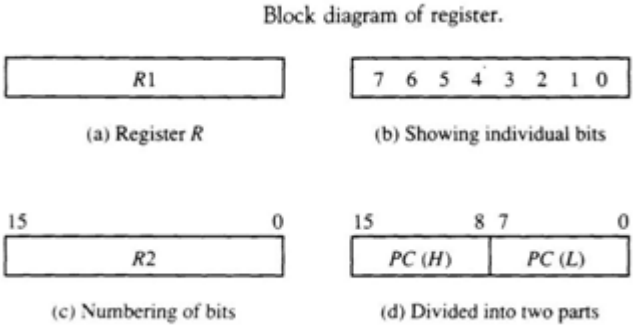
Registers in Computer Architecture:

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

P: R2 ← R1

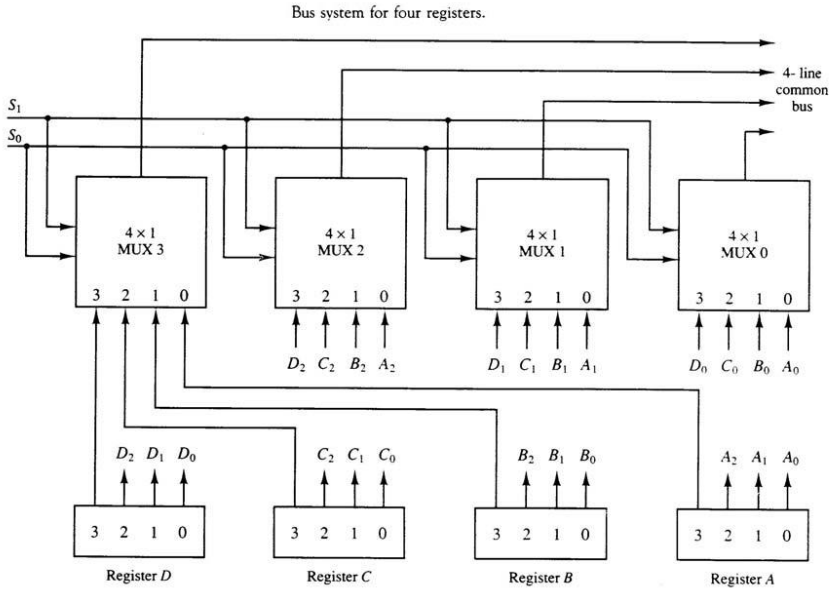
- The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

Bus System for Registers:

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.
- A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system.
- A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.
- In general, a bus system will multiplex *k* registers of *n* bits each to produce an *n*-line common bus.
- This requires *n* multiplexers – one for each bit.
- The size of each multiplexer must be *k* x 1.
- The number of select lines required is *log k.*
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated.

Functional table for Bus

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D



Three State-Bus Buffers

- A bus can be built using three-state buffers instead of multiplexers.
- A three-state gate has three states: 1, 0 and a high impedance state, which behaves like an open circuit.
- It is possible to connect a large number of three state gates in a common bus line without overloading it.
- The three-state buffer gate has a normal input and a control input which determines the output state.
- With control 1, the output equals the normal input.
- With control 0, the gate goes to a high-impedance state.
- This enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loading effects.

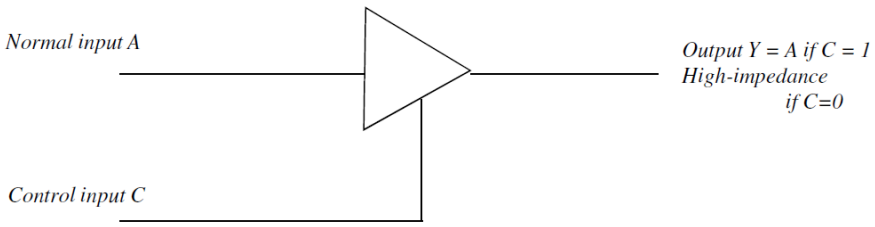


Fig: Graphic symbol for three-state buffer.

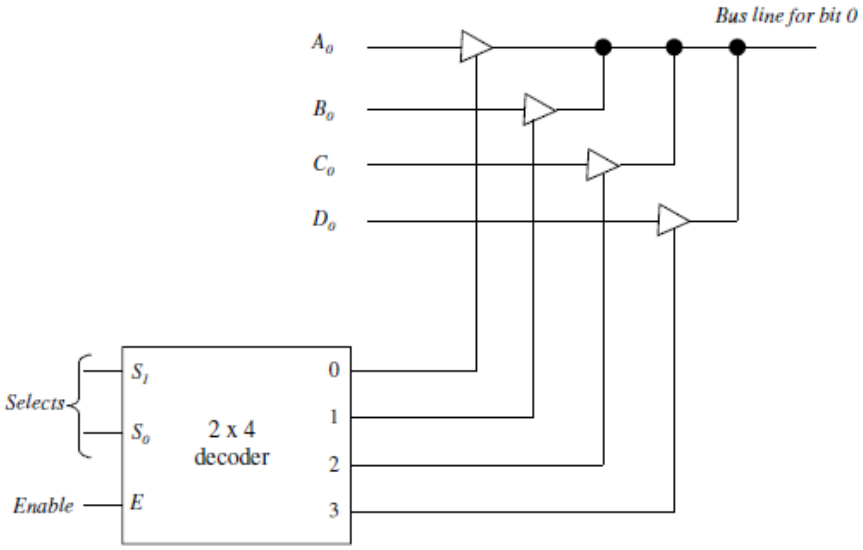


Fig: Bus Line with Three-state Buffers

- Decoders are used to ensure that no more than one control input is active at any given time. This circuit can replace the multiplexer.
- To construct a common bus for four registers of *n* bits each using three-state buffers, we need *n* circuits with four buffers in each.
- Only one decoder is necessary to select between the four registers.
- Designate a memory word by the letter M.
- It is necessary to specify the address of M when writing memory transfer operations.
- Designate the address register by AR and the data register by DR.
- The read operation can be stated as:

  Read: DR ← M[AR]
- The write operation can be stated as:

  Write: M[AR] ← R1

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.

a) 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 → 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 → 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**

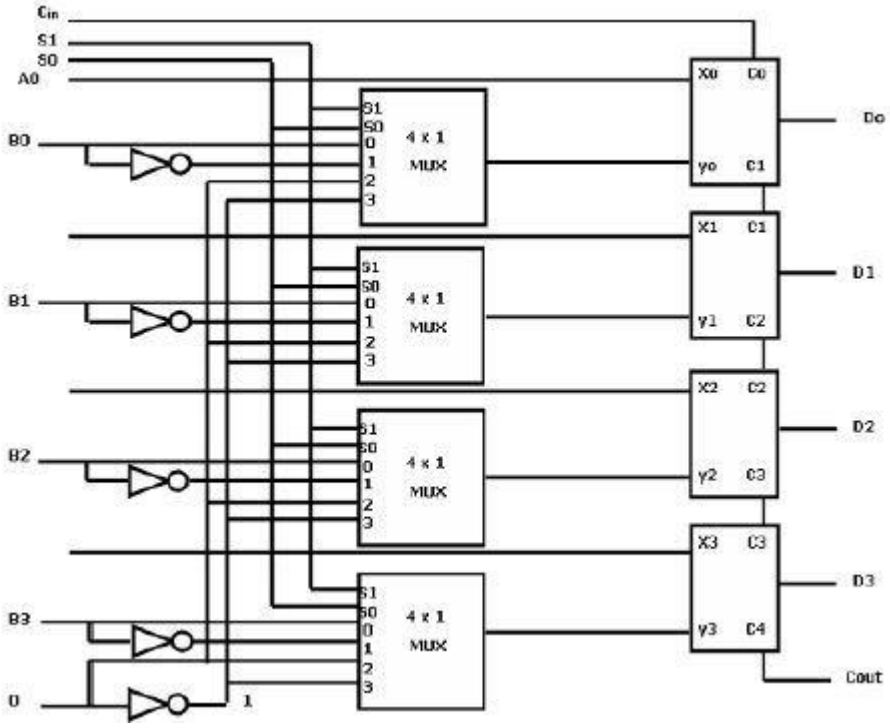
Increment/Decrement Micro-Operation

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

R1 → R1 + 1

R1 → 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.



s_1	s_0	c_{in}	Y	D=A+Y+c_{in}	Microoperation
0	0	0	B	D=A+B	Add
0	0	1	B	D=A+B+1	Add with Carry
0	1	0	B	D=A+$\bar{B}$	Subtract with barrow
0	1	1	B	D= A+$\bar{B}$+1	Subtract
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

b) 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.

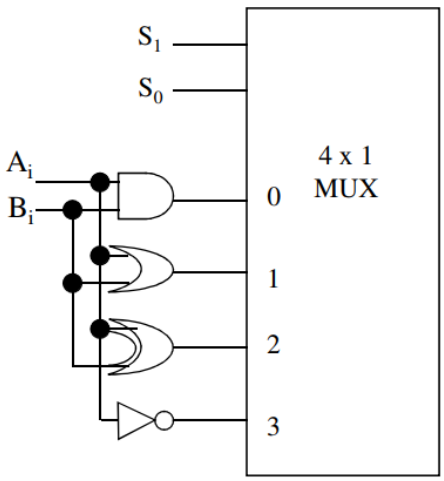
Assume that each register has 3 bits. Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

010 → R1

100 → R2

110 → R1 after P=1

One Stage of Logic Circuit



S_1	S_0	Output	Operation
0	0	E = A ^ B	AND
0	1	E = A ∨ B	OR
1	0	E = A ⊕ B	XOR
1	1	E = A'	Complement

c) 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.

$R1 \leftarrow she\ R1$

$R1 \leftarrow she\ R1$

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

b) Circular Shift

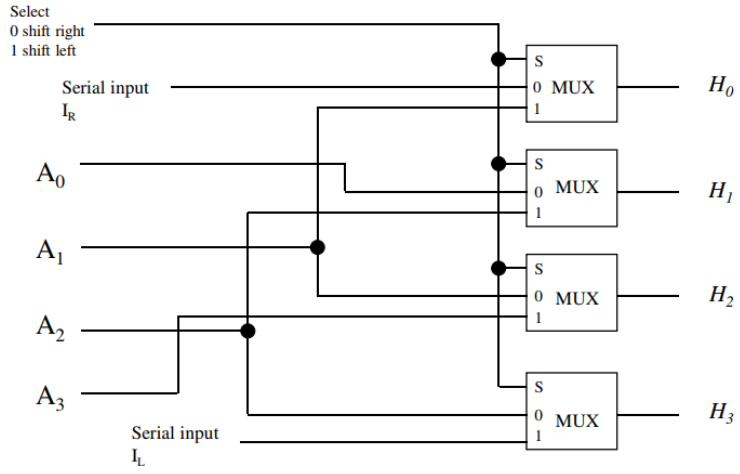
This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

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.

Symbolic Designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic Shift-left register R
R ← ashr R	Arithmetic Shift-right register R

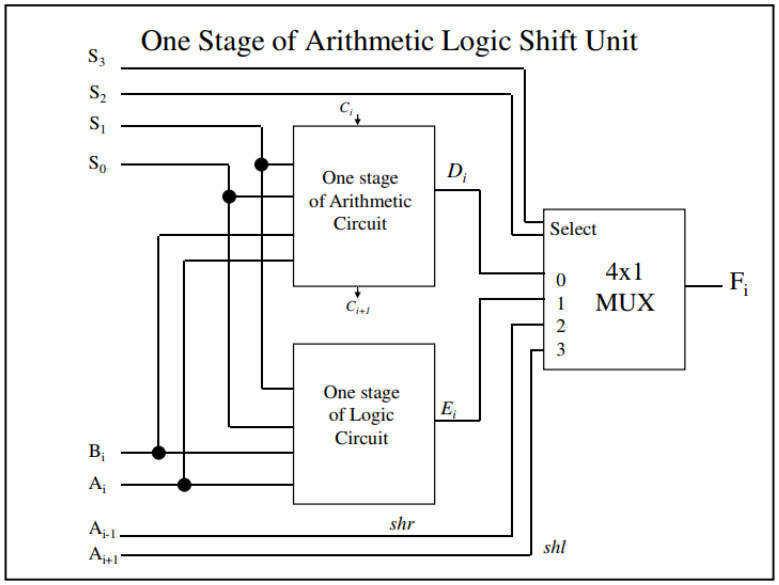
4-Bit Combinational Circuit Shifter



S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Arithmetic Logical Unit

Instead of having individual registers performing the micro-operations, computer system provides a number of registers connected to a common unit called as Arithmetic Logical Unit (ALU). ALU is the main and one of the most important unit inisde CPU of computer. All the logical and mathematical operations of computer are performed here. The contents of specific register is placed in the in the input of ALU. ALU performs the given operation and then transfer it to the destination register.



Instruction Code

An instruction code is a group of bits that instruct the computer to perform a specific operation.

Operation Code

The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Accumulator (AC)

Computers that have a single-processor register usually assign to it the name accumulator (AC) accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

Stored Program Organization

- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

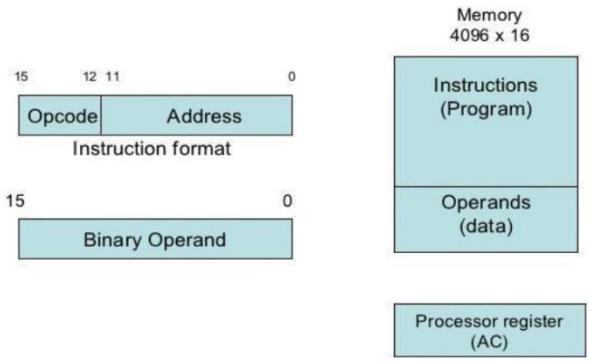


Figure : Stored Program Organization

- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words, we need 12 bits to specify an address since 2^{12}=4096.
- If we store each instruction code in one 16-bit memory word, we have available four bits for operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- The control reads a 16-bit instruction from the program portion of memory.
- It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
- It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name accumulator and label it AC.
- If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.
- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory. For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

Direct and Indirect addressing of basic computer.

- The second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The mode bit is 0 for a direct address and 1 for an indirect address.
- A direct address instruction is shown in Figure 2.2. It is placed in address 22 in memory.
- The I bit is 0, so the instruction is recognized as a direct address instruction.
- The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Figure 2.3 has a mode bit I = 1, recognized as an indirect address instruction.
- The address part is the binary equivalent of 300.
- The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC.

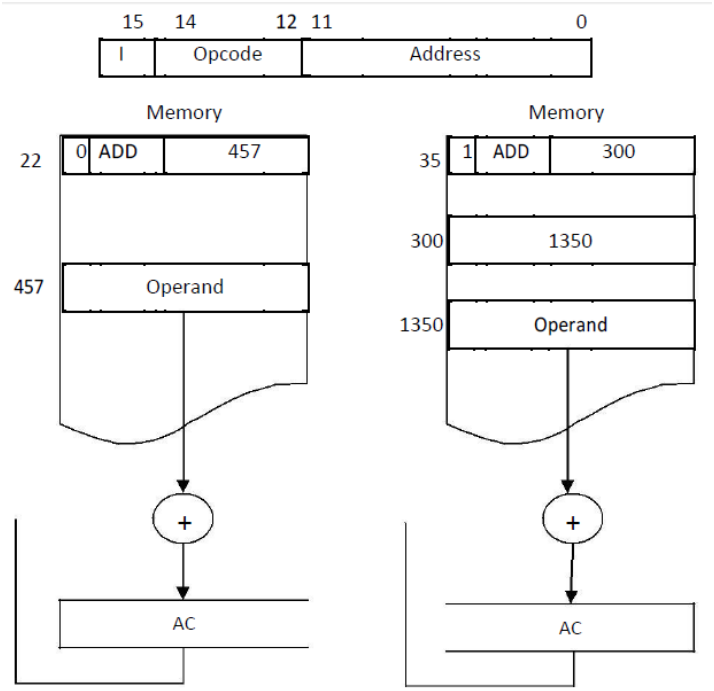


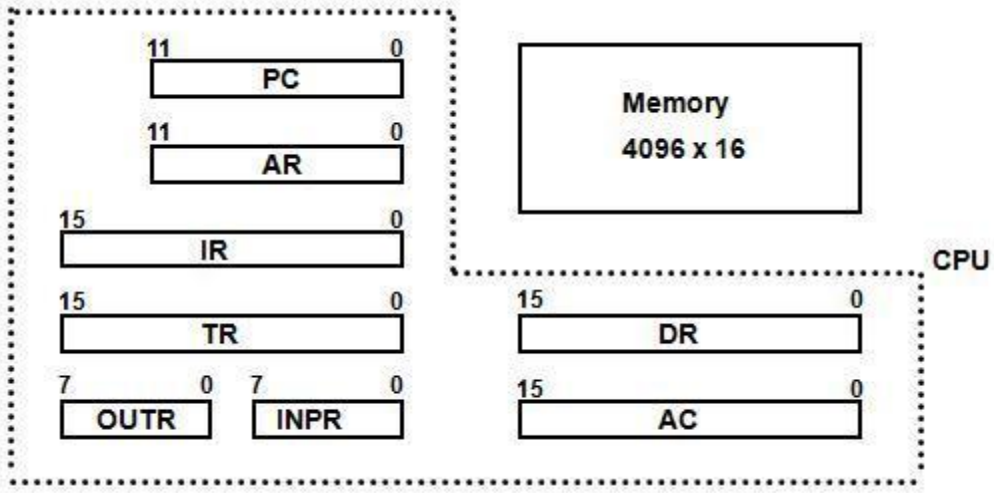
Figure 2.2: Direct Address

Figure 2.3: Indirect Address

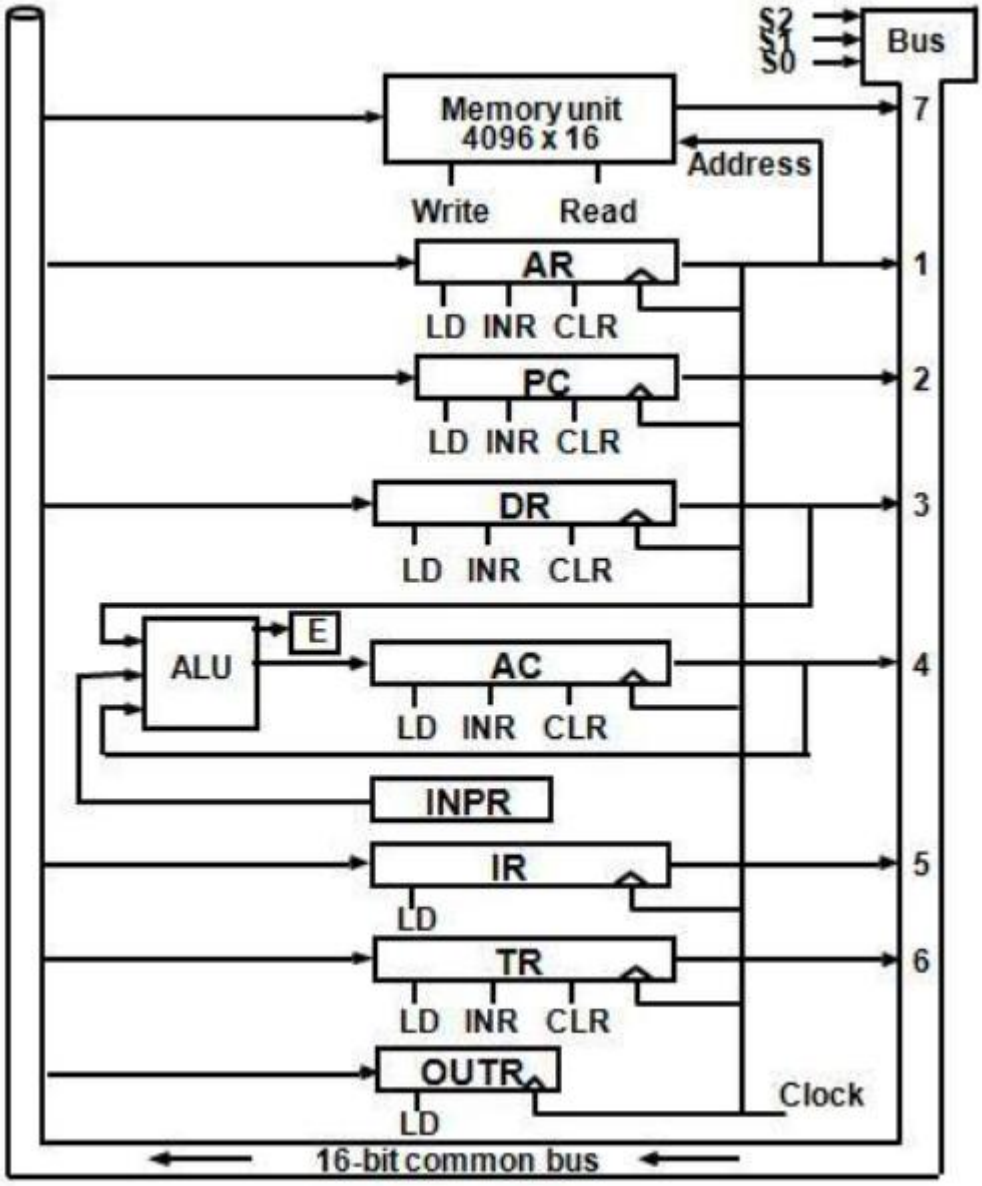
Direct Address	Indirect Address
When the second part of an instruction code specifies the address of an operand, the instruction is said to have a direct address.	When the second part of an instruction code specifies the address of a memory word in which the address of the operand, the instruction is said to have a direct address.
For instance the instruction MOV R000H. R0, when converted to machine language is the physical address of register R0. The instruction moves 0 to R0.	For instance the instruction MOV @R0 00H, when converted to machine language, @R0 becomes whatever is stored in R0, and that is the address used to move 0 to. It can be whatever is stored in R0.

Registers of basic computer

- It is necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address.
- These requirements dictate the register configuration shown in Figure.

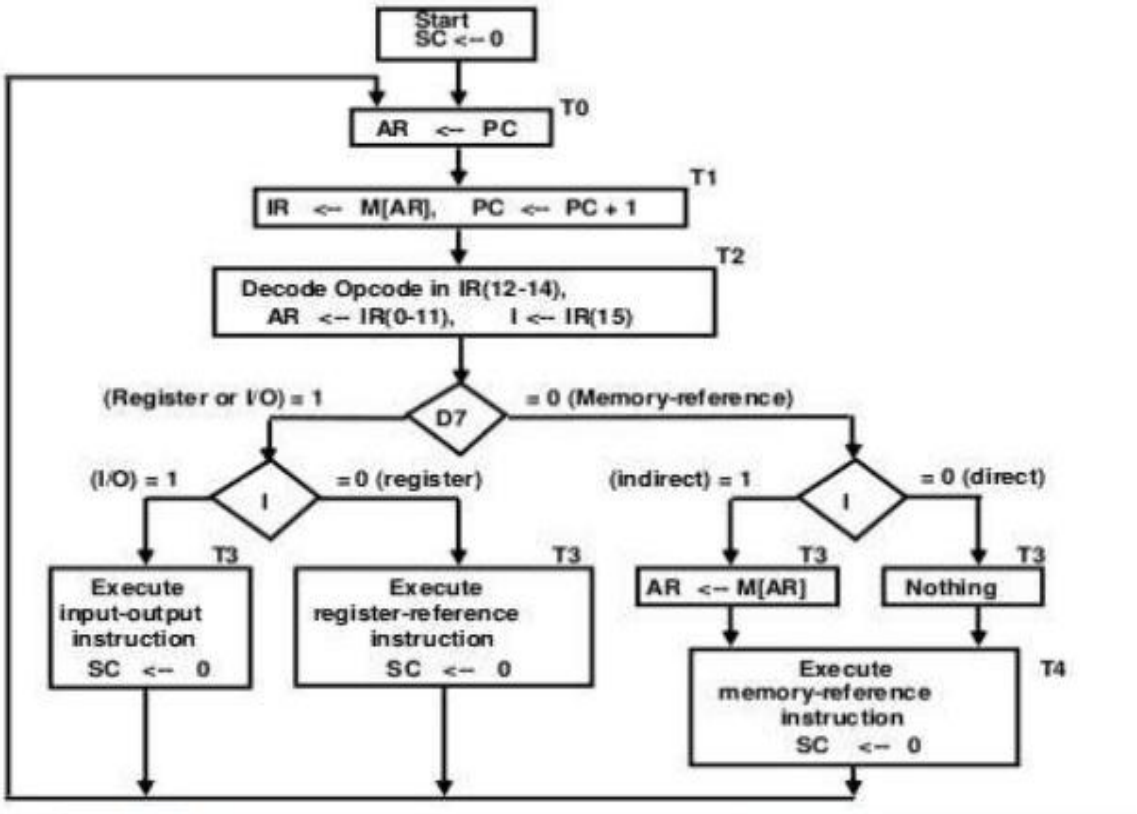


Register Symbol	Bits	Register Name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds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



Instruction cycle:

- A program residing in the memory unit of the computer consists of a sequence of instructions. 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.
- After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.
- This process continues unless a HALT instruction is encountered.
- The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- If D7 = 1, the instruction must be register-reference or input-output type. If D7 = 0, the operation code must be one of the other seven values 110, specifying a memory reference instruction. Control then inspects the value of the first bit of the instruction, which 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.
- 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'7 I T3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T4.



There are three types of instructions,

a) Register Reference Instructions
b) Memory Reference Instructions
c) Input-Output Instructions

a) Register Reference Instructions

There are 12 register-reference instructions listed below:

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

b) Memory Reference Instructions

The following table lists seven memory-reference instructions.

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

c) Input-Output Instructions

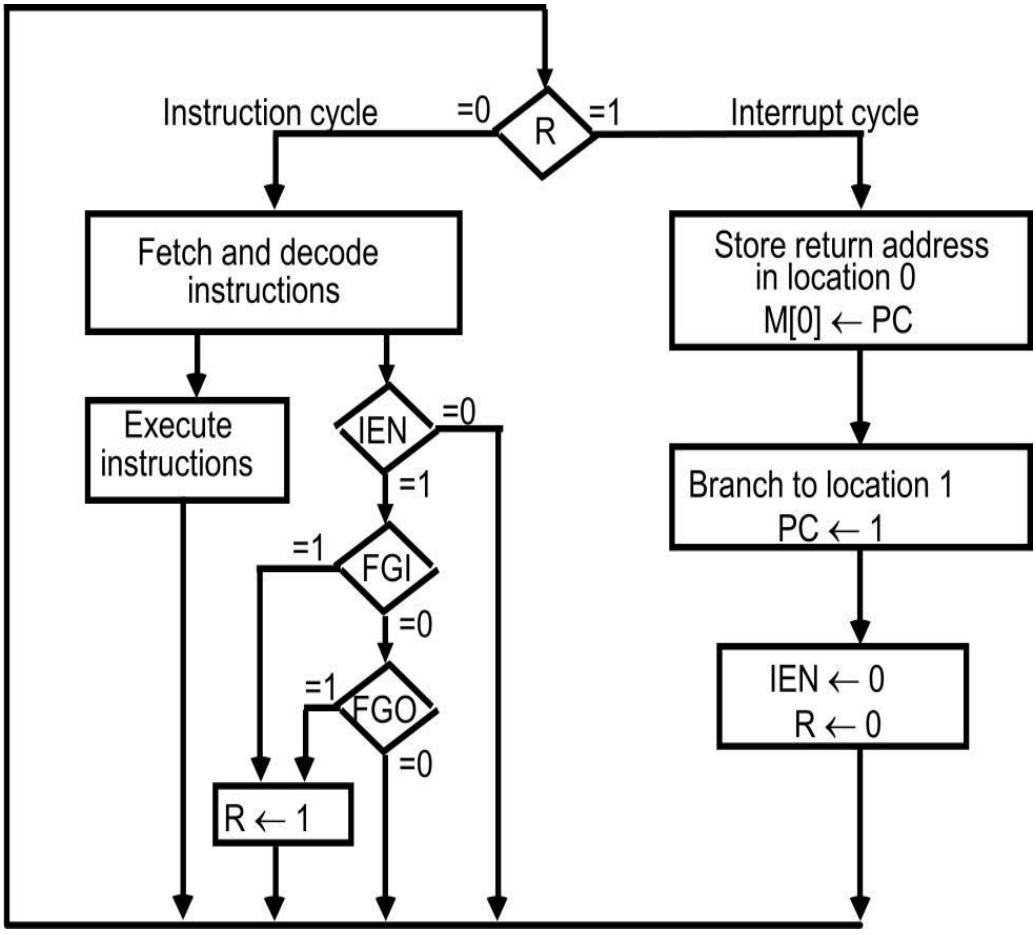
The control functions and microoperations for the input-output instructions are listed below.

INP	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	IEN ← 1	Interrupt enable on
IOF	IEN ← 0	Interrupt enable off

Interrupt cycle:

The way that the interrupt is handled by the computer can be explained by means of the flowchart shown in figure.

- 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.



Instruction Formats:

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 registers.
3. A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

1. General register organization.
2. Single accumulator organization.
3. Stack organization.

1. General register organization:

a. Three-Address Instructions-

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

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

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

b. Two-Address Instructions-

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

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

2. Single accumulator organization:

One-Address Instructions-

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X = (A + B) * (C + D) is

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

3. Stack Organization:

Zero-Address Instructions-

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

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

Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation will be executed on some data which is stored in computer registers or the main memory. The way any operand is selected during the program execution is dependent on the addressing mode of the instruction. The purpose of using addressing modes is as follows:

1. To give the programming versatility to the user.
2. To reduce the number of bits in addressing field of instruction.

Types of Addressing Modes

Below we have discussed different types of addressing modes one by one:

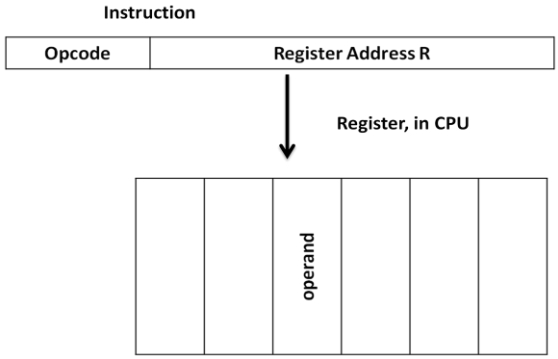
a. Immediate Mode

- In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

- **For Example:** ADD 7, which says Add 7 to contents of accumulator. 7 is the operand here.

b. Register Mode

- In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.
- **For Example:** MOV A, C (Move content of C Register to A register)



Advantages

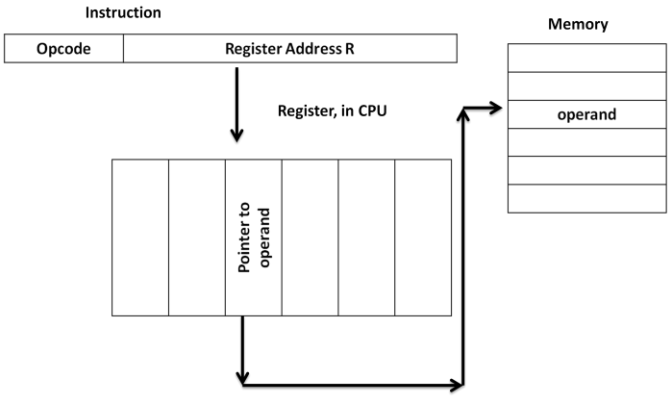
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

c. Register Indirect Mode

- In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.
- For Example: MOV A, [C] (Move the contents of memory location S addressed by C to Register A



d. Auto Increment/Decrement Mode

- In this the register is incremented or decremented after or before its value is used.
- **For Example:** Add R1, (R2)+

  R1 = R1 + M [R2]

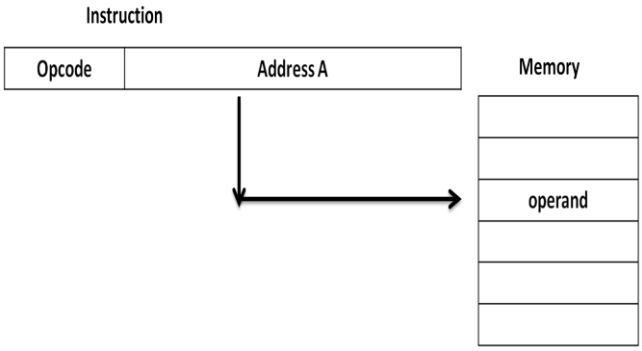
  R2 = R2 + d

- **For Example:** Add R1, -(R2)
  R2 = R2 - d
  R1 = R1 + M [R2]

e. Direct Addressing Mode

In this mode, effective address of operand is present in instruction itself.

- Single memory reference to access data.
- No additional calculations to find the effective address of the operand.
- **For Example:** Add A, [0030] (Add the Content of offset address 0030 to A)



f. Relative Addressing Mode

- It is a version of Displacement addressing mode.
- In this the contents of PC (Program Counter) is added to address part of instruction to obtain the effective address.
- EA = A + (PC), where EA is effective address and PC is program counter.
- The operand is A cells away from the current cell(the one pointed to by PC)

g. Base Register Addressing Mode

- It is again a version of Displacement addressing mode. This can be defined as
- EA = A + (R), where A is displacement and R holds pointer to base address.

h. Stack Addressing Mode

- In this mode, operand is at the top of the stack. For example: ADD, this instruction will *POP* top two items from the stack, add them, and will then *PUSH* the result to the top of the stack.

Data Transfer and Manipulation:

Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks. Most computer instructions can be classified into three categories:

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

1. Data transfer instructions-

Data transfer instructions move data from one place in the computer to another without changing the data content. The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers them.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

2. Data manipulation instructions-

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Arithmetic instructions:

Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and bit manipulation instructions:

Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift instructions-

Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

3. Program control instructions-

A program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered. In other words, program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data-processing operations.

The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution. This is an important feature in digital computers, as it provides control over the flow of program execution and a capability for branching to different program segments.

Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

RISC and CISC:

- As digital hardware became cheaper with the advent of integrated circuits, computer instructions tended to increase both in number and complexity.
- Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions.
- These computers also employ a variety of data types and a large number of addressing modes.
- The trend into computer hardware complexity was influenced by various factors, such as upgrading existing models to provide more customer applications, adding instructions that facilitate the translation from high-level language into machine language programs, and striving to develop machines that move functions from software implementation into hardware implementation.
- A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.

- In the early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often.
- This type of computer is classified as a reduced instruction set computer or RISC.

CISC Characteristics:

- A large number of instructions typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes typically from 5 to 20 different modes.
- Variable-length instruction formats.
- Instructions that manipulate operands in memory.

RISC Characteristics:

- Relatively few instructions.
- Relatively few addressing modes.
- Memory access limited to load and store instructions.
- All operations done within the registers of the CPU.
- Fixed-length, easily decoded instruction format.
- Single-cycle instruction execution.
- Hardwired rather than microprogrammed control.
- Compiler support for efficient translation of high-level language programs into machine language programs.
- A relatively large number of registers in the processor unit.
- Use of overlapped register windows to speed-up procedure call and return.
- Efficient instruction pipeline.