



COMPUTER SYSTEM ARCHITECTURE

(Course code: AECD04)

B.Tech III Semester

Regulation: IARE BT-23

Prepared by:

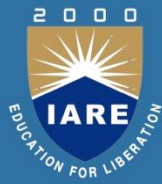
Mr. N Rajasekhar Reddy

Assistant Professor & Dean – Computer Center



Department of Computer Science and Engineering
INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
DUNDIGAL, HYDERABAD - 500 043

Course Outcomes



The course should enable the students to:

CO 1	Understand the organization and levels of design in computer architecture and To understand the concepts of programming methodologies.
CO 2	Describe Register transfer languages, arithmetic micro operations, logic micro operations, shift micro operations address sequencing, micro program example, and design of control unit.
CO 3	Understand the Instruction cycle, data representation, memory reference instructions, input-output, and interrupt, addressing modes, data transfer and manipulation, program control. Computer arithmetic: Addition and subtraction, floating point arithmetic operations, decimal arithmetic unit.
CO 4	Knowledge about Memory hierarchy, main memory, auxiliary memory, associative memory, cache memory, virtual memory Input or output Interface, asynchronous data transfer, modes of transfer, priority interrupt, direct memory access.
CO 5	Explore the Parallel processing, pipelining-arithmetic pipeline, instruction pipeline Characteristics of multiprocessors, inter connection structures, inter processor arbitration, inter processor Communication and synchronization

MODULE-II

ORGANIZATION OF A COMPUTER

Course Outcomes

CO 1	Understand the Instruction cycles and data representation in CPU design .
CO 2	Classify the input-output and interrupt, addressing modes.
CO 3	Describe the data transfer and manipulation and program control in CPU design .
CO 4	Knowledge about Addition and subtraction and floating point arithmetic operations in Computer arithmetic.
CO 5	Understand the decimal arithmetic unit Computer arithmetic.

Contents

- **Instruction Codes**
- **Computer Registers**
- **Computer Instructions**
- **Timing and Control**
- **Instruction cycle**

- **Program Input-output and interrupt**
- **Instruction Formats**
- **Addressing modes**
- **Data transfer and manipulation**
- **Program control**
- **RISC**

Instruction Codes

- An instruction code is a group of bits that instruct the computer to perform a specific operation.
- The operation code of an instruction is a group of bits that define operations such as addition, subtraction, shift, complement, etc.
- An instruction must also include one or more operands, which indicate the registers and/or memory addresses from which data is taken or to which data is deposited.

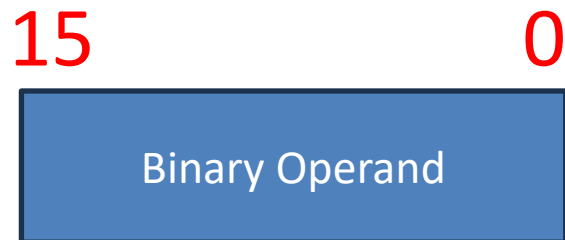
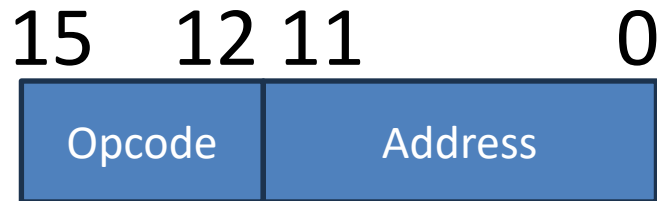
Micro Operations

- The instructions are stored in computer memory in the same manner that data is stored.
- The control unit interprets these instructions and uses the operations code to determine the sequences of microoperations that must be performed to execute the instruction

Stored Program Organization

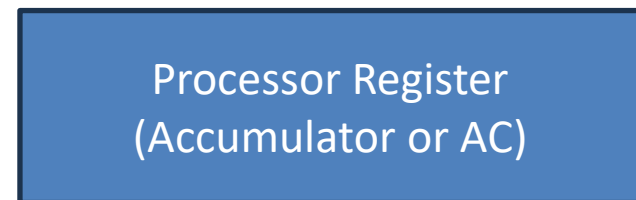
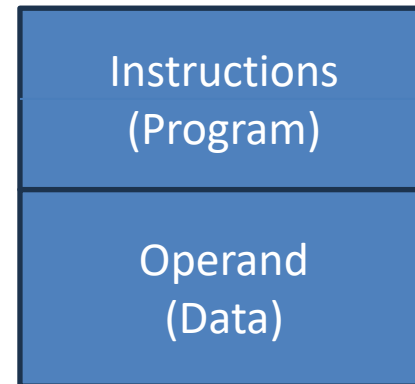
- The operands are specified by indicating the registers and/or memory locations in which they are stored.
 - **k bits can be used to specify which of 2^k registers (or memory locations) are to be used.**
- The simplest design is to have **one processor register** (called the **accumulator**) and two fields in the instruction, one for the opcode and one for the operand.
- Any operation that **does not need** a memory operand frees the other bits to be used for other purposes, such as specifying different operations.

Stored Program Organization



Memory

4096 x 16



Computer Registers

- The **Registers set** stores intermediate data used during the execution of the instructions.
- **Processor register** is a small amount of very fast computer memory used to speed the execution of computer programs by providing quick access to commonly used values—typically, the values being in the midst of a calculation at a given point in time. **(AC)**
- Data registers are used to store integer numbers **(DR - MDR)**
- Address registers hold memory addresses and are used to access memory. **(AR - MAR)**
- General Purpose registers **(GPRs)** can store both data and addresses, i.e., they are combined **Data/Address registers.**

- **Floating Point Registers (FPRs)** are used to store floating point Computer arithmetic .
- **Constant Registers** hold read-only values (**e.g., zero, one, pi, ...**).
- **Vector registers** hold data for vector processing .
- **Special Purpose Registers** store internal CPU data, like the **program counter** which indicates where the computer is in its instruction sequence.
- **Control Registers** which **control** the general behavior of the CPU.
- **Program Counter** Holds address for instruction (i.e. address of the next instruction after execution of the current instruction is completed (PC)
- **Instruction Register** holds the instruction code. (IR)
- **Temporary Register** holds temporary data (TR)
- **Input Register** holds input character (INPR)
- **Output Registers** holds output character (OUTR)

Computer Instructions

- Computer instructions are a **set of machine language instructions** that a particular processor understands and executes.
- A computer performs tasks on the basis of the **instruction provided**.
- An instruction comprises of **groups called fields**. These fields include:
 1. The **Operation code (Opcode)** field which specifies the operation to be performed.
 2. The **Address field** which contains the location of the operand, i.e., register or memory location.
 3. The **Mode field** which specifies how the operand will be located.



A basic computer has three instruction code formats which are:

1. Memory - reference instruction
2. Register - reference instruction
3. Input-Output instruction

Memory - Reference Instruction



(Opcode = 000 through 110)

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

Computer Instructions

Register - Reference Instruction



(Opcode = 111, I = 0)

- The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction.
- The Operation code (Opcode) of an instruction refers to a group of bits that define arithmetic and logic operations such as add, subtract, multiply, shift, and compliment.
- A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.

Computer Instructions

Input-Output Instruction



(Opcode = 111, I = 1)

Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed.

Computer Instructions

Note

- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type, and the bit in position 15 is taken as the addressing mode I.
- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

Computer Instructions

Instruction Set Completeness

- A **set of instructions** is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
- **Arithmetic, logical and shift instructions**
- A set of instructions for **moving information to and from memory and processor registers**.
- Instructions which **controls the program** together with instructions that check status conditions.
- **Input and Output** instructions.

Timing and Control

The **Timing** for all registers in the basic computer is controlled by a master clock generator.

- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.

The **Control signals** are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers and micro-operations for the accumulator.

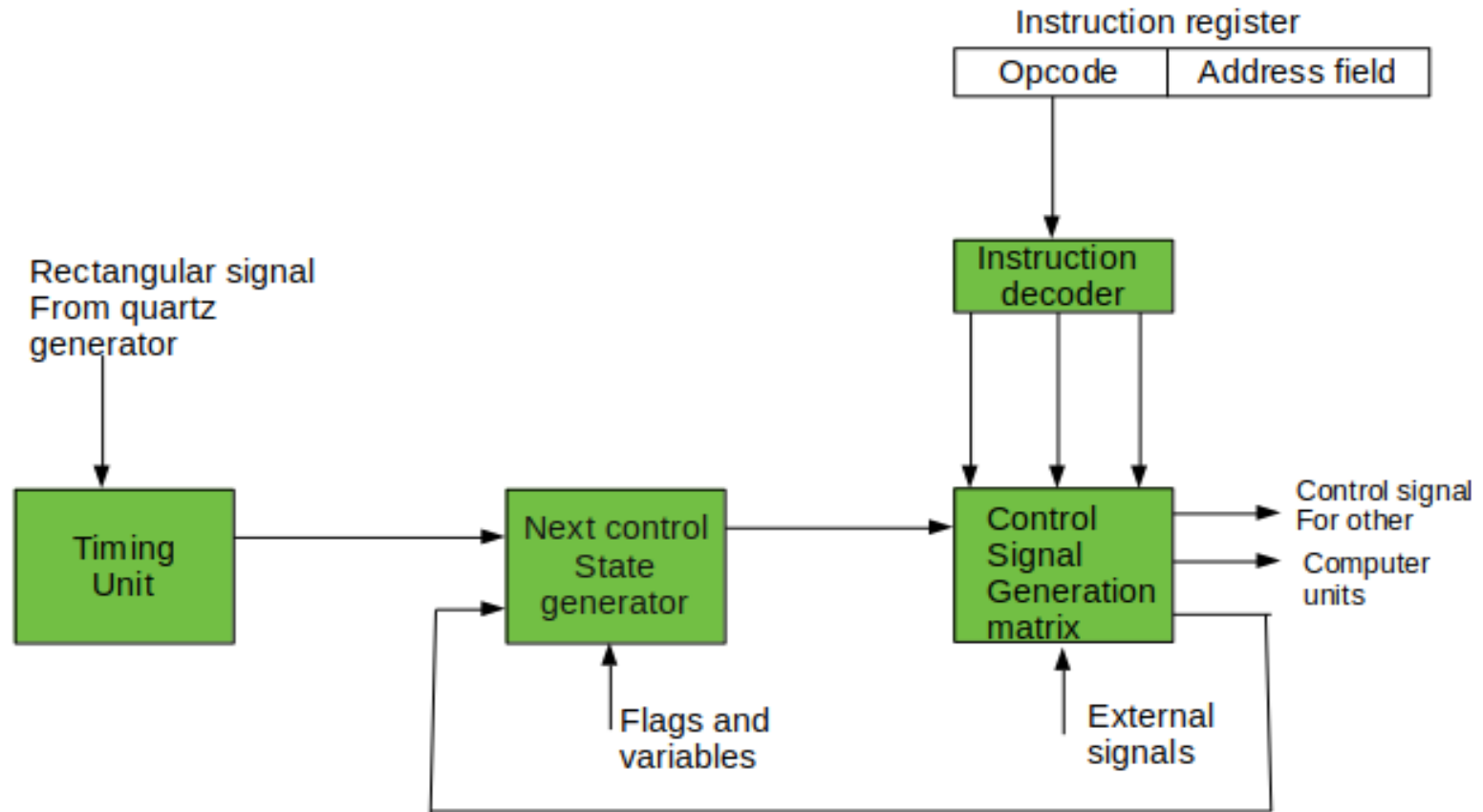
There are two major types of control organization:

- **Hardwired Control**
- **Micro-programmed Control**

Timing and Control

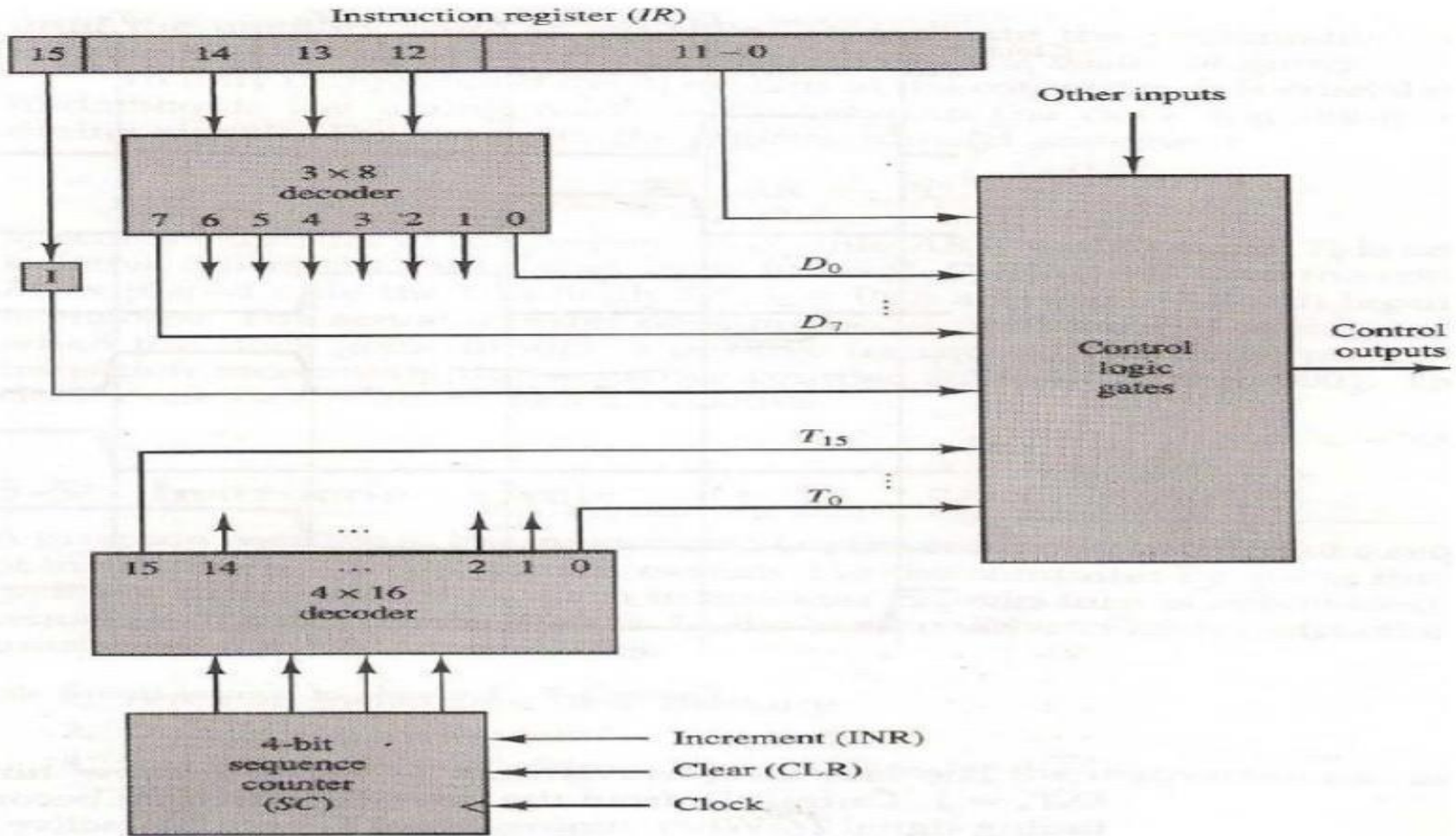
Hardwired Control	Micro-programmed Control
The control logic is implemented with gates, flip-flops, decoders and other digital circuits.	The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.
The advantage is that it can be optimized to produce a fast mode of operation.	Compared with the hardwired control operation is slow.
Requires changes in the wiring among the various components if the design has to be modified or changed.	Required changes or modifications can be done by updating the microprogram in control memory.

Hardwired Control



Block diagram of a hardwired control unit of a computer

Hardwired Control



The block diagram of the Hardwired Control Unit

The control unit of basic computers consists of two decoders, a sequence counter and a number of control logic gates.

- An instruction read from memory is placed in the instruction register (IR). It is divided into three parts:
 - The I bit
 - The operation code and
 - Bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3×8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates.

Hardwired Control

An instruction read from memory is placed in the **instruction register (IR)**.

- The instruction register is divided into three parts: the I bit, operation code, and address part.
- First 12-bits (0-11) to specify an address, next 3-bits specify the operation code (opcode) field of the instruction and last left most bit specify the addressing mode I.

I = 0 for direct address

I = 1 for indirect address

- First 12-bits (0-11) are applied to the control logic gates.
- The operation code bits (12 – 14) are decoded with a 3 x 8 decoder.
- The eight outputs (D0 through D7) from a decoder goes to the control logic gates to perform specific operation.

- Last bit 15 is transferred to a I flip-flop designated by symbol I
- The 4-bit sequence counter SC can count in binary from 0 through 15.
- The counter output is decoded into 16 timing pulses T0 through T15.
- The sequence counter can be incremented by INR input or clear by CLR input synchronously.

Hardwired Control

ADVANTAGES of Hardwired Control Unit:

- Hardwired Control Unit is fast because control signals are generated by combinational circuits.
- The delay in generation of control signals depends upon the number of gates.

DISADVANTAGES

- More is the control signals required by CPU; more complex will be the design of control unit.
- Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit.
- It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

Micro-programmed Control

- A micro-programmed control unit is implemented using programming approach. A sequence of microoperations are carried out by executing a program consisting of micro-instructions.
- Micro-program, consisting of micro-instructions is stored in the control memory of the control unit.
- Execution of a micro-instruction is responsible for generation of a set of control signals.
- A **micro-instruction** consists of:
 1. One or more micro-operations to be executed.
 2. Address of next microinstruction to be executed.
- **Micro-Operations:** The operations performed on the data stored inside the registers are called micro-operations.

Micro-programmed Control

Micro-Programs: Microprogramming is the concept for generating control signals using programs. These programs are called micro-programs.

Micro-Instructions: The instructions that make micro-program are called micro-instructions.

Micro-Code: Micro-program is a group of microinstructions. The micro-program can also be termed as micro-code.

Control Memory: Micro-programs are stored in the read only memory (ROM). That memory is called control memory.

Micro-programmed Control

ARCHITECTURE OF MICRO-PROGRAMMED CONTROL UNIT:

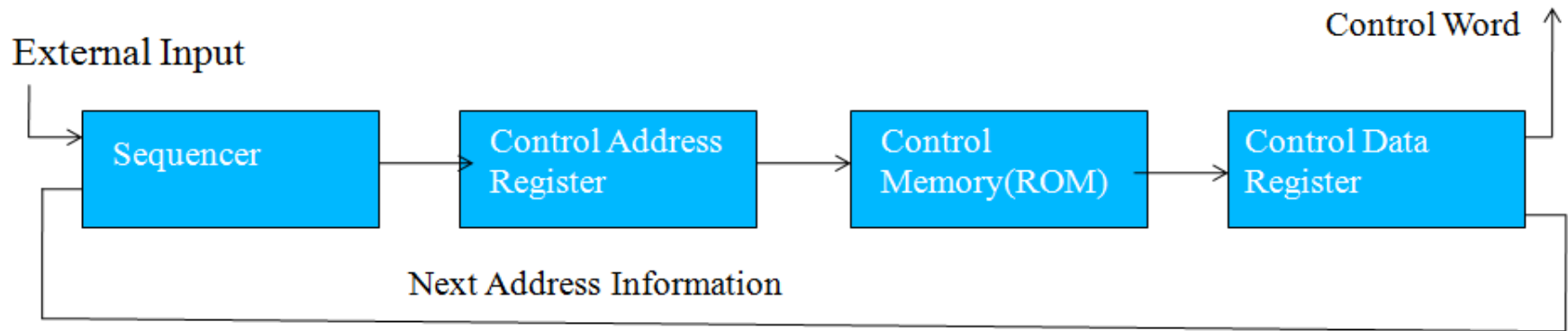


Figure : Microprogrammed Control Unit

Sequencer = Next Address Generator

Micro-programmed Control

- The address of micro-instruction that is to be executed is stored in the **control address register (CAR)**.
- Micro-instruction corresponding to the address stored in CAR is **fetches from control memory** and is stored in the control data register (CDR).
- This micro-instruction **contains control word to execute one or more micro-operations**.
- After the execution **of all micro-operations** of micro-instruction, the address of next micro-instruction is located.

Micro-programmed Control

ADVANTAGES

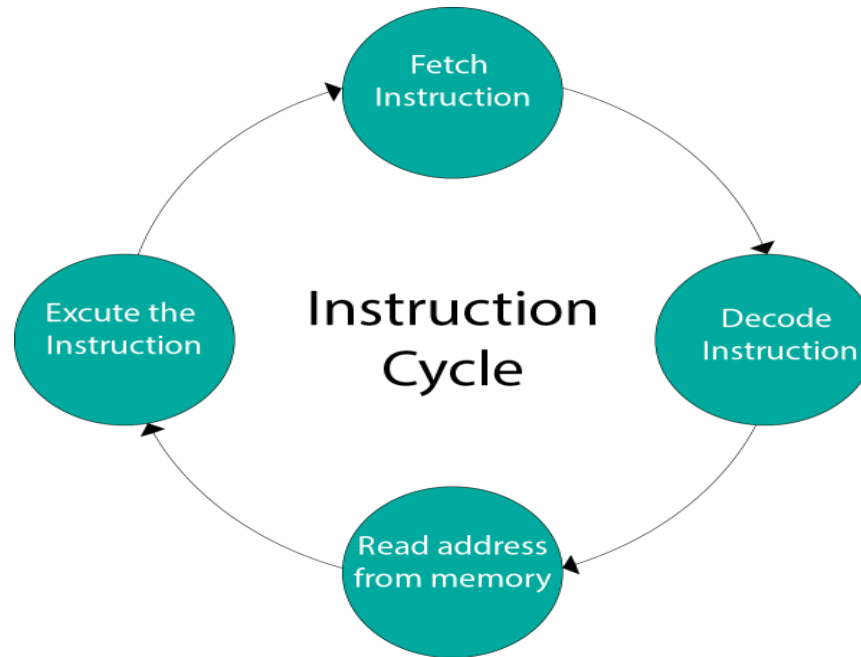
- The design of micro-program control unit is less complex because micro-programs are implemented using software routines.
- The micro-programmed control unit is more flexible because design modifications, correction and enhancement is easily possible.

DISADVANTAGES

- The micro-program control unit is slower than hardwired control unit. That means to execute an instruction in micro-program control unit requires more time.
- The micro-program control unit is expensive than hardwired control unit in case of limited hardware resources.
- The design duration of micro-program control unit is more than hardwired control unit for smaller CPU.

Instruction Cycle

- Each program is a sequence of instructions.
- The basic computer system each instruction is subdivided into Four phases:
 - 1. Fetch an Instruction from memory.**
 - 2. Decode the Instruction.**
 - 3. Read the effective address from the memory if the instruction has an indirect address.**
 - 4. Execute the Instruction.**
- The above process continues indefinitely unless a HALT instruction is encountered.



A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle of each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.

- 1. Fetch instruction:** Read instruction code from address in PC and place in IR.
(IR \leftarrow Memory[PC])
- 2. Decode instruction:** Hardware determines what the opcode/function is and determines which registers or memory addresses contain the operands.
- 3. Fetch operands from the memory if necessary:** If any operands are memory addresses, initiate memory read cycles to read them into CPU registers. If an operand is in memory, not a register, then the memory address of the operand is known as the effective address (EA). The fetching of an operand can therefore be denoted as Register \leftarrow Memory[EA].
- 4. Execute:** Perform the function of the instruction. If arithmetic or logic instruction, utilize the ALU circuits to carry out the operation on data in registers.

Instruction Cycle

Fetch and Decode

- Initially the program counter PC is loaded with the address of the first instruction.

T0: $AR \leftarrow PC$ ($S_0S_1S_2=010, T0=1$)

T1: $IR \leftarrow M[AR], PC \leftarrow PC + 1$ ($S0S1S2=111, T1=1$)

T2: $D0, \dots, D7 \leftarrow \text{Decode } IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$

Instruction Cycle

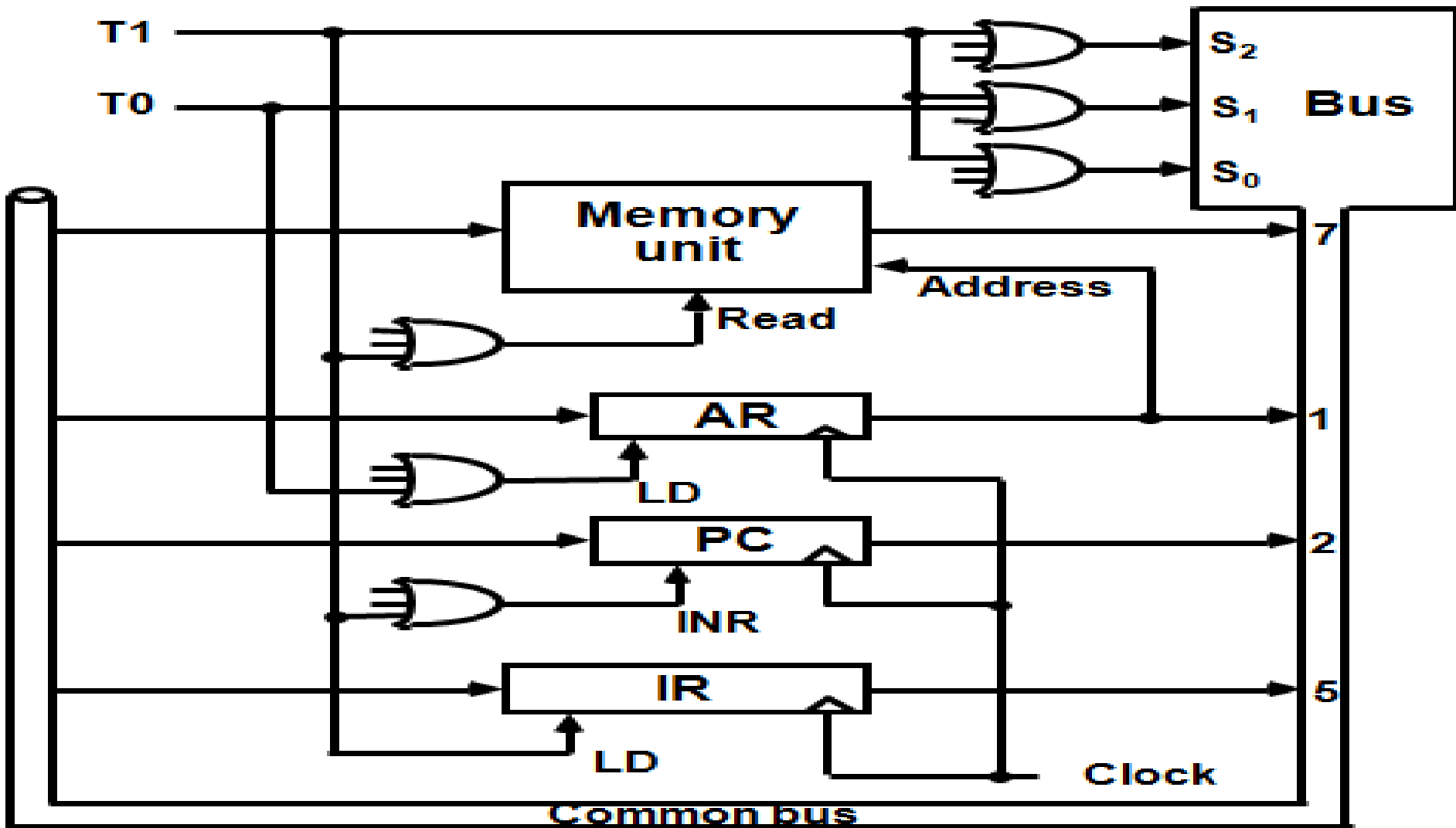


Fig: Register Transfer for the fetch phase

Instruction Cycle

- In the above diagram shows the transfer of first two statements(T0 and T1).
- When timing signal $T0=1$ then
 1. Place the contents of PC onto the bus by making the bus selection inputs $S2S1S0$ equal to 010.
 2. Transfer the contents of the bus to AR by enabling the LD input of AR.
- When timing signal $T1=1$ then
 1. Enable the read input of memory.
 2. Place the contents of Memory onto the bus by making $S2S1S0=111$.
 3. Transfer the contents of the bus to IR by enabling the LD input of IR.
 4. Increment PC by enabling the INR input of PC.

Instruction Cycle

Determine the Type of Instruction

- After executing the timing signal T1 the control unit determines the type of instruction that is read from memory.
- If D7=1 and the instruction must be a register-reference or input-out type.
- If D7=0 the operation code must be one of the other seven values 000 through 110 specifying a memory –reference Instruction.
- The symbolic representation is :

D'7IT3 :	$AR \leftarrow M[AR]$
D'7I'T3 :	Nothing
D7I'T3 :	Execute a register-reference instr.
D7IT3 :	Execute an input-output instr.

Instruction Cycle

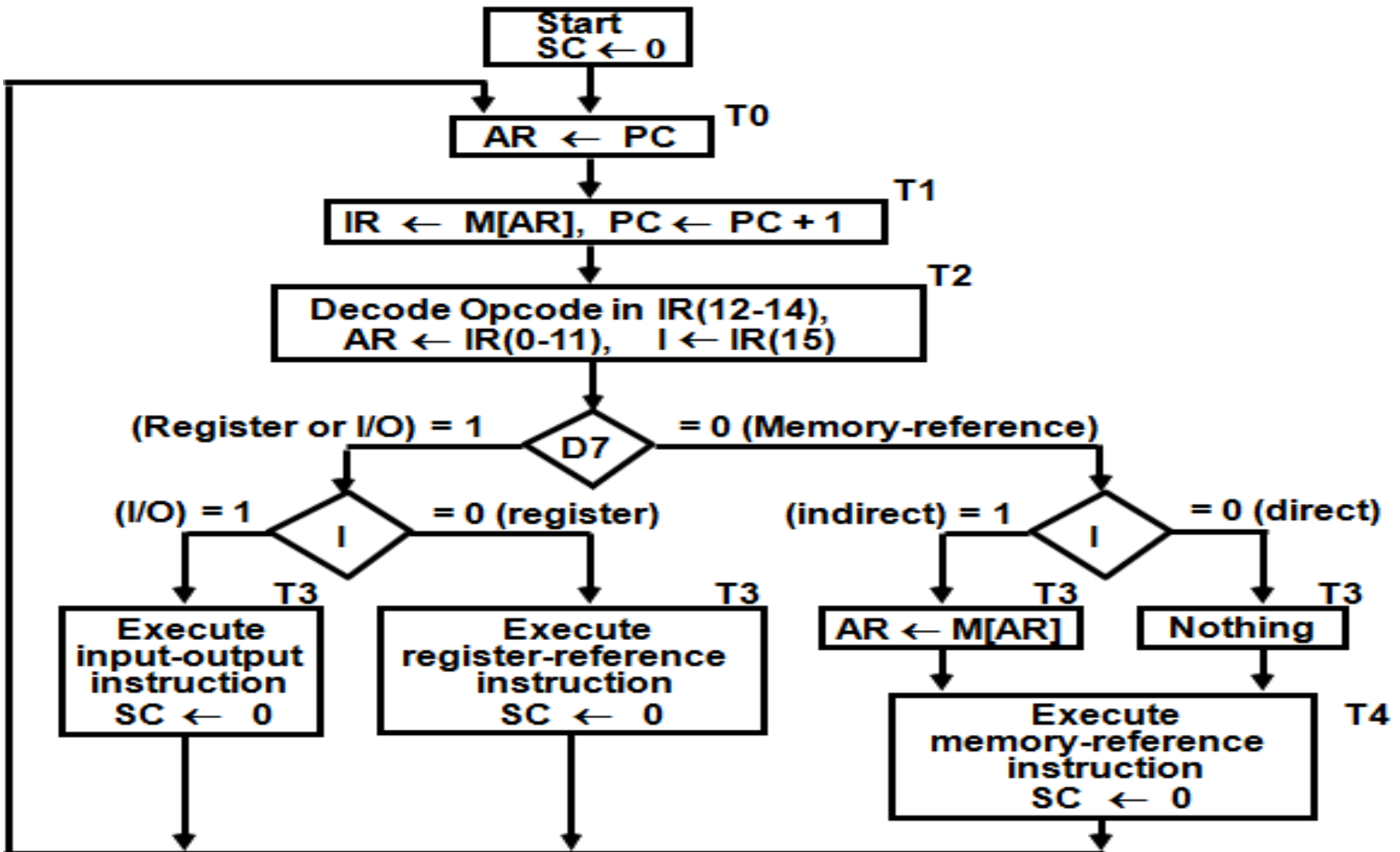


Fig : Flowchart for Instruction Cycle

Register Reference Instructions

- If D7=1 and I=0 then the instruction is recognized as register reference instruction.
- Register reference instructions use bits 0 to 11 of the instruction code to specify one of the 12 instructions. These bits are available in IR(0-11).

	r:	$SC \leftarrow 0$
CLA	rB_{11} :	$AC \leftarrow 0$
CLE	rB_{10} :	$E \leftarrow 0$
CMA	rB_9 :	$AC \leftarrow AC'$
CME	rB_8 :	$E \leftarrow E'$
CIR	rB_7 :	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6 :	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5 :	$AC \leftarrow AC + 1$
SPA	rB_4 :	if $(AC(15) = 0)$ then $(PC \leftarrow PC+1)$
SNA	rB_3 :	if $(AC(15) = 1)$ then $(PC \leftarrow PC+1)$
SZA	rB_2 :	if $(AC = 0)$ then $(PC \leftarrow PC+1)$
SZE	rB_1 :	if $(E = 0)$ then $(PC \leftarrow PC+1)$
HLT	rB_0 :	$S \leftarrow 0$ (S is a start-stop flip-flop)

Memory-Reference Instructions

Symbol	Operation Decoder	Symbolic Description
AND	D ₀	$AC \leftarrow AC \wedge M[AR]$
ADD	D ₁	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D ₂	$AC \leftarrow M[AR]$
STA	D ₃	$M[AR] \leftarrow AC$
BUN	D ₄	$PC \leftarrow AR$
BSA	D ₅	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	D ₆	$M[AR] \leftarrow M[AR] + 1, \text{ if } M[AR] + 1 = 0 \text{ then } PC \leftarrow PC + 1$

Table : Memory-Reference Instructions

- The effective address of the instruction is in AR and was placed there during timing signal T₂ when I = 0, or during timing signal T₃ when I = 1.
- The execution of MR instruction starts with T₄

Memory-Reference Instructions

- AND to AC

$D_0T_4:$ $DR \leftarrow M[AR]$ Read operand

$D_0T_5:$ $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ AND with AC

- ADD to AC

$D_1T_4:$ $DR \leftarrow M[AR]$ Read operand

$D_1T_5:$ $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$ Add to AC and store
carry in E

- LDA: Load to AC

D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR, SC \leftarrow 0$


- STA: Store AC

D_3T_4 : $M[AR] \leftarrow AC, SC \leftarrow 0$

Memory-Reference Instructions


- BUN: Branch Unconditionally
 $D_4T_4: PC \leftarrow AR, SC \leftarrow 0$
- BSA: Branch and Save Return Address

Memory, PC, AR at time T4

20	0	BSA	135
PC = 21	Next instruction		
AR = 135			
136	Subroutine		
			
	1	BUN	135

Memory

Memory, PC after execution

20	0	BSA	135
21	Next instruction		
135	21		
PC = 136	Subroutine		
			
	1	BUN	135

Memory

Fig: BSA Instruction Execution

Memory-Reference Instructions

- BSA:

$D_5T_4: M[AR] \leftarrow PC, AR \leftarrow AR + 1$

$D_5T_5: PC \leftarrow AR, SC \leftarrow 0$

- ISZ: Increment and Skip-if-Zero

$D_6T_4: DR \leftarrow M[AR]$

$D_6T_5: DR \leftarrow DR + 1$

$D_6T_4: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$

Memory-Reference Instructions

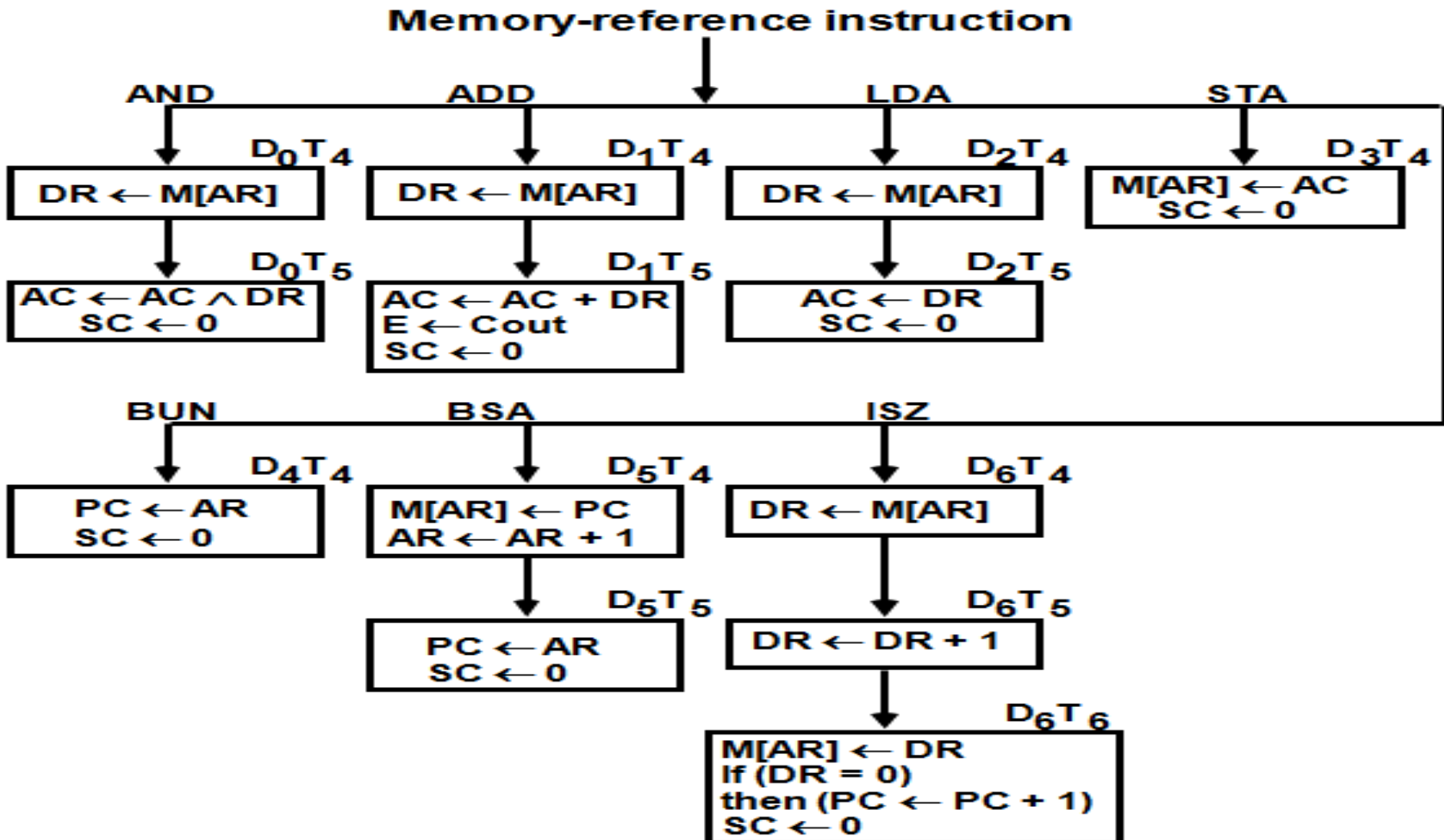


Fig: Flowchart for Memory Reference Instructions

Program Input-output and interrupt

Input-Output and Interrupt

- A Terminal with a keyboard and a Printer.

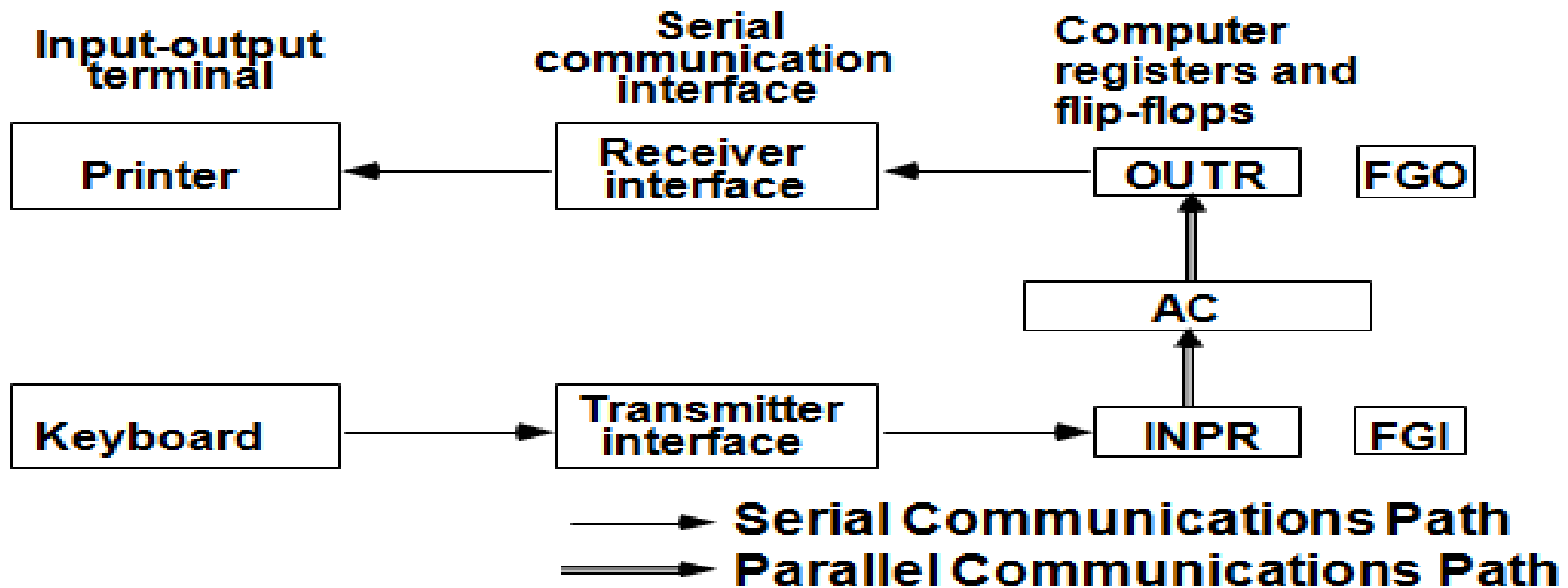


Fig: Input-Output Configuration

Input-Output and Interrupt

- The terminal sends and receives serial information.
- The serial info. from the keyboard is shifted into INPR .
- The serial info. for the printer is stored in the OUTR.
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing difference between I/O device and the computer.

FGO stands for "output flag" in computer architecture. It's a 1-bit control flip-flop that indicates whether information from the accumulator (AC) should be transferred to the output register (OUTR).

Here's how FGO works:

- Initially set to 1: When the computer starts, FGO is set to 1, which transfers information from the AC to the OUTR.
- Cleared to 0: When the operation is complete, the output device sets FGO to 0.
- Indicates output device is busy: When FGO is 0, the computer doesn't load a new character into the OUTR, which indicates that the output device is busy.

FGI stands for "first input flag" in computer architecture. It's a 1-bit control flip-flop that indicates when new information is available in an input device.

Here's how FGI works:

- Set**

When a key is pressed on the keyboard, the input flag FGI is set to 1.

- Cleared**

When the computer accepts the information, the input flag FGI is cleared to 0.

- Protected**

While the flag is set, the information in the input register cannot be changed by pressing another key.

- Transfer**

When the flag is cleared, the computer transfers the information from the input register to the accumulator in parallel.

Input-Output and Interrupt

$D_7|T_3 = p$

$IR(i) = B_i, i = 6, \dots, 11$

	p: $SC \leftarrow 0$	Clear SC
INP	pB₁₁: $AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input char. to AC
OUT	pB₁₀: $OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output char. from AC
SKI	pB₉: $\text{if}(FGI = 1) \text{ then } (PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB₈: $\text{if}(FGO = 1) \text{ then } (PC \leftarrow PC + 1)$	Skip on output flag
ION	pB₇: $IEN \leftarrow 1$	Interrupt enable on
IOF	pB₆: $IEN \leftarrow 0$	Interrupt enable off

Fig: Input-Output Instruction

Program Interrupt

- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. An interrupt flip-flop R is included in the computer. When $R = 0$, the computer goes through an instruction cycle. During the execute phase of the instruction cycle IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while $IEN = 1$, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

Program Interrupt

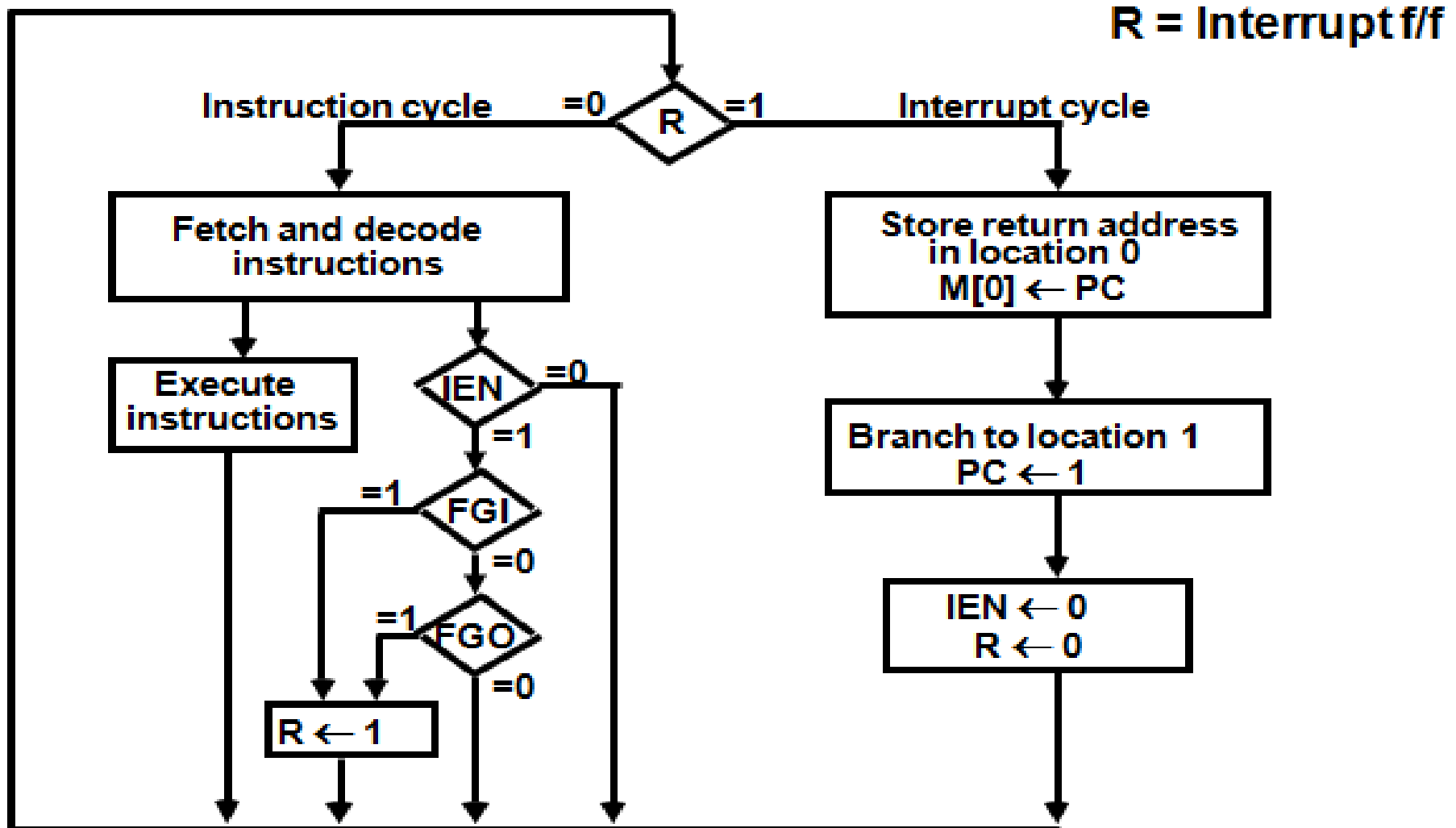


Fig: Flow chart for Interrupt Cycle

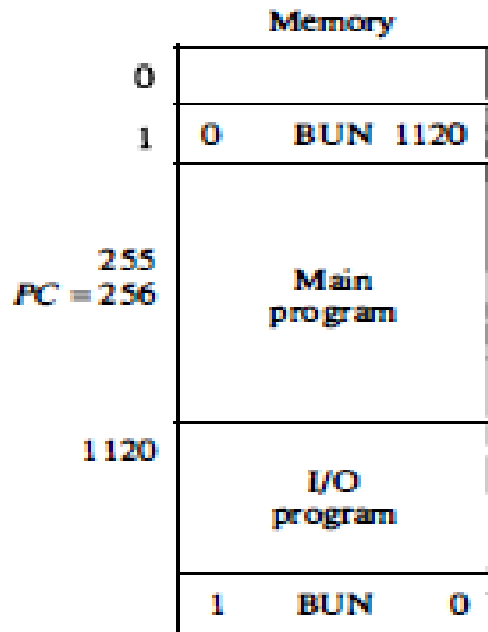
Program Interrupt

- An example that shows what happens during the interrupt cycle is shown in Fig. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig(a).
- When control reaches timing signal T_0 and finds that $R = 1$, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.

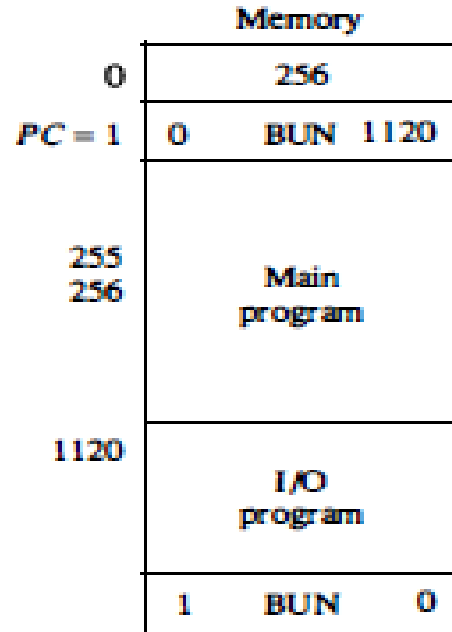
Program Interrupt

This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Fig. (b).

- Figure:** Demonstration of the interrupt cycle



(a) Before interrupt



(b) After interrupt cycle

Instruction Format

What is instruction format?

An instruction is normally made up of a combination of an operation code and some way of specifying an operand, most commonly by its location or address in memory



Fig. 9-3 Instruction Format with Mode Field

Types of address instructions

- Three address instructions
- Two address instructions
- One address instructions
- Zero address instruction

Three address instructions

- Memory addresses for the two operands and one destination need to be specified.
- It is also called **General register organization**.
- Instruction: ADD R1, R2, R3

Microoperation: $R1 \leftarrow R2 + R3$

Instruction Format

EVALUATE $X = (A + B) * (C + D)$

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

Two address instructions

- Two address registers or two memory locations are specified
- Assumes that the destination address is the same as that of the first operand.
- Instruction: ADD R1, R2

Microoperation: $R1 \leftarrow R1 + R2$

Instruction Format

EVALUATE $X = (A + B) * (C + D)$

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

Instruction Format

One address instructions

- One address can be a register name or memory address.
- **SINGLE ACCUMULATOR ORGANIZATION**
- It uses AC register for all data manipulation
- Instruction: ADD X

Microoperation: $AC \leftarrow AC + M[X]$

EVALUATE $X=(A+B)*(C+D)$

- LOAD A $AC \leftarrow M[A]$
- ADD B $AC \leftarrow AC + M[B]$
- STORE T $M[T] \leftarrow AC$
- LOAD C $AC \leftarrow M[C]$
- ADD D $AC \leftarrow AC + M[D]$
- MUL T $AC \leftarrow AC * M[T]$
- STORE X $M[X] \leftarrow AC$

Instruction Format

Zero address instruction

- Stack is used. Arithmetic operation pops two operands from the stack and pushes the result.
- Also called **stack organization**
- Evaluate $X = (A + B) * (C + D)$
- **PUSH A** **TOS \rightarrow A**
- **PUSH B** **TOS \rightarrow B**
- **ADD** **TOS \rightarrow (A+B)**
- **PUSH C** **TOS \rightarrow C**
- **PUSH D** **TOS \rightarrow D**
- **ADD** **TOS \rightarrow (C+D)**
- **MUL** **TOS \rightarrow (C+D)*(A+B)**
- **POP X** **M[X] \rightarrow TOS**

Push A
Push B
ADD
Push C
Push D
ADD
Mult
Store

Addressing Modes

Addressing Modes

- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction .

(OR)

- Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced).

(OR)

- Addressing mode specifies a rule for **interpreting or modifying** the **address field** of the instruction before the operand is actually executed.

Need for different addressing modes

1. To give programming flexibility to the user i.e pointers to memory, Counters for loop control, indexing of data, program relocation.
2. To use the bits in the address field of the instruction efficiently i.e reduce the number of bits.

Addressing Modes

Different Addressing Modes –

- Implied mode
- Immediate mode
- Register mode
- Register indirect mode
- Autoincrement or Autodecrement
- **Direct address mode**
- **Indirect address mode**
- Relative address mode
- Indexed addressing mode
- Base register addressing mode

Addressing Modes

Implied Mode

- In this mode the operands are specified implicitly in the definition of the instruction.
- All register reference instructions that use an accumulator are implied mode instruction.
- Example: CMA compliment accumulator.

Immediate Addressing

- Data needed by the processor is contained in the instruction
- Operand= address field

Ex. -ADD 5

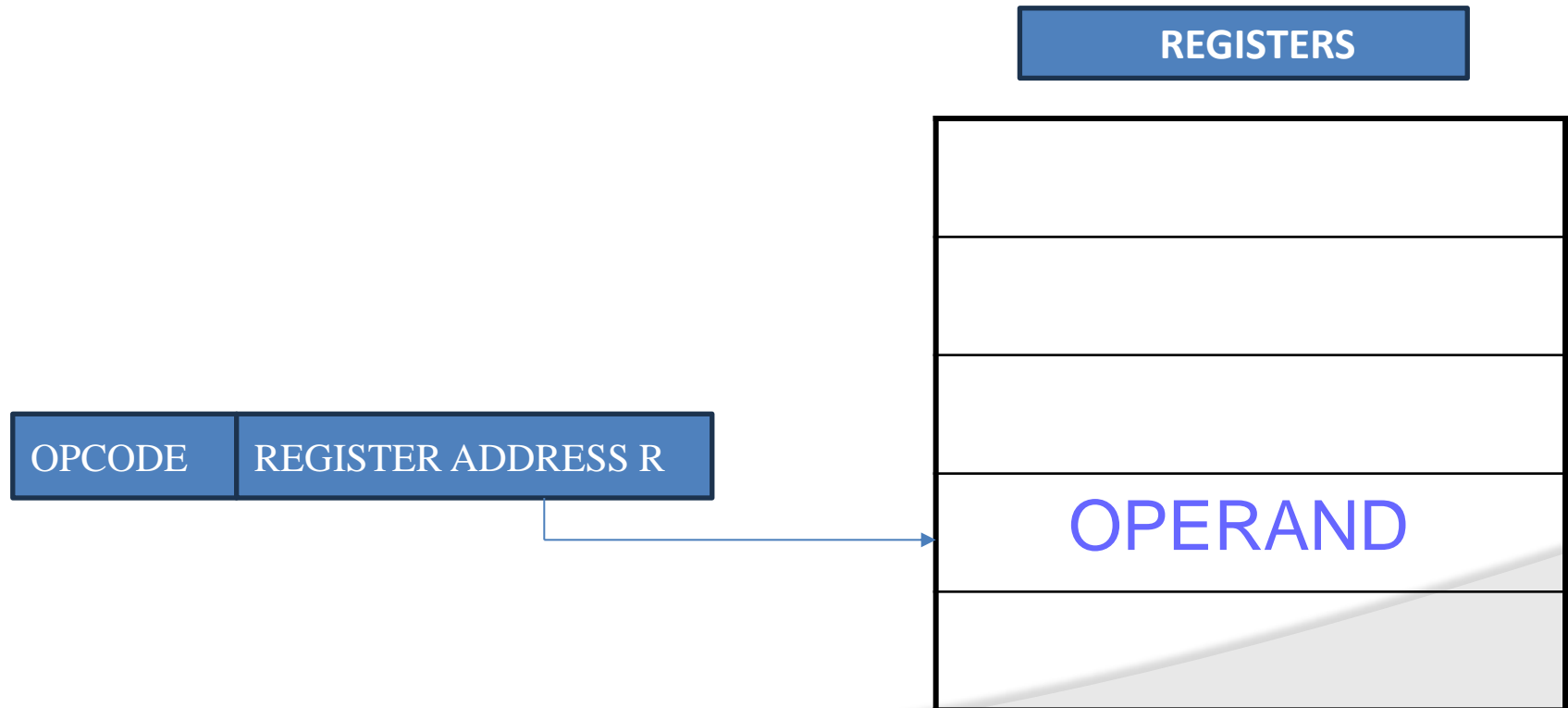
- add 5 to the content of accumulator
- 5 is the operand



Addressing Modes

Register Addressing

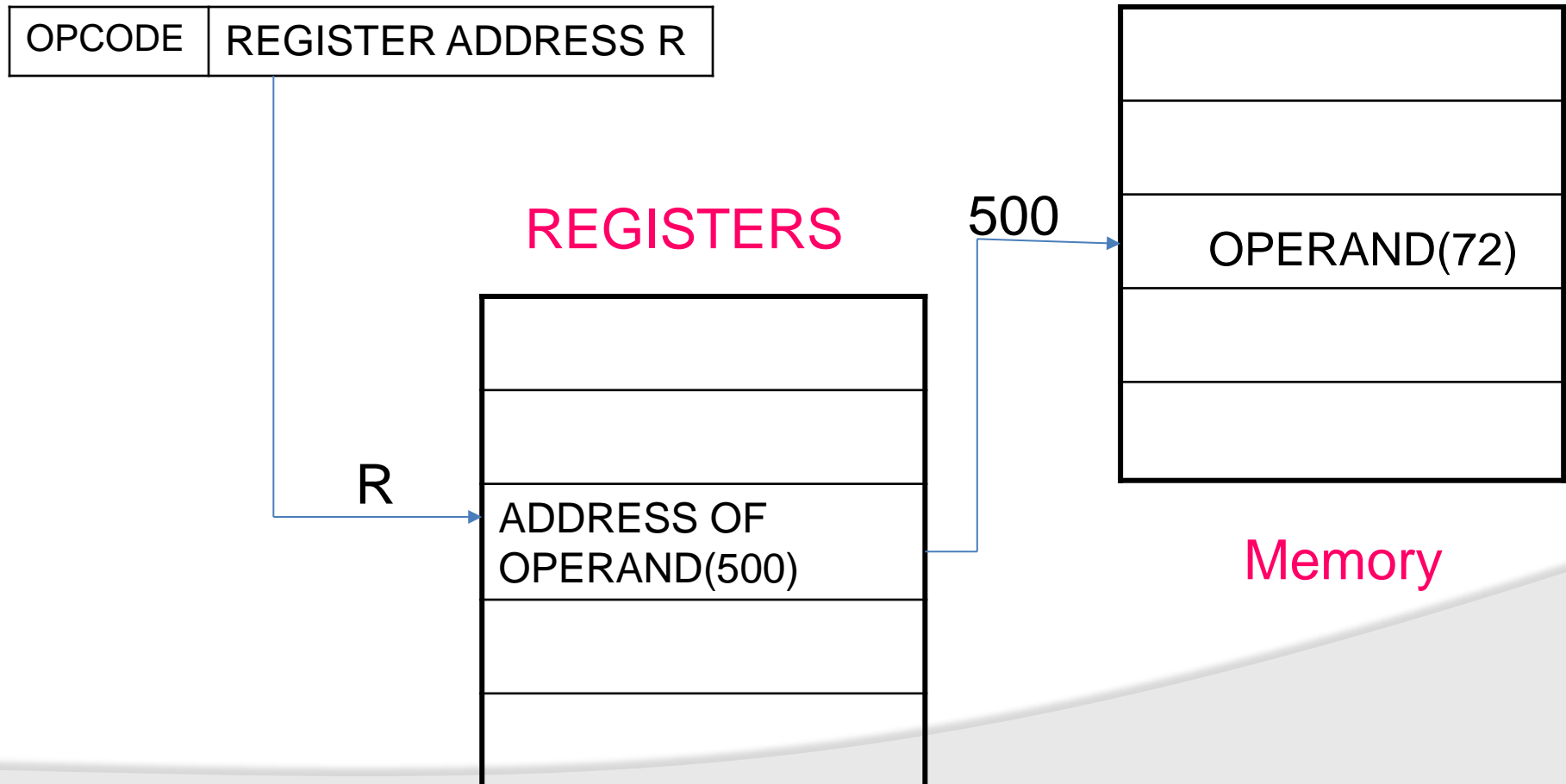
- Operand is held in register named in address field
- A k-bit field can specify any one of 2^k registers



Addressing Modes

Register Indirect Addressing

- The selected register contains the address of the operand rather than the operand itself.



Addressing Modes

Auto Increment or Auto Decrement Mode

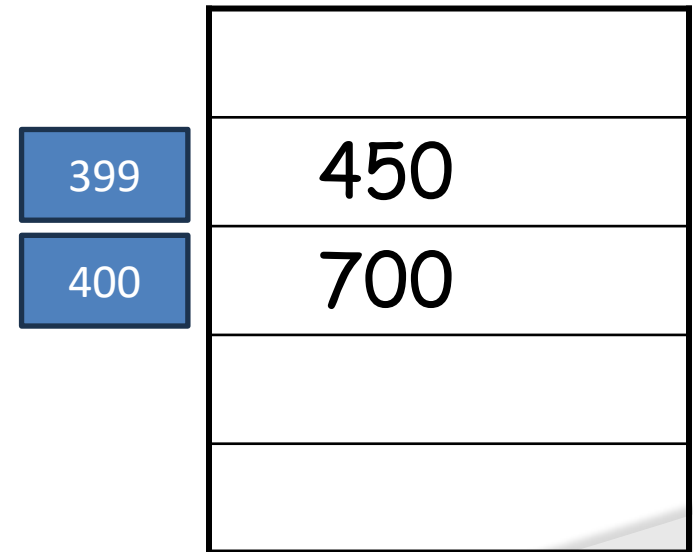
This is similar to the register indirect mode except that the register is **incremented or decremented after or before** its value is used to access memory.

R1=400

1. Auto incr.-E.A=400

R1 is incremented to 401 after the execution

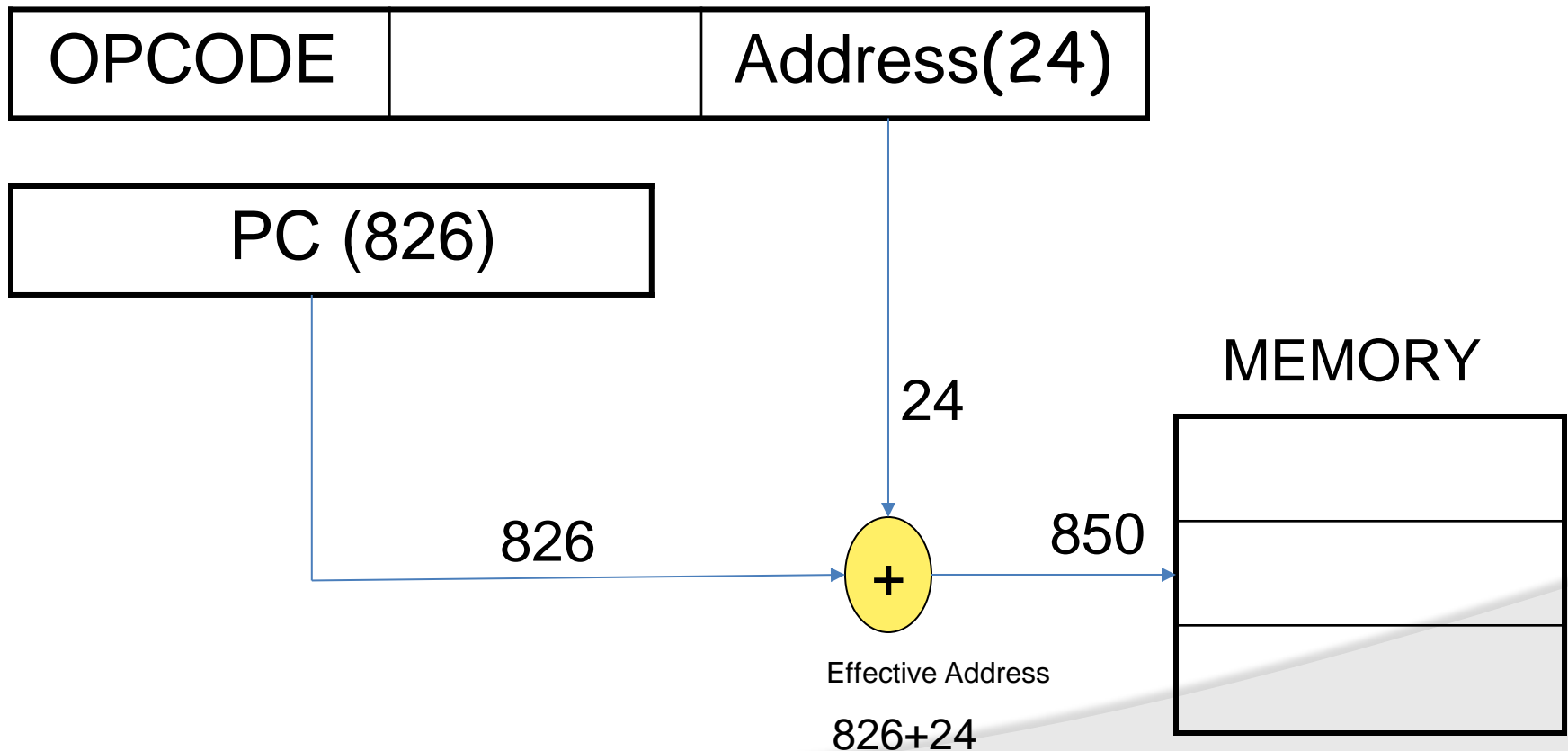
2. Auto decr.-R1 is decremented prior to the execution, therefore, R1=399 and E.A=399



Addressing Modes

RELATIVE ADDRESS MODE

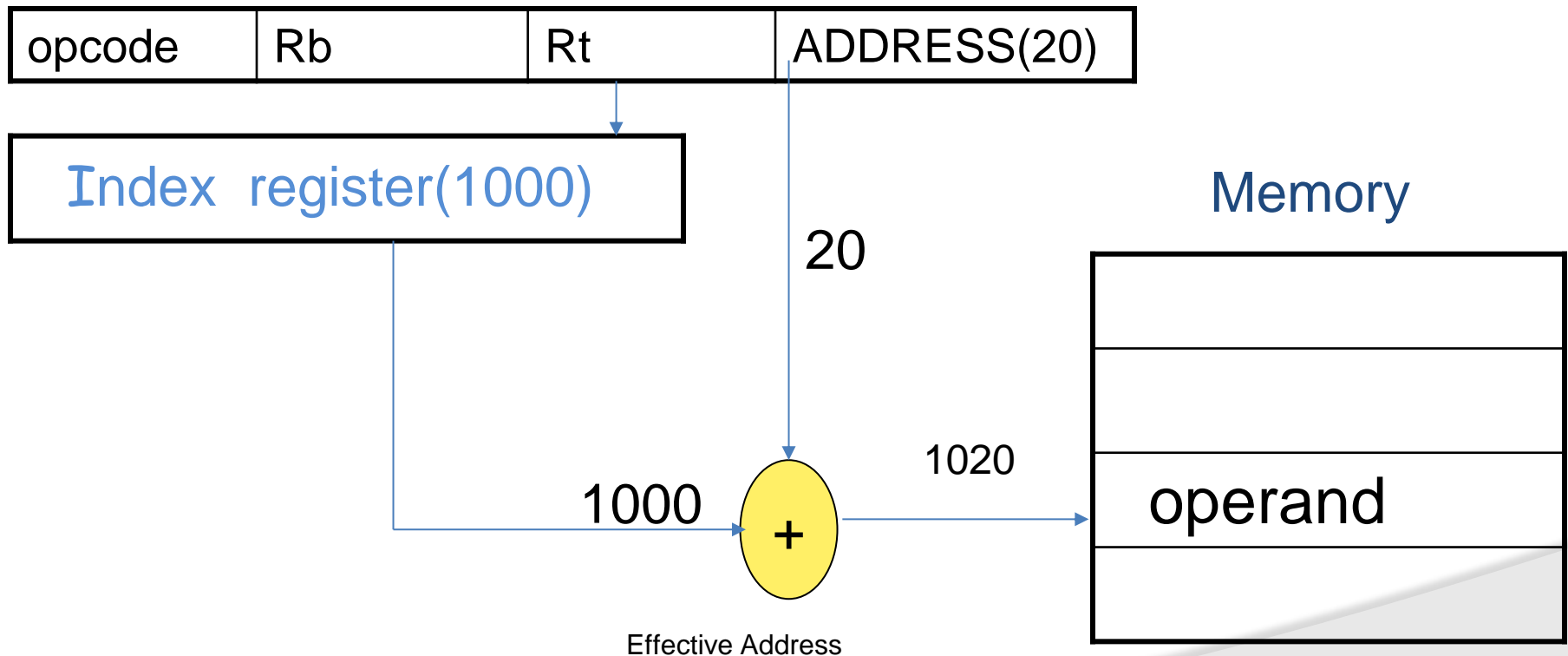
In this mode the **content of the program counter** is added to the **address part** of the instruction to obtain the effective address.



Addressing Modes

Indexed addressing mode

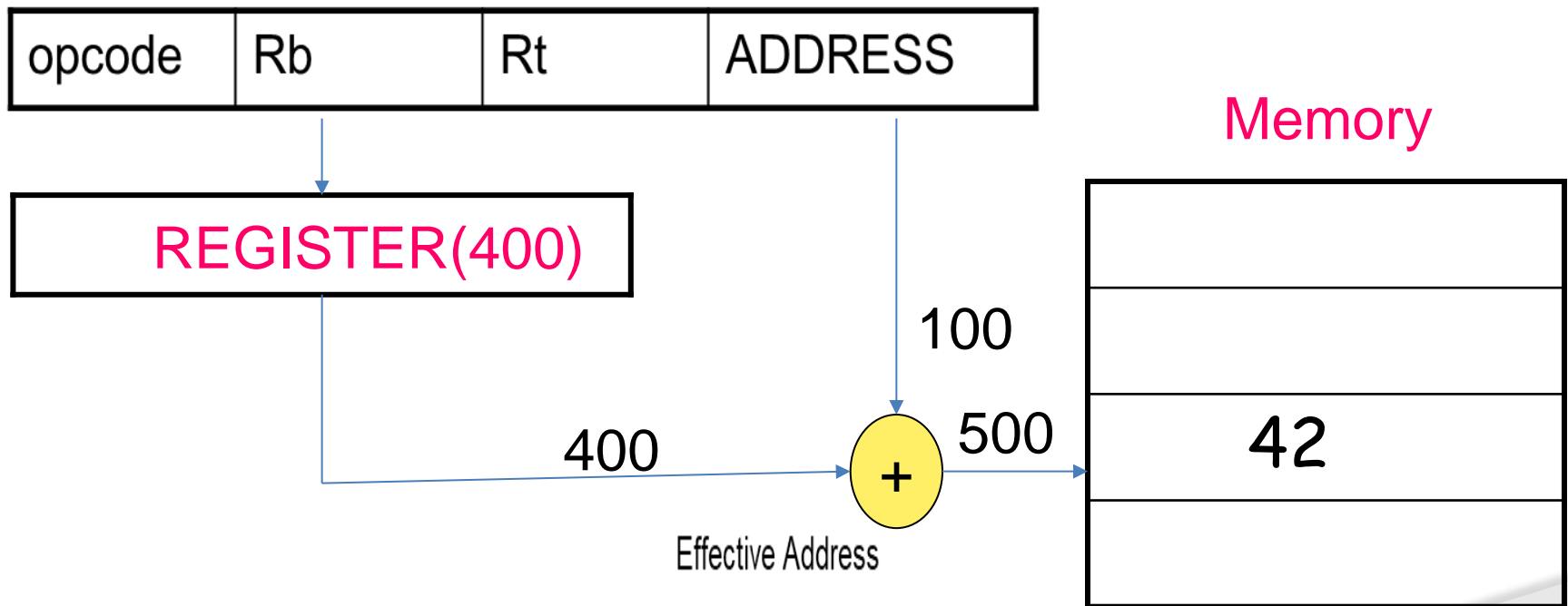
In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.



Addressing Modes

BASE REGISTER ADDRESSING MODE

In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.



Addressing Modes

Example :

PC = 200

R1 = 400

XR = 100

AC

Addressing Mode	Effective Address	Content of AC
Direct address	500	$/* AC \leftarrow (500) */$ 800
Immediate operand	-	$/* AC \leftarrow 500 */$ 500
Indirect address	800	$/* AC \leftarrow ((500)) */$ 300
Relative address	702	$/* AC \leftarrow (PC+500) */$ 325
Indexed address	600	$/* AC \leftarrow (RX+500) */$ 900
Register	-	$/* AC \leftarrow R1 */$ 400
Register indirect	400	$/* AC \leftarrow (R1) */$ 700
Autoincrement	400	$/* AC \leftarrow (R1)+ */$ 700
Autodecrement	399	$/* AC \leftarrow -(R) */$ 450

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

Data Transfer and Manipulation Instructions



- Computer instructions can be classified into three categories.
 1. Data Transfer Instructions
 2. Data Manipulation Instructions
 3. Program Control Instructions
- Data Transfer Instructions transfer the data from one location to another location.
- Data manipulation instructions performs arithmetic ,logic and shift operations on the data.
- Program control instructions provide decision making and change the path taken by the program when executed in the computer.

Data Transfer Instructions

- Data Transfer instructions move the data between
 - Memory ----- Processor register
 - Processor register ----- Input/output
 - Processor registers ----- Processor registers

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Table : Typical data Transfer Instructions

Data Transfer Instructions

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD \$ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$
Autodecrement	LD -(R1)	$R1 \leftarrow R1 - 1, AC \leftarrow M[R1]$

Table : Addressing modes for load instruction

Data Manipulation Instructions

- Data Manipulation Instructions are of three basic types.
 1. Arithmetic Instructions
 2. Logical and Bit Manipulation Instructions
 3. Shift Instructions

1. Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

Mnemonic
ADDI
ADDF
ADDD

Table: Typical Arithmetic Instructions

2. Logical and Bit manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Table : Logical and Bit Manipulation Instructions

Data Manipulation Instructions

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

Op REG TYPE RL COUNT-ShiftInstruction

Program Control

- When the program control instruction is executed it change the address value in the program counter and cause the flow of control to be altered.

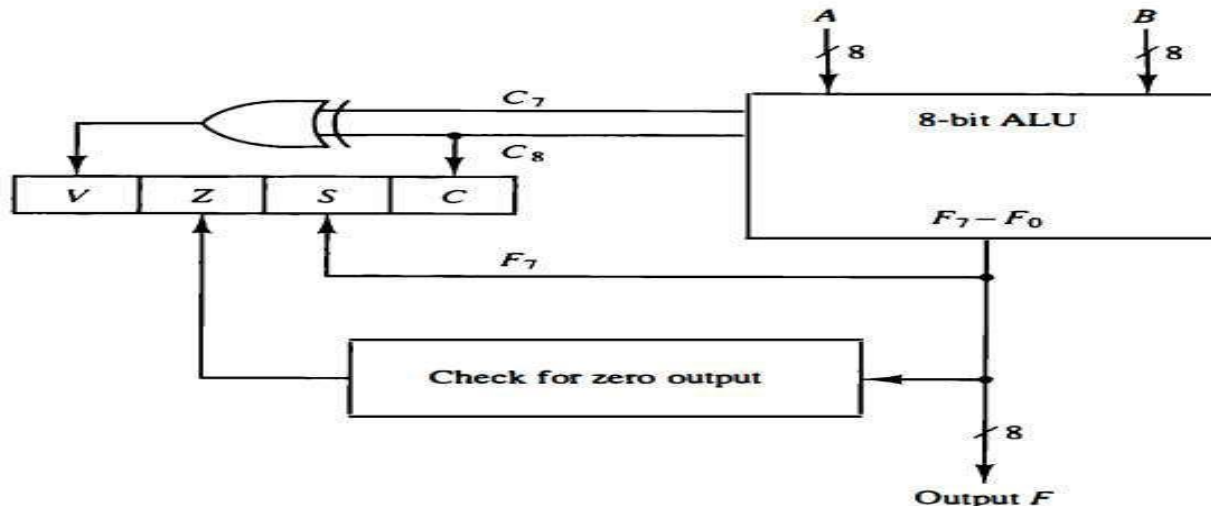
Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by —)	CMP
Test(by AND)	TST

Table: Program Control Instructions

- Branch and Jump instructions are conditional and Unconditional Instructions.
- CMP and Test set some of the bits in PSW(Processor status Word).

Program Control

- In Basic Computer, the processor had several (status) flags – 1 bit value that indicated various information about the processor's state – E, FGI, FGO, I, IEN, R.
- In some processors, flags like these are often combined into a register – the processor status register (PSR); sometimes called a processor status word (PSW).
- Common flags in PSW are
 - C (Carry): Set to 1 if the carry out of the ALU is 1
 - S (Sign): The MSB bit of the ALU's output
 - Z (Zero): Set to 1 if the ALU's output is all 0's
 - V (Overflow): Set to 1 if there is an overflow



Status Flag Circuit

Program Control

- Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
- Bit S (sign) is set to 1 if the highest-order bit F, is 1. It is set to 0 if the bit is 0.
- Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, $Z = 1$ if the output is zero and $Z = 0$ if the output is not zero.

•Subroutine Call and Return

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

During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program.

Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a branch is made back to the main program.

Program Control

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
<i>Unsigned compare conditions (A - B)</i>		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
<i>Signed compare conditions (A - B)</i>		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Table: conditional branch instructions

Program Control

- **Types of Interrupts**

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

- Internal interrupts are synchronous with the program while external interrupts are asynchronous.
- If the program is rerun, the internal interrupts will occur in the same place each time.
- External interrupts depend on external conditions that are independent of the program being executed at the time.
- External and internal interrupts are initiated from signals that occur in the hardware of the CPU.

Program Control

- A software interrupt is initiated by executing an instruction.
- Software interrupt is a special call instruction that behave 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.

RISC (Reduced Instruction Set Computer)

What is RISC?

RISC - Reduced Instruction Set Computer

RISC is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures (**i.e. CISC - Complex Instruction Set Computer**).

History Of RISC:

1. RISC approach developed in 1970's.
2. Increase in memory size.
3. Decrease in cost.
4. Advanced compilers.
5. In late 1970's IBM was the first to start.
6. In 1980 , David Patterson ,began the project that gives this approach RISC.

RISC (Reduced Instruction Set Computer)

Characteristics Of RISC:

- Simplified instructions , taking 1 clock cycle.
- Large number of general purpose registers.
- Circuit is much simpler.
- Fast to decode.
- Fast to execute.
- Pipelining- fetching of next instruction while previous instruction executes.

RISC Has Five Design Principles:

- Simple Instructions
- Register-to-Register Operations
- Simple Addressing Modes
- Large Register Set
- Fixed-Length

RISC (Reduced Instruction Set Computer)

Simple Instructions:

The objective is to design simple instruction so that each can execute in one cycle.

Register-to-Register Operations:

RISC processors only allow LOAD/STORE operations to access memory. Rest of the operations work on the register-to-register basis. This feature of restricting operands to registers also simplifies the control-unit.

Simple Addressing Modes:

- RISC processors employ register-to-register instruction so most instruction use register based addressing.
- Only LOAD/STORE instructions need memory addressing modes.

RISC (Reduced Instruction Set Computer)



Large Register Set:

- For register-to-register operation large number of registers required.
- Provide ample opportunities for the compiler to optimize their usage.

Fixed-Length:

- RISC design use fixed-length instructions.
- Variable length instructions cause implementation and execution inefficient.
- The boundaries of various fields in an instruction such as opcode and source operands are fixed.
- This allows efficient decoding and scheduling of instructions.

What Actually RISC Does?

- Break Operation into Simple Sub-Operation.

Example:-

Load X, Load Y,

add X and Y,

Store on Z

$$X * Y \rightarrow Z$$

RISC (Reduced Instruction Set Computer)

In Real Life Use of RISC Architectures:

- RISC architectures are now used across a wide range of platforms, from cellular telephones and tablet computers.
- Intel was able to spend vast amounts of money on processor development to offset the RISC advantages enough to maintain PC market share.
- New microprocessors can be developed and tested more quickly if being less complicated is one of its aims.