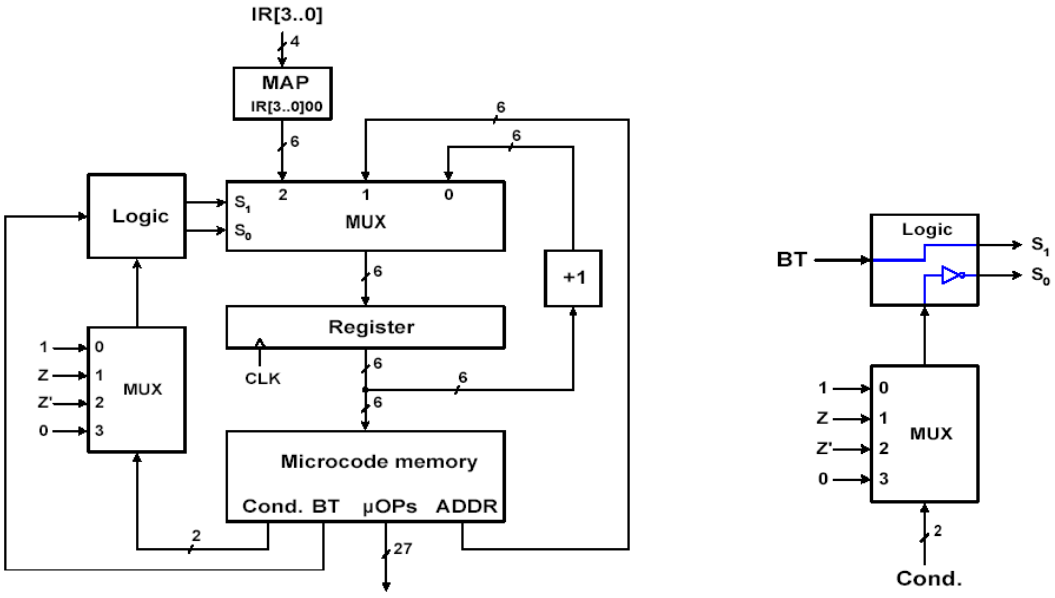
ΜΙΚΡΟΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

3

Σκοπός

Με αφορμή την σχεδίαση και την εξομοίωση με διάφορους τρόπους, απλών ψηφιακών κυκλωμάτων θα κατακτηθεί το αντικείμενο της άσκησης αυτής που είναι η σχεδίαση της μονάδας ελέγχου, χρησιμοποιώντας την μικροπρογραμματιζόμενη (microprogrammed) λογική η οποία θα χρησιμοποιηθεί κατά την τελική σύνθεση της σχετικά απλής ΚΜΕ.

Η μονάδα ελέγχου είναι αυτή που παρέχει στην ΚΜΕ τα απαραίτητα σήματα ελέγχου για τη λειτουργία της. Η λογική σχεδίασής της θα είναι η μικροπρογραμματιζόμενη λογική (microprogrammed logic), η οποία θα υλοποιηθεί με έναν microsequencer (Σχήμα 1).

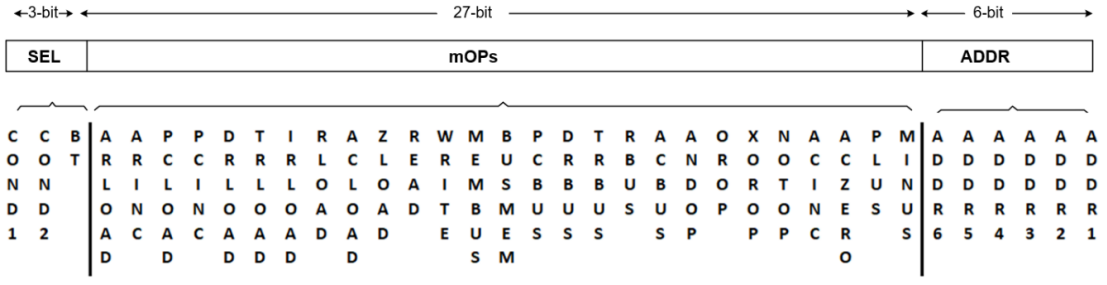


Σχήμα 1: *Λογικό διάγραμμα Μονάδας Ελέγχου και λογική παραγωγής σημάτων ελέγχου S_0,S_1.*

Καταρχάς θα πρέπει να καθοριστεί η δομή μιας μικροεντολής (microinstruction) της ΚΜΕ. Μια μικροεντολή χωρίζεται σε τρία επιμέρους πεδία: SEL, μOPs και ADDR. Το πεδίο των μικροδιεργασιών (micro-operations) ή μOPs είναι εκείνο που περιέχει στην ουσία όλα τα σήματα ελέγχου που παρέχονται σε κάθε παλμό στην ΚΜΕ. Έτσι βάσει του διαγράμματος της εσωτερικής οργάνωσης της ΚΜΕ τα σήματα ελέγχου είναι τα ακόλουθα :

- ✓ Σήματα καταχώρησης — *ARLOAD,PCLOAD,DRLOAD,IRLOAD,TRLOAD,RLOAD,ACLOAD*
- ✓ Σήματα προσαύξησης — *ARINC, PCINC*
- ✓ Σήματα ελέγχου της ALU — *ALUS 1, ALUS 2,…,ALUS 7*
- ✓ Σήματα που αφορούν τη μνήμη — *READ, WRITE*
- ✓ Σήματα ελέγχου των απομονωτών — *PCBUS, DRBUS, TRBUS, RBUS, ACBUS, MEMBUS, BUSMEM*

Τα σήματα αυτά μπορούν να ομαδοποιηθούν σε μία ψηφιολέξη, που θα αποτελεί το πεδίο των μικροδιεργασιών (μOPs) σε κάθε μικροεντολή. Στην πράξη όμως θα ακολουθηθεί μια διαφορετική προσέγγιση, όπου αντί των σημάτων ελέγχου της ALU (ALUS[1..7]) θα χρησιμοποιηθούν στο πεδίο των μικροδιεργασιών τα εξής οκτώ σήματα: ANDOP, OROP, XOROP, NOTOP, ACINC, ACZERO, PLUS και MINUS. Τα σήματα αυτά δεν αποτελούν απευθείας σήματα ελέγχου της ΚΜΕ, αλλά μέσω κάποιας λογικής παράγονται από αυτά τα επιθυμητά (σήματα). Κάθε ένα από αυτά αντιστοιχεί σε περισσότερες από μία μικροδιεργασίες.
Έτσι για παράδειγμα ενεργοποίηση του σήματος ADDOP, σημαίνει ότι λαμβάνει χώρα η διεργασία της πρόσθεσης (ADD) μεταξύ δύο αριθμών των 8-bits, η οποία για να υλοποιηθεί θα πρέπει να γίνουν κάποιες μικροδιεργασίες, μεταξύ των οποίων ενεργοποίηση των σημάτων ALUS 1 & 3 της ALU. Αντίστοιχα, η ενεργοποίηση του σήματος MINUS που αναφέρεται στην πράξη της αφαίρεσης, θα σημαίνει παράλληλα ενεργοποίηση των σημάτων ALUS 1, 2 & 4 μέσω κάποιας λογικής που θα παρεμβάλλεται μεταξύ της μονάδας ελέγχου και της ALU. Η ίδια λογική ισχύει και για τα υπόλοιπα σήματα που αντιστοιχούν σε διεργασίες που αφορούν την ALU και τον συσσωρευτή (AC). Έτσι τελικά το πεδίο των μOPs θα έχει εύρος 27-bits.
Θα πρέπει όμως να σημειωθεί ότι η επιλογή των παραπάνω οκτώ (8) σημάτων αντί των ALUS[1..7], δεν είναι η καλύτερη από πλευράς σχεδίασης, αλλά απλά επιλέχθηκε σαν μια διαφορετική προσέγγιση. Σε πιο πολύπλοκες όμως ΚΜΕ όπου τα σήματα ελέγχου θα είναι πολύ περισσότερα, η επιλογή έμμεσων κατά κάποιο τρόπο σημάτων ελέγχου (όπως τα ANDOP, OROP, ..κτλ), αποτρέπει το φαινόμενο το πεδίο των μικροδιεργασιών (μOPs) να είναι πολύ μεγάλου εύρους, ανάλογο του πλήθους των σημάτων ελέγχου.
Εκτός όμως από το πεδίο των μικροδιεργασιών, κάθε μικροεντολή περιλαμβάνει και τα πεδία SEL και ADDR, τα οποία χρησιμοποιούνται από τον microsequencer για τη λειτουργία του. Το πεδίο ADDR αποτελεί τη διεύθυνση της μνήμης μικροκώδικα της επόμενης μικροεντολής που θα εκτελεστεί. Για την ακρίβεια είναι μία από τις πιθανές διευθύνσεις της επόμενης μικροεντολής. Το τρίτο και τελευταίο πεδίο μιας μικροεντολής είναι το πεδίο SEL εύρους τριών (3) bits, τα οποία χρησιμοποιούνται σαν σήματα επιλογής διεύθυνσης της επόμενης μικροεντολής που θα εκτελεστεί.



Σχήμα 2: *Δομή μιας μικροεντολής (microinstruction).*

Ολοκληρώνοντας την περιγραφή της μονάδας ελέγχου κάθε μικροεντολή έχει εύρος 36-bits, από τα οποία τα 27-bits στο πεδίο των μικροδιεργασιών (μOPs) θα αποτελούν τα σήματα ελέγχου της ΚΜΕ. Βάσει των μικροδιεργασιών σε κάθε κατάσταση, προκύπτει το σύνολο των μικροεντολών για τη σχετικά απλή ΚΜΕ, το οποίο φαίνεται στον παρακάτω πίνακα.

		SEL			μOPs																											
State	ADDRESS	COND 1	COND 2	BT	ARLOAD	ARINC	PCLOAD	PCINC	DRLOAD	TRLOAD	IRLOAD	RLOAD	ACLOAD	ZLOAD	READ	WRITE	MEMBUS	BUSMEM	PCBUS	DRBUS	TRBUS	RBUS	ACBUS	ANDOP	OROP	XOROP	NOTOP	ACINC	ACZERO	PLUS	MINUS	ADDR (dec)
FETCH 1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2
FETCH 2	2	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
FETCH 3	3	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	X
NOP 1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LDAC 1	4	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
LDAC 2	5	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	6
LDAC 3	6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	7
LDAC 4	7	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
LDAC 5	33	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
STAC 1	8	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9
STAC 2	9	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	10
STAC 3	10	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	11
STAC 4	11	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	34
STAC 5	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
MVAC	12	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
MOVR	16	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
JUMP 1	20	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21
JUMP 2	21	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	22
JUMP 3	22	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
JMPZ 1	24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41
JMPZY 1	25	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26
JMPZY 2	26	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	27
JMPZY 3	27	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
JMPZN 1	41	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42
JMPZN 2	42	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
JPNZ 1	28	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
JPNZY 1	29	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
JPNZY 2	30	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	31
JPNZY 3	31	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
JPNZN 1	45	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
JPNZN 2	46	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ADD 1	32	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
SUB 1	36	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
INAC 1	40	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
CLAC 1	44	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
AND 1	48	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
OR 1	52	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1
XOR 1	56	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1
NOT 1	60	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1

Πίνακας 1: *Το σύνολο των μικροεντολών για τη σχετικά απλή CPU*

Για την υλοποίησή της Μονάδας Ελέγχου θα γραφεί κώδικας περιγραφής υλικού σε VHDL για κάθε υποκύκλωμα της ξεχωριστά, όπως φαίνεται από το σχήμα 1 και στη συνέχεια αυτά να χρησιμοποιηθούν σαν στοιχεία (components) για τη σύνθεση του τελικού κώδικα. Σημειώνεται ότι για το καταχωρητή παράλληλης φόρτωσης και για του πολυπλέκτες μπορούν να χρησιμοποιηθούν οι κώδικες περιγραφής που αναπτύχθηκαν στις προηγούμενες ασκήσεις.

Μνήμη μικροκώδικα

Το τμήμα αυτό του microsequencer περιέχει το σύνολο των μικροεντολών βάση των οποίων παράγονται τα επιθυμητά σήματα ελέγχου που τροφοδοτούν την ΚΜΕ. Η υλοποίηση της μνήμης μικροκώδικα σε VHDL, θα γίνει μέσω του δομικού στοιχείου του Quartus , ROM: 1-PORT με τη βοήθεια του Megafunction Wizard.

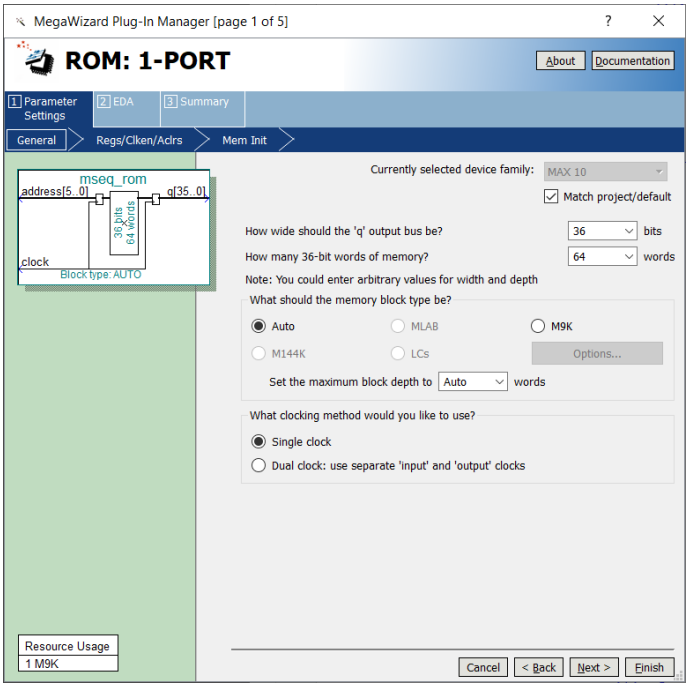
Με τη βοήθεια οποιουδήποτε Editor συντάξτε και αποθηκεύστε το πρόγραμμα 1 που ακολουθεί με όνομα "rs_microcode.mif". Το αρχείο αυτό θα χρησιμοποιηθεί για την αρχικοποίηση της μνήμης μικροκώδικα (μεταβλητή lpm_file) που θα δημιουργήσουμε στο επόμενο βήμα και περιέχει το σύνολο των μικροεντολών, εύρους 36-bits η κάθε μία, σε 39 θέσεις της μνήμης (αντίστοιχες των 39 καταστάσεων) μικροκώδικα.

```
-- Quartus Prime generated Memory Initialization File (.mif)

WIDTH=36;
DEPTH=64;

ADDRESS_RADIX=DEC;
DATA_RADIX=BIN;

CONTENT BEGIN
0 : 110000000000000000000000000000000001 ; -- NOP1
1 : 000100000000000001000000000000000010 ; -- FETCH1
2 : 000000110000010100000000000000000011 ; -- FETCH2
3 : 001100000100000001000000000000000000 ; -- FETCH3
4 : 000001011000001010000000000000000101 ; -- LDAC1
5 : 000000111000010100100000000000000110 ; -- LDAC2
6 : 000100000000000000110000000000000111 ; -- LDAC3
7 : 110000010000010100000000000000100001 ; -- LDAC4
8 : 000001011000001010000000000000001001 ; -- STAC1
9 : 000000111000010100100000000000001010 ; -- STAC2
10 : 000100000000000000110000000000001011 ; -- STAC3
11 : 110000010000000000000100000000100010 ; -- STAC4
12 : 110000000010000000000100000000000001 ; -- MVAC1
16 : 110000000001100000001000000000000001 ; -- MOVR1
20 : 000010010000010100000000000000010101 ; -- JUMP1
21 : 000000011000010100100000000000010110 ; -- JUMP2
22 : 110001000000000000110000000000000001 ; -- JUMP3
24 : 010000000000000000000000000000101001 ; -- JMPZ1
25 : 000010010000010100000000000000011010 ; -- JMPZY1
26 : 000000011000010100100000000000011011 ; -- JMPZY2
27 : 110001000000000000110000000000000001 ; -- JMPZY3
28 : 100000000000000000000000000000101101 ; -- JPNZ1
29 : 000010010000010100000000000000011110 ; -- JPNZY1
30 : 000000011000010100100000000000011111 ; -- JPNZY2
31 : 110001000000000000110000000000000001 ; -- JPNZY3
32 : 110000000001100000001000000010000001 ; -- ADD1
33 : 110000000001100000100000000000000001 ; -- LDAC5
34 : 110000000000001010100000000000000001 ; -- STAC5
36 : 110000000001100000001000000001000001 ; -- SUB1
```

```
40 : 11000000000110000000000000001000000001 ; -- INAC1
41 : 00000010000000000000000000000000101010 ; -- JMPZN1
42 : 11000010000000000000000000000000000001 ; -- JMPZN2
44 : 11000000000110000000000000100000001 ; -- CLAC1
45 : 00000010000000000000000000000000101110 ; -- JPNZN1
46 : 11000010000000000000000000000000000001 ; -- JPNZN2
48 : 11000000000110000000101000000000000001 ; -- AND1
52 : 11000000000110000000100100000000000001 ; -- OR1
56 : 11000000000110000000100010000000000001 ; -- XOR1
60 : 11000000000110000000000000010000000001 ; -- NOT1
END;
```

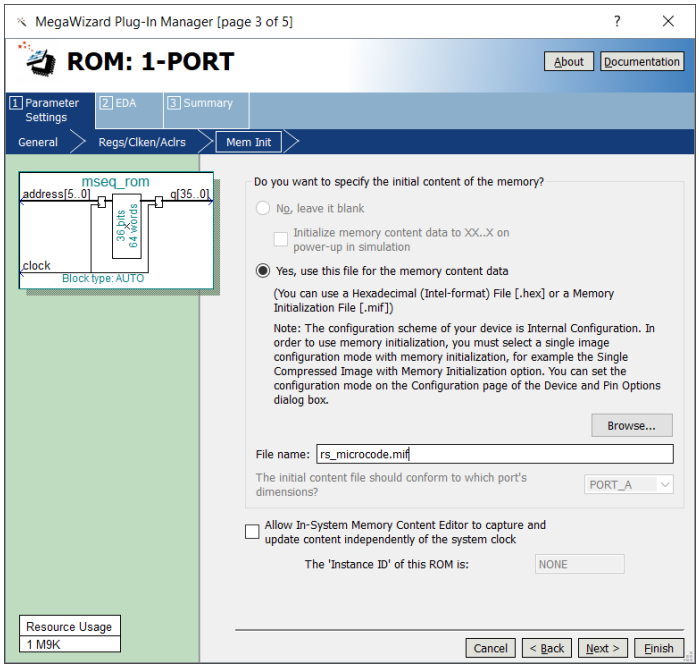
Πρόγραμμα 1: *Περιεχόμενα μνήμης μικροκώδικα*

Στη συνέχεια ¨καλέστε¨ τον MegaWizard Manager μέσω του IP Catalog (πάνω δεξιά περιοχή στο Quartus) αφού κάνετε διπλό click στην επιλογή Library|Basic Functions|On Chip Memory|ROM: 1-PORT και επιλέξετε στο παράθυρο που εμφανίζεται το όνομα (προτείνεται το mseq_rom.vhd), το path και το τύπο του αρχείου(VHDL) ξεκινάει η διαδικασία ορισμού των παραμέτρων της μνήμης:



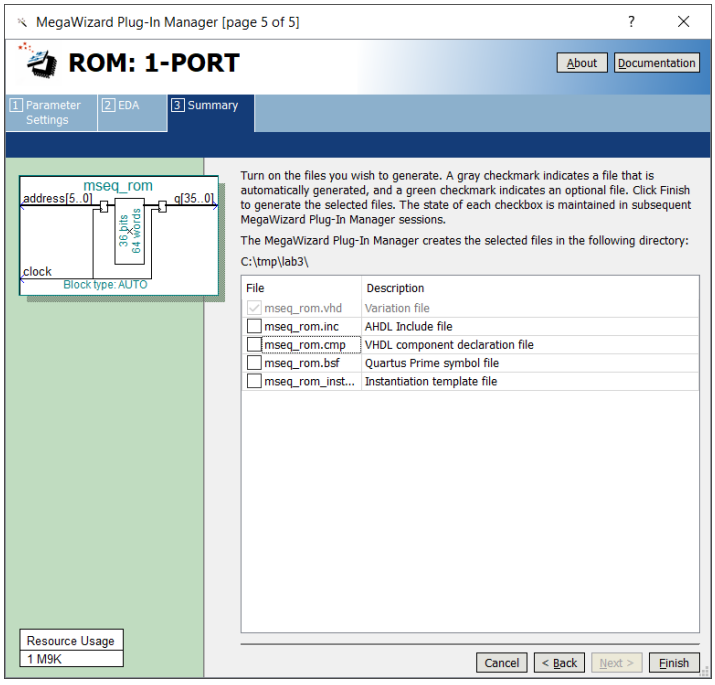
Εικόνα 1

Στο παράθυρο που εμφανίζεται εφαρμόζουμε τις ρυθμίσεις από την Εικόνα 1, πατάμε next και προσπερνάμε το επόμενο παράθυρο (page 2 of 5).



Εικόνα 2

Εφαρμόζουμε τις ρυθμίσεις από την Εικόνα 2, πατάμε next και προσπερνάμε το επόμενο παράθυρο (page 4 of 5).



Εικόνα 3

Τέλος καταλήγουμε στο τελευταίο παράθυρο (Εικόνα 3) στο οποίο επιλέγουμε την δημιουργία του αρχείου VHDL, το οποίο είναι και η default επιλογή, και πατάμε Finish για να ολοκληρωθεί. Κατά το τέλος της διαδικασίας το Quartus μας ρωτάει αν επιθυμούμε να προσθέσουμε το Quartus IP File στο

project. Εδώ επιλέγουμε όχι καθώς θα χρειαστούμε μόνο το αρχείο κώδικα της μνήμης το οποίο και θα προσθέσουμε αργότερα στο project.

Μονάδα Ελέγχου.

Έχοντας ολοκληρώσει τη συγγραφή του κώδικα για τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου, με δεδομένο ότι έχουμε στη διάθεση μας τους κώδικες για τον καταχωρητή n-bits και του πολυπλέκτη 4 σε 1 από προηγούμενες ασκήσεις και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γραφεί το συνολικό πρόγραμμα περιγραφής της μονάδας ελέγχου. Σημειώνεται εδώ ότι δεδομένου ότι κύκλωμα παραγωγής των σημάτων ελέγχου S_1 και S_0 (σχήμα 1) είναι εξαιρετικά απλό δεν είναι απαραίτητη η συγγραφή ξεχωριστού στοιχείου για αυτό.

Γράψτε τον κώδικα για τη βιβλιοθήκη (package), με το όνομα mseqlib, η οποία θα περιέχει τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου.

Γράψτε εδώ το πρόγραμμά σας:

Πρόγραμμα 2: *βιβλιοθήκη στοιχείων για την μονάδα ελέγχου.*

Στο πακέτο θα βάλουμε τους κώδικες από την εργασία 1 και 2, και θα χρησιμοποιήσουμε τα components για να φτιάξουμε την CPU.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

--=========================================================
-- Library package for Lab 3 microprogrammed control unit
-- Contains component declarations for ALU and related units
--=========================================================

package mseqlib is

    -- regnbit Component
    component regnbit is
        generic (n : integer := 8);
        port(
```

```
                -- Inputs
                din       : in std_logic_vector(n-1 downto 0);
                clk,rst,ld  : in std_logic;
                inc       : in std_logic;

                -- Outputs
                dout     : out std_logic_vector(n-1 downto 0)
        );
    end component;

-- 1-bit Full Adder Component
component adder1bit is
  port(
    -- Inputs
    a   : in std_logic;
    b   : in std_logic;
    cin  : in std_logic;

    -- Outputs
    s   : out std_logic;
    cout : out std_logic
  );
end component;

-- 8-bit Full Adder Component
component adder8bit is
  port(
    -- Inputs
    a   : in std_logic_vector(7 downto 0);
```

```
      b    : in std_logic_vector(7 downto 0);
      cin  : in std_logic;

      -- Outputs
      s    : out std_logic_vector(7 downto 0);
      cout : out std_logic
   );
end component;

-- 2-to-1 Multiplexer Component
component mux2 is
   port(
      -- Inputs
      Sig1   : in std_logic_vector(7 downto 0);
      Sig2   : in std_logic_vector(7 downto 0);
      Sel    : in std_logic;

      -- Outputs
      Output : out std_logic_vector(7 downto 0)
   );
end component;

-- 4-to-1 Multiplexer Component
component mux4 is
   port(
      -- Inputs
      Sig1   : in std_logic_vector(7 downto 0);
      Sig2   : in std_logic_vector(7 downto 0);
      Sig3   : in std_logic_vector(7 downto 0);
```

```
        Sig4   : in std_logic_vector(7 downto 0);
        Sel    : in std_logic_vector(1 downto 0);

        -- Outputs
        Output : out std_logic_vector(7 downto 0)
    );
  end component;

  -- ALU Component
  component alu is
    generic (n : integer := 8);
    port (
      ac   : in std_logic_vector(n-1 downto 0);
      db   : in std_logic_vector(n-1 downto 0);
      alus : in std_logic_vector(7 downto 1);
      dout : out std_logic_vector(n-1 downto 0)
    );
  end component;

end package mseqlib;
```

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 3) γράψτε τον κώδικα περιγραφής της μονάδας ελέγχου, δηλαδή του microsequencer, έτσι όπως διαμορφώνεται από τα επιμέρους στοιχεία και το σχήμα 1. Τα σήματα που θα δέχεται σαν είσοδο ο microsequencer εκτός των σημάτων clock και reset, θα είναι τα τέσσερα (4) λιγότερο σημαντικά bit του καταχωρητή εντολών (ir) και η τιμή του καταχωρητή σημαίας (z). Σαν έξοδοι λαμβάνονται τόσο το πεδίο των μικροδιεργασιών (μOPs) που περιέχει τα σήματα ελέγχου της CPU, όσο και το σήμα code που αντιστοιχεί στην κάθε μικροεντολή εύρους 36-bits. Ο λόγος που λαμβάνεται το σήμα code, είναι για να διακρίνεται πιο εύκολα στην εξομοίωση της CPU ποια μικροεντολή εκτελείται σε κάθε παλμό.

Γράψτε εδώ το πρόγραμμά σας:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library lpm;
use lpm.lpm_components.all;
```

```
use work.mseqlib.all;

entity mseq is
port( ir            : in std_logic_vector(3 downto 0);
      clock, reset  : in std_logic ;
      z             : in std_logic ;
      code          : out std_logic_vector(35 downto 0);
      mOPs          : out std_logic_vector(26 downto 0));
end mseq;
architecture arc of mseq is

end arc;
```

Πρόγραμμα 3: *Αριθμητική & Λογική Μονάδα.*

Χρησιμοποιώντας την βιβλιοθήκη *mseqlib* που περιέχει τους κώδικες από τις προηγούμενες εργασίες, δημιουργούμε τον παρακάτω κώδικα:

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

library lpm;

use lpm.lpm_components.all;

use work.mseqlib.all;


entity mseq is
  port(
    ir        : in std_logic_vector(3 downto 0);
    clock, reset : in std_logic;
    z         : in std_logic;
    code       : out std_logic_vector(35 downto 0);
    mOPs       : out std_logic_vector(26 downto 0)
  );
end mseq;


architecture arc of mseq is
```

```
signal microinstruction : std_logic_vector(35 downto 0);
signal current_address  : std_logic_vector(5 downto 0);
signal next_address     : std_logic_vector(5 downto 0);

signal sel_field        : std_logic_vector(2 downto 0);
signal addr_field       : std_logic_vector(5 downto 0);

signal s1, s0           : std_logic;
signal mux_sel          : std_logic_vector(1 downto 0);
signal bt               : std_logic;
signal condition        : std_logic;

signal incremented_addr : std_logic_vector(5 downto 0);
signal mapped_addr      : std_logic_vector(5 downto 0);

-- Extended 8-bit signals for mux4 compatibility
signal mux_in1          : std_logic_vector(7 downto 0);
signal mux_in2          : std_logic_vector(7 downto 0);
signal mux_in3          : std_logic_vector(7 downto 0);
signal mux_in4          : std_logic_vector(7 downto 0);
signal mux_out          : std_logic_vector(7 downto 0);

-- Signals for regnbit inputs/outputs
signal regnbit_ld       : std_logic;
signal regnbit_rst      : std_logic;
signal regnbit_inc      : std_logic := '0';  -- Not used for now
signal regnbit_din      : std_logic_vector(5 downto 0);
signal regnbit_dout     : std_logic_vector(5 downto 0);
```

```
begin

   -- ROM with REGISTERED address (required by MAX 10)
   ROM_inst : lpm_rom
      generic map (
         lpm_width => 36,
         lpm_widthad => 6,
         lpm_address_control => "REGISTERED",
         lpm_outdata => "UNREGISTERED",
         lpm_file => "rs_microcode.mif"
      )
      port map (
         address => regnbit_dout,
         inclock => clock,
         q => microinstruction
      );

   -- Field Extraction from ROM output
   sel_field  <= microinstruction(35 downto 33);
   addr_field <= microinstruction(5 downto 0);

   code <= microinstruction;
   mOPs <= microinstruction(32 downto 6);

   -- MAP Block: Maps IR to starting microcode addresses
   process(ir)
   begin
      case ir is
         when "0000" => mapped_addr <= "000000";  -- NOP -> 0
```

```
        when "0001" => mapped_addr <= "000001";  -- FETCH -> 1
        when "0010" => mapped_addr <= "000100";  -- LDAC -> 4
        when "0011" => mapped_addr <= "001000";  -- STAC -> 8
        when "0100" => mapped_addr <= "001100";  -- MVAC -> 12
        when "0101" => mapped_addr <= "010000";  -- MOVR -> 16
        when "0110" => mapped_addr <= "011000";  -- JMPZ -> 24
        when "0111" => mapped_addr <= "011100";  -- JPNZ -> 28
        when "1000" => mapped_addr <= "100000";  -- ADD -> 32
        when "1001" => mapped_addr <= "100100";  -- SUB -> 36
        when "1010" => mapped_addr <= "101000";  -- INAC -> 40
        when "1011" => mapped_addr <= "101100";  -- CLAC -> 44
        when "1100" => mapped_addr <= "110000";  -- AND -> 48
        when "1101" => mapped_addr <= "110100";  -- OR -> 52
        when "1110" => mapped_addr <= "111000";  -- XOR -> 56
        when "1111" => mapped_addr <= "111100";  -- NOT -> 60
        when others => mapped_addr <= "000001";
    end case;
end process;

-- Incrementer: Standard increment
incremented_addr <= regnbit_dout + 1;

-- Condition Evaluation LOGIC MUX
bt <= sel_field(2);
condition <= z when bt = '0' else not z;

-- S1, S0 Generation
s1 <= sel_field(1) and (sel_field(0) or condition);
s0 <= sel_field(0) and not sel_field(1);
```

```
mux_sel <= s1 & s0;

-- Extend 6-bit addresses to 8-bit for mux4 compatibility
mux_in1 <= "00" & incremented_addr;  -- Sel="00": Sequential
mux_in2 <= "00" & mapped_addr;       -- Sel="01": MAP jump
mux_in3 <= "00" & addr_field;        -- Sel="10": Branch
mux_in4 <= (others => '0');          -- Sel="11": Reset/Unused

-- MUX4 Component Instantiation (MAP ADDRESSES MUX)
MUX_Inst : mux4
   port map (
      Sig1   => mux_in1,
      Sig2   => mux_in2,
      Sig3   => mux_in3,
      Sig4   => mux_in4,
      Sel    => mux_sel,
      Output => mux_out
   );

-- regnbit load and data assignment
regnbit_ld <= '1';
regnbit_rst <= not reset;
regnbit_din <= mux_out(5 downto 0);

-- Instantiate regnbit
regnbit_inst : regnbit
   generic map (n => 6)
   port map (
```

```
        din  => regnbit_din,
        clk  => clock,
        rst  => regnbit_rst,
        ld   => regnbit_ld,
        inc  => regnbit_inc,
        dout => regnbit_dout
    );
end arc;
```

Εξομοίωση της Μονάδας Ελέγχου.

Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της μονάδας ελέγχου με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Με οδηγό τις προηγούμενες ασκήσεις, δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της μονάδας ελέγχου με τη βοήθεια του Waveform Editor για έξι (6) εντολές της ΚΜΕ , της επιλογής σας.

***ΠΡΟΣΟΧΗ**: Μέσω του μενού **Assigments|Device|Device and Pin Options..|Configuration** ρυθμίστε δώστε στο **Configuration mode** την τιμή **Single Uncompressed Image with Memory Initialization (256Kbits UFM)** για να μην έχετε προβλήματα – λάθη κατά το compilation του project.*

Τοποθετήστε εδώ τις κυματομορφές σας:

***Εικόνα 4:** Κυματομορφές εξομοίωσης της μονάδας ελέγχου*

Αρχικά επειδή έχουμε REGISTERED ROM, θα υπάρχει καθυστέρηση ενός κύκλου. Αυτό έχει ως αποτέλεσμα να εμφανίζεται πάντα μία "λανθασμένη" μικροεντολή ανάμεσα στις μεταβάσεις ροής ελέγχου. Για παράδειγμα κατά το άλμα από το στάδιο FETCH στην εκτέλεση της επόμενης εντολής όπως η ADD1. Ο λόγος που γίνεται αυτό είναι επειδή η ROM κλειδώνει (latches) την διεύθυνση της εισόδου σε κάθε ακμοπυροδότηση του ρολογιού με αποτέλεσμα να δημιουργεί pipeline latency για ένα παλμό του ρολογιού.

Όταν τρέξουμε τον κώδικα και δούμε τις κυματομορφές, αυτό που περιμένουμε να δούμε το παρακάτω μοτίβο:

NOP1 → FETCH1 → FETCH2 → FETCH3 → *LDAC1 → Η εντολή που επιλέξαμε → **Η εντολή που επιλέξαμε + 1

* Ο λόγος που θα δούμε τον LDAC1 είναι λόγου του pipeline latency.

** Προτού κάνει την λούπα για να τρέξει την εντολή NOP1, θα δείξει μία "λανθασμένη" εντολή λόγου του pipeline latency. Για παράδειγμα εάν βάζαμε την εντολή ADD1, θα βλέπαμε και την εντολή LDAC5 προτού ξεκινήσει να κάνει την λούπα πάλι.

Για την ευκολότερη κατανόηση των κυματομορφών, μετατρέπουμε τους δυαδικούς αριθμούς των εντολών σε δεκαεξαδικούς. Με αποτέλεσμα, προκύπτει ο παρακάτω πίνακας:

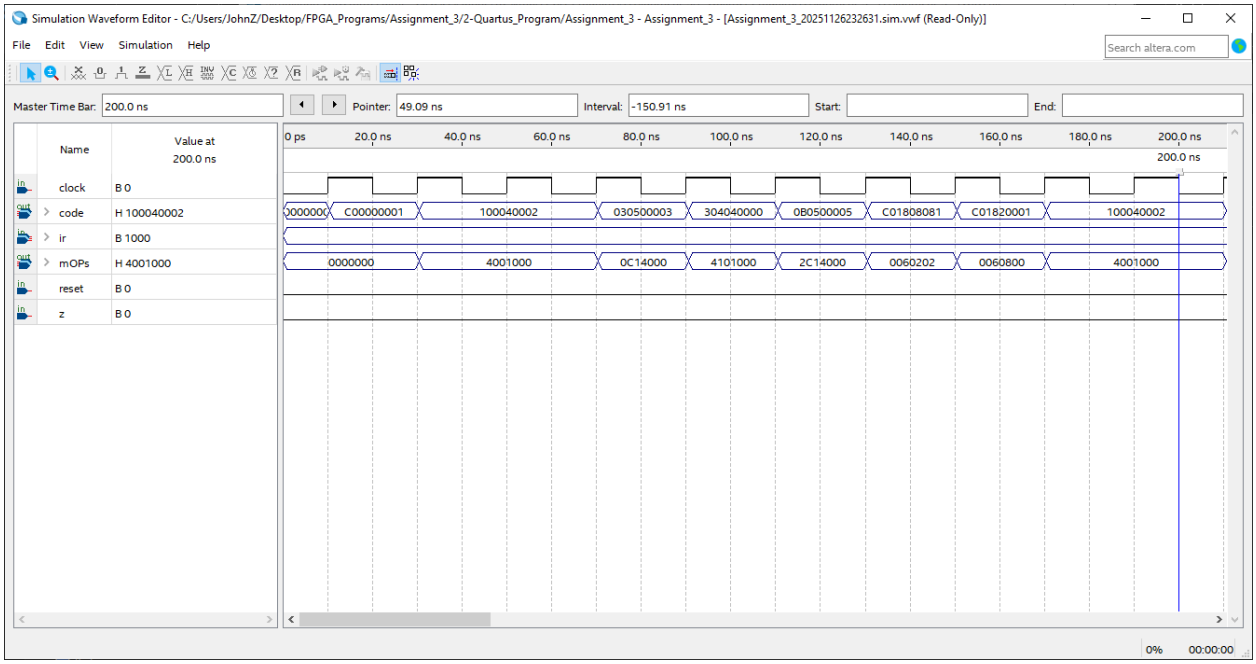
A/A	Operation	Binary (36-bit)	Hexadecimal
0	NOP1	110000000000000000000000000000000001	0xC00000001
1	FETCH1	000100000000000001000000000000000010	0x100040002
2	FETCH2	000000110000010100000000000000000011	0x030140003
3	FETCH3	001100000100000001000000000000000000	0x301040000
4	LDAC1	000010110000010100000000000000000101	0x0B0140005
5	LDAC2	000000111000010100100000000000000110	0x071148006
6	LDAC3	000100000000000000110000000000000111	0x100018007
7	LDAC4	110000010000010100000000000000100001	0xC08140021
8	STAC1	000010110000010100000000000000001001	0x0B0140009
9	STAC2	000000111000010100100000000000001010	0x07114800A
10	STAC3	000100000000000000110000000000001011	0x10001800B
11	STAC4	110000010000000000000100000000100010	0xC08001022
12	MVAC1	110000000010000000000100000000000001	0xC02001001
16	MOVR1	110000000001100000001000000000000001	0xC01802001
20	JUMP1	000010010000010100000000000000010101	0x090140015
21	JUMP2	000000011000010100100000000000010110	0x031148016
22	JUMP3	110001000000000000110000000000000001	0xC40018001
24	JMPZ1	010000000000000000000000000000101001	0x400000029
25	JMPZY1	000010010000010100000000000000011010	0x09014001A
26	JMPZY2	000000011000010100100000000000011011	0x03114801B
27	JMPZY3	110001000000000000110000000000000001	0xC40018001
28	JPNZ1	100000000000000000000000000000101101	0x80000002D
29	JPNZY1	000010010000010100000000000000011110	0x09014001E
30	JPNZY2	000000011000010100100000000000011111	0x03114801F
31	JPNZY3	110001000000000000110000000000000001	0xC40018001
32	ADD1	110000000001100000001000000010000001	0xC01802081
33	LDAC5	110000000001100000100000000000000001	0xC01808001
34	STAC5	110000000000001010100000000000000001	0xC000A8001
36	SUB1	110000000001100000001000000001000001	0xC01802041
40	INAC1	110000000001100000000000001000000001	0xC01800201
41	JMPZN1	000000100000000000000000000000101010	0x02000002A
42	JMPZN2	110000100000000000000000000000000001	0xC10000001
44	CLAC1	110000000001100000000000000100000001	0xC01800101
45	JPNZN1	000000100000000000000000000000101110	0x02000002E
46	JPNZN2	110000100000000000000000000000000001	0xC10000001
48	AND1	110000000001100000001010000000000001	0xC01802801
52	OR1	110000000001100000001001000000000001	0xC01802401
56	XOR1	110000000001100000001000100000000001	0xC01802101

Επειδή ο καταχωρητής IR είναι 4-bit, δεν μπορούμε να καλέσουμε τους 38 συνδυασμούς. Ο πίνακας 1 δείχνει συνολικά όλες τις μικροεντολές τις οποίες με τον παρακάτω πίνακα μπορούμε να τις καλέσουμε:

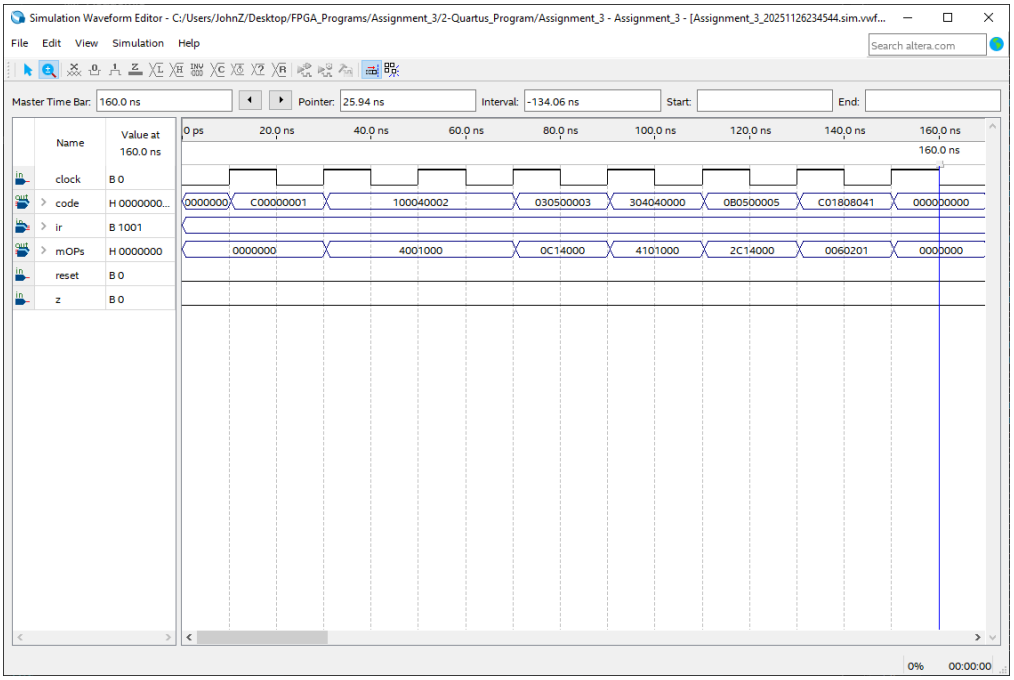
CPU Instruction	Microcode Start Address	First Microinstruction	IR (Binary)	IR (Decimal)
NOP	0	NOP1	0000	0
FETCH	1	FETCH1	0001	1
LDAC	4	LDAC1	0010	2
STAC	8	STAC1	0011	3
MVAC	12	MVAC1	0100	4
MOVR	16	MOVR1	0101	5
JMPZ	24	JMPZ1	0110	6
JPNZ	28	JPNZ1	0111	7
ADD	32	ADD1	1000	8
SUB	36	SUB1	1001	9
INAC	40	INAC1	1010	10
CLAC	44	CLAC1	1011	11
AND	48	AND1	1100	12
OR	52	OR1	1101	13
XOR	56	XOR1	1110	14
NOT	60	NOT1	1111	15

Ακολουθούν οι κυματομορφές

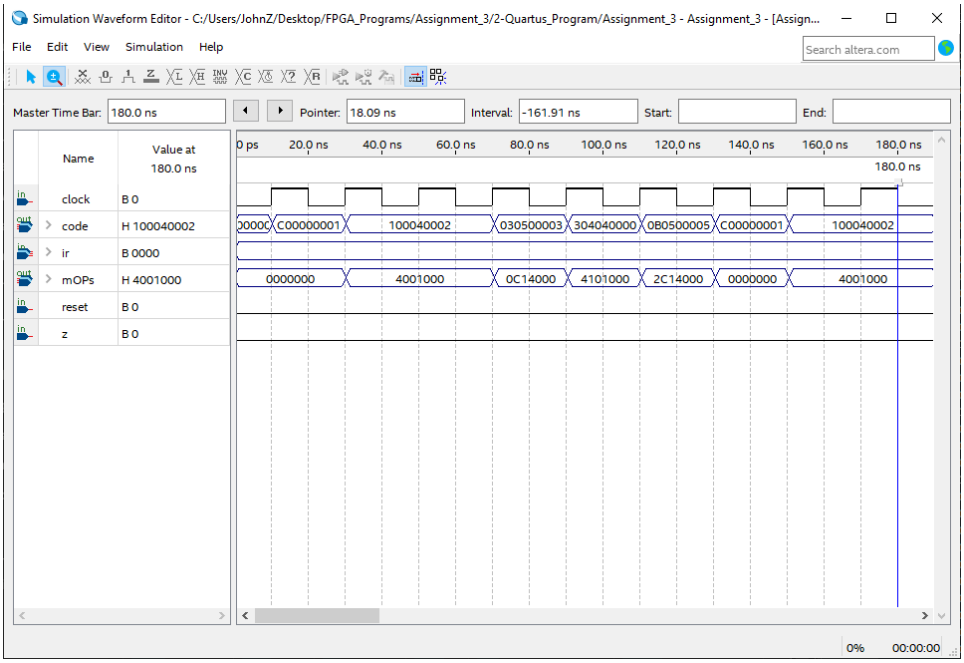
1. Εντολή ADD1:



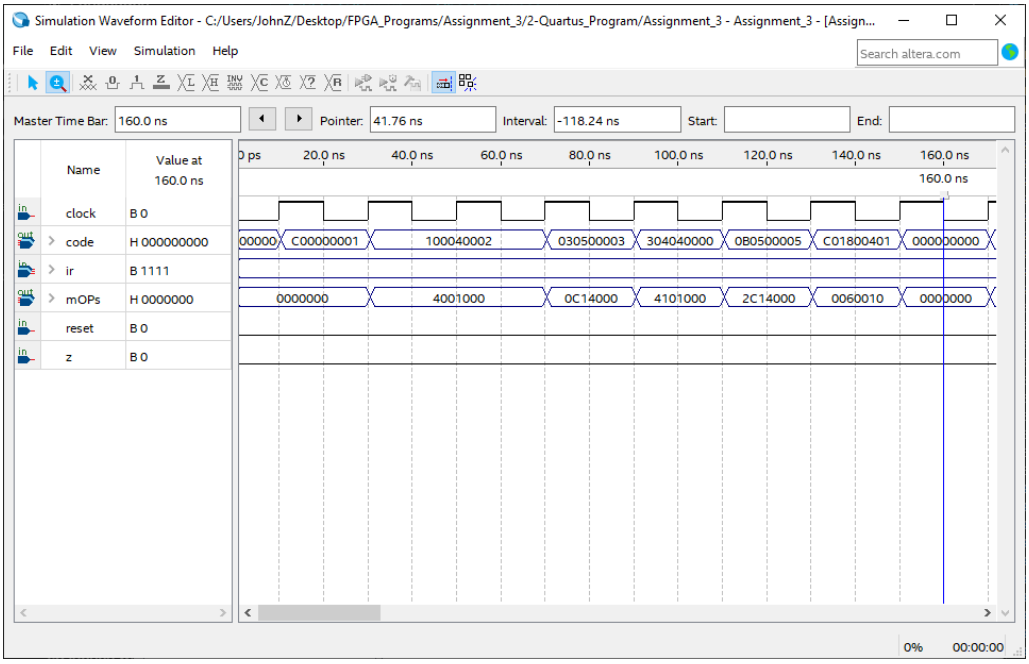
2. Εντολή SUB1:



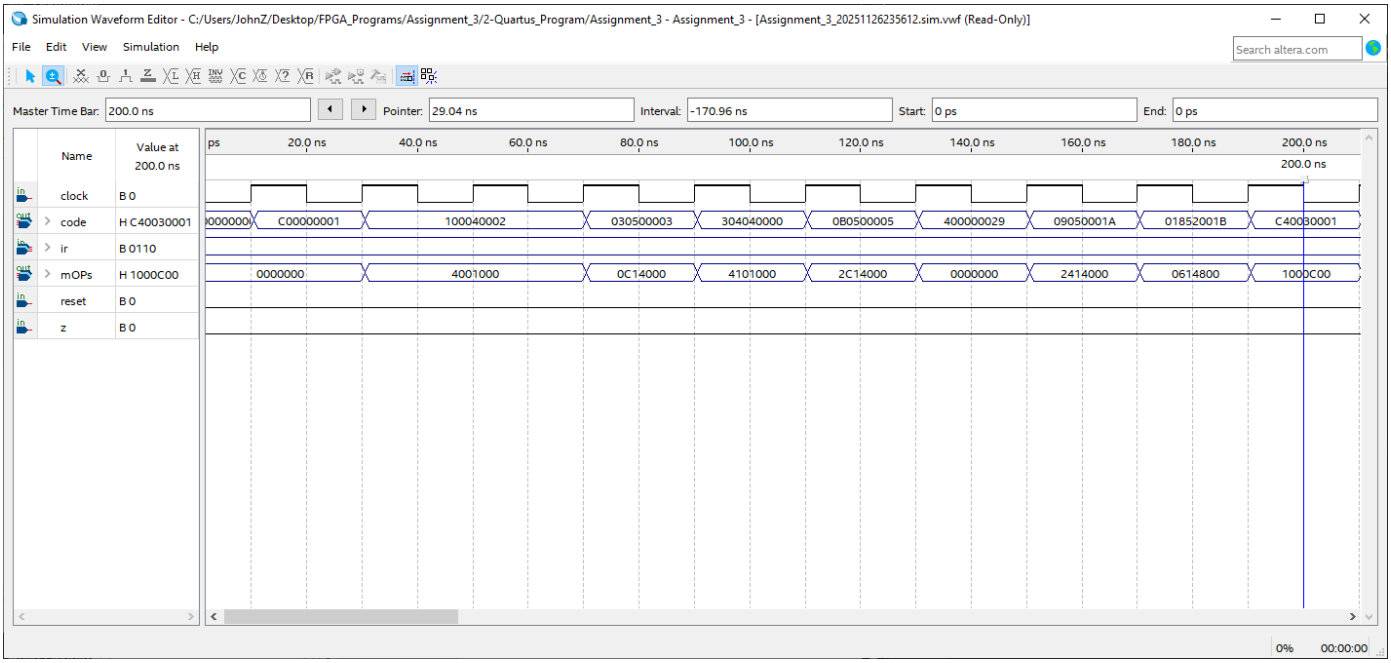
3. Εντολή NOP1:



4. Εντολή NOT1:



5. Εντολή JMPZ1:



6. Εντολή CLAC1: