ECS502U Lab: Week 9

QUESTION 1 Combine your work of the labs in weeks 6 and 8, by merging your implementation of the keypad loop of week 6 into the code of Question 2 in week 8. (If you have not completed one of these, you will want to start a fresh implementation based on q2.asm of week 8.) Specially, where you could just directly read/write to the connected virtual hardware in week 6, use the read and write/write C procedures of week 8 instead. Furthermore, you may have to adjust the bit patterns used for driving and reading the keypad to align with the instructions provided above (you have driven the columns high on the virtual keypad, whereas now the rows need to be driven high).Include your final code in the submission and ensure the following requirements are met, plus add describing text as stated below. The describing text preferably refers to source lines or procedures (otherwise labelled instructions). Unless stated otherwise below, such text may be as simple as \This is achieved by lines 7 through to 42."

```
; easy-to-use port names
RDpin equ 0xB3
WRpin equ 0xB2
A0pin equ 0xB5
A1pin equ 0xB4

ORG 8000h; it is set up to be more than 8000h in orden to interface it using the hyperterm

LJMP start;Long jump to start label

ORG 8100h
;Data base
KCODE0: DB 06h,5Bh,4Fh,71h  ; 1, 2, 3, f
KCODE1: DB 66h,6Dh,7Dh,79h  ; 4, 5, 6, e
KCODE2: DB 07h,7Fh,67h,5Eh  ; 7, 8, 9, d
KCODE3: DB 77H,3Fh,7Ch,58h  ; a, 0, b, c

ORG 8200h

start:
; set WRpin'and RDpin' high (off)
SETB RDpin ; read signal
SETB WRpin ; write signal

; set address to control register
SETB A0pin
SETB A1pin

; set control word
MOV  A,#81h
MOV  P1,A

; send write pulse to 8255
LCALL write

; set word to 0
MOV  A,#00h
MOV  P1,A
```

```
; set port to write to (A)
CLR  A0pin
CLR  A1pin
; send write pulse to 8255
LCALL write

; set port to B
SETB A0pin
CLR  A1pin
; send write pulse to 8255
LCALL write

; set port to C
CLR  A0pin
SETB A1pin
; send write pulse to 8255
LCALL write

; clear registers (just to be on safe side)
MOV R0,#00h
MOV R1,#00h
MOV R2,#00h
MOV R3,#00h
MOV R7,#00h
MOV R6,#00h
MOV R5,#00h

;------- Start Keypad Loop ---------

; for testing purposes
Kloop:; waiting until a key is pressed
        MOV P1, #0xF0; setting P1 as default one
        LCALL write_c
        LCALL read
        CJNE A, #0xF0, pressed; if key is pressed, it will go "pressed" label otherwise it will
go to the next line
        LJMP cont; displaying and making some delay
;Establish which key has been pressed
pressed:;looking for the the right COLUMN
        MOV R5, #0 ; R5 =0, it will be used as the index for my data base later
        JB A.0,row1; if the bit 0 of the Acc A is set, it will mean that the key pressed is on this
column and jump to row1, othersie it will move down

        MOV R5, #1; R5 =1, it will be used as the index for my data base later
        JB A.1,row1; if the bit 1 of the Acc A is set, it will mean that the key pressed is on this
column and jump to row1, othersie it will move down.

        MOV R5, #2 ; R5 =2, it will be used as the index for my data base later
        JB A.2,row1; if the bit 2 of the Acc A is set, it will mean that the key pressed is on this
column and jump to row1, othersie it will move down.
```

```
        MOV R5, #3 ; R5 =2, it will be used as the index for my data base later
        JB A.3,row1; if the bit 3 of the Acc A is set, it will mean that the key pressed is on this column and jump to row1, othersie it will move down.

row1:
        MOV P1, #0x1F; find row1
        LCALL write_c
        LCALL read
        MOV DPTR,#KCODE0; Moving to the Data Pointer KCODE0
        CJNE A, #0x10, load_data; comparing if the value of A is equal to 0x10 (in the bit 4 of the Acc A)
        SJMP row2
row2:
        MOV P1,#0x2F; find row2
        LCALL write_c
        LCALL read
        MOV DPTR,#KCODE1; Moving to the Data Pointer KCODE1
        CJNE A,#0x20,load_data; comparing if the value of A is equal to 0x20 (in the bit 5 of the Acc A)
        SJMP row3
row3:
        MOV P1,#0x4F ; find row3
        LCALL write_c
        LCALL read
        MOV DPTR,#KCODE2; Moving to the Data Pointer KCODE2
        CJNE A,#0x40, load_data; comparing if the value of A is equal to 0x10 (in the bit 6 of the Acc A)
        SJMP row4

row4:                   ;find row4
        MOV DPTR,#KCODE3; Moving to the Data Pointer KCODE3
        SJMP load_data

load_data:

        MOV A, R5;uploading the value of R5 in A
        MOVC A,@A+DPTR; moving to the index of value of A was store and storing back to A
        MOV R3,2; R3 get the valaue of the register 2
        MOV R2,1; R2 get the valaue of the register 1
        MOV R1,0; R1 get the valaue of the register 0
        MOV R0, A; R0 get the valaue of the accumulator A

        LCALL display;Long call dispaly label
        LCALL delay1; calling the long delay
        LJMP KLoop ;return to start of loop
;------- End Keypad Loop ---------
write:
        CLR WRpin ; writing is active−low
        NOP ; wait for the write to complete
```

```
        SETB WRpin ; writing done
        NOP ; let it settle
        RET
write_c:
; set the port to C
        CLR A0pin
        SETB A1pin
        NOP ; wait for the 8255 to settle
        ACALL write ; execute the write

        RET
read:
        MOV P1, #0FFH ; configure P1 as input
; we always read from port C
        CLR A0pin
        SETB A1pin
        NOP ; wait for the 8255 to settle
        CLR RDpin ; reading is active−low
        NOP ; wait for the 8255 to transmit the data
        MOV A,P1 ; read the data from P1 into the accumulator
        SETB RDpin ; reading done
        NOP ; wait for the 8255 to settle
        RET
cont:
;Refresh the display
        ACALL display
        ACALL delay
        LJMP KLoop ;return to start of loop
display:
   MOV   A,R0  ; load acc with first character
   MOV   P1,A
   CLR   A0pin
   CLR   A1pin ; select Port A
   ACALL write ; write character from acc to P1

   MOV   A,#00000001b
   MOV   P1,A
   CLR   A1pin
   SETB  A0pin ; select Port B
   ACALL write ; enable first display module
   LCALL delay

   MOV   A,#00h
   MOV   P1,A
   ACALL write ; clear displayed character before moving on


   MOV   A,R1  ; load acc with second character
   MOV   P1,A
   CLR   A0pin
   CLR   A1pin ; select Port A
```

```
    ACALL write ; write character from acc to P1

    MOV   A,#00000010b
    MOV   P1,A
    CLR   A1pin
    SETB  A0pin ; select Port B
    ACALL write ; enable second display module
    LCALL delay

    MOV   A,#00h
    MOV   P1,A
    ACALL write ; clear displayed character before moving on

    MOV   A,R2  ; load acc with third character
    MOV   P1,A
    CLR   A0pin
    CLR   A1pin ; select Port A
    ACALL write ; write character from acc to P1

    MOV   A,#00000100b
    MOV   P1,A
    CLR   A1pin
    SETB  A0pin ; select Port B
    ACALL write ; enable third display module
    LCALL delay

    MOV   A,#00h
    MOV   P1,A
    ACALL write ; clear displayed character before moving on


    MOV   A,R3  ; load acc with fourth character
    MOV   P1,A
    CLR   A0pin
    CLR   A1pin ; select Port A
    ACALL write ; write character from acc to P1

    MOV   A,#00001000b
    MOV   P1,A
    CLR   A1pin
    SETB  A0pin ; select Port B
    ACALL write ; enable fourth display module
    LCALL delay

    MOV   A,#00h
    MOV   P1,A
    ACALL write ; clear displayed character before moving on

    RET
delay1:; creating a long delay using loop1 and loop2
        MOV R6,#0xDF ; setting and storing data in the register R6
```

```
        MOV R7,#0xDF; setting and storing data in the register R7
loop1:  DJNZ R6,loop1;Decrement and jump to label if R6 is not zero, otherwise it goes to
the next line
loop2:  DJNZ R7,loop1;Decrement and jump to label if R7 is not zero, otherwise it goes to
the next line
        RET
delay:
   MOV   A, #0xFF   ; delay a bit for...
dly:
   DJNZ  acc, dly   ; ...for debouncing
   RET
END
```

Question1

1. Periodically write the contents of the registers R3 (left most digit), R2, R1 and R0 (of Bank 0) to the display. The contents of these registers hold the segment code for the corresponding decimal character such that if the value of the right-most digit is 3 decimal, the R0 register holds the segment code to display a "3" character. Add a brief statement to describe how your code achieves this.

The first time when the "display" label is called, it is when the "Kloop" label refreshes the display modules by calling "cont". In order to display something, first it is selected Port A to **write the character** from the accumulator A to Port P1. Then, It is selected Port B in order to enable **display module.** Lastly, it is **cleared display character** before moving on. This process is repeated for the next registers.

2. Scan the keypad for any single key press. If pressed, the key that is pressed must be identified and its numerical value used to ensure the left-most digit displays the correct value of this key. Again, add a brief statement to describe how your code achieves this.

My "Kloop" label scans and determines if any single key has been pressed at any time. It is done by calling the "write_c" label and then the "read" label. The bits are store in the accumulator A which then it is compared with the default set of bits 0F0h. It will jump to the label "pressed" if after comparing the two bytes are not equal. Otherwise, it will return to the "Kloop" label.

3. The keypad should include a debounce mechanism as necessary, and the display must be programmed so as to provide flicker-free illumination of the characters held in the R0, R1, R2 and R3 registers. Again, add a brief statement to describe how your code achieves this. Furthermore explain how you concluded that your results are satisfactory.

In order to provide flicker-free illumination of the characters held in the R0, R1, R2 and R3 registers, a long delay must be added/called after the "display" label. In the code written, it can be this done by calling the label **delay1,** where two different loops are added in order to stay there for an enough period of time. It can be seen that it works satisfactorily because it only displays one of the display module at a time and if any key is pressed, the current

display modules will be shifted to the left and the new value will be displayed in the right-most display module.

4. Describe how your code responds to concurrent key presses. Which of them is being picked up? Why is this the case?

When more than one key are concurrently pressed, the code first scans for the column, starting from the left-most column. When the rows are scanned, the top-most row is scanned first. For example, if the keys “5” and “3” are pressed concurrently, the left-most column is selected, which key “5” belongs to. The top-most row is selected, which belongs to the key “3”. Finally, the value of key “2” is displayed, as this corresponds to the selected column and row.

QUESTION 2

(a) Explain the difference between port mapped Input/Output and memory mapped Input/Out-put. Which of them apply to the 8051, and which assembly instructions are used?

- **Port-Mapped Input/Output (PMIO – also called “Isolated I/O** or **Separate I/O”)**
    - It uses **“a special class of CPU instructions”** designed specially for performing I/O.
    - Example, **Intel microprocessors based on the x86 and x86-64 architecture.**
    - I/O devices have a **“separate or different address space”** from the general memory, either accomplished by an extra “I/O” pin on the CPU’s physical interface, or an entire bus dedicated to I/O.

    Instruction use: MOV

- **Memory-Mapped I/O**
    - It uses **“the same bus”** to address both memory and I/O devices, and the CPU instructions used to read and write to memory are also used access I/O devices.
    - To accommodate the I/O devices, areas of the CPU **“address space must be reserved”,** this does not have to be permanent.
    - Example, the Commodore 64 could bank switch between its I/O devices and regular memory.
    - The **I/O devices monitor the CPU’s address bus and respond to any CPU access** of their assigned address space, mapping the address to their hardware registers.

    Instruction use: MOVX

The 8051 has components of both. However in Memory-Mapped configuration the MOVX is the only instruction available to access external data memory.

(b) With regard to memory mapped Input/Output describe what is meant by partial and full address decoding. Give a simple example of each.

- **Partial Address Decoding**

- It is the **"simplest** and **least expensive"** form of address coding.
- However, it is very inefficient scheme because of **5 of the 16 address lines are not used** and **one of the two memory chips is always reduced to 2048 bytes**.

Example: Provided A15 is low, memory device 1 is addressable at addresses from 0 to 1023, then again at addresses 1024 to 2947 and so on. The same memory device effectively appears multiple times in the memory map. All the memory is consumed between the two memory devices. (Memory Device 1 and Memory Device 2 are always active).

- **Full Address Decoding**

  - A computer stems is said to have full address decoding when each addressable location within a memory component corresponds to a single address on the CPU's address bus. It means that **"every address line is used"** to specify each physical memory location, through a combination of specifying a device and a location with it (there is no redundancy).
  - **Each memory device effectively appears only once in the memory map**. Furthermore, not memory is consumed unnecessarily; there is no redundancy.

Example:

- Memory Device 1 is active only when the address lines are: 111111ddddddddddd bin.
- Memory Device 1 is active only when the address lines are: 111110ddddddddddd bin

d = address lines that select different locations within the memory device.

ECS502U Lab: Week 10

QUESTION 1: In week 8 the exam-style questions asked to provide code snippets around timers. You may want to re-use these to complete the following tasks in the MCU8051 simulator or on the development board (where possible):

1.Provide 8051 assembly code to complement P1.0 after 128 machine cycles, using timers. Briefly describe, or provide code, how to solve this problem without timers. Then provide an implementation (with or without timers) to complement P1.0 after 1024 machine cycles.

128 machine cycles (using timers):

```
ORG 0000h     ; origin at 0000 hex
; **************
; initialisation
; **************
```

```
MOV TMOD,#01; Setting up time 0, mode 1 (16- bit timer or counter)
MOV P1,#0x00; clearing the pin port PORT1 ( P1 = 00h)
; *************
; main program
; *************
start:    ;128 machine cycles = 0080h (in hexadecimal)
          ;FFFF - 80 = FF7F counting from 7F(TL0) to FF(TH0)
          MOV TL0,#0x7F;loading 7Fh to low byte
          MOV TH0,#0xFF;loading 14h to high byte
          CPL P1.0; complementing bit 0 Port1 (toggling it)
          ACALL delay; calling subroutine delay: time delay: 138.88 us
          SJMP start; short jump to start again
; ******************
; end of main program
; ******************

; ********************
; delay loop routines
; ********************
delay:
          SETB TR0; setting timer 0

again:  JNB TF0,again; monitor when Timer rolls over or reaches again 00
          CLR TR0; stopping timer 0
          CLR TF0; clearing timer 0 flag
          RET
```

128 machine cycles (Not using timers)

```
org 0000h

MOV     P1,#00h   ; clear P1 pins (P1 will be used for output)
nop; wait to be set
start:
          MOV A, #3Ch ; Clock set to :12000Khz - > (1/12Mkhz)x12 = 1 us, then 128 machines
cycles will equal to 128x1 us = 128 us.
          ACALL delay; calling delay to make the 128 machine cycles
          CPL P1.0; complementing P1.0
          SJMP start;jumtp to start

delay:  DJNZ  A, delay; decrease the Acc and then return
          RET
```

1024 machine cycles (Using timers)

```
ORG 0000h     ; origin at 0000 hex
; **************
; initialisation
; **************
MOV TMOD,#01
MOV P1,#0x00
; *************
; main program
; *************
start:    ;1024 machine cycles = 400h (in hexadecimal)
```

```
        ;FFFF - 400 = FBFF counting from FF(TL0) to FB(TH0)
        MOV TL0,#0xFF;loading FFh to low byte
        MOV TH0,#0xFB;loading FBh to high byte
        CPL P1.0; complementing bit 0 Port1 (toggling it)
        ACALL delay; calling subroutine delay: time delay: 138.88 us
        SJMP start; short jump to start again
; ******************
; end of main program
; ******************

; ********************
; delay loop routines
; ********************
delay:
        SETB TR0; setting timer 0

again:  JNB TF0,again; monitor when Timer rolls over or reaches again 00
        CLR TR0; stopping timer 0
        CLR TF0; clearing timer 0 flag
        RET
```

1024 machine cycles (Not using timers)

```
org 0000h

MOV     P1,#00h   ; clear P1 pins (P1 will be used for output)
nop; wait to be set
start: ;Clock set to :12000Khz - > (1/12Mkhz)x12 = 1 us, then 1024 machines cycles will
equal to 1024x1 us =1ms 24 us.
        MOV R7, #0xFF ; storing ffh in R7 register
        MOV R6,#0xFC; storing fch in R6 register
        nop
        ACALL delay; calling delay to make the 1024 machine cycles
        CPL P1.0; complementing P1.0
        SJMP start;jumtp to start

delay:  DJNZ R7, delay; decrease R7
delay1: DJNZ R6, delay1; decrease R6 and then return
        RET
```

2. Develop a program with as few instructions as possible that displays the number of key presses of a button (up to 255). Do so using an LED panel (of 8 LEDs) and a simple keypad/buttons (either as virtual hardware in the simulator or using the LED/switch module for the development board), and interpret the 8 LEDs as a binary string. Argue why your solution is optimal. Hint: start by describing the essential steps of this process (e.g., setting values on the LED panel), and how these translate into micro-controller instructions/actions.

Using timers:

```
ORG 8000h     ; origin at 0000 hex
; **************
; initialisation
; **************
MOV     A, #0ffh
MOV     P3, A    ; configure P3 for input
MOV     A, #00h
MOV     P1, A    ; clear P1 pins (P1 will be used for output)

MOV TMOD,#0x50 ; setting up counter 1, mode2,C/T =1
SETB P3.5; make P3.5 input

start:
        MOV TL1,#0x00; setting timer 1 to low byte
        MOV TH1,#0xFF; setting timer 1 to high byte
slave:  SETB TR1; start the counter
        MOV A,TL1; getting a copy of count TL1
        MOV P1,A; displaying it on port 1
        JNB TF1,slave; keeping doing it until Timer flag (TF) = 0
        CLR TR1; stop the counter 1
        CLR TF1; make TF(Timer Flag) = 0
END
```

No timers:

```
org 0000h

MOV P1,0x00; clearing Port1 (output)
start:  JNB P3.0, start ;Jump if bit not set
        INC P1 ; increase P1 by one

pressed:JB P3.0,pressed; jump if bit set
        SJMP start; short jump to start
```

When we are not using timers, there is less use of machine cycles which make the code more optimal and efficient. It is started by configuring P1 to output by initializing it with 00h. In the “start” label, it will wait until the bit in P3.0 is set and then increment P1 by one, going to the “pressed” label waiting for the same bit to be low again and jump to start. The Virtual HW LED panel has been used to set the Port P1 with pins 0 to 7 and the simple Key pad has been set using the Port P3 and pin 0

.

3. Using the simulator's interrupt monitor (via Virtual MCU ! Interrupt monitor), develop and debug a program that starts or stops timer 1 whenever the external interrupt 1 is triggered. (The interrupt can be triggered by clicking the rocket symbol in the interrupt-monitor window.)

```
ORG 0000H
SJMP main; it jumps to main label

ORG 0013H
CPL TR1 ; complementing TR1 by toggling it
RETI ;return from interrupt

ORG 0030H
main:
```

```
MOV IE,#0x84 ;(Enabling interrupt IE = 0X10 and External Interrupt IE.2 = 0x04 )
MOV TMOD,#0x20 ;Time0, mode 2(auto reload)
MOV TH1,#0x00;(Setting timer 1 high to 00)

again: SJMP again; jump here until external interrupt is detected

END
```

QUESTION 2

(a) Within the 8051 micro-controller, Port 0 is used for both providing the lower half of the address bus and for the bidirectional data bus. Explain how this is possible and provide an illustration showing how you would connect an 8051 to an external 64Kbyte Random Access Memory (RAM) device including any interface logic that may be required. You should also clearly state the role of the ALE signal.

Since the Pc **(program counter)** of the 8051 is 16-bit, it is capable of accessing up to 64K bytes of program code.

- **Access External Memory:** In the 8051, the ports P0 and P2 provide the 16-bit address to access external memory.
  - P0 provides the lower 8-bit address A0-A7
  - P2 provides the upper 8-bit address A8-A15
  - P0 also provides the 8-bit data bus D0-D7
- P0.0-P0,7 are used for both the address and data paths (address/data multiplexing)

Example: In order to access an 8051 to an external 64K byte RAM (Random Access Memory), it is used the DPTR register and an instruction called MOVX, where X stands for external (the data memory space must be implemented externally)

Therefore:

MOVX A, @DPTR (**bringing** externally **stored data** into the CPU)

```
MYXDATA     EQU    1000H
COUNT       EQU     30
…
            MOV DPTR,#MYXDATA
            MOV R2,#COUNT

AGAIN:      MOVX A,@DPTR
            MOV P1,A
            INC DPTR
            DJNZ R2,AGAIN
```

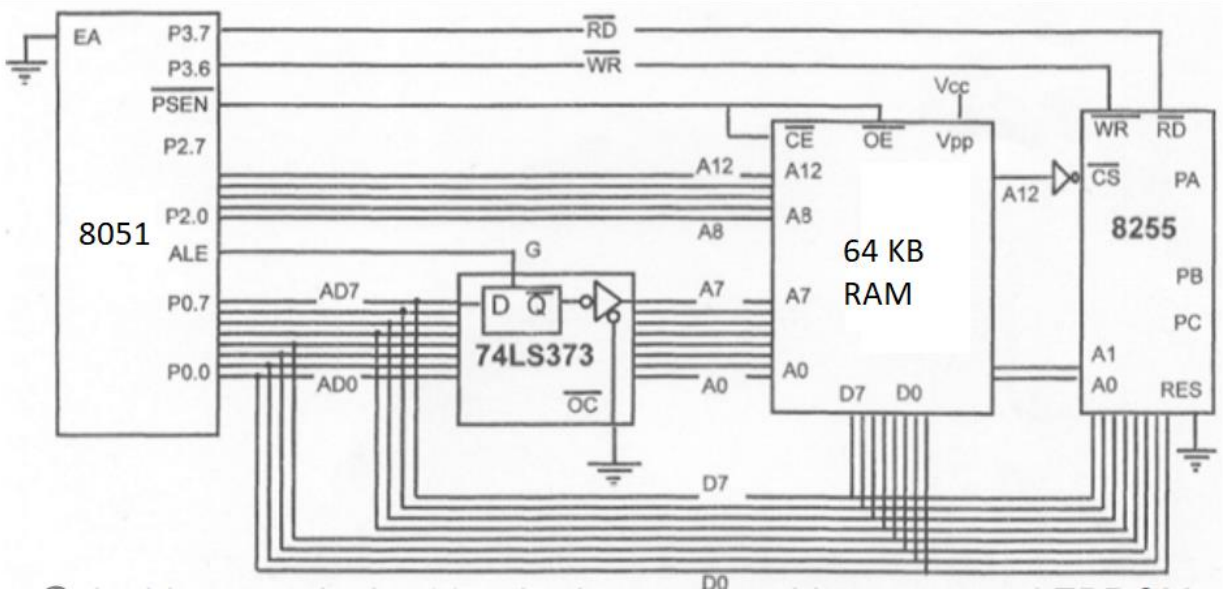
MOVX @DPTR, A (**writing data** to external data RAM)

```
            RAMDATA     EQU     5000H
            COUNT EQU 200

            MOV DPTR,#RAMDATA
            MOV R3,#COUNT

AGAIN:      MOV A,P1
```

```
        MOVX @DPTR,A
        ACALL DELAY
        INC DPTR

        DJNZ R3,AGAIN
HERE:   SJMP HERE
```

ALE(Address Latch Enable) pin is an output for 8051

- **ALE = 0,** P0 is used for **data path**.
  - It is used as a data bus, sending data out or bringing data in
- **ALE = 1,** P0 is used for **address path**
  - It puts the addresses A0-A7 on the P0 pins and activates ALE = 1



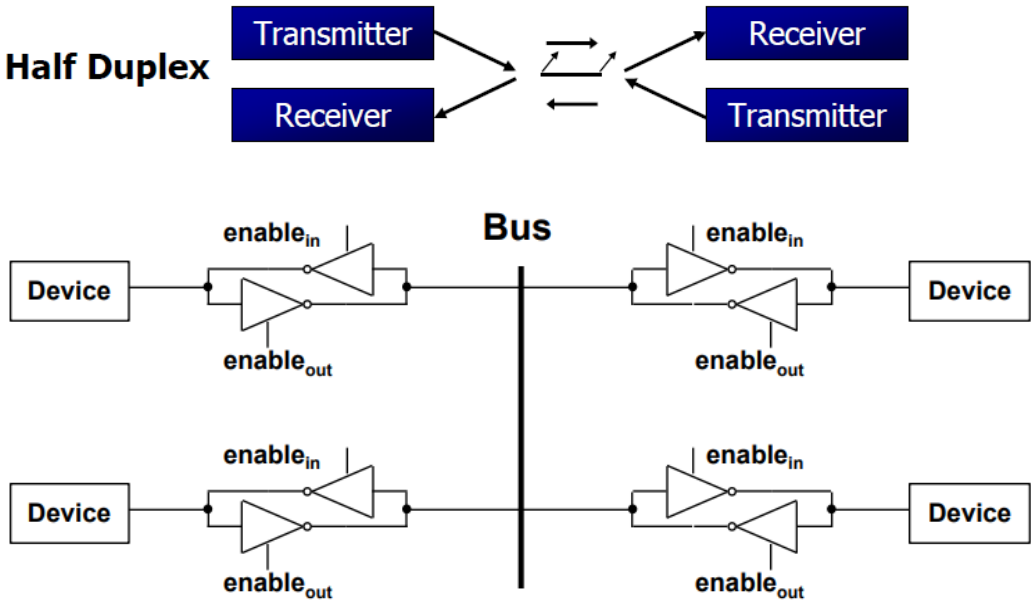
(b) Briefly explain what is meant by a bus and why it is necessary to control access to it. You are tasked with designing a means of interfacing devices via a half-duplex bus. Using an appropriate illustration explain how this can be done using buffering to avoid contention.

1. A bus is a collection of wires through which data is transmitted from one part of a computer system to another. It is an essential part of a computer System, and consist of 3 parts:
   - **Address bus:** transfer information about where the data should go
   - **Data bus:** transfer actual data
   - **Control Signals:** determines when a device should read the data and so forth.

   The size of a bus **(width)** determines how much data can be transmitted at one time.

2. It is necessary to control access to the bus since there can be multiple transmitters on a bus in order to coordinate when each entity can **have write access** to the bus. **Tri-state logic** does it and also **prevents a bus conflict**.

3.



Half-Duplex bus: data is transmitted one way a time, the direction can be changed. This type of communication can be used in some measurement systems and having the buffers will prevent bus conflict.