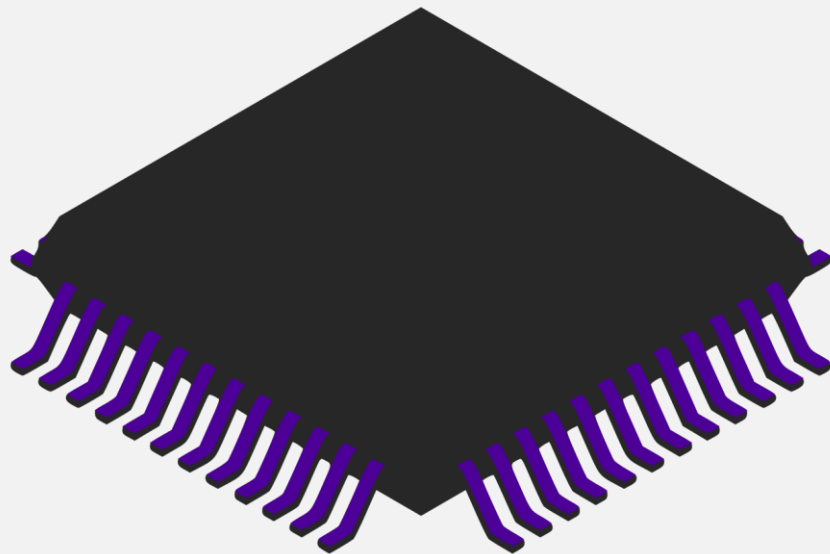


ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 4^{ης} Εργαστηριακής Άσκησης

«ΟΛΟΚΛΗΡΩΣΗ ΕΝΟΣ ΕΠΕΞΕΡΓΑΣΤΗ ΠΟΛΛΑΠΛΩΝ
ΚΥΚΛΩΝ & ΕΞΕΡΕΣΕΩΝ»

















 Ζαχαριουδάκης Νικόλας 2016030073

 Γαλάνης Μιχάλης 2016030036

ΠΕΡΙΕΧΟΜΕΝΑ

*Οι σύνδεσμοι για τις παρακάτω ενότητες είναι διαδραστικοί.
Πιέστε πάνω στο επιθυμητό τμήμα για τη μετάβαση σε αυτό.*

	ΕΙΣΑΓΩΓΗ	1
	Σκοπός Εργαστηρίου	1
	Προαπαιτούμενα	1
	A – ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ ΜΕ FSM	1
	A.1 Επισκόπηση	1
	A.2 Σχεδίαση & Υλοποίηση.....	4
	B – ΕΞΑΙΡΕΣΕΙΣ	4
	B.1 Επισκόπηση.....	4
	B.2 Υλοποίηση.....	5
	Γ – ΕΠΑΛΗΘΕΥΣΗ ΣΥΣΤΗΜΑΤΟΣ	6
	Γ.1. Προσομοίωση Π.Α #1 (Κυματομορφές)	6
	Γ.2. Προσομοίωση Π.Α #2 (Κυματομορφές)	7
	Γ.3. Προσομοίωση Π.Α #3 (Κυματομορφές)	8
	Δ – BLOCK DIAGRAMS.....	9

ΕΙΣΑΓΩΓΗ

Σκοπός Εργαστηρίου

Σκοπός της 4^{ης} εργαστηριακής άσκησης είναι η μετατροπή του single – cycle επεξεργαστή που είχαμε έως τώρα, σε έναν multi – cycle έτσι ώστε να χωρίσουμε 1 μεγάλο κύκλο εκτέλεσης εντολής σε μικρότερους και να έχουμε τελικά μικρότερες τιμές CPI. Η μετατροπή είναι ένα αναγκαίο ενδιάμεσο βήμα που χρειαζόμαστε για το pipelining που θα μελετηθεί σε βάθος στη 5^η εργαστηριακή άσκηση.

Ασχολούμαστε επίσης και με εξαιρέσεις που μας επιτρέπουν να διαχειριστούμε προβληματικές καταστάσεις που μπορεί να προκύψουν κατά την εκτέλεση των εντολών της μνήμης.

Προαπαιτούμενα

Χρειάζεται η άπαιστη γνώση της VHDL και των δομών λειτουργίας και η εξοικείωση στα εργαλεία σχεδιασμού της Xilinx. Θα χρησιμοποιηθεί επίσης πλήρως το υλικό που παράχθηκε στην προηγούμενη εργαστηριακή άσκηση.

Α – ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ ΜΕ FSM

Α.1 Επισκόπηση

Για επεξεργαστή με πολλαπλούς κύκλους χρειαζόμαστε μια μηχανή πεπερασμένων καταστάσεων. Αυτή επιτυγχάνεται στο Control Unit του επεξεργαστή και διαθέτει μια κατάσταση για κάθε ενέργεια της εντολής που μελετάμε. Πιο συγκεκριμένα έχουμε τις παρακάτω καταστάσεις:

Α) Γενικές Καταστάσεις

- **S0** : DECODE
- **S1** : FETCH

Β) Ανάγνωση / Εγγραφή στη Μνήμη

- **S2** : MEM_ADR (lw/sw)
- **S3** : MEM_READ (lw)
- **S4** : MEM_WRITEBACK (lw)
- **S5** : MEM_WRITE (sw)

Γ) Εκτέλεση R-Type Εντολών

- **S6** : R_EXECUTE
- **S7** : R_WRITE_BACK

Δ) Εκτέλεση Branch Εντολών

- **S8** : BRANCH

Ε) Εκτέλεση I-Type Εντολών

- **S9** : I_EXECUTE
- **S10** : I_WRITE_BACK

ΣΤ) Εκτέλεση J-Type Εντολών

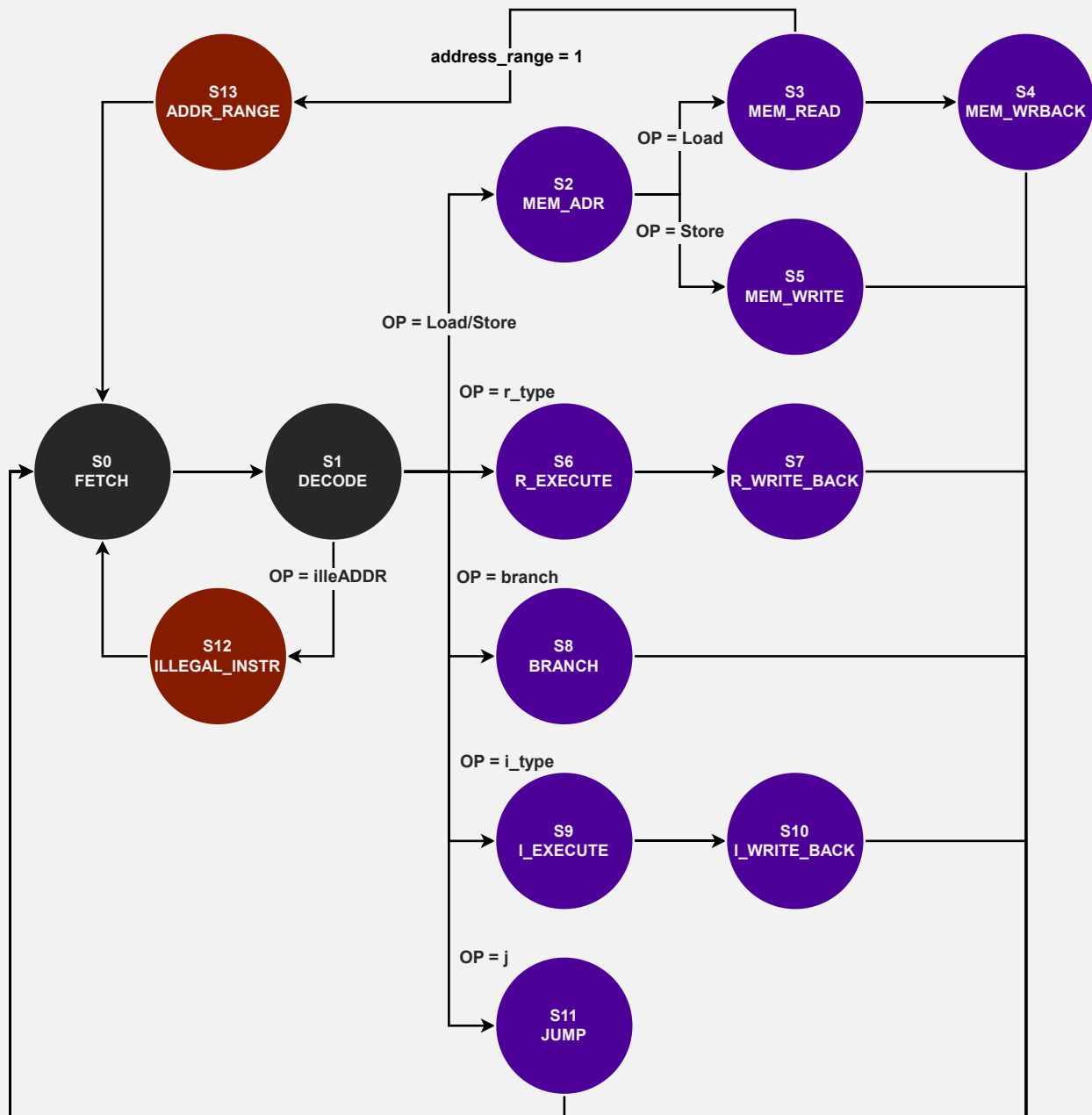
- **S11** : JUMP

Ζ) Καταστάσεις Εξαιρέσεων

- **S12** : ILLEGAL_INSTR
- **S13** : ADDR_RANGE

Παρατηρούμε ότι οι πρώτες δύο καταστάσεις S0, S1 είναι κοινές για εντολές οποιουδήποτε τύπου ενώ στη συνέχεια μεταβαίνουμε σε καταστάσεις αναλόγως με τις ανάγκες τις κάθε εντολής.


Σε επόμενο διάγραμμα φαίνεται η αναπαράσταση της μηχανής πεπερασμένων καταστάσεων. Έχει σχεδιαστεί οριζόντια έτσι ώστε να φαίνεται η χρονική διάρκεια (σε κύκλους) της κάθε εντολής. Το χρώμα στο παραπάνω διάγραμμα αποτυπώνει τη γενική κατηγορία των καταστάσεων.



Εύκολα παρατηρεί κάποιος ότι το critical path είναι στη περίπτωση που φορτώνουμε πληροφορία από τη μνήμη διότι απαιτεί 5 κύκλους ρολογιού. Οι έξοδοι δε φαίνονται στο συγκεκριμένο σχήμα διότι έχουμε μεγάλο αριθμό από αυτές και παρουσιάζονται σε ξεχωριστό πίνακα στη συνέχεια.

Όνομα Εξόδου	S0	S1	S2	S3	S4	S5	S6
ALUctr	0000	0000	0000	d	0001	0001	instr _[3:0]
Extop	0	0	1	d	0	0	
Rb_RF_sel	0	0	1	d	0	1	0
PC_LdEn	1	0	0	d	0	0	0
ALUSrcA	0	0	1	d	1	1	1
ALUSrcB	01	11	10	d	00	00	00
PC_source	00	00	00	d	01	01	00
IRWrite	1	0	0	d	0	0	0
MemWr	0	0	0	d	0	1	0
RF_WrEn	0	0	0	d	1	0	0
PC_LdEn_cond	0	0	0	d	0	0	0
shift16	0	0	0	d	0	0	0
ByteEnable	0	0	0	d	lb	sb	0
CAUSEwrite	0	0	0	d	0	0	0
intCause	0	0	0	d	0	0	0
EPCwrite	0	0	0	d	0	0	0
RF_WrData_sel	d	d	00	d	01	00	00

Όνομα Εξόδου	S7	S8	S9	S10	S11	S12	S13
ALUctr	0001	0001	*	0001	1111	0001	0001
Extop	0	1	li or addi	0	0	0	0
Rb_RF_sel	1	1	0	1	0	0	0
PC_LdEn	0	0	0	0	1	1	1
ALUSrcA	1	1	1	1	1	0	0
ALUSrcB	00	00	10	00	00	01	01
PC_source	01	01	00	01	10	11	11
IRWrite	0	0	0	0	0	0	0
MemWr	1	0	0	0	0	0	0
RF_WrEn	1	0	0	1	0	0	0
PC_LdEn_cond	0	*	0	0	0	0	0
shift16	0	0	Lui	0	0	0	0
ByteEnable	0	0	0	0	0	0	0
CAUSEwrite	0	0	0	0	0	1	1
intCause	0	0	0	0	0	0	1
EPCwrite	0	0	0	0	0	1	1
RF_WrData_sel	func _[5:4]	00	00	11	00	10	10

 **Παρατήρηση:** (d) ορίζεται ως don't care ενώ (*) δηλώνει ότι υπάρχει μια πολύπλοκη πράξη που δεν αναφέρεται.

A.2 Σχεδίαση & Υλοποίηση

Στο Control Unit έχουμε πλέον 5 processes από τα οποία τα 3 χρησιμοποιούνται αποκλειστικά για τη μηχανή πεπερασμένων καταστάσεων. Τα πρώτα δύο είναι υπεύθυνα για την αποκωδικοποίηση του OPCODE (που προέρχεται από τη μνήμη) και βοηθάνε αργότερα στην επιλογή των καταστάσεων της FSM. Έχουμε ένα process που διαχειρίζεται τη μετάβαση των καταστάσεων και άλλο ένα που αναθέτει εξόδους της κάθε κατάστασης. Τέλος, διαθέτουμε και ένα process που είναι υπεύθυνο για την αρχικοποίηση της FSM. Κώδικάς της δε θα παρουσιαστεί καθώς είναι εκτενής αλλά είναι διαθέσιμος μαζί με την αναφορά αυτή.

Με την εισαγωγή της FSM στο σύστημά μας, έγιναν διάφορες αλλαγές στη δομή του. Η κύριες αλλαγές αφορούσαν τη προσθήκη ενδιάμεσων καταχωρητών που διατηρούν αποτελέσματα της κάθε βαθμίδας. Στο TOP MODULE προστέθηκαν 2 registers (**IR, MDR**) για την διατήρηση των τιμών εξόδου της μνήμης (εντολές και δεδομένα). Για το MDR το we είναι ίσο με 1 καθώς κρατάμε την τιμή μόνο για ένα κύκλο, κάτι που δε συμβάνει στο IR. Στο Datapath προστέθηκαν 3 registers (**RF_A, RF_B, ALU_OUT_RG**) που κρατάνε επίσης τα αποτελέσματα των δύο καταχωρητών του Register File του DECSTAGE και το αποτέλεσμα της Αριθμητικής/Λογικής μονάδας αντίστοιχα. Κι αυτοί οι καταχωρητές έχουν write enable ίσο με 1 για τον ίδιο λόγο.

Καταργήσαμε το ifstage και το αντικαταστήσαμε με έναν καταχωρητή που κρατάει την τιμή του PC counter. Η τιμή που παίρνει επιλέγεται από έναν νέο πολυπλέκτη που παίρνει τιμές από την ALU. Ο καταχωρητής αυτός έχει enable που ενεργοποιείται μόνο όταν βρισκόμαστε στο FETCH ή έχουμε επιτυχημένο BRANCH.

Στη βαθμίδα εκτέλεσης εντολών ALUSTAGE αποτελείται πλέον από την αριθμητική/λογική μονάδα και 2 πολυπλέκτες. Ο πρώτος επιλέγει τον κατάλληλο πρώτο τελεστή μεταξύ RF_A και PC ενώ ο δεύτερος τον δεύτερο τελεστή μεταξύ RF_B,+4,immed και immed<<2.

B – ΕΞΑΙΡΕΣΕΙΣ

B.1 Επισκόπηση

Γενικώς, η διαδικασία που ακολουθείται όταν έχουμε μια εξαίρεση είναι η εξής:

1. Αποθηκεύουμε την τιμή του PC που προκάλεσε το exception στο EPC register.
2. Μεταβαίνουμε στον exception handler, αναζητούμε το cause και εκτελούμε τα περιεχόμενά του.
3. Εκτελούμε εντολή jump στην διεύθυνση EPC+4 και η ροή του προγράμματος συνεχίζεται κανονικά.

Στην περίπτωση μας μελετάμε 2 εξαιρέσεις:

Όνομα Εξαιρέσης	Αιτιολογία	Διεύθυνση Handler	Τιμή Cause Register
ILLEGAL_INSTR	Άγνωστη εντολή (Άγνωστο Opcode/Func)	0x030	0x00000111
ADDR_RANGE	Λανθασμένη διεύθυνση ανάγνωσης (τελική διεύθυνση λέξης εκτός της ζώνης [0..2K), η αντίστοιχα διεύθυνση byte εκτός της ζώνης [0..8K).	0x040	0x00111000

Για να επιτευχθούν τα παραπάνω, η κάθε εξαίρεση πρέπει να αποτελεί κατάσταση στην FSM. Στη κατάσταση **ILLEGAL_INSTR** μεταβαίνουμε μόνο από τη κατάσταση **DECODE** εφόσον έχουμε άγνωστο OPCODE ή func. Στη κατάσταση **ADDR_RANGE** μεταβαίνουμε μόνο από τη κατάσταση **MEM_READ** εφόσον έχουμε address_range = 1.

👁 B.2 Υλοποίηση

ΑΛΛΑΓΕΣ ΣΤΟ DATAPATH:

- Προστέθηκαν 3 σήματα εισόδου (CU_CAUSEwrite, CU_intCause, CU_EPCwrite) και ένα σήμα εξόδου (address_range). Το πρώτο είναι υπεύθυνο για τη ενεργοποίηση εγγραφής του cause exception στον register. Το δεύτερο είναι σήμα select από το οποίο παίρνουμε την αντίστοιχη τιμή cause register και διεύθυνση handler. Το τελευταίο από τα σήματα εισόδου δίνει enable για να εγγράψουμε στο exception PC. Προφανώς έχουμε address_range = 1 όταν είμαστε εκτός εμβέλειας για τη θέση μνήμης που θέλουμε να διαβάσουμε.
- Συνδέθηκε ο handler register με τον πολυπλέκτη του PC στη θέση D.
- Προστέθηκε νέος πολυπλέκτης έτσι ώστε να επιλέγουμε κάθε φορά τη διεύθυνση του handler ανάλογα με το cause που έχουμε.

ΑΛΛΑΓΕΣ ΣΤΟ DECSTAGE:

- Προστέθηκαν 2 registers (EPC & CAUSE). Ο EPC αναλαμβάνει την τιμή του program counter που προκάλεσε το exception και ο CAUSE διατηρεί την αιτία του exception.
- Προστέθηκε νέος πολυπλέκτης 2 σε 1 με τις τιμές των causes και επιλέγονται απλο το σήμα CU_intCause που περιεγράφηκε παραπάνω. Η τιμή 0 ισοδυναμεί με ILLEGAL_INSTR ενώ η τιμή 1 συνεπάγεται ADDR_RANGE.
- Αντικαταστάθηκε παλιός πολυπλέκτης 2 προς 1 σε νέο 4 εισόδων έτσι ώστε:
 - Με είσοδο 00 γράφει το cause από τον register (S_cause_post),
 - Με είσοδο 01 γράφει από την έξοδο της μνήμης (MEM_out)
 - Με είσοδο 10 έχουμε others => '0' και
 - Με είσοδο 11 γράφει το αποτέλεσμα που παράχθηκε από την ALU (ALU_out_post)

ΑΛΛΑΓΕΣ ΣΤΟ CONTROL UNIT:

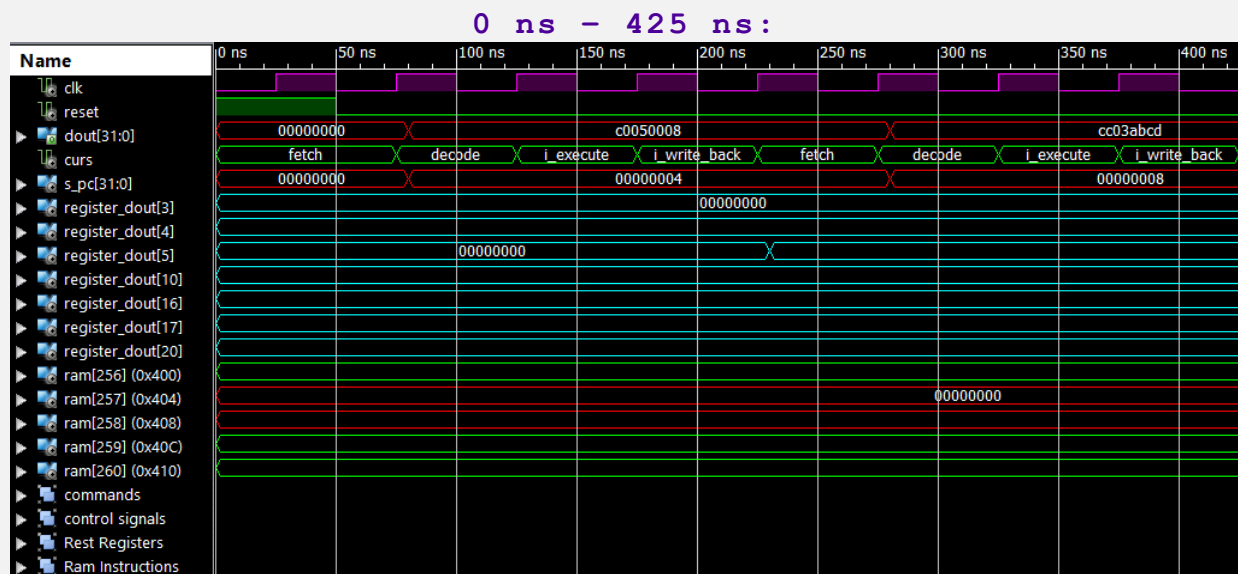
- Προστέθηκαν προφανώς οι 2 καταστάσεις εξαιρέσεων και η μετάβαση σε αυτές αναλύθηκε στο διάγραμμα της μηχανής πεπερασμένων καταστάσεων
- Προστέθηκαν κατάλληλα control signals έτσι ώστε η ALU να παράγει το άθροισμα $PC(ALUSrcA) + 4(ALUSrcB)$, γράφουμε ύστερα την αιτία που σταμάτησε και επιλέγεται η κατάλληλη τιμή του πολυπλέκτη ανάλογα με την κατάλληλη εξαίρεση. Γράφουμε τον EPC και τον PC με είσοδο το αποτέλεσμα της ALU.



Γ – ΕΠΑΛΗΘΕΥΣΗ ΣΥΣΤΗΜΑΤΟΣ

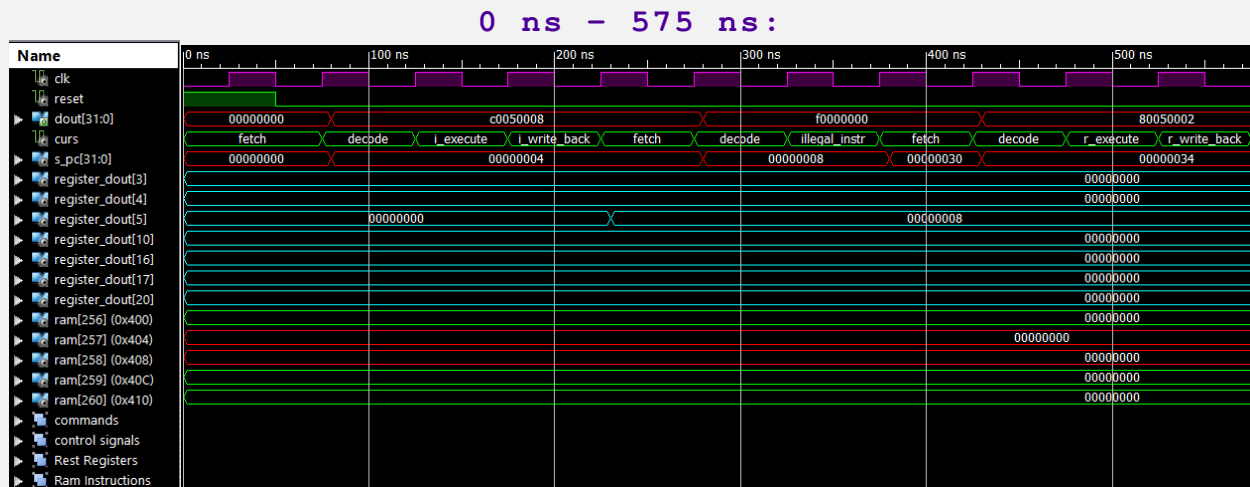
🕒 Γ.1. Προσομοίωση Π.Α #1 (Κυματομορφές)

Παρακάτω παρουσιάζεται μια περίπτωση ορθής λειτουργίας προγράμματος χωρίς εξαιρέσεις που επικεντρώνεται σε απλές πράξεις και αναγνώσεις / εγγραφές στη μνήμη. Παρατηρούμε πως μια εντολή χωρίζεται επιτυχώς σε πολλαπλούς κύκλους, όπου στον κάθε κύκλο υλοποιείται ένα συγκεκριμένο τμήμα της εντολής. Κάθε εντολή από ότι βλέπουμε ξεκινάει με ανάγνωση της από τη μνήμη και στη συνέχεια αποκωδικοποιείται για να μπορέσει να εκτελεστεί.

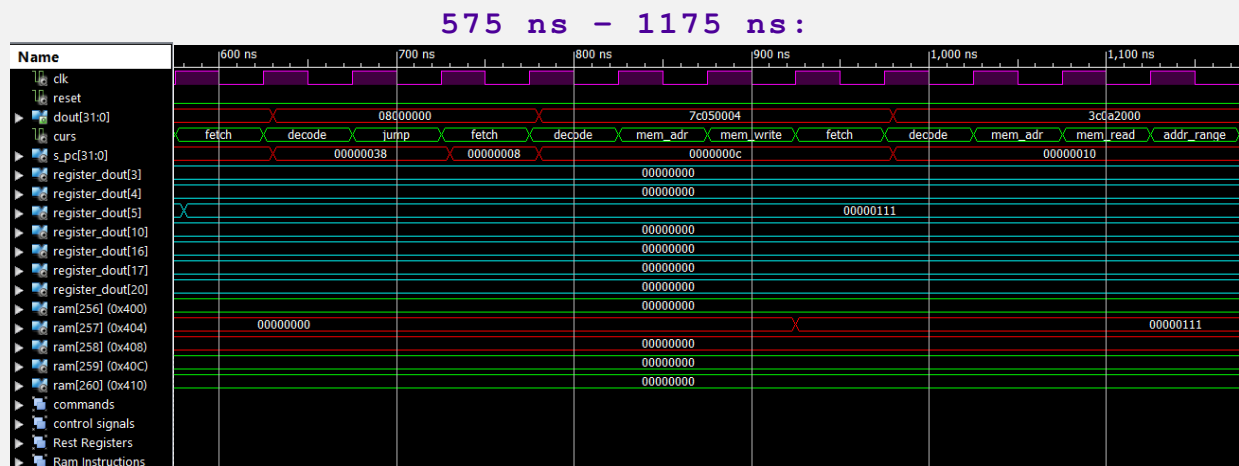


🕒 Γ.3. Προσομοίωση Π.Α #3 (Κυματομορφές)

Το τρίτο πρόγραμμα αναφοράς επιδεικνύει τη λειτουργία εξαιρέσεων. Η πρώτη εξαίρεση που συναντάμε είναι στη δεύτερη εντολή όταν διαβάζουμε μια άγνωστη εντολή. Μετά την κατάσταση αποκωδικοποίησης το πρόγραμμα μεταβαίνει στη κατάσταση ILLEGAL_INSTR και στη συνέχεια κάνει reset στη κατάσταση fetch.

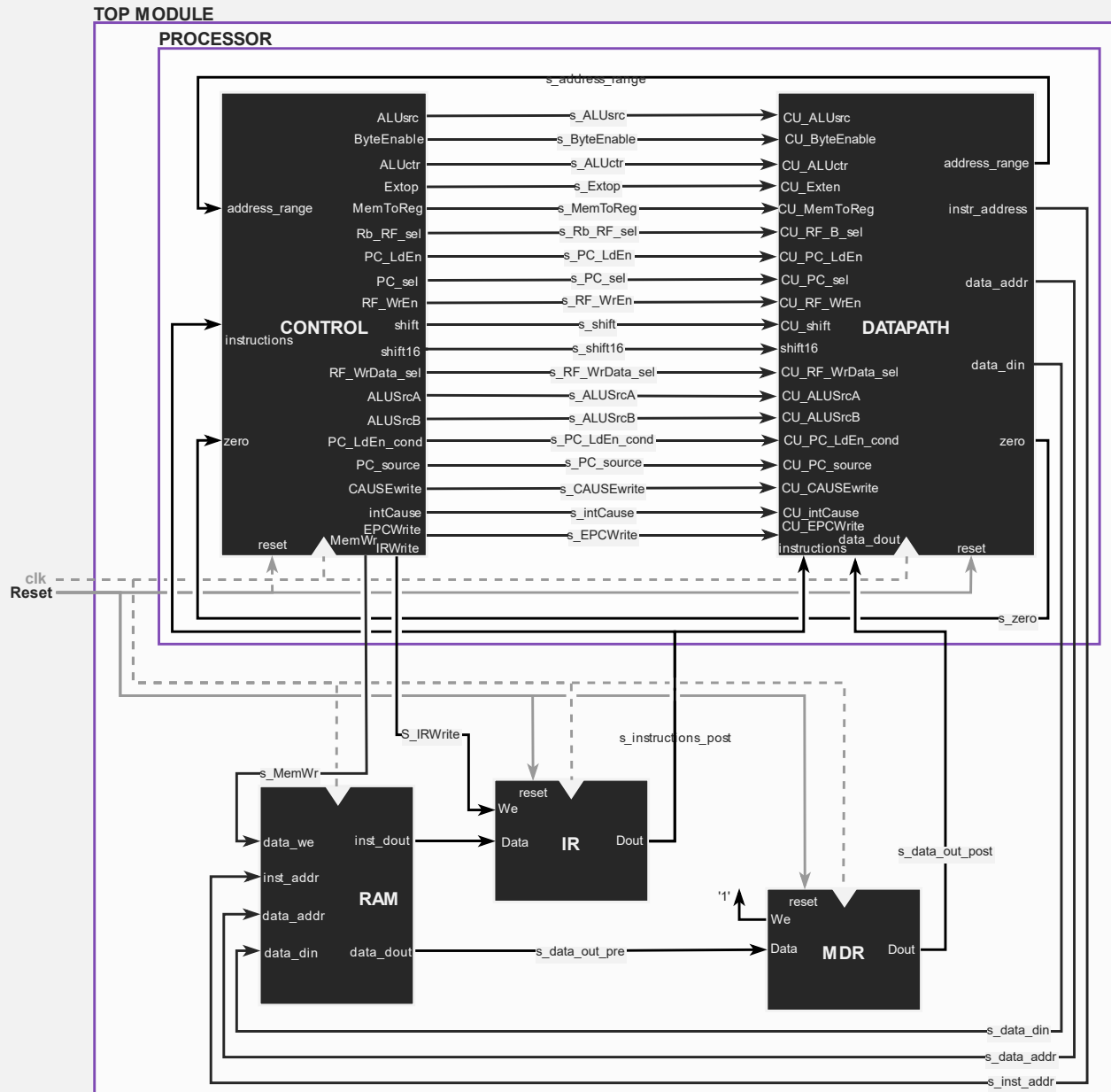


Συναντάμε μια 2^η εξαίρεση στην 6^η εντολή τύπου εμβέλειας μνήμης, όπου αυτή τη φορά το πρόγραμμα μεταβαίνει στην ADDR_RANGE κατάσταση μετά από προσπάθεια ανάγνωσης τμήματος μνήμης σε διεύθυνση που δεν της ανήκει.



Δ – BLOCK DIAGRAMS

CONTROL



DATAPATH

