Αναφορά Εργαστηριακής Άσκησης 2

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΜΜΥ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

ΕΡΓΑΣΤΗΡΙΑΚΕΣ ΑΣΚΗΣΕΙΣ ΓΙΑ ΤΟ ΜΑΘΗΜΑ:
ΗΡΥ 302 – Οργάνωση Υπολογιστών

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2025

Κύρκος Κωνσταντίνος 2022030112 „Χαράλαμπος Μυλωνάκης 2022030133
Ομάδα:24

1)Σκοπός της Εργασίας. Ο σκοπός της δεύτερης εργαστηριακής άσκησης ήταν να μετατρέψουμε τον επεξεργαστή που υλοποιήσαμε στην πρώτη άσκηση από ενός σε πολλών κύκλων. Για την προαναφερθείσα υλοποίηση χρησιμοποιήθηκαν καταχωρητές ενδιάμεσα από τις βαθμίδες, ώστε να διατηρούνται οι τιμές των σημάτων που παράγει η κάθε μια τους. Έπειτα σχεδιάσαμε εκ νέου την μηχανή πεπερασμένων καταστάσεων στο ροντρολ πατη του επεξεργαστή, ώστε να δίνει τα σωστά σήματα εκτέλεσης ανάλογα με την δοθείσα εντολή.

2) Περιγραφή σχεδιασμού.

Όπως προαναφέρθηκε ο επεξεργαστής πολλών κύκλων χρησιμοποιεί ρεγιστερς ανάμεσα στις βαθμίδες, ώστε τα αποτελέσματα της κάθε μιας να μην χάνονται. Από αυτό προκύπτει ότι μια εντολή της οποίας η εκτέλεση δεν απαιτεί όλες τις βαθμίδες θα εκτελεστεί πιο γρήγορα (Σε λιγότερους κύκλους ρολογιού), ενώ μια που χρησιμοποιεί όλες τις βαθμίδες θα χρειαστεί τον μέγιστο χρόνο εκτέλεσης. Παρακάτω φαίνονται όλες οι βαθμίδες του επεξεργαστή πολλών κύκλων.

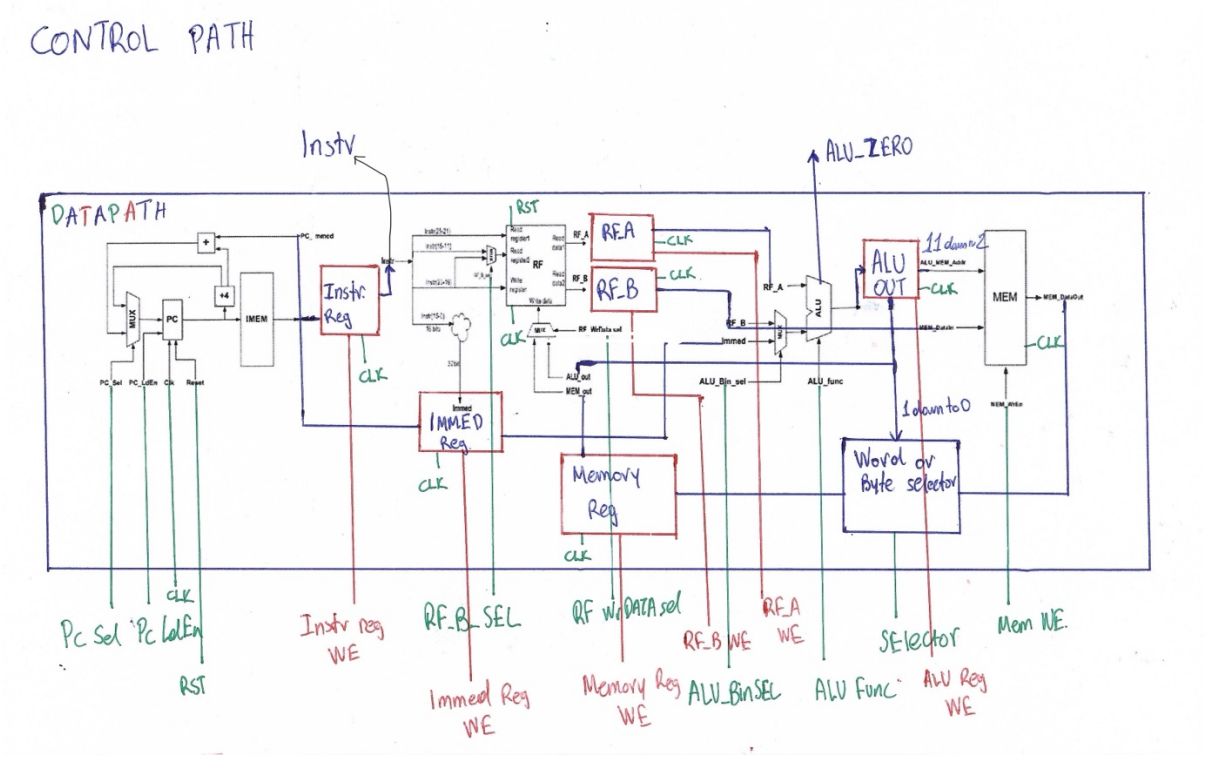
1) $If\ state$: Φορτώνεται η εντολή από την ROM στον $Program\ Counter$.
2) $Dec\ state$: Αποκωδικοποιείται η εντολή που ήρθε από τον $Program\ Counter$.
3) $Exec\ state$: Διαχωρισμός εντολών σε:
   A) $R-type$, li, lui, $addi$, $andi$,ori : Πράξη ALU
   B) B, Beq, Bne : Υπολογισμός διεύθυνσης επόμενης εντολής.
   Γ) lb, lw, sw : Υπολογισμός διεύθυνσης μνήμης
4) $Mem\ state$: εκτελείται το lb, lw, sw με την δοθείσα από το $exec\ state$ διεύθυνση μνήμης.
5) $Write\ back\ state$: Εκτελείται η εγγραφή στο αρχείο καταχωρητών.

6) *Bne, Beq initial state*: Ανάλογα το *ALU Zero* του *Exec state* αποφασίζεται αν ο επεξεργαστής θα μεταπηδήσει στην επόμενη εντολή ή αν θα κάνει *Branch*.

Για να επιτευχθεί η υλοποίηση που προαναφέρθηκε επαναχρησιμοποιήσαμε τα Register Modules που είχαμε ήδη δημιουργήσει για το Register File και δημιουργήσαμε τους επιπλέον 6 καταχωρητές:

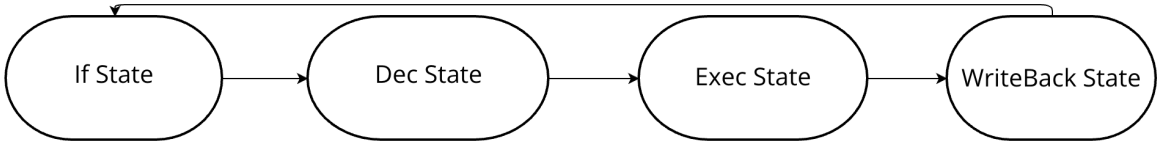
1) *Instruction Register*.
2) *Immed Register*.
3) *RF A Register*.
4) *RF B Register*.
5) *ALU Out Register*.
6) *WorB Register*.

Παρακάτω φαίνεται το ενιαίο *Data Path* του επεξεργαστή πολλών κύκλων. Με κόκκινο φαίνονται τα *registers* που προσθέσαμε.

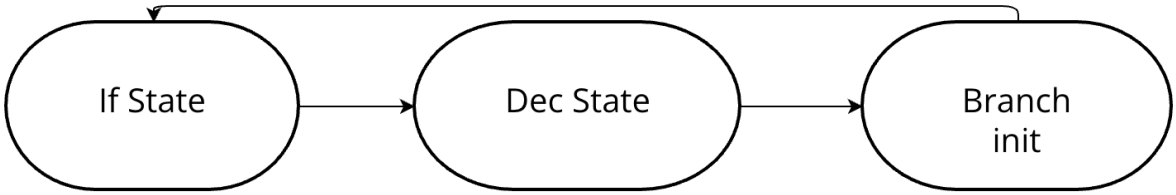


Όπως είπαμε η κάθε εντολή χρειάζεται διαφορετικό χρόνο εκτέλεσης. Αυτό το μέρος το αναλαμβάνει η νέα μηχανή πεπερασμένων καταστάσεων του επεξεργαστή. Παρακάτω φαίνεται η διαδρομή της κάθε εντολής από την αρχή μέχρι έως ότου εκτελεστεί. Κάθε *State* είναι και ένας κύκλος του ρολογιού.

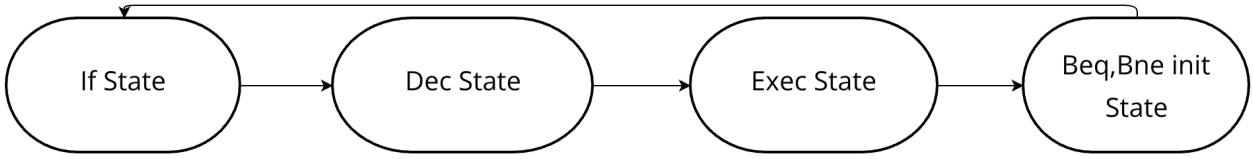
- *Rtype*, *li*, *lui*, *addi*, *andi*, *ori* (4 κύκλοι)



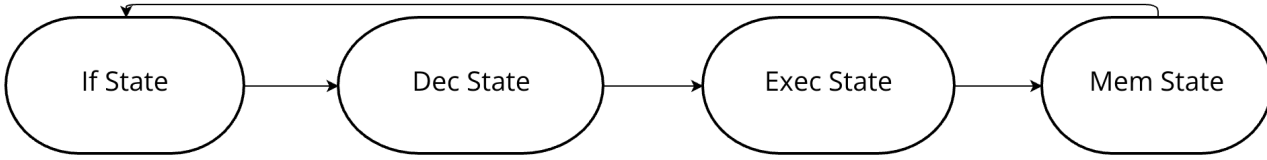
- *Branch* (3 κύκλοι)



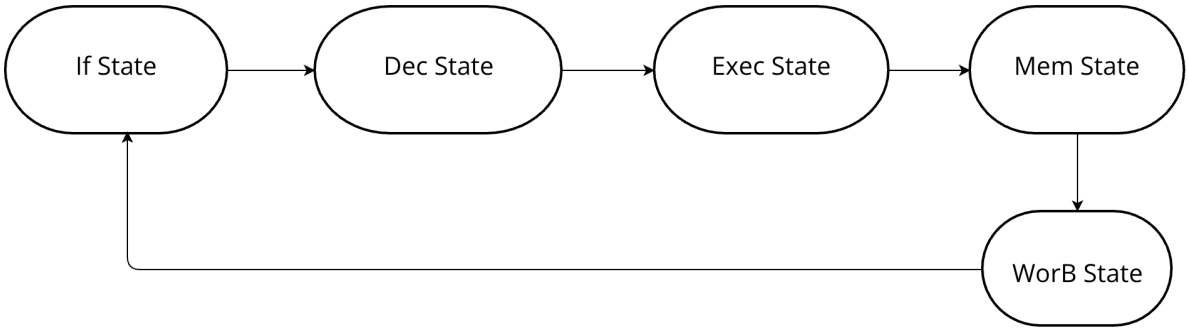
- *Branch equal*, *Branch not equal* (4 κύκλοι)



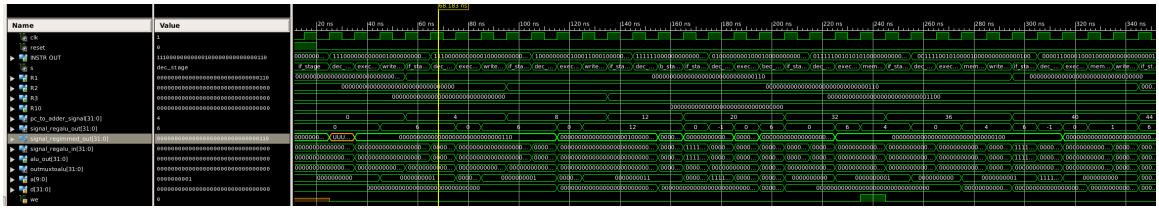
- *Store word* (4 κύκλοι)



- *Load word*, *Load Byte* (5 κύκλοι)



Τέλος δημιουργήθηκε ενδελεχές *test bench* για την επαλήθευση της ομαλής λειτουργίας του επεξεργαστή πολλών κύκλων.



3) Συμπεράσματα.

Σε αυτή την άσκηση κατασκευάσαμε έναν επεξεργαστή πολλών κύκλων. Κατανοήσαμε περαιτέρω την αρχιτεκτονική πίσω από αυτόν, καθώς και την ροή δεδομένων μέσα του. Κατανοήσαμε ότι για να έχουμε γρήγορο ρολόι ο επεξεργαστής πολλών κύκλων είναι απαραίτητος, καθώς στον ενός κύκλου για να ολοκληρωθούν οι πιο χρονοβόρες διεργασίες π.χ. lb/lw χρειάζεται αρκετά μεγάλη περίοδος του ρολογιού (χαμηλό $clock\ frequency$).