1η Εργαστηριακή Αναφορά – ΗΡΥ302

Σοφία Καφρίτσα Γεωργαντά, 2016030136

Σκοπός εργαστηριακής άσκησης

Στόχος της άσκησης είναι η σχεδίαση ενός single cycle MIPS-like επεξεργαστή. Υλοποιώντας ιεραρχικά τα βασικά modules του επεξεργαστή και ορίζοντας την αρχιτεκτονική συνόλου εντολών του, σχεδιάσαμε τις βασικές βαθμίδες του datapath ενός απλού επεξεργαστή.

Περιγραφή

Σχόλια για όλες τις φάσεις: Χρησιμοποίησα τις παρακάτω βιβλιοθήκες ανά περίπτωση: IEEE.NUMERIC_STD.ALL, IEEE.STD_LOGIC_1164.ALL, IEEE.STD_LOGIC_UNSIGNED.ALL, EEE.NUMERIC_STD.ALL, work.mux_pack.all (δική μου, αναφέρεται μετά)
Για την ομαλή λειτουργία των κυκλωμάτων, πολλές φορές χρησιμοποιώ ενδιάμεσα σήματα temp. Επίσης, έχω εφαρμόσει όπου χρειάζεται πρόσθετες καθυστερήσεις με τη χρήση του after. Τα σχήματα έγιναν στα λογισμικά Dia και Word και για το project χρησιμοποίησα το λογισμικό Xilinx ISE.

1η Φάση

Στην 1η φάση, ασχοληθήκαμε με τη μονάδα αριθμητικών και λογικών πράξεων ALU, ένα module που, με βάση την εκφώνηση, ανάλογα με κάθε κωδικό Op εκτελεί αριθμητικές και λογικές πράξεις, και με το αρχείο των καταχωρητών RF. Πιο συγκεκριμένα, τονίζουμε τα εξής:

ALU:

- Συνδυαστικό κύκλωμα χωρίς ρολόι, το οποίο δουλεύει με συμπληρώματα ως προς 2
- Χρησιμοποιώντας case-when, προσομοιώνω τη λειτουργία του Operation που θα εκτελεστεί
- Όσον αφορά το σήμα εξόδου **Zero**: είναι ενεργοποιημένο όταν το αποτέλεσμα, δηλαδή το σήμα Output, είναι μηδέν.
- Όσον αφορά το σήμα εξόδου **Ovf**: εξετάζοντας τη συνθήκη της υπερχείλισης, το σήμα **Ovf** είναι ενεργό όταν τα MSB των σημάτων A και B είναι '0' και το Out είναι '1' ή αντίστροφα τα MSB των σημάτων A και B είναι '1' και το Out είναι '0'.
- Όσον αφορά το σήμα εξόδου **Cout**: όρισα ένα ενδιάμεσο σήμα temp_Cout το οποίο ισούται με ('0'& A) + ('0' & B), ώστε να εξάγω το temp_Cout(32), το οποίο είναι το carry out.

RF:

- Αποτελείται από 3 sub-modules, 2 πολυπλέκτες (32 σε 1), 32 καταχωρητές και 1 αποκωδικοποιητή (5 σε 32), τα οποία έγιναν component/port map στο top-module RF. Τους 32 καταχωρητές τους δημιούργησα με for-generate, το οποίο το χρησιμοποίησα επίσης για να δημιουργήσω πύλες AND, στις οποίες συνέδεσα το WrEn του RF και την έξοδο του αποκωδικοποιητή.

  Πιο συγκεκριμένα για τα instances έχουμε:
- REG: καταχωρητής με 32 bits που αλλάζει την έξοδό του σε κάθε θετική ακμή του ρολογιού.
- MUX: πολυπλέκτης με 32 εισόδους των 32 bits. Επέλεξα να σχεδιάσω έναν Generic Mux δημιουργώντας νέο package, το οποίο ένα use και δημιούργησα με port map τα 2 instances των πολυπλεκτών.

- DEC: αποκωδικοποιητής στην αρχιτεκτονική του οποίου χρησιμοποίησα επίσης for-generate, ώστε να είναι μόνο ένα σήμα ενεργό κάθε φορά.

2η Φάση

RAM: Χρησιμοποιήθηκε ο δεδομένος κώδικας στις βαθμίδες IF και MEM, αφού πρώτα δημιούργησα testbench για να επιβεβαιώσω τη λειτουργία της.

IFSTAGE: ενημέρωση του PC ανάλογα με την κάθε εντολή.

- Αποτελείται από: τον *καταχωρητή PC*, τον αθροιστή *adder_incre*, ο οποίος αυξάνει κατά 4 τον καταχωρητή , τον αθροιστή *adder_immed*, ο οποίος είναι υπεύθυνος για τις εντολές διακλάδωσης και έναν *πολυπλέκτη MUX2x1,* ο οποίος αποφασίζει πώς ενημερώνεται ο PC, Αν PC_sel = 0 τότε έχουμε εντολή διακλάδωσης και πρέπει να συμπεριλάβουμε και την τιμή του Immediate, αλλιώς αν PC_sel = 1 αυξάνουμε απλά τον PC κατά 4.
- Σε single cycle επεξεργαστή, η τιμή του PC_LdEn ισούται πάντα με 1.
- Έχω, επίσης, δημιουργήσει IFSTAGE_topmodule (και αντίστοιχο testbench), στο οποίο ενώνω το IFSTAGE με τη μνήμη

DECSTAGE: βαθμίδα αποκωδικοποίησης εντολών.

- Αποτελείται από: το *RF*, έναν *πολυπλέκτη MUX2x1*, έναν μικρότερο *πολυπλέκτη 5 σε 1* και τον *CONVERTER (συννεφάκι)*
- Τα διάφορα τμήματα της εντολής Instr που θα αποκωδικοποιηθεί, στέλνονται ως είσοδο στα components RF, CONVERTER και MUX2x1.
- Ο CONVERTER είναι υπεύθυνος να ξεχωρίσει αν χρειάζεται sign-extend (ImmExt=01), ή zero-fill (ImmExt=00), και αν χρειάζεται ολίσθηση ή όχι (ImmExt=10 και ImmExt=11). Η έξοδός του είναι το Immed, το οποίο συνδέεται με το Immed του DECSTAGE.
- Το σήμα RF_B_sel είναι ενεργό όταν έχουμε εντολές Immediate, ώστε η έξοδος του RF_B του RF να είναι ο rd της εντολής.

EXSTAGE: βαθμίδα εκτέλεσης εντολών.

- Αποτελείται από: την *ALU* και ένα *πολυπλέκτη MUX2x1.*
- Ο sel του πολυπλέκτη ενώνεται με το ALU_Bin_sel και επιλέγεται η είσοδος B της ALU από RF_B ή Immediate.
- Το σήμα ALU_zero συνδέεται με το σήμα zero της ALU (στην επόμενη φάση χρησιμοποιείται ως είσοδος στο CONTROL)
- Το σήμα ALU_func ελέγχει τι πράξη θα κάνει η ALU (σχετίζεται με το σήμα func της εντολής που θα δούμε στην επόμενη φάση)

MEMSTAGE: εγγραφή και ανάγνωση μνήμης.

- Φροντίζω ώστε η διεύθυνση μνήμης που θα διαβάζεται θα είναι η διεύθυνση που στέλνει η ALU, προστιθέμενη κατά 0x400, ώστε τα δεδομένα να διαβάζονται και να γράφονται στο σωστό τμήμα.
- Χρησιμοποίησα το σήμα ByteOp, για να διαχωρίσω τη λογική των sw/lw από τη λογική των sb/lb, όπως φαίνεται στην εικόνα.
- Έχω, επίσης, δημιουργήσει MEMSTAGE_topmodule (και αντίστοιχο testbench), στο οποίο ενώνω το MEMSTAGE με τη μνήμη.

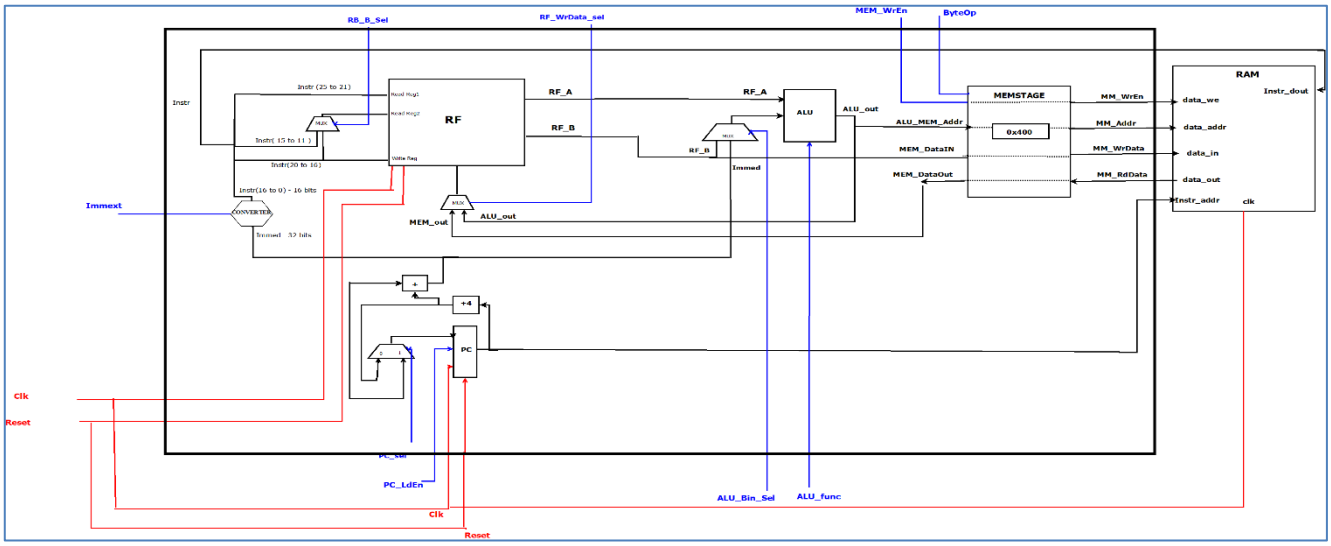
```
if ByteOp = '0' then
   MM_WrData   <= MEM_DataIn; --sw
   MEM_DataOut <= MM_RdData;  --lw
else
   temp_sb   <= MEM_DataIn(7 downto 0); --sb
   MM_WrData <= ("00000000000000000000000" & temp_sb);

   temp_lb     <= MM_RdData(7 downto 0); --lb
   MEM_DataOut <= ("00000000000000000000000" & temp_lb);
end if;
```

3η Φάση

DATAPATH

- Συνδέοντας όλα τα modules της 2ης φάσης, δημιούργησα το datapath που φαίνεται παρακάτω
- Η RAM μπήκε ως component στο testbench για την επαλήθευση της σωστής λειτουργίας.



CONTROL

- Το Control ουσιαστικά είναι η μονάδα ελέγχου που περιέχει τη ροή των πληροφοριών. Στο entity του Control έχω ορίσει, εκτός των άλλων, δύο σήματα OpCode και func, ώστε η συσχέτιση με τα μέρη των εντολών να είναι πιο κατανοητή. Με βάση τον πίνακα από την εκφώνηση για τις τιμές των OpCode και func, για κάθε εντολή ορίζω τα σωστά σήματα PC_Sel, PC_LdEn, RF_B_Sel, RF_WrEn, RF_WrData_Sel, ALU_Bin_Sel, ALU_func, Mem_WrEn και ImmExt.
- Παρατήρησα ότι το PC_sel θέλουμε να είναι '1' μόνο όταν έχουμε valid branch εντολές. Επίσης, για τις branch εντολές χρησιμοποιώ την πράξη της αφαίρεσης. Ακόμη, παρατήρησα ότι το RF_B_sel είναι '1' στις εντολές με immediate και το RF_WrEn είναι '0' μόνο στις branch και στις sw/sb. Το RF_WrData_ sel είναι ΄1΄ μόνο στις lw/lb εντολές. Το ALU_Bin_sel είναι ΄0΄ στις branch εντολές και στις εντολές με OpCode=100000. Το MEM_ WrEn είναι '1' μόνο στις εντολές sw/sb.
- Όσον αφορά τις τιμές του ALU_func που όρισα, αναλύω παρακάτω το σκεπτικό μου:
  Αρχικά, για τις εντολές li, lui έχω θεωρήσει ότι κάνουμε πρόσθεση το immediate με τον r0, ώστε να προκύψει το επιθυμητό ALU_out.
  1. ALU_func = 0000, καθώς παρατήρησα ότι οι εντολές li, lui, addi, lb, lw, sb, sw χρησιμοποιούν την ALU ως αθροιστή
  2. ALU_func = 0001, καθώς Παρατήρησα ότι οι εντολές b, beq, bne χρησιμοποιούν την ALU ως αφαιρέτη
  3. ALU_func = 0101 καθώς, η εντολή nandi χρησιμοποιεί την ALU για να κάνει την πράξη nand
  4. ALU_func = 0011, καθώς η εντολή ori χρησιμοποιεί την ALU για να κάνει την πράξη or
  5. ALU_func = υπόλοιπες τιμές, εξήγαγα τα 4 LSB της func για τις εντολές με OpCode="100000".

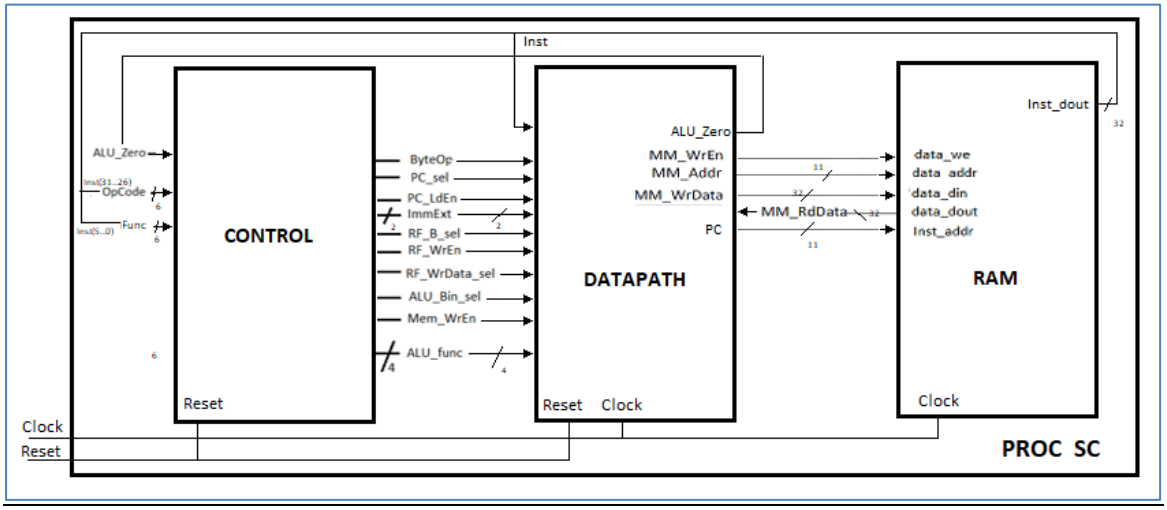
- Συγκεκριμένα, για τις branch εντολές, έβαλα μία ένθετη if στην οποία ανάλογα με το zero θα ορίζεται και το κατάλληλο PC_Sel. Για τις b, beq το PC_Sel είναι ενεργό με zero = '1', όπως βλέπουμε δίπλα.
  Για την bne ισχύει το αντίθετο, δηλαδή το
  το PC_Sel είναι ενεργό με zero = '0', όπως βλέπουμε δίπλα.

```
if(zero = '1') then
   PC_Sel   <= '1';
else
   PC_Sel   <= '0';
end if;
```

```
if(zero = '0') then
   PC_Sel   <= '1';
else
   PC_Sel   <= '0';
end if;
```

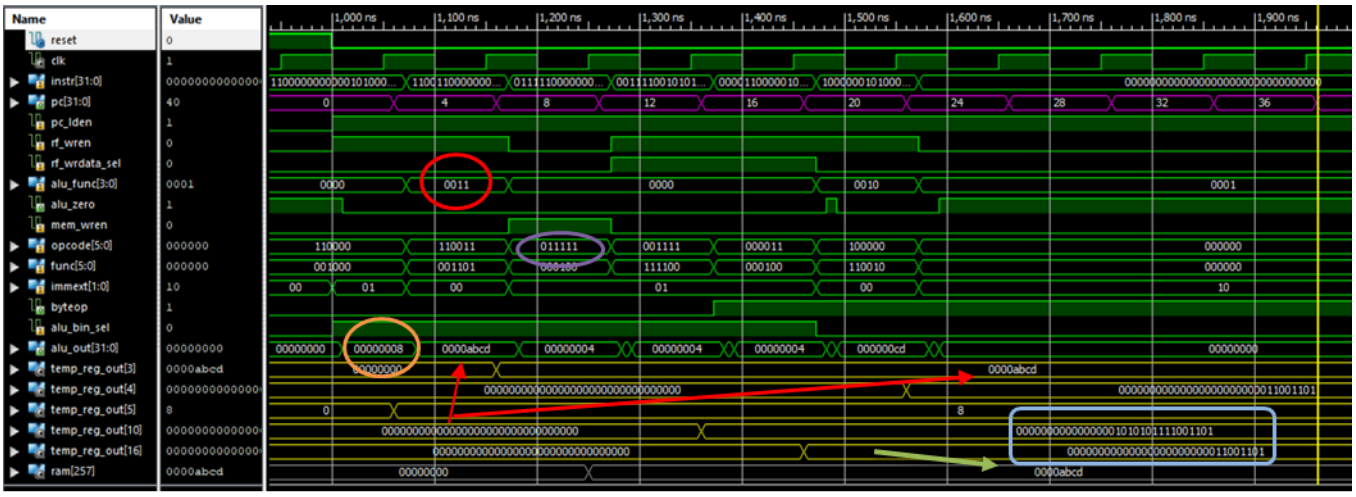
PROC_SC

Συνδέοντας κατάλληλα Control, Datapath και Ram, δημιούργησα το topLevel του επεξεργαστή ενός κύκλου.



Ανάλυση

Πρόγραμμα Αναφοράς #1

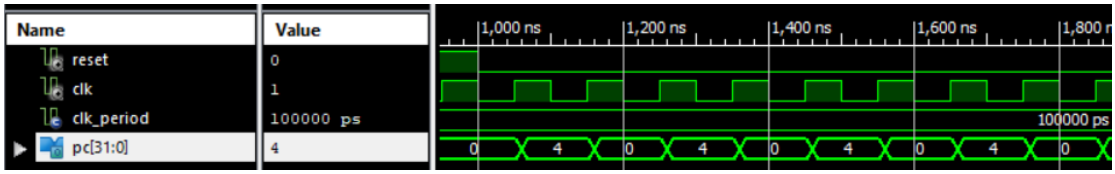


Πριν από τα 1000ns έχω το Reset ενεργό, επομένως η ανάλυση ξεκινάει με την εντολή addi μετά τα 1000ns.

addi r5, r0, 8: Έχουμε εντολή Immediate και περιμένουμε να δούμε ALU_Bin_ sel=1 και ALU_out= r0+8=8 (πορτοκαλί κύκλος). Βλέπουμε ότι το αποτέλεσμα γράφεται επιτυχώς στον r5.

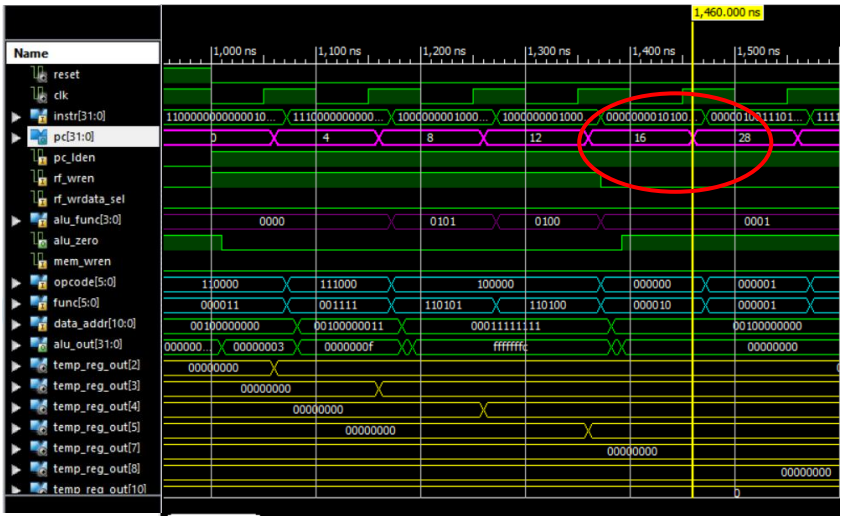
ori r3, r0, 0xABCD: Στα 1160 ns ο r3 παίρνει την τιμή ABCD και το ALU_func έχει την τιμή 0011 που αντιστοιχεί στην or (κόκκινα βέλη για ABCD και κύκλος).
sw r3, 4(r0): Στα 1200ns από την τιμή του Opcode βλέπουμε ότι κάνουμε sw (μωβ κύκλος), Βλέπουμε ότι και τα υπόλοιπα σήματα μπαίνουν σωστά, δηλαδή ByteOp=0, MEM_WrEn=1, RF_WrEn=0 και ImmExt=01. Το ALU_out παίρνει την σωστή τιμή, το 4 και εφόσον έχουμε προσθέσει το offset 0x400, η διεύθυνση της μνήμης που θα γράψουμε είναι $x404_{16} = 257_{10}$. (πράσινο βέλος που δείχνει στην τιμή της RAM)
lw r10, -4(r5): Γράφουμε στον r10 στα 1350ns.
lb r16, 4(r0): Γράφουμε στον r16. Παρατηρούμε ότι η τιμή του r16 ταυτίζεται με τα 8 LSB του r10, αφού κάνουμε load από την ίδια διεύθυνση μνήμης(μπλε παραλληλόγραμμο).
nand r4, r10, r16: Δεν έχουμε immediate, άρα ALU_Bin_sel=0 και ο r4 ενημερώνεται σωστά.

Πρόγραμμα αναφοράς #2

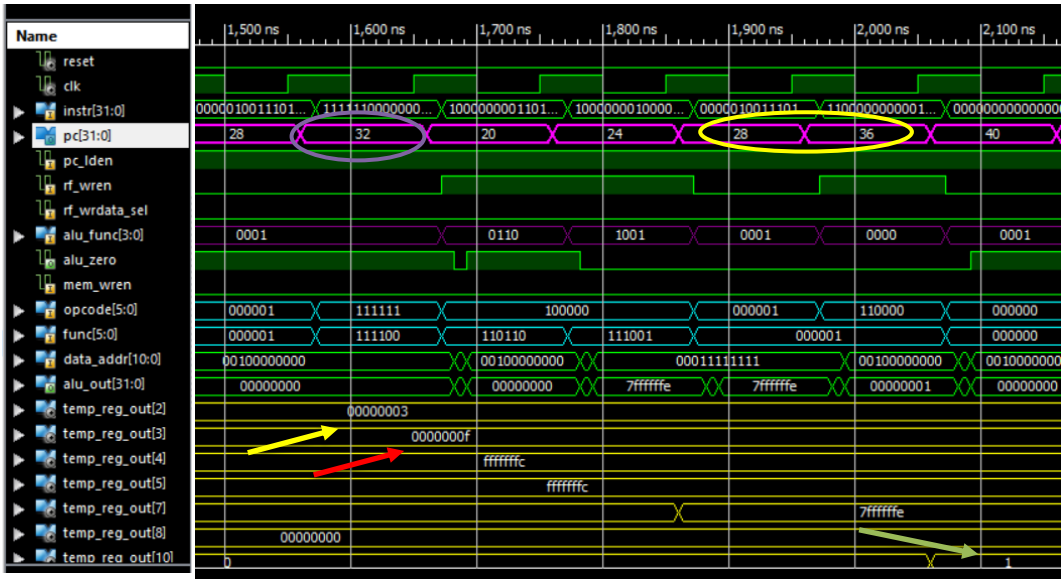


Σε αυτό το πρόγραμμα θέλουμε να επιβεβαιώσουμε ότι το infinity loop δουλεύει όταν έχουμε αποτυχημένη διακλάδωση. Όπως φαίνεται και στην παρακάτω εικόνα, η τιμή του PC εναλλάσσεται από 0 σε 4 και επαληθεύεται, έτσι, το infinity loop.

Πρόγραμμα αναφοράς #4 (για rom4.data)



Πριν από τα 1000ns έχω το Reset ενεργό, επομένως η ανάλυση ξεκινάει μετά τα 1000ns.
addi r2,r0,3: Εκτελείται η addi και ο r2 παίρνει την τιμή 3 (η τιμή 3 φαίνεται στο επόμενο σχήμα με κίτρινο βέλος).
li r3, 15: Εκτελείται στα 1100 ns. Το ALU_func έχει ορθά την τιμή 0000 και ο r3 την τιμή 15. (η τιμή 3 φαίνεται στο επόμενο σχήμα με κόκκινο βέλος).
nand r4, r2, r3: Εκτελείται στα 1200 ns. Το RF_B_sel είναι '0' καθώς έχουμε εντολή χωρίς immediate. Η καθυστέρηση που βλέπουμε στην ενημέρωση του r4 είναι αναμενόμενη, καθώς υπάρχουν καθυστερήσεις απόκρισης της ALU και του RF.
Μέχρι στιγμής ο PC αυξάνεται κατά 4.
not r5, r2: Ο r5 ισούται με τον r4, καθώς not(0011) = 1100
beq r4, r5, 2: Λόγω του παραπάνω, το beq στα 1400 ns είναι valid και επειδή r5-r4 τότε ALU_zero=1 και συνεπώς PC_Sel=1. Επομένως, ο PC αυξάνεται από τα 16 στα 28 σύμφωνα με την τιμή του immediate. (κόκκινος κύκλος)



nor r8, r3, r4: -- Δεν μπαίνει την πρώτη φορά.
srl r7, r4: -- Δεν μπαίνει την πρώτη φορά.
bne r8,r7,1: Εκτελείται invalid bne (με PC=28) αφού r7=r8 και το PC γίνεται 32. (μωβ κύκλος)
b -4: Ο PC γίνεται 20. Μετά από δύο κύκλους (για να γίνει ο PC=28) θα εκτελεστεί πάλι η bne η οποία τώρα είναι valid και ο καταχωρητής PC παίρνει την τιμή 36. (κίτρινος κύκλος)
addi r10, r0, 1: Εκχωρείται η τιμή 1 στον καταχωρητή r10. (πράσινο βέλος)

Συμπεράσματα – Παρατηρήσεις

- Επειδή ο επεξεργαστής είναι single cycle, η κάθε εντολή θα διαρκεί όσο και η περίοδος του ρολογιού, ακόμα κι αν έχει ολοκληρωθεί σε μικρότερο χρόνο.
- Λόγω των καθυστερήσεων που υπάρχουν στα testbenches, θέλουμε η περίοδος του ρολογιού να είναι τουλάχιστον 100 ns.
- Όταν κάποια μέρη του κυκλώματος χρειάζονται λιγότερο χρόνο από 100 ns για να ολοκληρώσουν τη λειτουργία τους, μπορεί να εμφανιστούν σκουπίδια στο κύκλωμα, τα οποία, όμως, αγνοούνται εφόσον ορίζονται σωστά τα σήματα ελέγχου.
- Δημιούργησα δύο επιπλέον ακόμα αρχεία, το rom3.data και rom4.data
- Τα testbench των IFSTAGE, MEMSTAGE αναφέρονται στα topModules.