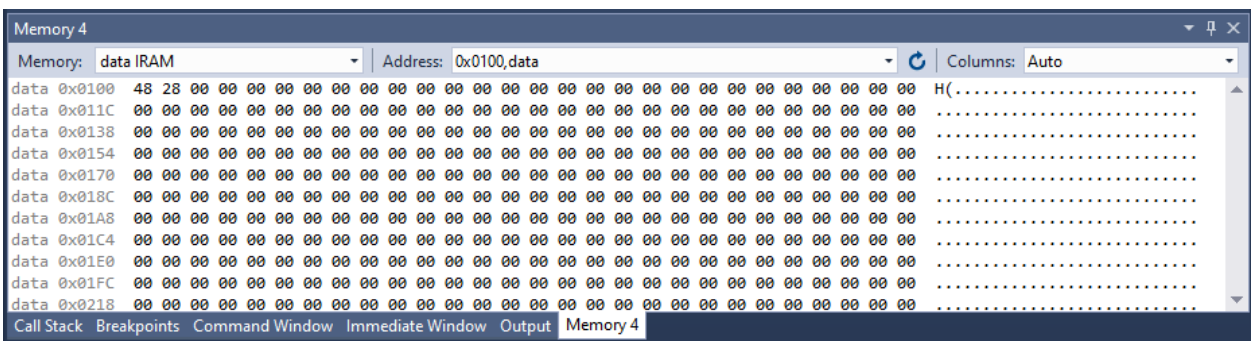
Εργαστήριο Μικροεπεξεργαστών

1η Εργαστηριακή άσκηση

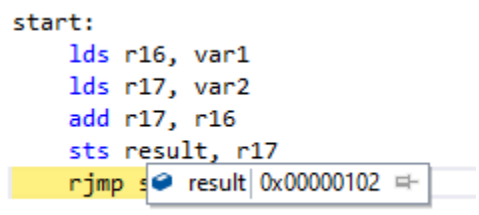
Γέροντας Αλέξανδρος 321 - 2015029

1ο Ζητούμενο – Πρόσθεση δύο μεταβλητών

(1) Εισάγουμε τις τιμές 48, 0x28 στην Ram.



Κάνοντας “hover” πάνω από την τιμή του result βλέπουμε ότι αντιστοιχεί στην 3η θέση μνήμης 0x00000102 η οποία αρχικά είναι 0x00.



Πατάμε F10 και οι καταχωρητές σε κάθε βήμα 1-4 παίρνουν τις τιμές:

1. R16 = 0x48
   R17 = 0x00
   result = 0x00

2. R16 = 0x48
   R17 = 0x28
   result = 0x00

3. R16 = 0x48
   R17 = 0x70
   result = 0x00

4. R16 = 0x48
   R17 = 0x70
   result = 0x70

Όπου το 1ο βήμα είναι η εντολή `lds r16, var1` και το 4ο η εντολή `sts result, r17`.

Στο 4ο βήμα παρατηρούμε ότι η θέση μνήμης 0x00000102 παίρνει την τιμή 0x70.

Κανένας από τους ενδείκτες που μας ενδιαφέρουν (S, V, N, Z, C) δεν έχει πάρει την τιμή 1.

(2) Βάζουμε πάλι τις τιμές στην Ram και τρέχουμε το πρόγραμμα.
Το 1ο βήμα είναι η εντολή `lds r16, var1` και το 4ο η εντολή `sts result, r17`.

i) Τιμές 0x48, 0x28 - με βάση το προηγούμενο ερώτημα.

ii) Για τις τιμές: 0x48, 0x58

1. R16 = 0x48
   R17 = 0x00
   result = 0x00

2. R16 = 0x48
   R17 = 0x58
   result = 0x00

3. R16 = 0x48
   R17 = 0xA0
   result = 0x00

4. R16 = 0x48
   R17 = 0x70
   result = 0xA0

Η τιμή του result στο δεκαδικό είναι 160:

Hex value:
48 + 58 = **A0**

Decimal value:
72 + 88 = **160**

Παρατηρούμε ότι οι ενδείκτες V, N μετά το 3ο βήμα έχουν πάρει την τιμή 1.

- Overflow έχουμε όταν η πρόσθεση δύο θετικών αριθμών μας δίνει αρνητικό αριθμό. Καθώς οι προσημασμένοι ακέραιοι 8 bit μπορούν να πάρουν τιμές από -128 ως 127 οποιαδήποτε τιμή πάνω από το 127 (όπως το 160) συνεπάγει overflow.
- Ο ενδείκτης N μας δείχνει ότι η τιμή του result θα μπορούσε να είναι αρνητική. Αν το πρόγραμμα μας δουλεύει με μη προσημασμένους μπορούμε να αγνοήσουμε αυτόν τον ενδείκτη.

iii) Για τις τιμές: 0x72, 0x81

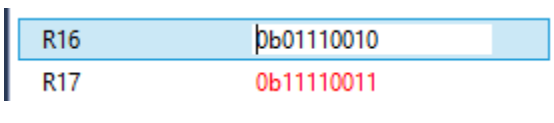
1. R16 = 0x72
   R17 = 0x00
   result = 0x00

2. R16 = 0x72
   R17 = 0x81
   result = 0x00

3. R16 = 0x72
   R17 = 0xF3
   result = 0x00

4. R16 = 0x72
   R17 = 0xF3
   result = 0xF3

Στο 3ο βήμα οι ενδείκτες S, N έχουν πάρει την τιμή 1. Ο ενδείκτης S μας δείχνει ότι το πιο σημαντικό bit έχει πάρει την τιμή 1. Πράγματι αν δούμε την τιμή του ενδείκτη R17 στο δυαδικό, βλέπουμε ότι το 1ο bit έχει πάρει την τιμή 1.



Ο καταχωρητής R17 έχει πάρει την τιμή $0xF3_{16}$ -> $243_{10.}$ Είναι εκτός του ορίου των προσημασμένων και για αυτό ανοίγει το flag N.

iv) Για τις τιμές: 0x72, 0xff

1. R16 = 0x72
   R17 = 0x00
   result = 0x00

2. R16 = 0x72
   R17 = 0xff
   result = 0x00

3. R16 = 0x72
   R17 = 0x71
   result = 0x00

4. R16 = 0x72
   R17 = 0x71
   result = 0x71

Στο 3ο βήμα η τιμή του ενδείκτη C έχει πάρει την τιμή 1. Ο ενδείκτης carry ενεργοποιείται όταν στο δυαδικό μας έχει μείνει κρατούμενο μετά την πράξη στο πιο σημαντικό bit. Πράγματι αν δώσουμε σε τρεις καταχωρητές τις τιμές 0x72, 0xff, 0x71 και δούμε τις τιμές τους στο δυαδικό θα δούμε ότι αν προσθέταμε στο δυαδικό τις τιμές 0x72, 0xff στο τέλος της πράξης θα περίσσευε ένας άσσος.

R08	0b01110010
R09	0b11111111
R10	0b01110001

v) Για τις τιμές: 0xFF, 0x01

1. R16 = 0xFF
   R17 = 0x00
   result = 0x00

2. R16 = 0xFF
   R17 = 0x01
   result = 0x00

3. R16 = 0xFF
   R17 = 0x00
   result = 0x00

4. R16 = 0xFF
   R17 = 0x00
   result = 0x00

Στο 3ο βήμα η τιμή των ενδεικτών Z, C έχει πάρει την τιμή 1. Η τιμή του καταχωρητη R16 - 0xFF στο δεκαδικό είναι ίση με 255 δηλαδή στο όριο των μη προσημασμένων αριθμών 8 bit. Καθώς δεν μπορούμε να αναπαραστήσουμε αριθμούς πάνω από 255 με 8 bit οποιαδήποτε πράξη πρόσθεσης με τον καταχωρητή R16 θα μας δώσει έναν αριθμό μεταξύ 0-255. Εφόσον παίρνουμε αποτέλεσμα 0x00 ή 0 στο δεκαδικό ενεργοποιείται το Zero flag, και καθώς στην πρόσθεση του 0xFF με το 0x01 στο δυαδικό θα μας περίσσευε ένας άσσος μετά την πράξη στο πιο σημαντικό bit ενεργοποιείται το Carry flag.

R00	0b11111111
R01	0b00000001
R02	0b00000000

(3) Σύμφωνα με το documentation του επεξεργαστή οι register του είναι των 8-bit

AVR Register File

- 32 8-bit GP registers

και σύμφωνα με τις δηλώσεις των μεταβλητών, βλέπουμε ότι το μέγεθος των δεδομένων είναι 1 byte (ή ακόμα ένας ακέραιος unsigned από [0, 255] και ένας ακέραιος signed από [-128, 127] έχει μέγεθος 8 bit.

```
var1: .byte 1
var2: .byte 1
result: .byte 1
```

Represents a **signed** integer number stored with 8, 16 or 32 bit.
...

Number of bits	Min. value	Max. value
8 bit	–128	127
16 bit	–32768	32767
32 bit	–2147483648	2147483647

Άρα το μέγεθος των δεδομένων είναι 1 byte.

Οι μεταβλητές βρίσκονται στις διευθύνσεις:

var1: 0x00000100

var1: 0x00000101

result: 0x00000102

(4) Το μέγεθος του κώδικα σε bytes είναι 16. Για να δούμε την μνήμη του προγράμματος βλέπουμε την μνήμη prog Flash. Ο κώδικας βρίσκεται στα πρώτα 16 byte, και τα byte με τιμή ff είναι τα byte που δεν έχουμε χρησιμοποιήσει ακόμα.

Memory 4

Memory: prog FLASH Address: 0x0000,prog

```
prog 0x0000  00 91 00 01 10 91 01 01 10 0f 10 93 02 01 f8 cf ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x001C  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0038  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0054  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0070  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x008C  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00A8  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00C4  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00E0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x00FC  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0118  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0134  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0150  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x016C  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
prog 0x0188  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Η 1η εντολή κώδικα βρίσκεται στην θέση μνήμης 0x00000000 και η τελευταία στην θέση 0x0000000F.

Για να βρούμε το μέγεθος σε λέξεις βλέπουμε το manual του AVR όπου κάτω από κάθε εντολή βλέπουμε τις λέξεις και τα bytes της εντολής.

74.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
lds r2,$FF00 ; Load r2 with the contents of data space location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back
```

Words	2 (4 bytes)
Cycles	2
Cycles XMEGA	2 If the LDS instruction is accessing internal SRAM, one extra cycle is inserted

Συγκεντρώνουμε τις παρακάτω πληροφορίες:

Lds 2 word 4 byte

Add 1 word 2 byte

Sts 2 word 4 byte

Rjmp 1 word 2 byte

Οπότε 2*Lds_words + 1*Add_words + 1*Sts_words + 1*Rjump_words = 2*2 + 1*1 + 1*2 + 1*1 = 8 words

(5) Η εντολή add r17, r16 κωδικοποιείται σε γλώσσα μηχανής:
0f 10 -> 0000 1111 0001 0000

16-bit Opcode:

0000	11rd	dddd	rrrr

Σύμφωνα με το εγχειρίδιο τα 1^{α} έξι bit αφορούν το αναγνωριστικό της εντολής add -> 0000 1111 0001 0000, τα bit r αφορούν τον καταχωρητή R16 όπου στην περίπτωση μας: $16_{10} = 10000_2$

Άρα -> 0000 1111 0001 0000

και τα bit d τον καταχωρητή R17 όπου: $17_{10} = 10001_2$

Άρα -> 0000 1111 0001 0000

Description

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) (i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

(6) Η εντολή sts result, r17 κωδικοποιείται σε γλώσσα μηχανής:
9310 0102-> 1001 0011 0001 0000 0000 0001 0000 0010

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Σύμφωνα με το εγχειρίδιο τα 1^{α} εφτά bit αφορούν το αναγνωριστικό της εντολής sts -> 1001 0011 0001 0000 0000 0001 0000 0010, τα bit k αφορούν την διεύθυνση μνήμης στην οποία θα αποθηκευτεί ένα byte όπου στην περίπτωση μας: $0000\ 0001\ 0000\ 0010_2 = 102_{16}$

Άρα -> 1001 0011 0001 0000 0000 0001 0000 0010

και τα bit d τον καταχωρητή R17 όπου: $17_{10} = 10001_2$

Άρα -> 1001 0011 0001 0000 0000 0001 0000 0010

121.1. Description

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the Register File, I/O memory, and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64KB. The STS instruction uses the RAMPD Register to access memory above 64KB. To access another data segment in devices with more than 64KB data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	0 ≤ r ≤ 31, 0 ≤ k ≤ 65535	PC ← PC + 2

2^{ο} Ζητούμενο – Αφαίρεση δύο μεταβλητών

- Για τις τιμές: 0x48, 0x28

  result = 0xe0
  flag: S,N,C

- Για τις τιμές: 0x48, 0x58

  result = 0x10
  flag: -

- Για τις τιμές: 0x00, 0x01

  Result = 01
  flag: -

- Για τις τιμές: 0x72, 0xff

  Result = 8d
  flag: S,N

3ο Ζητούμενο – Λογικές πράξεις

(1)

- Για τις τιμές: 0x34, 0x0F

  result = 0x04
  flag: -

- Για τις τιμές: 0x30, 0x0F

  result = 0x00
  flag: Z

(2) Καταχωρούμε στον καταχωρητή R18 την τιμή του καταχωρητή R16 πριν γίνει η πράξη or: ldi r18, 0x34. Στη συνέχεια εκτελούμε το τμήμα κώδικα και βλέπουμε τις τιμές των καταχωρητών σε μορφή binary:

R16	0b00111111
R17	0b00001111
R18	0b00110100

Βλέπουμε ότι:

- όταν γίνεται η πράξη or μεταξύ 2 μηδενικών το αποτέλεσμα είναι 0
- όταν κάνουμε 0 or 1 παίρνουμε αποτέλεσμα 1
- και όταν γίνεται or μεταξύ 2 άσων παίρνουμε αποτέλεσμα 1.

(3) Δίνουμε πάλι στον καταχωρητή R18 την τιμή του R16 πριν γίνει η πράξη EOR: ldi r18, 0x55 και βλέπουμε τα αποτελέσματα σε μορφή binary.

R16	0b10101010
R17	0b11111111
R18	0b01010101

Παρατηρούμε πως η πράξη Eor μεταξύ 2 άσων μας δίνει 0, ενώ όταν κάνουμε Eor μεταξύ άσου και μηδενικού το αποτέλεσμα είναι 1.

4ο Ζητούμενο – Εντολές ελέγχου ροής του προγράμματος

Για την τιμή: 0xA5:

Δίνουμε στον καταχωρητή R16 την τιμή 0xA5 και στον καταχωρητή R17 την τιμή 0x00:

```
ldi r16, 0xA5
ldi r17, 0x00
```

Γίνεται σύγκριση των καταχωρητών με την εντολή cp:

```
cp r16, r17
```

Παρατηρούμε πως τα flag S, N έχουν πάρει την τιμή 1. Καθώς το 0xA5 στο δεκαδικό είναι 165 είναι εκτός του ορίου ενός προσημασμένου ακεραίου 8 bit, άρα είναι αρνητικός και το πιο σημαντικό του bit είναι 1.

Στη συνέχεια εκτελείται η εντολή Brpl η οποία ελέγχει αν το flag N είναι ανοικτό, και αν δεν είναι πηγαίνουμε στο branch positive.

```
brpl positive
```

Καθώς είναι ανοιχτό δεν μπορούμε να κάνουμε jump και πηγαίνουμε στην παρακάτω εντολή όπου αφαιρούμε την τιμή του καταχωρητή R16 από το μηδέν για να πάρουμε την θετική τιμή του.

```
sub r17, r16
```

Βλέποντας την τιμή του καταχωρητή R17 στο δεκαδικό βλέπουμε ότι έχει πάρει την τιμή 91.

R17	91

Όντως καθώς ένας ακέραιος 8 bit μπορεί να πάρει τιμές από 0-255 δηλαδή 256 διαφορετικές τιμές – αν κάνουμε την πράξη 256 – 165 παίρνουμε 91.

Τέλος η εντολή rjmp μας πηγαίνει στο τέλος του προγράμματος.

Για την τιμή: 0x64:

Δίνουμε στον καταχωρητή R16 την τιμή 0x64 η οποία στο δεκαδικό είναι ο αριθμός 100.

Γίνεται σύγκριση των καταχωρητών με την εντολή cp:

```
cp r16, r17
```

Παρατηρούμε αυτή τη φορά κανένα flag δεν έχει πάρει την τιμή 1.

Στη συνέχεια εκτελείται η εντολή `brpl positive`

Καθώς το flag N δεν είναι ανοικτό παραλείπονται οι επόμενες δύο εντολές και πηγαίνουμε στο branch positive όπου απλά αποθηκεύουμε την τιμή του καταχωρητή R16 στον καταχωρητή R17. Οντως ο καταχωρητής R17 από 0x00 παίρνει την τιμή 0x64.

R16	0x64
R17	0x64