Basic Computer Organization and Design

Registers in Computer Architecture

Register is a very fast computer memory, used to store data/instruction in-execution.
A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information. An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.
A register consists of a group of flip-flops and gates. The flip-flops hold the binary information and gates control when and how new information is transferred into a register. Various types of registers are available commercially. The simplest register is one that consists of only flip-flops with no external gates.
These days registers are also implemented as a register file.

Loading the Registers
The transfer of new information into a register is referred to as loading the register. If all the bits of register are loaded simultaneously with a common clock pulse than the loading is said to be done in parallel.

*****Register Transfer Language**

The symbolic notation used to describe the micro-operation transfers amongst registers is called **Register transfer language**.
The term **register transfer** means the availability of **hardware logic circuits** that can perform a stated micro-operation and transfer the result of the operation to the same or another register.
The word **language** is borrowed from programmers who apply this term to programming languages. This programming language is a procedure for writing symbols to specify a given computational process.
Following are some commonly used registers:

1. **Accumulator**: This is the most common register, used to store data taken out from the memory.
2. **General Purpose Registers**: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.
3. **Special Purpose Registers**: Users do not access these registers. These registers are for Computer system,
    - **MAR:** Memory Address Register are those registers that holds the address for memory unit.
    - **MBR:** Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
    - **PC:** Program Counter points to the next instruction to be executed.
    - **IR:** Instruction Register holds the instruction to be executed.

Register Transfer

Information transferred from one register to another is designated in symbolic form by means of replacement operator.
R2 ← R1
It denotes the transfer of the data from register R1 into R2.
Normally we want the transfer to occur only in predetermined control condition. This can be shown by following **if-then** statement: if (P=1) then (R2 ← R1)
Here P is a control signal generated in the control section.

Control Function

A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:

P: R2 ← R1

The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

***Micro-Operations

The operations executed on data stored in registers are called micro-operations. A micro-operation is an elementary operation performed on the information stored in one or more registers.

Example: Shift, count, clear and load.

Types of Micro-Operations

The micro-operations in digital computers are of 4 types:

1. Register transfer micro-operations transfer binary information from one register to another.
2. Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operations perform bit manipulation operation on non-numeric data stored in registers.
4. Shift micro-operations perform shift micro-operations performed on data.

Arithmetic Micro-Operations

Some of the basic micro-operations are addition, subtraction, increment and decrement.

Add Micro-Operation

It is defined by the following statement:

$R3 \rightarrow R1 + R2$

The above statement instructs the data or contents of register R1 to be added to data or content of register R2 and the sum should be transferred to register R3.

Subtract Micro-Operation

Let us again take an example:

$R3 \rightarrow R1 + R2' + 1$

In subtract micro-operation, instead of using minus operator we take **1's compliment** and add 1 to the register which gets subtracted, i.e **R1 - R2** is equivalent to **R3 → R1 + R2' + 1**

Increment/Decrement Micro-Operation

Increment and decrement micro-operations are generally performed by adding and subtracting 1 to and from the register respectively.

$R1 \rightarrow R1 + 1$
$R1 \rightarrow R1 - 1$

Symbolic Designation	Description
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.

R1 ← R1 - 1	Decrement the contents of R1 by 1.

Logic Micro-Operations

These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.

```
P: R1 ← R1 X-OR R2
```

In the above statement we have also included a Control Function.

Assume that each register has 3 bits. Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

******Shift Micro-Operations***

These are used for serial transfer of data. That means we can shift the contents of the register to the left or right. In the **shift left** operation the serial input transfers a bit to the right most position and in **shift right** operation the serial input transfers a bit to the left most position.

There are three types of shifts as follows:

a) Logical Shift

It transfers 0 through the serial input. The symbol **"shl"** is used for logical shift left and **"shr"** is used for logical shift right.

```
R1 ← she R1
R1 ← she R1
```

The register symbol must be same on both sides of arrows.

b) Circular Shift

This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

c) Arithmetic Shift

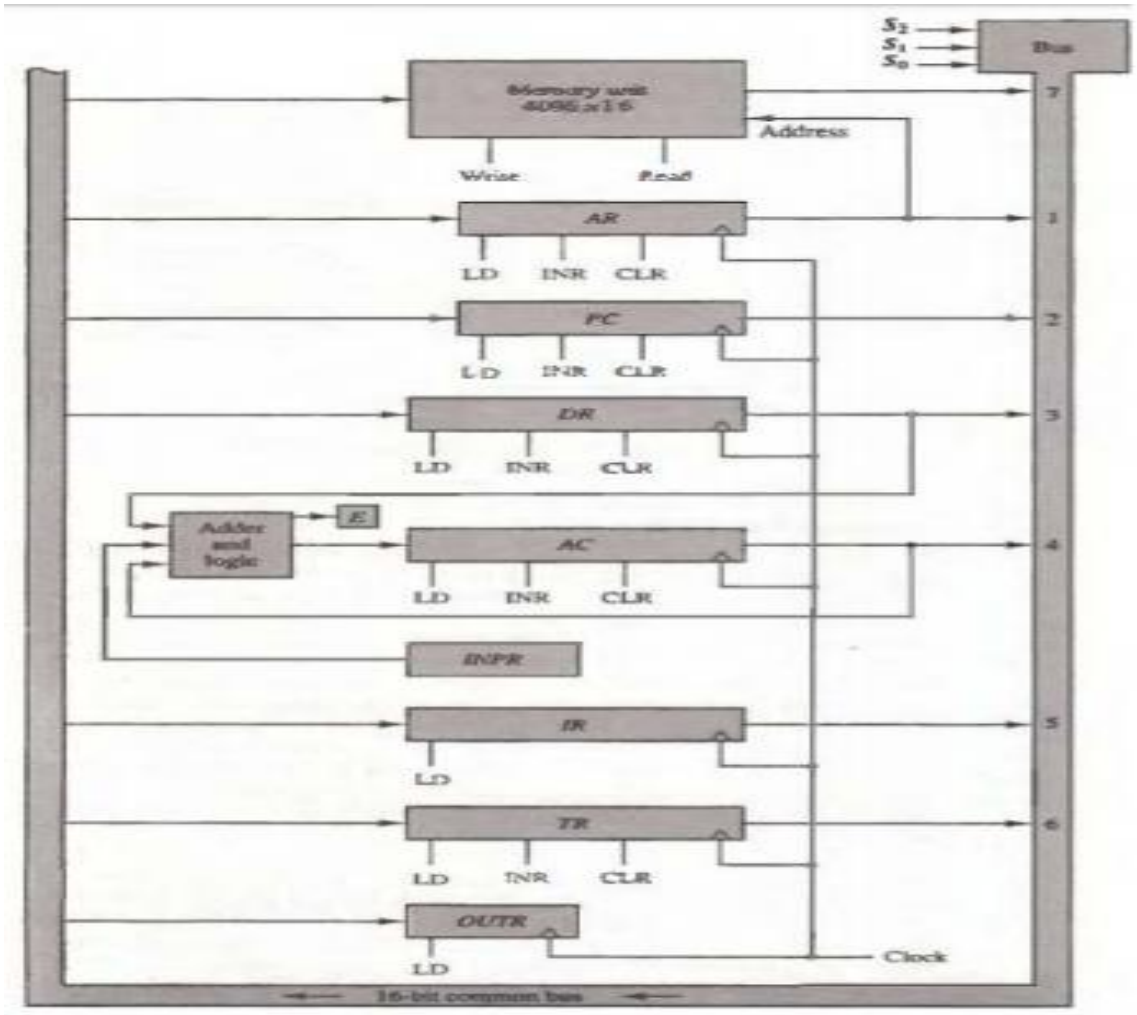
This shifts a signed binary number to left or right. An **arithmetic shift left** multiplies a signed binary number by 2 and **shift left** divides the number by 2. Arithmetic shift micro-operation leaves the sign bit unchanged because the signed number remains same when it is multiplied or divided by 2.

Common Bus System

- **The basic computer has eight registers**, a memory unit, and a control unit . Paths must be provided to transfer information from one register to another and between memory and registers.
- **The number of wires will be excessive** if connections are made between the outputs of each register and the inputs of the other registers.
- **A more efficient** scheme for transferring information in a system with many registers is to use a common bus.
- **The connection of the registers** and memory of the basic computer to a common bus system is shown in Fig. below. The outputs of seven registers and memory are connected to the common bus.

- **The specific output** that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.

- **The number along** each output shows the decimal equivalent of the required binary selection. For example, the number along the output of DR is 3.
- **The 16-bit outputs of DR** are placed on the bus lines when $S_2S_1S_0 = 011$ since this is the binary value of decimal 3.
- **The lines from the common bus** are connected to the inputs of each register and the data inputs of the memory. The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- **The memory** receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.
- **Four registers**, DR, AC, IR, and TR, have 16 bits each. Two registers, AR



- **Figure 5-4** Basic computer registers connected to a common bus.

- **and PC, have 12 bits** each since they hold a memory address. When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's.
- **When AR or PC** receive information from the bus, only the 12 least significant bits are transferred into the register. The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- **INPR is connected to provide information** to the bus but OUTR can only receive information from the bus.

- **This is because INPR** receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.
- **The 16 lines of the common bus** receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).
- **This type of register** is equivalent to a binary counter with parallel load and synchronous clear. The increment operation is achieved by enabling the count input of the counter. Two registers have only a LD input.
- **The input data and output data** of the memory are connected to the common bus, but the memory address is connected to AR. Therefore, AR must always be used to specify a memory address.
- **By using a single register** for the address, we eliminate the need for an address bus that would have been needed otherwise. The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC .
- **The 16 inputs of AC** come from an adder and logic circuit. This circuit has three sets of inputs. One set of 16-bit inputs come from the outputs of AC . They are used to implement register microoperations such as complement AC and shift AC .
- **Another set of 16-bit inputs** come from the data register DR. The inputs from DR and AC are used for arithmetic and logic rnlcrooperations, such as add DR to AC or AND DR to AC.
- **The result of an addition** is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit). A third set of 8-bit inputs come from the input register INPR.
- **Note that the content of any register** can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.
- **For example**, the two microoperations

- DR ← AC and AC ← DR

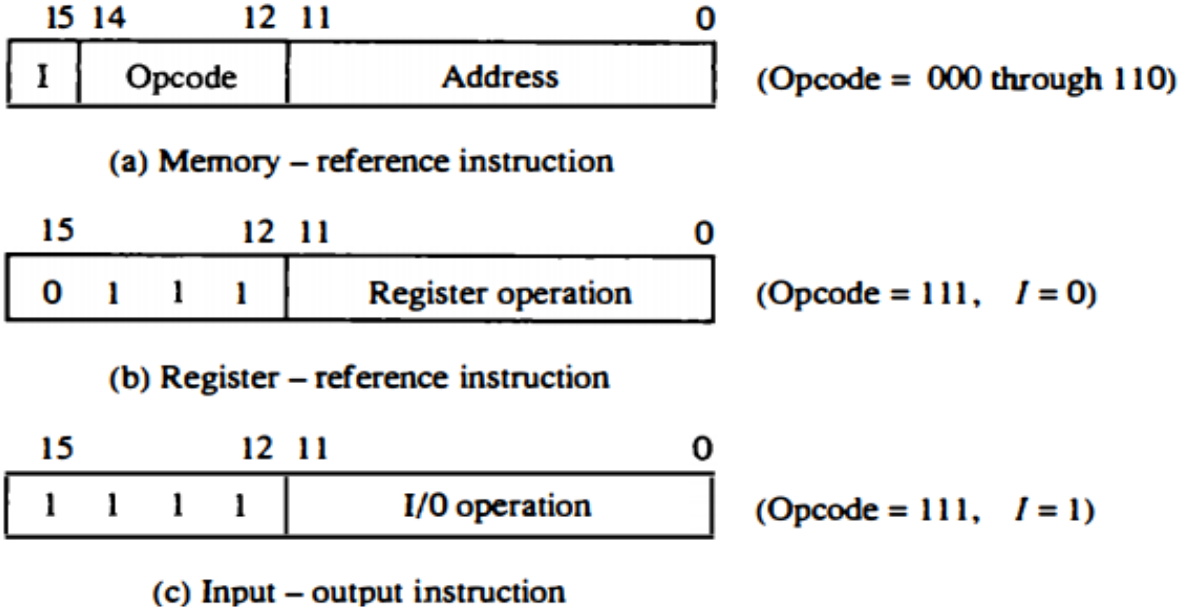
- **can be executed at the same time**. This can be done by placing the content of AC on the bus (with $S_2S_1S_0 = 100$), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of AC, all during the same clock cycle.
- **The two transfers** occur upon the arrival of the clock pulse transition at the end of the clock cycle.

Computer Instructions

- **The basic computer** has three instruction code formats, as shown in Fig. below. Each format has 16 bits.
- **The operation code (opcode)** part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
- **A memory-reference** instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
- **The register reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

- **A register-reference** instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- **Similarly**, an input-output instruction does not need a reference to memory and is recognized by the operation code Ill with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.
- **The type of instruction** is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 though 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I.
- **If the 3-bit opcode** is equal to 111, control then inspects the bit in position 15. If this bit is 0, the

Figure Basic computer instruction formats.



- **instruction** is a register-reference type. If the bit is 1, the instruction is an input-output type. Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.
- **Only three bits of the instruction** are used for the operation code. It may seem that the computer is restricted to a maximum of eight distinct operations.
- **However**, since register-reference and input-output instructions use the remaining 12 bits as part of the operation code, the total number of instructions can exceed eight.
- **In fact**, the total number of instructions chosen for the basic computer is equal to 25.
- **The instructions** for the computer are listed in Table below. The symbol designation is a three-letter word and represents an abbreviation intended for programmers and users. The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.
- **By using the hexadecimal equivalent** we reduced the 16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits.
- **A memory-reference instruction** has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address.
- **The last bit of the instruction** is designated by the symbol I. When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0.

- **When I = I**, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is I. Register-reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7.
- **The other three hexadecimal digits** give the binary equivalent of the remaining 12 bits. The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

TABLE Basic Computer Instructions

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

****Instruction Set Completeness**

- **A computer should have a set of instructions** so that the user can construct machine language programs to evaluate any function that is known to be computable.
- **The set of instructions** are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- 1. Arithmetic, logical, and shift instructions
- 2. Instructions for moving information to and from memory and processor
- registers
- 3. Program control instructions together with instructions that check
- status conditions

- 4. Input and output instructions

- **Arithmetic, logical, and shift instructions** provide computational capabilities for processing the type of data that the user may wish to employ.
- **The bulk of the binary information** in a digital computer is stored in memory, but all computations are done in processor registers.
- **Therefore**, the user must have the capability of moving information between these two units. Decision making capabilities are an important aspect of digital computers.
- **For example**, two numbers can be compared, and if the first is greater than the second, it may be necessary to proceed differently than if the second is greater than the first.
- **Program control instructions** such as branch instructions are used to change the sequence in which the program is executed. Input and output instructions are needed for communication between the computer and the user.
- **Programs and data** must be transferred into memory and results of computations must be transferred back to the user. The instructions listed in Table above constitute a minimum set that provides all the capabilities mentioned above.
- **There is one arithmetic instruction**, ADD, and two related instructions, complement AC(CMA) and increment AC(INC).
- **With these three instructions** we can add and subtract binary numbers when negative numbers are in signed-2's complement representation. The circulate instructions, CIR and CIL, can be used for arithmetic shifts as well as any other type of shifts desired.
- **Multiplication and division** can be performed using addition, subtraction, and shifting. There are three logic operations: AND, complement AC(CMA), and clear AC(CLA).
- **The AND and complement provide a NAND operation**. It can be shown that with the NAND operation it is possible to implement all the other logic operations with two variables.
- **Moving information** from memory to AC is accomplished with the load AC(LDA) instruction. Storing information from AC into memory is done with the store AC(STA) instruction.
- **The branch instructions BUN, BSA, and ISZ**, together with the four skip instructions, provide capabilities for program control and checking of status conditions. The input (lNP) and output (OUT) instructions cause information to be transferred between the computer and external devices.
- **Although the set of instructions** for the basic computer is complete, it is not efficient because frequently used operations are not performed rapidly. An efficient set of instructions will include such instructions as subtract, multiply, OR, and exclusive-OR.

*****Timing and Control**

- **The timing for all registers** in the basic computer is controlled by a master clock generator.
- **The clock pulses** are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- **The clock pulses do not change the state** of a register unless the register is enabled by a control signal.
- **The control signals** are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.
- **There are two major types** of control organization: hardwired control and microprogrammed control. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- **It has the advantage** that it can be optimized to produce a fast mode of operation.
- **In the microprogrammed organization**, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.

- **A hardwired control**, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.
- **The block diagram of the control unit** is shown in Fig. below. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the nstruction register (IR).
- **The position of this register** in the common bus system is indicated in Fig. previously seen. The instruction register is shown again in Fig. below, where it is divided into three parts: the I bit, the operation code, and bits 0 through 11.
- **The operation code in bits 12 through 14** are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7
- **The subscripted decimal number is equivalent** to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- **Bits 0 through 11** are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15
- **The sequence counter SC** can be incremented or cleared synchronously
- **. Most of the time**, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T0.
- **As an example**, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active.
- **This is expressed symbolically** by the statement D3T4: SC ← 0. The timing diagram of Fig. below shows the time relationship of the control signals.
- **The sequence counter SC** responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the

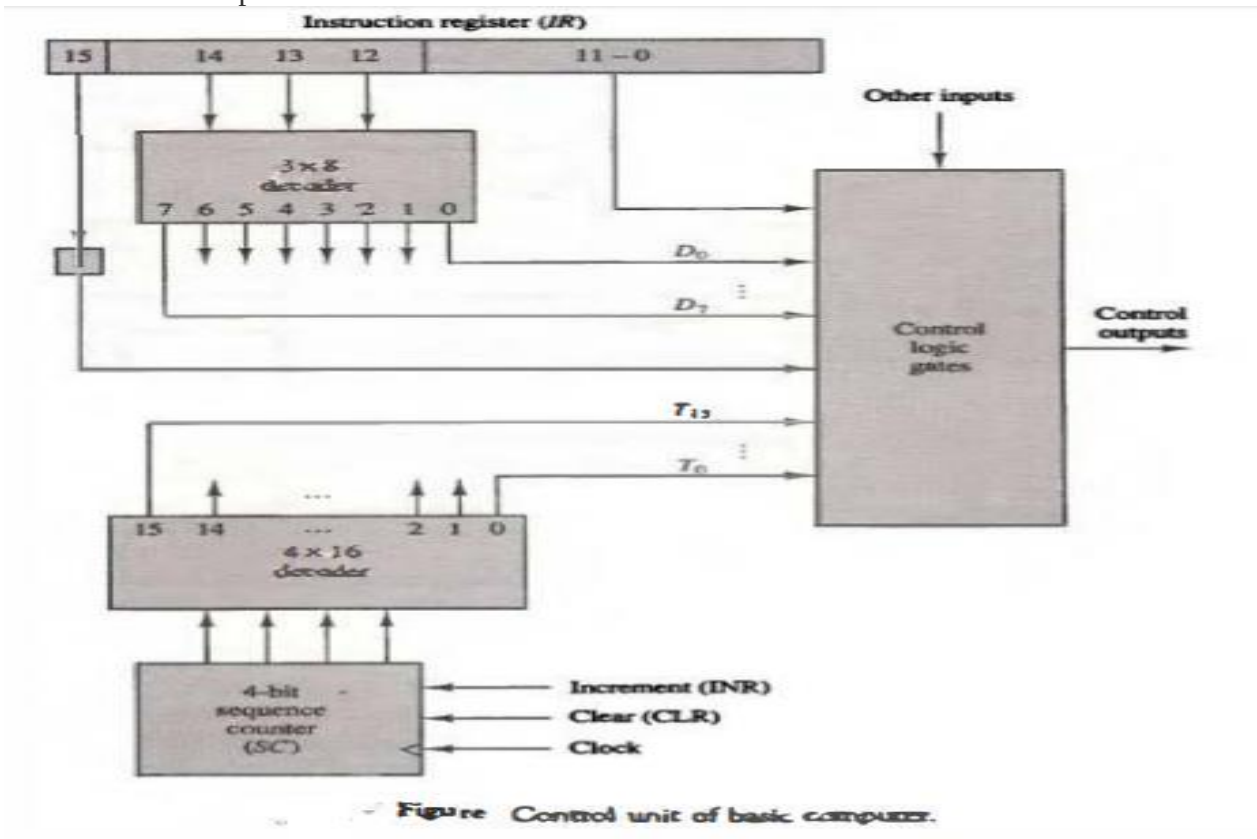


Figure Control unit of basic computer.

- **clock clears SC to 0**, which in turn activates the timing signal T0 out of the decoder. T0 is active during one clock cycle.

- **The positive clock transition** labeled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.
- **SC is incremented** with every positive clock unless its CLR input is active.
- **This produces the sequence** of timing signals T0,T1,T2,T3,T4 and so on, as shown in the diagram. (Note the relationship between the timing signal and its corresponding positive clock transition.)
- **If SC is not cleared**, the timing signals will continue with T_5, T_6 up to T15 and back to T_0.

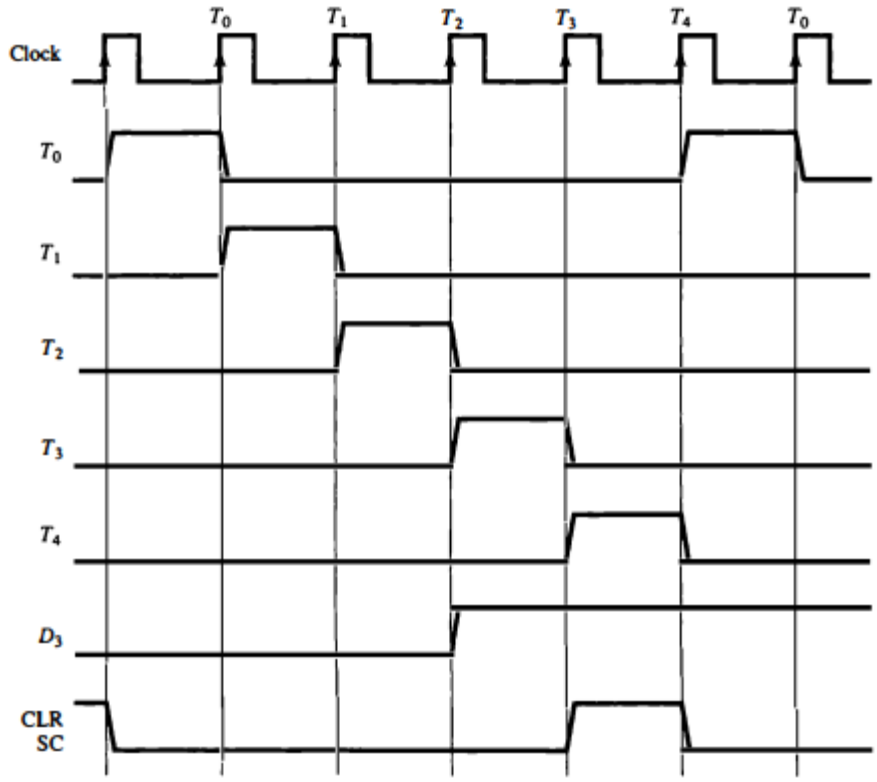


Figure Example of control timing signals.

- **The last three waveforms in Fig. above** show how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2
- **When timing signal T4 becomes active**, the output of the AND gate that implements the control function D3T4 becomes active.
- **This signal is applied to the CLR input of SC**. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.
- **A memory read or write cycle** will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition.
- **The clock transition** will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle.
- **In such a case** it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer.
- **To fully comprehend the operation of the computer**, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement T0: AR ← PC specifies a transfer of the content of PC into AR if timing signal T0 is active.

- **T0 is active during an entire clock cycle interval**. During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition.
- **This same positive clock transition** increments the sequence counter SC from 0000 to 0001 . The next clock cycle has T1 active and T0 inactive.

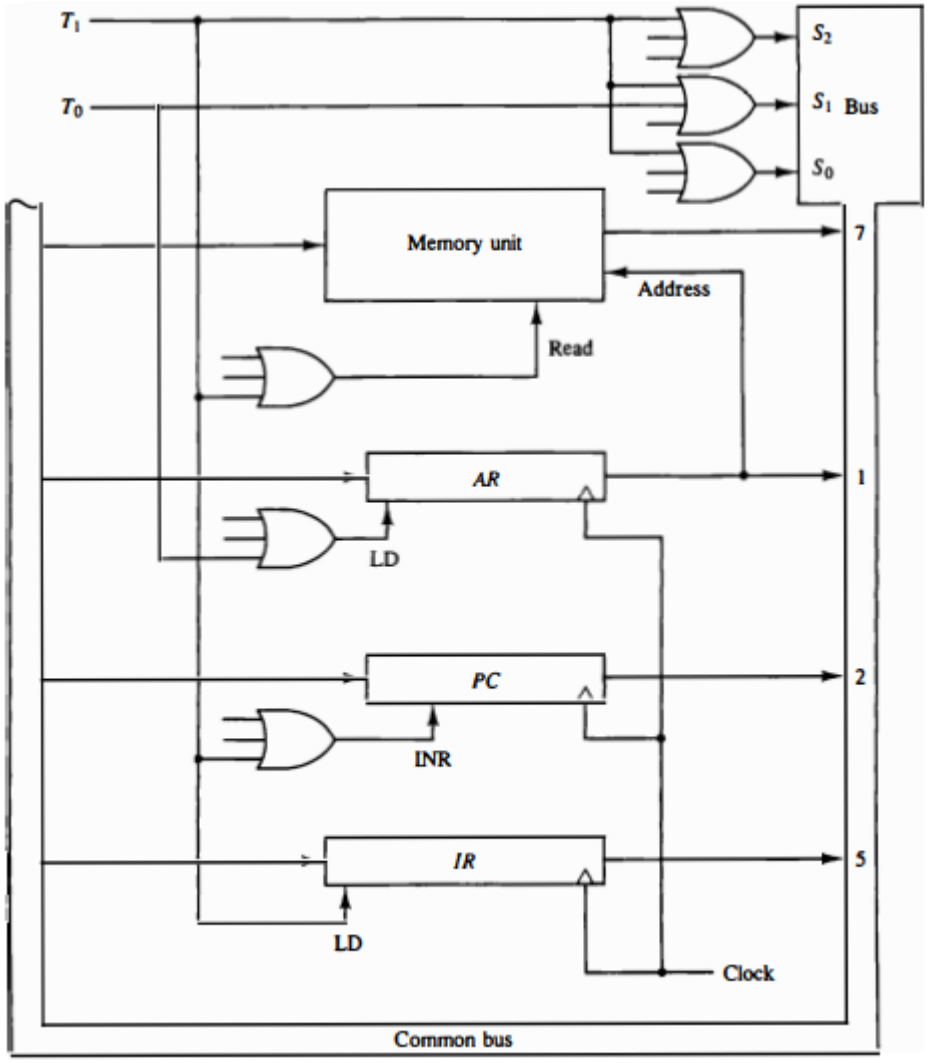
*****Instruction Cycle**

- **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 through 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**
    1. **Initially, the program counter PC** is loaded with the address of the first instruction in the program.
    2. **The sequence counter SC** is cleared to 0, providing a decoded timing signal T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on.
    3. **The rnicrooperations** for the fetch and decode phases can be specified by the following register transfer statements.

    4. T0: AR ← PC
    5. T1: IR ← M[AR], PC ← PC + 1
    6. T2: D0, ... , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)
    7.
    8. **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 T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.



9. **Figure** Register transfers for the fetch phase.

10. **At the same time**, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, 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 .
11. **Note that SC** is incremented after each clock pulse to produce the sequence T0, T1, and T2
12. **Figure above** shows how the first two register transfer statements are implemented in the bus system.
13. **To provide** the data path for the transfer of PC to AR we must apply timing signal T0 to achieve the following connection:

14. 1. Place the content of PC onto the bus by making the bus selection inputs
15. $S_2S_1S_0$ equal to 010.
16.
17. 2. Transfer the content of the bus to AR by enabling the LD input of AR .

18. **The next clock transition** initiates the transfer from PC to AR since T0 = 1. In order to implement the second statement : T1: IR ← M[AR], PC ← PC + 1 it is necessary to use timing signal T1 to provide the following connections in the bus system.

19. 1. Enable the read input of memory.
20. 2. Place the content of memory onto the bus by making $S_2S_1S_0$ = 111.

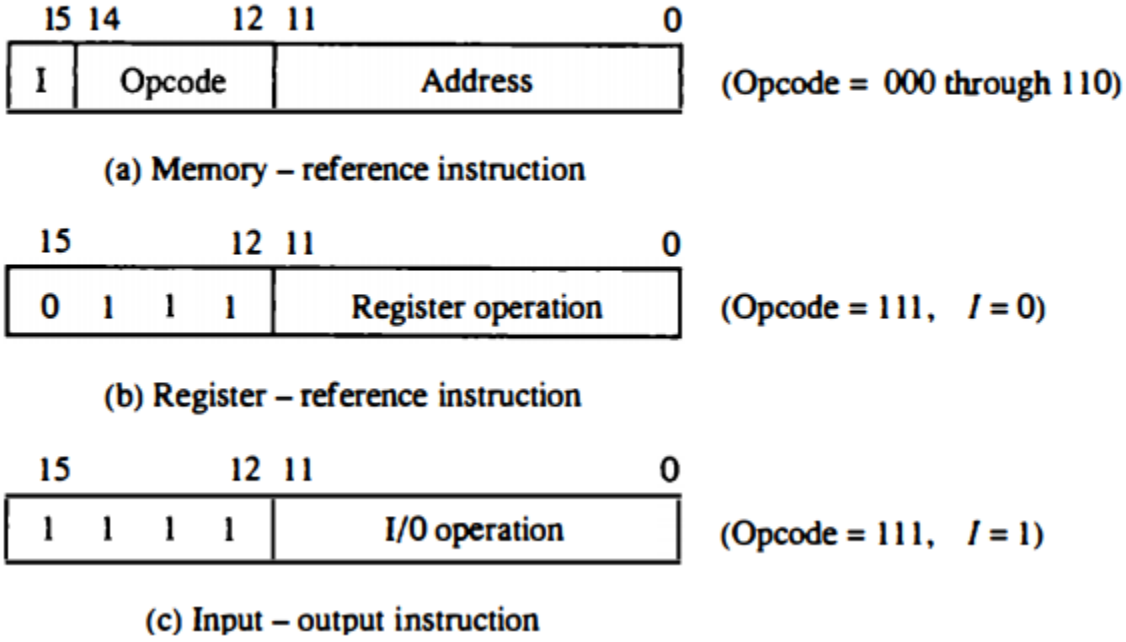
21. 3. Transfer the content of the bus to IR by enabling the LD input of IR.
22. 4. Increment PC by enabling the INR input of PC.

23. **The next clock transition** initiates the read and increment operations since T1 = 1.
24. **Figure above** duplicates a portion of the bus system and shows how T0 and T1 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. below** 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. on basic computer formats.

Figure Basic computer instruction formats.



- **Decoder output D_7 is equal to 1** if the operation code is equal to binary 111. From Fig. on basic computer formats we determine that if $D_7 = 1$, the instruction must be a

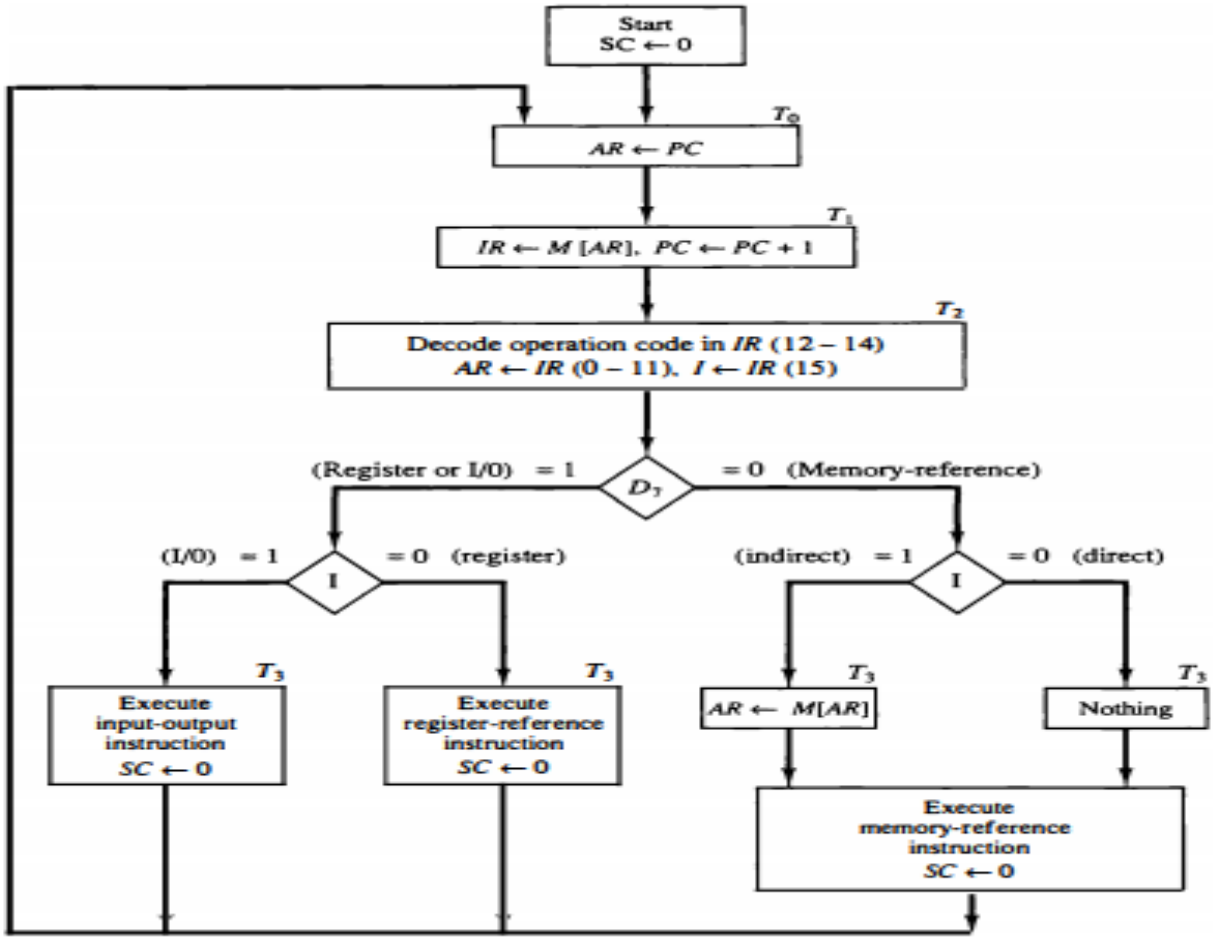


Figure Flowchart for instruction cycle (initial configuration).

- 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 D7 = 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 ← 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:

- $D'_7 IT_3$: AR ← M[AR]
- $D'_7 I'T_3$: Nothing
- $D_7 I'T_3$: 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 $T_0 = 1$.
- **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.

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 IR(0-11). They were also transferred to AR during time T_2.
- **The control functions** and microoperations for the register-reference instructions are. listed in Table below. These instructions are executed with the clock transition associated with timing variable T_3.**Each control function** needs the Boolean 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 IR(0-11). By assigning the symbol B_i to bit i of IR, all control functions can be simply denoted by rB_i.
- **For example**, the instruction CLA has the hexadecimal code 7800, 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 I and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is $D_7I'T_3B_{11} = rB_{11}$.
- **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.
- **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 all the 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.

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 A
CME	rB_8:	$E \leftarrow \overline{E}$	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

******Memory-Reference Instructions***

- **In order** to specify the rnicrooperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.
- **We will now show** that the function of the memory-reference instructions can be defined precisely by means of register transfer notation.
- **Table below** lists the seven memory-reference instructions. The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from 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 microoperations needed for their execution.

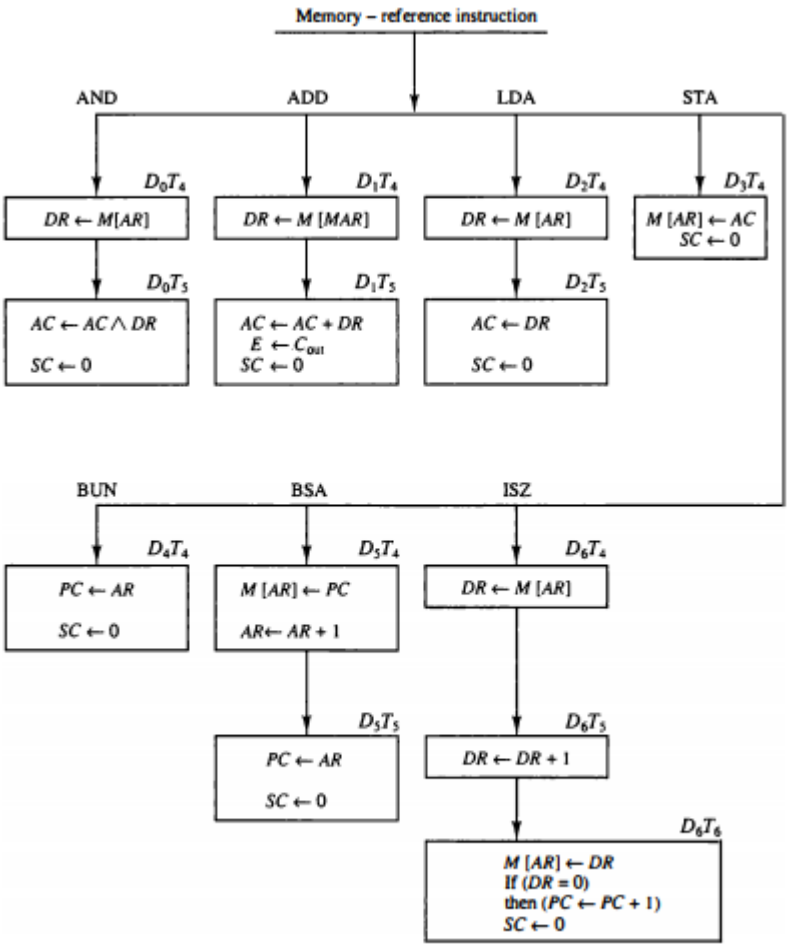
TABLE Memory-Reference Instructions

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

-

Control Flowchart

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



- Figure Flowchart for memory-reference instructions.

***Input-Output and Interrupt

- **A computer can serve no useful purpose** unless it communicates with the external environment. 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.
- **Input-Output Configuration**

1. **The terminal** sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code.
2. **The serial** information from the keyboard is shifted into the input register INPR.
3. **The serial information** for the printer is stored in the output register OUTR.
4. **These two registers** communicate with a communication interface serially and with the AC in parallel. The input-output configuration is shown in Fig. below. The transmitter interface receives serial information from the keyboard and transmits it to INPR.
5. **The receiver** interface receives information from OUTR and sends it to the printer serially.
6. **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

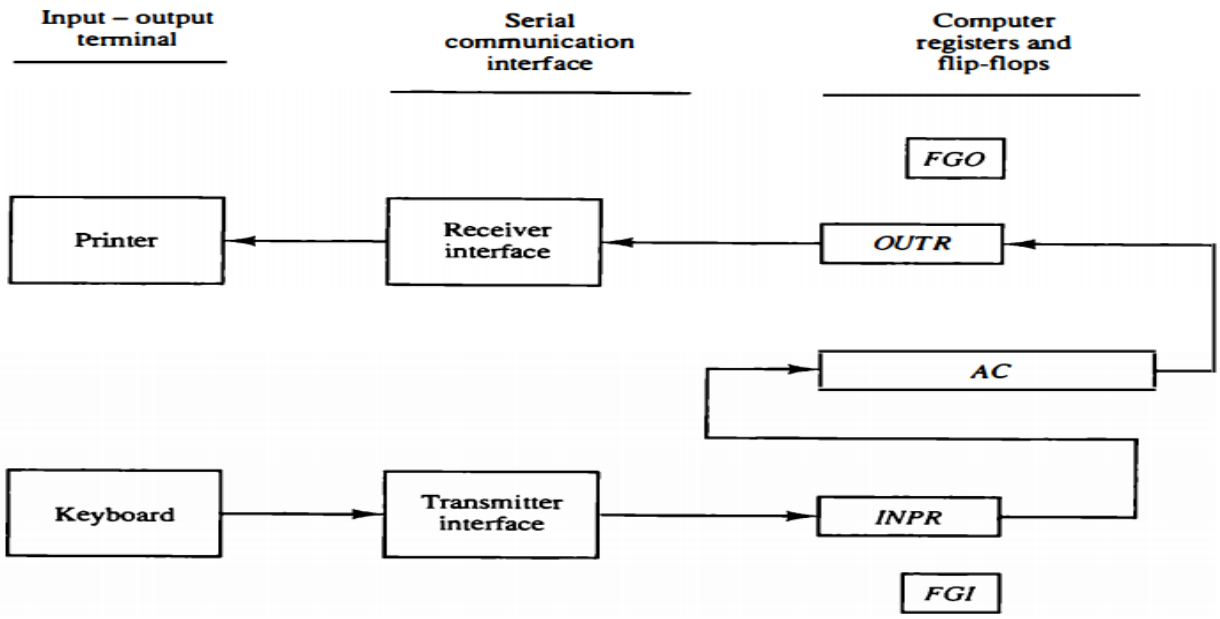
available in the input device and is cleared to 0 when the information is accepted by the computer.

7. **The flag** is needed to synchronize the timing rate difference between the input device and the computer.
8. **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.
9. **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 it is 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.
10. **The output register OUTR** works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1.
11. **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.
12. **The computer** does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

- **Input-Output Instructions**

1. **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.
2. **Input-output instructions** have an operation code 1111 and are recognized by the control when $D_7 = 1$ and $I = 1$. The remaining bits of the instruction specify the particular operation.
3. **The control functions** and microoperations for the input-output instructions are listed in Table below. 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 IR(6-11).
4. **By assigning** the symbol B_i to bit i of IR, all control functions

Figure Input-output configuration.



can be denoted by pB_i for i = 6 though 11. The sequence counter SC is cleared to 0 when $p = D_7IT_3 = 1$.

5. **The INP instruction** transfers the input information from lNPR into the eight low-order bits of AC and also clears the input flag to 0.
6. **The OUT instruction transfers** the eight least significant bits of AC into the output register OUTR and clears the output flag to 0.
7. **The next two instructions** in Table above check the status of the flags and cause a skip of the next instruction if the flag is 1.
8. **The instruction** that is skipped will normally be a branch instruction to return and check the flag again.
9. **The branch instruction** is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed.
10. **The last two instructions set** and clear an interrupt enable flipflop IEN. The purpose of IEN is explained in conjunction with the interrupt operation

****8Program 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 computer that can go through an instruction cycle in 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 100,000 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.

TABLE Input-Output Instructions

$D_7IT_3 = p$ (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}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	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	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

- **The way that the interrupt** is handled by the computer can be explained by means of the flowchart of Fig. above. 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 neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1.
- **At the end of the execute phase**, control checks the value of R, and if it 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 so 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. below. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at 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 Part (a) in Fig. below. 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, PC 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** at 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 or 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 where it was interrupted. This is shown in Part (b) in Fig. below.
- **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 UO service program.

- **After** this instruction is

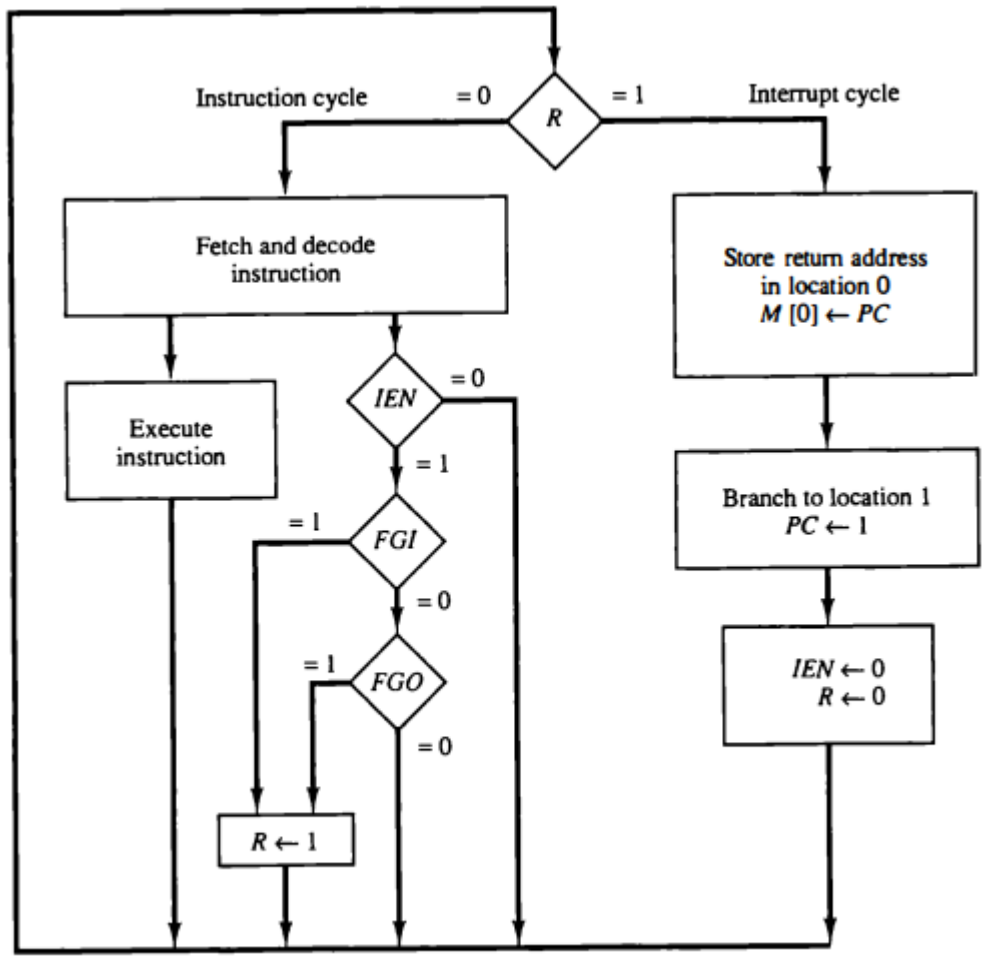


Figure Flowchart for interrupt cycle.

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**

- **The interrupt cycle** is initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IEN = 1 and either FGI or 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(IEN)(FGI+FGO):R←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 or T_2 (as shown in Figure in Section - Determine the Type of Instruction) 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 ← 0, TR ← PC
- RT_1: M[AR] ← TR, PC ← 0
- RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 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 through an instruction cycle that fetches and executes the BUN instruction in location 1.