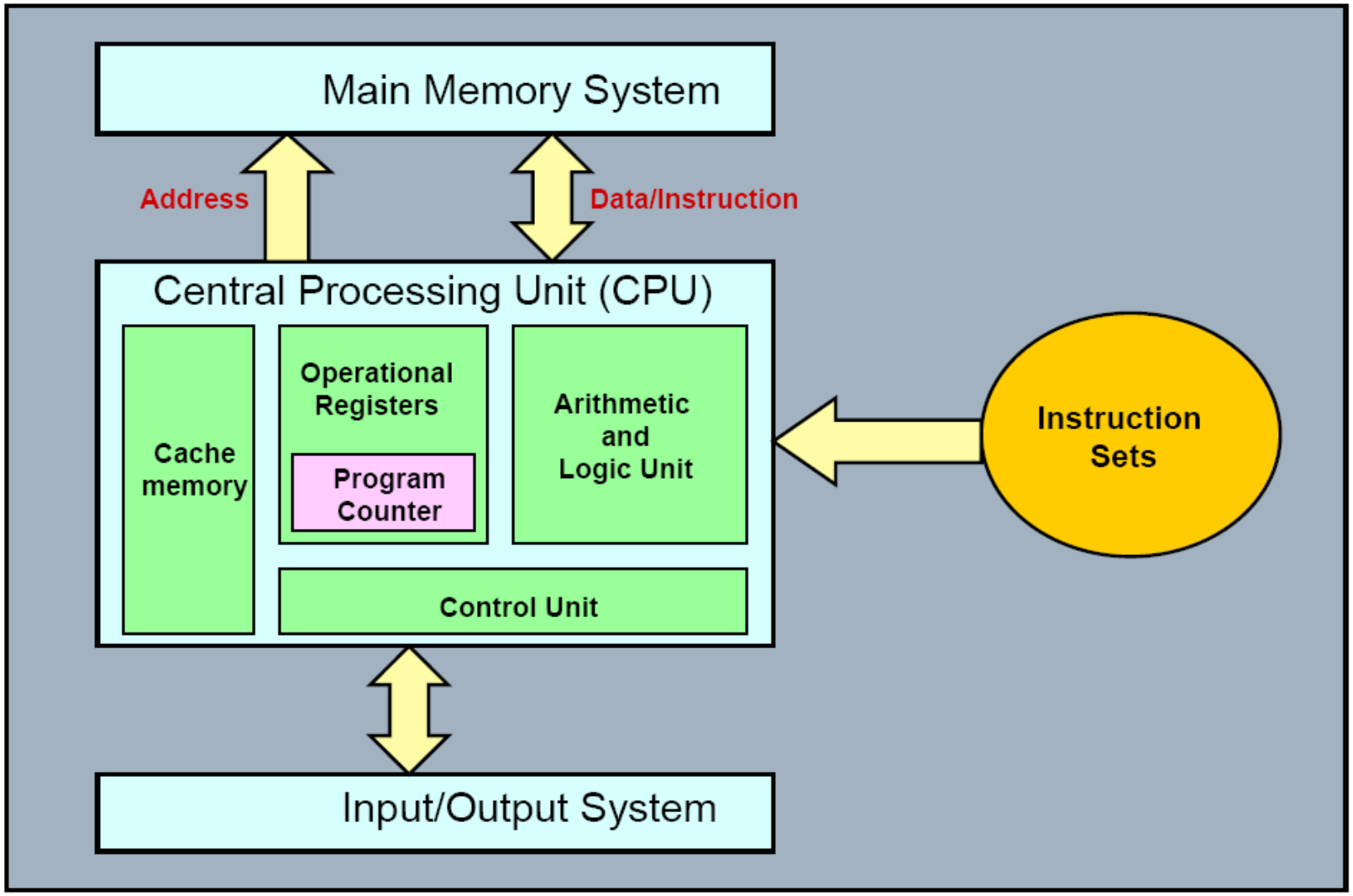
Computer Organization and Architecture
CST 202

Module - I Part - 1
Basic Structure of Computers

Structure of a Computer



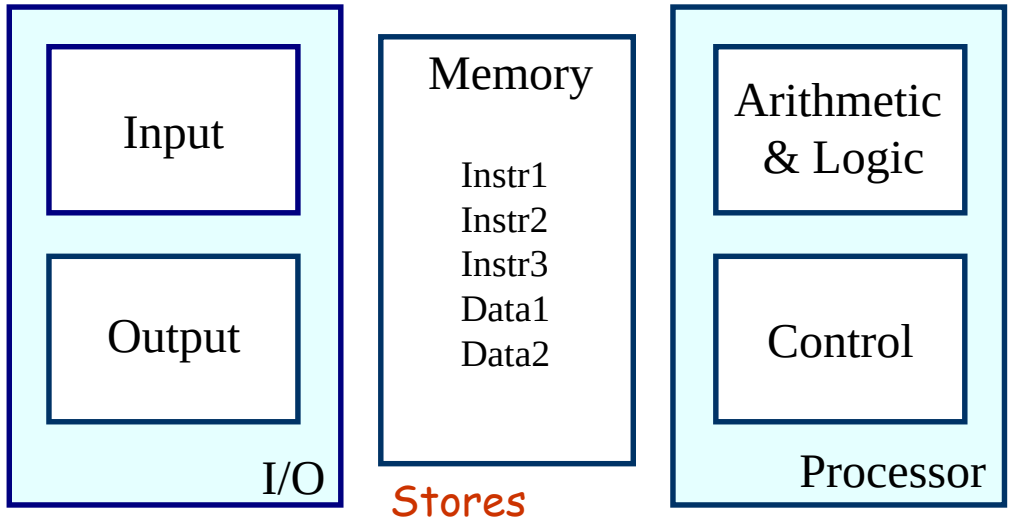
What is a computer?

- a computer is a sophisticated electronic calculating machine that:
  - Accepts input information,
  - Processes the information according to a list of internally stored instructions and
  - Produces the resulting output information.
- Functions performed by a computer are:
  - Accepting information to be processed as input.
  - Storing a list of instructions to process the information.
  - Processing the information according to the list of instructions.
  - Providing the results of the processing as output.
- What are the functional units of a computer?

Functional units of a computer

Input unit accepts information:
- Human operators,
- Electromechanical devices (keyboard)
- Other computers

Arithmetic and logic unit(ALU):
- Performs the desired operations on the input information as determined by instructions in the memory



Output unit sends results of processing:
- To a monitor display,
- To a printer

Stores information:
- Instructions,
- Data

Control unit coordinates various actions
- Input,
- Output
- Processing

Information in a computer -- ***Instructions***

- Instructions specify commands to:
  - Transfer information within a computer (e.g., from memory to ALU)
  - Transfer of information between the computer and I/O devices (e.g., from keyboard to computer, or computer to printer)
  - Perform arithmetic and logic operations (e.g., Add two numbers, Perform a logical AND).
- A sequence of instructions to perform a task is called a program, which is stored in the memory.
- Processor fetches instructions that make up a program from the memory and performs the operations stated in those instructions.
- What do the instructions operate upon?

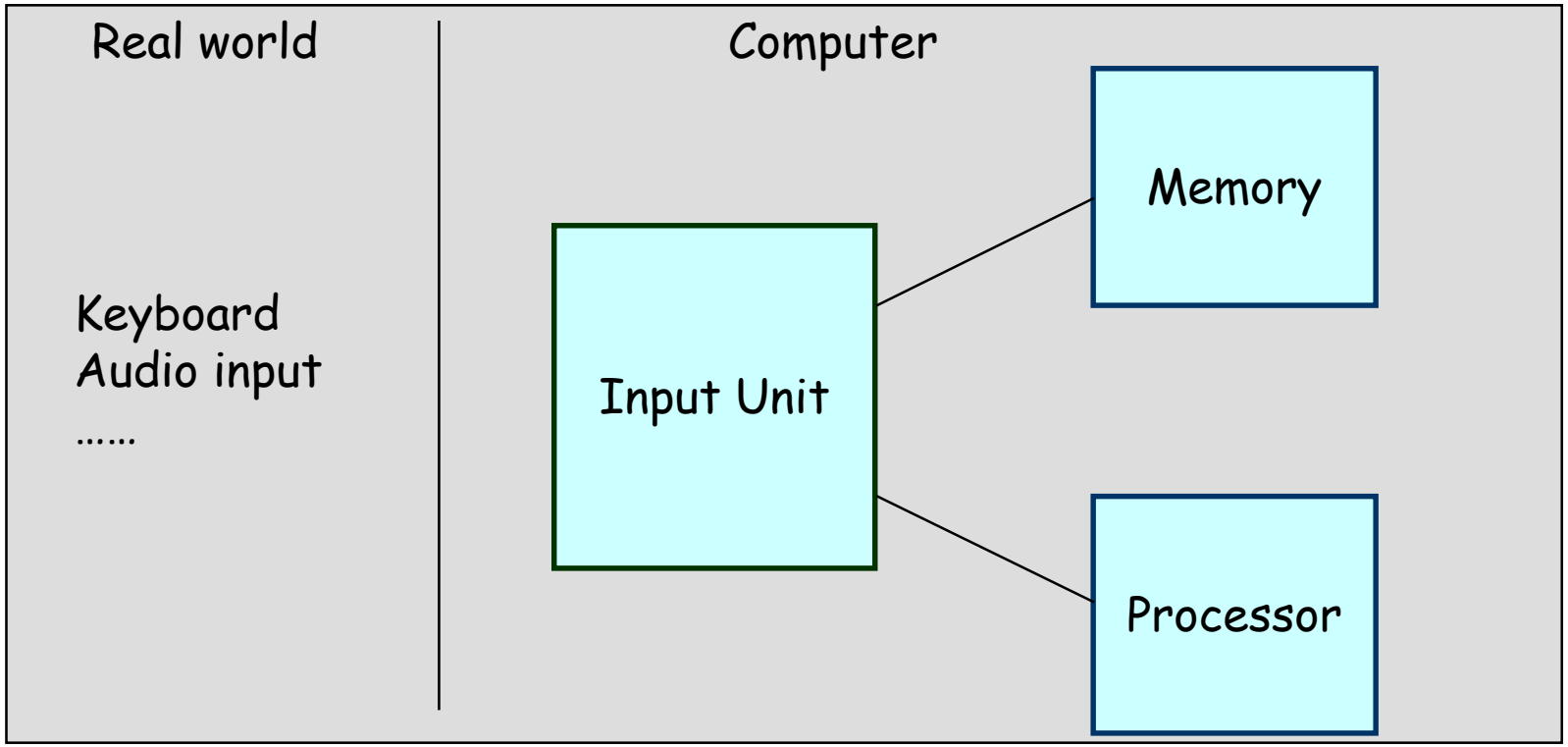
Information in a computer -- **Data**

- Data are the "operands" upon which instructions operate.
- Data could be:
  - Numbers,
  - Encoded characters.
- Data, in a broad sense means any digital information.
- Computers use data that is encoded as a string of binary digits called bits.

Input unit

Binary information must be presented to a computer in a specific format. This task is performed by the input unit: When a key is pressed, the corresponding digit or letter is automatically translated into its binary code and transmitted over a cable either to memory or the processor

- Interfaces with input devices.
- Accepts binary information from the input devices.
- Presents this binary information in a format expected by the computer.
- Transfers this information to the memory or processor.



Memory unit

- Memory unit stores instructions and data.
  - Recall, data is represented as a series of bits.
  - To store data, memory unit thus stores bits.
- Processor reads instructions and reads/writes data from/to the memory during the execution of a program.
  - In theory, instructions and data could be fetched one bit at a time.
  - In practice, a group of bits is fetched at a time.
  - Group of bits stored or retrieved at a time is termed as "word"
  - Number of bits in a word is termed as the "word length" of a computer.
- In order to read/write to and from memory, a processor should know where to look:
  - "Address" is associated with each word location.

Memory unit (contd..)

- Processor reads/writes to/from memory based on the memory address:
  - Access any word location in a short and fixed amount of time based on the address.
  - Random Access Memory (RAM) provides fixed access time independent of the location of the word.
  - Access time is known as "Memory Access Time".
- Memory and processor have to "communicate" with each other in order to read/write information.
  - In order to reduce "communication time", a small amount of RAM (known as Cache) is tightly coupled with the processor.
- Modern computers have three to four levels of RAM units with different speeds and sizes:
  - Fastest, smallest known as Cache
  - Slowest, largest known as Main memory.

Memory unit (contd..)

- Primary storage of the computer consists of RAM units.
  - Fastest, smallest unit is Cache.
  - Slowest, largest unit is Main Memory.
- Primary storage is insufficient to store large amounts of data and programs.
  - Primary storage can be added, but it is expensive.
- Store large amounts of data on secondary storage devices:
  - Magnetic disks and tapes,
  - Optical disks (CD-ROMS).
  - Access to the data stored in secondary storage in slower, but take advantage of the fact that some information may be accessed infrequently.
- Cost of a memory unit depends on its access time, lesser access time implies higher cost.

Memory Unit

- Store programs and data
- Two classes of storage

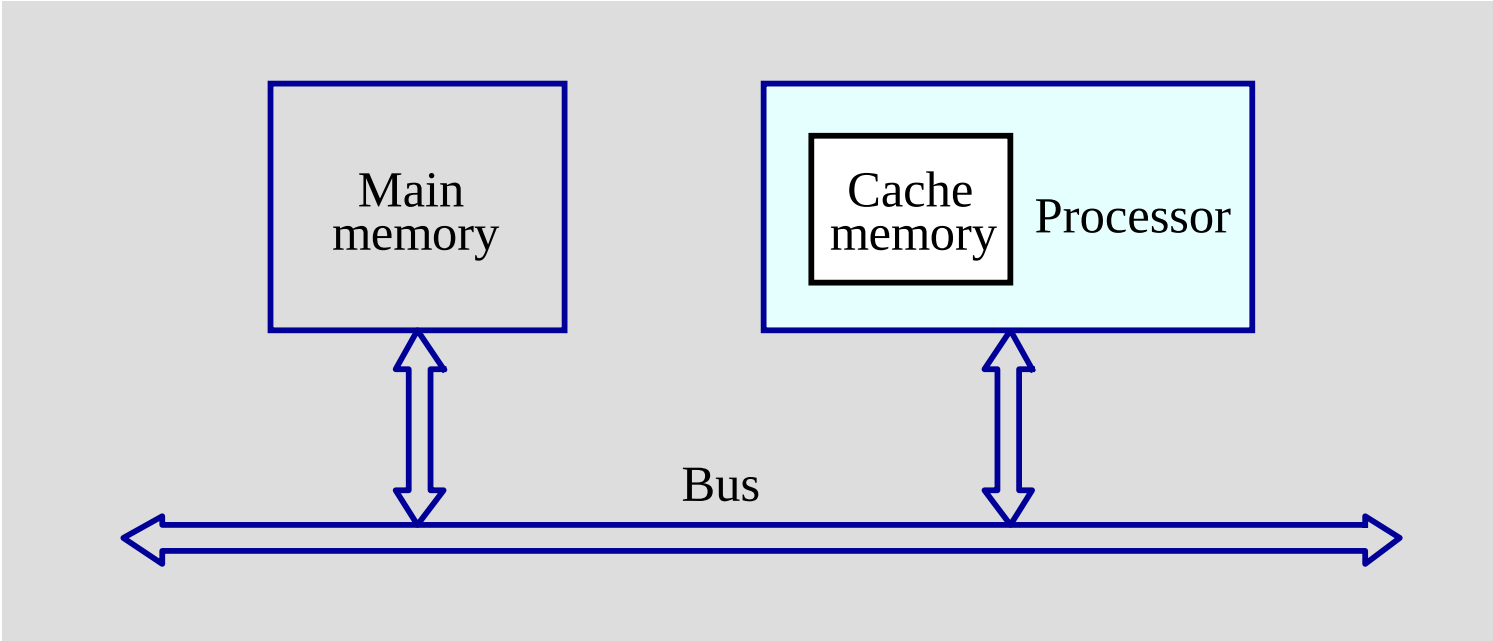
➢ Primary storage

  - Fast
  - Programs must be stored in memory while they are being executed
  - Large number of semiconductor storage cells
  - Processed in words
  - Address
  - RAM and memory access time
  - Memory hierarchy – cache, main memory

➢ Secondary storage

  - larger and cheaper

Organization of cache and main memory



Why is the access time of the cache memory lesser than the access time of the main memory?

Arithmetic and logic unit (ALU)

- Operations are executed in the Arithmetic and Logic Unit (ALU).
  - Arithmetic operations such as addition, subtraction.
  - Logic operations such as comparison of numbers.
- In order to execute an instruction, operands need to be brought into the ALU from the memory.
  - Operands are stored in general purpose registers available in the ALU.
  - Access times of general purpose registers are faster than the cache.
- Results of the operations are stored back in the memory or retained in the processor for immediate use.

Computer Organization and Architecture
CST 202

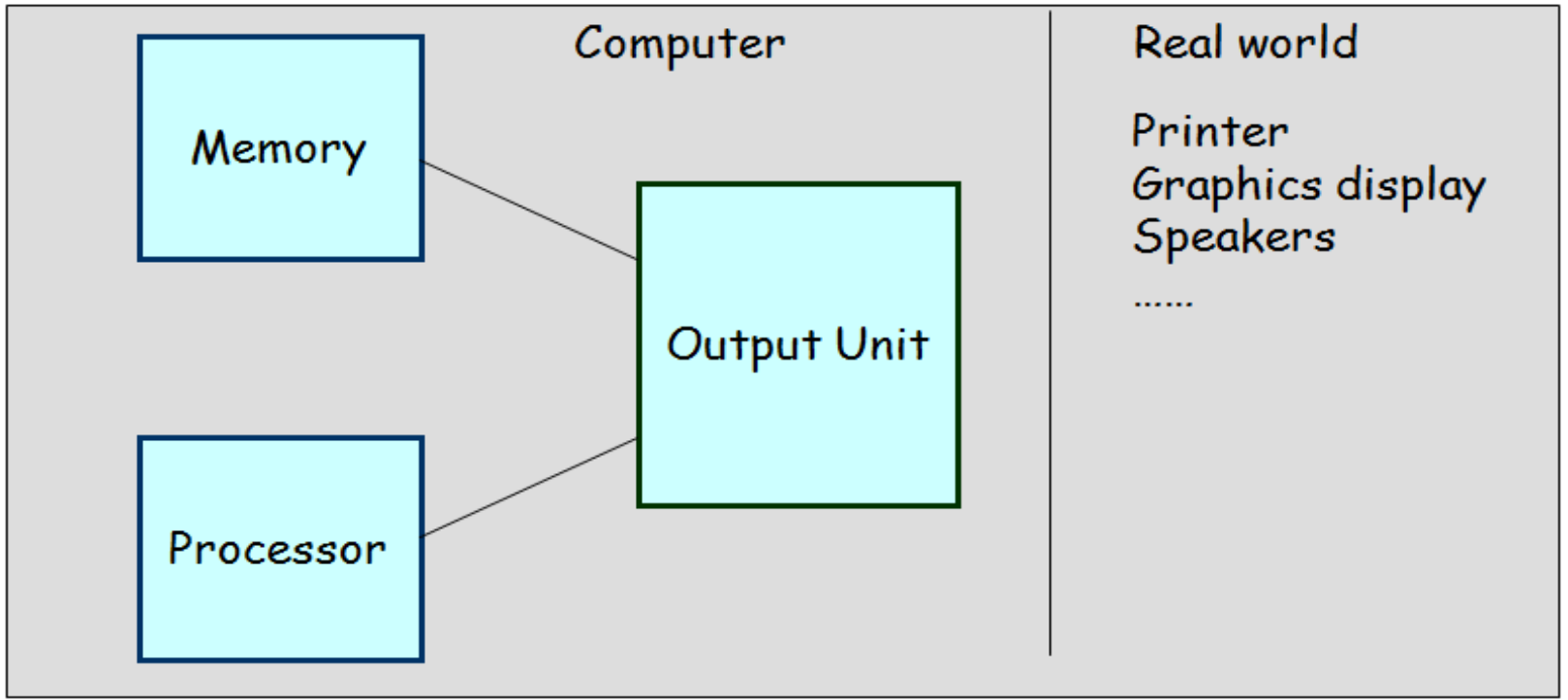
Module - I Part - 1
Basic Structure of Computers
Lecture 3

Output unit

•Computers represent information in a specific binary form.

Output units:

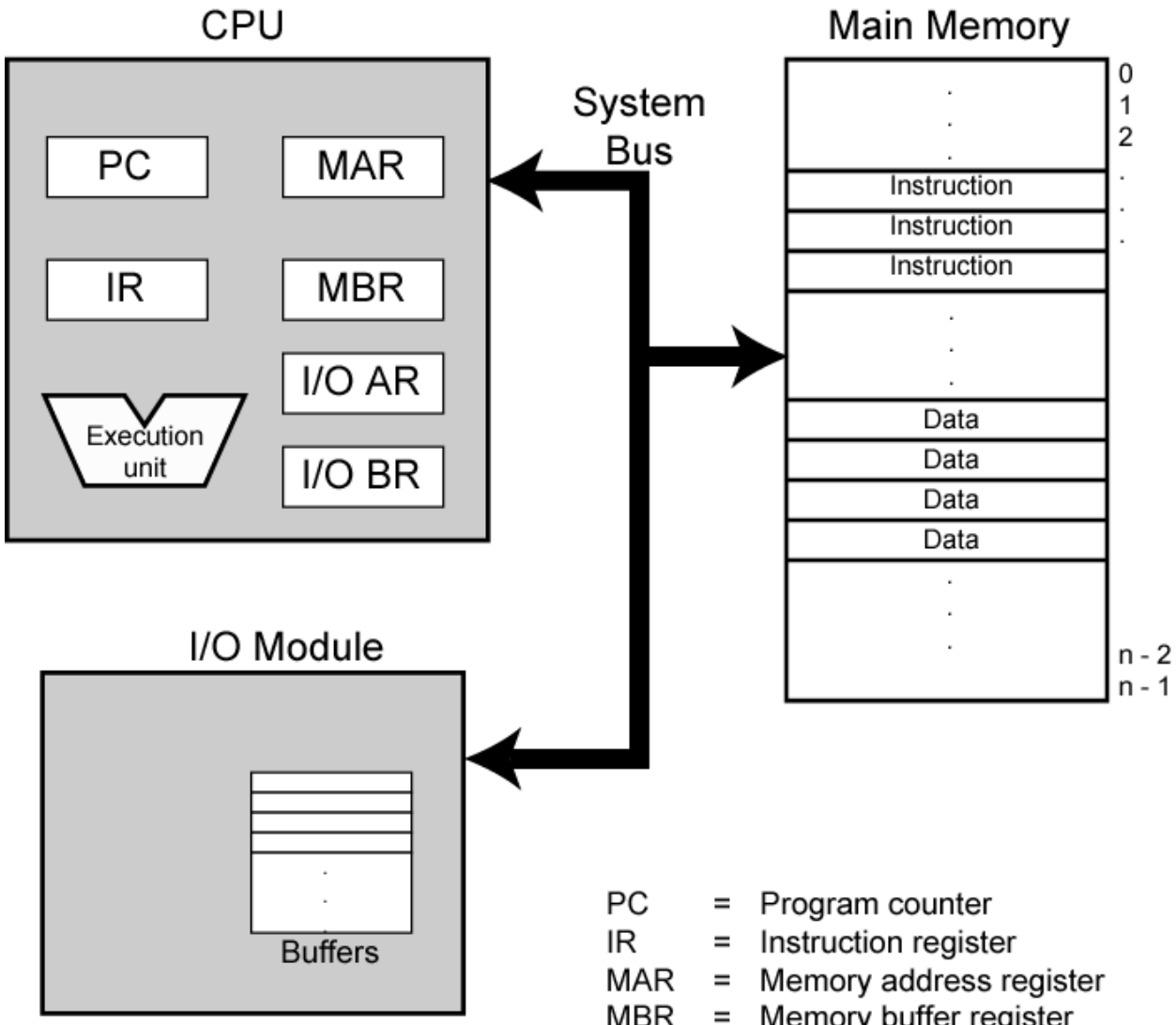
- Interface with output devices.
- Accept processed results provided by the computer in specific binary form.
- Convert the information in binary form to a form understood by an output device.



Control unit

- Operation of a computer can be summarized as:
  - Accepts information from the input units (Input unit).
  - Stores the information (Memory).
  - Processes the information (ALU).
  - Provides processed results through the output units (Output unit).
- Operations of Input unit, Memory, ALU and Output unit are coordinated by Control unit.
- Instructions control "what" operations take place (e.g. data transfer, processing).
- Control unit generates timing signals which determines "when" a particular operation takes place.

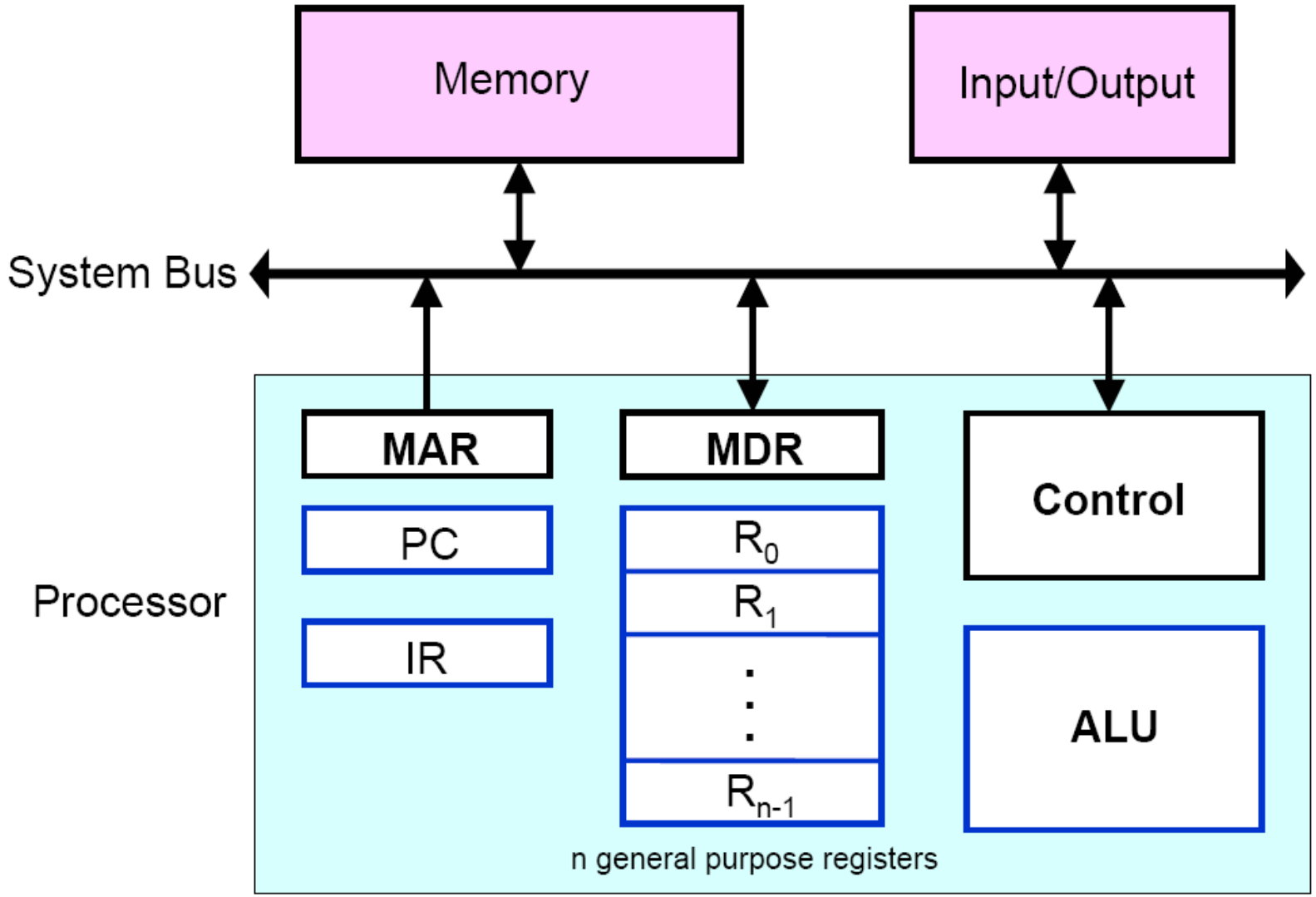
Computer Components: Top Level View



PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

Basic Operational Concepts

Connection Between the Processor and Memory



Typical Operating Steps (Fetch Cycle)

- Programs reside in the memory, entered through input devices
- PC (Program counter) has address of the first instruction or next instruction to be executed
- The contents of PC are transferred to MAR (Memory Address Register) and send over system bus to memory
- A Read signal is sent to the memory by control unit
- The instruction (addressed word) is fetched and loaded into MDR (Memory Data Register) after duration of memory access time
- The contents of MDR transferred to IR (Instruction Register)
- Now the instruction is ready to be decoded and executed

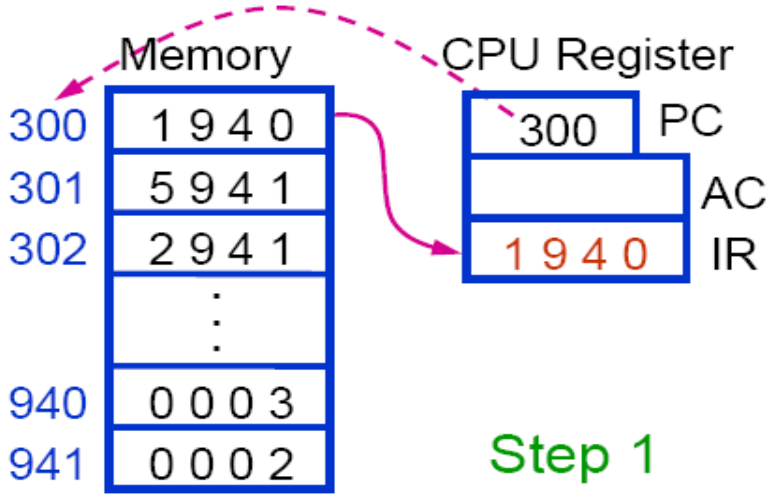
Typical Operating Steps (Execution Cycle)

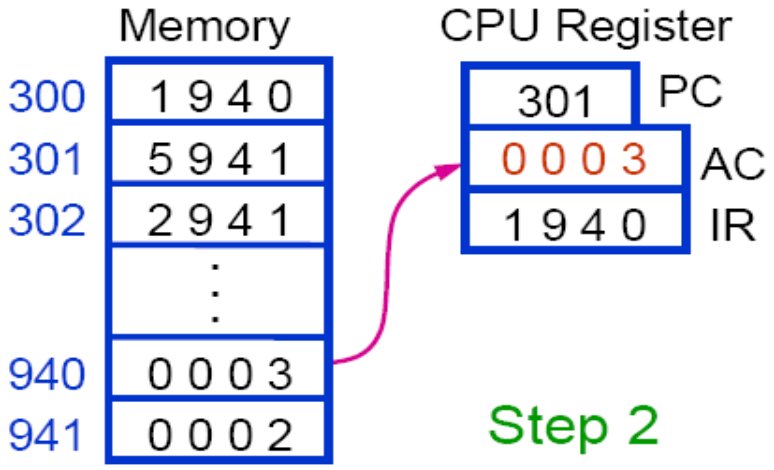
- Get operands for ALU from:
  - General-purpose register (R0, … Rn-1)
  - Memory
    - (address to MAR, Read frm mem, data to MDR, to ALU)
- Perform operation in ALU
- Store the result back to:
  - General-purpose register (R0, … Rn-1)
  - Memory
    - (address to MAR, result to MDR, Write to mem)
- During the execution, PC is incremented to point to the next instruction
- In addition to transferring data between memory and processor, computer accepts data from input devices and sends data to output devices. Instructions to handle I/O transfers, I/O Address Register, I/O Data Register, etc provided.

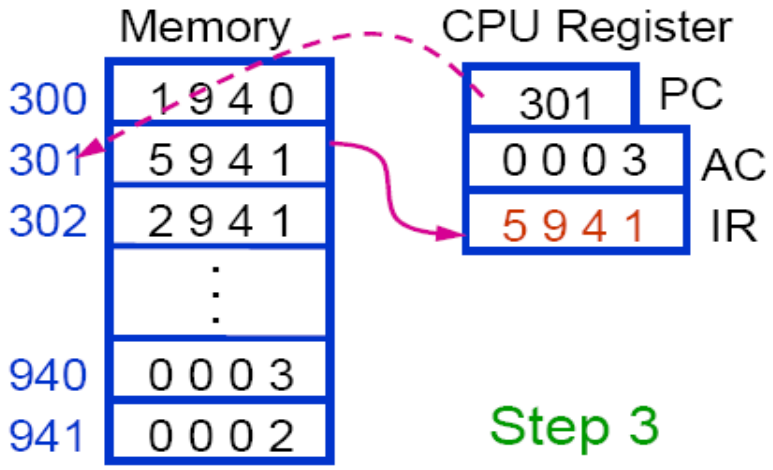
A Partial Program Execution Example

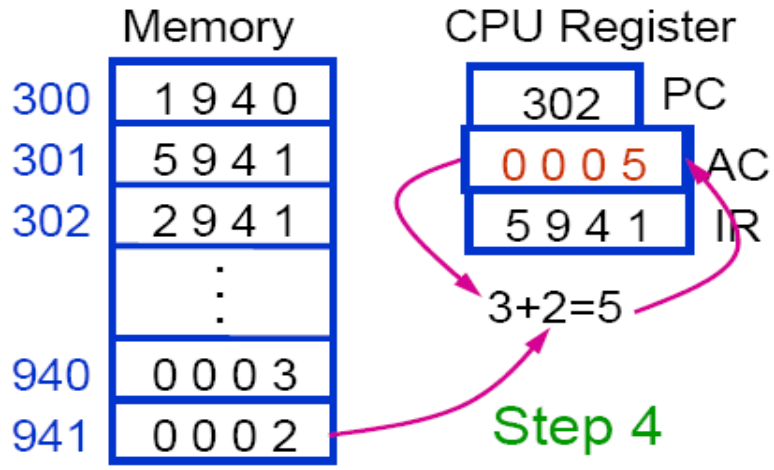
Ex : Load 940 , Add 941 , Store 941

opcodes Load-1 , Add-5 , Store-2





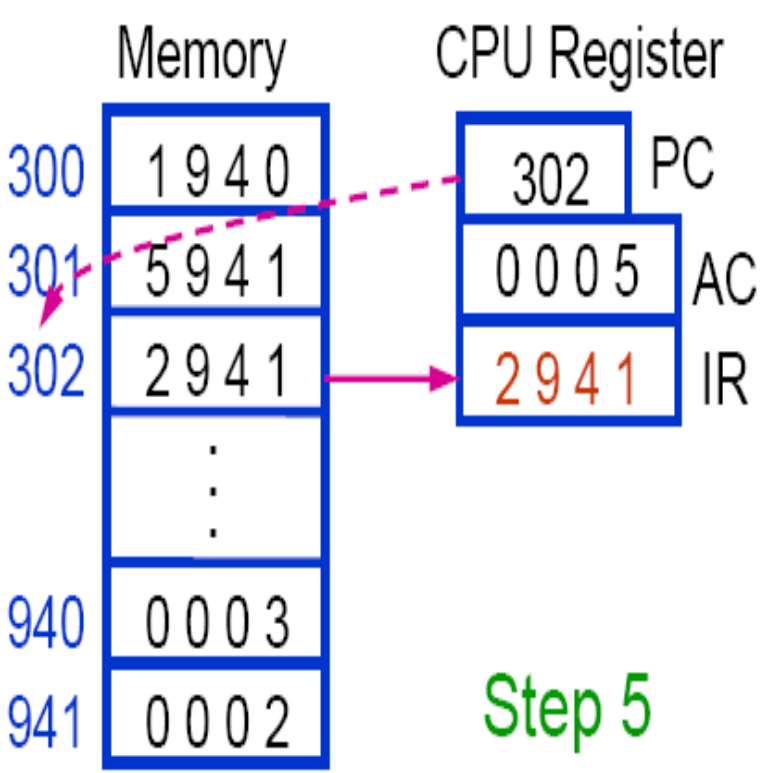


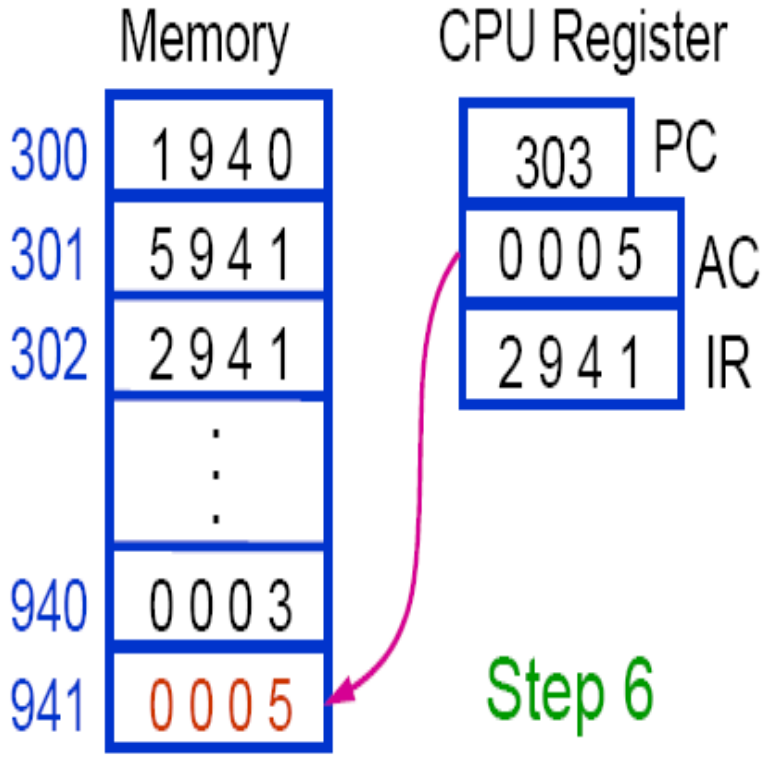


A Partial Program Execution Example

Ex : Load 940 , Add 941 , Store 941

opcodes Load-1 , Add-5 , Store-2





Computer Organization and Architecture
CST 202

Module - I Part - 1
Basic Structure of Computers

Interrupt

- Normal execution of programs may be interrupted if some device requires urgent servicing
- (eg : a monitoring device in a computer -controlled industrial process may detect a hazardous situation)
  - To deal with the situation immediately, the normal execution of the current program must be interrupted
    - An interrupt is a request from an I/O device for service by the processor
- Procedure of interrupt operation
  - The device raises an interrupt signal
  - The processor provides the requested service by executing an appropriate interrupt-service routine (ISR)
  - The state of the processor is first saved before servicing the interrupt . (Normally, the contents of the PC, the general registers, and some control information and specific information are stored in memory)
  - When the interrupt-service routine is completed, the state of the processor is restored so that the interrupted program may continue.

Types of Interrupts

Three major types of interrupts that cause a break in the normal execution of a program

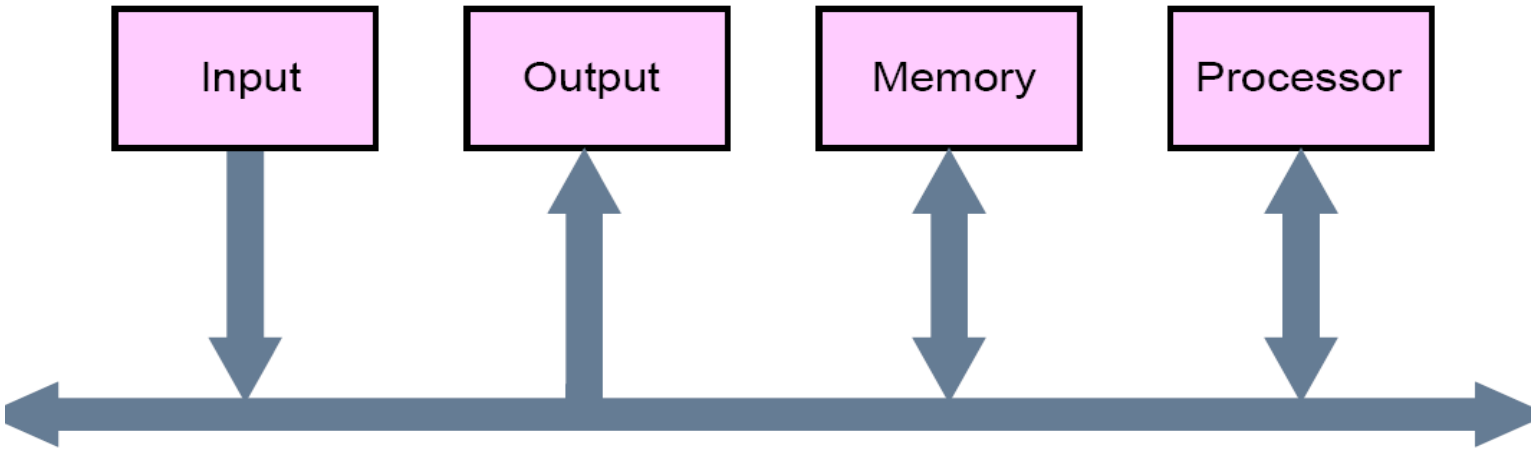
- External interrupts - come from input-output (I/O) devices
  Eg:- from a timing device, a circuit monitoring the power supply, or any other external source.
- Internal interrupts (traps) - arise from illegal or erroneous use of an instruction or data.
  Eg:- interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, protection violation and stack overflow.
- Software interrupt - is initiated by executing an instruction.
  - It is a special call instruction that behaves like an interrupt rather than a subroutine call.
  - It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

Classes of Interrupts

- Program
  - Generated by some condition that occurs as a result of an instruction execution such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space
- Timer
  - Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis
- I/O
  - Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions
- Hardware failure
  - Generated by a failure such as power failure or memory parity error

Bus Structures

- For a computer to achieve its operation, the functional units need to communicate with each other.
- In order to communicate, they need to be connected.



- Functional units may be connected by a group of parallel wires.
- The group of parallel wires is called a ***bus***.
- Each wire in a bus can transfer one bit of information.
- The number of parallel wires in a bus is equal to the word length of a computer

Bus Structures

- A group of lines that serves a connecting path for several devices is called a bus
  - In addition to the lines that carry the data, the bus must have lines for address and control purposes
  - The simplest way to interconnect functional units is to use a single bus.
  - Single bus is cheaper and flexible to attach peripherals
  - All units are connected to a bus, since bus can be used for only one transfer at a time, only two devices can actively use it at any given time
  - Bus control lines are used to arbitrate multiple requests for bus
- Multiple buses have concurrency in operations, better performance, but expensive.

s

Drawbacks of the Single Bus Structure

- The devices connected to a bus vary widely in their speed of operation
  - Devices relatively slow (such as printer and keyboard)
  - Devices considerably fast (such as optical disks)
  - fastest parts (Memory and processor units)
- Efficient transfer mechanism is needed to cope with this problem:
  - A common approach is to include buffer registers with the devices to hold the information during transfers
  - An another approach is to use two-bus structure and an additional transfer mechanism
    - A high-performance bus, a low-performance, and a bridge for transferring the data between the two buses.

      Eg: ARMA Bus belongs to this structure

Computer Organization and Architecture
CS 202

Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Memory Location, Addresses, and Operation

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups called word.
- *n* is called word length.

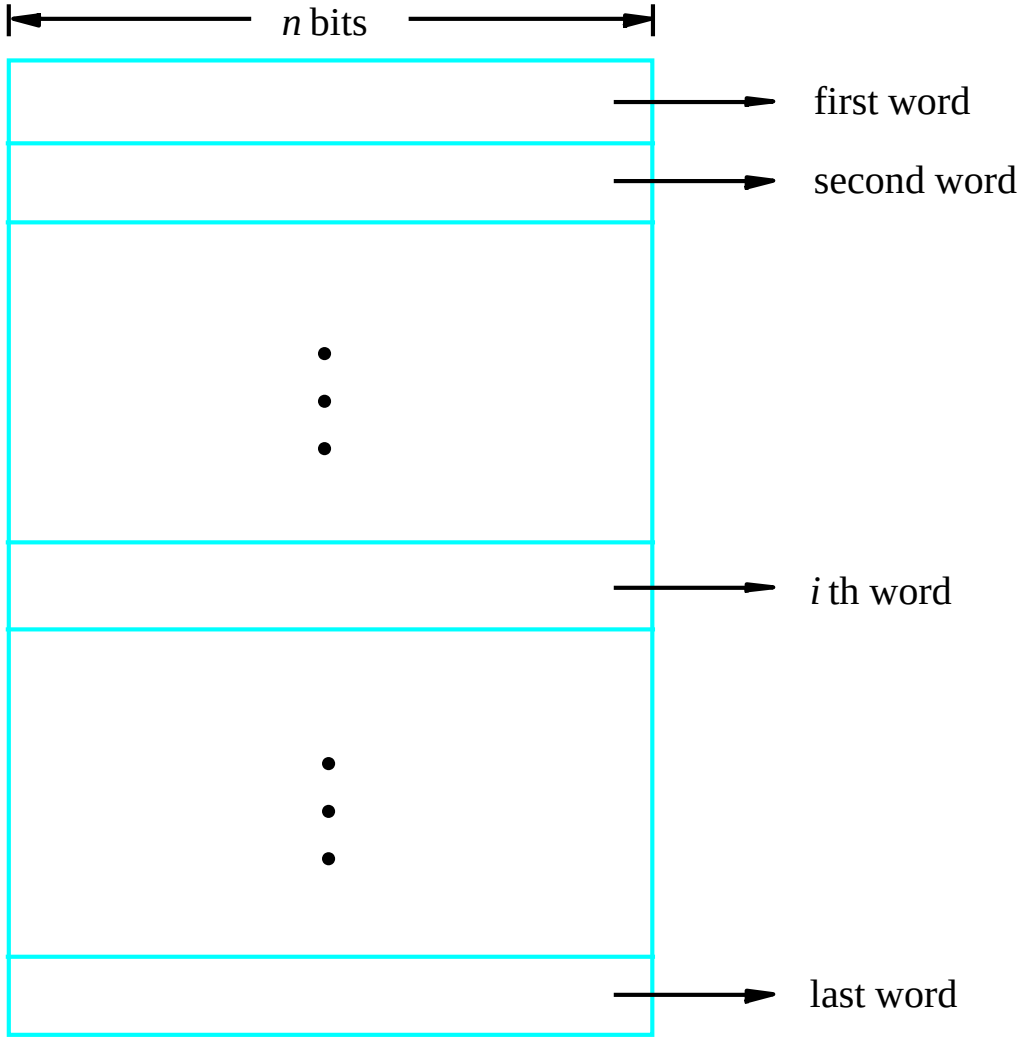
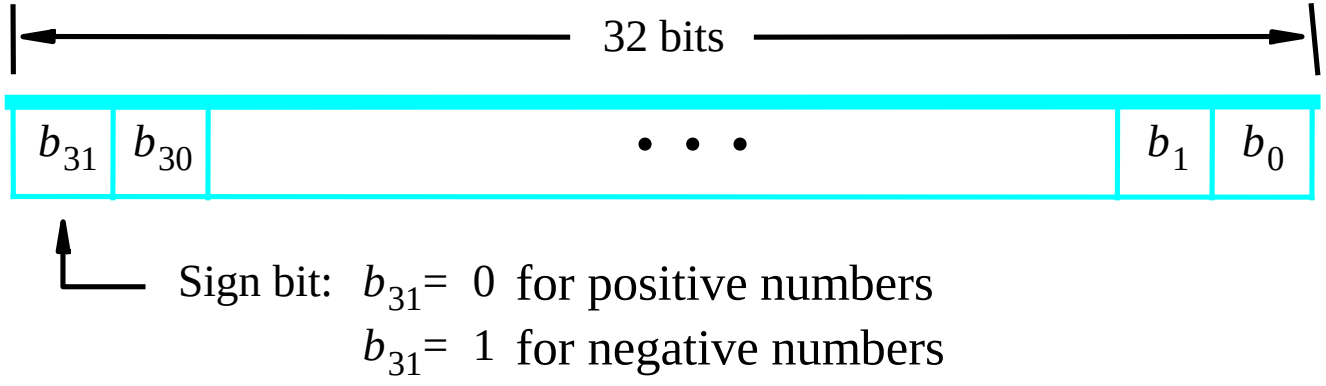


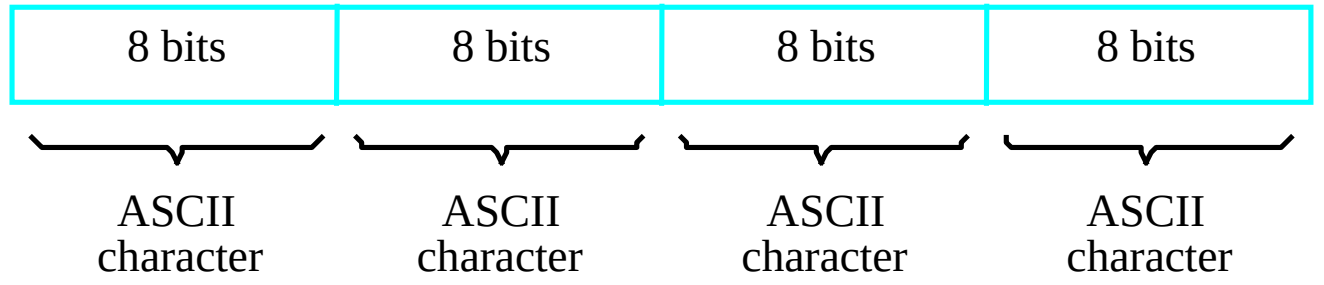
Figure 2.5. Memory words.

Memory Location, Addresses, and Operation

- 32-bit word length example



(a) A signed integer



(b) Four characters

Accessing numbers, characters and strings

- Number
  - By its word address as it usually occupies one word
- Character
  - By byte address
- Strings
  - They are of variable length
  - Beginning of the string by giving the beginning byte address which contains first character
  - Successive bytes contains successive characters
  - Termination?
    - Either by a special control character
    - Or a separate memory word location/ register containing a number indicating the string length

Memory Location, Addresses, and Operation

- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A *k*-bit address memory has 2^k memory locations, namely $0 – 2^k-1$, called memory space.
- 24-bit addr memory: 2^{24} = 16,777,216 = 16M ($1M=2^{20}$)
- 32-bit addr memory: 2^{32} = 4G ($1G=2^{30}$)
- 1K(kilo)$=2^{10}$
- 1T(tera)$=2^{40}$
- 48- bit address memory : 2^{48} = 256 T

Memory Location, Addresses, and Operation

- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory is called byte-addressable memory.
- Word alignment - Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word.
- Byte locations have addresses - 0,1, 2, 3, etc …
- word length is 16 bits, the successive words are located at addresses - 0, 2, 4, 6, etc…
- word length is 32 bits, the successive words are located at addresses - 0, 4, 8,12, etc…

Byte 0 (8 bits) (0-7	Byte 1(8-15) 16 word length
Byte 2 (16-23)	Byte 3 (24-31)
Byte 4	Byte 5

BYTE 0	BYTE 1	BYTE 2	BYTE 3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

Computer Organization and Architecture
CS 202

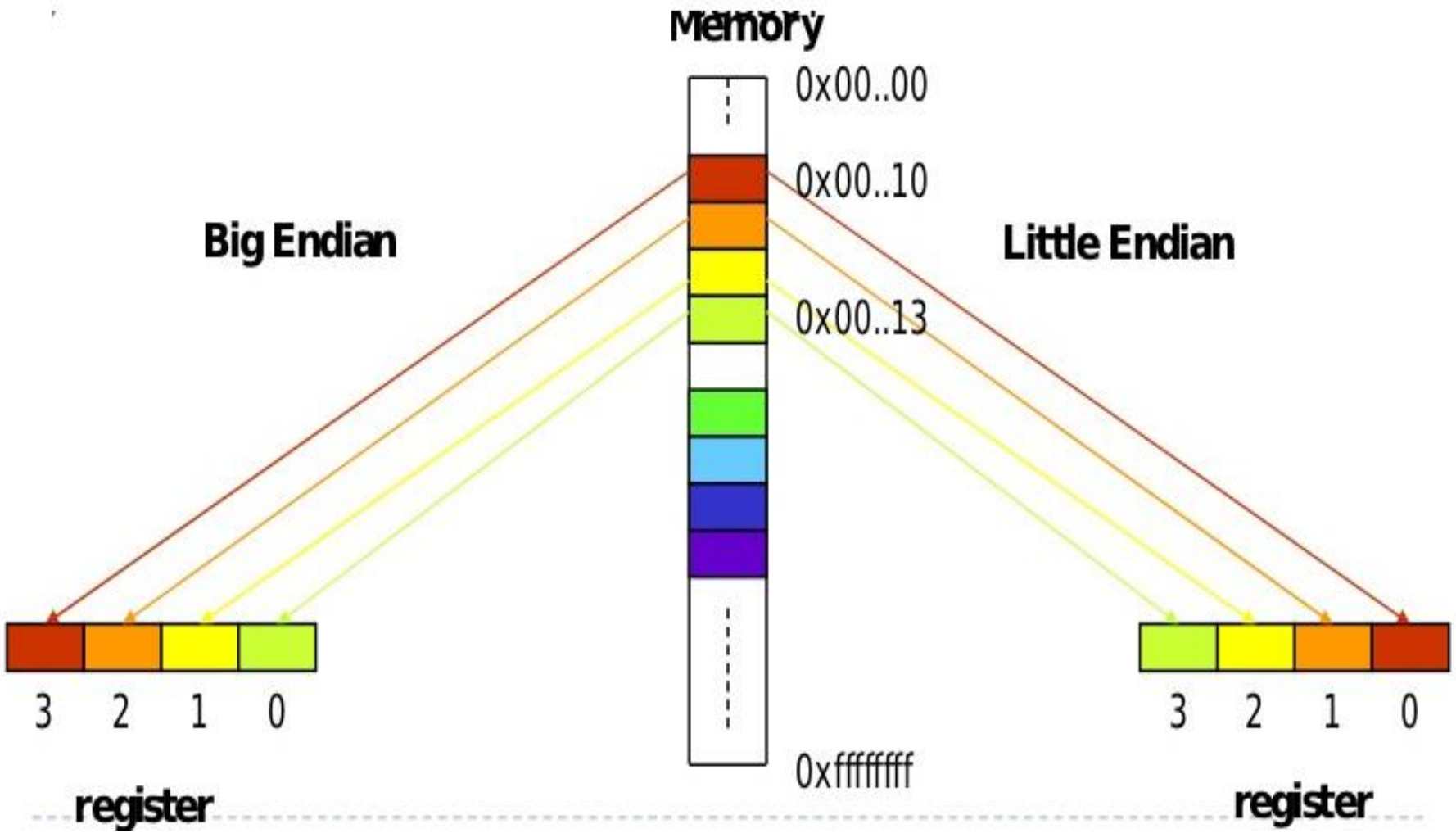
Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Big-Endian and Little-Endian Assignments

Endianness: ordering of bytes within a larger object, eg how a word is stored in memory

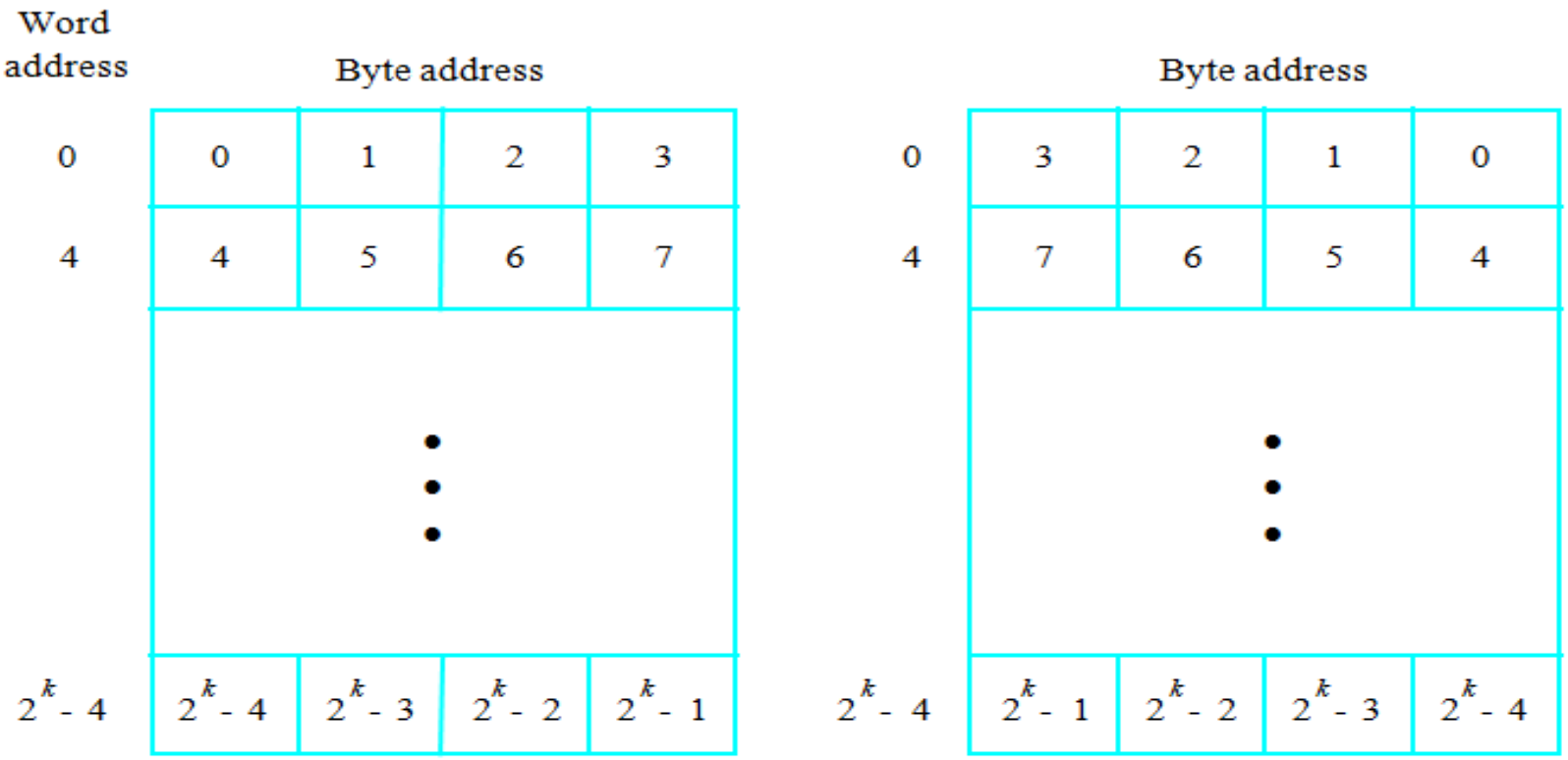
Eg: 68000 is a big endian processor



Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the least significant bytes of the word



Byte and word addressing

Problems

1. A memory has 32-bit address and byte-addressable, what is the size of the memory (in bytes)?
2. A memory has 24-bit address and word-addressable with a word length of 32 bits, what is the size of the memory (in bytes)?
3. A memory has 16-bit address and byte addressable. Word length is 32 bits. How many words can we store in such a memory?

- Note: A memory with k-bit address
- a) byte-addressable → 2^k bytes
- b) word-addressable → 2^k words

Answers

1. 32-bit address, byte addressable memory

- No of bytes = 2^{32} = $2^{30} \times 2^{2}$
- = 4G bytes (1G = 2^{30})

- 2. 24-bit address, word addressable, 1Word =32 bits (4 bytes)
- No of words = 2^{24} = $2^{20} \times 2^{4}$
- No of bytes = No of words x (bytes/word)
- = $2^{20} \times 2^{4} \times 2^{2}$ (1 word = 4 bytes)
- = 64M bytes (1M = 2^{20})

- 3. 16-bit address, byte addressable, 1Word =32 bits (4 bytes)
- No of words = No of bytes / (bytes/word)
- = $2^{16} / 2^{2}$ = $2^{10} \times 2^{6} / 2^{2}$ (1 word = 4 bytes)
- = 16 K words (1K = 2^{10})

Problem

1. A memory has 26-bit address and word-addressable with a word length of 64 bits, what is the size of the memory (in bytes & word)?

- No.of words = $2^{26} = 2^{20} * 2^{6}$ = 64 G words
- No of bytes = No of words x (bytes/word)
- $= 2^{20} * 2^{6} * 2^{2}$
- = 256 GB

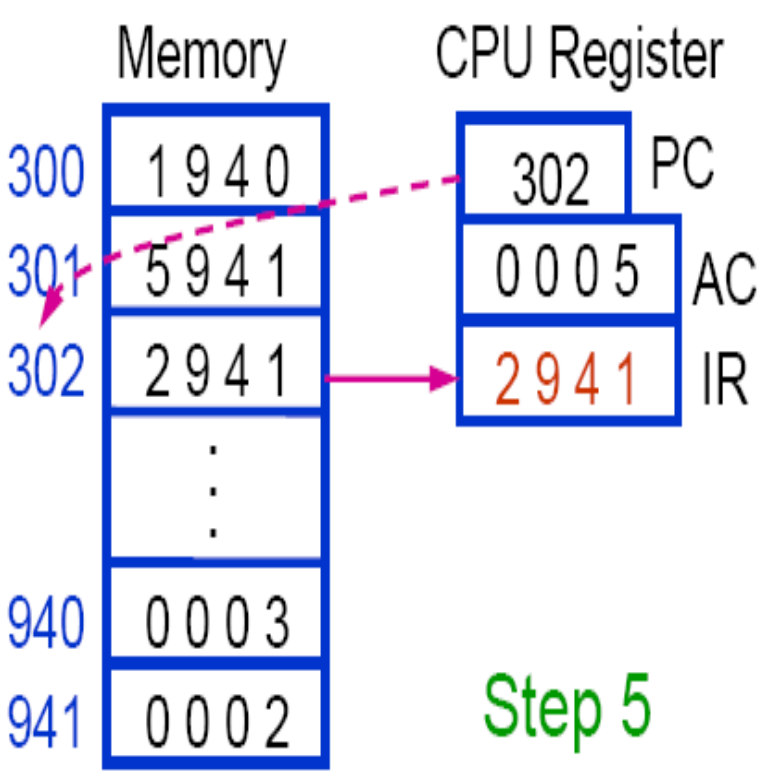
Memory Operations

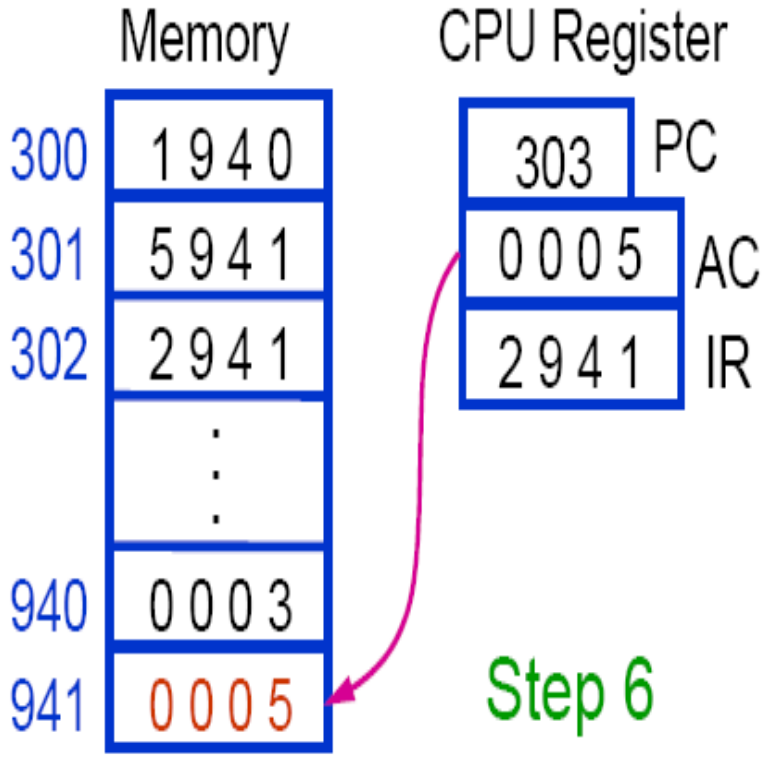
- Load (or Read or Fetch)
  - Copy the content
  - Memory content doesn’t change
  - Address → MAR, read frm mem, content → MDR
  - Registers can be used
- Store (or Write)
  - Overwrite the content in memory
  - Address → MAR and Data → MDR, write to mem
  - Registers can be used

A Partial Program Execution Example

Ex : Load 940 , Add 941 , Store 941

opcodes Load-1 , Add-5 , Store-2



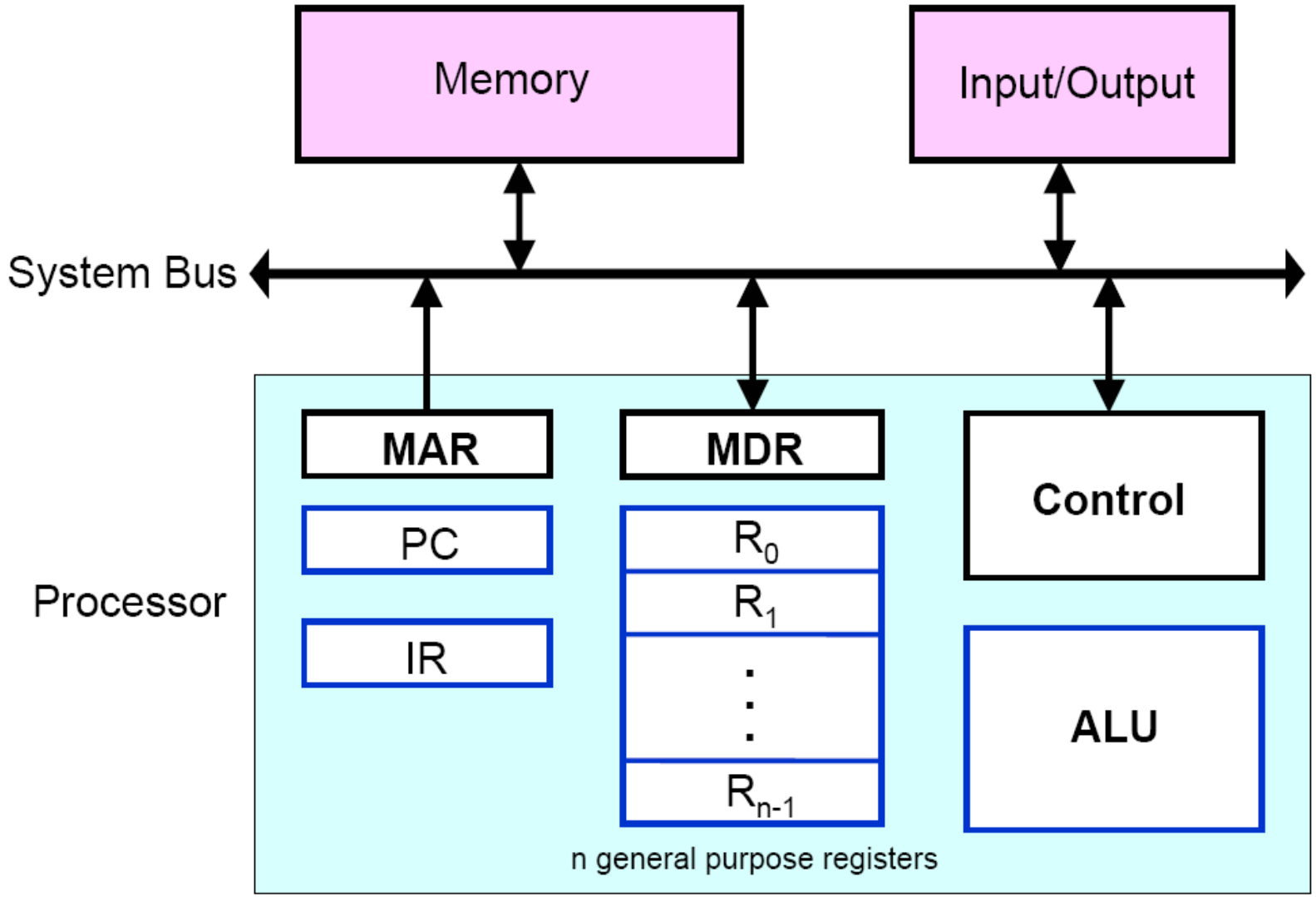


Instruction and Instruction Sequencing

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Basic Operational Concepts

Connection Between the Processor and Memory



Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location
  - R1← [LOC] R1 ← [3000]
  - R3 ← [R1] + [R2] R3 = 5 + 3
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs
  - Move R1, LOC => R1← [LOC]
  - ADD R1, R2, R3=> R1 ← [R2] + [R3]
  - R2 & R3 SOURCE OPERANDS R1 DESTINATION OPERAND

  - Move LOC, R1 => LOC ← [R1]
  - ADD R3,R1,R2 => R3 ← [R1] +[R2]
  - [R1]=3 [R2]=8 R3 =11

Computer Organization and Architecture
CS 202

Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Instruction cycle

Instruction cycle consists of following phases:

- Fetching an instruction from memory.
- Decoding the instruction.
- Reading the effective address from memory in case of the instruction having an indirect address.
- Execution of the instruction.

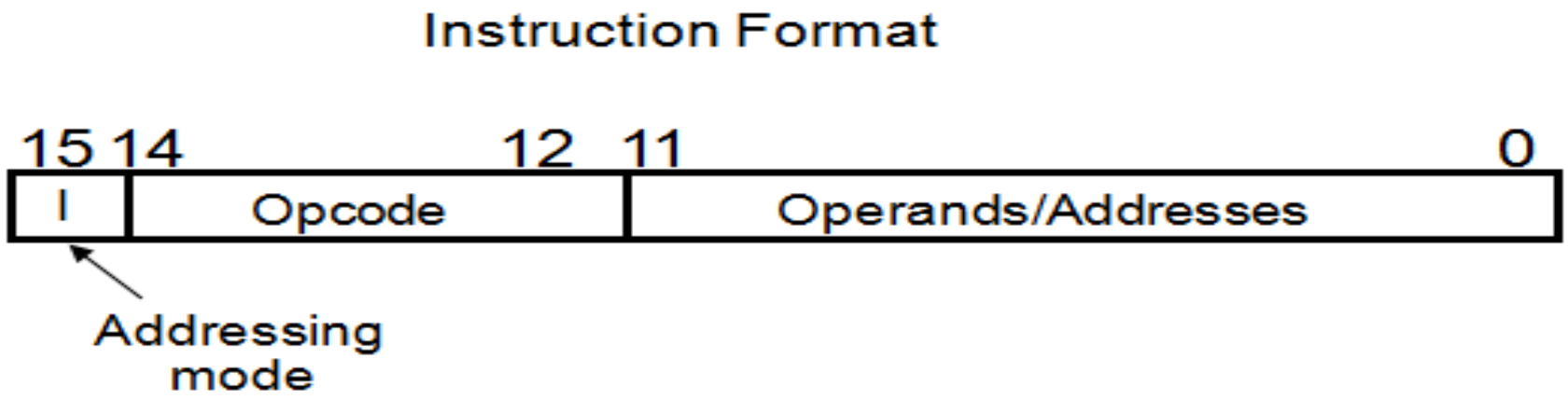
Instruction Format

➢Instruction Format

An instruction consists of bits and these bits are grouped upto make fields.

➢Fields in instruction format :

- Opcode - which tells about the operation to be performed.
- Address field - designating a memory address or a processor register.
- Mode field - specifying the way the operand or effective address is determined.



Instruction Formats

- Three-Address Instructions
  (operation destn, src1, src2)
  - ADD R1, R2, R3 ; R1 ← R2 + R3
- Two-Address Instructions
  (operation destn, src)
  - ADD R1, R2 ; R1 ← R1 + R2
- One-Address Instructions
  (in arithmetic operations, accumulator is implicit)
  - ADD B ; AC ← AC + M[B]
  - LOAD A ; AC ← M[A]
  - PUSH X ; TOS ← M[X]
- Zero-Address Instructions
  (all operands are implied to be in a pushdown stack)
  - ADD ; TOS ← TOS + (TOS – 1) (stack grows up)

  (This operation has effect of popping the two top numbers from stack, adding the numbers, and pushing the sum into the stack)

7	81	TOS
6	76	TOS-1
5	5	
4	4	
3		
2		
1		
0		

Instruction Formats

Eg: Evaluate (A+B)*(C+D) store result in X

- Three-Address

1. ADD R1, A, B ; R1 ← M[A] + M[B]
2. ADD R2, C, D ; R2 ← M[C] + M[D]
3. MUL X, R1, R2 ; M[X] ← R1 * R2

Instruction Formats

Eg: Evaluate (A+B)∗(C+D) store result in X

- Two-Address

1. MOV R1, A ; R1 ← M[A]
2. ADD R1, B ; R1 ← R1 + M[B]
3. MOV R2, C ; R2 ← M[C]
4. ADD R2, D ; R2 ← R2 + M[D]
5. MUL R1, R2 ; R1 ← R1 ∗ R2
6. MOV X, R1 ; M[X] ← R1

Instruction Formats

Eg: Evaluate (A+B)*(C+D) store result in X

- One-Address

1. LOAD A ; AC ← M[A]
2. ADD B ; AC ← AC + M[B]
3. STORE T ; M[T] ← AC
4. LOAD C ; AC ← M[C]
5. ADD D ; AC ← AC + M[D]
6. MUL T ; AC ← AC * M[T]
7. STORE X ; M[X] ← AC

Instruction Formats

Eg: Evaluate (A+B)*(C+D) store result in X

- Zero-Address

1. PUSH A ; TOS ← A, 5
2. PUSH B ; TOS ← B, 10
3. ADD ; TOS ← (A + B)
4. PUSH C ; TOS ← C , 2
5. PUSH D ; TOS ← D , 5
6. ADD ; TOS ← (C + D)
7. MUL ; TOS ← (C+D)*(A+B)
8. POP X ; M[X] ← TOS

Using Registers

- Registers are faster
- Shorter instructions
  - The number of registers is smaller

    (e.g. 32 registers need 5 bits)
- Potential speedup
- Minimize the frequency with which data is moved back and forth between the memory and processor registers.

Addressing Modes

- Opcode field - specifies the operation to be performed.
- Operand - data on which the operation is to be performed.
- operand(data) may be in accumulator, general purpose register or at some specified memory location
- Addressing mode -The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

❖ Effective Address - is the actual location of an operand of an instruction.

Types of Addressing Modes

- Immediate mode
- Register mode
- Absolute/ Direct mode
- Indirect mode
- Index mode
- Base with index
- Base with index and offset
- Relative mode
- Auto - increment mode
- Auto - decrement mode

Immediate Addressing

- Operand is part of instruction
- Operand = Value,
- EA = address of data in memory (in instruction area)
- e.g. MOV R1, #5 ; R1 ← 5 $5_{\text{immediate}}$

–Move 5 to register R1

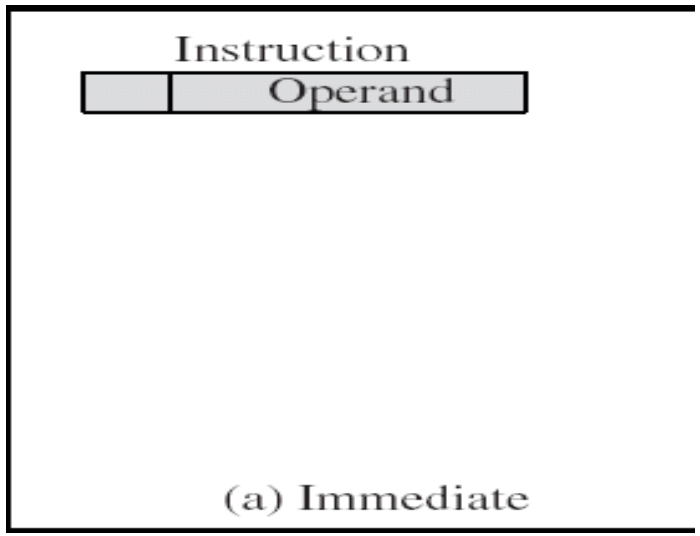
–5 is operand

- No memory reference to fetch data
- Fast
- Limited range

Example

MOV R1, #5

5 → R1



Computer Organization and Architecture
CS 202

Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Addressing Modes

- Opcode field - specifies the operation to be performed.
- Operand - data on which the operation is to be performed.
- operand(data) may be in accumulator, general purpose register or at some specified memory location
- Addressing mode -The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

❖ Effective Address - is the actual location of an operand of an instruction.

Types of Addressing Modes

- Immediate mode
- Register mode
- Absolute/ Direct mode
- Indirect mode
- Index mode
- Base with index
- Base with index and offset
- Relative mode
- Auto - increment mode
- Auto - decrement mode

Immediate Addressing

- Operand is part of instruction
- Operand = Value,
- EA = address of data in memory (in instruction area)
- e.g. MOV R1, #5 ; R1 ← 5 $5_{\text{immediate}}$

–Move 5 to register R1

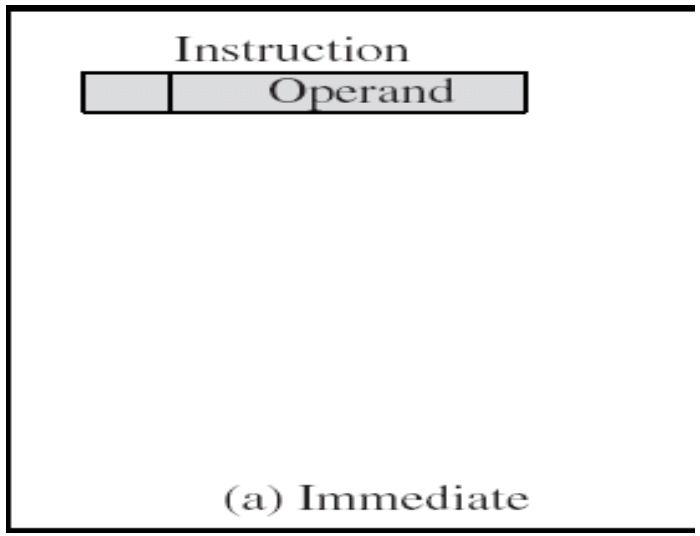
–5 is operand

- No memory reference to fetch data
- Fast
- Limited range

Example

MOV R1, #5

5 → R1



200	MOV mode
201	#5
202	Next instruction

Register Addressing

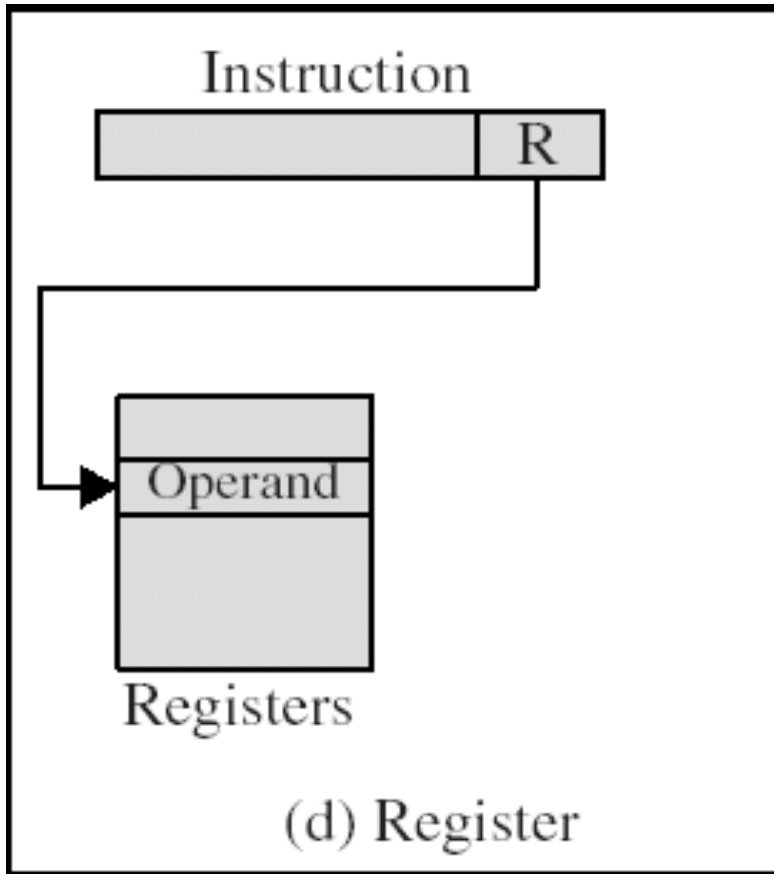
- Operand is held in register, whose name (address) given in instructn in address field
- EA = Ri (register address)
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch and execution
- No memory access
- Very limited address space

Eg : MOV R1, R2

ADD R1, R2

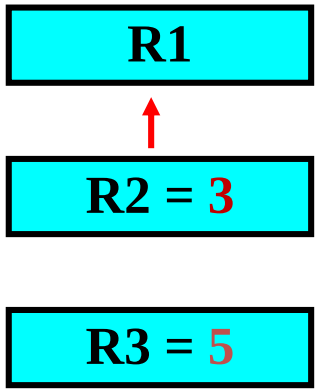
200	MOV R1 mode
201	R2
202	Next instruction
R2	46

Register Addressing Diagram



Example

MOV R1, R2



Direct /Absolute Addressing

- Address field contains address of operand (memory location)
- Effective address EA = A (memory address)

  eg: MOV R1, A

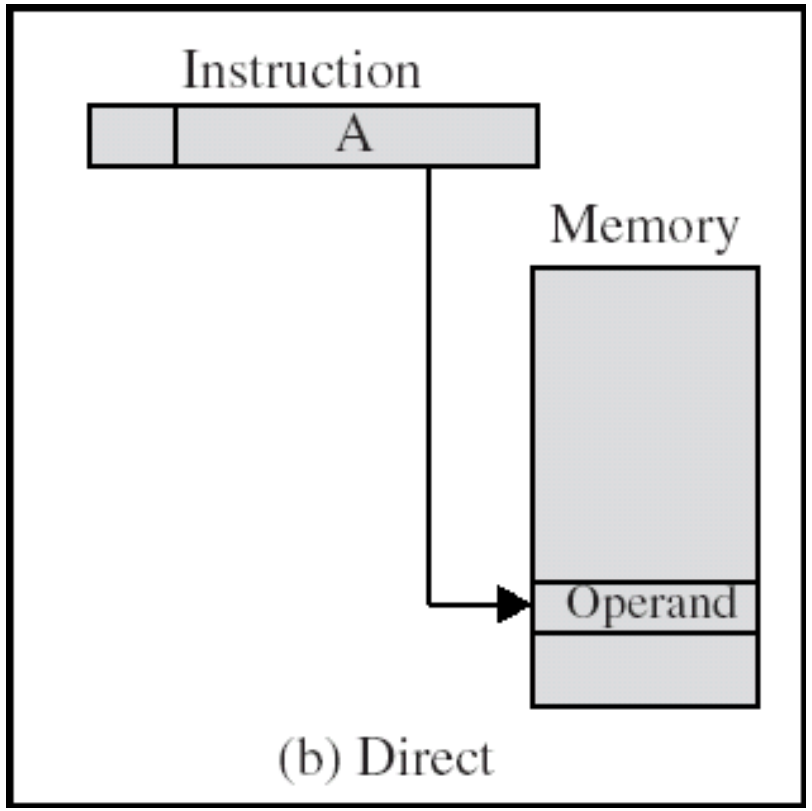
Eg : - MOV R1,300

MEMORY LOCATION 300 = 67H, R1 = 67H

- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

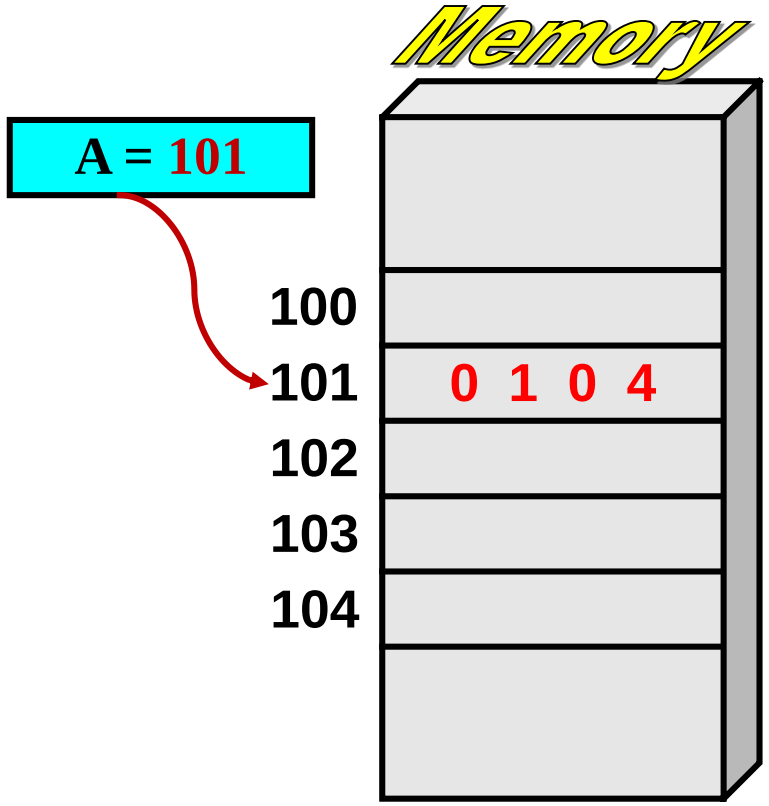
Direct /Absolute Addressing

MOV R1, 101



Example

MOV R1, A



Indirect Addressing (1/2)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA =[A]
  - –Look in A, find an address (content of A) and look in memory at that address location for operand
  - – In C programming, E=*D, D is a Pointer Variable
  - –Can be compiled into commands

    MOV R1, D

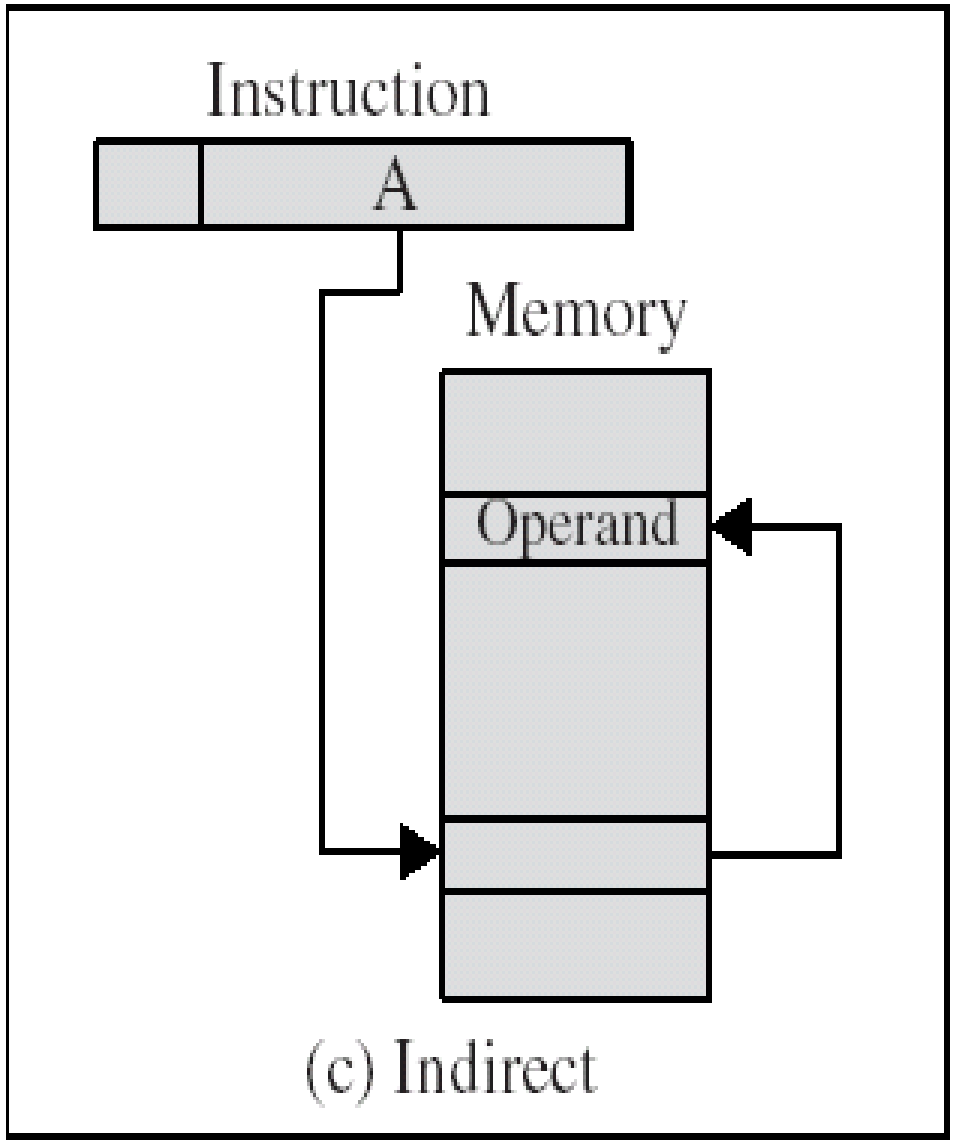
    MOV E, (R1)
  - – using indirect addressing => MOV E, (D)
- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded, e.g. EA = (((A)))
- Multiple memory accesses ,Hence slower

- MOV R1, (3000) R1 = 46H
- 3000 – 3007
- 3007- 46H
- R1 = 46H

- MOV R1, (((3000))) R1 = 46H

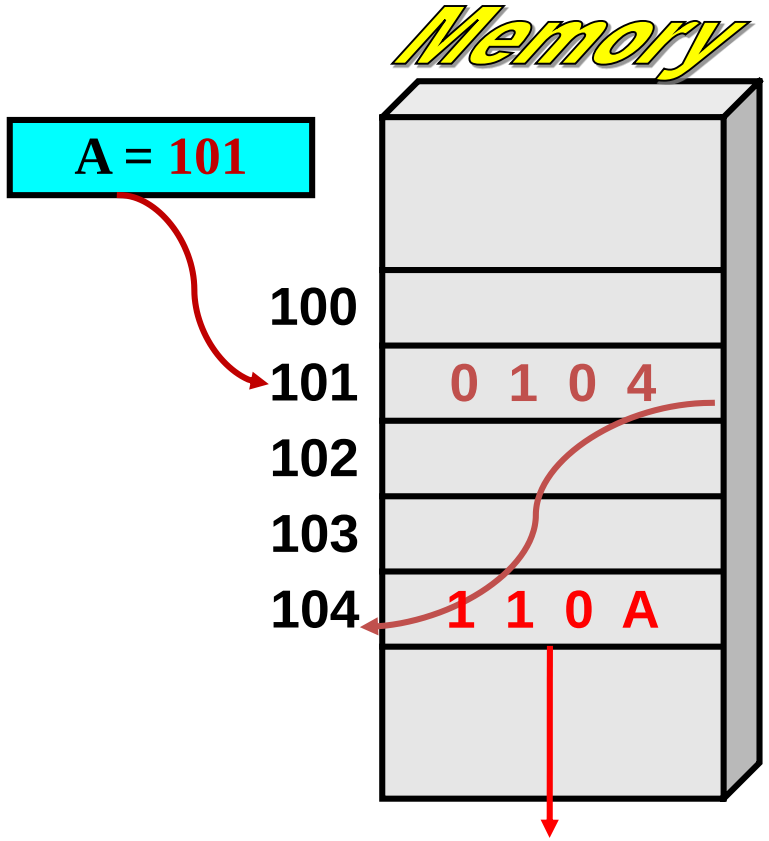
- 3000 – 3007
- 3007- 3009
- 3009- 3100
- 3100 – 46H
- R1 = 46H

Indirect Addressing Diagram



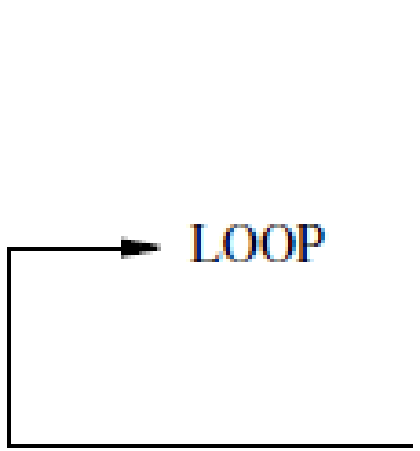
Example

MOV R1, (A)



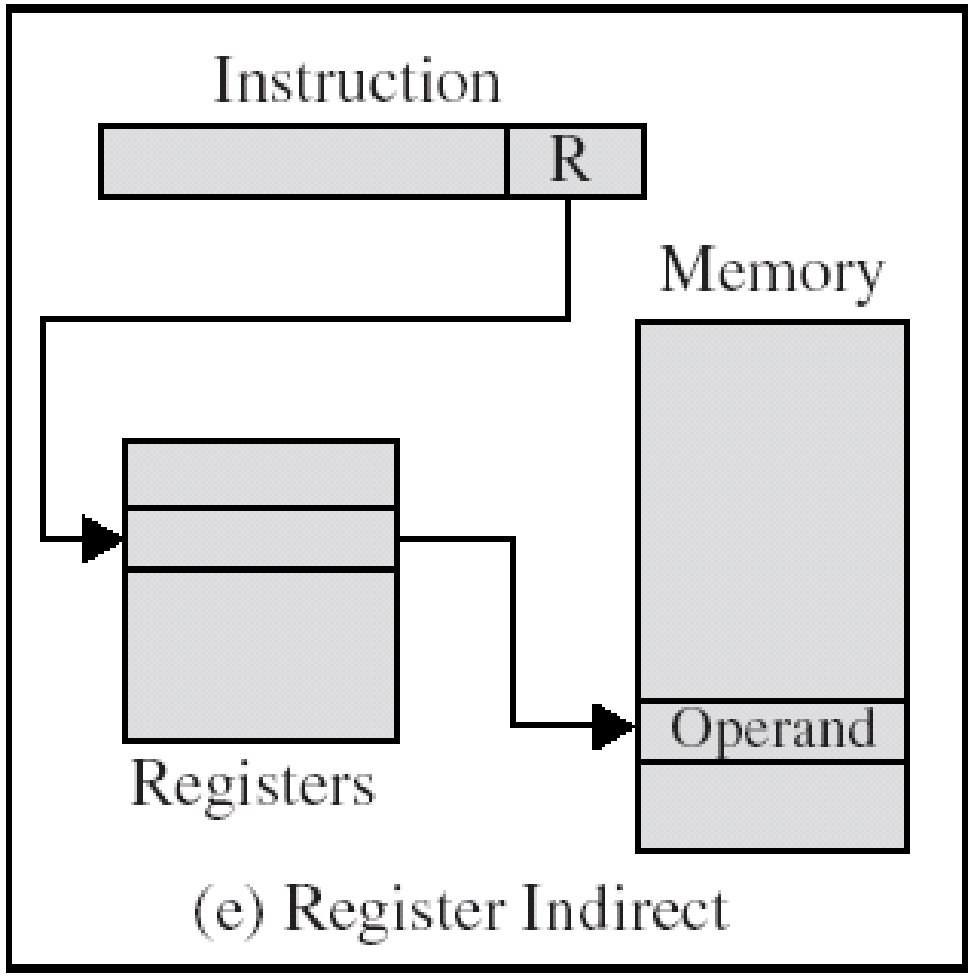
Register Indirect Addressing

- Operand is in register or memory cell pointed to by contents of register Ri
- EA = [Ri]
- Large address space (2^n)
- One fewer memory access than indirect addressing



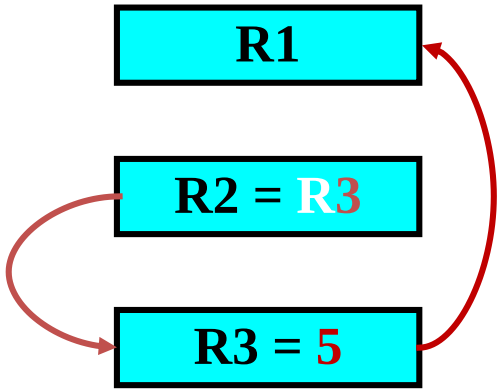
Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Register Indirect Addressing Diagram



Example

MOV R1, (R2)



Computer Organization and Architecture
CS 202

Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Indexed / Displacement Addressing

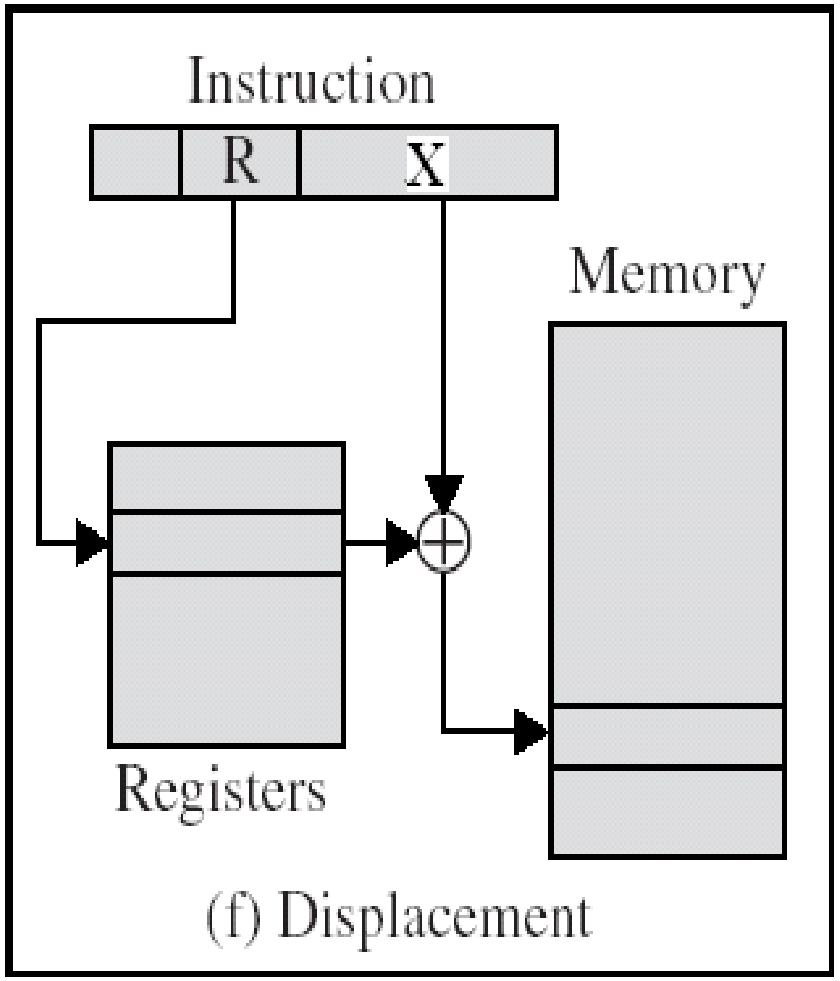
- **Indexing and Arrays**
- Index mode – the effective address of the operand is generated by adding a constant value to the contents of a register.
  - **X(Ri) ; EA = [Ri] + X 5(R0) , R0=1000**
  - **EA = 1000+5 =1005**

  Ri → Index register

  X → constant given either as an explicit number or as a symbolic name representing a numerical value.
    - **Effective address= start address + displacement**
  - (Index register holds address of a new location and value of X defines an offset (displacement))
  - Good for accessing arrays

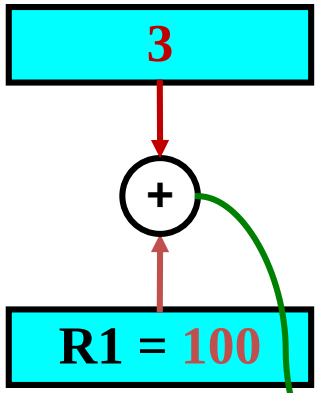
Indexed / Displacement Addressing

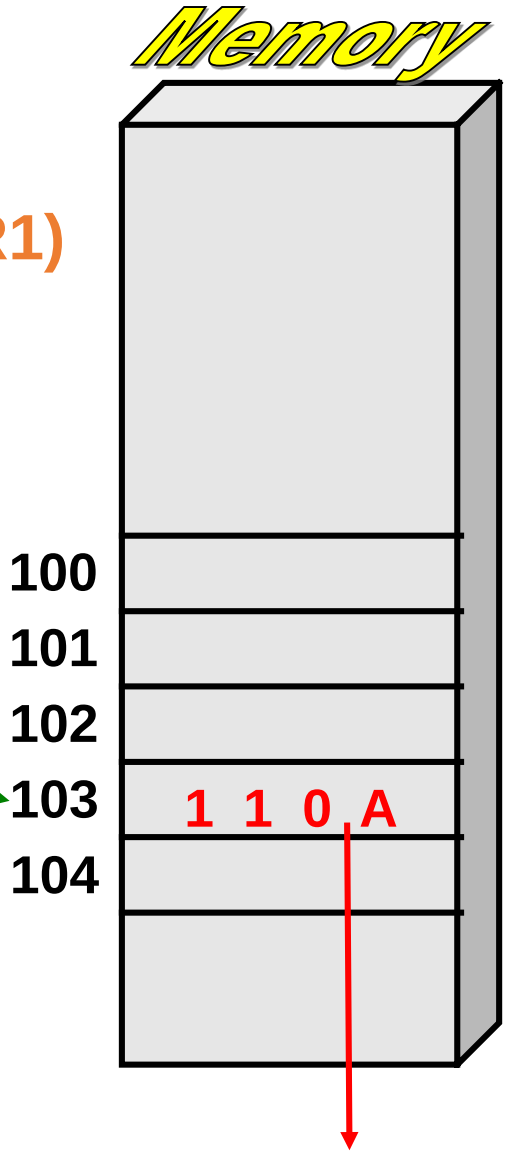
- Indexed Address
  - EA = Index Register value + Relative Addr



Example

MOV R2, 3(R1)





Indexing and Arrays…

- Index mode facilitates access to an operand whose location is defined relative to a reference point within the data structure in which the operand appears.
- Several variations:

➢ X(Ri) ; EA = [Ri] + X (Index)
EA is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register.

➢ (Ri, Rj) ; EA = [Ri] + [Rj] (Base with Index)
A second register (base register) may be used to contain the offset X. EA is the sum of the contents of registers R*i and Rj.*

➢ X(Ri, Rj) ; EA = X + [Ri] + [Rj] (Base with Index and offset)
EA is sum of constant X and contents of registers R*i and Rj .*
It is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by (R*i, Rj) part of addressing mode* (3D arrays).

Relative Addressing (PC-Relative)

- Effective address is determined by Index mode using the program counter in place of the general-purpose register.
- Can be used to access data operands. But, mostly used to specify the target address in branch instructions
- **X(PC) ; EA = [PC] + X (X is a signed number)**

  eg. Branch>0 LOOP

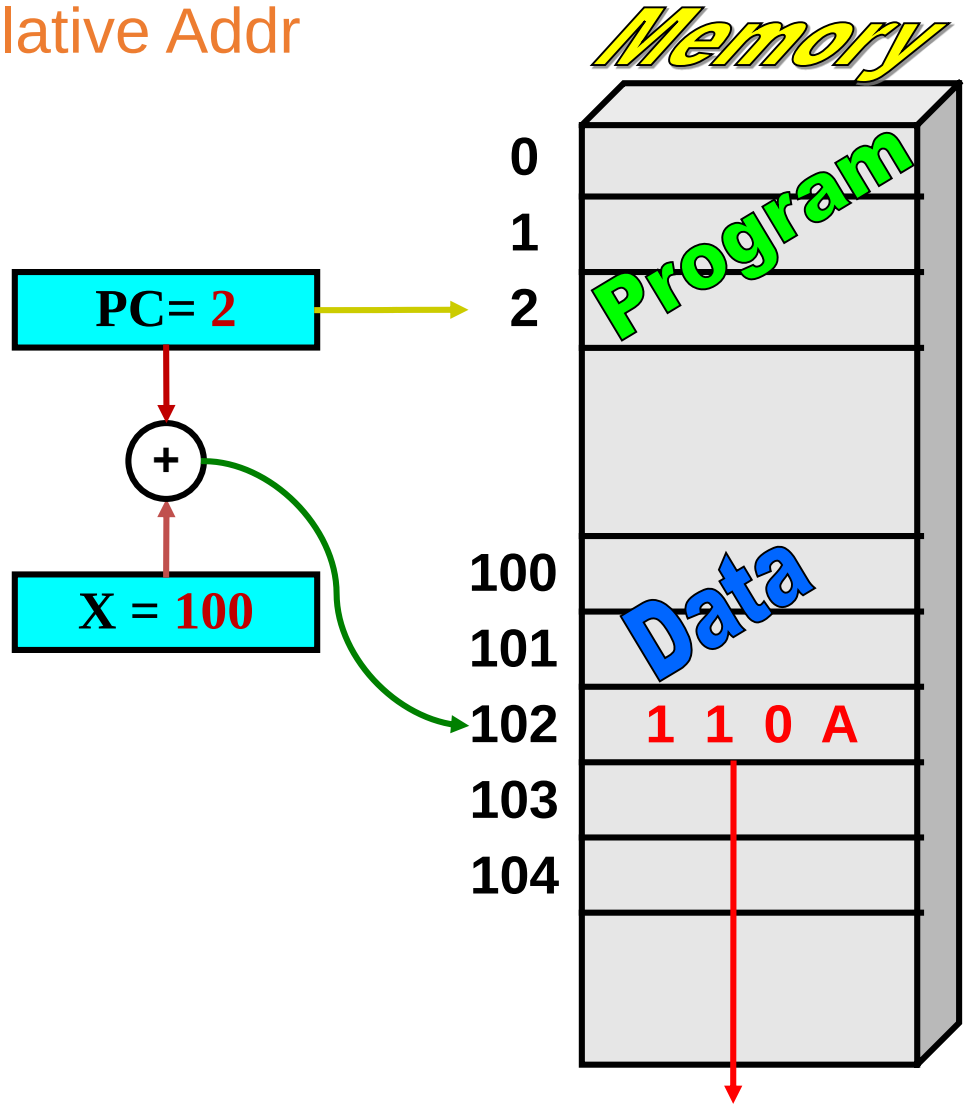
  branch target location (LOOP) is computed by specifying it as an offset from the current value of PC
- Branch target may be either before or after the branch instruction, the offset is given as a singed number.

Relative Addressing (PC-Relative)

- Relative Address
  - EA = PCvalue + Relative Addr

Example

MOV R1, 100(PC)



Autoincrement mode

- the effective address of the operand is the contents of a register specified in the instruction. ADD (R2)+,R0
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.

  (Ri)+ → EA = [Ri] ;

  Increment Ri

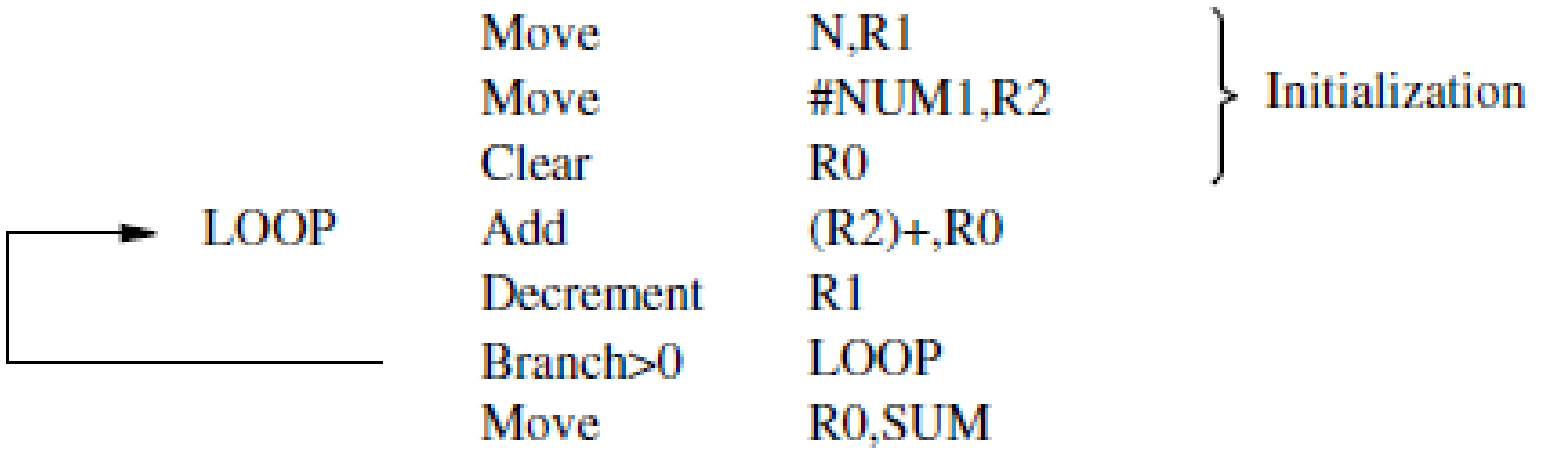
- The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.

Example

ADD (R1)+, 3 ; M[R1]←M[R1]+3, R1 ←R1+1

[R1] = 3000 [R1]+ = 3001 3000 = 5 3001 = 4
OPERAND =4 M[R1]←M[R1]+3, R1 ←R1+1

ANSWER = 7

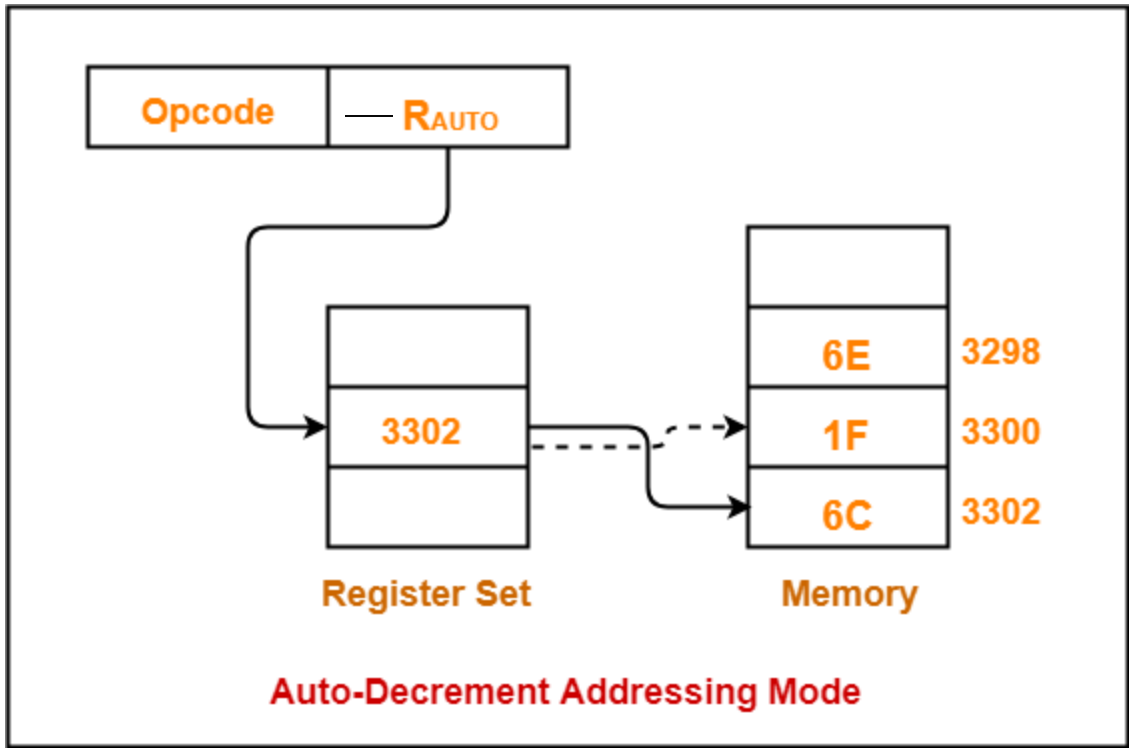


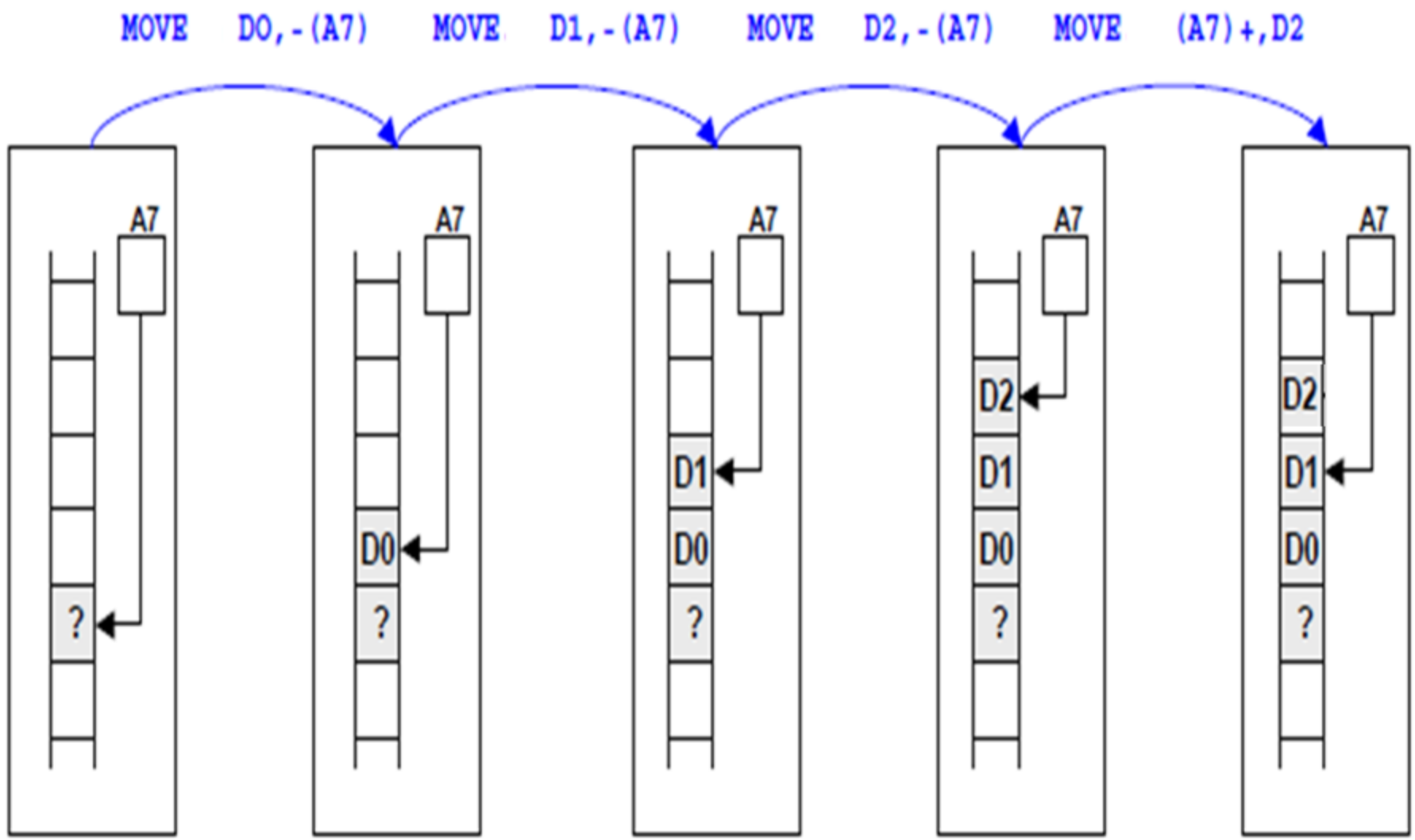
Autodecrement mode

- contents of a register specified in the instruction is automatically decremented and used as effective address of the operand.

- (Ri) → Decrement Ri ;
EA = [Ri];

- Assume operand size = 2 bytes.
- Here, First, the instruction register R_{AUTO} will be decremented by 2.
- Then, updated value of R_{AUTO} will be 3302 – 2 = 3300.
- At memory address 3300, the operand will be found.
- In auto-decrement addressing mode, First, the instruction register R_{AUTO} value is decremented by step size ‘d’.
- Then, the operand value is fetched.




MOVE D0,-(A7)
MOVE D1,-(A7)
MOVE D2,-(A7)
MOVE (A7)+,D2
A7
?
D0
D1
D2

Addressing Modes

- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Computer Organization and Architecture
CS 202

Module -1 Part -2

- Memory Locations, Addresses, and Operations
- Addressing Modes

Addressing Modes

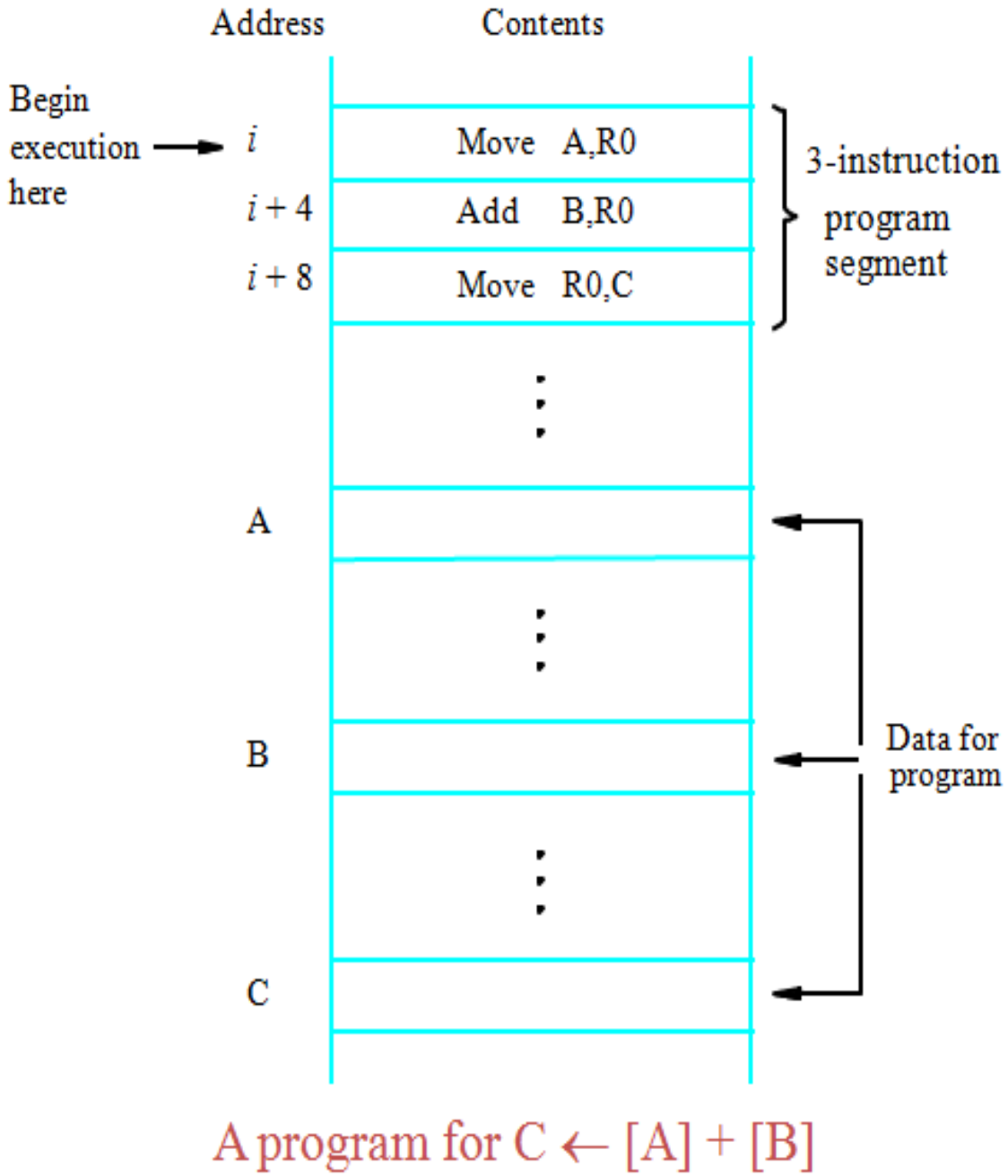
- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Instruction Execution and Straight-Line Sequencing

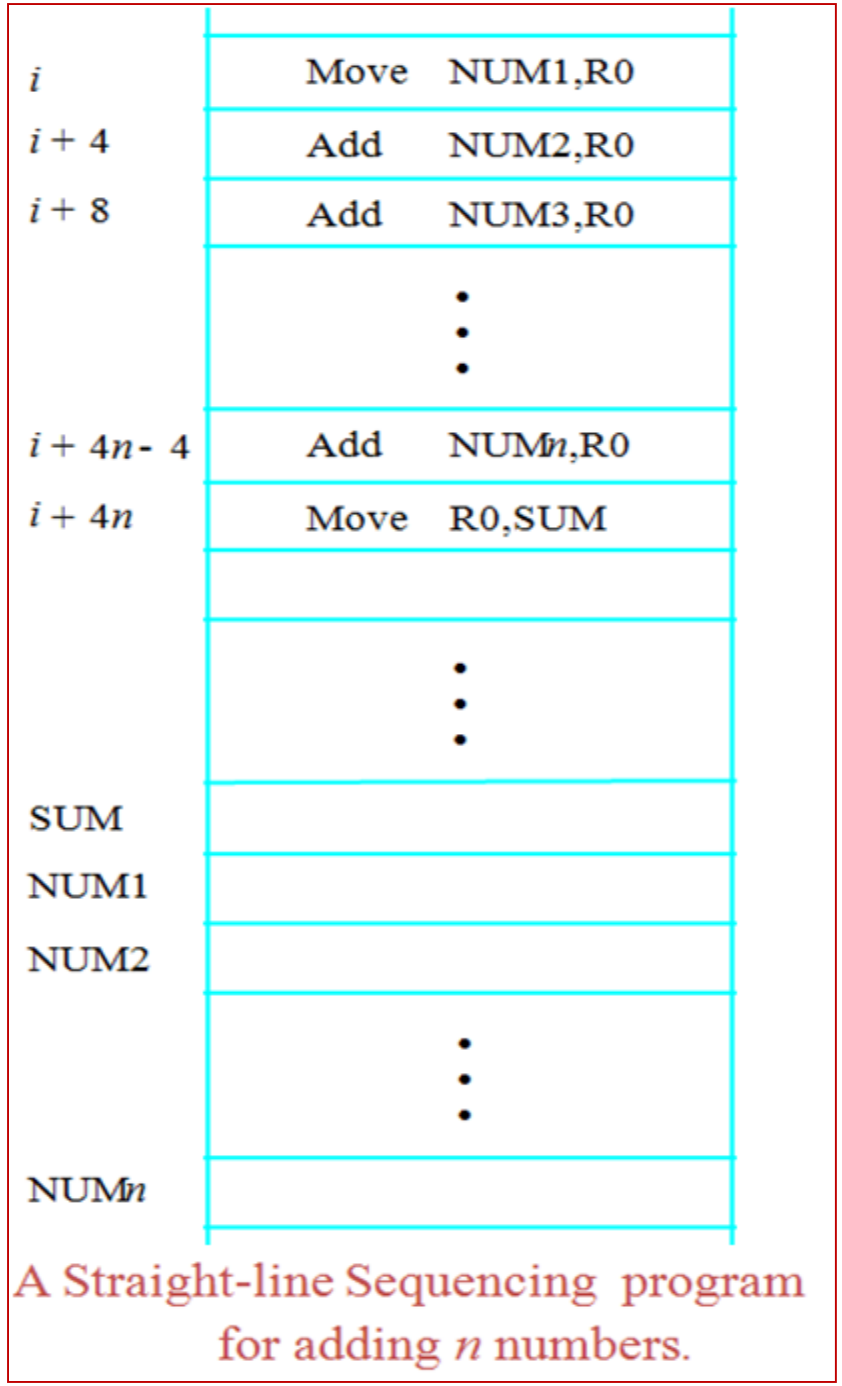


A program for C ← [A] + [B]

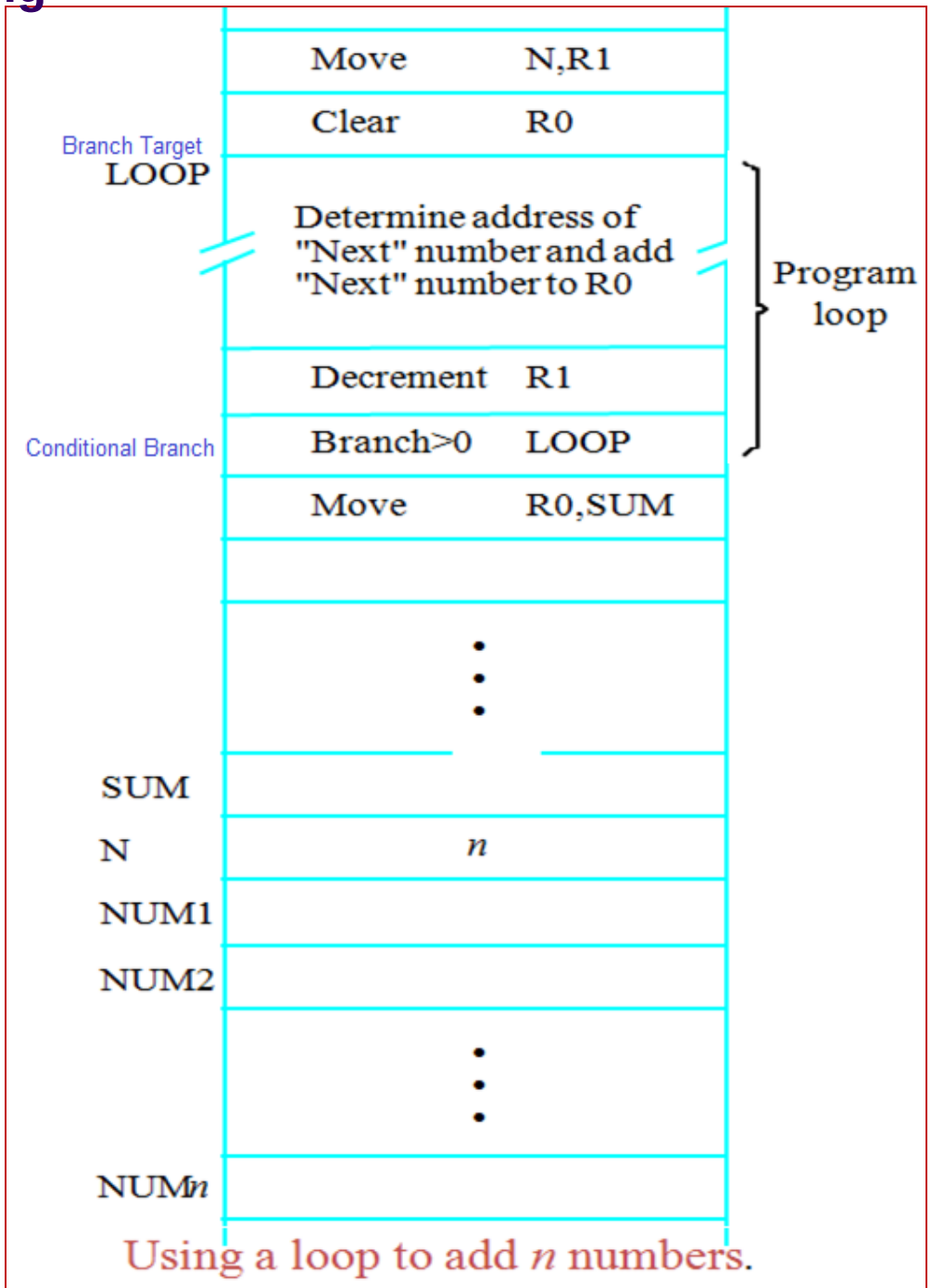
Assumptions:
- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure:
- Instruction fetch
- Instruction execute

Branching



A Straight-line Sequencing program for adding n numbers.



Using a loop to add n numbers.

Branching

Address	Contents	Src, Destn	
Initialization	Move	N,R1	Direct addressing
	Move	#NUM1,R2	Immediate
	Clear	R0	
LOOP	Add	(R2),R0	Register Indirect
	Add	#4,R2	Immediate
	Decrement	R1	Register
	Branch>0	LOOP	
	Move	R0,SUM	Register

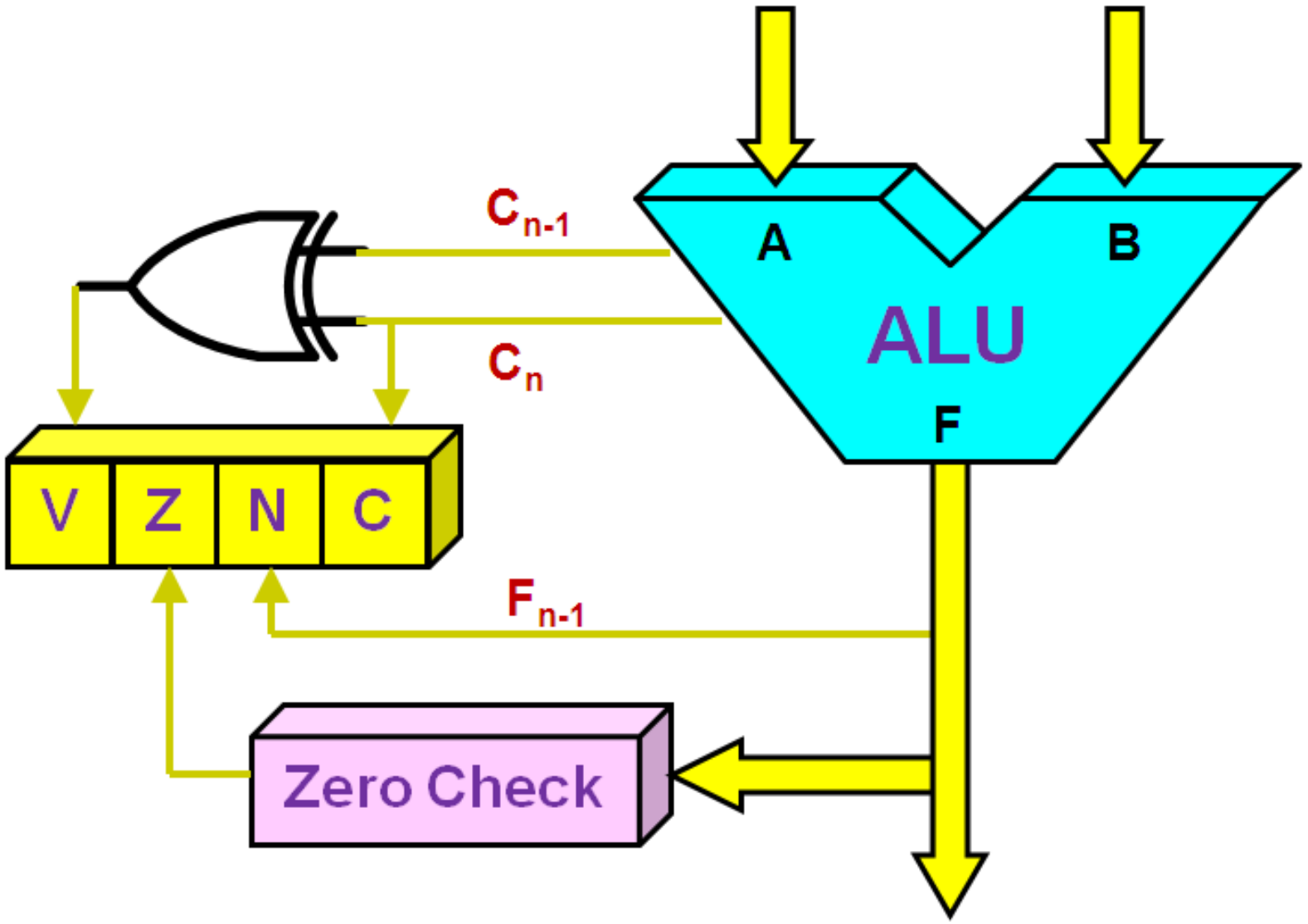
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2)+,R0	Auto Increment
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Condition Codes

- Condition-code bits are also called status bits or flag bits.
- Condition code flags => bits are set or cleared as a result of an operation performed in the ALU.
- Condition code register / status register:
  - Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries (C_n and C_{n-1}) is equal to 1, else V=0. This is the condition for an overflow when negative numbers are in 2's complement.

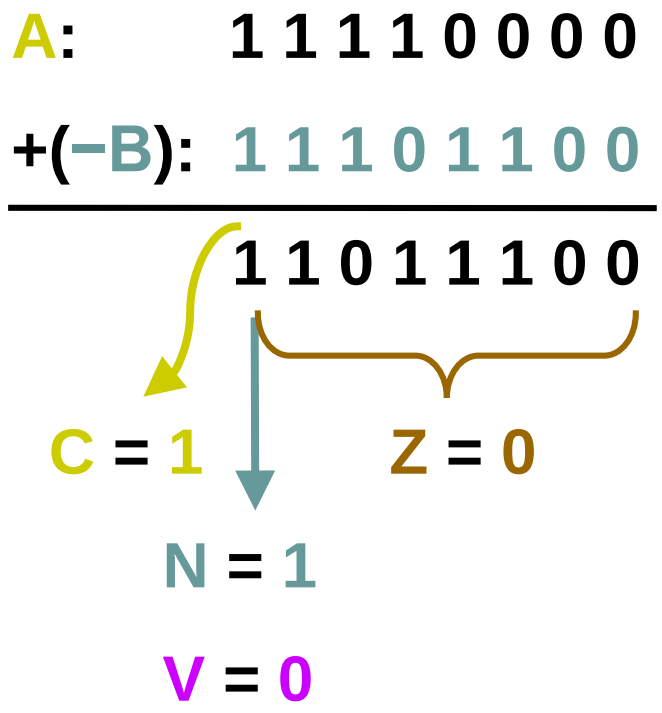
    eg (In 8-bit ALU, V=1 if the output is greater than +127 or less than -128)
  - Bit Z (zero) is set to 1 if the output of the ALU contains all 0's, else Z=0.
  - Bit N (negative) is set to 1 if the highest-order bit F_{n-1} is 1, else N=0.
  - Bit C (carry) is set to 1 if the end carry C_n is 1, else C=0.
- Different instructions affect different flags

Condition Codes



Conditional Branch Instructions

- Example:
  - A: 1 1 1 1 0 0 0 0
  - B: 0 0 0 1 0 1 0 0



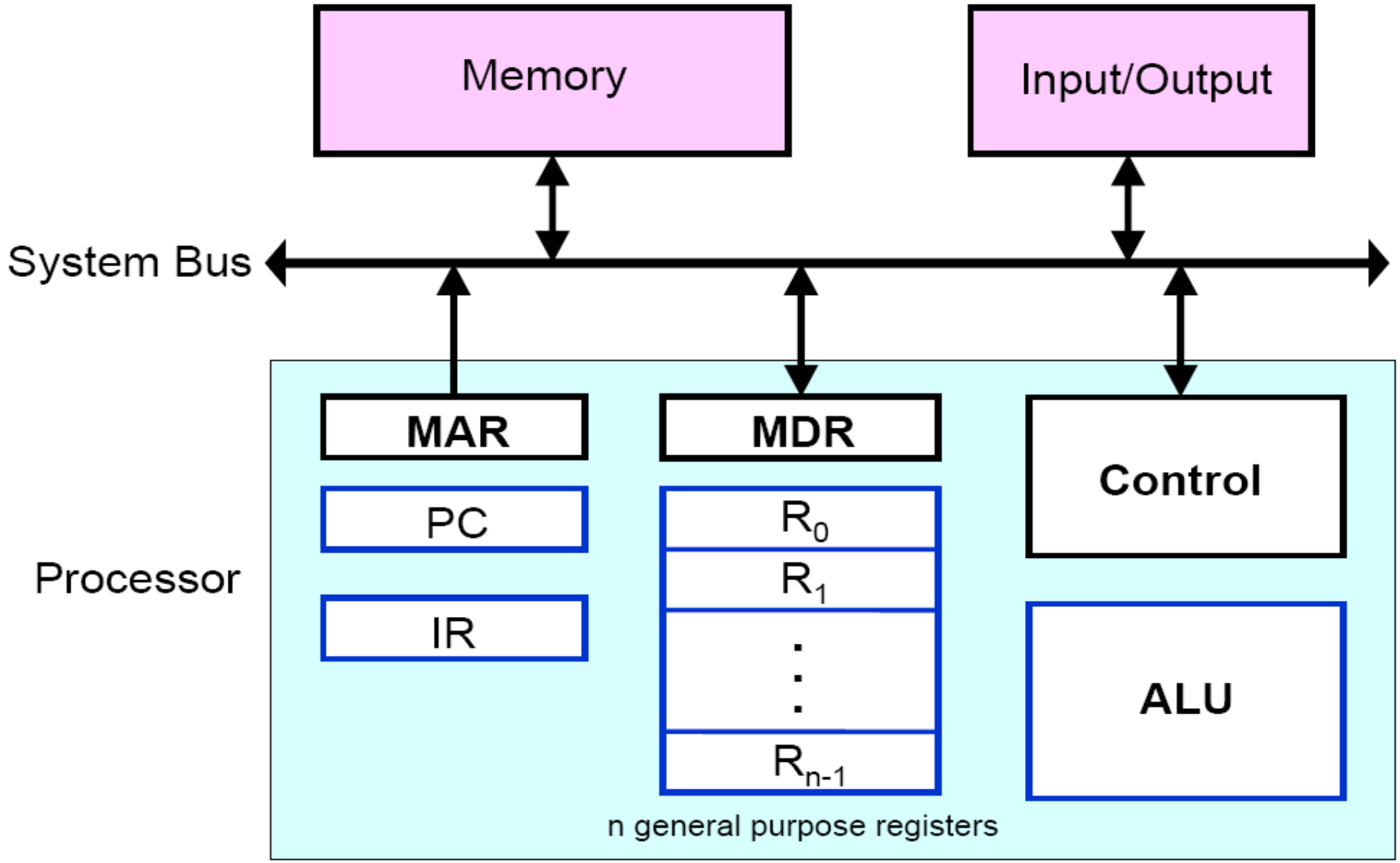
Computer Organization and Architecture
CS 202

Module - 1 Part – III

Basic Processing Unit

Overview

- Instruction Set Processor (ISP)
- Central Processing Unit (CPU)
- A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program.
- An instruction is executed by carrying out a sequence of more rudimentary operations.


Memory
Input/Output
System Bus
Processor
MAR
MDR
Control
PC
R_0
R_1
IR
ALU
R_{n-1}
n general purpose registers

Fundamental Concepts

- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch 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).**
- **Instruction Register (IR)-** Instructions fetched from memory are kept in IR in CPU

Executing an Instruction

❖Fetch phase

- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR.

IR ← [[PC]]

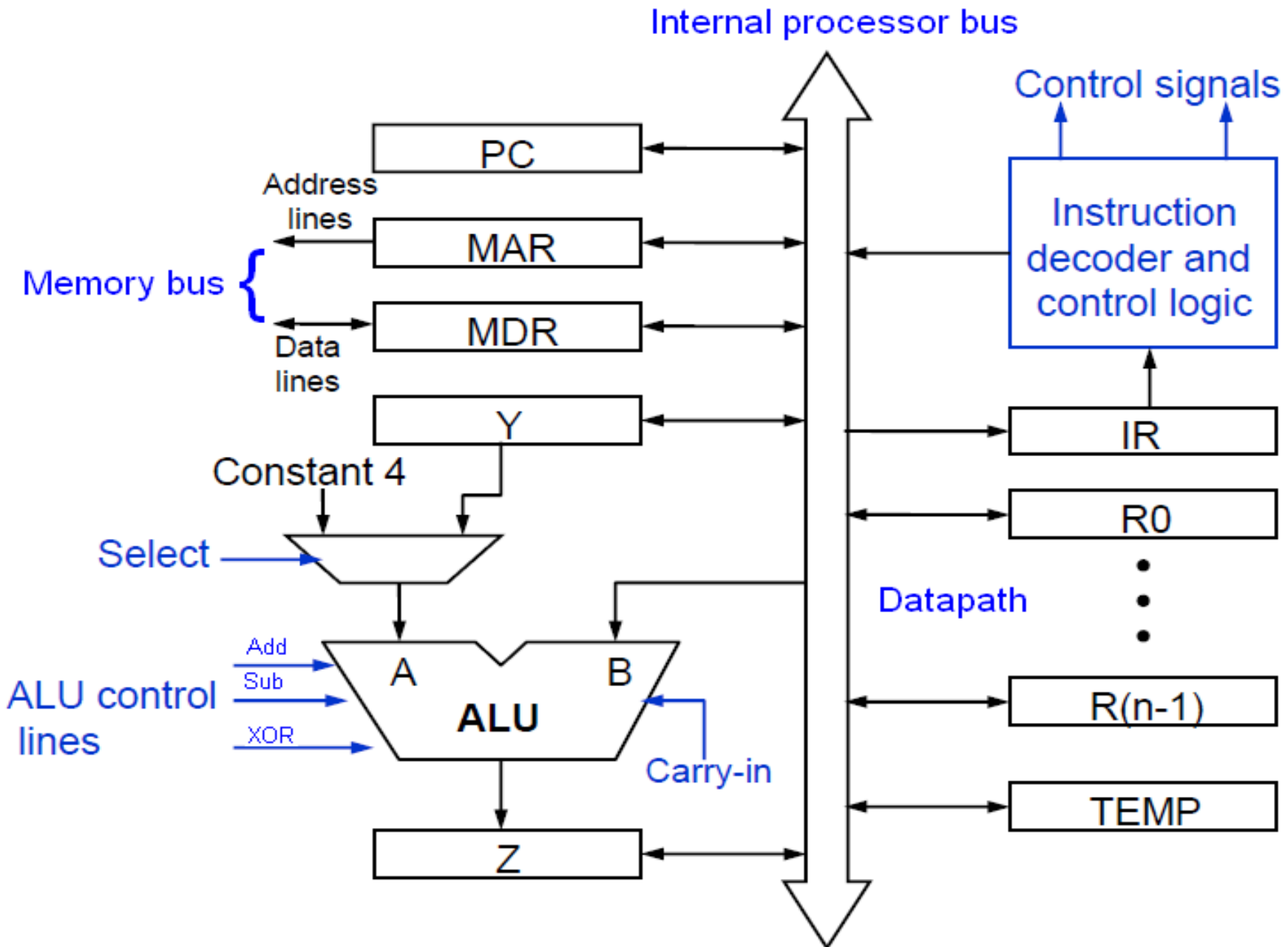
- Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

PC ← [PC] + 4

❖Execution phase

- Carry out the actions specified by the instruction in the IR

Processor Organization



Internal organization of the processor

- ALU
- Registers for temporary storage
- Various digital circuits for executing different micro operations. (gates, MUX, decoders, counters)
- Internal data path for movement of data between ALU and registers.
- Driver circuits for transmitting signals to external units.
- Receiver circuits for incoming signals from external units.

Internal organization of the processor

PC: Program Counter

- Keeps track of execution of a program
- Contains the memory address of the next instruction to be fetched and executed.

MAR: Memory Address Register

- Holds the address of the location to be accessed.
- I/P of MAR is connected to Internal bus and an O/p to external bus.

MDR: Memory Data Register

- Contains data to be written into or read out of the addressed location.
- It has 2 inputs and 2 Outputs.
- Data can be loaded into MDR either from memory bus or from internal processor bus.

The data and address lines are connected to the internal bus via MDR and MAR

Internal organization of the processor

Registers: The processor general purpose registers R0 to Rn-1 are provided for use by programmer, vary considerably from one processor to another.

- Special purpose registers - index & stack registers.
- Registers Y,Z &TEMP are temporary registers used by processor during the execution of some instruction.

Multiplexer: Select either the output of the register Y or a constant value 4 to be provided as input A of the ALU.

- Constant 4 is used by CPU to increment the contents of PC.

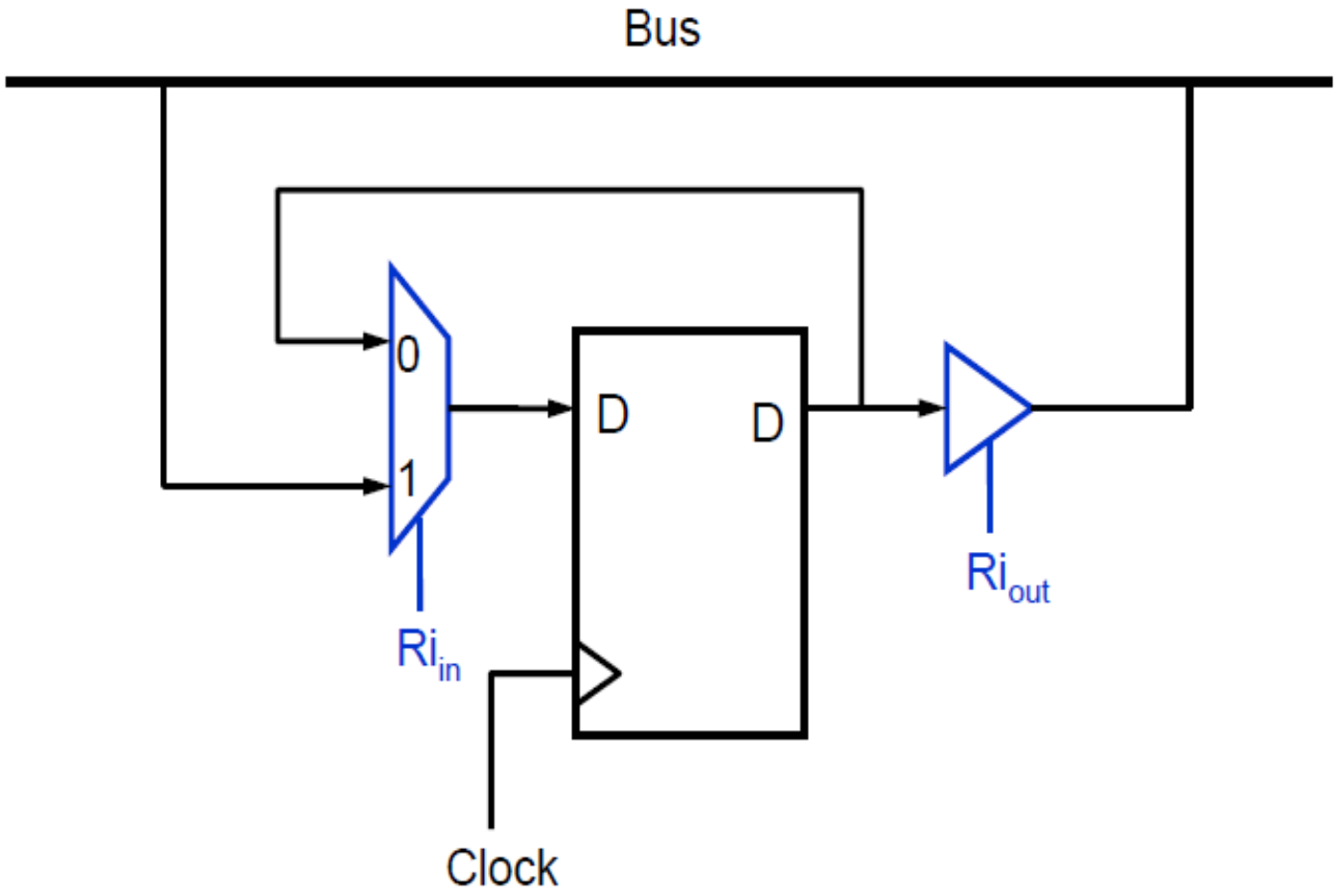
ALU: Used to perform arithmetic and logical operation.

Data Path:

- The registers, ALU and interconnecting bus are collectively referred to as the data path.

Register Transfers

- All operations and data transfers are controlled by the processor clock.



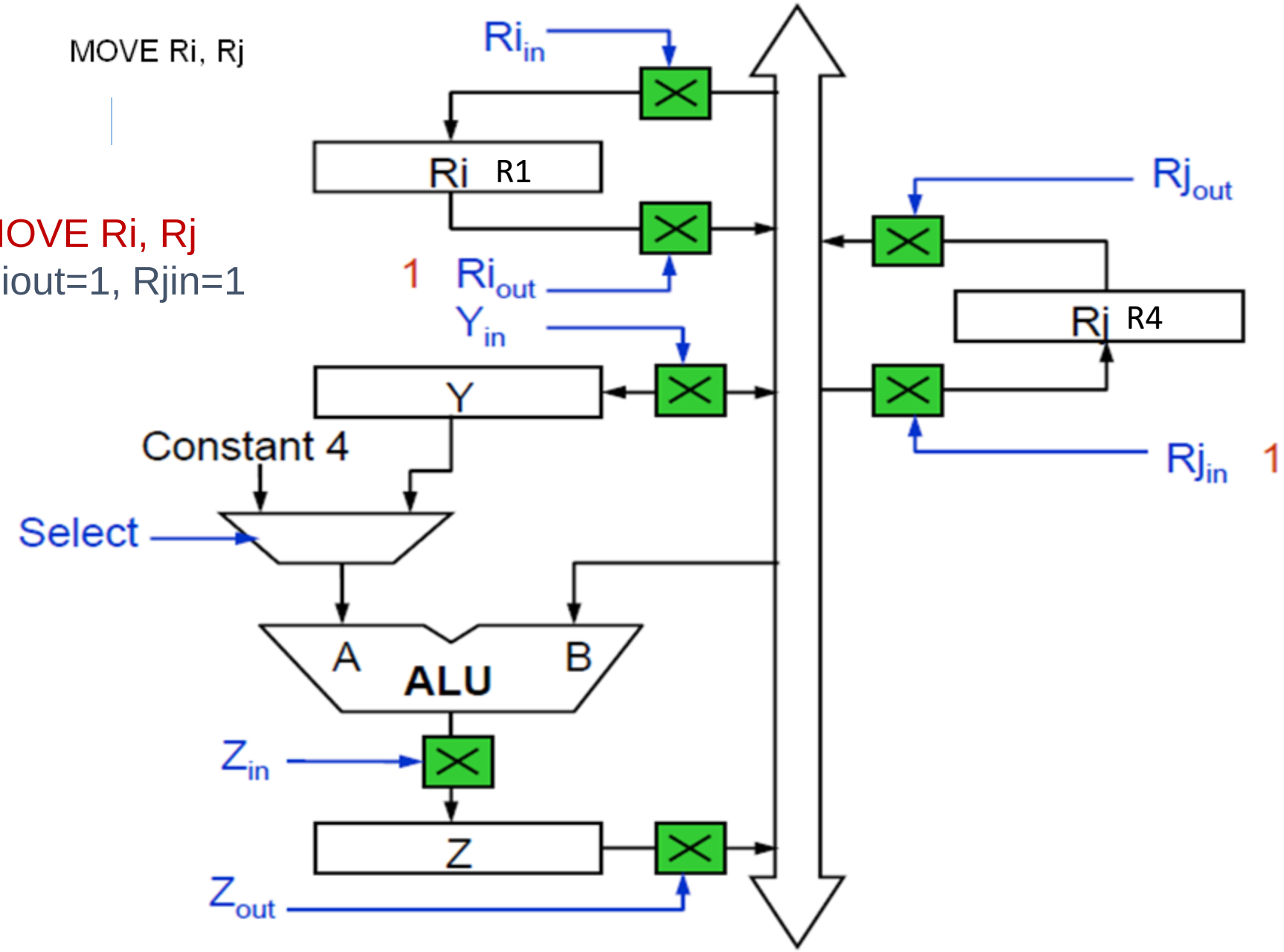
Input and output gating for one register bit.

1. Register Transfers

MOVE Ri, Rj

❖MOVE Ri, Rj

1. Riout=1, Rjin=1



- The input and output gates for register Ri are controlled by signals is R_{in} and R_{iout} .
- Initially R_{in} is set to1 – data available on common bus are loaded into Ri.
- R_{iout} is set to1 – the contents of register are placed on the bus.
- R_{iout} is set to 0 – the bus can be used for transferring data from other registers .

Data transfer between two registers:

EX: Transfer the contents of R1 to R4.

Enable output of register R1 by setting R1out=1. This places the contents of R1 on the processor bus.

Enable input of register R4 by setting R4in=1. This loads the data from the processor bus into register R4.

- MOVE R1, R4

  R1out=1, R4in=1.

2. Performing an Arithmetic or Logic Operation

- 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.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

Add R1, R2, R3

1. R1out, Yin
2. R2out, Select Y, Add, Zin
3. Zout, R3in,End

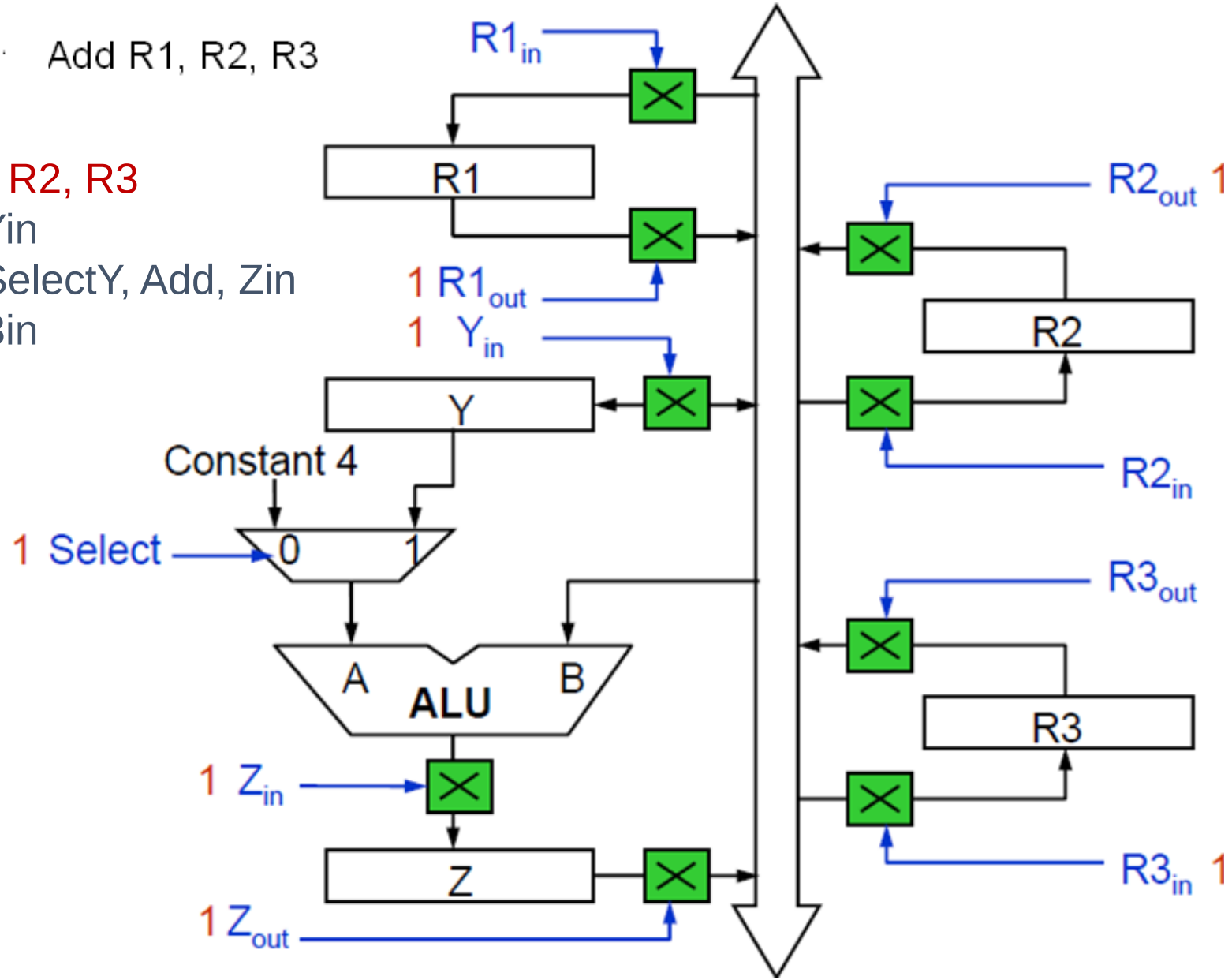
Step 1: Output of the register R1 and input of the register Y are enabled, causing the contents of R1 to be transferred to Y.

Step 2: The multiplexer's select signal is set to select Y causing the multiplexer to gate the contents of register Y to input A of the ALU.

Step 3: The contents of Z are transferred to the destination register R3.

Arithmetic Operation

❖ Add R1, R2, R3
1. R1out, Yin
2. R2out, SelectY, Add, Zin
3. Zout, R3in



Computer Organization and Architecture
CS 202

Module - 1 Part – III

Basic Processing Unit

3.Fetching a Word from Memory

- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).
- Move (R1), R2

➢ MAR ← [R1]

➢ Start a Read operation on the memory bus

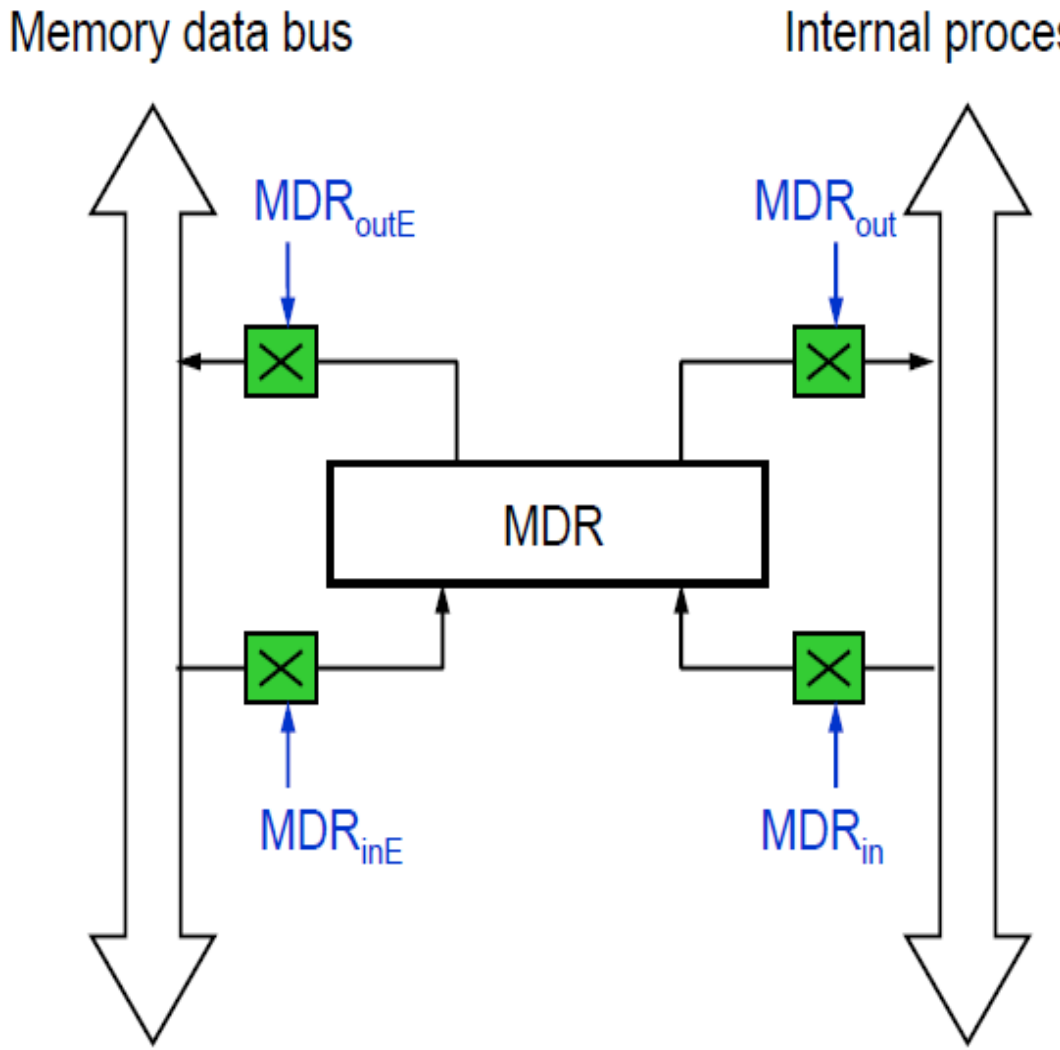
➢ Wait for the MFC response from the memory

➢ Load MDR from the memory bus

➢ R2 ← [MDR]

Fetching a Word from Memory

- Address into MAR; issue Read operation; data into MDR.



Move (R1), R2

1. R1$_{out}$, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

WMFC: is the control signal that causes the processor's control circuitry to wait for the arrival of the MFC signal.

Connection and control signals for register MDR.

Timing

Assume MAR is always available on the address lines of the memory bus.

- Move (R1), R2

1. R1out, MARin, Read
2. MDRinE, WMFC
3. MDRout, R2in

Computer Organization and Architecture
CS 202

Module - 1 Part – III

Basic Processing Unit

4. Storing a word in memory

- Address is loaded into MAR
- Data to be written loaded into MDR.
- Write command is issued.
- Example:
- Move (R1),R2

  $R1_{out}$,MAR_{in}

  $R2_{out}$,MDR_{in},Write

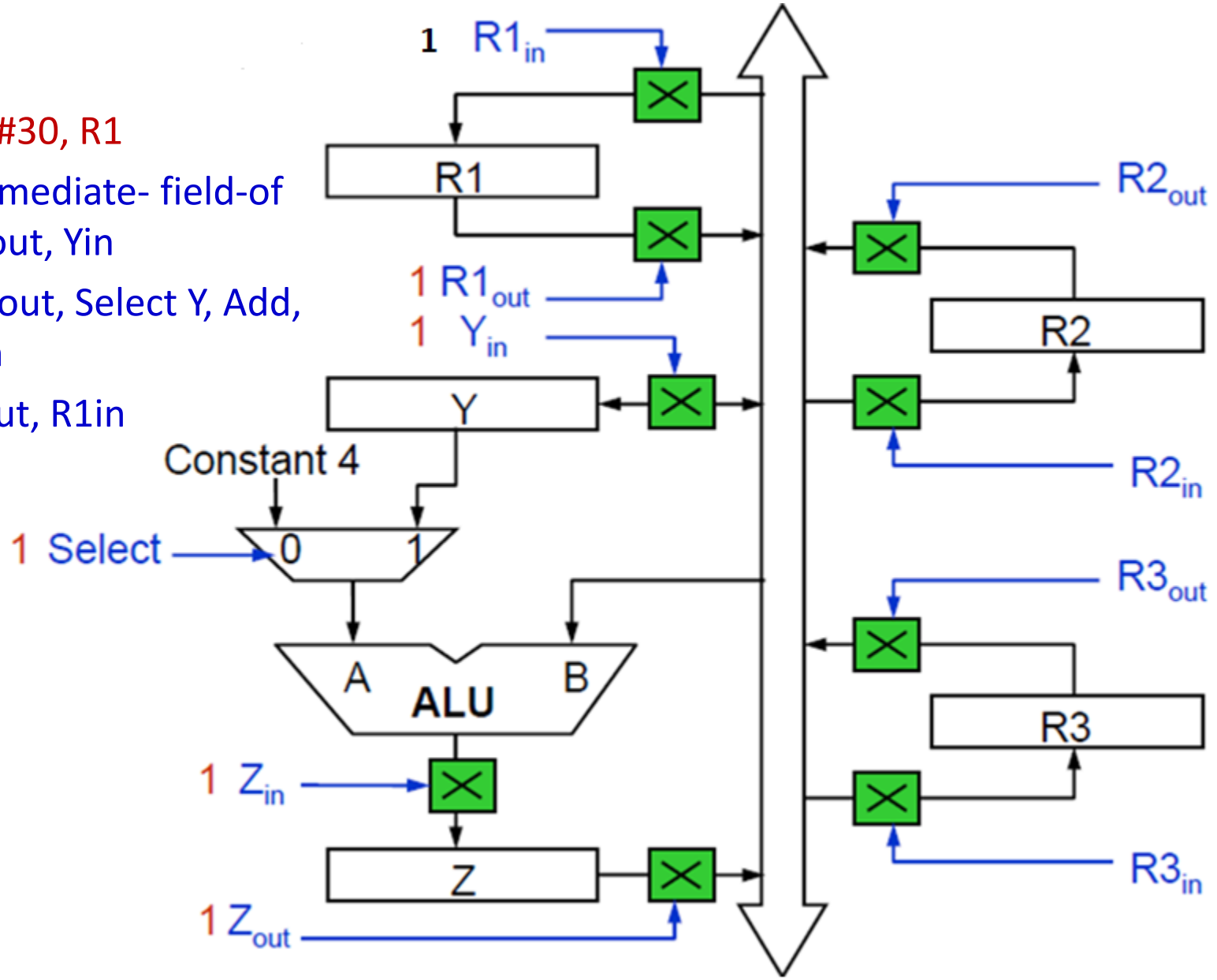
  MDR_{outE}, WMFC, End

5.Adding an immediate operand

- Add #30, R1
  1. Immediate- field-of IRout, Yin
  2. R1out, Select Y, Add, Zin
  3. Zout, R1in

Arithmetic Operation

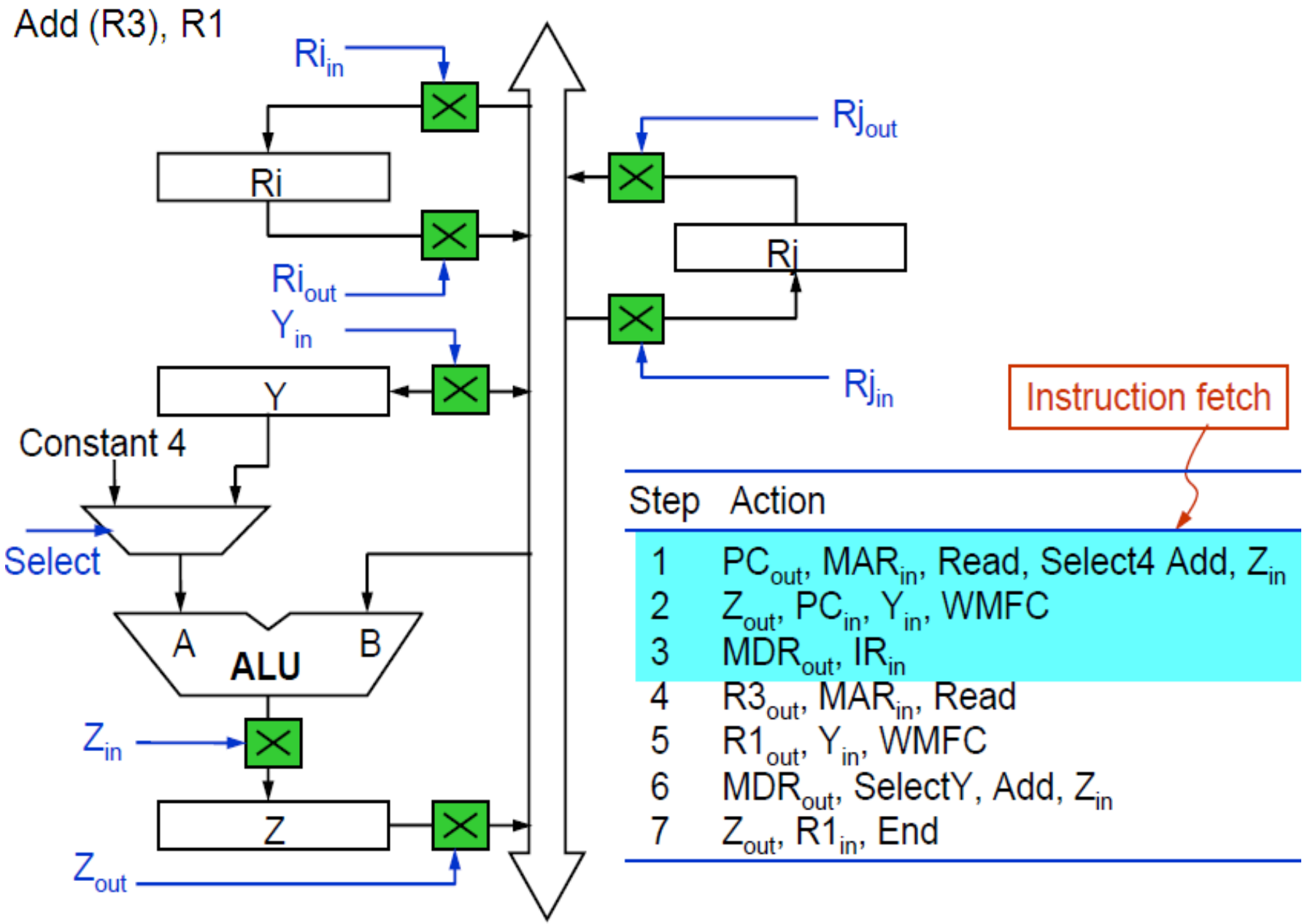
- Add #30, R1
  1. Immediate- field-of IRout, Yin
  2. R1out, Select Y, Add, Zin
  3. Zout, R1in



Execution of a Complete Instruction

- Add (R3), R1
- Fetch the instruction
- Fetch the first operand (the contents of the memory location pointed to by R3)
- Perform the addition
- Load the result into R1

Control Sequence - Execution of a Complete Instruction



Execution of Branch Instructions

❖Unconditional branch

- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- (eg:- if the branch instruction is at location 1000 and branch target-address is 1200, then the value of X must be 196, since the PC will be containing the address 1004 after fetching the instruction at location 1000).

Execution of Unconditional Branch Instructions

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Select Y, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Control sequence for an Unconditional branch instruction.

Computer Organization and Architecture
CS 202

Module - 1 Part – III

Basic Processing Unit

Execution of Branch Instructions

- The processing starts as usual, the fetch phase ends in step 3.
- In step 4, offset-value is extracted from IR by instruction-decoding circuit.
- Since updated value of PC is already available in register Y, offset X is gated onto the bus, and an addition operation is performed.
- In step 5, the result, which is the branch-address,is loaded into PC.

❖ Conditional branch

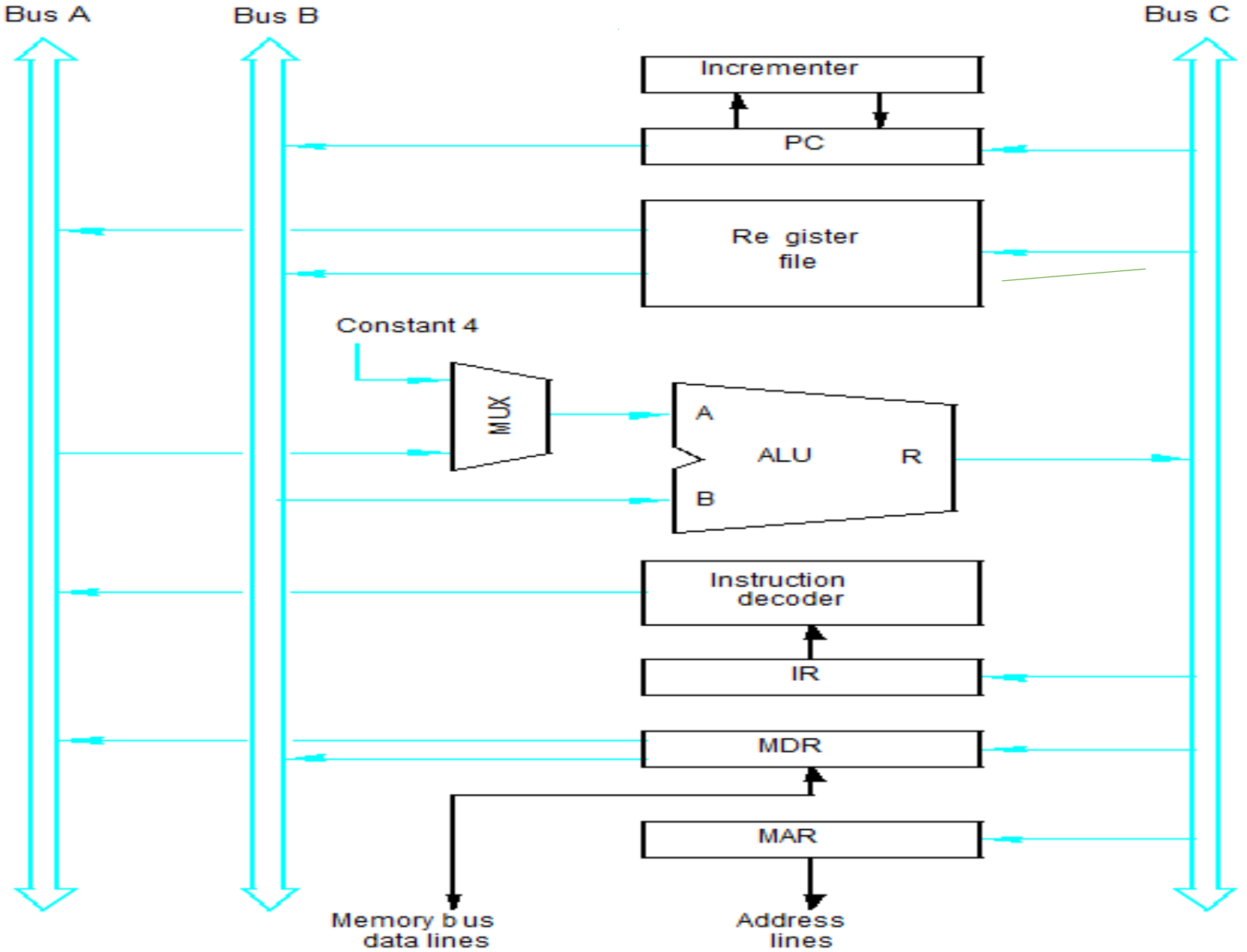
- In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC.
- e.g. Offset-field-of-IRout, Select Y, Add, Zin, If N=0 then END.
- If N=0, processor returns to step 1 immediately after step 4.
- If N=1, step 5 is performed to load a new value into PC.

Execution of Conditional Branch Instructions

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Select Y, Add, Z_{in}, If N=0, then End
5	Z_{out}, PC_{in}, End

Control sequence for an Conditional branch instruction.

Multiple-Bus Organization



Three bus Organisation of datapath

- ALU may simply pass one of its 2 input operands unmodified to bus C.
- The ALU control signals for such an operation R=A or R=B.
- Incrementer unit is used to increment the PC by 4.
- Using the incrementer eliminates the need to add the constant value 4 to the PC using the main ALU.
- The source for the constant 4 at the ALU input multiplexer can be used to increment other address such as loadmultiple & storemultiple

Multiple-Bus Organization

- Allow the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.
- Allow the data on bus C to be loaded into a third register during the same clock cycle.
- Incrementer unit.
- ALU just passes one of its two input operands unmodified to bus C.
- → control signal: R=A or R=B
- General purpose registers are combined into a single block called register file.
- 3 ports,2 output ports –access two different registers and have their contents on buses A and B
- Third port allows data on bus c during same clock cycle.
- Bus A & B are used to transfer the source operands to A & B inputs of the ALU.
- ALU operation is performed.
- The result is transferred to the destination over the bus C.

Multiple-Bus Organization

- Add R4, R5, R6

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

Figure 7.9. Control sequence for the instruction. Add R4,R5,R6, for the three-bus organization in Figure 7.8.

- Step 1:The contents of PC are passed

  through the ALU using R=B control signal & loaded into MAR to start a memory read operation

  At the same time PC is incrementer by 4
- Step 2:The processor waits for MFC
- Step 3: Loads the data ,received into MDR ,then transfers them to IR.
- Step 4: The execution phase of the instruction requires only one control step to complete.

Exercise

- What is the control sequence for execution of the instruction

  Add R1, R2

  including the instruction fetch phase? (Assume single bus architecture)

References

- Computer Organization By Carl Hamacher, Zvonko Vranesic, Safwat Zaky, fifth Edition, McGraw-Hill, ISBN 007-120411-3