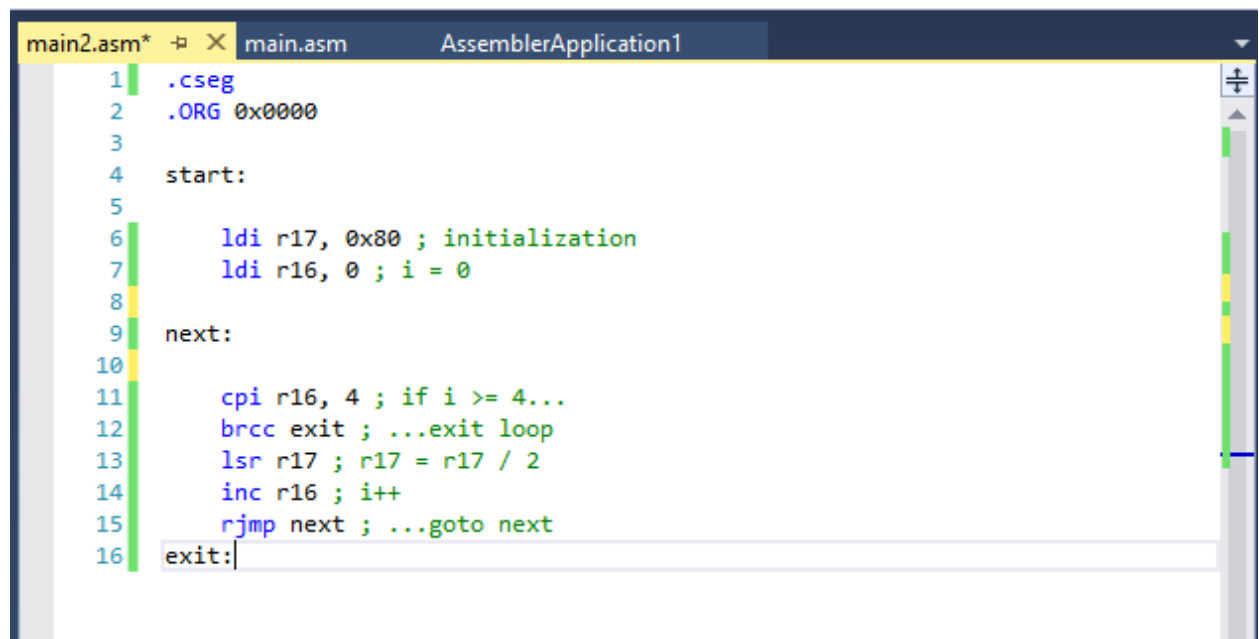


## Εργασία 1

Συμπληρώνουμε τον κώδικα ως εξής:



```
1 .cseg
2 .ORG 0x0000
3
4 start:
5
6     ldi r17, 0x80 ; initialization
7     ldi r16, 0 ; i = 0
8
9 next:
10
11     cpi r16, 4 ; if i >= 4...
12     brcc exit ; ...exit loop
13     lsr r17 ; r17 = r17 / 2
14     inc r16 ; i++
15     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<sup>ος</sup> αριθμός είναι μικρότερος από τον 2<sup>ο</sup> χρειαζόμαστε ένα ακόμα bit στον 1<sup>ο</sup> 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 χρησιμοποιείται 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<sup>ο</sup> 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.

```
1  .cseg
2  .ORG 0x0000
3
4  start:
5
6  |  ldi r17, 0x80 ; initialization
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:

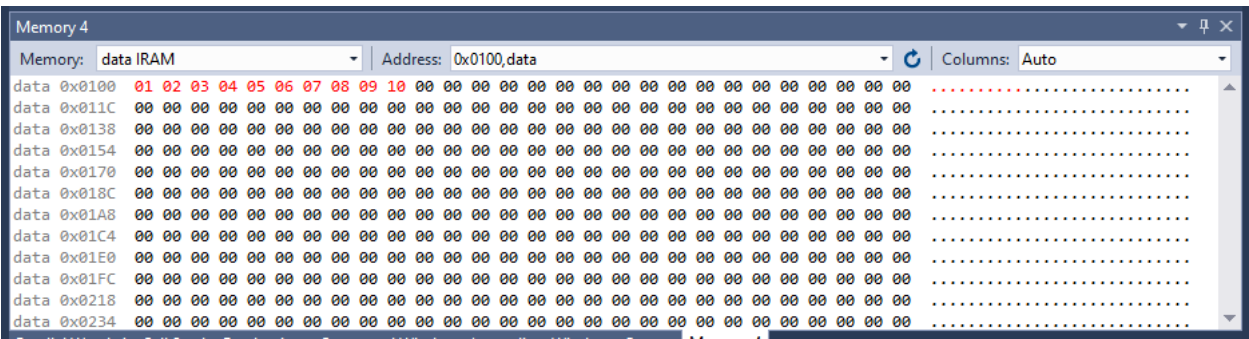
R16	0
R17	8

Εργασία 4

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
```

Δίνουμε στην ram τις ακόλουθες τιμές:



1<sup>η</sup> 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.

## 10<sup>η</sup> 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 X στο δυαδικό ξεκινάει από 0x100 και φτάνει στο 0x10A στην τελευταία επανάληψη.

Το flag Z έχει ανοίξει καθώς στην εντολή cpi r16, 10 η αφαίρεση του 10 από το  $r16_{10}$  μας δίνει 0. ( $r16_{10} - 10 = 10 - 10 = 0$ )

Ερώτηση 3: Μετατρέποντας την τιμή FFFF στο δεκαεξαδικό βλέπουμε ότι μας δίνει τον αριθμό 65535. Καθώς ο X 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.

Εργασία 5

Τροποποιούμε το πρόγραμμα μας 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 r17, r17 ; tmp = 0
10     eor r18, r18 ; sum = 0
11     eor r19, r19 ; borrow
12
13     ldi r27, high(array) ; initialize X
14     ldi r26, low(array)
15
16     start:
17         rjmp main
18
19     main:
20         ld r17, X+ ; tmp = array[i]
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++
27         cpi r16, 10 ; if i < 10...
28         ;cpi r16, 4 ; if i < 4
29
30         brcs main ; goto main
31
32     borrow:
33         inc r19 ; borrow++
34         rjmp cont
35
36
```

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’.

Το 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<sub>16</sub>: 11110000 + 14<sub>16</sub>: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.

R18	00000100
R19	00000001

Παρατηρούμε πως αν δούμε αυτούς τους καταχωρητές σαν ζεύγος παίρνουμε το ίδιο αποτέλεσμα δηλαδή 0100000100<sub>2</sub> -> 260<sub>10</sub>.







Εργασία 9

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<sub>2,3</sub> παίρνει την τιμή 1.

Πρόγραμμα 8:

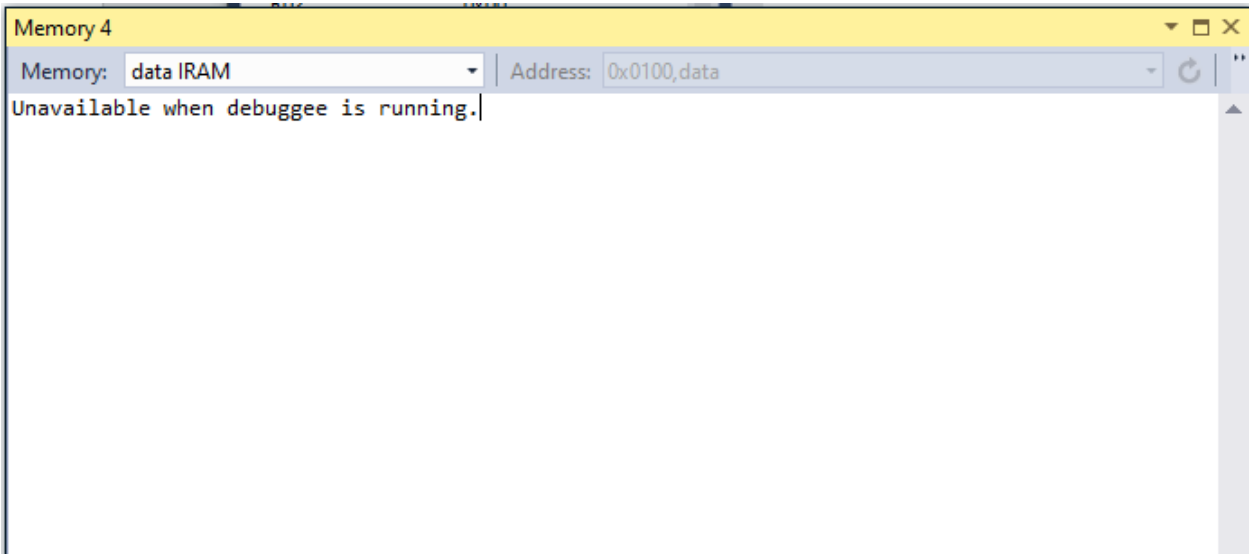
- Μετά την εκτέλεση των εντολών `ldi r16, 0b00100000` και `out DDRB, r16` το DDRB<sub>5</sub> παίρνει την τιμή 1.
- Μετά την εκτέλεση των `ldi r16, 0b11111111` και `out PORTB, r16` τα Port 0-7 παίρνουν την τιμή 1 και το PB<sub>5</sub> παίρνει την τιμή 1.
- Μετά την εκτέλεση των `ldi r16, 0b11011111` και `out PORTB, r16` τα Port 0-7 παίρνουν την τιμή 1 εκτός το PortB5 που παίρνει την τιμή 0 και το PB<sub>5</sub> παίρνει την τιμή 0.

Εργασία 10

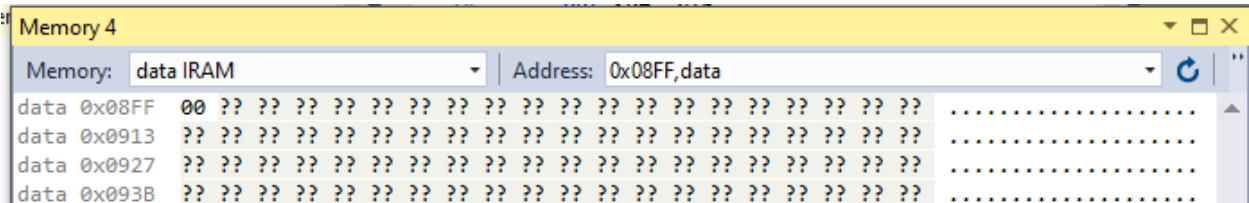
Τρέχοντας τον εξής κώδικα:

```
1  .dseg
2  .ORG 0x0100
3      A: .byte 1
4      B: .byte 1
5      C: .byte 1
6
7  .cseg
8  .ORG 0x0000
9
10 start:
11     ldi r16, low(RAMEND) ; set up the stack
12     out SPL, r16
13     ldi r16, high(RAMEND)
14     out SPH, r16
15     lds r16, A ; Input arguments
16     lds r17, B
17     rcall calc
18     ;call calc
19     sts C, r18 ; Output argument
20     rjmp start
21
22 calc:
23     add r16, r16 ; 2*A
24     mov r18, r16
25     sub r18, r17 ; 2*A - B
26     add r18, r18 ; 4*A - 2*B
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: `Program Counter 0x00000008` και η τιμή του stack pointer στην θέση μνήμης 0x08FF ήταν 00.



Δυστυχώς λόγω αυτού του προβλήματος δεν μπόρεσα να σημειώσω άλλα αποτελέσματα / παρατηρήσεις.



## Εργασία 11

Δυστυχώς στην εργασία 11 δεν έτρεξε όπως θα ήθελα. Συγκεκριμένα η εντολή `rcall calc` 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.

```
1  .dseg
2  .ORG 0x0100
3      array: .byte 10
4
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
12
13     eor r16, r16 ; i = 0
14     eor r17, r17 ; tmp = 0
15     eor r18, r18 ; sum = 0
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...
26
27 calc:
28     add r18, r17 ; sum = sum + tmp
29     brcc return ; if carry flag is off return
30
31     call borrow
32     ret
33
34 return:
35     ret
36
37 borrow:
38     inc r19 ; borrow++
39     ret
40
```