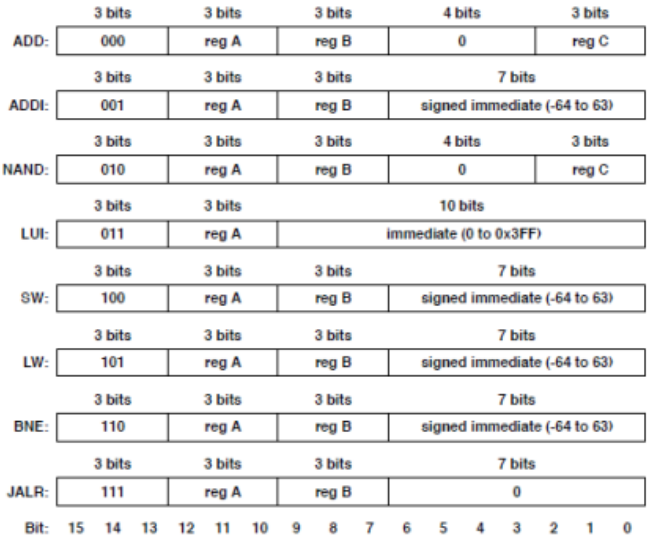
Συστήματα VLSI

Εργαστηριακή άσκηση 2: Επεξεργαστής

Γιώργος Δημητρακόπουλος

Στο εργαστήριο αυτό καλείστε να σχεδιάσετε σε VHDL και να υλοποιήσετε στην FPGA ένα μικρό επεξεργαστή των 16bit. Ο επεξεργαστής αποτελείται από 8 καταχωρητές των 16bit ο καθένας, ενώ κάθε διεύθυνση αναφέρεται και αυτή σε μια ποσότητα των 16bit. Ο καταχωρητής 0 περιέχει πάντα την τιμή 0. Οποιαδήποτε ανάγνωση του καταχωρητή 0 επιστρέφει την τιμή 0 η οποία δε μπορεί να αλλάξει από καμία εγγραφή. Το σύνολο εντολών του επεξεργαστή αποτελείται από 8 εντολές. Τα πεδία της κάθε εντολής μαζί με τον ορισμό της και την περιγραφή της λειτουργίας της φαίνεται στους πίνακες που ακολουθούν:



Assembly-Code		Meaning
add	regA, regB, regC	R[regA] <- R[regB] + R[regC]
addi	regA, regB, immed	R[regA] <- R[regB] + immed
nand	regA, regB, regC	R[regA] <- ~(R[regB] & R[regC])
lui	regA, immed	R[regA] <- immed & 0xffc0
sw	regA, regB, immed	R[regA] -> Mem[R[regB] + immed]
lw	regA, regB, immed	R[regA] <- Mem[R[regB] + immed]

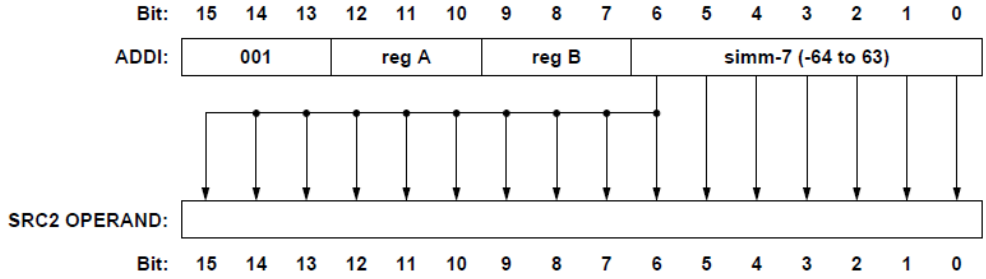
bne	regA, regB, immed	if (R[regA] != R[regB]) {
		PC <- PC + 1 + immed
		(if label, PC <- label)
		}

jalr	regA, regB	PC <- R[regB], R[regA] <- PC + 1

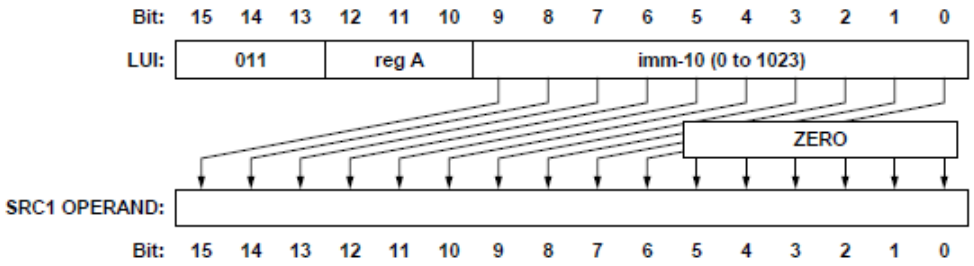
Assembly Format	Action
add rA, rB, rC	Add contents of **regB** with **regC**, store result in **regA**.
addi rA, rB, imm	Add contents of **regB** with **imm**, store result in **regA**.
nand rA, rB, rC	Nand contents of **regB** with **regC**, store results in **regA**.
lui rA, imm	Place the 10 ten bits of the 16-bit **imm** into the 10 ten bits of **regA**, setting the bottom 6 bits of **regA** to zero.
sw rA, rB, imm	Store value from **regA** into memory. Memory address is formed by adding **imm** with contents of **regB**.
lw rA, rB, imm	Load value from memory into **regA**. Memory address is formed by adding **imm** with contents of **regB**.
bne rA, rB, imm	If the contents of **regA** and **regB** are not the same, branch to the address PC+1+**imm**, where PC is the address of the bne instruction.
jalr rA, rB	Branch to the address in **regB**. Store PC+1 into **regA**, where PC is the address of the jalr instruction.

Οι προς εκτέλεση εντολές είναι αποθηκευμένες στη μνήμη εντολών και η επόμενη προς εκτέλεση εντολή καθορίζεται από τη διεύθυνση που αποθηκεύει ο program counter (PC). Η επικοινωνία με τη μνήμη δεδομένων γίνεται μέσω των εντολών load και store ενώ η εντολή jalr μπορεί να χρησιμοποιηθεί για την εκτέλεση συναρτήσεων. Ως καταχωρητής επιστροφής από συνάρτηση μπορείτε να χρησιμοποιήσετε τον καταχωρητή r7.

Η εκτέλεση των εντολών με ορίσματα σταθερές τιμές όπως η addi, lui, lw, sw, bne απαιτεί την επέκταση προσήμου στα 16bit των σταθερών τιμών πριν αυτές χρησιμοποιηθούν. Για παράδειγμα στην περίπτωσης της addi η σταθερή τιμή των 7 bit πρώτα επεκτείνεται στα 16 bit σύμφωνα με το παρακάτω σχήμα και μετά χρησιμοποιείται σαν όρισμα της πρόσθεσης.



Με παρόμοιο τρόπο υλοποιείτε η επέκταση κρατουμένου στην περίπτωση των εντολών load, store και bne. Αντίθετα η lui είναι κάπως διαφορετική. Στην περίπτωση αυτή ο καταχωρητής reg A στον ορισμό της lui δέχεται τα 10bit της σταθεράς που είναι αποθηκευμένα μέσα στην εντολή αφού όμως πρώτα αυτά έχουν τοποθετηθεί στις 10 πιο σημαντικές θέσεις της 16bit λέξης. Τα 6 λιγότερο σημαντικά ψηφία είναι ίσα με 0.



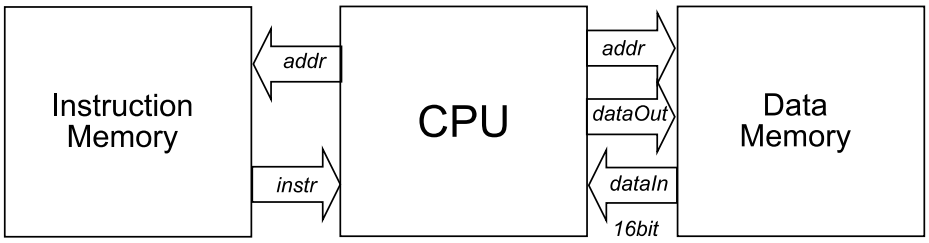
Η φόρτωση μιας 16bit σταθεράς απαιτεί πέρα από τα 10 περισσότερο σημαντικά ψηφία που καλύπτει η lui τη φόρτωση και των 6 λιγότερο σημαντικών ψηφίων της σταθεράς. Αυτό μπορεί να συμβεί μέσω της addi. Αν για παράδειγμα θέλουμε φορτώσουμε στον καταχωρητή X τα 6 λιγότερα ψηφία της σταθεράς imm αρκεί να εκτελέσουμε addi X,X,imm6 όπου στη θέση του imm6 έχουμε βάλει την τιμή σταθερά & 0x3f .

Οι 8 εντολές του επεξεργαστή αν και λίγες ουσιαστικά καλύπτουν ένα ευρύ φάσμα από πιθανές εφαρμογές. Λειτουργίες οι οποίες δεν υποστηρίζονται απευθείας από το σύνολο εντολών, όπως η αφαίρεση, μπορούν εκτελεστούν σε περισσότερους κύκλους ρολογιού χρησιμοποιώντας περισσότερες από μία εντολές. Για παράδειγμα η αφαίρεση R4 <= R1-R2 μπορεί να υλοποιηθεί ως Reg[1]+~Reg[2]+1 ως εξής:

```
NAND R3, R2, R2   // R3 <= ~R2
ADDI R3, R3, 1    // R3 <= ~R2+1
ADD  R4, R1, R3   // R4 <= R1+~R2+1 (=R1-R2)
```

Ο επεξεργαστής σας καλό είναι να εκτελεί την κάθε εντολή σε πολλούς κύκλους ρολογιού ανάλογα με την οργάνωση της FSMD που εσείς θα επιλέξετε. Ο επεξεργαστής επικοινωνεί με τη μνήμη εντολών και

δεδομένων χρησιμοποιώντας τους διαδρόμους διευθύνσεων και δεδομένων που φαίνονται στο σχήμα που ακολουθεί.



Η υλοποίηση των μνημών εντολών και δεδομένων που θα χρειαστείτε θα σας δοθεί νωρίτερα. Για το δικό σας πειραματισμό κατά τη φάση της σχεδίασης μπορείτε να χρησιμοποιείσετε μοντέλα μνημών σε VHDL σαν αυτά του αρχείου καταχωρητών των σημειώσεων σας. Η μνήμη εντολών μπορεί να είναι απλά μια ROM (ένα μεγάλο case statement).

Για την επίδειξη της ορθής λειτουργίας του κυκλώματος σας καλείστε να υλοποιήσετε μικρά προγράμματα σε assembly τα οποία είτε να οδηγούν διευθύνσεις/δεδομένα (μέσω load/store εντολών) στα 7segment displays της πλακέτας ή να οδηγούν καταχωρητές του VGA controller αλλάζοντας σε software το χρώμα που προβάλει.

Προσέξτε απλά το αποτέλεσμα της επεξεργασίας να φαίνετια στο ανθρώπινο μάτι. Αν διαρκεί λίγους κύκλους ρολογιού με ρυθμό 50MHz δε θα είναι εφικτό.