CMR
INSTITUTE OF
TECHNOLOGY

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

Internal Assesment Solution- for IAT II

Sub:	Computer Organization and Architecture							Code:	18EC35
Date:	15/10/ 2019	Duration:	90 mins	Max Marks:	50	Sem:	3rd	Branch:	ECE
Answer Any FIVE FULL Questions									

1 Explain any five shift and rotate instructions with relavent diagrams and examples [10]

Shift and Rotate Instructions

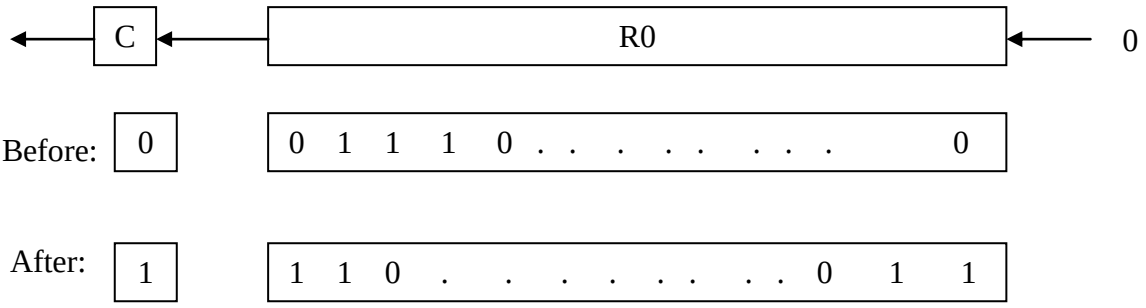
There are applications that require bits of an operand to be shifted to the right or left some specified number of bit positions. For general operands we use a logical shift. For a number we use an arithmetic shift which preserves the sign of the number.

Logical Shifts

Two logical shift instructions are needed, one for shifting left (LShiftL) and another for shifting right (LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction. The general form of logical left shift instruction is

LShiftL count, dst

The count operand may be given as an immediate operand or it may be contained in the processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag C, and then dropped. Involving the C flag in shifts is useful in arithmetic operations on large numbers that occupy more than one word. Fig 2.10 illustrates all the shift operations.



(a) Logical shift Left LShiftL #2, R0

2ª) Explain the data structure stack with the help of suitable diagrams. Provide instructions used to manipulate stack.

b) Given that register R5 is used to point to the top of the stack. Write sequence of instructions using auto increment/ auto decrement addressing modes to perform the operation

(i) Remove the top item from stack

(ii) Insert a new item to the stack

(i) Remove the top item from stack

Move (R5)+, R4

(ii) Insert a new item to the stack

Move R4, -(R5)

3) Explain DMA controller with the help f suitable diagrams. Explain the registers involved in DMA operation

To transfer large blocks of data at high speeds, an alternate approach is used. A special control unit may be provided to allow transfer of block of data directly between external device and main memory without intervention by processor. This approach is called direct memory access or DMA.

DMA transfers are performed by control circuits that are part of I/O interface called DMA controller. The DMA controller performs functions that would normally be carried out by processor when accessing main memory.

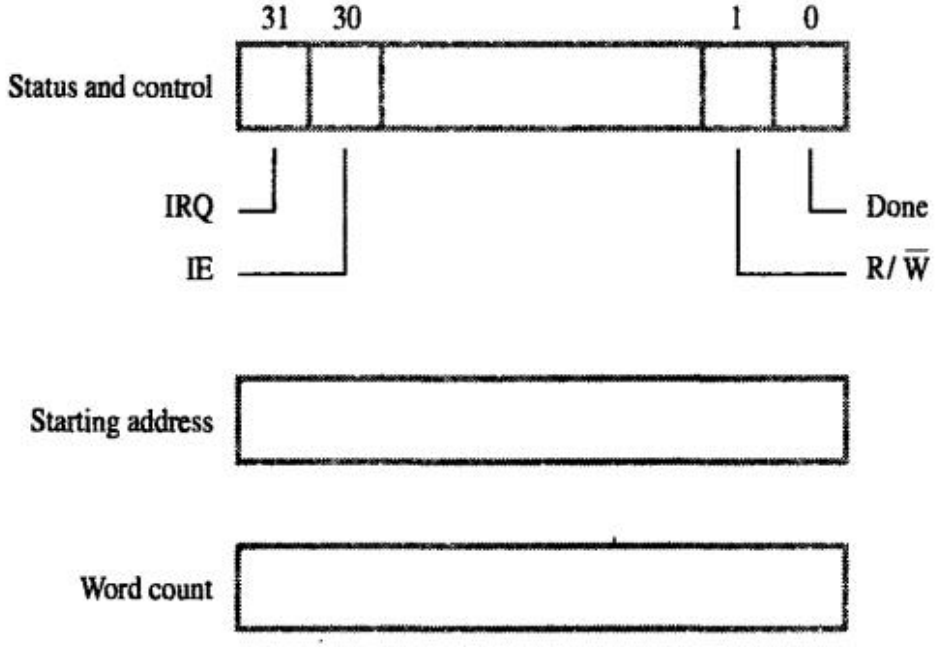


Fig 9: Registers in DMA interface

The R/$\overline{W}$ bit determine the direction of transfer. When this bit is set to 1 by a program instruction, the controller performs read operation that is it transfers data from memory to I/O device. When transfer is complete, it sets done flag to 1. When IE is1, it causes the controller to raise an interrupt after it has completed transferring block of data. Finally IRQ bit is set to 1 when it has requested interrupt.

Requests from DMA devices are given high priority than processor requests. Among different DMA devices high priority is given to high speed peripherals such as disks, high speed network interface or graphic display device.

The processor originates most memory cycles, the DMA controller is said to steal memory cycles from processor. This technique is called cycle stealing. DMA controller is given access to main memory to transfer a block of data without interruption. This is called as block or burst mode.

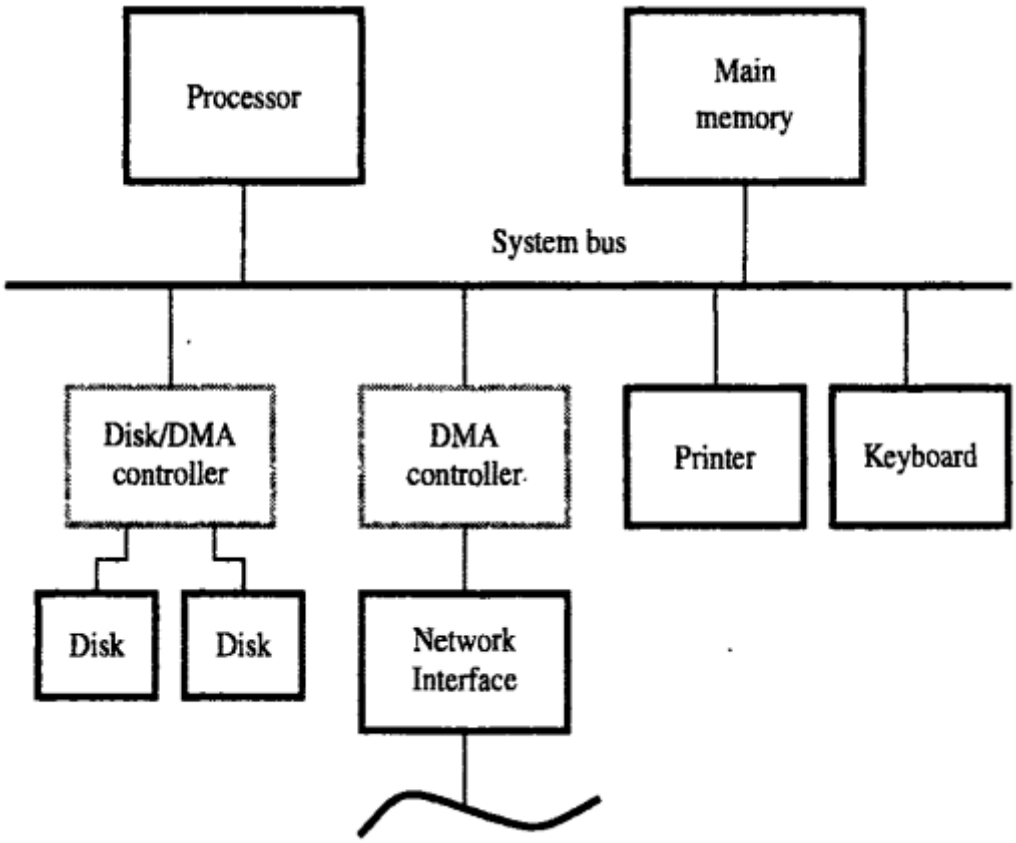


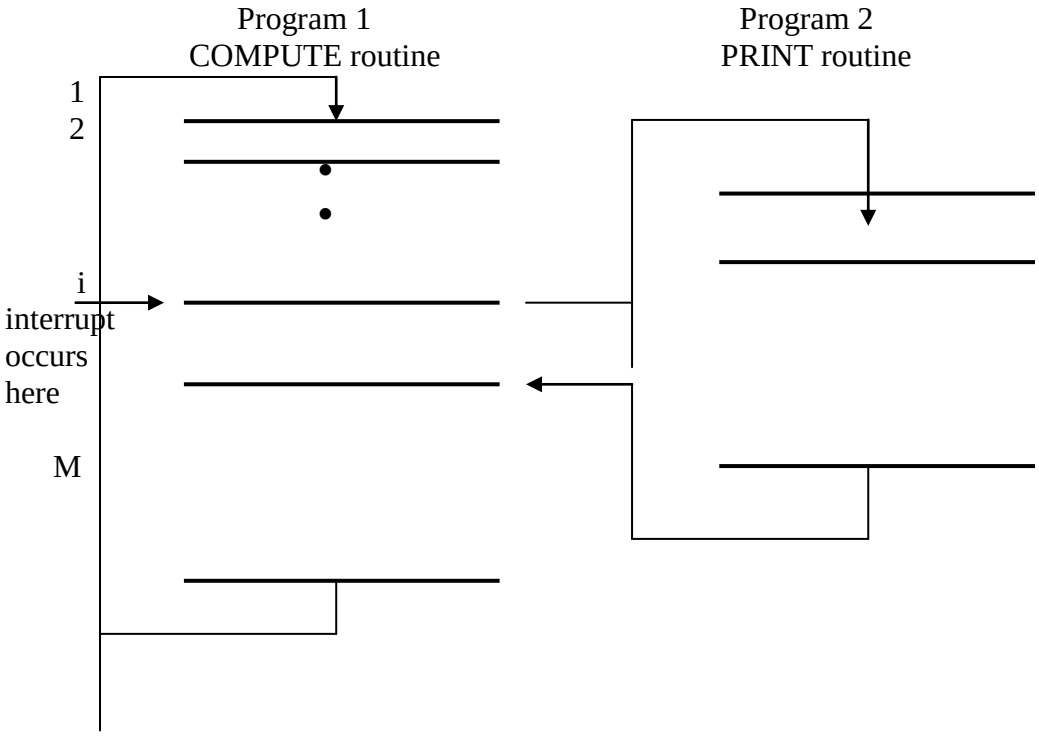
Fig 10: Use of DMA controllers in computer system

4) a) What is an Interrupt. Illustrate the concept with example

Interrupts

The other tasks can be performed by the processor while waiting for the I/O device to become ready. When the I/O device becomes ready, it sends a hardware signal called interrupt to the processor. Using the interrupts waiting periods can be eliminated.

Consider a task that requires some computations to be performed and the results to be printed on a line printer. This is followed by more computations and output and so on. Let the program consists of two routines COMPUTE and PRINT. Assume COMPUTES produces a set of ‘n’ lines of output to be printed by PRINT routine.



It is possible to overlap printing and computation ie to execute COMPUTE routine while printing is in progress, a faster overlap speed of execution will result. Whenever printer becomes ready, it alerts the processor by sending a interrupt request signal. In response the processor interrupts the COMPUTE routine and transfers the control to the PRINT routine. This process continues until all 'n' lines are printed and PRINT routine ends.

If COMPUTE takes longer to generate 'n' lines than the time required to print them, then the processor will be performing useful computations all the time.

Saving registers also increases the delay between the time the interrupt request is received and the start of execution of interrupt service routine. This delay is called interrupt latency.

b)With a neat diagram explain interrupt hardware

Interrupt Hardware

I/O device requests an interrupts by activating a bus line called interrupt request. A single interrupt may be used to serve 'n' devices.

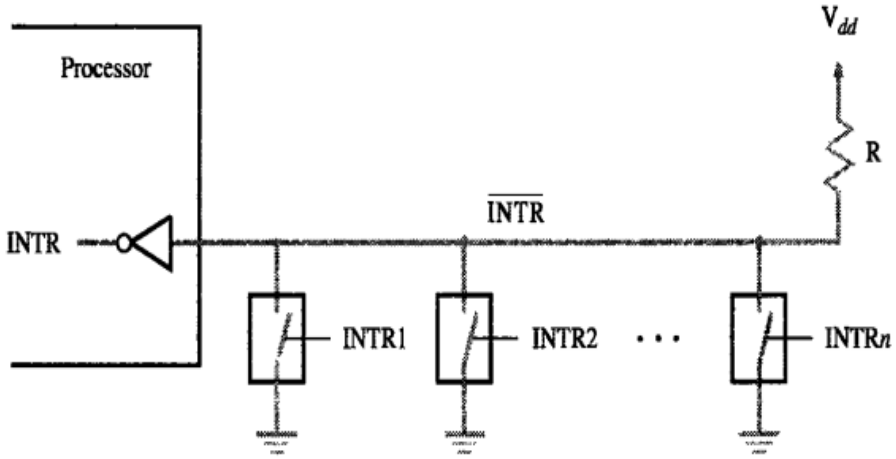


Fig : Circuit for common interrupt request line

To request an interrupt device closes its associated switch. Thus if all the interrupt request signals INTR1 to INTRn are inactive, the voltage on the interrupt request line is Vdd. This is the inactive state of the line. When the device requests the interrupt by closing its switch, the voltage line drops to zero causing the interrupt request line INTR received by the processor to go to 1. The value of INTR is the logical OR of the requests from individual devices, that is

$$INTR=INTR_1 + . \ . \ . \ .+ INTR_n$$

R is the pull up register because it pulls line voltage up to high voltage when the switches are open.

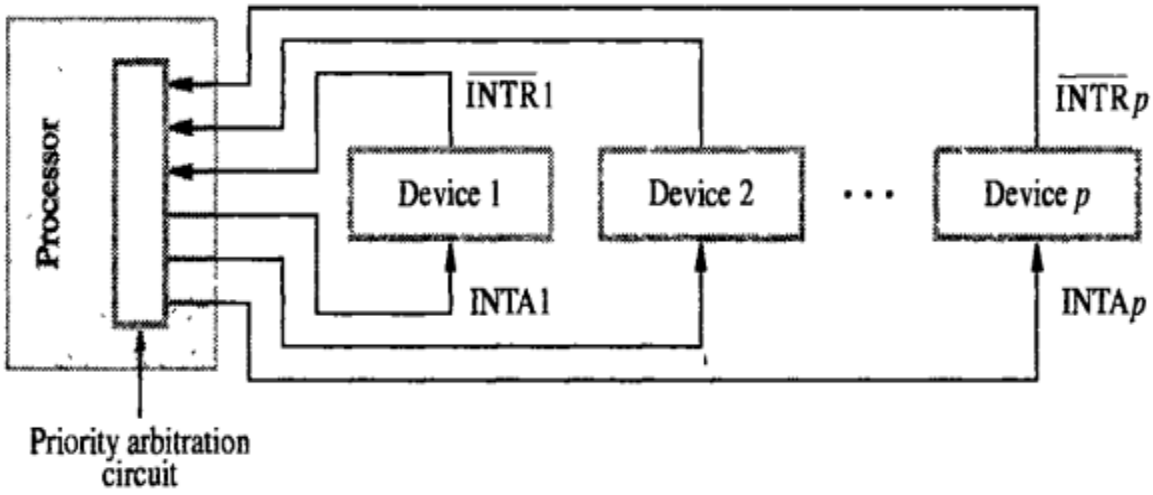
5)With supporting diagrams explain (i) Interrupt Nesting (ii) Vectored Interrupt (iii) Simultaneous Interrupt requests.

Interrupt Nesting

I/O devices should be organized in a priority structure. An interrupt request from a high priority should be accepted while the processor is serving another request from the lower priority device.

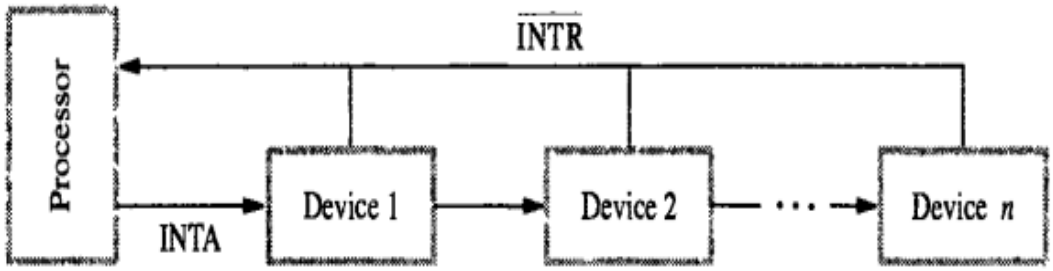
We can assign priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts from devices that have priorities higher than its own.

The processor is in supervisory mode when it is executing the OS routines. It switches to User mode before beginning to execute application programs. The privileged instructions can be executed only while the processor is running in the supervisory mode. A multiple priority scheme can be implemented by using separate interrupt request and interrupt acknowledge lines from each device.

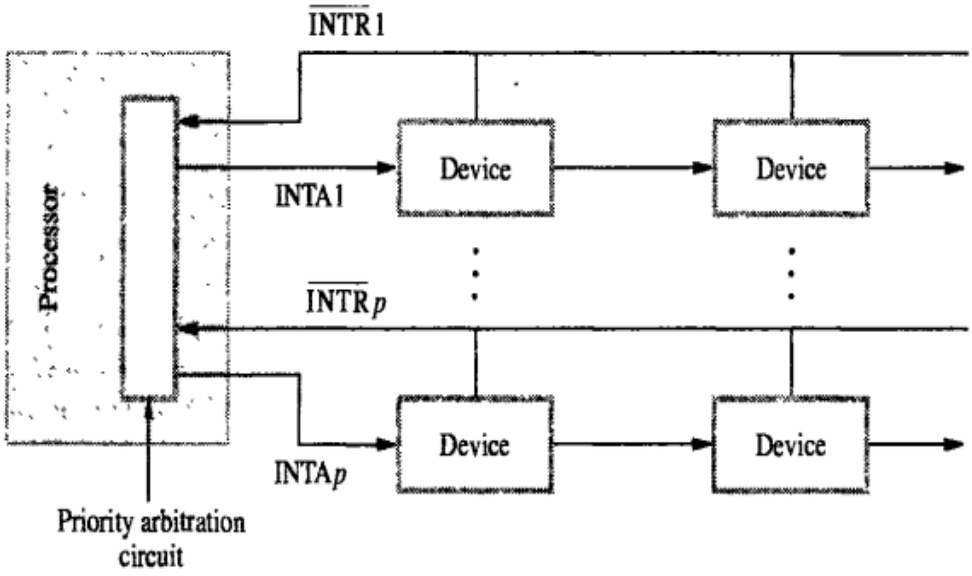


Simultaneous Requests

If several devices share one interrupt request line, some other mechanism is needed. When several devices raises interrupt request and $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1. The signal is received by device 1. Device 1 passes the signal onto device 2 only if it does not require any service. If device 1 has pending request for interrupt, it blocks the INTA signal and proceeds to put its identification code on to data lines. In daisy chain the device that is electrically closest to the processor has the highest priority.



Devices can be organized in groups and each group is connected at a different priority level. Within group devices are connected in daisy chain.



6) a) Explain basic input output operations . Include necessary diagrams, buffer registers, status bits and instructions.

Vectored Interrupts

A device requesting an interrupt can identify itself by sending special code to the processor over the bus. The code supplied by the device represents the starting address of the interrupt service routine. The code length is 4 to 8 bits. The processor reads this address called the interrupt vector and stores it in to the PC. The interrupt vector may also include a new value for a processor status register.

The interrupted device must wait to put on the bus only when the processor is ready to receive it. When the processor is ready to receive the vector interrupt code, it activates the interrupt acknowledge line INTA. The I/O device responds by sending its interrupt vector code and turning off INTR signal.

Basic Input/Output Operations

Consider a task that reads in a character input from a keyboard and produces a character output on a display. A simple way of performing such tasks is to use methods known as *program-controlled I/O*. The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.

Consider the problem of moving a character code from the keyboard to the processor. Striking a key store the corresponding character code in an 8-bit buffer register associated with the keyboard. Let us call this register DATAIN as shown in Fig 2.3. To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1. A program monitor SIN, and when SIN is set to 1, the processor reads the contents of DATAIN. When the character is transferred to processor, SIN is automatically cleared to 0. If the second character is entered at the keyboard, SIN is again set to 1 and the process repeats.

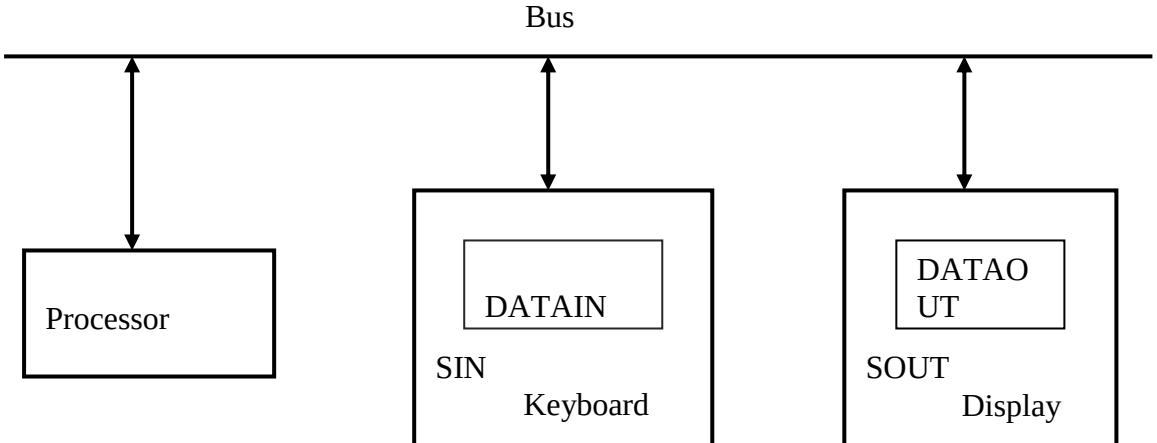


Fig 2.3: Bus connection for processor, keyboard and display

An analogous process takes place when the characters are transferred from the processor to display. A buffer register, DATAOUT and a status control register SOUT are used for this transfer. When SOUT equals 1, the display is ready to receive a character. Under program control, the processor monitors SOUT and when SOUT is set to 1, the processor transfers a character code to DATAOUT. The transfer of character to DATAOUT clears SOUT to 0, when the display is ready to receive a second character; SOUT is set again to 1. The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry known as *device interface.*

The processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to register R! By the following sequence of operations:

READWAIT Branch to READWAIT if SIN=0
Input from DATAIN to R1

The first instruction tests the status flag and the second performs the branch. The processor monitors the status flag by executing a short *wait loop* and proceeds to transfer the input data when SIN is set to 1 as a result of key being struck. The input operation resets SIN to 0. The sequence of operations are used for transferring the output to display are

WRITEWAIT Branch to WRITEWAIT if SOUT=0
Output from R1 to DATAOUT

Many computers use an argument called *memory mapped I/O* in which some memory address values are used to refer to the peripheral device buffer registers such as DATAIN and DATAOUT. The keyboard character buffer DATAIN can be transferred to register R1 in the processor by the instruction

b) Write an assembly language program to add 10 numbers using subroutine.

Calling program

```
Move    N, R1           R1 serves as a counter
Move    #NUM1, R2       R2 points to the list
Call    LISTADD         Call subroutine
Move    R0, SUM         Save result
```

-
-
-

Subroutine

```
LISTADD   Clear          R0              Initialize sum to 0
LOOP      Add            (R2)+, R0       Add entry from list
          Decrement      R1
          Branch > 0     LOOP
          Return                         Return to calling program
```

---

7) Write a program that reads a line of character through keyboard, echoes it on the diaplay and stores it in memory buffer. SIN and SOUT are mapped to bit-0 and bit-1 position of status register.

Main Program

```
        Move       #LINE, PNTR          initialize buffer pointer
        Clear      EOL                  clear end of line indicator
        BitSet     # 2, CONTROL         Enable keyboard interrupt
        BitSet     #9, PS               Set Interrupt enable bit in PS register
```

Interrupt-Service Routine

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