Computer Organization

CSEN2203

Module I

- **Basics of Computer Organization:**
- Basic organization of the stored program computer and operation sequence for execution of a program
- Von Neumann & Harvard Architecture.
- RISC vs. CISC based architecture.
- Fetch, decode and execute cycle,
- Concept of registers and storage,
- Instruction format,
- Instruction sets
- addressing modes.

Computer Organization and Architecture

- Introduction

Computer technology has made incredible improvement in the past half century. In the early part of computer evolution, there were no stored-program computer, the computational power was less and on the top of it the size of the computer was a very huge one.

Today, a personal computer has more computational power, more main memory, more disk storage, smaller in size and it is available in affordable cost.

This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design.

In this course we will mainly deal with the innovation in computer design.

Computer architecture

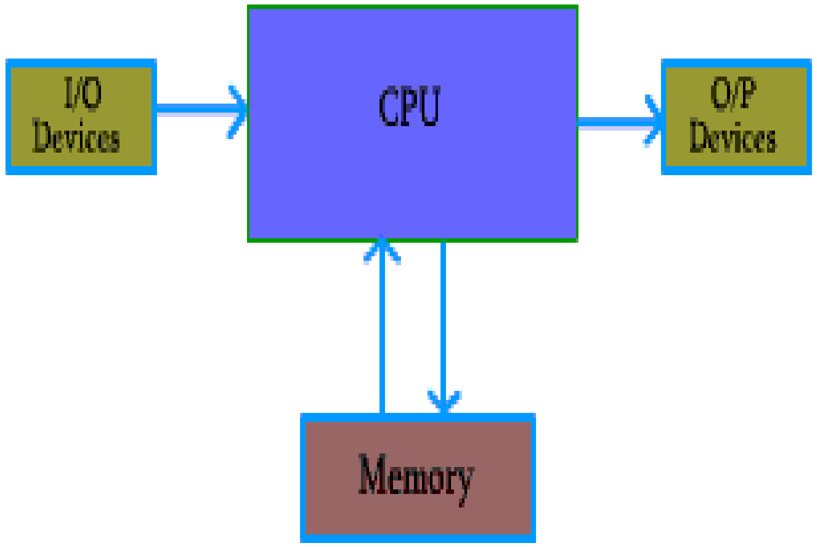
- Computer architecture refers to those parameters of a computer system that are visible to a programmer or those parameters that have a direct impact on the logical execution of a program.
- Examples of architectural attributes include the
- instruction sets
- the number of bits used to represent different data types
- I/O mechanisms
- techniques for addressing memory.

Computer organization

- Computer organization refers to the operational units and their interconnections that realize the architectural specifications.
- Examples of organizational attributes include those hardware details transparent to the programmer, such as
  - control signals,
  - interfaces between the computer and peripherals, and the memory technology used.

Figure 1.1: Basic Unit of a Computer

- **Central Processor Unit**
- **Input Unit**
- **Output Unit**
- **Memory Unit**



Central Processor Unit (CPU) :

- Central processor unit consists of two basic blocks :
  - The program control unit has a set of registers and control circuit to generate control signals.
  - The execution unit or data processing unit contains a set of registers for storing data and an Arithmetic and Logic Unit (ALU) for execution of arithmetic and logical operations.
- In addition, CPU may have some additional registers for temporary storage of data.

Input Unit

- With the help of input unit data from outside can be supplied to the computer.
- Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction.
- Example of input devices:
- Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

Output Unit

- With the help of output unit computer results can be provided to the user or it can be stored in storage device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.
- Example of output devices:
- Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

Memory Unit

- Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semi conductor memories.
- There ate two types of semiconductor memories -
- Volatile Memory : RAM (Random Access Memory).
- Non-Volatile Memory :
  - ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Secondary Memory

- Secondary memories are non volatile memory and it is used for permanent storage of data and program**.**
- Example of secondary memories
- Hard Disk, Floppy Disk, Magnetic Tape :magnetic devices
- CD-ROM: optical device
- Thumb drive (or pen drive):semiconductor memory

Basic Working Principle of a Computer

- Before going into the details of working principle of a computer, we will analyse how computers work with the help of a small hypothetical computer.
- In this small computer, we do not consider about Input and Output unit. We will consider only CPU and memory module. Assume that somehow we have stored the program and data into main memory. We will see how CPU can perform the job depending on the program stored in main memory

Arithmetic and Logic Unit (ALU) of Central Processing Unit

- Consider an ALU which can perform four arithmetic operations and four logical operations
  To distingish between arithmetic and logical operation, we may use a signal line,
- 0 - signal, represents an arithmetic operation a
- 1 - signal, represents a logical operation.
- In the similar manner, we need another two signal lines to distinguish between four arithmetic operations and four logic operations.

Operational Table

The different operations and their binary code is as follows:

Arithmatic	operations	Logical	operations
000	ADD	100	OR
001	SUB	101	AND
010	MULT	110	NAND
011	DIV	111	NOR

Consider the part of control unit, its task is to generate the appropriate signal at right moment.

Arithmetic and Logic Unit (ALU)

- There is an instruction decoder in CPU which decodes this information in such a way that computer can perform the desired task
- The simple model for the decoder may be considered that there is three input lines to the decoder and correspondingly it generates eight output lines. Depending on input combination only one of the output signals will be generated and it is used to indicate the corresponding operation of ALU.

Storage units

- In our simple model, we use three storage units in CPU,

  Two -- for storing the operands and

  one -- for storing the results.

  These storage units are known as register.
- But in computer, we need more storage space for proper functioning of the Computer.
- Some of them are inside CPU, which are known as register. Other bigger chunk of storage space is known as primary memory or main memory. The CPU can work with the information available in main memory only.

Interfacing with the Primary Memory

- To access the data from memory, we need two special registers one is known as
- Memory Data Register (MDR)
- Memory Address Register (MAR).
- Data and program is stored in main memory. While executing a program, CPU brings instruction and data from main memory, performs the tasks as per the instruction fetch from the memory. After completion of operation, CPU stores the result back into the memory.

Main Memory Organization

- Main memory unit is the storage unit, There are several location for storing information in the main memory module.
- The capacity of a memory module is specified by the number of memory location and the information stored in each location.
- A memory module of capacity 16 X 4 indicates that, there are 16 location in the memory module and in each location, we can store 4 bits of information.
- We have to know how to indicate or point to a specific memory location. This is done by address of the memory location.

Memory operations

- We need two operation to work with memory
- **READ** : This operation is to retrieve the data from memory and bring it to CPU register
- **WRITE:** This operation is to store the data to a memory location from CPU register
- With the help of one signal line, we can differentiate these two operations. If the content of this signal line is
  0, we say that we will do a READ operation; and if it is

  1, then it is a WRITE operation.
- To transfer the data from CPU to memory module and vice-versa, we need some connection. This is termed as DATA BUS

DATA BUS &ADDRESS BUS.

- The size of the data bus indicate how many bit we can transfer at a time. Size of data bus is mainly specified by the data storage capacity of each location of memory module.
- We have to resolve the issues how to specify a particular memory location where we want to store our data or from where we want to retrieve the data.
- This can be done by the memory address. Each location can be specified with the help of a binary address.
- These signal lines use to identify a memory location is termed as ADDRESS BUS.

ADDRESS BUS

- Size of address bus depends on the memory size.
- For a memory module of capacity of 2^n location, we need *n* address lines, that is, an address bus of size *n*.
- We use a address decoder to decode the address that are present in address bus
- As for example, consider a memory module of 16 location and each location can store 4 bit of information.
  - The size of address bus is 4 bits
  - size of the data bus is 4 bits
  - The size of address decoder is 4 X 16.

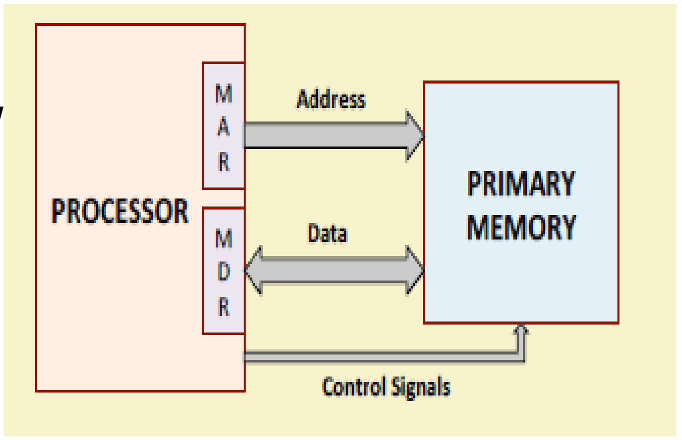
control signals for read or write

- There is a control signal named R/W.
  If R/W = 0, indicates READ operation
- if R/W = 1, indicates WRITE operation

- If the contents of address bus is 0101 and contents of data bus is 1100 and R/W = 1, then 1100 will be
  written in location 5.
- If the contents of address bus is 1011 and R/W=0, then the contents of location 1011 will be placed in data bus.

Interfacing with the Primary Memory

- To read data from memory
  - Load the memory address into MAR.
  - Issue the control signal *READ.*
  - The data read from the memory is stored into MDR.
- To write data into memory
  - Load the memory address into MAR.
  - Load the data to be written
  - Issue the control signal *WRI*



Registers

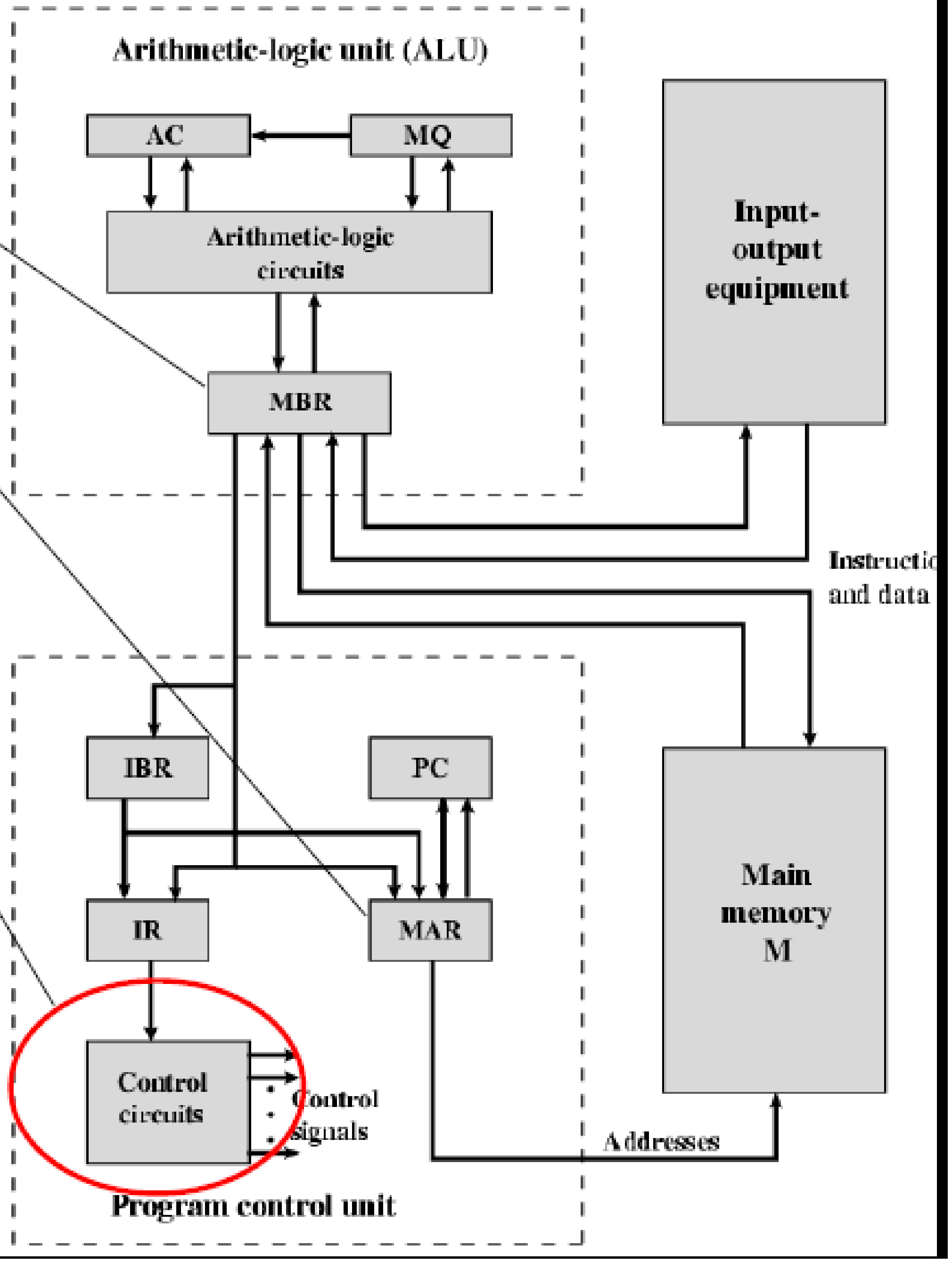
- The memory has 4096 words in it
- 4096 = 2^{12}, so it takes 12 bits to select a word in memory
- Each word is 16 bits long
- Size of MAR=12 bits
- Size of MDR=16bits
- Two special-purpose registers are used:
- ***Program Counter (PC):*** *Holds the memory address of the next* instruction to be executed.
- Automatically incremented to point to the next instruction when an instruction is being executed.
- ***Instruction Register (IR):*** *Temporarily holds an instruction that has* been fetched from memory.
- Need to be decoded to find out the instruction type.
- Also contains information about the location of the data.

IAS organization

Memory Buffer Register either sends data to or receives data from Mem. or I/O

Memory Address Register specifies which Mem. location will be read or written next

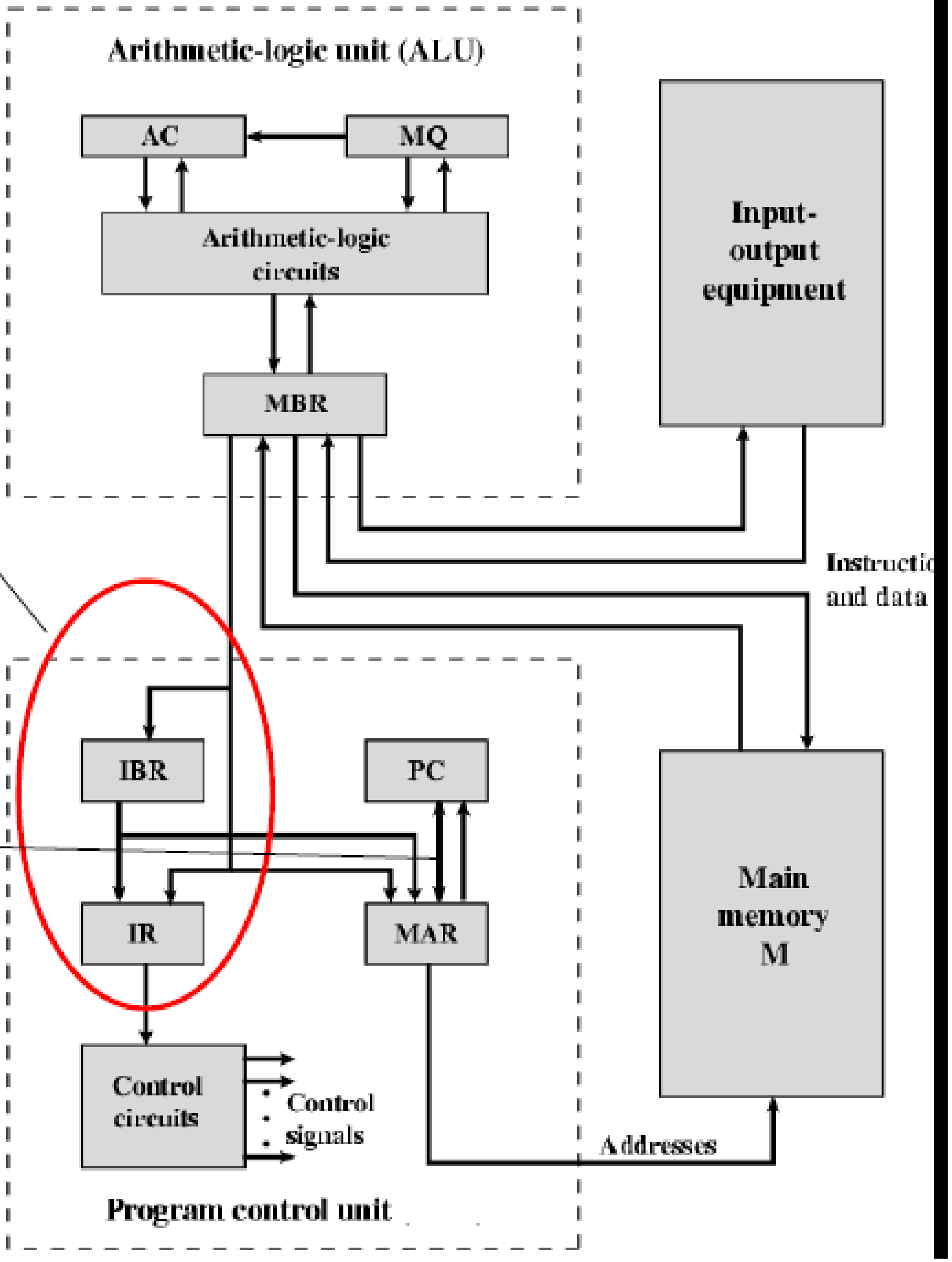
Control signals are set by the opcode part of the instruction bits.
Examples:
• Bring a new instruction from Mem. (fetch)
• Perform an addition (execute)



IAS organization

Why do we need both a register and a buffer register to hold instructions?

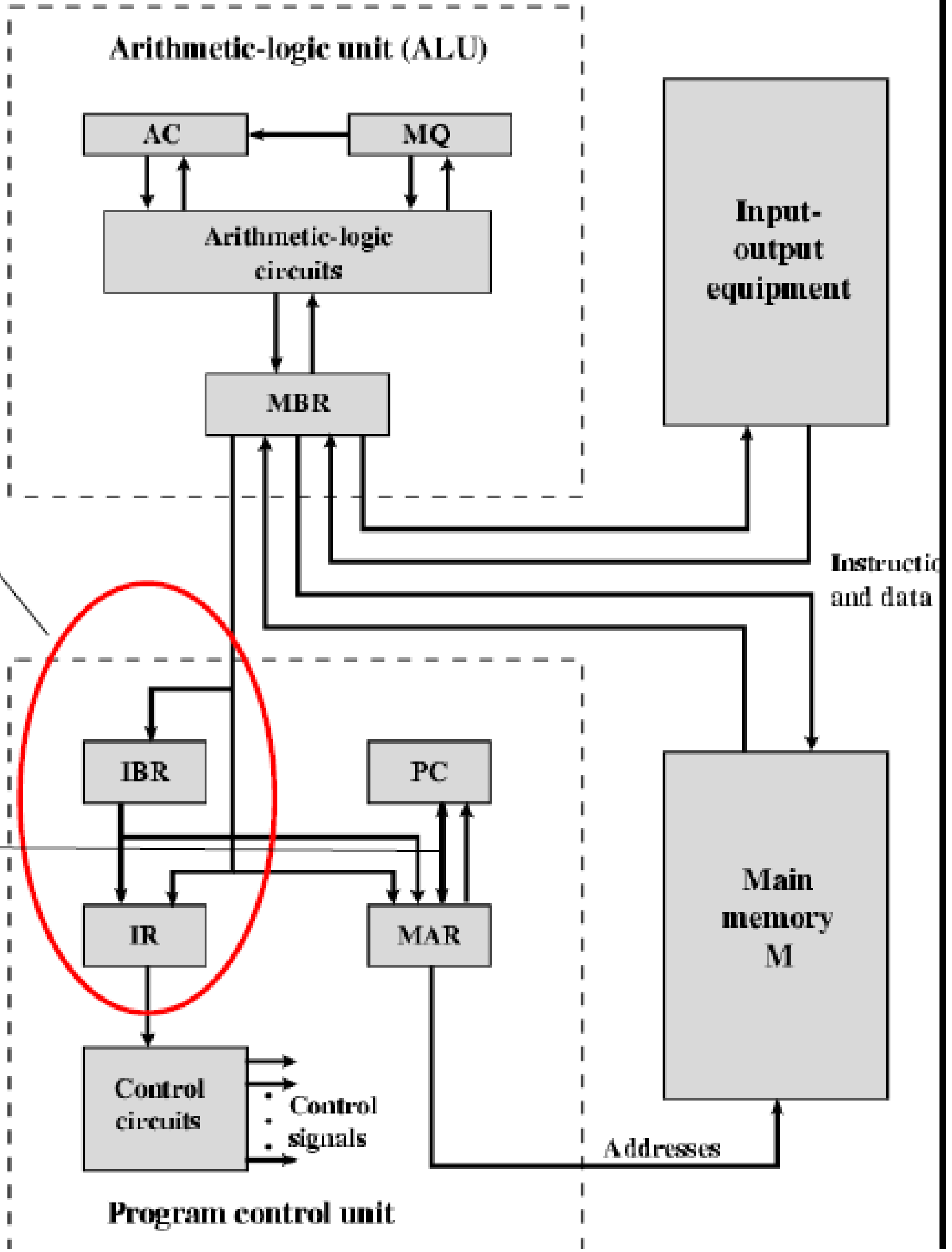
Why does the arrow between PC and MAR point both ways?



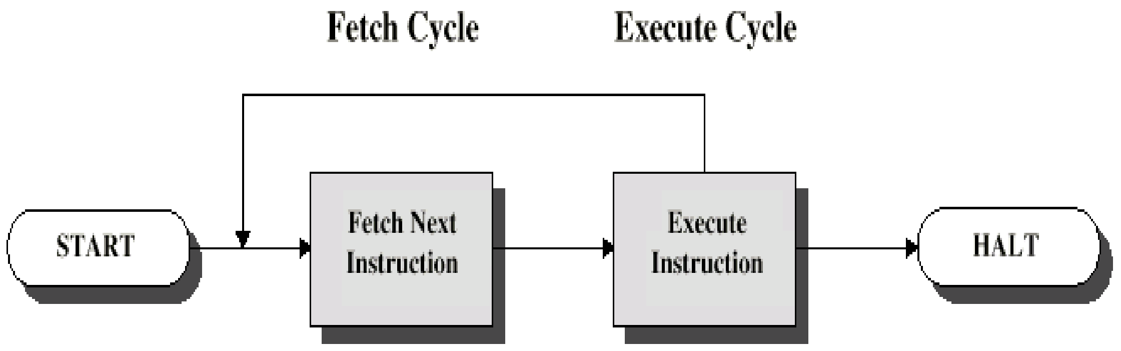
IAS organization

Hint: 2 instructions are stored in each memory word

Hint: the next instruction can be found either sequentially, or through a branch (jump)



Instruction Cycle



- Two steps:
  - Fetch
  - Execute
- **Fetch Cycle**
  - Program Counter (PC) holds address of next instruction to fetch
  - Processor fetches instruction from memory location pointed to by PC
  - Increment PC
    - Unless told otherwise
  - Instruction loaded into Instruction Register (IR)
  - Processor interprets instruction and performs required actions

IAS –the FETCH-EXECUTE cycle

- **Each instruction is executed in the same two-step manner:**
- FETCH load the binary code of the instr. from Memory (or IBR)
- Opcode goes into IR
- Address goes into MAR

- EXECUTE send appropriate control signals to do what the instr. needs to do

Execute Cycle

- Processor-memory
  - data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
  - Combination of above

IAS – instruction set (architecture!)

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD –M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD –|M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP+ M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)

Specifies one of 21 instructions

There was no assembly language back then!

IAS – instruction set (continued)

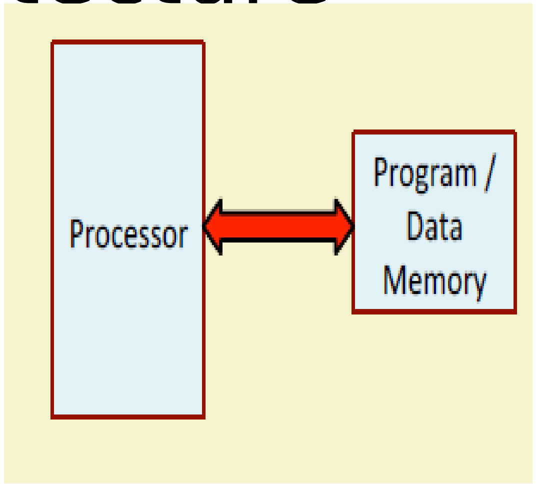
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2; i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

Input Output

- Input devices are used to put the information into computer. With the help of input devices we can store information in memory so that CPU can use it. Program or data is read into main memory from input device or secondary storage under the control of CPU input instruction.
- Output devices are used to output the information from computer. If some results are evaluated by computer and it is stored in computer, then with the help of output devices, we can present it to the user. Output data from the main memory go to output device under the control of CPU output instruction.

von-Neumann Architecture

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- More flexible and easier to implement.
- Suitable for most of the general purpose



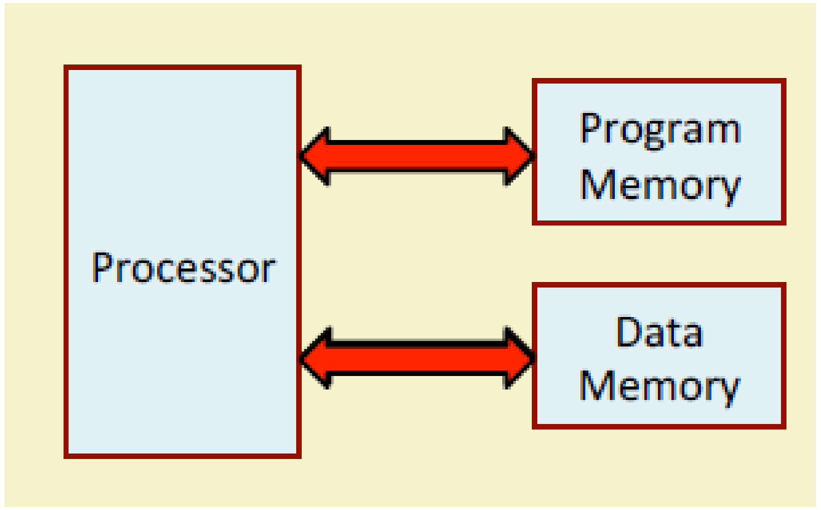
- Disadvantage:
- The processor-memory bus acts as the bottleneck.
- All instructi**ons** and data are moved back and forth through the pipe

von-Neumann Architecture

- **How can this be reduced?**
- This performance problem is reduced by using cache memory.(details in memory part)
- Using RISC architecture as it uses less number of memory reference instruction and uses large number of registers.

Harvard Architecture

- Separate memory for program and data.
- Instructions are stored in program memory
- Data are stored in data memory.
- Instruction and data accesses can be done in parallel.
- Some microcontrollers and pipelines with separate instruction and data caches follow this concept.
- The processor-memory bottleneck remains.



Instruction Set & Addressing

- Various addressing modes
- Machine Instruction
- Instruction Format

Instructions

- **Program**
  - **A sequence of (machine) instructions**
- **(Machine) Instruction**
  - **A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)**
- **The instructions of a program, along with any needed data are stored in memory**
- **The CPU reads the next instruction from memory**
- **It is placed in an *Instruction Register* (IR)**
- **Control circuitry in control unit then translates the instruction into the sequence of micro operations necessary to implement it**

Instruction Set Architecture (ISA)

- **Serves as an interface between software and hardware.**
- **Typically consists of information regarding the programmer’s view of the architecture (i.e. the registers, address and data buses, etc.).**
- **Also consists of the instruction set.**
- **Many ISA’s are not specific to a particular computer architecture.**
- **They survive across generations.**
- **Classic examples: IBM 360 series, Intel x86 series, etc.**

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed

Instruction Addressing

- **We have examined the types of** *operands* and *operations* that may be specified by *machine instructions*. Now we have to see how is the address of an operand specified, and how are the *bits* of an instruction organized to define the *operand addresses* and operation of that instruction.

Addressing Modes:

- The most common addressing techniques are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement
  - Stack

Addressing Modes:

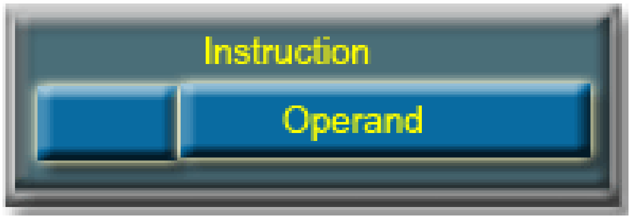
- All computer architectures provide more than one of these *addressing modes*. The question arises as to how the *control unit* can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different *opcodes* will use different addressing modes. Also, one or more bits in the *instruction format* can be used as a *mode field*. The value of the mode field determines which addressing mode is to be used.
- What is the interpretation of *effective address*. In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer.

Addressing Modes:

- To explain the addressing modes, we use the following notation:
- **A**=contents of an address field in the instruction that refers to a memory
- **R**=contents of an address field in the instruction that refers to a register
- **EA**=actual (effective) address of the location containing the referenced operand
- **(X)**=contents of location X

Immediate Addressing:

- The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:
- OPERAND   =   A
- This mode can be used to define and use constants or set initial values of variables.



Immediate Addressing:

- The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand.
- The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.

Direct Addressing:

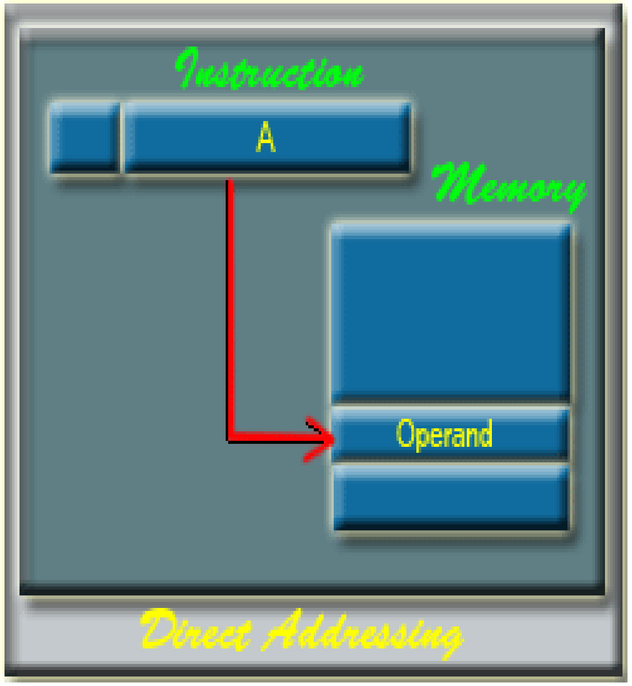
- A very simple form of addressing is direct addressing, in which the address field contains the effective add
operand:
- EA = A
- It requires only

one memory reference

and no special calculation.

- Here, 'A' indicates the me

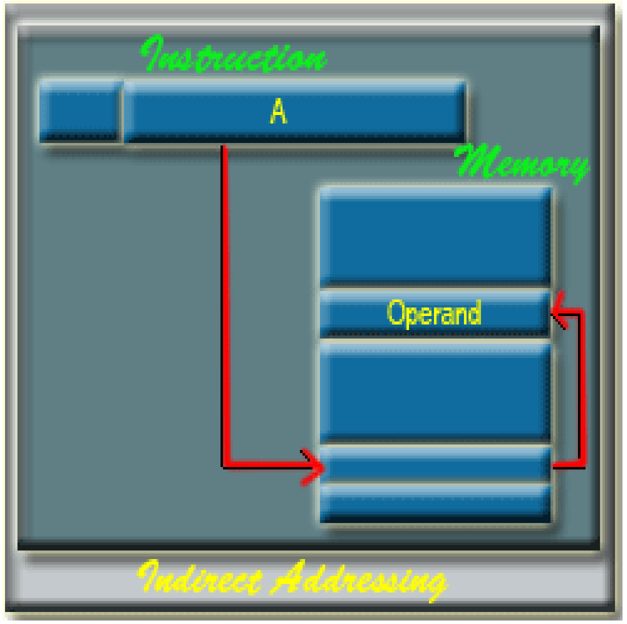
address field for the operand.



Indirect Addressing:

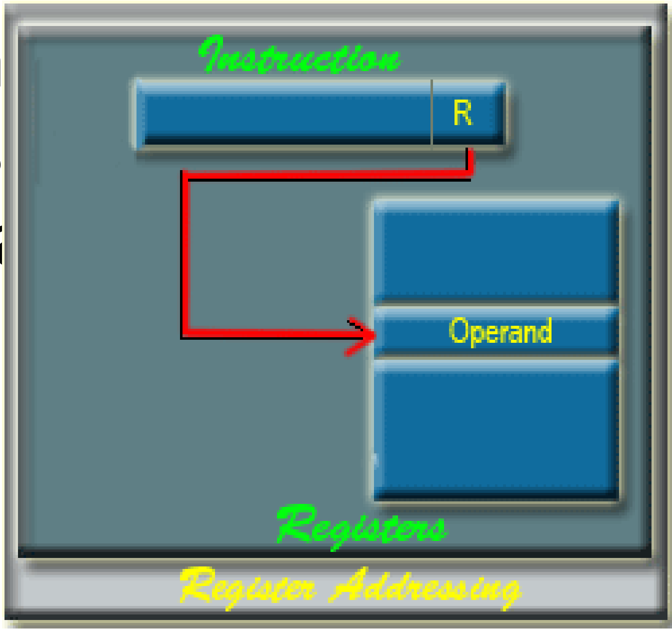
- With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is know as indirect addressing:
- EA = (A)

Here 'A' indicates the memory address field of the required operands.



Register Addressing:

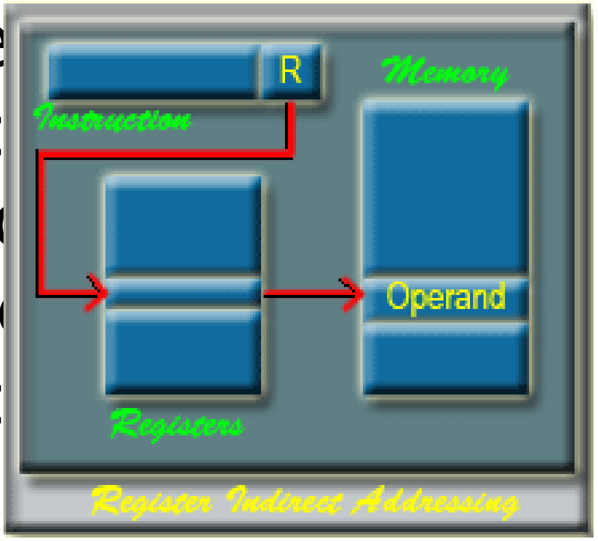
- Register addressing is similar to direct addressing. The only difference is that the address field referes to a register rather than a main memory address:
- EA = R
- The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required.
- The disadvantage of register addressing is that the address space is very limited.
- 'R' in regis opera



Register Indirect Addressing:

- Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location.
  It requires only one memory reference and no special calculation.
- EA = (R)
- Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address.

- From that memory location, we use to get the data or information. In
  ge
  ac
  me
  th
  ac

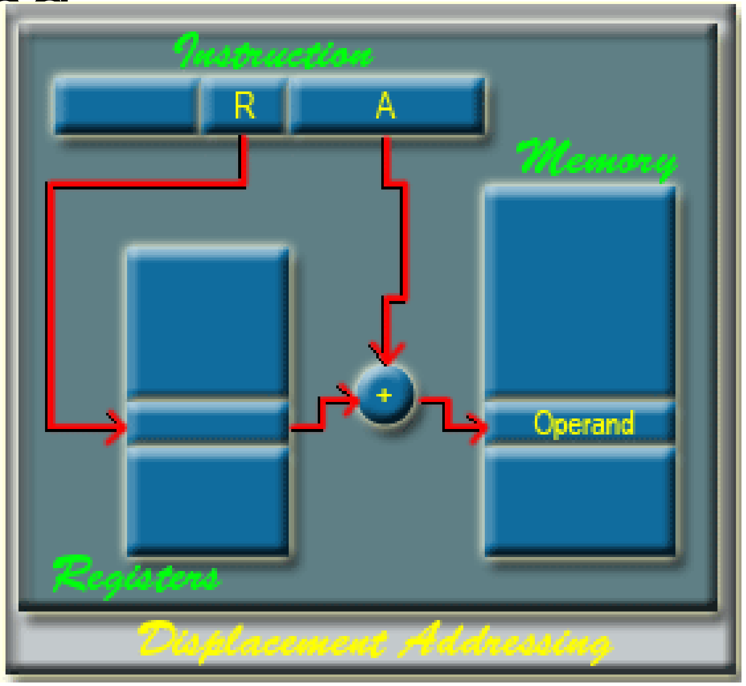


Displacement Addressing:

- A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing, which is broadly categorized as displacement addressing:
- EA = A + (R)
- Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly.
- The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

Displacement Addressing:

- Three of the most common use of displacement addressing are:
- Relative addressing
- Base-register addressing
- Indexing



Displacement Addressing:

- **Relative Addressing:**
- It implicitly referenced register is the *program counter* (PC).
- That is, the current instruction address is added to the address field to produce the EA.
- Thus, the effective address is a displacement relative to the address of the

- **Base-Register Addressing:**
- The reference register contains a memory address, and the address field contains a displacement from that address. The register reference may be *explicit* or *implicit*.
- In some implementation, a single segment/base register is employed and is used implicitly. In others, the programmer may choose a register to hold the base address of a segment, and the instruction must reference it explicitly.

Indexing:

- The address field references a main memory address, and the reference register contains a positive displacement from that address. In this case also the register reference is sometimes explicit and sometimes implicit.
- Generally index register are used for iterative tasks, it is typical that there is a need to increment or decrement the index register after each reference to it. Because this is such a common operation, some system will automatically do this as part of the same instruction cycle.

- This is known as *auto-indexing.* We may get two types of auto-indexing:
  - -- auto-incrementing
  - -- auto-decrementing.
- If certain registers are devoted exclusively to indexing, then auto-indexing can be invoked implicitly and automatically.
- If general purpose register are used, the auto index operation may need to be signaled by a bit in the instruction.

- Auto-indexing using *increment* can be depicted as follows:

$$EA = (R)$$
$$R = (R) + 1$$

- Auto-indexing using *decrement* can be depicted as follows:
- $R = (R) - 1$
- $EA = (R)$

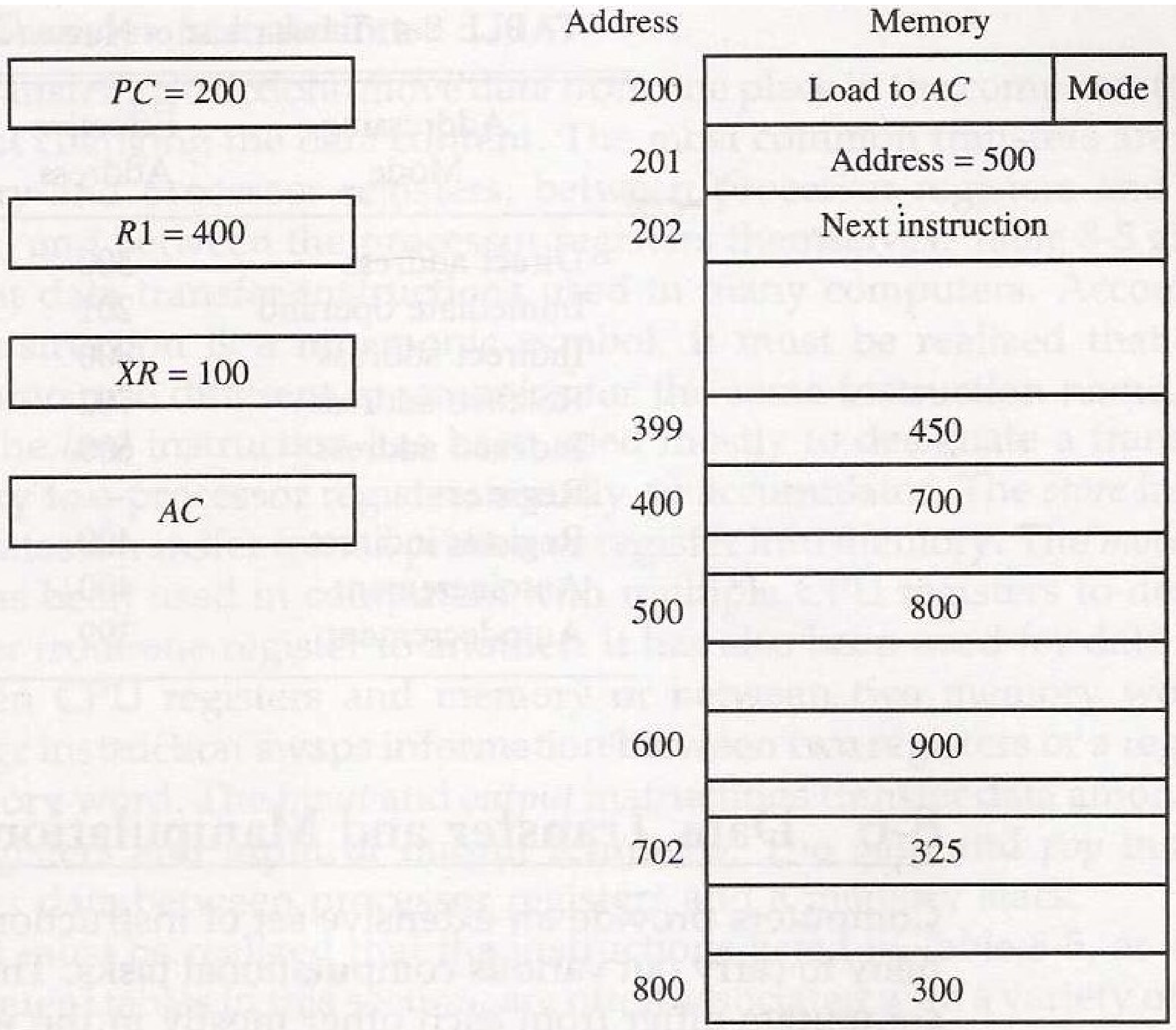
- In some machines, both *indirect addressing* and *indexing* are provided, and it is possible to employ both in the same instruction. There are two possibilities: The indexing is performed either before or after the indirection.
- If indexing is performed after the indirection, it is termed *postindexing*
- EA = (A) + (R)
- First, the contents of the address field are used to access a memory location containing an address. This address is then indexed by the register value.
- With preindexing, the indexing is performed before the indirection:
- EA = (A + (R))
- An address is calculated, the calculated address contains not the operand, but the address of the operand.

Stack Addressing:

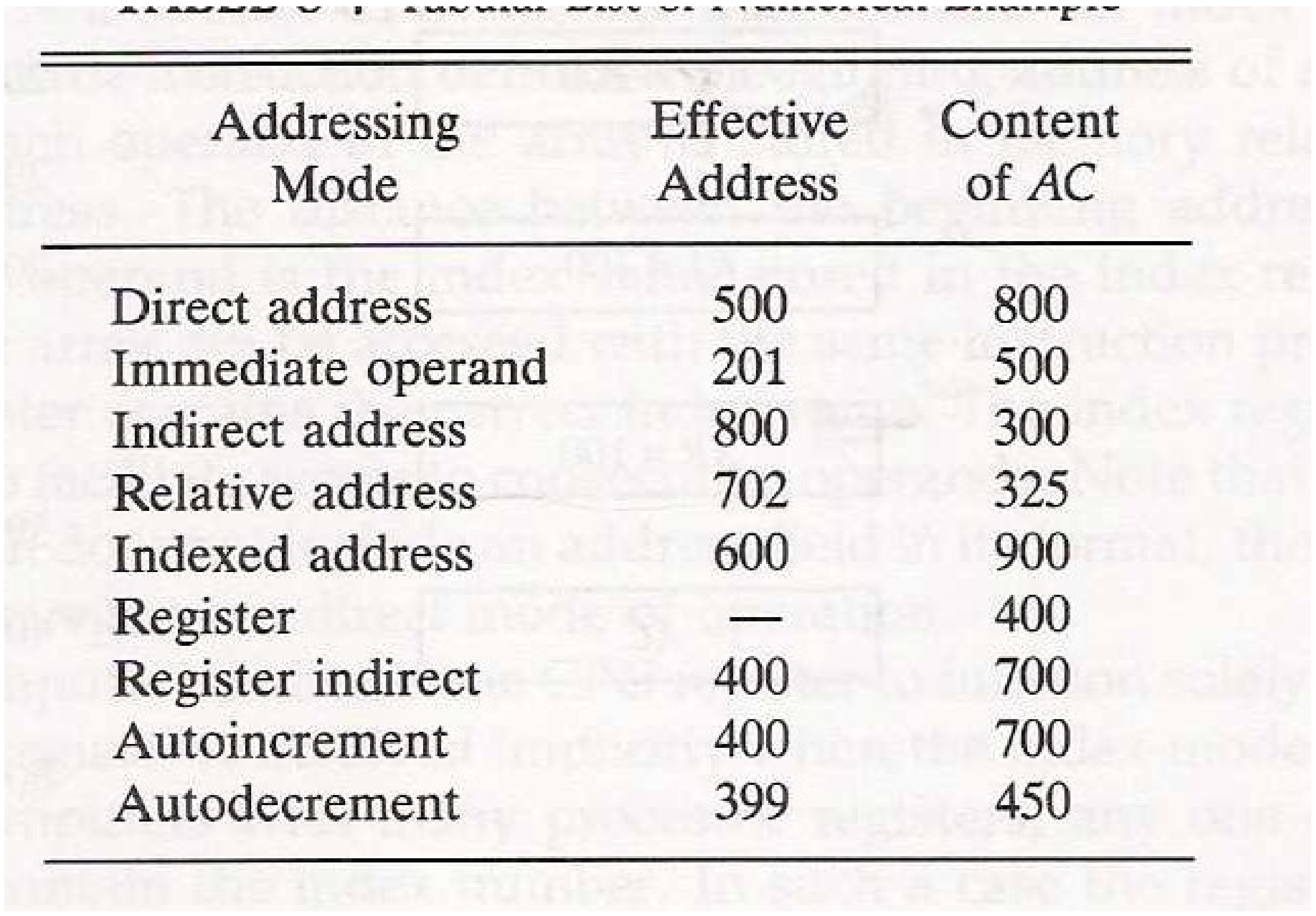
- A stack is a linear array or list of locations. It is sometimes referred to as a *pushdown list* or *last-in-first-out queue*. A stack is a reserved block of locations. Items are appended to the top of the stack so that, at any given time, the block is partially filled. Associated with the stack is a pointer whose value is the address of the top of the stack. The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact register indirect addresses.
- The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

Questions

- A two address instruction available at memory location 200. 1st word ” LOAD AC” and mode stored at location 200 with its address field at location 201. The address field has the value 500. Pc has the value 200 for fetching the instruction .The processor register R1 contains the number 400, index register has value 100. AC receives the operand after the instruction executed.
- Evaluate the effective address of operand and content of accumulator after this statement if the addressing mode of the instruction is as follows **i)** direct **ii)** immediate **iii)** relative **iv)** register indirect **v)**index vi)Register vii)Auto increment viii) Auto decrement



Address
Memory
PC = 200
R1 = 400
XR = 100
AC
200
Load to AC
Mode
201
Address = 500
202
Next instruction
399
450
400
700
500
800
600
900
702
325
800
300



Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Addressing Modes

- The different ways in which the location of an operand is specified in an instruction are referred to as 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*]

questions

- Registers R1 and R2 contain decimal values 1200 and 4600. What is the effective address of the memory operand and the type of addressing modes used in each of the following instructions?
- a) load 20(R1), R5
- b) move #3000, R5
- c) store R5, 30(R1,R2)
- d) add –(R2) , R5
- e) sub (R1)+, R5

Questions and answer

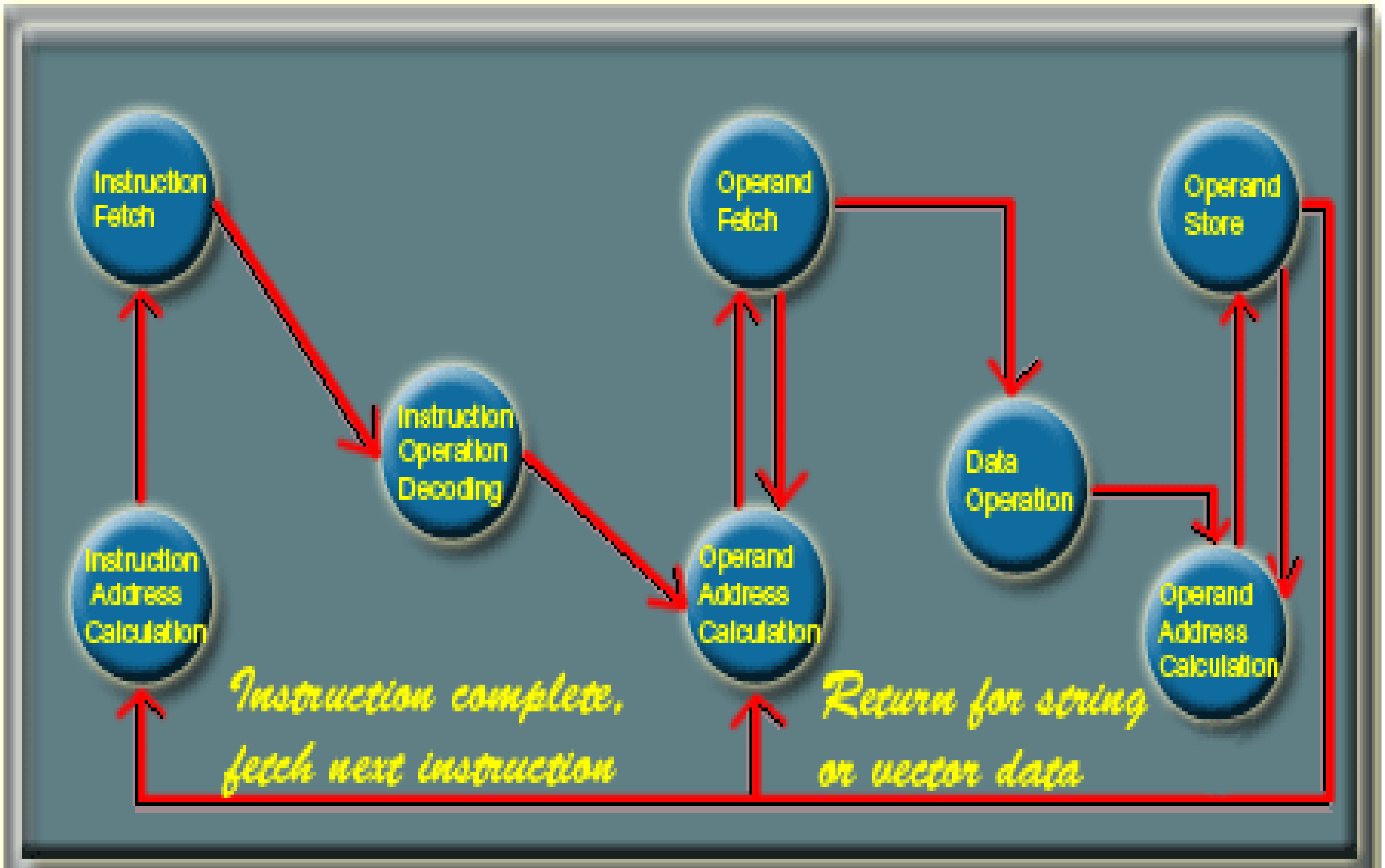
- Registers R1 and R2 contain decimal values 1200 and 4600. What is the effective address of the memory operand and the type of addressing modes used in each of the following instructions?
- a) load 20(R1), R5// index EA=1220
- b) move #3000, R5// immediate R5<-3000
- c) store R5, 30(R1,R2) // base with index and offset
    - //EA=5830
- d) add –(R2) , R5 //Auto decrement EA=4599
- e) sub (R1)+, R5 //auto increment EA=4600

- **Machine Instruction**
  The operation of a CPU is determine by the instruction it executes, referred to as machine instructions or computer instructions. The collection of different instructions is referred as the *instruction set of the CPU*.
- Each instruction must contain the information required by the CPU for execution. The elements of an instruction are as follows:
- **Operation Code:**
  Specifies the operation to be performed (e.g., add, move etc.). The operation is specified by a binary code, know 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 reference:**
  The operation may produce a result.

- **Next instruction reference:**
  This tells the CPU where to fetch the next instruction after the execution of this instruction is complete.

- The next instruction to be fetched is located in main memory. But in case of virtual memory system, it may be either in main memory or secondary memory (disk). In most cases, the next instruction to be fetched immediately follow the current instruction. In those cases, there is no explicit reference to the next instruction.When an explicit reference is needed, then the main memory or virtual memory address must be given.

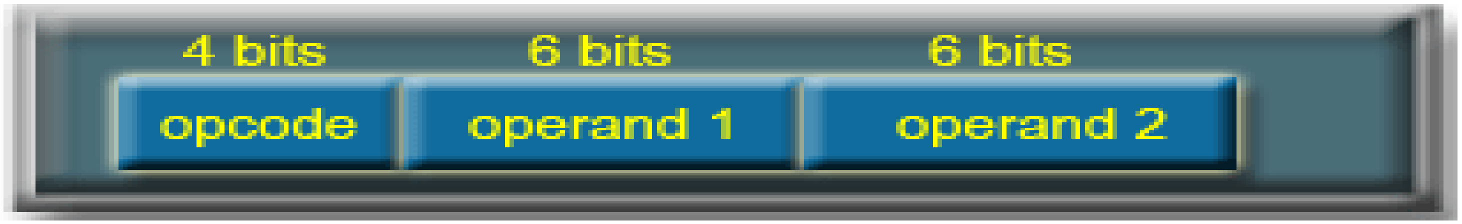
- Source and result operands can be in one of the three areas:
  - main or virtual memory,
  - CPU register or
  - I/O device.


Instruction Fetch
Instruction Operation Decoding
Operand Fetch
Data Operation
Operand Store
Instruction Address Calculation
Operand Address Calculation
Operand Address Calculation
Instruction complete, fetch next instruction
Return for string or vector data

Steps involved in instruction execution

Instruction Representation

- Within the computer, each instruction is represented by a sequence of bits.
- The instruction is divided into fields, corresponding to the constituent elements of the instruction.
- The instruction format is highly machine specific and it mainly depends on the machine architecture



- A simple example of an instruction format is shown in the Figure .
- It is assume that it is a 16-bit CPU.
- 4 bits are used to provide the operation code. So, we may have to 16 ($2^4 = 16$) different set of instructions.
- With each instruction, there are two operands.
- To specify each operands, 6 bits are used.
- It is possible to provide 64 ($2^6 = 64$) different operands for each operand reference.

Instruction Types

- The instruction set of a CPU can be categorized as follows:
- **Data Processing:**
  - Arithmatic and Logic instructions Arithmatic instructions provide computational capabilities for processing numeric data. Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers.
  - Logic instructions thus provide capabilities for processing any other type of data. There operations are performed primarily on data in CPU registers.
- **Data Storage:**
  - Memory instructions are used for moving data between memory and CPU registers.

Instruction Types

- **Data Movement:**
  - I/O instructions are needed to transfer program and data into memory from storage device or input device and the results of computation back to the user.
- **Control:**
  - Test and branch instructions
    Test instructions are used to test the value of a data word or the status of a computation. Branch instructions are then used to branch to a different set of instructions depending on the decision made.

Number of Addresses

- What is the maximum number of addresses one might need in an instruction?
- Most of the arithmantic and logic operations are either *unary* (one operand) or *binary* (two operands).
- Thus we need a maximum of two addresses to reference operands.
- The result of an operation must be stored, suggesting a third address.
- Finally after completion of an instruction, the next instruction must be fetched, and its address is needed.
- This reasoning suggests that an instruction may require to contain *four address references:* two operands, one result, and the address of the next instruction. In practice, four address instructions are rare. Most instructions have one, two or three operands addresses, with the address of the next instruction being implicit (obtained from the program counter).

Instruction Set Design

- One of the most interesting, and most analyzed, aspects of computer design is instruction set design. The instruction set defines the functions performed by the CPU.
- The instruction set is the programmer's means of controlling the CPU.
- Thus programmer requirements must be considered in designing the instruction set.

Instruction Set Design

- **Operations performed :**How many and which operations to provide, and how complex operations should be.
- **Data Types :**The various type of data upon which operations are performed.
- **Instruction format:** Instruction length (in bits), number of addresses, size of various fields and so on.
- **Registers:** Number of CPU registers that can be referenced by instructions and their use.
- **Addressing:** The mode or modes by which the address of an operand is specified.

Types of Operands

- Machine instructions operate on data. Data can be categorized as follows :
- **Addresses:** It basically indicates the address of a memory location. Addresses are nothing but the unsigned integer, but treated in a special way to indicate the address of a memory location. Address arithmetic is somewhat different from normal arithmetic and it is related to machine architecture.
- **Numbers:** All machine languages include numeric data types. Numeric data are classified into two broad categories: integer or fixed point and floating point.

- **Characters:** A common form of data is text or character strings. Since computer works with bits, so characters are represented by a sequence of bits. The most commonly used coding scheme is ASCII (American Standard Code for Information Interchange) code.
- **Logical Data:** Normally each word or other addressable unit (byte, halfword, and so on) is treated as a single unit of data. It is sometime useful to consider an *n*-bit unit as consisting of *n* 1-bit items of data, each item having the value 0 or 1. When data are viewed this way, they are considered to be logical data. Generally 1 is treated as true and 0 is treated as false.

Types of Opearations

- The number of different *opcodes* and their types varies widely from machine to machine. However, some general type of operations are found in most of the machine architecture. Those operations can be categorized as follows:
  - **Data Transfer**
  - **Arithmetic**
  - **Logical**
  - **Conversion**
  - **Input Output [I/O]**
  - **System Control**
  - **Transfer Control**

Commonly used data transfer operation:

Operation Name	Description
Move (Transfer)	Transfer word or block from source to destination
Store	Transfer word from processor to memory
Load (fetch)	Transfer word from memory to processor
Exchange	Swap contents of source and destination
Clear (reset)	Transfer word of 0s to destination
Set	Transfer word of 1s to destination
Push	Transfer word from source to top of stack
Pop	Transfer word from top of stack to destination

arithmetic operations

Operation Name	Description
Add	Compute sum of two operands
Subtract	Compute difference of two operands
Multiply	Compute product of two operands
Divide	Compute quotient of two operands
Absolute	Replace operand by its absolute value
Negate	Change sign of operand
Increment	Add 1 to operand
Decrement	Subtract 1 from operand

logical operations

Operation Name	Description
AND	Performs the logical operation AND bitwise
OR	Performs the logical operation OR bitwise
NOT	Performs the logical operation NOT bitwise
Exclusive OR	Performs the specified logical operation Exculsive-OR bitwise
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmatic comparison Set flag(s) based on outcome
Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control etc.
Shift	Left (right) shift operand, introducing constant at end
Rotate	Left (right) shift operation, with wraparound end

I/O operations are:

Operation Name	Description
Input (Read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (Write)	Transfer data from specified source to I/O port or device.
Start I/O	Transfer instructions to I/O processor to initiate I/O operation.
Test I/O	Transfer status information from I/O system to specified destination

- **Conversion:**
  - Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.
- **System Control:**
  - System control instructions are those which are used for system setting and it can be used only in privileged state. Typically, these instructions are reserved for the use of operating systems. For example, a system control instruction may read or alter the content of a control register. Another instruction may be to read or modify a storage protection key.

Branch Instruction

- A branch instruction, also called a jump instruction, has one of its operands as the address of the next instruction to be executed.
- Basically there are two types of branch instructions:
  - Conditional Branch instruction
    - the branch is made only if a certain condition is met. Otherwise, the next instruction in sequence is executed.
  - unconditional branch instruction.
    - the branch is made by updating the program counter to address specified in operand.

Branch Instruction

- BRP X ; Branch to location X if result is positive
  BRN X ; Branch to location X if result is negative
  BRZ X ; Branch to location X is result is zero
  BRO X ; Branch to location X if overflow occurs
- For example,
- BRE R1, R2, X ; Branch to X if contents of R1 = Contents of R2.

Instruction Format:

- An instruction format defines the layout of the bits of an instruction, in terms of its constituents parts.
- An instruction format must include an opcode and, implicitly or explicitly, zero or more operands.
- Each explicit operand is referenced using one of the addressing mode that is available for that machine.
- The format must, implicitly or explicitly, indicate the addressing mode of each operand

Instruction Format: CPU Design

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Format:

- Three-Address Instructions
  - ADD R1, R2, R3 $R1 \leftarrow R2 + R3$
  - Three address instruction formats are not common because they require a relatively long instruction format to hold the three address reference.

- Two-Address Instructions
  - ADD R1, R2 $R1 \leftarrow R1 + R2$
  - With two address instructions, and for binary operations, one address must do double duty as both an operand and a result.

Instruction Format

- One-Address Instructions
  - ADD M   AC ← AC + M[AR]
  - In one address instruction format, a second address must be implicit for a binary operation. For implicit reference, a processor register is used and it is termed as accumulator(AC). the accumulator contains one of the operands and is used to store the result.
- Zero-Address Instructions
  - ADD   TOS ← TOS + (TOS – 1)
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store

Three-Address

- Consider a simple arithmetic expression to evaluate:
  X= (A + B) * (C + D)

Example: Evaluate (A+B) ∗ (C+D)

- 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

Two-Address

Example: Evaluate (A+B) ∗ (C+D)

- 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
```

One-Address

Example: Evaluate (A+B) ∗ (C+D)

- 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

Zero-Address

Example: Evaluate (A+B) ∗ (C+D)

- Zero-Address

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

RISC

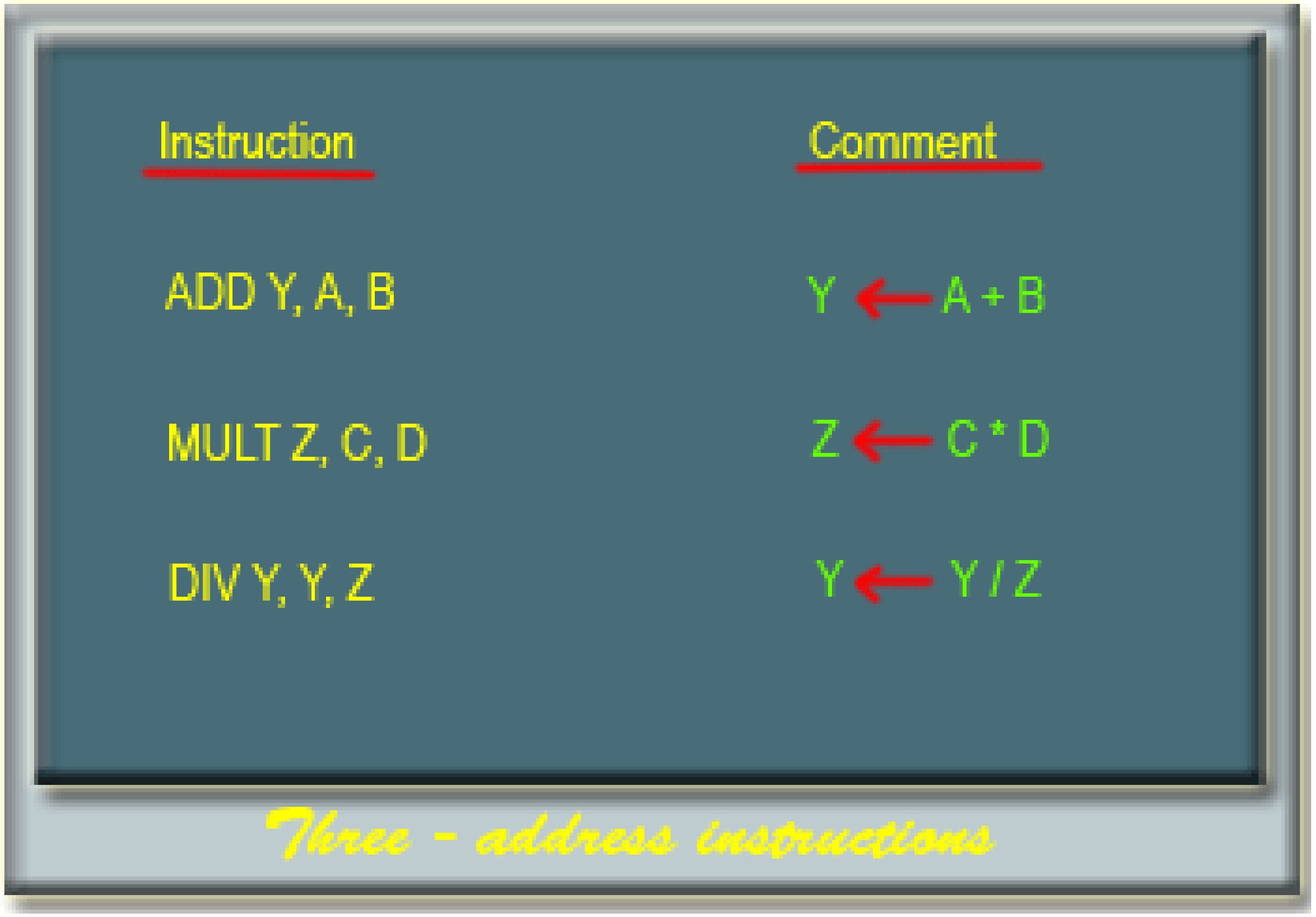
Example: Evaluate (A+B) ∗ (C+D)

- RISC

```
1.  LOAD     R1, A          ; R1 ← M[A]
2.  LOAD     R2, B          ; R2 ← M[B]
3.  LOAD     R3, C          ; R3 ← M[C]
4.  LOAD     R4, D          ; R4 ← M[D]
5.  ADD      R1, R1, R2     ; R1 ← R1 + R2
6.  ADD      R3, R3, R4     ; R3 ← R3 + R4
7.  MUL      R1, R1, R3     ; R1 ← R1 ∗ R3
8.  STORE    X, R1          ; M[X] ← R1
```

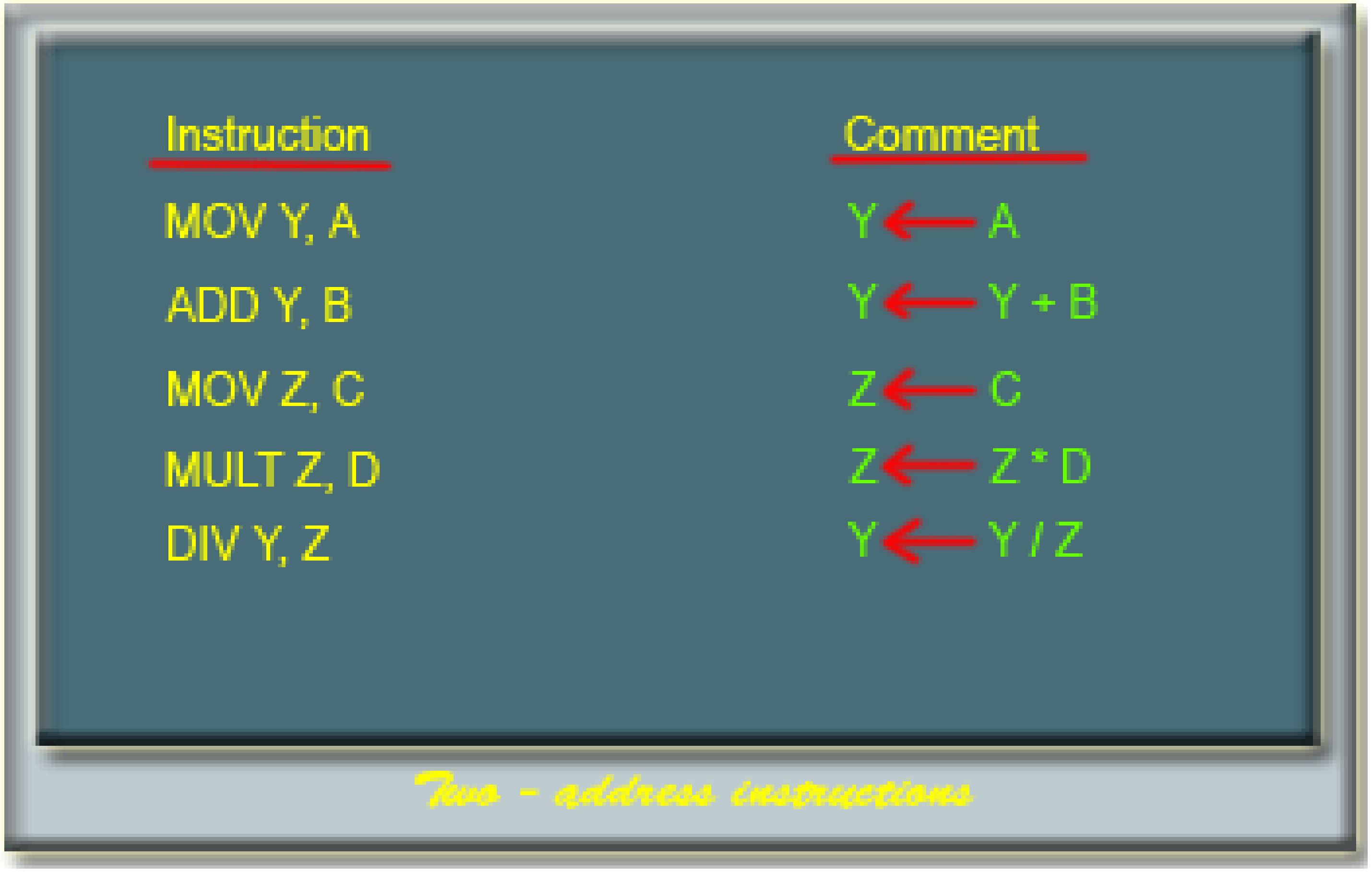
Questions

- Evaluate the arithmetic statement $Y=(A+B)/(C*D)$ in zero, one, two, three addresses machine instructions. A,B,C,D are variables.



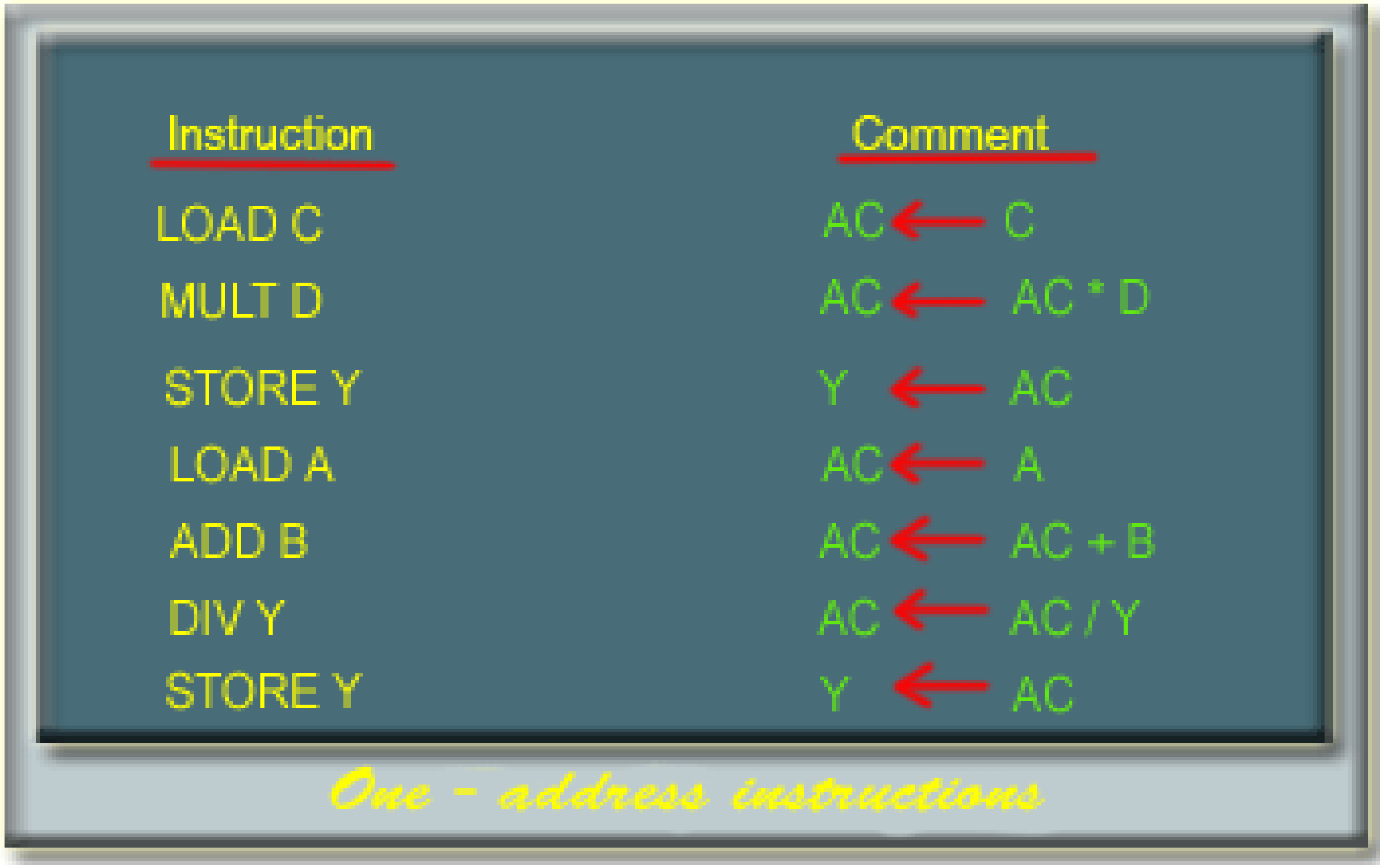
Instruction	Comment
ADD Y, A, B	$Y \leftarrow A + B$
MULT Z, C, D	$Z \leftarrow C * D$
DIV Y, Y, Z	$Y \leftarrow Y / Z$

Three - address instructions



Instruction	Comment
MOV Y, A	$Y \leftarrow A$
ADD Y, B	$Y \leftarrow Y + B$
MOV Z, C	$Z \leftarrow C$
MULT Z, D	$Z \leftarrow Z * D$
DIV Y, Z	$Y \leftarrow Y / Z$

Two - address instructions



Instruction	Comment
LOAD C	$AC \leftarrow C$
MULT D	$AC \leftarrow AC * D$
STORE Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
ADD B	$AC \leftarrow AC + B$
DIV Y	$AC \leftarrow AC / Y$
STORE Y	$Y \leftarrow AC$

One - address instructions

Questions

- A processor has 16 integer registers (R0, R1, .. , R15) and 64 floating point registers (F0, F1,… , F63). It uses a 2-byte instruction format. There are four categories of instructions:Type-1, Type-2, Type-3, and Type-4. Type-1 category consists of four instructions, each with 3 integer register operands (3Rs). Type-2 category consists of four instructions, each with 2 floating point register operands (2Fs). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of N instructions, each with a floating point register operand (1F).
- Specify instruction format for each type.
- The maximum value of N is __________.

End of Module I

Thank you