Unit 3 Basic of Computer Organization

3.1 System Bus

The CPU has to be able to send various data values, instructions, and information to all the devices and components inside your computer as well as the different peripherals and devices attached. If you look at the bottom of a motherboard you'll see a whole network of lines or electronic pathways that join the different components together. These electronic pathways are nothing more than tiny wires that carry information, data and different signals throughout the computer between the different components. This network of wires or electronic pathways is called the 'Bus'.

All communication between the individual major components is via the system bus. The bus is merely a cable which is capable of carrying signals representing data from one place to another. The bus within a particular individual computer may be specific to that computer or may (increasingly) be an industry-standard bus. If it is an industry standard bus then there are advantages in that it may be easy to upgrade the computer by buying a component from an independent manufacturer which can plug directly into the system bus. For example most modern Personal Computers use the PCI bus.

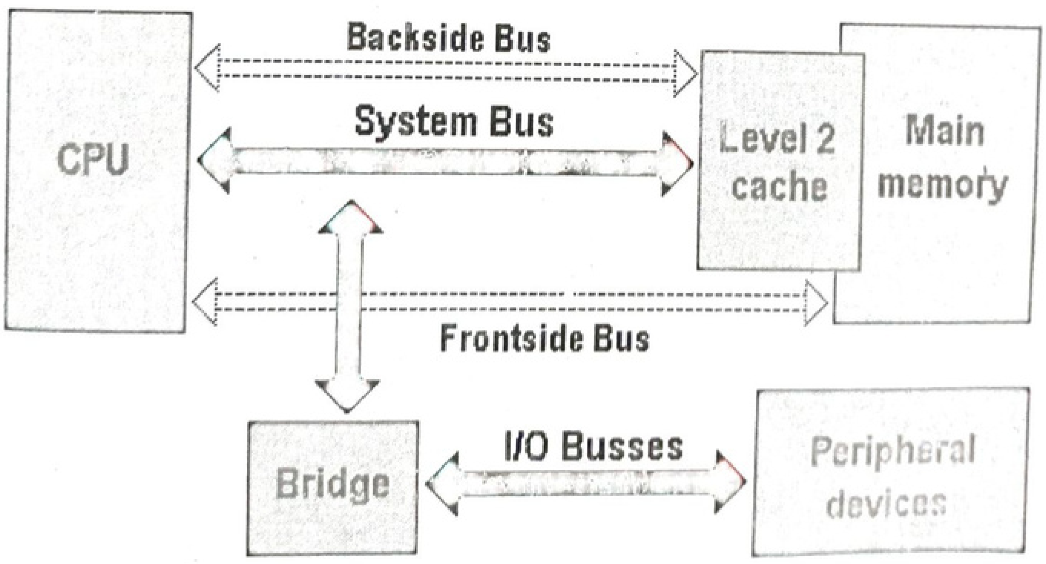


Fig. 1 : System Bus

A computer's bus can be divided into two different types, Internal and External. The Internal Bus connects the different components inside the case: The CPU, system memory, and all other components on the motherboard. It's also referred to as the System Bus. The External Bus connects the different external devices, peripherals, expansion slots, I/O ports and drive connections to the rest of the computer. In other words, the External Bus allows various devices to be added to the computer. It allows for the expansion of the computer's capabilities. It is generally slower than the system bus. Another name for the External Bus is the Expansion Bus.

After this study we can divide the system bus in three types. These are as follows:

3.2.2 Address bus

3.2.3 Data bus

3.2.4 Control bus

Now we can discuss one by one as in following way:

3.1.1 Address bus:

Data is stored, manipulated and processed in system memory at various locations. System memory is like a large sea of information full of fish (data). Our computer has to move information in and out of memory, and it has to keep track of which data is stored where. The computer knows where all the fishes are, but it has to transmit that information to the CPU and other devices. It has to keep a map of the different address locations in memory, and it has to be able to transmit and describe those memory locations to the other components so that they can access the data stored there. The info used to describe the memory locations travels along the address bus. The size or width of the address bus directly corresponds to the number of address locations that can be accessed. This simply means that the more memory address locations that a processor can address, the more RAM it has the capability of using. It makes sense, right? So An address bus is a computer bus that is used to specify a physical address. When a processor or DMA (Direct Memory Access)-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus). The width of the address bus determines the amount of memory a system can address. For example, a system with a 32-bit address bus can address 232 (4,294,967,296) bytes, or 4 GB, of memory. Early processors used a wire for each bit of the address width. For example, a 16-bit address bus had 16 physical wires making up the bus. As the bus becomes wider, this approach becomes less convenient and more expensive to implement. Instead, some modern processors make the address bus faster than the data bus, and send the address in two parts. For example a 32-bit address bus can be implemented by using 16 wires and sending the first half of the memory address, immediately followed by the second half. For example A 286 processor with a 16 bit address bus can access over 16 million locations, or 16 Mb of RAM. A 386 CPU with a 32 bit address bus can access up to 4 GB of RAM. Of course, at the present time, due to space and cost limitations associated with the average home computer, 4GB of RAM is not practical. But, the address bus could handle it if it wanted to! **Address bus is also known as memory bus.**

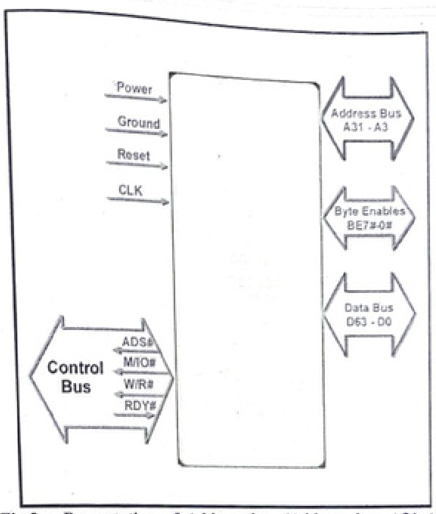


Fig.2 : Presentation of Address bus (Address bus A31-A3)

3.1.2 Data Bus

A data bus is a group of wires connecting different parts of a circuit with wire carrying a different signal. The data bus is connected to the inputs of several gates and to the outputs of several gates. A data bus may be time multiplexed to serve different functions at different times. At any time only one gate may drive information onto the bus line but several gates may receive it. In general, information may flow on the bus wires in both directions. This type of bus is referred to as a bidirectional data bus.

There are two types of data bus; they are address bus and a data bus. The data bus transfers data and address bus transfers information or address about where the data should go. A bus which allows data to be transferred faster, which makes applications run faster. The local bus is of high-speed that connects directly to the processor. Many military systems are compatible with the 1553, but most new commercial systems are not. This system is used for command and telemetry transfer between military spacecraft components, subsystems and instruments, and within complex components themselves.

The main work of the data bus is to carry digital information. A data bus can be viewed as a group or collection of wires connecting different parts of a circuit with wire carrying a different signal. The data bus is connected to the inputs of several gates and to the outputs of several gates. A data bus may be time multiplexed to serve different functions at different times. The ISA architecture which is also developed by IBM is used for defector standard, and is widely used for high performance. Thus, the main function of the data bus is to connect the 'system to the external devices. There are many types of data bus, which can connect the PC to the mobile and other external device in order to download software's. Thus data bus is a media which is transfer the data from one place to another.

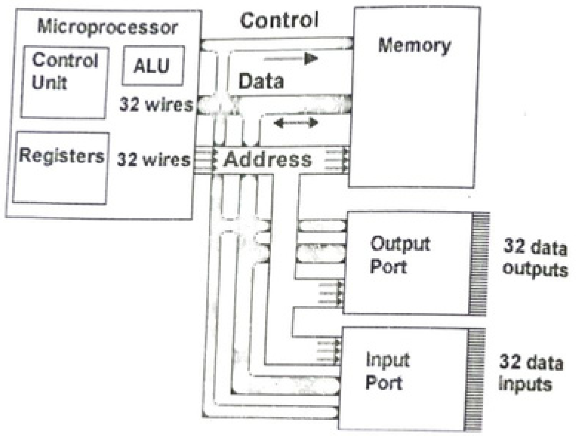


Fig. 3: Representation of Data Bus

3.1.3 Control Bus

A control bus is a computer bus, used by CPUs for communicating with other devices within the computer. While the address bus carries the information on which device the CPU is communicating with and the data bus carries the actual data being processed, the control bus carries commands from the CPU and returns status signals from the devices, for example if the data is being read or written to the device the appropriate line (read or write) will be active.

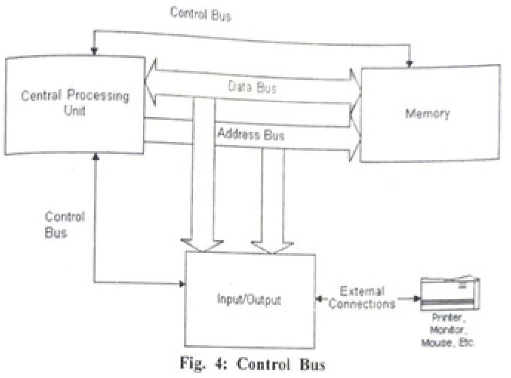


Fig. 4: Control Bus

3.2. Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going thought a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases :

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2 and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: $AR \leftarrow PC$

T_1: $IR \leftarrow M[AR], PC \leftarrow PC + 1$

T_2: $D_0 \ldots\ldots\ldots D_1 \leftarrow \text{Decode } IR\ (12 - 14), AR \leftarrow IR\ (0 - 11)\ 1 \leftarrow IR(15)$

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T_0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T_1. At the same time, PC is incremented by one to prepare if for the address of the next instruction in the program. At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. Note that SC is incremented after each clock pulse to produce the sequence T_0, T_1 and T_2.

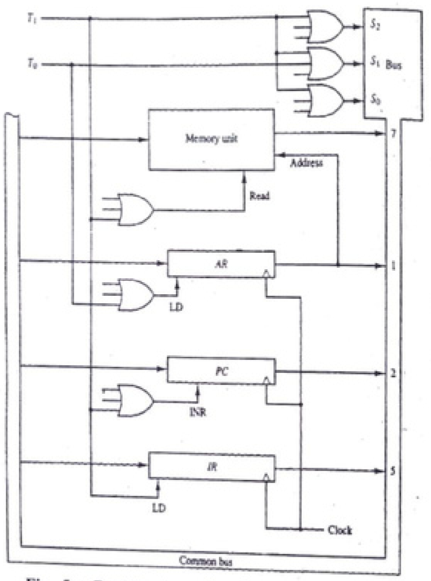


Fig. 5 : Register transfers for the fetch phases.

Figure shows how the first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection.

1. Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.

2. Transfer the content of the bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since $T_0 = 1$. In order to implement the second statement

T_1: $IR \leftarrow M[AR],\ PC \leftarrow PC + 1$

it is necessary to use timing signal T_1 to provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since $T_1 = 1$.

Figure duplicates a portion of the bus system and shows how T_0 and T_1 are connected to the control inputs of the registers, the memory, and the bus selection inputs. Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory. The flowchart of Fig. presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. The three possible instruction types available in the basic computer are specified in Fig.

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. From Fig. we determine that if $D_7 = 1$, the instruction must be a register-reference or input-output type. If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory reference instruction. Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I. If $D_7 = 0$ and $I = 1$; we have a memory reference instruction with an indirect address. It is then necessary to read the effective address from memory. The microoperation for the indirect address condition can be symbolized by the register transfer statement.

$AR \leftarrow M\ [AR]$

Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows :

D'_7IT_3 :	$AR \leftarrow M[AR]$
D'_7IT_3:	Nothing
$D_7I'T_3$:	Execute a register-reference instruction
D_7IT_3:	Execute an input-output instruction

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when $D'_7T_3 = 1$, so that the execution of the memory reference instruction can be continued with timing variable T_4. A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.

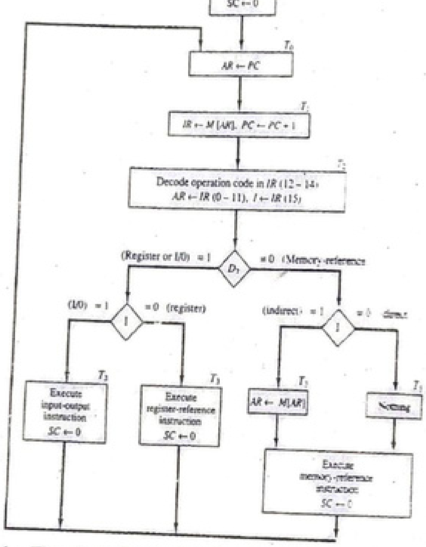


Fig. 6 : Flowchart for instruction cycle (initial configuration)

Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. We will adopt the convention that if SC is incremented, we will not write the statement SC ← SC + 1, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared we will include the statement SC ← 0.

The register transfers needed for the execution of the register-reference instructions are presented in this section. The memory reference instructions are explained in the next section. The input-output instructions are included in Sec. 5-7.

Register-Reference Instructions

Register-reference instructions are recognized by the control when $D_7 = 1$ and $I = 0$. These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in [R(0-1)]. They were also transferred to AR during time T_2.

The control functions and microoperations for the register-reference instructions are listed in Table. These instructions are executed with the clock transition associated with timing variable T_3. Each control function needs the Boolen relation $D_7I'T_3$, which we designate for convenience by the symbol r. The control function is distinguished by one of the bits in [R(0 –11)]. By assigning the symbol B_i to bit i of IR, all control functions can be simply denoted by rB. For example, the instruction CLA has the hexadecimal code 78000 (see table), which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent to I'. The next three bits constitute the operation code and are recognized from decoder output D_7. Bit 11 in IR is 1 and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is $D_7I'T_3\,B_{11} = rB_{11}$. The execution of a register-reference instruction is completed at time T_3. The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T_0.

Table : Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in IR (0–11) that specifies the operation]

	r: $SC \leftarrow 0$	Clear SC
CLA	rB_{11}: $AC \leftarrow 0$	Clear AC
CLE	rB_{10}: $E \leftarrow 0$	Clear E
CMA	rB_9 : $AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8 : $E \leftarrow E$	Complement E
CIR	rB_7 : $AC \leftarrow \text{shr}\,AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6 : $AC \leftarrow \text{shl}\,AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5 : $AC \leftarrow AC + 1$	Increment AC
SPA	rB_4 : If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3 : If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2 : If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1 : If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0 : $S \leftarrow 0$ (S is a start-stop flip-flop)	Halt computer

The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers. The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing PC once again (in addition, it is being incremented during the fetch phase at time T_1). The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC (15) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if allthe flip-flops of the register are zero. The HLT instruction clears a start stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

3.3. MEMORY SUBSYSTEM ORGANIZATION

In order to specify the microoperation needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Looking back to table, where the instruction are listed, we find that some instructions have a ambiguous description. This is because the explanation of an instruction in workds is usually lengthy, and not enough space is available in the table for such a lengthy explanation. We will now show that the function of the memory-reference instructions can be defined precisely by means of register transfer notation.

Table lists the seven memory-reference instructions. The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 rom the operation decoder that belongs to each instruction is included in the table. The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1. The execution of the memory-reference instructions starts with timing signal T_4. The symbolic description of each instruction is specified in the table in terms of register transfer notation. The actual execution of the instruction in the bus system will require a sequence of microoperations. This is because data stored in memory cannot be processed directly. The data must be read from memory to a register where they can be operated on with logic circuits. We now explain the operation of each instruction and list the control functions and microoperation needed for their execution. A flowchart that summarizes all the microoperations is presented at the end of this section.

Table : Memory Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AV \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR],\ E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC,\ PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC : This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC. The microoperations that execute this instruction are :

$$D_0T_4:\ DR \leftarrow M[AR]$$
$$D_0T_5:\ AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$$

The control function for this instruction uses the operation decoder D_0 since this output of the decoder is active when the instruction has an AND operation whose binary code value is 000. Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T_4 transfers the operand from memory into DR. The clock transition associated with the next timing signal T_5 transfers to AC the result of the AND logic operation between the

contents of DR and AC. The same clock transition clears SC to 0, transferring control to timing signal T_0 to start a new instruction cycle.

ADD to AC : This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are

$$D_1T_4:\ DR \leftarrow M[AR]$$
$$D_1T_5:\ AC \leftarrow AC + DR,\ \ E \leftarrow C_{out},\ SC \leftarrow 0$$

The same two timing signals, T_4 and T_5, are used again but with operation decoder D_1 instead of D_0 which was used for the AND instruction. After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory reference instruction.

LDA Load to AC : This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

$$D_2T_4:\ DR \leftarrow M[AR]$$
$$D_2T_5:\ AC \leftarrow DR,\ SC \leftarrow 0$$

Looking back at the bus system shown in Fig. we note that thee is no direct path from the bus into AC. The adder and logic circuit receive information from DR which can be transferred into AC. Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC. The reason for not connecting the bus tothe inputs of AC is the delay encountered in the adder and logic circuit. It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle. By not connecting the bus to the inputs of AC we can maintain one clock cycle per microoperation.

STA : Store AC : This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation :

$$D_3T_4:\ M[AR] \leftarrow AC,\ SC \leftarrow 0$$

BUN : Branch Unconditionally : This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation :

$$D_4T_4:\ PC \leftarrow AR,\ SC \leftarrow 0$$

The effective address from AR is transferred through the common bus to PC. Resetting SC to 0 transfers control to T_0. The next instruction is then fetched and executed from the memory address given by the new value in PC.

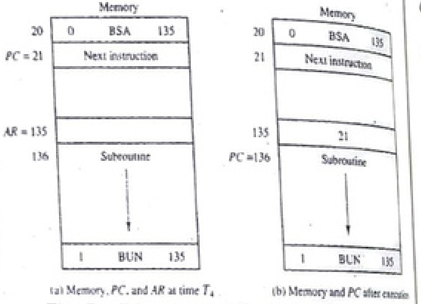
BSA : Branch and Save Return Address : This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in Table with the following register transfer :

$$M[AR] \leftarrow PC,\ PC \leftarrow AR + 1$$

A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. The BSA instruction is assumed to be in memory at address 20. The 1 bit is 0 and the address part of the instruction has the binary equivalent of 135. After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135. This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation :

$$M(135) \leftarrow 21,\ PC \leftarrow 135 + 1 = 136$$

The result of this operation is shown in part (b0 of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine. When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. When the BUN instruction is executed, the effective address 21 is transferred to PC. The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.



(a) Memory, PC, and AR at time T_4 (b) Memory and PC after execution

Fig. 7 : Example of BSA instruction execution

The BSA instruction performs the function usually referred to as a subroutine call. The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return. In most commercial computers, the return address associated with a subroutine is stored in either a processor register in a portion of memory called a stack. This is discussed in more detail in Sec. 8-7.

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations :

$$D_5T_4\text{: } M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$$
$$D_5T_5\text{: } PC \leftarrow AR,\ SC \leftarrow 0$$

Timing signal T_4 initiates a memory write operation places the content of PC into the bus, and enables the INR input of AR. The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T_3 to transfer the content of AR to PC.

ISZ : Increment and Skip if Zero : This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PCis incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations :

$$D_6T_4\text{: } DR \leftarrow M[AR]$$
$$D_6T_5\text{:. } DR \leftarrow DR + 1$$
$$D_6T_6\text{: } M[AR] \leftarrow DR,\ \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1),\ SC \leftarrow 0$$

Control Flowchart

A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. The control functions are indicated on top of each box. The

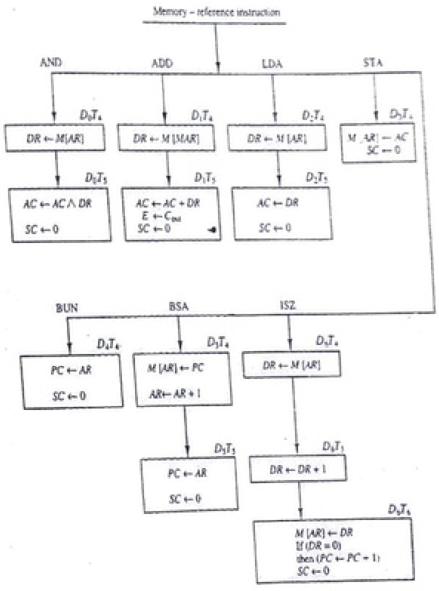


Fig. 8 : Flowchart for memory-reference instructions

microoperatiing that are performed during time T_4, T_5, or T_6 depend on the operation code value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded. The sequence counter SC is cleared to 0 with the last timing signal in each case. This causes a transfer of control to timing signal T_0 to start the next instruction cycle.

Note that we need only seven timing signals to execute the longest instruction (ISZ). The computer can be designed with a 3-bit sequence counter. The reason for using a 4-bit counter for SC is to provide additional timing signals for other instructions that are presented in the problems section.

Input-Output and Interrupt

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device. Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

3.4 Input-Output Subsystem Organization

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The input-output configuration is shown in Fig. The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver serial interface receives information from OUTR and sends it to the printer serially. The operation of the serial communication interface is explained in Sec. 11-3.

The input register INPR consists of eight bits and holds an alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer. The flag is needed to synchronize the timing rate difference between the input device and the computer. The process of information transfer is as follows. Initially the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1. As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if its 1 the information from INPR is transferred in parallel into AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed it sets FGO to 1. The computer does not load a new character into outer when FGO is 0 because this conditions indicates that the output device is in the process of printing the character.

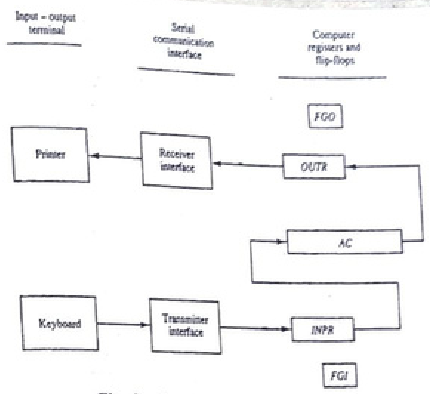


Fig. 9 : Input-output configuration

Input-Output Instructions : Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 111 and are recognized by the control when $D_7 = 1$ and $I = 1$. The remaining bit of the instruction specify the particular operation. The control functions and microoperation specify the particular operation. The control functions and microoperation for the input-output instructions are listed in table. These instructions are executed with the clock transition associated with timing signal T_3. Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol p. The control function is distinguished by one of the bits in [R(6-11). By assigning the symbol B, to bit i of IR, all control functions can be denoted by pB, for i = 6 though 11. The sequence counter SC is cleared to 0 when $p = D_7IT_3 = 1$.

$D_7IT_3 = p$ (common to all input-output instructions)

$IR(i) = B_i$ [bit in IR (6–11) that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTER \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9 :	If$(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8 :	If$(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7 :	$IEN \leftarrow 1$	Interrupt enable on
IOF	rB_6 :	$IEN \leftarrow 0$	Interrupt enable off

The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0. The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0. The next two instructions in Table check the status of the flags and cause a skip of the next instruction if the flag is 1. The instruction that is skipped will normally be a branch instruction to return and check the flag again. The branch instruction is not skipped if then flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed. (Examples of input and output programs are given in Sec 6.8). The last two instructions set and clear an interrupt enable flip-flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

Program Interrupt

The process of communication just described is referred to as programmed control transfer. The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer. The difference of information flow rate between the computer and that of the input-output device makes this type of transfer inefficient. To see why this is inefficient, consider a

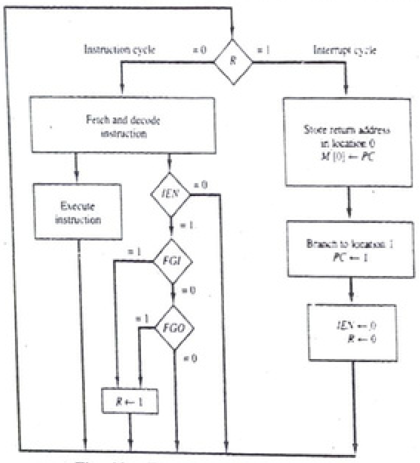


Fig. 10 : Flowchart for interrupt cycle

computer that can go through an instruction cycle is 1 μs. Assume that the input-output device can transfer information at a maximum rate of 10 characters per second. This is equivalent to one character every 1,00,000 μs. Two instructions are executed when the computer checks the flag bit and decides not to transfer the information. This means that at the maximum rate, the computer will check the flag 50,000 times between each transfer. The computer is wasting time

while checking the flag instead of doing some other useful processing task.

An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility. While the computer is running a program, it does not check the flags. However, when a flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set. The computer deviates momentarily from what it is doing to take care of the input or output transfer. It then returns to the current program to continue what it was doing before the interrupt.

The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be interrupted. These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.

The way that the interruptus handled by the computer can be explained by means of the flowchart of Fig. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. During the execute phase of the instruction cycle IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that nightjar the input nor the output registers are ready of transfer of information. n this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flip R is set to 1. At the end of the execute phase, control checks the value of R, and fit is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

The interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R sot that no more interruptions can occur until the interrupt request from the flag has been serviced.

An example that shows what happens during the interrupt cycle is shown in Fig. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction of address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig..

When control reaches timing signal T_0 and finds that R = 1, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0. OC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction of address 1 causes the program to transfer to the input-output service program at address 1120. This program checks the flags, determines which flag is set, and then transfers the required input o output information.

Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location wheel it was interrupted. This is shown in Fig..

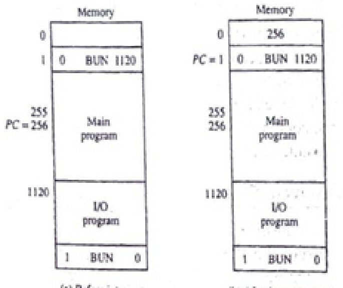


Fig. 11 : Demonstration of the interrupt cycle

The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O service program. After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address. The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle. The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Interrupt Cycle

We are now ready to list the register transfer statements for the interrupt cycle. We interrupt cycle is initiated after the last execute phase if the interrupt flip-flip R is equal to 1. This flip-Flop is set to 1 if IEN = 1 and either FGI of FGO are equal to 1. This can happen with any clock transition except when timing signals T_0, T_1 or T_2 are active. The condition for setting flip-flop R to 1 can be expressed with the following register transfer statement :

$$T'_0T'_1T'_2 \text{ (IEN(FGI + FGO):} \qquad R \leftarrow 1$$

The symbol + between FGI and FGO in the control function designates a logic OR operation. This is ANDed with IEN and $T'_0T'_1T'_2$:

We now modify the fetch and decode phases of the instruction cycle. Instead of using only timing signals T_0, T_1 and T_2 (as shown in Fig.) we will AND the three timing signals with R' so that the fetch and decode phases will be recognized from the three control functions $R'T_0$, $R'T_1$ and $R'T_2$. The reason for this is that after the instruction is executed and SC is cleared to 0, the control will go through a fetch phase only if R = 0. Otherwise, if R = 1, the control will go through an interrupt cycle. The interrupt cycle stores the return address (available in PC) into memory location 0, branches to memory location 1, and clears IEN R, and SC to 0. This can be done with the following sequence of microoperations :

RT_0: $AR \leftarrow 0$, $TR \leftarrow PC$

RT_1: $M[AR] \leftarrow TR$, $PC \leftarrow 0$

RT_2: $PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$

During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR. With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0. The third timing signal increments PC to 1, clears IEN and R, and control goes back to T_0 by clearing SC to 0. The beginning of the next instruction cycle has the condition $R'T_0$ and the content of PC is equal to 1. The control then goes though an instruction cycle that fetches and executes the BUN instruction in location 1.

3.5 Complete Computer Description

The final flowchart of the instruction cycle, including the interrupt cycle for the basic computer, is shown in Fig. The interrupt flip-flop R may be set at any time during the indirect or execute phases. Control returns to timing signal T_0 after SC is cleared to 0. If R = 1, the computer goes through an interrupt cycle. If R = 0, the computer goes through an instruction cycle. If the instruction is one of the memory-reference instruction, the computer first check if thee is an indirect address and then continues to execute the decoded instruction according to the flowchart of Fig. If the instruction is one of the register-reference instructions, it is executed with one of the microoperations listed in table. If it is an input-output instruction, it is executed with one of the microoperations listed in Table.

Instead of using a flowchart, we can describe the operation of the computer with a list of register transfer statements. This is done by accumulating all the control functions and microoperations in one table. The entries in the table are taken from figs. and tables

The control functions and microoperations for the entire computer are Islamized in Table. The register transfer statements in this table describe in a concise from the internal organization of the basic computer. They also give all the information necessary for the design of the logic circuits of the computer. The control functions and conditional control statements listed in the table formulate the Boolean functions for the gates in the control unit. The list of microoperations specifies the type of control inputs needed for the registers and memory. A register transfer language is useful not only for describing the internal organization of a digital system but also for specifying the logic circuits needed for its design.

3.6 Register Transfer Languages

A digital system is an interconnected of digital hardware modules that accomplish a specific information processing task. Digital system vary in size and complexity from a few integrated circuits to a complex of interconnected and interacting digital computers. Digital system design invariably uses a modular approach. The modules are constructed from such digital components as register, decodes, arithmetic elements and control logic. The various modules are interconnected with common data and control paths to form a digital computer system.

Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations executed on data stored in registers are called microoperations. A microoperation is an elemental operation performed on the information stored in one are more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another registers : Examples of microoperations are shift, count, clear and load. Some of the digital components are registers that implement microoperations for example a counter with parallel load is capable of performing the microoperations increments and load. A bidirectional shift register is capable of perfuming the shift right and shift left microoperations.

The internal hardware organization of a digital computer is best defined by specifying :

1. The set of register it contain and the function.
2. The sequence of microoperation performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

The symbolic notation used to describe the microoperation transfer among registers is called a register transfer language. The term "register transfer" implies the availability of hardware logic circuits that can perform a another register. The word language is borrowed from programmers, who apply this term to programming language. A programming language is a procedure for writing symbols to specify a given computational process. Similarly a natural language such as English is a system for writing symbols and combining then into words and sentences for the purpose of communication.

"A register transfer language is a system for expressing in symbolic form the microoperation sequence among the register of a digital module."

It is a convenient tool for describing the internal organization of a digital computer in concise and precise manner. It can also be used to facilitate the design process of digital system.

Register transfer : In computer science, register transfer language (RTL) is a kind of intermediate representation (I) that is very close to assembly language, such as that which is used in a compile. It is basically an operation which is used to transfer information from one place to another Academic papers and textbooks also often use a from of RTL as an architecture neutral assembly language.

RTL is also name of a specific IR used in the GNU compiler collection (GCC) and several other compilers such as zephyr or the European compiler project cerco and composure.

3.7 Design of Basic Computer

The basic computer consists of the following hardware components:

1. A memory unit with 4096 words of 16 bits each
2. Nine registers: AR, PC, DR, AC, IR, TR, OUTR, INPR and SC
3. Seven flip-flop: I, S, E, R, IEN, FGI and FGO
4. Two decoders : a 3 × 8 operation decoder and a 4 × 16 timing decoder
5. A 16-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of AC

The memory unit is a standard component that can be obtained readily from a commercial source. The registers are of the type shown in Fig. and are similar to integrated circuit type 74163. The flip-flops can be either of the D or JK type, as described in Sec. 1-6. The two decoders are standard components similar to the ones presented in Sec. 2-2. The common bus system can be constructed with sixteen 8 × 1 multiplexers in a configuration similar to the one shown in Fig. 4-3. We are going to show how to design the control logic gates. The next section deals with the design of the adder and logic circuit associated with AC.

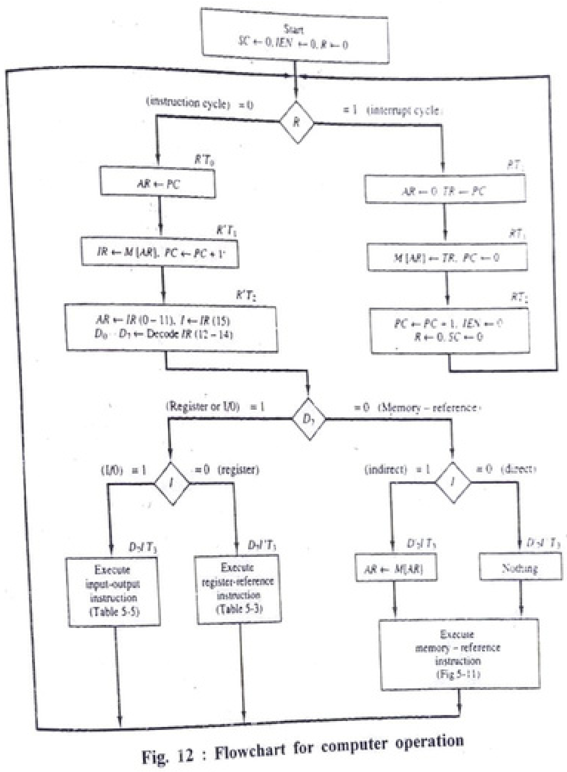


Fig. 12 : Flowchart for computer operation

Table : Control function and Microoperations for the Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
Decode	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
	$R'T_2$:	$D_0, \ldots\ldots, D_7 \leftarrow$ Decode $IR(12-14)$,
		$AR \leftarrow IR(0-11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt :		
$T'_0T'_1T'_2(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-reference :		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out} SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$.
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, If $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Register-reference:		
$D_7I'T_3 = r$ (common to all register-reference instructions)		
$IR(i) = B_i$ [bit in IR (0–11) that specifies the operation]		
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9 :	$AC \leftarrow \overline{AC}$
CME	rB_8 :	$E \leftarrow \overline{E}$
CIR	rB_7 :	$AC \leftarrow$ shr $AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6 :	$AC \leftarrow$ shl $AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5 :	$AC \leftarrow AC + 1$
SPA	rB_4 :	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3 :	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2 :	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1 :	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0 :	$S \leftarrow 0$
Input-output		
$D_7IT_3 = p$ (common to all input-output instructions)		
$IR(i) = B_i$ [$i = 6, 7, 8, 9, 10, 11$)		
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0-7) \leftarrow INPR, \quad FGI \leftarrow 0$
OUT	pB_{10}:	$OUTER \leftarrow AC(0-7), FGO \leftarrow 0$
SKI	pB_9 :	If$(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8 :	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7 :	$IEN \leftarrow 1$
IOF	rB_6 :	$IEN \leftarrow 0$

CONTROL LOGIC GATES

The block diagram of the control logic gates is shown in Fig. The inputs to this circuit come from the two decoders, the I flip-flop, and bits 0 through 11 of IR. The other inputs to the control logic are : AC bits 0 through 15 to check if AC = 0 and to detect the sign bit in AC (15); DR bits 0 through 15 to check if DR = 0; and the values of the seven flip-flops.

The outputs of the control logic circuit are :

1. Signals to control the inputs of the nine registers.
2. Signals to control the read and write inputs of memory.
3. Signals to set, clear, or complement the flip-flops.
4. Signals for S_2, S_1 and S_0 to select a register for the bus
5. Signals to control the AC adder and logic circuit.

The specifications for the various control signals can be obtained directly from the list of register transfer statements in Table.

7.8 CONTROL OF REGISTERS AND MEMORY

The registers of the computer connected to common bus system are shown in Fig.. The control inputs of the registers are LD (load), INR (increment), and CLR (clear). Suppose that we want to derive the gate structure associated with the control inputs of AR. We scan Table to find all the statements that change the content of AR:

$R'T_0$:	$AR \leftarrow PC$
$R'T_1$:	$AR \leftarrow IR(0-11)$
D'_7IT_3:	$AR \leftarrow M[AR]$
RT_0:	$AR \leftarrow 0$
D_5T_4:	$AR \leftarrow AR + 1$

The first three statements specify transfer of information from a register or memory to AR. The content of the source register or memory is placed on the bus and the content of the bus is transferred into AR by enabling its LD control input. The fourth statement clears AR to 0. The last statement increments AR by 1. The control functions can be combined into three Boolen expulsions as follows :

$$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$

where LD(AR) is the load input of AR, CLR(AR) is the clear input of AR, and INR(AR) is the increment input of AR. The control gate logic associated with AR is shown in Fig.

In a similar fashion we can derive the control gates for the other registers as well as the logic needed to control the read and write inputs of memory. The logic gates associated with the read input of memory is derived by scanning Table to find the statements that specify a read operation. The read operation is recognized from the symbol $\leftarrow M[AR]$

$$\text{Read} = R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)T.$$

The output of the logic gates that implement the Boolean expression above must be connected to the read input of memory.

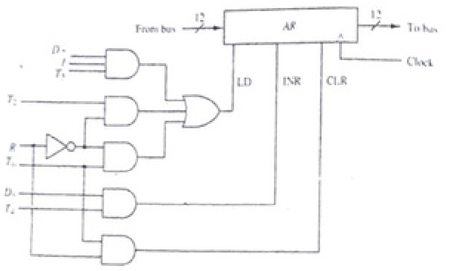


Fig. 13 : Control gates associated with AR.

Control of Single Flip-flops

The control gates for the seven flip-flops can be determined in a similar manner. For example, Table shoes that IEN may change as a result of the two instructions ION and IOF.

$$pB_7: \quad IEN \leftarrow 1$$
$$pB_6: \quad IEN \leftarrow 0$$

where $p = D_7IT_3$ and B_7 and B_6 are bits 7 and 6 of IR, respectively. Moreover, at the end of the interrupt cycle IEN is cleared to 0.

$$RT_2: \quad IEN \leftarrow 0$$

If we use of a JK flip-flop for IEN, the control gate logic will be as shown in Fig.

Control of Common Bus

The 16-bit common bus shown in Fig. is controlled by the selection inputs S_2, S_1 and S_0. The decimal number shown with each bus input specifies the equivalent binary number that must be applied to the selection inputs in order to select the corresponding register. Table specifies the binary numbers for S_2, S_1 and S_0. The decimal number shown with each bus input specifies the equivalent binary number that must be applied to the selection inputs in order to select the corresponding register. Table specifies the binary numbers of S_2, S_1 and S_0 that select each register. Each binary number is associated with a Boolean variable X_1 through X_7, corresponding to the gate structure that must be active in order to select the register or memory for the bus. For example, when $x_1 = 1$, the value of $S_2S_1S_0$ must be 001 and the output of AR will be selected for the bus. Table is recognized as the truth table of a binary encoder. The placement of the encoder at the inputs of the bus selection logic is shown in Fig. The Boolean functions for the encoder are

$$S_0 = x_1 + x_3 + x_5 + x_7$$
$$S_1 = x_2 + x_3 + x_6 + x_7$$
$$S_2 = x_4 + x_5 + x_6 + x_7$$

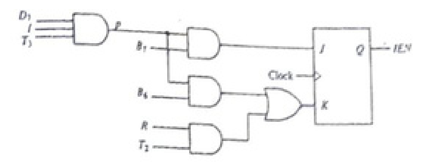


Fig. 14 : Control inputs for IEN

Table : Encoder for Bus Selection Circuit

Inputs							Output			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

To determine the logic for each encoder input, it is necessary to find the control functions that place the corresponding register onto the bus. For example, to find the logic that makes $x_1 = 1$, we scan all register transfer statements in Table and extract those statements that have AR as a source.

$$D_4T_4: \quad PC \leftarrow AR$$
$$D_5T_5: \quad PC \leftarrow AR$$

Therefore, the Boolean function for x_1 is

$$x_1 = D_4T_4 + D_5T_5$$

The data output from memory are selected for the bus when $x_7 = 1$ and $S_2S_1S_0 = 111$. The gate logic that generates x_7 must also be applied to the read input of memory. Therefore, the Boolean function for x_7 is the same as the one derived previously for the read operation.

$$x_7 = R'T_1 + D'_7IT_3 + (D_0 + D_2 + D_6)T_4$$

In a similar manner we can determine the gate logic for the other registers.

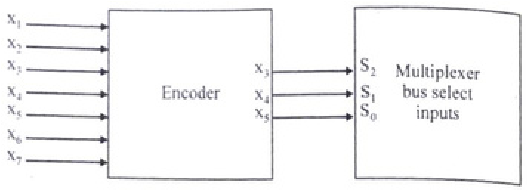


Fig. 15 : Encoder for the base selection input

3.9 Design of Accumulator Logic

The circuits associated with the AC register are shown in Fig. The adder and logic circuit has three sets of inputs. One set of 16 inputs comes from the outputs of AC. Another set of 16 inputs comes from the data register DR. A third set of eight inputs comes from the input register INPR. The outputs of the adder and logic circuit provide the data inputs for the register. In addition it is necessary to include logic gates for controlling the LD, INR and CLR in the register and for controlling the operation of the adder and logic circuit.

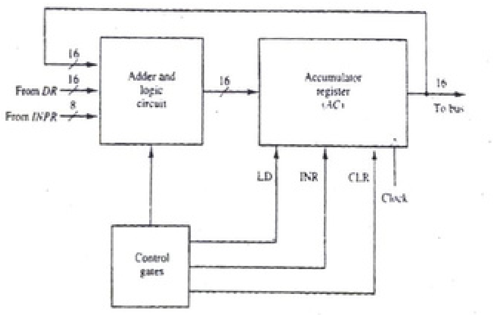


Fig. 16 : Circuits associated with AC

In order to design the logic associated with AC, it is necessary to go over the register transfer statements in Table and extract all the statements that change the content of AC.

D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5:	$AC \leftarrow AC + DR$	Add with DR
D_2T_5:	$AC \leftarrow DR$	Transfer from DR

pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shr } AC, AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

From this list we can derive the control logic gates and the adder and logic circuit.

Control of AC Register

The gate structure that controls the LD, INR, and CLR inputs of AC is shown in Fig.. The gate configuration is derived from the control functions in the list above. The control function for the clear microoperation is rB_{11}, where $r = D_7I'T_3$ and $B_{11} = IR(11)$. The output of the AND gate that generates this control function is connected to the CLR input of the register. Similarly, the output of the gate that implements the increment microoperation is connected to the INR input of the register. The other seven microoperations are generated in the adder and logic circuit and are loaded into AC at the proper time. The outputs of the gates for each control function is marked with a symbolic name. These outputs are used in the design of the adder and logic circuit.

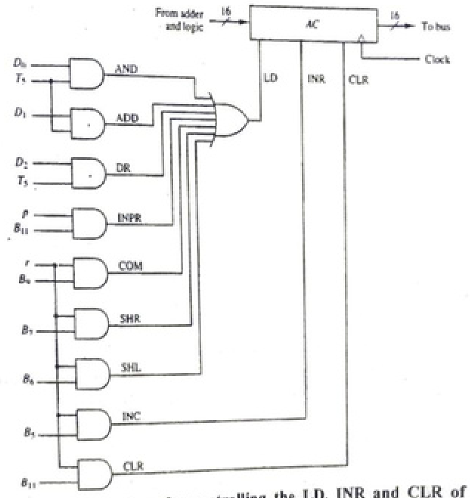


Fig. 17 : Gate structure for controlling the LD, INR and CLR of AC.

Adder and Logic Circuit

The adder and logic circuit can be subdivided into 16 stages, with each stage corresponding to one bit of AC. The internal construction of the register is as shown in Fig. Looking back at that figure we note that each stage has a JK flip-flop, two OR gates, and two AND gates. The load (LD) input is connected to the inputs of the AND gates. Figure shows one such AC register stage (with the OR gates removed). The input is labeled I_i and the output AC(i). When the LD input is enabled, the 16 inputs I_i for i = 0, 1, 2.......... 15 are transferred to AC (0-15).

One stage of the adder and logic circuit consists of seven AND gates, one OR gate and a full-adder (FA) as shown in Fig. The inputs of the gates with symbolic names come from the outputs of gates marked with the same symbolic name in Fig. For example, the input marked ADD in Fig. is connected to the output marked ADD in Fig..

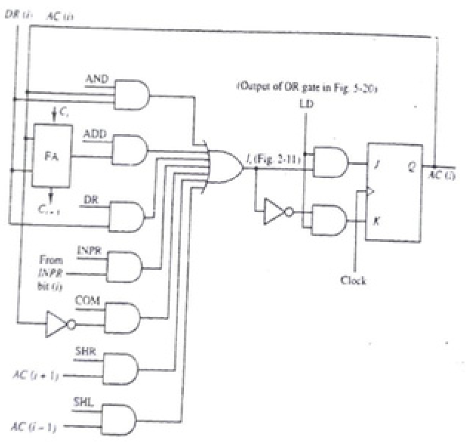


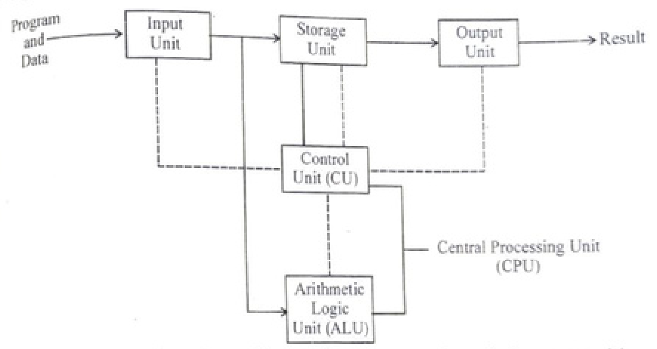
Fig. 18 : One stage of adder and logic circuit

The AND operation is achieved by ANDing AC(i) with the corresponding bit in the data register DR(i). The AD operation is obtained using a binary adder similar to the one shown in Fig. One stage of the adder uses a full-adder with the corresponding input and output carries. The transfer from INPR to AC is only for bits 0 through 7. The complement microoperation is obtained by inverting the bit value in AC. The shift-right operation transfers the bits from AC(i + 1), and the shift-left operation transfers the bit from AC(i − 1). The complete adder and logic circuit consists of 16 stages connected together.

3.10 CPU Design

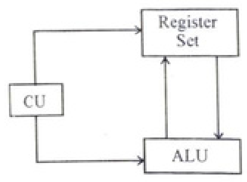
It is the brain of any computer system. The control unit (CU) and Arithmetic Logic Unit (ALU) are jointly known as central processing unit (CPU). As in the human body, all decisions (major) are taken by the brain, similarly in a computer system all major calculation and comparison are made inside the CPU. CPU is also responsible for activating and controlling the operations of other units of a computer system.

Organization



Computer system is made up of integrated components :– Input devices, output deivces, storage unit, CPU, that work together to perform program execution.

Input or output units can not function until they receive signals from central processing unit. The same way, the storage unit is of use with but CPU, So usefullness of each unit depends on other units, and when all units are put together, they form a system.



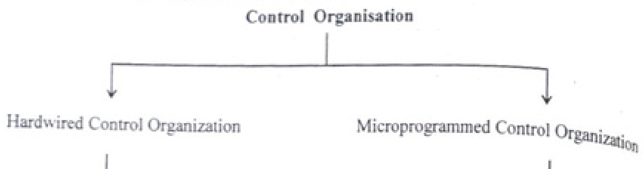
The CPU is basically made up of three major parts :–

1. Control Unit (CU)
2. Arithmetic Logic Unit (ALU) and
3. Register set

The register set stores intermediate data used during the execution of the instructions. The ALU performs the required operations for execution the instructions. The control unit supervises the transfer of information among the register and instructs all to perform the operations.

3.10.1 Control Organization

These are two major types of control organization :–

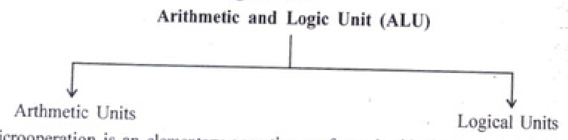

Control Organisation
Hardwired Control Organization
Microprogrammed Control Organization

In this organzation, the control logic is implemented with gates, flipflops, decoders, and other digital circuits. There is no concept of control memory. Once a hardwired control is designed for a particular task then any small modification in task might need a complete circuit design all over again. Hardwired control can be optimized to produce a fast mode of operation. It requires changes in the wiring among the various components if the design has to be modified or changed.

In this organization, the control infromation is stored in a control memory. The control memory is programmed to initiate the required sequence of operations. Here no need to change wiring if modification or any changes are needed. Any required changes or modifications can be done by updating the microprogram stored in control memory.

3.11 Arithmetic and Logic Unit (ALU) Design

ALU is the combination of following 2 units :–



A microoperation is an elementary operation performed with the data stored in registers. Microoperation can classified in following four categories :–

1. Register Transfer Microoperation
2. Arithmetic Microoperation
3. Logical Microoperation
4. Shift Microoperation

3.11.1 Register Transfer Microoperation

$R_2 \leftarrow R_3$

It denotes a transfer of the content of register R_3 into register R_2 i.e. information transfer from one register to another by means of replacement operator. Here content of R_2 is replaced by the content of R_3 though the content of R_3 remains unchanged.

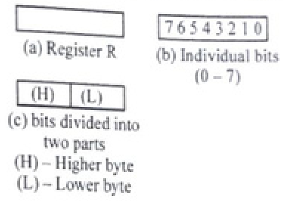
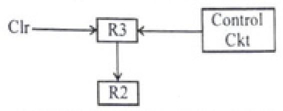


Fig. Block diagram of register

To perform a particular register transfer, register transfer language is used. Every element written in a register transfer notation implies a hardware construction for implementing the transfer.



Transfer from R_3 to R_2 when $P = 1$

i.e. if $(P = 1)$ then $(R_2 \leftarrow R_3)$

Where P is control signal generated from control circuit. i.e. $P : R_2 \leftarrow R_3$

Basic symbol used in register transfer :–

Symbol	Meaning	Example
1. Letter	Denote a register	R_1, R_2, MAR
2. Parentheses ( )	Denote part of a register	$R_3(0 - 7)$, $R_3(L)$ ↓ Lower byte $R_3(H)$, $R_3(8 - 15)$ ↓ Higher byte
3. Arrow ←	Denote direction transfer of information	$R_3 \leftarrow R_2$, $R_4 \leftarrow R_5$
4. Comma	Separates two microoperations	$R_3 \leftarrow R_2$, $R_1 \leftarrow R_2$

3.11.2 Arithmetic Microoperation

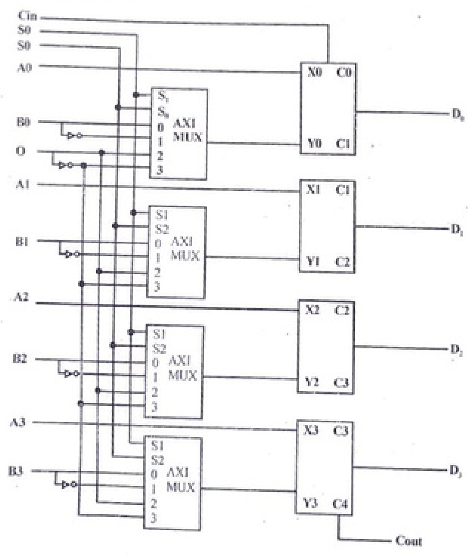
Basic arithmetic operations are as follows :–

- Addition
- Subtraction
- Increment
- Decrement
- Shift

Symbols	Description
1. $R_3 \leftarrow R_1 + R_2$	Content of $R_1 + R_2$ transferred to R_3
2. $R_3 \leftarrow R_1 - R_2$	Content of $R_1 - R_2$ transferred to R_3
3. $R_2 \leftarrow \overline{R}_2$	1's complement of contents of R_2
4. $R_3 \leftarrow \overline{R}_2 + 1$	2's complement of contents of R_2
5. $R_3 \leftarrow R_1 + \overline{R}_2 + 1$	R_1 complement of R_2 transferred to R_3 (Subtraction)
6. $R_3 \leftarrow R_1 + 1$	Increment of content of R_1 by one
7. $R_3 \leftarrow R_1 - 1$	Decrementation of content of R_1 by one

Basic complement of an arithmetic cricuits is the parallel adder. By restricting the data inputs to the adder, it is possible to obtain different types of arithmetic operation.

4-bit arithmetic circuits



There are 4-bit input (A & B) and 4-bit output (D). The four input from A (A_0, A_1, A_2, A_3) go directly to the X inputs of binary full adder (FA). Each of the four input from B(B_0, B_1, B_2, B_3) are connected to the data input of the 4 × 1 multiplexer. The multiplexer data inputs also receive the complement of B as second data input of multiplexer. Other two data inputs are logic – 0 and logic –1 which is a fixed voltage value – 0 voltage for logic 0 and nonzero voltage (depends upon signal generated through invester) as logic – 1 respectively.

There are 2 selection inputs S_0 and S_1 input carry C_{in} goes to the FA in the least significant position. Other carries are connected from one stage to the next.

Output of FA can be expressed as,

$$\boxed{D = A + Y + Cin}$$

$$\text{i.e.}\quad \left.\begin{aligned} D_0 &= A_0 + Y_0 + C_0 \\ D_1 &= A_1 + Y_1 + C_1 \\ D_2 &= A_2 + Y_2 + C_2 \end{aligned}\right\} \text{4-bit output}$$

Function Table

Select			Input	Output	Microoperation
S_1	S_0	Cin	Y	D = A + Y + Cin	
0	0	0	B	D = A + B	Addition
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtraction
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A –1	Decrement A
1	1	1	1	D = A	Transfer A

Logical Microoperations

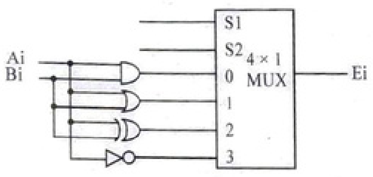
Logic microoperations consider each bit of the register separately. Special symbols are adopted for the logic microoperations OR, AND, EX-OR, complement. Symbol '$\vee$' will be used to denote an OR microoperation and '$\wedge$' symbol is used to denote an AND microoperation.

Logical Microoperations

Boolean Function	Microoperation	Description
$f_0 = 0$	$f \leftarrow 0$	Clear
$f_1 = xy$	$f \leftarrow A \wedge B$	AND
$f_2 = xy'$	$f \leftarrow A \wedge \overline{B}$	AND
$f_3 = x$	$f \leftarrow A$	Transfer A

$f_4 = x'y$	$f \leftarrow A \wedge B$	AND
$f_5 = y$	$f \leftarrow B$	Transfer B
$f_6 = x \oplus y$	$f \leftarrow A \oplus B$	Exclusive OR
$f_7 = x + y$	$f \leftarrow A \vee B$	OR
$f_8 = (x + y)'$	$f \leftarrow \overline{A \vee B}$	NOR
$f_9 = (x \oplus y)$	$f \leftarrow (A' \oplus B)'$	Exclusive NOR
$f_{10} = y'$	$f \leftarrow \overline{B}$	Complement B
$f_{11} = x + y'$	$f \leftarrow A \vee \overline{B}$	OR
$f_{12} = x'$	$f \leftarrow \overline{A}$	Complement A
$f_{13} = x' + y$	$f \leftarrow \overline{A} \vee B$	OR
$f_{14} = (xy)'$	$f \leftarrow \overline{A \wedge B}$	NAND
$f_{15} = 1$	$f \leftarrow$ All 1s	Set all 1's

One Stage of Logic Circuit



Function Table

S1	S2	Output (Ei)	Operation
Select			
0	0	E = A^B	AND
0	1	E = A∨B	OR
1	0	E = A⊕B	XOR
1	1	E = $\overline{A}$	Complement

3.11.3 Shift Microoperation

Shift microoperation are use for serial transfer of data.

4-bit Shifter

4-bit Shifter

Shift Microoperation

Shift – Left Operation
↓
Serial input transfers a bit into the right most position

Shift – Left Operation
↓
The serial input transfer a bit into the left most position

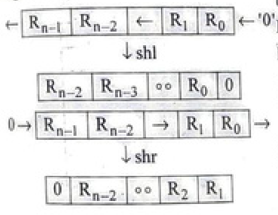
Control Organisation

Logical Shift
↓
- Logical Shift Left (Shl)
- Logical Shift Righr (Shr)

A logical shift is one that transfer '0' through the serial input.

$R_3 \leftarrow$ Shl R_3, $R_4 \leftarrow$ Shr R_4

1 – bit shift to the left content of register R_3 and a 1-bit shift to the right of the content of register R_4.

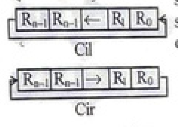


Circular Shift
↓
- Circular Shift Left (Cil)
- Circular Shift Right (Cir)

It is also known as rotate operation. It circulate the bits of the register around the two ends without loss of information.

$R_3 \leftarrow$ Cil R_3

$R_4 \leftarrow$ Cir R_3

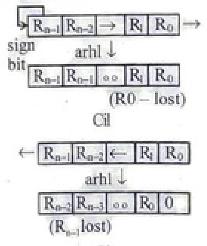


A sign reversal occurs if the bit in R_{n-1} change in value after the shift.

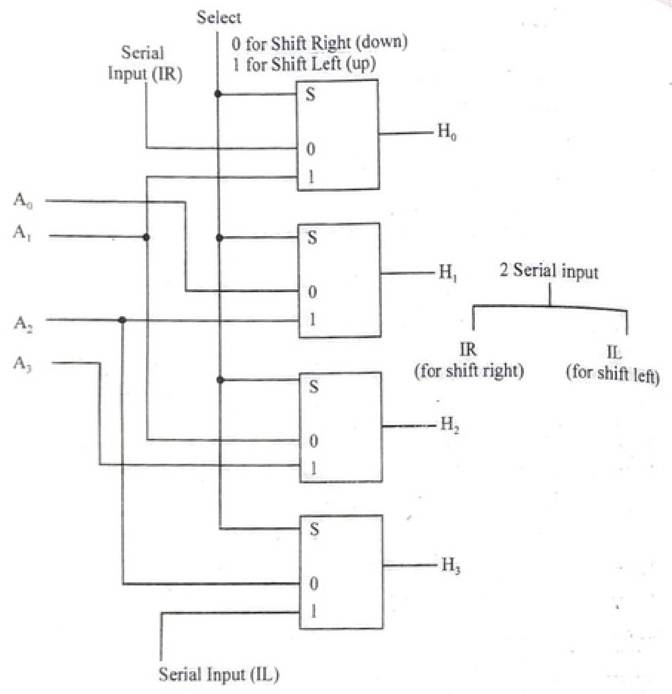
Arithmetic Shift
↓
- Arithmetic Shift Left (ashl)
- Arthmetic Shift Righr (ashr)

This operation shifts a signed binary number to the left or right.

An arithmetic shift – Left multiplies a 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.



Cir



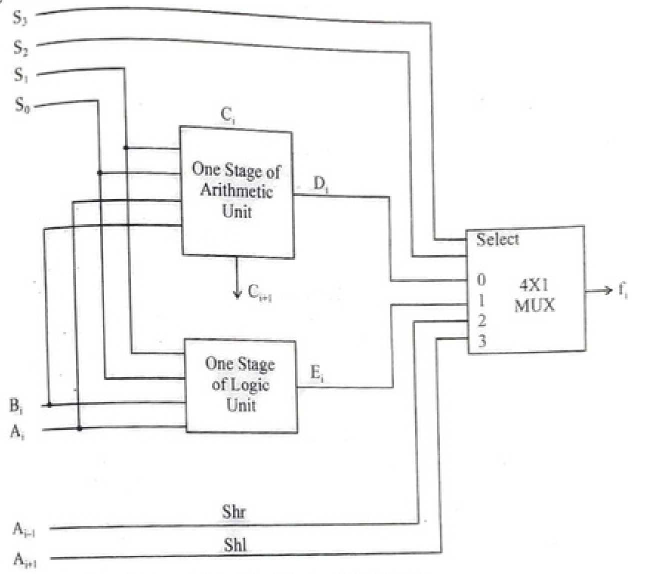
Function Table

Select	Output			
S	H_0	H_1	H_2	H_3
0	IR	A_0	A_1	A_2
1	A_1	A_2	A_3	IL

3.12 Arithmetic Logic Shift Unit

Arithmetic unit, logic unit and shift unit is combined into one unit i.e. ALU.

One Stage of ALU



Function Table for Arithmetic Logic Shift Unit

Operation Select						
S_3	S_2	S_1	S_0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Transfer A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with Carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtractioin
0	0	1	1	0	F = A –1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND ing
0	1	0	1	×	$F = A \vee 1$	OR ing

0	1	1	0	×	$F = A \oplus B$	XOR ing
0	1	1	1	×	$F = \bar{A}$	Complement A
1	0	×	×	×	F = Shr A	Shift Right A
1	1	×	×	×	F = Shl A	Shift Lift A

The CPU of a small computer is a microprocessor. The CPU of a large computer contains a number of microprocessor. Each microprocessor in a large central processing unit perform a specified task within the CPU.

3.13 SUMMARISE VIEW

3.13.1 Arithmetic and Logic Unit (ALU)

The function of an arithmetic and logic unit is perform arithmetic and logic operation. Usually and ALU performs the following arithmetic and logic operation.

(i) Addition
(ii) Subtraction
(iii) Multiplication
(iv) Division
(v) Logical AND
(vi) Logical OR
(vii) Logical EXCLUSIVE - OR
(viii) Complement (Logical NOT)
(ix) Increment (i.e. addition of 1)
(x) Decrement (i.e. subtraction of 1)
(xi) Left or right shift (the content of the accumulator can be shifted left or right by one bit)
(xii) Clear (the content of the accumulator or carry flag can be made zero).

Other mathematical operations such as exponential, logarithmic, trigonometric and floating-point operations are not performed by ALU. These operations are performed by special purpose math processor called floating-point unit (FPU). Modern microprocessors contain an FPU on the microprocessor which are used for simple automatic control applications may not contain on-chip FPU. Such processors use either software for above mentioned mathematical operations or employ a math processor IC (or math coprocessor) in the microprocessor based system. The use of software for such mathematical operations make execution slower. Math processors speed up program execution and reduce programming complexity. The choice depends on actual and cost involved in a particular application.

3.13.2 Control Unit (CU) design

The control unit of a CPU controls the entire operation of the computer. This very section of the CPU really acts as the brain of the computer. It also controls all other devices such as memory, input and output devices connected to the CPU. It fetches instruction from the memory, decodes the instruction, interprets the instruction to know what tasks are to be performed. It

generates timing and control signals, and provides them for all operation. It controls the data flow between CPU and peripherals (including memory). It provide status, control and timing signals that the memory and I/O devices require.

Under the control of the control unit the instructions are fetched from the memory one after another for execution until all the instructions are excuted. For fetching and executing an instruction the following steps are performed under its control:

(i) The address of the memory location where instruction lies, is placed on the address bus.
(ii) Instruction is read from the memory.
(iii) The instruction is sent to the decoding circuitary for decoding.
(iv) Addresses and data required for the execution of the instruction are read from the memory.
(v) These data/addresses are sent to the other sections for processing.
(vi) The result are sent to the memory or kept is some register.
(vii) Necessary steps are taken for next instruction. For this the content of the program counter is incremented.

A CPU contains a number of special purpose registers for different purposes. These are :

(i) Program Counter (PC)
(ii) Stack Pointer (SP)
(iii) Status Register
(iv) Instruction Register (IR)
(v) Index Register
(vi) Memory Address Register (MAR)
(vii) Memory Buffer Register (MBR) or Data Register (DR)

All CPUs do not contain all of these special registers A powerful CPU contains most of them.

3.14 DESIGN AND IMPLEMENTATION OF A SIMPLE MICRO-SQUENCER

In computer architecture and engineering, a sequencer or microsequencer generates the addresses used to step through the microprogram of a control store. It is used as a part of the control unit of a CPU or as astand alone generator for address ranges.

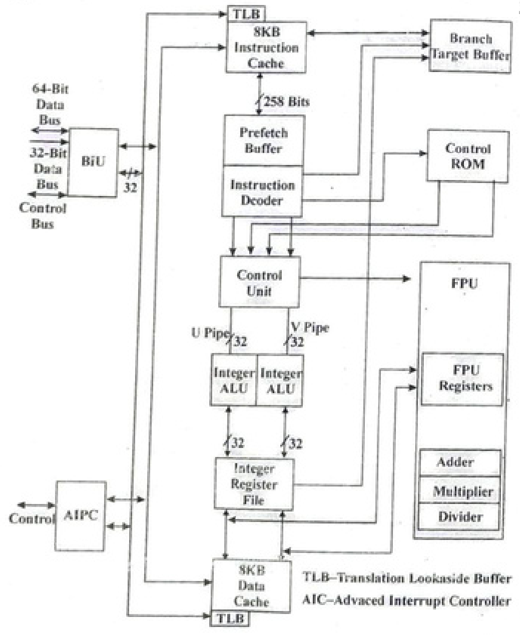
Usually the addresses are generated by same combination of a counter, a field from a microinstruction and some subset of the instruction register. A counter is used for the typical case that the next micro instructions is the one of execute or other logic. Since CPU implement an instruction set, its very useful to able to decode the instruction's bit directly into the sequencer to select a set of microinstructions to perform a CPU's instructions.

Most modern CPU's are considerably more complex then this description suggest. They tend to have multiple cooperating micromachines with specialized logic to detect and handle interface between the micromachines.

3.15 Features of Pentium Microprocessor

The pentium is a high performance, superscalar, CISC microprocessor developed by Intel corporation. It was introduced in the year 1993. It has 32-bit address bus and 64-bit data bus. It contains 3.1 million transistors. It is housed in a 273-pin grid array package. The clock frequencies for is different versions are: 60-233MHz. It provides traditional memory page size of 4KB or larger size of 4MB. It is suitable for multitasking/multiuser system. It is used in high-end desktop PCs, workstations and network file servers where high speed data processing is required. Its MIPS capability is 112 at 66MHz. Its data transfer rate is 528MB/see at 66MHz. The computing power of pentium at 100MHz is 166MIPS. 100MHz Pentium is manufactured using 0.8 micron 3 layer process technology. 120MHz 3.3V Pentium uses 0.35 micron process technology. 150MHz pentium uses 0.6 micron 4 layer process technology.

Block Diagram



The pentium processor contain two 8KB cache memories. one for code and the other for data. There is an on chip floating-point math coprocessor unit. It incorporates 7 stage popelining and hard-wired functions. It is five to ten times faster then the floating-point unit used in the 486 microprocessor. Though the pentium processor, for example, superscalar architecture a microprocessor architecture which incorporates more than one execution unit or pipeline is called **superscalar architecture**. The superscalar architecture executes multiple instructions per clock cycle. The Pentium is designed to have dual-pipeline architecture. The dual pipeline is known as U pipeline and V pipeline. Both the pipeline execute instructions in 5 stages. Each pipeline has its own arithmetic logic unit, address generation circuitary and data cache interface. It is capable of executing two instruction simultaneously per clock cycle. This type of feature is generally provided in a RISC processor.

Advantages

The main advantage of pentium over the power PC is that it can run a large number of already-existing software which is a powerful marketing advantage. The pinetum has the features of parity-based internal error detection for both the instruction and data cache, tags and cache TLBs (translation lookside buffers), as well as for the microcode ROM. It contains a circuitry to monitor the performance of one pentium with another. It has the features of the built-in error detection for multiprocessor servers not found in RISC chips, which is crucial to distribute computing. The pinetum processor id provided with branch prediction ability which is an advanced computing technique. This technique was available in traditional mainframe computers. By this technique the processor can predict what instructions are to be executed next.

EXERCISES

Very Short Answer Type Questions (2 Marks each)

1. One kilo byte is.......... (Raj. B.C.A. 2010)
2. Bit is an abbreviation of .... (Raj. B.C.A. 2009)
3. What does RAM and DRAM stands for............ (Raj. B.C.A. 2009)
4. A multiplexer is a circuit with .... (Raj. B.C.A. 2008)
5. A byte is .... (Raj. B.C.A. 2007)

Short Answer Type Questions (4 Marks each)

1. Explain design and implementation of a simple micro sequencer.
2. What are the features of pentium processor.
3. RTL (Register Transfer Language) Define it.
4. Explain the concept of input/output interfacing ?
5. Write short notes on

(a) Instruction execution cycle
(b) System bus

Long Answer Type Questions (12 Marks each)

1. What is an I/O processor ? Why are I/O processor used in large system. **(Raj. B.C.A.2006)**
2. Explain the concept of input-output interfacing and I/O processor. **(Raj. B.C.A. 2008)**
3. Explain the features of Pentium processor **(Raj. B.C.A. 2006)**
4. Define RTL (Register Transfer Language) briefly.
5. Draw a block diagram to illustrate the basic organization of a computer system and explain the function of the logic units. **(Raj. B.C.A. 2007)**
6. Write short notes on-
   (a) System Buses
   (b) Pentium Microprocessor
   (c) Asynchronous data transfer
7. What is the design of a simple sequencer ? Explain the features of pentium micoprocessor.
8. What is memory interfaces and memory organization. Explain in detail.
9. Explain in briefly-
   (i) Instruction cycle and system buses
   (ii) RTL (Register Transfer Language)
10. What do you understand by register Transfer ? Explain the use of register transfer language.
11. What is Hard disk ? Explain the various harddisk interface ?