Lecture of Module 2

Instruction Set Architecture

Overview

- **Introduction**
- **Instruction Format**
- **Different types of Instruction format**
- **Addressing Modes**
- **Types of Instruction**
- **Instruction Cycle**

Introduction

- A program is a set of instructions that specify the operations, operands and sequence of operations by which processing has to occur.
- A computer instruction is a binary code that specifies a sequence of micro-operations.
- Instruction code together with data stored in the computer.
- The collection of different instructions that the processor can execute is referred to as the processor's *instruction set.*
- Every processor has its own specific Instruction Set.
- Each instruction must contain the information required by the processor for execution.
- The ability to store and execute instruction, the stored program concept is most important property of general purpose computer.
- The instruction code is a group of bits that instruct the computer to perform specific operation.

Elements of Instruction

Operation code (opcode)

- Specifies the operation to be performed. The operation is specified by a binary code, known as the operation code, or *opcode*

Source operand reference

- The operation may involve one or more source operands, that is, operands that are inputs for the operation

Result operand

- The operation may produce a result

Next instruction reference

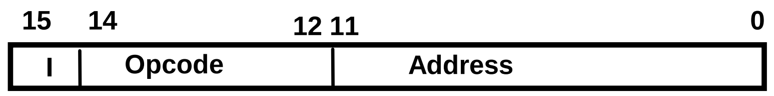
- This tells the processor where to fetch the next instruction after the execution of this instruction is complete

General Instruction Format

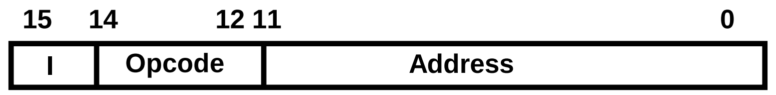
Opcode	Operand/Address

- General instruction format consists of two fields
  - Opcode
  - Operand
- The number of bits in opcode or operational code depends on the number of operations available in the Instruction set.
- When the Opcode decoded in the Control unit, it issues Control signals to read the operand and initiates micro-operation to perform complete operation of that instruction.
- The Operand/Address part may be the data or address of data or Register or I/O device etc.
- This also specifies where the result to be stored.
- The number of bits in operand field depends on the number of registers in that processor or address bits required to locate memory location.

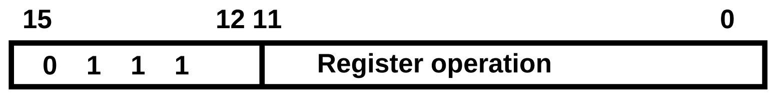
Example



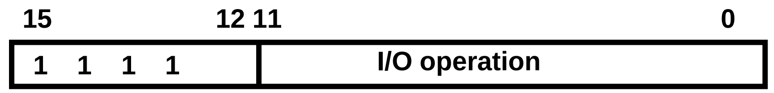
Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)



Input-Output Instructions (OP-code =111, I = 1)



- Let the instruction code having 16 bits
- 12 bits for Operand/Address, 3 bits for Opcode and 1 bit for mode (I) bit
- If I = 0 Direct addressing memory reference instruction
- If I = 1 Indirect addressing memory reference instruction
- Opcode 111 are for Register or I/O reference instruction
- If I = 0, it is Register reference instruction
- If I = 1, it is I/O reference instruction

- Opcodes are represented by abbreviations called *mnemonics*
- Examples:

ADD	Add
SUB	Subtract
MUL	Multiply
DIV	Divide
LOAD	Load data from memory
STOR	Store data to memory

- Operands are also represented symbolically
- Each symbolic opcode has a fixed binary representation
  - The programmer specifies the location of each symbolic operand

Symbol	*Description*
AND	AND memory word to AC
ADD	Add memory word to AC
LDA	Load AC from memory
STA	Store content of AC into memory
BUN	Branch unconditionally
BSA	Branch and save return address
ISZ	Increment and skip if zero
CLA	Clear AC
CLE	Clear E
CMA	Complement AC
CME	Complement E
CIR	Circulate right AC and E
CIL	Circulate left AC and E
INC	Increment AC
SPA	Skip next instr. if AC is positive
SNA	Skip next instr. if AC is negative
SZA	Skip next instr. if AC is zero
SZE	Skip next instr. if E is zero
HLT	Halt computer
INP	Input character to AC
OUT	Output character from AC
SKI	Skip on input flag
SKO	Skip on output flag
ION	Interrupt on
IOF	Interrupt off

Instruction Format

- A computer will usually have a variety of Instruction Code format.
- It is the function of Control unit within the CPU to interpret each instruction code and provide necessary control functions needed to process the instruction.
- The bits of instructions are divided into groups called fields.
- The most common fields found in instruction format are:
  - An Operation code field (Opcode)
  - Operand or Address fields
  - A mode field that specifies the way the operand or effective address of the operand is determined
  - Other special fields are sometimes employed under certain circumstances
- Computers may have instructions of several lengths containing varying number of address fields.
- The number of address field in the instruction format of a computer depends on the internal organization of its registers.

- Most of the computers fall into one of three types of CPU organization.
  - Single Accumulator organization
  - General Register organization
  - STACK organization

Single Accumulator organization

ADD X	$AC \leftarrow AC + M[X]$

General Register organization

ADD R1, R2, R3	$R1 \leftarrow R2 + R3$
ADD R1, R2	$R1 \leftarrow R1 + R2$
ADD R1, X	$R1 \leftarrow R1 + M[X]$

STACK organization

PUSH A

PUSH B

ADD

Different types of Instruction format

- Some computers may follow one or more than one or all mentioned features in their structure.
- There are various types of Instruction format according to the number of Address fields.
  - Three-address instruction
  - Two-address instruction
  - One-address instruction
  - Zero-address instruction

Example:

Let the expression is $X = (A + B) \times (C + D)$

Three-address instruction

ADD R1, A, B	$R1 \longleftarrow M[A] + M[B]$
ADD R2, C, D	$R2 \longleftarrow M[C] + M[D]$
MUL X, R1, R2	$M[X] \longleftarrow R1 \times R2$

Expression is $X = (A + B) \times (C + D)$

Two-address instruction

MOV R1, A	$R1 \leftarrow M[A]$
ADD R1, B	$R1 \leftarrow R1 + M[B]$
MOV R2, C	$R2 \leftarrow M[C]$
ADD R2, D	$R2 \leftarrow R2 + M[D]$
MUL R1, R2	$R1 \leftarrow R1 \times R2$
MOV X, R1	$M[X] \leftarrow R1$

One-address instruction

LOAD A	$AC \leftarrow M[A]$
ADD B	$AC \leftarrow AC + M[B]$
STOR T	$M[T] \leftarrow AC$
LOAD C	$AC \leftarrow M[C]$
ADD D	$AC \leftarrow AC + M[D]$
MUL T	$AC \leftarrow AC \times M[T]$
STOR X	$M[X] \leftarrow AC$

Expression is $X = (A + B) \times (C + D)$

Zero-address instruction

PUSH	A	$Top \leftarrow A$
PUSH	B	$Top \leftarrow B$
ADD		$Top \leftarrow (A + B)$
PUSH	C	$Top \leftarrow C$
PUSH	D	$Top \leftarrow D$
ADD		$Top \leftarrow (C + D)$
MUL		$Top \leftarrow (A + B) \times (C + D)$
POP	X	$M[X] \leftarrow Top$

Addressing Modes

- Implied/Implicit
- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- STACK
- Auto Increment / Auto Decrement

- In an instruction format the way the operands are chosen during program execution is dependent on the Addressing mode of the instruction.
- Before any operand is referenced, the addressing mode specifies the operand or address of the operand.
- Various types of Addressing modes are there as stated.
- A particular Instruction Set Architecture may follow some or all of these addressing modes to find the effective address of the operand.

Implied/Implicit Addressing mode

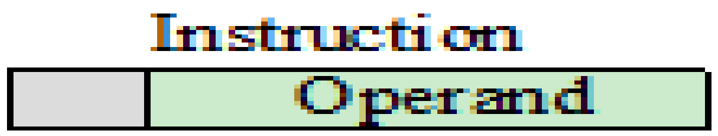
- No address field required.

Instruction format

- The operand is specified within the instruction implicitly.
- All the instructions that uses accumulator implicitly within the instruction called implied addressing mode.
- CLA – Clear the content of Accumulator
- CMA – Complement the content of the Accumulator
- No memory reference other than the instruction fetch.

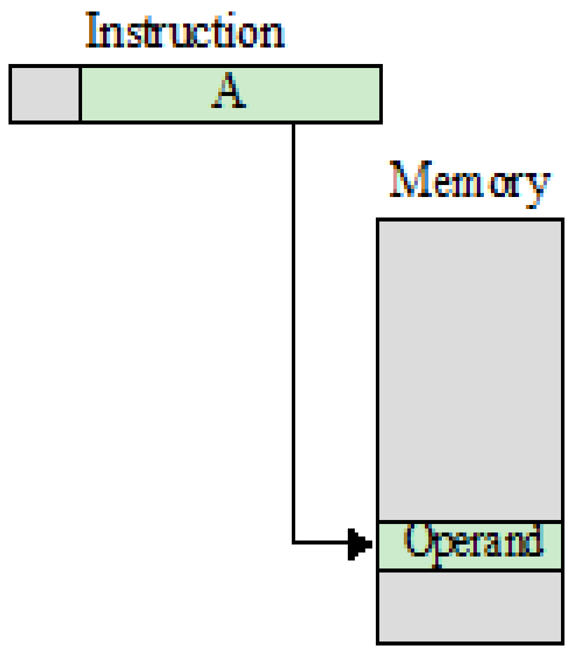
Immediate Addressing Mode

- Simplest form of addressing.
- Operand = A
- The required operand is present in the instruction.
- This mode can be used to define and use constants or set initial values of variables.
- No memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle.
- Size of the data restricted to the size of the Operand/Address field. It may be less than the word length, which is a drawback.



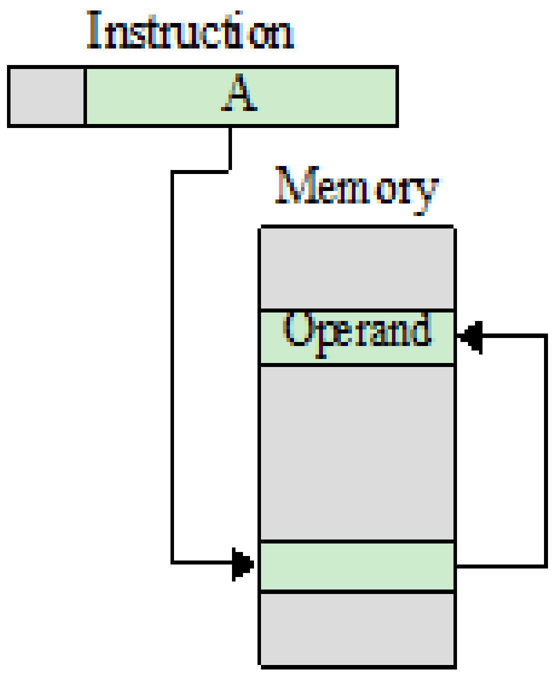
Direct Addressing Mode

- In the instruction format the address field contains the effective address of the operand.
- Effective address (EA) = address field A
- It was common in earlier generations of computers.
- Except instruction fetching, it requires only one memory reference for data reading.
- No special calculation necessary for effective address.
- Limitation is that it provides only a limited address space.



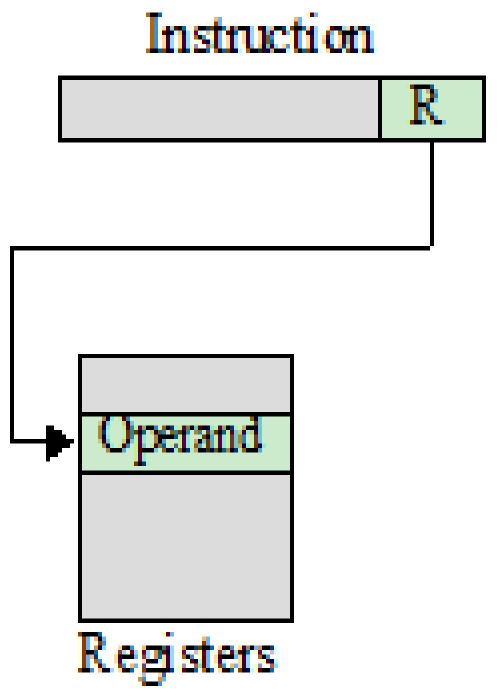
Indirect Addressing Mode

- In the instruction format the Operand/Address field contains an address where the address of the data is present.
- Effective address (EA) = [A]
- Reference to the address of a word in memory which contains a full-length address of the operand.
- For a word length of N an address space of 2^N is now available, which resolves the limitation of Direct addressing.
- Instruction execution requires two memory references to fetch the operand. One to get its address and a second to get its value.



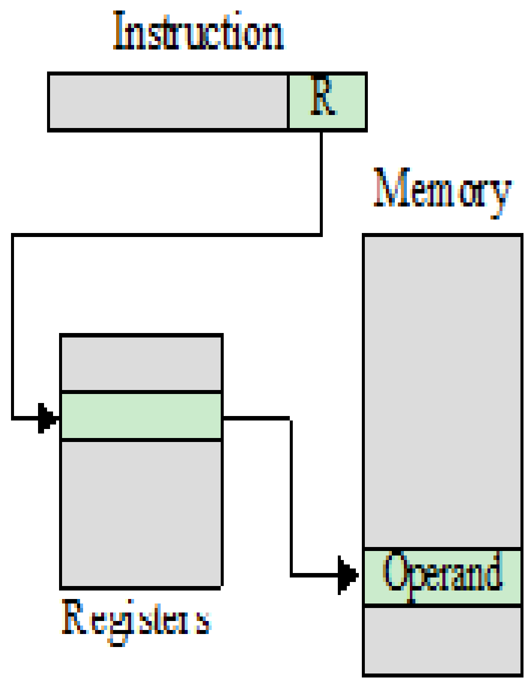
Register Addressing Mode

- In the instruction format the Operand/Address field specifies a register rather than a memory address which contains the required operand.
- EA = R
- Only a small Operand/Address field is needed in the instruction to specify a particular register.
- No memory references rather register reference to find the required data.
- The address space is very limited



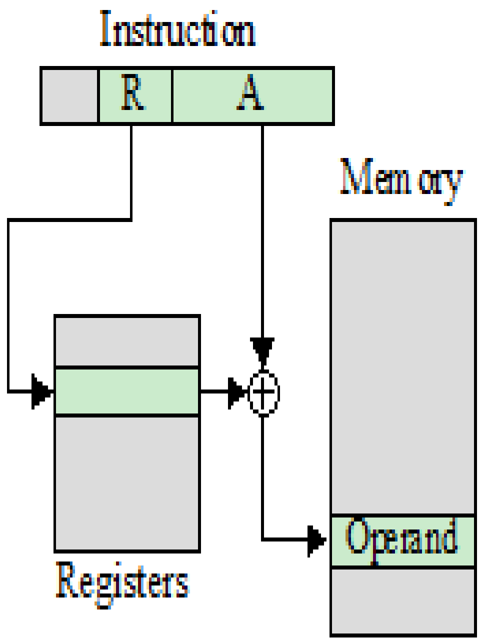
Register Indirect Addressing Mode

- In the instruction format the Operand/Address field specifies a register which contains the address of an operand in the memory location.
- Analogous to indirect addressing
  - difference is whether the address field refers to a memory location or a register.
  - Uses one less memory reference than indirect addressing.
  - Instruction format may be smaller than indirect addressing.
- EA = [R]
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address.
- Before using a Register Indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.



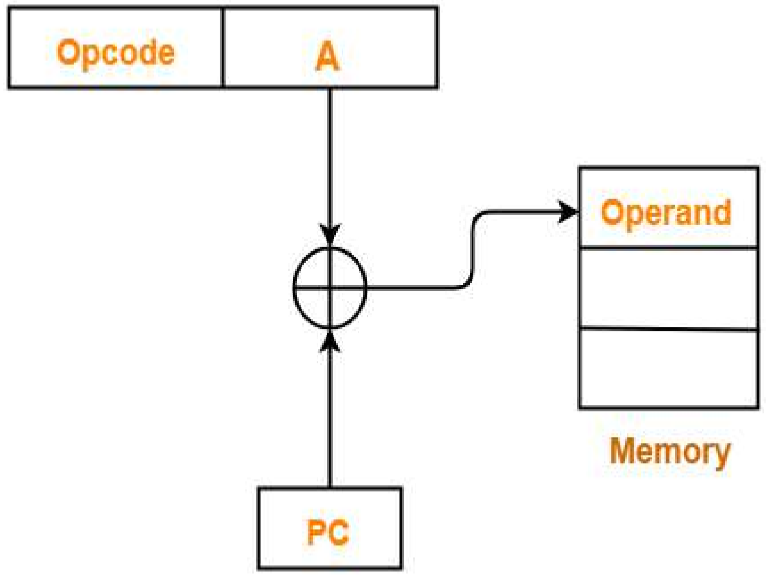
Displacement Addressing Mode

- Very powerful addressing mode
- Combines the capabilities of register indirect addressing and direct addressing
- EA = [R] + A
- The instruction may have two address fields
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the effective address
- Most common uses are:
  - Relative addressing
  - Base-register addressing
  - Indexed addressing



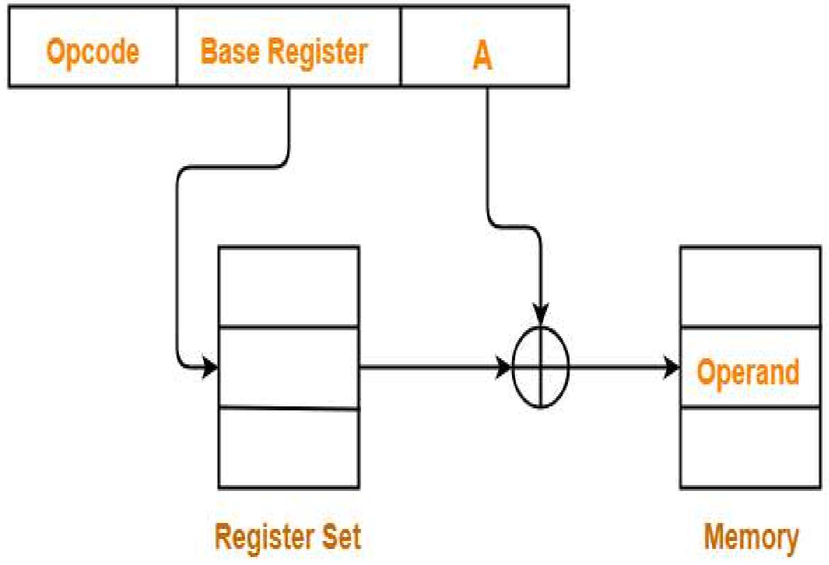
Relative Addressing Mode

- The referenced register is the Program Counter (PC).
- The next instruction address (PC) is added to the address field A to produce the effective address (EA).
- Typically the address field is treated as a twos complement number for this operation.
- The effective address is a displacement relative to the address of the instruction.
- Exploits the concept of locality.
- Saves address bits in the instruction if most memory references are relatively near to the instruction being executed.
- Also called Limit Addressing Mode.



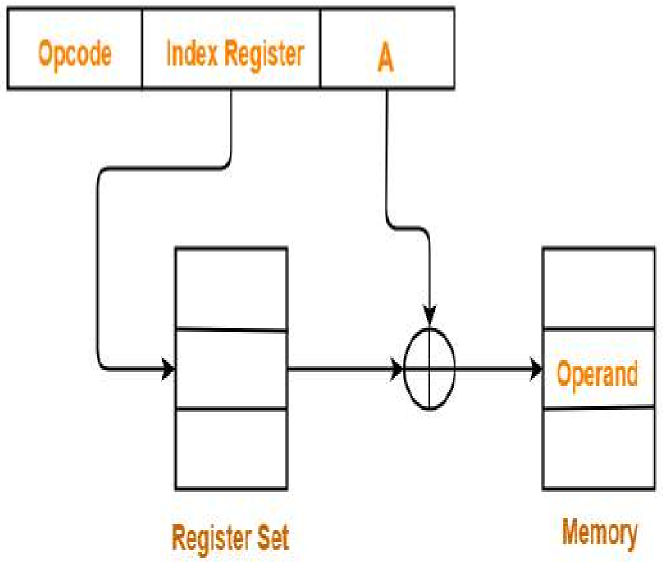
Base-register Addressing Mode

- In this type of addressing mode the content of the Base register is added to the address part, that is, A of the instruction to obtain effective address.
- The Base register contains a main memory address and the address field contains a displacement from that address.
- Exploits the locality of memory references.
- Convenient means of implementing segmentation.
- In some implementations a single segment base register is employed.
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it.



Indexed Addressing Mode

- Index Register is a special CPU register contains an Index value.
- The address field (A) references a main memory address and the index register contains a positive displacement from that address.
- The method of calculating the EA is the same as for base-register addressing.
- An important use is to provide an efficient mechanism for performing iterative operations.
- Autoindexing
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (XR)
  - (XR) 🡐 (XR) + 1



STACK Addressing Mode

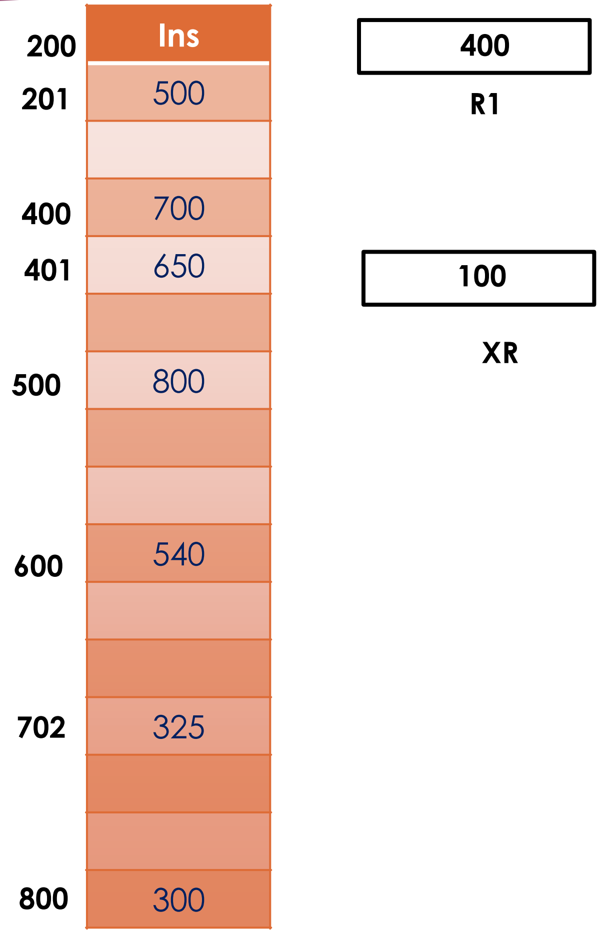
- A stack is a linear array of locations.
- Items are appended to the top of the stack so that the block is partially filled.
- Associated with the Stack Pointer (SP) whose value is the address of the top of the stack.
  - The stack pointer is a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- This is also a form of implied addressing mode.
- In this mode, operand is at the top of the stack.
- For example: ADD, this instruction will POP top two operands from the stack, add them, and will then PUSH the result to the top of the stack.

Auto increment/Auto decrement Addressing

- It is similar to Register or Register Indirect Addressing mode.
- If it Register addressing mode then the content of the specified register is incremented or decremented by 1.
- If it Register Indirect addressing mode then the address of the operand is incremented or decremented by 1.
- To access consecutive location the address value is stored in the register and then incremented or decremented

Example:

- An instruction is stored at location 200 with address field at location 201. The address field has the value 500. A processor register R1 contains 400. Evaluate the effective address for different addressing modes if XR value is 100.
  - Direct Addressing Mode: EA = 500, AC = 800
  - Indirect Addressing Mode: EA = 800, AC = 300
  - Immediate Addressing Mode: EA = 201, AC = 500
  - Register Addressing Mode: EA = No, AC = 400
  - Register Indirect Addressing Mode: EA = [R1] = 400, AC = 700
  - Relative Addressing Mode: EA = [PC]+500 = 702, AC = 325
  - Indexed Addressing Mode: EA = [XR] + 500 = 600, AC = 540
  - Auto increment (Register addressing): EA = No, AC = 401
  - Auto increment (Register indirect addressing): EA = 401, AC = 650



Types of Instruction

- A computer provides an extensive set of instructions to give the user flexibility to carry out various computational tasks.
- The instruction set for different processors differ from each other mostly in the way the operands are determined and mode field.
- The actual operations available in the instruction set are not very different from one computer to another.
- The binary code of the Opcode may differs from processor to processor, even for the same operation.
- Most computer instructions can be classified into following categories.
  - Data transfer instructions
  - Data manipulation instructions
    - Arithmetic instructions
    - Logical and bit manipulation instructions
    - Shift instructions
  - Program control instructions

Data Transfer Instructions

- Data transfer instructions move data from one place to another without changing the data content.
- Most common transfers are between memory and processor registers, between processor registers and I/O, and between processor registers themselves.
- LOAD, STOR, MOV, XCHG, IN, OUT, PUSH, POP etc.

Data manipulation instructions

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
  - Arithmetic instructions
  - Logical and bit manipulation instructions
  - Shift instructions

Arithmetic Instructions

- Basic arithmetic operations are addition, subtraction, multiplication and division.
- Most computers provide instructions for all four operations.
- Some small computers have only addition and possibly subtraction instructions.
- The multiplication and division must then be performed by help of addition and other instructions.
- ADD, SUB, MUL, DIV, INC, DEC, ADDC, SUBB etc.

Logical and Bit manipulation Instructions

- AND, OR, XOR, CLR, COM, CLRC, SETC, COMC etc.

Shift Instructions

- Instructions to shift the content of a register or accumulator are quite useful.
- Shift operations may specify either logical shifts or arithmetic shifts or rotate type operations.
- In either case the shift may be to the left or to the right.
- SHR, SHL, ASHR, ASHL, RAR, RAL, RRC, RLC etc.

Program Control instructions

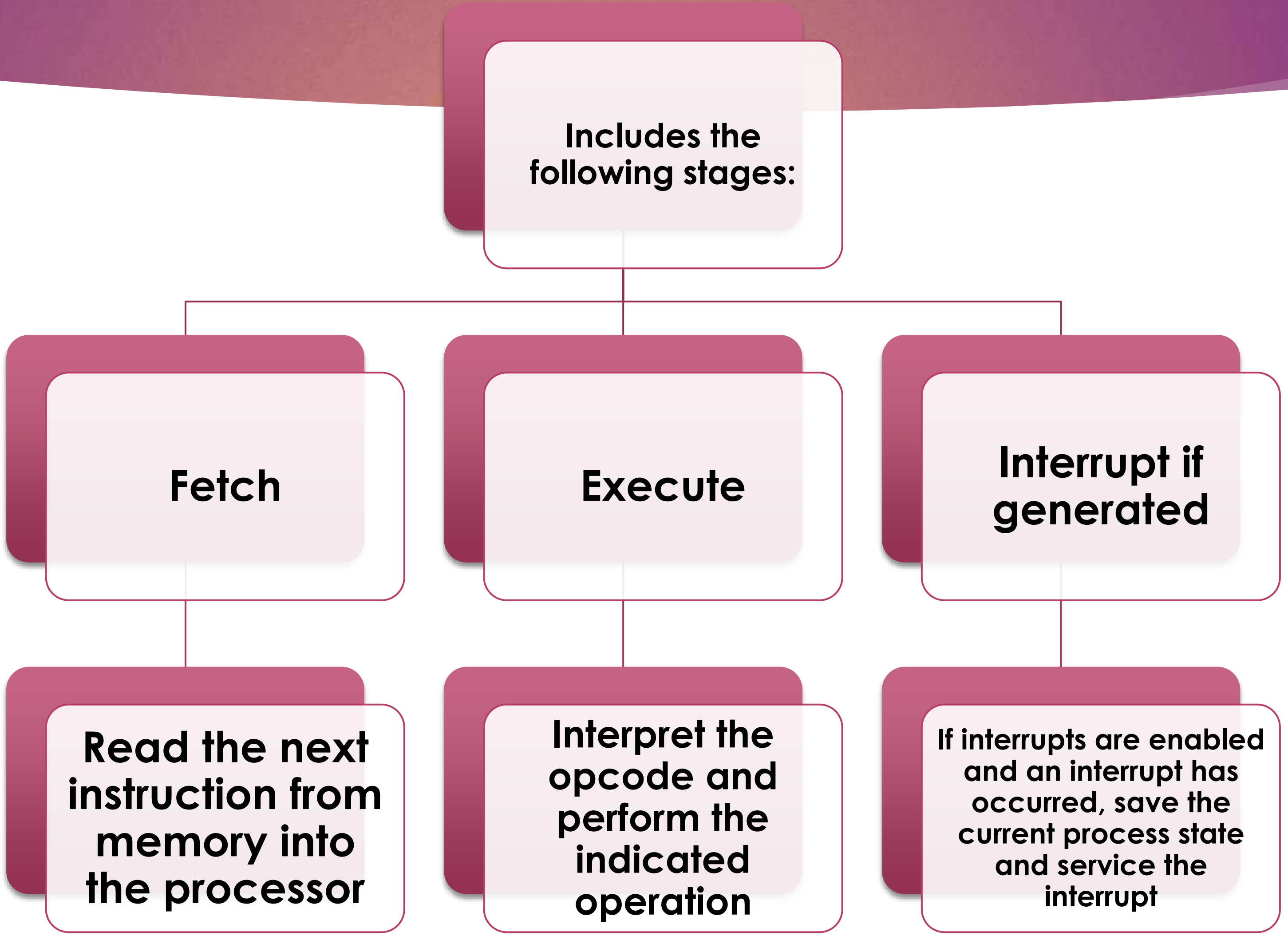
- Instructions are always stored in successive memory locations.
- Instructions are fetched from memory and executed.
- Just after fetching of an instruction, the Program counter is incremented for the next instruction in sequence.
- On the other hand, a program control type of instruction, when executed may change the address value in the Program counter and cause the flow of control to be altered.
- The change in value of the Program counter due to program control instruction cause a break in sequence of instruction execution.
- BR, JUMP, JC, JNC, JZ, JNZ, JP, JN, SKP, CALL, RET etc.

Instruction Cycle

- An instruction is a command given to the computer to perform specific operation on given data.
- To perform a particular task a sequence of instructions are written, called a program.
- Generally instruction and data are stored in the memory.
- The necessary steps that a CPU carries out to fetch an instruction and required data and then to process it constitute an Instruction Cycle.
- Each Instruction cycle is subdivided into a sequence of sub cycles or phases.
- In Basic Computer, an instruction passes through following sub cycles:
  1. Fetch an instruction from memory
  2. Decode the instruction
  3. Read the required Operand
  4. Entertain the Interrupt if it is generated
  5. Execute the instruction

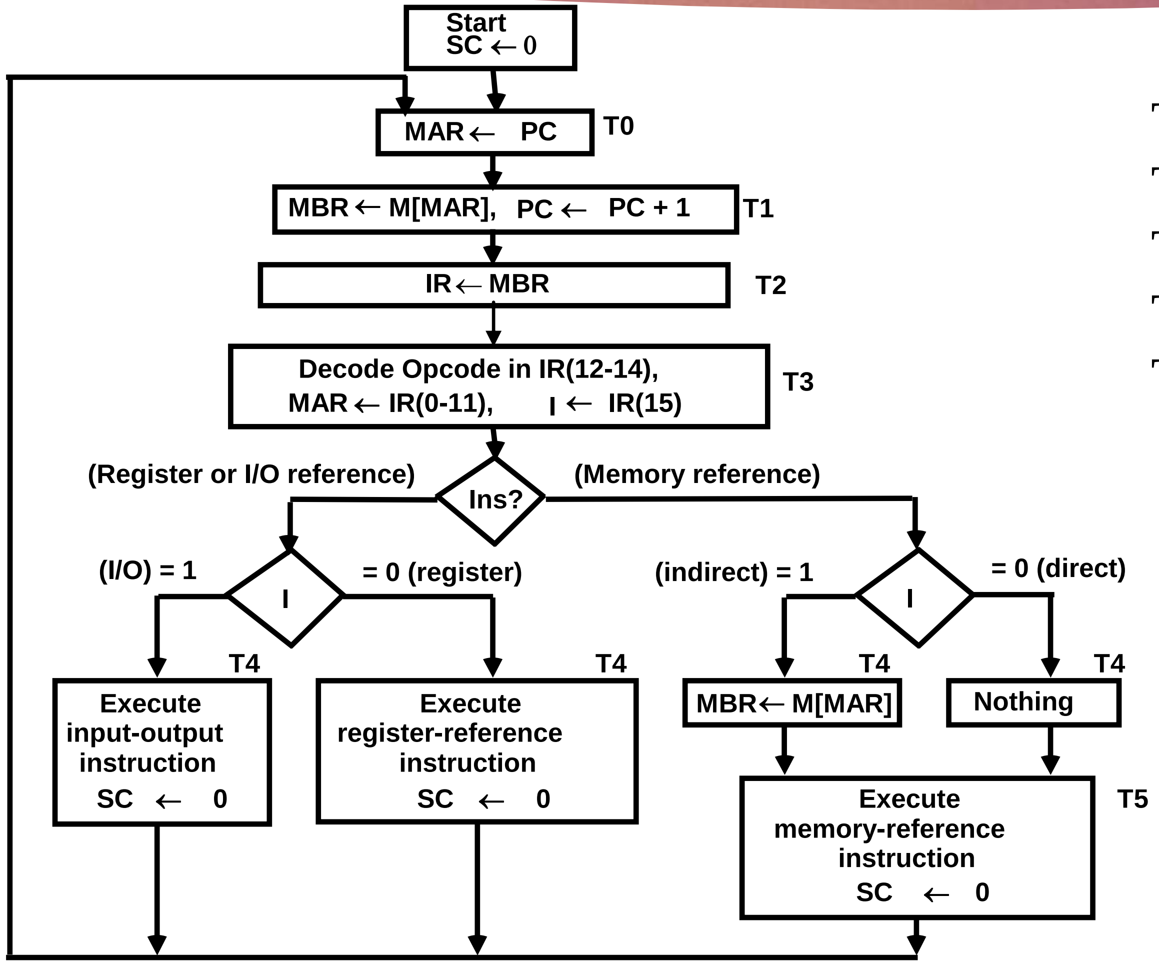
***Note*:** Every different processor has its own (different) instruction cycle

Instruction Cycle



Includes the following stages:
Fetch
Execute
Interrupt if generated
Read the next instruction from memory into the processor
Interpret the opcode and perform the indicated operation
If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt

Instruction Cycle Flow Diagram


Start
SC ← 0
MAR ← PC
T0
MBR ← M[MAR], PC ← PC + 1
T1
IR ← MBR
T2
Decode Opcode in IR(12-14),
MAR ← IR(0-11), I ← IR(15)
T3
(Register or I/O reference)
Ins?
(Memory reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T4
T4
T4
T4
Execute input-output instruction SC ← 0
Execute register-reference instruction SC ← 0
MBR← M[MAR]
Nothing
Execute memory-reference instruction SC ← 0
T5

T0: MAR ⟵ PC

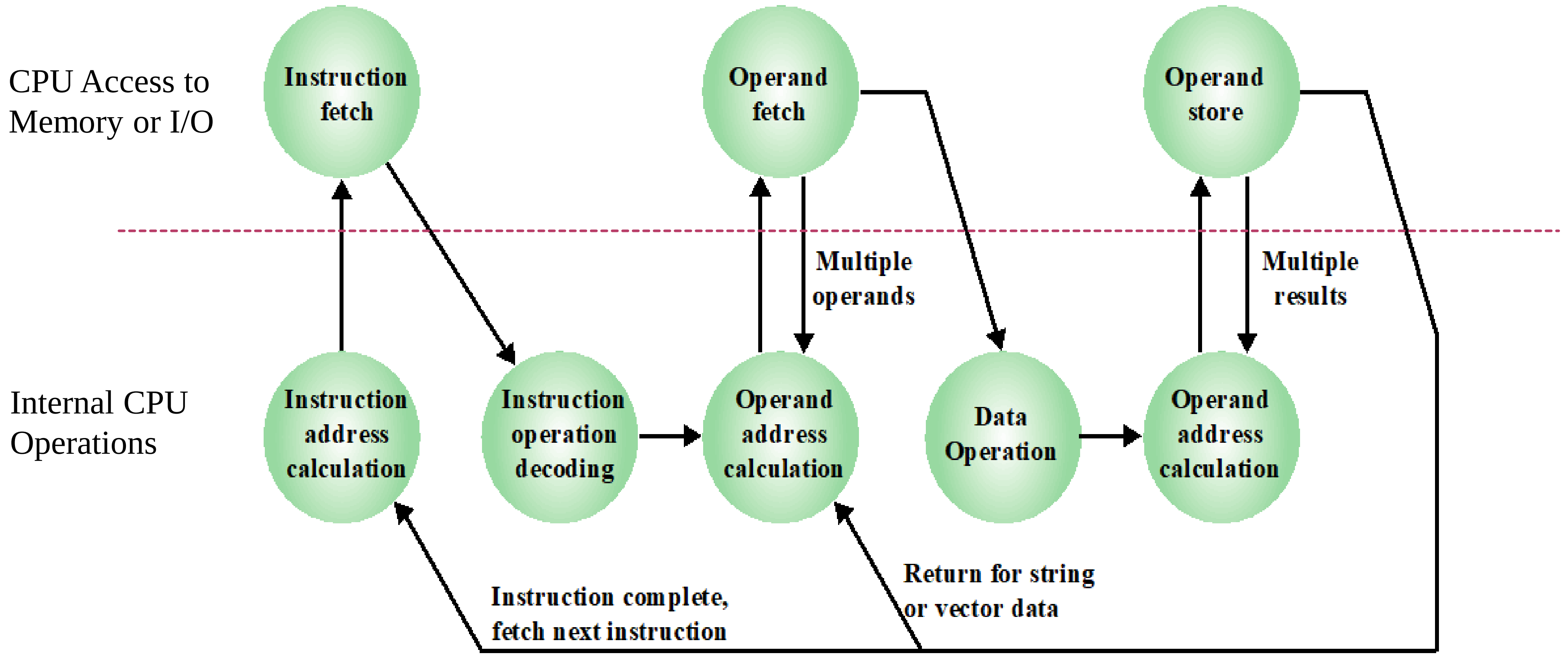
T1: MBR ⟵ M[MAR], PC ⟵ PC+1

T2: IR ⟵ MBR

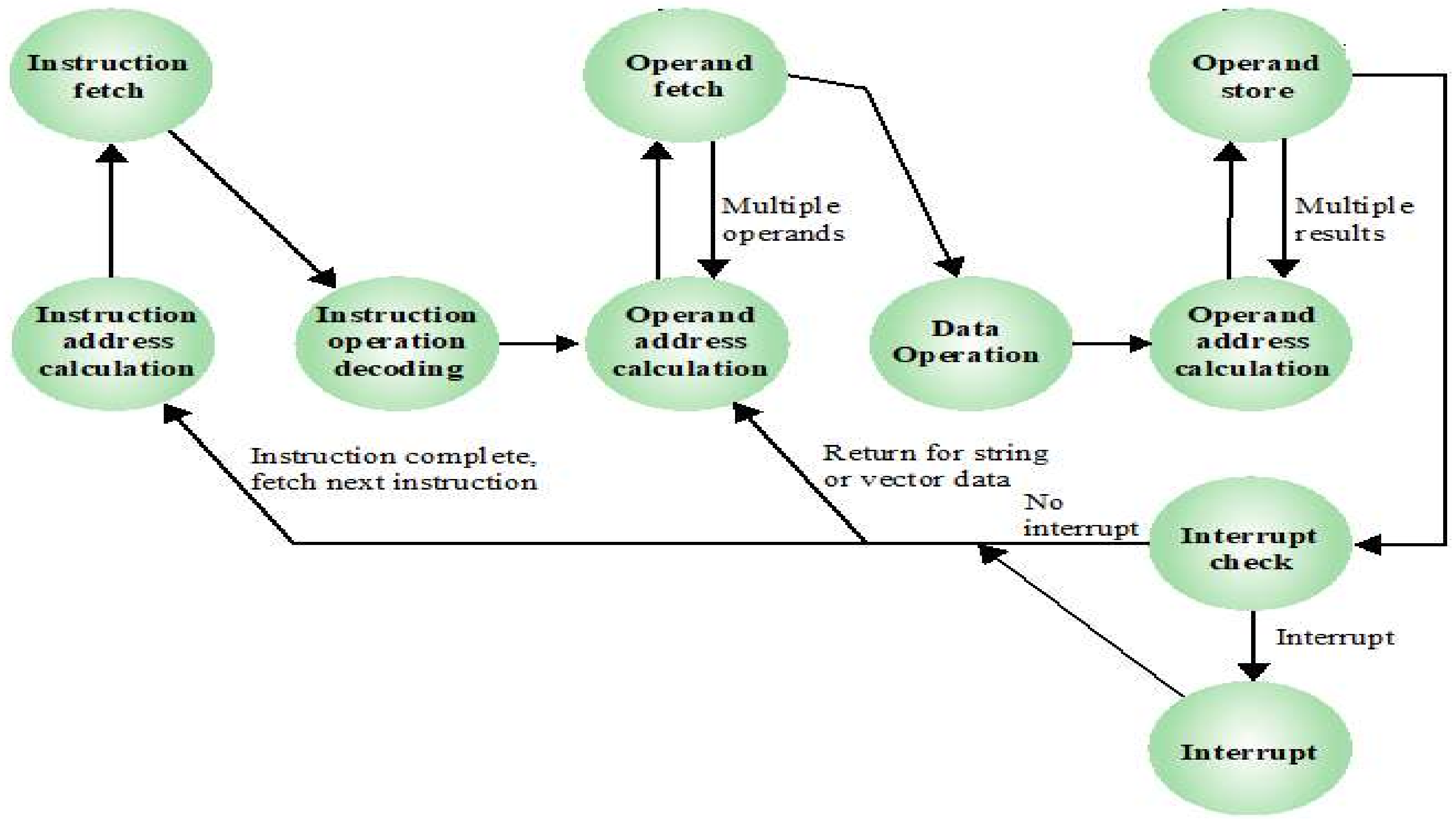
T3: Decode

T4, T5: Execute

Instruction Cycle State Diagram


CPU Access to Memory or I/O
Internal CPU Operations
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Return for string or vector data
Instruction complete, fetch next instruction

Instruction Cycle State Diagram with Interrupt


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt
Interrupt check
Interrupt
Interrupt

CPU Organization

Overview

Introduction

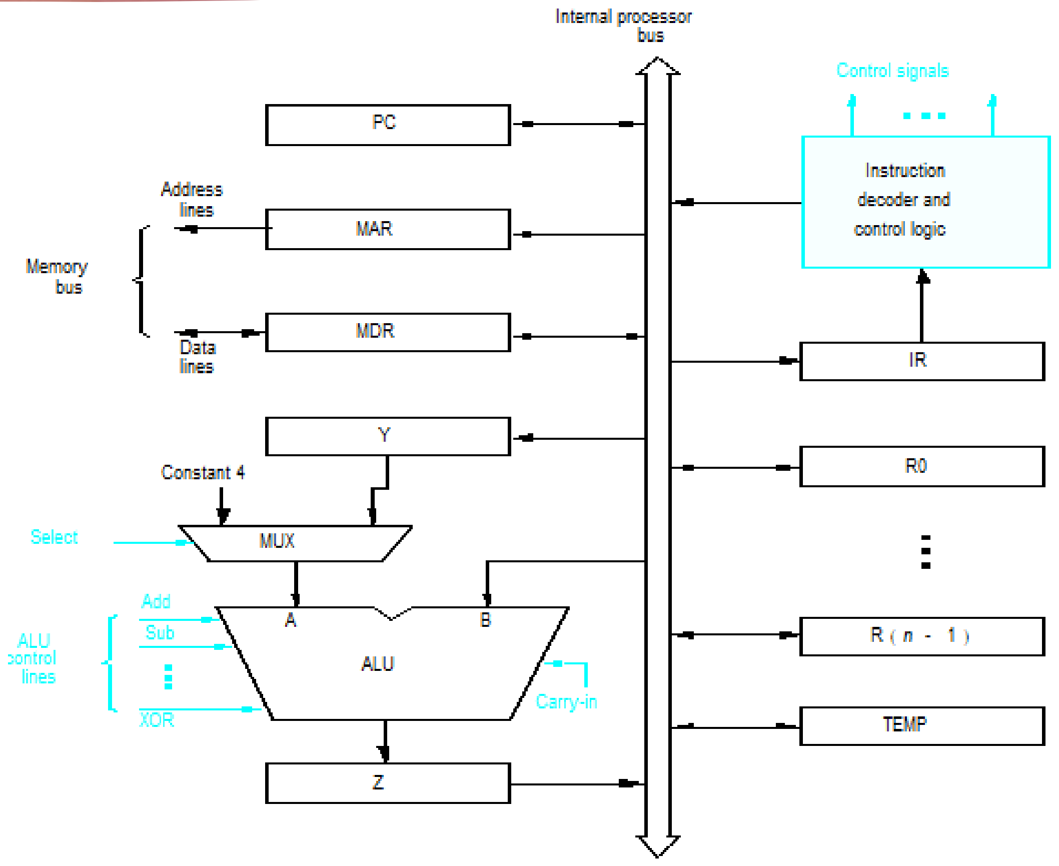
- Processor fetches one instruction at a time and perform the required operation.
- Instructions are fetched from successive memory locations until a Branch, Call or a Jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- After fetching from memory instruction is stored in Instruction Register (IR) for decoding.

1. Fetch the contents of the memory location pointed to by the PC.
2. Assuming that the memory is byte addressable, and each instruction comprises 4 bytes, increment the contents of the PC by 4. $PC \leftarrow [PC] + 4$
3. Carry out the operations specified by the instruction.

Steps 1 and 2 constitute the fetch phase, and step 3 constitutes the execution phase.

Single Bus organization (Data Path) inside Processor

- Generally there are two sub units within the processor, that is, Control Unit and Datapath.
- Physically there is hardly any difference between the Control unit and Datapath.
- The hardware of both subunits are tightly coupled in a single physical unit.
- The Datapath includes the internal path for movement of data within the Processor between ALU and registers and other hardware like temporary storage, lathes, multiplexers, demultiplexers, decoders, counters, delay logics etc.
- Other buses are external buses that connects memory and I/O devices.

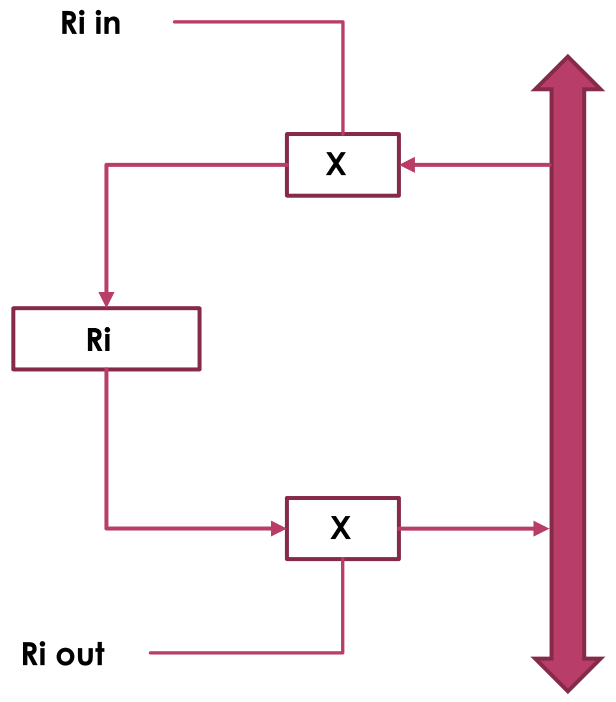


Single-bus organization of the datapath inside a processor.

- The ALU and all the registers are interconnected via a single common bus. This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- The data and address lines of the external memory bus are connected to the internal processor bus via the memory data register (MDR), and the memory address register (MAR), respectively.
- The input of the MAR is connected to the internal bus, and its output is connected to the external bus.
- The number and use of the processor registers R0 through R(n-1) vary from one processor to another.
- Three registers, Y, Z, and TEMP are transparent to the programmer. They are used by the processor for temporary storage during execution of some instructions.
- The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input of A of the ALU. The constant 4 is used to increment the content of the program counter (PC).
- The instruction decoder and the control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register.

Register Transfers

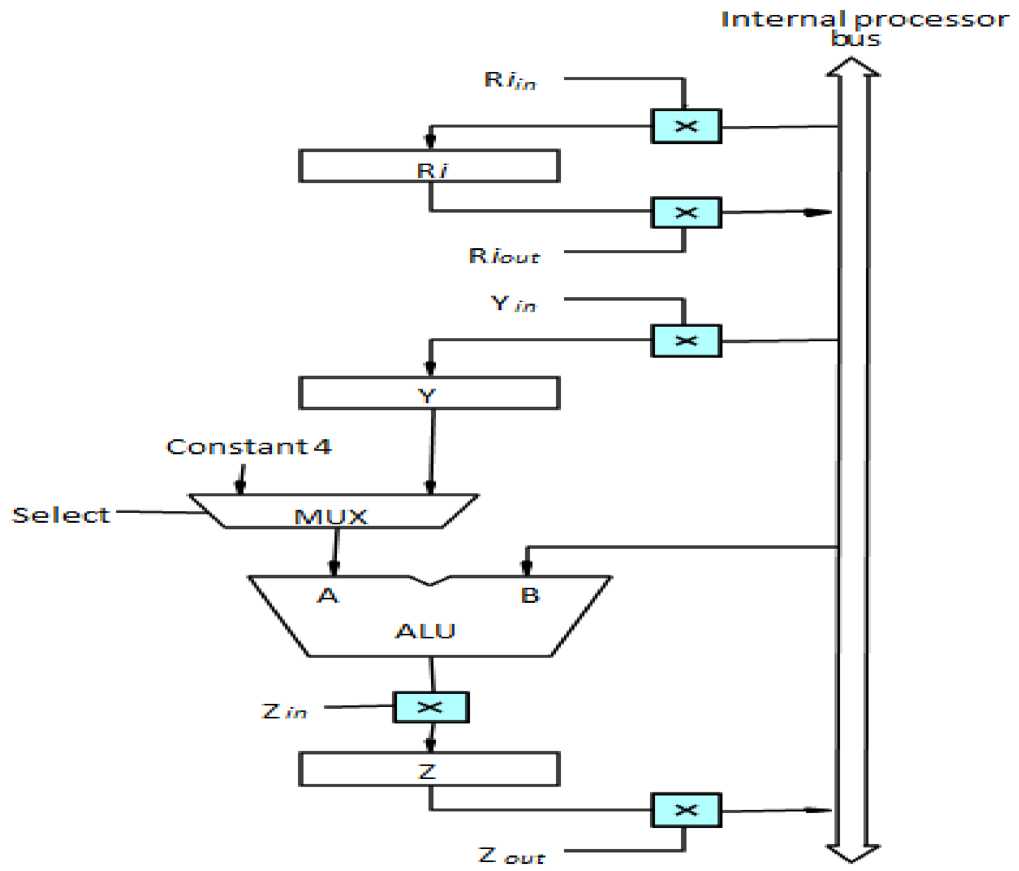
- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register, two control signals are used to place the content of that register on the bus or from the bus to the register.
- The input and output of register R*i* are connected to the bus via switches controlled by the signal Ri_{in} and Ri_{out} respectively.
- When $Ri_{in} = 1$, the data on the bus are loaded into R*i*.
- When $Ri_{out} = 1$, the contents of register R*i* are placed on the bus.
- While $Ri_{out} = 0$, the bus can be used for transferring data from other registers.
- All the operations and data transfers within the processor take place within time periods defined by the processor clock.



Control Sequences

- The ALU is a combinational circuit that has no internal storage.
- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- A sequence of operations to add the contents of register R1 to R2 and store the result in R3 is shown below.

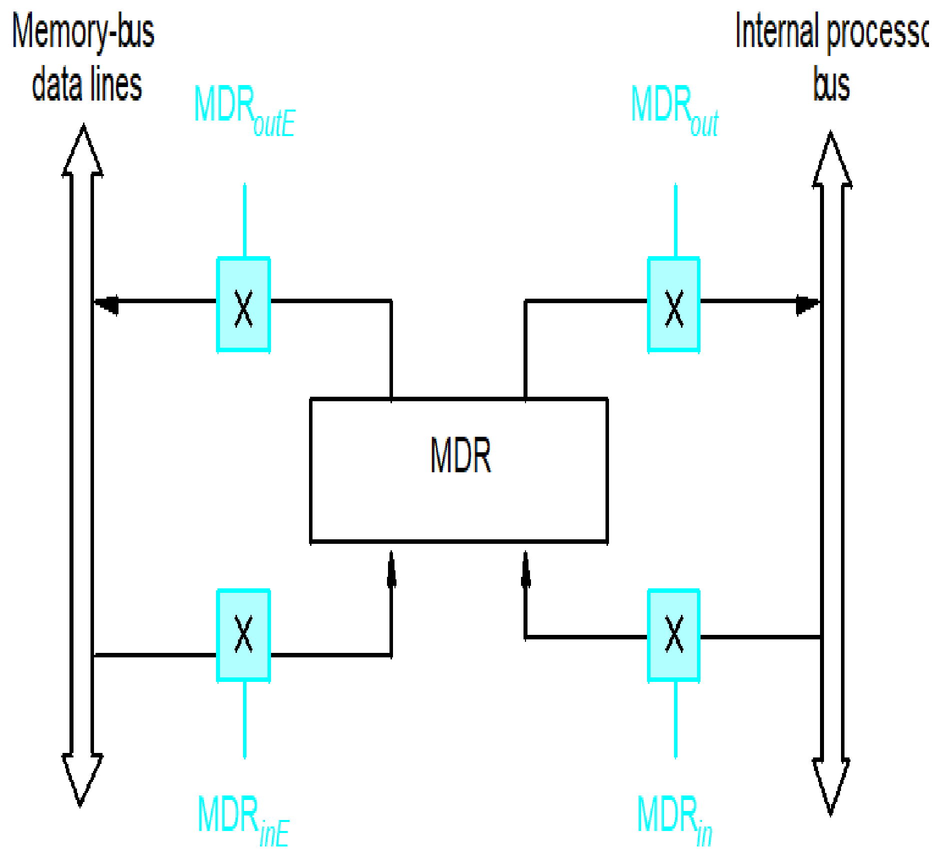
1. $R1_{out}$, Y_{in}
2. $R2_{out}$, SelectY, Add, Z_{in}
3. Z_{out}, $R3_{in}$



Input and output gating for the registers

Fetching a Word from Memory

- To fetch a word from the memory, the processor has to specify the address of the memory location
- Address into MAR; issue Read operation; data into MDR.
- It has four control signals.
- MDR_{in} and MDR_{out} control the connection to the internal bus.
- MDR_{inE} and MDR_{outE} control the connection to the external bus.



- During memory Read and Write operations, the timing of internal processor operations must be coordinated with the response of the addressed device on the memory bus.
- The processor completes one internal data transfer in one clock cycle.
- The speed of the operation of I/O devices are slower than the processor speed and also different device speed is different from each other.
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed.
- A control signal called Memory-Function-Complete (MFC) is used for this purpose.
- During Read or Write operation to or from memory, the other operations Wait for MFC signal.
- WMFC is a control signal that causes processor control circuitry to wait for arrival of MFC signal.

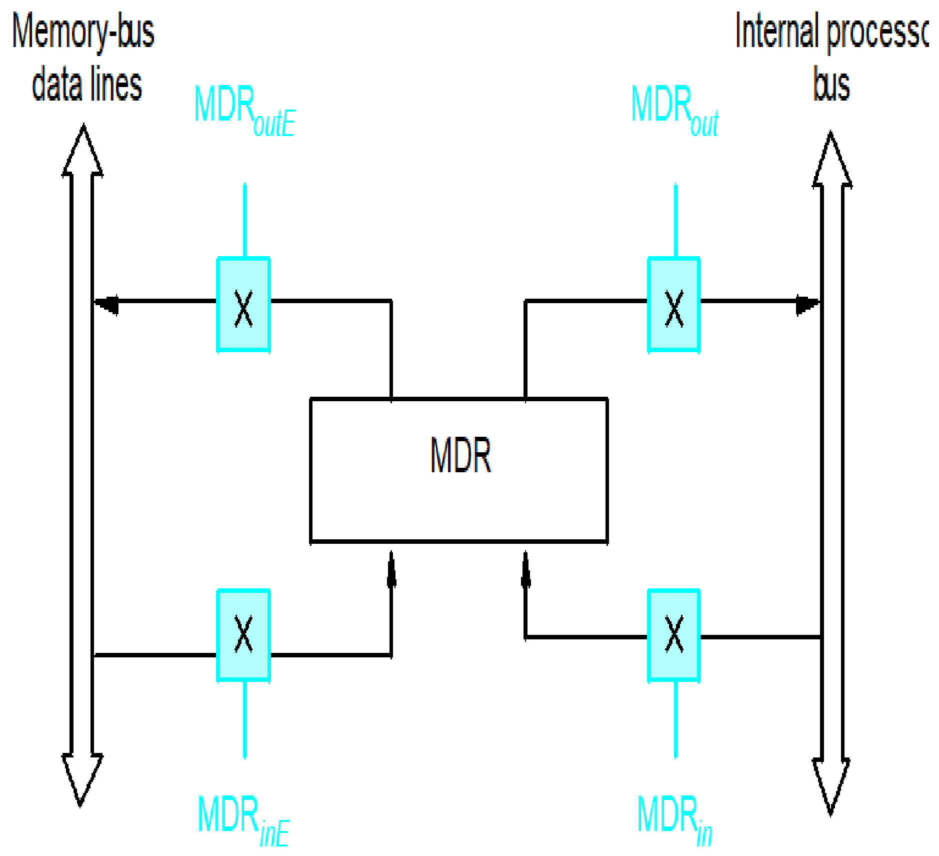
Example: Consider an instruction MOV R2, [R1]

- The actions needed for memory read are:

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← MDR

- Control Sequences are:

1. R*1out*, MAR$_{in}$, Read
2. MDR$_{inE}$, WMFC
3. MDR$_{out}$, R2$_{in}$



Storing a Word in Memory

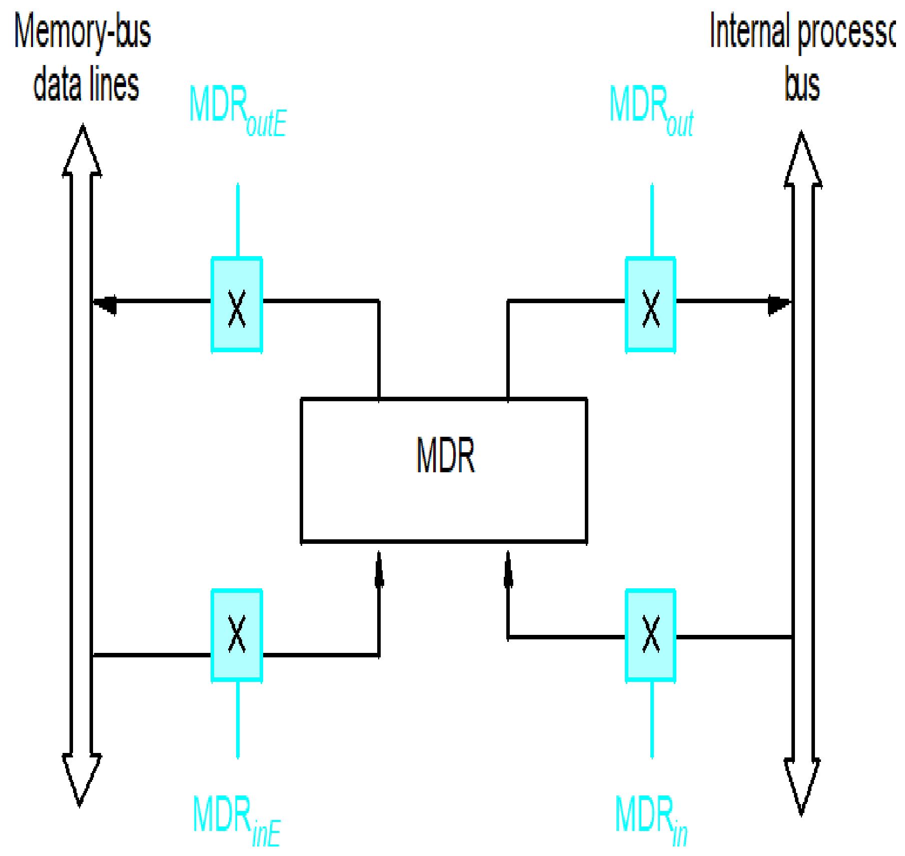
Example: Consider an instruction Move [R1], R2

- The actions needed for memory write are:

1. MAR ← [R1]
2. MDR ← R2
3. Start a write operation on the memory bus
4. Wait for the MFC response from the memory
5. Store memory bus from MDR

- Control Sequences are:

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, Write
3. $MDR_{out\ E}$, WMFC



Execution of a Complete Instruction

Example: Consider an instruction ADD R1, [R3]

- The actions needed for execution of complete instruction are:

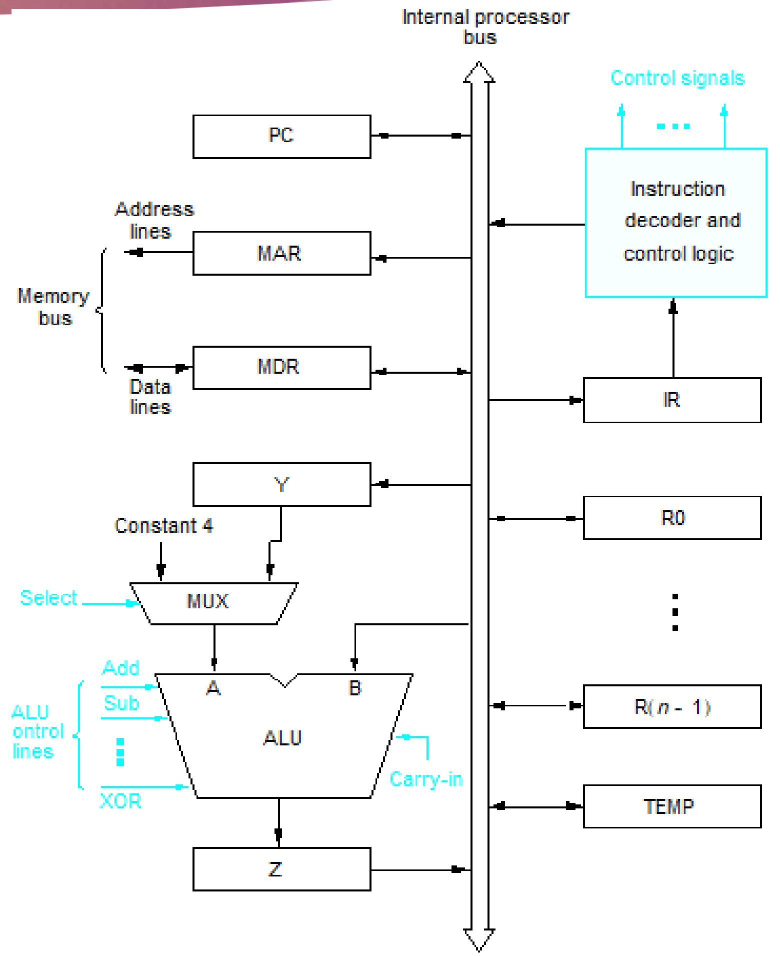
1. Fetch the instruction
2. Read the operand (the contents of the memory location pointed to by R3)
3. Perform the addition
4. Load the result into R1

Example: Consider an instruction ADD R1, [R3]

- Control Sequences are:

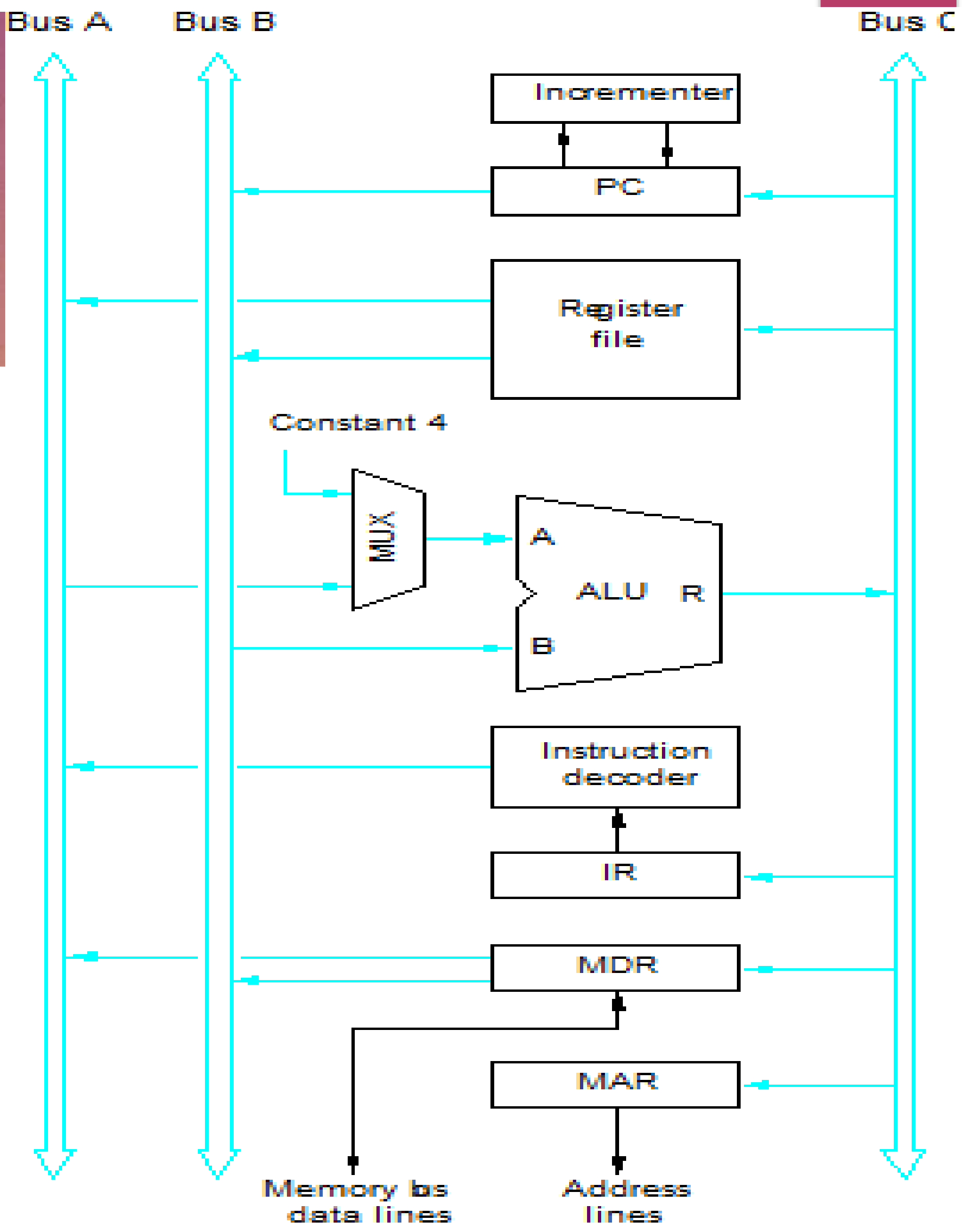
1. PC $_{out}$, MAR $_{in}$, Read, Select4, Add, Z $_{in}$
2. Z $_{out}$, PC $_{in}$, WMFC
3. MDR$_{in\,E}$, MDR $_{out}$, IR $_{in}$
4. R3 $_{out}$, MAR $_{in}$, Read
5. R1 $_{out}$, Y $_{in}$, WMFC
6. MDR$_{in\,E}$, MDR $_{out}$, SelectY, Add, Z $_{in}$
7. Z $_{out}$, R1 $_{in}$, End

- Sequence 1, 2 and 3 for Opcode fetching and PC increment for next instruction.
- Sequence 4 for locating memory location to read operand.
- Sequence 5 for reading operand from register.
- Sequence 6 for operand reading from memory to ALU and arithmetic operation.

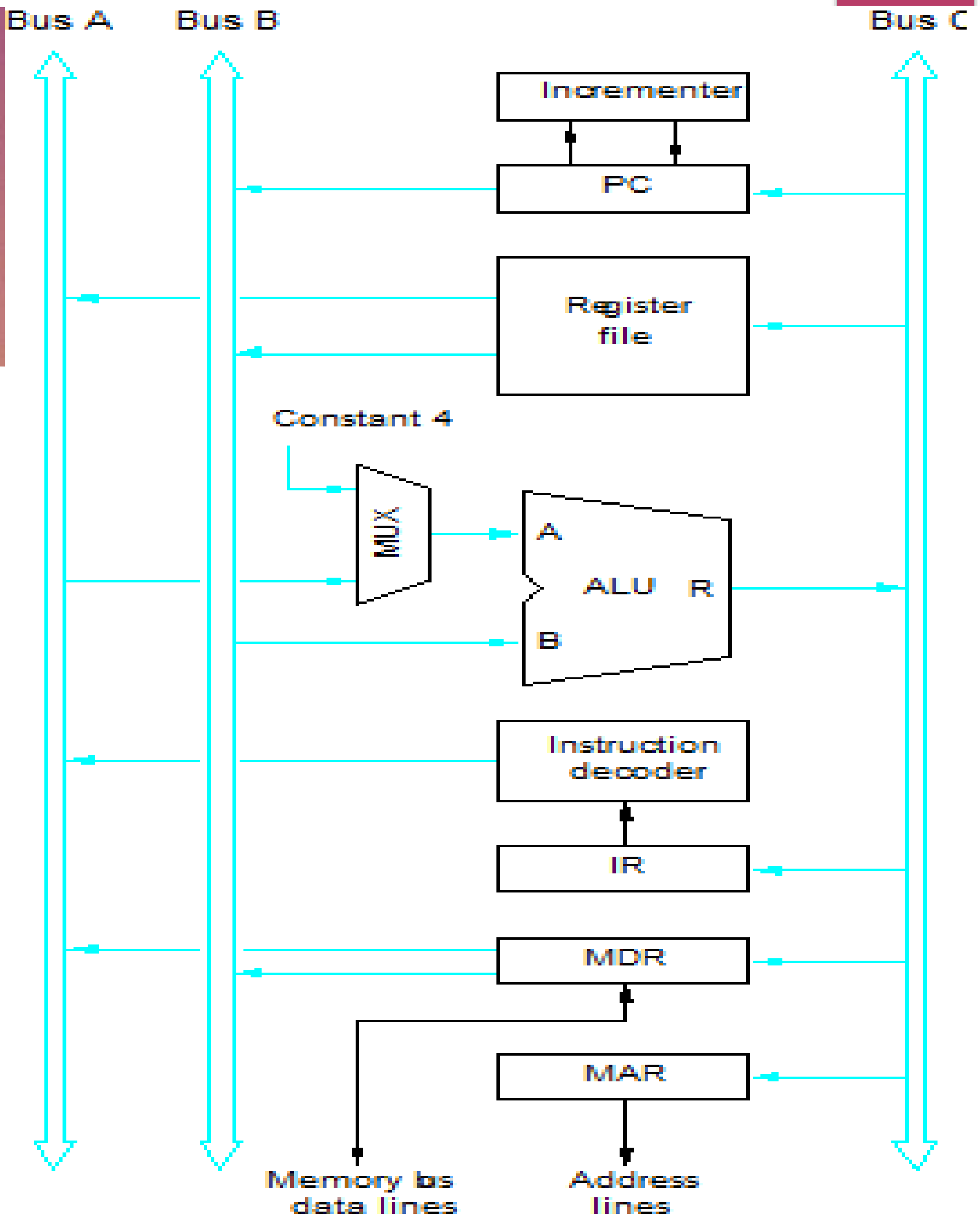


Multi Bus organization (Data Path)inside Processor

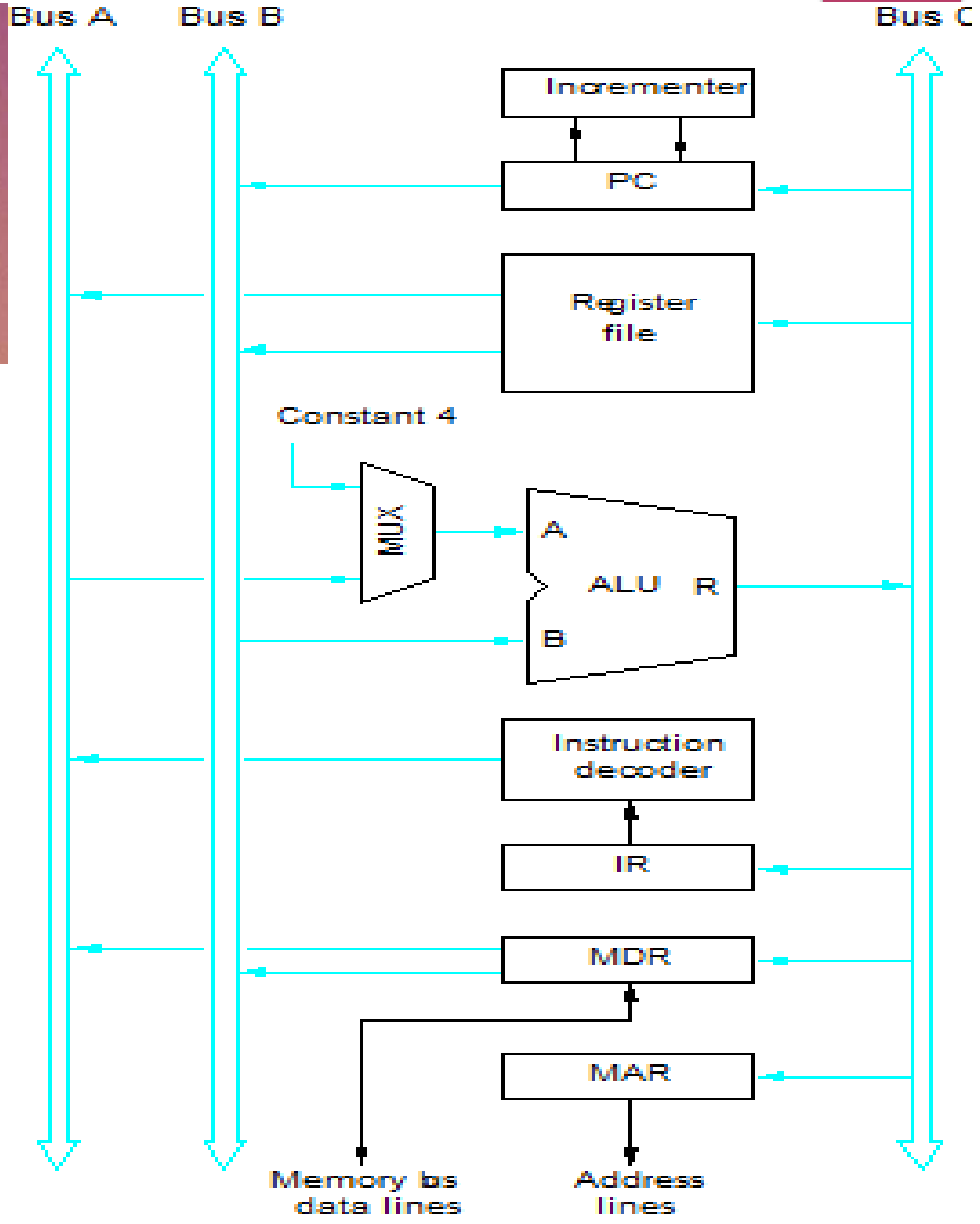
- With a single bus organization, the resulting control sequences are quite long because only one data item can be transferred over the bus in a single clock cycle.
- To reduce the number steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.
- All general-purpose registers are combined into a single block called the register file.
- The register file is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.



- The third port allows the data on the bus C to be loaded into a third register during the same clock cycle.
- Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.
- The result is transferred to the destination over bus C.
- If needed, the ALU may simply pass one of its two input operands unmodified to bus C.
- The ALU control signals for such an operation may be called as R = A or R = B.



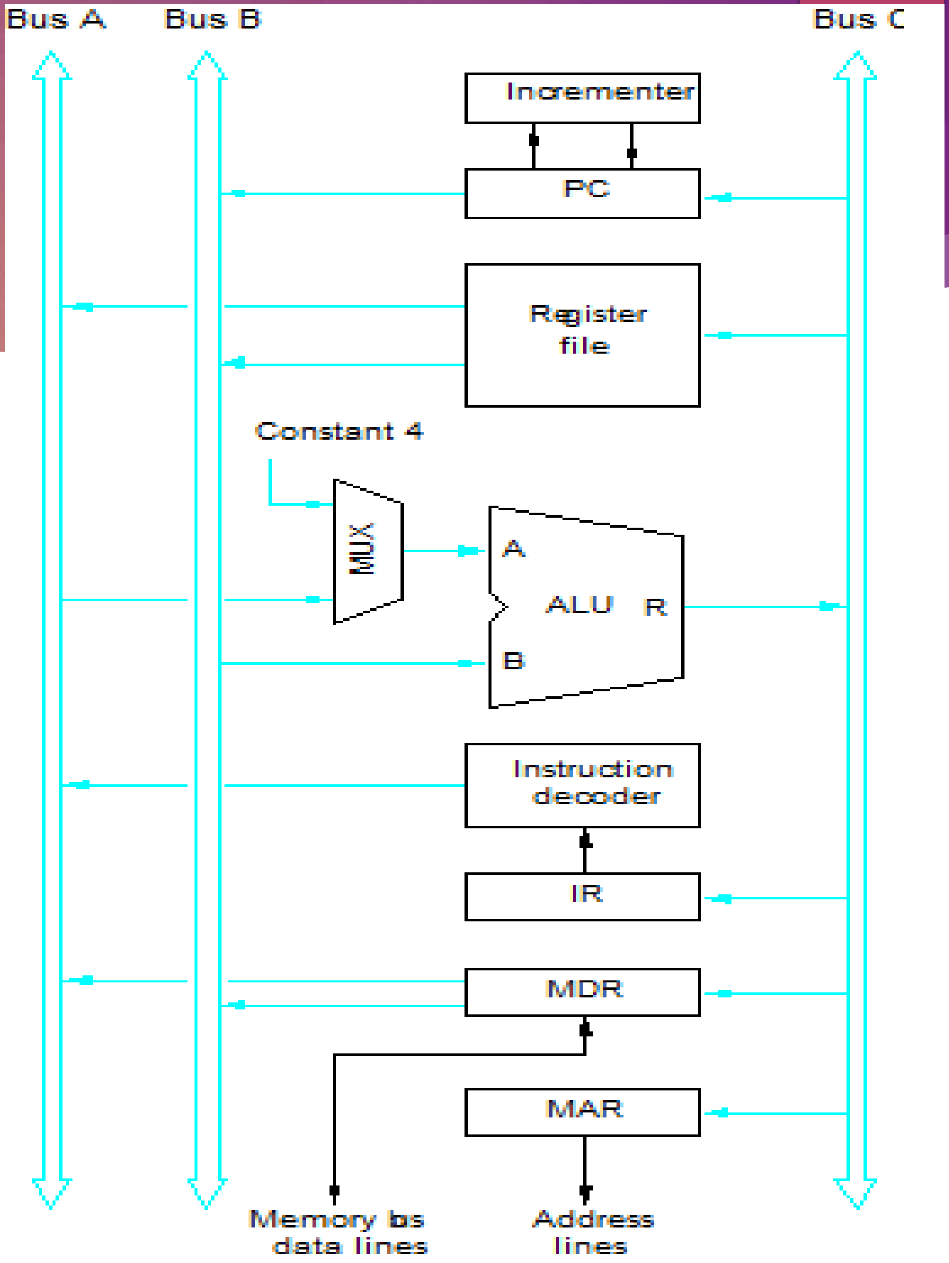
- The three-bus arrangement obviates the need for the registers Y and Z (available in the single-bus organization).
- Another feature is the introduction of the Incrementor unit, which is used to increment the PC by 4.
- Using the Incrementor eliminates the need to add 4 to the PC using the main ALU.
- The source for the constant 4 at the ALU input multiplexer is still useful. It can be used to increment other addresses, such as the memory addresses in LoadMultiple and StoreMultiple instruction.



- Add R6, R5, R4

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	$MDR_{inE,}$ MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Control sequence for the instruction for the three-bus organization



Control Unit

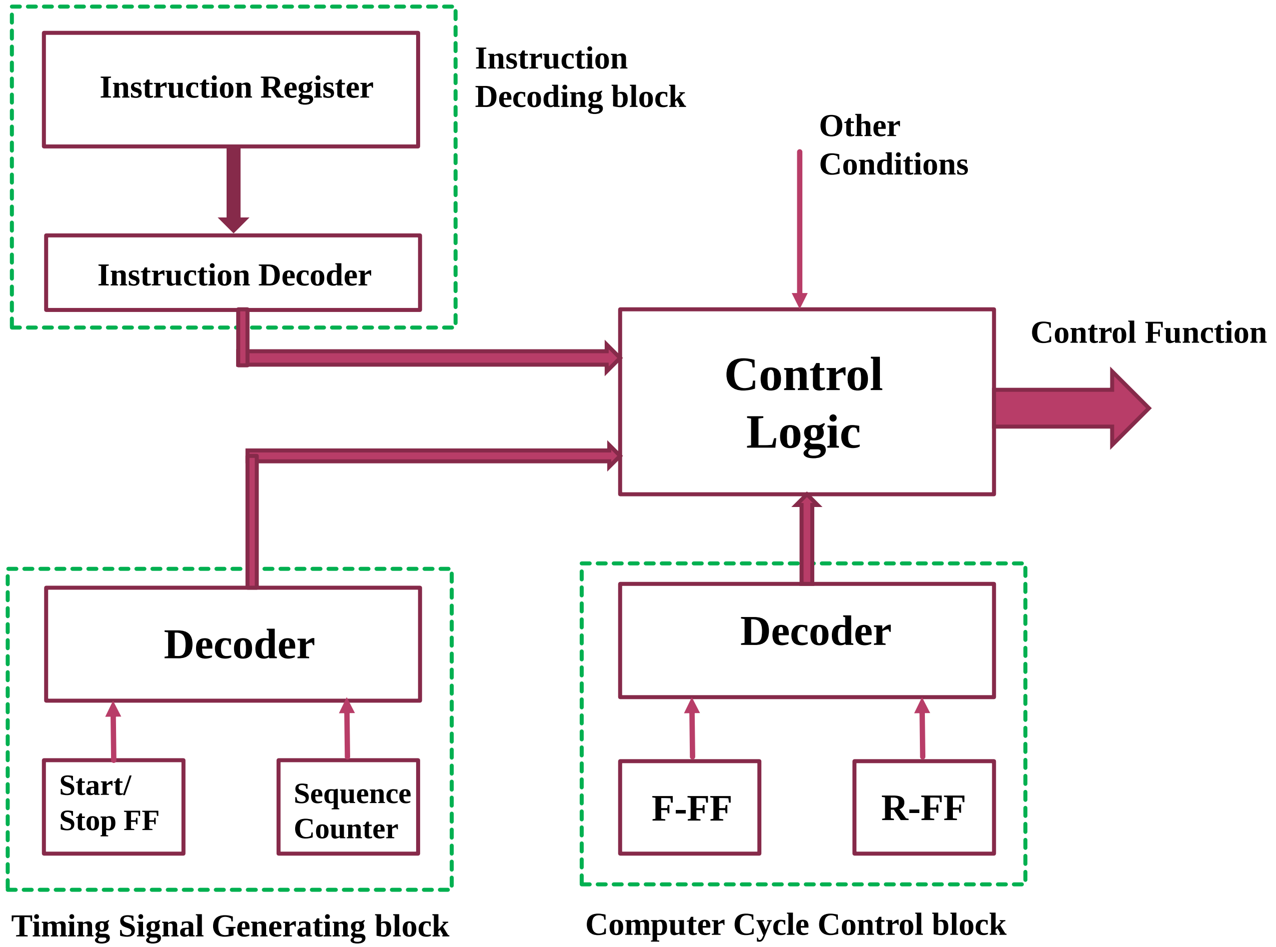
- All the operations in a computer system must be coordinated in some synchronized way, which is the task of control unit.
- This unit effectively the nerve center of the system.
- In concept it is reasonable to think of a control unit as a well defined physically separate unit that interacts with other parts of the system but, in practice much of the control circuitry is physically distributed throughout the machine.

Organization of basic Control Unit

- The organization of basic control unit of a system consists of four blocks.
  - Instruction Decoding block
  - Timing Signal Generating block
  - Computer Cycle Control block
  - Control Logic

Instruction Decoding block

- It consists of an Instruction register and Instruction decoder.
- It decodes the instruction and provides necessary information to the Control Logic.



Timing Signal Generating block

- It is essentially consists of a Sequence Counter and a Start/Stop flip flop.
- Start/Stop flip flop is used to enable the decoder.
- The Decoder provides necessary timing signals to the Control Logic.

Computer Cycle Control block

- This block consists of two flip flops **F** and **R** and a Decoder.
- The computer cycle is determined by this circuit depending on the following Truth Table.

F	R	Cycle
0	0	Fetch Cycle
0	1	Operand Cycle
1	0	Execute Cycle
1	1	Interrupt Cycle

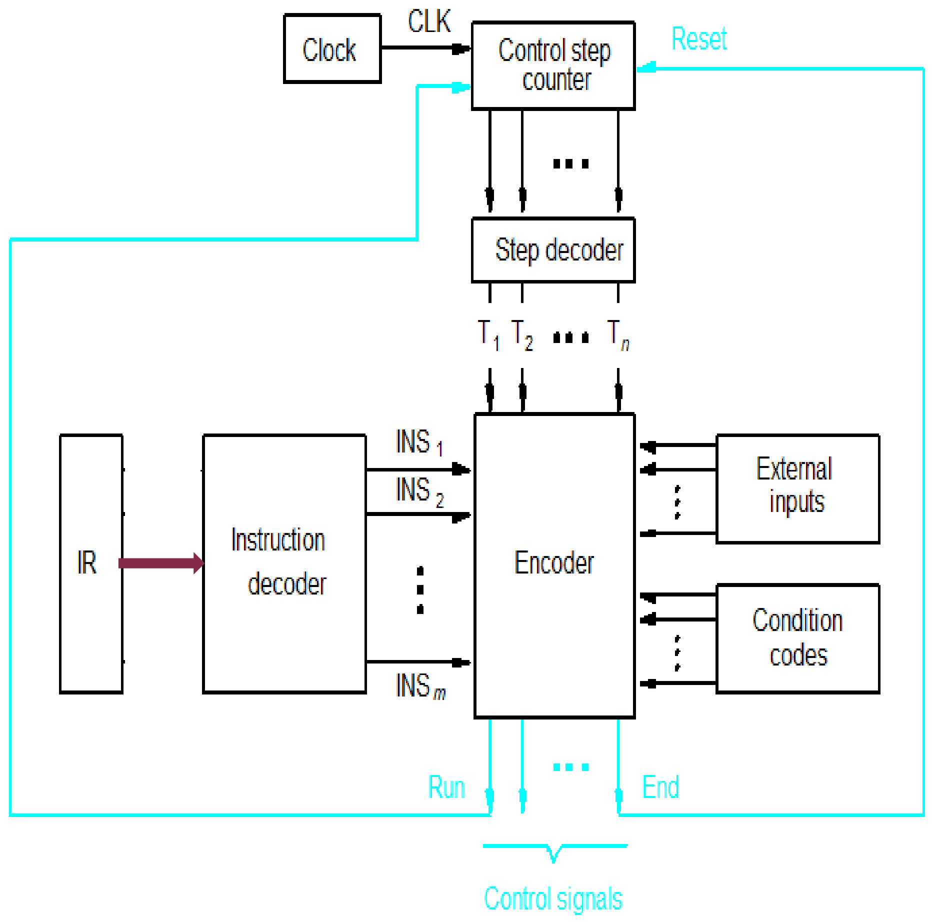
Control Logic

- It is a complex circuit which receives inputs from Instruction decoding block, Timing signal generating block and Computer cycle control block.
- It also takes some control conditions from outside and generates appropriate Control Function or Control word (CW).

- After decoding the instructions the Control unit must have some means of generating control signals needed to complete the operations.
- The computers are designed by implementing a wide variety of technologies to perform this.
- Most of these technologies, however, fall into two categories.
  - Hardwired Control
  - Microprogrammed Control

Hardwired Control Unit

- Each step in sequence is completed in one clock period.
- A counter may be used to keep track of the control steps.
- Each state, or count, of this counter corresponds to one control step.
- The required control signals are determined by the following information.
  - Contents of the instruction register and decoder
  - Contents of the condition code flags
  - External input signals, such as MFC and interrupt requests
  - Signals from Step decoder



- The step decoder provides a separate signal line for each step, or time slot, in the control sequence.
- Similarly, the output of the instruction decoder consists of a separate line for each machine instruction.
- For any instruction loaded in the IR, one of the output lines INS_1 through INS_m is set to 1, and all other lines are set to 0.
- The input signals to the encoder block are combined to generate the individual control signals like Y_{in}, PC_{out}, Add, End, and so on.
- The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes, and the external inputs.
- The outputs of the state machine are the control signals from which all or some of the lines are activated.
- The sequence of operations carried out by this machine is determined by the wiring of the logic elements, hence the name “hardwired”.

Advantages

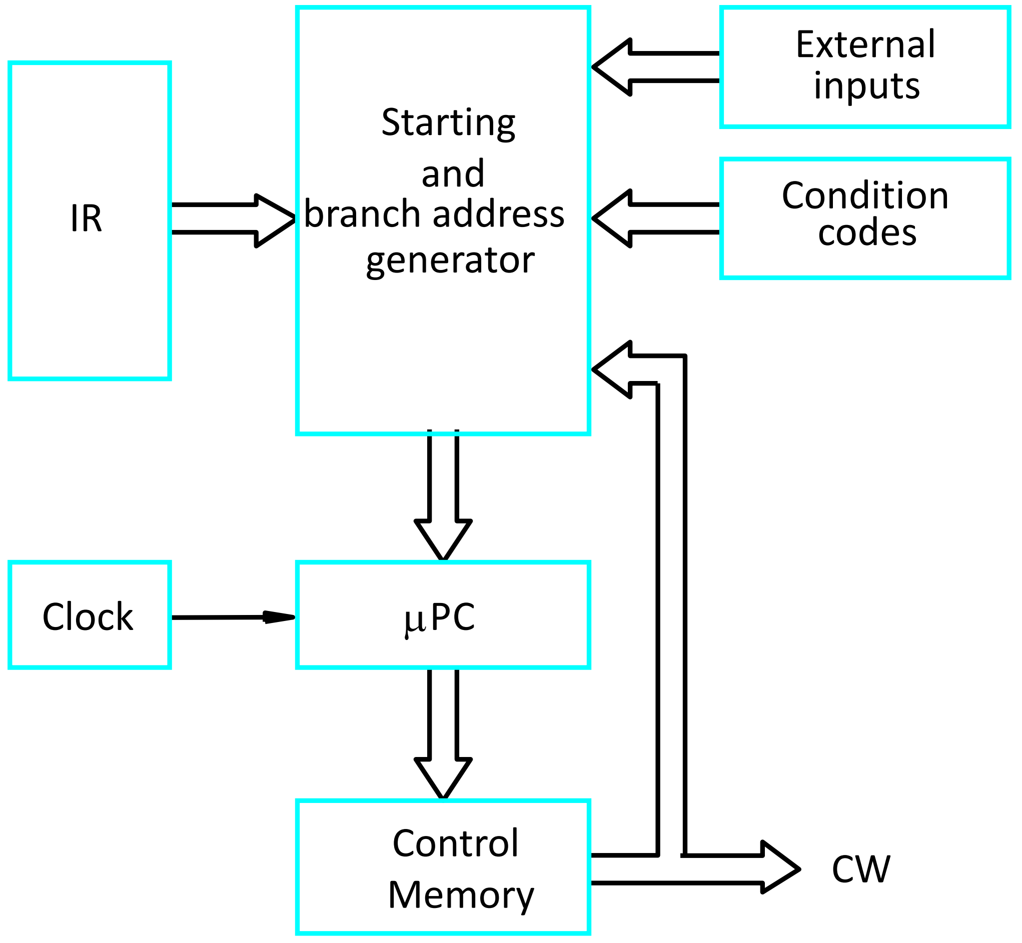
- A hardwired control unit works fast.
- The combinational circuits generate the control signal based on the input signals status. The delay between the output generation to the input availability depends on the number of gates in the path and propagation delay of each gate in the path.

Disadvantages

- If the CPU has a large number of control points, the control unit design becomes very complex.
- The design does not give any flexibility.
- If any modification instruction set is required, it is extremely difficult to make the correction.

Microprogrammed Control Unit

- Microprogramming is a method of Control unit design in which the control signal selection and sequencing information is stored in ROM or RAM, called Control Memory (CM).
- The control signal to be activated at any time are specified by a microinstruction which is fetched from the Control memory.
- Each microinstruction also provides necessary information for microoperation sequencing.
- A set of microinstructions forms Microprogram.
- Microprogram can be changed relatively easily by changing the contents of Control Memory (CM).
- So, Microprogrammed control unit is more flexible than the Hardwired control unit.



- The instruction is loaded to the Instruction Register (IR).
- The address field of the microinstruction contains the next address and that to be used when branching condition is satisfied.
- The Microprogram Counter (µPC) provides the next microinstruction address when no branching is needed.
- The starting address or branch address is generated by taking condition code, external inputs and CW into consideration.
- By decoding the microinstruction appropriate control signals, that is, Control Word (CW) is generated.
- Generally the control memory is ROM, it can not be modified.
- For flexibility now writeable control memory (WCM) is used.
- So, a processor with writeable control memory (WCM) in the control unit is said to be dynamically micro-programmable because the control memory content can be altered.

Advantages

- Less design complexity
- Flexible due to WCM
- A given CPU instruction set can be easily modified by changing the microprogram.

Disadvantages

- A microprogram control unit works slow than Hardwired control unit.

Microprogram

- Program stored in control memory that generates all the control signals required to execute the instruction set correctly
- Consists of microinstructions

Microinstruction

- Contains a control word and a sequencing word

  Control Word - All the control information required for one clock cycle

  Sequencing Word - Information needed to decide the next microinstruction address

Control Memory (Control Storage: CS)

  Storage in the microprogrammed control unit to store the microprogram

Writeable Control Memory (Writeable Control Storage: WCS)

- CS whose contents can be modified

  Allows the microprogram can be changed

  Instruction set can be changed or modified

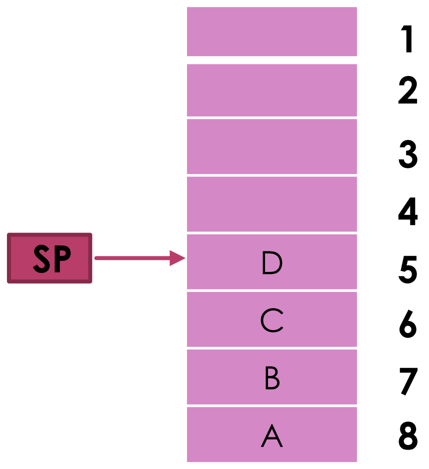
Dynamic Microprogramming

- Computer system whose control unit is implemented with a microprogram in WCS
- Microprogram can be changed by a systems programmer or a user

STACK

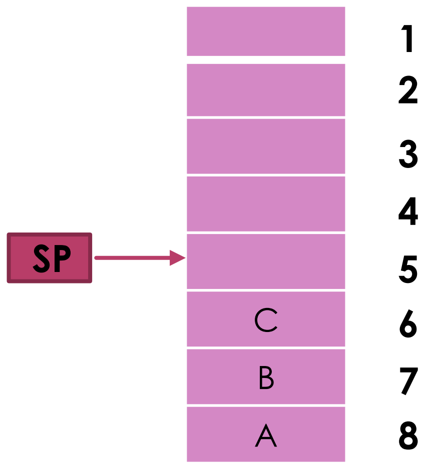
- Stack is a contiguous memory location or collection of finite number of registers defined by the user that stores information.
- If it is contiguous memory locations, called memory stack.
- If it is collection of finite number of registers, called register stack.
- The stack is a reserved area of the memory where we can store temporary information.
- The item stored last into the stack will be retrieved first. So, the operation is called LIFO.
- The register that hold the address for the stack, that is, top of the stack is called Stack pointer (SP).
- The two operations of a stack are the insertion and deletion of item.
- The operation of insertion is called PUSH.
- The operation of deletion is called POP.
- These operations are simulated by incrementing or decrementing the Stack Pointer register.

- A stack may be Full stack or Empty stack.
- It may be Descending stack or Ascending stack.



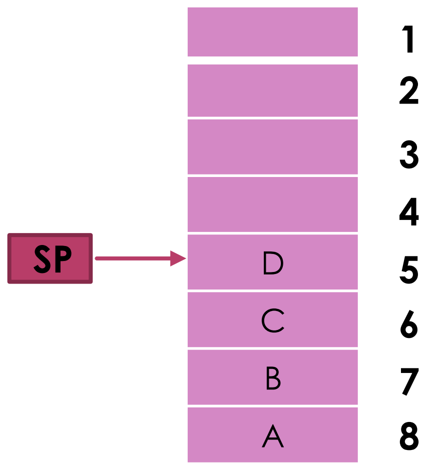
Full Stack

SP ← SP-1
PUSH



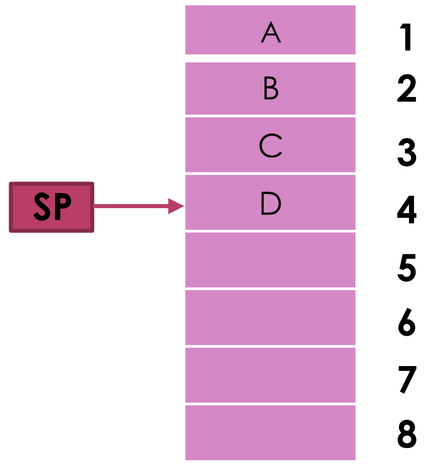
Empty Stack

PUSH
SP ← SP-1



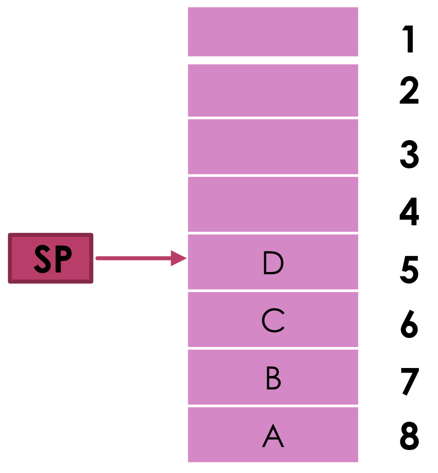
Descending Stack

SP ← SP -1
PUSH



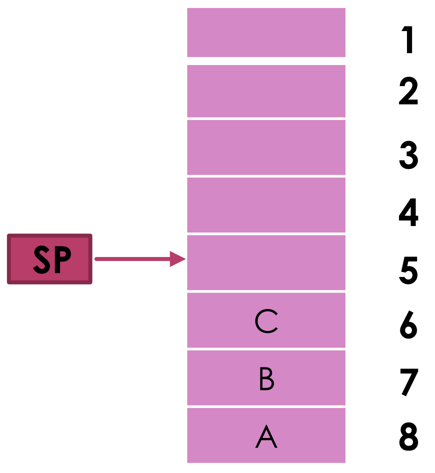
Ascending Stack

SP ← SP +1
PUSH



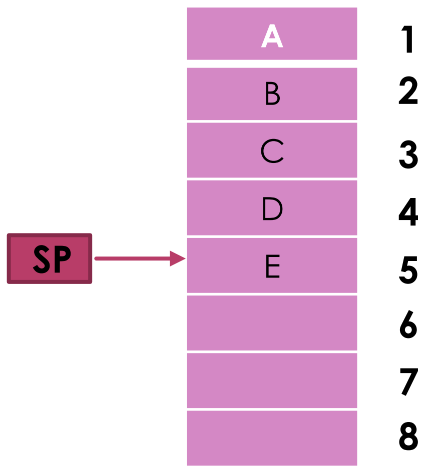
Full Descending Stack

SP ← SP-1
PUSH



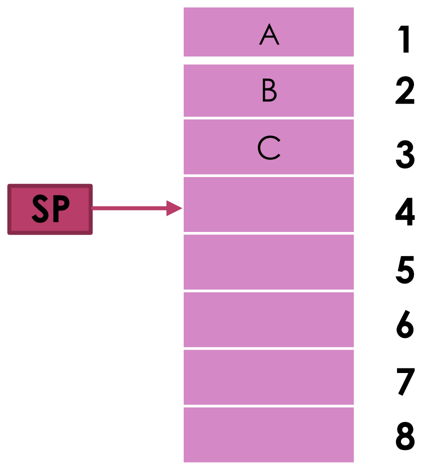
Empty Descending Stack

PUSH
SP ← SP-1



Full Ascending Stack

SP ← SP +1
PUSH



Empty Ascending Stack

PUSH
SP ← SP +1

Polish Notation

- The way to write arithmetic expression is known as **notation**.
- An arithmetic expression can be written in three different but equivalent notations, that is, without changing the essence or output of an expression.
- These notations are –
  - Infix Notation
  - Prefix (Polish) Notation
  - Postfix (Reverse-Polish) Notation
- Infix, Prefix and Postfix notations are three different but equivalent notations of writing algebraic expressions.
- While writing an arithmetic expression using infix notation, the operator is placed between the operands.
- For example, *A+B*; here, plus operator is placed between the two operands A and B.
- Although it is easy to write expressions using infix notation, computers find it difficult to parse as they need a lot of information to evaluate the expression.
- Information is needed about operator precedence, associativity rules, and brackets which overrides these rules.
- So, computers work more efficiently with expressions written using prefix or postfix notations.

Reverse Polish Notation

- Postfix notation was given by Jan Łukasiewicz who was a Polish logician, mathematician, and philosopher.
- His aim was to develop a parenthesis-free prefix notation (also known as Polish notation) and a postfix notation which is known as Reverse Polish Notation or RPN.
- In postfix notation, the operator is placed after the operands. For example, if an expression is written as *A+B* in infix notation, the same expression can be written as *AB+* in postfix notation.
- The order of evaluation of a postfix expression is always from left to right.
- The expression (A + B) * C is written as: AB+C* in the postfix notation.
- A postfix operation does not even follow the rules of operator precedence.
- No parenthesis required.
- The operator which occurs first in the expression is operated first on the operands.
- For example, given a postfix notation AB+C*. While evaluation, addition will be performed prior to multiplication.

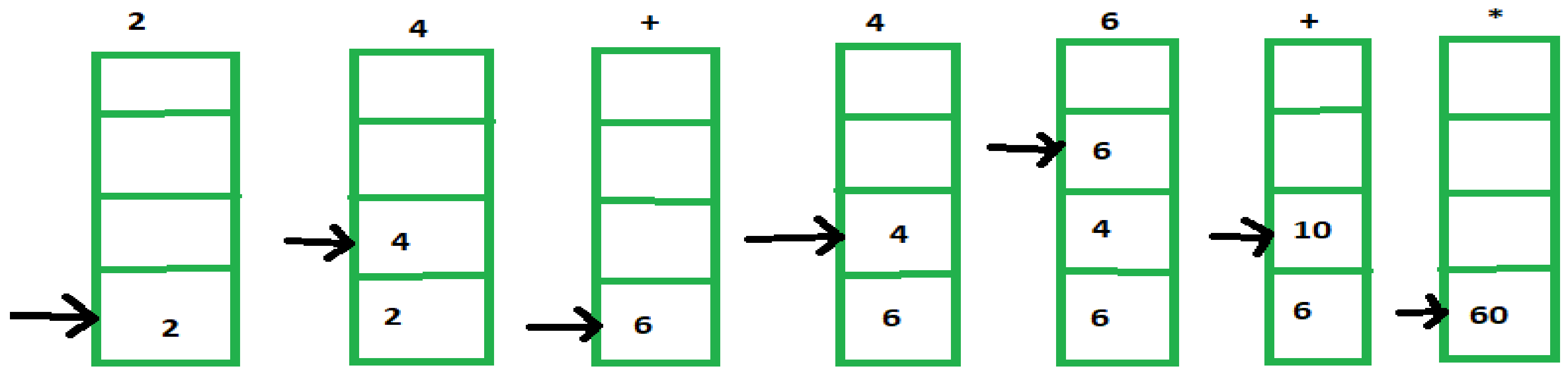
- Now we need to calculate the value of the arithmetic operations by using stack.
- The procedure for getting the result is:

1. Convert the expression in Reverse Polish notation (post-fix notation).
2. Push the operands into the stack in the order they appear.
3. When any operator encounter then pop two topmost operands for executing the operation.
4. After execution push the result into the stack.
5. After the complete execution of expression the final result remains on the top of the stack.

Evaluation of Arithmetic Expression

Example

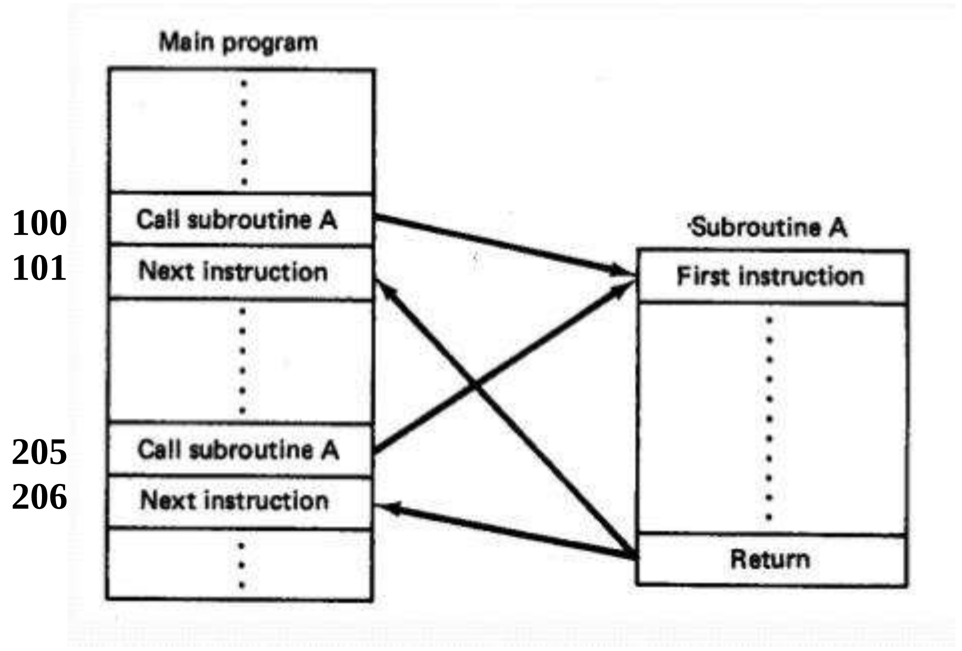
- Infix notation: (2+4) * (4+6)
- Post-fix notation: 2 4 + 4 6 + *
- Result: 60
- The stack operations for this expression evaluation is shown below:

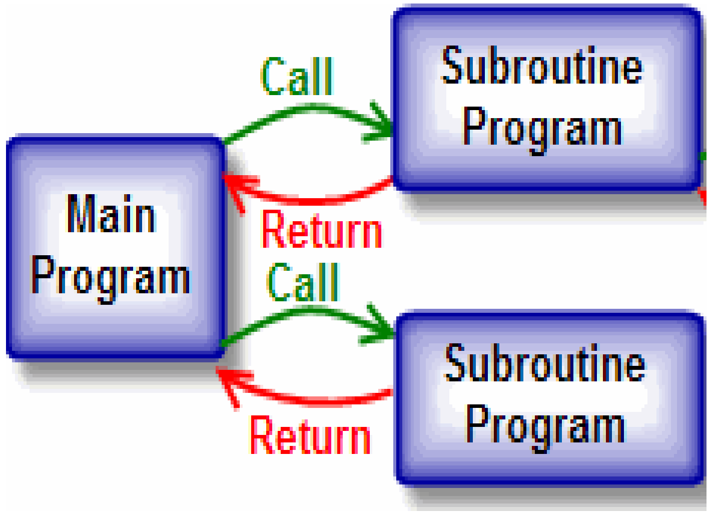


Stack operations to evaluate (2+4)*(4+6)

Subroutine

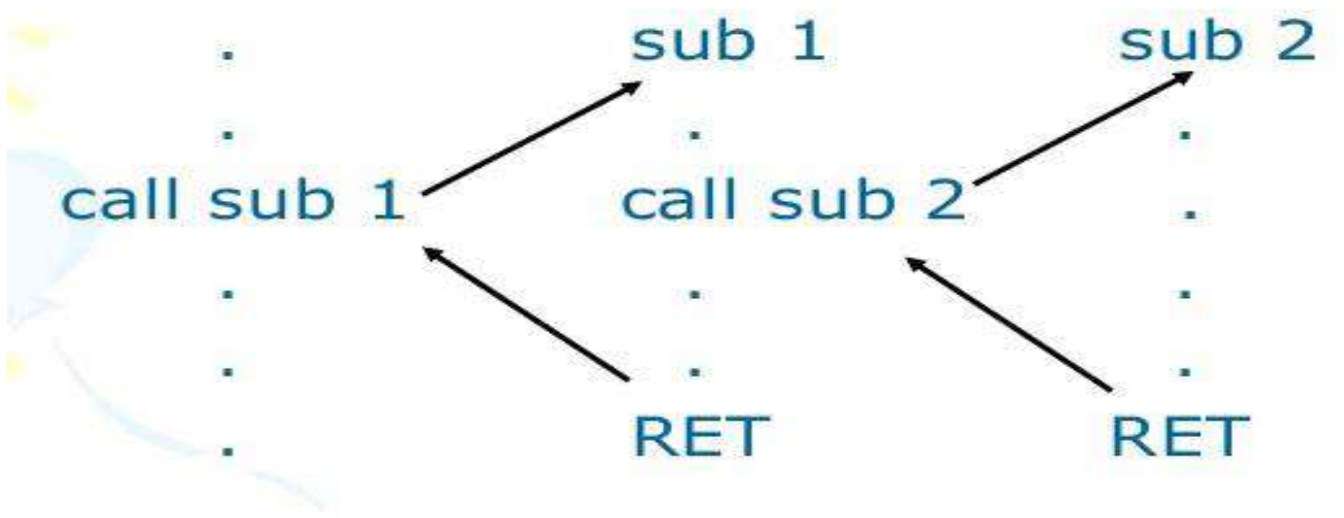
- In programming, a **subroutine** is a sequence of instructions that performs a specific task, packaged as a unit.
- This unit can then be used in programs wherever that particular task should be performed.
- Subroutines may be defined within programs, or separately in libraries that can be used by many programs.
- In different programming languages, a subroutine may be called a **routine**, **subprogram**, **function**, **method**, or **procedure**.
- So, a subroutine is a group of instructions that will be used repeatedly in different locations of the program.
- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.


Main program
100
101
205
206
Call subroutine A
Next instruction
Call subroutine A
Next instruction
Subroutine A
First instruction
Return

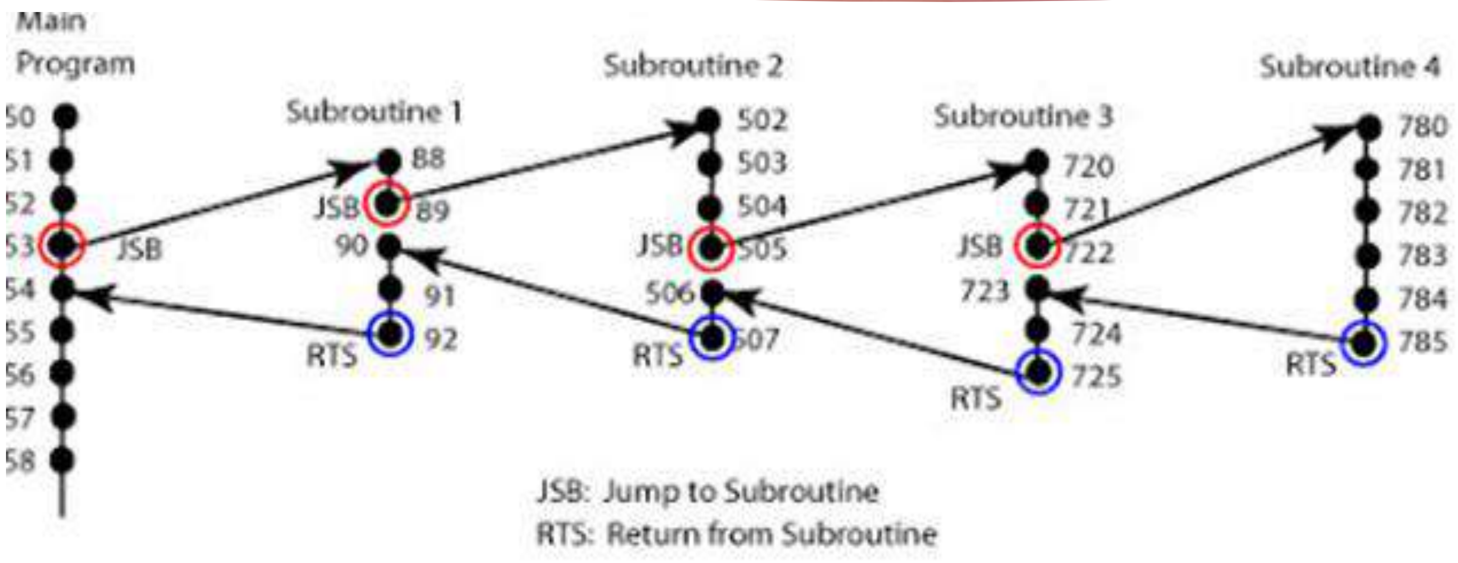

Main Program
Call
Subroutine Program
Return
Call
Subroutine Program
Return

Nested Subroutine

- A nested subroutine is a subroutine that is called from other subroutine.
- Stack operations can be very useful at subroutine entry and exit to avoid losing register contents if other subroutines are called.
- At the start of a subroutine, content of PC required to be stored on the stack, and at exit they can be popped off again.



Example



			723
		506	506
	90	90	90
54	54	54	54