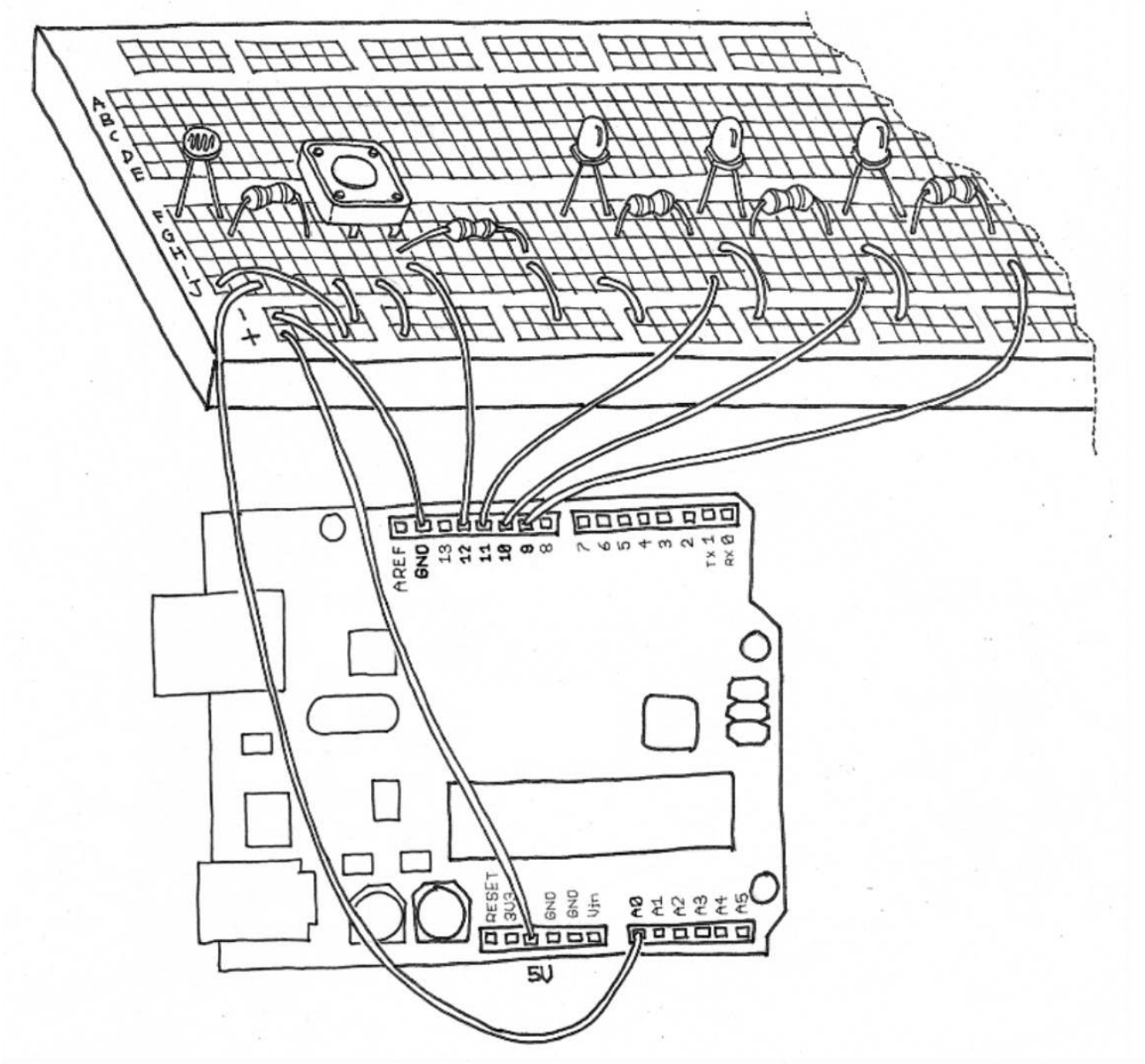
CS/EEE/INSTR F241
Microprocessor Programming and Interfacing

Lab 9 - Advanced BIOS Interrupts for Display



Dr. Vinay Chamola and Anubhav Elhence

What is a Procedure?

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called.

The syntax for procedure declaration:

***name* PROC**

; here goes the code

; of the procedure ...

RET

***name* ENDP**

(*name* - is the procedure name); The same name should be used for both the PROC and ENDP directives! (This is used to check the correct closing of procedures)

PROC and ENDP are compiler directives, so they are not assembled into any real machine code. The compiler just remembers the address of the procedure.

The CALL Instruction

The CALL instruction is used to call a procedure. The RET instruction is used to return to the operating system. The same instruction is used to return from a procedure (actually, the operating system sees your entire program as a special procedure).

For example, in the code below, the program calls the procedure m1, performs MOV BX, 5, and proceeds to the next instruction (MOV AX, 2)

```
CALL m1

MOV AX, 2

RET             ; Return to the OS

m1 PROC     ; Define the procedure 'm1'

MOV BX, 5

RET             ; Return to Caller.

m1 ENDP
```

There are several ways to pass parameters to a procedure. The easiest way to pass parameters is by using registers. Here is another example of a procedure that receives two parameters in AL and BL registers, multiplies these parameters, and returns the result in AX register. Since m2 is called four times, the final result in AX will be 2^4 (1 or 10H)

```
MOV AL, 1

MOV BL, 2

CALL m2

CALL m2

CALL m2

CALL m2

RET             ; Return to the OS

m2 PROC

MUL BL              ; The product of AL, BL is stored in AX

RET             ; Return to the Caller

m2 ENDP
```

The Stack

The Stack is an area of memory for keeping temporary data. The stack is used by the CALL instruction to keep return address for procedure, and the RET instruction gets this value from the stack and returns to that offset.

This also happens when INT instruction calls an interrupt (Recall INT 21h and INT 10h!). It stores the code segment and offset in the stack flag registe. Similar to RET, the IRET instruction is used to return from interrupt call.

The PUSH and POP Instructions

The stack is a LIFO data structure (Last In, First Out) can be accessed to store or retrieve data using these two instructions-

PUSH

PUSH - stores a 16 bit value (from a register or memory location) in the stack.

Syntax:

```
PUSH REG          ; AX, BX, DI, SI etc.

PUSH SREG         ; DS, SS, ES etc.

PUSH memory       ; [BX], [BX+SI] etc.

PUSH immediate ; 5. 3Fh, 10001000b etc.
```

POP

POP - gets 16 bit value from the stack and stores it in a register or a memory location.

Syntax:

```
POP REG         ; AX, BX, DI, SI etc.

POP SREG        ; DS, SS, ES etc.

POP memory      ; [BX], [BX+SI] etc.
```

The following example shows how the stack can be used to swap the values in the registers AX and BX. Notice the order of registers in the pop operation! (What would happen if we perform POP BX first?)

```
MOV AX, 1212h

MOV BX, 3434h

PUSH AX

PUSH BX

POP AX

POP BX
```

Tasks to be Completed

1. ***Reverse a string (your first name) in place using only the stack***

Note: Use dw instead of db in the data section since you'll use PUSH and POP

2. ***Write a procedure to calculate nPr, where the parameter n is stored in BX, the parameter r is stored in DX, and the result is stored in AX.***

Hint:

- **First, write a recursive procedure to calculate the factorial of a number**
- **nPr = n!/(n-r)!**
- **So, 5P2 = 5!/3! = 20 (14h)**

Sample Input:

BX = 05h, DX = 02h

Sample Output:

AX = 14h


BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
ज्ञानं परमं बलम्