ΕΜΠ - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2021-2022

ΑΘΗΝΑ 31-10-2021

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ ''Εργαστήριο Μικροϋπολογιστών''
2η Εργ. Άσκ. στον Μικροελεγκτή AVR – Λογικές Πράξεις / Χρήση εξωτερικών διακοπών
(υλοποίηση στον προσομοιωτή Atmel Studio)

Ομάδα 41
Σιόκουρου Αιμιλία – 03117703
Πεγειώτη Νάταλυ – 03117707

Ζήτημα 2.1

1η Λύση

Για την υλοποίηση του προγράμματος υλοποιήσαμε τις ακόλουθες λογικές πράξεις:

F0 = (A'B + B'CD)'

F1 = AC(B+D)

Σχηματίσαμε τον ακόλουθο κώδικα σε assembly, ο οποίος εξηγείται αναλυτικά με σχόλια:

```
.include "m16def.inc"
.DEF input=r22
.DEF output=r23
.DEF inputC=r15
.DEF F0=r20
.DEF F1=r21
.DEF A=r16
.DEF B=r17
.DEF C=r18
.DEF D=r19
.DEF temp1=r12
.DEF temp2=r13
.DEF ace=r14

reset: ldi r24, low(RAMEND)        ; initialize stack pointer
       out SPL, r24
       ldi r24, high(RAMEND)
       out SPH, r24

       clr input                   ; initialize PORTC for input
       out DDRC , input
       ser output                  ; initialize PORTB for output
       out DDRB , output

main:  in inputC, PINC
       mov A, inputC          ; A <- 0000 DCBA, load number
       andi A, 1              ; A <- 0000 DCBA and 0000 0001 = 0000 000A
       mov B, inputC          ; B <- 0000 DCBA, load number
       andi B, 2              ; B <- 0000 DCBA and 0000 0010 = 0000 00B0
       lsr B                  ; right shift => B <- 0000 000B
       mov C, inputC          ; C <- 0000 DCBA, load number
       andi C, 4              ; C <- 0000 DCBA and 0000 0100 = 0000 0C00
       lsr C                  ; right shift => C <- 0000 00C0
       lsr C                  ; right shift => C <- 0000 000C
       mov D, inputC          ; D <- 0000 DCBA, load number
```

```
        andi D, 8           ; D <- 0000 DCBA and 0000 1000 = 0000 D000
        lsr D               ; right shift => D <- 0000 0D00
        lsr D               ; right shift => D <- 0000 00D0
        lsr D               ; right shift => D <- 0000 000D

        clr ace
        inc ace

F0_lbl:     mov temp1, ace      ; temp1 = 1
            eor temp1, A        ; 1 xor A => temp1 = A'
            and temp1, B        ; temp1 and B => temp1 = A'B

            mov temp2, ace      ; temp2 = 1
            eor temp2, B        ; 1 xor B => temp2 = B'
            and temp2, C        ; temp2 and C => temp2 = B'C
            and temp2, D        ; temp2 and D => temp2 = B'CD

            mov F0, temp1       ; F0 = temp1 = A'B
            or F0, temp2        ; F0 add temp2 => F0 = A'B + B'CD
            eor F0, ace         ; F0 xor 1 => F0' = (A'B + B'CD)'

F1_lbl:     mov temp1, A        ; temp1 = A
            and temp1, C        ; temp1 and C => temp1 = AC

            mov temp2, B        ; temp2 = B
            or temp2, D         ; temp2 add D => temp2 = B+D

            mov F1, temp1       ; F1 = temp1 = AC
            and F1, temp2       ; F1 and temp2 => F1 = AC(B+D)


RES:    lsl F1              ; (F1) <- 0000 00(F1)0  (shift)
        add F1, F0          ; (F1) <- 0000 00(F1)(F0)
        out PORTB, F1
        rjmp main           ; start from the beginning
```

Με τον ίδιο τρόπο υλοποιήσαμε το πρόγραμμα στη γλώσσα C:

```
#include <avr/io.h>

int main(void)
{
        char x,A,B,C,D,F0,F1;
        DDRB=0xFF; // use PORTB as output
        DDRC=0x00; // use PORTC as input
        while (1)
        {
                x = PINC & 0x0F; //isolation of the 4 LSBs
                A = x & 0x01; //A is PC0
                B = x & 0x02; //B is PC1
                B = B>>1;
                C = x & 0x04; //C is PC2
                C = C>>2;
                D = x & 0x08; //D is PC3
                D = D>>3;
                F0 = !((!A && B ) || (!B && C && D)); //calculation of F0
                F1 = ((A && C) && (B || D)); //calculation of F1
                F1 = F1<<1;
                PORTB = (F0 | F1); //output
        }
}
```

2η Λύση (εναλλακτική)

Για την υλοποίηση του προγράμματος υπολογίσαμε τα ακόλουθα:

→ F0 = (A'B + B'CD)', επομένως:
F0 = 0 ⇔ A'B + B'CD = 1 ⇔ A'B = 1 or B'CD = 1 ⇔ (A,B) = (0,1) or (B,C,D) = (0,1,1)
F0 = 1 , σε όλες τις υπόλοιπες περιπτώσεις

→ F1 = AC(B+D) , οπότε
F1 = 0 ⇔ AC(B+D) = 0 ⇔ A = 0 or C = 0 or B+D = 0 ⇔ A = 0 or C = 0 or (B,D) = (0,0)
F1 = 0 , σε όλες τις υπόλοιπες περιπτώσεις

Ακολουθώντας όλες τις παραπάνω παρατηρήσεις σχηματίζουμε τον ακόλουθο κώδικα assembly:

```
.include "m16def.inc"
.DEF input=r22
.DEF output=r23
.DEF inputC=r15
.DEF F0=r20
.DEF F1=r21

reset: ldi r24, low(RAMEND)      ; initialize stack pointer
       out SPL, r24
       ldi r24, high(RAMEND)
       out SPH, r24
       clr input           ; initialize PORTC for input
       out DDRC , input
       ser output          ; initialize PORTB for output
       out DDRB , output

main:  in inputC, PINC

F0_lbl:      mov r16, inputC     ; (r16) <- 0000 DCBA, load number
             andi r16, 3         ; (r16) <- 0000 DCBA and 0000 0011 = 0000 00BA
             cpi r16,2           ; compare 0000 00BA to 0000 0010
             breq F0_zero        ; if (A'B=1 <=> ((r16) == 0000 0010)) => F0=0 =>
                                 jump to F0_zero
             mov r16, inputC     ; (r16) <- 0000 DCBA, load number again
             andi r16,14         ; (r16) <- 0000 DCBA and 0000 1110 = 0000 DCB0
             cpi r16,12          ; compare 0000 DCB0 to 0000 1100
             breq F0_zero        ; if (B'CD=1 <=> ((r16) == 0000 1100)) => F0=0 =>
                                 jump to F0_zero
             rjmp F0_ace         ; else F0=1 => jump to F0_ace
F0_zero:     ldi F0, 0
             rjmp F1_lbl
F0_ace:      ldi F0, 1
             rjmp F1_lbl


F1_lbl:      mov r17, inputC     ; (r17) <- 0000 DCBA, load number
             andi r17, 1         ; (r17) <- 0000 DCBA and 0000 0001 = 0000 000A
             cpi r17, 0          ; compare 0000 000A to 0000 0000
             breq F1_zero        ; if (A=0 <=> ((r17) == 0000 0000)) => F1=0 =>
                                 jump to F1_zero
             mov r17, inputC     ; (r17) <- 0000 DCBA, load number again
             andi r17, 4         ; (r17) <- 0000 DCBA and 0000 0100 = 0000 0C00
```

```
            cpi r17, 0          ; compare 0000 0C00 to 0000 0000
            breq F1_zero        ; if (C=0 <=> ((r17) == 0000 0000)) => F1=0 =>
                                jump to F1_zero
            mov r17, inputC     ; (r17) <- 0000 DCBA, load number again
            andi r17, 10        ; (r17) <- 0000 DCBA and 0000 1010 = 0000 D0B0
            cpi r17, 0          ; compare 0000 D0B0 to 0000 0000
            breq F1_zero        ; if (B+D=0 <=> ((r17) == 0000 0000)) => F1=0 =>
                                jump to F1_zero
            rjmp F1_ace         ; else F1=1 => jump to F1_ace
F1_zero:    ldi F1, 0
            rjmp RES
F1_ace:     ldi F1, 1
            rjmp RES


RES:  lsl F1                    ; (F1) <- 0000 00(F1)0  (shift)
      add F1, F0                ; (F1) <- 0000 00(F1)(F0)
      out PORTB, F1
      rjmp main                 ; start from the beginning
```

Με τους ίδιους υπολογισμούς υλοποιήσαμε το πρόγραμμα στη γλώσσα C:

```
#include <avr/io.h>

char f0 , f1 ;

int main ( void )
{
      DDRB = 0xFF; // Initialize PORTB as output
      DDRC = 0x00; // Initialize PORTC as input

      while (1) {
            // F0
            if ((( PINC & 0x03 )==2) || (( PINC & 0x0e )==12)) {
                  f0 = 0;
            }
            else {
                  f0 = 1;
            }
            // F1
            if (((PINC & 0x01)==0) || ((PINC & 0x04)==0) || ((PINC & 0x0A)==0)) {
                  f1 = 0;
            }
            else {
                  f1 = 1;
            }

            f1 = f1 << 1;
            PORTB = f0 | f1 ; // Output in PORTB
      }
      return 0;
}
```

Ζήτημα 2.2

Για την υλοποίηση της άσκησης αυτής χρησιμοποιήσαμε τον σκελετό του παραδείγματος, τον οποίο και διαμορφώσαμε κατάλληλα.
Στο ζητούμενο ερώτημα η διακοπή που χρησιμοποιείται είναι η INT1 στην ανερχόμενη ακμή, επομένως ρυθμίσαμε κατάλληλα τις μάσκες.
Ο αριθμός των διακοπών καταμετράται στη μεταβλητή intercount. Όταν γίνει διακοπή (PIND3 on) και εφόσον τα bits 6 και 7 της θύρας A είναι σε θέση on, ο μετρητής αυξάνεται κατά ένα και το περιεχόμενο του εμφανίζεται στην έξοδο της θύρας B.

Ο κώδικας για την υλοποίηση του προγράμματος είναι ο ακόλουθος, ο οποίος επεξηγείται με αναλυτικά σχόλια:

```
.include "m16def.inc"

.def intercount = r16     ;interrupts counter
.def temp = r17
.def count = r26

.org 0x0     ; the main program (reset) starts
rjmp reset   ; in address 0x0
.org 0x4     ; the interrupt INT1 routine starts
rjmp ISR1    ; in address 0x4

reset: ldi r20, low(RAMEND)      ; initialise stack pointer
       out SPL, r20
       ldi r20, high(RAMEND)
       out SPH, r20

       ldi r24,(1 << ISC11) | (1 << ISC10)    ;set interrupt on rising edge
       out MCUCR, r24
       ldi r24,(1 << INT1)             ;set INT1
       out GICR, r24
       sei

       ser r26             ; initialise  PORTB, PORTC for output
       out DDRC, r26
       out DDRB, r26
       clr r26             ; initialise  PORTA for input
       out DDRA, r26

loop:  out PORTC , count         ; display counter on LEDs
       ;ldi r24 , low(100)       ; load r25:r24 with 100
       ;ldi r25 , high(100)      ; delay 100 ms
       ;rcall wait_msec          ; cannot be used on the simulator
       inc count                 ; increase counter
       rjmp loop                 ; repeat

ISR1:  in temp, PINA             ; read input from PINA
       andi temp,0b11000000      ; isolate bits 6 and 7
       cpi temp,0b11000000       ; if bit 6 or 7 is off then skip
       brne loop
       inc intercount            ; increase counter of interrupts
       out PORTB, intercount     ; display counter of interrupts on PORTB
       ;ldi r24 , low(998)       ; load r25:r24 with 980
       ;ldi r25 , high(998)      ; delay 1sec
       ;rcall wait_msec          ; cannot be used on the simulator
       reti
```

Ζήτημα 2.3

Για την υλοποίηση του ζητούμενου προβλήματος πρέπει να χρησιμοποιήσουμε εξωτερικές διακοπές γι' αυτό χρησιμοποιούμε και τη βιβλιοθήκη "<avr/interrupt.h> " εκτός από την "<avr/io.h> ".
Για την ενεργοποίηση του interrupt0, αρχικοποιούνται κατάλληλα οι σημαίες.
Εντός της ρουτίνας εξυπηρέτησης της διακοπής γίνεται ο έλεγχος του διακόπτη PA2 και ανάλογα με το αν είναι ON ή OFF εκτελείται η αντίστοιχη διαδικασία ώστε να παραχθεί η ζητούμενη έξοδος και να τοποθετηθεί στα leds PC7 - PC0.

Ο κώδικας για την υλοποίηση του προγράμματος είναι ο ακόλουθος, ο οποίος επεξηγείται με αναλυτικά σχόλια:

```
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char a,b,temp,count,output;    //a->PORTA, b->PORTB

ISR (INT0_vect)
{
	int i;
	int output;
	output=0x00;
	a=PINA & 0x04;		//isolate the 2nd lsb for PA2
	a=a>>2;			//shift right 2 positions to get the value in the LSB
	b=PINB;
	for(i=0; i<8; i++)
	{
		temp=b;
		b=b & 0x01;  //isolate the lsb
		if(b==1)
		{
			count++;	//counter for dip switches ON
		}
		b=temp;			//restore number
		b=b>>1;			//right shift to check the next LSB dip switch
	}
	if(a==0)
	{
		while(count > 0)
		{
			output=output+1;
			output=output<<1;
			count=count-1;
		}
		PORTC=output>>1;
	}
	else
	{
		PORTC=count;
	}
	return;
}
```

```c
int main(void)
{
	DDRA=0x00;	//set A and B as inputs
	DDRB=0x00;
	DDRC=0xFF;	//set PORTC as output
	GICR=0x40;	//INT0 ON
	MCUCR=0x03;	//INT0 MODE: FALL EDGE
	sei();		//Enable global interrupts
			by setting global interrupt enable bit in SREG

	while (1)
	{
		asm("nop");
	}
}
```