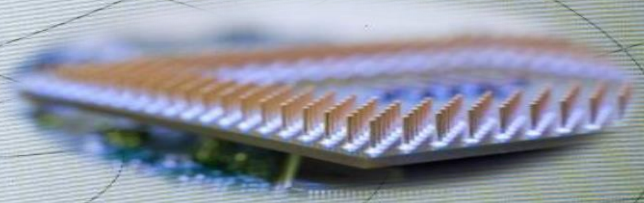
Computer Architecture and Organization

Chapter: PROCESSOR DESIGN



Prof. Ram Sarkar
Jadavpur University, Kolkata
West Bengal, INDIA

Compiled by
MCA (2016-2019), CSE Dep.
Jadavpur University

Processor Design – Hayes

CPU-> executes sequence of instructions stored in a memory which is external to CPU.

-The seq. of operations constitute instruction cycle.

- it has two parts.

i) **Fetch cycle**: instruction is obtained from Main Memory.

ii)**Execution cycle:** decodes the instruction, fetches any required operands, and then performs the operation specified by the opcode.

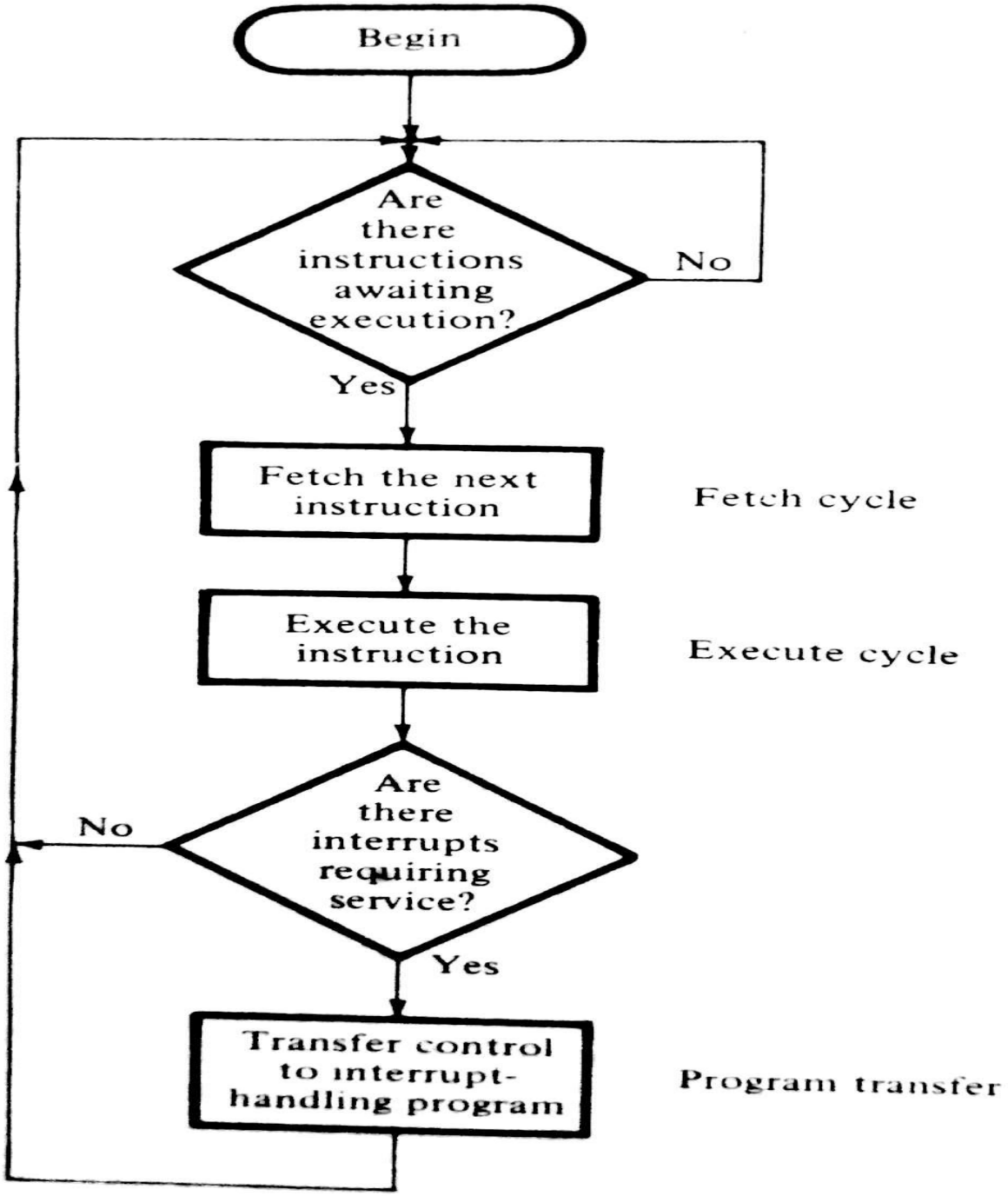
-sequence of micro operations, involves a register transfer operation.

CPU cycle time : (tcpu) time required for shortest well-defined CPU micro operation.

Clock rate : reciprocal of tcpu (in MHz)

-CPU supervises the other system components via special control lines

-Infrequent events are handled by interrupts .



: overview of CPU

CPU designs have following two permises:-

-it should be as fast as technology permits; number of components in the CPU must be kept relatively small.

-Main Memory (MM) of relatively large capacity is needed to store programs and data.

Memory cycle time(tm): time elapses between two successive read or write operations.

tm/ tcpu: 1-10

-CPU contains a small no. of storage device: **registers**

-Instructions whose operands are in fast CPU registers can be executed more rapidly.

Program execution:-

1) Transfer the required operands from MM to CPU registers.
2) Compute the results
3)Transfer the results from CPU to MM

CPU communicates with I/O devices through addressable registers called I/O ports.

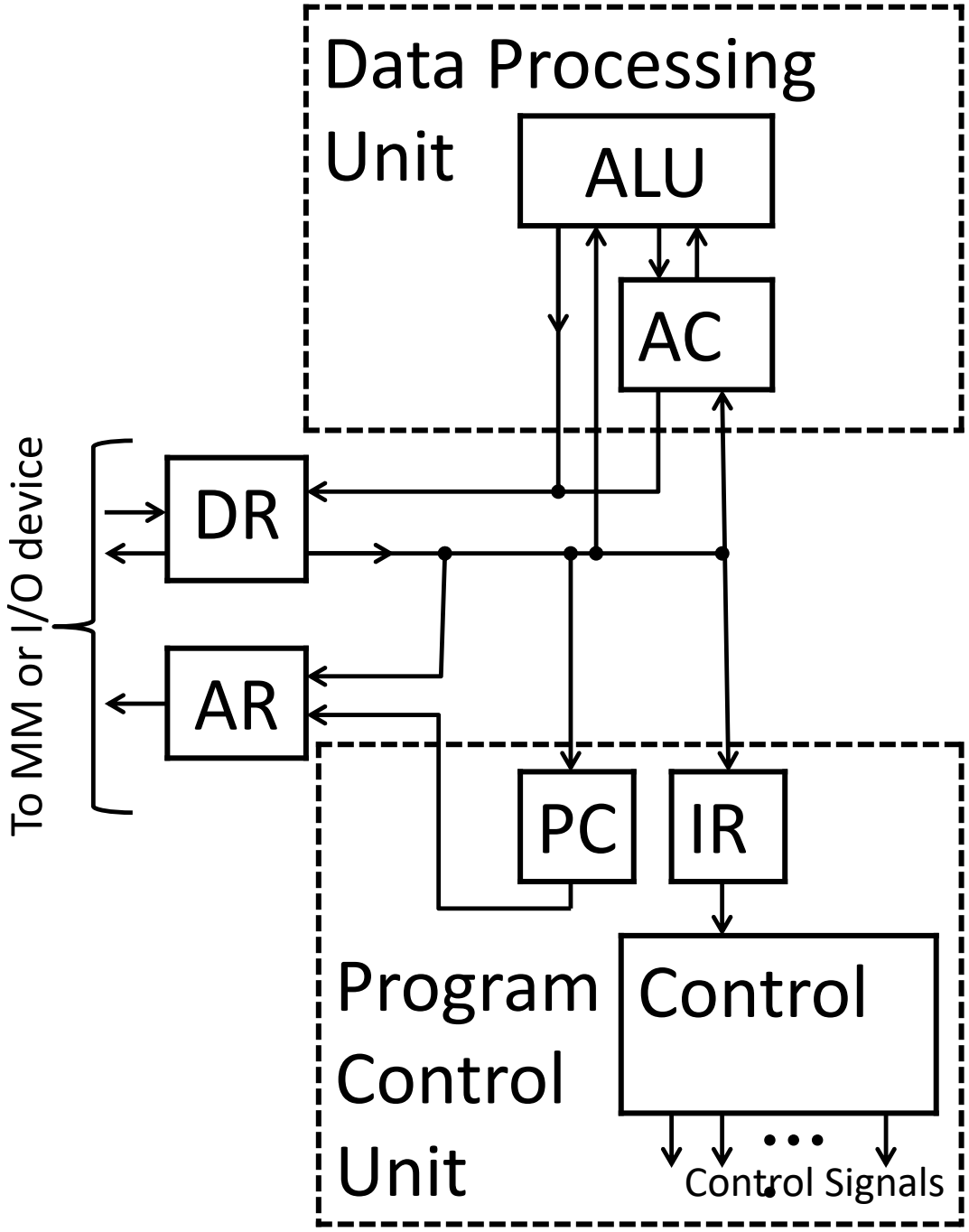
Memory mapped I/O:- No I/O instruction.

- All I/O data transfers are implemented by memory referencing instructions.
- Memory locations and I/O ports share the same address space

I/O Mapped or port- addressed I/O:-

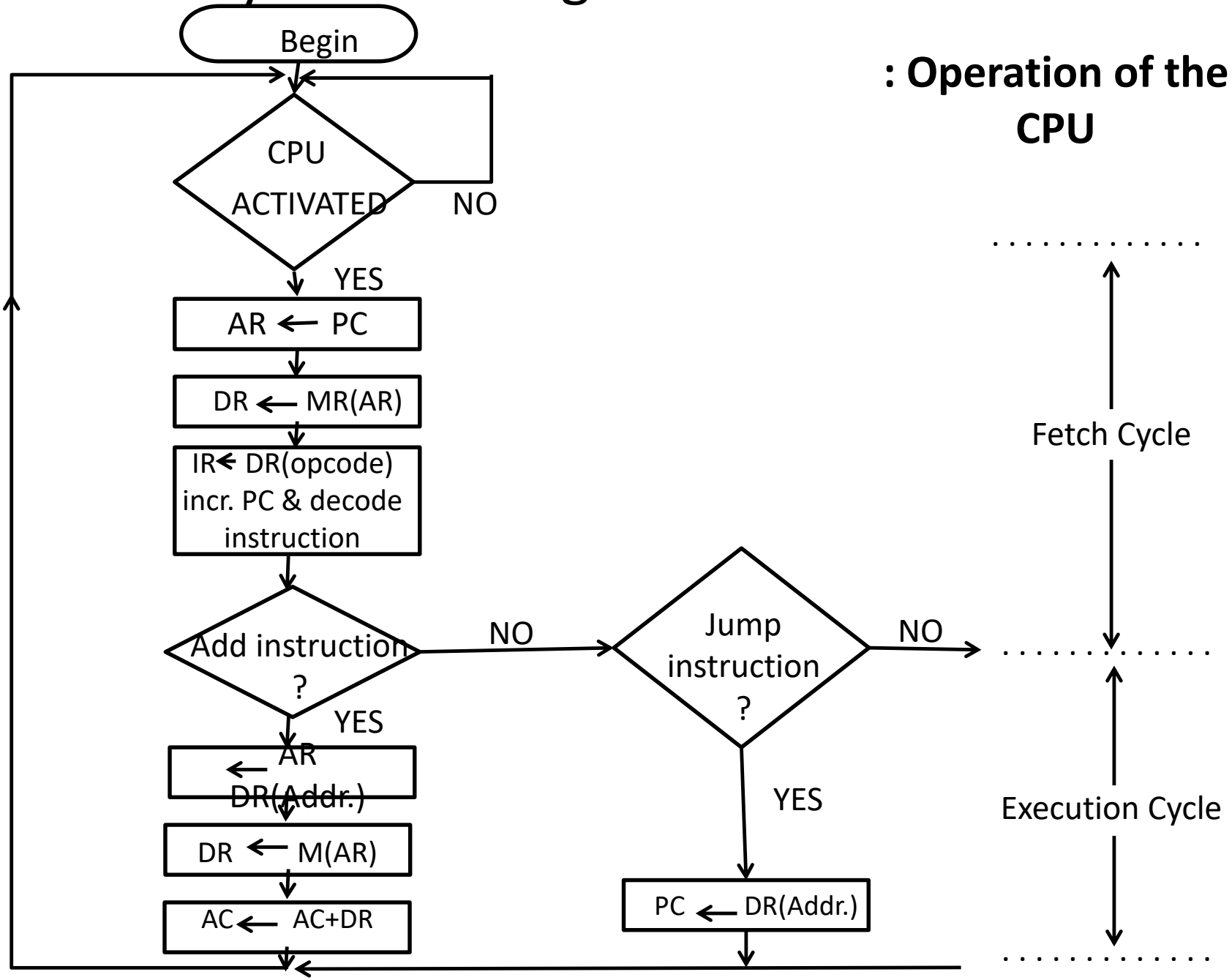
-have I/O instructions that are distinct from memory-referencing instructions.

Accumulator:- CPU register; plays a central role; being used to store an input or output operand in the execution of most instructions.



AC : Main operand register of ALU.
Data Register(DR) : Acts as a buffer between the CPU and MM or input operand register.
Program Counter (PC): Stores the address of the next instruction.
Instruction Register (IR): Holds the opcode of the current instruction.
: <u>A Simple Accumulator based CPU</u>

AR : Memory Address Register



: Operation of the CPU

Ways to improve Processor Organization :

1. Additional Addressable registers can be provided for storing operands and addresses.
   - General Register Organization: registers are for multipurpose, called Register file or Scratch-Pad memory.
   - Also have special address registers such as Base registers.
2. Capabilities of ALU can be extended.
   - Add extra circuiting for multiplication/division.

- Include floating-point operations.

3. Special Register can be included to facilitate the transfer of Control between instructions within a program.

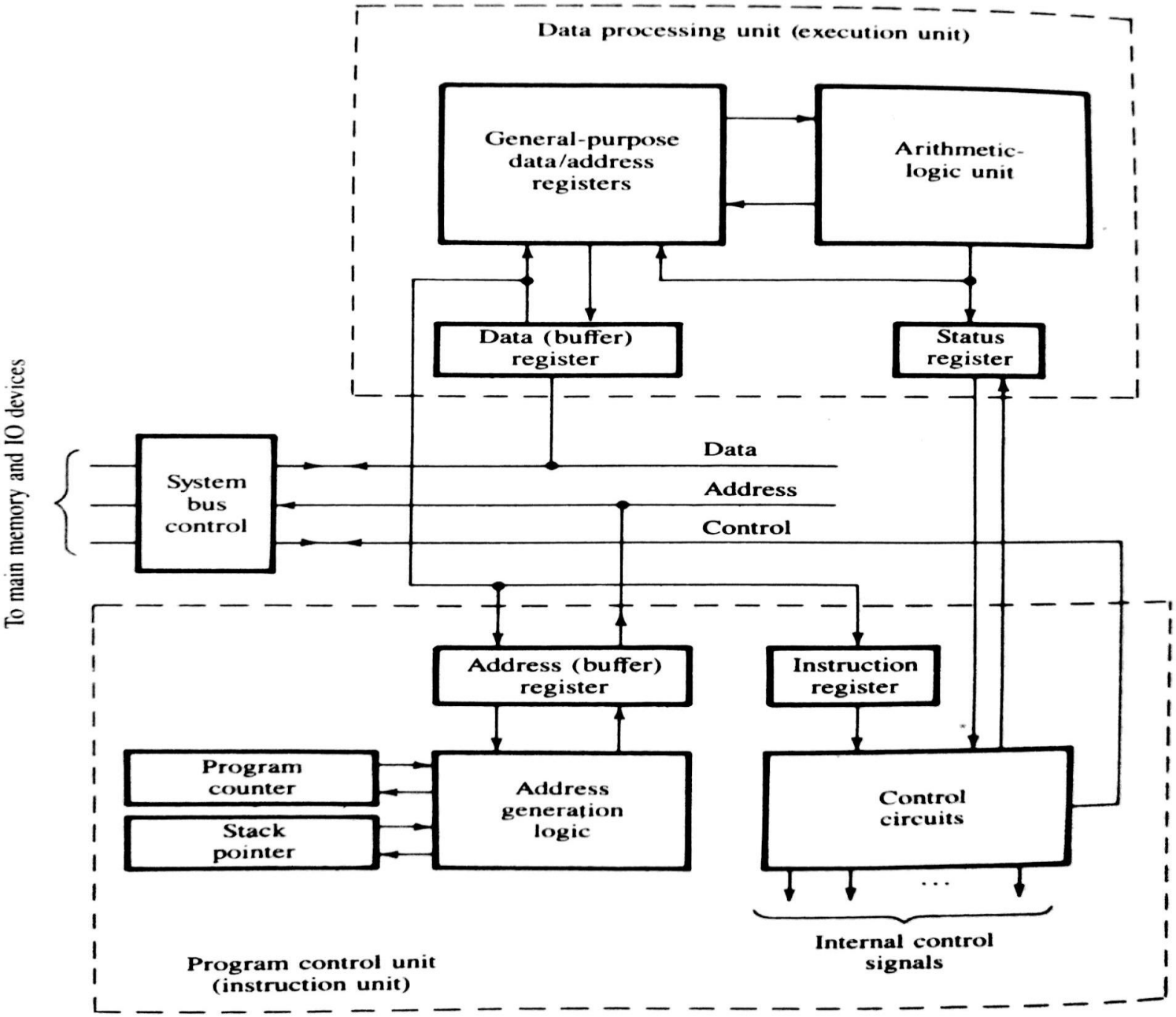
   e.g. Status register is used to indicate conditions resulting from the execution of the previous instruction.

4. Transfer of control between the different subprograms due to interrupts or subroutine calls & return is also facilitated by Special Registers.

- Control is transferred by saving the current PSW (Program Status Word) in a designed location in MM & loading a new PSW into the CPU.
- Control is returned to first program by retrieving the previously saved PSW from memory & restoring it to the CPU.
- Uses stack (Stack Pointer register: SP)
  - Advantage of stack: handled repeated program transfers

5. Facilities can be provided for the simultaneous processing of two or more distinct instructions.

- Can be fetched simultaneous by extending the memory addressing circuit & adding sufficient buffer storage to CPU.
- Execution of several instructions can also be overlapped.



: Typical CPU with general register organization

-ALU stores/obtains most of its results/operands into/from this register set.
-Special logic is included for address computation.
- control circuit derives the input from the instruction register, which stores the opcode of the current instruction and the status register.
- communication with outside world is via a system bus that is used for the transmission of address, data and control information to & from the CPU.

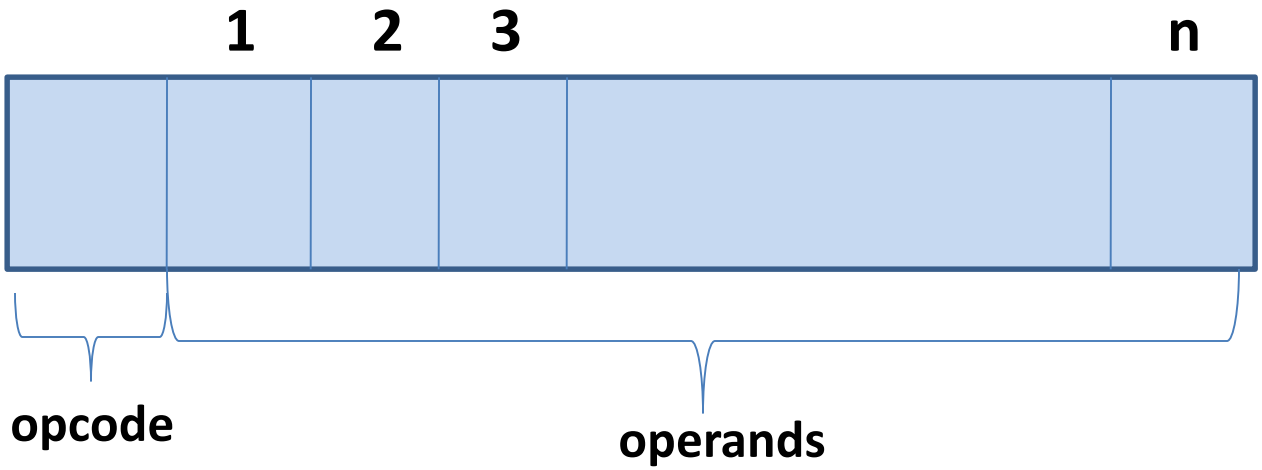
3.2: Instruction sets:-

-The purpose of an instruction is to specify an operation to be carried out and the set of operands or data to be used.

-The operation is specified by a field called the opcode.

- The operand fields contain the addresses of storage locations in main memory or in the processor.

Most instructions specify a register transfer operation of the form $X_1 \leftarrow f(X_1, X_2, \ldots, X_n)$, which involves n operands.



-To reduce the instruction size, specify m<n operands explicitly in the instruction, the remaining operands are implicit.

- explicit address : Main memory

  Implicit address: Register

  -It is called *m-address machine*

  - Implicit input operands must be placed in locations known to CPU before the instruction that refers to them is executed.

Addressing Modes :-

-Operand associates with data X.

- for execution CPU needs the current value of x.

- this value can be specified in several ways: addressing modes

- if the X is constant, then its value can be placed in the operand field; X is called immediate operand and the mode of operand specification is called immediate addressing.

- quantity of interest is a variable
- corresponding operand field contains the address X of the storage location containing the required value .
- operand value can be varied without modifying any instruction address.

- **Direct Addressing**

- It is frequently useful to change the location (as opposed to the value) of X without changing the address fields of any instructions that refer to X.

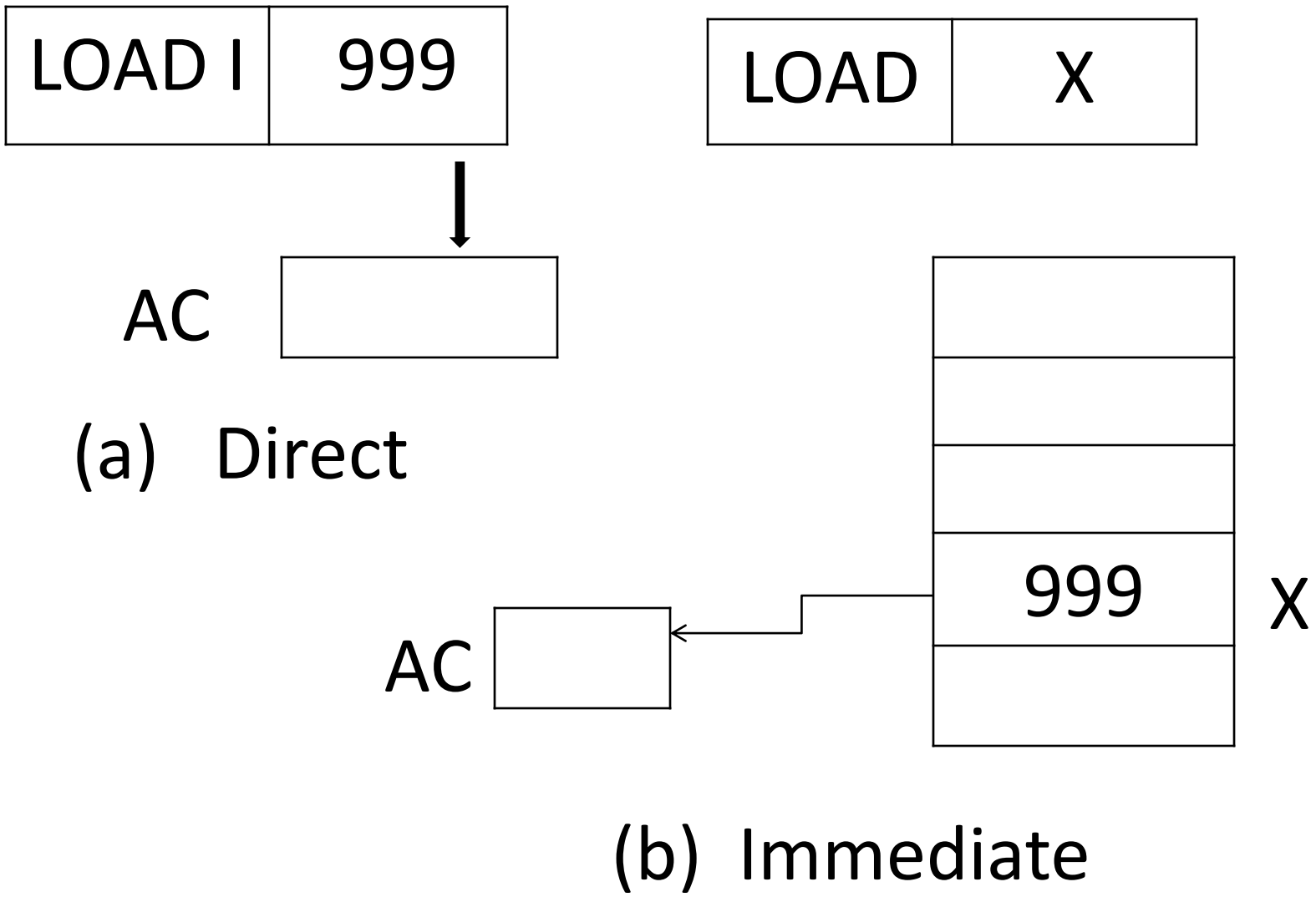
- **Indirect Addressing**

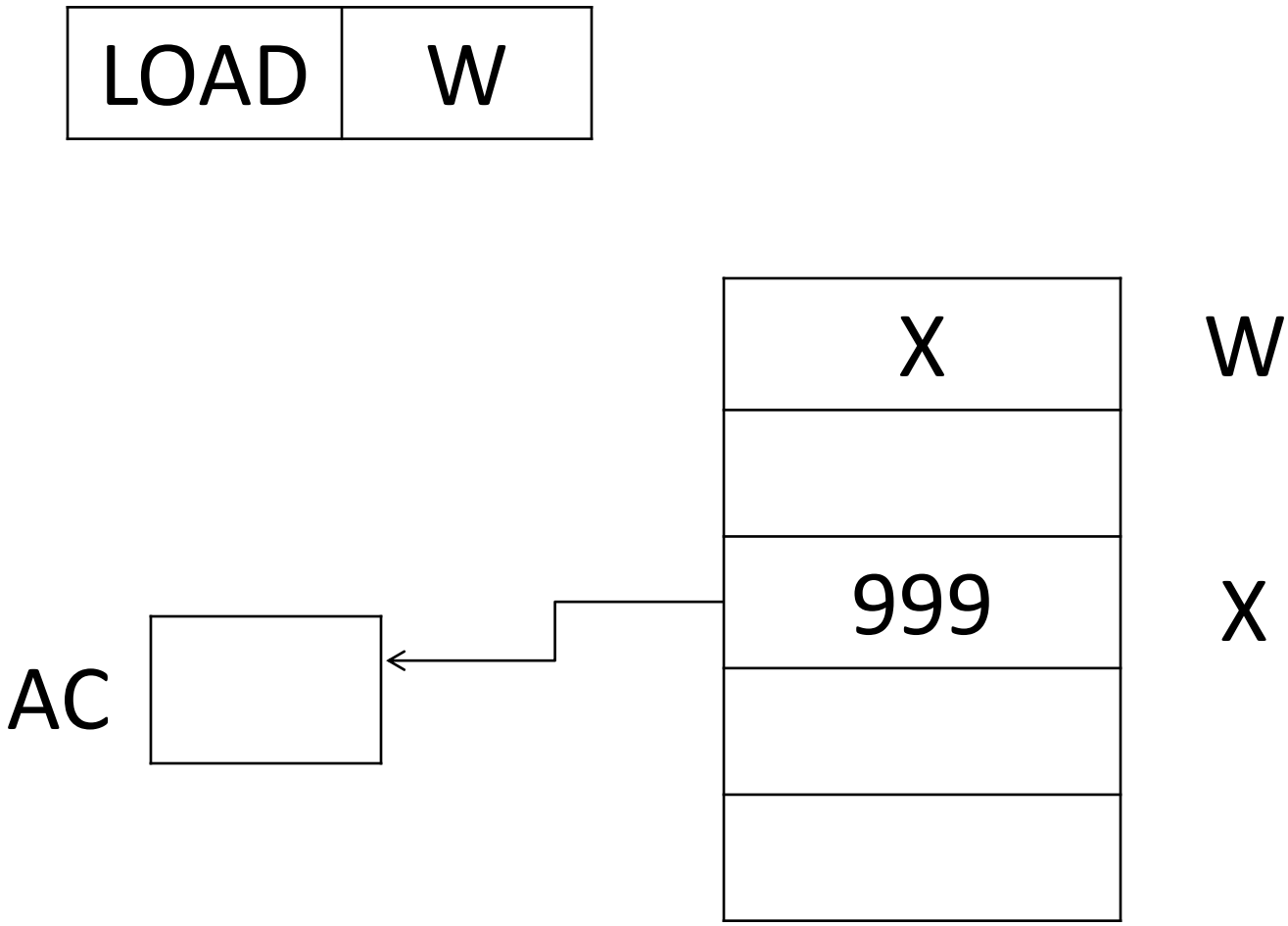
- Direct Addressing requires one fetch operation to obtain an operand value , while indirect addressing requires two.

MOV A , B (A<-B): Direct Addressing (register-to-register transfer)

MVI A , 99 (A<- 99): Immediate operand

- uses both D. & I. Addressing modes


LOAD I
999
AC
(a) Direct
LOAD
X
999
X
AC
(b) Immediate



: Indirect

Orthogonality:- The ability to use all relevant addressing modes in a uniform and consistent way with all opcodes of an instruction set.

- Simplifies programming both by reducing the number of opcodes needed & by simplifying the rules for operand address specification.

- Most computers have little orthogonality since processor cost can be reduced by restricting instructions to a few frequently used addressing modes that vary from instruction to instruction.

Absolute Addressing :- Simplest mode of (direct) address formation.

- requires complex operand address to appear in the instruction operand field.

- address is used without further modification to access the desired data item.

Relative Addressing:-

- partial addressing information is included in the instruction.
- complete address must be completed by CPU.
- operand field contains a relative address of displacement D.

- instruction also identifies storage locations $R_1, R_2, \ldots$
- effective address A is $f(D, R_1, R_2, \ldots, R_k)$

In general, A = R + D , R may be PC.

Advantages:

1) Since all the address information need not be included, instruction length is reduced.

2) By changing the contents of R, CPU can change the absolute address referred to by a block of instruction B.
   - Relocation of entire region becomes easier
   - R acts as a base register. (content is called base address)

3. Facilitates the process of indexed data

- R is called the INDEX REGISTER.

-Indexed terms X(0), X(1),…,X(n) are stored in the consecutive addresses of the Main Memory.

-D contains the Address of X(0)

R contains the Index ‘i’.

-X(i)=D+R; is by changing the contents of R , a single instruction can be made to refer to any item X(i) in the list.

Limitations:-

Extra logic circuits and extra processing time required for address computation.
-Indexed items are frequently accessed sequentially, a process called ‘AUTOINDEXING’ is required which automatically increments or decrements an address.

-(A) and (A)+: First one indicates that contents of A should be decremented automatically before the instruction is executed, whereas the latter one increments automatically after the current instruction has been executed.

-Autoindexing facilitates another addressing mode-

-Stack addressing: Part of Main Memory

-Two options:

push- store a new item at the top of the stack

pop- removes the item stored at the top

-controlled by the register called STACK POINTER.

-automatically adjusted by the push/pop operation.

- Stack grows towards the low address.

NUMBER OF ADDRESSES:-

---How many explicit operand addresses to include in instructions?

---Fewer the address, the shorter the instruction.

---Fewer addresses means more primitive instructions and longer program required to perform any given task.

---while the storage requirements of shorter instructions and longer programs tend to balance, larger programs require longer execution time.

---But long instructions with multiple addresses require more complex decoding and processing circuits.

ADD Z, X, Y (add X, Y and store result in Z)

ADD X (AC ← AC + X)

ADD X, Y (AC ← X+Y)

EXAMPLE- X=A*B+C*C

One Address:-

- LOAD A
- MULTIPLY B
- STORE T
- LOAD C
- MULTIPLY C
- ADD T
- STORE X

- Transfer A into accumulator AC
- AC ← AC + B
- Transfer Accumulator to memory location T
- AC ← C
- AC ← AC*C
- AC ← AC + T
- Transfer result to memory location X

Two Address:-

- MOVE T, A
- MULTIPLY T, B
- MOVE X, C
- MULTIPLY X, C
- ADD X, T

- T ← A
- T ← T*B
- X ← C
- X ← X*C
- X ← X+T

Three Address:-

- MULTIPLY T, A, B
- MULTIPLY X, C, C
- ADD X, X, T

- T ← A*B
- X ← C*C
- X ← X+T

- Some computers have most instructions contain no explicit addresses, called Zero address machine.
  ---Store operands in a push-down stack.

  ADD: Causes the top two operands X and Y to be removed from the stack and added. The sum is placed at the top of the stack.

  PUSH A -> Transfer A to top of stack.
  PUSH B -> Transfer B to top of stack.

MULTIPLY Y -> Remove A,B from stack and replace by A x B.

PUSH C -> Transfer C to top of stack.

PUSH C -> Transfer C to top of stack.

MULTIPLY Y -> Remove C,C and replace by C x C

ADD -> Remove C x C and A x B and replace by their sum.

POP X -> Transfer result from to of stack to X.

Polish Notation (Polish logician Jan Lukasiewicz)

- Postfix : X * Y => X Y * (suffix or reverse polish notation.)
- Prefix : X * Y => *X Y

A x B + C x C => AB x CC x + (suffix)

Advantage : No requirement of parentheses.

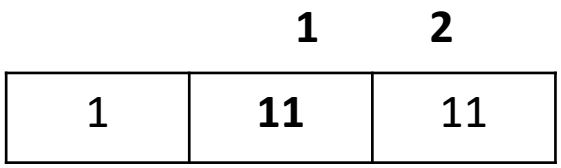
X (Y + Z) => X Y Z + X

: OPCODE :

A fixed length field of k bits within each instruction permitting up to 2k distinct operations to be specified.

k=8/16 (if k=16, # of pattern 65, 536)

- Some addressing information may also be included in opcode, e.g. addresses of processor registers used to store operands.
- Number of registers are small, so only a few bits are required.



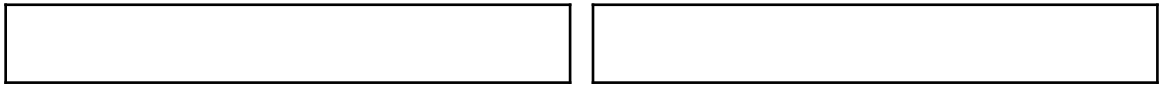
Opcode byte

1. Destination Register address
2. Source Register address.

Opcode byte

Data (immediate operand) or address

Opcode byte



Data (immediate operand) or address

- Intel 8085 microprocessor instruction format.
(1st byte is always opcode.)

Example

ADD r , (1 byte instruction)

i.e. A <- A + r , r is any 8 bit register

ADI d , d is a 8 bit word , 2 bytes long

JMP address,

i.e. PC <- address, is a 3 byte branch instruction.

- Some programs occupy a considerable amount of storage space , it is desirable to reduce their length as much as possible.
- Assign shortest format to the most frequently used instruction, and the longest format to the least frequently used.
- The frequency with which operand addresses occur cannot be reasonably determined; all may be assumed to have equal probability.

- The frequency with which specific instruction types occur can be determined.
- One can therefore attempt to base opcode lengths on their probability of occurrence.

Instruction Types :-

- What type of instructions should be included in a general purpose processor's instruction set ?
- A typical machine instruction defines one or two register transfer (micro) operations, and a sequence of such instructions is needed to implement a statement in a high level programming language.
- Due to complexity of operation, data type and syntax H.L.P.L., few successful attempts have been made to construct computers whose machine language directly corresponds to a H.L.P.L. .
- Thus there is a semantic gap between H.L.P.L. and M.I. that implements it, a gap that compiler must bridge.

Requirements:-

1) **Complete**:- One should be able to construct a machine language program to evaluate any function that is computable using a reasonable amount of memory space.
2) **Efficiency**:- Frequently required functions can be performed rapidly using relatively few instructions.
3) **Regular**:- Instruction set should contain expected opcodes and addressing modes (e.g. left shift or right shift). The instruction set should also be reasonably orthogonal with respect to the addressing modes.
4) **Compatible**:- To reduce both h/w and s/w design costs, the instructions may be required to be compatible with those of existing machines.

Completeness:-

A function f(x) is defined to be computable if it can be evaluated in a finite number of steps by a Turing machine.

But real computer have finite amount of memory, and can be used to any computable function, at least to a reasonable degree of approximation.

- Turning machine has only four basic operations.
- While simple instruction sets require simple, therefore inexpensive, logic circuits to implement them, they can load to excessively complex programs.

Instructions are divided into five major types :-

1) Logical Instructions : Include Boolean and other non-numerical operation.
2) Program Control Instructions : Such as branch instructions, which change the sequence in which program executes.
3) Arithmetic Instructions : Perform operations on numerical data.
4) Data-transfer Instructions : Cause information to be copied from one location to another.

5) Input-Output (IO) Instructions: Cause information to be transferred between the processor or its MM & external I/O devices.

These are not always mutually exclusive.

e.g. Arithmetic operation A <- B+C can be used to implement simple data transfer A <- B by setting C = 0.

RISC Vs CISC :-

- No general agreement about what constitutes the appropriate size of a general-purpose instruction set.
- Early computers had small & simple instruction sets, forced by the need to minimize the amount of h/w.
- As h/w became cheaper, instruction set tends to increase both in number and complexity.
- Contains hard-to-program operations, line floating-point division.
- Reduce the Semantic gap.
- Increase the complexity in designing h/w & s/w.

Suppose that a particular option F can be implemented either by a single complex instruction I_F , or by a

multi-instructions routine P_F composed of single instructions.

- Execution of P_F will generally be slower than that of I_F due to the fact that more time must be spent fetching instructions of P_F
- It occupies more memory space than I_F .
- But the disadvantage of I_F is that it adds complexity of a processor's control unit , thereby both the size of the CPU & the time required to design it.

- A compiler translates F into I_F , if available, which uses fixed CPU registers & has a fixed execution time.
  - Otherwise optimization is employed
  - Generates object code Q_F corresponds to P_F that exploits information known as compilation time to reduce the execution time for F.

e.g. for multiplication(fixed point) :

- if one operand is small constant or zero, the compiler generates shorter term of P_F which executes faster than generic n-step multiply instruction I_F

- The speed gap between I_F & P_F can be narrowed by designing the small I. Set required for P_F to reduce the fetch & execution cycles times.

- Another advantages of P_F over J_F is that it can be interrupted in the middle of every option, whereas J_F must proceed to termination before CPU respond to an interrupt.

Reduced Instructions Set Computer(RISC):

Contains small & relatively simple J.sets.

1) Few instruction types &addressing nodes.
2) Fixed &easily decoded instruction formats.
3) Fast single-cycle instruction execution.

4) Hardwired rather than micro programmed control.

5) Memory access limited mainly to load &store options.

6) Use of compilers to optimize object code performance.

- Interval of these RISC attributes are closely related.

  E.g. the small size and regularity of the instruction set simplifies the design of a hardwired program control unit , which in turn facilitates the achievement of fast single-cycle execution.

- But efficient compilation requires the architects & compiler writers to cooperate closely in the design process.

- To support register-register transfers, a lager-than-usual number of registers may be placed in CPU.
- Likely to have more code in a program than the corresponding CISC code.
- Likely to have more code in a program than the corresponding CISC code.
- But CISCs outperform RISCs in scientific computations requiring lots of floating-point arithmetic.

IMPLEMENTATION

The design of circuits to implement an J.set

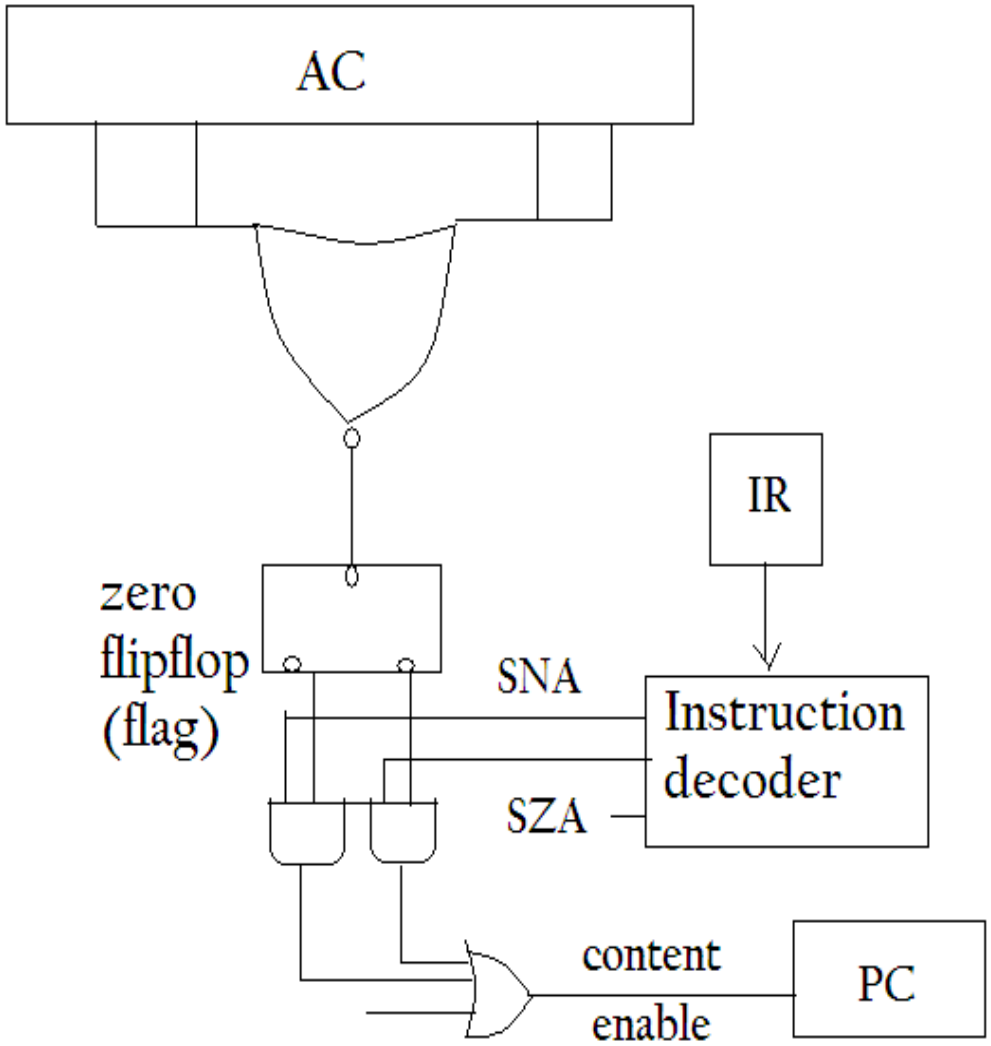
Is an exercise in register-level logic design.

-Necessary to devise an algo for appropriate register transfers or other micro operations.

-Particular component technology used to implement them.

-Also should balance circuit cost & execution speed.

-common to choose alogos that permit cks used by different instructions to be stored.

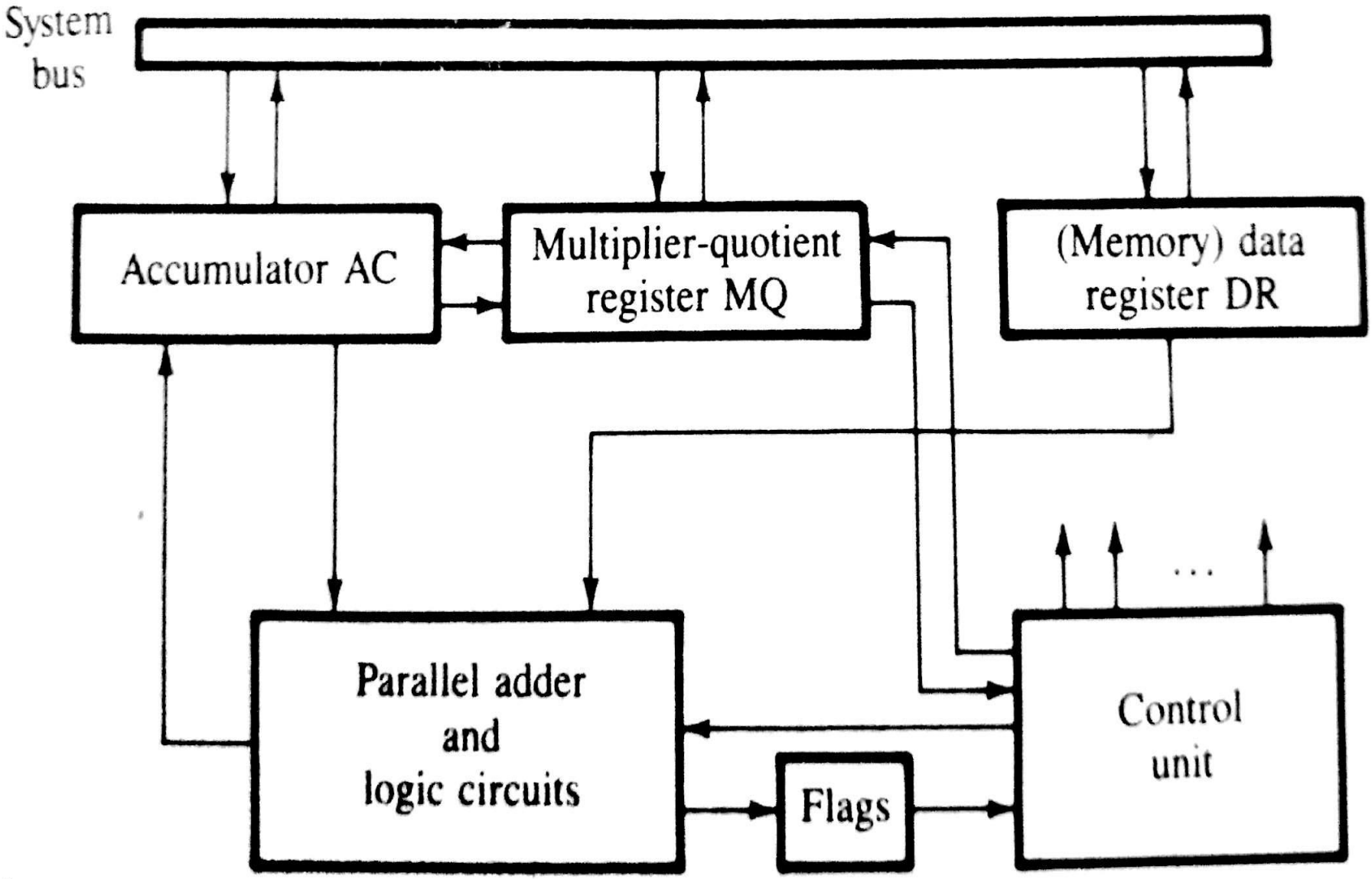

AC
IR
zero
flipflop
(flag)
SNA
Instruction
decoder
SZA
content
enable
PC

: Implementation of two conditional branch instructions SZA (skip on zero accumulator) SNA (skip on non zero accumulator)

ALU designs:

Various circuits used in the execution of data processing instructions by a processor are usually merged into a single unit called Arithmetic-Logic Unit (ALU).

- fixed-point / floating point or both
- Can have auxiliary unit called arithmetic co-processor.



: Structure of a basic fixed point ALU

- Multiplication and Division is done by sequential digit-by-digit shift-and-add / subtract algorithm.
- Three one-word registers: AC, MQ & DR
- **AC.MQ**: Capable of left and right shifting (acts as a single register)
- Parallel adder derives its input from AC & DR and places the results in AC.
- DR also stores the multiplicand or divisor.

Usage of the registers:

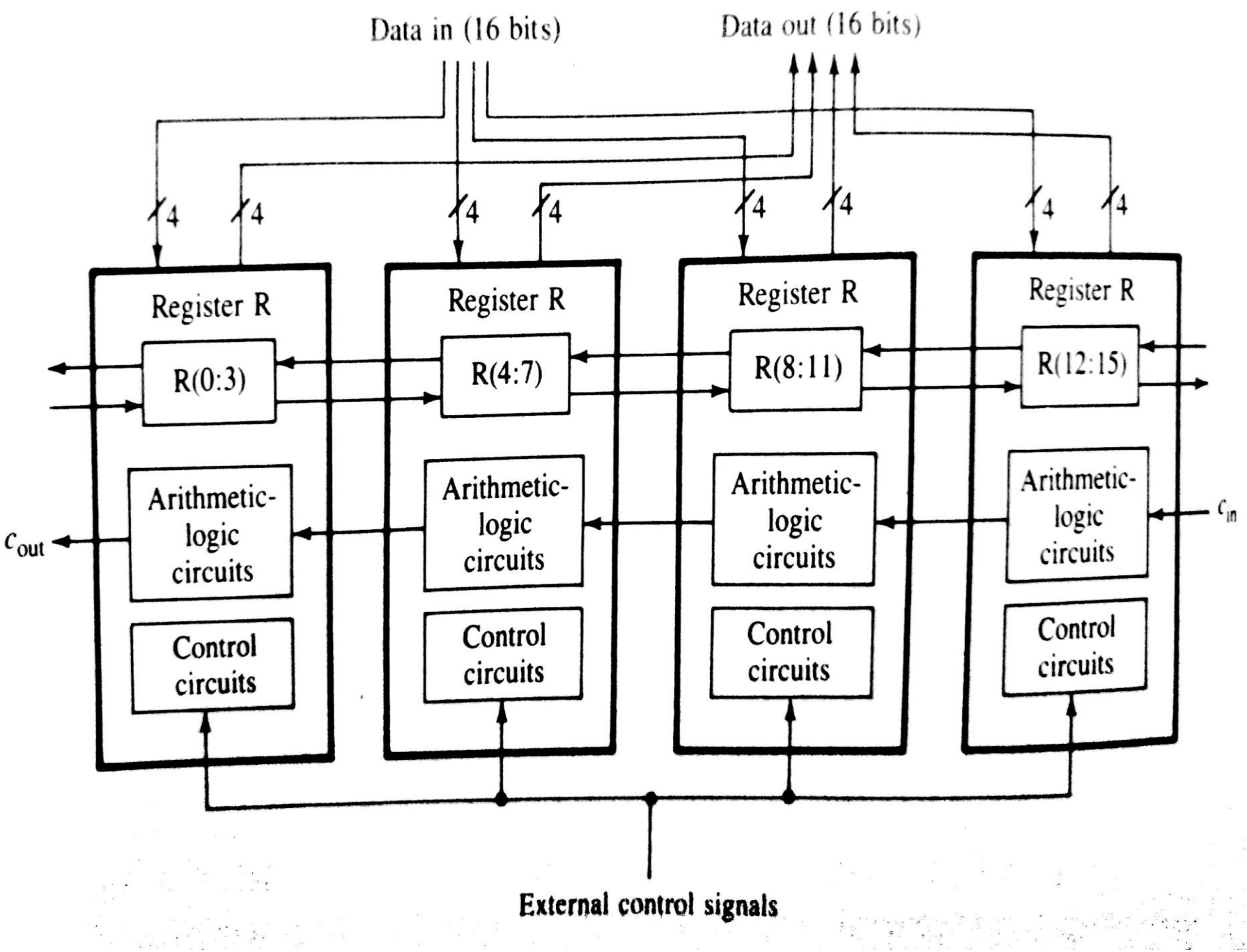
ADD: AC←AC + DR	Mult : AC.MQ ← DR X MQ
SUB: AC←AC – DR	Div : AC.MQ ← MQ ÷DR
AND: AC←AC ^ DR	OR: AC← AC V DR
X-OR: AC←AC⊕ DR	NOT: AC ← $\overline{AC}$

-Something DR serves as a memory buffer register to store data addressed by an instruction address field ADR. Then DR can be replaced by M(ADR), results in a one-address-memory-referencing format.

Bit-Sliced ALU:

- Feasible to design an entire fixed-points ALU on a single-IC chip. (if the word size m is 4/8 bits)
- Can be designed to be expandable in that K copies of the ALU chip can be connected to form a single ALU of K m bit words directly.
- Resulting array like circuit is called bit-sliced, because each component chip processed an independent ‘slice’ of m bits from each Km-bit operand.

Adv: any desired word size can be handled efficiently by selecting the appropriate no. of components (bit slice) to use.



: A 16-bit bit-sliced ALU composed of 4 4-bit slices

- Data buses & Registers of the individual slices are effectively juxtaposed to increase their size from 4 to 16 bits.
- The control lines that select and sequence the operations to be performed are connected to every lice, so that all the slices execute the same actions in step with one another
- Each slice thus performs the same operation on a different slice of the input operands, and produces only the corresponding part of the results.

- Required control signals are derived form and external control unit, which is usually microprogrammed.
- Also performs shift and carry-forward operations (needs exchange of bits)

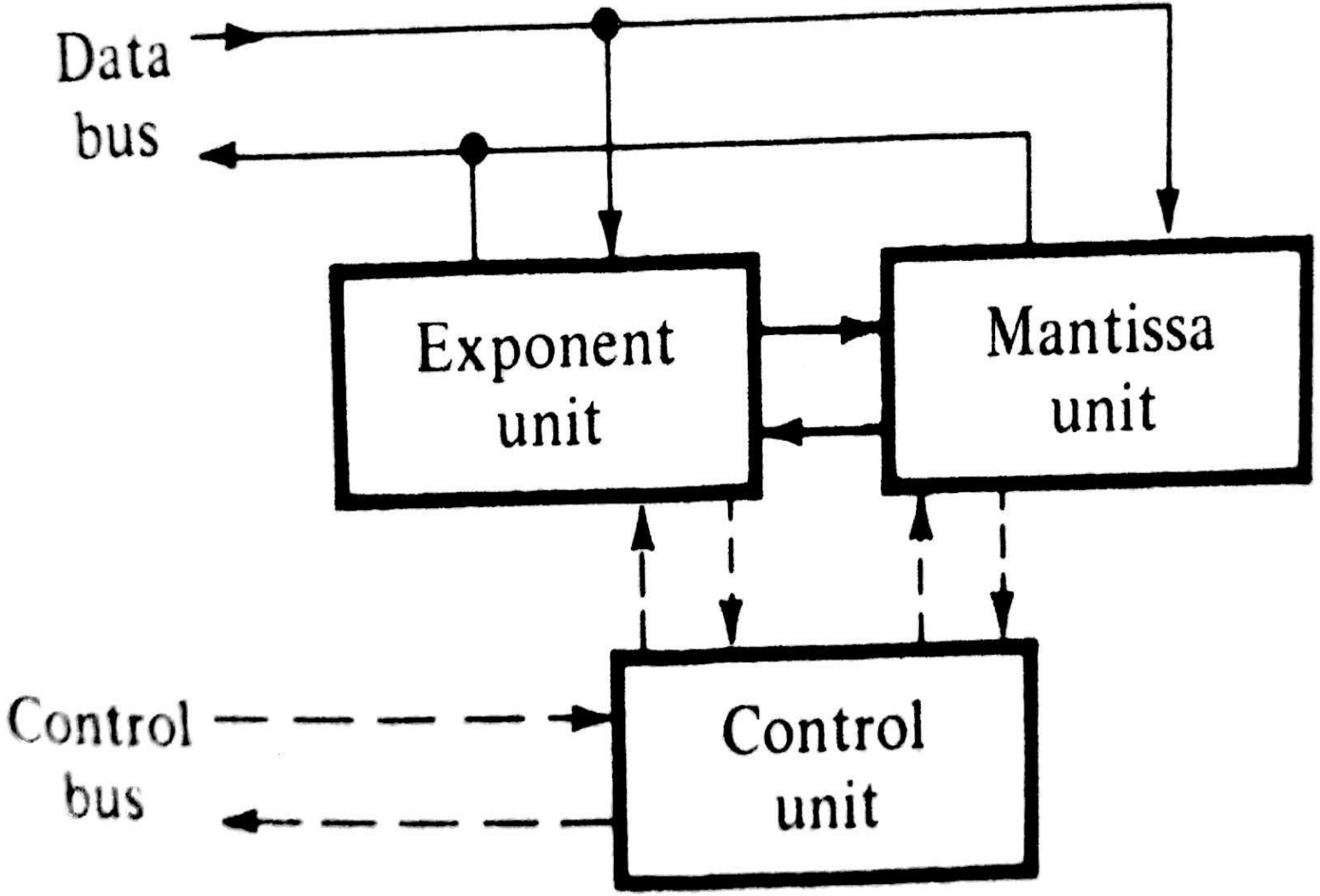
3.4.2 : Floating –Point Arithmetic

$X\ (X_M, X_E)$, i.e. $X = X_M * B^{X_E}$

Assumptions:

1. $X_M = n_M$ – bit binary (2's complement) fraction
2. $X_E = n_E$ – bit integer in excess – $2^{n_E - 1}$ code, implying an exponent bias of $2^{n_E - 1}$
3. $B = 2$
4. All numbers are started in normalized form only

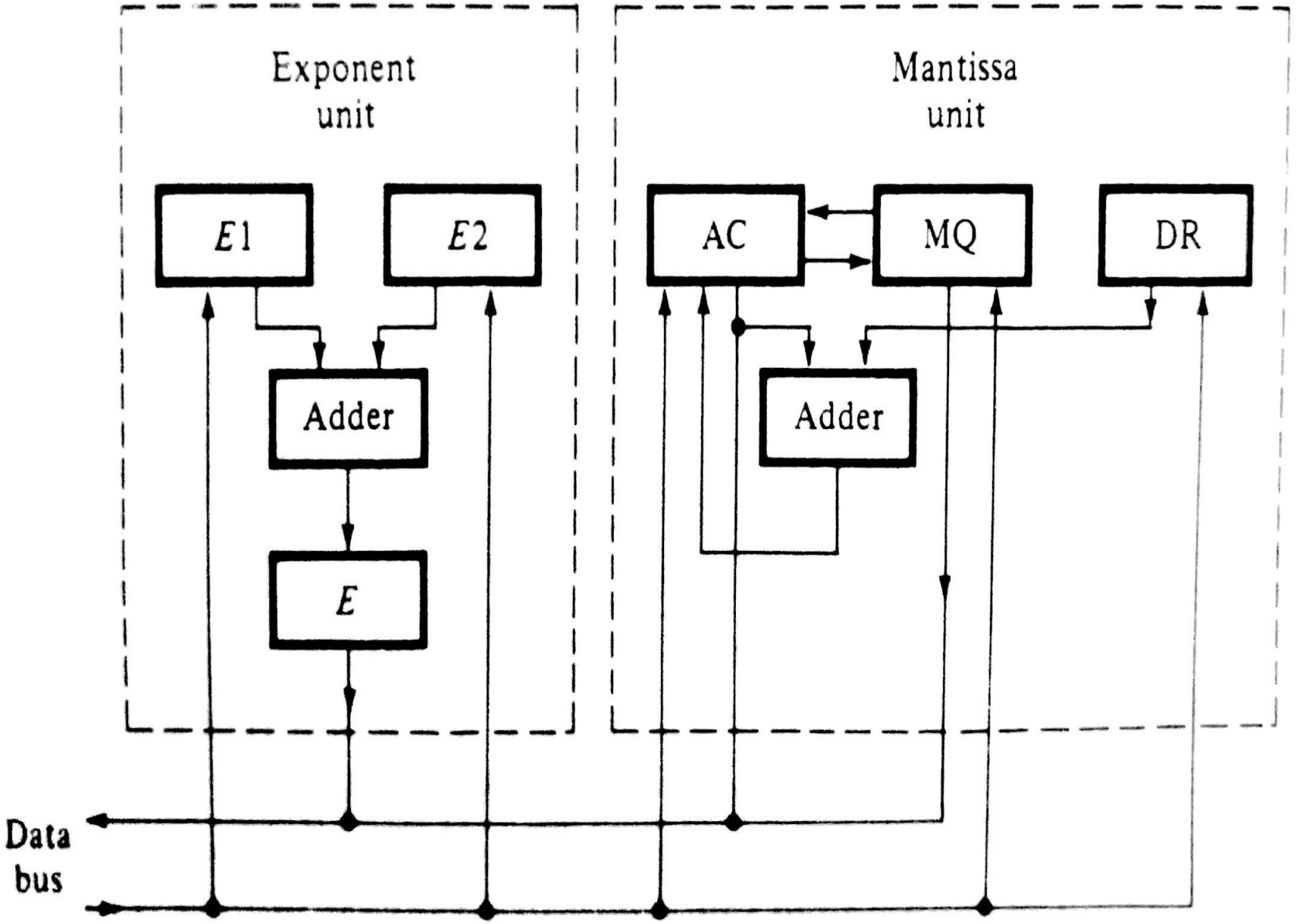
-Can be implemented by two loosely connected fixed-point arithmetic circuits, an exponent unite and a mantissa unit.



: A floating point arithmetic unit

-For performing all four open, general-purpose fixed point ALU can (Mantissa) be used

-**for exponent**: add/sub/compare => simple circuit can do that exponent comparison is performed by comparator.



: Data Processing part of a simple floating point arithmetic unit

- The exponents of inputs are placed in E_1 & E_2, which are connected to parallel adder to perform $E_1 \pm E_2$.
- The result of the exponents comparison is placed in E.

Dis: it can be used for n-bit floating-point operations only, even though it has most of the facilities required for n-bit fixed-point operations. Since all computers with floating-point instructions also have fixed-point instructions, it is often desirable to design a single unit for both types of instructions(a bit difficult circuit).

3.4.3. ARITHMETIC PROCESSORS:

- CPU devotes a considerable amount of control & data processing h/w for non-arithmetic functions. Therefore, h/w costs due to IC count & chip area for implementing complex arithmetic opens often prevent their inclusion in the CPU's instruction set.
- Such CPU must rely on slower s/w routines.

- Alternatively, a processor is devoted exclusively to arithmetic functions, so a full range of numerical operations can be implemented in h/w at relatively low cost e.g. in a single IC
    - It is an auxiliary special-purpose arithmetic processor that are separated from CPU.
- Auxiliary processors speed up program execution by replacing s/w routine with h/w; they can also reduce programming complexity.
    - Can do add/sub/mult/div/exponentiation /logarithms/trigonometric functions

To implement it:

Two ways:

1. It can be treated as a peripheral or I/O device to which the CPU sends data & instructions, & from which it receives results.(Peripheral Processor)

Or,

2. It is closely coupled to the CPU so that its instructions and register set are extensions to those of the CPU. The CPU's instruction set has a special subset of opcodes reserved for the auxiliary processors. (co-processor).

Peripheral processor :-

Such as AMD 95211/12 one-chip floating point processor which have advantage that they can be used with any host CPU.

Dis :- Need for explicitly programmed and relatively slow communication links the host.

It is used as follows :

1) CPU executes a series of data-transfer instructions, e.g. the type of option to be performed, to the registers in the peripheral processor.

2) The PP decodes & executes commands received from the CPU, generating a result , and place it in the register accessible to the CPU.
3) CPU determines the completion of the task by PP by checking the status i.e. by polling a pre-set status register, or else by receiving an interrupt from PP.
4) CPU then obtains the result from PP by executing more data transfer instructions.

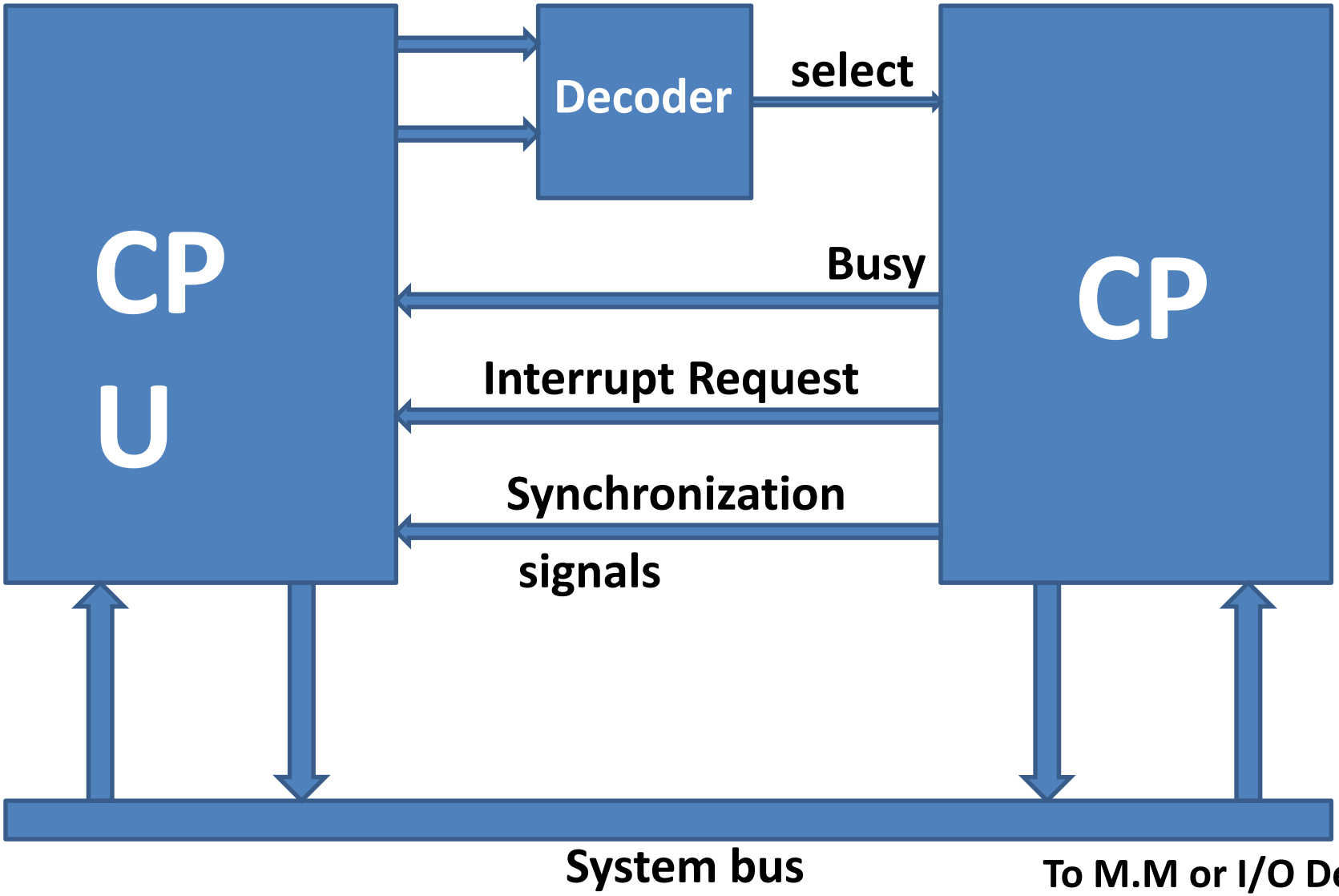
-sometimes, CPU must wait for the result of the PP's computations, in which case CPU can be idle for an extended period.

Coprocessor :- (CP)

- Unlike a PP, a co-processor is tailored to a particular CPU family. Each CPU is designed with a co-processor interface that includes special control circuits linking the CPU with co-processor, and special instructions designed for the execution by the CP.

-Communication is made by h/w , so CP's instruction appear in assembly/ machine level language.

- The CPU knows there is no CP, then it can transfer program control to a specific memory location to get the service done. This CPU generated interruption is termed as trap. So CP approach makes it possible to provide either h/w or s/w support for certain instructions without changing the source code.



: Typical connection between CPU & CP

-CP sits idly unit an instruction from CPU comes.
- links to CPU by some control lines, which allow the CPU & CP to synchronize rapidly.
- Both connect to system Bus , can be used in normal fashion for inter processor data transfers.
- CP can initiate data transfers to & from CPU.
- CP can acts as passive device like M.M

CP Instruction : 3 fields

F_0 = opcode, that identifies CP instruction

F_1 = address of particular CP, If multiple CPs

F_2 = particular command to be performed by CP or it can be partial operand information (address)

-After decoding F_0 & F_1, the CPU 'wakes up' the CP by sending it certain control signals. The CPU then transmits F_2 to a predefined location that serves as the latter's command register. F_2 is decoded by CP, and the execution is initiated.

Ways of providing the address of operand required for CP by CPU :-

1) CPU can read a special status register that specifies CP's operand requirement.
2) CPU automatically responds to a CP opcode by initiating a dummy memory read cycle, which places a memory address on the system address Bus. This address is read by CP, which subsequently uses it & other addresses derived from the first to

fetch/store memory operands without further CPU involvement.
- On termination, CP deactivates its busy signal & places appropriate information in its status register.