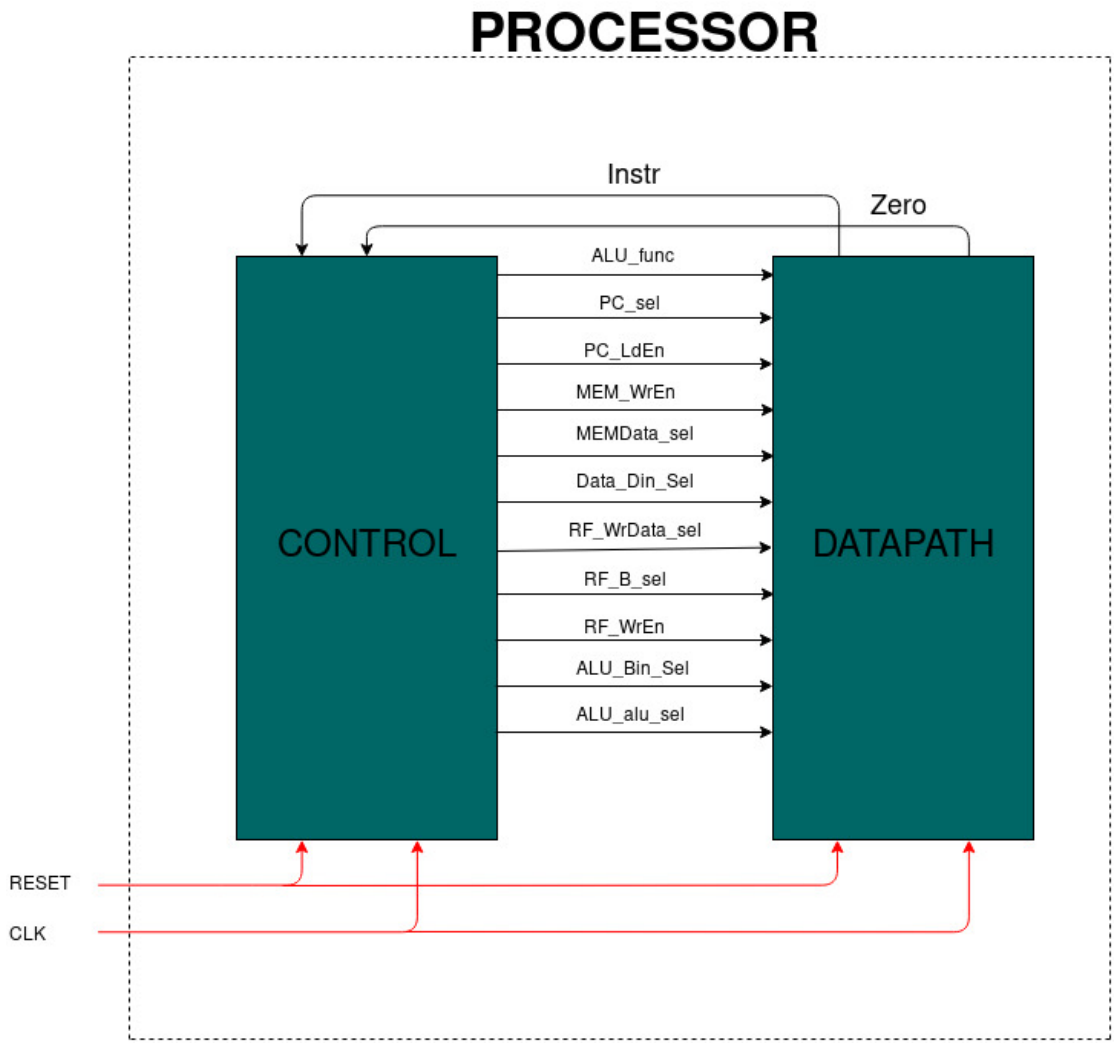
Προεργασία 4ου Εργαστηρίου

Ομάδα Εργασίας:	**LAB31239629**
Κονιδάρη Ηρώ	**Α.Μ. 2012030049**
Μάνεσης Αθανάσιος	**Α.Μ. 2014030061**

Σκοπός της Άσκησης

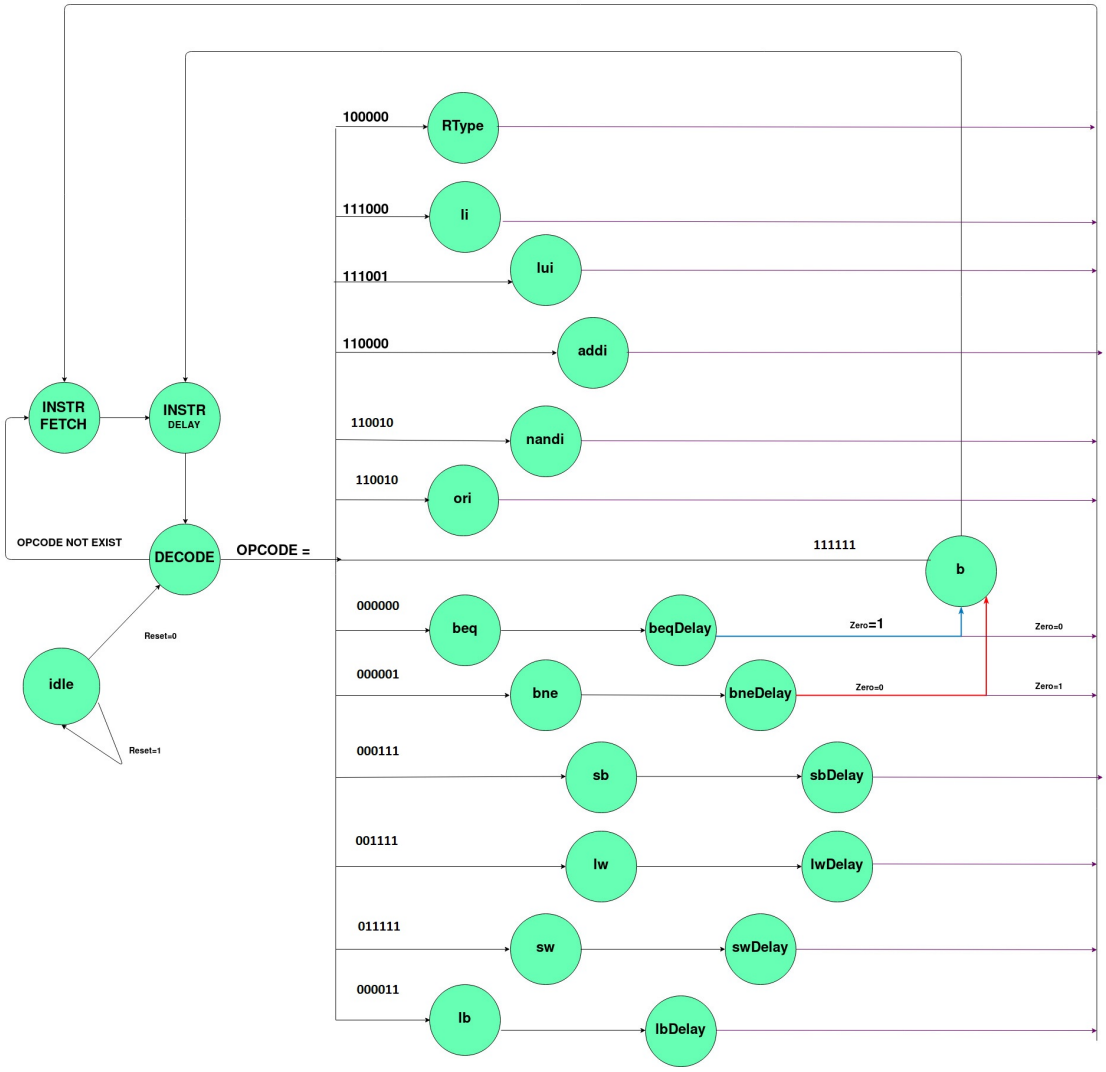
Ολοκλήρωση επεξεργαστή και προσθήκη exceptions.

Περιγραφή/Υλοποίηση της Άσκησης



Για την ολοκλήρωση του processor, προσθέσαμε στο datapath δύο registers 32 bit, ώστε να κρατάνε τις τιμές των RF_A, RF_B.

Για το control σχεδιάσαμε την παρακάτω fsm.



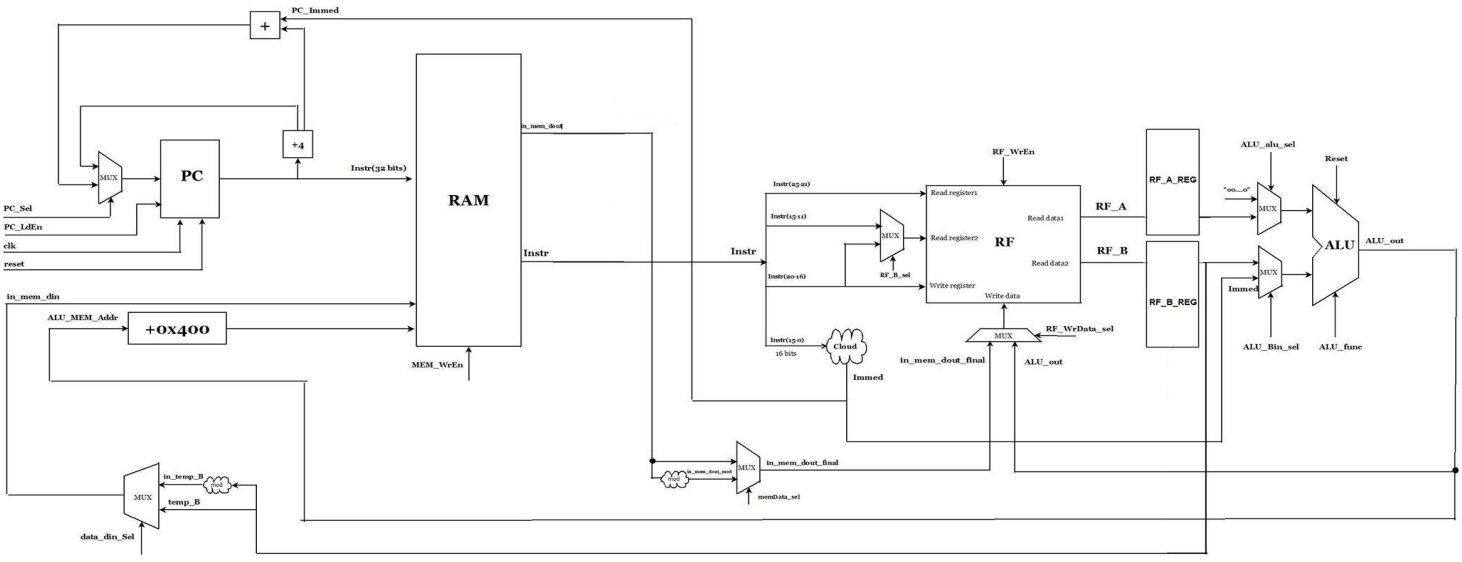
Η FSM χρησιμοποιώντας το opcode της εντολής και το zero της ALU παράγει τα κατάλληλα σήματα για την σωστή εκτέλεση της εντολής σε multicycle.

Ανάλυση καταστάσεων:

1. **Rtype instructions:** Η fsm ξεκινάει από την κατάσταση InstFetch στην οποία γράφεται το PC=PC+4(PC_sel=0) στον καταχωρητή PC (PC_LdEn=1). Μετά στην InstrDelay όπου με καθυστέρηση ενός κύκλο ρολογιού περιμένουμε να έρθει η εντολή από την μνήμη. Ακολουθεί η κατάσταση Decode όπου αποκωδικοποιούμε το instruction και πέρνουμε τα δεδομένα που χρειαζόμαστε για τους καταχωρητές rt, rs. Ξέροντας τα

opcode και ALU_func η fsm ενεργοποιεί τα κατάλληλα control σήματα της ALU.

2. **Itype instructions:** Ομοίως με τις R-type, μόνο που επιλέγεται το Immediate ως δεύτερος τελεσταίος στην ALU(ALU_bin_sel=1).
3. **Branch instructions:** Διαφοροποίειται στο IfStage αφού αν λάβουμε μια εντολή b τότε PC_LdEn=1, PC_sel=1 και γράφεται στον PC το PC+4+4*PC_Immed. Στις εντολές bne , beq πρέπει να γίνει η σύγκριση(sub) μεταξύ των δύο καταχωρητών ώστε να αποφασίσουμε ποια θα είναι η καινούρια τιμή του PC. Για αυτό έχουμε προσθέσει μία delay κατάσταση. Ανάλογα με την τιμή του zero θα αποφανθούμε για την καινούριο τιμή του PC.
4. **Store-Load instructions:** Ομοίως με Itype εντολές, τα κομμάτια δέσμευσης εντολής από την μνήμη και η αποκωδικοποίηση της εντολής είναι ίδια. Σε τέτοιου τύπου εντολές το ALU_FUNC=0000 (add), οπότε επίσης χρειαζόμαστε delay ώστε να γίνει η πράξη.



Κυματομορφές

Παρατηρούμε ότι όλες οι εντολές αποκωδικοποιούνται και εκτελούνται σωστά, καθώς η εγγραφές στην RF ή στην μνήμη γίνονται σωστά. Επίσης μπορούμε να δούμε και το κομμάτι του infinite loop όπου το PC πηγαίνει συνεχώς από το 32 στο 36.

Το simulation αφορά το παρακάτω πρόγραμμα:

```
addi r5,r0,8                // $R5=00000008
ori r3,r0,ABCD              // $R3=0000abcd
sw r3,4(r0)                 // RAM[1028] = 0000abcd
lw r10,-4(r5)               // $R10 = 0000abcd
lb r16,4(r0)                // $R16 = 000000cd
nand r4,r10,r16             // $R4 = ffffff32
not r1, r5                  // $R1 = fffffff7
ror r2,r4                   // $R2 = 7ffff99
bne r5,r5,8                 // false, pc=pc+4
b -2 (infinite loop)        // infinite loop, back to bne
add r1.r2,r3                // not executed
```