Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

**ΟΜΑΔΑ 23**

**Συνεργάτες**

Νικόλαος Αναγνώστου Νικόλαος Λάππας

03121818 03121098

**Ζήτημα 1.1**

Σε αυτή την άσκηση ζητήθηκε να παρέχουμε σε avr assembly μια συνάρτηση wait\_x\_msec η οποία ανάλογα με το τι αριθμός αναπαριστά το ζεύγος καταχωρητών r25 – r24 να παράγει την αντίστοιχη καθυστέρηση σε msec.

Προχωρούμε στην εξήγηση. Αφού ο μικροεπεξεργαστής τρέχει με συχνότητα 16 Mhz πρέπει για κάθε msec η ζητούμενη συνάρτηση wait\_x\_msec να φροντίζει εσωτερικά αυτής να καταναλώνονται 16000 κύκλοι αθροιστικά για όλα τα ζητούμενα msec πέραν του τελευταίου όπου για να πετύχουμε την απόλυτη ακρίβεια κάναμε κάτι διαφορετικό!

Βλέποντας τον κώδικα πιο κάτω παρατηρούμε πως καταναλώνονται συνολικά για κάθε msec εκτός του τελευταίου:

1+1+(3997\*2+3996\*2+1)”helper for each msec” +2+1+10+2 = 16000 κύκλοι.

Για το τελευταίο msec έχουμε:

1+1+(3997\*2+3996\*2+1)”helper for each msec” +2+2+4+**3+4** = 16000 κύκλους, συμπεριλαμβάνοντας και τους κύκλους της rcall εντολής (3) και της ret (4).

wait\_x\_msec:

ldi r26,LOW(15984);1 cycle

ldi r27,HIGH(15984);1 cycle

helper:

sbiw r26,4 ;2 cycles

brne helper ;2 cycles or 1 cycle for the last iteration

;15984 -> helper consumes 15983 cycles

;so after helper we consume totally 15985 cycles

sbiw r24,1 ;2 cycle

breq last\_msec ;1 cycle but if last msec 2 cycles

;for all msec except from the last -> 15985 + 2 + 1 = 15988 cycles

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop ;10 cycles

;extra 10 cycles -> 15998

brne wait\_x\_msec ;2 cycles total 16000 cycles with this operation

last\_msec:

;in the last iteration (last msec) we have 15989 cycles

nop

nop

nop

nop

;extra 4 cycles -> 15993 cycles

ret ;4 cycles

;with ret and rcall we calculated exactly 16000 cycles again

;so in both cases we end up having 16000 cycles -> 1 msec \* (desired time)

****Ενδεικτικά παραδείγματα

****x = 1msec

x = 2000msec or 2 sec

**Ζήτημα 1.2**

Στην συγκεκριμένη άσκηση καλούμαστε να υπολογίσουμε τις λογικές συναρτήσεις

σε avr assembly για τον μικροελεγκτή ATmega328.

Στον κώδικά μας, αφού φορτώσουμε στους καταχωρητές τις αρχικές τιμές των A, B, C και D, υπολογίσουμε μεμονωμένα τις λογικές τιμές σιγά σιγά με την βοήθεια προσωρινών καταχωρητών και τον εντολών OR (λογικό Ή) και AND (λογικό ΚΑΙ). Τα αποτελέσματα τα τυπώνουμε στο PORTD του μικροελεγκή, αφού πρόκειται για λογικές συναρτήσεις με 8-bit η καθεμία (και το PORTD έχει 8 leds) – Δεν ζητείται η εκτύπωση, απλά για οπτικοποίηση.

Το loop υπολογισμού των λογικών συναρτήσεων εκτελέστηκε 6 φορές, και σε κάθε loop αυξάναμε τους καταχωρητές τόσο όσο καθορίζει η εκφώνηση της άσκησης. Για να το πετύχουμε αυτό, επειδή αυξάνουμε απευθείας τους καταχωρητές όσο μας λέει η εκφώνηση, τους αρχικοποιήσαμε εκτός του loop με τιμή ανάλογη (μικρότερη τόσες φορές όσες έπρεπε). Τα αποτελέσματα φαίνονται στον επόμενο πίνακα:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **F0** | **F1** |
| 0x51 | 0x41 | 0x21 | 0x01 | 0xEF | 0xDF |
| 0x52 | 0x43 | 0x24 | 0x05 | 0xEB | 0xDB |
| 0x53 | 0x45 | 0x27 | 0x09 | 0xE5 | 0xD3 |
| 0x54 | 0x47 | 0x2A | 0x0D | 0xE7 | 0xD5 |
| 0x55 | 0x49 | 0x2D | 0x11 | 0xEB | 0xC7 |
| 0x56 | 0x4B | 0x30 | 0x15 | 0xEB | 0xCB |

main:

;initialize the registers with their values

;minus the number we add in each iteration

ldi A, 0x50

ldi B, 0x3F

ldi C, 0x1E

ldi D, 0xFD

ldi counter, 0x06

ser r24

out DDRD,r24; Init PORTD as output

main\_loop:

subi A, -1

subi B, -2

subi C, -3

subi D, -4

mov complement\_B, B

mov complement\_C, C

mov complement\_D, D

mov temp\_F0, A

mov temp\_F1, A

mov temp, D

com complement\_B

com complement\_C

com complement\_D

;calculate F0

and temp\_F0, complement\_B

and temp, complement\_B

or temp\_F0, temp

com temp\_F0 ;value F0

out PORTD, temp\_F0

;calculate F1

or temp\_F1, complement\_C

or complement\_D, B

and temp\_F1, complement\_D ;value F1

out PORTD, temp\_F1

dec counter

brne main\_loop ;loop 6 times

**Ζήτημα 1.3**

Σε αυτή την άσκηση ζητήθηκε να προσομοιώσουμε ένα τρενάκι που ξεκινάει από δεξιά και πάει αριστερά και σε κάθε βήμα κάνει στάση 1 sec και όταν φτάσει στο MSB της PORTD τότε κάνει 2 sec στάση και ξεκινάει κίνηση προς τα δεξιά όπου πάλι κάθε βήμα πρέπει να καθυστερεί 1 sec μέχρι να φτάσει στο LSB όπου κάνει στάση πάλι 2 sec και έπειτα πάει αριστερά ….και αυτό έπρεπε να εκτελείται ατέρμονα.

Αυτό υλοποιήθηκε με την σύνθετη συνάρτηση της άσκησης 1.1 wait\_x\_msec.

Πέρα από αυτή τη συνάρτηση η λογική της άσκησης είναι απλή. Θέσαμε την ddrd σε 0xFF ώστε η portd να είναι η έξοδος και έπειτα φορτώσαμε την τιμή 0x01 στην portd μιας και το τρένο κλήθηκε να ξεκινά από δεξιά. Κατόπιν ακολουθούν πολλαπλά lsl με ανάλογες καθυστερήσεις για κίνηση αριστερά, όπου θέταμε το T flag του sreg στο λογικό 0 (εντολή clt) και μετά με lsr για την κίνηση προς τα δεξιά με ανάλογες καθυστερήσεις, όπου θέταμε το T flag του sreg στο λογικό 1 (εντολή set). Αυτές οι δύο εντολές επενεργούν στο 6ο μόνο bit του καταχωρητή.

Τέλος, να σημειωθεί πως τροποποιήσαμε την wait\_x\_msec ώστε στο τέλος αυτής τα r24 και r25 να αρχικοποιούνται ξανά στα 1000msec. Αυτό επέφερε μια ακόμη πιο μικρή απόκλιση από τους 6 κύκλους που εξηγήσαμε στην 1.1, όμως σε πραγματικά νούμερα αυτή η απόκλιση δεν έπαψε να είναι παρά ελάχιστη και ο χρόνος μας είχε ως αποτέλεσμα ακρίβεια τουλάχιστον 3 δεκαδικών αν όχι και παραπάνω.

ldi train, 0x01 ;start from right -> left : code 0

;left -> right : code 1

;ldi r28,0xFF

out PORTD,train

transportation:

clt ;right -> left : code 0 in 6th bit of register SREG

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

lsl train

out PORTD,train

rcall wait\_x\_msec

rcall wait\_x\_msec ;reached the edge

set ;left -> right : code 1 in 6th bit of register SREG

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec

lsr train

out PORTD,train

rcall wait\_x\_msec ;reached the edge

rjmp transportation ;loop again