Computer architecture

Machine Instructions

Machine instruction

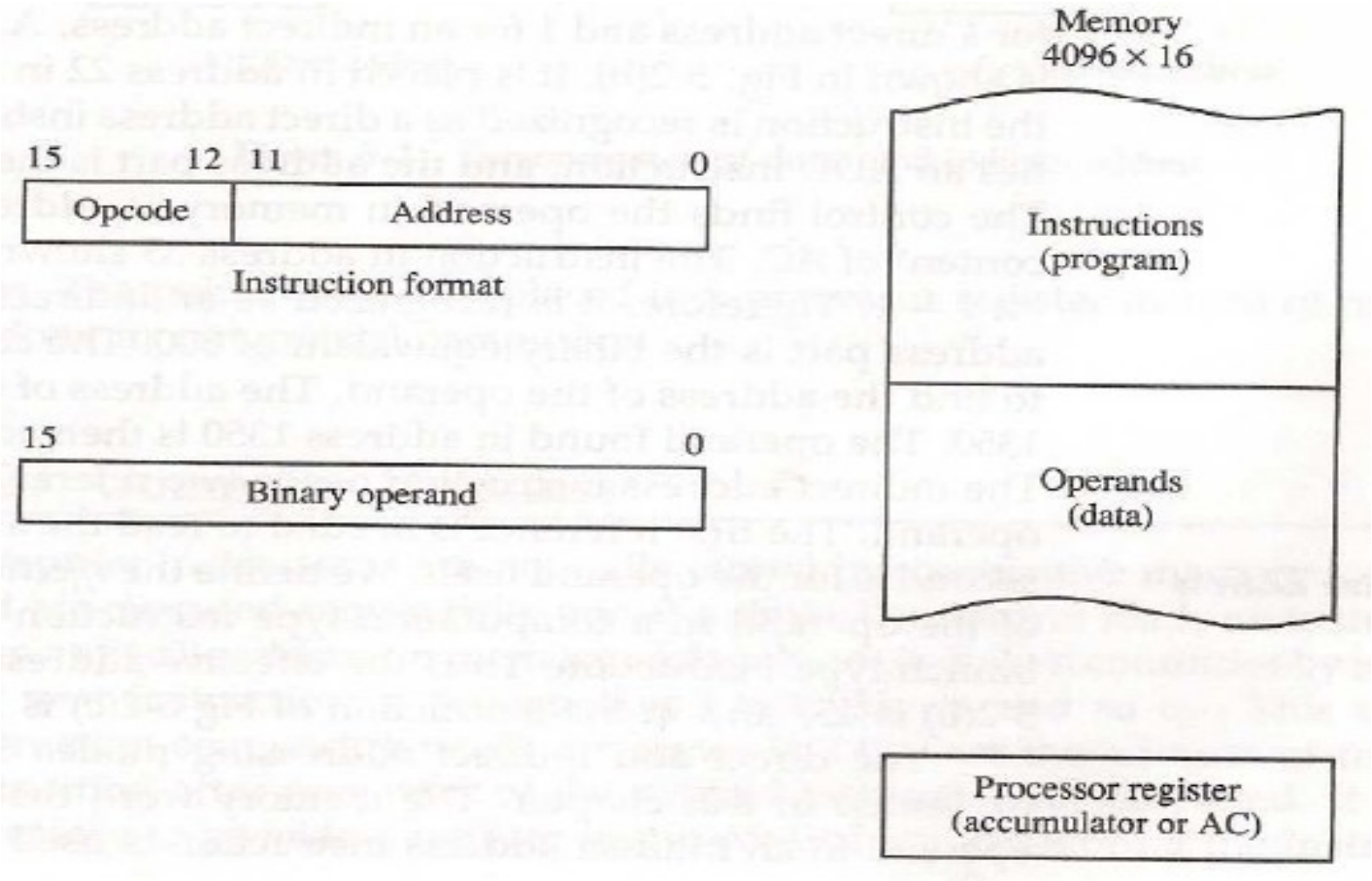
- Instruction Codes
  - A program is a set of *instructions* that specify the operations, operand, and the sequence
  - An instruction is a binary code that specifies a sequence of micro-operations
  - Codes and data are stored in memory
  - The computer reads each instruction from memory and *places it in a control register*.
  - The control then *interprets the binary code* and proceeds to *execute it*

Machine instruction

- Instruction Code:
  - A group of bits that instruct the computer to perform a specific operation
  - It is usually divided into parts

- Operation Code:
  - The most basic part of an instruction code
  - A group of bits that define such operations as add, subtract, multiply, shift, and complement

Machine instruction



Machine instruction

- **Stored Program Organization**
  - Instruction code format with two parts:
    Op. Code + Address
  - Op. Code: specify 16 possible operations *(4 bit)*
  - Address: specify the address of an operand *(12 bit)*
  - Unused part is used for other purpose
  - Memory: 12 bit = 4096 word
    (Instruction and Data are stored)
  - Store each instruction code (***program***) and operand (***data***) in 16-bit memory word

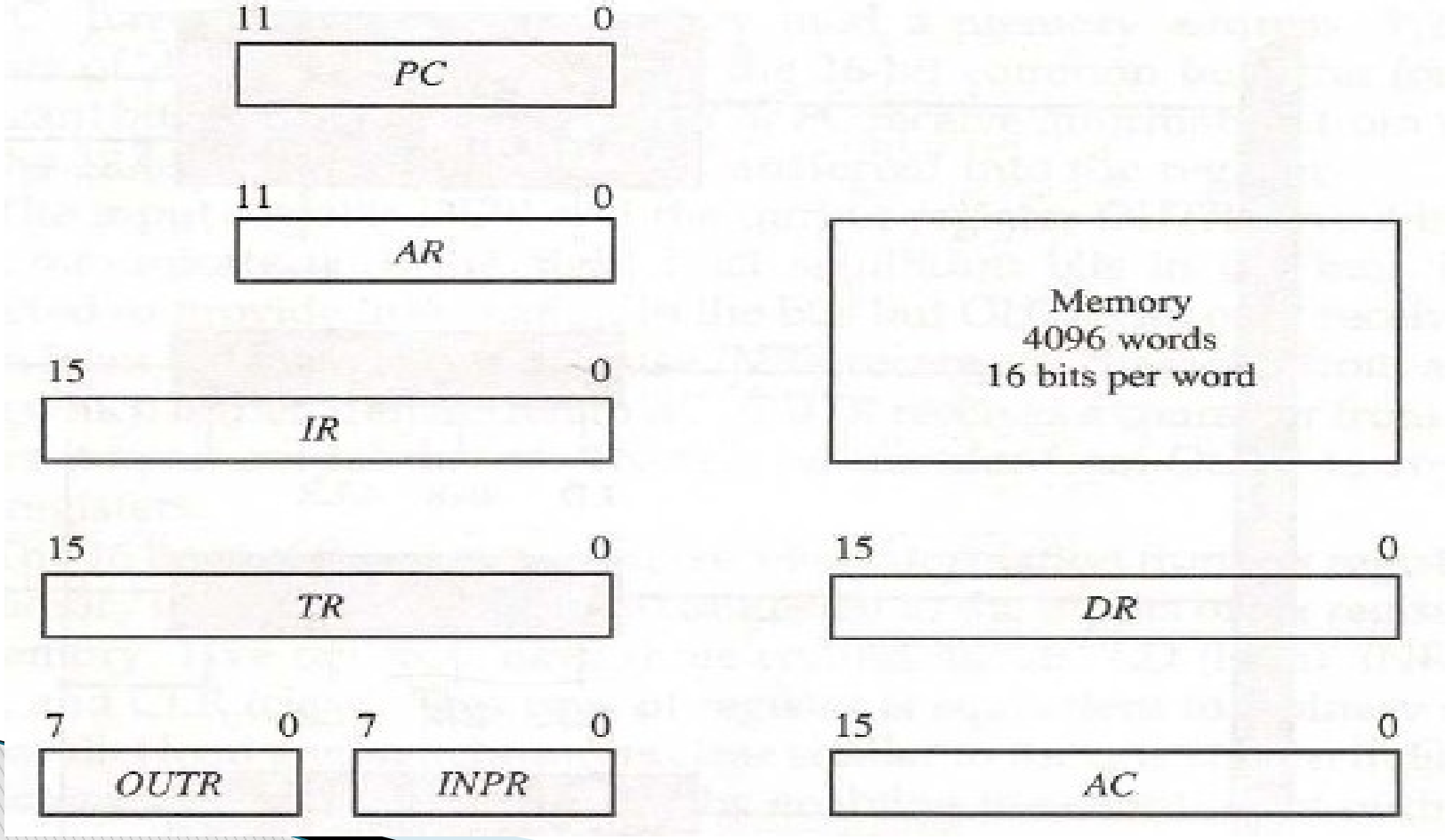
Machine instructions

- List of registers in basic computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Machine instructions

- Registers and memory of basic computer:



Machine instruction

- **Functions of basic registers:**
  - Data Register (***DR***): hold the operand (data) read from memory
  - Accumulator Register (***AC***): general purpose processing register
  - Instruction Register (***IR***): hold the instruction read from memory
  - Temporary Register (***TR***): hold a temporary data during processing
  - Address Register (***AR***): hold a memory address, 12 bit width
  - Input Register (***INPR***): receive an 8-bit character from an input device
  - Output Register (***OUTR***): hold an 8-bit character for an output device

Machine instructions

- Program Counter (***PC***):
  - hold the address of the next instruction to be read from memory after the current instruction is executed
  - A branch instruction calls for a transfer to a non-consecutive instruction in the program
  - The address part of a branch instruction is transferred to PC to become the address of the next instruction
  - To read instruction, memory read cycle is initiated, and PC is incremented by one

Machine instructions

- **Timing and Control:**
  - A master clock generator controls the timing for all registers in the basic computer
  - The clock pulses are applied to all F/Fs and registers in system
  - 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
  - It provides controls for the mux in the common bus, registers etc.

Machine instructions

- Hardwired Control
  - The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
  - For any change, need to change hardware
  - Fast operation

- Microprogrammed Control
  - The control information is stored in a programmed control memory
  - Any required change can be done by updating the microprogram in control memory,
  - Slow operation

Machine instructions

- **Instruction Cycle:**
- 1) Instruction Fetch from Memory – read instruction from memory through the data bus
- 2) Instruction Decode – the control unit determine the type of instruction that was just read from memory
- 3) Fetch the operand – accumulate the operands to make operation; read from memory in case of indirect addressing mode (operands are residing data in memory)
- 4) Instruction Execution – perform the operation based on decoded Opcode
- 5) Go to step 1): Next Instruction [PC+1]

Machine instructions

- **Instruction Formats:**
- 1) *Operation Code Field*: the operation code field of an instruction is a group of bits that specifies various operations such as add, subtract, complement and shift.
- 2) *Address Field*: specifies the memory address or a processor register
- 3) *Mode Field*: defines a variety of alternatives for choosing the operands from the given address.

Machine instructions

The address field depends on the arrangement of registers

Most computers fall into one of three types of CPU organizations:

- 1) Single Accumulator Organization: **ADD X**
- 2) General Register Organization: **ADD R1**, **R2**, **R3**
- 3) Stack Organization: **PUSH X**

Machine instructions

Instructions with different address fields

- 1) Three-Address Instruction
- ADD R1, A, B R1 M[A]+M[B]
- MUL X, R1, R2 M[X] R1*R2
  - Each address fields specify either a processor register or a memory operand
  - Short program
  - Require too many bit to specify 3 address
- 2) Two-Address Instruction
- MOV R1, A R1 M[A]
- MUL R1, R2 R1 R1*R2
- MOV X, R1 M[X] R1
  - The most common in commercial computers
  - Each address field specifies either a processor register or a memory operand

Machine instructions

Instructions with different address fields

- 3) One-Address instruction
- LOAD A AC □M[A]
- ADD B AC □AC+M[B]
  - All operations are done between the AC register and memory operand
- 3) Zero-Address instruction
- PUSH A TOS □A
- PUSH B TOS □B
- ADD TOS □A+B
  - Stack-organized computer does not use an address field for the instructions ADD, and MUL
  - PUSH, and POP instructions need an address field to specify the operand
  - Zero-Address: absence of address (ADD, MUL)

Machine instructions

Addressing Mode

- It specifies a rule for interpreting the address field operands
- Implied Mode: Operands are specified implicitly in definition of the instruction
- *Examples*
  - COM: Complement Accumulator
  - Operand in AC is implied in the definition of the instruction
  - POP: Stack pop
  - Operand is implied to be on top of the stack

Machine instructions

Addressing Mode

- Immediate Mode
- Operand field contains the actual operand. It is useful for initializing registers to a constant value.
  - *Example*: LD #NBR AC ←NBR
- Register Mode
- Operands are in registers. Register is selected from a register field in the instruction
  - *Example*: LD R1 AC R1

Machine instructions

Addressing Mode

- Register Indirect Mode
- Selected register contains the address of the operand rather than the operand itself
  - *Example*: LD (R1) AC ← M[R1]
- Auto-increment or Auto-decrement Mode
- Similar to the register indirect mode except that
- the register is *incremented after* its value is used to access memory
- the register is *decrement before* its value is used to access memory
  - *Example* (*Auto-increment*): LD (R1)+
    AC ← M[R1], R1 ← R1+1

Machine instructions

Addressing Mode

- Direct Addressing Mode
- Effective address is equal to the address field of the instruction (Operand)
  - *Example*: LD ADR AC M[ADR]
-
- Indirect Addressing Mode
- Address field of instruction gives the address where the effective address is stored in memory
  - *Example*: LD @ADR AC M[M[ADR]]

Machine instructions

Addressing Mode

- Relative Addressing Mode
- PC is added to the address part of the instruction to obtain the effective address
  - *Example*: LD $ADR AC M[PC+ADR]
-
- Indexed Addressing Mode
- XR (*Index register*) is added to the address part of the instruction to obtain the effective address
  - *Example*: LD ADR(XR) AC M[ADR+XR]

Machine instructions

Addressing Mode

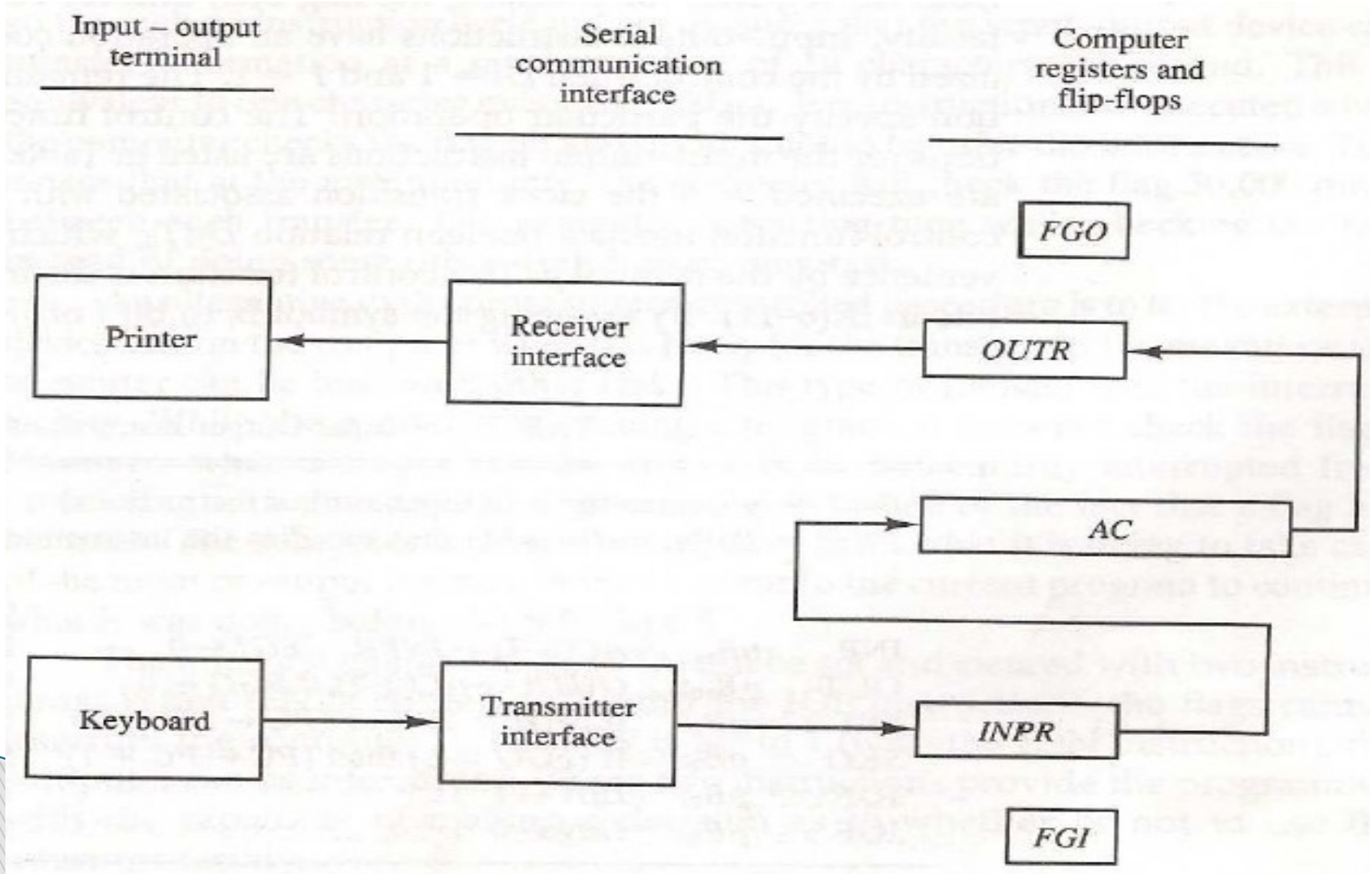
- Base Register Addressing Mode
- the content of a base register is added to the address part of the instruction to obtain the effective address
  - base register (BR) : LD ADR(BR)
    AC □M[BR+ADR]
  - base register hold a base address

Machine instructions

Input-Output Configuration

- The communication with I/O is performed through two registers:
  Input Register (***INPR***), Output Register (***OUTR***)
- The registers communicate with I/O serially and with the AC in parallel
- Each quantity of information has eight bits of an alphanumeric code
  - The serial data from the keyboard is shifted into the input register INPR.
  - The serial information for the printer is stored in the output register OUPR.

Machine instructions



Machine instructions

I/O Instruction

- INP and OUT are the common instructions for I/O processing.
- INP – AC(0-7) INPR, FGI 0; Input character, FGI is the input flag
- OUT – OUTR AC(0-7), FGO 0; Output character, FGO is the output flag

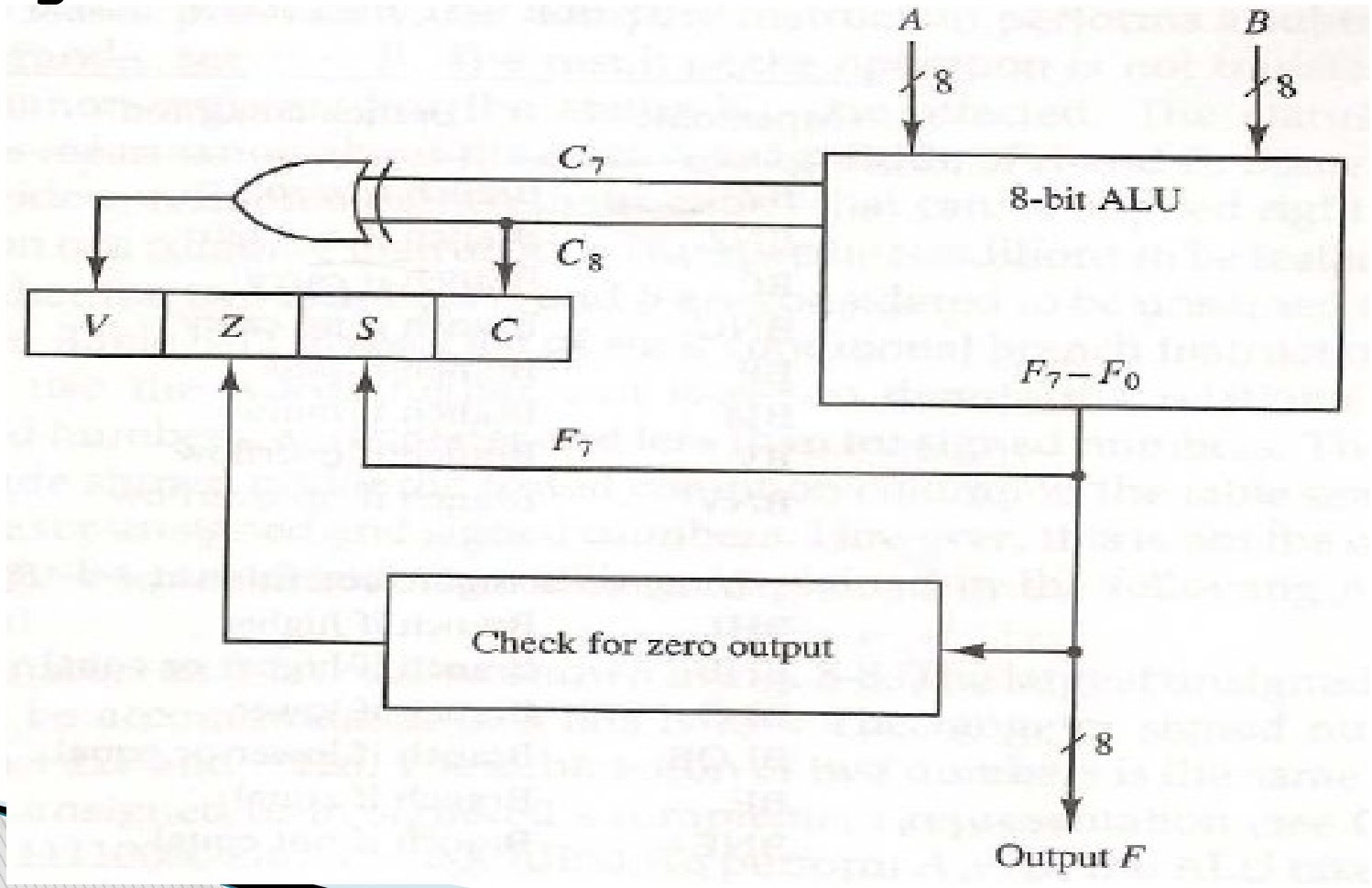
Machine instructions

Program control with Status bits

- The status of 4-bits status register is used to control the program.
- The bits are set or cleared as a result of an operation performed in the ALU.
  - Bit **C** (*carry*) : set to 1 if the end carry C8 is 1
  - Bit **S** (*sign*) : set to 1 if F7 of the output (F7-F0) is 1
  - Bit **Z** (*zero*) : set to 1 if the output of the ALU contains all 0's
  - Bit **V** (*overflow*) : set to 1 if the XOR of the last two carries (C8 and C7) is equal to 1

Machine instructions

Program control with Status bits



Machine instructions

Data transfer and manipulation:

- Most computer instructions can be classified into three categories:
  - 1) Data transfer
  - 2) Data manipulation
  - 3) Program control instructions

Machine instructions

Data Transfer Instruction: Transfer of data from one location to another without changing content.

- Typical Data Transfer Instruction:
- Load transfer from memory to a processor register, usually an AC (*memory read*)
- Store transfer from a processor register into memory (*memory write*)
- Move transfer from one register to another register
- Exchange swap information between two registers or a register and a memory word
- Input/Output transfer data among processor registers and input/output device
- Push/Pop transfer data between processor registers and a memory stack

Machine instructions

Data manipulation instruction: Perform operations on data and provide the computational capabilities for the computer.

- *Arithmetic Instructions*: Performs basic arithmetic operations on the specified operands. Examples: INC, DEC, ADD, SUB, MUL, DIV
- *Logical and Bit Manipulation Instructions*: Logical instructions perform binary operations on the strings of bits stored in registers. Example: CLR, OR, AND, COM
- *Shift Instructions*: Shits are operations in which the bits of a word are moved to the left or right. Example: SHR, ROR, SHL

Machine instructions

Program control instructions:

- Program control instruction is used to make the flow of program controlled to be altered.
- It specifies conditions for altering the content of the program counter breaking the sequence of instruction execution.
- Example: BR, JMP, CALL.

Machine instructions

Interrupt Handling:

- Interrupt is an event that changes the sequence in which the processor executes instructions.
- Transfer program control from a currently running program to another service program
- Control returns to the original program after the service program is executed

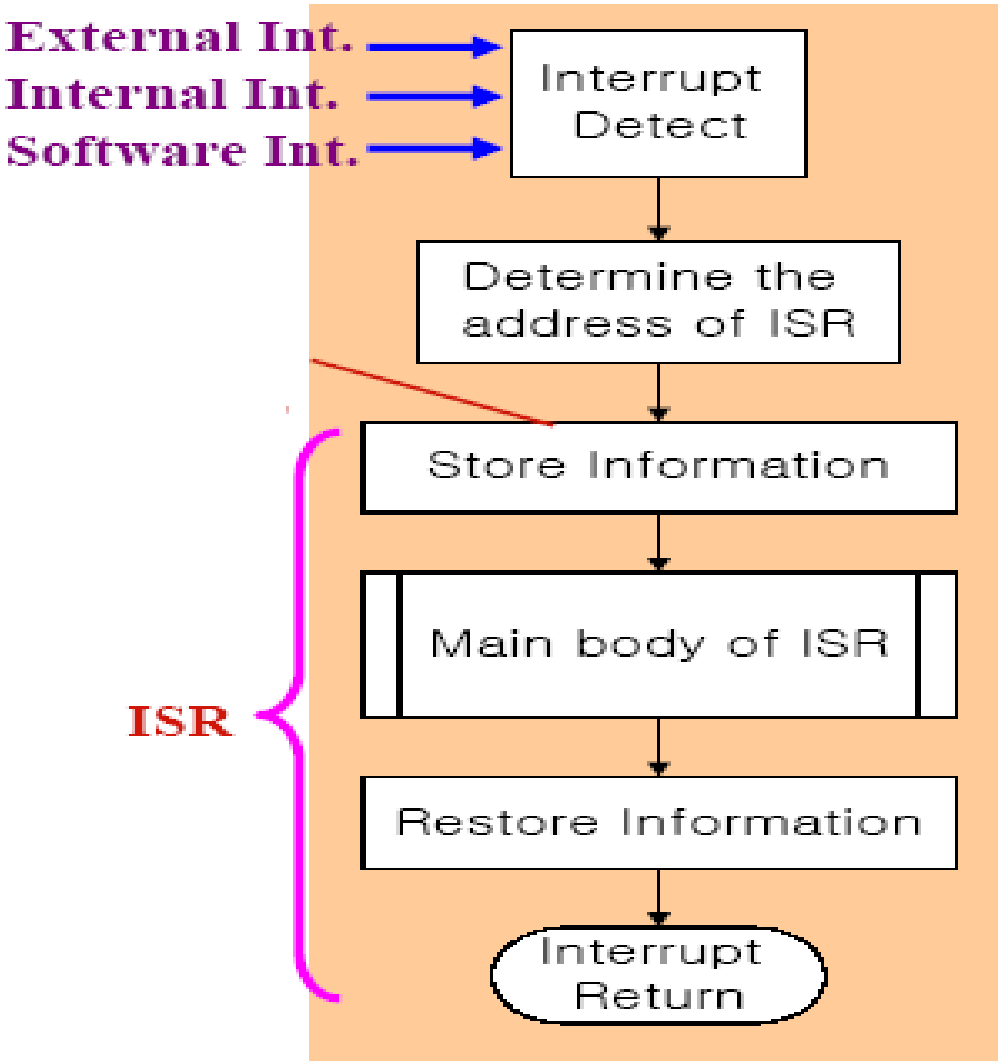
Machine instructions

Interrupt Handling:

- **Steps of Interrupt Service**
- An interrupt is initiated by an internal or external signal
- The address of the interrupt service routine (ISR) is determined by the hardware
- An interrupt procedure stores all the information necessary to define the state of the CPU
- Execute the ISR
- Restore the original state (control comes to the original program)

Machine instructions

Interrupt Handling:



Machine instructions

Types of Interrupts:

- External Interrupts - come from I/O device, from a timing device, from a circuit monitoring the power supply, or from any other external source
- Internal Interrupts - caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation
- Software Interrupts - initiated by executing an instruction used by the programmer to initiate an interrupt procedure at any desired point in the program