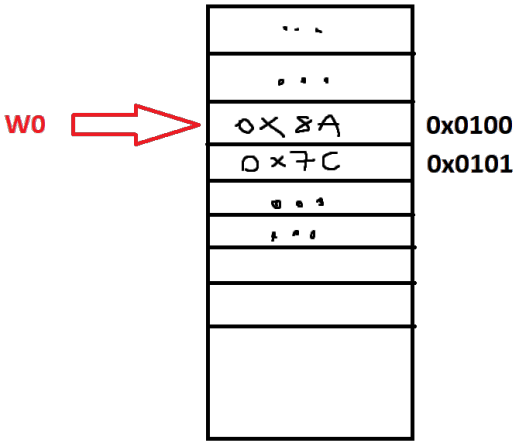
EE308 HOMEWORK ASSIGNMENT #3

AYBARS YAZICI 25330

0.1. **Question Q1.** We know that the content of W0=0x0100 and W1=0xEA1C, while the memory location 0x0100 contains 0x7C8A,



we execute the following code:

```
1       mov.b       [++W0], W1
```

We notice that we do a Pre-incremental operation on W0, this means that before we execute the move operation we will increment the content of W0, that means we will be executing the following code:

```
1       mov.b       0x0101, W1
```

Thus after the execution of this mov.b operation, W1 will contain EA7C instead of EA1C, as we are moving a single byte.

0.2. **Question Q2.** The 3 lines of code given in the question are as follows:

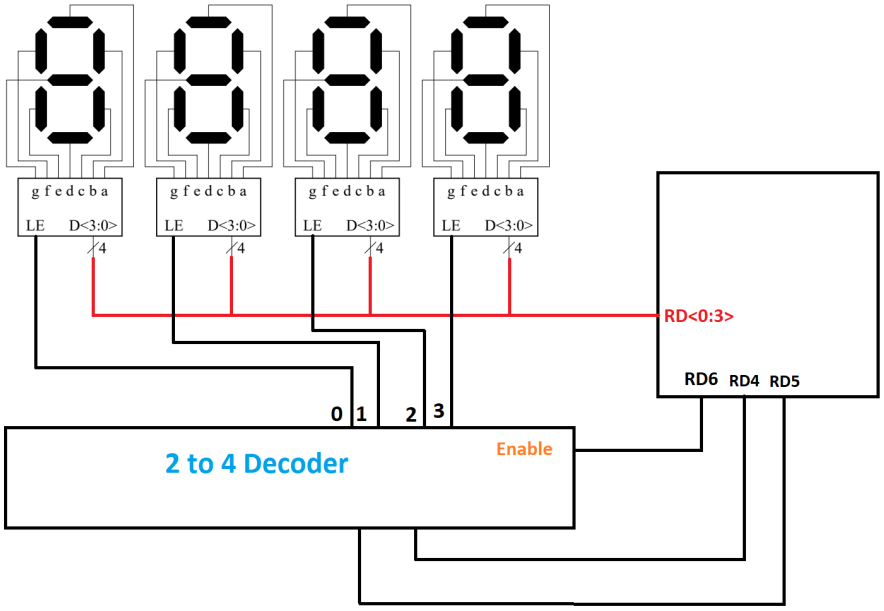
```
1     repeat      #3
2     add         W0, W0, W0
3     bra         C, label1
```

We know that repeat #3 means that the following operation that's on the next line will execute (3+1=4) many times. W0 initially contains 0x1000. Let's trace each execution of the add operation.

(1) We add 0x1000 to 0x1000 and store it in W0. W0 now contains 0x2000
(2) We add 0x2000 to 0x2000 and store it in W0. W0 now contains 0x4000
(3) We add 0x4000 to 0x4000 and store it in W0. W0 now contains 0x8000
(4) We add 0x8000 to 0x8000 and store it in W0. W0 now contains 0x0000

As the last operation will produce a carry out of the most significant bits added. The carry flag will be set, because of this the branch operation will execute, and program counter will be loaded with the label1's address.

0.3. **Question Q3.** We are asked to interface the SSD and their decoders to RD<6:0>while utilizing the given 2-to-4 decoder with enable input. Knowing that the SSD decoders require a 4 bit input, that is, D<3:0>, we can map them to RD<3:0>, so we can use the keyboard as an input device to the SSD's, we can use RD6 as an enable signal, while using the remaining two bits RD4 and RD5 as a way to choose which SSD to enable... So we get something as follows:



Now on the following part we are required write a simple function that displays the Hour and Minute variables on the SSDs. Knowing that Hour and Minute variables are in packed BCD format, where the ones-digit is in the 4 lower bits, and tens-digit is in upper 4 we can do the following:

```
displayOnSSD:
    push W0
    push W1
    mov HOUR, W0
    mov HOUR, W1
    lsr W0, #4, W0; Shift W0 4 bits to the right
    and #0xFF9F, W1 ,PORTD ;This clears Bit #6 and #5 while keeping the rest of W1
    bset PORTD #4; We set the Bit #4
    ;Thanks to these two operations, the decoder will select the output-1.
    bset PORTD, #6 ;Set the data available signal to enable the decoder
    ;And PORTDs Lower 4 bits contain the ones-digit of the hour variable in BCD format
    ;At this point we have already displayed the ones-digit of the hour variable.
    nop ;wait a bit for the SSD to finish its job.
    bclr PORTD #6
    ;Now to display the tens-digit, we do the following
    and #0xFF8F, W0 ,PORTD; This clears Bit #4,#5 and #6, keeps the rest of W0
    bset PORTD #6
    ;The decoder selects output 0.
    ;Now we have displayed the tens digit of the hour variable
    nop
    bclr PORTD #6
    mov MINUTE, W0
    mov MINUTE, W1
    lsr W0, #4, W0; Shift W0 4 bits to the right
    and #0xFFBF, W1, PORTD ;This clears Bit #6 while keeping the rest of W1 the same
    bset PORTD #4
    bset PORTD #5
    ;Now the decoder will select output-3;
    bset PORTD #6
    ;We have now displayed the ones-digit of the minute variable
    nop
    bclr PORTD #6
    and #0xFFAF, W0, PORTD; This clears Bit #6 and Bit #4 while keeping the rest of W0
    bset PORTD #5;Decoder will now choose output-2
    bset PORTD #6;We enable the decoder
    nop
    bclr PORTD #6
    pop W1
    pop W0
    return
```

Normally we need to loop this function to trick the human eye that all the SSD's are on. Because what we are doing is enabling an SSD, writing to it, then disabling it and enabling another one and writing to it. Then do this for all the SSDs. If we do this operation once, we as the user will only see the last SSD being open, as thats the last one we enable.

But if we loop this function continously as the CPU is really fast, we as the user will not even see SSD's switching on and off continously but we will be seeing them as ON all the time.

So what we can do is the following:

```
    infiniteLoop:
    call displayOnSSD
    bra infiniteLoop
```

We can also increment HOUR and MINUTE variables here, in this loop, or even better we can use a timer interrupt, to have the minute variable increment every 60 seconds, and hour variable to increment every 60 minutes. We can put an exit condition as well if necessary to go out of this infinite loop. But this is the basic idea of displaying the HOUR and MINUTE variables, that are in packed BCD format, on the SSD.

0.4. **Question Q4.** We asked to write a code fragment which modifies the Bits 4 until 11(included) of TRISA while not modifying the rest. Note that IOR'ing with 1 will set the bit to 1, while IOR'ing 0 will not modify the bit. Thus the code fragment would be as follows:

```
    push W0
    push W1
    mov TRISA, W1 ;Read TRISA into W1
    mov 0x0FF0 , W0
    ior W0, W1 , W1 ;inclusive or W0 and W1, store it in W1
    mov W1, TRISA ;Store the info back to TRISA
    pop W1
    pop W0
```

0.5. **Question Q5.** Let's say we want to calculate the factorial of number 3, which is stored in W0,

```
    mov #3, W0 ;Store the number of which we want to calculate the factorial of to W0.
    push W1 ;Save W1 the stack as we will be using it for the factorial calculation.
    mov W0, W1
    call factorial
    pop  W1 ;Don't forget to store the content of W1 back after we are done.
```

Let's take a look at the factorial function we have:

```
    factorial:
        cp W1, #1 ;Is W1 equal to 1 ?
        bra NZ, calculate ;If it is not equal to 1 goto the calculation part
        return ;If it is equal to 1, return from the function.
        calculate:
        dec W1,W1 ;We came here because W1 was NOT equal to 1. Thus decrement it by one
        mul.ss W0,W1,W0 ;Multiply it with W0 and store the new result in W0.
        call factorial ; Call the function again.
```

0.6. **Question Q6.** We are asked to write a C function that displays a digital clock on PicSIMLab, First the main.c will be as follows:

```
    void main(void) {
    TRISD = 0x00;
    TRISA = 0x00;
    PORTD = 0b00111111;
    LATA =  0b00111100; //Clock displays 00:00 at the start
    while(1){
        if(PORTBbits.RB5==0) //Has the button been pressed?
        {
            showOnSSD(0,0); //If yes, start the clock from HOUR = 0, MINUTE = 0;
            while(PORTBbits.RB5==0){} //Wait Release
        }
    }
    return;
}
```

When we press RB5 the clock will start from the given parameters, which are 0 and 0. Let's take a look at the showOnSSD function:

```
#define _XTAL_FREQ 2000000
int table[10] = {0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111101,0b00000111,0b01111111,0b01101111};
//This is an integer array representing the SSD codes for the digits ranging from 0 till 9(inclusive)
void showOnSSD(int HOUR, int MINUTES)
{
    TRISD = 0x00; //PORTD All output
    TRISA = 0x00; //PORTA All outputa
    //As the same button, RB5, will be used as a way to stop the digital clock
    while(PORTBbits.RB5==0){} //We need to make sure if the user has released the key before starting.
    int onesDigit;
    int tensDigit;
    restart:;
    int cnt = 0;
    while(1)
    {
        onesDigit = HOUR%10;
        tensDigit = HOUR/10;
        LATA =  0b00000100;
        LATD = table[tensDigit];
        __delay_ms(1.5);
        LATA =  0b00001000;
        LATD = table[onesDigit];
        onesDigit = MINUTES%10;
        tensDigit = MINUTES/10;
        LATA =  0b00010000;
        LATD = table[tensDigit];
        __delay_ms(1.5);
        LATA =  0b00100000;
        LATD = table[onesDigit];
        cnt++;
        if(cnt==295){ //This way the clock will increment it's minute every sec
            MINUTES++; //at CPU=4MHz,we can modify this value,to change Clock's speed.
            if(MINUTES == 60) //Set it to 2950, and it will increment every sec
            { //At 40Mhz
                MINUTES = 0;
                HOUR++;
                if(HOUR == 24)
                {
                    HOUR = 0;
                }
            }
            goto restart;
        }
        if(PORTBbits.RB5==0)
        {
            PORTD = 0b00111111;
            LATA =  0b00111100;
            return;
        }
    }
}
```

Note that we need the delays to have the first and third SSDs stay on as much as the 2nd and 4th. Because we do some calculations before displaying the number on first and third SSD, they will stay on for shorter amount of period compared to second and fourth SSDs, thus we introduce a small amount of delay, just to keep them on for roughly the same amount of time.

As we have talked about in the 3rd question, it is possible to use the timer interrupt to have increase the minute and hour variables. Our main function will not change. Let's take a look at our show on SSD function:

```
#define _XTAL_FREQ 2000000
int table[10] = {0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0
    b01111101,0b00000111,0b01111111,0b01101111};

unsigned int counter = 0;
unsigned int HOUR = 0, MINUTES = 0;

void showOnSSD(int hours, int minutes)
{
    HOUR = hours;
    MINUTES = minutes;
    TRISD = 0x00;
    TRISA = 0x00;
    while(PORTBbits.RB5==0){} //Wait Release
    int onesDigit;
    int tensDigit;
    while(1)
    {
        onesDigit = HOUR%10;
        tensDigit = HOUR/10;
        LATA =  0b00000100;
        LATD = table[tensDigit];
        __delay_ms(0.1);
        LATA =  0b00001000;
        LATD = table[onesDigit];
        onesDigit = MINUTES%10;
        tensDigit = MINUTES/10;
        LATA =  0b00010000;
        LATD = table[tensDigit];
        __delay_ms(0.1);
        LATA =  0b00100000;
        LATD = table[onesDigit];
        if(PORTBbits.RB5==0)
        {
            PORTD = 0b00111111;
            LATA =  0b00111100;
            return;
        }
    }
}
```

And our interrupt service routine:

```
void __interrupt(high_priority) tcInt(void)
{
    if (TMR0IE && TMR0IF)
    {
        TMR0IF=0;
        counter++;
        if(counter==140)
        {
            MINUTES++;
            if(MINUTES == 60)
            {
                HOUR = (HOUR+1)%24;
                MINUTES = 0;
            }
            counter = 0;
        }
    }
    return;
}
```

Note that in timer0's initilization I am using a prescaler of 256, a crystal clock of 20Mhz, thus the above code in PicSIMLab will increment itself every second at CPU = 40Mhz. But if we change the if(counter==140) to around if(counter==8700) it should increment itself every minute, at 40Mhz.