Θεόδωρος Μπάρκας 2016030050 LAB4

Σκοπός άσκησης

Στη συγκεκριμένη άσκηση σκοπός ήταν η εξοικείωση με τη γλωσσά C μεσώ της μετατροπής του κωδικά μας από μόνο Assembly σε ένα συνδυασμό των δυο .Συγκεριμένα τα Interrupts είναι γραμμένα σε Assembly και ο υπόλοιπος κώδικας σε C.

Υλοποίηση C και Initialization

Αρχικά αρχικοποιήσαμε την μνήμη ως εξής :

```
void INIT(){
    //memory map for data
    unsigned char *displayNumbers=(unsigned char *) 0x60;
    displayNumbers[0]=(unsigned char)0x05;
    displayNumbers[1]=(unsigned char)0x06;
    displayNumbers[2]=(unsigned char)0x07;
    displayNumbers[3]=(unsigned char)0x08;
    displayNumbers[4]=(unsigned char)0x09;
    displayNumbers[5]=(unsigned char)0x00;
    displayNumbers[6]=(unsigned char)0x01;
    displayNumbers[7]=(unsigned char)0x02;
    unsigned char *seven_segment_decoder=(unsigned char *) 0x68;
    seven_segment_decoder[0]=(unsigned char)0b00000001; //0
    seven_segment_decoder[1]=(unsigned char)0b01001111; //1
    seven_segment_decoder[2]=(unsigned char)0b00010010; //2
    seven_segment_decoder[3]=(unsigned char)0b00000110; //3
    seven_segment_decoder[4]=(unsigned char)0b01001100; //4
    seven_segment_decoder[5]=(unsigned char)0b00100100; //5
    seven_segment_decoder[6]=(unsigned char)0b00100000; //6
    seven_segment_decoder[7]=(unsigned char)0b00001111; //7
    seven_segment_decoder[8]=(unsigned char)0b00000000; //8
    seven_segment_decoder[9]=(unsigned char)0b00000100; //9
    seven_segment_decoder[10]=(unsigned char)0b11111111; //A
    unsigned char *curNumberIndex=(unsigned char *) 0x73; //r17 value
    *curNumberIndex=(unsigned char)0x07;
    unsigned char *ringCounter=(unsigned char *) 0x74; //r17 value
    *ringCounter=(unsigned char)0x80;
```

Εδώ βλέπουμε ότι αρχίζοντας από την αρχή της Ram έχουμε δεσμεύσει 8 Bytes για τους αριθμούς που προβάλουμε στον τελικό χρήστη,11 Bytes εκ των οποίων 10 για το δεκαδικό σύστημα και ένα για την κενή οθόνη(δηλαδή χωρίς κανένα Segment ανοιχτό).
Εν συνεχεία πραγματοποιήσαμε αλλαγές δημιουργώντας δυο νέες θέσεις μνήμης οι οποίες θα μας βοηθήσουν να βγάλουμε τους χαμένους καταχωρημένες r17,r19 από την τωρινή μας υλοποίηση. Οι τιμές που περιέχουν αυτοί είναι:

- ο curNumberIndex αριθμός που αν προστεθεί στην αρχή της Ram θα μας δώσει τη θέση του ψηφίου που προβάλουμε την εκάστοτε στιγμή ,μέχρι στιγμής δέσμευε τον καταχωρητή r17.
- Ο ringCounter οπού διώχνει κάθε στιγμή την άνοδο που ενεργοποιούμε για τη χρονική πολυπάθεια, μέχρι στιγμής δέσμευε τον καταχωρητή r19.

Αυτοί οι δυο καταχωρητές χρειάζονταν στο Interrupt του timerOvf κατά το οποίο πραγματοποιείτε η πολυπλεξία στο χρόνο ,επομένως αρχικοποιούνται στην αρχή του στον κωδικά Assembly:

```
#define curNumberIndex 0x73;
```

```
#include <avr/io.h>
#define curNumberIndex 0x73;
.global ISR_timerOVF
    ISR_timerOVF: ;Interrrupt Handler Timer OVF

        ldi XH, hi8(RAMSTART);New Code For C compatibility so that anyone can mess up with X in main
        ldi XL ,curNumberIndex
        ld r17,X+
        ld r19,X
```

Ύστερα και γυρνώντας στην αρχικοποίηση στο κωδικά C , αρχικοποιήσαμε τα Data Direction για τα Pins/Ports σε αυτά παρατηρήθηκε ένα λάθος από προηγούμενα εργαστήρια που μου είχε περάσει απαρατήρητο. Συγκεκριμένα αντί να χρησιμοποιήσω το PORTC χρησιμοποιούσα το PORTB ,πλέον έχει διορθωθεί.
Έκτος αυτού καλούμε την `USART_Init(ubrr);` και την `TIMER0_Init();` καθώς και ενεργοποιούμε τα Interrupts με την εντολή sei();

```
    //masks for ports a and c
    //i just realise in the past labs i use port B as my port A
    //so i fixed this in this one
    DDRC=(unsigned char) 0xff;
    DDRA=(unsigned char)0b01111111;
    asm("nop");


    unsigned int ubrr= (unsigned int)UBRR;

    USART_Init(ubrr);
    TIMER0_Init();
    sei();
    return;
}
```

USART_INIT() comments:
Αρχικοποιούμε ton CurState ολοκληρώνοντας το Memory Map μας.
USART_TRANSMIT / USART_TRANSMIT_OK comments
Η USART_TRANSMIT καλείται από της Assembly με rcall και η υλοποίηση της είναι σε C.

```
void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
TCNT2 = data;
}

void USART_Transmit_OK( unsigned char data )
{
USART_Transmit((unsigned char)0x4F); //O
USART_Transmit((unsigned char)0x4B); //K
USART_Transmit((unsigned char)0x0D); //<CR>
USART_Transmit((unsigned char)0x0A); //<LF>
}
```

Σχόλια και αλλαγές σε κωδικά Assembly

Όπως είδαμε παραπάνω στην αρχή του κωδικά περνούμε τις τιμές r17,r19 στο τέλος όμως για να είμαστε συνεπείς με την υλοποίηση μας πρέπει να θέσουμε τις καινούργιες τιμές των καταχωρητών στις εκάστοτε θέσεις μνήμης :

```
ldi XL ,curNumberIndex
st X+,r17
st X,r19
```

Αλλάζουμε την διεύθυνση του ControlUsart για να αντιπροσωπεύει το τωρινό Memory Map

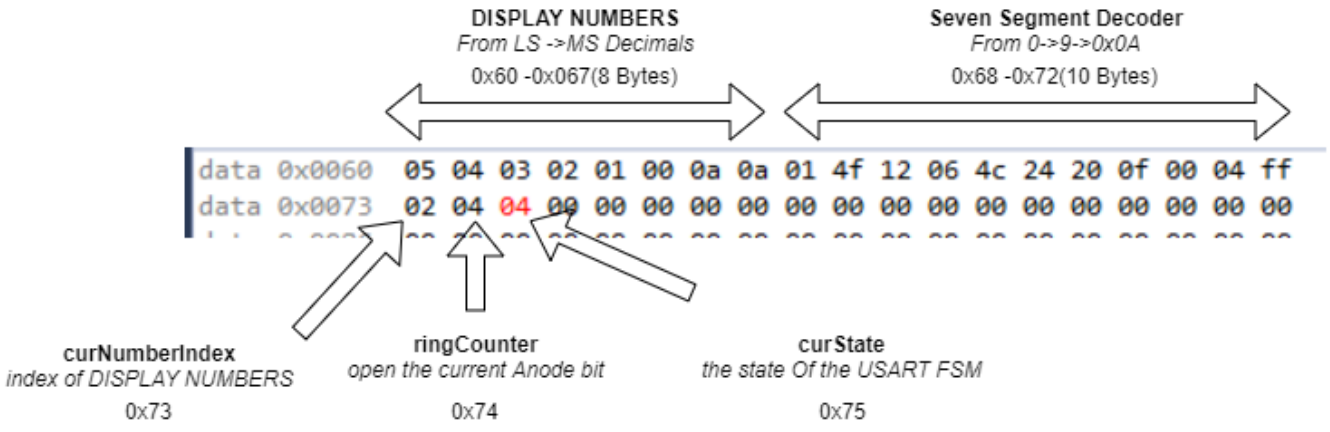
```
.global ISR_USART_Receive
;------------------------
;Receive in r18 the value Of UDR
ISR_USART_Receive:
;STATE MACHINE
 #define INITstate 0x00;
 #define TWAITstate 0x01;
 #define BYTEOUTstate 0x02;
 #define CRWAITstate 0x03;
 #define LFWAITstate 0x04;

 #define ControlUsart 0x75;
```

Επίσης έβαλα τον κωδικά εσωτερικά αποφεύγοντας συνάρτησεις (rcall) και χρήση jump εκτός του interrupt γενικότερα , όπως αναφέρετε στην εκφώνηση.
Τέλος όταν έρχεται <LF> την σωστή στιγμή (στο `LFWAITstate`) καλούμε την συνάρτηση σε C :

```
rcall USART_Transmit_OK
```

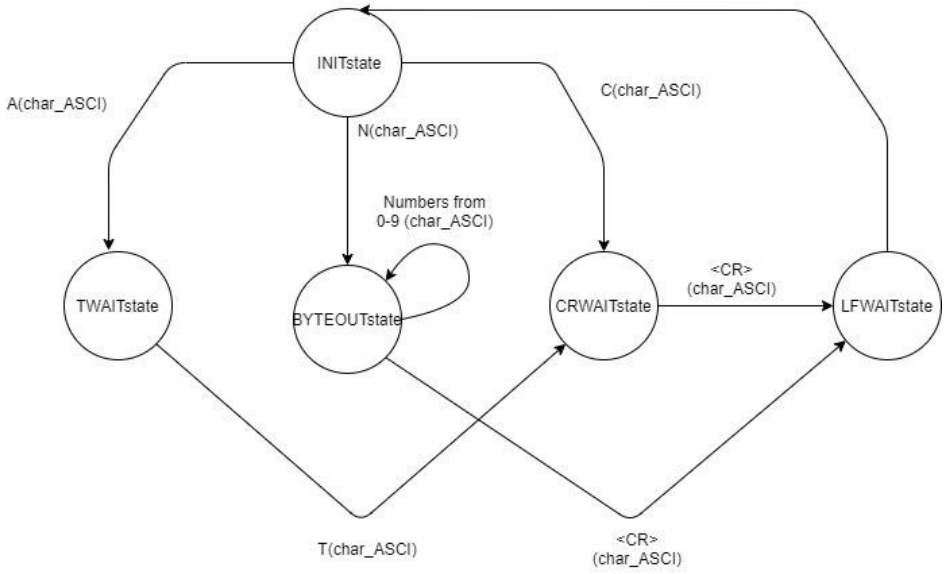
η οποία λειτουργεί με Polling και αναφέρθηκε παραπάνω.

MEMORY MAP



FSM

Υπενθυμίζουμε για μια ακόμα φορά την FSM



TESTING

Τρέχοντας το Simulation παρατηρούμε ακριβός ίδια συμπεριφορά με το προηγούμενο εργαστήριο .