Program controlled Input/Output

- **Program-controlled I/O**

  -**Continuous CPU involvement**
  CPU keeps checking flag bit. If 1 then initiates transfer
  I/O takes valuable CPU time

  -**Difference in information flow rate makes this type of transfer inefficient**

- **Alternative approach is to let external device inform the computer when it is ready for transfer, in meantime computer can be busy with other task**

- **Interrupt**

Interrupt Initiated Input/Output

- Open communication only when some data has to be passed --> *interrupt*.

- The I/O interface, instead of the CPU, monitors the I/O device.

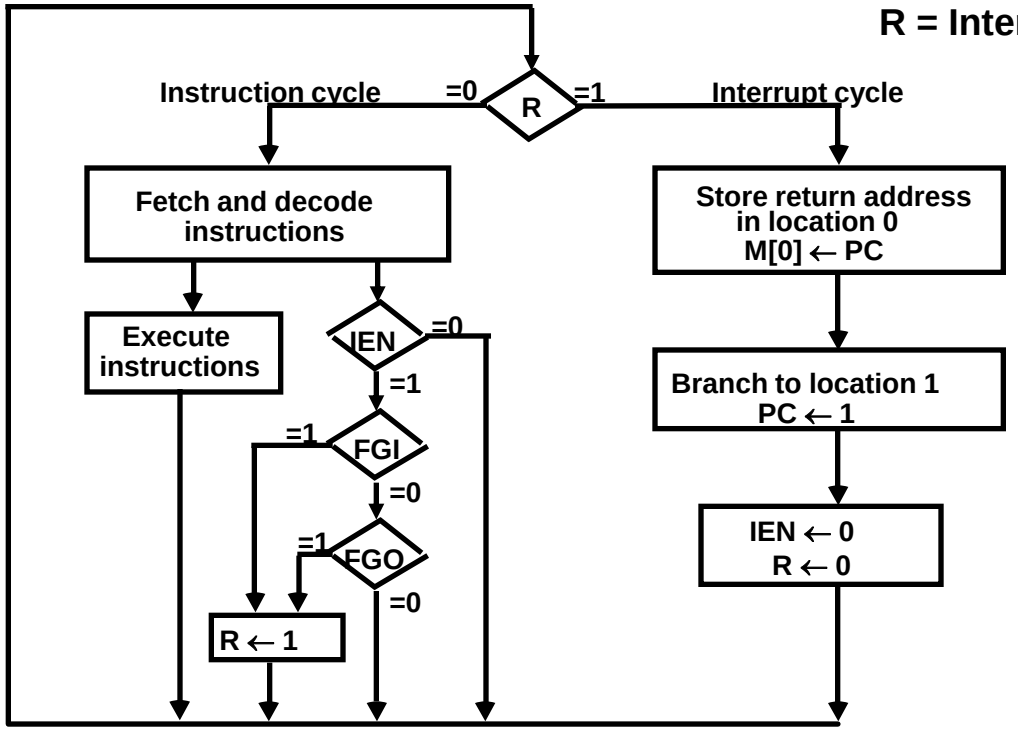
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

IEN (Interrupt-enable flip-flop)

- can be set and cleared by instructions
  - When cleared (IEN=0) the computer cannot be interrupted
  - When set (IEN=1) the computer can be interrupted

Flow Chart of Interrupt Cycle



INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- The interrupt cycle is a HW implementation of a branch and save return address operation.

- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.

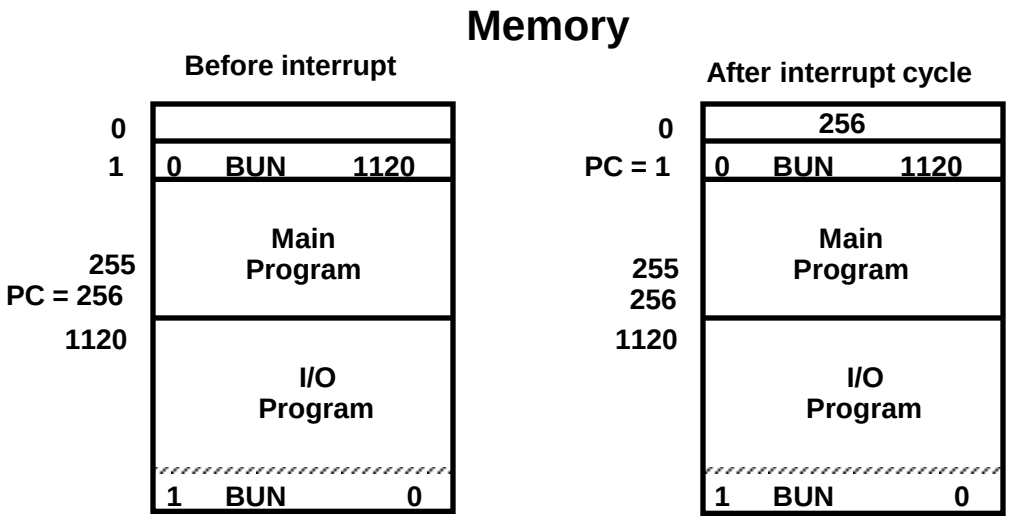
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine

- The instruction that returns the control to the original program is "indirect BUN 0"

Which of the following is not a flip flop ?

A) IEN
B) E
C) I
D) None of the above

Register Transfer Operations in Interrupt Cycle



Register Transfer Statements for Interrupt Cycle

- R F/F ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$

  ⇔ $T_0'T_1'T_2'$ (IEN)(FGI + FGO): R ← 1

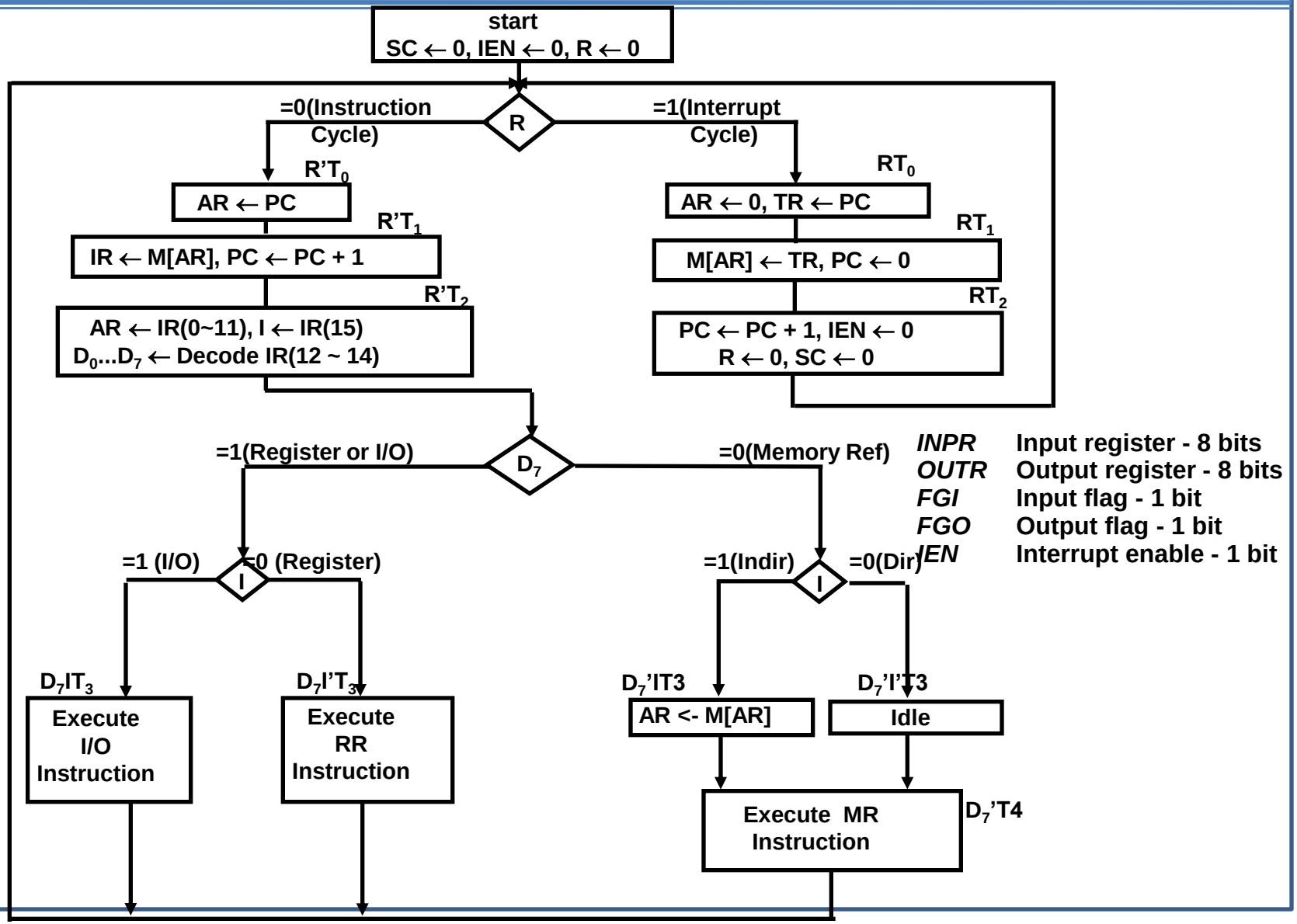
- The fetch and decode phases of the instruction cycle must be modified ➔Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle :

  RT_0: AR ← 0, TR ← PC

  RT_1: M[AR] ← TR, PC ← 0

  RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

Complete Computer Description



CSE211

Computer Organization and Design

- *General Register Organization*
- *Stack Organization*
- *Instruction Formats*
- *Addressing Modes*

Overview

- **General Register Organization**
- **Stack Organization**
- **Instruction Formats**
- **Addressing Modes**
- Data Transfer and Manipulation
- Program Control
- RISC and CISC

Major Components of CPU

- **Storage Components**
  - **Registers**
  - **Flags**
- **Execution (Processing) Components**
  - **Arithmetic Logic Unit(ALU)**
    - **Arithmetic calculations, Logical computations, Shifts/Rotates**
- **Transfer Components**
  - **Bus**
- **Control Components**
  - **Control Unit**

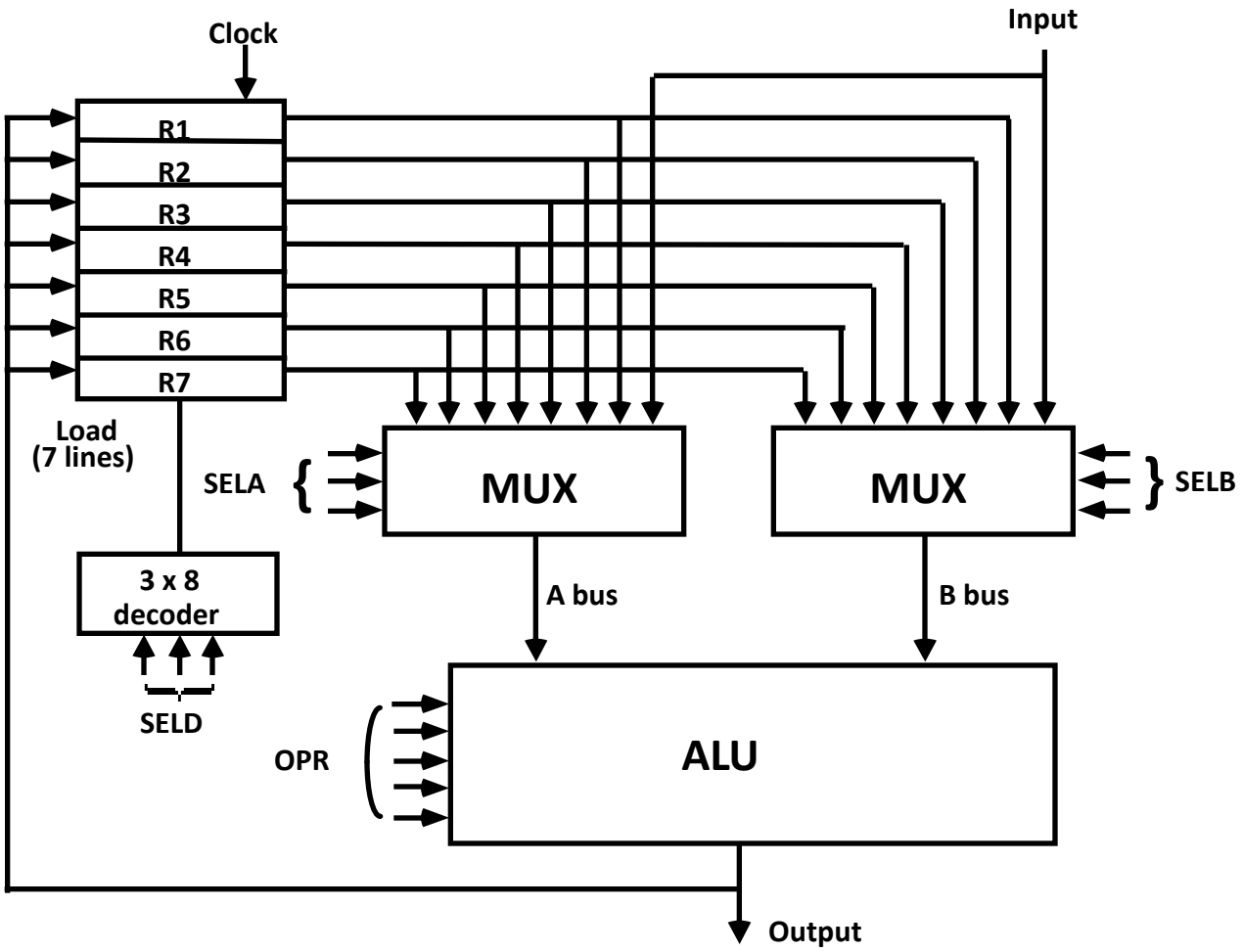
Register

- **In Basic Computer, there is only one general purpose register, the Accumulator (AC)**
- **In modern CPUs, there are many general purpose registers**
- **It is advantageous to have many registers**
  - **Transfer between registers within the processor are relatively fast**
  - **Going "off the processor" to access memory is much slower**

Processor Organization

- **In general, most processors are organized in one of 3 ways**
  - **Single register (Accumulator) organization**
    - **Basic Computer is a good example**
    - **Accumulator is the only general purpose register**
  - **General register organization**
    - **Used by most modern computer processors**
    - **Any of the registers can be used as the source or destination for computer operations**
  - **Stack organization**
    - **All operations are done using the hardware stack**
    - **For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack**

General Register Organization



Operation of ControlUnit

The control unit

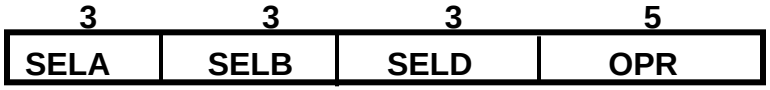
Directs the information flow through ALU by

- **Selecting various *Components* in the system**
- **Selecting the *Function* of ALU**

Example: R1 ← R2 + R3

[1] MUX A selector (SELA): BUS A ← R2
[2] MUX B selector (SELB): BUS B ← R3
[3] ALU operation selector (OPR): ALU to ADD
[4] Decoder destination selector (SELD): R1 ← Out Bus

Control Word



Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU Control

Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 – R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ← R1	R1	-	R7	TSFA	001 000 111 00000
Output ← R2	R2	-	None	TSFA	010 000 000 00000
Output ← Input	Input	-	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

In General Register Organization, OPR is of how many bits ?

A) 3

B) 4

C) 5

D) 6