REGISTERS

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 any binary information of n bits.

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

The computer registers are designated by capital letters to denote the function of the register.

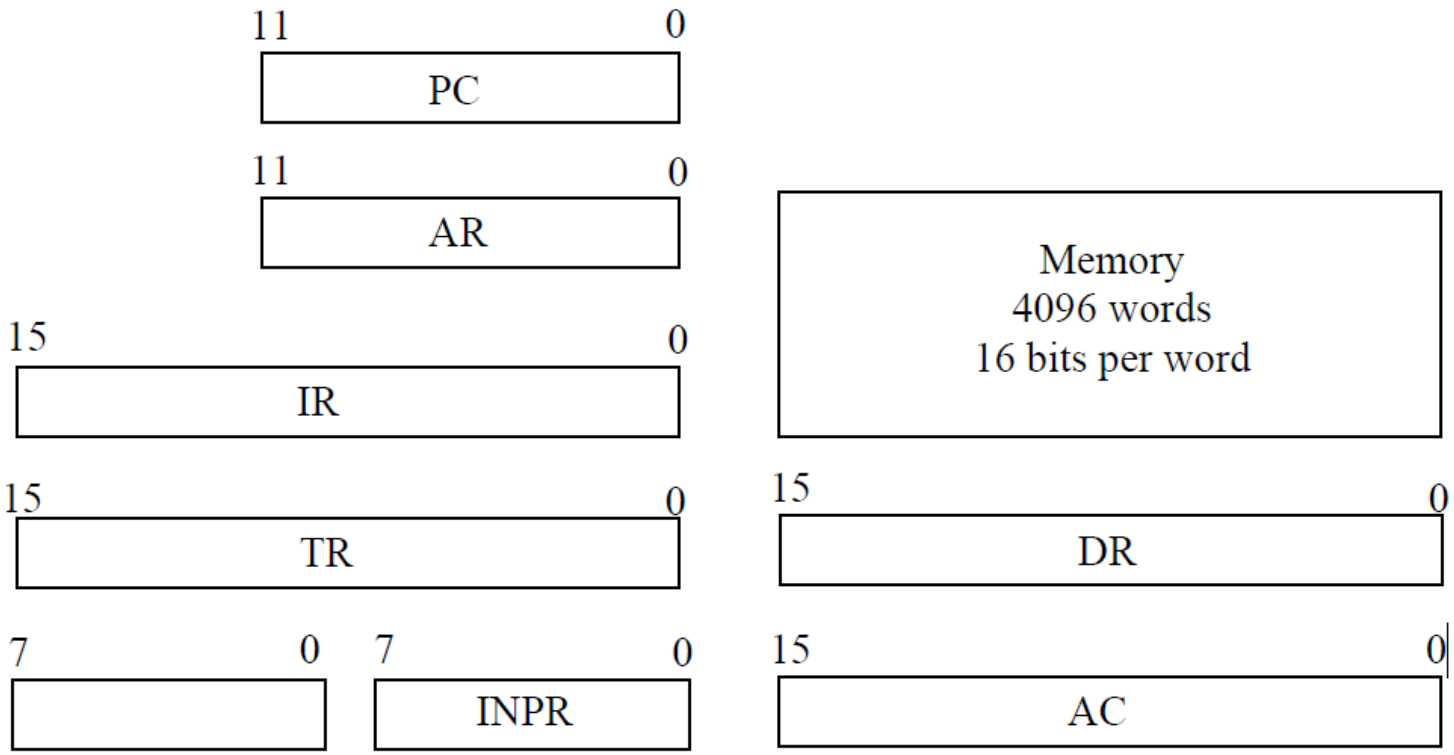
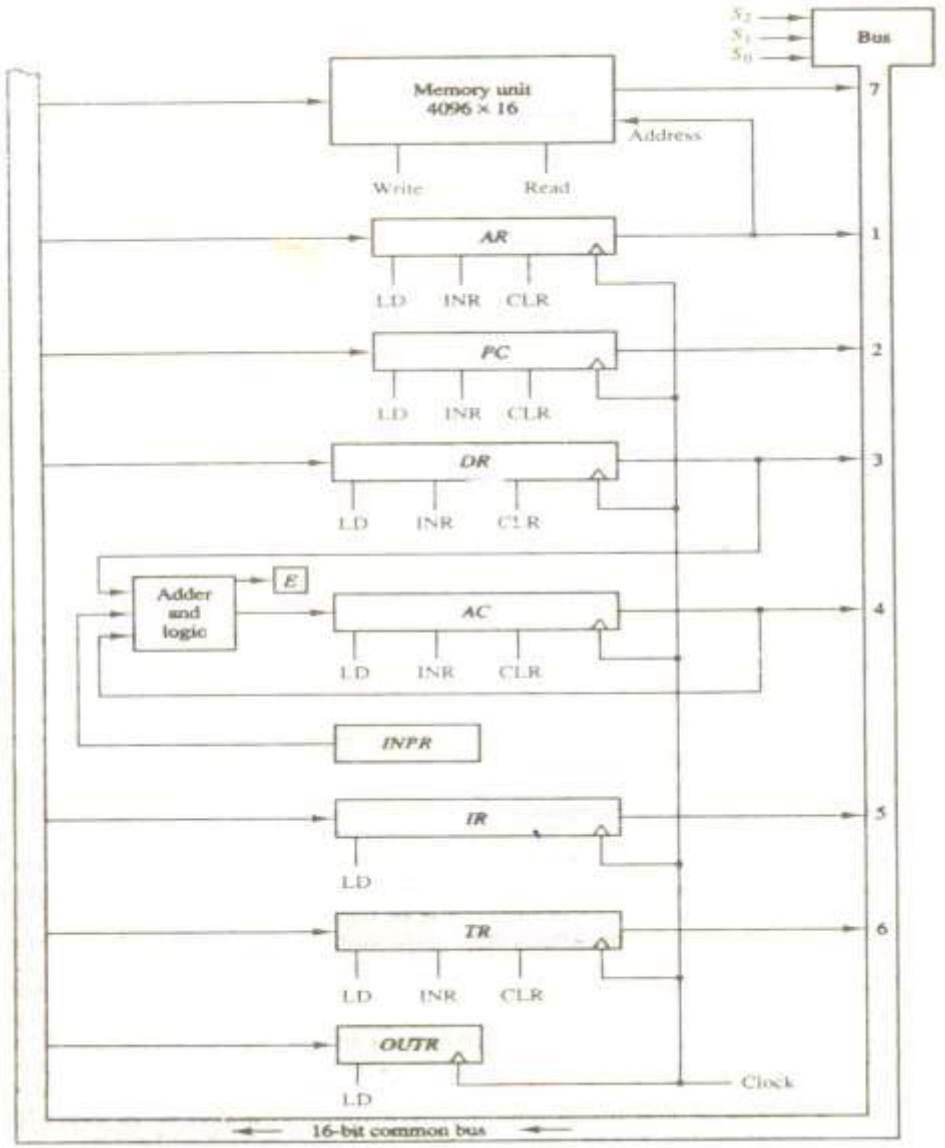


Figure 1-3 Basic computer registers and memory.

COMMON BUS SYSTEM



Computer Instruction

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

Computer Instruction

- The basic computer has three instruction code formats, 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.
- **1. Memory reference instruction**
- **2. Register-reference instruction**
- **3.Input-Output instruction**

- **A memory reference:(opcode=000 through 110,I=0 or 1)** instruction uses 12 bits to specify an address and one bit to specify the addressing mode and 3 bits specify an opcode
  - I is equal to 0 for direct address and to 1 for indirect address.
- **The register-reference: (opcode=111, I=0)** 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.
- **Input-Output instruction: (opcode=111, I=1)**

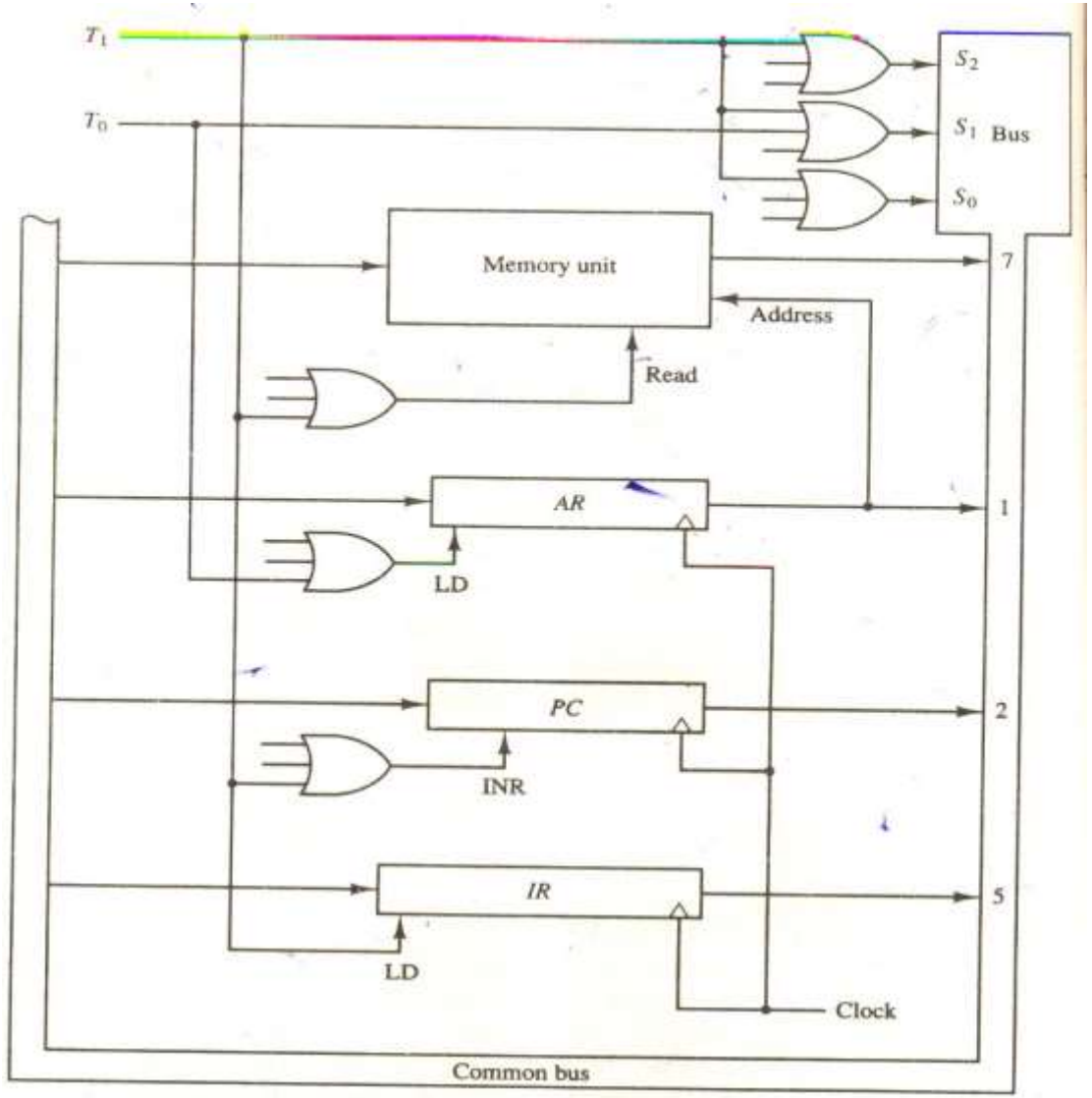
INSTRUCTION CYCLE

- 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 sub cycles or phases.
- In the basic computer each instruction cycle consists or 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.**

FETCH AND DECODE

- T0: AR ← PC
- T1: IR ← M[AR], PC ← PC + 1
- T2: D0,…, D1 ← Decode IR (12-14), AR ← IR(0 −11), 1 ← IR (15)

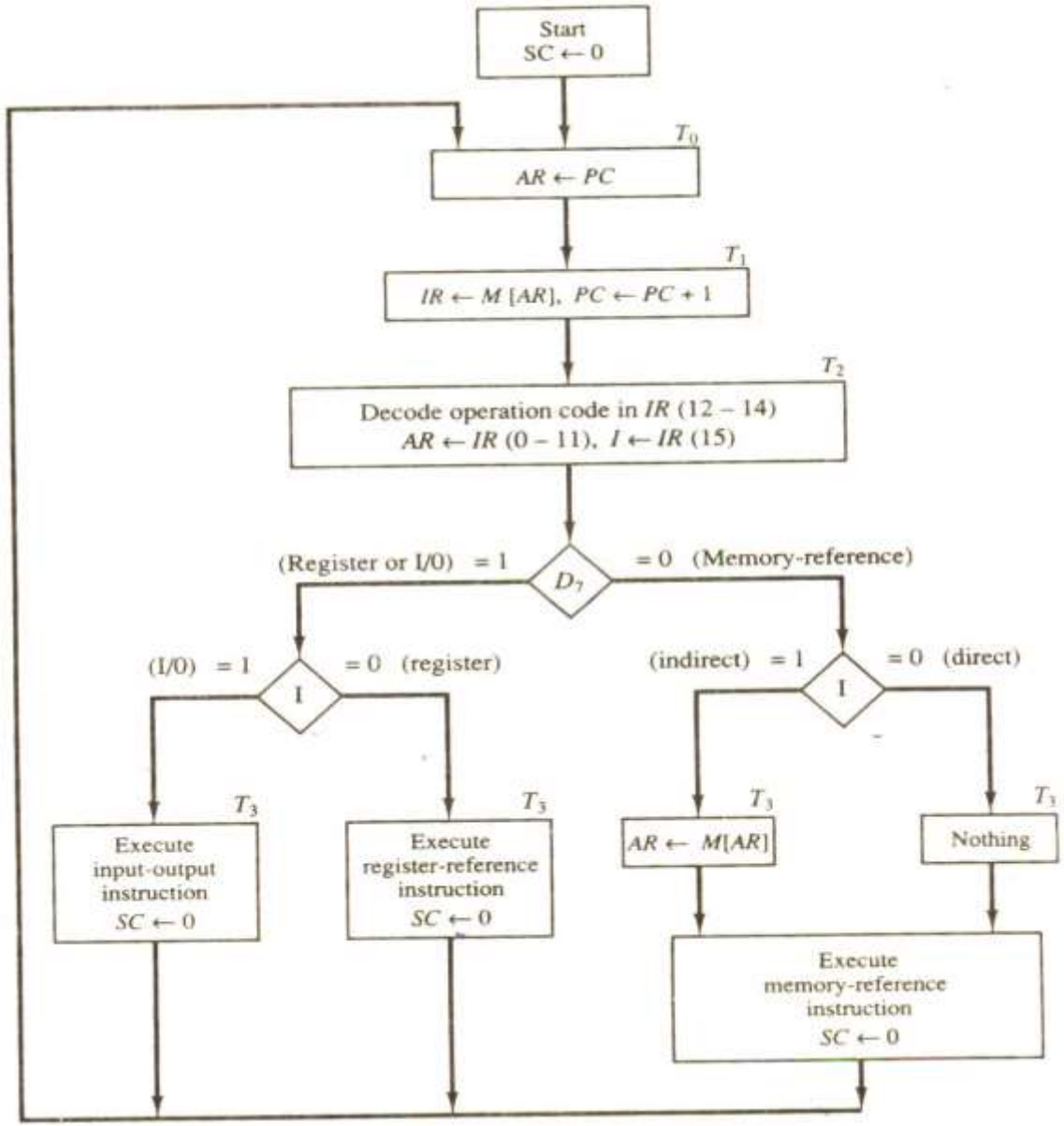
Instruction cycle diagram



Instruction cycle

- The Above 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 T0 to achieve the following connection.:
- 1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.
- 2. Transfer the content of the bus to AR y enabling the LD input of AR.
- The next clock transition initiates the transfer form 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.
- 1. Enable the read input of memory.
- 2. Place the content of memory onto the bus by making S2S1S0 = 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.

Flowchart for instruction cycle



Addressing Modes

- The operation field of an instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers or memory words.
- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

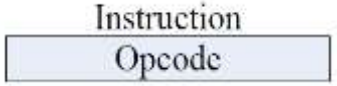
1. To give programming versatility to the user by providing such facilities as pointers to Memory, counters for loop control, indexing of data, and program relocation
2. To reduce the number of bits in the addressing field of the instruction.
3. The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.

Types of addressing modes

1. Implied Addressing Mode.
2. Immediate Addressing Mode
3. Register Addressing Mode
4. Register Indirect Addressing Mode
5. Auto-increment or Auto-decrement Addressing Mode
6. Direct Addressing Mode
7. Indirect Addressing Mode
8. Relative Addressing Mode
9. Index Addressing Mode
10. Base Register Addressing Mode

Implied Addressing Mode.

- In this mode the operands are specified implicitly in the definition of the instruction.
- For example, the instruction “complement accumulator” is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.



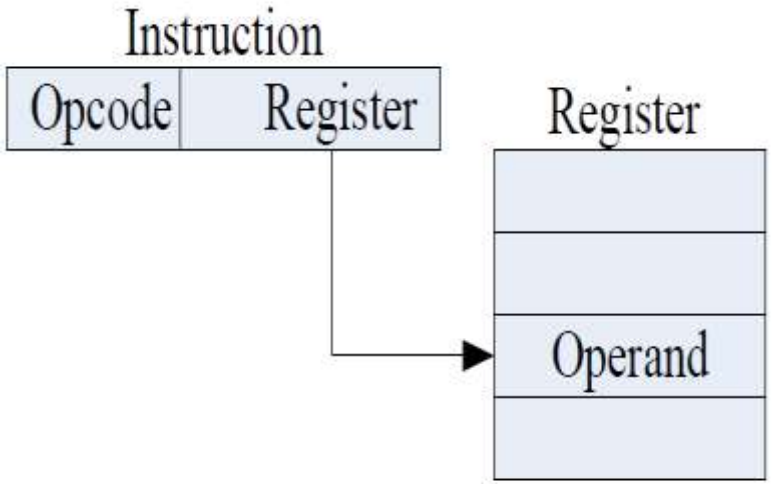
- Advantage: no memory reference.
- Disadvantage: limited operand
-

Immediate Addressing Mode

- In this mode the operand is specified in the instruction itself.
- In other words, an immediate-mode instruction has an operand field rather than an address field.
- For example MVI B, 50H

Register Addressing Mode

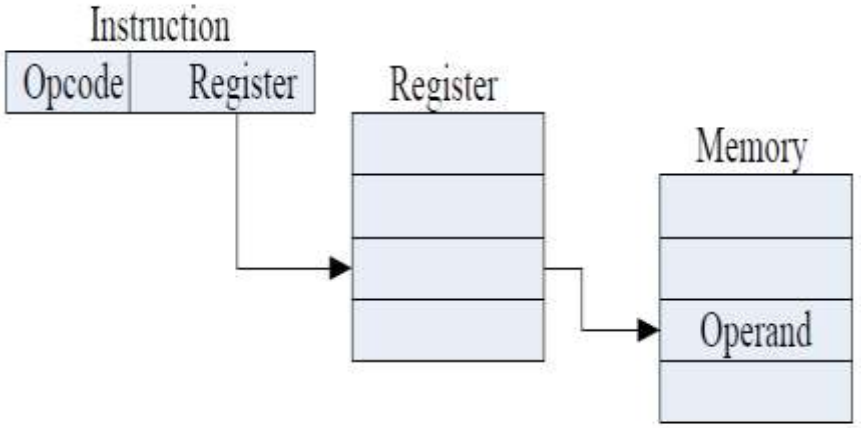
- In this mode, the operands are in registers that reside within the CPU. The particular register is selected from the register field in the instruction. A k-bit field can specify any one of 2^k registers
- For example MOV A, B



- Effective Address (EA) = R
- Advantage: no memory reference.
- Disadvantage: limited address space

Register Indirect Addressing Mode

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.



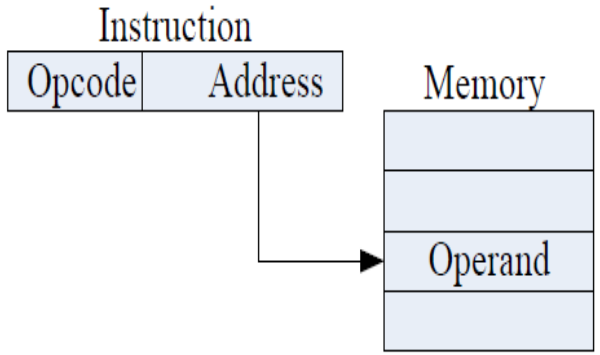
- Effective Address (EA) = (R)
- Advantage: Large address space.
  - The address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
- Disadvantage: Extra memory reference

Auto-increment or Auto-decrement Addressing Mode

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

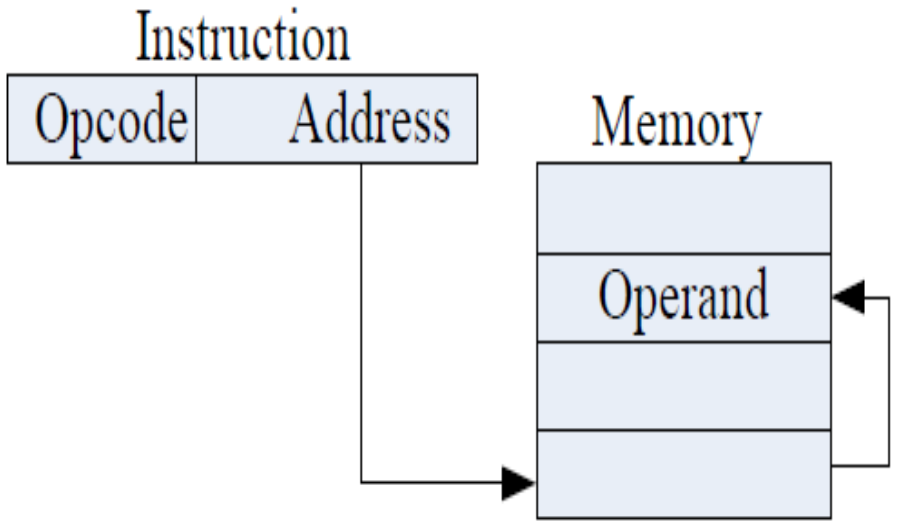
Direct Address Mode:

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the adder field of the instruction.



Indirect Address Mode:

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.



- **Effective address=address part of the instruction + content of CPU register**

Relative Addressing Mode

- In this mode the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (either a +ve or a –ve number).
- When the number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
- Effective address= address part of the instruction + content of the PC.

Indexed Addressing Mode

- In this mode the content of an index register (XR) is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- Note: If an index-type instruction does not include an address field in its format, the instruction is automatically converted to the register indirect mode of operation.
- Effective Address (EA) = Content of indexed register(XR) + Address part of the instruction.

• Base Register Addressing Mode :

- In this mode the content of a base register (BR) is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of the index register.
- Effective Address (EA) = Content of indexed register(BR) + Address part of the instruction.

Example

ADRS or NBR =500

PC=250

R1=400

ACC

Addressing mode	Symbolic conversion	Register transfer	Efective address	Contet of ACC
Immediate	LDA #NBR	ACC←NBR	-	500
Register	LDA R1	ACC←R1	-	400
Register-indirect	LDA (R1)	ACC←M[R1]	400	700
Direct	LDA ADRS	ACC←M[ADRS]	500	800
Indirect	LDA [ADRS]	ACC←M[M[ADRS]]	800	300
Relative	LDA $ADRS	ACC←M[ADRS+PC]	750	600
Index	LDA ADRS(R1)	ACC←M[ADRS+R1]	900	200

250	Opcode	Mode
251		
	Next instruction	
252		
	700	
400		
	800	
500		
750	600	
	300	
800		
	200	
900		

REGISTER TRANSFER LANGUAGE

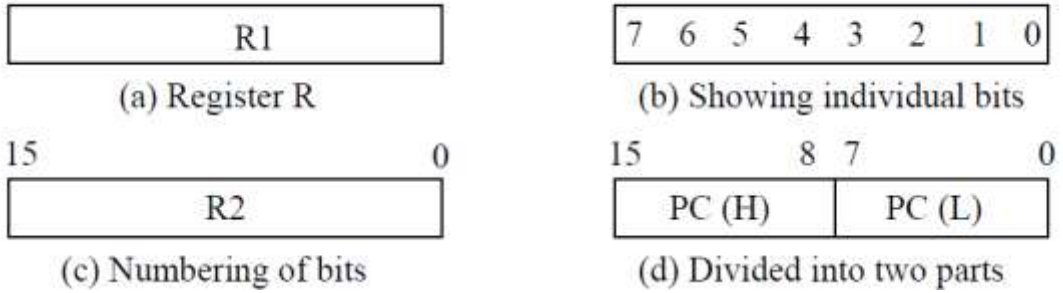
- The operations executed on date stored in registers are called microoperations.
- A microoperation is an elementary operation performed on the on the information stored in one or more registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- Examples of microoperations are shift, count, clear, and load.
- Registers that implement microoperations. For example, a counter with parallel load is capable of performing the micro operations increment and load.
- A bidirectional shift register is capable of performing the shift right and shift left microoperations.
- The internal hardware organization of a digital computer is best defined by specifying.
- 1. The set of registers it contains and their function.
- 2. The sequence of microoperations performed on the binary information stored in the registers.
- 3. The control that initiates the sequence of microoperations.

REGISTER TRANSFER LANGUAGE

- The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.
- The term “register transfer” implies the availability of hardware logic circuits that can perform a stated microoperation 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.
- A programming language is a procedure for writing symbols to specify a given computational process.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement denotes a transfer of the content of register R1 into register R2

$$R2 \leftarrow R1$$

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



REGISTER TRANSFER LANGUAGE

- This can be shown by means of an if-then statement.

$$\text{If } (P = 1) \text{ then } (R2 \leftarrow R1)$$

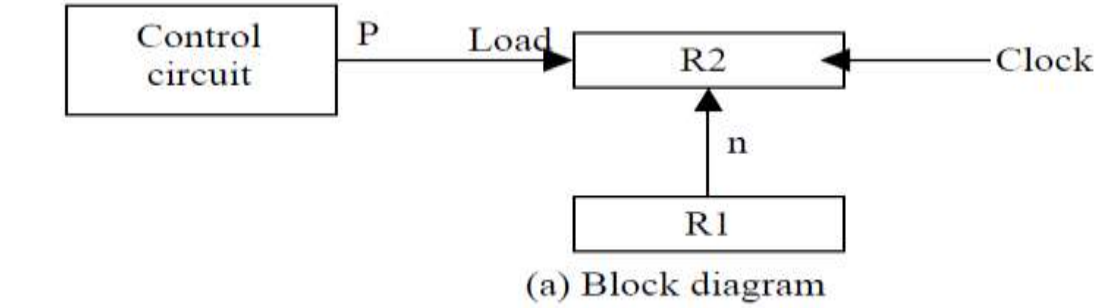
- where P is a control signal generated in the control section. It is sometimes convenient to separate the
- control variables from the register transfer operation by specifying a control function.
- A control function is a Boolean variable that is equal to 1 or 0. The control function is included in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

$$P : R2 \leftarrow R1$$

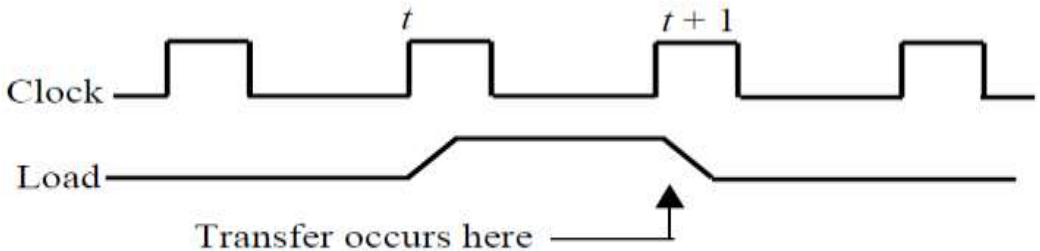
- The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1.

REGISTER TRANSFER LANGUAGE

- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
- Figure 2.2 shows the block diagram that depicts the transfer from R1 to R2.
- The letter n will be used to indicate any number of bits for the register.
- It will be replaced by an actual number when the length of the register is known. Register R2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.



(a) Block diagram



(b) Timing diagram

REGISTER TRANSFER LANGUAGE

TABLE 2-1 Basic Symbols for Register Transfers

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

Bus system for four registers

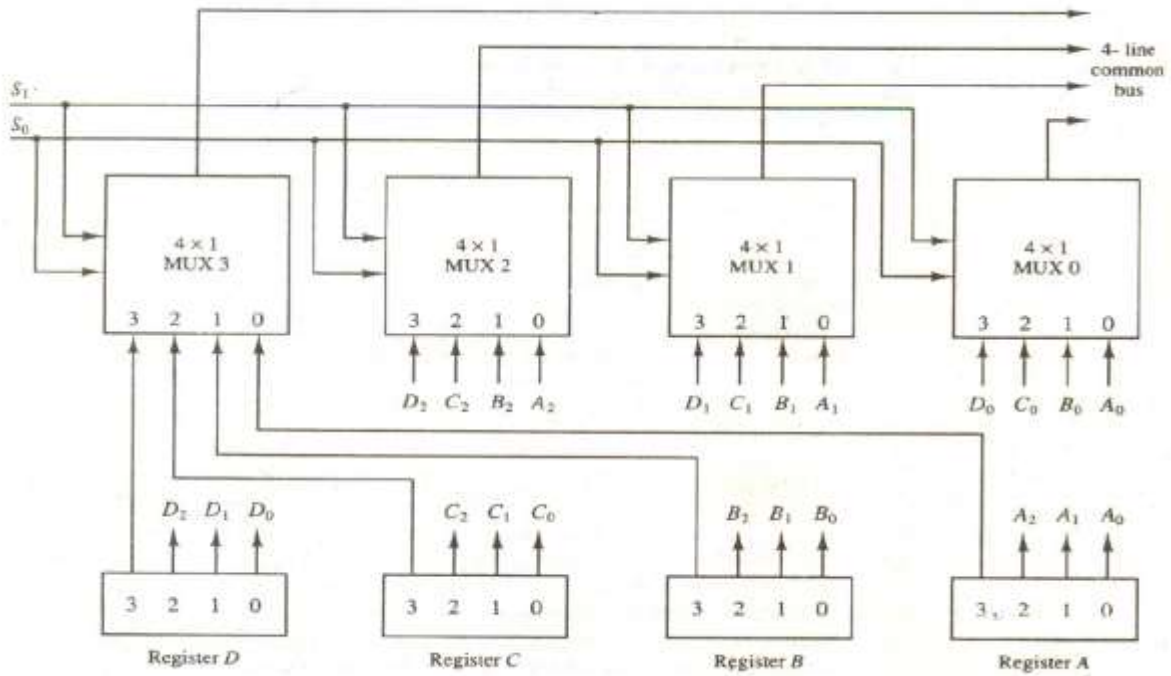


TABLE 2-2

Function Table for Bus of Fig. 2-3

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

INSTRUCTION FORMAT:

- **The most common fields are:**

  1).Operation field which specifies the operation to be performed like addition.

  2).Address field which contain the location of operand, i.e., register or memory location.

  3).Mode field which specifies how operand is to be founded.

- An instruction is of various length depending upon the number of addresses it contain.
- Generally CPU organization are of three types on the basis of number of address fields:

  1).Single Accumulator organization

  2).General register organization

  3).Stack organization

INSTRUCTION FORMAT

- In first organization operation is done involving a special register called accumulator.
- In second on multiple registers are used for the computation purpose.
- In third organization the work on stack basis operation due to which it does not contain any address field.
- It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.
- On the basis of number of address, instruction are classified as:
- Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

One Address Instructions:

- This use a implied ACCUMULATOR register for data manipulation.
- One operand is in accumulator and other is in register or memory location.
- Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

One Address Instructions:

- Implied address: a register called *an accumulator ACC* for one operand and the result, *single-accumulator architecture*

LD	A	ACC ← M[A]
ADD	B	ACC ← ACC + M[B]
ST	X	M[X] ← ACC
LD	C	ACC ← M[C]
ADD	D	ACC ← ACC + M[D]
MUX	X	ACC ← ACC X M[X]
ST	X	M[X] ← ACC

7 instructions

- All operations are between the ACC register and a memory operand

Two Address Instructions:

- This is common in commercial computers.
- Here two address can be specified in the instruction.
- Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

Two Address Instructions

- The first operand address also serves as the implied address for the result

MOVE T1, A	M[T1] ← M[A]
ADD T1, B	M[T1] ← M[T1] + M[B]
MOVE X, C	M[X] ← M[C]
ADD X, D	M[X] ← M[X] + M[D]
MUX X, T1	M[X] ← M[X] X M[T1]

- 5 instructions

Three Address Instructions:

- This has three address field to specify a register or a memory location.
- Program created are much short in size but number of bits per instruction increase.
- These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

Three Address Instructions

- Example: X=(A+B)(C+D)
- Operands are in memory address symbolized by the letters A,B,C,D, result stored memory address of X

	ADD T1, A, B	M[T1] ← M[A] + M[B]
	ADD T2, C, D	M[T2] ← M[C] + M[D]
	MUX X, T1, T2	M[X] ← M[T1] X M[T2]
OR		
	ADD R1, A, B	R1 ← M[A] + M[B]
	ADD R2, C, D	R2 ← M[C] + M[D]
	MUX X, R1, R2	M[X]← R1 X R2

- +: Short program, 3 instructions
- -: Binary coded instruction require more bits to specify three addresses

Zero Address Instructions:

- A stack based computer do not use address field in instruction.
- To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Zero Address Instructions:

- Use stack(FILO):
  - ADD
  - PUSH X
  - POP X

	TOS ← TOS + TOS_{-1}
PUSH A	TOS ← M[X]
PUSH B	M[X] ← TOS
ADD PUSH	TOS ← M[A]
C PUSH D	TOS ← M[B]
ADD	TOS ← TOS + TOS_{-1}
MUX	TOS ← M[C]
POP X	TOS ← M[D]
	TOS ← TOS + TOS_{-1}
	TOS ← TOS X TOS_{-1}
	M[X] ← TOS

8 instructions

- Data manipulation operations: between thestack elements
- Transfer operations: between the stack andthe memory

Stack Instructions

- Push:
  - SP ← SP-1; TOS ← R1
- Pop:
  - R1 ← TOS; SP ← SP +1