2η Αναφορά – Οργάνωση Υπολογιστών

Σοφία Καφρίτσα Γεωργαντά, 2016030136

Σκοπός 2ου μέρους

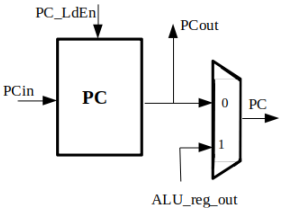
Σκοπός των επόμενων φάσεων είναι η βελτίωση ενός single cycle επεξεργαστή μέσω δύο διαφορετικών μεθόδων. Πρώτα θα υλοποιήσω τη μέθοδο μετατροπής του επεξεργαστή σε multi-cycle επεξεργαστή κι, έπειτα, θα μετατρέψω τον αρχικό επεξεργαστή σε pipeline επεξεργαστή.

Φάση 4η

- DATAPATH_MC

Για την υλοποίηση πρόσθεσα 5 καταχωρητές, με σκοπό να ξεχωρίσω την κάθε εντολή σε στάδια, κάθε στάδιο και ένας ξεχωριστός παλμός ρολογιού. Η θέση των καταχωρητών καθορίζεται από το ποιες συνδυαστικές μονάδες θα χωρέσουν σε έναν κύκλο ρολογιού και ποια δεδομένα χρειάζονται σε επόμενους κύκλους που υλοποιούν την εντολή. Έτσι, ένας είναι για το Instr, δύο για τις εξόδους του DECSTAGE, ένας για το αποτέλεσμα της ALU και ένας για το αποτέλεσμα των δεδομένων μνήμης. Με αυτή την λογική, κάποιες εντολές δεν βρίσκονται στο critical path. Το critical path προκύπτει από τις εντολές lw/lb, οι οποίες χρειάζονται 5 κύκλους ρολογιού για να ολοκληρωθούν, καθώς χρησιμοποιούν όλες τις βαθμίδες του datapath. Αντιθέτως, οι εντολές branch χρειάζονται 3 κύκλους ρολογιού για να ολοκληρωθούν, καθώς δεν απαιτείται MEMSTAGE και WRITE_BACK_STAGE, με αποτέλεσμα να μη γίνεται πλήρης χρήση του datapath, όπως συμβαίνει σε single-cycle επεξεργαστή.

Σχόλια:



1. Γι' αυτή την υλοποίηση χρειάστηκαν κάποιες αλλαγές στα βασικά components, όπως το IFSTAGE και EXSTAGE. Επειδή η ALU δεν χρησιμοποιείται στους 2 πρώτους κύκλους κάθε εντολής (IFSTAGE, DECSTAGE) την χρησιμοποιώ παράλληλα με τους 2 πρώτους κύκλους ως εξής: όταν εκτελείται το IFSTAGE θα χρησιμοποιώ την ALU για την πράξη PC=PC+4, ενώ όταν εκτελείται το DECSTAGE θα χρησιμοποιώ την ALU για την πράξη PC=(PC+4)+Immed. Η πράξη (PC+4) έχει ήδη υπολογιστεί στον 1ο κύκλο. Έτσι, αφαίρεσα από το IFSTAGE τους αθροιστές και άλλαξα τη θέση στον πολυπλέκτη.
2. Στη συνέχεια, μετασχημάτισα το EXSTAGE, προσθέτοντας έναν πολυπλέκτη σε κάθε είσοδο της ALU. Ο πρώτος έχει δύο εισόδους, την τιμή που έχει ο PC στο IFSTAGE και την έξοδο του καταχωρητή RF_A, ο οποίος είναι η πρώτη έξοδος του RF. Ο δεύτερος καταχωρητής έχει 4 εισόδους, όπου στην πρώτη συνδέω το RF_B, στην δεύτερη συνδέω τη σταθερά 4 που χρησιμοποιείται για την πρόσθεση του PC+4, στην τρίτη συνδέω το Immed, όπως προκύπτει από το DECSTAGE και η τέταρτη είσοδος γειώνεται. Η έξοδος της ALU συνδέεται πριν και μετά τον καταχωρητή σε έναν ακόμη πολυπλέκτη 2 εισόδων, ο οποίος ελέγχεται από το σήμα PC_Source και θα χρησιμοποιηθεί για να συνδεθεί η σωστή τιμή με τον καταχωρητή του PC για τις πράξεις PC=PC+4 και PC=(PC+4)+Immed. Αναλύεται και παρακάτω.

- CONTROL_MC

Για το control σε αυτή την υλοποίηση, χρησιμοποίησα μία παρατήρηση που είχα κάνει στις προηγούμενες φάσεις της εργασίας: το ByteOp για τις εντολές sw, sb, lw, lb αντιστοιχεί στο NOT(OpCode(3)) και το PC_Sel θέλουμε να είναι '1'μόνο όταν έχουμε valid branch εντολές, δηλαδή συνδυάζοντας το ALU_zero με τα beq,b και το NOT(ALU_zero) με το bne. Για να είναι κατανοητή και ξεκάθαρη η υλοποίηση, όρισα δύο σήματα tempbne, tempbeq τα οποία είναι ενεργά όταν εκτελούνται bne και b,beq αντίστοιχα, και δουλεύω με αυτά, όπως φαίνεται παρακάτω:

tempPC_Sel<=((zero and t_tempbeq) or ((not zero) and t_tempbne)) after 4 ns;

Στη συνέχεια, το τελικό CONTROL_MC αποτελείται από το CONTROL_ ALU και το CONTROL_FSM, έχοντας κάνει αυτό τον διαχωρισμό και για τις δύο υλοποιήσεις. Στο CONTROL_ ALU υπολογίζω την τιμή της ALU_func όπως εξήγησα στην προηγούμενη αναφορά, εφαρμόζοντας κάποιες ελαχιστοποιήσεις, όπως φαίνεται στον παρακάτω κώδικα.

```
process(func,ALU_op)
   begin
      temp(3)<=ALU_op(3) and func(3);
      temp(2)<=((not ALU_op(3)) and ALU_op(2)) or (ALU_op(3) and func(2));
      temp(1)<=((not ALU_op(3)) and (not ALU_op(2)) and ALU_op(1)) or (ALU_op(3) and func(1));
      temp(0)<=((not ALU_op(3)) and ALU_op(0))  or (ALU_op(3) and func(0));
end process;

ALU_func<=temp;
```

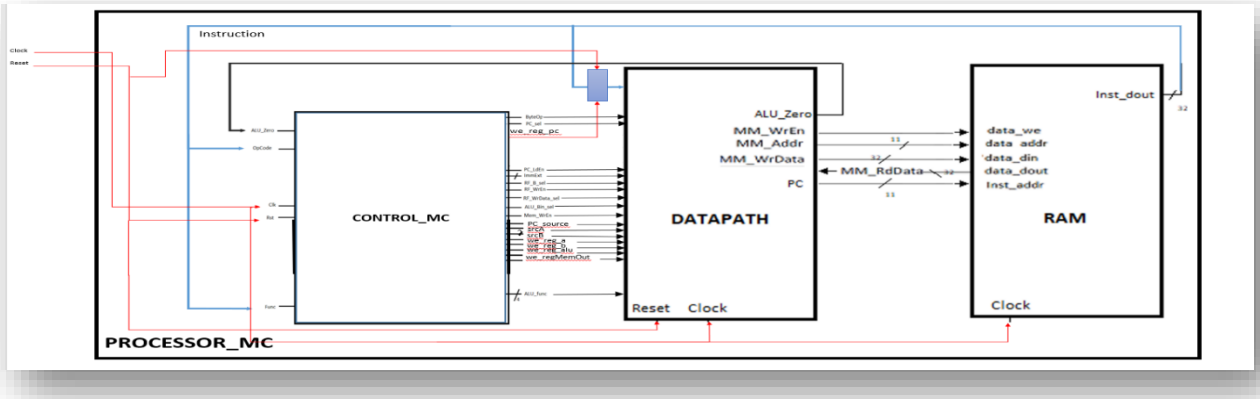
Η βασική λειτουργικότητα του Control υλοποιήθηκε μέσω μίας FSM στο CONTROL_FSM. Οι πρώτες δύο καταστάσεις αντιστοιχούν στα Instruction Fetch και Instruction Decode. Στο τελευταίο γίνεται η επιλογή των καταστάσεων της FSM ανάλογα με το OpCode της εντολής. Για τον διαχωρισμό του EXSTAGE έχω τις εξής 4 καταστάσεις: για R-type εντολές, για I-type εντολές, για branch εντολές και για load/store εντολές, σύμφωνα με το CHARIS. Όσον αφορά το MEMSTAGE, απομονώνω και ελέγχω μέρος του OpCode για να διαχωρίσω τις load από τις store εντολές. Το WRITE_BACK (stage 9) αφορά τις R-type, I-type και load εντολές. Επιστρέφουμε στο τέλος πάντα στο Stage 1.

Σχόλια:

1. Το σήμα ALU_Bin_sel το αντικατέστησα με το srcB, το οποίο ελέγχει πολυπλέκτη 4x1 και όχι 2x1.
2. Ακόμη, το σήμα PC_Source ελέγχει τον πολυπλέκτη που έχω ορίσει στο EXSTAGE ως εξής: στο Stage 1 (Instruction Fetch) επιλέγεται PC_Source = 0, καθώς όπως προανέφερα, τότε η ALU χρησιμοποιείται για την πράξη PC+4, και το αποτέλεσμα είναι στην είσοδο του καταχωρητή της. Στη συνέχεια προχωράμε στο DECSTAGE και αν το OpCode αντιστοιχεί σε εντολή branch τότε χρησιμοποιούμε πάλι την ALU για να υπολογίσουμε την πράξη PC=(PC+4)+Immed (η πράξη στην παρένθεση έχει προϋπολογιστεί). Στο EXSTAGE η τιμή PC=(PC+4)+Immed έχει περάσει στον καταχωρητή της ALU και αν έχουμε valid branch (**PC_Sel = 1**), τότε έχουμε και **PC_Source = 1** και PC_LdEn = 1. Τότε ο καταχωρητής PC θα πάρει τη σωστή τιμή σύμφωνα με την διακλάδωση.

PROCESSOR_MC

Στον PROCESSOR συνδέω όλα τα submodules και στο testbench συνδέοντα μαζί με τη μνήμη. Ο καταχωρητής για την αποθήκευση του Instruction έχει δηλωθεί στον PROCESSOR. Στο παρακάτω σχήμα φαίνεται η λογική που ακολούθησα.



Φάση 5η

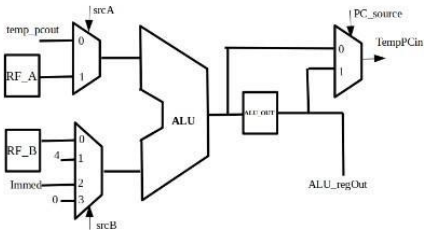
- DATAPATH_PIPELINE

Για το datapath επέλεξα να επεκτείνω εκείνο του multi-cycle επεξεργαστή και να προσθέσω καταχωρητές για τα ενδιάμεσα στάδια και για τα σήματα ελέγχου. Στην πρώτη βαθμίδα καταχωρητών των σημάτων ελέγχου,

αποθηκεύω τα σήματα που χρειάζονται τα EXSTAGE, MEMSTAGE, WRITEBACK STAGE, στη δεύτερη βαθμίδα αποθηκεύω τα σήματα που χρειάζονται τα MEMSTAGE, WRITEBACK STAGE και στην τρίτη βαθμίδα αποθηκεύω τα σήματα που χρειάζεται το WRITEBACK STAGE. Η μνήμη RAM και ο καταχωρητής που ελέγχεται από το σήμα we_ifid_hdu πριν το DECSTAGE βρίσκονται εκτός του Datapath και εντός του Processor αλλά για λόγους σαφήνειας αποτυπώνονται στο παραπάνω σχήμα.

Σχόλια:

1. Αρχικά, έχω απλοποιήσει το IFSTAGE καθώς δεν εξυπηρετούνται branch εντολές και δεν υπολογίζεται η πράξη PC=(PC+4)+Immed. Επομένως, δεν χρειάζεται ο πολυπλέκτης, ο αθροιστής και το PC_Sel.
2. Το EXSTAGE τροποποιήθηκε έτσι ώστε να περιέχει ξανά τον πολυπλέκτη που ελέγχεται από το ALU_Bin_sel. Στην πρώτη είσοδο βάζω το αποτέλεσμα του καταχωρητή της ALU, στην δεύτερη βάζω την έξοδο του πολυπλέκτη AM_out, στην τρίτη βάζω τις δύο εξόδους του RF και την τέταρτη τη γειώνω. Η έξοδος του πολυπλέκτη που ελέγχεται από το σήμα forwardB συνδέεται στον πολυπλέκτη που ελέγχεται από το ALU_Bin_sel, στον οποίο συνδέω και την έξοδο Immed του καταχωρητή ID/EX. Η έξοδος του πολυπλέκτη που ελέγχεται από το σήμα forwardB συνδέεται επίσης στον καταχωρητή EX/MEM για τη σωστή λειτουργία του Forward Unit.



3. Το DECSTAGE τροποποιήθηκε ώστε να βγάζει έξοδο πρώτον την έξοδο του πολυπλέκτη που ελέγχεται από το σήμα RF_B_sel (rd_rt_hdu - θα χρησιμοποιηθεί στο Hazard Detection Unit και ως είσοδος στον καταχωρητή ID/EX) και, δεύτερον, την έξοδο του πολυπλέκτη που ελέγχεται από το σήμα RF_WrData_sel (AM_out, αποτέλεσμα Mem/WriteBack stage, θα χρησιμοποιηθεί ως είσοδος στους πολυπλέκτες του EXSTAGE για να εξυπηρετηθεί το Forward Unit).
4. Στην πρώτη βαθμίδα, αποθηκεύω τις τιμές του Source Register και την έξοδο του πολυπλέκτη rd_rt_hdu που ελέγχεται από το σήμα RF_B_Sel.

- CONTROL_PIPELINE

Έχοντας, πλέον, χωρίσει τα control και αφήνοντας το Control_alu ως έχει, δουλεύω με το Control_if ως εξής. Όρισα τα σήματα ελέγχου των καταχωρητών κάθε βαθμίδας, τα οποία είναι ενεργά πάντα εκτός αν έχω Reset. Επειδή οι καταχωρητές έχουν sensitivity list για τα Clock, Reset και write enable σήματα και όχι για τα δεδομένα εισόδου, κάθε καταχωρητής παίρνει την τιμή του προηγούμενου στη θετική ακμή του ρολογιού. Τα σήματα εξόδου δεν συνδέονται απευθείας με το Datapath αλλά με έναν πολυπλέκτη, ο οποίος ελέγχεται από τα σήματα βαθμίδων Hazard Detection Unit και Instruction Control.

- HAZARD DETECTION UNIT (HDU)

Μέσω αυτού του module επιτυγχάνεται η διαχείρηση των data hazards μέσω stalls. Πρόκειται για συγκριτή που ελέγχει την εξής συνθήκη: memread = 1 και (rs=rd_idex ή rd_rt= rd_idex). Δηλαδή έχει έχει ως είσοδο τα bit του Instr που αντιστοιχούν στον rs, την έξοδο του πολυπλέκτη rd_rt_hdu που ελέγχεται από το σήμα RF_B_sel, την έξοδο του καταχωρητή ID/EX που αφορά τον rd και ένα σήμα MemRead το οποίο είναι 1 μόνο όταν έχουμε load εντολή. Αντίθετα, τα σήματα εξόδου είναι το PC_LdEn, το we_reg_pc και το control_sel, το οποίο το βάζω είσοδο σε μία OR. Όταν έχουμε στο EXSTAGE εντολή load και στο DECSTAGE κάποια εντολή που απαιτεί πληροφορία από τον καταχωρητή που η load χρησιμοποιεί ως rd, τότε παγώνω τα IFSTAGE, DECSTAGE απενεργοποιώντας τα writeEnable του PC και του Instruction καταχωρητη, εισάγωντας μηδενικά σήματα ελέγχου στην πρώτη βαθμίδα σημάτων ελέγχου. Στη συνέχεια ενεργοποιείται το Forward Unit για να εξυπηρετηθεί η εντολή που βρίσκεται στο DECSTAGE.

- FORWARD UNIT

Δέχεται είσοδο τις εξόδους του καταχωρητή ID/EX, rs και rd_rt. Ο rd_rt κρατάει την έξοδο του πολυπλέκτη rd_rt_hdu που ελέγχεται από το σήμα RF_B_sel. Επίσης, έχει ως εισόδους τις εξόδους των καταχωρητών rd των βαθμίδων EX/MEM και MEM/WB και τις εξόδους των καταχωρητών RF_we των βαθμίδων EX/MEM και MEM/WB. Οι αριθμοί 2,3 που είναι στα ονόματα αναφέρονται σε ποιο stage βρισκόμαστε και όποια σήματα έχουν “we” στο όνομά τους, πρόκειται για σήματα enable που χρειάστηκα γι αυτή την υλοποίηση, τα οποία

ενεργοποιούνται ενιαία. Η ιδέα είναι ότι για να βρω την τιμή που παίρνουν τα σήματα forwardA και forwardB, τα οποία ελέγχουν τους δύο πολυπλέκτες του EXSTAGE, πρέπει να κοιτάξω τους καταχωρητές που χρησιμοποιούν οι τρέχουσες και προηγούμενες εντολές. Συγκεκριμένα, αν μία εντολή στο EXSTAGE απαιτεί πληροφορία από καταχωρητή που είναι rd στο MEMSTAGE, τότε προωθώ στην κατάλληλη είσοδο της ALU το περιεχόμενο του καταχωρητή της ALU. Από την άλλη, αν μία εντολή στο EXSTAGE απαιτεί πληροφορία από καταχωρητή που χρησιμοποιείται ως rd στο WRITE BACK STAGE, τότε προωθώ στην κατάλληλη είσοδο της ALU την έξοδο του πολυπλέκτη (AM_out) που ελέγχεται από το σήμα RF_WrData_sel και αποτελεί το αποτέλεσμα του WRITE BACK STAGE.

- INSTRUCTION CONTROL

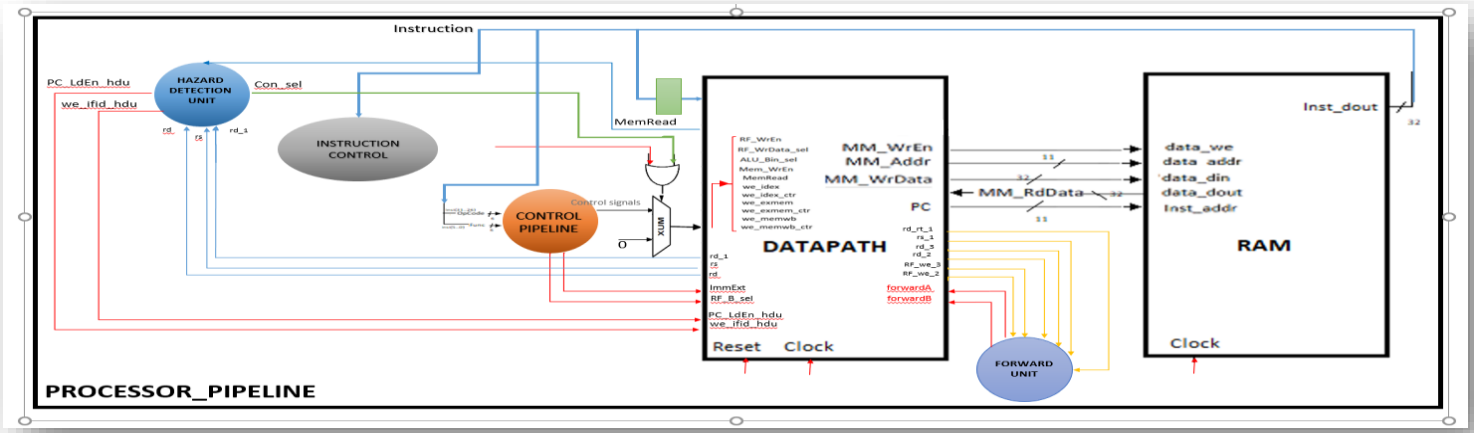
Έχει ως είσοδο το Instruction και ως έξοδο το enable, το οποίο συνδέεται επίσης στην OR που προαναφέρθηκε στο HDU. Αν τα 32 bits της εντολής είναι 0, τότε το ControlEnable ισούται με 1 και η OR βγάζει αποτέλεσμα 1 στο ControlMUX, ανεξαρτήτως τιμής του ελέγχου του HDU.

- CONTROL_MUX

Αποτελείται από πολλούς πολυπλέκτες 2x1 με ένα κοινό σήμα ελέγχου. Όταν είναι 1, τότε στην πρώτη βαθμίδα καταχωρητών ελέγχου όλα τα σήματα έχουν την τιμή 0, ενώ όταν το σήμα ελέγχου είναι 0, τότε περνούν εκείνα τα σήματα του Control που αφορούν την εντολή που κάνουμε decode.

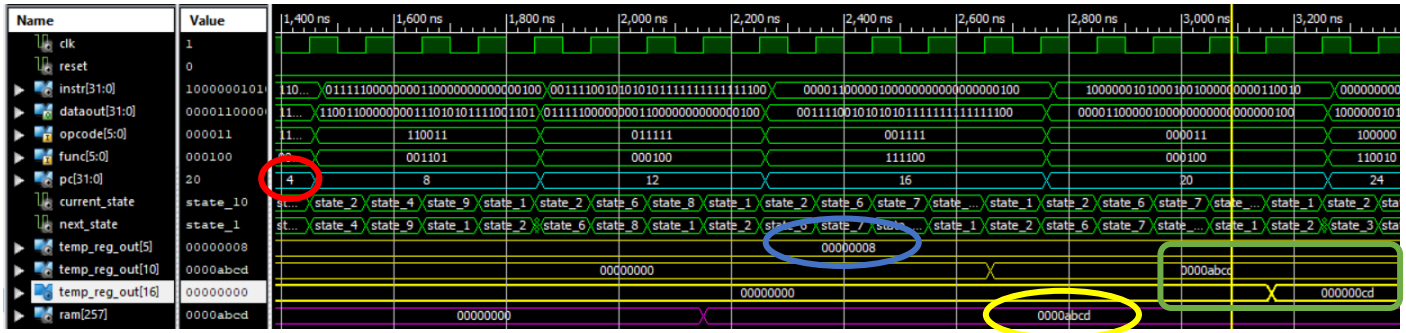
- PROCESSOR_PIPELINE

Συνδέω τα submodules που αναλύθηκαν παραπάνω και στο testbench συνδέω και τη μνήμη. Ο καταχωρητής για το Instruction δηλώνεται και πάλι στον Processor και όχι στο Datapath.



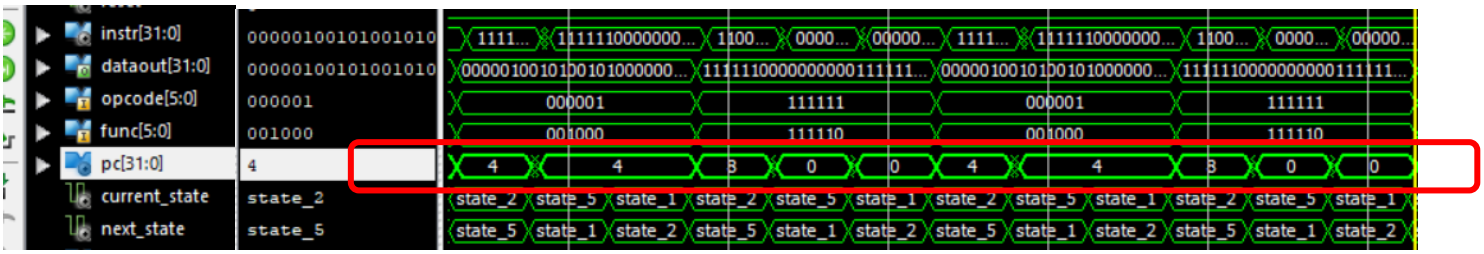
Ανάλυση Αποτελεσμάτων

Πρόγραμμα Αναφοράς 1 – MC



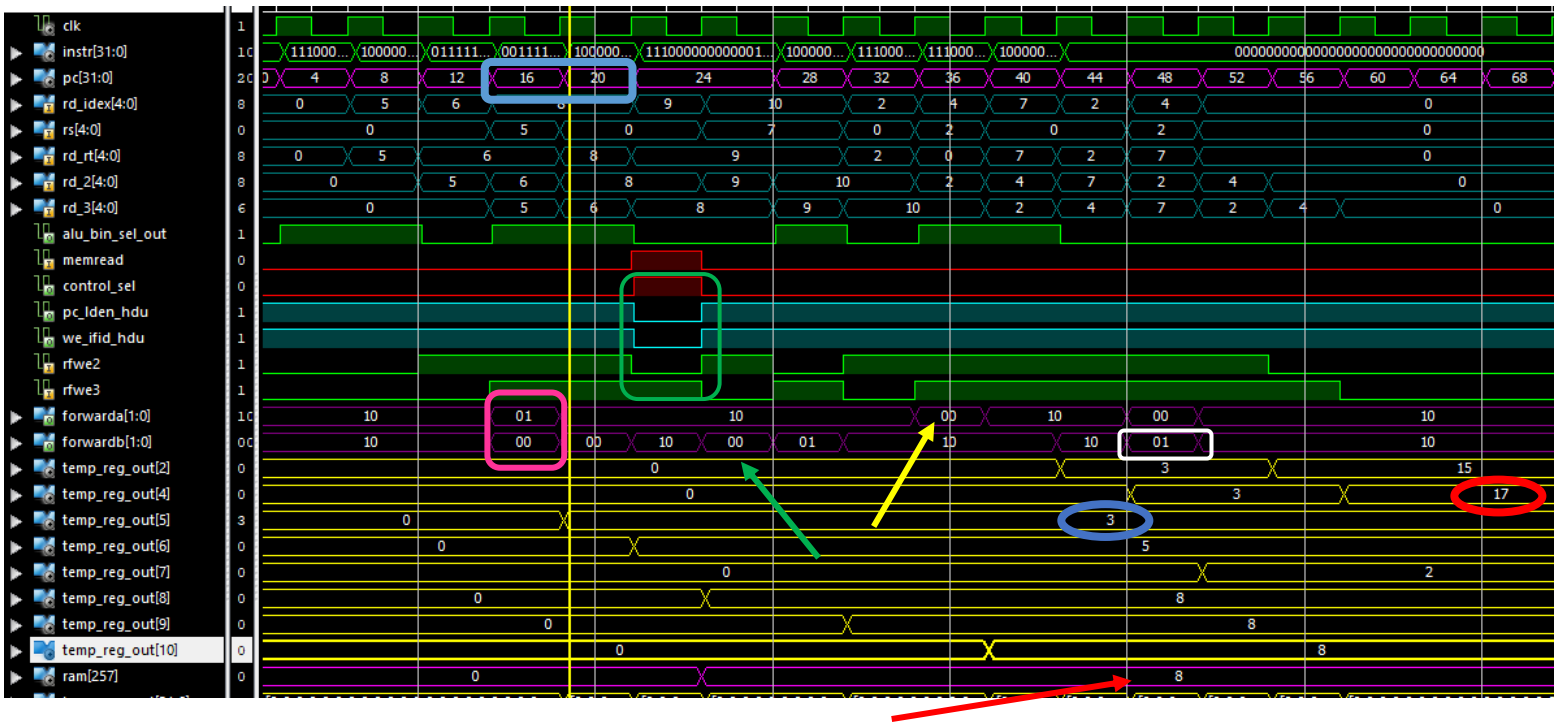
Βλέπουμε ότι στο τέλος του state 1 ενημερώνεται σωστά ο PC και παίρνει την τιμή 4 (κόκκινος κύκλος). Στο τέλος του stage 9 (WRITE BACK STAGE) γίνεται η εγγραφή της τιμής 8 στον r5 (μπλε κύκλος). Επιβεβαιώνεται, επίσης, η σωστή αλληλουχία των states και η λειτουργία των εντολών load και store. Στο Stage 8 (MEMSTAGE) γίνεται sw γράφεται στη θέση 4(r0)=257 της μνήμης η τιμή abcd (κίτρινος κύκλος). Στο stage 10 (τέλος του WRITE BACK STAGE) έχουμε δύο load εντολές, και βλέπουμε ότι γράφονται σωστά οι τιμές abcd και cd στους καταχωρητές r10 και r16 αντίστοιχα (πράσινο πλαίσιο).

Πρόγραμμα Αναφοράς 2 – MC



Σκοπός είναι και πάλι να δείξουμε το ατέρμονο loop το οποίο δημιουργείται, μέσω του καταχωρητή PC. Αυτό σημαίνει πως η εντολή addi r1, r0, 1 δεν ανακτάται και δεν εκτελείται. Αρχικά, ανακτάται η εντολή bne και ο PC παίρνει σωστά την τιμή 4. Επειδή έχουμε εντολές διακλάδωσης, περιμένουμε ότι κατά τη διάρκεια του state 2 (DECSTAGE) χρησιμοποιείται η ALU, προκειμένου να υπολογιστεί η τιμή PC + PC + Immediate, ώστε στο EXSTAGE να ενημερωθεί ο PC, αν η εντολή διακλάδωσης είναι έγκυρη. Για την b εντολή, υπολογίζεται στο τέλος του IFSTAGE το PC+4=8 και στο τέλος του DECSTAGE υπολογίζεται το PC= 8 + (-2) <<2 = 0, το οποίο ακολουθείται μετά το EXSTAGE (state 5), επειδή η εντολή διακλάδωσης είναι έγκυρη. Στον επόμενο κύκλο στο state 1 το Instruction Fetch ανακτά πάλι την bne εντολή και δημιουργείται ατέρμονο loop.

Πρόγραμμα pipeline.rom - PIPELINE



1) li r5, 3: Βλέπουμε ότι γράφεται ο αριθμός 3 στον καταχωρητή r5 στο τέλος του WRITE BACK STAGE, δηλαδή 4 κύκλους μετά το IF, όταν ο PC θα αλλάζει από 16 σε 20 (μπλε κύκλος και μπλε πλαίσιο).
2) li r6, 5: Στον επόμενο κύκλο γράφεται η τιμή 5 στον καταχωρητή r6.

3) add r8, r5, r6: θα ενεργοποιηθεί η FORWARD UNIT καθώς έχουμε εντολή στο EXSTAGE, η οποία απαιτεί forward και από το MEMSTAGE και από το WRITE BACK STAGE. Ο r5, τον οποίο χρησιμοποιεί η add, είναι ο rd της load όταν αυτή είναι στο WRITE BACK STAGE και ο r6 είναι ο rd της load όταν αυτή είναι στο MEMSTAGE. Επομένως, γίνεται και forwardA=01 (αποτέλεσμα καταχωρητή ALU, και forwardB=00 (αποτέλεσμα πολυπλέκτη που ελέγχεται από το σήμα RF_WrData_sel – ροζ πλαίσιο).
4) sw r8, 4(r0): Η εντολή βάζει στη σωστή θέση μνήμης την τιμή 8 (κόκκινο βέλος).
5) lw r9, 4(r0): Φορτώνεται στον καταχωρητή 9 η τιμή 8.
6) add r10, r9, r6: Θα φορτωθεί στον καταχωρητή r10 η πρόσθεση των r6 και r9. Εδώ δημιουργείται stall για την αντιμετώπιση hazard. Το DECSTAGE παγώνει και εκτελείται έναν κύκλο μετά , ώστε η add να εξυπηρετηθεί μέσω forwardB = 00 (πράσινο βέλος). Στο πράσινο πλαίσιο φαίνονται τα σήματα που δημιουργούν καθυστέρηση στο datapath.
7) li r2, 3: Φορτώνω την τιμή 3 στον καταχωρητή r2.
8) add r4, r2, r0: Περνάω την τιμή του r2 στον r4. Επειδή ο r2 είναι ο rs της εντολής, θα γίνει forward της τιμής του καταχωρητή της ALU στο EX/MEM stage στην πάνω είσοδο της ALU, έχοντας forwardA = 00 (κίτρινα βέλος)
9) li r7, 2: Φορτώνω την τιμή 2 στον καταχωρητή r7.
10) li r2, 15: Φορτώνω την τιμή 15 στον καταχωρητή r2. Αυτή την εντολή την έβαλα για να δημιουργήσω το forwarding που ακολουθεί.
11) add r4, r2, r7: Περνάει η πρόσθεση των r2, r7 στον καταχωρητή r4. Θα χρειαστούμε forwardB = 01 (πορτοκαλί βέλος), αφού ο καταχωρητής r7 λειτουργεί ως rd στην εντολή li και ως rt στην εντολή add, και η εντολή li είναι valid, άρα και η τιμή που θα έχει εγγραφεί στο RF μετά το WRITE BACK STAGE της li. Αποτέλεσμα είναι η τιμή 17 [που φαίνεται στον κόκκινο κύκλο

<u>Συμπεράσματα – Παρατηρήσεις</u>

Όσον αφορά τον multi cycle επεξεργαστή, παρατηρώ ότι επιβεβαιώνεται η θεωρία και υπάρχουν εντολές που δεν περνούν από όλα τα stages του Datapath, δηλαδή από το critical path, και δεν χρειάζονται το MEMSTAGE ή το WRITE BACK STAGE. Επίσης, κάθε φορά στο datapath θα βρίσκεται μία εντολή, αλλά μπορεί να διαρκέσει λιγότερο από ό,τι στο single cycle επεξεργαστή, πχ μία load εντολή δεν χρειάζεται το WRITE BACK STAGE, άρα θα χρειαστεί λιγότερους κύκλους. Το κύκλωμα CONTROL υλοποιείται με FSM. Έχουμε οικονομία σε hardware, γιατί μπορούμε να χρησιμοποιήσουμε το ίδιο functional unit πολλές φορές στην ίδια εντολή, σε διαφορετικούς κύκλους.

Όσον αφορά τον pipeline επεξεργαστή, συναντάμε data, control και structural hazards τα οποία εμποδίζουν την ομαλή λειτουργία του κυκλώματος. Επίσης, δεν χρειάζεται να τελειώσει κάποια εντολή την εκτέλεσή της για να ξεκινήσει η επόμενη, σε αντίθεση με τον multicycle. Το κύκλωμα control είναι συνδυαστικό και δεν υλοποιείται με FSM, όπως και η HDU και η FU. Στην HDU με τη χρήση stall, εμποδίζουμε το PC και τον IF/ID να αλλάξουν, άρα διαβάζεται σε δύο διαδοχικούς κύκλους η ίδια εντολή και αποκωδικοποιείται η ίδια επόμενή της δύο φορές συνεχόμενα. Επίσης για τα data hazards, οι εντολές ανταλλάσσουν μεταξύ τους δεδομένα μέσω του Register File και της μνήμης. Όταν η επόμενη εντολή χρειάζεται για όρισμα (ανάγνωση) κάτι που δεν έχει προλάβει να γράψει η προηγούμενη, τότε έχουμε κίνδυνο για hazard. Εκτός από το stall, η δεύτερη λύση είναι το forwarding μέσω της FORWARD UNIT, κατά το οποίο προωθούμε το EX/MEM αποτέλεσμα ως είσοδο για την ALU πράξη της επόμενης εντολής και το ίδιο κάνουμε για τη μεθεπόμενη εντολή, όπου προωθούμε το MEM/WB αποτέλεσμα ως είσοδο για την ALU πράξη της μεθεπόμενης εντολής.