ΑΝΑΦΟΡΑ tomasulo 2ο μέρος

Κουρκουλος Αγγελος ΑΜ:2017030111

Σκοπός αυτού του μέρους είναι ο έλεγχος του συνολικού κυκλώματος tomasulo για κάθε διαφορετική είδους εισόδου που θα μπορούσε να υπάρξει , επαληθεύοντας έτσι την ορθότητα του κυκλώματος ακόμα και για τις ακραίες περιπτώσεις .

Στην αρχή της εκτέλεσης όλων των simulation ξεκινάμε κάνοντας rst=1 για ένα κύκλο .

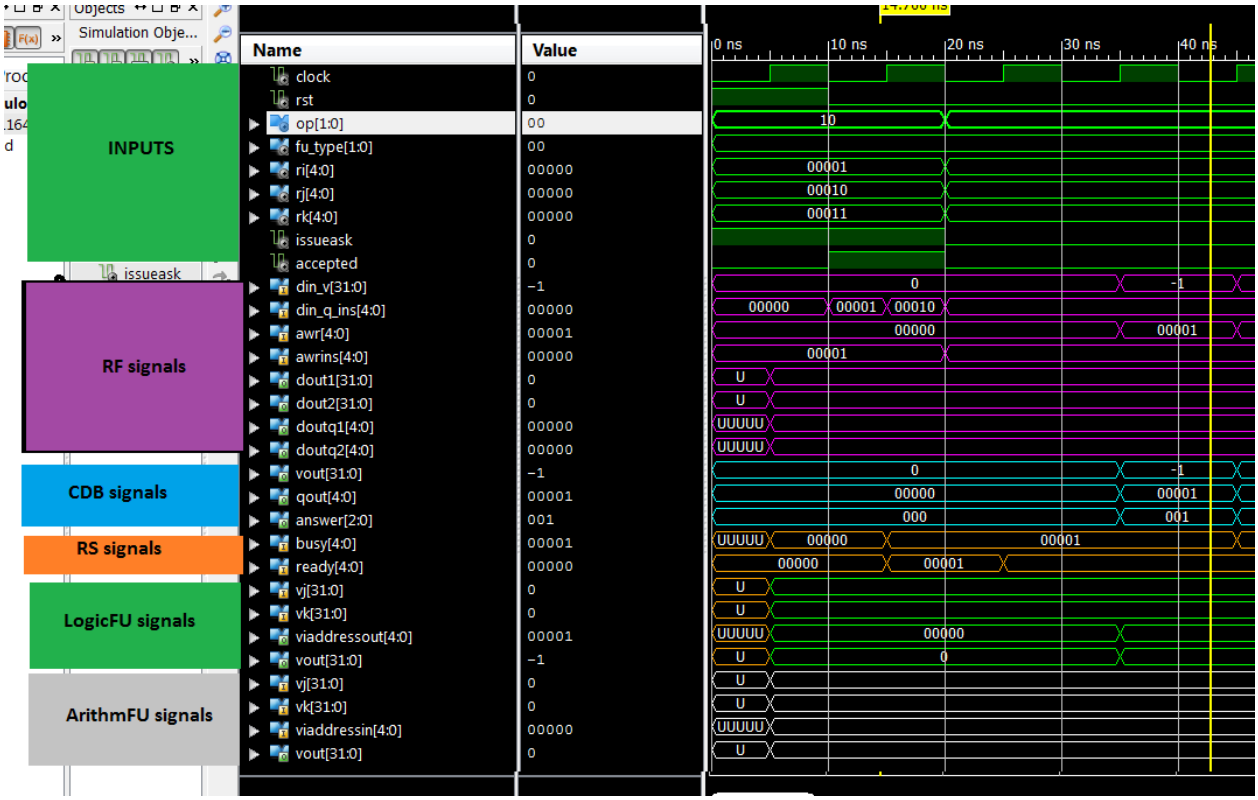
1.

Αρχικά ελέγχουμε αν το κύκλωμα εκτελείτε σωστά όταν του ζητάμε να κάνει μια απλή εντολή NOT χωρίς να ακολουθεί κάποια άλλη εντολή :

Inps:

```
 wait for Clock_period*1;
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00001";
Rj <="00010";
Rk <="00011";
Issueask<='1';
```

Simulation:



Στην αρχή του δεύτερου κύκλου γίνεται issue η εντολή και τη στιγμή του παλμού του ρολογιού τα κατάλληλα δεδομένα αποθηκεύονται στο πρώτο RS το οποίο επιβεβαιώνεται

από το σήμα busy που γίνεται 00001 καθώς ενεργοποιείται το αντίστοιχου σήμα ready του RS δύοντας τη δυνατότητα στην FU να ξεκινήσει την πράξη . Ταυτόχρονα το σήμα awrIns του RF έχει την τιμή της διεύθυνσης του καταχωρειτή προορισμού που είναι 00001 και το din_q_ins έχει το tag του RS που εκτελεί την εντολή που θα καταλήξει στο αντίστοιχο register του πεδίου V του RF. 2 κύκλους μετά βλέπουμε ότι το αποτέλεσμα (-1) της πράξεις υπάρχει στην έξοδο της LogicFU και δημοσιεύεται στο πεδίο V του CDB καταλήγοντας στην είσοδο din_v του RF όπως γίνεται και με την τιμή Q αντίστοιχα ώστε να γραφεί ο κατάλληλος register. Η εντολή έχει φτάσει στο τέλος της και το RS απελευθερώνεται κάνοντας το κατάλληλο σήμα busy =0.

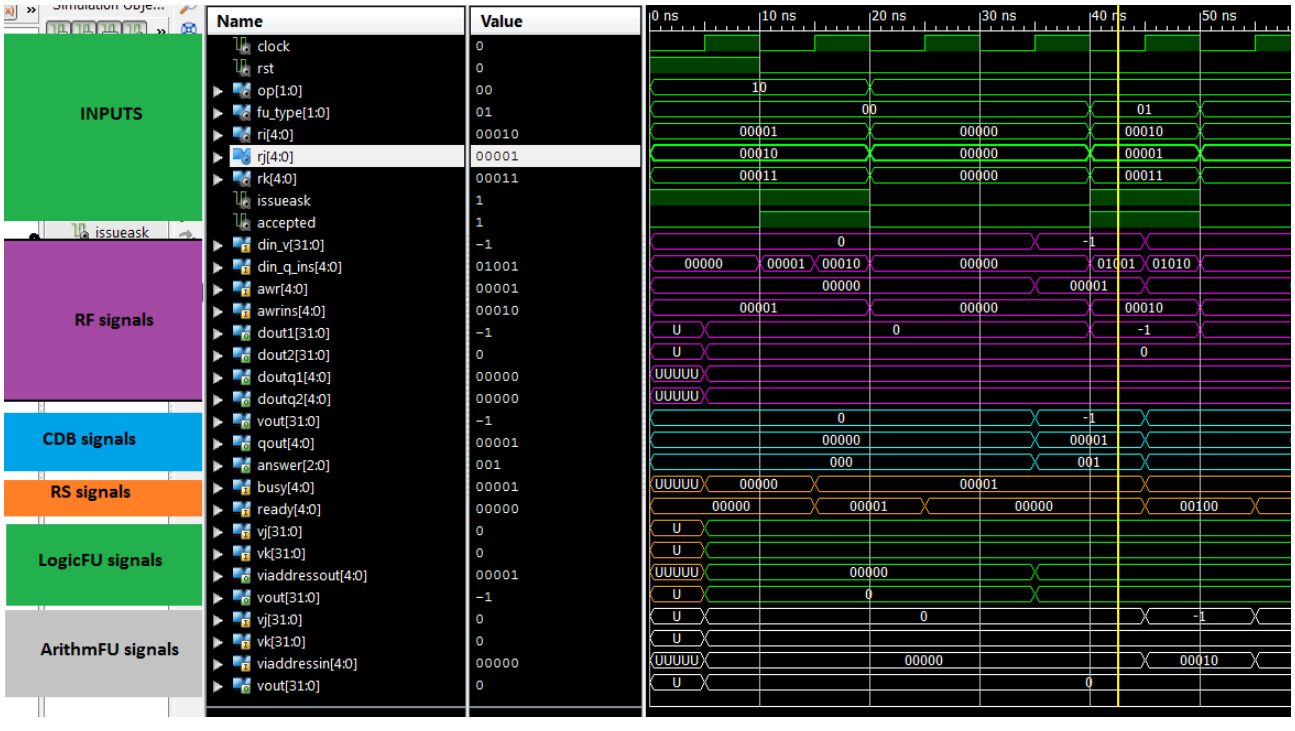
2.

Σε αυτήν την προσομοίωση αφού έρθει ο κύκλος στον οποίο θα γραφεί το αποτέλεσμα της πράξης που έγινε προηγούμενος, χρησιμοποιούμε το αποτέλεσμα για να κάνουμε μια αριθμητική πράξη πρόσθεσης τσεκάροντας έτσι αν γίνεται σωστά η μέθοδος fall-through που έχει υλοποιηθεί στο RF.

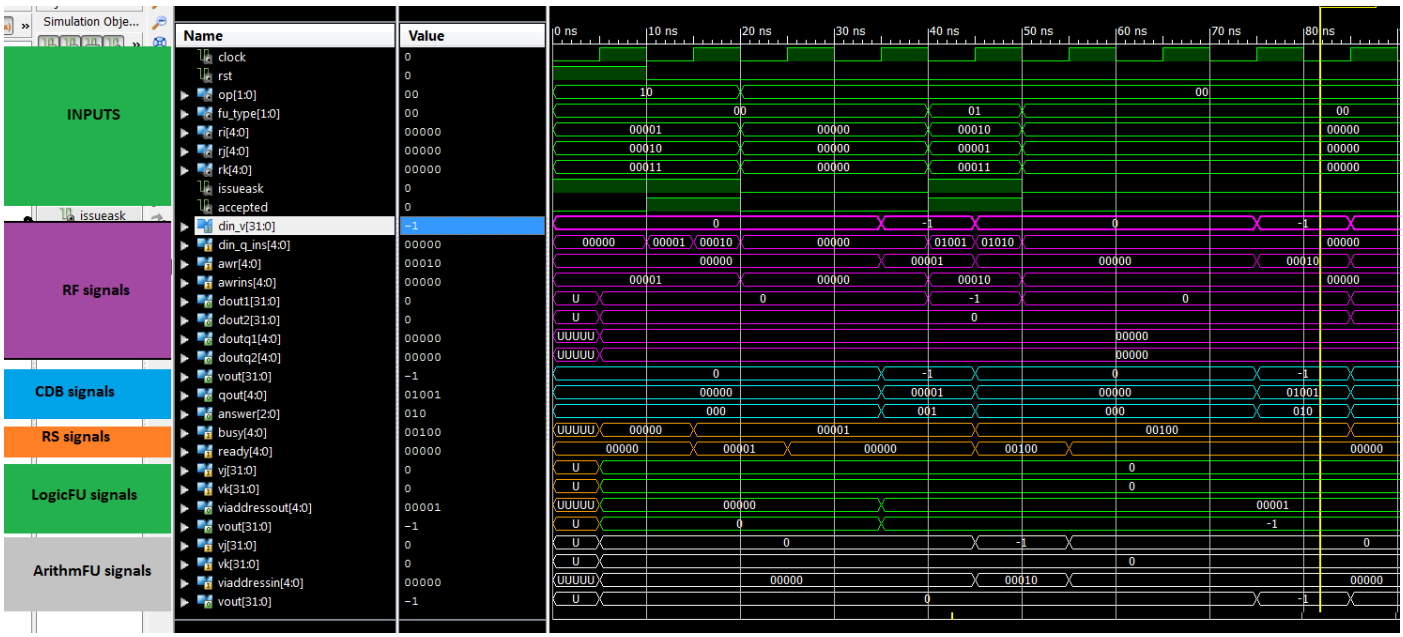
Inps:

```
  wait for Clock_period*1;
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00001";
Rj <="00010";
Rk <="00011";
Issueask<='1';
  wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="00";
Ri <="00000";
Rj <="00000";
Rk <="00000";
Issueask<='0';
  wait for Clock_period*2;
 Rst <='0';
op <="00";
FU_type <="01";
Ri <="00010";
Rj <="00001";
Rk <="00011";
Issueask<='1';
```

Simulation:



Μετά την ολοκλήρωση της πράξης NOT έρχονται τα σήματα για την εκτέλεσης της εντολής της πράξης + . Προτού έρθει ο παλμός του ρολογιού βλέπουμε ότι τα σήματα awrIns γίνεται 00010 που είναι η διεύθυνση που θα γραφεί το αποτέλεσμα της πράξης + καθώς το din_q_ins γίνεται 01001 που είναι το tag του RS που θα εκτελέσει την πράξη . Ταυτόχρονα η έξοδος dout1 του RF γίνεται -1 ίση δηλαδή με το αποτέλεσμα της πράξης NOT που προηγήθηκε επαληθεύοντας την σωστή λειτουργία του fall-through . Μόλις έρθει ο παλμός του ρολογιού οι έξοδοι του RF και όλα υπόλοιπα χρήσιμα σήματα γράφονται στο RS , γίνεται τα κατάλληλα busy και ready 1 και ξεκινά η εκτέλεση της πράξης από την ArithmFU .



3 κύκλους αργότερα βλέπουμε και έχει τελειώσει η πράξη της ArithmFU καθώς έχει βγει το αποτέλεσμα και έχει δημοσιευθεί στο CDB καταλήγοντας στο RF όπου και γράφεται. Η εντολή έχει φτάσει στο τέλος της και το RS απελευθερώνεται κάνοντας το κατάλληλο σήμα busy =0.

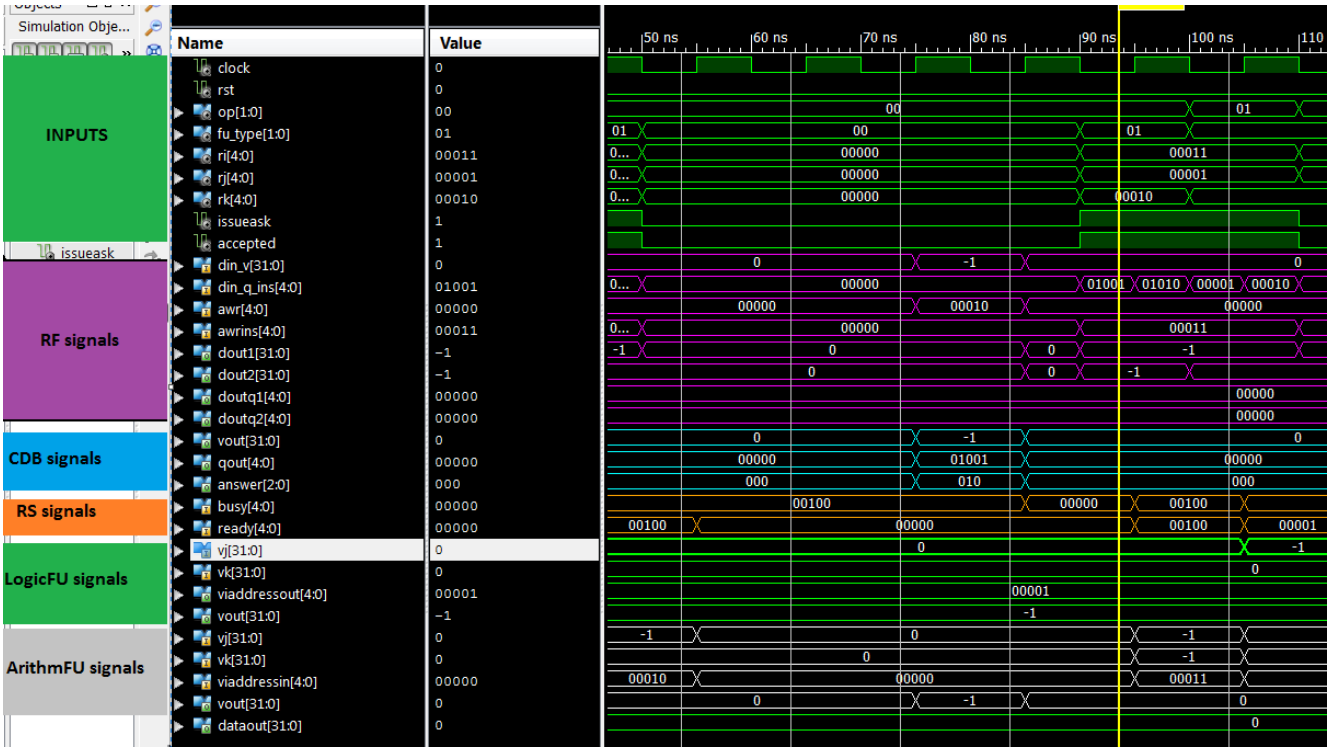
3.

Μετά την ολοκλήρωση και εγγραφή των 2 παραπάνω πράξεων χρησιμοποιώντας τα αποτελέσματα τους θα εξετάσουμε την περίπτωση που 2 πράξεις γίνονται παράλληλα και χρησιμοποιούν διαφορετικές λειτουργικές μονάδες . Για την προσομοίωση αυτής της πιθανότητας θα εκτελέσουμε μια πράξη + μεταξύ των αποτελεσμάτων των αρχικών πράξεων και έπειτα μια πράξη AND με το αποτέλεσμα της πράξης NOT και του 0.

Inps:

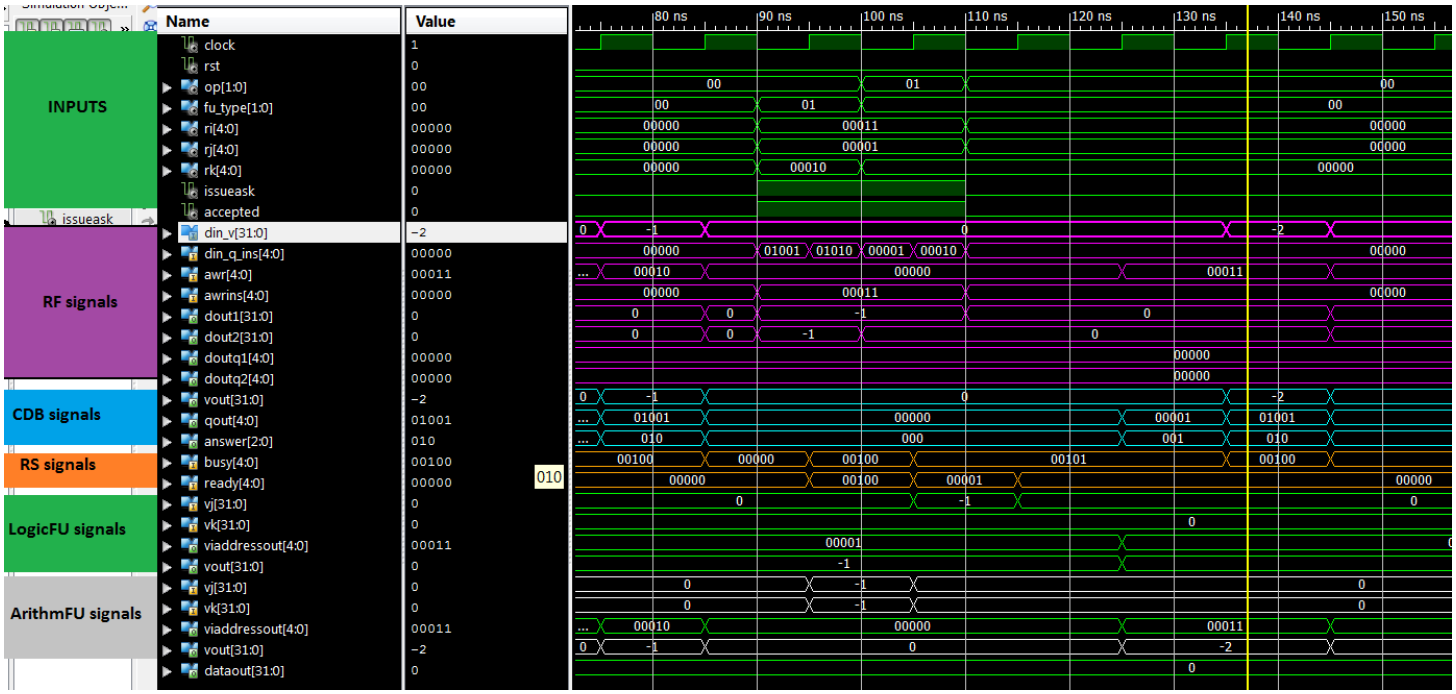
```
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00011";
Rj <="00001";
Rk <="00010";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="01";
FU_type <="00";
Ri <="00011";
Rj <="00001";
Rk <="00000";
Issueask<='1';
wait for Clock_period*1;
```

Simulation:



Στον κύκλο που ξεκινά στα 90 ns γίνεται issue η εντολή + και μετά τον παλμό γράφεται στο πεδίο Q του RF το tag του RS το οποίο θα γράψει στην αντίστοιχη θέση προορισμού V που είναι η θέση 00011 καθώς γράφεται και το κατάλληλο RS. Ένα κύκλο μετά βλέπουμε ότι η ArithmFU είναι έτυμη να εκτελέσει την πράξη -1 +(-1) . Παράλληλα η επόμενη εντολή (and) ετοιμάζει τα σήματα που θα γραφούν στο επόμενο RS και πανωγράφει το tag που έχει ο register 00011 στο πεδίο Q του RF αφού και αυτή η εντολή θέλει να γράψει στην ίδια θέση μνήμης με την προηγούμενη.

2 κύκλους μετά την εγγραφή της εντολής AND βγαίνουν ταυτόχρονα τα αποτελέσματα και των 2 πράξεων (-2 και 0) και θέλουν και οι 2 να γραφούν στον ίδιο προορισμό που είναι ο register 00011. Λόγο της ταυτόχρονης έκδοσης των αποτελεσμάτων υπάρχει σύγκρουση στην προσπάθεια των FUs να χρησιμοποιήσουν το CDB και αναγκαστικά πρέπει να μπουν σε μια σειρά και να χρησιμοποιήσουν το CDB σειριακά . Το CDB επιλέγει να αφήσει την εντολή AND να γράψει πρώτη τα αποτελέσματα της και γιαυτο βλέπουμε το Vout του CDB να είναι 0.



Το τελευταίο σήμα που φαίνεται στο simulation dataout είναι η έξοδος του καταχωριτή 00011 του πεδίου V από τον RF το οποίο βλέπουμε και παραμένει 0 δηλαδή γράφεται σωστά μετά την ακμή του ρολογιού . Ταυτόχρονα έρχεται η σειρά της εντολής + να γράψει το CDB και να μεταδώσει τα αποτελέσματα του στο RF το οποίο γίνεται επιτυχώς . Μετά τον επόμενο παλμό του ρολογιού όμως το σήμα dataout του register 00011 παραμένει 0 το οποίο δείχνει ότι δεν γράφτηκε το αποτέλεσμα -2 που μόλις μεταδόθηκε . Το γεγονός αυτό συνέβη επειδή η εντολή + ξεκίνησε την εκτέλεση της πριν την εντολή AND και αν έγραφε το αποτέλεσμα της μετά από αυτή θα είχαμε WAW hazard το οποίο αποφεύγουμε με τον αυτόματο έλεγχο που έχει ο RF για το αν η τιμή που πάει να γραφεί έχει tag ίδιο με αυτό της αντίστοιχης θέσης στο πεδίο Q, το οποίο στην περίπτωση μας δεν ισχύει διότι η εντολή AND πανώγραψε το tag της +. Έτσι τελειώνει η εντολή και απελευθερώνετε και αυτό το RS .

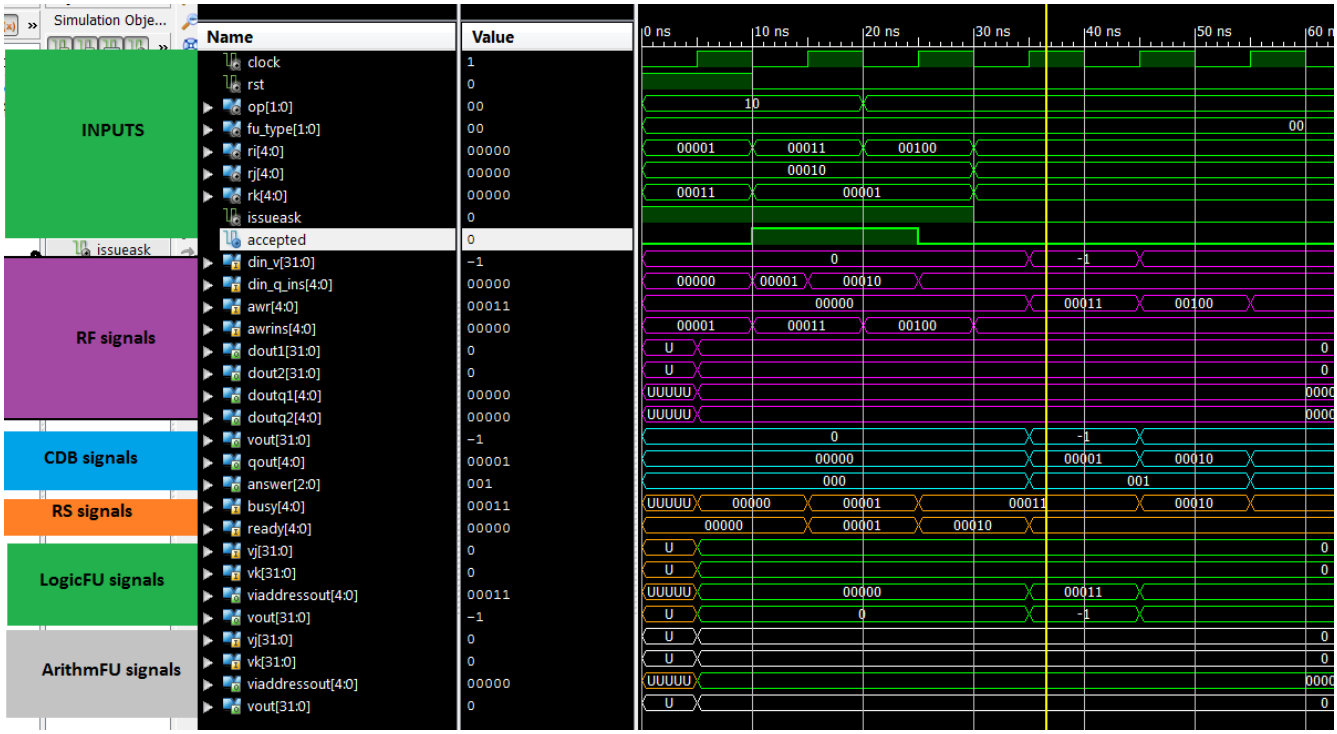
4.

Σε αυτή την προσομοίωση θα ελέγξουμε την περίπτωση που 2 συνεχόμενες λογικές εντολές χρησιμοποιούν την ίδια λειτουργική μονάδα χωρίς να έχουν εξάρτιση η μια από την άλλη. Η πρώτη εντολή είναι μια NOT Και η δεύτερη μια OR.

Ipns:

```
 wait for Clock_period*1;
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00011";
Rj <="00010";
Rk <="00001";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="00";
Ri <="00100";
Rj <="00010";
Rk <="00001";
Issueask<='1';
```

Simulation:



Σε αυτή την προσομοίωση βλέπουμε ότι στο 2^{ο} κύκλο εισέρχονται τα σήματα της πράξης NOT στο κύκλωμα και την στιγμή της ακμής του ρολογιού γράφονται οι απαραίτητοι registers του RS και του RF αποθηκεύοντας την πληροφορία της εντολής . Τη στιγμή εκείνη βλέπουμε το σήμα busy και ready του RS-1 και ενεργοποιούνται το οποίο δηλώνει ότι το RS είναι απασχολημένο και έτοιμο για να εκτελέσει την εντολή. Στον επόμενο κύκλο εισέρχονται τα σήματα της εντολής AND και γράφονται οι κατάλληλοι registers ενεργοποιώντας τα σήματα busy και ready του RS-2 . Ταυτόχρονα σβήνει το σήμα ready του RS-1 καθώς έχει ξεκινήσει ήδη για ένα κύκλο η πράξη της προηγούμενης εντολής και έπειτα ξεκινά η εκτέλεση της εντολής AND . Αφού περάσει ένας κύκλος ακόμα βγαίνει το αποτέλεσμα της πρώτης εντολής (-1) , δημοσιεύεται στο CDB καταλήγοντας στην είσοδο του RF και το σήμα ready του RS-2 γίνεται 0 . Ένα κύκλο αργότερα η τιμή γράφεται στον σωστό καταχωρειτή του RF , το σήμα busy του RS-1 γίνεται 0 απελευθερώνοντας το για να μπορεί να δεχτεί άλλη εντολή και το αποτέλεσμα της πράξης AND που μόλις τελείωσε δημοσιεύεται στο CDB καταλήγοντας στην είσοδο του RF όπου και γράφεται στον επόμενο κύκλο .Τέλος το σήμα busy του RS-2 γίνεται 0 και η εντολή έχει φτάσει στο τέλος της .

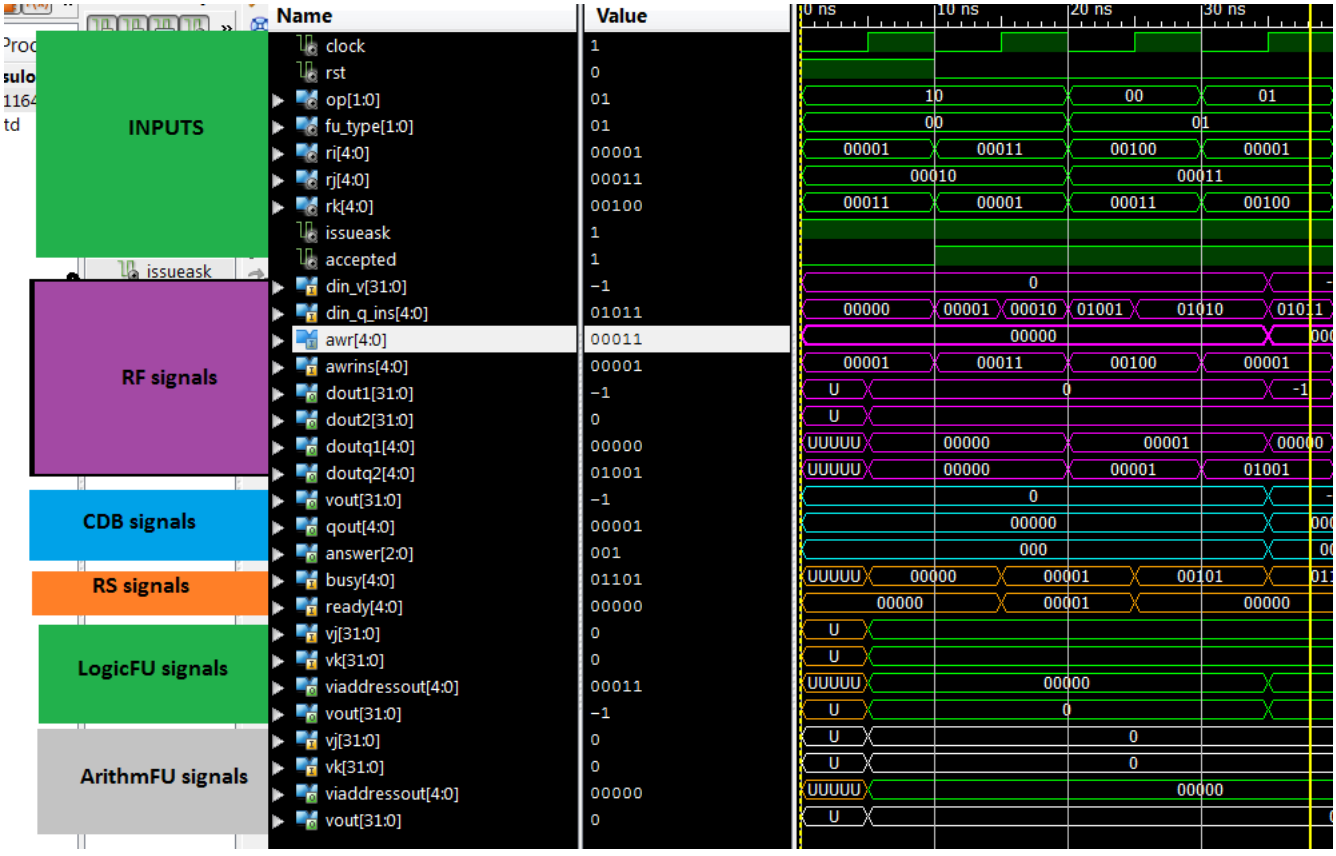
5.

Σε αυτή την προσομοίωση θα ελέγξουμε την περίπτωση που 3 συνεχόμενες εντολές έχουν εξάρτιση η μια από την άλλη. Η πρώτη εντολή είναι μια NOT , η δεύτερη μια αριθμητική ADD με εισόδους την έξοδο της NOT και τέλος μια αριθμητική SUB με εισόδους την έξοδο της NOT και την έξοδο της ADD .

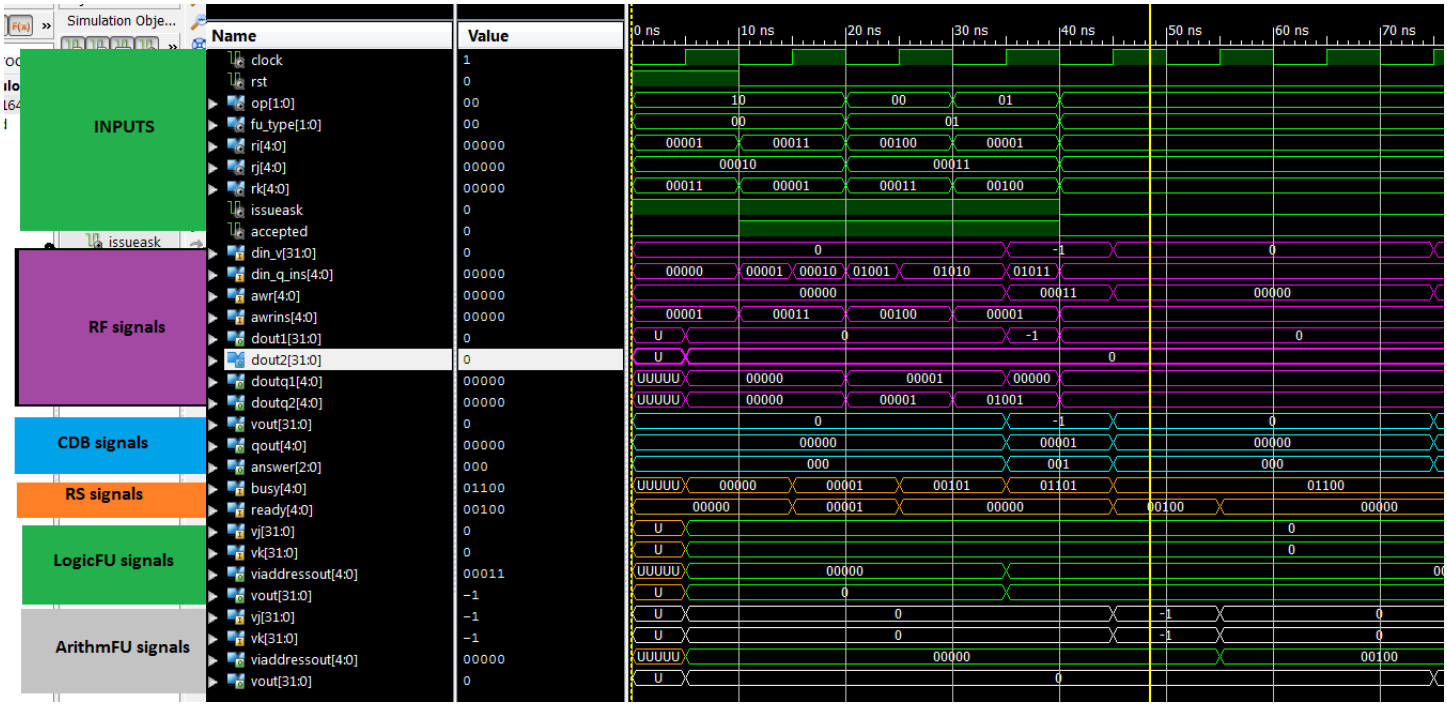
INPS:

```
 wait for Clock_period*1;
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00011";
Rj <="00010";
Rk <="00001";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00100";
Rj <="00011";
Rk <="00011";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="01";
FU_type <="01";
Ri <="00001";
Rj <="00011";
Rk <="00100";
Issueask<='1';
```

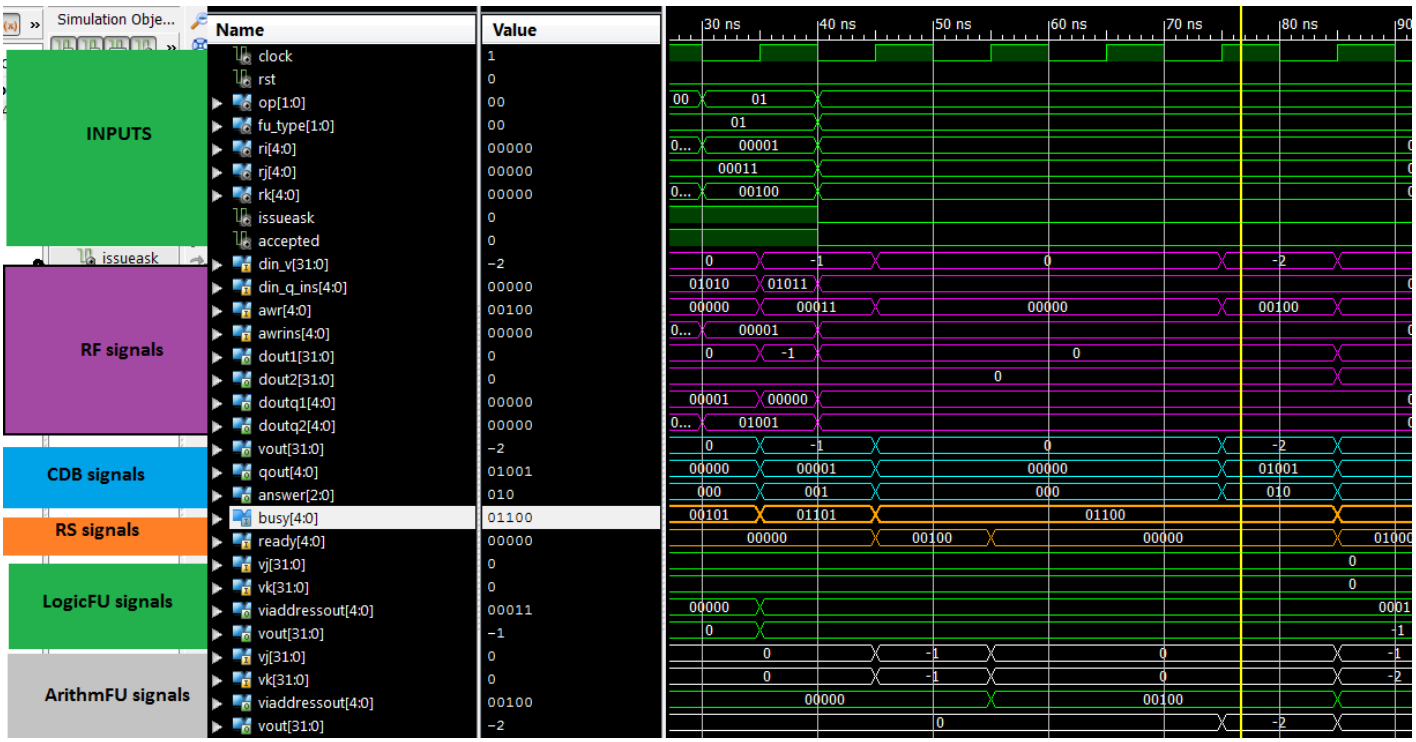
Simulation :



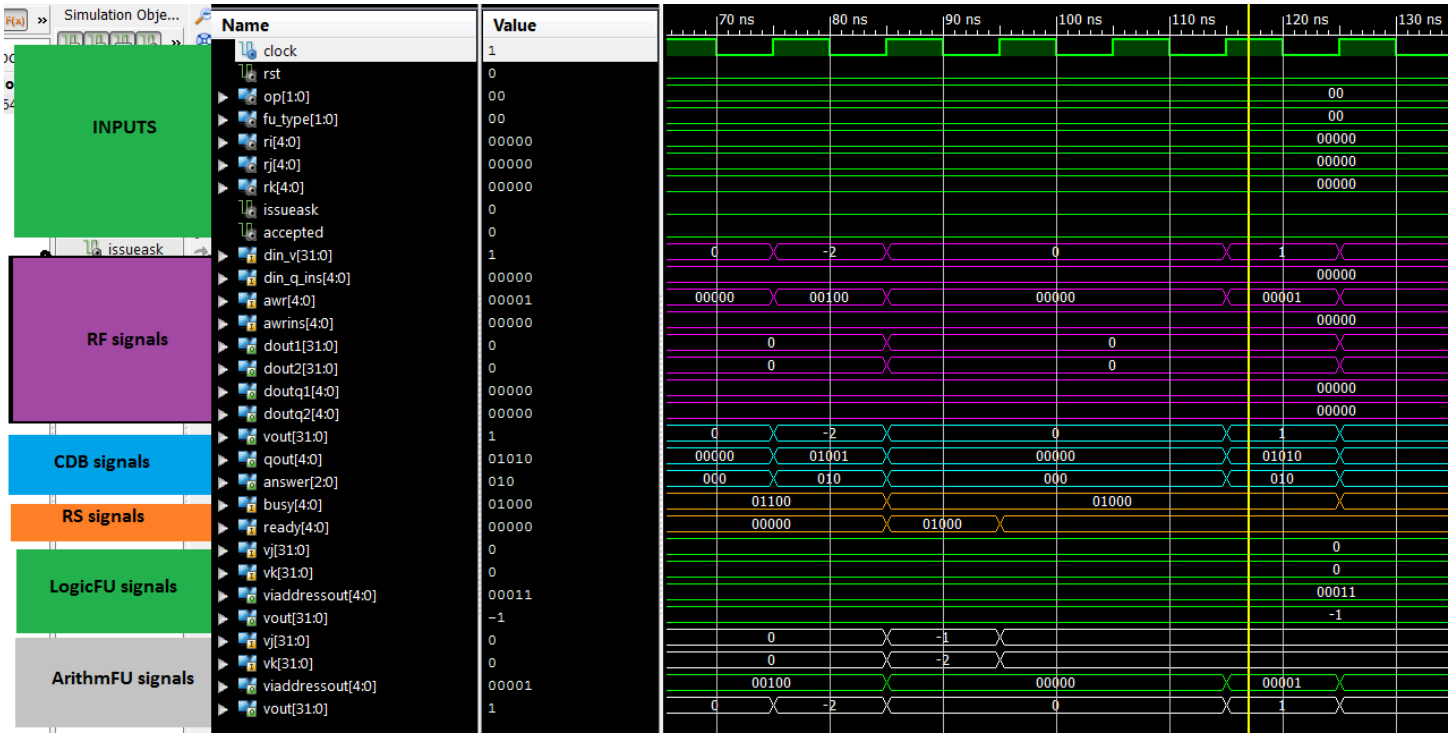
Στον 2^{ο} κύκλο της παραπάνω προσομοίωσης γίνεται issue και γράφονται τα σήματα της εντολής NOT , στον 3^{ο} γίνεται issue και γράφονται τα σήματα της εντολής ADD και στον 4^{ο} γίνεται issue και γράφονται τα σήματα της εντολής SUB. Επίσης στον 4^{ο} κύκλο τελειώνει και δημοσιεύεται το αποτέλεσμα (-1) της πράξης NOT μέσο του CDB και φτάνει στην είσοδο του RF. Σε αυτόν τον κύκλο μπορούμε να παρατηρήσουμε τα σήματα busy των RS που είναι 01101 επαληθεύοντας έτσι ότι όντως έχουν καταχωρηθεί 2 αριθμητικές και 1 λογική εντολή και προσπαθούν να εκτελεστούν .



Στον επόμενο κύκλο δηλαδή στον 5^{ο} το σήμα busy του RS-1 γίνεται 0 ,γράφεται το αποτέλεσμα της εντολής NOT στον RF και τα RS που χρειάζονται το αποτέλεσμα αυτής της πράξης το βρίσκουν στο CDB και το αποθηκεύουν . Σε αυτό τον κύκλο βλέπουμε λιπών ότι το σήμα ready του RS-3 που είναι υπεύθυνο για την πράξη ADD γίνεται 1 καθώς οι είσοδοι της ArithmFU γίνονται -1 και -1 και ξεκινά ο αριθμητικός υπολογισμός της πράξης.



3 κύκλους αργότερα δηλαδή στα 75 ns εκδίδεται το αποτέλεσμα(-2) της πράξης ADD και μέσο του CDB φτάνει στην είσοδο του RF . Ένα κύκλο αργότερα δηλαδή στα 85 ns αποθηκεύεται το αποτέλεσμα στο RF και ταυτόχρονα το RS-4 που περιμένει το αποτέλεσμα της πράξης ADD το βρίσκει στο CDB και το παίρνει από εκεί. Τη στιγμή αυτή το σήμα ready του RS-4 γίνεται 1 , οι είσοδοι τη ArithmFU γίνονται -1 και -2 και η πράξη της αφαίρεσης ξεκινά .



3 κύκλους αργότερα δηλαδή στα 115 ns εκδίδεται το αποτέλεσμα της πράξης(1) SUB και μέσο του CDB φτάνει στην είσοδο του RF. Ένα κύκλο αργότερα δηλαδή στα 125 ns αποθηκεύεται το αποτέλεσμα στο RF , το busy και αυτού του RS μηδενίζεται και το simulation φτάνει στο τέλος του.

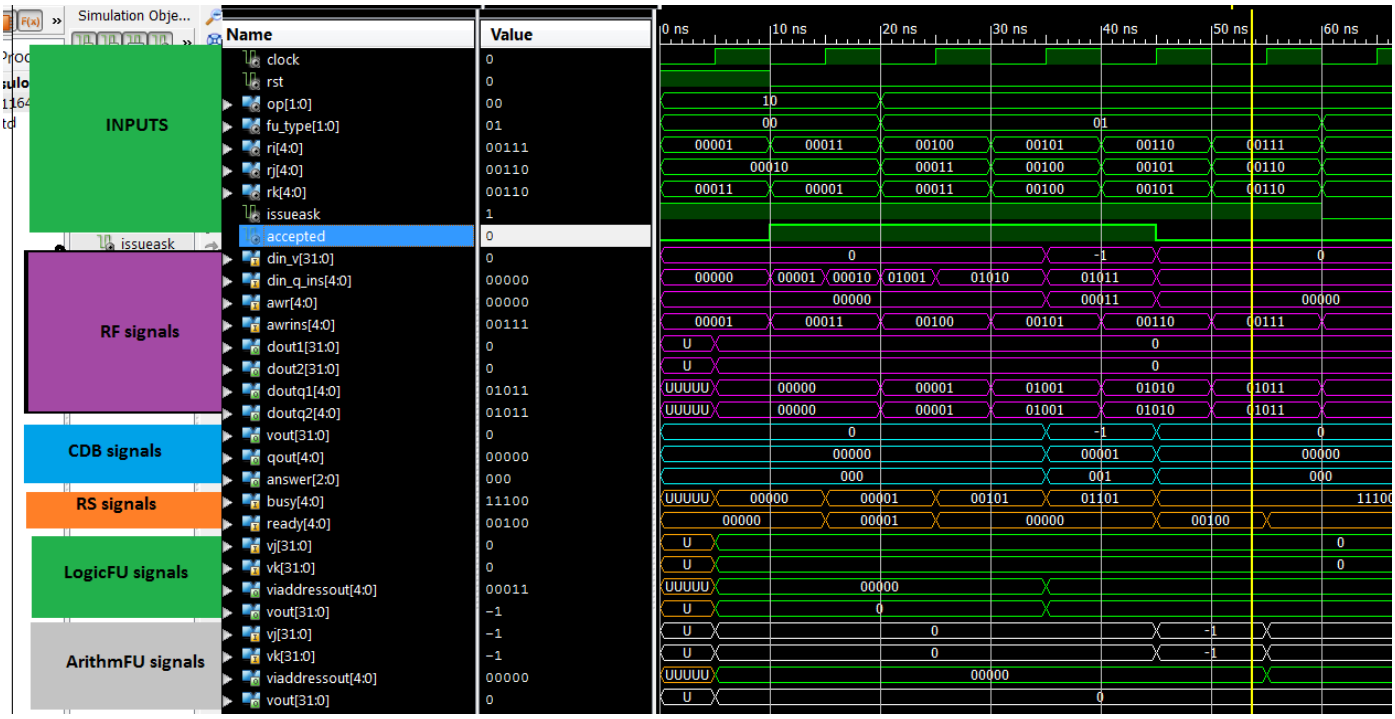
6.

Σε αυτή την προσομοίωση θα ελέγξουμε την περίπτωση που προσπαθούν να εκδοθούν 4 συνεχόμενες αριθμητικές πράξεις κάνοντας το σύστημα να γεμίσει και να μην μπορεί να δεχτεί και τις 4 αφού έχουμε μόνο 3 RS . Η πρώτη εντολή είναι μια NOT ώστε να μπορέσουμε να κάνουμε τις αριθμητικές πράξεις χρησιμοποιώντας το αποτέλεσμα της , η δεύτερη μια αριθμητική ADD με εισόδους την έξοδο της NOT , η Τρίτη ξανά μια ADD με εισόδους την έξοδο της προηγούμενης ADD , η τέταρτη Τρίτη ξανά μια ADD με εισόδους την έξοδο της προηγούμενης ADD και τέλος ακόμα μια ADD που σύμφωνα με το προσδοκώμενο αποτέλεσμα δεν πρέπει να γίνει δεχτή από το σύστημα .

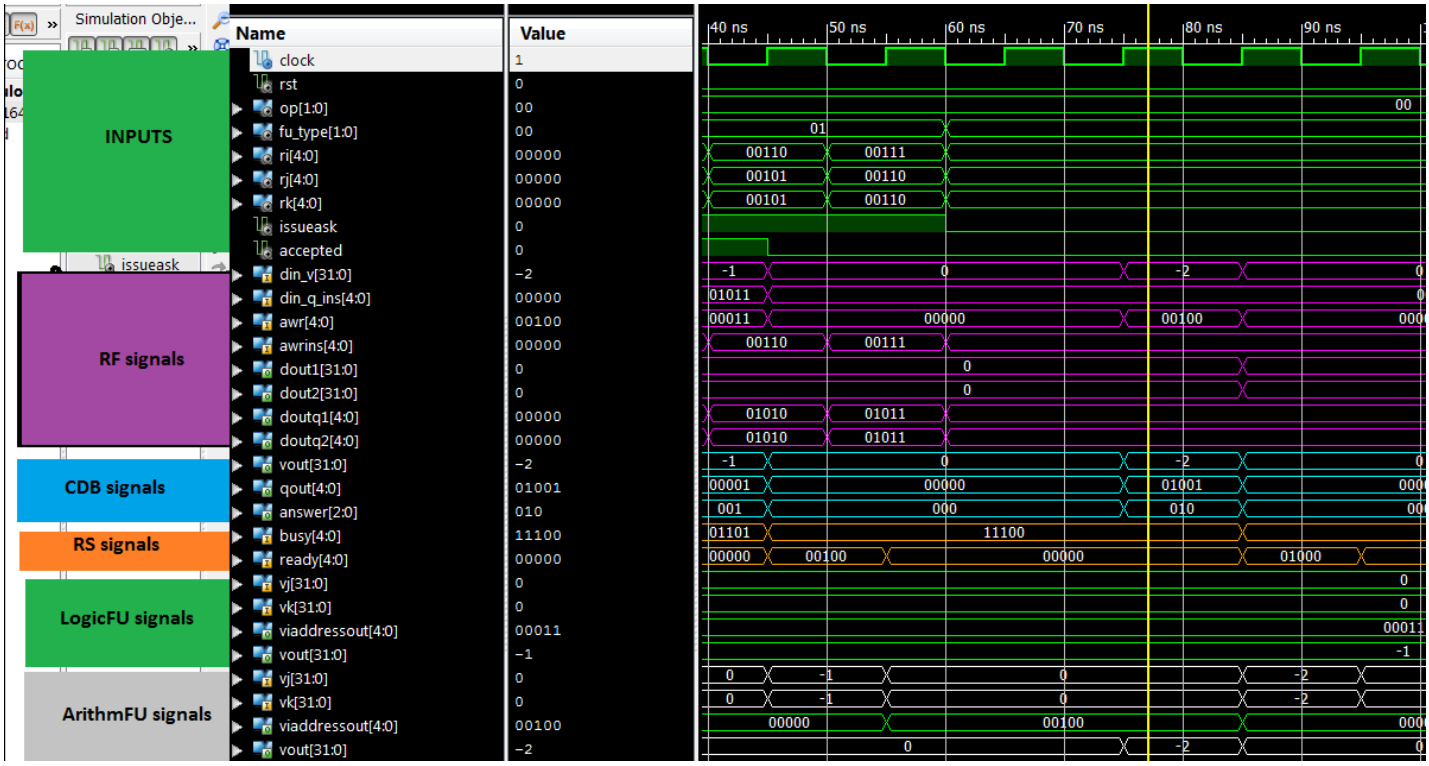
Inps:

```
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00011";
Rj <="00010";
Rk <="00001";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00100";
Rj <="00011";
Rk <="00011";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00101";
Rj <="00100";
Rk <="00100";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00110";
Rj <="00101";
Rk <="00101";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00111";
Rj <="00110";
Rk <="00110";
Issueask<='1';
```

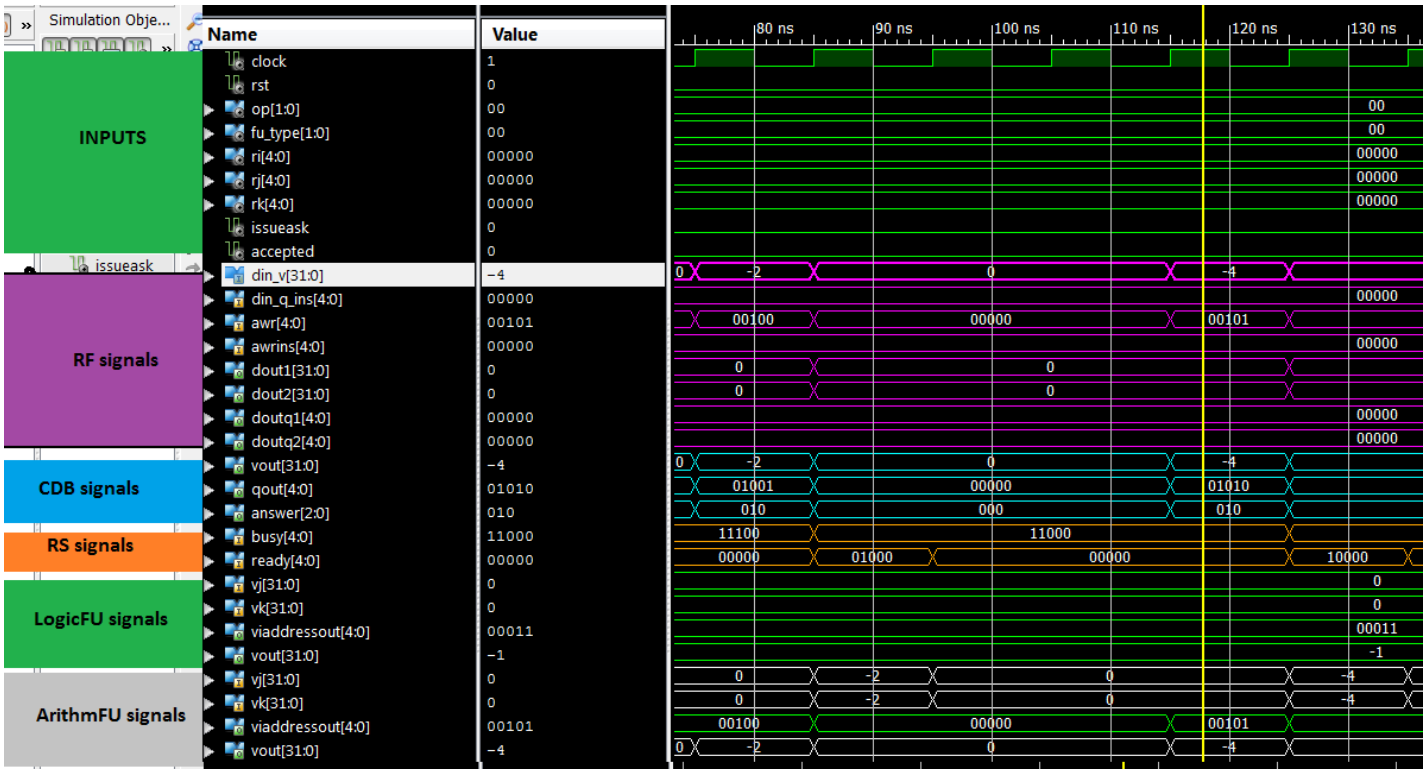
Simulation:



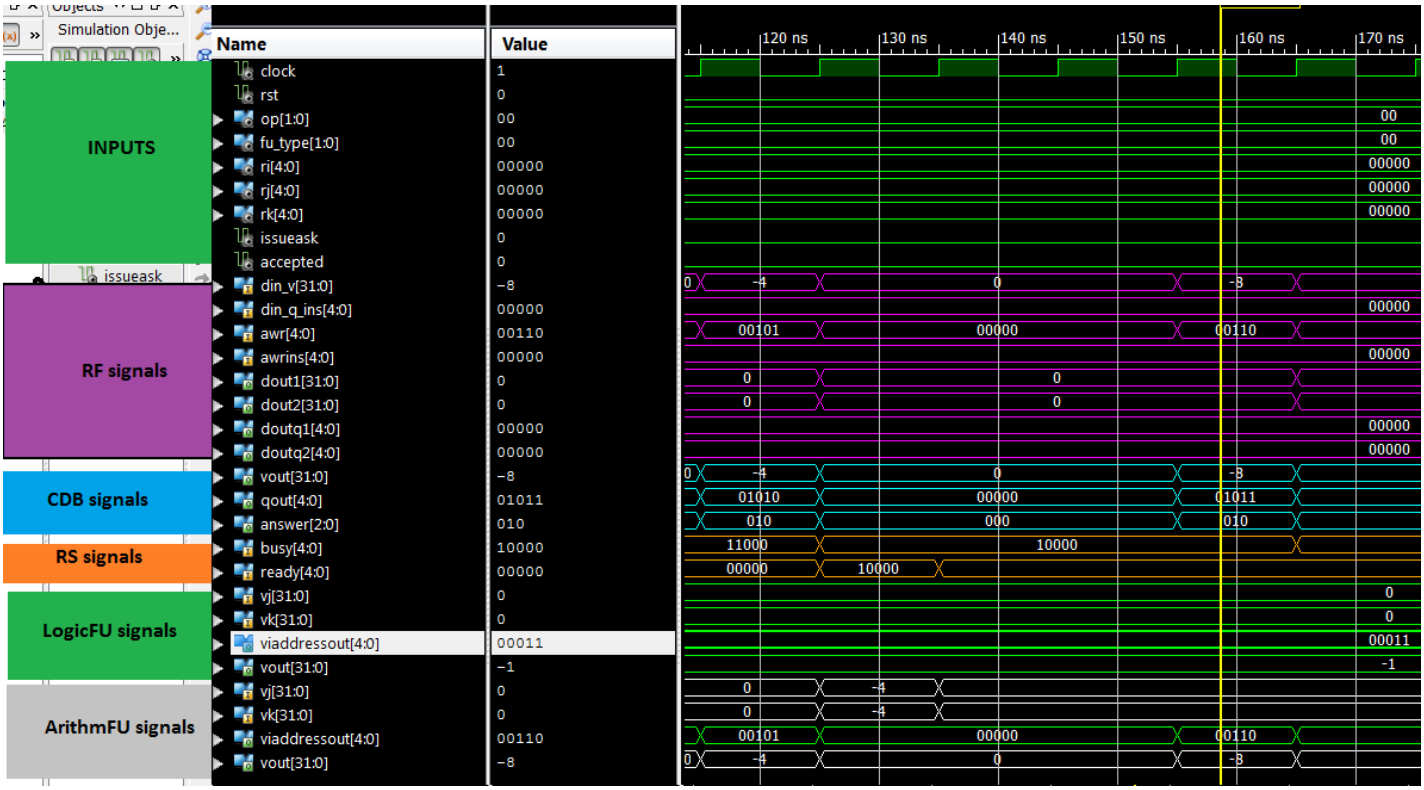
Όπως βλέπουμε στην παραπάνω προσομοίωση γίνονται issue και εισέρχονται σωστά στο κύκλωμα οι εντολή NOT και η 3 εντολές ADD ενώ όταν φτάσει η στιγμή για να εισέρθει η 4^{η} ADD τα σήματα busy είναι 11100 το οποίο σημαίνει ότι δεν υπάρχει χώρος για να μπει και άλλη αριθμητική πράξη και γι’ αυτό η απάντηση του κυκλώματος προς την εξωτερική μονάδα Instruction Fetch γίνεται 0 και αγνοείτε η εντολή αυτή σε αυτό τον κύκλο . Επίσης μπορούμε να δούμε στο σχήμα της προσομοίωσης ότι στον κύκλο 4 βγαίνει το αποτέλεσμα της πράξης NOT και ένα κύκλο αργότερα γράφεται στο RF και το RS που το χρειάζεται ξεκινώντας τον υπολογισμό της πρώτης ADD.



3 κύκλους αργότερα δηλαδή στα 75 ns βγαίνει το αποτέλεσμα(-2) της πρώτης πρόσθεσης και ένα κύκλο μετά στα 85 ns γράφεται στο RF και στο RS-4 το οποίο γίνεται ready και προωθεί στη ArithmFU τα περιεχόμενα του για να ξεκινήσει η πράξη .



3 κύκλους αργότερα δηλαδή στα 115 ns βγαίνει το αποτέλεσμα(-4) της δεύτερης πρόσθεσης και ένα κύκλο μετά στα 125 ns γράφεται στο RF και στο RS-5 το οποίο γίνεται ready και προωθεί στη ArithmFU τα περιεχόμενα του για να ξεκινήσει η πράξη .



3 κύκλους αργότερα δηλαδή στα 155 ns βγαίνει το αποτέλεσμα(-8) της τρίτης πρόσθεσης και ένα κύκλο μετά στα 165 ns γράφεται στο RF και το πρόγραμμα τελειώνει καθώς είχε απορρίψει την τελευταία ADD την οποία δεν ξαναδώσαμε ως είσοδο αργότερα .

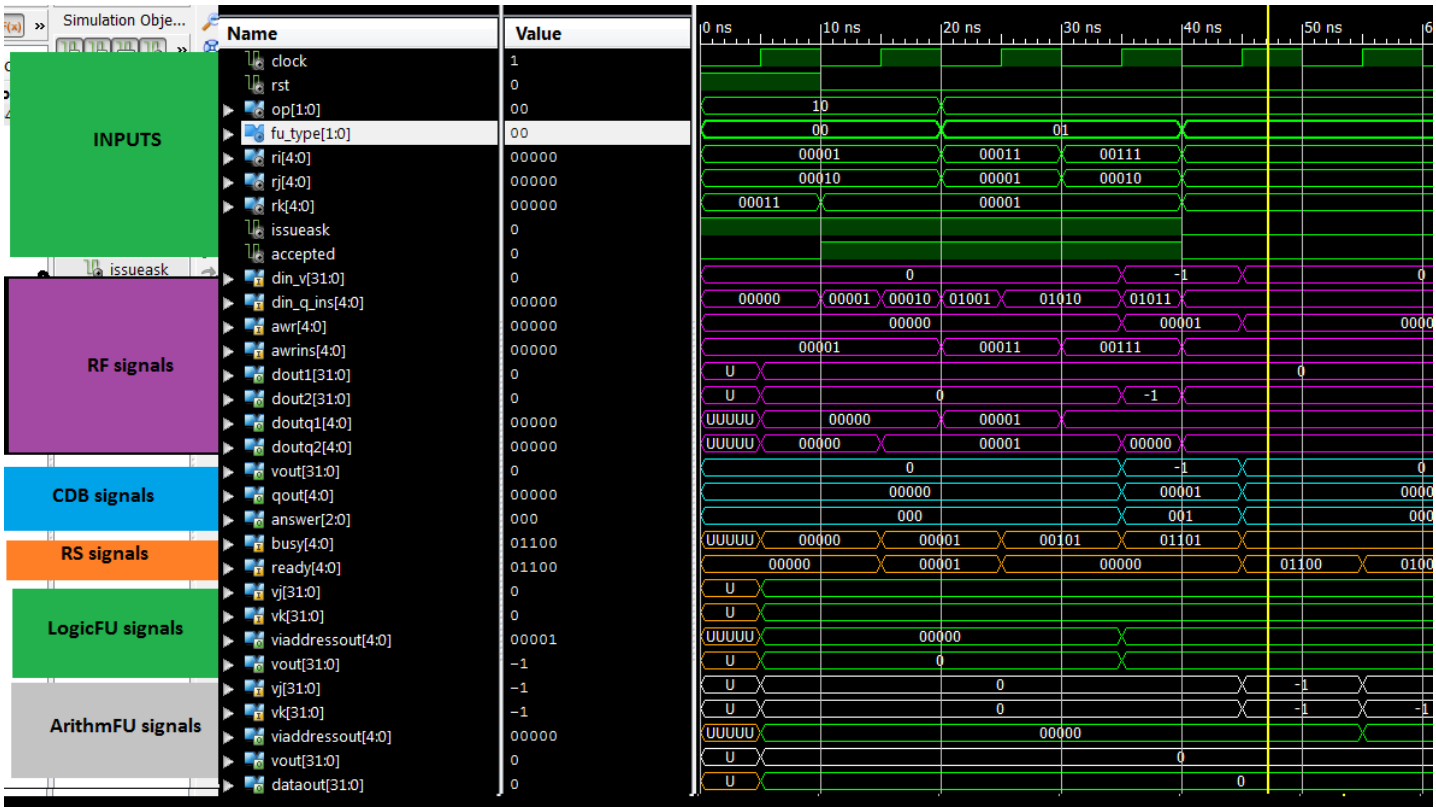
7.

Σε αυτή την προσομοίωση θα ελέγξουμε την περίπτωση που 2 αριθμητικές εντολές ADD περιμένουν το αποτέλεσμα μια λογικής NOT για να ξεκινήσουν την εκτέλεση τους συγκρουόμενες για το ποια θα χρησιμοποιήσει πρώτη την ArithmFU καθώς γίνονται ταυτόχρονα ready. Επίσης μέσω της συγκεκριμένης προσομοίωσης μπορούμε να ελέγξουμε την ορθότητα της μεθόδου fall-through του RF αφού το αποτέλεσμα της NOT βγαίνει στον κύκλο που εκδίδεται η εντολή ADD.

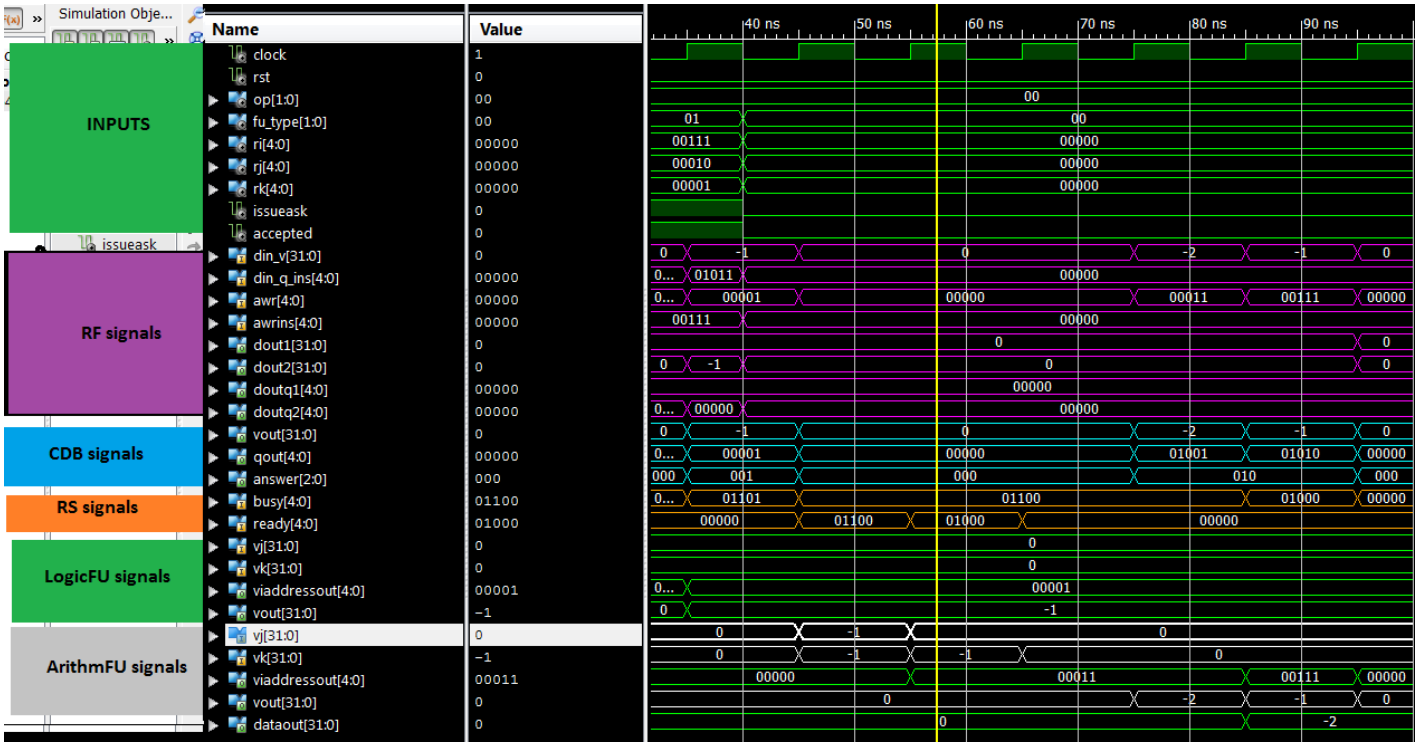
Inps:

```
Rst <='0';
op <="10";
FU_type <="00";
Ri <="00001";
Rj <="00010";
Rk <="00001";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00011";
Rj <="00001";
Rk <="00001";
Issueask<='1';
wait for Clock_period*1;
Rst <='0';
op <="00";
FU_type <="01";
Ri <="00111";
Rj <="00010";
Rk <="00001";
Issueask<='1';
```

Simulation :



Στην προσομοίωση βλέπουμε ότι γίνεται κανονικά issue η εντολή NOT , η $1^{η}$ εντολή ADD και τη στιγμή που εκδίδεται η $2^{η}$ ADD πραγματοποιείτε σωστά η μέθοδος fall-through και το RS-4 παίρνει τα απαραίτητα inputs που χρειάζεται . Εκείνη τη στιγμή το RS-3 βρίσκει στο CDB τα πεδία τιμών που χρειάζεται και ενεργοποιούνται τα σήματα ready και των 2 RS προσπαθώντας να χρησιμοποιήσουν ταυτόχρονα την ArithmFU. Σε αυτή την περίπτωση το σύστημα επιτρέπει στο RS-3 να χρησιμοποιήσει πρώτο το FU και το RS-4 είναι αναγκασμένο τα περιμένει .



Ένα κύκλο αργότερα το σύστημα επιτρέπει και στην δεύτερη πρόσθεση να χρησιμοποιήσει την FU για την εκτέλεση της πράξης της . 2 κύκλους αργότερα δηλαδή στα 75 ns βγαίνει το αποτέλεσμα της(-2) της 1^{ης} πρόσθεσης , δημοσιεύεται στο CDB και καταλήγει στο RF . Στα 85 ns το αποτέλεσμα -2 γράφεται στον register 00011 του RF του οποίου η έξοδος φαίνεται ως τελευταίο σήμα στο simulation για επαλήθευση και βγαίνει το αποτέλεσμα της 2^{ης} πρόσθεσης το οποίο και δημοσιεύετε. Τέλος γράφεται το αποτέλεσμα της 2^{ης} πρόσθεσης (-1) στον RF και τελειώνει η προσομοίωση .

Ο κώδικας του συστήματος είναι ίδιος με τον κώδικα που σας δόθηκε στο πρώτο μέρος της εργασίας . Το σύστημα βλέπουμε και λειτουργεί σωστά για όλες τις απλές περιπτώσεις που ελέγξαμε καθώς και στις ακραίες περιπτώσεις (corner cases) που είδαμε στα τελευταία simulations . Σύμφωνα με τα αποτελέσματα που έχουμε μπορούμε να πούμε με αρκετή σιγουριά ότι το κύκλωμα λειτουργεί όπως θέλουμε .