Εργαστήριο Μικροεπεξεργαστών

2^η Εργαστηριακή άσκηση

Γέροντας Αλέξανδρος 321 - 2015029

Εργασία 1

Συμπληρώνουμε τον κώδικα ως εξής:

```
main2.asm* → × main.asm
                             AssemblerApplication1
     1
         .cseg
         .ORG 0x0000
        start:
     6
            ldi r17, 0x80 ; initialization
     7
            ldi r16, 0 ; i = 0
     9 next:
    10
            cpi r16, 4 ; if i >= 4...
    11
    12
            brcc exit; ...exit loop
            lsr r17 ; r17 = r17 / 2
    13
            inc r16 ; i++
    14
             rjmp next ; ...goto next
    16 exit:
```

Τρέχουμε το πρόγραμμα και μετά από την έξοδο από τον βρόγχο βλέπουμε τις τελικές τιμές των καταχωρητών r16 και r17 στο δεκαδικό:

```
R16 4
R17 8
R18 0x00
```

Κατά την εκτέλεση του προγράμματος παρατηρούμε ότι μετά την εντολή cpi r16, 4 για i<4 ανοίγει το carry flag. Για να γίνει η πράξη στο δυαδικό όταν ο $1^{\circ\varsigma}$ αριθμός είναι μικρότερος από τον 2° χρειαζόμαστε ένα ακόμα bit στον 1° αριθμό. Όταν χρειάζεται να προστεθεί αυτό το επιπλέον bit ανοίγει το "borrow flag". Το borrow flag αν και χρησιμοποιεί το ίδιο flag με το carry δεν χρησιμοποείται για τον ίδιο σκοπό.

Για i=4 το zero flag ανοίγει καθώς για να γίνει η σύγκριση μεταξύ της τιμής του καταχωρητή και του 4 γίνεται η αφαίρεση: r16 – 4. Καθώς η αφαίρεση αυτή μας δίνει μηδέν ανοίγει το zero flag.

Ερώτηση 1: Κάνοντας αντικατάσταση της εντολής rjmp με την jmp παρατηρούμε πως το πρόγραμμα τρέχει κανονικά. Άρα στο συγκεκριμένο πρόγραμμα δεν έχει καμία διαφορά. Βλέποντας το manual παρατηρούμε πως η rjmp μπορεί να μεταβεί σε συγκεκριμένες θέσεις μνήμης ενώ η jmp μπορεί να μεταβεί σε οποιαδήποτε θέση μνήμης.

Ερώτηση 2: Ναι μπορεί καθώς με την εντολή brne όσο το i<4 θα γίνεται το jump στο branch next μέχρι το i να γίνει 4. (Με τον 2° τρόπο)

Εργασία 2

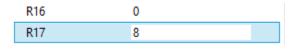
Τρέχοντας το πρόγραμμα παρατηρούμε ότι το flag carry ανοίγει όταν εκτελείται η εντολή cpi r16, 4 για i < 4, και για i = 4 ανοίγει μόνο το zero flag. Οι καταχωρητές r16, r17 έχουν τις ίδιες τελικές τιμές.

Τροποποιούμε το πρόγραμμα ως εξής:

- Αντί για ldi r16, 0 δίνουμε στον καταχωρητή r16 την τιμή 4
- Αντί να αυξάνουμε την τιμή του i (καταχωρητής r16) την μειώνουμε κατά 1. dec r16, i
- Αντί να ελέγχουμε την τιμή του carry flag ελέγχουμε την τιμή του zero flag. Γίνεται επανάληψη όσο το flag zero είναι κλειστό, και μόλις ανοίξει βγαίνουμε από τον βρόγχο.

```
2
    .ORG 0x0000
3
4
    start:
5
    ldi r17, 0x80 ; initialization
6
7
        ldi r16, 4 ; i = 4
8
9 while:
10
        lsr r17 ; r17 = r17 / 2
11
        dec r16 ; i--
12
        cpi r16, 0 ; if i != 4...
13
        brne while
14
        jmp start
```

Οι τιμές των καταχωρητών r16, r17 στο δεκαδικό όταν το πρόγραμμα τερματίσει είναι:



Συμπληρώνουμε τον κώδικα ως εξής:

```
1    .dseg
2    .ORG 0x0100
3     array: .byte 10
4
5    .cseg
6    .ORG 0x0000
7
8     eor r16, r16 ; i = 0
9     eor r18, r18 ; sum = 0
10     ldi r27, high(array) ; initialize X
11    ldi r26, low(array)
12
13
14    next:
15     ld r17, X+ ; r17 = array[i]
16
17     add r18, r17
18     inc r16 ; i++
19     cpi r16, 10 ; if i < 10...
20
21     brcs next ; ...goto next</pre>
```

Δίνουμε στην ram τις ακόλουθες τιμές:

```
Memory: data IRAM

▼ Columns: Auto

data 0x0100
 data 0x011C
 data 0x018C
 data 0x01A8
data 0x01E0
 data 0x01FC
```

1^η Επανάληψη

• Εκτέλεση εντολής ld:

X: 0101

R16 = 0

R17 = 1

R18 = 0

Flags: -

Add

X: 0101

R16 = 0

R17 = 1

R18 = 1 Flags: -

Inc

X: 0101

R16 = 1

R17 = 1

R18 = 1 Flags: -

• Cpi

X: 0101

R16 = 1

R17 = 1

R18 = 1

Flags: S, N, C

Παρατηρούμε πως πάλι ανοίγει το carry (borrow) flag καθώς ο αριθμός που αφαιρούμε είναι μεγαλύτερος από την τιμή του r16. Ανοίγουν επίσης και τα flag Sign και Negative, το negative flag γιατί το αποτέλεσμα είναι αρνητικό, και το sign flag γιατί το αποτέλεσμα πρέπει να είναι αρνητικό.

10η Επανάληψη

• Εκτέλεση εντολής ld:

X: 010A

R16 = 9

R17 = 16

R18 = 45

Flags: -

Add

X: 010A

R16 = 9

R17 = 16

R18 = 61

Flags: -

Inc

X: 010A

R16 = 10

R17 = 16

R18 = 61

Flags: -

• Cpi

X: 010A

R16 = 10

R17 = 16

R18 = 61

Flags: Z

Παρατηρούμε πως η τιμή του Χ στο δυαδικό ξεκινάει από 0x100 και φτάνει στο 0x10Α στην τελευταία επανάληψη.

Το flag Z έχει ανοίξει καθώς στην εντολή cpi r16, 10 η αφαίρεση του 10 από το r 16_{10} μας δίνει 0. (r 16_{10} - 10 = 10 - 10 = 0)

Ερώτηση 3: Μετατρέποντας την τιμή FFFF στο δεκαεξαδικό βλέπουμε ότι μας δίνει τον αριθμό 65535. Καθώς ο Χ που μας δείχνει την θέση του πίνακα αποτελείται ένα ζευγάρι δύο καταχωρητών όπου η μέγιστη τιμή που μπορούν να πάρουν είναι το 65535 μπορούμε να προσθέσουμε με αυτόν τον τρόπο 65535 στοιχεία.

Τροποποιούμε το πρόγραμμα μας ως εξής:

```
.ORG 0x0100
     3
           array: .byte 10
     4
     5 .cseg
     6 .ORG 0x0000
     7
     8
          eor r16, r16 ; i = 0
    9
          eor r17, r17 ; tmp = 0
    10
          eor r18, r18 ; sum = 0
    11
          eor r19, r19 ; borrow
    12
    13
          ldi r27, high(array) ; initialize X
          ldi r26, low(array)
    15
    16 start:
    17
          rjmp main
    18
    19 main:
    21
           add r18, r17 ; sum = sum + tmp
    22
           brcs borrow ; if carry flag is on goto borrow
    23
    24 cont:
   25
    26
           inc r16 ; i++
          cpi r16, 10 ; if i < 10...
    27
    28
           ;cpi r16, 4 ; if i < 4
    29
          brcs main ; goto main
    30
    32 borrow:
          inc r19 ; borrow++
          rjmp cont
    35
    36
100 % - 4
```

Χρησιμοποιούμε τον καταχωρητή r16 για την μέτρηση των επαναλήψεων, τον r17 ως tmp και τον r18 για το άθροισμα όπως και στην προηγούμενη εργασία.

Στο αρχικό branch 'main' διαβάζουμε ένα στοιχείο από τον πίνακα και το αποθηκεύουμε στον καταχωρητή r17 και στην συνέχεια το προσθέτουμε στο άθροισμα δηλαδή τον καταχωρητή r18. Αν από αυτή την πράξη προκύψει κρατούμενο δηλαδή ανοίξει το flag carry κάνουμε jump στο branch 'borrow'. Αλλιώς το πρόγραμμα συνεχίζει στο branch 'cont'.

To branch 'borrow' αυξάνει τον καταχωρητή r19 τον οποίο χρησιμοποιούμε για το κρατούμενο κατά 1.

Στο branch 'cont' αυξάνουμε τον μετρητή i (καταχωρητής r16) κατά 1 και τον συγκρίνουμε με το 10. Αν το i είναι μικρότερο του 10 το πρόγραμμα πηγαίνει στο branch main αλλιώς πηγαίνει στο branch start.

Ξεκινάμε το πρόγραμμα και δίνουμε σαν είσοδο f0, 14 για να έχουμε υπερχείλιση. Μετατρέπουμε τους αριθμούς αυτούς στο δυαδικό και τους προσθέτουμε -> $f0_{16}$: 11110000 + 14_{16} :00010100 = 0100000100

```
Binary value:
 11110000 + 00010100
 = 0100000100
Decimal value:
 240 + 20
 = 260
```

Στην συνέχεια τρέχουμε το πρόγραμμα μέχρι να γίνει η πρόσθεση και να φτάσουμε στο branch όπου αυξάνουμε τον μετρητή του κρατουμένου. Οι καταχωρητές r18 και r19 έχουν τις εξής τιμές στο δυαδικό -> r19: 00000001 r18: 00000100.

00000100 00000001

Παρατηρούμε πως αν δούμε αυτούς τους καταχωρητές σαν ζεύγος

παίρνουμε το ίδιο αποτέλεσμα δηλαδή 01000001002 -> 26010.

Επαναλαμβάνουμε την διαδικασία δίνοντας σαν 3° στοιχείο του πίνακα το FC. Προσθέτουμε τις τιμές των Fc, 14, F0 και βλέπουμε ότι στο δεκαδικό μας δίνουν την τιμή 512.



Τρέχουμε το πρόγραμμα και μετά την πρόσθεση και την αύξηση του μετρητή του κρατουμένου ο καταχωρητής r18 έχει στο δυαδικό την τιμή 00000000 και ο καταχωρητής r19 την τιμή 00000010. Βλέποντας την τιμή των καταχωρητών σαν ζεύγος παίρνουμε τον αριθμό 1000000000 στο δυαδικό ο οποίος μας δίνει 512 στο δεκαδικό:

```
512 = Ob100000000
```

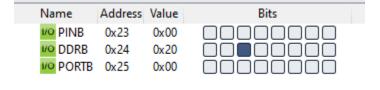
Ερώτηση 4: -----

Εργασία 6

Συμπληρώνουμε τις απαραίτητες δηλώσεις ως εξής:

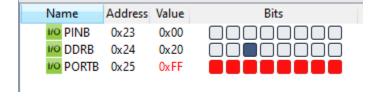
```
.dseg
     .ORG 0x0100
 2
 3
     .cseg
     .ORG 0x0000
     start:
 8
         ldi r16, 0b00100000
         out DDRB, r16
 9
10
         nop
11
         nop
         ldi r16, 0b11111111
12
         out PORTB, r16
13
14
         nop
15
         nop
16
         rjmp start
```

Τρέχουμε το πρόγραμμα μέχρι το 2° nop. Παρατηρούμε πως το 5° bit του DDRB έχει πάρει την τιμή 1.

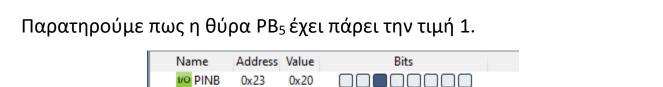


Δηλαδή η θύρα PB_5 θα είναι έξοδος.

Συνεχίζουμε την εκτέλεση του προγράμματος μέχρι την εντολή out PORTB, r16. Παρατηρούμε πως όλες οι τιμές του portB έχουν πάρει την τιμή 1.



Καθώς μόνο η θύρα PB_5 είναι έξοδος μόνο αυτή θα πάρει την τιμή 1.



0xFF

Συνεχίζουμε την εκτέλεση του προγράμματος μέχρι τα επόμενα 2 nop.

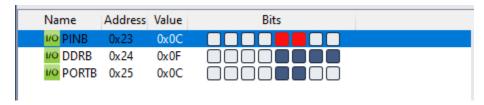
Τροποποιούμε το πρόγραμμα ως εξής:

```
.dseg
 2
     .ORG 0x0100
 3
 4
     .cseg
 5
     .ORG 0x0000
 6
 7
     start:
 8
       ldi r16, 0b00001111
 9
        out DDRB, r16
10
        nop
11
         nop
12
        ldi r16, 0b00001100
13
         out PORTB, r16
14
         nop
15
         nop
         rjmp start
```

Δίνουμε στο DDRB την τιμή 0b00001111 καθώς θέλουμε να δώσουμε έξοδο στα Pb₀ – Pb₃.

Δίνουμε στο PortB την τιμή 0b00001100 για να δώσουμε στα bit Pb₃, Pb_2 την τιμή 1, και στα bit Pb_1 , Pb_0 την τιμή 0.

Τρέχουμε το πρόγραμμα και παίρνουμε το εξής αποτέλεσμα:



Εργασία 8

Τροποποιούμε το πρόγραμμα ως εξής:

```
.dseg
     .ORG 0x0100
3
4
    .cseg
5
    .ORG 0x0000
7
    start:
        ldi r16, 0b00100000
8
9
         out DDRB, r16
10
         nop
11
         nop
12
13
     loop:
         ldi r16, 0b11111111
14
         out PORTB, r16
15
16
         nop
17
         nop
18
         ldi r16, 0b11011111
19
         out PORTB, r16
20
         nop
21
         nop
22
         rjmp loop
```

Δίνουμε στον καταχωρητή DDRB την τιμή 0b00100000 καθώς θέλουμε να πειράξουμε το 5° bit της θύρας.

Στη συνέχεια δίνουμε εναλλάξ τις τιμές 0b11111111, 0b11011111 στον καταχωρητή PortB έτσι ώστε το PB₅ να παίρνει τις τιμές 1, 0 αντίστοιχα.

Τρέχουμε το πρόγραμμα μέχρι την γραμμή 17 και βλέπουμε ότι το Pb₅ έχει πάρει την τιμή 1.



Συνεχίζοντας την εκτέλεση μέχρι την γραμμή 21, το PB5 παίρνει την τιμή 0.



0xDF

0x24

PORTB 0x25

Πρόγραμμα 7:

- Μετά την εκτέλεση των εντολών ldi r16, 0b00001111 και out DDRB, r16 το DDRB παίρνει την τιμή 00001111.
- Μετά την εκτέλεση των ldi r16, 0b00001100 και out PORTB, r16
 τα Port 2,3 παίρνουν την τιμή 1 και τα PB_{2,3} παίρνει την τιμή 1.

Πρόγραμμα 8:

- Μετά την εκτέλεση των εντολών ldi r16, 0b00100000 και out DDRB, r16 το DDRB $_5$ παίρνει την τιμή 1.
- Μετά την εκτέλεση των ldi r16, 0b111111111 και out PORTB, r16 τα Port 0-7 παίρνουν την τιμή 1 και το PB_5 παίρνει την τιμή 1.
- Μετά την εκτέλεση των ldi r16, 0b11011111 και out PORTB, r16 τα Port 0-7 παίρνουν την τιμή 1 εκτός το PortB5 που παίρνει την τιμή 0 και το PB₅ παίρνει την τιμή 0.

Εργασία 10

Τρέχοντας τον εξής κώδικα:

```
.dseg
    .ORG 0x0100
    A: .byte 1
        B: .byte 1
      C: .byte 1
7
    .cseg
    .ORG 0x0000
10 start:
   ldi r16, low(RAMEND) ; set up the stack
11
12
       out SPL, r16
       ldi r16, high(RAMEND)
13
14
       out SPH, r16
15
        lds r16, A ; Input arguments
16
        lds r17, B
17
       rcall calc
     ;call calc
18
19
        sts C, r18; Output argument
20 rjmp start
21
22 calc:
23
       add r16, r16 ; 2*A
      mov r18, r16
      sub r18, r17 ; 2*A - B
25
        add r18, r18; 4*A - 2*B
26
27
    ret
28
```

Όταν πήγαινε να εκτελεστεί η εντολή rcall calc μου εμφανιζόταν το εξής μήνυμα και το πρόγραμμα τερμάτιζε / συνέχιζε την εκτέλεση χωρίς να βλέπω αποτελέσματα:

Δοκίμασα επίσης και με την εντολή call calc την οποία έχω σε σχόλιο αλλά πήρα το ίδιο αποτέλεσμα. Πριν μου εμφανιστεί το μήνυμα αποτυχίας η τιμή του PC ήταν: $\frac{Program Counter}{Program Counter}$ και η τιμή του stack pointer στην θέση μνήμης 0x08FF ήταν 00.

Δυστυχώς λόγο αυτού του προβλήματος δεν μπόρεσα να σημειώσω

άλλα αποτελέσματα / παρατηρήσεις.

Δυστυχώς στην εργασία 11 δεν έτρεξε όπως θα ήθελα. Συγκεκριμένα η εντολή reall cale αγνοούνταν τελείως από το πρόγραμμα, και συνέχιζε με την εκτέλεση της εντολής inc r16. Παρακάτω έχω βάλει ένα screenshot με το πως θα μπορούσε να υλοποιηθεί η εργασία 5 με υπορουτίνα ωστόσο ο κώδικας δεν τρέχει σωστά έτσι ώστε να σημειώσω τα αποτελέσματα/παρατηρήσεις.

```
.dseg
 2
    .ORG 0x0100
 3
       array: .byte 10
 5
    .cseg
 6
    .ORG 0x0000
 7
8
       ldi r16, low(RAMEND) ; set up the stack
9
       out SPL, r16
10
       ldi r16, high(RAMEND)
11
       out SPH, r16
    eor r16, r16; i = 0
eor r17, r17; tmp = 0
eor r18, r18: sum
12
13
15
16
       eor r19, r19 ; borrow
17
18
       ldi r27, high(array) ; initialize X
19
        ldi r26, low(array)
20
21 start:
22
       ld r17, X+ ; tmp = array[i]
23
       rcall calc ; goto calc
24
       inc r16 ; i++
25
       cpi r16, 11 ; if i < 10...</pre>
26
27 calc:
28
       add r18, r17 ; sum = sum + tmp
29
       brcc return ; if carry flag is off return
30
       call borrow
31
32
        ret
33
34
   return:
35
36
37 borrow:
38
       inc r19 ; borrow++
39
        ret
40
```