USN | | | | | | | | | | |

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assesment Test I – Sept. 2017

Sub:	COMPUTER ORGANIZATION			Sub Code:	15CS34	Branch:	CSE,ISE
Date:	20 / 09 / 2017	Duration:	90 mins	Max Marks:	50	Sem / Sec:	3 (A,B,C)

OBE

Answer **FIVE** FULL questions selecting AT LEAST ONE question **FROM EACH PART** — MARKS | CO | RBT

PART A

1 (a) List the steps needed to execute the machine instruction — [10] CO1 L3

ADD A,B,R1

in terms of transfers between the components and some simple control commands. Assume that the instruction itself is stored in the memory at location INSTR and that this address is initially in register PC.

OR

2 (a) Assume a program with 1000 instructions. 25% of the instructions take 4 clock cycles, 40% of the instructions take 5 clock cycles, and the remaining instructions take 3 clock cycles to execute respectively. Calculate the time of execution assuming basic performance equation if the clock rate of the machine is 1GHz. — [4] CO3 L3

(b) Calculate the SPEC rating for the program suite under test. Running times of the program suite for reference PC and PC under test are given below: — [6] CO3 L3

Programs	Running time for reference PC	Running time for PC under test
P1	20	10
P2	100	50
P3	40	20
P4	10	5
P5	60	30

PART B

3 (a) Explain basic instruction types with example. — [6] CO1 L2

(b) Register R1 and R2 of a computer contains the decimal values 1000 and 2000 .What is the effective address of the memory operand in each of the following machine instructions. — [4] CO1 L3

a. Load 10(R1),R5
b. Store R5,50(R1,R2)
c. Add –(R2),R5
d. Subtract (R1)+,R5

OR

4 (a) Explain addressing modes with example of each mode. — [10] CO1 L2

PART C

5 (a) Explain the operation of stack with example. — [10] CO1 L2

OR

6 (a) Define Subroutine. Explain subroutine linkage using a link register. — [7] CO1 L2

(b) The subroutine call instruction of a computer saves the return address in a processor register called the Link register RL. What would you do to allow — [3] CO1 L3

	subroutine nesting? Would your scheme allow the subroutine to call itself?			
	PART D			
7 (a)	With a neat diagram, explain registers in DMA interface. Also, explain any one of the bus arbitration approaches.	[5+5]	CO2	L2
	OR			
8 (a)	Explain various methods for handling interrupts from multiple devices.	[10]	CO2	L2
	PART E			
9 (a)	With a neat diagram, explain I/O interface for an I/O device. Also, explain various registers involved in it.	[10]	CO2	L2
	OR			
10 (a)	What is an Interrupt? With an example, illustrate the concept of interrupt.	[10]	CO2	L2

USN

Internal Assesment Test I – Sept. 2017 – **Scheme of evaluation and solution**

Sub:	COMPUTER ORGANIZATION				Sub Code:	15CS34	Branch:	CSE,ISE
Date:	20 / 09 / 2017	Duration:	90 mins	Max Marks: 50	Sem / Sec:	3 (A,B,C)		OBE

Answer **FIVE** FULL questions selecting AT LEAST ONE question **FROM EACH PART** — MARKS — CO — RBT

PART A

1 (a) List the steps needed to execute the machine instruction [10] CO1 L3

ADD A,B,R1

in terms of transfers between the components and some simple control commands. Assume that the instruction itself is stored in the memory at location INSTR and that this address is initially in register PC.

[1 mark x 10 steps]

PC --> MAR
Read cycle, Memory --> MDR
MDR --> IR
Operand1 address in MAR
Read cycle, Memory --> MDR
Operand2 address in MAR
Read cycle, Memory --> MDR
Addition in ALU
Result stored in R1
PC+1

OR

2 (a) Assume a program with 1000 instructions. 25% of the instructions take 4 clock cycles, 40% of the instructions take 5 clock cycles, and the remaining instructions take 3 clock cycles to execute respectively. Calculate the time of execution assuming basic performance equation if the clock rate of the machine is 1GHz. [4] CO3 L3

$$\mathrm{T} = \frac{\overset{\text{1 mark}}{250\text{x}4} + \overset{\text{1 mark}}{400\text{x}5} + \overset{\text{1 mark}}{350\text{x}3}}{1\text{x}10^9}$$

1 mark

$=4.05\text{x}10^{6}$ seconds or 4.05 microseconds

(b) Calculate the SPEC rating for the program suite under test. Running times of the program suite for reference PC and PC under test are given below: [6] CO3 L3

Programs	Running time for reference PC	Running time for PC under test
P1	20	10
P2	100	50
P3	40	20
P4	10	5
P5	60	30

S_1=20/10=2 1 mark

S_2= 100/50=2 1 mark

S_3=40/20=2 1 mark

S_4=10/5=2 1 mark

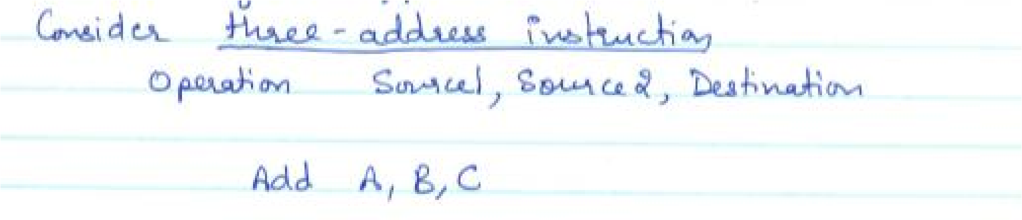
S_5=60/30=2 1 mark

Overall SPEC rating= $(2 \times 2 \times 2 \times 2 \times 2)^{1/5} = 2$ 1 mark

PART B

3 (a) Explain basic instruction types with example. [6] CO1 L2

(Definition=1 mark , Example= 1 mark)

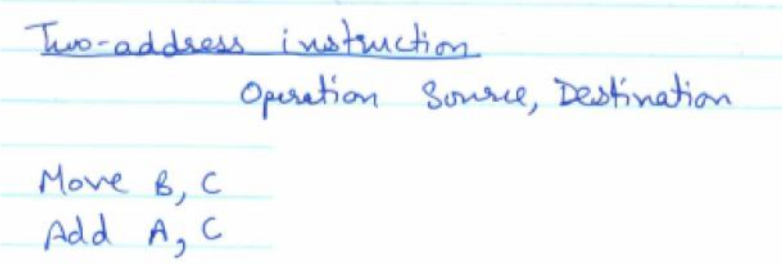


Consider three-address instruction

Operation Source1, Source2, Destination

Add A, B, C

(Definition=1 mark , Example= 1 mark)



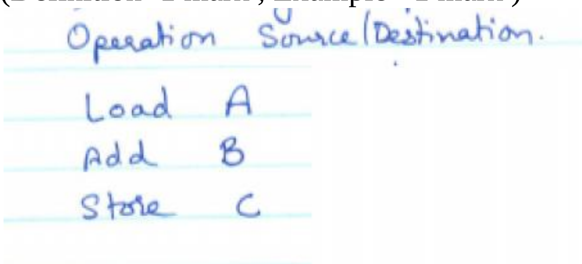
Two-address instruction

Operation Source, Destination

Move B, C

Add A, C

(Definition=1 mark , Example= 1 mark)



Operation Source/Destination.

Load A

Add B

Store C

(b) Register R1 and R2 of a computer contains the decimal values 1000 and 2000 .What is the effective address of the memory operand in each of the following machine instructions. [4] CO1 L3

a. Load 10(R1),R5
b. Store R5,50(R1,R2)
c. Add –(R2),R5
d. Subtract (R1)+,R5

1010 – 1 mark
3050 – 1 mark
1999 or 1996 – 1 mark
1000 – 1 mark

OR

4 (a) Explain addressing modes with example of each mode. [10] CO1 L2

(Index mode- 3 marks, other modes 1 mark each)

① Immediate mode

The operand is given explicitly in the instruction.

Assembler function - # Value

Addressing function - Operand = Value

Eg. Move #200, R0

Above instruction places value 200 in register R0.

Generally, this mode is used to represent constants.

② Register mode

The operand is the contents of a processor register; the name of the register is given in the instruction.

Assembler syntax - Ri

Addressing function - EA= Ri

EA means effective address of operand i.e, where operand is located.

Eg. - Move R1, R2.

Above instruction moves the contents of register R1 to R2.

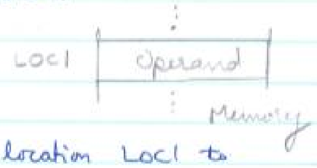
Generally, this mode is used to access variables.

③ Absolute (Direct) mode

The address of the memory location where operand is located is given explicitly in the instruction.

Assembler syntax - LOC

Addressing function - EA= LOC



Eg- Move LOC1, LOC2

Above instruction moves the operand at location LOC1 to location LOC2.

It is generally used for representing global variables.

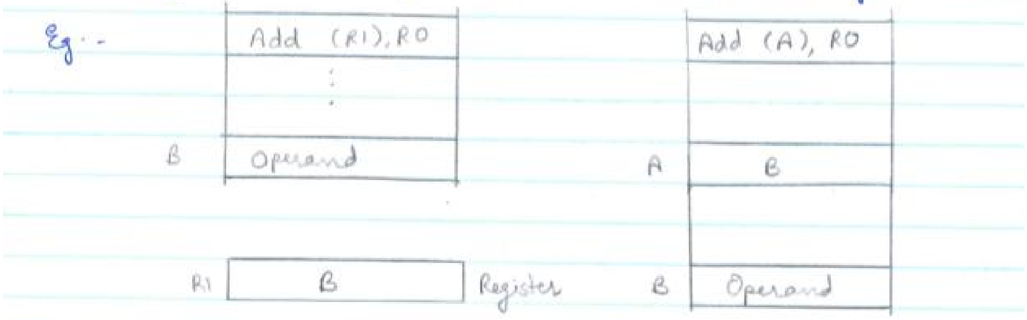
④ Indirect mode

The contents of register or memory location given in the instruction gives the effective address of the operand.

Assembler syntax - (Ri)
(LOC)

Addressing function - EA= [Ri]
EA = [LOC]

Eg. -



(a) Through a general purpose register (b) Through a memory location

Indirect addressing

③ Index mode

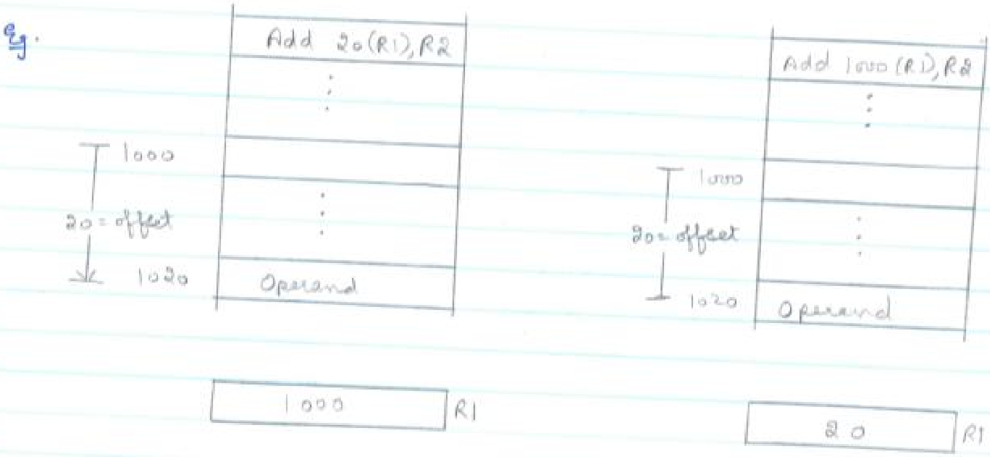
The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be a special register or a general - purpose register and is known as an index register.

Assembler syntax - $X(R_i)$

Addressing function - $EA = [R_i] + X$

X defines an offset (also called a displacement).

Eg.



(a) offset is given as a constant (b) offset is in the index register

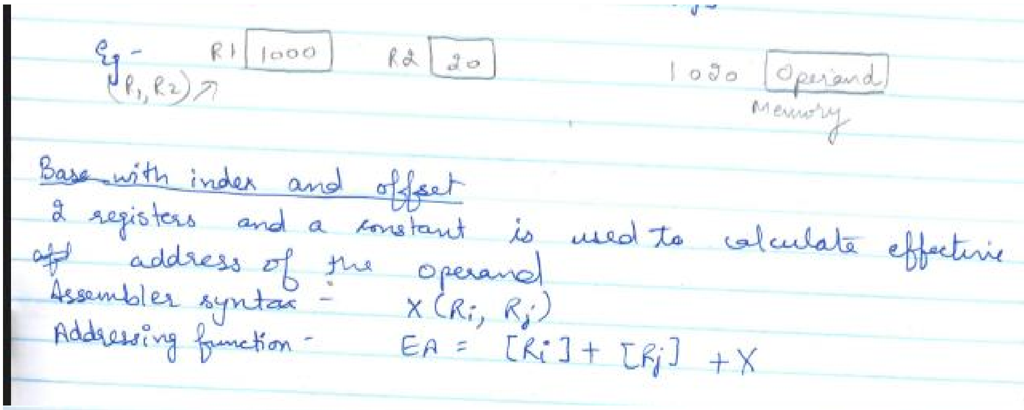
Indexed addressing

Base with index mode

The effective address is the sum of the contents of of multiple registers.

Assembler syntax - (R_i, R_j)

Addressing function - $EA = [R_i] + [R_j]$



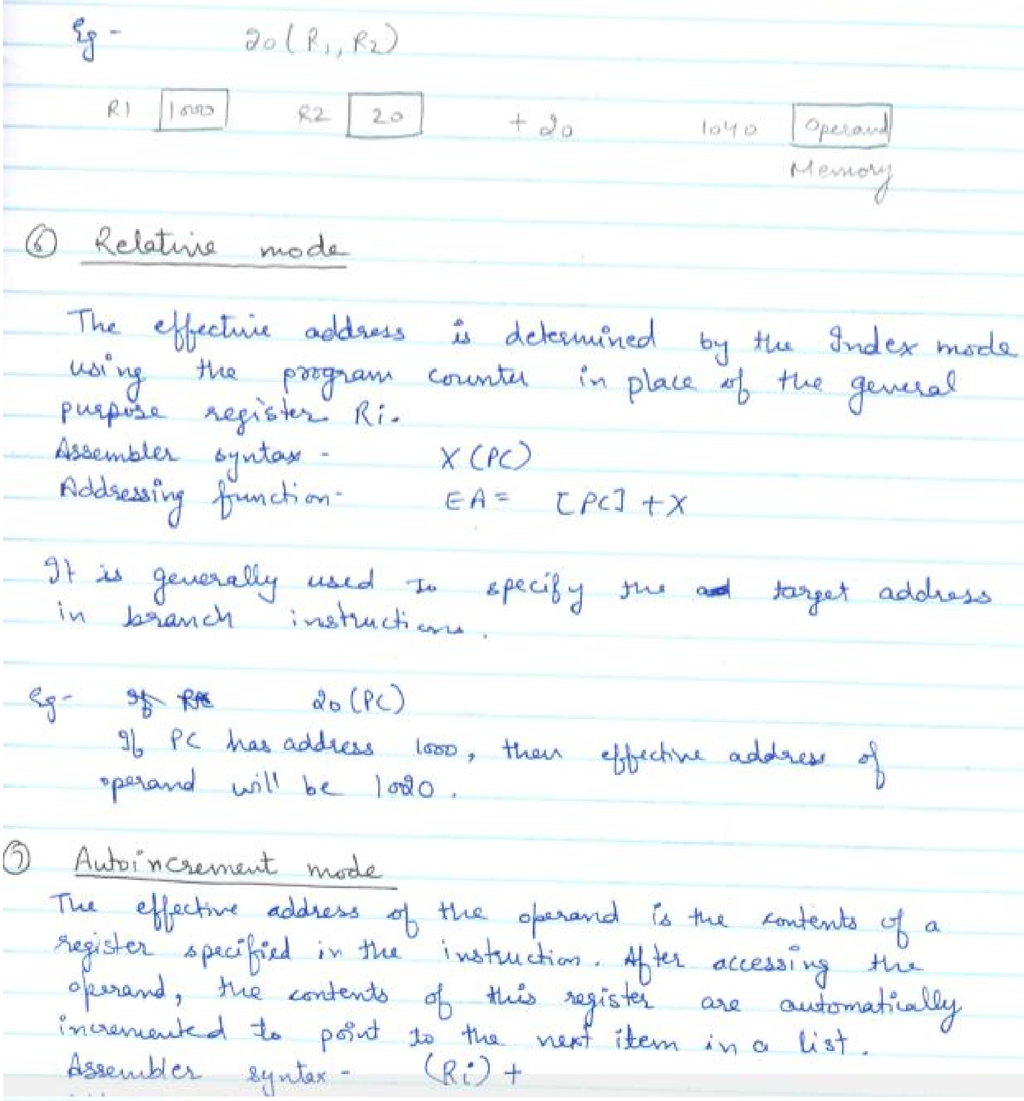
Eg - R1 $\boxed{1000}$ R2 $\boxed{20}$ 1020 $\boxed{\text{Operand}}$
(R1, R2) ↗ Memory

Base with index and offset

2 registers and a constant is used to calculate effective ~~apf~~ address of the operand

Assembler syntax - $X(R_i, R_j)$

Addressing function - $EA = [R_i] + [R_j] + X$



Eg - $20(R_1, R_2)$

R1 $\boxed{1000}$ R2 $\boxed{20}$ + 20 1040 $\boxed{\text{Operand}}$
Memory

⑥ Relative mode

The effective address is determined by the Index mode using the program counter in place of the general purpose register Ri.

Assembler syntax - $X(PC)$

Addressing function - $EA = [PC] + X$

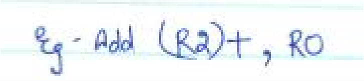
It is generally used to specify the ~~ad~~ target address in branch instructions.

Eg - ~~If R0~~ 20(PC)
If PC has address 1000, then effective address of operand will be 1020.

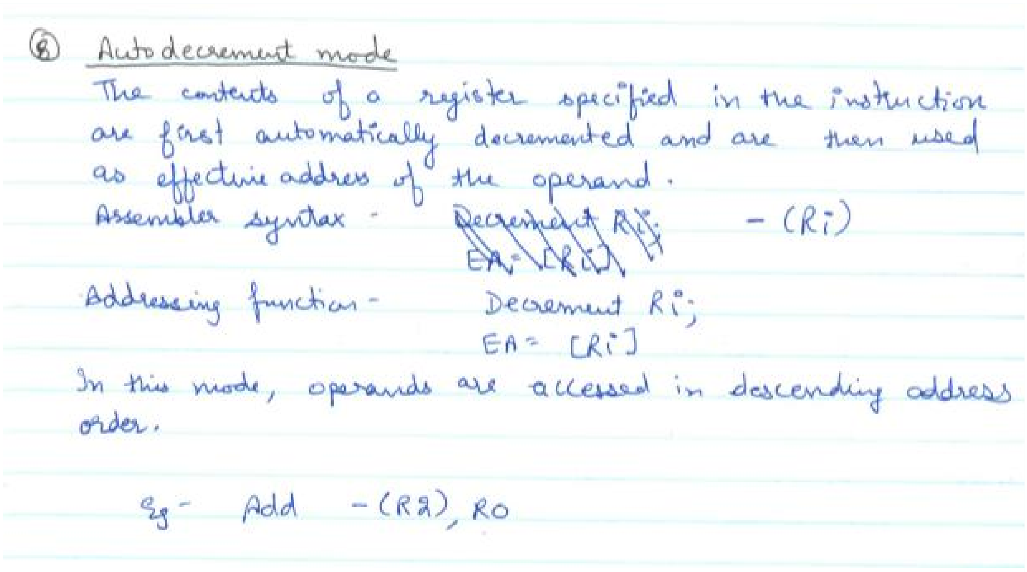
⑦ Autoincrement mode

The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.

Assembler syntax - $(R_i)+$



Eg - Add (R2)+, R0



(8) Autodecrement mode

The contents of a register specified in the instruction are first automatically decremented and are then used as effective address of the operand.

Assembler syntax - ~~Decrement Ri; EA=[Ri]~~ -(Ri)

Addressing function - Decrement Ri;
EA= [Ri]

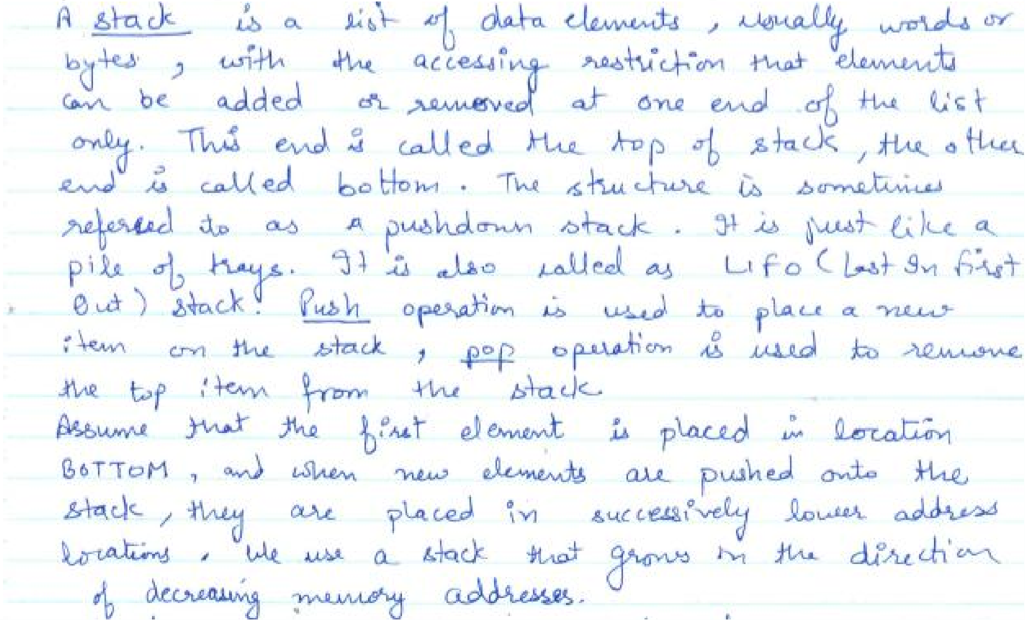
In this mode, operands are accessed in descending address order.

Eg - Add -(R2), R0

PART C

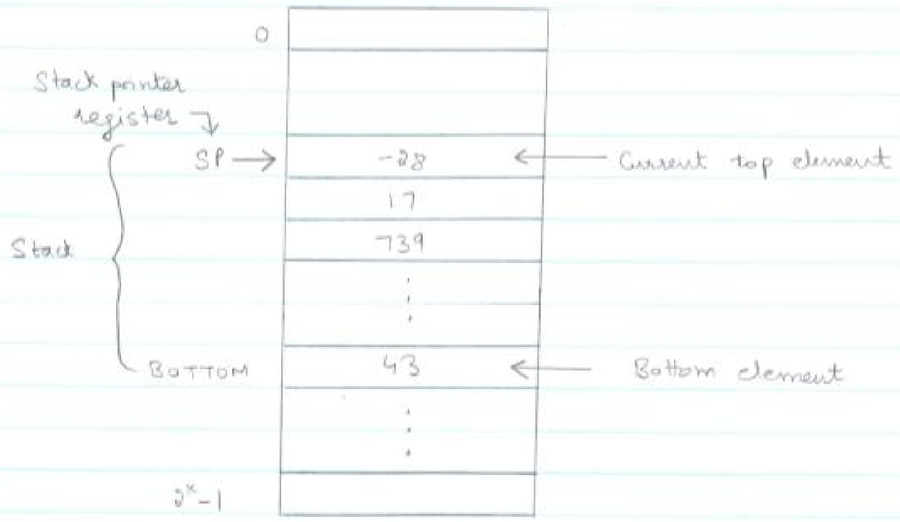
5 (a) Explain the operation of stack with example. [10] CO1 L2

(Diagram and explanation – 8 marks, SafePush, SafePop- 2 marks)



A stack is a list of data elements, usually words or bytes, with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of stack, the other end is called bottom. The structure is sometimes referred to as a pushdown stack. It is just like a pile of trays. It is also called as LIFO (Last In first Out) stack. Push operation is used to place a new item on the stack, pop operation is used to remove the top item from the stack.

Assume that the first element is placed in location BOTTOM, and when new elements are pushed onto the stack, they are placed in successively lower address locations. We use a stack that grows in the direction of decreasing memory addresses.

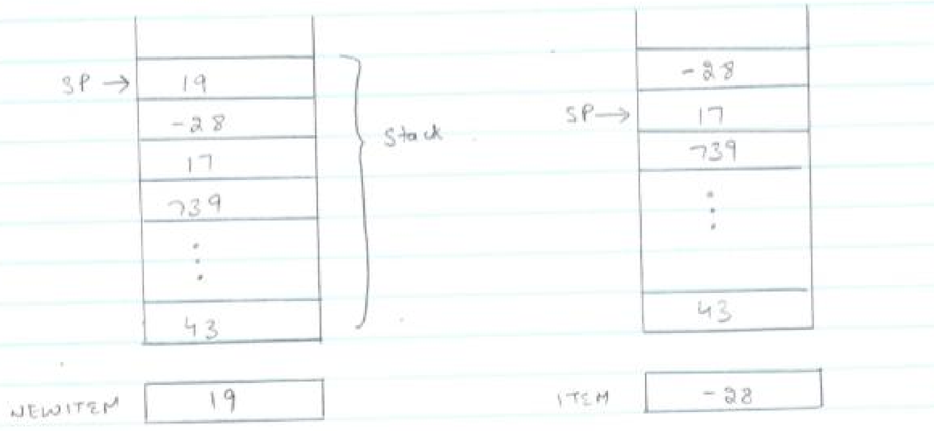


A Stack of words in the memory

Assume a byte-addressable memory with a 32-bit word length.

```
Push :      Subtract   #4, SP
            Move       NEWITEM, (SP)

Pop :       Move       (SP), ITEM
            Add        #4, SP
```

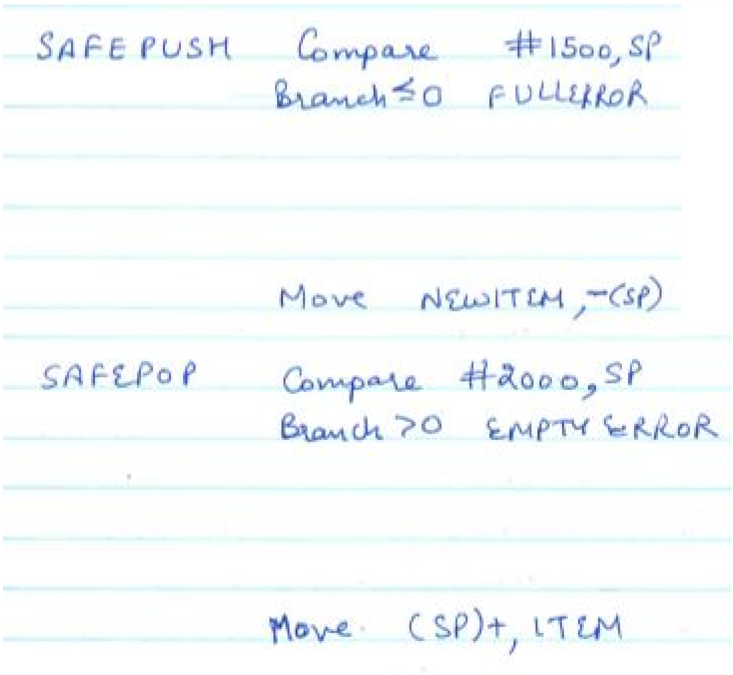
```
Push :-     Move   NEWITEM, -(SP)

Pop :-      Move   (SP)+, ITEM
```



a) After push from NEWITEM

b) After pop into ITEM

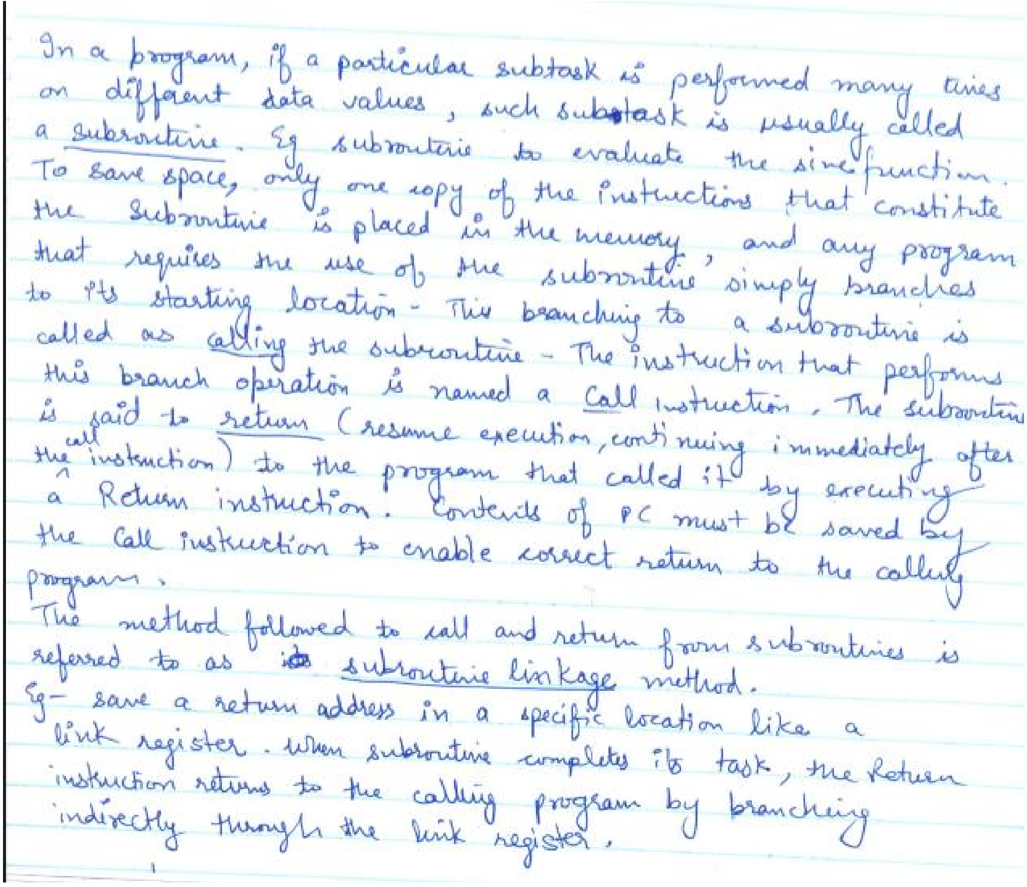
* Consider stack siz from $\xrightarrow{1500 \text{ to}}$ 2000 memory location addresses.



```
SAFEPUSH   Compare     #1500,SP
           Branch≤0    FULLERROR

           Move        NEWITEM,-(SP)
SAFEPOP    Compare     #2000,SP
           Branch>0    EMPTYERROR

           Move        (SP)+,ITEM
```

OR

6 (a)	Define Subroutine. Explain subroutine linkage using a link register. (Definition- 1 mark, Diagram- 3 marks, Explanation and example- 3 marks)	[7]	CO1	L2



In a program, if a particular subtask is performed many times on different data values, such subtask is usually called a subroutine. Eg subroutine to evaluate the sine function. To save space, only one copy of the instructions that constitute the subroutine is placed in the memory, and any program that requires the use of the subroutine simply branches to its starting location. This branching to a subroutine is called as calling the subroutine. The instruction that performs this branch operation is named a Call instruction. The subroutine is said to return (resume execution, continuing immediately after the call instruction) to the program that called it by executing a Return instruction. Contents of PC must be saved by the Call instruction to enable correct return to the calling program.

The method followed to call and return from subroutines is referred to as ~~its~~ subroutine linkage method.

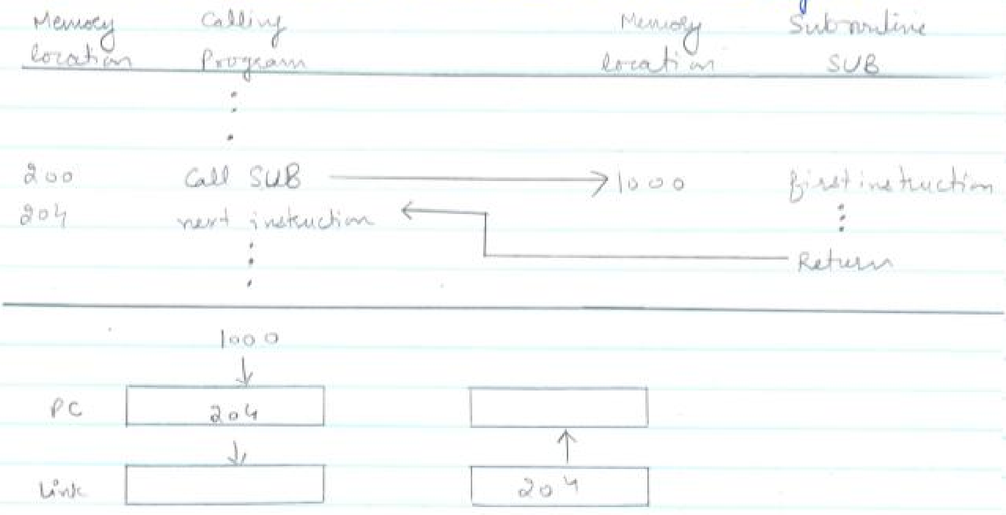
Eg- save a return address in a specific location like a link register. When subroutine completes its task, the Return instruction returns to the calling program by branching indirectly through the link register.

Call instruction

- Store contents of PC in link register
- Branch to the target address specied by the instruction.

Return instruction

- Branch to the address contained in the link register.



Subroutine linkage using a link register

(b) The subroutine call instruction of a computer saves the return address in a processor register called the Link register RL. What would you do to allow subroutine nesting? [3] CO1 L3

Would your scheme allow the subroutine to call itself?

(Scheme chosen- 1.5 marks, Recursion allowed or not- 1.5 marks)

Store the contents of Link Register in Memory. No, it does not support recursion.

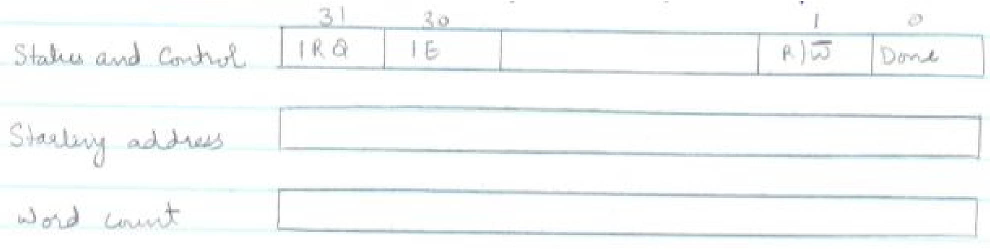
Or

Store the contents of Link Register in Stack. Yes, it supports recursion.

PART D

7 (a) With a neat diagram, explain registers in DMA interface. Also, explain any one of the bus arbitration approaches. [5+5] CO2 L2

(Registers diagram and its explanation- 5 marks, Any one of the bus arbitration techniques explanation – 5 marks)



Registers in a DMA interface

Bits & Flags	1	0
R/W	READ	WRITE
Done	Data transfer finishes	
IRQ	Interrupt request	
IE	Raise interrupt (enable) after Data Transfer.	

- <u>Centralized Bus Arbitration Technique</u>

→ A 'single bus arbiter' performs the required arbitration.

→ Bus arbiter may be the processor or a separate device connected to the bus (eg. DMA controller having highest priority).

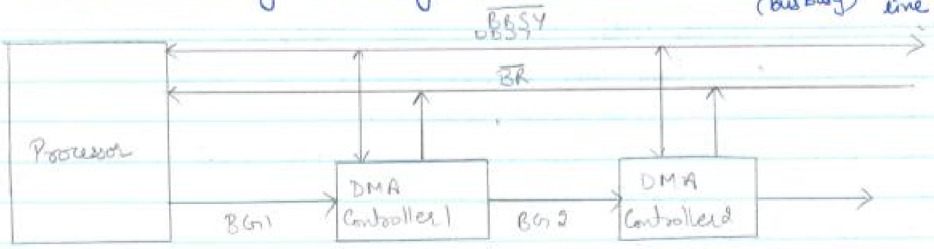
→ Initially the processor will act as 'bus master'.

→ Whenever a DMA controller generates a request ($\overline{BR}$), by ~~setting~~ activating <u>Bus Request</u> line, processor activates <u>Bus-Grant</u> (BG1) signal indicating DMA controller can become a master

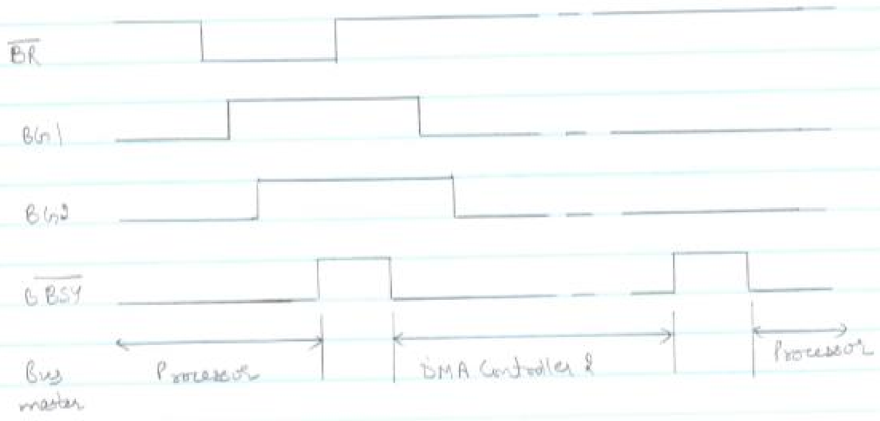
→ BG ~~signal~~ line is connected to all DMA devices using a daisy chain link.

→ If DMA controller 1 has enabled the request, it blocks the signal (BG1) and acts as master for bus arbitration, thereby disabling the bus for other device use.

→ If DMA controller 1 has not enabled bus arbitration-request, It simply forwards BG1 signal to its <u>downstream neighbour</u> DMA controllers by asserting BG2. New bus master activates $\overline{BBSY}$ (Bus Busy) line.



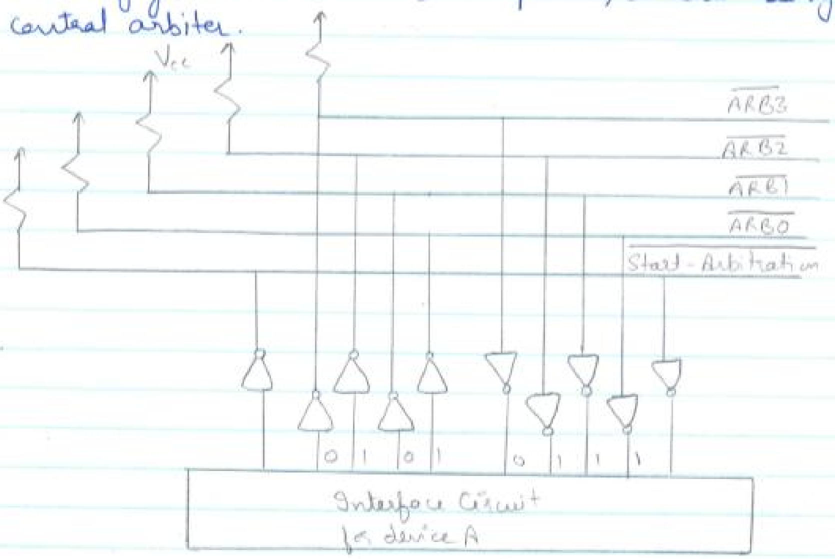
Bus arbitration using Daisy chain



Sequence of signals for Transfer of bus mastership from processor to DMA controller2

Distributed Bus Arbitration

- All devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.



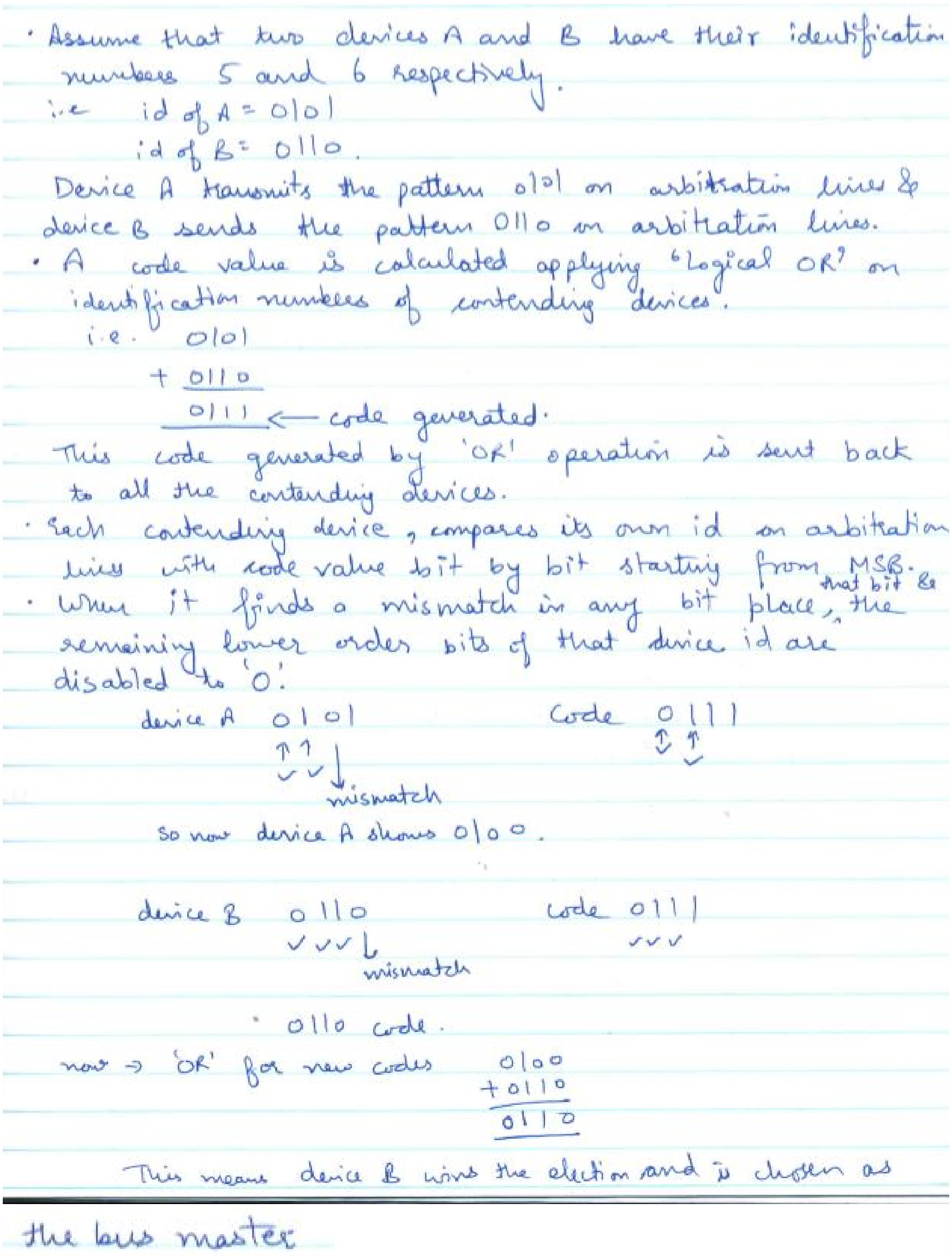
Distributed arbitration scheme

→ Each device is identified by using a 4bit identification number.

→ Devices start contending for bus by enabling "start-arbitration signal and place their 4-bit identification ~~number~~ on the bus (4 lines $ARB_0 - ARB_3$).

→ Device having ~~the~~ highest identification number is selected to get the granted service.

→ Selection procedure :



- Assume that two devices A and B have their identification numbers 5 and 6 respectively.

i.e id of A = 0101

id of B = 0110.

Device A transmits the pattern 0101 on arbitration lines & device B sends the pattern 0110 on arbitration lines.

- A code value is calculated applying 'Logical OR' on identification numbers of contending devices.

i.e.

$$\begin{array}{r} 0101 \\ +\ 0110 \\ \hline 0111 \end{array} \leftarrow \text{code generated.}$$

This code generated by 'OR' operation is sent back to all the contending devices.

- Each contending device, compares its own id on arbitration lines with code value bit by bit starting from MSB.
- When it finds a mismatch in any bit place, that bit & the remaining lower order bits of that device id are disabled to '0'.

device A 0101 Code 0111

✓✓ mismatch

So now device A shows 0100.

device B 0110 code 0111

✓✓✓ mismatch ✓✓✓

0110 code.

now → 'OR' for new codes

$$\begin{array}{r} 0100 \\ +0110 \\ \hline 0110 \end{array}$$

This means device B wins the election and is chosen as the bus master

OR

8 (a)	Explain various methods for handling interrupts from multiple devices. (Polling explanation - 2 marks, Vectored interrupts explanation - 2 marks, Interrupt Nesting explanation with diagram - 3 marks, Daisy Chaining explanation with diagram - 3 marks)	[10]	CO2	L2

① POLLING

- The IRQ (interrupt request) bit in the status register of device is set ~~to~~ when a device is requesting an interrupt.
- The Interrupt service routine polls the I/O devices connected to the bus.
- The first device encountered with the IRQ bit set is serviced and ISR is invoked.
- It is easy to implement, but too much time is spent on checking the IRQ of all devices, though some devices may not be requesting service.

② VECTORED INTERRUPTS

- Device requesting an interrupt identifies itself directly to the processor.

The device sends a special code (4 to 8 bits) to the processor over the bus.

- The code contains:
  identification of the device,
  starting address of ISR,
  address of the branch to ISR (if ISR not at that location).

The location pointed to by the interrupting device is used to store the starting address of the interrupt service routine. This address is called interrupt vector. Processor reads it and loads it into PC.

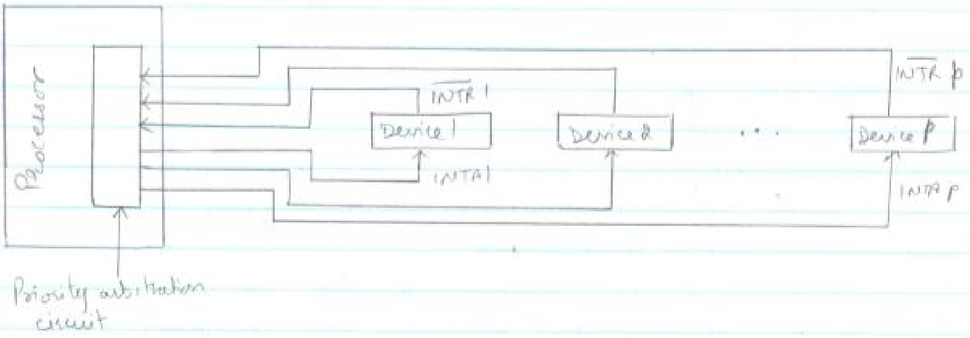
* When the processor is ready to receive interrupt-vector code, it may activate interrupt-acknowledge line, INTA. The I/O device responds by sending its interrupt-vector code and turning off the INTR signal.

③ INTERRUPT NESTING

- Disabling Interrupts during execution of the ISR may not favor devices which need immediate attention. eg, keeping track of time of day.
- Pre-emption of low priority interrupt by another higher priority interrupt is known as Interrupt nesting.
- Only interrupts requests of higher priority will be accepted during execution of ISR of lower priority interrupt.
- A priority level is assigned to processor which is the priority of the program that is currently being executed. Only higher priority interrupts than this are accepted.
- Processor's priority is encoded in a few bits of processor

status word which can be changed by program instruction called privileged instructions, which can be executed only while processor is running in supervisor mode (i.e. when executing OS routines).

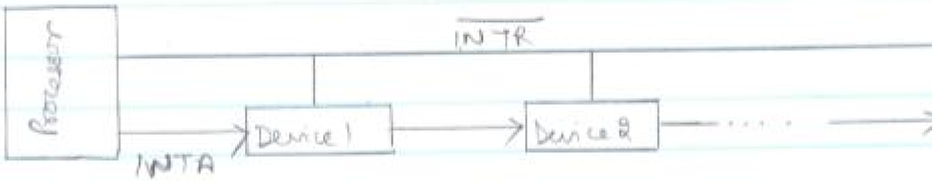
- An attempt to execute a privileged instruction while in user mode leads to a special type of interrupt called a privilege exception.
- A multiple-priority scheme can be implemented by using separate interrupt request and interrupt-acknowledge lines for each device. Each interrupt-request lines is assigned a different priority level. Interrupt requests received over these lines are sent to a priority arbri arbitration circuit in the processor. A request is accepted only if it has a higher priority level than that currently assigned to the processor.



Implementation of interrupt priority using individual interrupt-request and acknowledge lines

④ DAISY CHAINING

- The interrupt request line INTR is common
  the devices.
- The interrupt acknowledgement line INTA is c
  devices in a daisy chain way.
- INTA propagates serially through the devices
- Device that is electrically closest to the
  gets high priority.
- Low priority device may have a danger of

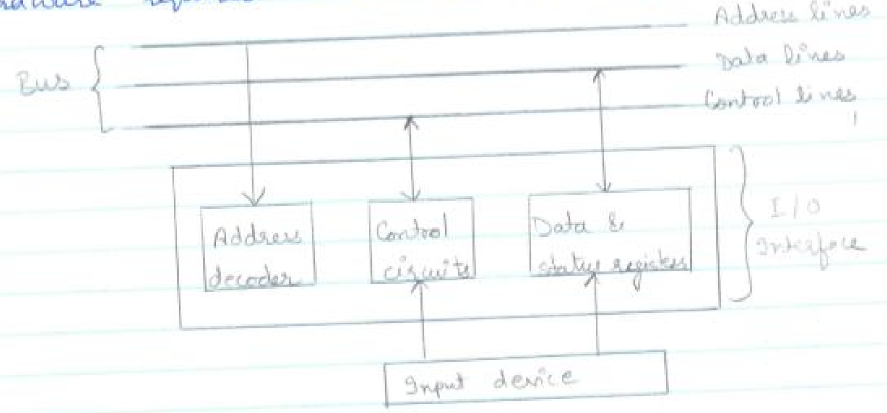


PART E

9 (a) With a neat diagram, explain I/O interface for an I/O device. Also, explain various registers involved in it. [10] CO2 L2

(I/O interface diagram- 2marks, interface explanation- 3 marks, Registers diagram- 2 marks, registers explanation- 3 marks)

Hardware required to connect an I/O device to the bus.



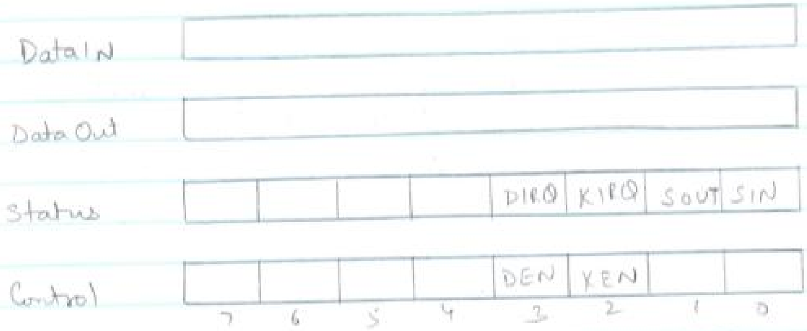
I/O interface for an input device

Address decoder - Enables the device to recognize its address when this address appears on address lines

Data register - Holds the data being transferred to or from the processor.

Status register - Contains information relevant to the operation of the I/O device.

The address decoder, the data & status registers and control circuitry required to coordinate I/O transfers constitute the device's interface circuit



Registers in keyboard & display interfaces

Registers: DATAIN, DATAOUT, STATUS, CONTROL

Flags :- SIN, SOUT - provide status information for keyboard & display unit.

KIRQ, DIRQ - Keyboard, Display Interrupt bits

DEN, KEN - Keyboard, Display Enable bits

OR

10 (a) What is an Interrupt? With an example, illustrate the concept of interrupt. [10] CO2 L2

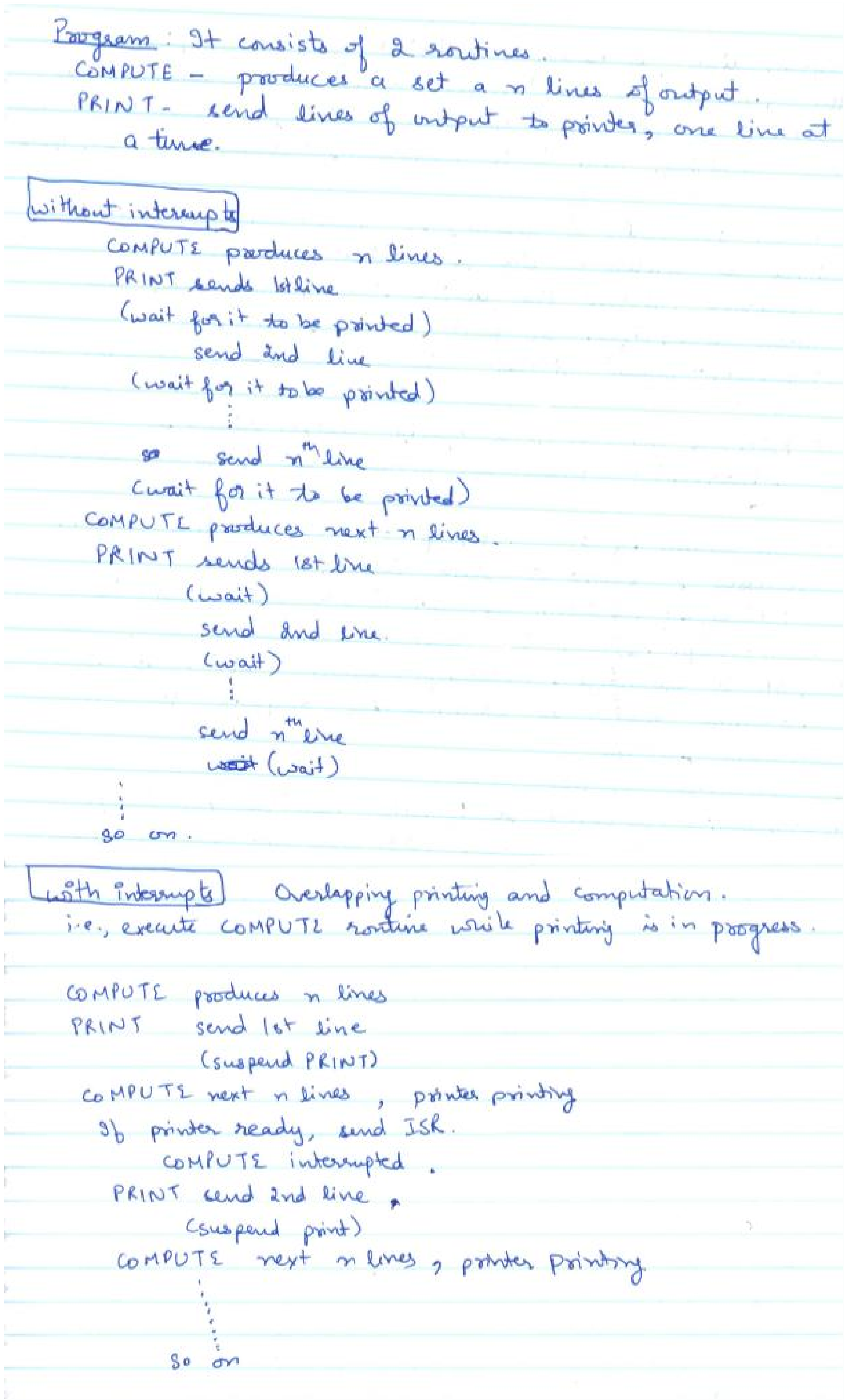
(Definition- 1 marks, Explanation using example or a program- 9 marks)

In program-controlled I/O processor repeatedly tests the device status. During this wait loop, processor is not performing any useful computation. There are many situations where other tasks can be performed while waiting for the I/O device to become ready. I/O device may alert the processor when it becomes ready. This alert may be sent using a hardware signal called an interrupt to the processor. Generally, at least one of the bus control lines, called an interrupt

request line, is usually dedicated for this purpose. Since the processor is no longer required to continuously check the status of external devices, it can use the waiting period to perform other useful functions

- When I/O device is ready, it sends the INTERRUPT signal to processor via a dedicated control line.
- Processor informs the device that its request has been recognized so that it may remove its interrupt-request signal. This can be done in two ways:
  - → Processor may send a special control signal (interrupt-acknowledge signal) on bus to device.

    or
  - → Instruction in the interrupt-service routine (ISR) accesses a status or data register in the device interface; implicitly informing the device that its interrupt request has been recognized.
- ISR is executed. Interrupt-service routine is the routine which is executed in response to an interrupt request.



Program : It consists of 2 routines.

COMPUTE – produces a set a n lines of output.

PRINT- send lines of output to printer, one line at a time.

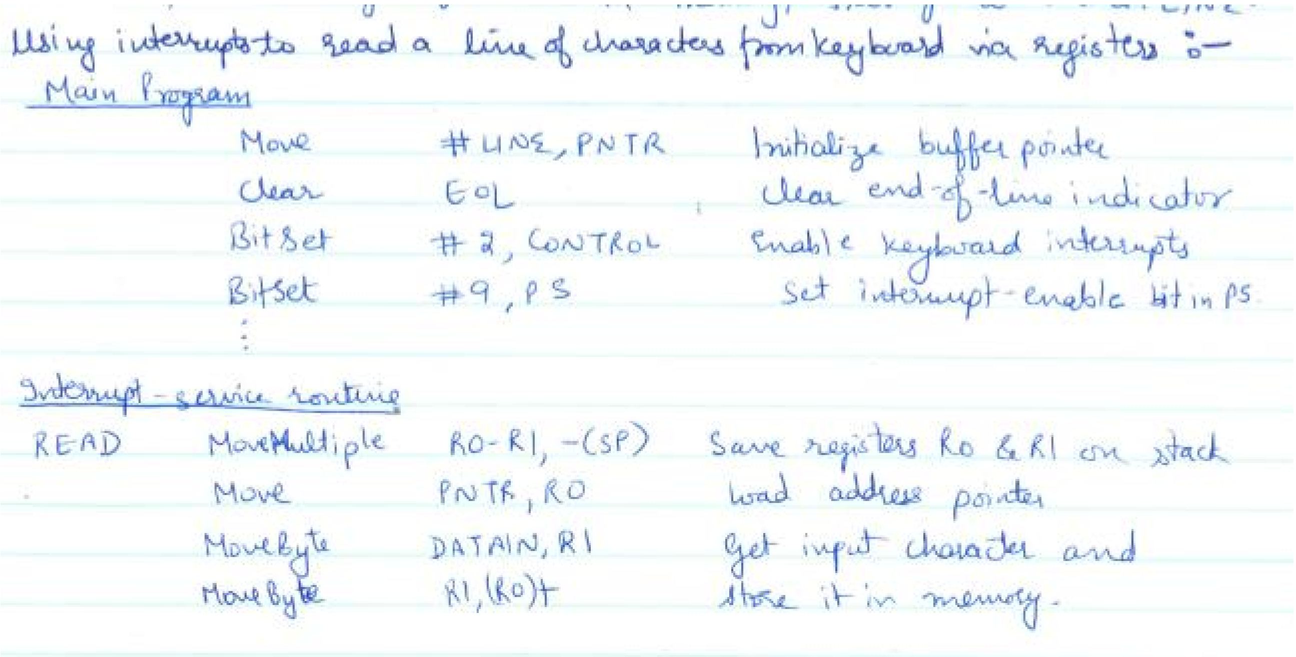
without interrupts

```
COMPUTE produces n lines.
PRINT sends 1st line
  (wait for it to be printed)
        send 2nd line
  (wait for it to be printed)
           ⋮
  ~~so~~   send $n^{th}$ line
  (wait for it to be printed)
COMPUTE produces next n lines.
PRINT sends 1st line
        (wait)
        send 2nd line.
        (wait)
           ⋮
        send $n^{th}$ line
        ~~wait~~ (wait)
  ⋮
so on.
```

with interrupts Overlapping printing and computation.

i.e., execute COMPUTE routine while printing is in progress.

```
COMPUTE produces n lines
PRINT   send 1st line
        (suspend PRINT)
COMPUTE next n lines , printer printing
 If printer ready, send ISR.
      COMPUTE interrupted .
  PRINT send 2nd line .
        (suspend print)
  COMPUTE  next n lines , printer printing
             ⋮
        so on
```



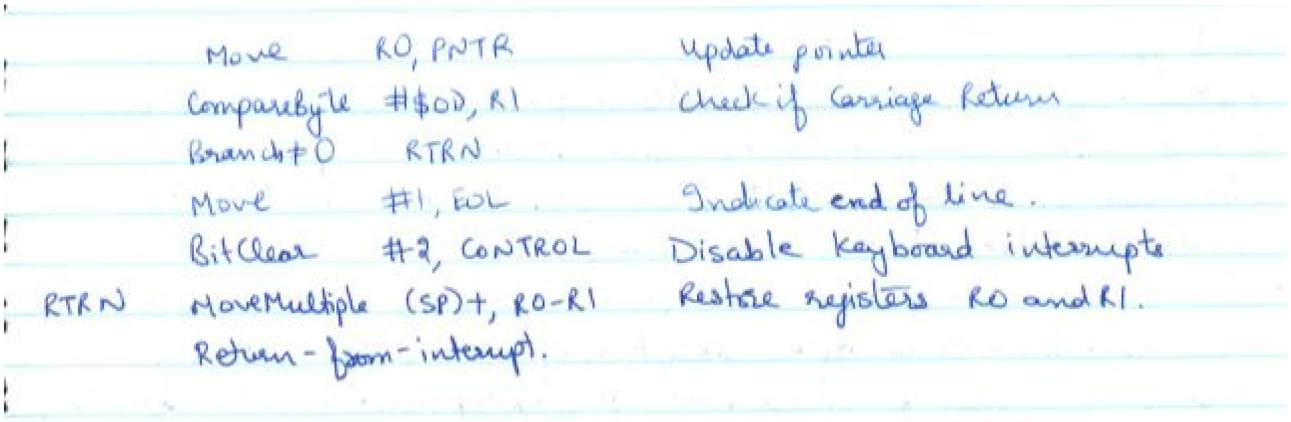
Using interrupts to read a line of characters from keyboard via registers :-

Main Program

```
            Move            #LINE, PNTR        Initialize buffer pointer
            Clear           EOL                Clear end-of-line indicator
            BitSet          #2, CONTROL        Enable keyboard interrupts
            BitSet          #9, PS             Set interrupt-enable bit in PS
              ⋮
```

Interrupt-service routine

```
READ        MoveMultiple    R0-R1, -(SP)       Save registers R0 & R1 on stack
            Move            PNTR, R0           Load address pointer
            MoveByte        DATAIN, R1         Get input character and
            MoveByte        R1, (R0)+          store it in memory.
```



```
            Move            R0, PNTR           Update pointer
            CompareByte     #$0D, R1           Check if Carriage Return
            Branch≠0        RTRN
            Move            #1, EOL            Indicate end of line.
            BitClear        #2, CONTROL        Disable keyboard interrupts
RTRN        MoveMultiple    (SP)+, R0-R1       Restore registers R0 and R1.
            Return-from-interrupt.
```