Lecture # 32

Interrupts and related concepts # 1

Covered by these slides and chapter # 17
(section 17.4)

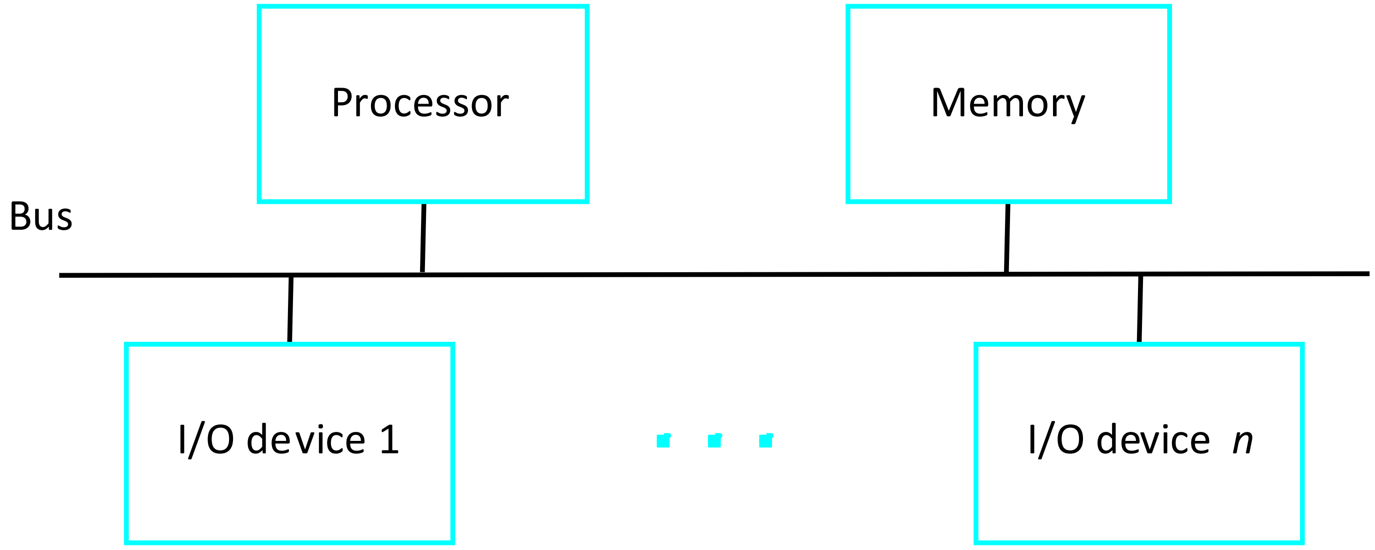
Basic Input/Output Operations

- We have seen instructions to:
    - Transfer information between the processor and the memory.
    - Perform arithmetic and logic operations
    - Program sequencing and flow control.
- Input/Output operations which transfer data from the processor or memory to and from the real world are essential.
- In general, the rate of transfer from any input device to the processor, or from the processor to any output device is likely to the slower than the speed of a processor.
    - The difference in speed makes it necessary to create mechanisms to synchronize the data transfer between them.

Accessing I/O devices

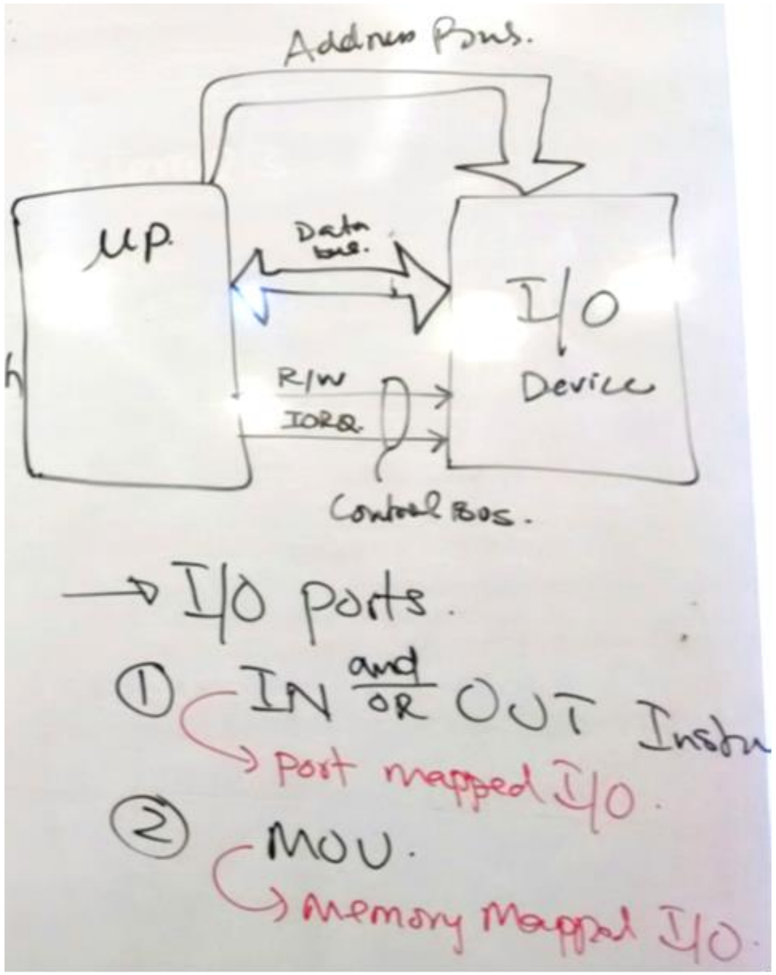
- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
  - Simpler software.
- I/O devices and the memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices.
  - I/O devices may have to deal with fewer address lines.
  - I/O address lines need not be physically separate from memory address lines.
  - In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Accessing I/O devices

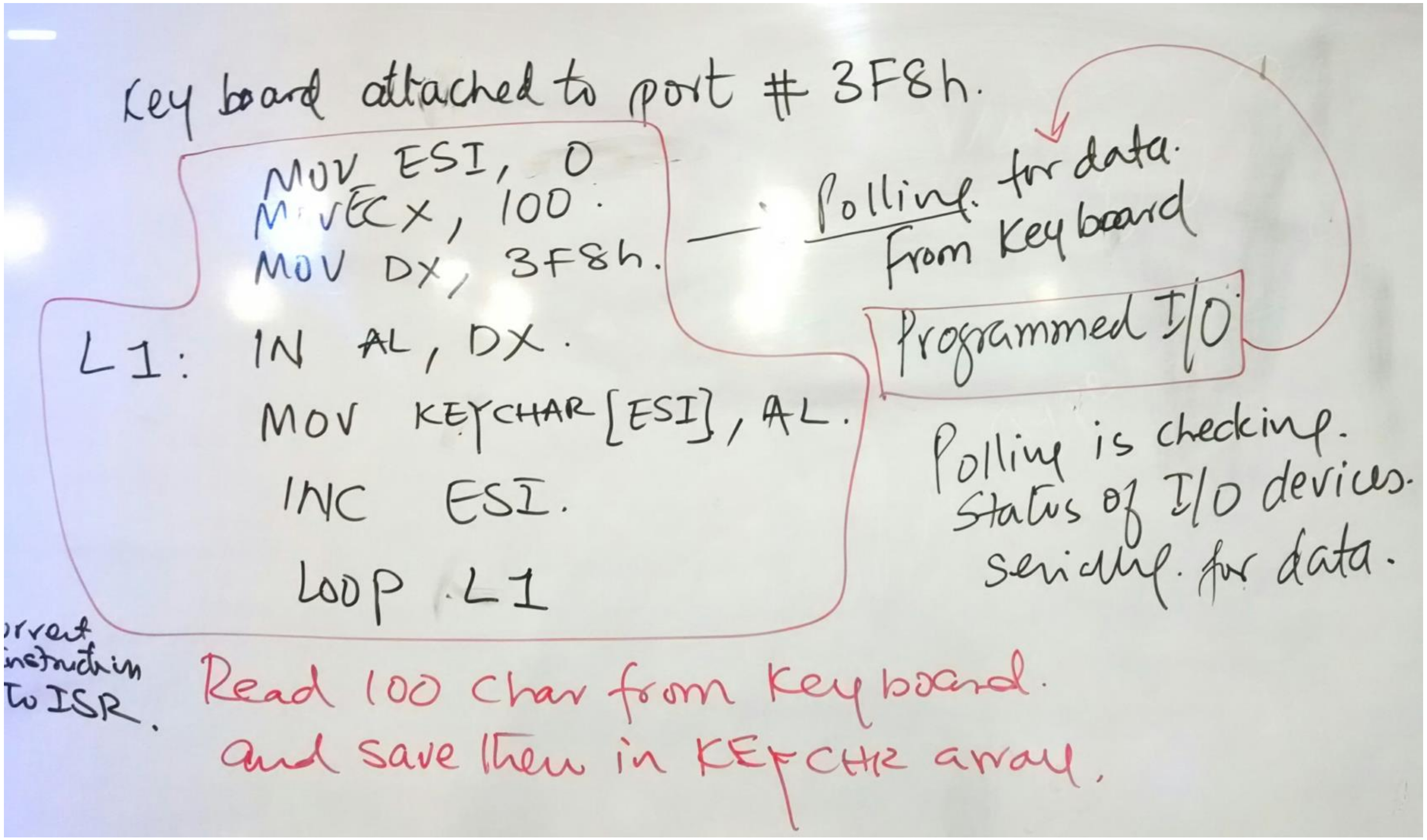


•*Multiple I/O devices may be connected to the processor and the memory via a bus.*
•*Bus consists of three sets of lines to carry address, data and control signals.*
•*Each I/O device is assigned an unique address.*
•*To access an I/O device, the processor places the address on the address lines.*
•*The device recognizes the address, and responds to the control signals.*

Processing with IORQ uses IN and OUT Instruct.



Class example of Programmed I/O



Key board attached to port # 3F8h.

```
    MOV ESI, 0
    MOV ECX, 100.
    MOV DX, 3F8h.
L1: IN AL, DX.
    MOV KEYCHAR[ESI], AL.
    INC ESI.
    LOOP L1
```

Polling for data. From Key board

Programmed I/O

Polling is checking. Status of I/O devices. servicing. for data.

rrupt instruction to ISR.

Read 100 Char from Key board and save them in KEYCHR array.

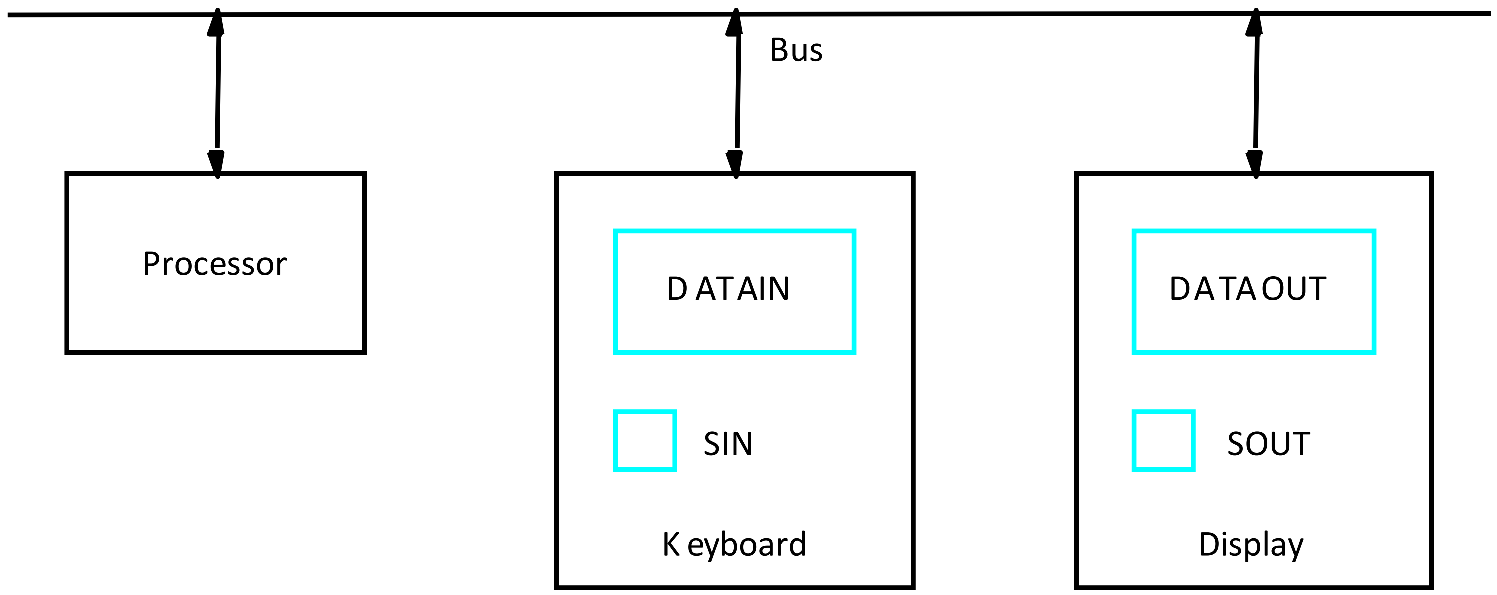
Q: How to do a synchronized programmed I/O?

- Let us consider a simple task of reading a character from a keyboard and displaying that character on a display screen.

- Buffer registers DATAIN and DATAOUT, and status flags SIN and SOUT are part of a circuitry known as device interface.

- **Write Assembly programs using IN / OUT instructions from the two scenarios described in the next two slides.**

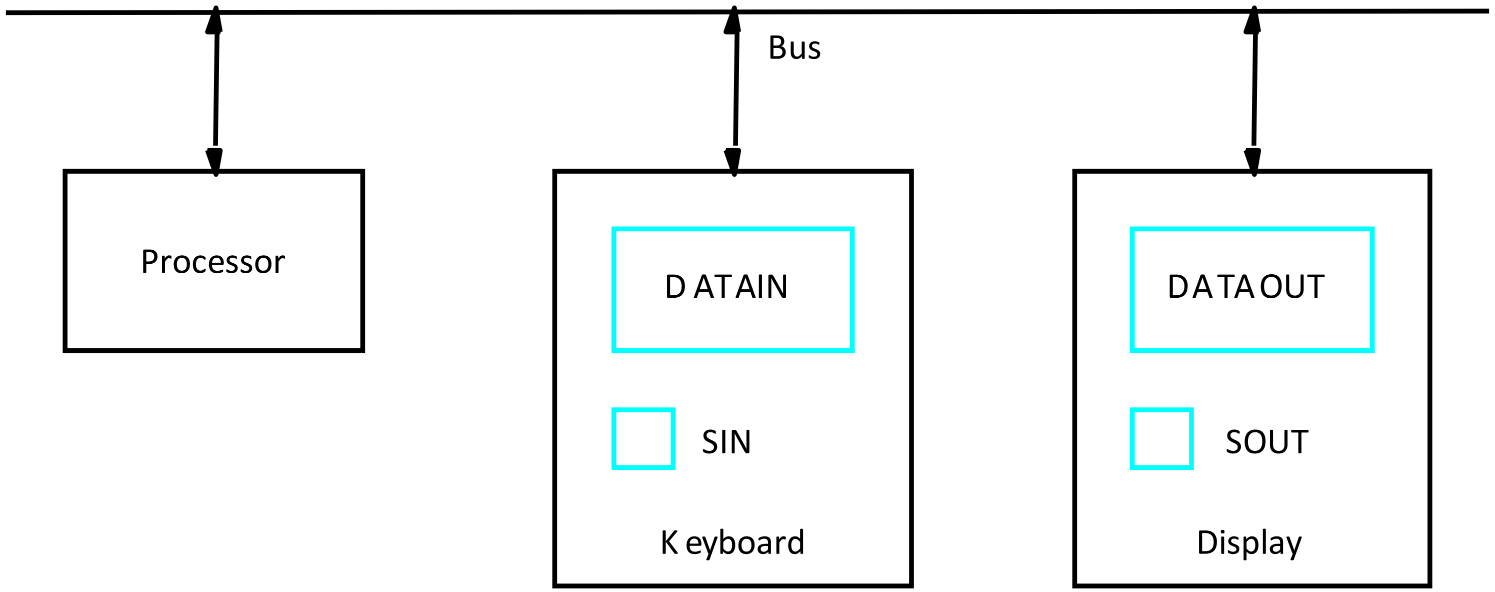
Data read from DATAIN by checking SIN



Input:

- *When a key is struck on the keyboard, an 8-bit character code is stored in the buffer register DATAIN.*
- *A status control flag SIN is set to 1 to indicate that a valid character is in DATAIN.*
- *A program monitors SIN, and when SIN is set to 1, it reads the contents of DATAIN.*
- *When the character is transferred to the processor, SIN is automatically cleared.*
- *Initial state of SIN is 0.*

Data written to DATAOUT by checking SOUT



Output:

- *When SOUT is equal to 1, the display is ready to receive a character.*
- *A program monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to the buffer DATAOUT.*
- *Transfer of a character code to DATAOUT clears SOUT to 0.*
- *Initial state of SOUT is 1.*

Accessing I/O devices (contd..)

- Program-controlled I/O:
  - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
  - Processor polls the I/O device.

- Two other mechanisms used for synchronizing data transfers between the processor and memory:
  - Interrupts.
  - Direct Memory Access (Not covered in this course).

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.

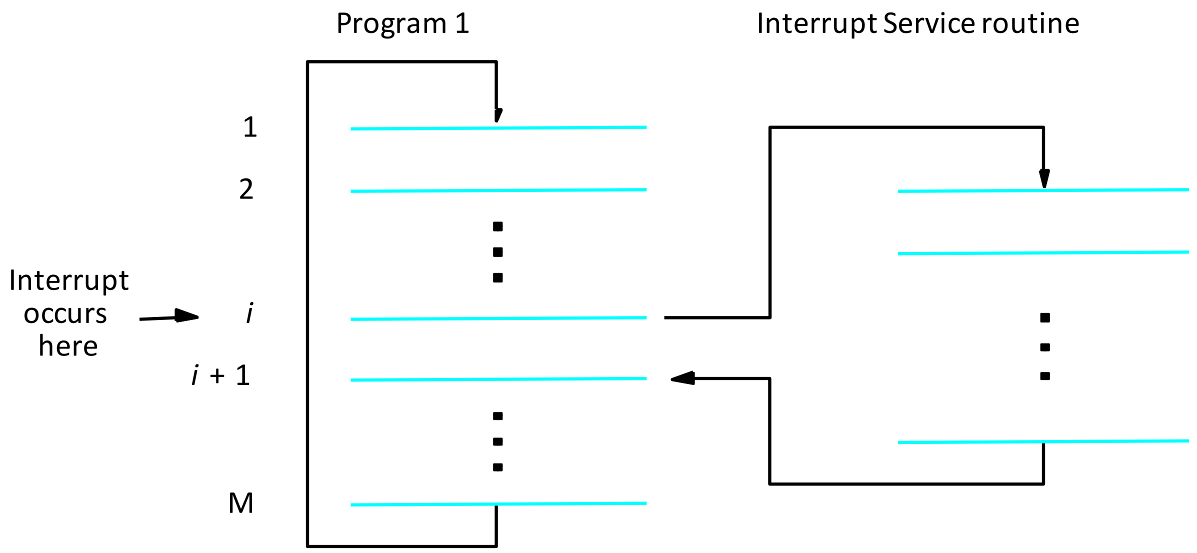
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.

- Processor can perform other useful tasks while it is waiting for the device to be ready.

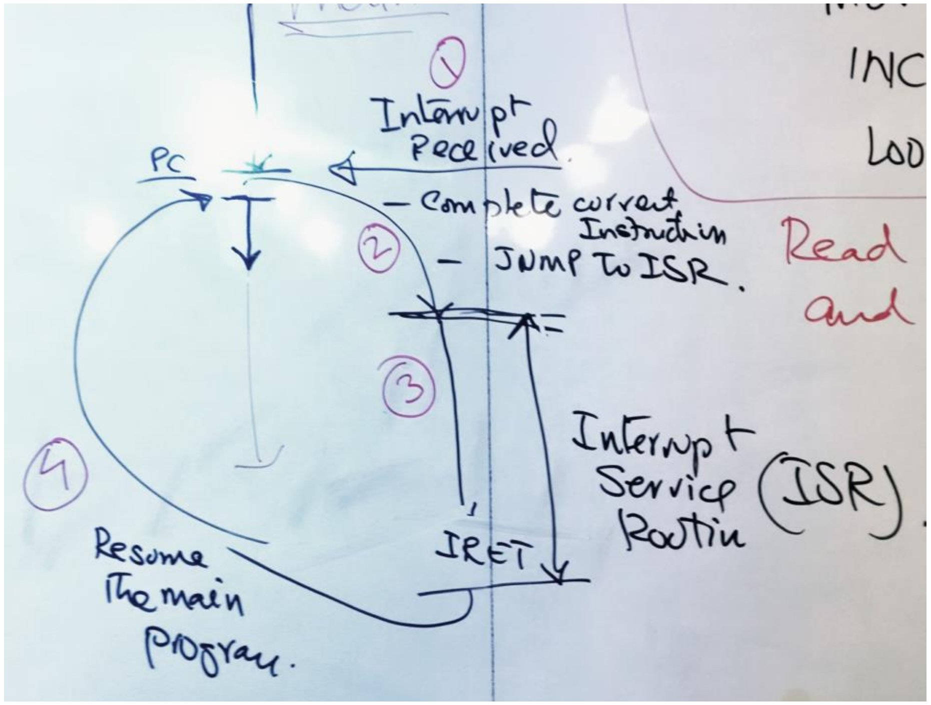
Polled IO versus Interrupt Driven IO

- Polled Input/Output (IO) – processor continually checks IO device to see if it is ready for data transfer
  - Inefficient, processor wastes time checking for ready condition
  - Either checks too often or not often enough
- Interrupt Driven IO – IO device interrupts processor when it is ready for data transfer
  - Processor can be doing other tasks while waiting for last data transfer to complete – very efficient.
  - All IO in modern computers is interrupt driven.

Interrupts (contd..)



- *Processor is executing the instruction located at address i when an interrupt occurs.*
- *Routine executed in response to an interrupt request is called the interrupt-service routine.*
- *When an interrupt occurs, control must be transferred to the interrupt service routine.*
- *But before transferring control, the current contents of the PC (i+1), must be saved in a known location.*
- *This will enable the return-from-interrupt instruction to resume execution at i+1.*
- *Return address, or the contents of the PC are usually stored on the processor stack.*



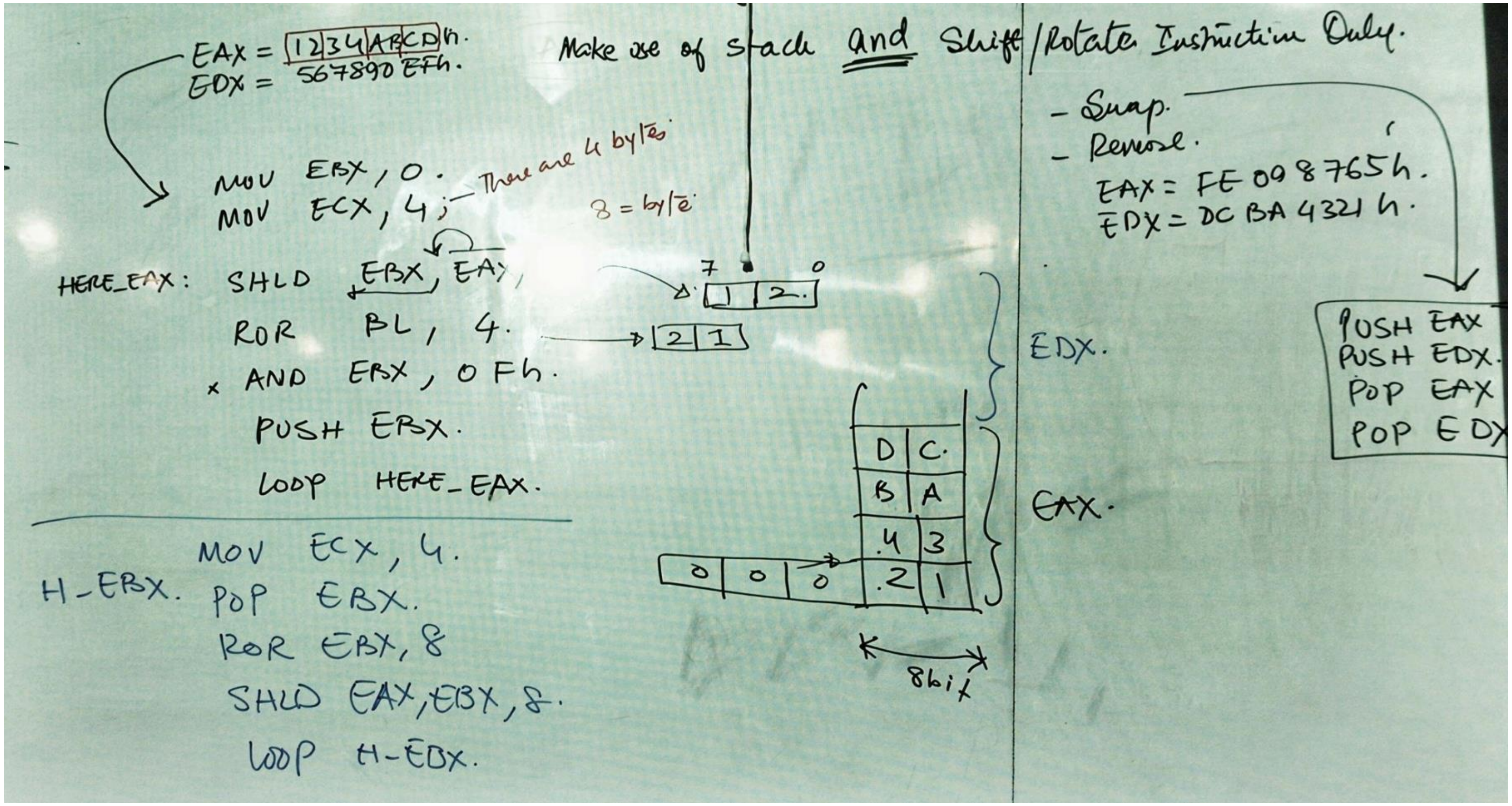
1. Processor received Interrupt
2. Save context (return address, registers, flags, etc.) and JMP to ISR
3. Interrupt Service Routing (ISR) or Interrupt Handler is executed
4. IRET instruction is last in the ISR and it returns controls to the main program

Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.
  - Interrupt-service routine and the program that it interrupts may belong to different users.
  - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
  - This will enable the interrupted program to resume execution upon return from interrupt service routine.

Midterm # 2 Selected Solutions

Question 3(a)



EAX = 1234ABCDh.
EDX = 567890EFh.

Make use of stack and Shift/Rotate Instruction Only.

- Swap.
- Reverse.

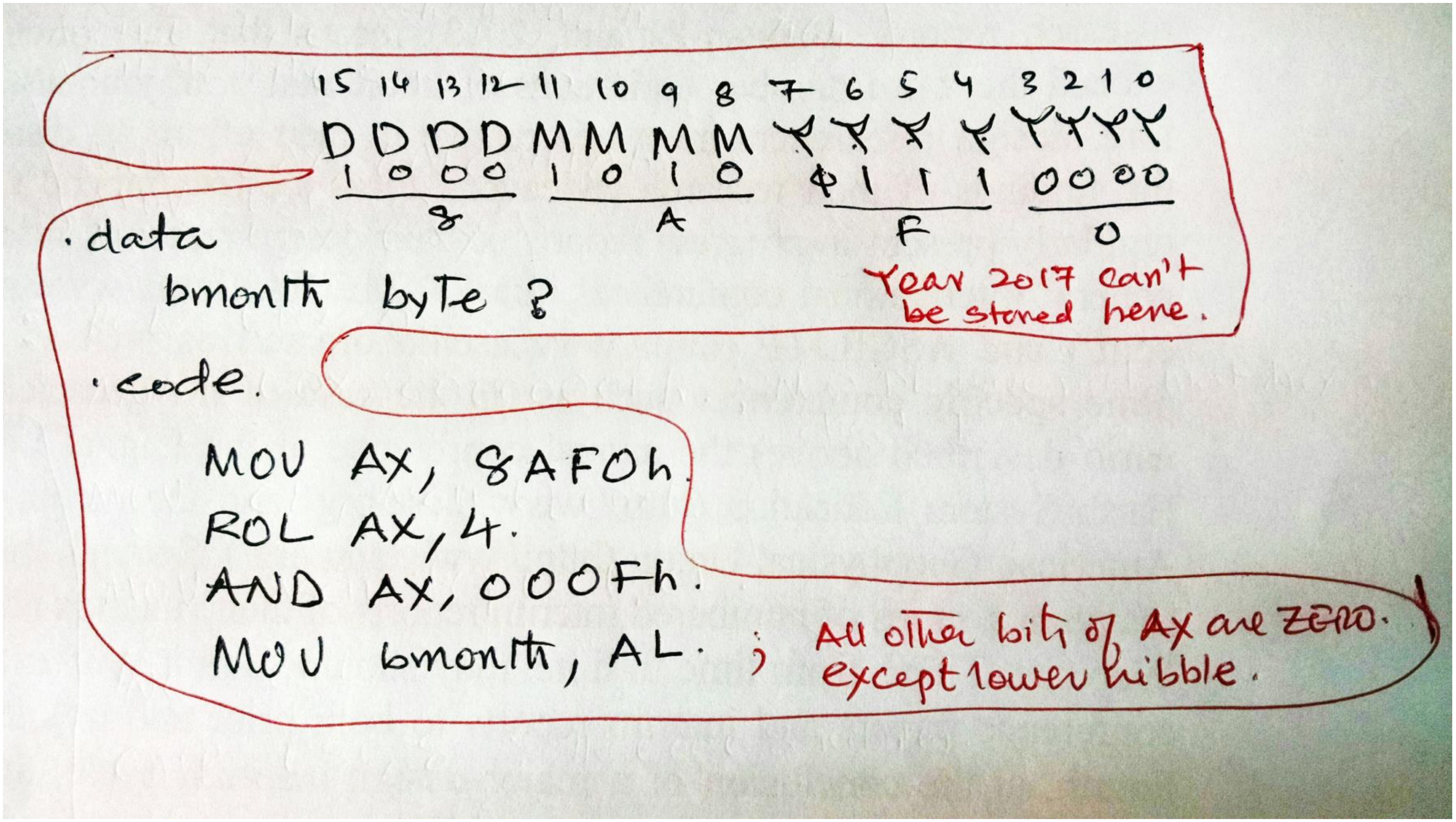
EAX = FE098765h.
EDX = DCBA4321h.

```
          MOV   EBX, 0.
          MOV   ECX, 4; — There are 4 bytes
                          8 = byte
HERE_EAX: SHLD  EBX, EAX,
          ROR   BL, 4.
        × AND   EBX, 0Fh.
          PUSH  EBX.
          LOOP  HERE_EAX.
```

```
       MOV   ECX, 4.
H_EBX. POP   EBX.
       ROR   EBX, 8
       SHLD  EAX, EBX, 8.
       LOOP  H-EBX.
```

```
PUSH EAX
PUSH EDX.
POP  EAX
POP  EDX
```

Question 3(b)



```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
D  D  D  D  M  M  M M Y Y Y Y Y Y Y Y
1  0  0  0  1  0  1 0 ϕ 1 1 1 0 0 0 0
    8           A         F       0
```

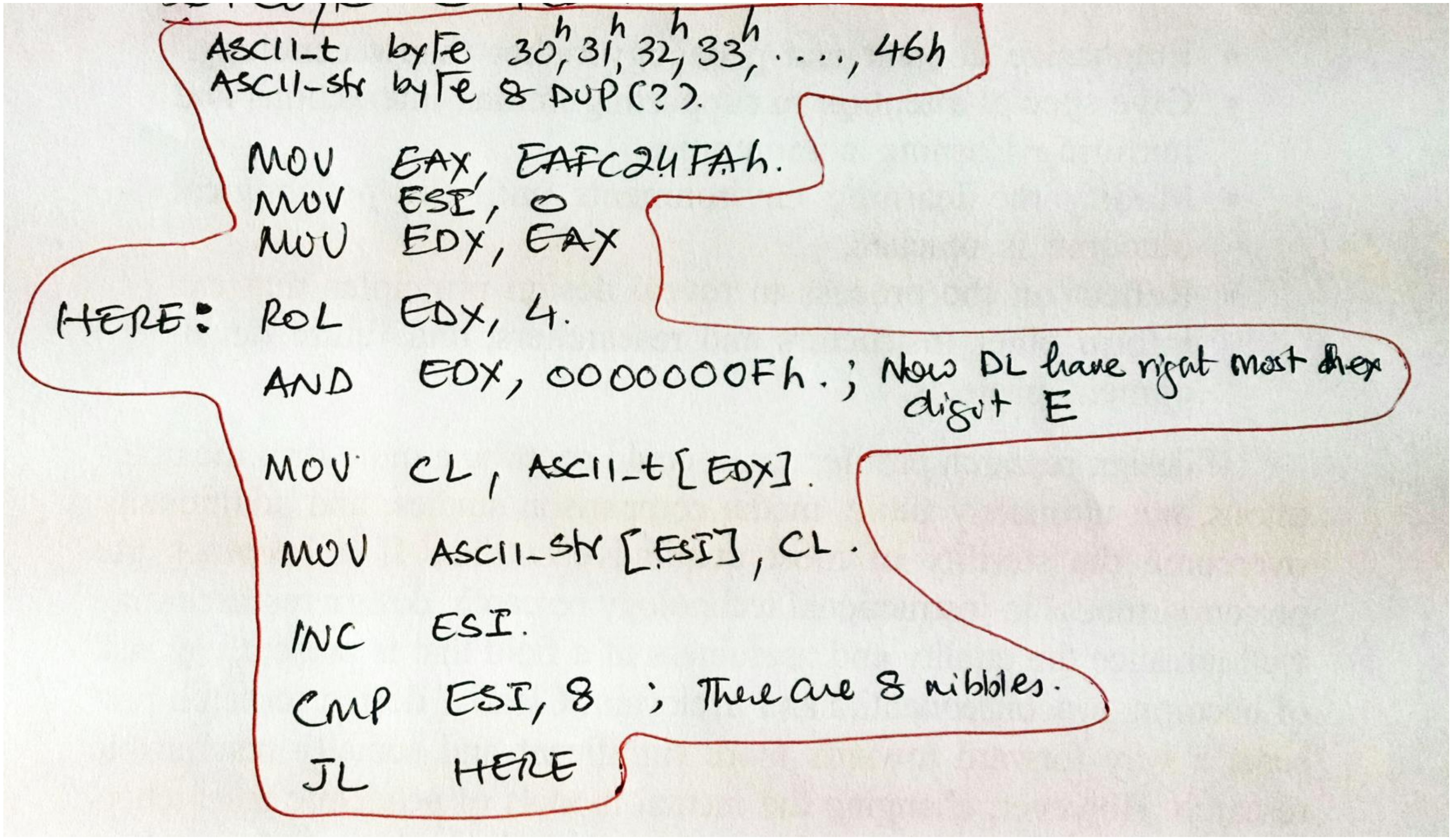
Year 2017 can't be stored here.

```
.data
    bmonth  byte ?

.code

    MOV  AX, 8AF0h
    ROL  AX, 4.
    AND  AX, 000Fh.
    MOV  bmonth, AL.  ; All other bits of AX are ZERO.
                        except lower nibble.
```

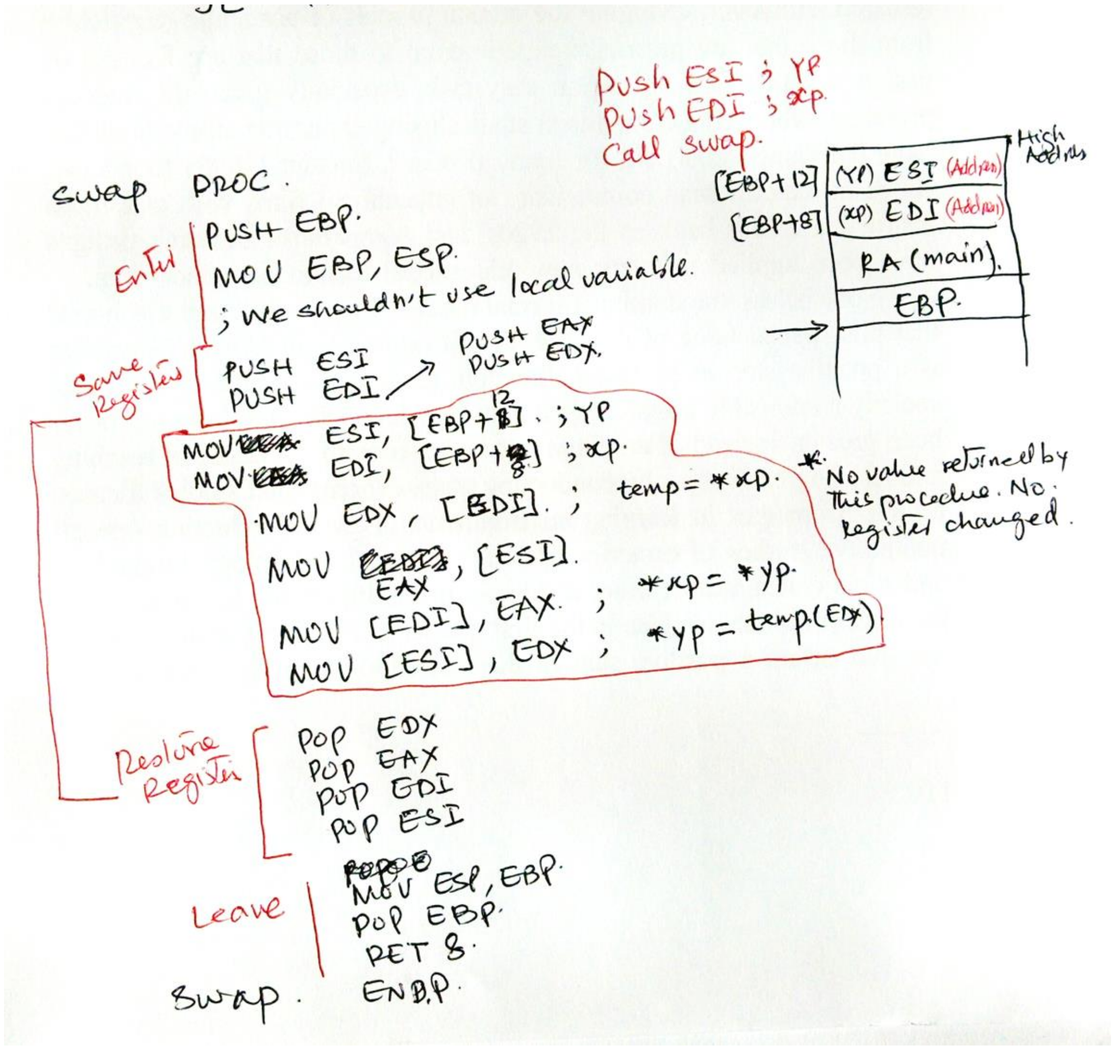
Question 4(a)



```
            ASCII_t    byte 30h, 31h, 32h, 33h, ...., 46h
            ASCII_str  byte 8 DUP(?).

            MOV   EAX, EAFC24FAh.
            MOV   ESI, 0
            MOV   EDX, EAX
HERE:       ROL   EDX, 4.
            AND   EDX, 0000000Fh. ; Now DL have right most hex
                                    digit E
            MOV   CL, ASCII_t[EDX].
            MOV   ASCII_str[ESI], CL.
            INC   ESI.
            CMP   ESI, 8   ; There are 8 nibbles.
            JL    HERE
```

Question 4(b)



```
Push ESI ; YP
Push EDI ; xp.
Call swap.

                              High Address
[EBP+12]  | (YP) ESI (Address) |
[EBP+8]   | (xp) EDI (Address) |
          | RA (main)          |
          | EBP.               |
      --> |                    |

swap  PROC.
Enter      PUSH EBP.
           MOV EBP, ESP.
           ; We shouldn't use local variable.
Save       PUSH ESI        PUSH EAX
Registers  PUSH EDI        PUSH EDX.
           MOV ~~EBX~~ ESI, [EBP+~~8~~12] . ; YP
           MOV ~~EBX~~ EDI, [EBP+~~12~~8] ; xp.
           MOV EDX, [EDI].  ;  temp = *xp.
           MOV ~~[EDI]~~ EAX, [ESI].
           MOV [EDI], EAX.  ;  *xp = *yp.
           MOV [ESI], EDX   ;  *yp = temp.(EDX)
                    * No value returned by this procedure. No register changed.
Restore    Pop EDX
Register   Pop EAX
           Pop EDI
           Pop ESI
Leave      ~~POPOE~~
           MOV ESP, EBP.
           Pop EBP.
           RET 8.
Swap.      ENDP.
```