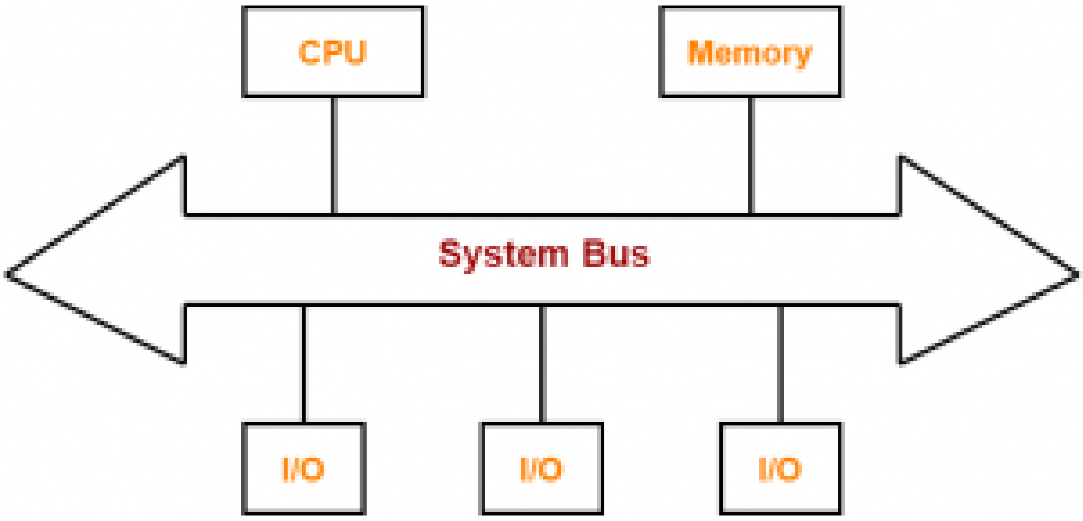
SYSTEM BUS

What Is A System Bus?

- A bus is a set of electrical wires (lines) that connects the various hardware components of a computer system.
- It works as a communication pathway through which information flows from one hardware component to the other hardware component.

A bus that connects major components (CPU, memory and I/O devices) of a computer system is called as a **System Bus**.

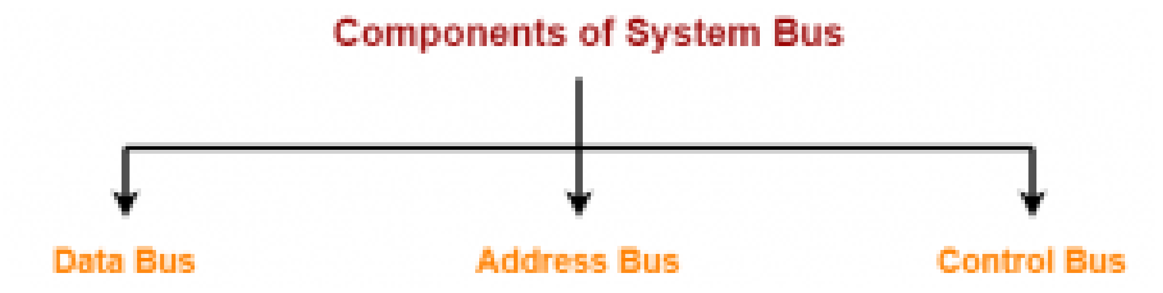


Why Do We Need Bus?

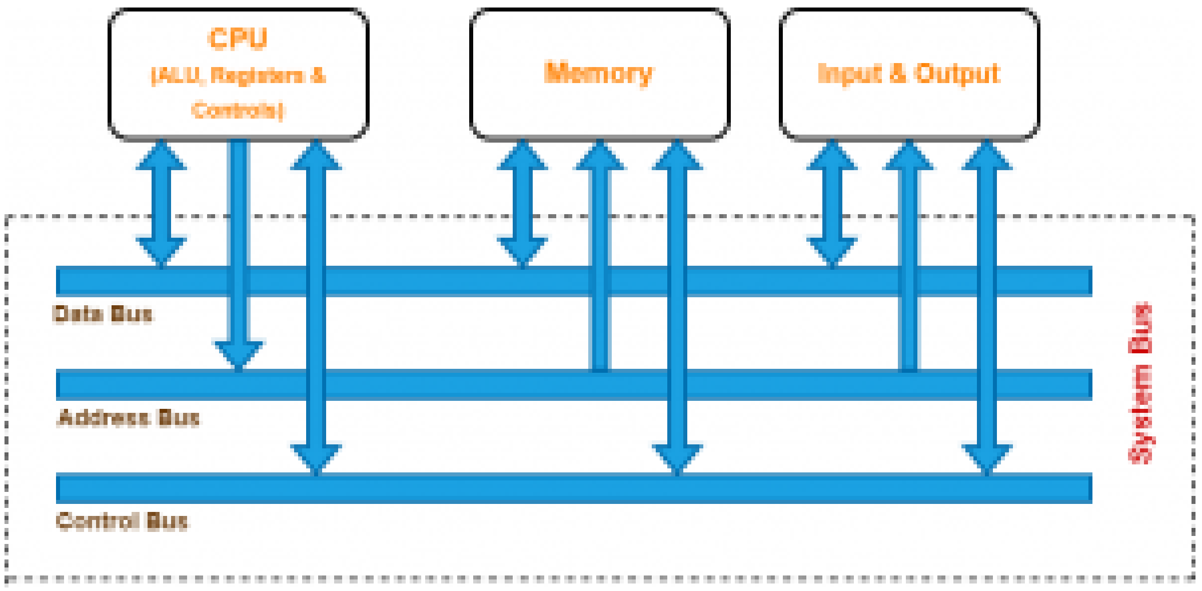
- A computer system is made of different components such as memory, ALU, registers etc.
- Each component should be able to communicate with other for proper execution of instructions and information flow.
- If we try to implement a mesh topology among different components, it would be really expensive.
- So, we use a common component to connect each necessary component i.e. BUS.

Components Of A System Bus-

The system bus consists of three major components-



1. Data Bus
2. Address Bus
3. Control Bus



Let us learn about each component one by one.

1) Data Bus-

- As the name suggests, data bus is used for transmitting the data / instruction from CPU to memory/IO and vice-versa.
- It is bi-directional.

Data Bus Width

- The width of a data bus refers to the number of bits (electrical wires) that the bus can carry at a time.
- Each line carries 1 bit at a time. So, the number of lines in data bus determine how many bits can be transferred parallely.
- The width of data bus is an important parameter because it determines how much data can be transmitted at one time.
- The wider the bus width, faster would be the data flow on the data bus and thus better would be the system performance.

Examples-

- A 32-bit bus has thirty two (32) wires and thus can transmit 32 bits of data at a time.
- A 64-bit bus has sixty four (64) wires and thus can transmit 64 bits of data at a time.

2) Control Bus-

- As the name suggests, control bus is used to transfer the control and timing signals from one component to the other component.
- The CPU uses control bus to communicate with the devices that are connected to the computer system.
- The CPU transmits different types of control signals to the system components.
- It is bi-directional.

What Are Control & Timing Signals?

Control signals are generated in the control unit of CPU.

Timing signals are used to synchronize the memory and I/O operations with a CPU clock.

Typical control signals hold by control bus-

- **Memory read –** Data from memory address location to be placed on data bus.
- **Memory write –** Data from data bus to be placed on memory address location.
- **I/O Read –** Data from I/O address location to be placed on data bus.
- **I/O Write –** Data from data bus to be placed on I/O address location.

Other control signals hold by control bus are interrupt, interrupt acknowledge, bus request, bus grant and several others.

The type of action taking place on the system bus is indicated by these control signals.

Example-

When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory. Similarly, when the processor wants to read from an I/O device, it generates the I/O read signal.

<u>3) Address Bus-</u>

- As the name suggests, address bus is used to carry address from CPU to memory/IO devices.
- It is used to identify the particular location in memory.
- It carries the source or destination address of data i.e. where to store or from where to retrieve the data.
- It is uni-directional.

Example-

When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory and the address of the memory location is sent on the address bus.

If CPU wants to read data stored at the memory location (address) 4, the CPU send the value 4 in binary on the address bus.

<u>Address Bus Width</u>

- The width of address bus determines the amount of physical memory addressable by the processor.
- In other words, it determines the size of the memory that the computer can use.
- The wider is the address bus, the more memory a computer will be able to use.
- The addressing capacity of the system can be increased by adding more address lines.

Examples-

- An address bus that consists of 16 wires can convey 2^{16} (= 64K) different addresses.
- An address bus that consists of 32 wires can convey 2^{32} (= 4G) different addresses.

PRACTICE PROBLEMS BASED ON SYSTEM BUS-

Problem-01:

Which of the following system bus is used to designate the source or destination of the data on the bus itself?

1. Control bus
2. Data bus
3. Address bus
4. System bus

Solution-

The correct option is (C) Address bus.

Address bus carries the source or destination address of data i.e. where to store or from where to retrieve the data.

Problem-02:

The bus which is used to transfer data from main memory to peripheral device is-

1. Data bus
2. Input bus
3. DMA bus
4. Output bus

Solution-

The correct option is (A) Data bus.

Data bus carries data / instruction from CPU to memory/IO and vice-versa.

Problem-03:

How many memory locations a system with a 32-bit address bus can address?

1. 2^8

2. 2^{16}
3. 2^{32}
4. 2^{64}

Solution-

The correct option is (C) 2^{32}.

2^{32} memory locations can be addressed by a 32-bit address bus.

Problem-04:

How many bits can be transmitted at a time using a bus with 32 data lines?

1. 8 bits
2. 16 bits
3. 32 bits
4. 1024 bits

Solution-

Each line carries one bit. So, a bus with 32 data lines can transmit 32 bits at a time.

Problem-05:

A microprocessor has a data bus with 64 lines and an address bus with 32 lines. The maximum number of bits that can be stored in memory is-

1. 32×2^{12}
2. 32×2^{64}
3. 64×2^{32}
4. 64×2^{64}

Solution-

The correct option is (C) 64×2^{32}.

The amount of blocks that could be located is 2^{32}. Now, since data bus has 64 lines, so each block is 64 bits. Thus, maximum number of bits stored in memory is $2^{32} \times 64$ bits.

Problem-06:

The address bus with a ROM of size 1024 x 8 bits is-

1. 8 bits
2. 10 bits
3. 12 bits
4. 16 bits

Solution-

The correct option is (B) 10 bits.

The size of the ROM is 1024 x 8 = 2^{10} x 8. Here, 10 indicates the address bus and 8 indicates the data bus width.

Problem-07:

The data bus width of a ROM of size 2048 x 8 bits is-

1. 8
2. 10
3. 12
4. 16

Solution-

The correct option is (A) 8.

The size of the ROM is 2048 x 8 = 2^{11} x 8. Here, 11 indicates the address bus and 8 indicates the data bus width.

InstructionTypes:

Computers must have instructions capable of performing the following four types of operations:

 Data transfers between the memory and the processor registers

 Arithmetic and logic operations on data

 Program sequencing and control

 I/O transfers

Instruction Types

Instructions are divided into the following five types:

1 Data-transfer instruction, which copy information from one location to another location

either in the processor's internal register set or in the external main memory.

Operation: MOV, LOAD, STORE, XCH, PUSH, POP

Eg: MOV A, R1 LOAD A STORE T XCH A PUSH A POPA

2 Arithmetic instructions, which perform operations on numerical data.

Operation: ADD,ADD WITH CARRY,SUBTRACT,MULTIPLY, DIV

Eg: ADDA, B ADDC A,B SUBA,B MULA,B

3 Logical instructions, which include Boolean and other non-numerical Operations.

Operation: AND,OR,NOT,XOR, SHIFTLEFT, SHIFTRIGHT

Eg: ANDA, B OR A,B NOT A,B SHLA SHR A

4 Program control instructions, such as branch instruction, which change the sequence in which

programs are executed.

Operation: JUMP, RETURN,EXECUTE, SKIP, CONDITIONAL

COMPARE,TEST,WAIT

Eg: JMP CALL RET SKP TST

5 Input-output (IO) instructions, which cause information to be transferred between the

processor or its main memory and external IO devices.

Operation: INPUT, OUTPUT, START IO, TEST IO, HALT IO

Eg: IN OUT

ADDRESSING MODES:

Addressing modes	Example Instruction	Meaning	When used
Register	Add R4,R3	R4 <- R4 + R3	When a value is in a register
Immediate	Add R4, #3	R4 <- R4 + 3	For constants
Displacement	Add R4, 100(R1)	R4 <- R4 + M[100+R1]	Accessing local variables
Register deffered	Add R4,(R1)	R4 <- R4 + M[R1]	Accessing using a pointer or a computed address
Indexed	Add R3, (R1 + R2)	R3 <- R3 + M[R1+R2]	Useful in array addressing: R1 - base of array R2 - index amount
Direct	Add R1, (1001)	R1 <- R1 + M[1001]	Useful in accessing static data
Memorydeferred	Add R1, @(R3)	R1 <- R1 + M[M[R3]]	If R3 is the address of a pointer p, then mode yields *p
Autoincrement	Add R1, (R2)+	R1 <- R1 +M[R2] R2 <- R2 + d	Useful for stepping through arrays in a loop. R2 - start of array d - size of an element
Autodecrement	Add R1,-(R2)	R2 <-R2-d R1 <- R1 + M[R2]	Same as autoincrement. Both can also be used to implement a stack as push and pop
Scaled	Add R1,100(R2)[R3]	R1<-R1+M[100+R2+R3*d]	Used to index arrays. May be applied to any base addressing mode in some machines.

Problem-01:

The most appropriate matching for the following pairs is-

Column-1:

X: Indirect addressing

Y: Immediate addressing

Z: Auto decrement addressing

Column-2:

1. Loops
2. Pointers
3. Constants

A X-3, Y-2, Z-1
B X-1, Y-3, Z-2
C X-2, Y-3, Z-1
D X-3, Y-1, Z-2

Solution-

Option (C) is correct.

Problem-02:

In the absolute addressing mode,

1. The operand is inside the instruction
2. The address of the operand is inside the instruction
3. The register containing the address of the operand is specified inside the instruction
4. The location of the operand is implicit

Solution-

Option (B) is correct.

Problem-03:

Which of the following addressing modes are suitable for program relocation at run time?

1. Absolute addressing
2. Base addressing
3. Relative addressing
4. Indirect addressing

1. 1 and 4
2. 1 and 2
3. 2 and 3
4. 1, 2 and 4

Solution-

Option (C) is correct.

Problem-04:

What is the most appropriate match for the items in the first column with the items in the second column-

Column-1:

X: Indirect addressing

Y: Indexed addressing

Z: Base register addressing

Column-2:

1. Array implementation

2. Writing relocatable code

3. Passing array as parameter

1. X-3, Y-1, Z-2
2. X-2, Y-3, Z-1
3. X-3, Y-2, Z-1
4. X-1, Y-3, Z-2

Solution-

Option (A) is correct.

Problem-05:

Which of the following addressing modes permits relocation without any change whatsoever in the code?

1. Indirect addressing
2. Indexed addressing
3. Base register addressing
4. PC relative addressing

Solution-

Option (C) is correct.

Problem-06:

Consider a three word machine instruction-

ADD A[R_0], @B

The first operand (destination) "A[R_0]" uses indexed addressing mode with R_0 as the index register. The second operand operand (source) "@B" uses indirect addressing mode. A and B are memory addresses residing at the second and the third words, respectively. The first word of the instruction specifies the opcode, the index register designation and the source and destination addressing modes. During execution of ADD instruction, the two operands are added and stored in the destination (first operand).

The number of memory cycles needed during the execution cycle of the instruction is-

1. 3
2. 4
3. 5
4. 6

Solution-

For the first operand,

- It uses indexed addressing mode.
- Thus, one memory cycle will be needed to fetch the operand.

For the second operand,

- It uses indirect addressing mode.
- Thus, two memory cycles will be needed to fetch the operand.

After fetching the two operands,

- The operands will be added and result is stored back in the memory.
- Thus, one memory cycle will be needed to store the result.

Total number of memory cycles needed

= 1 + 2 + 1

= 4

Thus, Option (B) is correct.

Problem-07:

Consider a hypothetical processor with an instruction of type LW R_1, 20(R_2), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R_1. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R_2. Which of the following best reflects the addressing mode implemented by this instruction for operand in memory?

1. Immediate Addressing
2. Register Addressing
3. Register Indirect Scaled Addressing
4. Base Indexed Addressing

Solution-

Clearly, the instruction uses base indexed addressing mode.

Thus, Option (D) is correct.

Problem-08:

The memory locations 1000, 1001 and 1020 have data values 18, 1 and 16 respectively before the following program is executed.

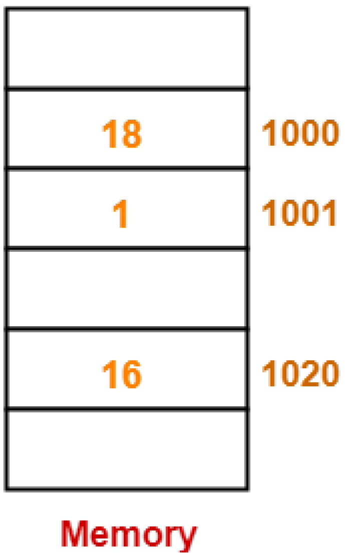
MOVI	R_s, 1	Move immediate
LOAD	R_d, 1000(R_s)	Load from memory
ADDI	R_d, 1000	Add immediate
STOREI	0(R_d), 20	Store immediate

Which of the statements below is TRUE after the program is executed?

1. Memory location 1000 has value 20
2. Memory location 1020 has value 20
3. Memory location 1021 has value 20
4. Memory location 1001 has value 20

Solution-

Before the execution of program, the memory is-



Now, let us execute the program instructions one by one-

Instruction-01: MOVI R_s, 1

- This instruction uses immediate addressing mode.
- The instruction is interpreted as Rs ← 1.
- Thus, value = 1 is moved to the register R_s.

Instruction-02: LOAD R_d, 1000(R_s)

- This instruction uses displacement addressing mode.
- The instruction is interpreted as R_d ← [1000 + [R_s]].
- Value of the operand = [1000 + [R_s]] = [1000 + 1] = [1001] = 1.
- Thus, value = 1 is moved to the register R_d.

Instruction-03: ADDI R_d, 1000

- This instruction uses immediate addressing mode.
- The instruction is interpreted as $R_d \leftarrow [R_d] + 1000$.
- Value of the operand = $[R_d] + 1000 = 1 + 1000 = 1001$.
- Thus, value = 1001 is moved to the register R_d.

Instruction-04: STOREI 0(R_d), 20

- This instruction uses displacement addressing mode.
- The instruction is interpreted as $0 + [R_d] \leftarrow 20$.
- Value of the destination address = $0 + [R_d] = 0 + 1001 = 1001$.
- Thus, value = 20 is moved to the memory location 1001.

Thus,

- After the program execution is completed, memory location 1001 has value 20.
- Option (D) is correct.

To watch video solution, click **here**.

Problem-09:

Consider the following memory values and a one-address machine with an accumulator, what values do the following instructions load into accumulator?

- Word 20 contains 40
- Word 30 contains 50
- Word 40 contains 60
- Word 50 contains 70

Instructions are-

1. Load immediate 20
2. Load direct 20
3. Load indirect 20
4. Load immediate 30
5. Load direct 30
6. Load indirect 30

Solution-

Instruction-01: Load immediate 20

- This instruction uses immediate addressing mode.
- The instruction is interpreted as Accumulator $\leftarrow 20$.

- Thus, value 20 is loaded into the accumulator.

Instruction-02: Load direct 20

- This instruction uses direct addressing mode.
- The instruction is interpreted as Accumulator ← [20].
- It is given that word 20 contains 40.
- Thus, value 40 is loaded into the accumulator

Instruction-03: Load indirect 20

- This instruction uses indirect addressing mode.
- The instruction is interpreted as Accumulator ← [[20]].
- It is given that word 20 contains 40 and word 40 contains 60.
- Thus, value 60 is loaded into the accumulator.

Instruction-04: Load immediate 30

- This instruction uses immediate addressing mode.
- The instruction is interpreted as Accumulator ← 30.
- Thus, value 30 is loaded into the accumulator.

Instruction-05: Load direct 30

- This instruction uses direct addressing mode.
- The instruction is interpreted as Accumulator ← [30].
- It is given that word 30 contains 50.
- Thus, value 50 is loaded into the accumulator

Instruction-06: Load indirect 30

- This instruction uses indirect addressing mode.
- The instruction is interpreted as Accumulator ← [[30]].
- It is given that word 30 contains 50 and word 50 contains 70.
- Thus, value 70 is loaded into the accumulator.

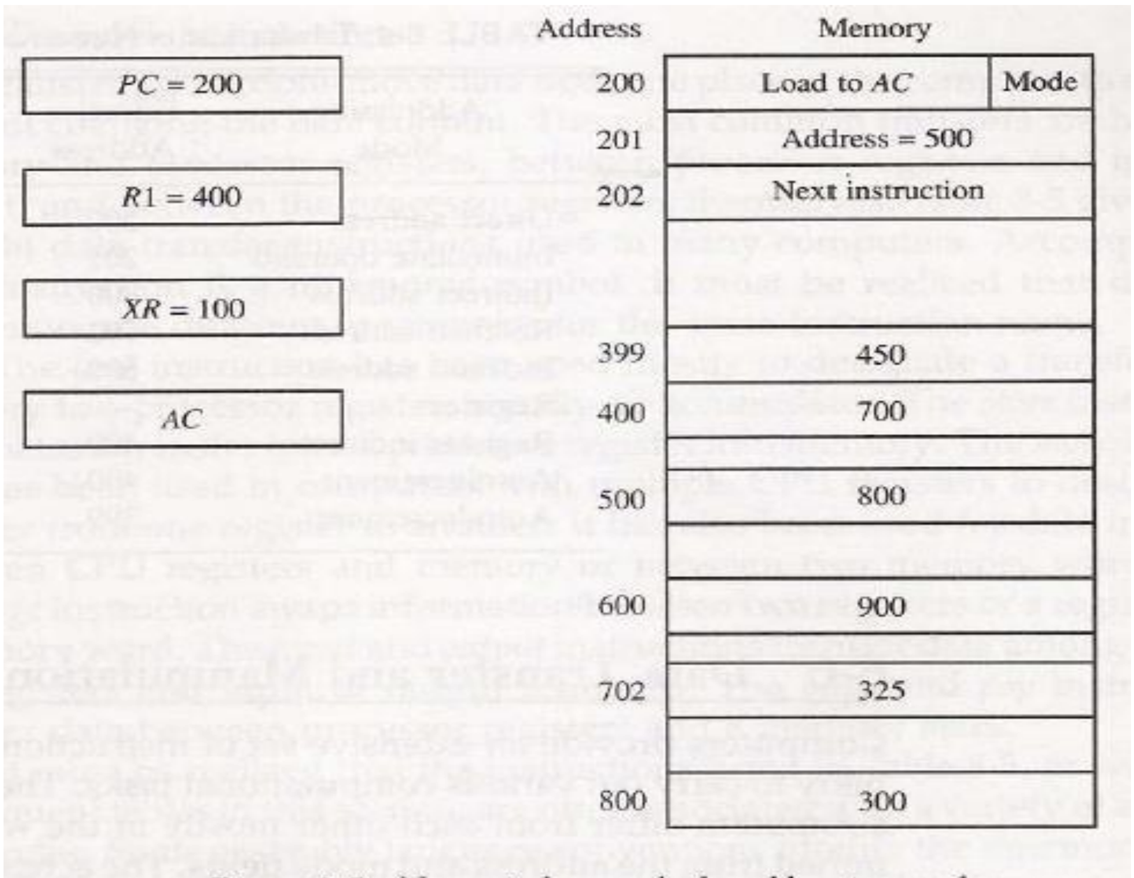
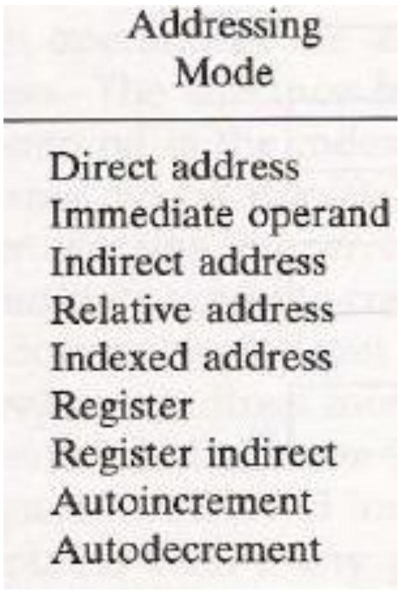


Figure 8-7 Numerical example for addressing modes.



Addressing Mode
Direct address
Immediate operand
Indirect address
Relative address
Indexed address
Register
Register indirect
Autoincrement
Autodecrement

Write effective address of each addressing mode and content of accumulator.

Subroutine Call and Return

- A subroutine is a self-contained sequence of instructions that performs a given computational task.

- The instruction that transfers program control to a subroutine is known by different names. The most common names used are call subroutine, jump to subroutine, branch to subroutine, or branch and save address.

- The instruction is executed by performing two operations:

(1) The address of the next instruction available in the program counter (the return address) is

Stored in a temporary location so the subroutine knows where to return

(2) Control is transferred to the beginning of the subroutine.

Different computers use a different temporary location for storing the return address.

- Some store the return address in the first memory location of the subroutine, some store it in a fixed location in memory, some store it in a processor register, and some store it in a memory stack. The most efficient way is to store the return address in a memory stack. The advantage of using a stack for the return address is that when a succession of subroutines is called, the sequential return addresses can be pushed into the stack. The return from subroutine

instruction causes the stack to pop and the contents of the top of the stack are transferred to the program counter.

- A subroutine call is implemented with the following micro operations:

SP $\leftarrow$SP - 1 Decrement stack pointer

M [SP] $\leftarrow$PC Push content of PC onto the stack

PC $\leftarrow$ effective address Transfer control to the subroutine

- If another subroutine is called by the current subroutine, the new return address is pushed into

The stack and so on. The instruction that returns from the last subroutine is implemented by the

Micro operations:

PC$\leftarrow$M [SP] Pop stack and transfer to PC

SP $\leftarrow$SP + 1 Increment stack pointer