

Aσκηση 4.2

4a). Σε αυτή την περίπτωση δεν μπορούμε να εξουψιάσουμε καθιδικά εντολή R-format, επειδή κάθε εντολή θα "μεταφράζεται" ως I-format. Αυτό συμβαίνει επειδή έχουν εξαντληθεί όλες οι $8(2^3)$ επιλογές σε εντολές I-format και. Assembler είσι θα μπορεί να γράψει το R-format, και επομένως τα "βλέπει" όπως σε I-format. Εστώ λοιπόν ότι οι εντολές:

bi	000
ci	001
di	010
fi	011
gi	100
hi	101
li	110
mi	111

και οι κωνταχωρίτες:

x0	000
x1	001
x2	010
x3	011
x4	100
x5	101
x6	110
x7	111

Επομένως αν επιμένουμε και γράψουμε την εντολή σε R-format 'RRR' με $op=001$ και $funct=10$, τότε η εντολή "RR x3" θα γίνεται $\mu\epsilon$ την εντολή (στο I-format) $ci \ 1g \ \frac{\text{binary}}{\text{I-format}} \ \frac{\overbrace{10011001}^{\text{op}}}{\text{Imm}} \ \frac{\text{binary}}{\text{R-format}} \ \frac{\overbrace{10011001}^{x3}_{funct}}{\text{op}} = RRR \ x3$

Όπως το παραπάνω παραδείγμα, αυτό θα συμβαίνει σε οποιαδήποτε εντολή R-format έχουμε αφού o assembler θα 'διαβάσει' όλες τις εντολές ως I-format, διότι τα 3-LS bits ως opcode και τα επόμενα 5 bits ως αριθμοί (σε R-format θα έπρεπε αφού διαβάσει τα 3-LS bits ως opcode έπειτα να διαβάσει τα επόμενα 3 bits ως κωνταχωρίτες και τα 2-Ms bits ως funct). Αυτό συμβαίνει εξουψιά 'ξεδέψει' όλους τους 8 διαθέσιμους δυνητικούς του πεδίου op για εντολές I-format, o assembler δεν μπορεί να αναγνωρίσει εντολές R-format και επομένως τα 'μεταφράζει' όλα σε I-format.

4b) Εστια λογικόν τύπο όπως 'Θυροί/Γαρές' είναι συνδυαζόμενος με αυτό το I-format για το R-format. Ετοιμάζεται η επόμενη τις εντολές:

για I-format:

ci	001
di	010
fi	011
gi	100
hi	101
li	110
mi	111

για R-format:

	op	funct
cr	000	00
dr	000	01
fr	000	10
gr	000	11

Αν λογικόν 'Θυροί/Γαρές', ενώ αυτό τους 8 διαθέσιμους συνδυασμούς του πεδίου op για μια εντολή R-format και τις υπόλοιπους 7 συνδυασμούς τους έχουμε για το I-format, τότε όπως βλέπουμε θα έχουμε 4-εντολές για το R-format.

Ο op για κάθε R-format

Θα είναι ίδιος με 000 και το μόνο που θα αλλαγή

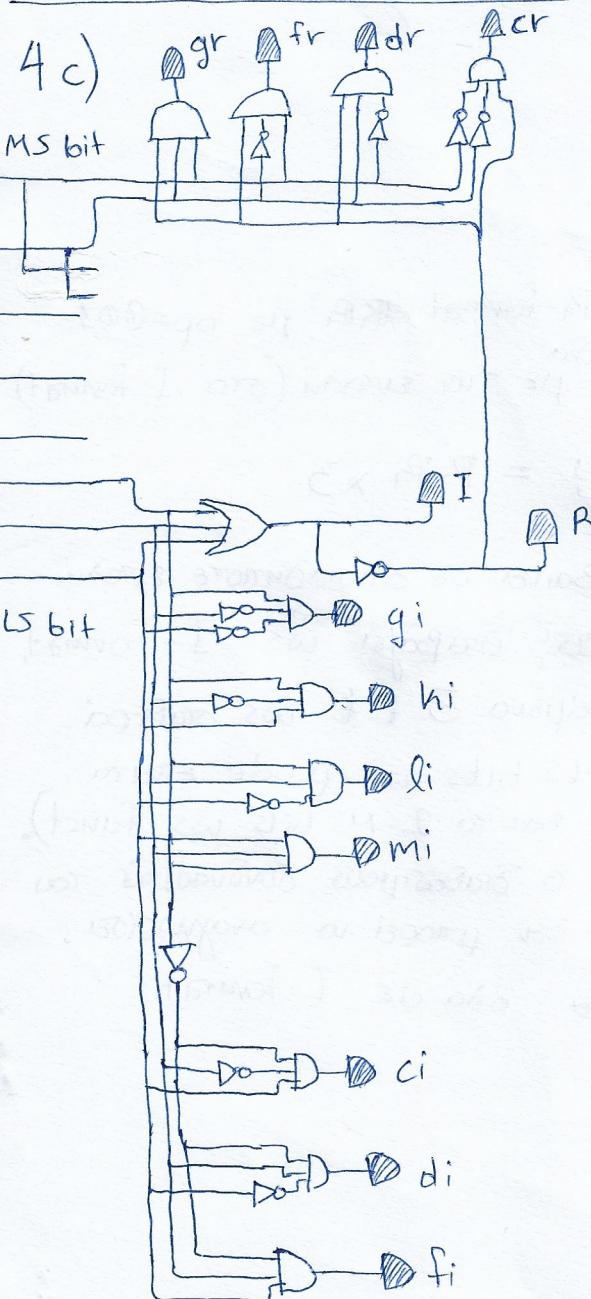
Θα είναι ο κωδικός του funct. Δεδομένου ότι ο

funct είναι μόνο 2-bits

Θα έχουμε $2^2 = 4$ εντολές

R format. Αλλά μπορούμε να έχουμε περισσότερες εντολές R-format επειδή δεν έχουμε αρκετά bits για νέους συνδυασμούς.

Καταλαβαίνει κανείς ότι για κάθε ~~μεταφέρει~~ εντολή εναντίο συνδυασμού op θυρών με το I format μεριδίζεται 4 συνδυασμούς (=εντολές) στο R-format.



4 d) Εάν έχουμε 6 εντολές I-format, τότε θα έχουμε 2 συμβασιών op για εντολές R-format. Από το 4 b) μαργίζουμε ότι για κάθε συμβασή op 'Έδειξαμε' από το I-format περδίζουμε 4 (2^2) εντολές R-format. Άυτο σημαίνει ότι με 2 συμβασιών op έχουμε $2 \cdot 2^2 = 2 \cdot 4 = 8$ εντολές σε R-format. Ετοι έχουμε τις εντολές:

Fia I-format (6 εντολές):

	op
ci	010
di	110
fi	001
gi	101
hi	111
li	111

Fia R-format
op. funct (8-εντολές)

	op.	funct
br	000	00
cr	000	01
dr	000	10
fr	000	11
gr	001	00
hr	001	01
lr	001	10
mr	001	11

a) Ο RISC-V δεν έχει την εντολή ble, ούτε την εντολή logt και αυτό συμβαίνει επειδή δεν τις χρειάζεται. Οι παραπάνω εντολές μπορούν εύκολα να γίνεται/αναπαρασταθούν όμως τη χρονικής αύλων εντολών διακρίσιμων.

Εστω $x_{22} = i$ και $x_{23} = j$

ble \rightarrow if($i \leq j$) go to label

Στο RISC-V θα μπορούσε

να γραφτεί ws if($j \geq i$) go to label

Δηλαδή

bge x_{23}, x_{22} , label

Άγου ~~ταχινή~~ η παρακάτω ισότιτη

$$i \leq j \Leftrightarrow j \geq i$$

bg t \rightarrow if($i > j$) go to label

Στο RISC-V θα μπορούσε

να γραφτεί ws if($j < i$) go to label

Δηλαδή

blt x_{23}, x_{22} , label

Άγου ~~ταχινή~~ η παρακάτω ισότιτη

$$i > j \Leftrightarrow j < i$$

B) Οι εντολές beq, bne δεν υπάρχει λόγος να υπάρχουν. Οι εντολές beq/bne κανουν αντίθετην ιδίαν έλεγχο. Οι beq/bne συγχέονται καθε bit του καταχωριτή rs1 με το αντίστοιχο bit (σε βαθύτερη σημασία) του καταχωριτή rs2. Αν έχουν τα bits ίδια τότε πηγαίνουν στο label αν beq και οχι αν bne. Αν από την αλλή τα bits τους δεν είναι ίδια τότε πηγαίνουν στο label αν bne και οχι αν beq. Αν υπήρχαν οι εντολές bequ/bneu θα συγκρίνονταν και αυτές καθε ~~είδους~~ bit του rs1 με rs2 (την ίδια 'διαδοχή' με τις beq/bne). Επειδή δοιπού δεν κάνουν κάτι διαφορετικό, είναι αίχνη το να έχουν bequ/bneu εντολές. Οι συγκεκριμένες εντολές beq/bne είναι ιδιαίτερα αριθμητικές και unsigned, αφού η ισότιτη απαιτεί όλα τα bits να είναι ίδια.

y) Συμφωνα με το format για τις εντολές branch του RISC-V, δεν υπάρχει διαθέσιμος χώρος για να έχουμε εναν αρκετά μεγάλο σταθερό αριθμό. Αν αντικαταστήσουμε τα bits του καταχωριτή rs1 από τους 2 καταχωριτές τοτε θα μπορούσαμε να συγκρίνουμε αριθμους (5 bits) $2^5 = 32 \rightarrow$ από -16 μέχρι 15 που είναι επαρκώς περιορισμένος γεγονός αριθμών. Έτσι εκ τούτου, επειδή δεν χρειάζεται "usable" σταθερη πλούτη, δεν υπάρχουν εντολές τύπου beq i κτλ.

*Συνέχεια B) ενα πρόσω που είναι ανεξαρτήτως τη παρούσαι ο καθε συνδυασμός από bits. Άρα οι εντολές beq και bne θα μπανε σε όλης ισοδύναμες με τις εντολές beq και bne που υπάρχουν.

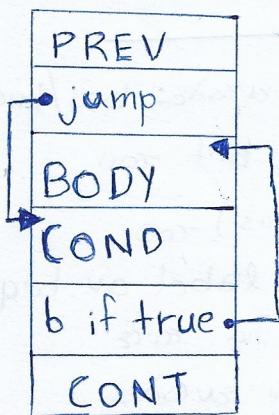
δ) Γνωρίζοντας ότι δεν υπάρχει η εντολή ble (\leq) και δεν μπορούμε να κανουμε αγκύρα με σταθερή ποσότητα, θα πρέπει να κανουμε καταλληλες μικρές αλλαγές ώστε να μεταφράσουμε τον φυσικό κώδικα "branch if i ≤ 13 " σε Assembly των RISC-V. Αρχικά θα αποθηκεύσουμε την σταθερά 13 σε εναν καταχωριτη (εστω x23). Επειτα θα κανουμε $13 \geq i$ δηλαδη την εντολή bge. Να σημειωθεί πως οι αποθηκευσης της σταθερας 13 στο προσωρινό καταχωριτη θα γίνει μεσω της εντολής addi. Εποι έχουμε: εστω x22 = i text.

addi x23, x0, 13 # Αποθηκευση στο καταχωριτη x23 = 13

bge x23, x22, label # if $13 \geq i$ go to label

$$13 \geq i = i \leq 13$$

Assembly 5.5



add t1, x0, x0 # $i=0$
 j cond
 body: addi t1, t1, 1 # $i = i + 1$ (loop BODY)
 cond: slli t0, t1, 3 # $tmp = 8 * i$ (start COND evaluation)
 add t0, s0, t0 # $tmp = \text{διάλυθμον του } table[i]$
 ld t0, 0(t0) # $tmp = table[i]$
 bne t0, t2, body # if $table[i] \neq \text{key}$ goto body (continue loop)
 # continue with the rest of the program

Με τον νέο κιβώτιο αποφεύγουμε την εκτέλεση της εντολής jump σε κάθε ανακύκλωση. Ως εκ τούτου, αν το σώμα των πικρατικών βρόχου επαναλαμβάνεται 10 φορές τότε με το νέο κιβώτιο θα εκτελούνται συνολικά 9 λιγότερες εντολές ~~σε σχέση με το παλαιό~~ (9 φορές διότι την πρώτη φορά ^{του} θα εκτελεστεί το BODY θα εκτελεστεί και το jump αναγκαστικά στην είσοδο του βρόχου) απότινα από τις 10 εκτελέσεις στην μία θα εκτελέσουν και του jump, αρα $10 - 1 = 9$ φορές δεν θα εκτελεστεί το jump) Δεξιόμενο ότι στο παλαιό κιβώτιο το jump πάντα εκτελείται (σε καθε loop) στη νέα εκτελείται μόνο την πρώτη φορά (αρα $10 - 1 = 9$ φορές δεν θα εκτελεστεί \rightarrow 9 ~~εκτελέσεις~~ λιγότερες εντολές από την παλαιά).

Aonnon 5.10

i. $B = (i < j)$

slt x26, x22, x23

ii. $B = (i >= j)$

slti x26, x22, x23

xori x26, x26, 1

iii. $B = (i > j)$

slt x26, x23, x22

iv. $B = (i <= j)$

slt x26, x23, x22

xori x26, x26, 1

v. $B = (i < \text{CONST})$

slti x26, x22, CONST

vi. $B = (i >= \text{CONST})$

slti x26, x22, CONST

xori x26, x26, 1

vii. $B = (i > \text{CONST})$

slti x26, x22, CONST

xori x26, x26, -1

viii. $B = (i <= \text{CONST})$

~~slti x26, x22, CONST~~

slti x26, x22, CONST + 1

xori x26, x26, -1

a. $B = (i == j)$

sub x22, x22, x23

sltiu x26, x22, 1

b. $B = (i != j)$

sub x22, x22, x23

sltu x26, x0, x22

c. $B = (i == \text{CONST})$

addi x22, x22, -CONST

sltiu x26, x22, 1

d. $B = (i != \text{CONST})$

addi x22, x22, -CONST

sltu x26, x0, x22

vii. $B = (i > \text{CONST})$

~~slti x26, x22, CONST~~

slti x26, x22, CONST + 1

viii. $B = (i <= \text{CONST})$

slti x26, x22, CONST + 1