Εργαστήριο Μικροεπεξεργαστών

Θεωρητικές ασκήσεις

Γέροντας Αλέξανδρος 321 - 2015029

Άσκηση 3

Για να χρησιμοποιήσουμε την θύρα B σαν είσοδο πρέπει να δώσουμε στον καταχωρητή DDRBn την τιμή 0.

```
start:

    ldi r18, 0b00000000

    out DDRB, r18 ; Η θύρα B θα χρισιμοποιηθεί για είσοδο
    out PORTB, r18 ; αρχικοποίηση της θύρας B portB

    ldi r16, 0 ; αρχικοποίηση του r16 για την αποθηκευση των bit με τιμή 1.
```

Δίνουμε λοιπόν στον καταχωρητή r18 την τιμή 0b00000000 και με την εντολή out την περνάμε στον DDRB. Επίσης περνάμε και στην θύρα B την τιμή αυτή για να την αρχικοποιήσουμε.

Καλούμε την ρουτίνα με την χρήση της rcall:

```
main:
    rcall routine ; κλήση της ρουτίνας
    rjmp main
```

Στην ρουτίνα με την εντολή in διαβάζουμε τα δεδομένα από την θύρα B και τα αποθηκεύουμε στον καταχωρητή r17. Καλούμε την ρουτίνα bit cnt για να μετρήσουμε τα bit με τιμή 1 και επιστρέφουμε από την ρουτίνα.

```
routine:
    in r17, PORTB ; Διαβάζουμε τα δεδομένα της θύρας B στον καταχωρητή r17

    rcall bit_cnt ; Καλούμε την ρουτίνα bit_cnt για να μετρήσουμε τα bit με τιμή 1

    ret ; Επιστροφή από την ρουτίνα
```

Στην ρουτίνα bit cnt κάνουμε δεξί shift στα bit του r17 (κόβουμε δηλαδή κάθε φορά το τελευταίο bit). Αν το bit που έχουμε κόψει έχει την τιμή 1 το carry flag θα ανοίξει. Αν το carry flag είναι ανοικτό καλούμε το inc bin για να αυξήσουμε την τιμή του r16 κατά 1. Στη συνέχεια ελέγχουμε αν το r17 έχει πάρει την τιμή 0 και αν όχι καλούμε ξανά την bit cnt μέχρι να διαβαστούν όλα τα bit.

```
bit_cnt:
lsr r17 ; Διαβάζουμε ένα bit από το r17 (από το τελευταίο στο πρώτο)
brcs inc_bin ; Αν το bit έχει τιμή 1 καλούμε την inc_bin για να αυξήσουμε την τιμή του r16

cpi r17, 0 ; Ελέγχουμε αν το r17 έχει μηδενιστεί
brne bit_cnt ; Αν όχι ξανακαλούμε την bit_cnt και συνεχίζουμε να διαβάζουμε τα επόμενα bit

ret ; Αλλιώς επιστρέφουμε από την ρουτίνα
```

Ξεκινάμε το πρόγραμμα και μετα την αρχικοποίηση της θύρας της δίνουμε την εξής τιμή:

Name	Address	Value	Bits
I/O PINB	0x23	0x00	☐☐☐☐☐☐☐☐
I/O DDRB	0x24	0x00	☐☐☐☐☐☐☐☐
I/O PORTB	0x25	0x4A	☐■☐☐■☐■☐

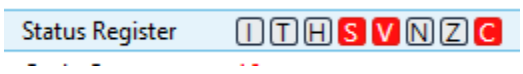
Η τιμή έχει αποθηκευτεί στον καταχωρητή r17:

R16	0x00
R17	0b01001010

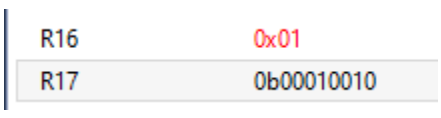
Με την εντολή lsr κόβουμε το τελευταίο bit του r17. Καθώς το bit ήταν 0 δεν καλείται η inc bin.

R16	0x00
R17	0b00100101

Διαβάζουμε το επόμενο bit. Καθώς το bit είναι 1 το carry flag ανοίγει.

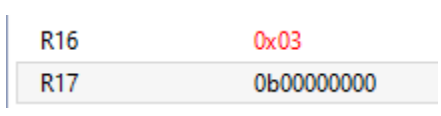


Καλούμε το inc bin και αυξάνεται η τιμή του r16 κατά 1.



Συνεχίζουμε την επανάληψη μέχρι να διαβαστούν όλα τα bit ή το r17 να πάρει την τιμή 0.

Όταν επιστρέψουμε από την ρουτίνα ο καταχωρητής r16 θα έχει την τιμή 3 όσα και τα bit που είχαμε δώσει στο portB.



Άσκηση 4

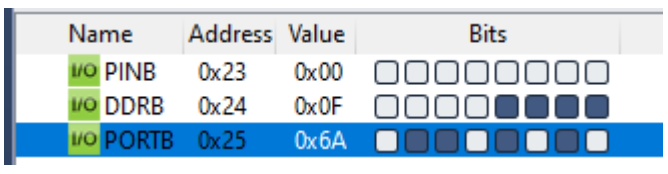
Αρχικοποιούμε την θύρα ως εξής – δίνουμε στα λιγότερο σημαντικά bit την τιμή 0 καθώς τα θέλουμε για είσοδο, και στα υπόλοιπα bit την τιμή 1 καθώς θα στείλουμε στο πιο σημαντικό bit ένα θετικό παλμό.

```
init_port:
    ldi r16, 0b00001111
    out DDRB, r16 ; Τα λιγότερο σημαντικά bit της θύρας B θα χρησιμοποιηθούν για είσοδο

    ldi r16, 0b00000000
    out PORTB, r16 ; αρχικοποίηση της θύρας B
```

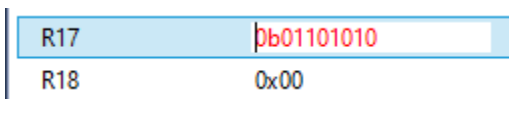
Διαβάζουμε την τιμή από το portB και την αποθηκεύουμε στο r17. Κόβουμε τα 4 πιο σημαντικά bit με την εντολή lsr (x4). Στη συνέχεια στέλνουμε τον θετικό παλμό, καλούμε την ρουτίνα delay η οποία καθυστερεί περίπου όσα msec όσο ο αριθμός που διαβάστηκε και επαναφέρουμε στο bit την τιμή 0.

```
start:
    in r17, PORTB ; Διάβασμα της τιμής από το portB

    lsr r17 ; Αποκοπή τον τελευταίων 4 bit
    lsr r17
    lsr r17
    lsr r17

    ldi r16, 0b00000001 ; Αποστολή θετικού παλμού στο πιο σημαντικό bit της θύρας
    out PORTB, r16

    rcall delay ; Καθυστέρηση που διαρκεί όσο τα bit που διαβάστηκαν

    ldi r16, 0b00000000 ; Επαναφορά του bit στο 0
    out PORTB, r16
```

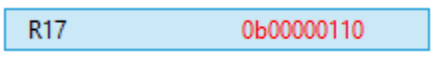
Για παράδειγμα δίνουμε στην θύρα B την εξής τιμή:



Διαβάζουμε την τιμή της θύρας B και την αποθηκεύουμε στον καταχωρητή r17.

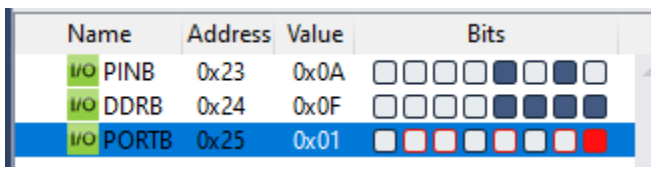


Κόβουμε τα 4 πιο σημαντικά bit με την χρήση της εντολής lsr (x4):

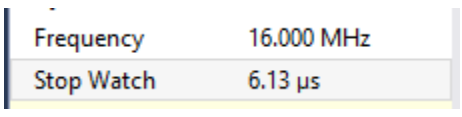


Όπου στο δεκαδικό αντιστοιχεί στην τιμή 6:

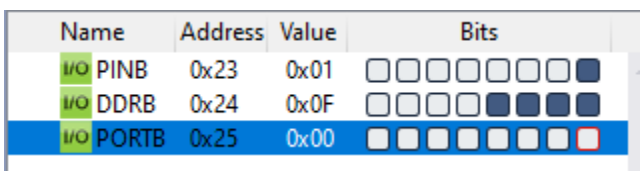
Στέλνουμε τον θετικό παλμό στην θύρα B:



Καθυστερούμε για περίπου 6 sec:



Το bit επανέρχεται στο 0:



Άσκηση 5

Το πρόγραμμα ξεκινάει με την start. Αρχικοποιούμε τον καταχωρητή X και με την χρήση της fill array γεμίζουμε τον πίνακα με ένα μοτίβο. Στη συνέχεια δίνουμε στον καταχωρητή r16 το μέγεθος του πίνακα, στον r17 το στοιχείο που ψάχνουμε και στον r18 θα αποθηκευτεί η θέση του πίνακα στην οποία βρίσκεται το στοιχείο.

```
start:

    ldi r27, high(array) ; αρχικοποίηση του δείκτη X
    ldi r26, low(array)

    ldi r16, 10 ; r16 = 10
    ldi r17, 2 ; r17 = 2

    rcall fill_array ; Γεμίζουμε τον πίνακα με την χρήση της fill array

    ldi r27, high(array) ; Καθώς ο δείκτης X θα δείχνει τωρα στο τέλος του πίνακα
    ldi r26, low(array) ; τον αρχικοποιούμε ξανά για να δείχνει στο πρώτο κελί

    ldi r16, 10 ; Δίνουμε στον καταχωρητή r16 το μέγεθος του πίνακα

    ldi r17, 8 ; Το στοιχείο του πίνακα που ψάχνουμε -> π.χ 8

    ldi r18, 0 ; η θέση αποθηκεύεται στον καταχωρητή r18.

    rcall find_element ; Αναζήτη του στοιχείου με την find element

    rjmp start
```

Στην ρουτίνα αποθηκεύουμε την τιμή του r17 σε ένα στοιχείο του πίνακα. Ο r17 έχει αρχική τιμή 3. Στη συνέχεια προσθέτουμε στο r17 την τιμή του r18 ο οποίος έχει αρχική τιμή 2. Αυξάνουμε τον r18 κατά 1. Η επανάληψη σταματάει όταν το r16 που αρχικά έχει το μέγεθος του πίνακα πάρει την τιμή 0, δηλαδή όταν δοθεί μια τιμή σε κάθε στοιχείο του πίνακα.

```
fill_array:
    st X+, r17 ; array[i] = r17

    add r17, r18 ; r17 = r17 + r18
    inc r18 ; r18++
    dec r16 ; r16 = r16 - 1

    breq return ;if r16 == 0 return
    rjmp fill_array ; else goto fill array
```

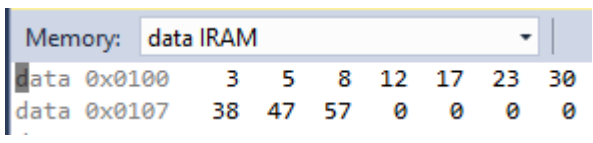
Στην ρουτίνα find element συγκρίνουμε κάθε στοιχείο του πίνακα με το r17 δηλαδή το στοιχείο που ψάχνουμε. Όταν βρεθεί το στοιχείο αποθηκεύουμε την θέση του με την get pos. Αν το στοιχείο δεν βρεθεί η τιμή του r18 παραμένει 0.

```
find_element:

    ld r19, X+ ; r19 = array[i]

    cp r17, r19 ; σύγκριση της τιμής του r19 με το στοιχείο που ψάχνουμε
    breq get_pos ; αν r17 = r19 αποθηκεύουμε την θέση στην get_pos

    cpi r16, 0
    breq return ; Αν έχουμε προσπελάσει όλα τα στοιχεία επιστρέφουμε απο την ρουτίνα.

    dec r16 ; r16 = r16 - 1

    rjmp find_element ; Αλλιώς συνεχίζουμε να ελέγχουμε τα υπόλοιπα στοιχεία του πίνακα
```

Όταν βρεθεί το στοιχείο αποθηκεύουμε την τιμή του r26 δηλαδή την θέση του στοιχείου στον καταχωρητή r18.

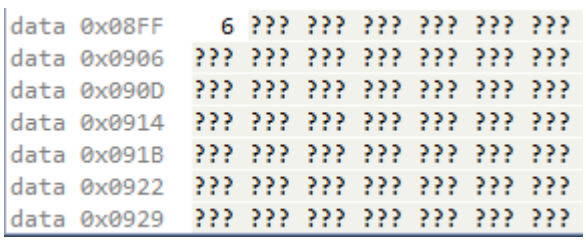
```
get_pos:
    mov r18, r26 ; Αποθήκευση της θέσης του στοιχείο στον καταχωρητή r18
```

Για παράδειγμα έστω ότι θέλουμε να δούμε αν υπάρχει το στοιχείο 8 στον πίνακα. Τρέχουμε το πρόγραμμα μέχρι η ρουτίνα fill array να δώσει τιμές στα στοιχεία του πίνακα:

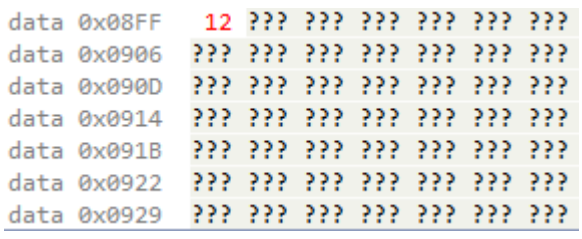


Στοιχεία του πίνακα unsigned

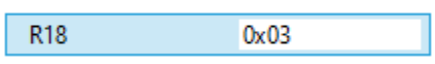
Η τιμή του pc πριν την κλήση της ρουτίνας ήταν 5. Αν πάμε στην διεύθυνση του stack pointer 0x08FF βλέπουμε πως η θέση της επόμενη εντολής που θα εκτελεστεί μετά την επιστροφή από την ρουτίνα δηλαδή η 6 έχει καταχωρηθεί στην στοίβα.



Στη συνέχεια καλείται η ρουτίνα find element για την αναζήτηση του στοιχείου. Η τιμή του pc πριν την κλήση της ρουτίνας είναι 11 (στο δεκαδικό). Η τιμή στη στοίβα τώρα είναι 12.



Γίνεται αναζήτηση του στοιχείου. Μετά την επιστροφή από την ρουτίνα το r18 έχει πάρει την τιμή 3 καθώς το στοιχείο βρίσκεται στην 3^{η} θέση του πίνακα.



Δοκιμάζουμε την αναζήτηση με τα ίδια στοιχεία στον πίνακα άλλα για ένα στοιχείο το οποίο δεν υπάρχει στον πίνακα όπως το 10. Βλέπουμε πως όλα τα στοιχεία του πίνακα προσπελάστηκαν καθώς το r16 το οποίο μειώνεται σε κάθε επανάληψη έχει πάρει την τιμή 0, και πως η τιμή του r18 έχει μείνει 0 καθώς το στοιχείο δεν βρέθηκε.