Ψηφιακή Σχεδίαση

4.2a)

Αν ξοδέψουμε και τους 8 διαθέσιμους συνδυασμούς του πεδίου op για 8 εντολές I-Format όχι δεν θα μπορούμε να έχουμε κάποια άλλη εντολή R-format. Αυτό συμβαίνει καθώς όλοι οι δυνατοί συνδυασμοί έχουν χρησιμοποιηθεί και έτσι το πρόγραμμα διαβάζοντας τα πρώτα LS bits δεν μπορεί να καταλάβει τη format έχει η εντολή οπότε δεν μπορεί να καταλάβει αν υπάρχει πεδίο funct. Όμως έστω ότι επιμέναμε να έχουμε μια εντολή με R-format με op=001 και funct=10. Τότε η εντολή RR x5 θα παρερμηνευόταν με εντολή I-Format δηλαδή τα 3 LS bits θα ήταν το opcode

001 και μετά τα υπόλοιπα 5 bits θα θωρούντουσαν Imm με τα MS bits 10101 με 101 = 5 καθώς έχουμε το register 5 και 10 αφού funct = 10. Οπότε θα γινόταν η εντολή li με imm 21.

Το ίδιο θα ίσχυε και για οποιαδήποτε άλλη εντολή μεταφράζαμε από R-format σε I-Format αφού δεν υπάρχει τρόπος το πρόγραμμα να καταλάβει ποια από τις εντολές είναι σε R-format αφού έχουν χρησιμοποιηθεί όλοι οι δυνατοί συνδυασμοί των opcode.

4.2b)

Σε αυτήν την περίπτωση μπορούμε να έχουμε 4 εντολές με R-format και αυτές θα είναι :

FUNCT R OP

00 xxx 111

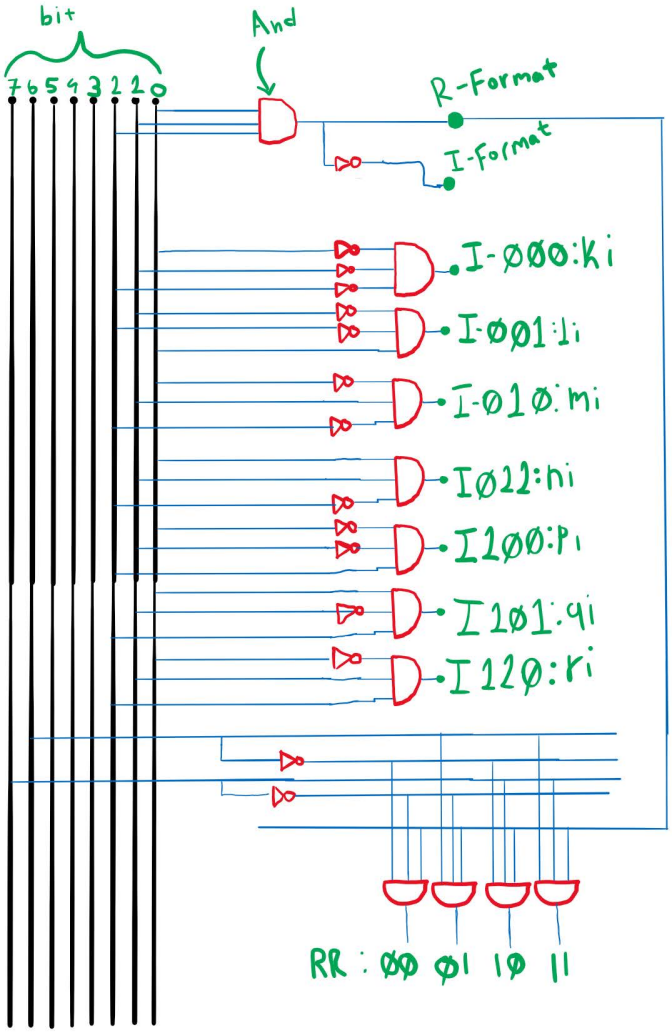
01 xxx 111

10 xxx 111

11 xxx 111

Μπορούμε να έχουμε μόνο αυτές τις τέσσερις και καμία παραπάνω εντολή R-format καθώς αφενός έχουμε μόνο τον συνδυασμό op 111 που περισσεύει και όλοι οι άλλοι είναι δεσμευμένοι και αφετέρου με τα 2 bits που μας δίνονται από τον funct μπορούμε να έχουμε 2^2 =4 συνδυασμούς με opcode 111 και funct 00,01,10,11.Επιπλέον έχουμε στην διάθεση μας μόνον 8 bits.

4.2c)



4.2d)

Έχοντας τώρα 6 εντολές με 8 bits εισόδου σημαίνει ότι έχουμε χρησιμοποιήσει ήδη τους 6 συνδυασμούς μας μένουν λοιπόν 2 bits που μπορούν να γίνουν R-Format και εφόσον έχουμε 2 bits funct έχουμε 8 συνδυασμούς των bit που να δίνουν R-Format εντολές. Αρά έχουμε 6 εντολές με

I-Format και 8 εντολές με R-Format οπότε σύνολο θα έχουμε 14 εντολές. Και θα είναι οι εξής :

FUNCT	R	OP
xx	xxx	000
xx	xxx	001
xx	xxx	010
xx	xxx	011
xx	xxx	100
xx	xxx	101
00	xxx	110
01	xxx	110
10	xxx	110
11	xxx	110
00	xxx	111
01	xxx	111
10	xxx	111
11	xxx	111

Εργαστήριο 5

Άσκηση 5.2)

Δεν υπάρχουν εντολές bgt , ble στον Risc-V καθώς δεν είναι αναγκαίες. Με τις ήδη υπάρχοντα εντολές μπορούμε να τις δημιουργήσουμε .Π.Χ.

riscv1.asm*

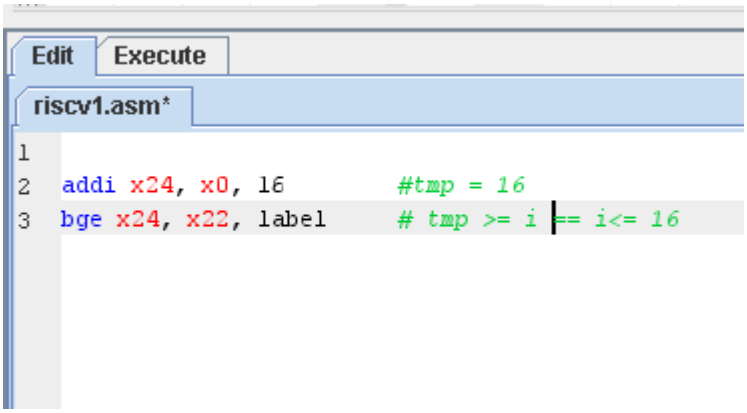
```
1
2  # if i <= j goto label
3  sub x24, x23, x22 # k = j-i
4  bge x24, x0, label # k >= 0
5
6  #if i > j goto label
7  sub x24, x23, x22  # k = j-i
8  blt x24, x0, label # k < 0
9
10 #OR
11 bge x23,x22, label # j >= i == i <= j
12
13 blt x23,x22, label # j < i == i > j
14
```

(β)

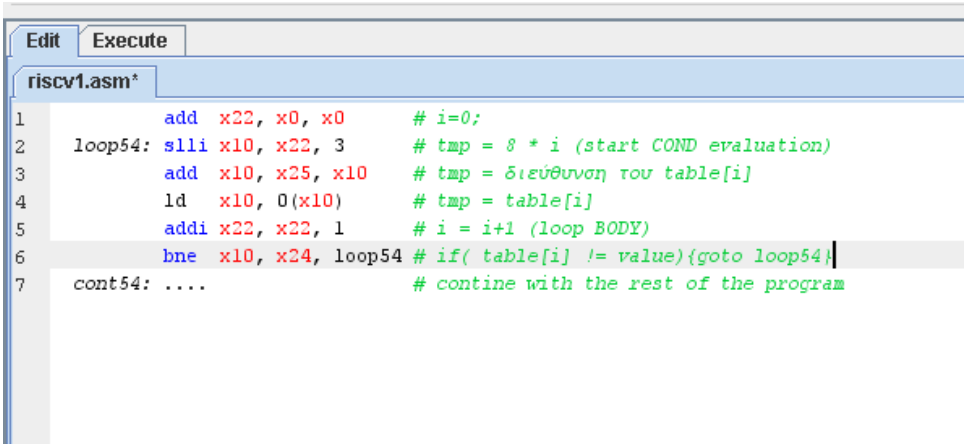
Δεν γίνεται να έχουμε εντολές με Immidiate σε branch καθώς στον Risc-V οι εντολές είναι 32 bits και έχουν B-Format. Άρα χρησιμοποιούν 5 bits για τον ένα καταχωρητη (rs1), 7 bits για τον opcode, 3 bits για το funct και 12 bits για το label καθώς και αυτό είναι ένας Imm. Οπότε περισσεύουν 32-(7+3+5+12) = 5 bit στα οποία δεν χωράει ένας Immidiate αριθμός και έτσι θα ξεπερνούσαμε τα bits.

(γ)

Για να δημιουργήσουμε μια εντολή σαν και αυτή δεν έχουμε πάρα να αποθηκεύσουμε την τιμή 13 σε ένα tmp καταχωρητη και να χρησιμοποιήσουμε την bge ανάποδα.



Άσκηση 5.5)



Αν υποθέσουμε ότι το BODY του loop επαναλαμβάνεται 10 φορές τότε στον παλιό κώδικα εκτελούνται 58 εντολές ενώ στον νέο κώδικα 50 και υπάρχει μια εντολή διακλαδώσεις μεταξύ κάθε επανάληψης.

Άσκηση 5.9)

riscv1.asm*

```
1
2  #ASKISI 5.9)
3  # i)
4  slt x26, x22, x23       # i < j
5  # ii)
6  slt x26, x22, x23       # i < j
7  xori x26, x26, 1        # NOT(i < j) == i >= j
8  # iii)
9  slt x26, x23, x22       # i > j
10 # iv)
11 slt x26, x23, x22       # i > j
12 xori x26, x26, 1        # NOT(i > j) == i <= j
13 # v)
14 slti x26, x22, CONST    # i < CONST
15 # vi)
16 slti x26, x22, CONST    # i < CONST
17 xori x26, x26, 1        # NOT( i < CONST) == i>= CONST
18 # vii)
19 slti x26, x22, CONST+1  # i < CONST+1
20 xori x26, x26, 1        # NOT( i < CONST + 1) == i >= CONST + 1 ( i > CONST )
21 # viii)
22 slti x26, x22, CONST+1  # i < CONST +1  == i <= CONST
23
24 #a)
25 sub x26, x22, x23
26 sltiu x26, x26, 1
27
28 #b)
29 sub x26, x22, x23
30 sltiu x26, x26, 1
31 xori x26,x26, 1
32
33 #c)
34 addi x26, x22, -CONST
35 sltiu x26, x26, 1
36
37 #d)
38 addi x26, x22, -CONST
39 sltiu x26, x26, 1
40 xori x26, x26 ,1
41
```