data Bus

This is used for the exchange of data between the processor, memory and peripherals, and is bi-directional so that it allows data flow in both directions along the wires. Again, the number of wires used in the data bus (sometimes known as the 'width') can differ. Each wire is used for the transfer of signals corresponding to a single bit of binary data. As such, a greater width allows greater amounts of data to be transferred at the same time.

Address Bus

The address bus contains the connections between the microprocessor and memory that carry the signals relating to the addresses which the CPU is processing at that time, such as the locations that the CPU is reading from or writing to. The width of the address bus corresponds to the maximum addressing capacity of the bus, or the largest address within memory that the bus can work with. The addresses are transferred in binary format, with each line of the address bus carrying a single binary digit. Therefore the maximum address capacity is equal to two to the power of the number of lines present (2^lines).

Bus structure is divided in two types

1. Single Bus Structure

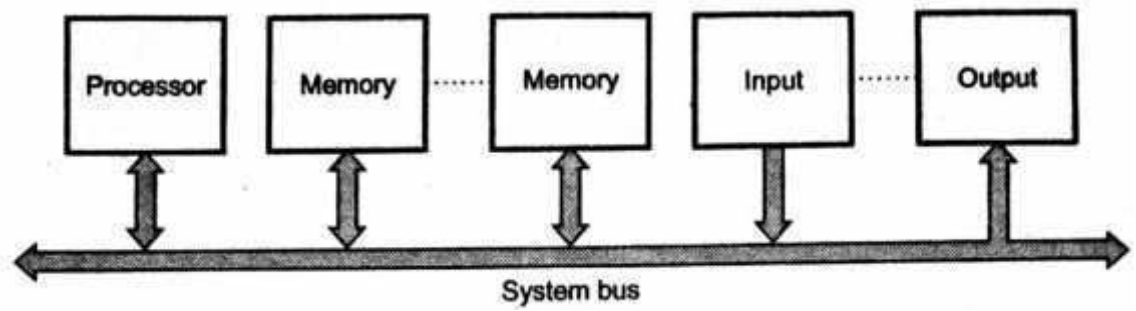


Figure 11. Single bus structure

Another way to represent the same bus connection scheme is shown in Fig. 11. Here address bus, data bus and control bus are shown by single bus called system bus. Hence such interconnection bus structure is called single bus structure. In a Single bus structure all units are connected to common bus called system bus. With single bus only two units can communicate with each other at a time. The bus control lines are used to arbitrate multiple requests for use of the bus. The main advantage of single bus structure is its low cost and its flexibility for attaching peripheral devices.

2. Multiple Bus Structure

Multiple architecture is divided in to two type. A great number of devices on a bus will cause performance to suffer due to

1. Propagation delay - the time it takes for devices to coordinate the use of bus.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

Now-a-days the data transfer rates for video controllers and network interfaces are growing rapidly. The need of high speed shared bus is impractical to satisfy with a single bus. Thus, most computer systems use the multiple buses. These buses have the hierarchical structure as shown in the Fig. 12

2.1 Traditional hierarchical bus architecture - The traditional bus connection uses three buses : local bus, system bus and expanded bus. Use of a cache structure insulates CPU from frequent accesses to main memory. Main memory can be moved off local bus to a system bus. Expansion bus interface is used to buffers data transfers between system bus and an I/O controller on expansion bus insulates memory-to-processor traffic from I/O traffic. Traditional hierarchical bus breaks down as higher and higher performance is seen in the I/O devices.

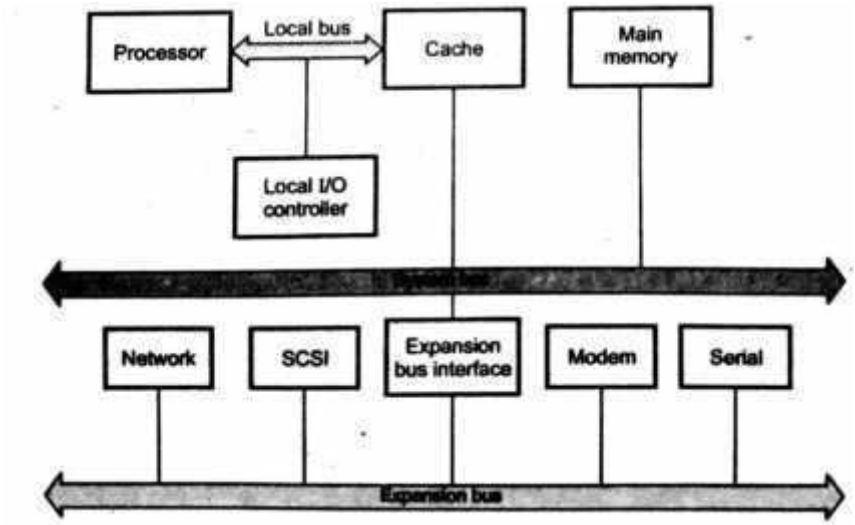


Figure 12 Traditional bus configuration

2.2 High-performance hierarchical bus I architecture - Specifically designed to support high—capacity I/O devices. It uses high-speed bus along with the three buses used in the traditional bus connection. Here cache controller is connected to high-speed bus. This bus supports connection to high-speed LANs, such as Fiber Distributed Data Interface (FDDI), video and graphics workstation controllers, as well as interface controllers to local peripheral buses including SCSI and P1394.Changes in processor architecture do not affect the high speed bus and vice versa

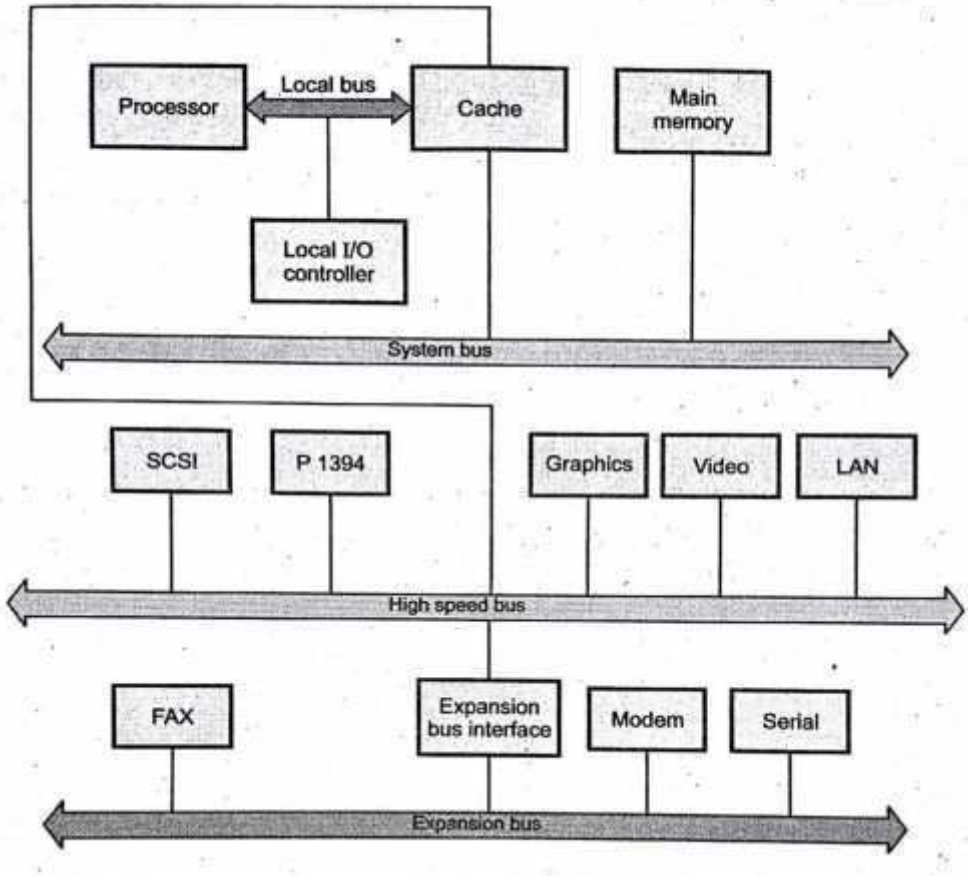


Figure 13 High-speed bus configuration

REGISTER TRANSFER LANGUAGE

The digital system can be constructed with flip flops and gates using sequential logic design. However, it is difficult redesign large digital systems using this approach. To overcome this difficulty, large digital systems are designed using modular approach. In the modular approach, system is partitioned into subsystems such that each subsystem performs some functional task. Such modules are constructed from registers, counters, decoders, multiplexers, arithmetic and logic elements and control Logic. The modules are then interconnected with common data and control paths to form a digital system.

The features of register transfer logic

1. Uses registers as a primitive component in the digital system instead of flip-flops and gates.
2. The information flow and processing tasks among the data stored in the registers is described in a concise and precise manner.
3. Uses a set of expressions and statements which resemble the statements used in programming languages.

4. The presentation of digital functions in register transfer logic is very user friendly.

Basic Components of Register Transfer Logic

The register transfer logic method uses four basic components to describe digital systems. These are as follows:

1. Registers and their functions: The total number of registers in the system and their functions includes all its counterparts such as shift registers, counters and memory unit. Counter is register, which increments the information stored in it by
   1. A memory unit is a set of storage registers where information can be stored.

2. Information: The information is stored in the registers may be binary numbers, BCD numbers, alphanumeric characters, control information or any other binary coded information.

3. Operations: The operations performed on the information stored in the registers.
   The operations performed on the data stored in registers are called micro-operations and it is performed in one clock cycle. The operation may be arithmetic operation or logical, operation. A statement that requires a sequence of micro-operations for its implementation is called a macro-operation.

4. Control function: The control functions that activate operations and central the sequence of operations. They consists of timing signals that sequence the operations one at a time the control function is basically a binary variable, when it is logic 1 it initiates the operation otherwise it inhibits the operation.-

Definition of Register Transfer Language

A micro operation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may by transfer to another register. The symbolic notation used to describe the micro operation transfer among registers is called a *register transfer language*. The term "register transfer" implies the availability of hardware logic circuits that can perform stated micro operation and transfer the results to the operation to the same or another register. Examples of micro-operation are lead, store, clear, shift, and count and so on. Sequences of micro- operations are performed to perform one complete operation. For example, to add two numbers following micro-operation sequence has to be performed.

1. Load first number in register 1
2. Load second number in register 2.
3. Perform add micro-operation.
4. Store the result in the destination register.

As shown above it is possible to specify the sequence of micro-operation in a word, but it involves lengthy descriptive explanation. Hence, it is preferred to use symbolic notations to describe the micro-operations. These symbolic notations are also called a register transfer language or computer hardware description language.

The micro-operations used in the digital system can be classified as:

1. Register transfer micro-operations: The micro-operations that transfer information from one register to another. 2. Arithmetic micro-operations: The micro-operations that perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operation: The micro-operations that perform bit manipulation operations on non numeric data stored in registers.

4. Shift micro-operations: The micro-operations that perform shift operations on data stored in registers.

1. *Arithmetic Micro-operations*

These micro-operations perform some basic arithmetic operations on the numeric data stored in the registers. These basic operations may be addition, subtraction, incrementing a number, decrementing a number and arithmetic shift operation. An **'add'** micro-operation can be specified as: R3 ← R1 + R2

It implies: add the contents of registers R1 and R2 and store the sum in register R3. The add

operation mentioned above requires three registers along with the addition circuit in the ALU. Subtraction is implemented through complement and addition operation as:

R3 ← R1 – R2 is implemented as
R3 ← R1 + (2's complement of R2)
R3 ← R1 + (1's complement of R2 + 1)
R3 ← R1 + R2 + 1

An increment operation can be symbolized as: R1 ← R1 + 1

While a decrement operation can be symbolized as: R1 ← R1 – 1

We can implement increment and decrement operations by using a combinational circuit or binary up/down counters. In most of the computers multiplication and division are implemented using add/subtract and shift micro-operations. If a digital system has implemented division and multiplication by means of combinational circuits then we can call these as the micro-operations for that system. An arithmetic circuit is normally implemented using parallel adder circuits. Each of the multiplexers (MUX) of the given circuit has two select inputs. This 4-bit circuit takes input of two 4-bit data values and a carry-in-bit and outputs the four resultant data bits and a carry-out-bit. With the different input values we can obtain various micro-operations.

Equivalent micro-operation	Micro- operation name
R ← R1 + R2	Add
R ←R1 + R2 +1	Add with carry
R ←R1 + R2	Subtract with borrow
R ←R1 + 2's	Subtract
R ←R1	Transfer
R ←R1 + 1	Increment
R ←R1 – 1	Decrement

2. *Logic Micro-operations*

These operations are performed on the binary data stored in the register. For a logic micro-operation each bit of a register is treated as a separate variable. For example, if R1 and R2 are 8 bits registers and

R1 contains 10010011 and
R2 contains 01010101

R1 AND R2 00010001

Some of the common logic micro-operations are AND, OR, NOT or complements Exclusive OR, NOR, NAND. We can have four possible combinations of input of two variables. These are 00, 01, 10 and 11. Now, for all these 4 input combination we can have 2^4 = 16 output combinations of a function. This implies that for two variables we can have 16 logical operations.

Logic Micro Operations

SELECTIVE SET

The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. it does not affect bit positions that have 0's in B. the following numerical example clarifies this operation:-

```
1010        A before
1100        B (logic operand)
----------------------------
1110        A after
```

It is clear that the OR micro-operation can be used to selectively set the bits of a register.

SELECTIVE COMPLEMENT

The selective-complement operation complements bits in register A where there are corresponding 1's in register B. it does not affect bit positions that have 0's in B. The following numerical example clarifies this
operation:-

```
1010        A before
1100        B (logic operand)
----------------------------
0110        A after
```


Hence the exclusive OR micro-operation can be used to selectively complement bits of a register.

SELECTIVE CLEAR

The selective-clear operation clears to 0 the bits in register A only where there are corresponding 1's in register B. For example:-

```
1010        A before
1100        B (logic operand)
----------------------------
0010        A after
```

Hence the logic micro-operation corresponding to this is: A B ^ B

MASK OPERATION

The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. the mask operation is an AND micro operation, for example:-

```
1010        A before
1100        B (logic operand)
----------------------------
1000        A after masking
```


The two right most bits of A are cleared because the corresponding bits of B are 0's.
The two right most bits are left unchanged due to the corresponding bits of B (i.e.
1). The mask operation is more convenient to use than the selective clear because most computers provide an AND instruction, and few provide an introduction that executes the micro operation is an AND micro operation.

INSERT OPERATION

The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value. For example, suppose that an A register contains eight bits, 0110 1010. To replace the four unwanted bits:-

```
0110 1010        A before
0000 1111        B (mask)
---------------------------
0000 1010        A after masking
```

and then insert the new value by replace the four leftmost bits by the value 1001

```
0000 1010        A before
1001 0000        B (insert)
---------------------------
1001 1010        A after insertion
```

The mask operation is an AND micro operation and the insert operation is an OR micro operation.

CLEAR OPERATION

The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal. This operation is achieved by an exclusive-OR micro operation as has own by the following example:

```
1010          A
1010          B
-----------------
0000    ←   A⊕     A  B
```

When A and B are equal, the two corresponding bits are either both 0 or both 1. In either case the exclusive-OR operation produces a 0. The all-0's result is then checked to determine if the tow numbers were equal.

3. Shift Micro operations

Shift micro operation can be used for serial transfer of data. They are used generally with the arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. During a shift-right operation the serial input transfers a bit into the leftmost position. The serial input transfers a bit into the rightmost position during a shift-left operation. There are three types of shifts, logical, circular and arithmetic.

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

Logical shift

A logical shift operation is one that transfers 0 through the serial input. We use the symbols shl and shr for logical shift left and shift right micro operations, e.g.

R1 ← shl R1
R2 ← shr R2

are the two micro operations that specify a 1-bit shift left of the content of register R1 and a 1- bit shift right of the content of register R2. In logical shift 1change in 0 and 0 change in 1.

Circular shift

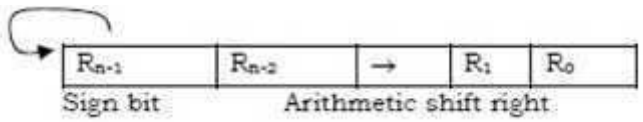
The circular shift is also known as rotate operation. It circulates the bits of the register around the two ends and there is no loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. We use the symbols cil and cir for the circular shift left and circular shift right. In circular shift while shifting value not change only position change. E.g. suppose Q1 register contains 01101101 then

Q1 register bit 01101101 *Q1 register bit*

After cir operation, it contains *1*0110110 *0*1101101 And after cil operation it will contain 11011010.

Arithmetic Shift

An arithmetic shift micro operation shifts a signed binary number to the left or right. The effect of an arithmetic shift left operation is to multiply the binary number by 2. Similarly an arithmetic shift right divides the number by 2. Because the sign of the number must remain the same, arithmetic shift-right must leave the sign bit unchanged, when it is multiplied or divided by 2. The left most 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. Following figure shows a typical register of n bits.



Bit R_{n-1} in the left most 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 bits) to the right. Thus R_{n-1} remains the same; R_{n-2} receives the bit from R_{n-1}, and so on for other bits in the register. In arithmetic shift value change 1 from 0 and 0

from 1, 1 indicate negative and 0 indicate positive, if negative change in positive we say overflow otherwise vice versa.

Register transfer

The symbolic notation used to describe the micro-operation belongs to transfers among registers forms a syntax/ statement of a register transfer language. For example, the statement denotes a transfer of the content of register R1 into register R2. After transfer the contents of register R2 are replaced by the content of register R1, however the content of register R1 remain unchanged. The term register transfer implies the availability of hardware logic circuit that can perform a stated micro-operation and transfer the result of operation to the same or another register. Therefore just like various programming languages we have register transfer language to specify the micro-operations. Each statement in the register transfer language specify the unique micro-operation.

A register transfer language also allows to specify control conditions in the statement. To specify control conditions it uses control variable along with the colon at the leftmost side of the statement as shown below.

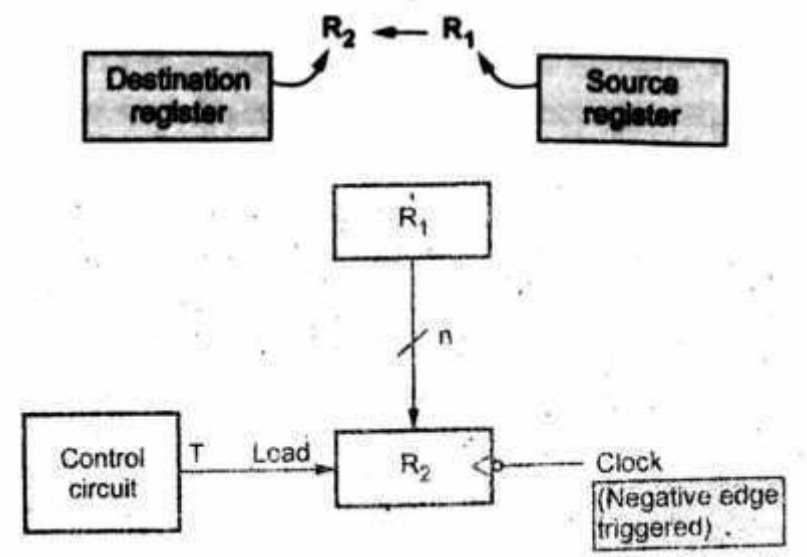


Figure 14 block diagram for representing R2<— R1 micro-operation

Here, T is used as a control variable it is basically a Boolean variable having value equal to 1 or 0. This statement indicates that the content of R1 are transferred to R2 only when T=1; otherwise transfer operation is not performed. In a computer, micro-operations are synchronized with the help of clock signal. The micro- operation is initiated either at the rising edge or at the falling edge of the clock signal depending on the circuit.

It is important to 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 either positive or negative. If hardware permits two micro-operations can be executed at the same time. For example, statement exchanges the contents of two registers, they perform R2 ? R1, R3 ? R4 micro operations simultaneously they are separated by coma in a statement.

BUS AND MEMORY TRANSFER

Bus Transfer

A digital computer has many registers and it is necessary to provide data path between them to transfer information from one register to another. If separate lines are used between each register,

there will be excess number of wires and controlling of those wires make circuit complex. Therefore, in multiple-register configuration a common bus system is used to transfer information between two registers. A common bus 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 are used to determine which is the source register and which is a destination register for that particular transfer. The common bus scheme can be implemented in two ways

1. Using multiplexers.
2. Using tri-state bus buffers.

Implementation of common bus using multiplexers

The Fig. 15 shows the implementation of common bus system for four registers using multiplexers. Each register has four bits, numbered 0 through 3 and they are routed through multiplexers to the common bus. Here, four multiplexers are used to select four bits of the source register. Each multiplexers has four input lines, two select lines and one output line. The four input lines of multiplexer 0 (MUX 0) are connected to the bit 0 outputs of four registers such that bit 0 of register is connected to input 0, bit 0 of register 1 is connected to input 1, bit 0 of register 2 is connected to input 2 and bit 0 of register 3 is connected to input 3. Similarly, inputs for MUX 1 are connected to bit 1 outputs, inputs for MUX 2 are connected to bit 2, and inputs for MUX 3 are connected to bit 3 outputs of registers
0 through 3. To avoid the complexity of the diagram, only input connections for
MUX 3 are physically shown. To show other connections labels are used. Two same labels have connection between them.

The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers. These lines choose the four bits of one register and transfer them into the four-line common bus through OUT lines. When S_1S_0 = 00, the input 0 of all four multiplexers are selected and applied to the outputs to transfer them on the common bus. As a result a common bus receives the contents of register 0. Since the outputs of register 0 are connected to the input 0 of the multiplexers. Similarly, the content of register 1 are transferred on the common bus when S_1S1. The Table below shows the register selection according to the status of S_1 and S_0 lines.

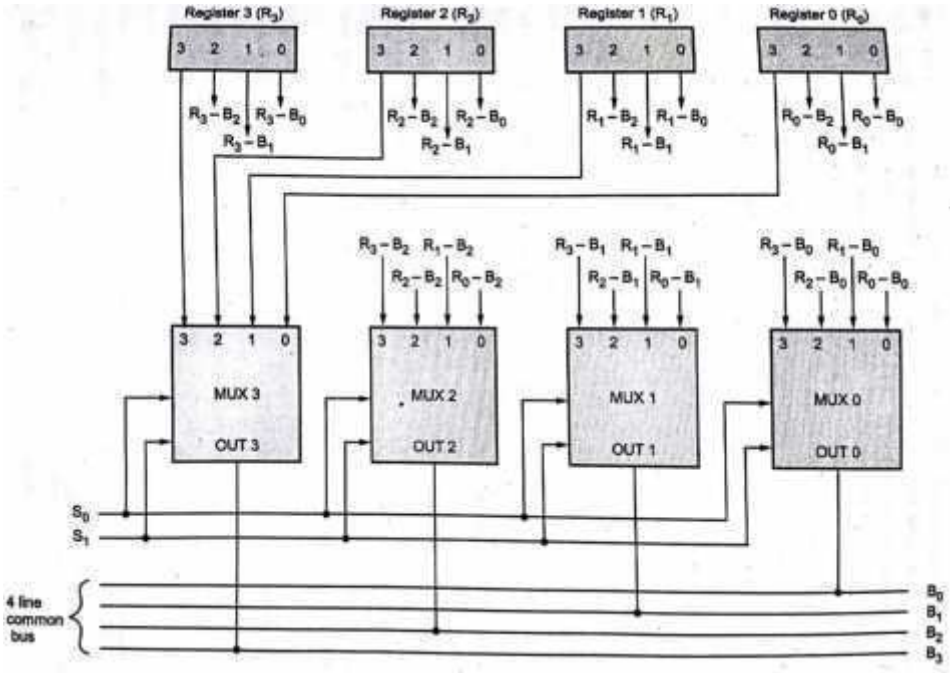


Figure 16. Computer bus System using multiplexer

S_1	S_0	Selected register
0	0	Register 0
0	1	Register 1
1	0	Register 2
1	1	Register 3

Table 1 Selection table

In general, common bus system will have n multiplexers and n line common bus where n represents the number of bits in each register and k number of inputs to each multiplexer where it represents the total number of registers. Therefore, the size of each multiplexer will be k x 1. The data transfer from a bus to any destination register can be accomplished by connecting the bus lines to the inputs. of all destination registers and activating the load control input of the particular (selected) destination register.

Implementing common bus using tri-state buffer

A common bus system can be implemented using tri-state or three state gates instead of multiplexers. A tri-state gate is a digital circuit that can have three output states: HIGH, LOW and high impedance. The high impedance state behaves like an open circuit, which means that the output is disconnected.

Bus is a common connection between number of registers and other units. Therefore, the data transfer through tri-state gates may require larger sinking and sourcing of current. The tri-state gate which provides more sinking and sourcing capacities is called tri-state buffer. The Fig. 17 shows the logic symbol for tri-state

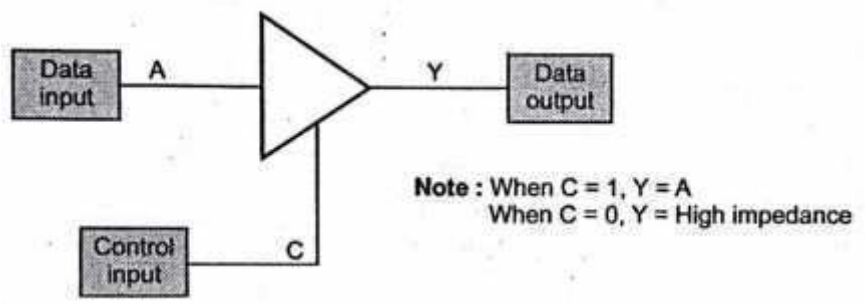
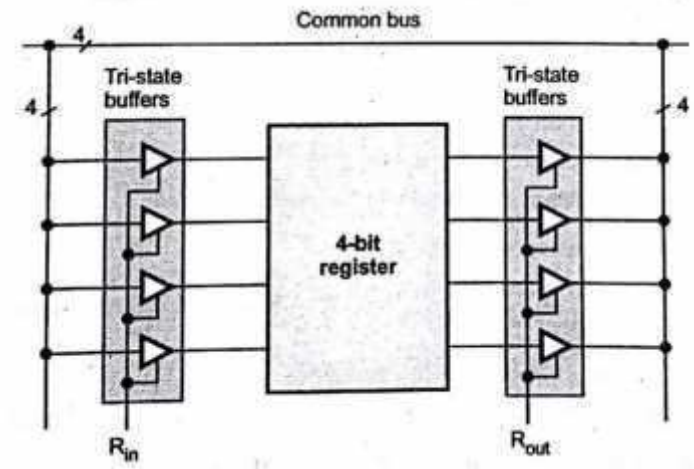


Figure 17 (a) Logic symbol of tri-state buffer



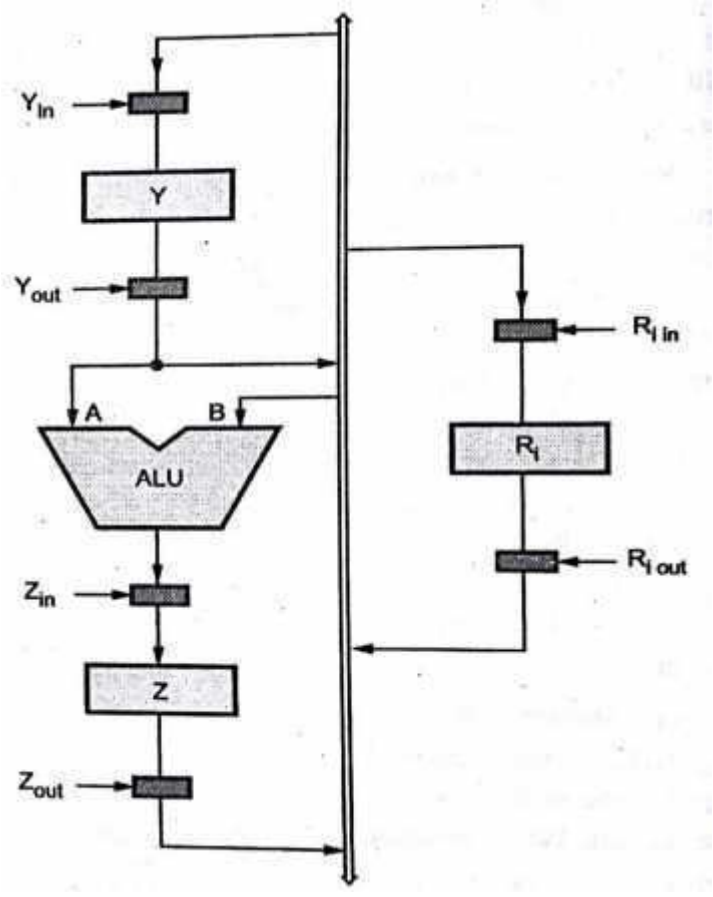
(b) working of tri state buffer with register

As shown in the Fig.17 the control input decides the state of the output. When control input is 1, the data is available at the output, i.e. Y = A. On the other hand, when control input is 0, the output gets to '0' high-impedance state.

The Fig. 17 (b) shows how the data transfer takes place between register and common bus. Here, 4-bit register is shown with tri-state buffers at input and output Sides. Each data line requires two tri-state buffers one at the input side and one at the output side. Therefore, for 8-bit register we require eight tri-state buffers at the input side and eight tri-state buffers at the output side.

As shown in the Fig. 17 all tri-state buffers at the input side are control by a common control signal R_{in}. When R_{in} is active the n-bit data from the common bus is loaded into the register. Similarly, at the output, a common signal R_{out} controls all output tri—state buffers. When R_{out} is active the 4-bit data from register is placed on the common bus.

The Fig. 18 shows the bus structure for the data transfer between various registers and the common bus. As shown in the Fig. 18 each register has input and output tri-state buffers and these tri-state buffers are controlled by corresponding control signals. Control signals R_{in} and R_{out} controls the input and output gating of register R. When R_{in} is set to 1, the data available on the common data bus is loaded into register $R_{i\ in}$. Similarly, when $R_{i\ out}$ is set to 1, the contents of register Ri are placed on the common data bus. The signals $R_{i\ in}$ and $R_{i\ out}$ are commonly known as input enable and output enable signals of registers, respectively.



Flg.18 Input and output tri-state buffers for the register

We can transfer the data from r1 register to R2 register by

1. Activate the output enable signal of R_1, $R_{1\ out}$ = 1. This places the contents of R_1 on the common bus.
2. Activate the input enable signal of R_2, $R_{2\ out}$ = 1. This loads data from the common bus into the register R_2.

All operations and data transfers within the processor take place in synchronization with the clock signal. The control signals which controls a particular transfer are activated either at the rising edge or at the falling edge of the clock cycle.

Memory Transfer

A memory is a collection of storage cells. Each cell store l-bit of information. The memory stores binary information in groups of bits called words. To access information from a particular word from main memory each word in the main memory has distinct address. This allows to access any word from the main memory by specifying corresponding address. 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.

The number of words in the memory decides the size of the memory and the number of address bits. For example, 8-bit address can access upto 28 = 256 different words. The number of information bits can be read or written at a time is decided by the word length (numbers of bits in one word) of the memory. The Fig. 19 shows the typical connection between processor and memory. As shown in the Fig. 1.11.7 the control lines from the processor decides the memory operation.

In case of read operation Read signal is activated. It is used to enable the active low output enable signal of the memory. In case of write operation Write signal is activated to indicate the write operation. The data transfer between the memory and processor takes place through the use of two processor registers, usually called MAR (Memory Address Register) or simply AR (Address Register) and MDR (Memory Data Register) or simply DR (Data Register).

If MAR is k-bit long and MDR is n-bit long, it is possible to access upto 2^k memory locations and during one memory cycle, it is possible to transfer n bit data. In read operation, the address of the memory word is specified by address register, AR and the data word read from the memory is loaded into the data register, DR. This read operation can be represented as,

Read : DR ?M [AR]

Letter M represents memory word whose address is specified by address register, AR. The control signal Read indicates that the operation is performed only when Read signal is active. Once the data is available in the DR register it can be transferred to any other register within the CPU.

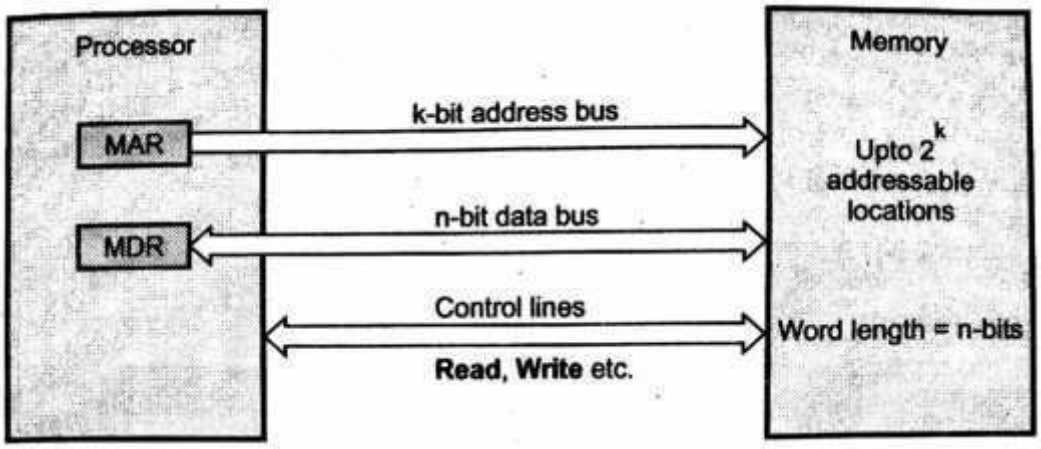


Figure 19. Connection between memory and processor

It is important to note that for above operation the activation of Read signal-is not necessary; it is necessary only when data is read from memory unit. The write operation transfers the contents of a

data register to a memory word M selected by the address in the address register, AR. The write operation can be represented as,

Write : M [AR] ? DR

The control signal Write indicates that the operation is performed only when Write signal is active. When it is necessary to transfer data from any other CPU register to the memory. we have to transfer the data from that register to the data register, DR before write operation.

The Fig. 20 shows the communication between memory unit and multiple registers. Register A0 through A3 are the address registers. The MUX 1 selects one of the address source for memory. The MUX 2 selects one of data source for write operation of memory. The decoder selects the destination register to read data from memory.

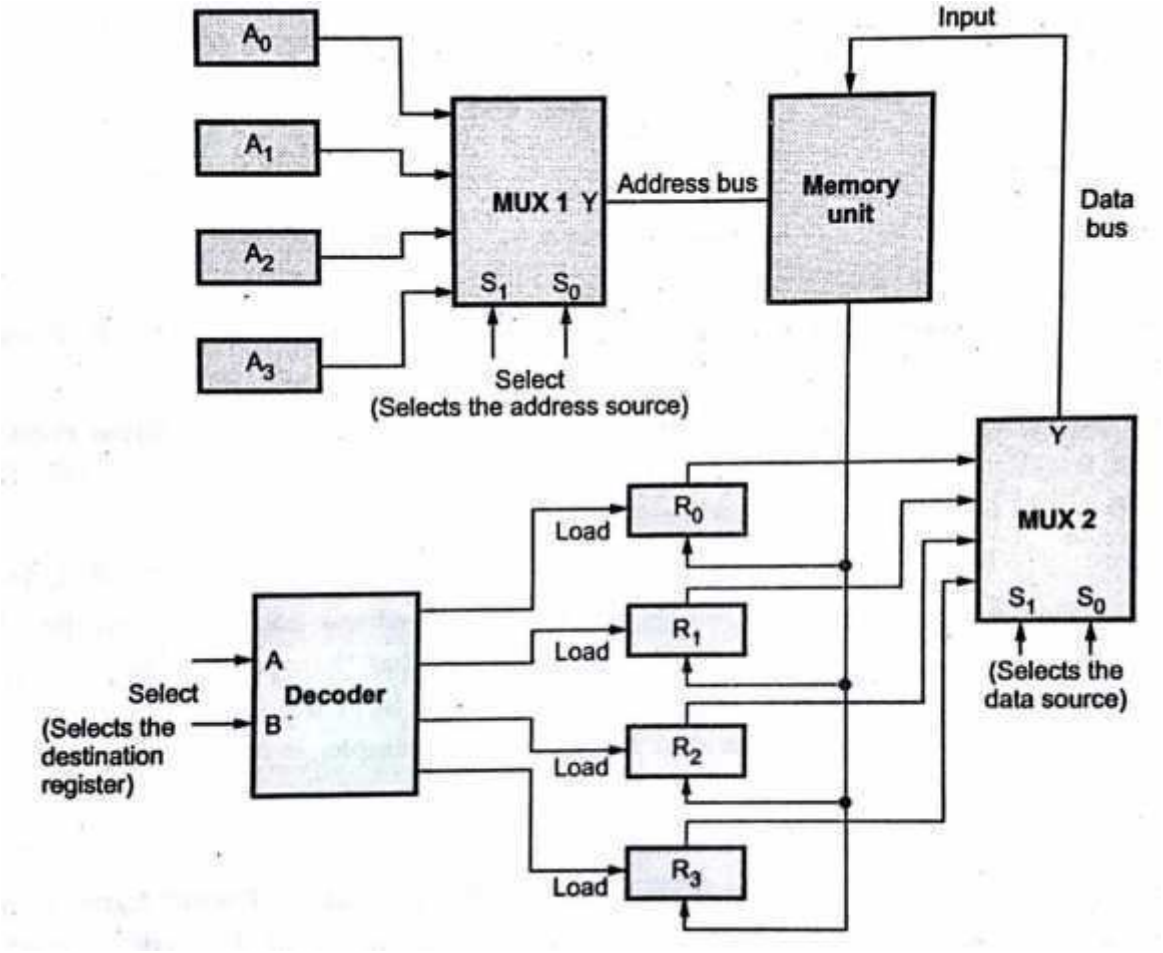


Figure 20 Communication between memory unit and multiple register

ADDRESSING MODES

The address of the memory can be specified directly within the instruction. For example, MOV [2000H], R1. In this instruction the memory address is fix, it can not be dynamically changed in the program itself. There are some situations where we need to change the memory address dynamically. For example program of array, to add the data and get the total sum of all array elements. In this we have to repeat the add instruction number of times equal to the array elements and each time we have to add a number from a new successive memory location. Every time the address of memory is different. So to change the address of memory each time when we enter the loop address variable is used. Such addressing is called indirect addressing.

The operation to be performed is specified by the operation field of the instruction. The execution of the operation is performed on some data stored in computer registers or memory words and the

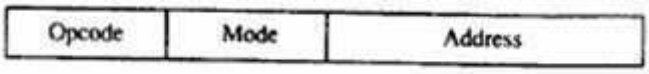
way the operands are chosen during program. Selection of operands during program execution depends on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referred. The purpose of using address mode techniques by the computer is to accommodate one or both of the following provisions:

1. To give programming flexibility 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.

Part of the programming flexibility for each processor is the number and different kind of ways the programmer can refer to data stored in the memory or I/O device. The different ways that a processor can access data are referred to as addressing schemes or addressing modes. Usage of addressing modes lends programming versatility to the user and helps to write program data mode affection in terms of instruction and implementation. The basic operation cycle of the computer must be understood to have a thorough knowledge of addressing modes. The instruction cycle is executive in the control unit of the computer and is divided into three principal phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Execute the instruction.

The program counter or PC register in the computer hold the instruction in the program stored memory. Each time when instruction is fetched from memory the PC is incremented, for it holds the address of the instruction to be executed next. Decoding involves determination of the operation to be performed, the addressing mode of the instruction, and the returns of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence. Figure show the distinct addressing mode field of in instruction format.



Instruction format with mode field

The operation code (opcode) specifies the operation to be performed. The mode field issue to locate the operands needed for the operation. An address field in an instruction may or may not be present.

1. Implied Mode: This mode specify the operands 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 references instructions that use an accumulator are implied mode instructions. Zero-address introductions are implied mode instructions.

2. Immediate Mode: The operand is specified in the instruction itself in this mode i.e. the immediate mode instruction has an operand field rather than an address field. The actual operand to be used in conjunction with the operation specified in the instruction is contained in the operand field.
   Example : MOVE A, #20

3. Register Mode: In this mode the operands are in registers that reside within the CPU. The register required is chosen from a register field in the instruction. Example: MOV R1, R2

4. Register Indirect Mode: In this mode the instruction specifies a register in the CPU that contains the address of the operand and not the operand itself. Usage of register indirect mode instruction necessitates the placing of memory address of the operand in the processor register with a previous instruction. EA=R
   Example: MOVE A, (R0)
5. Auto increment or Auto decrement Mode: After execution of every instruction from the data in memory it is necessary to increment or decrement the register. This is done by using the increment or decrement instruction. Given upon its sheer necessity some computers use special mode that increments or decrements the content of the registers automatically.
   Example: MOVE (R2), + R0
   MOVE (R2), - R0
6. Direct Address Mode: In this mode the operand resides in memory and its address is given directly by the address field of the instruction such that the affective address is equal to the address part of the instruction. Example: MOVE A, 2000
7. Indirect Address Mode: Unlike direct address mode, in this mode give the address field gives the address where the effective address is stored in memory. The instruction from memory is fetched through control to read is address part to access memory again to read the effective address. 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 equation:

   Effective address = address part of instruction + context of CPU register

   The CPU Register used in the computation may be the program counter, Index Register or a base Register.

8. Relative Address Mode: This mode is applied often with branch type instruction where the branch address position is relative to the address of the instruction word itself. As such in the mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address whose position in memory is relative to the address of the next instruction. Since the relative address can be specified with the smaller number of bits then those required the design the entire memory address, it results in a shorter address field in the instruction format. EA = PC + Address part of instruction
9. Indexed Addressing Mode: In this mode the effective address is obtained by adding the content of an index register to the address part of the instruction. The index register is a special CPU register that contains an index value and can be incremented after its value is used to access the memory. EA= offset + R
   Example: MOVE 20 [R1], R2
10. Base Register Addressing Mode: In this mode the effective address is obtained by adding the content of a base register to the part of the instruction like that of the indexed addressing mode though the register here is a base register and not a index register.
    The difference between the two modes is based on their usage rather than their computation. 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, and gives a displacement relative to this base address. The base register addressing mode is handy for relocation of programs in memory to another as required in multi programming systems. The address values of instruction must reflect this change of position with a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

Mode	Assembly Convention	Register Transfer
Direct Address	LD ADR	AC<-M[ADR]
Indirect Address	LD @ADR	AC<-M[M[ADR]]
Relative Address	LD $ADR	AC<-M[PC+ADR]
Immediate Operand	LD #NBR	AC<-NBR
Index Addressing	LD ADR(X)	AC<-M[ADR+XR]
Register	LD R1	AC<-R1
Register Indirect	LD(R1)	AC<-M[R1]
Auto increment	LD (R1)	AC<-M[R1],R1<-R1+1

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in