



Σχολή Θετικών Επιστημών και Τεχνολογίας  
Τμήμα Πληροφορικής

Πτυχιακή Εργασία

Σχεδιασμός και προγραμματισμός σε VHDL  
μιας νέας CPU για εκπαιδευτικούς σκοπούς

Παναγιώτης Στόκας

Επιβλέπων καθηγητής: Βασίλειος Φωτόπουλος

Πάτρα, Ιούλιος 2025

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Σχεδιασμός και προγραμματισμός σε VHDL  
μιας νέας CPU για εκπαιδευτικούς σκοπούς

Παναγιώτης Στόκας

Επιτροπή Επίβλεψης Πτυχιακής Εργασίας

Επιβλέπων Καθηγητής:

Βασίλειος Φωτόπουλος

Σ.Ε.Π., Ελληνικό Ανοικτό Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής:

Ευάγγελος Τοπάλης

Σ.Ε.Π., Ελληνικό Ανοικτό Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής:

Κωνσταντίνος Γιαννακόπουλος

Ε.ΔΙ.Π., Πανεπιστήμιο Πατρών

Θα ήθελα να ευχαριστήσω ιδιαίτερος τον επιβλέποντα καθηγητή μου, Βασίλη Φωτόπουλο, για την καθοδήγηση που μου προσέφερε στην πορεία της εργασίας.

Θερμές ευχαριστίες οφείλω και στον Χαρίδημο Βέργο, για τις εξαιρετικές φροντιστηριακές διαλέξεις που παρέδωσε στο έτος 2022-23 σε όλους τους φοιτητές της ΠΛΗ21 στο ΕΑΠ.

*Η παρούσα εργασία αφιερώνεται στον Άγιο Εφραίμ Νέας Μάκρης.*

## Περίληψη

Στην πτυχιακή αυτή εργασία παρουσιάζεται ο σχεδιασμός, η υλοποίηση και η αξιολόγηση του E80, ενός νέου επεξεργαστή 8-bit. Βασιζόμενος στις αρχές του Πάπερτ για τους μικρόκοσμους μάθησης, ο E80 σχεδιάστηκε εξ αρχής με γνώμονα την απλότητα και την παιδαγωγική αξία, στοχεύοντας να αποτελέσει ένα προσβάσιμο εκπαιδευτικό εργαλείο. Ο καθορισμός των αρχιτεκτονικών απαιτήσεων προέκυψε από τη συγκριτική αξιολόγηση των συνόλων εντολών από εισαγωγικά πανεπιστημιακά μαθήματα Assembly, καθώς και από άλλες εκπαιδευτικές CPU. Το σύνολο εντολών που προέκυψε, αν και λιτό, είναι λειτουργικά πλήρες, υποστηρίζοντας όλες τις απαραίτητες λειτουργίες για τη συγγραφή μη-τετριμμένων αλγορίθμων, συμπεριλαμβανομένης της διαχείρισης στοίβας και των υπορουτινών. Ο κώδικας VHDL σχεδιάστηκε χωρίς έτοιμες αριθμητικές βιβλιοθήκες, ώστε τα εσωτερικά υποσυστήματα του ελέγχου και της διόδου δεδομένων να είναι κατανοητά μέσω στοιχειώδους θεωρίας ψηφιακής σχεδίασης. Το έργο επεκτάθηκε στη δημιουργία ενός αυτόνομου οικοσυστήματος λογισμικού, το οποίο συμπεριλαμβάνει έναν συμβολομεταφραστή γραμμένο σε C και έτοιμα σενάρια για προσομοίωση και υλοποίηση σε υλικό. Η ανάπτυξη του οικοσυστήματος αξιοποίησε αρχές από τον Σχεδιασμό Λογισμικού: η συμβολική γλώσσα του E80 περιγράφηκε με ασυμφραστική γραμματική, η λεκτική και συντακτική ανάλυση καθοδηγήθηκε από τη θεωρία μεταγλωττιστών, ενώ για τη διαχείριση των εκδόσεων του κώδικα και των ρυθμίσεων ολόκληρου του έργου χρησιμοποιήθηκε το σύστημα Git. Η λειτουργικότητα και η στιβαρότητα του συστήματος επαληθεύτηκαν μέσω προσομοίωσης σε πολλαπλά περιβάλλοντα (ModelSim, GHDL) και επικυρώθηκαν με την επιτυχή υλοποίηση σε δύο διαφορετικές πλακέτες FPGA (DSD-il, Tang Primer 25K), αποδεικνύοντας την μεταφερσιμότητα του σχεδιασμού. Ως μελέτη περίπτωσης, αναπτύχθηκε ένας μη-τετριμμένος αλγόριθμος διαίρεσης, καταδεικνύοντας πώς το οικοσύστημα του E80 μπορεί να λειτουργήσει ως ένα γόνιμο περιβάλλον για την ανακάλυψη ισχυρών ιδεών, όπως η αλγοριθμική βελτιστοποίηση και ο δομημένος προγραμματισμός. Τελικά, η εργασία αυτή επαληθεύει την θέση ότι το πρόγραμμα σπουδών Πληροφορικής του Ελληνικού Ανοικτού Πανεπιστημίου μπορεί να καθοδηγήσει την ανάπτυξη ενός υπολογιστή από το μηδέν.

## Λέξεις κλειδιά

Αρχιτεκτονική υπολογιστών, Απλή CPU, VHDL, Συμβολογλώσσα, FPGA, Μικρόκοσμος μάθησης

## Abstract

This undergraduate thesis presents the design, implementation, and evaluation of the E80, a new 8-bit processor. Drawing upon Seymour Papert's principles of learning microworlds, the E80 was designed from the ground up with simplicity and pedagogical value as core tenets, aiming to serve as an accessible educational tool. The architectural requirements were determined through a comparative analysis of the instruction sets from introductory university Assembly courses, as well as from other educational CPUs. The resulting instruction set, though concise, is functionally complete, supporting all necessary operations for writing non-trivial algorithms, including stack management and subroutines. The VHDL code was designed without high-level numeric libraries to ensure that the internal subsystems of the control unit and the datapath are understandable through elementary digital design theory. The project was extended to the creation of a self-contained software ecosystem, which includes an assembler written in C and ready-to-use scripts for simulation and hardware implementation. The ecosystem's development utilized principles from Software Design: the assembly language of E80 was described with a context-free grammar, the lexical and syntactic analysis was guided by compiler theory, while the Git version control system was used to manage the entire project's code and settings. The system's functionality and robustness were verified via simulation across multiple environments (ModelSim, GHDL) and validated by successful implementation on two different FPGA platforms (DSD-i1, Tang Primer 25K), demonstrating the design's portability. As a case study, a non-trivial division algorithm was developed, demonstrating how the E80 ecosystem can serve as a fertile ground for discovering powerful ideas, such as algorithmic optimization and structured programming. Ultimately, this work verifies the thesis that the curriculum of the Hellenic Open University's Computer Science program can guide the development of a computer from scratch.

## Keywords

Computer Architecture, Simple CPU, VHDL, Assembly, FPGA, Learning Microworld

## Πίνακας περιεχομένων

Περίληψη.....	v
Abstract.....	vi
Εικόνες.....	xiii
Πίνακες.....	xv
Συντομογραφίες και ακρωνύμια.....	xvi
1. Εισαγωγή.....	1
1.1. Στόχοι.....	1
1.2. Μεθοδολογία.....	2
2. Υπόβαθρο.....	3
2.1. Στοιχεία ψηφιακής σχεδίασης.....	3
2.1.1. Μανδαλωτής και flip-flop.....	3
2.1.2. Σύγχρονο και ασύγχρονο reset.....	3
2.1.3. Αρχείο καταχωρητών (register file).....	4
2.1.4. Αριθμητική-λογική μονάδα (ALU).....	4
2.2. Στοιχεία αρχιτεκτονικής υπολογιστών.....	5
2.2.1. Αρχιτεκτονική Neumann.....	5
2.2.2. Δίοδος δεδομένων και μονάδα ελέγχου.....	5
2.2.3. Επεξεργαστής ενός κύκλου.....	6
2.2.4. Αρχιτεκτονική συνόλου εντολών.....	6
2.2.5. Διευθυνσιοδότηση.....	7
2.2.6. Αρχιτεκτονική Load/Store.....	7
2.2.7. Μορφές στοίβας.....	8
2.3. VHDL.....	8
2.3.1. Η αφαιρετικότητα της VHDL.....	9
2.3.2. Η δήλωση PROCESS της VHDL.....	10
2.3.3. Πάγκος δοκιμών (testbench).....	11
2.4. Συμβολομεταφραστής.....	12
2.4.1. Backus-Naur Form (BNF).....	12
2.4.2. Λεκτική ανάλυση, συλλογή συμβόλων, και συντακτική ανάλυση.....	12
2.4.3. Παραγωγή κώδικα.....	13
2.5. Εργαλεία προσομοίωσης.....	13
2.5.1. Το ρίσκο της εξάρτησης από εμπορικά εργαλεία.....	13
2.5.2. GHDL και GTKWave: η επιλογή του ελεύθερου λογισμικού.....	14
2.6. FPGA.....	14
2.6.1. Αναπτυξιακή πλακέτα DSD-i1 και Quartus.....	14
2.6.2. Quartus Prime Lite.....	15
2.6.3. Tang Primer 25K και Gowin IDE.....	16
2.7. Οικοδομισμός και εκπαιδευτικοί μικρόκοσμοι.....	16
2.7.1. Αρχές σχεδιασμού μικρόκοσμων.....	17
2.7.2. “Ισχυρές ιδέες” κατά Πάπερτ.....	17

3. Καθορισμός των απαιτήσεων του E80.....	18
3.1. Σύνολο συμβολικών εντολών που χρησιμοποιούνται στην εκπαίδευση.....	18
3.2. Σύνολα εντολών σε άλλες υλοποιήσεις απλών CPU.....	20
3.2.1. Aizup (Li).....	20
3.2.2. Simple CPU (Freeman).....	20
3.2.3. VSCPU (Yıldız).....	21
3.2.4. SC91-A (Pinault).....	22
3.2.5. TINYCPU.....	23
3.3. Συνηθέστερες συμβολικές εντολές σε βιβλιοθήκες Windows & Linux.....	24
3.4. Σύνολο εντολών.....	25
3.4.1. Λειτουργίες που δεν θα υποστηριχθούν.....	26
3.4.2. Λειτουργίες που θα υποστηριχθούν.....	26
3.5. Σημαίες κατάστασης: Ρόλος και Σημασία στη Σχεδίαση του E80.....	28
3.6. Σημαία Αλτ.....	29
3.7. Σύνταξη εντολών.....	30
3.8. Μνήμη, καταχωρητές, μέγεθος λέξης και μορφή συνόλου εντολών.....	32
3.9. Στοίβα.....	33
3.10. Είσοδος δεδομένων με διακόπτες DIP.....	34
4. Η αρχιτεκτονική εντολών του E80.....	35
4.1. Καταχωρητές.....	35
4.2. Σημαίες.....	35
4.3. Κωδικοποίηση εντολών.....	36
4.4. Εντολές μεταφοράς δεδομένων.....	37
4.4.1. MOV - μεταφορά τιμών μεταξύ καταχωρητών.....	37
4.4.2. LOAD - φόρτωση απο την μνήμη.....	37
4.4.3. STORE - αποθήκευση στην μνήμη.....	38
4.5. Αριθμητικές εντολές.....	38
4.5.1. ADD - πρόσθεση.....	38
4.5.2. SUB - αφαίρεση.....	39
4.5.3. CMP - σύγκριση με αφαίρεση.....	39
4.6. Εντολές λογικών πράξεων.....	40
4.6.1. AND - σύζευξη.....	40
4.6.2. BIT - σύγκριση με σύζευξη.....	40
4.6.3. OR - διάζευξη.....	41
4.6.4. XOR - αποκλειστική διάζευξη.....	41
4.7. Εντολές ελέγχου ροής.....	42
4.7.1. JMP - άλμα χωρίς συνθήκη.....	42
4.7.2. JC - άλμα αν υπάρχει κρατούμενο.....	42
4.7.3. JNC - άλμα αν δεν υπάρχει κρατούμενο.....	43
4.7.4. JZ - άλμα αν υπάρχει μηδενικό αποτέλεσμα.....	43
4.7.5. JNZ - άλμα αν δεν υπάρχει μηδενικό αποτέλεσμα.....	43



4.7.6. JS - άλμα αν υπάρχει πρόσημο αρνητικού.....	44
4.7.7. JNS - άλμα αν δεν υπάρχει πρόσημο αρνητικού.....	44
4.7.8. JV - άλμα αν υπάρχει υπερχείλιση.....	45
4.7.9. JNV - άλμα αν δεν υπάρχει υπερχείλιση.....	45
4.7.10. CALL - κλήση υπορουτίνας.....	46
4.7.11. RETURN - επιστροφή απο υπορουτίνα.....	46
4.8. Εντολές στοίβας.....	47
4.8.1. PUSH - ώθηση στην στοίβα.....	47
4.8.2. POP - απώθηση απο την στοίβα.....	47
4.9. Εντολές ολίσθησης.....	48
4.9.1. RSHIFT - δεξιά ολίσθηση.....	48
4.9.2. LSHIFT - αριστερή ολίσθηση.....	48
4.9.3. ROR - δεξιά περιστροφή.....	49
4.10. Ειδικές εντολές.....	49
4.10.1. NOP - καμία λειτουργία.....	49
4.10.2. HLT - παύση εκτέλεσης.....	50
4.11. Φυλλάδιο αρχιτεκτονικής εντολών του E80.....	51
5. Σχεδιασμός και υλοποίηση του E80 σε VHDL.....	52
5.1. Σχεδιαστικές επιλογές και περιορισμοί.....	52
5.1.1. Ανάπτυξη χωρίς αριθμητικές βιβλιοθήκες.....	52
5.1.2. Χρήση της PROCESS αποκλειστικά στην κατασκευή του D flip flop.....	52
5.1.3. Συμβατότητα με το υποσύνολο της VHDL 2008 που υποστηρίζει το Quartus....	52
5.1.4. Άμεση χρήση στιγμιοτύπων αντί για δηλώσεις συστατικών.....	53
5.1.5. Ένας κύκλος ανα εντολή.....	53
5.1.6. Περιορισμός της ενθυλάκωσης υποσυστημάτων.....	53
5.2. Βιβλιοθήκη υποστήριξης.....	54
5.3. Πλήρης αθροιστής.....	55
5.4. Αριθμητική-Λογική Μονάδα (ALU).....	57
5.4.1. Αποκωδικοποίηση (ALUop Decoder).....	57
5.4.2. Προσθαφαίρεση (Full Adder / Subtractor).....	58
5.4.3. Κυκλικός ολισθητής (Barrel shifter).....	58
5.4.4. Υπολογισμός αποτελέσματος και σημαιών.....	59
5.4.5. Έξοδος ή απόρριψη (διατήρηση προηγούμενου) αποτελέσματος.....	59
5.4.6. ALU testbench.....	60
5.5. D flip-flop.....	61
5.6. Αρχείο καταχωρητών.....	61
5.6.1. Αρχικοποίηση σε απροσδιοριστία όταν γίνεται Reset.....	63
5.7. Μνήμη.....	63
5.7.1. Φόρτωμα προγράμματος στην μνήμη μέσω Reset.....	65
5.7.2. Διεύθυνση εισόδου δεδομένων (DIP input).....	65
5.8. Υπολογιστής.....	65

5.8.1. Διαγραμματική αναπαράσταση της επιλογής WHEN.....	66
5.9. CPU.....	67
5.9.1. Αποκωδικοποίηση (Instruction Decoder).....	67
5.9.2. Σήματα ελέγχου.....	68
5.9.3. Έλεγχος ροής και ο μετρητής προγράμματος.....	69
5.9.4. Σενάριο εκτέλεσης της NOP.....	70
5.9.5. Η εκτέλεση της ADD (καταχωρητή-καταχωρητή).....	72
5.9.6. Η εκτέλεση της LOAD με κατευθείαν διεύθυνση.....	74
5.9.7. Η εκτέλεση της STORE με έμμεση διεύθυνση με χρήση καταχωρητή.....	76
5.9.8. Η εκτέλεση της CALL.....	78
5.9.9. Η εκτέλεση της POP.....	80
5.10. Ο σχεδιασμός του D flip flop με σύγχρονο Reset χωρίς είσοδο επίτρεψης.....	82
6. Η συμβολική γλώσσα του E80.....	83
6.1. Μορφή αριθμών.....	83
6.2. Ετικέτες.....	83
6.3. Στοιχίση και σχόλια.....	84
6.4. Οδηγίες.....	84
6.4.1. Όνομα προγράμματος .TITLE.....	84
6.4.2. Δήλωση συμβολικής σταθερής .NAME.....	84
6.4.3. Αρχικοποίηση περιοχών μνήμης .DATA.....	84
6.4.4. Προεπιλεγμένη συχνότητα CPU .FREQUENCY.....	85
6.4.5. Είσοδος DIP στην προσομοίωση .SIMDIP.....	85
6.5. Ενοποίηση συμβολικών σταθερών και ετικετών.....	85
6.6. Διατύπωση με ασυμφραστική γραμματική.....	86
6.7. Φυλλάδιο συμβολικής γλώσσας του E80.....	88
7. Υλοποίηση του συμβολομεταφραστή σε C.....	89
7.1. Μεταγλωττιστής και προγραμματιστικό περιβάλλον.....	89
7.2. Είσοδος/Εξόδος μέσω Stdin/Stdout.....	89
7.3. Δομές δεδομένων.....	90
7.4. Ροή λειτουργίας του Assembler.....	90
7.4.1. Ανάγνωση εισόδου και αποθήκευση γραμμών σε λίστα.....	90
7.4.2. Λεκτική ανάλυση.....	90
7.4.3. Συλλογή συμβόλων.....	91
7.4.4. Συντακτική ανάλυση οδηγιών και εγγραφή .DATA στην μνήμη.....	91
7.4.5. Συντακτική ανάλυση εντολών και παραγωγή κώδικα μηχανής.....	91
7.4.6. Έλεγχοι σφαλμάτων.....	92
7.4.7. Τελική παραγωγή εξόδου σε VHDL.....	93
7.5. Παράδειγμα μετατροπής ιδιωματικού κώδικα.....	93
8. Λειτουργία του E80 σε προσομοίωση.....	95
8.1. ModelSim.....	96
8.1.1. Προετοιμασία του έργου και εκτέλεση σεναρίου c.do.....	96

8.1.2. Το σφάλμα με τα άγκιστρα στο ModelSim.....	99
8.1.3. Αυτόματος τερματισμός προσομοίωσης μέσω Halt flag.....	99
8.2. Ένα σενάριο χρήσης του E80 με το ModelSim.....	100
8.3. GHDL & GTKWave.....	102
8.4. Σενάριο προσομοίωσης του E80 με τα GHDL/GTKWave.....	103
8.5. Απο την Assembly ως τις κυματομορφές με ένα πλήκτρο.....	104
8.6. Aldec Active-HDL.....	106
9. Εκτέλεση του E80 σε FPGA.....	107
9.1. Δοκιμαστικό πρόγραμμα.....	107
9.2. DIP Input.....	108
9.3. LED.....	108
9.4. Γεννήτρια αργού ρολογιού.....	109
9.4.1. Παύση ρολογιού.....	109
9.5. LED ρολογιού.....	110
9.6. Joystick.....	111
9.7. Quartus & DSD-i1.....	113
9.7.1. Ανάθεση ακροδεκτών.....	114
9.7.2. Οδηγίες σύνθεσης και εκτέλεσης.....	117
9.8. Gowin & Tang Primer 25K.....	120
9.8.1. Ανάθεση ακροδεκτών.....	121
9.8.2. Σύνθεση και εκτέλεση.....	124
9.9. Αναφορές συνδυαστικών βρόχων.....	127
9.10. Μετατροπή σε VHDL-93 για συμβατότητα με οποιαδήποτε FPGA.....	127
10. Μια νέα CPU για εκπαιδευτικούς σκοπούς.....	128
10.1. Η ιδέα για ένα αλγόριθμο διαίρεσης.....	128
10.2. Απο την χειρόγραφη ιδέα στον αλγόριθμο.....	128
10.3. Βελτιστοποιημένη υλοποίηση συμβολικού προγράμματος.....	131
10.4. Συνδυασμός με πολλαπλασιασμό a la russe και υπορουτίνες.....	133
10.5. Μεταγλώττιση, προσομοίωση και υλοποίηση.....	134
10.6. Ο εκπαιδευτικός μικρόκοσμος του E80.....	136
11. Συμπεράσματα.....	138
11.1. Περιορισμοί και κριτική.....	138
11.2. Μελλοντικές επεκτάσεις.....	139
Βιβλιογραφία.....	140
Παράρτημα A: Πρωτογενή δεδομένα.....	146
Παράρτημα B: Ο κώδικας του E80 σε VHDL.....	152
B.1 FPGA.vhd - FPGA implementation.....	152
B.2 Support.vhd - support library.....	154
B.3 Computer.vhd - E80 Computer.....	155
B.4 Computer_TB.vhd - E80 Computer test bench.....	156
B.5 RAM.vhd - 256x8 RAM.....	156

B.6 Firmware.vhd - multiplication and division with subroutines.....	157
B.7 CPU.vhd - E80 CPU.....	158
B.8 RegisterFile.vhd - 8x8 register file.....	161
B.9 DFF8.vhd - 8-bit D flip-flop.....	162
B.10 ALU.vhd - arithmetic logic unit.....	162
B.11 ALU_TB.vhd - ALU test bench.....	164
B.12 FA8.vhd - 8-bit ripple carry full adder / subtractor.....	164
Παράρτημα Γ: Κυκλωματικά διαγράμματα RTL απο Quartus.....	166
Παράρτημα Δ: Σενάρια και ρυθμίσεις GHDL/GTKWave.....	171
Δ.1 g.bat - Βασικό αρχείο δεσμίδας GHDL-GTKWave.....	171
Δ.2 alu_tb.gtkw - GTKWave E80 ALU preset.....	172
Δ.3 computer_tb.gtkw - GTKWave E80 Computer preset.....	173
Δ.4 CPU_DSDi1_synthesis.bat - GHDL all-in-one synthesis.....	175
Παράρτημα Ε: Σενάρια και ρυθμίσεις ModelSim.....	176
Ε.1 a.do - σενάριο προσομοίωσης της ALU.....	176
Ε.2 c.do - σενάριο προσομοίωσης του E80 Computer.....	176
Ε.3 LayoutE80.reg - ρυθμίσεις παραθύρων ModelSim.....	177
Ε.4 E80.mpf - αρχείο ρυθμίσεων έργου.....	178
Παράρτημα ΣΤ: Αρχεία ρυθμίσεων Quartus.....	181
ΣΤ.1 E80.qsf - ρυθμίσεις και αναθέσεις ακροδεκτών.....	181
ΣΤ.2 E80.out.sdc - ρυθμίσεις χρονικών περιορισμών.....	183
Παράρτημα Ζ: Αρχεία ρυθμίσεων Gowin.....	184
Ζ.1 Gowin.gprj - FPGA project.....	184
Ζ.2 Gowin_process_config.json - ρυθμίσεις.....	184
Ζ.3 Gowin.sdc - χρονισμός ρολογιού.....	186
Ζ.4 Gowin.cst - ακροδέκτες.....	186
Παράρτημα Η: Κώδικας συμβολομεταφραστή E80ASM σε C.....	188
Η.1 main.c - συντακτική ανάλυση και μετάφραση.....	188
Η.2 error_handler.h - επικεφαλίδα χειρισμού σφαλμάτων.....	193
Η.3 error_handler.c - μηνύματα σφαλμάτων.....	193
Η.4 parse_functions.h - επικεφαλίδα συντακτικών εργαλείων.....	196
Η.5 parse_functions.c - εργαλεία συντακτικής ανάλυσης.....	197
Η.6 data_structures.h - επικεφαλίδα δομών δεδομένων.....	201
Η.7 data_structures.c - λειτουργίες δομών δεδομένων.....	202
Η.8 template.vhd - πρότυπο αρχείο VHDL firmware.....	204
Η.9 E80ASM.dev - ρυθμίσεις Red Panda Cpp & Dev Cpp.....	204
Παράρτημα Θ: Σενάρια αυτοματοποίησης του Notepad++.....	208
Θ.1 Ανάθεση πλήκτρου συντόμευσης μεταγλώττισης & προσομοίωσης.....	208
Θ.2 Χρώματα σύνταξης.....	209
Παράρτημα Ι: Ιστορικό αποθετηρίου github.com/Stokpan/E80.....	210

## Εικόνες

Εικόνα 1: D μανδαλωτής και D flip-flop.....	3
Εικόνα 2: ALU με εισόδους A, B, FlagsIn, ALUop και εξόδους Result και FlagsOut.....	4
Εικόνα 3: Αρχιτεκτονική Neumann.....	5
Εικόνα 4: Επίπεδα αφαιρετικότητας της VHDL.....	9
Εικόνα 5: Αντιστοίχιση της WHEN σε κύκλωμα απο τον RTL Viewer του Quartus.....	10
Εικόνα 6: Ασύγχρονος απαριθμητής με δήλωση PROCESS (ΕΑΠ, DSMC lab).....	11
Εικόνα 7: Αναπτυξιακή πλακέτα DSD-i1 του DSMC lab.....	15
Εικόνα 8: Tang Primer 25K τοποθετημένο στην κατάλληλη dock base board.....	16
Εικόνα 9: Άσκηση προσθαφαίρεσης αριθμών (ΕΑΠ, ΠΛΗ21).....	28
Εικόνα 10: Αντιστοίχιση της μορφής “Εντολή reg, op2” με στοιχειώδη δίοδο δεδομένων.....	30
Εικόνα 11: Πλήρης αθροιστής 8-bit και ένα στιγμιότυπό του (δεξιά).....	55
Εικόνα 12: Κυκλωματικό διάγραμμα 1-bit full adder απο τον RTL viewer του Quartus.....	56
Εικόνα 13: Κυκλωματικό διάγραμμα 8-bit full adder απο τον RTL viewer του Quartus.....	56
Εικόνα 14: Η ALU του E80.....	57
Εικόνα 15: Κυματομορφές προσομοίωσης δοκιμών της ALU.....	60
Εικόνα 16: Εστίαση σε στιγμιαία διαταραχή (hazard) στην αφαίρεση της ALU.....	60
Εικόνα 17: Το 8-bit D flip-flop του E80 και η μορφή του στον RTL viewer του Quartus.....	61
Εικόνα 18: Το αρχείο καταχωρητών (Register File) του E80.....	61
Εικόνα 19: Το υποσύστημα μνήμης του E80.....	63
Εικόνα 20: Υπολογιστής E80.....	66
Εικόνα 21: Μέρος του διαγράμματος του E80 στο Active HDL της Aldec.....	66
Εικόνα 22: Σύμβολο επιλογής WHEN (αριστερά) και η υλοποίησή του με πολυπλέκτες.....	67
Εικόνα 23: Διάγραμμα του ελέγχου ροής προγράμματος.....	70
Εικόνα 24: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της NOP.....	71
Εικόνα 25: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της ADD (register-register).....	73
Εικόνα 26: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της LOAD (register-direct).....	75
Εικόνα 27: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της STORE (register indirect).....	77
Εικόνα 28: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της CALL.....	79
Εικόνα 29: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της POP.....	81
Εικόνα 30: Εκτέλεση του Assembler με ανακατεύθυνση εισόδου/εξόδου.....	89
Εικόνα 31: Μήνυμα σφάλματος σε όρισμα.....	92
Εικόνα 32: Μήνυμα συντακτικού σφάλματος.....	92
Εικόνα 33: Μετατροπή προγράμματος σε κώδικα VHDL απο τον Assembler.....	94
Εικόνα 34: Αρχικοποιήσεις σημάτων και μέτρηση βημάτων εκτέλεσης.....	96
Εικόνα 35: Αρχική εκτέλεση του ModelSim με τις προτεινόμενες ρυθμίσεις.....	96
Εικόνα 36: Σειρά μεταγλώττισης των συστατικών.....	97
Εικόνα 37: Εκτέλεση σεναρίου προσομοίωσης μέσω εντολής “do c.do”.....	97
Εικόνα 38: Περιεχόμενα μνήμης στο τέλος της προσομοίωσης μέσω εντολής “do c.do”.....	98
Εικόνα 39: Περιεχόμενα μνήμης σε σημείο εκτέλεσης μέσω “mouse hover”.....	99
Εικόνα 40: Αποτέλεσμα εκτέλεσης απλού-δοκιμαστικού προγράμματος.....	101

Εικόνα 41: Εκτέλεση δεσμίδας alu_tb.bat για προσομοίωση GHDL/GTKWave.....	102
Εικόνα 42: Προσομοίωση alu_tb.vhd μέσω GHDL/GTKWave.....	103
Εικόνα 43: Εκτύπωση κυματομορφών σε Postscript απο το GTKWave.....	103
Εικόνα 44: Προσομοίωση computer_tb.vhd μέσω GHDL/GTKWave.....	104
Εικόνα 45: Συγγραφή, μετάφραση, προσομοίωση με ένα πλήκτρο.....	105
Εικόνα 46: Προσομοίωση computer_tb.vhd μέσω Aldec Active-HDL.....	106
Εικόνα 47: Περιορισμός της άδειας χρήσης του Active-HDL.....	106
Εικόνα 48: Προσομοίωση προγράμματος 256 ROR στην στιγμή τερματισμού.....	108
Εικόνα 49: Joystick ελέγχου ταχύτητας, ενδείξεων και Reset.....	111
Εικόνα 50: Σύνδεση εξαρτημάτων στην DSD-i1.....	113
Εικόνα 51: Ανάθεση ακροδεκτών μέσω του Pin Planner στο Quartus.....	114
Εικόνα 52: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής R0.....	118
Εικόνα 53: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής SP (R7).....	118
Εικόνα 54: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής FLAGS (R6).....	119
Εικόνα 55: Σύνδεση εξαρτημάτων στην Tang Primer 25K.....	120
Εικόνα 56: Ανάθεση ακροδεκτών μέσω του Floor Planner στο Gowin.....	121
Εικόνα 57: Προγραμματισμός της Tang Primer 25K μέσω USB.....	124
Εικόνα 58: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής R0.....	125
Εικόνα 59: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής SP (R7).....	126
Εικόνα 60: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής FLAGS (R6). ..	126
Εικόνα 61: Το αρχικό χειρόγραφο της ιδέας του αλγορίθμου “ανέβα-κατέβα”.....	129
Εικόνα 62: Εκτέλεση του αλγορίθμου διαίρεσης “ανέβα-κατέβα” στον E80.....	131
Εικόνα 63: Βελτιωμένος αλγόριθμος διαίρεσης “ανέβα-κατέβα”.....	132
Εικόνα 64: Προσομοίωση εκτέλεσης divmul.asm στον E80 (GHDL/GTKWave).....	135
Εικόνα 65: Προσομοίωση εκτέλεσης divmul.asm στον E80 (ModelSim).....	135
Εικόνα 66: Αποτέλεσμα εκτέλεσης divmul.asm στον E80 (Tang Primer 25K).....	136
Εικόνα 67: Διάγραμμα Computer (RTL Viewer / Quartus).....	166
Εικόνα 68: Διάγραμμα RegisterFile (RTL Viewer / Quartus).....	166
Εικόνα 69: Διάγραμμα FPGA (RTL Viewer / Quartus).....	167
Εικόνα 70: Διάγραμμα CPU (RTL Viewer / Quartus).....	168
Εικόνα 71: Διάγραμμα RAM (RTL Viewer / Quartus).....	169
Εικόνα 72: Διάγραμμα ALU (RTL Viewer / Quartus).....	170



## Πίνακες

Πίνακας 1: Συχνότητα συμβολικών εντολών σε εκπαιδευτικό υλικό.....	19
Πίνακας 2: Σύνολο εντολών Aizup (Li).....	20
Πίνακας 3: Σύνολο εντολών Simple CPU (Freeman).....	21
Πίνακας 4: Σύνολο εντολών VSCPU (Yildiz).....	21
Πίνακας 5: Σύνολο εντολών SC91-A (Pinault).....	22
Πίνακας 6: Σύνολο εντολών TINYCPU (Nakano & Ito).....	23
Πίνακας 7: Οι συχνότερες εντολές σε βιβλιοθήκες Windows & Linux.....	24
Πίνακας 8: Το σύνολο εντολών του E80 με ευέλικτο 2ο τελούμενο.....	31
Πίνακας 9: Απαιτούμενα bits για την υποστήριξη των εντολών του E80.....	33
Πίνακας 10: Αναθέσεις σημάτων εισόδου στους ακροδέκτες της DSD-i1.....	115
Πίνακας 11: Αναθέσεις σημάτων εξόδου στους ακροδέκτες της DSD-i1.....	116
Πίνακας 12: Αναθέσεις σημάτων εισόδου στους ακροδέκτες της Tang Primer 25K.....	122
Πίνακας 13: Αναθέσεις σημάτων εξόδου στους ακροδέκτες της Tang Primer 25K.....	123
Πίνακας 14: Συχνότητα εντολών σε παραδείγματα MIPS (ΠΛΗ10, ΕΑΠ).....	146
Πίνακας 15: Συχνότητα εντολών σε ασκήσεις αρχιτεκτονικής (ΠΛΗ21, ΕΑΠ).....	147
Πίνακας 16: Συχνότητα εντολών σε ασκήσεις μικροεπεξεργαστών (ΠΛΗ21, ΕΑΠ).....	148
Πίνακας 17: Συχνότητα εντολών σε ασκήσεις x86-64 (CS61, Harvard).....	149
Πίνακας 18: Συχνότητα εντολών σε ασκήσεις αρχιτεκτονικής (Νικολός, ΤΜΗΥΠ).....	150
Πίνακας 19: Συχνότητα εντολών στον οδηγό εκμάθησης 8086 (Δασυγένης, Π.Δ.Μ.).....	151

## Συντομογραφίες και ακρωνύμια

E80	Οκτάμπιτος επεξεργαστής για εκπαιδευτικούς σκοπούς που έχει σχεδιαστεί στην παρούσα εργασία
ΕΑΠ	Ελληνικό Ανοικτό Πανεπιστήμιο
ΠΛΗ10	Εισαγωγική ενότητα του ΕΑΠ στην Πληροφορική· συμπεριλαμβάνει προγραμματισμό MIPS, δομές δεδομένων και προγραμματισμό σε C.
ΠΛΗ21	Ενότητα ψηφιακών συστημάτων του ΕΑΠ· περιλαμβάνει στοιχειώδη ψηφιακή σχεδίαση, αρχιτεκτονική υπολογιστών, και μικροεπεξεργαστές.
ΠΛΗ24	Ενότητα σχεδιασμού και τεχνολογιών λογισμικού του ΕΑΠ· συμπεριλαμβάνει θεωρία ασυμφραστικών γλωσσών και μεταγλωττιστών.
ΠΛΗΨΙ	1ος κύκλου εργαστηρίου ψηφιακών συστημάτων του ΕΑΠ.
ΠΛΗΨΙΙ	2ος κύκλου εργαστηρίου ψηφιακών συστημάτων του ΕΑΠ.
ΛΣΨ	Λιγότερο Σημαντικό Ψηφίο
ΠΣΨ	Πιο Σημαντικό Ψηφίο
ALU	(Arithmetic-Logic Unit) Αριθμητική-λογική μονάδα
ALUop	(ALU Operation) Επιλογή λειτουργίας της ALU
ASCII	(American Standard Code for Information Interchange) Αντιστοίχιση λατινικών χαρακτήρων και συμβόλων σε αριθμητικές τιμές
BNF	(Backus-Naur Form) Μορφή Backus-Naur για ασυμφραστικές γραμματικές
C, Z, S, V, H	(Carry, Zero, Sign, Overflow, Halt) Σημαίες κατάστασης
CA	(Central Arithmetic) Κεντρικό αριθμητικό τμήμα κατα Neumann
CC	(Central Control) Κεντρικός έλεγχος κατα Neumann
CPU	(Central Processing Unit) Κεντρική μονάδα επεξεργασίας
CST	(Constraint File) Αρχείο περιορισμών για το Gowin IDE
DIP	(Dual In-line Package) Πακέτο διακοπών σε διπλή σειρά ακροδεκτών
DSMC lab	(Digital Systems and Media Computing laboratory) Εργαστήριο ψηφιακών συστημάτων του ΕΑΠ
FIFO	(First-In, First-Out) Πρώτη είσοδος, πρώτη έξοδος
FPGA	(Field-Programmable Gate Array) Συστοιχία επιτοπίως προγραμματιζόμενων πυλών για υλοποίηση ψηφιακών σχεδιάσεων
GHDL	(G Hardware Design Language) Προσομοιωτής VHDL υπο την άδεια GPL
GPL	(General Public License) Άδεια που διασφαλίζει τα δικαιώματα του κοινού ως προς τα παράγωγα ενός έργου
HDL	(Hardware Description Language) Γλώσσα περιγραφής δομής και λειτουργίας κυκλωμάτων



I/O	(Input/Output) Είσοδος/Εξοδος
IDE	(Integrated Development Environment) Ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού
IEEE	Institute of Electrical and Electronics Engineers
ISA	(Instruction Set Architecture) Αρχιτεκτονική συνόλου εντολών
LCD	(Liquid Crystal Display) Οθόνη υγρών κρυστάλλων
LEA	(Load Effective Address) Εντολή φόρτωσης ενεργής διεύθυνσης
LED	(Light Emitting Diode) Δίοδος εκπομπής φωτός
LSB	(Least Significant Bit) Λιγότερο σημαντικό ψηφίο
MASM	(Microsoft Macro Assembler) Συμβολομεταφραστής της Microsoft για αρχιτεκτονικές x86
MinGW	(Minimalist GNU for Windows) Περιβάλλον που προσφέρει μεταγλωττιστές GNU σε Windows
MIPS	(Microprocessor without Interlocked Pipelined Stages) Επεξεργαστής RISC που χρησιμοποιείται ως εκπαιδευτικό πρότυπο
MSB	(Most Significant Bit) Πιο σημαντικό ψηφίο
Op2	Ευέλικτο 2ο τελούμενο σε εντολές ARM assembly
Opcode	(Operation Code) Κωδικός λειτουργίας
PC	(Program Counter) Μετρητής προγράμματος
PWM	(Pulse-Width Modulation) Διαμόρφωση εύρους παλμού
QPF	(Quartus Prime Project File) Αρχείο έργου του Quartus
RAM	(Random-Access Memory) Μνήμη τυχαίας προσπέλασης
RBF	(Raw Binary File) Αρχείο του Quartus για διαμόρφωση FPGA
RISC	(Reduced Instruction Set Computer) Υπολογιστής μειωμένου συνόλου εντολών
RTL	(Register-Transfer Level) Περιγραφή κυκλώματος με βάση τη ροή των δεδομένων μεταξύ των καταχωρητών του
SDC	(Synopsys Design Constraints) Αρχείο χρονικών περιορισμών
SP	(Stack Pointer) Δείκτης στοίβας
stdin/stdout	(Standard Input / Output) Πρότυπη ροή δεδομένων που επιτρέπει την είσοδο/έξοδο δεδομένων απο/προς συσκευές ή άλλα προγράμματα
VHDL	(Very High-Speed Integrated Circuits Hardware Description Language) Γλώσσα HDL που αναπτύχθηκε σε πρόγραμμα του υπουργείου Αμύνης των ΗΠΑ στην δεκαετία του 1980

## 1. Εισαγωγή

Το ενδιαφέρον για απλές CPU σε FPGA καταγράφεται στον Παγκόσμιο Ιστό από τον 20ο αιώνα (Guccione, [1994](#)) και η ενσωμάτωση εικονικών σχεδιασμών στην διδασκαλία στοιχειώδους αρχιτεκτονικής υπολογιστών έχει διερευνηθεί εκτενώς (Calazans & Moraes, [2001](#)) με μετρήσιμες βελτιώσεις στις επιδόσεις των φοιτητών (Salazar & Birrer, [2020](#)).

Οι σχεδιασμοί αυτοί κρίνονται προτιμότεροι από τις εμπορικές υλοποιήσεις σε τέτοιες βαθμίδες (Clements, [1999](#)) και έχουν προταθεί απλές CPU για αυτόν τον σκοπό από τους Pinault ([2006](#)), Nakano & Ito ([2008](#)), Yıldız ([2018](#)), Freeman ([2019](#)) κ.α.

### 1.1. Στόχοι

Σε αυτό το πλαίσιο, σκοπός της παρούσας εργασίας είναι ο σχεδιασμός μιας κεντρικής μονάδας επεξεργασίας (εφεξής “E80”) σε VHDL, η οποία στοχεύει σε τέσσερα σημεία που δεν συναντώνται ταυτόχρονα σε άλλες γνωστές υλοποιήσεις. Οι στόχοι αυτοί είναι οι εξής:

1. **Χρησιμότητα:** Το οικοσύστημα του E80 να μπορεί να χρησιμοποιηθεί για τη διδασκαλία της Assembly, προσφέροντας ένα πλήρες σετ εντολών με απλή σύνταξη που να είναι εύκολη για τους αρχάριους και ταυτόχρονα να επιτρέπει σχετικά προχωρημένο προγραμματισμό Assembly. Η προσομοίωσή του πρέπει να είναι εφικτή με χρήση ελεύθερου ή έστω δωρεάν λογισμικού.
2. **Συνέχεια:** Η σχεδίαση του βασικού Υπολογιστή E80 θα πρέπει να γίνει με αποκλειστική χρήση της ύλης της στοιχειώδους ψηφιακής σχεδίασης, ώστε η μελέτη του κώδικά του να βοηθά στην μετάβαση στην αρχιτεκτονική υπολογιστών σε μια ενότητα όπως η ΠΛΗ21 του ΕΑΠ.
3. **Εφαρμογή:** Για την σχεδίαση της συμβολικής γλώσσας και την ανάπτυξη του συμβολομεταφραστή, θα γίνει χρήση της θεωρίας ασυμφραστικών γλωσσών, συντακτικής ανάλυσης, μεταγλωττιστών, καθώς και των αρχών της τεχνολογίας λογισμικού, συμπεριλαμβανομένης της χρήσης συστήματος ελέγχου εκδόσεων, έτσι ώστε να εφαρμοστεί στον μέγιστο βαθμό το πρόγραμμα σπουδών Πληροφορικής του ΕΑΠ.
4. **Υλοποίηση:** Παράλληλα με την απαιτούμενη λειτουργία στην προβλεπόμενη FPGA του Ψηφιακού Εργαστηρίου του Ελληνικού Ανοικτού Πανεπιστημίου (ΕΑΠ), θα πρέπει να εξασφαλιστεί η υλοποίηση και σε μια επιπλέον FPGA χαμηλού κόστους για επαλήθευση της μεταφερσιμότητας του σχεδιασμού.

Εν κατακλείδι, η παρούσα εργασία θα επιχειρήσει να δείξει ότι το πρόγραμμα Πληροφορικής του ΕΑΠ μπορεί να καθοδηγήσει την σχεδίαση ενός απλού αλλά χρήσιμου επεξεργαστή με συμβολομεταφραστή, που μπορεί να υλοποιηθεί με οικονομικό υλικό και να προσομοιωθεί με ελεύθερο ή έστω δωρεάν λογισμικό. Το οικοσύστημα αυτό θα προταθεί ως ένας Παπερτιανός Μικρόκοσμος για την εξερεύνηση των θεμελιωδών αρχών που διέπουν τη λειτουργία κάθε υπολογιστικής μηχανής, την διδασκαλία της Assembly και το γεφύρωμα της ψηφιακής σχεδίασης με την αρχιτεκτονική υπολογιστών.

## 1.2. Μεθοδολογία

Κατά τα αρχικά στάδια της συγγραφής του κώδικα VHDL, αναδείχθηκε μια σειρά από εγγενείς αντιθέσεις μεταξύ των στόχων που είχαν τεθεί. Για παράδειγμα, η υλοποίηση στοίβας και κλήσης υπορουτίνας (Στόχος 1: Χρησιμότητα) απαιτούσε έναν σχεδιασμό που ερχόταν σε αντίθεση με την απλότητα που επέβαλλε ο Στόχος 2 (Συνέχεια). Ταυτόχρονα, η απαίτηση για συμβατότητα με το Quartus Lite (Στόχος 4) περιόριζε τη δυνατότητα χρήσης της πλήρους προδιαγραφής VHDL 2008, οδηγώντας σε έναν λιγότερο ευανάγνωστο κώδικα που παρεμπόδιζε την επίτευξη του Στόχου 2.

Οι αντιθέσεις αυτές κατέστησαν αναγκαία την επανεξέταση της προσέγγισης, θέτοντας ερωτήματα όπως “ποιό είναι το ελάχιστο σύνολο εντολών που πρέπει να υποστηριχθεί;” ή “θα πρέπει να υπάρχει μεταβλητό πλήθος κύκλων ανα εντολή;”. Για την απάντηση στα ερωτήματα αυτά, ακολουθήθηκε μια πολυφασική προσέγγιση.

Στην πρώτη φάση έγινε ο καθορισμός των απαιτούμενων λειτουργιών. Διεξήχθη βιβλιογραφική επισκόπηση εστιάζοντας στις απαιτήσεις συγγραφής Assembly σε ακαδημαϊκά πλαίσια, σε πρότερες υλοποιήσεις απλών CPU, καθώς και στις συνηθέστερες εντολές και συντακτικές συμβάσεις των αρχιτεκτονικών Intel και ARM. Η έρευνα αυτή οδήγησε στον καθορισμό του τελικού συνόλου εντολών του E80.

Στην δεύτερη φάση έγιναν δοκιμαστικές υλοποιήσεις σε VHDL. Για την εξισορρόπηση των αντικρουόμενων στόχων, αναπτύχθηκαν υλοποιήσεις επιμέρους υποσυστημάτων, οι οποίες συγκρίθηκαν ως προς την πολυπλοκότητα και τη συμβατότητά τους, οδηγώντας στην τελική αρχιτεκτονική. Στο σημείο αυτό ξεκίνησε η χρήση του ελέγχου εκδόσεων στο αποθετήριο [github.com/Stokpan/E80](https://github.com/Stokpan/E80), με την προσθήκη του επιβλέποντα καθηγητή ως παρατηρητή. Το πλήρες ιστορικό του αποθετηρίου παρουσιάζεται στο Παράρτημα I (σ. 210).

Στην τρίτη φάση αναπτύχθηκε το οικοσύστημα του λογισμικού. Αρχικά σχεδιάστηκε η κατάλληλη συμβολική γλώσσα η οποία περιγράφηκε με τη χρήση ασυμφραστικής γραμματικής. Ακολούθως, υλοποιήθηκε ο αντίστοιχος συμβολομεταφραστής σε C, ακολουθώντας τις κλασικές προσεγγίσεις συντακτικής ανάλυσης.

Στην τέταρτη φάση επαληθεύτηκε η ορθότητα της σχεδίασης του E80 μέσω προσομοίωσης σε πολλαπλά περιβάλλοντα, και μέσω της σύνθεσης και της λειτουργίας του σε δύο διαφορετικές πλακέτες FPGA: την προβλεπόμενη DSD-i1 του Ψηφιακού Εργαστηρίου του ΕΑΠ μέσω του Quartus, και την Tang Primer 25K μέσω του Gowin.

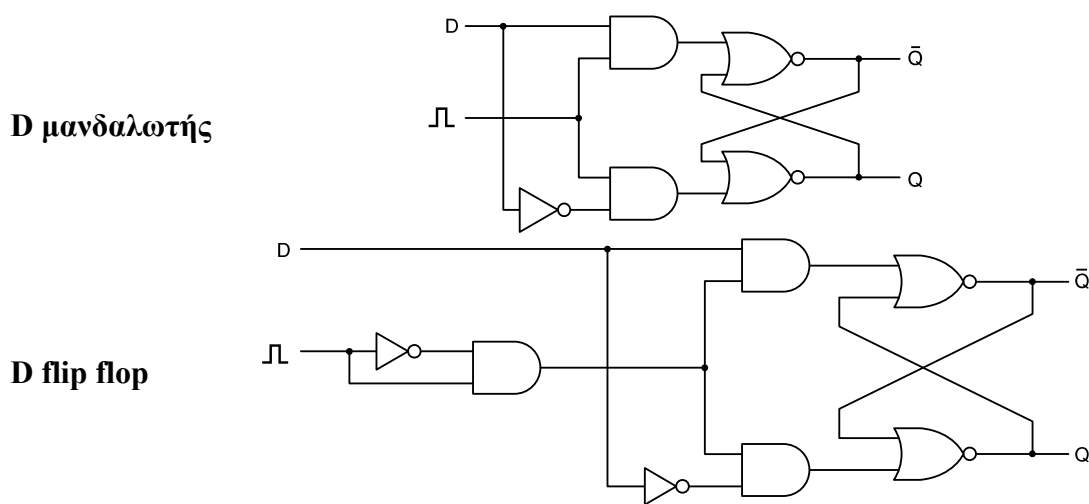
Η εργασία ολοκληρώθηκε με την ανάπτυξη και βελτιστοποίηση ενός μη τετριμμένου αλγορίθμου διαίρεσης. Ο αλγόριθμος αυτός λειτούργησε ως σενάριο χρήσης για την τελική αξιολόγηση του συστήματος, ως προς τις τεχνικές του δυνατότητες, και ως προς τον στόχο της χρήσης του ως Παπερτιανός Μικρόκοσμος.

## 2. Υπόβαθρο

### 2.1. Στοιχεία ψηφιακής σχεδίασης

#### 2.1.1. Μανδαλωτής και flip-flop

Ο μανδαλωτής (latch) είναι ένα στοιχειώδες κύκλωμα που χρησιμοποιείται για την αποθήκευση μιας κατάστασης. Ο μανδαλωτής στην Εικόνα 1, θέτει  $Q = D$  αν το ρολόι είναι σε θετικό παλμό, αλλιώς η  $Q$  διατηρεί την τιμή της. Η δυνατότητα αυτή βασίζεται στην χρονική καθυστέρηση του σήματος και στην ανατροφοδότησή του μεταξύ των πυλών που οδηγούν τις  $Q$  και  $\bar{Q}$  (Vahid, [2011](#), σ. 107).



Εικόνα 1: D μανδαλωτής και D flip-flop <sup>1</sup>

Τα flip flop βασίζονται στην ίδια αρχή με τους μανδαλωτές αλλά είναι ακμοπυροδοτούμενα (Βέργος, [2020](#), σ. 29) με τον Tanenbaum ([2013](#), σ. 172) να τονίζει ότι οι όροι latch και flip-flop συχνά συγχέονται στην βιβλιογραφία. Στο παράδειγμα της Εικόνας 1, το flip-flop συμπεριλαμβάνει τη δημιουργία ενός σύντομου παλμού κατά την ανοδική ακμή του ρολογιού, ο οποίος τροφοδοτεί ένα μανδαλωτή που αποθηκεύει το bit της εισόδου D.

#### 2.1.2. Σύγχρονο και ασύγχρονο reset

Τα flip-flop συνήθως διαθέτουν μια είσοδο reset η οποία μπορεί να είναι συγχρονισμένη ή ασύγχρονη με το ρολόι. Στην πρώτη περίπτωση η αρχικοποίηση της τιμής του Q εφαρμόζεται όταν το σήμα reset είναι ενεργό στην ακμή του ρολογιού. Στην δεύτερη περίπτωση η αρχικοποίηση του Q γίνεται άμεσα όταν ενεργοποιηθεί το reset ανεξάρτητα από το ρολόι.

Οι Cummings & Mills ([2002](#)) αναφέρουν ένα αριθμό πλεονεκτημάτων του σύγχρονου reset σε σχέση με το ασύγχρονο, όπως:

- Το ρολόι λειτουργεί ως φίλτρο σε παράσιτα του reset.
- Το ασύγχρονο reset μπορεί να δημιουργήσει προβλήματα μεταβατικών καταστάσεων.

Οι συγγραφείς καταλήγουν γλαφυρά στο ότι “*there are many reasons given by engineers as to why asynchronous resets are evil*” κάτι το οποίο επιβεβαιώθηκε στην πράξη όπως θα συζητηθεί στην Ενότητα 5.5 (σ. 61).

<sup>1</sup> Tanenbaum ([2013](#), σ. 172-173)

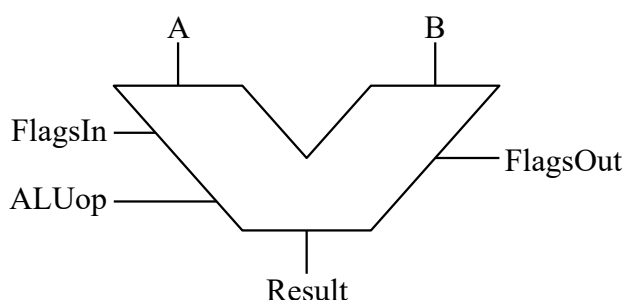
### 2.1.3. Αρχείο καταχωρητών (register file)

Όπως σημειώθηκε προηγουμένως, ένα flip-flop αποθηκεύει ένα bit πληροφορίας στην ακμή του ρολογιού. Μια ομάδα από N flip-flop σχηματίζει έναν N-bit καταχωρητή. Ένα αρχείο καταχωρητών αποτελεί μια οργανωμένη συλλογή καταχωρητών που διευκολύνει την αποθήκευση και πρόσβαση σε προσωρινές μεταβλητές σε ψηφιακά συστήματα (IEEE, [1995](#)). Συνήθως υλοποιείται με τυπική δομή MxN (M καταχωρητές × N bits) και υποστηρίζει ταυτόχρονες λειτουργίες ανάγνωσης/εγγραφής μέσω πολλαπλών θυρών.

Ένα αρχείο καταχωρητών ενσωματώνει γενικούς και ειδικούς καταχωρητές, εξυπηρετώντας ρόλους όπως η αποθήκευση τελουμένων, ενδιάμεσων αποτελεσμάτων, διευθύνσεων μνήμης και πληροφοριών κατάστασης. Για παράδειγμα μια εντολή όπως η “ADD A, B, C” απαιτεί ένα αρχείο καταχωρητών τύπου “2 Read, 1 Write” για την ταυτόχρονη ανάγνωση των B και C και την εγγραφή του αθροίσματός τους στον A σε ένα κύκλο (Vahid, [2011](#), σ. 229).

### 2.1.4. Αριθμητική-λογική μονάδα (ALU)

Σύμφωνα με το IEEE, η ALU ορίζεται ως το λειτουργικό στοιχείο που εκτελεί μαθηματικές πράξεις και λογικές συναρτήσεις. Όπως περιγράφουν οι Harris & Harris ([2022](#)), μια τυπική ALU ενσωματώνει πολλαπλές λειτουργίες όπως πρόσθεση, αφαίρεση, ολίσθηση και λογικές πράξεις σε ένα ενιαίο μπλοκ, με βάση ένα διάνυσμα ελέγχου. Στους Hennessy & Patterson ([2012](#)) το διάνυσμα αυτό ονομάζεται ALUop.



Εικόνα 2: ALU με εισόδους A, B, FlagsIn, ALUop και εξόδους Result και FlagsOut

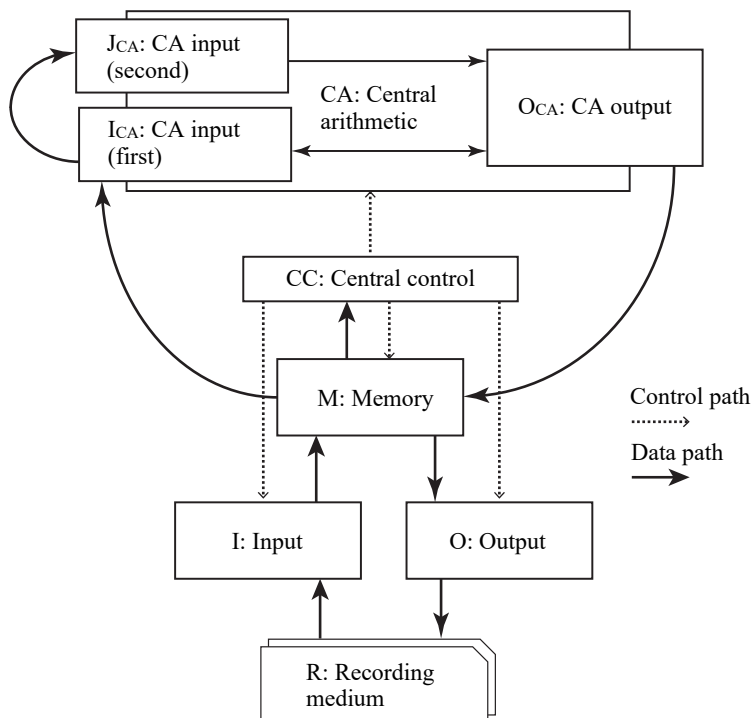
Η δομή μιας ALU βασίζεται συνήθως σε έναν αθροιστή/αφαιρέτη, συνδυαστικές λογικές πύλες και πολυπλέκτες για την επιλογή της λειτουργίας. Σημαντικό χαρακτηριστικό είναι η δυνατότητα παραγωγής σημαίων κατάστασης, όπως το κρατούμενο, το μηδέν, το πρόσημο και η υπερχειλίση, βάσει του αποτελέσματος της πράξης. Οι σημαίες αυτές είναι απαραίτητες για τις αποφάσεις που αφορούν στην ροή της εκτέλεσης.

Στην Εικόνα 2 παρουσιάζεται το τυπικό σύμβολο της ALU με εισόδους τα τελούμενα διανύσματα A και B, τις σημαίες FlagsIn και το διάνυσμα ελέγχου ALUop. Βάσει του ALUop η ALU εκτελεί μια πράξη στα A και B και εξάγει το αποτέλεσμα στο Result και τις σημαίες εξόδου στο διάνυσμα FlagsOut. Σε λειτουργίες όπως η σύγκριση, η ALU αγνοεί το αποτέλεσμα της πράξης και εξάγει το αρχικό διάνυσμα A μεταβάλλοντας μόνο τις σημαίες.

## 2.2. Στοιχεία αρχιτεκτονικής υπολογιστών

### 2.2.1. Αρχιτεκτονική Neumann

Ο Neumann (1945, σ. 3) όρισε το αυτόματο ψηφιακό υπολογιστικό σύστημα πολύ υψηλής ταχύτητας σύμφωνα με το ανθρώπινο νευρικό σύστημα. Το κεντρικό αριθμητικό τμήμα (CA) με τον κεντρικό έλεγχο (CC) συνιστούν το τμήμα C ενώ η μνήμη το τμήμα M. Τα μέρη αυτά τα αντιστοίχισε με τους συνειρμικούς νευρώνες του ανθρώπινου νευρικού συστήματος, ενώ στους αισθητικούς και κινητικούς νευρώνες αντιστοίχισε τα όργανα εισόδου (I) και εξόδου (O) της μηχανής συνδέοντάς τα σύμφωνα με την Εικόνα 3.



Εικόνα 3: Αρχιτεκτονική Neumann<sup>1</sup>

Χαρακτηριστικό της αρχιτεκτονικής Neumann είναι ότι το πρόγραμμα και τα δεδομένα προσπελούνται στον ενιαίο χώρο μνήμης M.

### 2.2.2. Δίοδος δεδομένων και μονάδα ελέγχου

Σε ότι αφορά στις εντολές ενός προγράμματος που εκτελείται στην αρχιτεκτονική Neumann, σύμφωνα με τον Hayes (1998, σ. 21), το τμήμα που τις λαμβάνει από την μνήμη και τις μεταφράζει σε σήματα ελέγχου λέγεται μονάδα ελέγχου (control unit), ενώ το τμήμα που αναλαμβάνει την εκτέλεση τους λέγεται δίοδος δεδομένων (datapath). Βασικά συστατικά της μονάδας ελέγχου είναι ο αποκωδικοποιητής εντολών και ο μετρητής προγράμματος, ενώ η δίοδος δεδομένων περιέχει την ALU και τους γενικούς καταχωρητές.

Οι Harris & Harris (2022, σ. 457-461) παρουσιάζουν τις δύο αυτές δομές ως διακριτά συστατικά, ενώ οι Patterson & Hennessy (2013, σ. 17) προσδιορίζουν ότι μερικές φορές υφίστανται ως ένα ενιαίο τμήμα που συνιστά τον επεξεργαστή.

<sup>1</sup> Το διάγραμμα, κατά τους Haigh, Priestley & Rope (2016, σ. 145) παρουσιάζει την πλήρη αρχιτεκτονική Neumann σύμφωνα με τις ακριβείς ονομασίες του αρχικού σχεδίου που έγραψε το 1945.



### 2.2.3. Επεξεργαστής ενός κύκλου

Σε ένα επεξεργαστή ενός κύκλου, η διαδικασία λήψης, αποκωδικοποίησης και εκτέλεσης λαμβάνει χώρα σε ένα κύκλο του ρολογιού. Αυτό με πρώτη ματιά δίνει την εντύπωση ότι ένας τέτοιος επεξεργαστής είναι γρηγορότερος ή οικονομικότερος από ένα επεξεργαστή πολλών κύκλων αλλά στην πραγματικότητα ισχύει το αντίστροφο.

- Στον επεξεργαστή ενός κύκλου, ο κύκλος (άρα και η κάθε εντολή) θα πρέπει να έχει την διάρκεια που απαιτεί η πιο αργή εντολή (Patterson & Hennessy, [2013](#), σ. 274) το οποίο έχει μετρήσιμα πλεονεκτήματα για τους επεξεργαστές πολλών & σύντομων κύκλων και σε υλοποιήσεις σε FPGA με VHDL (Solanki & Sharma, [2021](#)).
- Ο επεξεργαστής πολλών κύκλων μπορεί να χρησιμοποιήσει τα ίδια υποσυστήματα για διαφορετικές λειτουργίες στην εκτέλεση μιας εντολής. Για παράδειγμα η προσαύξηση του μετρητή προγράμματος μπορεί να γίνει μέσω της ALU η οποία προηγουμένως χρησιμοποιήθηκε για την πρόσθεση καταχωρητών. Αυτό μειώνει το κόστος του σε σχέση με τον επεξεργαστή ενός κύκλου που χρειάζεται ένα ξεχωριστό υποσύστημα για την κάθε λειτουργία που θα πραγματοποιήσει στον ένα κύκλο.

Πλεονέκτημα της προσέγγισης του ενός κύκλου είναι η απλότητα, με τον Tanenbaum ([2013](#), σ. 334-335) να δίνει ως παράδειγμα τον ATmega168, ένα πραγματικό επεξεργαστή που εκτελεί τις περισσότερες εντολές του σε ένα κύκλο (Microchip Technology, [2016](#)). Στο πλαίσιο της εργασίας αυτής, η εκτέλεση όλων των εντολών σε ένα κύκλο επιτρέπει την ευκολότερη μελέτη της προσομοίωσης, καθώς ο κάθε κύκλος ρολογιού ισοδυναμεί με την εκτέλεση μιας εντολής, επιτρέποντας την εύκολη μέτρηση των βημάτων του προγράμματος.

### 2.2.4. Αρχιτεκτονική συνόλου εντολών

Ο Pesler ([1982](#)) ορίζει την αρχιτεκτονική συνόλου εντολών (Instruction Set Architecture - ISA) ως “τα χαρακτηριστικά του επεξεργαστή όπως τα βλέπει ένας προγραμματιστής *Assembly*”. Ο ορισμός αυτός δεν είναι το ίδιο με τον όρο “γλώσσα *Assembly*” καθώς η ISA εκτός από σύνολο εντολών περιλαμβάνει τον τρόπο κωδικοποίησής τους, τα είδη των τελεματών, τους τρόπους διευθυνσιοδότησης, την διαχείριση της μνήμης, τους καταχωρητές και γενικά οτιδήποτε ορίζει την διασύνδεση μεταξύ των μεταγλωττιστών και του υλικού (Tanenbaum, [2013](#)). Ταυτόχρονα η γλώσσα *Assembly* μπορεί να περιέχει οδηγίες και κανόνες σύνταξης που δεν έχουν σχέση με την αρχιτεκτονική του συνόλου εντολών.

Ο Tanenbaum ορίζει δυο παράγοντες που κάνουν μια “καλή” ISA: αφενός να προσφέρει ένα σύνολο εντολών που μπορεί να υλοποιηθεί αποτελεσματικά και οικονομικά σε τρέχουσες και μελλοντικές τεχνολογίες, και αφετέρου να επιτρέπει την εύκολη υλοποίηση καθαρού και ποιοτικού κώδικα.

### 2.2.5. Διευθυνσιοδότηση

Στη βιβλιογραφία δεν υφίσταται ενιαία ονοματολογία για τις μεθόδους διευθυνσιοδότησης. Συγγραφείς και κατασκευαστές χρησιμοποιούν συχνά διαφορετικούς όρους για την περιγραφή της ίδιας μεθόδου ή αποδίδουν το ίδιο όνομα σε διαφορετικές μεθόδους διευθυνσιοδότησης.

Στην εργασία αυτή θα χρησιμοποιηθούν τέσσερις τρόποι διευθυνσιοδότησης με τις ονομασίες που καθορίζονται από τους Βέργο (2022 σ. 133) και Tanenbaum (2013 σ. 372-373) λόγω της ομοιότητας που παρουσιάζουν.

Έστω ότι σε μια εντολή χρησιμοποιούμε ένα τελούμενο  $X$  με το οποίο θα προσπελάσουμε μια διεύθυνση. Έχουμε τους εξής τρόπους διευθυνσιοδότησης:

- Άμεση: το  $X$  είναι το δεδομένο, μια σταθερά.
- Κατευθείαν σε καταχωρητή: το  $X$  είναι το όνομα (ή η διεύθυνση) ενός καταχωρητή που προσπελάνεται.
- Κατευθείαν στην μνήμη: το  $X$  είναι μια διεύθυνση μνήμης που προσπελάνεται.
- Έμμεση με καταχωρητή: το  $X$  είναι διεύθυνση ενός καταχωρητή που περιέχει διεύθυνση μνήμης η οποία προσπελάνεται.

Θα δοθεί ένα παράδειγμα για την κάθε μορφή διευθυνσιοδότησης.

- `MOV R0, 10` (ψευδοκώδικας:  $R0 \leftarrow 10$ )  
Υπάρχει κατευθείαν διευθυνσιοδότηση για τον  $R0$  ο οποίος προσπελάνεται για αλλάξει το περιεχόμενό του. Στην σταθερά 10 υπάρχει άμεση διευθυνσιοδότηση.
- `LOAD R0, 100` (ψευδοκώδικας:  $R0 \leftarrow [100]$ )  
Υπάρχει κατευθείαν διευθυνσιοδότηση για τον  $R0$  όπως στο προηγούμενο παράδειγμα. Υπάρχει κατευθείαν διευθυνσιοδότηση με το 100 να εκφράζει θέση μνήμης που περιέχει το δεδομένο που θα φορτωθεί στον  $R0$ .
- `STORE R2, R1` (ψευδοκώδικας:  $R2 \rightarrow [R1]$ )  
Υπάρχει κατευθείαν διευθυνσιοδότηση για τον  $R2$  η τιμή του οποίου αντλείται. Υπάρχει έμμεση διευθυνσιοδότηση μέσω του  $R1$  ο οποίος περιέχει μια διεύθυνση μνήμης η οποία προσπελάνεται ώστε να της αποθηκευτεί η τιμή του  $R2$ .

### 2.2.6. Αρχιτεκτονική Load/Store

Η αρχιτεκτονική Load/Store (Φόρτωσης/Αποθήκευσης) ορίζει ότι η διευθυνσιοδότηση στην μνήμη, είτε κατευθείαν είτε έμμεσα από καταχωρητή, περιορίζεται αποκλειστικά σε εντολές `LOAD` και `STORE` όπως στο παράδειγμα της προηγούμενης υποενότητας (Hayes, 1998, σ. 142). Κατα συνέπεια, στις υπόλοιπες εντολές γίνεται αποκλειστική χρήση άμεσης διευθυνσιοδότησης ή κατευθείαν από καταχωρητή.

Η σύγχρονη τεχνολογία ευνοεί την αρχιτεκτονική Load/Store. Οι ταχύτητες των CPU είναι πολύ μεγαλύτερες σε σχέση με τις ταχύτητες της μνήμης και η ελαχιστοποίηση των αργών προσβάσεων στη μνήμη, που επιτυγχάνεται με την αρχιτεκτονική αυτή, βελτιώνει συνολικά την απόδοση (Tanenbaum, 2013, σ. 422).



### 2.2.7. Μορφές στοίβας

Για την διατήρηση και την επαναφορά των τιμών των καταχωρητών, αλλά και για την κλήση ρουτινών, είναι απαραίτητη η υλοποίηση στοίβας. Οι Harris & Harris (2022, σ. 322) και ο Thind κ.α. (2017) ορίζουν τέσσερις διαφορετικές οργανώσεις στοίβας:

- Πλήρης καθοδική: Ο δείκτης δείχνει το τελευταίο στοιχείο που ωθήθηκε και μειώνεται στην ώθηση. Για παράδειγμα αν ο δείκτης είναι 100 και γίνει ώθηση, ο δείκτης θα μειωθεί στο 99 και το στοιχείο θα ωθηθεί στην θέση 99.
- Κενή καθοδική: Παρόμοια με την πλήρη καθοδική, αλλά ο δείκτης δείχνει την θέση μνήμης που είναι υποψήφια να δεχτεί το επόμενο στοιχείο. Για παράδειγμα αν ο δείκτης είναι 100 και γίνει ώθηση, το στοιχείο θα ωθηθεί στην θέση 100 και ο δείκτης θα μειωθεί στο 99.
- Πλήρης ανοδική: Ο δείκτης δείχνει το τελευταίο στοιχείο που ωθήθηκε και αυξάνεται στην ώθηση. Για παράδειγμα αν ο δείκτης είναι 100 και γίνει ώθηση, ο δείκτης θα αυξηθεί στο 101 και το στοιχείο θα ωθηθεί στην θέση 101.
- Κενή ανοδική: Παρόμοια με την πλήρη ανοδική, αλλά ο δείκτης δείχνει την θέση μνήμης που είναι υποψήφια να δεχτεί το επόμενο στοιχείο. Για παράδειγμα αν ο δείκτης είναι 100 και γίνει ώθηση, το στοιχείο θα ωθηθεί στην θέση 100 και ο δείκτης θα αυξηθεί στο 101.

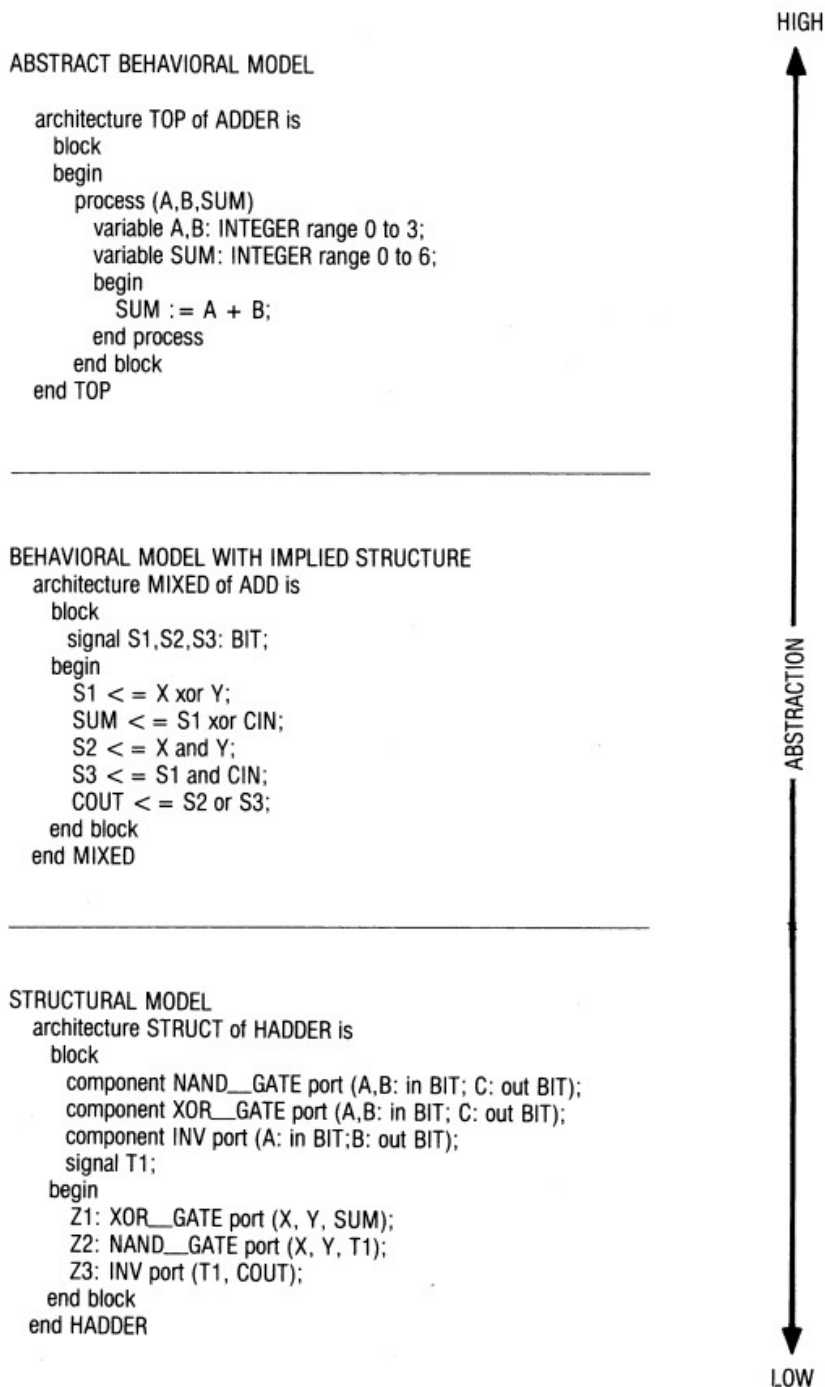
Στην εισαγωγική θεωρία δομών δεδομένων (Χατζηλυγερούδης, 2008, σ. 136) και στην σχολική βιβλιογραφία η στοίβα παρουσιάζεται ως πλήρης ανοδική, αλλά στους επεξεργαστές η πιο συνηθισμένη είναι η πλήρης καθοδική (π.χ. 8085 και RISC-V).

### 2.3. VHDL

Η γλώσσα VHDL, ακρωνύμιο του Very High Speed Integrated Circuits Hardware Description Language, ξεκίνησε το 1981 από το υπουργείο άμυνας των Ηνωμένων Πολιτειών Αμερικής με σκοπό την κατασκευή μιας γλώσσας περιγραφής της δομής και της λειτουργίας των ολοκληρωμένων κυκλωμάτων. Τυποποιήθηκε το 1987 από το IEEE καθώς θεωρήθηκε ότι υπερτερούσε σε χαρακτηριστικά σε σχέση με άλλες γλώσσες περιγραφής υλικού (Aylor, 1986). Θα πρέπει να σημειωθεί ότι η δημοσίευση του Aylor δεν συμπεριλαμβάνει την Verilog που είχε εμφανιστεί το 1984.

Μια συνηθισμένη εφαρμογή της VHDL είναι ο προγραμματισμός συστοιχιών προγραμματιζομένων πυλών (FPGA). Ο κώδικας VHDL συντίθεται, μέσω εφαρμογών όπως το Quartus ή το Gowin, σε αρχείο διαμόρφωσης το οποίο ρυθμίζει τα λογικά στοιχεία της συστοιχίας έτσι ώστε να λειτουργήσουν σύμφωνα με την περιγραφή στην VHDL.

Αρχικά η VHDL χαρακτηρίστηκε ως εκτενής και σημασιολογικά πολύπλοκη (Bhasker, 1998). Χαρακτηριστικό είναι το υποκοριστικό “Verbose HDL” με την έννοια της “πολυλογία” που της είχε αποδοθεί. Τα προβλήματα αυτά αντιμετωπίστηκαν από την VHDL 2008, η οποία διατήρησε μεν το χαρακτηριστικό της ισχυρής τυποποίησης αλλά απλοποίησε σε σημαντικό βαθμό την αναγνωσιμότητα του κώδικα (Lewis, 2013).



Εικόνα 4: Επίπεδα αφαιρετικότητας της VHDL<sup>1</sup>

### 2.3.1. Η αφαιρετικότητα της VHDL

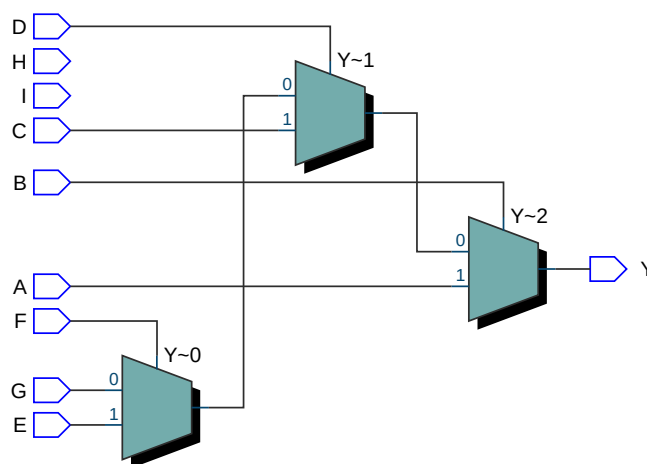
Η αφαιρετικότητα και η ανεξαρτησία από τις τεχνικές λεπτομέρειες είναι κοινό χαρακτηριστικό των γλωσσών περιγραφής κυκλωμάτων, έτσι ώστε να διευκολύνουν την τεκμηρίωση, την προσομοίωση και την σύνθεση κυκλώματος ανεξάρτητα από συγκεκριμένες πλατφόρμες και εργαλεία. Στην Εικόνα 4 καταγράφεται ένα παράδειγμα των επιπέδων αφαιρετικότητας της VHDL.

<sup>1</sup> Σύμφωνα με τον Aylor (1986), το διάγραμμα προέρχεται από την 1η δημοσιευμένη έκδοση της VHDL.

Για να δοθεί ένα παράδειγμα της αφαιρετικότητας, έστω ότι θέλουμε να αναθέσουμε στο **Y** το σήμα **A** αν **B=1**, αλλιώς το **C** αν **D=1**, αλλιώς το **E** αν **F=1**, αλλιώς το **G**. Η αντίστοιχη ανάθεση σε VHDL γράφεται σχεδόν ως φυσική γλώσσα:

```
Y <=
  A WHEN B ELSE
  C WHEN D ELSE
  E WHEN F ELSE
  G;
```

Η δήλωση αυτή μετατρέπεται σε ένα συνδυαστικό κύκλωμα από λογικές πύλες και συστατικά που διδάσκονται στις αρχικές ενότητες της ψηφιακής σχεδίασης. Για παράδειγμα το Quartus μετατρέπει την παραπάνω δήλωση στο κύκλωμα της Εικόνας 5 όπως φαίνεται παρακάτω.



Εικόνα 5: Αντιστοίχιση της WHEN σε κύκλωμα από τον RTL Viewer του Quartus

### 2.3.2. Η δήλωση PROCESS της VHDL

Το παράδειγμα της WHEN που δόθηκε στην προηγούμενη υποενότητα αφορά σε δηλώσεις που εκτελούνται ταυτόχρονα και περιγράφουν ένα συνδυαστικό κύκλωμα.

Η VHDL δεν υποστηρίζει την περιγραφή ακολουθιακών κυκλωμάτων, όπως τα flip flop, με τέτοιες δηλώσεις. Για παράδειγμα, η προσπάθεια να περιγραφεί ο μανδαλωτής που παρουσιάστηκε στην Ενότητα 2.1.1 (σ. 3) με τον παρακάτω κώδικα...

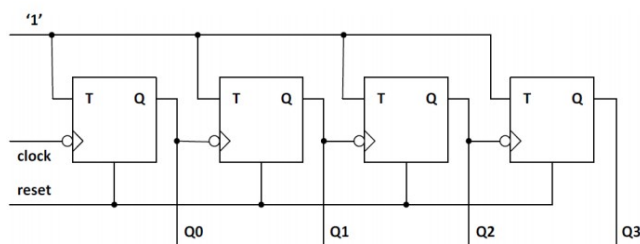
```
NotQ <= Q NOR (CLK AND D)
Q <= NotQ NOR (CLK AND NotD)
```

...καταλήγει σε ατέρμονα συνδυαστικό βρόχο (combinational loop) και δεν μπορεί να προσομοιωθεί.

Για την περιγραφή των ακολουθιακών δομών, υπάρχει η δήλωση PROCESS που ορίζει ένα μπλοκ σειριακών εντολών και μια λίστα ευαισθησίας. Οι εντολές ενεργοποιούνται και εκτελούνται σειριακά όταν αλλάζει οποιοδήποτε σήμα της λίστας ευαισθησίας. Η δυνατότητα αυτή αποτελεί βασικό πλεονέκτημα της VHDL και επιτρέπει την κατασκευή πολύπλοκων κυκλωμάτων με αφαιρετικό κώδικα, και με ταυτόχρονη αποφυγή προβλημάτων χρονισμού. Ένα χαρακτηριστικό παράδειγμα είναι ο τρόπος υλοποίησης ενός Ασύγχρονου Απαριθμητή mod16 όπως φαίνεται στην Εικόνα 6 που ακολουθεί.

## 4-bit Ασύγχρονος Προσθετικός mod-16

T-FF (toggle) (T='1')



```
Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_unsigned.all;

Entity counter_up_async is
port (clock,reset: In std_logic;
      Q: out std_logic_vector (3 downto 0));
End counter_up_async;

Architecture behavioral of counter_up_async is
Signal tmp: std_logic_vector (3 downto 0);
begin

Process (clock,reset)
begin
    if reset = '1' then
        tmp <= "0000";
    elsif clock'event and clock = '0' then
        tmp <= tmp + 1;
    end if;
End process;

Q <= tmp;
End behavioral;
```

Εικόνα 6: Ασύγχρονος απαριθμητής με δήλωση PROCESS (EAI, DSMC lab)

Ο κώδικας αυτός μπορεί να εκτελεστεί σε μια FPGA όπου τα απαραίτητα ακολουθιακά στοιχεία θα δημιουργηθούν αυτόματα από έναν μεταγλωττιστή της VHDL. Τα αποτελέσματά του είναι αξιόπιστα: οι Harris & Harris (2022, σ. 221) υποστηρίζουν την χρήση της PROCESS με συμπερίληψη όλων των σημάτων στην λίστα ευαισθησίας ακόμα και σε συνδυαστικά κυκλώματα.

Ωστόσο, σε αντίθεση με τις ταυτόχρονες εντολές που αντιπροσωπεύουν πιστά την παράλληλη λειτουργία των συνδυαστικών κυκλωμάτων, η PROCESS εισάγει μια σειριακή εκτέλεση που μπορεί να απομακρύνει το μοντέλο από τη φυσική συμπεριφορά του υλικού (Ashenden, 2008, σ. 141). Η κριτική αυτή αποτελεί ένα κεντρικό στοιχείο του σχεδιασμού του E80. Στο παράδειγμα της Εικόνας 6, ο κώδικας έχει αποκρύψει εντελώς την λεπτομέρεια της υλοποίησης με αποτέλεσμα να μην μπορεί να χρησιμοποιηθεί σαν εργαλείο εκμάθησης του T flip flop.

### 2.3.3. Πάγκος δοκιμών (testbench)

Ένα testbench είναι μια μονάδα που χρησιμοποιείται για την δοκιμή μιας άλλης μονάδας. Ο χρήστης πρέπει να εξετάσει τα αποτελέσματα της προσομοίωσης και να επιβεβαιώσει ότι οι έξοδοι είναι σωστές ή μπορεί να χρησιμοποιήσει κώδικα για να αυτοματοποιήσει τον έλεγχο. Στο παρακάτω παράδειγμα testbench, ελέγχεται η μονάδα ALU της Εικόνας 2 (σ. 4):

```
SIGNAL A      : STD_LOGIC_VECTOR(7 DOWNTO 0) := "11000010";
SIGNAL B      : STD_LOGIC_VECTOR(7 DOWNTO 0) := "01111110";
SIGNAL FlagsIn : STD_LOGIC_VECTOR(7 DOWNTO 0) := "UUUUUUUU";
SIGNAL ALUOp   : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
SIGNAL Result, FlagsOut : WORD;
BEGIN
    ALUOp <= STD_LOGIC_VECTOR(UNSIGNED(ALUOp) + 1) AFTER 50 ps;
    ALU : ENTITY work.ALU PORT MAP(A, B, FlagsIn, ALUOp, Result, FlagsOut);
```

Στο παράδειγμα εισάγονται δοκιμαστικές τιμές στα διανύσματα A, B, και ALUOp, ενώ το FlagsIn τίθεται σε απροσδιοριστία. Το ALUOp προσανξάνεται διαδοχικά κατά 1, και σε κάθε μεταβολή του θα πρέπει να ελεγχθούν οι έξοδοι Result και FlagsOut.

Τα testbenches προσομοιώνονται κανονικά αλλά δεν είναι υλοποιήσιμα σε κύκλωμα. Στο παράδειγμα που δόθηκε, η δήλωση “AFTER 50ps” δεν αντιστοιχεί σε πραγματικό υλικό. Επίσης το σήμα ‘U’ (undefined) δεν υπάρχει σε ένα πραγματικό ψηφιακό κύκλωμα όπου όλα τα σήματα έχουν τιμή ‘0’ ή ‘1’ ή το ‘Z’ που εκφράζει υψηλή εμπέδηση.

## 2.4. Συμβολομεταφραστής

Ένας συμβολομεταφραστής (assembler) είναι ένα εργαλείο που μεταφράζει εντολές συμβολικής γλώσσας (συμβολογλώσσα<sup>1</sup> ή assembly) σε κώδικα συμβατό με την αρχιτεκτονική εντολών του επεξεργαστή. Η μετάφραση είναι σχεδόν άμεση, σε αντιδιαστολή με τις εντολές μιας γλώσσας υψηλού επιπέδου που μεταγλωττίζονται απο μια πολύπλοκη διαδικασία. Για παράδειγμα, η κατασκευή ενός συμβολομεταφραστή είναι απλούστερη από αυτήν ενός μεταγλωττιστή λόγω της απουσίας αναδρομής στη σύνταξη των εντολών, καθώς δεν υπάρχει εμφώλευση και σύνθετες εκφράσεις. Ένας συμβολομεταφραστής χρησιμοποιεί οδηγίες (directives) και ετικέτες που σηματοδοτούν συγκεκριμένες διευθύνσεις στην μνήμη για να κάνει τον κώδικα πιο ευανάγνωστο και συντηρήσιμο.

Η βιβλιογραφία της αρχιτεκτονικής υπολογιστών συμπεριλαμβάνει και οδηγίες για συγγραφή συμβολομεταφραστή (πχ. Stallings, [2015](#)) αλλά στην παρούσα εργασία ο σχεδιασμός του Assembler θα βασιστεί στις αρχές που διατυπώνονται στο “Dragonbook” των Aho κ.α. ([2006](#)) το οποίο χρησιμοποιείται στο πρόγραμμα σπουδών Πληροφορικής του ΕΑΠ.

### 2.4.1. Backus-Naur Form (BNF)

Ως γλώσσα προγραμματισμού, η συμβολική γλώσσα ενός επεξεργαστή χαρακτηρίζεται από την ίδια αυστηρότητα που έχουν οι γλώσσες υψηλού επιπέδου και επομένως, μπορεί και πρέπει να περιγραφεί με ακρίβεια μέσω ασυμφραστικής γραμματικής.

Η μορφή Backus-Naur αποτελεί ένα πρότυπο για την αναπαράσταση αυτών των γραμματικών, επιτρέποντας τον σαφή ορισμό των συντακτικών κανόνων της γλώσσας. Παρά την παλαιότητά της (χρησιμοποιήθηκε για την περιγραφή της σύνταξης της αρχικής έκδοσης της ALGOL), η BNF συνεχίζει να χρησιμοποιείται σε διάφορες παραλλαγές.

### 2.4.2. Λεκτική ανάλυση, συλλογή συμβόλων, και συντακτική ανάλυση

Η λεκτική ανάλυση αποτελεί το αρχικό στάδιο της διαδικασίας μεταγλώττισης. Σε αυτή τη φάση, ο λεκτικός αναλυτής διαβάζει τον πηγαίο κώδικα ως μια ακολουθία χαρακτήρων και τον ομαδοποιεί σε μια σειρά από tokens. Τα tokens είναι οι θεμελιώδεις συντακτικές μονάδες της γλώσσας, όπως αναγνωριστικά, αριθμοί και σύμβολα. Ο λεκτικός αναλυτής είναι υπεύθυνος για την αναγνώριση αυτών των μονάδων και την παράβλεψη στοιχείων όπως οι περιττοί κενοί χαρακτήρες και τα σχόλια.

Για την ακριβή αντιστοίχιση συμβόλων σε διευθύνσεις μνήμης ή τιμές, ένας συμβολομεταφραστής θα πρέπει αρχικά να αποθηκεύσει τα σύμβολα και τις τιμές ή τις διευθύνσεις τους σε ένα πρώτο πέρασμα, και μετά να αντιστοιχίσει τις αναφορές τους με τις τιμές αυτές σε δεύτερο πέρασμα.

Μετά τη λεκτική ανάλυση, η συντακτική ανάλυση αναλαμβάνει να ελέγξει αν η ακολουθία των token που παρήχθη από το λεκτικό αναλυτή συμμορφώνεται με τους συντακτικούς κανόνες της ασυμφραστικής γραμματικής. Για τους λόγους προσδιορίστηκαν στην αρχή αυτής της ενότητας, η συντακτική ανάλυση της Assembly είναι σχετικά εύκολη.

<sup>1</sup> [https://search.eleto.gr/termbank?term\\_value=assembly&search\\_mode=complete&database\\_field=TERMEN-&databases=inforterm&display\\_english=true](https://search.eleto.gr/termbank?term_value=assembly&search_mode=complete&database_field=TERMEN-&databases=inforterm&display_english=true)

### 2.4.3. Παραγωγή κώδικα

Στο τελικό στάδιο της παραγωγής κώδικα οι εσωτερικές αναπαραστάσεις του προγράμματος μετατρέπονται στον τελικό κώδικα μηχανής. Για έναν συμβολομεταφραστή, αυτό συνεπάγεται την αντιστοίχιση κάθε συμβολικής εντολής και των τελουμένων της στον αντίστοιχο κώδικα μηχανής όπως αυτός ορίζεται από την αρχιτεκτονική εντολών του επεξεργαστή. Το αποτέλεσμα είναι ένα αρχείο που περιέχει την εκτελέσιμη δυαδική μορφή του προγράμματος, έτοιμο για εκτέλεση.

## 2.5. Εργαλεία προσομοίωσης

Η προσομοίωση (simulation) αναφέρεται στη διαδικασία μοντελοποίησης της συμπεριφοράς ενός συστήματος σε λογισμικό. Στην περίπτωση της VHDL, ο κώδικας εκτελείται σε έναν προσομοιωτή, ο οποίος υπολογίζει τις τιμές των σημάτων σε διακριτές χρονικές στιγμές. Η διαδικασία της προσομοίωσης είναι απαραίτητη για την διόρθωση σφαλμάτων, αλλά ταυτόχρονα, και ειδικά στο πλαίσιο αυτής της εργασίας, παρέχει και ένα εργαλείο πειραματισμού και εκμάθησης με μηδενικό κόστος.

### 2.5.1. Το ρίσκο της εξάρτησης από εμπορικά εργαλεία

Για τις ανάγκες της εργασίας εξετάστηκαν δύο διαδεδομένα εμπορικά περιβάλλοντα, το ModelSim και το Active-HDL, τα οποία παρέχουν ολοκληρωμένες λύσεις για την προσομοίωση ψηφιακών συστημάτων.

Το ModelSim, προϊόν της Siemens, αποτελεί ένα από τα πιο αναγνωρισμένα εμπορικά περιβάλλοντα προσομοίωσης για VHDL. Παρότι η Intel εξακολουθεί να προσφέρει μια δωρεάν έκδοση του<sup>1</sup>, η διαθεσιμότητα των φοιτητικών εκδόσεων από την ίδια την Siemens έχει καταστεί αβέβαιη, με την εταιρεία να επικαλείται αλλαγές στην ομοσπονδιακή νομοθεσία των ΗΠΑ<sup>2</sup>.

Αντίστοιχα, το Active-HDL της εταιρείας Aldec, προσφέρει ένα ισχυρό προσομοιωτή που υποστηρίζει μέχρι και την VHDL 2019, και εύχρηστο γραφικό περιβάλλον, αλλά η δωρεάν φοιτητική του έκδοση απαιτεί εγγραφή με παροχή προσωπικών δεδομένων<sup>3</sup>. (Και κατά την χρήση του στην Ενότητα 8.6, σ. 106, προέκυψε ένα σοβαρότερο μειονέκτημα).

Η εξάρτηση από τέτοιου είδους εμπορικά εργαλεία, ακόμη και στις δωρεάν εκδόσεις τους, ενέχει ρίσκο. Η διαθεσιμότητά τους μπορεί να ανακληθεί ή να περιοριστεί ανά πάσα στιγμή, όχι μόνο λόγω αλλαγών στην εμπορική πολιτική της εκάστοτε εταιρείας, αλλά και εξαιτίας ευρύτερων γεωπολιτικών παραγόντων, όπως εμπορικοί περιορισμοί μεταξύ κρατών. Αυτή η αβεβαιότητα καθιστά τα εμπορικά εργαλεία, ή τουλάχιστον την προσκόλληση σε ένα από αυτά, επισφαλή για μακροπρόθεσμα ακαδημαϊκά ή ερευνητικά έργα που απαιτούν απρόσκοπτη πρόσβαση.

<sup>1</sup> <https://www.intel.com/content/www/us/en/software-kit/750666/modelsim-intel-fpgas-standard-edition-software-version-20-1-1.html>

<sup>2</sup> <https://eda.sw.siemens.com/en-US/modelsim-student-edition-unavailable/>

<sup>3</sup> <https://www.aldec.com/students/student.php?id=56>



## 2.5.2. GHDL και GTKWave: η επιλογή του ελεύθερου λογισμικού

Ενώ τα εμπορικά πακέτα προσφέρουν ένα αδιαμφισβήτητο ισχυρό και ενοποιημένο περιβάλλον, οι κίνδυνοι διαθεσιμότητας που αναλύθηκαν κατέστησαν αναγκαία την διερεύνηση ενός εναλλακτικού τρόπου προσομοίωσης της VHDL. Η διερεύνηση αυτή κατέληξε στον συνδυασμό GHDL & GTKWave.

Το GHDL<sup>1</sup> είναι ένας προσομοιωτής VHDL ο οποίος διατίθεται υπο την άδεια GPL και επομένως είναι δωρεάν και ελεύθερος. Παρουσιάζει σημαντικό πλεονέκτημα στην ταχύτητα εκτέλεσης, καθώς λειτουργεί ως μεταγλωττιστής, μετατρέποντας τον κώδικα VHDL απευθείας σε εκτελέσιμο κώδικα μηχανής. Το κύριο μειονέκτημά του είναι η απουσία ενσωματωμένου γραφικού περιβάλλοντος.

Για την κάλυψη αυτής της ανάγκης, το GHDL συνδυάζεται με το GTKWave<sup>2</sup>, το *defacto* δωρεάν και ελεύθερο λογισμικό προβολής ψηφιακών κυματομορφών υπο την άδεια GPL. Το GTKWave διαβάζει τα αρχεία αποτελεσμάτων που παράγει το GHDL (.ghw) και τα παρουσιάζει σε γραφική μορφή, επιτρέποντας την ανάλυση της χρονικής συμπεριφοράς του κυκλώματος.

Ο συνδυασμός GHDL και GTKWave αποτελεί μια λύση μηδενικού κόστους που είναι ανθεκτική σε εξωτερικές εξαρτήσεις, καθιστώντας τον μια σημαντική εναλλακτική λύση για την ακαδημαϊκή έρευνα και εκπαίδευση.

## 2.6. FPGA

Οι συστοιχίες επιτοπίως προγραμματιζομένων πυλών προσφέρουν έναν μεγάλο αριθμό λογικών στοιχείων και άλλων ψηφιακών (και ενίοτε αναλογικών) λειτουργιών που προγραμματίζονται απο εφαρμογές όπως το Quartus ή το Gowin. Οι λειτουργίες αυτές δεν είναι εικονικές· μια FPGA δεν προσομοιώνει ένα κύκλωμα, αλλά μετατρέπεται σε αυτό.

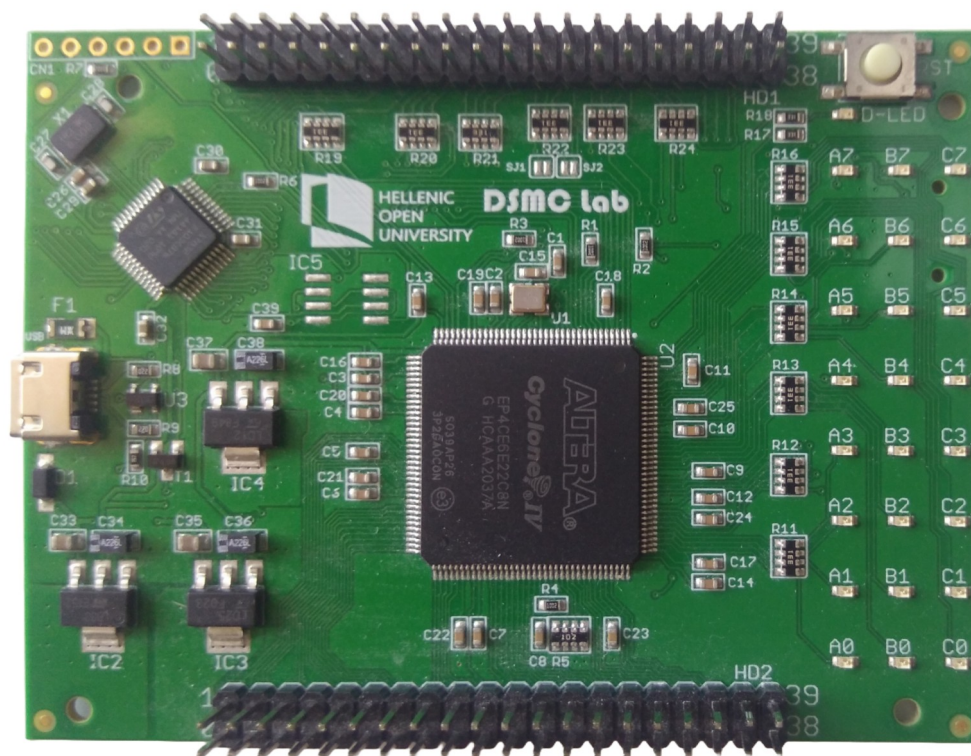
### 2.6.1. Αναπτυξιακή πλακέτα DSD-i1 και Quartus

Η αναπτυξιακή πλακέτα DSD-i1 όπως φαίνεται στην Εικόνα 7 αποτελεί μια ολοκληρωμένη εκπαιδευτική πλατφόρμα ψηφιακής σχεδίασης, που αναπτύχθηκε στο DSMC lab του ΕΑΠ (Φαναριώτης, Ορφανουδάκης, Φωτόπουλος, Κίτσος, 2019) και έχει αποδείξει την εκπαιδευτική αξία και την στιβαρότητά της ως εργαλείο των φοιτητών των ενοτήτων ΠΛΗΨ I & II.

Συνδυάζει το FPGA Cyclone IV της Intel/Altera και ένα μικροελεγκτή που βασίζεται στον ARM Cortex-M3. Ο εξοπλισμός περιλαμβάνει ρολόι 50 MHz, 6000 λογικά στοιχεία, 25 LED, 80 ακροδέκτες, και θύρα micro-USB για τροφοδοσία και επικοινωνία με υπολογιστή. Παρέχει δυνατότητες προγραμματισμού απο το Keil MDK και το Arduino IDE αλλά το κύριο ενδιαφέρον ως προς την εργασία αυτή εστιάζεται στον προγραμματισμό της ως FPGA απο το Quartus.

<sup>1</sup> <https://ghdl.github.io/ghdl/about.html>

<sup>2</sup> <https://sourceforge.net/projects/gtkwave/files/gtkwave-3.3.100-bin-win32>



Εικόνα 7: Αναπτυξιακή πλακέτα DSD-i1 του DSMC lab

Το Quartus είναι το περιβάλλον ανάπτυξης της Intel για τον σχεδιασμό, τη σύνθεση, την προσομοίωση και τον προγραμματισμό των δικών της FPGA. Αποτελεί ένα επαγγελματικό εργαλείο βιομηχανικών προδιαγραφών που καλύπτει ολόκληρη τη ροή εργασιών από τον σχεδιασμό κυκλωματικού διαγράμματος και την συγγραφή του κώδικα HDL, μέχρι τον έλεγχο των χρονικών περιορισμών και την διαμόρφωση του τελικού υλικού.

### 2.6.2. Quartus Prime Lite

Το Quartus Prime Lite είναι η δωρεάν έκδοση του Quartus. Η Intel υποστηρίζει ότι το Prime Lite δεν πληροί<sup>1</sup> την VHDL 2008 αλλά αυτό δεν ισχύει απόλυτα. Κατα την διάρκεια της συγγραφής έγινε αντιληπτό ότι υποστηρίζεται ένα υποσύνολο της VHDL 2008, και μέσα απο μια διαδικασία δοκιμών προέκυψε η ακόλουθη λίστα με επιβεβαιωμένες ασυμβατότητες:

- Δεν υποστηρίζονται τελεστές σύγκρισης με αδιάφορα σήματα όπως για παράδειγμα η συνθήκη  $x \neq "10--1"$ .
- Δεν υποστηρίζονται μονομερείς λογικοί τελεστές/πύλες σε διανύσματα όπως για παράδειγμα η έκφραση NOR Vector.
- Δεν επιτρέπεται ανάθεση με WHEN εντός επιλογής IF σε ακολουθιακή διαδικασία.
- Η VHDL 2008 επιτρέπει την χρήση θυρών εξόδου σε δηλώσεις ανάθεσης<sup>2</sup>. Στο Quartus Prime Lite αυτό απαιτεί την δήλωση των εν λόγω θυρών ως BUFFER.
- Δεν υποστηρίζονται εντολές ολίσθησης και περιστροφής (πχ. ROR).

<sup>1</sup> <http://www.intel.com/content/dam/www/central-libraries/us/en/documents/quartus-prime-compare-editions-guide.pdf>

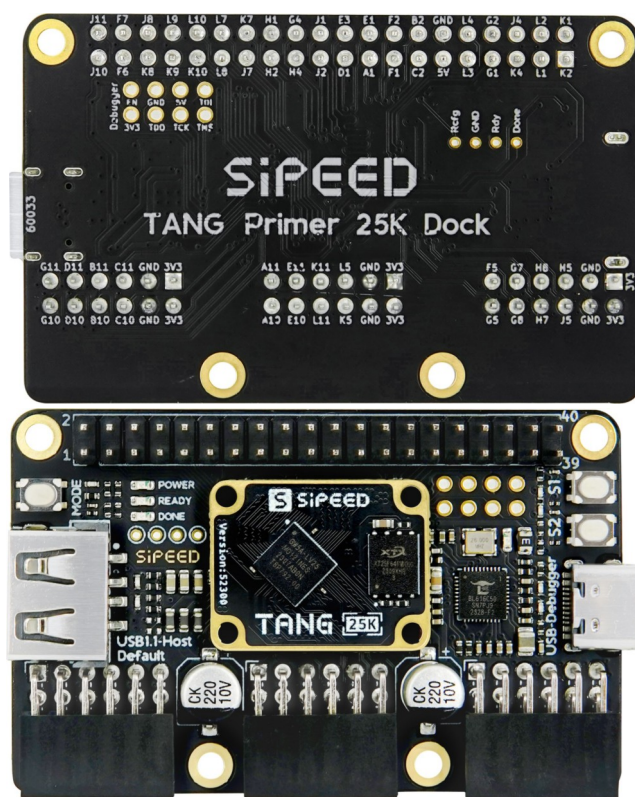
<sup>2</sup> <https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Reading-Output-Ports>



### 2.6.3. Tang Primer 25K και Gowin IDE

Η αναπτυξιακή πλακέτα Tang Primer 25K όπως φαίνεται στην Εικόνα 8 είναι μια συμπαγής πλατφόρμα ψηφιακής σχεδίασης, σχεδιασμένη από τη Sipeed. Βασίζεται στο FPGA Aurora V, προσφέροντας 23000 λογικά στοιχεία και ρολόι 50 MHz. Για τους τελικούς χρήστες η Tang Primer 25K συνοδεύεται από την ειδική βάση (25K dock base board) η οποία παρέχει 88 ακροδέκτες και θύρα USB-C.

Το Tang Primer 25K υποστηρίζεται από το Gowin IDE, επιτρέποντας προγραμματισμό μέσω USB χωρίς εξωτερικά εργαλεία. Αν και το Gowin δεν ανταγωνίζεται στο επίπεδο του Quartus ως προς τις δυνατότητες, στις δοκιμές που έγιναν στην εργασία αυτή, προσέφερε γρηγορότερη μεταγλώττιση, ευκολότερο ανέβασμα του κώδικα στην πλακέτα, και πλήρη υποστήριξη της VHDL 2008 αλλά και της VHDL 2019.



Εικόνα 8: Tang Primer 25K τοποθετημένο στην κατάλληλη dock base board<sup>1</sup>

### 2.7. Οικοδομισμός και εκπαιδευτικοί μικρόκοσμοι

Οι εκπαιδευτικοί σκοποί του συστήματος που θα κατασκευαστεί πηγάζουν από τις θέσεις του Jean Piaget και, κυρίως, του Seymour Papert. Ο Πιαζέ διατύπωσε τη θεωρία του δομικού οικοδομισμού (constructivism), η οποία υποστηρίζει ότι η γνώση δεν μεταδίδεται παθητικά, αλλά οικοδομείται ενεργά μέσα στο μυαλό του κάθε ατόμου μέσα από τις εμπειρίες του και την αλληλεπίδρασή του με τον κόσμο. Ο Πάπερτ, μαθητής και συνεργάτης του Πιαζέ, επέκτεινε αυτή τη θεωρία, δημιουργώντας τον όρο του οικοδομισμού (constructionism) ο οποίος ενθαρρύνει την κατασκευή και διαμοίραση τεχνουργημάτων σε ένα κατάλληλα διαμορφωμένο περιβάλλον που ονομάζεται Εκπαιδευτικός Μικρόκοσμος.

<sup>1</sup> <https://wiki.sipeed.com/hardware/en/tang/tang-primer-25k/primer-25k.html>

Παρόλο που τα αρχικά παραδείγματα μικρόκοσμων όπως η Logo συνδέθηκαν στενά με την εκπαίδευση σε μικρές ηλικίες, η έρευνα έδειξε ότι οι αρχές αυτές έχουν εφαρμογή και στην τριτοβάθμια εκπαίδευση. Οι Psenka κ.α. (2017) μελετούν την οικοδομιστική μάθηση στην εκπαίδευση μηχανικών σε πανεπιστημιακό επίπεδο, ενώ οι Rojprasert κ.α. (2013) προτείνουν μοντέλα που συνθέτουν τον οικοδομισμό με την αυτομελέτη στο περιβάλλον της τριτοβάθμιας εκπαίδευσης. Η Sacristán (2017) καταγράφει επιτυχή εφαρμογή του οικοδομισμού στην εκμάθηση στατιστικής στην τριτοβάθμια εκπαίδευση μέσω ενός μικρόκοσμου που έχει κατασκευαστεί στην γλώσσα R.

Η θεμελιώδης αρχή της οικοδόμησης της γνώσης μέσω της κατασκευής παραμένει καθολική· αυτό που διαφοροποιείται είναι η πολυπλοκότητα του μικρόκοσμου και το ακαδημαϊκό επίπεδο των εννοιών που αυτός πραγματεύεται.

### 2.7.1. Αρχές σχεδιασμού μικρόκοσμων

Για να είναι ένας μικρόκοσμος αποτελεσματικός ως εκπαιδευτικό εργαλείο, ο σχεδιασμός του πρέπει να διέπεται από συγκεκριμένες αρχές. Ο Πάπερτ και οι συνεργάτες του στο MIT ανέπτυξαν ένα πλαίσιο αρχών για τον σχεδιασμό περιβαλλόντων, το οποίο έχει γίνει ευρέως γνωστό με τον όρο “χαμηλό κατώφλι, υψηλό ταβάνι και ευρείς τοίχοι” (Φράγκου & Παπανικολάου, 2010).

- Το “χαμηλό κατώφλι” αναφέρεται στην ανάγκη να μπορούν οι αρχάριοι χρήστες να εισέλθουν εύκολα στο περιβάλλον και να ξεκινήσουν άμεσα να δημιουργούν, βιώνοντας μια αίσθηση επιτυχίας που τους δίνει κίνητρο για περαιτέρω ενασχόληση.
- Το “υψηλό ταβάνι” υποδηλώνει ότι το περιβάλλον πρέπει να παρέχει στους χρήστες τον χώρο να εξελιχθούν και να εργαστούν πάνω σε ολοένα και πιο σύνθετα και απαιτητικά έργα, καθώς οι δεξιότητές τους αυξάνονται.
- Οι “ευρείς τοίχοι”, αναφέρονται στην ικανότητα του περιβάλλοντος να υποστηρίζει μια ευρεία γκάμα διαφορετικών τύπων έργων, επιτρέποντας στους εκπαιδευόμενους να εξερευνήσουν τα δικά τους ενδιαφέροντα και να εκφραστούν με ποικίλους τρόπους.

### 2.7.2. “Ισχυρές ιδέες” κατά Πάπερτ

Ο τελικός στόχος της ενασχόλησης ενός εκπαιδευόμενου με έναν μικρόκοσμο, κατά τον Πάπερτ, δεν είναι η εκμάθηση μιας δεξιότητας, αλλά η επαφή του με “ισχυρές ιδέες” (Powerful Ideas). Η “ισχυρή ιδέα” είναι μια θεμελιώδης έννοια που, όταν κατακτηθεί, παρέχει στον εκπαιδευόμενο έναν νέο τρόπο να σκέφτεται και να κατανοεί τον κόσμο, επιτρέποντάς του να συνδέσει γνώσεις από διαφορετικούς τομείς (Papert, 1980 & 2000).

Σε αντίθεση με την αποστήθιση, η κατάκτηση μιας ισχυρής ιδέας αλλάζει τον τρόπο που το άτομο προσεγγίζει και λύνει προβλήματα. Ο ρόλος του μικρόκοσμου είναι ακριβώς να δημιουργήσει ένα γόνιμο έδαφος όπου ο εκπαιδευόμενος μπορεί να ανακαλύψει και να οικοδομήσει αυτές τις ιδέες. Όπως αναφέρει ο Stager (2012), τα έργα που αναπτύσσονται σε ένα οικοδομιστικό περιβάλλον Πληροφορικής πρέπει να ενσωματώνουν ισχυρές ιδέες από τα μαθηματικά, τις θετικές επιστήμες, την επιστήμη των υπολογιστών και την μηχανική.

### 3. Καθορισμός των απαιτήσεων του E80

#### 3.1. Σύνολο συμβολικών εντολών που χρησιμοποιούνται στην εκπαίδευση

Για την υλοποίηση του στόχου της **Χρησιμότητας**, διερευνήθηκε το σύνολο εντολών που χρησιμοποιούνται στις σχετικές ενότητες του ΕΑΠ, καθώς και απο άλλες πηγές. Απο την έρευνα αυτή κατασκευάστηκαν οι πίνακες πρωτογενών δεδομένων που καταγράφονται στο Παράρτημα Α (σ. 146–151). Για την κανονικοποίηση των πρωτογενών δεδομένων έγιναν οι εξής συμβάσεις:

1. Οι εντολές δεν κατηγοριοποιήθηκαν με το όνομά τους αλλά με την λειτουργία τους. Για παράδειγμα οι εντολές της κατηγορίας “Άλμα αν ισότητα / μηδενικό αποτέλεσμα” έχουν διαφορετικό όνομα ανάλογα με την αρχιτεκτονική, αλλά η λειτουργία είναι πρακτικά η ίδια.
2. Οι πίνακες πρωτογενών δεδομένων θεωρήθηκαν ισοδύναμοι οπότε και το κανονικοποιημένο ποσοστό προέκυψε απο την εξίσωση:

$$N(\text{Λειτουργία}) = \frac{1}{6} \sum_{t=1}^6 P_t(\text{Λειτουργία})$$

Όπου  $P_t(\text{Λειτουργία})$  το ποσοστό μιας λειτουργίας στον πίνακα  $t$  και  $N(\text{Λειτουργία})$  το κανονικοποιημένο ποσοστό συνολικά.

Με βάση την παραπάνω εξίσωση και χρησιμοποιώντας την συνάρτηση SUMPRODUCT στο LibreOffice, προέκυψε ο Πίνακας 1 που καταγράφει τις πιο συνηθισμένες εντολές σε εκπαιδευτικό υλικό που αφορά στην εισαγωγή στην assembly.

Λειτουργία	Κανονικοποιημένο ποσοστό
Μεταφορά	11,49%
Πρόσθεση	11,16%
Αποθήκευση στην μνήμη	7,59%
Επιστροφή απο υπορουτίνα	5,56%
Φόρτωση απο μνήμη ή άμεσης τιμής	5,55%
Αφαίρεση	5,49%
Φόρτωση άμεσης τιμής	4,34%
Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα	3,81%
Φόρτωση απο μνήμη	3,38%
Παύση εκτέλεσης	3,13%
Άλμα αν ισότητα / μηδενικό αποτέλεσμα	3,13%
Ωθηση	2,97%
Άλμα χωρίς συνθήκη	2,93%
Απόθεση	2,74%
Πολλαπλασιασμός	2,72%

Σύγκριση	2,63%
Αύξηση κατα 1	2,04%
AND	1,93%
Κλήση υπορουτίνας	1,75%
Διαίρεση	1,66%
XOR	1,65%
Πρόσθεση άμεσης τιμής	1,65%
Μείωση κατα 1	1,35%
Αριστερή ολίσθηση	1,01%
Καμία λειτουργία	0,91%
OR	0,64%
Φόρτωση διεύθυνσης με υπολογισμούς	0,63%
Σύγκριση με AND	0,62%
Φόρτωση διεύθυνσης σε ζεύγος καταχωρητών	0,61%
Άλμα αν μεγαλύτερο ή ίσο	0,56%
Αύξηση ζεύγους καταχωρητών κατα 1	0,45%
Εκκαθάριση καταχωρητή	0,45%
Δεξιά ολίσθηση	0,45%
Πολλαπλασιασμός με πρόσημο	0,39%
NOT	0,38%
Άλμα αν δεν υπάρχει κρατούμενο	0,35%
Άλμα αν υπάρχει κρατούμενο	0,30%
Ανταλλαγή τιμών	0,27%
Αποθήκευση μέσω ζεύγους καταχωρητών	0,20%
Αριστερή περιστροφή	0,20%
Σύγκριση με άμεση τιμή	0,15%
NAND	0,13%
NOR	0,13%
Αφαίρεση άμεσης τιμής	0,10%
Δεξιά ολίσθηση μέσω κρατούμενου	0,10%
Δεξιά περιστροφή	0,10%
Πρόσθεση με κρατούμενο	0,10%
Φόρτωση μέσω ζεύγους καταχωρητών	0,10%

Πίνακας 1: Συχνότητα συμβολικών εντολών σε εκπαιδευτικό υλικό

## 3.2. Σύνολα εντολών σε άλλες υλοποιήσεις απλών CPU

### 3.2.1. Aizup (Li)

Ο Aizup είναι ένας επεξεργαστής αρχιτεκτονικής RISC με γραμμή διοχέτευσης (pipeline) που υλοποιήθηκε σε Xilinx FPGA στο πλαίσιο του μαθήματος Οργάνωσης Υπολογιστών στο Πανεπιστήμιο του Aizu (Li, [1996](#)). Μέσα από τον σχεδιασμό του επεξεργαστή οι φοιτητές μελέτησαν τεχνικές βελτιστοποίησης απόδοσης.

Εντολή	Περιγραφή
NOP	Καμία λειτουργία
ADD	Πρόσθεση
SUB	Αφαίρεση
OR	OR
AND	AND
XOR	XOR
MOV	Μεταφορά
LD	Φόρτωση από μνήμη
ST	Αποθήκευση στην μνήμη
ADDI	Πρόσθεση άμεσης τιμής
SUBI	Αφαίρεση άμεσης τιμής
SR0H/SR0L	Φόρτωση άμεσης τιμής
BZ	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
BNZ	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
BRA	Άλμα χωρίς συνθήκη

Πίνακας 2: Σύνολο εντολών Aizup (Li)

Το σύνολο εντολών φαίνεται μικρό, αλλά για την εποχή στην οποία σχεδιάστηκε είναι εντυπωσιακό. Οι εντολές προσθαφαίρεσης άμεσης τιμής περιορίζονται στα 2 bits ενώ η φόρτωση άμεσης τιμής γίνεται ξεχωριστά για το άνω και κάτω nibble ώστε να μπορεί η εντολή να χωρέσει σε 8 bits. Η αρχική σχεδίαση δεν χωρούσε ολόκληρη στο Xilinx 4006PC84 οπότε οι συγγραφείς μείωσαν την μνήμη από 256 στα 224 bytes.

Αν και τα προβλήματα αυτά δεν απασχολούν πλέον σε τέτοιο βαθμό, η βέλτιστη αξιοποίηση του διαθέσιμου υλικού είναι ένας βασικός στόχος της ψηφιακής σχεδίασης, και ο Aizup αποτέλεσε έμπνευση ως προς την σχεδίαση του E80.

### 3.2.2. Simple CPU (Freeman)

Η Απλή CPU που έχει επιλεγθεί στο εργαστήριο ΠΛΗΨΠ έχει βασιστεί στον σχεδιασμό της Simple CPU του Freeman ([2019](#)). Χρησιμοποιεί αρχιτεκτονική ενός συσσωρευτή και έχει το σύνολο εντολών που καταγράφεται στον Πίνακα 3. Η περιγραφή ακολουθεί τις συμβάσεις που γίνονται στους πίνακες του Παραρτήματος Α (σ. 146).

Εντολή	Περιγραφή
LOAD	Φόρτωση άμεσης τιμής
ADD	Πρόσθεση
AND	AND
SUB	Αφαίρεση
INPUT	Φόρτωση από μνήμη
OUTPUT	Αποθήκευση στην μνήμη
JUMP U	Άλμα χωρίς συνθήκη
JUMP NC	Άλμα αν δεν υπάρχει κρατούμενο
JUMP C	Άλμα αν υπάρχει κρατούμενο
JUMP Z	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
JUMP NZ	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα

Πίνακας 3: Σύνολο εντολών Simple CPU (Freeman)

Η Simple CPU του Freeman κατασκευάστηκε με σκοπό την απεικόνιση της διαδικασίας υλοποίησης μιας απλής αρχιτεκτονικής επεξεργαστή από λογικές πύλες. Το σύνολο εντολών της είναι μικρότερο από αυτό του συνόλου που παρουσιάζεται στο εκπαιδευτικό υλικό της εισαγωγικής ΠΛΗ10 για τον MIPS (σ. 146).

### 3.2.3. VSCPU (Yildiz)

Η Very Simple CPU (Yildiz, [2018](#)) έχει παρόμοιους στόχους με την Simple CPU του Freeman αλλά έχει δοκιμαστεί στην πράξη στα πανεπιστήμια Yeditepe και Özyeğin (Τουρκία). Όπως καταγράφεται στον Πίνακα 4 (σ. 21), όλες οι εντολές υποστηρίζουν και άμεση διευθυνσιοδότηση.

Εντολή	Περιγραφή
ADD/ADDi	Πρόσθεση
NAND/NANDi	NAND
SRL/SRLi	Ολίσθηση δεξιά/αριστερά
LT/LTi	Σύγκριση
MUL/MULi	Πολλαπλασιασμός
CPi	Αποθήκευση άμεσης τιμής
CP	Αποθήκευση από θέση μνήμης
CPi	Αποθήκευση από θέση μνήμης σε έμμεση θέση
CPI	Αποθήκευση από έμμεση θέση σε έμμεση θέση
BZJ	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
BZJi	Άλμα χωρίς συνθήκη

Πίνακας 4: Σύνολο εντολών VSCPU (Yildiz)



Η VSCPU διαθέτει δύο καταχωρητές. Το σύνολο είναι αντίστοιχο της Απλής CPU με την προσθήκη εντολών ολίσθησης και πολλαπλασιασμού. Στην σχετική δημοσίευση αναφέρεται η ύπαρξη καταχωρητή στοίβας, αλλά αυτό δεν προκύπτει από το σύνολο εντολών της. Σε αντίθεση με την Simple CPU, ο πηγαίος κώδικάς της<sup>1</sup> δεν είναι πλήρως δημοσιευμένος.

### 3.2.4. SC91-A (Pinault)

Η SC91A χρησιμοποιεί 32-bit RISC αρχιτεκτονική με 32 καταχωρητές (R00-R31), με τους R00-R02 να έχουν ειδικές λειτουργίες (PC, SP, flags). Όλες οι εντολές της εκτελούνται σε ένα κύκλο ρολογιού. Το σύνολο εντολών της παρουσιάζεται στον παρακάτω πίνακα:

Εντολή	Περιγραφή
LOADd	Φόρτωση άμεσης τιμής
LOADr	Φόρτωση τιμής από μνήμη
LOADm	Φόρτωση από μνήμη
STORE	Αποθήκευση στην μνήμη
AND	AND
OR	OR
XOR	XOR
JMPr	Άλμα χωρίς συνθήκη
CPYB	Αντιγραφή bits
ADDr	Πρόσθεση
MUL	Πολλαπλασιασμός
DIV	Διαίρεση
ROT	Αριστερή/Δεξιά περιστροφή
PUSH	Ωθηση
POP	Απόθηση
CALLr	Κλήση υπορουτίνας
NOP	Καμία λειτουργία

Πίνακας 5: Σύνολο εντολών SC91-A (Pinault)

Σε αντίθεση τις δύο προηγούμενες απλές CPU, η SC91-A συνοδεύεται από πλήρη τεκμηρίωση και κώδικα υπο την άδεια GPL. Οι εντολές του Πίνακα 5 είναι πλήρως υλοποιημένες, αλλά η αρχιτεκτονική αναφέρει ένα ακόμα μεγαλύτερο σύνολο εντολών που δεν έχει υλοποιηθεί καθώς το έργο βρίσκεται σε ημιτελές “*non testée*”<sup>2</sup> στάδιο ανάπτυξης από το 2005. Για παράδειγμα η τρέχουσα υλοποίηση προσφέρει κλήση υπορουτίνας αλλά όχι επιστροφής από αυτήν.

<sup>1</sup> <https://github.com/MC2SC/VerySimpleCPU-public>

<sup>2</sup> <https://www.simple-cpu.com/telechargements-sc91-fr.php>

### 3.2.5. TINYCPU

Η TINYCPU (Nakano & Ito, [2008](#)) έχει σχεδιαστεί με αρχιτεκτονική στοίβας και δεν διαθέτει καταχωρητές γενικού σκοπού. Όλες οι εντολές, που καταγράφονται στον Πίνακα 6 πλην της HALT, ωθούν ή απωθούν στην στοίβα και δεν υπάρχουν σημαίες. Για παράδειγμα η εντολή JZ πραγματοποιεί άλμα αν η κορυφή της στοίβας είναι μηδενική και μετά την απωθεί. Οι συμβολικές εντολές χρησιμοποιούν την ίδια ονοματολογία και σύνταξη με τα σχετικά παραδείγματα και τις ασκήσεις που διδάσκονται στο ΕΑΠ (Νικολός, [2008](#), σ. 78 και Αλεξίου κ.α., [2013α](#))

Εντολή	Περιγραφή
HALT	Τερματισμός προγράμματος
PUSHI	Ωθηση άμεσης τιμής
PUSH	Ωθηση απο διεύθυνση μνήμης
POP	Απόθεση
JMP	Άλμα χωρίς συνθήκη
JZ	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
JNZ	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
IN	Ωθηση απο θύρα εισόδου
OUT	Απόθεση σε buffer εξόδου
ADD	Πρόσθεση
SUB	Αφαίρεση
MUL	Πολλαπλασιασμός
SHL	Αριστερή ολίσθηση
SHR	Δεξιά ολίσθηση
BAND/AND	AND / λογικό AND
BOR/OR	OR / λογικό OR
BXOR	XOR
EQ/NE/GE/LE/GT	Σύγκριση
NEG	Συμπλήρωμα ως προς 2
BNOT/NOT	NOT / λογικό NOT

Πίνακας 6: Σύνολο εντολών TINYCPU (Nakano & Ito)

Η TINYCPU αποτελεί ένα εξαιρετικό παράδειγμα πρακτικής εφαρμογής της αρχιτεκτονικής στοίβας, και συνοδεύεται απο συμβολομεταφραστή και μεταγλωττιστή C. Χρησιμοποιείται από το 2008 στο Πανεπιστήμιο της Χιροσίμα όπου, σύμφωνα με τους Nakano και Ito, συνέβαλε ουσιαστικά στη διδασκαλία θεμάτων Ενσωματωμένου Υλικού και Λογισμικού. Επιπλέον, έχει χρησιμοποιηθεί και στη διεθνή βιβλιογραφία (McLoughlin, [2017](#)) και ο πηγαίος κώδικας σε Verilog είναι διαθέσιμος υπο την άδεια GPL<sup>1</sup>.

<sup>1</sup> <http://www.cs.hiroshima-u.ac.jp/~nakano/wiki>



Μια άλλη, πιο σύγχρονη υλοποίηση αρχιτεκτονικής στοίβας είναι η MINICPU (Στουρνάρας, 2022) η οποία σχεδιάστηκε για την υλοποίηση εφαρμογής γραφικών σε FPGA με χρήση παράλληλων CPUs και διαμοιραζόμενη μνήμη.

### 3.3. Συνηθέστερες συμβολικές εντολές σε βιβλιοθήκες Windows & Linux

Για τις ανάγκες δημιουργίας ενός εργαστηρίου εισαγωγής στο Reverse Engineering και την Assembly, οι Coldwind & Murray (2024) κατέγραψαν τη συχνότητα εμφάνισης των 100 πιο συνηθισμένων συμβολικών εντολών σε βιβλιοθήκες των Windows και Ubuntu. Στον Πίνακα 7 (σ. 24) καταγράφονται οι πρώτες 20. Στο σύνολό τους οι 20 αυτές εντολές είναι παραπλήσιες όσων έχουν καταγραφεί στον Πίνακα 1 (σ. 19) στην εισαγωγή αυτού του κεφαλαίου.

Εντολή	Συχνότητα	Περιγραφή
MOV	35.14%	Μεταφορά δεδομένων
CALL	7.97%	Κλήση υποπρογράμματος
LEA	6.83%	Φόρτωση διεύθυνσης με υπολογισμούς
CMP	4.98%	Σύγκριση
JZ	4.15%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
TEST	4.04%	Σύγκριση με AND
POP	4.03%	Απόθεση
JMP	4.00%	Άλμα χωρίς συνθήκη
PUSH	3.98%	Ωθηση
ADD	3.07%	Πρόσθεση
JNZ	2.79%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
XOR	2.51%	XOR
SUB	1.76%	Αφαίρεση
RET	1.15%	Επιστροφή απο υπορουτίνα
MOVZX	1.02%	Μεταφορά σε ευρύτερο καταχωρητή
AND	0.91%	AND
NOP	0.82%	Καμία λειτουργία
MOVUPS	0.74%	Μεταφορά μη-ευθυγραμμισμένων δεδομένων
ENDBR64	0.67%	Σήμανση κώδικα για προστασία απο ανασφαλείς διακλαδώσεις
MOVAPS	0.61%	Μεταφορά ευθυγραμμισμένων δεδομένων

Πίνακας 7: Οι συχνότερες εντολές σε βιβλιοθήκες Windows & Linux

Η λίστα βασίζεται σε στατική ανάλυση και όχι στην πραγματική εκτέλεση, άρα “συχνά εμφανιζόμενες” δεν σημαίνει υποχρεωτικά και “συχνά εκτελούμενες”. Το θέμα αυτό απασχόλησε στην συγγραφή της παρούσας εργασίας ως προς την επεξεργασία των πρωτογενών δεδομένων και έγινε προσπάθεια να αντιμετωπιστεί καταμετρώντας τις εντολές ως προς το πλήθος των παραδειγμάτων ή ασκήσεων στις οποίες εμφανίζονταν, και όχι ως προς το πλήθος των εμφανίσεων τους στα επιμέρους παραδείγματα κώδικα (βλ. σ. 146).

Στο σημείο αυτό αξίζει να σημειωθεί ότι στον πλήρη πίνακα των 100 εντολών από τους Coldwin & Murray, οι εντολές πολλαπλασιασμού και διαίρεσης καταγράφηκαν ανάμεσα στις λιγότερο συχνές (0.02%), ακόμα πιο κάτω και από την πρόσθεση με κρατούμενο (0.03%).

### 3.4. Σύνολο εντολών

Για να εξυπηρετήσει τον στόχο της **Χρησιμότητας** όπως έχει περιγραφεί στην εισαγωγή της εργασίας, ο E80 θα πρέπει να μπορεί να χρησιμοποιηθεί σε εισαγωγικά μαθήματα Assembly και επομένως θα πρέπει να καλύπτει τις σημαντικότερες από τις λειτουργίες που περιγράφονται στον Πίνακα 1 (σ. 19).

Παράλληλα όμως, για τον ίδιο στόχο της **Χρησιμότητας** αλλά και για τον στόχο της **Υλοποίησης**, το σύνολο θα πρέπει να είναι όσο το δυνατόν μικρό:

- Σύμφωνα με τους Coulter & Kelly (1986), οι περιορισμένες επιλογές απλοποιούν τη λήψη αποφάσεων και μειώνουν τον γνωστικό φόρτο.
- Μικρό σύνολο εντολών ισοδυναμεί με χαμηλές απαιτήσεις στην σύνθεση, άρα επιτρέπει την υλοποίηση σε FPGA χαμηλού κόστους.
- Μικρό μέγεθος συνόλου εντολών αντιστοιχεί σε μικρό μήκος εντολής και συνεπώς σε πιο ευανάγνωστο κώδικα μηχανής σε δυαδική ή δεκαεξαδική μορφή.

Τέλος για την κάλυψη του στόχου της **Συνέχειας**, οι εντολές θα πρέπει να μπορούν να υλοποιηθούν με την ύλη της ψηφιακής σχεδίασης. Όπως έχει τονιστεί στην Ενότητα 2.3, η VHDL δίνει την δυνατότητα αφαιρετικής υλοποίησης πολύπλοκων λειτουργιών μέσω των βιβλιοθηκών της. Για παράδειγμα, χρησιμοποιώντας την βιβλιοθήκη IEEE.NUMERIC\_STD η υλοποίηση των βασικών πράξεων γίνεται σε 4 γραμμές:

```
S <= A + B;  
D <= A - B;  
P <= A * B;  
Q <= A / B;
```

Μια τέτοια μινιμαλιστική υλοποίηση (όπου η πραγματικότητα κρύβεται κάτω από το “χαλί” της βιβλιοθήκης) θέτει το ερώτημα: “και ποιός ήταν ο σκοπός της εκμάθησης ψηφιακής σχεδίασης;”. Αυτό το ερώτημα θα οδηγήσει στην ουσιαστική διαφορά του E80 από τις υλοποιήσεις που παρουσιάστηκαν στην Ενότητα 3.2: ο σχεδιασμός του θα γίνει αποκλειστικά με χρήση της θεωρίας βασικών εννοιών ψηφιακής σχεδίασης του ΕΑΠ με επιπλέον στοιχεία από Βέργο (2007) χωρίς χρήση αριθμητικών βιβλιοθηκών.

### 3.4.1. Λειτουργίες που δεν θα υποστηριχθούν

Αρχικά θα πρέπει να προσδιοριστούν οι λειτουργίες που μπορούν να υποκατασταθούν είτε άμεσα, είτε πολύ εύκολα απο άλλες υπάρχουσες. Συγκεκριμένα, οι παρακάτω λειτουργίες κρίνονται ως πλεονάζουσες:

- Αύξηση κατά 1· γίνεται με πρόσθεση άμεσης τιμής.
- Μείωση κατά 1· γίνεται με αφαίρεση άμεσης τιμής.
- Εκκαθάριση καταχωρητή· γίνεται με μεταφορά άμεσης τιμής.
- Αριστερή περιστροφή· γίνεται με συμμετρικό πλήθος δεξιών περιστροφών.
- Πρόσθεση με κρατούμενο· γίνεται με άλμα αν δεν υπάρχει κρατούμενο.
- Αύξηση ζεύγους καταχωρητών κατά 1· γίνεται με άλμα αν δεν υπάρχει κρατούμενο.
- Ανταλλαγή τιμής· γίνεται με επιπλέον καταχωρητή ή ώθηση/απόθεση απο στοίβα.
- Άλμα αν μεγαλύτερο ή ίσο· υλοποιείται με έλεγχο προσήμου ή κρατουμένου.
- NAND / NOR· υλοποιούνται με XOR επι του AND / OR.

Οι λειτουργίες που είναι στενά συνδεδεμένες με αρχιτεκτονικές του εμπορίου θεωρούνται ανεπιθύμητες γιατί προκαλούν σύγχυση όταν είναι εκτός του πλαισίου της αρχιτεκτονικής που αιτιολογεί την ύπαρξή τους.

- Φόρτωση διεύθυνσης με υπολογισμούς (LEA)  
Βασίζεται σε συγκεκριμένα μοντέλα διευθυνσιοδότησης και δεν προσφέρει εκπαιδευτικό όφελος χωρίς γνώση της εσωτερικής οργάνωσης μνήμης του επεξεργαστή.
- Φόρτωση διεύθυνσης σε ζεύγος καταχωρητών (LXI)  
Απαιτεί ζεύγη καταχωρητών τα οποία δεν μπορούν να αιτιολογηθούν σε μια απλή αρχιτεκτονική όπου λέξη και διεύθυνση έχουν ίδιο μήκος.
- Αποθήκευση μέσω ζεύγους καταχωρητών (STAX)  
Όπως στην LXI.
- Φόρτωση μέσω ζεύγους καταχωρητών (LDAX)  
Όπως στην LXI.

Τέλος, οι λειτουργίες πολλαπλασιασμού και διαίρεσης είναι μεν επιθυμητές καθώς χρησιμοποιούνται στην εκπαίδευση, αλλά δεν μπορούν να υλοποιηθούν με τις βασικές γνώσεις Ψηφιακής Σχεδίασης του ΕΑΠ. Οι λειτουργίες αυτές μπορούν να υλοποιηθούν αποδοτικά με Assembly όπως παρουσιάζεται στο Κεφάλαιο 10 (σ. 128).

### 3.4.2. Λειτουργίες που θα υποστηριχθούν

Αφαιρώντας τις λειτουργίες που δεν θα υποστηριχθούν, απο όσες έχουν καταγραφεί στον Πίνακα 1 (σ. 19) στην εισαγωγή του κεφαλαίου, απομένουν αυτές που θα πρέπει να υποστηριχθούν με την διαμόρφωση της κατάλληλης αρχιτεκτονικής:

- Μεταφορά
- Πρόσθεση
- Αποθήκευση στην μνήμη
- Επιστροφή απο υπορουτίνα
- Φόρτωση απο μνήμη ή άμεσης τιμής
- Αφαίρεση
- Φόρτωση άμεσης τιμής
- Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
- Φόρτωση απο μνήμη
- Άλμα αν ισότητα / μηδενικό αποτέλεσμα
- Παύση εκτέλεσης
- Ώθηση
- Άλμα χωρίς συνθήκη
- Απώθηση
- Σύγκριση
- AND
- Κλήση υπορουτίνας
- XOR
- Πρόσθεση άμεσης τιμής
- Αριστερή ολίσθηση
- Καμία λειτουργία
- OR
- Σύγκριση με AND
- Δεξιά ολίσθηση
- Άλμα αν δεν υπάρχει κρατούμενο
- Άλμα αν υπάρχει κρατούμενο
- Σύγκριση με άμεση τιμή
- Αφαίρεση άμεσης τιμής
- Δεξιά περιστροφή

Για την υποστήριξη της λειτουργίας “Άλμα αν μεγαλύτερο ή ίσο” στην σύγκριση προσημασμένων αριθμών θα υλοποιηθούν και οι παρακάτω λειτουργίες:

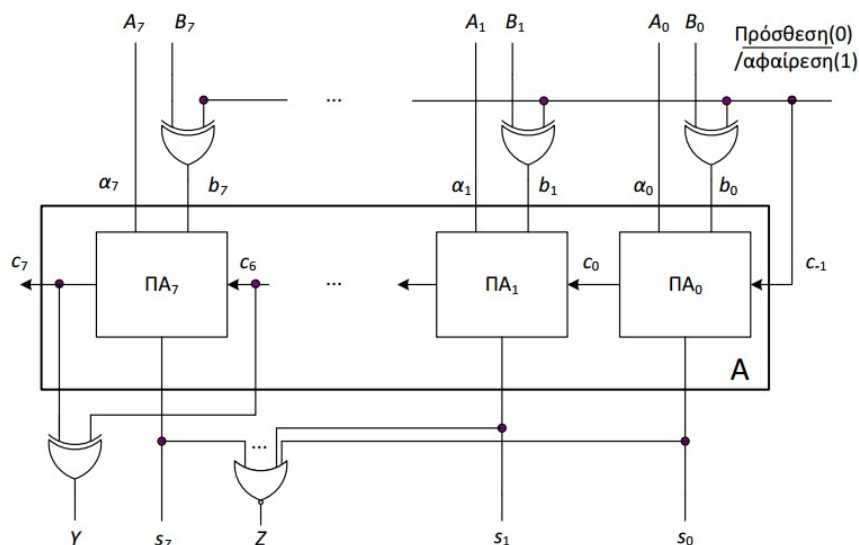
- Άλμα αν υπάρχει αρνητικό πρόσημο
- Άλμα αν δεν υπάρχει αρνητικό πρόσημο
- Άλμα αν υπάρχει υπερχειλίση
- Άλμα αν δεν υπάρχει υπερχειλίση

### 3.5. Σημαίες κατάστασης: Ρόλος και Σημασία στη Σχεδίαση του E80

Οι περισσότερες απο τις οκτώ προτεινόμενες λειτουργίες άλματος βάσει σημαίας (μηδενικού, κρατούμενου, προσήμου και υπερχειλίσης) δεν συναντώνται στις υλοποιήσεις απλών CPU. Η απόφαση να συμπεριληφθούν στον E80 έχει να κάνει με την κατανόηση της αναπαράστασης προσημασμένων ή μη αριθμών. Συγκεκριμένα, στις πολυπληθείς φροντιστηριακές διαλέξεις της ΠΛΗ21 στον Μάρτιο του 2023 είχε συζητηθεί εκτενώς το παρακάτω θέμα:

Χρησιμοποιώντας τη μονάδα του σχήματος που ακολουθεί να εκτελέσετε τις πράξεις  $\alpha+\beta$ ,  $\alpha+\gamma$ ,  $\alpha-\beta$  και  $\alpha-\delta$  και να δώσετε τις τιμές των σημαίων C ( $C_7$ , κρατούμενο εξόδου), Y= $C_7 \text{ XOR } C_6$  και Z (Zero, μηδενικού αποτελέσματος).

$\alpha = 11110001$   
 $\beta = 10000001$   
 $\gamma = 01000000$   
 $\delta = 11111100$



Για κάθε μία των περιπτώσεων A και B:

- Να αιτιολογήσετε εάν το αποτέλεσμα που λαμβάνουμε στις εξόδους  $S_7, \dots, S_1, S_0$  είναι σωστό ή όχι και να αναφέρετε τη σημαία στην τιμή της οποίας βασιστήκατε για να βγάλετε το συμπέρασμά σας.
- Να εκτελέσετε τις αντίστοιχες πράξεις στο δεκαδικό και να επιβεβαιώσετε την ορθότητα των συμπερασμάτων που εξαγάγατε στην περίπτωση i.

A. Οι αριθμοί  $\alpha$ ,  $\beta$ ,  $\gamma$  και  $\delta$  είναι προσημασμένοι σε αναπαράσταση συμπληρώματος ως προς 2.

B. Οι αριθμοί  $\alpha$ ,  $\beta$ ,  $\gamma$  και  $\delta$  είναι μη προσημασμένοι.

Εικόνα 9: Άσκηση προσθαφάιρησης αριθμών (ΕΑΠ, ΠΛΗ21)

Απο την παρακολούθηση στο ακροατήριο των φοιτητών έγινε αντιληπτό ότι υπήρχανε οι εξής προβληματισμοί και απορίες:

- “Πως νοείται η αφαίρεση απροσήμων αριθμών αφού ισχύει  $x-y = x+(-y)$  και επομένως ο  $y$  μπορεί να θεωρείται προσημασμένος;”.
- “Πως ελέγχεται η υπερχειλίση στους απρόσημους αριθμούς;”.

Αυτό που παρατηρήθηκε ήτανε ότι ενώ στους προσημασμένους αριθμούς η υπερχειλίση προκύπτει εύκολα απο το σήμα Y ( $C_7 \text{ XOR } C_6$ ), για τους απρόσημους κρίθηκε ότι η υπερχειλίση προκύπτει απο το τελικό κρατούμενο  $C_7$ . Αυτό όμως δεν ισχύει στην αφαίρεση. Για παράδειγμα η αφαίρεση των  $0001 - 0110$  μέσω συμπληρώματος ως προς 2 προκύπτει ως  $0001 + 1010 = 1011$  χωρίς κρατούμενο. Στο σημείο αυτό ο φοιτητής απαντούσε εσφαλμένα ότι δεν υπάρχει υπερχειλίση ή ότι το αποτέλεσμα είναι αρνητικό. Για την αντιμετώπιση της σύγχυσης στην διάλεξη, τονίστηκε επανειλημμένα ότι η υπερχειλίση στην αφαίρεση απροσήμων εκφράζεται απο την **μη**-ύπαρξη κρατούμενου.

Οι δυσκολίες αυτές έχουν προσδιοριστεί από την έρευνα των Herman, Zilles, & Loui (2011, σ. 301) που κατέδειξαν την αδυναμία φοιτητών που έχουν περάσει ενότητες ψηφιακής σχεδίασης να αντιληφθούν τα θέματα αναπαράστασης αριθμών και υπερχειλίσης.

Η χρήση της VHDL στο σημείο αυτό μπορεί να περιπλέξει την κατάσταση. Συγκεκριμένα, ένας απλός τρόπος που προτείνεται<sup>1</sup> για την προσθαφαίρεση αριθμών με κρατούμενο είναι ο εξής:

```
(c, o) <= std_logic_vector(unsigned('0' & a) - unsigned('0' & b));
```

Στον υπολογισμό αυτό γίνεται επέκταση των όρων της αφαίρεσης έτσι ώστε στο επεκτεταμένο ψηφίο να υπολογιστεί το κρατούμενο. Όμως επειδή το συμπλήρωμα ως προς 2 του αφαιρέτη συμπεριλαμβάνει και την επέκταση, το κρατούμενο πλέον δεν αντιστοιχεί με τον τρόπο που υπολογίζεται από τον απλό αθροιστή/αφαιρέτη της Εικόνας 9 (σ. 28).

Στο παράδειγμα του 0001 - 0110 ή επέκταση θα οδηγούσε σε 00001 - 00110, η συμπλήρωση ως προς 2 σε 00001 + 11010 = 11011 και επομένως στην ανάθεση στο ζεύγος (c, o) το κρατούμενο c θα είχε τιμή 1. Άρα με την μέθοδο αυτή, το κρατούμενο στην προσθαφαίρεση απροσήμων πάντα συμβαδίζει με την υπερχειλίση. Αυτό δίνει ένα φαινομενικό πλεονέκτημα και απλοποιεί την λογική των αλμάτων σύγκρισης στην προσθαφαίρεση κρατουμένου.

Σύμφωνα όμως με τον στόχο της **Συνέχειας** με την ύλη της ψηφιακής σχεδίασης, το πλεονέκτημα αυτό δεν ισχύει γιατί δημιουργεί σύγχυση σε ένα θέμα που είναι ήδη δύσκολο όπως καταγράφηκε προηγουμένως.

Απο τα παραπάνω προκύπτουν τα εξής συμπεράσματα:

1. Οι σημαίες κατάστασης θα πρέπει να υποστηριχθούν ώστε να βοηθήσουν στην κατανόηση πράξεων προσημασμένων και απροσήμων αριθμών.
2. Οι σημαίες κατάστασης θα πρέπει να συμβαδίζουν με την θεωρία ψηφιακής σχεδίασης όπως παρουσιάζονται στην Εικόνα 9 και στα διδακτικά εγχειρίδια (πχ. Λιναρδής, 2008, σ. 276).

### 3.6. Σημαία Αλτ

Κατα την δοκιμή της CPU κρίθηκε ότι η αδυναμία πρόβλεψης του χρόνου εκτέλεσης ενός προγράμματος Assembly δυσκόλευε την προσομοίωσή του. Στο ModelSim αυτό δεν είναι ιδιαίτερο πρόβλημα καθώς δίνει την δυνατότητα βηματικής προσομοίωσης, αλλά στο GHDL οι κυματομορφές εξάγονται σε αρχείο το οποίο μετά διαβάζεται από το GTKWave. Μπορούσε βεβαίως να οριστεί ένα μεγάλο χρονικό διάστημα που να καλύπτει κάθε λογικό σενάριο εκτέλεσης, αλλά αυτό δυσκόλευε την ανάγνωση σύντομων εκτελέσεων και απαιτούσε χειρισμούς zoom για την στόχευση στον χρόνο της εκτέλεσης.

Τα προβλήματα αυτά λύθηκαν με την υλοποίηση εντολής HLT η οποία αφενός έδωσε την δυνατότητα της οπτικοποίησης του τερματισμού εκτέλεσης στην FPGA, και αφετέρου χρησι-

<sup>1</sup> <https://stackoverflow.com/questions/54933059/how-can-i-check-for-carry-out-while-using-unsigned-vector-subtraction>



μπουήθηκε στα GHDL και ModelSim για να τερματίσει την προσομοίωση, επιτρέποντας στα ίδια τα προγράμματα να καθορίσουν το απαιτούμενο χρονικό διάστημά τους.

### 3.7. Σύνταξη εντολών

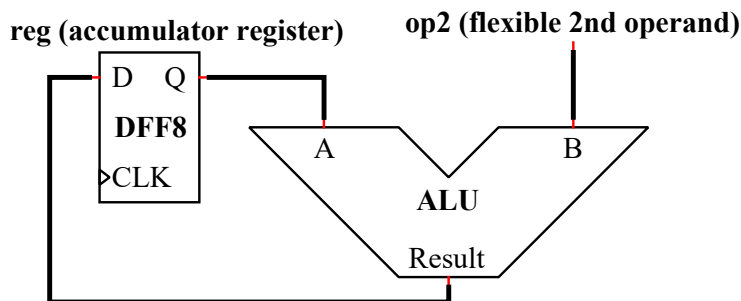
*“As an aside, this program is really for the Core i7, except that we have renamed the instructions and registers and changed the notation to make it easy to read because the Core i7’s standard assembly-language syntax (MASM) verges on the bizarre, a remnant of the machine’s former life as an 8088”* -- Tanenbaum (2013, σ. 374)

Η ρήση του Tanenbaum έχει εξέχουσα σημασία στην εργασία αυτή. Η σύνταξη της γλώσσας μιας μηχανής που έχει κατασκευαστεί για εκπαιδευτικούς σκοπούς έχει το πλεονέκτημα ότι δεν επιβαρύνεται από το φορτίο άλλων στόχων όπως η ταχύτητα και η συμβατότητα. Η αρχιτεκτονική του E80 θα πρέπει επομένως να σχεδιαστεί ώστε να επιτρέπει την εύκολη υλοποίηση ευανάγνωστου κώδικα.

Ως προς το επίπεδο εντολών γλώσσας μηχανής, ο E80 θα χρησιμοποιήσει αρχιτεκτονική καταχωρητών γενικού σκοπού, καθώς αυτό ισχύει πρακτικά σε όλους τους σύγχρονους υπολογιστές (Νικολός, 2008, σ. 79). Ταυτόχρονα θα χρησιμοποιηθεί αρχιτεκτονική Load/Store όπως έχει περιγραφεί στην Ενότητα 2.2.6 (σ. 7). Η συγκεκριμένη αρχιτεκτονική αποτελεί σήμα κατατεθέν των επεξεργαστών RISC, αλλά ως προς τους εκπαιδευτικούς σκοπούς του E80, η αποκλειστική χρήση διευθυνσιοδότησης για προσπέλαση μνήμης σε δύο εντολές απλοποιεί σημαντικά την σύνταξη της γλώσσας.

Ως προς την μορφή των εντολών, αυτή θα επιλεγθεί έτσι ώστε να επιτρέπει την υλοποίηση με μια απλή αρχιτεκτονική. Η σύνταξη με δύο τελούμενα της μορφής “**εντολή reg, op2**” που υλοποιείται σύμφωνα με την Εικόνα 10 έχει ακριβώς αυτό το χαρακτηριστικό. Το op2 που εκφράζει ένα “ευέλικτο 2ο τελούμενο” είναι ένας όρος που χρησιμοποιείται στην Assembly του ARM<sup>1</sup>. Στην απλοποιημένη προσέγγιση του E80, το op2 θα μπορεί να είναι:

- μια άμεση τιμή,
- ένας καταχωρητής ή μια κατευθείαν διεύθυνση μνήμης,
- μια έμμεση διεύθυνση από καταχωρητή στις περιπτώσεις των LOAD/STORE.



Εικόνα 10: Αντιστοίχιση της μορφής “Εντολή reg, op2” με στοιχειώδη δίοδο δεδομένων

Η σύνταξη των εντολών με ευέλικτο τελούμενο μειώνει το απαιτούμενο γνωστικό φορτίο. Ο εκπαιδευόμενος δεν χρειάζεται να αποστηθίζει τις ξεχωριστές εκδοχές των εντολών ανάλογα

<sup>1</sup> <https://developer.arm.com/documentation/dui0489/i/arm-and-thumb-instructions/flexible-second-operand--operand2->



με την διευθυνσιοδότηση (πχ. ADD/ADDI) καθώς το έργο αυτό θα ανατεθεί στον συμβολο-μεταφραστή μέσω μιας μη-διφορούμενης ασυμφραστικής γραμματικής.

Οι σχεδιαστικές επιλογές ως προς το τελικό σύνολο εντολών καταγράφονται στον Πίνακα 8 που ακολουθεί. Το n εκφράζει μια άμεση τιμή ή διεύθυνση, ενώ το op1 στην εντολή JMP μπορεί να είναι διεύθυνση άλματος ή καταχωρητής που παραπέμπει σε αυτήν.

Εντολή	Περιγραφή
LOAD reg, op2	Φόρτωση στον reg απο άμεση ή έμμεση θέση μνήμης op2
STORE reg, op2	Αποθήκευση του reg στην άμεση ή έμμεση θέση μνήμης op2
MOV reg, op2	Μεταφορά
ADD reg, op2	Πρόσθεση
SUB reg, op2	Αφαίρεση
CMP reg, op2	Σύγκριση
AND reg, op2	AND
OR reg, op2	OR
XOR reg, op2	XOR
BIT reg, n	Σύγκριση με AND
LSHIFT reg	Αριστερή ολίσθηση
RSHIFT reg	Δεξιά ολίσθηση
ROR reg, op2	Περιστροφή δεξιά κατα op2 bits
NOP	Καμία λειτουργία
HLT	Παύση εκτέλεσης
JMP op1	Άλμα στην διεύθυνση op1
JC n	Άλμα αν υπάρχει κρατούμενο
JNC n	Άλμα αν δεν υπάρχει κρατούμενο
JZ n	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
JNZ n	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
JS n	Άλμα αν υπάρχει αρνητικό πρόσημο
JNS n	Άλμα αν δεν υπάρχει αρνητικό πρόσημο
JV n	Άλμα αν υπάρχει υπερχειλίση
JNV n	Άλμα αν δεν υπάρχει υπερχειλίση
CALL n	Κλήση υπορουτίνας
RETURN	Επιστροφή απο υπορουτίνα
PUSH reg	Ωθηση
POP reg	Απώθηση

Πίνακας 8: Το σύνολο εντολών του E80 με ευέλικτο 2ο τελούμενο

### 3.8. Μνήμη, καταχωρητές, μέγεθος λέξης και μορφή συνόλου εντολών

Ως επεξεργαστής για εκπαιδευτικούς σκοπούς, ο E80 δεν χρειάζεται πολύ μνήμη, δεδομένου ότι τα προγράμματα που εμφανίζονται στις ασκήσεις των πηγών που διερευνήθηκαν στο Παράρτημα Α (σ. 146) αρκούνται σε λίγες γραμμές. Επίσης στην πλειοψηφία των ασκήσεων από τις πηγές αυτές, και ειδικά στις ασκήσεις Μικροϋπολογιστών του ΕΑΠ ([2013β](#)) οι τιμές των καταχωρητών περιορίζονται σε 1 byte.

Για την διευκόλυνση της συγγραφής προγραμμάτων θα είναι χρήσιμο να συγκλίνει το μήκος της διεύθυνσης μνήμης με το μέγεθος της λέξης έτσι ώστε το ευέλικτο τελούμενο op2 να έχει την ίδια μορφή ανεξάρτητα με το αν είναι διεύθυνση ή άμεση τιμή σε υπολογισμό.

Επομένως το μήκος της λέξης θα οριστεί στα 8 bits και αντίστοιχα το εύρος της διεύθυνσης μνήμης θα οριστεί στα 8 bits, άρα θα υποστηρίξονται 256 λέξεις.

Για τους καταχωρητές θα πρέπει να γίνει επιλογή ανάμεσα στους 4 και τους 8. Περισσότεροι από 8 καταχωρητές θα επιτρέπουν την επίλυση ασκήσεων Assembly με τρόπο που δεν επιτυγχάνει τον εκπαιδευτικό τους στόχο, ενώ λιγότεροι καταχωρητές θα αυξάνουν τις απαιτήσεις για χρήση προσωρινής αποθήκευσης και θα κάνουν τα προγράμματα δυσανάγνωστα. Επειδή ένας καταχωρητής θα χρησιμοποιηθεί για δείκτη στοίβας και ένας καταχωρητής θα χρησιμοποιηθεί για αποθήκευση σημαιών, επιλέγονται οι 8 από τους οποίους οι 6 θα είναι γενικού σκοπού.

Για τον υπολογισμό του πλήθους των bits που απαιτούνται για την αναπαράσταση της κάθε εντολής θα πρέπει να υπολογιστεί το συνολικό πλήθος διαφορετικών συνδυασμών εντολών και τελουμένων με χρήση συνδυαστικής. Συγκεκριμένα, σε κάθε εντολή γίνεται κατανομή **κ** διακεκριμένων θέσεων bit σε 2 διακεκριμένες υποδοχές (0 ή 1) στις οποίες η σειρά τοποθέτησης δεν παίζει ρόλο. Άρα έχουμε διατάξεις με επανάληψη χωρίς σειρά. Από την μελέτη των εντολών του πίνακα 8 (σ. 31), προκύπτουν οι εξής τύποι σύνταξης:

- Τύπος 1: Εντολές χωρίς τελούμενα (πχ. NOP)  
Κάθε τέτοια εντολή έχει μία μόνο διάταξη.
- Τύπος 2: Εντολές με ένα τελούμενο καταχωρητή (πχ. LSHIFT reg)  
Εφόσον υπάρχουν 8 καταχωρητές, κάθε τέτοια εντολή έχει 8 διατάξεις.
- Τύπος 3: Εντολές με ένα τελούμενο άμεσης διεύθυνσης (πχ. JMP n)  
Ο διάυλος διευθύνσεων έχει 8 bits άρα κάθε τέτοια εντολή έχει 256 διατάξεις.
- Τύπος 4: Εντολές με δύο τελούμενα καταχωρητή-καταχωρητή (πχ. LOAD reg, reg)  
Σύμφωνα με την βασική αρχή της απαρίθμησης, υπάρχουν  $8 \times 8$  διατάξεις.
- Τύπος 5: Εντολές με καταχωρητή και άμεση διεύθυνση (πχ. LOAD reg, n)  
Σύμφωνα με την βασική αρχή της απαρίθμησης θα υπάρχουν  $8 \times 256$  επιλογές.

Οι υπολογισμοί που προκύπτουν καταγράφονται στον πίνακα 9 (σ. 33). Όπως σημειώθηκε στην προηγούμενη ενότητα, εντολές που δέχονται είτε καταχωρητή είτε άμεση τιμή θα πρέπει να αντιστοιχίζονται σε διαφορετικούς κωδικούς ανάλογα με το τελούμενο. Αυτό θα πρέπει να διασφαλιστεί **και** μέσα από την σύνταξη της Assembly έτσι ώστε να μην είναι διφορούμενη.

Εντολή	Διατάξεις τελουμένων				
	κανένα $2^0$	reg $2^3$	reg1,reg2 $2^3 \times 2^3$	word $2^8$	reg,word $2^3 \times 2^8$
HLT	1				
NOP	1				
JMP op1		8		256	
JC n				256	
JNC n				256	
JZ n				256	
JNZ n				256	
JS n				256	
JNS n				256	
JV n				256	
JNV n				256	
CALL n				256	
RETURN	1				
MOV reg, op2			64		2048
ADD reg, op2			64		2048
SUB reg, op2			64		2048
ROR reg, op2			64		2048
AND reg, op2			64		2048
OR reg, op2			64		2048
XOR reg, op2			64		2048
STORE reg, op2			64		2048
LOAD reg, op2			64		2048
RSHIFT reg		8			
CMP reg, op2			64		2048
LSHIFT reg		8			
BIT reg, n					2048
PUSH reg		8			
POP reg		8			
Σύνολο				25771	
log <sub>2</sub> (Σ)				14,65	

Πίνακας 9: Απαιτούμενα bits για την υποστήριξη των εντολών του E80

Απο το σύνολο των διατάξεων προκύπτουν τα απαιτούμενα bits ως  $\log_2(\Sigma) = 14.65$ . Συνεπώς θα πρέπει να χρησιμοποιηθούν το πολύ 2 λέξεις για την κωδικοποίηση των εντολών.

### 3.9. Στοίβα

Η σημασία της στοίβας τονίζεται απο τον Δασυγένη (2021) που την χρησιμοποιεί για υλοποίηση διαφανούς διαδικασιακού προγραμματισμού, και τους Patterson & Hennesey (2014, σ. 98-99) που δείχνουν την αναγκαιότητά της για την μεταγλώττιση μιας απλής συνάρτησης σε C σε Assembly. Αυτό δείχνει οτι η ύπαρξη στοίβας σε έναν επεξεργαστή είναι απαραίτητη για την επίτευξη των χαρακτηριστικών υψηλού ταβανιού και ευρέων τοίχων σε ένα Παπερτιανό μικρόκοσμο όπως έχουν παρουσιαστεί στην Ενότητα 2.7.1 (σ. 17).

Η υλοποίηση της στοίβας αποτέλεσε μια θεμελιώδη σχεδιαστική απόφαση για την αρχιτεκτονική του E80, καθώς καθόρισε σε μεγάλο βαθμό την δομή του ελέγχου και της διόδου δεδομένων. Σχεδιάζοντας αυτή τη δίοδο, ειδικά ως προς την POP, καλύφθηκαν όλες οι ανάγκες των υπολοίπων εντολών. Για την υλοποίηση επιλέχθηκε η οργάνωση πλήρους κατιούσας στοίβας σύμφωνα με τον σχολιασμό της Ενότητας 2.2.7 (σ. 8) στο Υπόβαθρο.

Ο ρόλος του Δείκτη Στοίβας (Stack Pointer - SP) ανατέθηκε στον καταχωρητή R7. Η επιλογή αυτή έγινε για πρακτικούς λόγους που αφορούν την αποσφαλμάτωση του συστήματος στην πλατφόρμα υλοποίησης. Συγκεκριμένα, κατά τη λειτουργία του επεξεργαστή στην FPGA, η επισκόπηση των τιμών των καταχωρητών γίνεται μέσω ενός υποστηριζόμενου joystick, όπου η κίνηση πάνω-κάτω εναλλάσσει κυκλικά την προβολή των καταχωρητών. Η τοποθέτηση του SP στον “τελευταίο” καταχωρητή R7 επιτρέπει στον χρήστη να μεταβεί άμεσα σε αυτόν με μία μόνο κίνηση προς τα κάτω από την αρχική κατάσταση (που δείχνει τον R0), διευκολύνοντας την ταχεία παρακολούθηση της κατάστασης της στοίβας. Αξίζει να σημειωθεί ότι η χρήση του τελευταίου διαθέσιμου καταχωρητή ως δείκτη στοίβας έχει ιστορικό προηγούμενο, όπως στον επεξεργαστή MIL-STD-1750A (Pesler, [1982](#)).

Ο δείκτης στοίβας αρχικοποιείται στην τιμή 0xFF το οποίο, λόγω πλήρους κατιούσας οργάνωσης, σημαίνει ότι η θέση αυτή δεν χρησιμοποιείται από την στοίβα. Αυτό έγινε για δυο λόγους:

Αφενός η θέση αυτή είναι διαθέσιμη για την είσοδο δεδομένων με διακόπτες DIP που περιγράφεται στην Ενότητα 3.10 που ακολουθεί. Αφετέρου, η άδεια στοίβα ισοδυναμεί με SP=11111111, η ώθηση ενός στοιχείου ισοδυναμεί με SP=11111110 δηλαδή συμπληρωμένο 1 κ.ο.κ. Δεδομένου ότι τα LED ανάβουν στο 1 και σβήνουν το 0, ο δείκτης στοίβας διαβάζεται εύκολα ως δυαδικός αριθμός με αντεστραμμένο φωτισμό.

Ταυτόχρονα η υποχείλιση που εκφράζεται από μηδενισμένο δείκτη στοίβας, γίνεται άμεσα αντιληπτή δεδομένου ότι τα LED του R7 έρχονται σε πλήρη αντίθεση με την κανονική λειτουργία καθώς σβήνουν όλα, παρέχοντας μια σαφή οπτική ένδειξη για τη δυσλειτουργία του προγράμματος.

### 3.10. Είσοδος δεδομένων με διακόπτες DIP

Για την πληρέστερη αναπαράσταση της αρχιτεκτονικής Neumann όπως έχει καταγραφεί στην Ενότητα 2.2.1 (σ. 5), ο σχεδιασμός του E80 επεκτάθηκε έτσι ώστε μαζί με την έξοδο αποτελεσμάτων σε LED, να συμπεριλάβει και οκτάμπιτη είσοδο μέσω διακοπών DIP από την πλακέτα. Αυτό επιτρέπει στον φοιτητή να παρατηρήσει ένα λειτουργικό υπολογιστικό σύστημα με πλήρη κύκλο επεξεργασίας: φυσική είσοδο δεδομένων, επεξεργασία από την CPU, και φυσική έξοδο αποτελεσμάτων.

Η ενσωμάτωση της δυνατότητας αυτής έγινε με χαρτογράφηση της διεύθυνσης 0xFF στον χώρο της μνήμης. Επομένως η είσοδος από τους διακόπτες DIP γίνεται φορτώνοντας το δεδομένο από την εν λόγω διεύθυνση σε ένα καταχωρητή με την εντολή LOAD reg, 0xFF.

## 4. Η αρχιτεκτονική εντολών του E80

### 4.1. Καταχωρητές

Όπως αποφασίστηκε στο προηγούμενο κεφάλαιο, ο E80 χρησιμοποιεί μια συστοιχία 8 καταχωρητών, εφεξής R0-R7 από τους οποίους οι R0-R5 είναι γενικού σκοπού, ο R6 είναι ο καταχωρητής σημαιών κατάστασης και ο R7 είναι ο δείκτης στοίβας. Ως προς τους R6 και R7 θα μπορούν να χρησιμοποιούνται εναλλακτικά και τα ονόματα FLAGS και SP. Το πλήθος των 8 καταχωρητών απαιτεί 3 bits διευθυνσιοδότησης.

### 4.2. Σημαίες

Οι σημαίες αποθηκεύονται στον καταχωρητή FLAGS (R6) εκπροσωπώντας τις ακόλουθες καταστάσεις:

7	6	5	4	3	2	1	0
+	+	+	+	+	+	+	+
	C		Z		S		V
+	+	+	+	+	+	+	+

C Carry / Κρατούμενο

Αποθηκεύει το κρατούμενο στις εντολές πρόσθεσης και αφαίρεσης. Στην περίπτωση της ολίσθησης, το ολισθέν bit αποθηκεύεται στο Κρατούμενο.

Z Zero / Μηδενικό

Ενεργοποιείται όταν ένας καταχωρητής μεταβάλλεται σε 0. Εξαίρεση θα αποτελέσουν οι πράξεις στοίβας οι οποίες δεν θα επηρεάζουν τις σημαίες.

S Sign / Πρόσημο

Αναπαριστά το πιο σημαντικό ψηφίο του αποτελέσματος μιας εντολής. Αν οι αριθμοί θεωρούνται προσημασμένοι, τότε S=1 εκφράζει αρνητικό αριθμό.

V Overflow / Υπερχείλιση

Εξάγεται από τον αθροιστή και εκφράζει υπερχειλίση σε πράξεις προσημασμένων αριθμών. Στις εντολές ολίσθησης, τίθεται V=1 όταν μεταβάλλεται το πιο σημαντικό ψηφίο του τελουμένου.

H Halt / Αλτ

Όταν τίθεται H=1, παύει εκτέλεση στην FPGA, και τερματίζεται η προσομοίωση σε GHDL και ModelSim.

Οι σημαίες ενημερώνονται σύμφωνα με έναν απλό κανόνα, εμπνευσμένο από τον 6502 (MOS Technology, [1976](#), σ. 25):

- Στις εντολές που δεν μεταβάλλουν καταχωρητή, οι σημαίες παραμένουν αμετάβλητες. Αυτό ισχύει και στις εντολές στοίβας οι οποίες χρησιμοποιούνται ενδιάμεσα σε λειτουργίες που βασίζονται σε αποτελέσματα προηγούμενων πράξεων.
- Στις υπόλοιπες εντολές, οι σημαίες Z και S ενημερώνονται πάντα, ενώ οι C και V ενημερώνονται στην προσθαφαίρεση και την ολίσθηση.

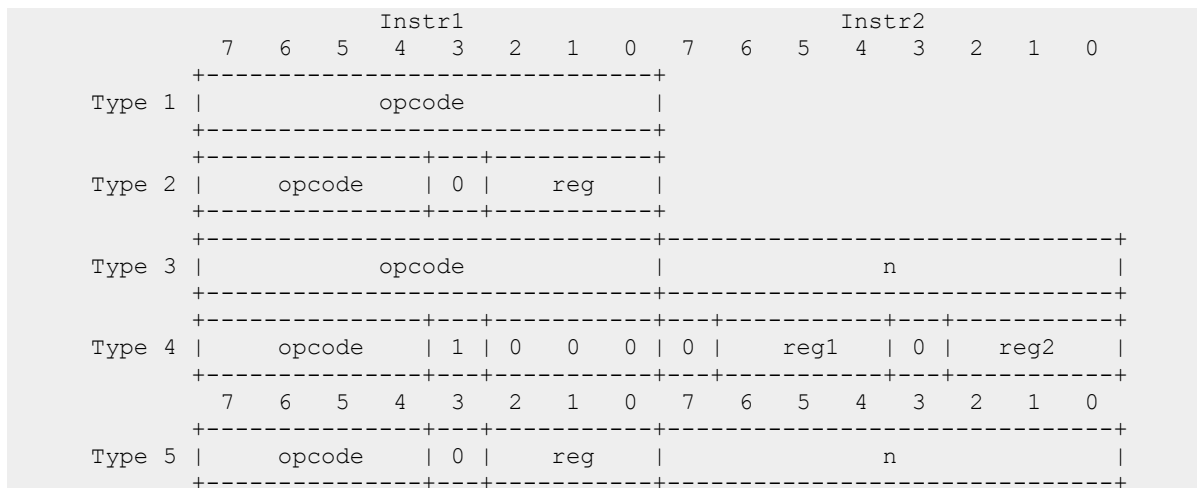
### 4.3. Κωδικοποίηση εντολών

Για την παρουσίαση της κωδικοποίησης των εντολών συνδυάστηκε η πρακτική προσέγγιση του Pessler ([1982](#)) με την πιο θεωρητική του Hayes ([1998](#), σ. 180).

Οι εντολές αποτελούνται από μία ή δύο λέξεις. Τα τμήματα αυτά εφεξής θα ονομάζονται Instr1 και Instr2.

Μετά από δοκιμαστικές υλοποιήσεις του E80, έγινε αντιληπτό ότι η σχεδίαση με μεταβλητό μήκος εντολής είχε μικρή επίπτωση στην πολυπλοκότητα, και επομένως επιλέχθηκε μεταβλητό μέγεθος 1 ή 2 bytes. Αυτό επιτρέπει την βέλτιστη αξιοποίηση της λιγοστής διαθέσιμης μνήμης του συστήματος αλλά και δίνει δυνατότητα εξάσκησης σε κώδικα με μεταβλητό μήκος εντολής (στόχοι **Υλοποίησης** και **Χρησιμότητας**).

Όπως φαίνεται παρακάτω, η κωδικοποίηση χρησιμοποιεί ένα σχήμα “Αναπτυσσόμενου Κωδικού Λειτουργίας” (Tanenbaum, [2012](#), σ. 366). Το bit Instr1[3] χρησιμοποιείται για να διακρίνει αν το ευέλικτο 2ο τελούμενο (op2) είναι καταχωρητής ή όχι. Στην περίπτωση που δεν υπάρχει 2ο τελούμενο, η τιμή του εν λόγω bit είναι 0 ή αδιάφορη.



Οι εντολές είναι κωδικοποιημένες με τέτοιο τρόπο ώστε το κάθε nibble να περιέχει είτε κωδικό λειτουργίας είτε διεύθυνση μνήμης, είτε διεύθυνση καταχωρητή. Για παράδειγμα:

- Η εντολή ADD R<sup>5</sup>,255 κωδικοποιείται ως 0x2<sup>5</sup>FF και το nibble 5 αντιστοιχεί στον καταχωρητή R5.
- Η εντολή ADD R<sup>3</sup>,R<sup>4</sup> κωδικοποιείται ως 0x28<sup>34</sup>. Το nibble 8 δεν μπορεί να είναι καταχωρητής· εκφράζει περίπτωση Instr1[3]=1 οπότε οι καταχωρητές εντοπίζονται στα nibble 3 και 4.

Το χαρακτηριστικό αυτό είναι δύσκολο να περιγραφεί στο κείμενο αλλά πιο εύκολο να φανεί στην πράξη μέσω του φυλλαδίου της αρχιτεκτονικής που παρουσιάζεται στην Ενότητα 4.11 (σ. 51). Το πρακτικό πλεονέκτημα είναι ότι η δομή της εντολής παραμένει ευανάγνωστη και στην δεκαεξαδική της αναπαράσταση, επιτρέποντας την χρήση “συμπυκνωμένων” δεκαεξαδικών στις κυματομορφές το οποίο με τη σειρά του επιτρέπει την επισκόπηση πολύ μεγαλύτερου πλήθους βημάτων εκτέλεσης, όπως θα φανεί στο Κεφάλαιο 8 (σ. 95).

## 4.4. Εντολές μεταφοράς δεδομένων

### 4.4.1. MOV - μεταφορά τιμών μεταξύ καταχωρητών

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	+	0	0	0	1	1	0	0	0	0	0	reg	+	0	+	op2
Type 5	+	0	0	0	1	0	+	reg	+	+	+	+	+	op2	+	+

**Μνημονικό:** MOV reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:** reg ← op2

Μεταφέρει στον reg την τιμή του op2.

**Σημαίες:** Μεταβάλλονται οι Z και S βάσει της νέας τιμής του reg.

**Παράδειγμα:** MOV R0, R1 μεταφέρει στον R0 την τιμή του R1.

MOV R2, 10 μεταφέρει στον R2 την τιμή 10.

### 4.4.2. LOAD - φόρτωση απο την μνήμη

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	+	1	0	0	1	1	0	0	0	0	0	reg	+	0	+	op2
Type 5	+	1	0	0	1	0	+	reg	+	+	+	+	+	op2	+	+

**Μνημονικό:** LOAD reg, [op2]

reg κατευθείαν διεύθυνση καταχωρητή

op2 έμμεση διεύθυνση μνήμης απο καταχωρητή (αν Instr1[3]=1)  
ή κατευθείαν διεύθυνση μνήμης (αν Instr1[3]=0)

**Λειτουργία:** reg ← [op2]

Φορτώνει στον reg την λέξη απο την διεύθυνση μνήμης op2.

**Σημαίες:** Μεταβάλλονται οι Z και S βάσει της νέας τιμής του reg.

**Παράδειγμα:** LOAD R0, [R1] φορτώνει στον R0 την λέξη απο την διεύθυνση μνήμης R1.

LOAD R2, [100] φορτώνει στον R2 την λέξη απο την διεύθυνση μνήμης 100.

LOAD R3, [255] φορτώνει στον R3 την είσοδο απο τους διακόπτες DIP.



#### 4.4.3. STORE - αποθήκευση στην μνήμη

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	1	0	0	0	1	0	0	0	0	reg			0		op2	
Type 5	1	0	0	0	0		reg						op2			

**Μνημονικό:** STORE reg, [op2]

reg κατευθείαν διεύθυνση καταχωρητή

op2 έμμεση διεύθυνση μνήμης απο καταχωρητή (αν Instr1[3]=1)  
ή κατευθείαν διεύθυνση μνήμης (αν Instr1[3]=0)

**Λειτουργία:** reg → [op2]

Αποθηκεύει την τιμή του reg στην θέση μνήμης op2.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** STORE R0, [R1] αποθηκεύει τον R0 στην διεύθυνση μνήμης R1.

STORE R2, [100] αποθηκεύει τον R2 στην διεύθυνση μνήμης 100.

#### 4.5. Αριθμητικές εντολές

##### 4.5.1. ADD - πρόσθεση

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	0	0	1	0	1	0	0	0	0	reg			0		op2	
Type 5	0	0	1	0	0		reg						op2			

**Μνημονικό:** ADD reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:** reg ← reg + op2

Προσθέτει στον reg τον op2.

**Σημαίες:** Μεταβάλλονται όλες οι σημαίες, C, Z, S, V σύμφωνα με την λειτουργία του οκτάμπιτου αθροιστή/αφαιρέτη (σ. 55). Ειδικά για τις σημαίες κρατούμενου και υπερχείλισης, δίνεται ο παρακάτω συνοπτικός πίνακας για ADD a, b:

Σημαία	Προσημασμένοι	Απρόσημοι
C=1		a+b > 255 (υπερχείλιση)
C=0		a+b ≤ 255
V=1	a+b ∉ [-128,127] (υπερχείλιση)	
V=0	a+b ∈ [-128,127]	

**Παράδειγμα:** ADD R0, R1 προσθέτει στον R0 τον R1

ADD R2, 0x64 προσθέτει στον R2 το 100

#### 4.5.2. SUB - αφαίρεση

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	0	0	1	1	1	0	0	0	0	reg			0		op2	
Type 5	0	0	1	1	0		reg						op2			

**Μνημονικό:** SUB reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:**  $reg \leftarrow reg + (\sim op2) + 1$

Αφαιρεί από τον reg τον op2 χρησιμοποιώντας πρόσθεση με συμπλήρωμα ως προς 2.

**Σημαίες:** Μεταβάλλονται όλες οι σημαίες, C, Z, S, V σύμφωνα με την λειτουργία του οκτάμπιτου αθροιστή/αφαιρέτη (σ. 55). Ειδικά για τις σημαίες κρατούμενου και υπερχείλισης, δίνεται ο παρακάτω συνοπτικός πίνακας για SUB a, b:

Σημαία	Προσημασμένοι	Απρόσημοι
C=1		$a \geq b$
C=0		$a < b$ (υπερχείλιση)
V=1	$a-b \notin [-128, 127]$ (υπερχείλιση)	
V=0	$a-b \in [-128, 127]$	

**Παράδειγμα:** SUB R0, R1 αφαιρεί από τον R0 τον R1.

SUB R2, 0b01100010 αφαιρεί από τον R2 το 100.

#### 4.5.3. CMP - σύγκριση με αφαίρεση

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	1	0	1	1	1	0	0	0	0	reg			0		op2	
Type 5	1	0	1	1	0		reg						op2			

**Μνημονικό:** CMP reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:** Λειτουργεί όπως η SUB αλλά δεν μεταβάλλει τον reg.

Χρησιμοποιείται για να μεταβάλλει σημαίες για έλεγχο ροής εκτέλεσης.

**Σημαίες:** Όπως στην SUB.

**Παράδειγμα:** CMP R0, R1 εκτελεί SUB R0, R1 χωρίς να μεταβάλλει τον R0.

CMP R2, 100 εκτελεί SUB R2, 100 χωρίς να μεταβάλλει τον R2.

## 4.6. Εντολές λογικών πράξεων

### 4.6.1. AND - σύζευξη

		Instr1									Instr2																	
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0											
		+-----+																										
Type 4			0	1	0	1		1		0	0	0	0		0		reg					0		op2				
		+-----+																										
		+-----+																										
Type 5			0	1	0	1		0		reg					op2													
		+-----+																										
		+-----+																										

**Μνημονικό:** AND reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:**  $reg \leftarrow reg \& n$

Καταχωρεί στον reg την σύζευξη των bits του reg και op2.

**Σημαίες:** Μεταβάλλονται οι Z και S βάσει της νέας τιμής του reg.

**Παράδειγμα:** AND R0, R1 καταχωρεί στον R0 την σύζευξη των bits του με αυτά του R1.

AND R2, 0x0F μηδενίζει το αριστερό nibble του R2.

### 4.6.2. BIT - σύγκριση με σύζευξη

Instr1												Instr2											
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0							
Type 5	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----							
		1	1	0	1		0		reg		n												
	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----							

**Μνημονικό:** BIT reg, n

reg κατευθείαν διεύθυνση καταχωρητή

n άμεση τιμή

**Λειτουργία:** Λειτουργεί όπως η AND αλλά δεν μεταβάλλει τον reg.

Χρησιμοποιείται για να μεταβάλλει σημαίες για έλεγχο ροής εκτέλεσης.

**Σημαίες:** Όπως στην AND.

**Παράδειγμα:** BIT R2, 1 θέτει Z=0 αν το ΛΣΨ του R2 είναι 1 (δηλ. αν ο R2 είναι περιττός).

#### 4.6.3. OR - διάζευξη

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	0	1	1	0	1	0	0	0	0	0	reg	0	op2			
Type 5	0	1	1	0	0	reg					op2					

**Μνημονικό:** OR reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:**  $reg \leftarrow reg \mid op2$

Καταχωρεί στον reg την διάζευξη των bits του reg και op2.

**Σημείες:** Μεταβάλλονται οι Z και S βάσει της νέας τιμής του reg.

**Παράδειγμα:** OR R0, R1 καταχωρεί στον R0 την διάζευξη των bits του με αυτά του R1.  
OR R2, 1 θέτει 1 στο ΛΣΨ του R2.

#### 4.6.4. XOR - αποκλειστική διάζευξη

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	0	1	1	1	1	0	0	0	0	0	reg	0	op2			
Type 5	0	1	1	1	0	reg					op2					

**Μνημονικό:** XOR reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:**  $reg \leftarrow reg \wedge op2$

Καταχωρεί στον reg την αποκλειστική διάζευξη των bits του reg και op2.

**Σημείες:** Μεταβάλλονται οι σημείες Z και S βάσει της νέας τιμής του reg.

**Παράδειγμα:** XOR R0, R1 καταχωρεί στον R0 την αποκλειστική διάζευξη των bits του με αυτά του R1.  
XOR R2, 0xF0 αντιστρέφει τα bits στο αριστερό nibble του R2.  
XOR R3, 0xFF συμπληρώνει (NOT) τον R3.

Όπως φαίνεται στο παραπάνω παράδειγμα, η XOR μπορεί να αντικαταστήσει την NOT.

## 4.7. Εντολές ελέγχου ροής

### 4.7.1. JMP - άλμα χωρίς συνθήκη

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3	0	0	0	0	0	0	1	0	op1							
Type 4	0	0	0	0	0	0	1	1	0	0	0	0	0	0	op1	

**Μνημονικό:** JMP op1

op1 κατευθείαν διεύθυνση μνήμης (αν Instr1[0]=0)  
ή έμμεση διεύθυνση μνήμης απο καταχωρητή (αν Instr1[0]=1)

**Λειτουργία:** PC ← op1

Θέτει στον Program Counter την τιμή n και επομένως οδηγεί την εκτέλεση στην εντολή που βρίσκεται σε αυτή τη διεύθυνση.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** JMP 100 γίνεται άλμα στην διεύθυνση 100.

MOV R0, 90

JMP R0 γίνεται άλμα στην διεύθυνση 90.

Οι εντολές άλματος χρησιμοποιούνται συνήθως με ετικέτες, στις οποίες ο assembler αντιστοιχεί αυτόματα τις κατάλληλες διευθύνσεις. Για παράδειγμα, η εντολή JMP x κάνει άλμα στην διεύθυνση που αντιστοιχεί ο assembler στην ετικέτα x.

### 4.7.2. JC - άλμα αν υπάρχει κρατούμενο

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3	0	0	0	0	0	1	0	0	n							

**Μνημονικό:** JC n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν C=1 τότε PC ← n

Αν υπάρχει κρατούμενο, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** CMP R0, R1 αν R0 ≥ R1 (θεωρώντας τους απρόσημους)

JC x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή

#### 4.7.3. JNC - άλμα αν δεν υπάρχει κρατούμενο

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3		0	0	0	0	0	1	0	1	n							

**Μνημονικό:** JNC n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν C=0 τότε PC ← n

Αν δεν υπάρχει κρατούμενο, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** CMP R0, R1 αν R0 < R1 (θεωρώντας τους απρόσημους)

JNC x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή

#### 4.7.4. JZ - άλμα αν υπάρχει μηδενικό αποτέλεσμα

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3		0	0	0	0	0	1	1	0	n							

**Μνημονικό:** JZ n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν Z=1 τότε PC ← n

Αν υπάρχει μηδενικό αποτέλεσμα, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** CMP R0, R1 αν R0 = R1

JZ x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή

#### 4.7.5. JNZ - άλμα αν δεν υπάρχει μηδενικό αποτέλεσμα

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3		0	0	0	0	0	1	1	1	n							

**Μνημονικό:** JNZ n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν Z=0 τότε PC ← n

Αν υπάρχει μη-μηδενικό αποτέλεσμα, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** CMP R0, R1 αν R0 ≠ R1

JNZ x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή.



#### 4.7.6. JS - άλμα αν υπάρχει πρόσημο αρνητικού

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type	3	0	0	0	0	1	0	1	0	n							

**Μνημονικό:** JS n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν  $S=1$  τότε  $PC \leftarrow n$

Αν υπάρχει αρνητικό αποτέλεσμα, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** SUB R0, R1 αν  $R1 > R0$  (θεωρώντας τους προσημασμένους)

JS x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή.

#### 4.7.7. JNS - άλμα αν δεν υπάρχει πρόσημο αρνητικού

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type	3	0	0	0	0	1	0	1	1	n							

**Μνημονικό:** JNS n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν  $S=0$  τότε  $PC \leftarrow n$

Αν υπάρχει μη-αρνητικό αποτέλεσμα, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** OR R0, 0 αν R0 προσημασμένος και  $\geq 0$ , ή απρόσημος και  $< 128$

JNS x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή.

Όπως φαίνεται παραπάνω, οι σημαίες προσήμου και υπερχείλισης μπορούν να χρησιμοποιηθούν δημιουργικά και σε απρόσημους αριθμούς.

#### 4.7.8. JV - άλμα αν υπάρχει υπερχείλιση

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type	3	+-----+-----+-----+-----+								+-----+-----+-----+-----+							
		0	0	0	0	1	1	0	0	n							

**Μνημονικό:** JV n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν  $V=1$  τότε  $PC \leftarrow n$

Αν υπάρχει υπερχείλιση σε πράξη προσημασμένων, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** ADD R0, R1 αν για προσημασμένους  $R0+R1 \notin [-128, 127]$

JV x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή.

Η σημαία της υπερχείλισης χρησιμοποιείται και στις πράξεις ολίσθησης σύμφωνα με τον τρόπο που έχει διατυπωθεί στην Ενότητα 4.2 (σ. 35).

#### 4.7.9. JNV - άλμα αν δεν υπάρχει υπερχείλιση

		Instr1								Instr2							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type	3	+-----+-----+-----+-----+								+-----+-----+-----+-----+							
		0	0	0	0	1	1	0	1	n							

**Μνημονικό:** JNV n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:** Αν  $V=0$  τότε  $PC \leftarrow n$

Αν δεν υπήρχε υπερχείλιση σε πράξη προσημασμένων, οδηγεί την εκτέλεση στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** ADD R0, R1 αν  $R0+R1 \in [-128, 127]$

JNV x γίνεται άλμα στην διεύθυνση της ετικέτας x

... αλλιώς εκτελείται η επόμενη εντολή.

#### 4.7.10. CALL - κλήση υπορουτίνας

Instr1									Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 3	0	0	0	0	1	1	1	0	n							

**Μνημονικό:** CALL n

n κατευθείαν διεύθυνση μνήμης

**Λειτουργία:**  $PC+2 \rightarrow [--SP]$  και μετά  $PC \leftarrow n$

Αρχικά ωθεί στην στοίβα την διεύθυνση της εντολής που ακολουθεί την CALL, και μετά κάνει άλμα στην διεύθυνση n.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** CALL x ωθεί την διεύθυνση της y, και κάνει άλμα στην διεύθυνση x  
y

Η CALL έχει μέγεθος 2, άρα η εντολή y βρίσκεται στην θέση PC+2.

#### 4.7.11. RETURN - επιστροφή απο υπορουτίνα

				Instr1							
				7	6	5	4	3	2	1	0
Type 1	+-----+-----+-----+										
		0	0	0	0	1	1	1	1		
+-----+-----+-----+											

**Μνημονικό:** RETURN

**Λειτουργία:**  $PC \leftarrow [SP++]$

Απωθεί απο την στοίβα μια διεύθυνση εντολής στον μετρητή προγράμματος και επομένως κάνει άλμα σε αυτή τη διεύθυνση.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** 1: CALL 5 ωθεί την διεύθυνση 3, και κάνει άλμα στην διεύθυνση 5

3: ... εδώ επιστρέφει η RETURN

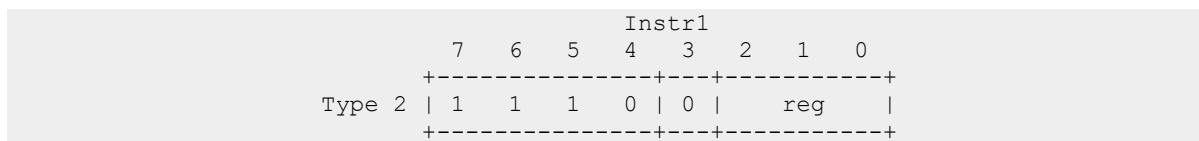
5: ... εδώ γίνεται το άλμα απο την CALL

7: RETURN απωθεί την διεύθυνση 3 και κάνει άλμα σε αυτήν

8: ...

## 4.8. Εντολές στοίβας

### 4.8.1. PUSH - ώθηση στην στοίβα



**Μνημονικό:** PUSH reg

reg κατευθείαν διεύθυνση καταχωρητή

**Λειτουργία:** reg  $\rightarrow$  [--SP]

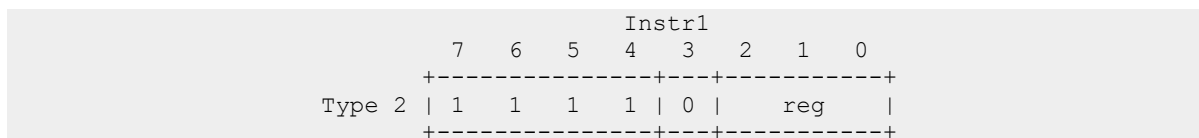
Μειώνει τον Stack Pointer (R7/SP) κατά 1 και μετά αποθηκεύει την τιμή του reg στην διεύθυνση που υποδεικνύει έμμεσα ο SP.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** PUSH R0 ωθεί την τιμή του R0 στην στοίβα.

PUSH FLAGS αποθηκεύει την κατάσταση σημαιών (R6) στην στοίβα.

### 4.8.2. POP - απόθεση από την στοίβα



**Μνημονικό:** POP reg

reg κατευθείαν διεύθυνση καταχωρητή

**Λειτουργία:** reg  $\leftarrow$  [SP++]

Φορτώνει στον reg την τιμή στην έμμεση διεύθυνση του SP και μετά αυξάνει τον SP κατά 1.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** POP R0 απωθεί από την στοίβα στον R0.

POP FLAGS επαναφέρει τις σημαίες εφόσον είχαν αποθηκευτεί στην στοίβα.

## 4.9. Εντολές ολίσθησης

### 4.9.1. RSHIFT - δεξιά ολίσθηση

		Instr1							
		7	6	5	4	3	2	1	0
		+-----+-----+-----+							
Type 2		1	0	1	0		0		reg
		+-----+-----+-----+							

**Μνημονικό:** RSHIFT reg

reg κατευθείαν διεύθυνση καταχωρητή

**Λειτουργία:**  $(reg, C) \gg 1, V \leftarrow 1$  αν αλλάξει το S

Ολισθαίνει τα ψηφία του reg μια θέση δεξιά και τοποθετεί 0 στο ΠΣΨ.

**Σημαίες:** Μεταβάλλονται όλες οι σημαίες. Στο κρατούμενο καταχωρείται το ολισθημένο bit. Αν αλλάξει το πρόσημο τότε η σημαία υπερχειλίσσης τίθεται στο 1.

**Παράδειγμα:** MOV R0, 0b01000001

RSHIFT R0

Απο  $01000001_2$  (65) ο R0 γίνεται  $00100000_2$  (32).

Το ολισθημένο bit είναι 1, άρα  $C=1$ .

Προφανώς  $Z=0$  και  $S=0$ .

Το S δεν άλλαξε, οπότε  $V=0$ .

Στην δεξιά ολίσθηση γίνεται με ακέραια διαίρεση με το 2 ενώ στο κρατούμενο καταχωρείται το modulo 2.

### 4.9.2. LSHIFT - αριστερή ολίσθηση

		Instr1							
		7	6	5	4	3	2	1	0
		+-----+-----+-----+							
Type 2		1	1	0	0		0		reg
		+-----+-----+-----+							

**Μνημονικό:** LSHIFT reg

reg κατευθείαν διεύθυνση καταχωρητή

**Λειτουργία:**  $(C, reg) \ll 1, V \leftarrow 1$  αν αλλάξει το S

Ολισθαίνει τα ψηφία του reg μια θέση αριστερά και τοποθετεί 0 στο ΛΣΨ.

**Σημαίες:** Μεταβάλλονται όλες οι σημαίες. Στο κρατούμενο καταχωρείται το ολισθημένο bit. Αν αλλάξει το πρόσημο τότε η σημαία υπερχειλίσσης τίθεται στο 1.

**Παράδειγμα:** MOV R0, 0b01000001

LSHIFT R0

Απο  $01000001_2$  (65) ο R0 γίνεται  $10000010_2$  (130 ή προσημασμένος -126).

Το ολισθημένο bit είναι 0, άρα  $C=0$ .

Προφανώς  $Z=0$  και  $S=1$ .

Το S άλλαξε, άρα  $V=1$ .

Η αριστερή ολίσθηση ισοδυναμεί με πολλαπλασιασμό με το 2. Κρατούμενο ισοδυναμεί με γινόμενο εκτός του εύρους των 8 bit.

### 4.9.3. ROR - δεξιά περιστροφή

	Instr1								Instr2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Type 4	0	1	0	0	1	0	0	0	0	0	reg	0	op2			
Type 5	0	1	0	0	0	reg						op2				

**Μνημονικό:** ROR reg, op2

reg κατευθείαν διεύθυνση καταχωρητή

op2 κατευθείαν διεύθυνση καταχωρητή (αν Instr1[3]=1)  
ή άμεση τιμή (αν Instr1[3]=0)

**Λειτουργία:** reg >> op2 | reg << (8-op2)

Περιστρέφει δεξιά τον reg κατα op2 mod 8 bits.

**Σημαίες:** Μεταβάλλονται οι Z και S.

**Παράδειγμα:** ROR R0, R1 περιστρέφει δεξιά τον R0 κατα τα 3 ΛΣΨ του R1.

ROR R2, 2 περιστρέφει δεξιά τον R2 κατα 2 bits (αριστερά κατα 6 bits)

ROR R3, 5 περιστρέφει δεξιά τον R3 κατα 5 bits (αριστερά κατα 3 bits)

Η δεξιά περιστροφή κατα x bits ισοδυναμεί με αριστερή ολίσθηση 8-x bits (σε οκτάμπιτους).

## 4.10. Ειδικές εντολές

### 4.10.1. NOP - καμία λειτουργία

	Instr1							
	7	6	5	4	3	2	1	0
Type 1	0	0	0	0	0	0	0	1

**Μνημονικό:** NOP

**Λειτουργία:** Δεν εκτελεί καμία λειτουργία.

**Σημαίες:** Δεν μεταβάλλονται.

**Παράδειγμα:** NOP Συνεχίζει στην εκτέλεση της επόμενης εντολής

Η NOP χρησιμοποιείται για εισαγωγή καθυστερήσεων, στοίχιση κώδικα και υπόδειξη κενού χώρου που μπορεί να συμπληρωθεί με κώδικα.

Ειδικά στον E80 η NOP συμπεριλήφθηκε για καθαρά εκπαιδευτικούς σκοπούς. Δεδομένου ότι δεν εκτελεί καμία λειτουργία, είναι χρήσιμη για την εισαγωγική επίδειξη της διόδου δεδομένων και της μονάδας ελέγχου.



#### 4.10.2. HLT - παύση εκτέλεσης

		Instr1							
		7	6	5	4	3	2	1	0
Type 1		+-----+							
		0	0	0	0	0	0	0	0
		+-----+							

**Μνημονικό:** HLT

**Λειτουργία:** Τερματίζει την εκτέλεση στον τρέχοντα κύκλο οδηγώντας σε ατέρμονα βρόχο. Στην FPGA το LED του ρολογιού σταματά να αναβοσβήνει και παραμένει φωτεινό. Στο GHDL και το ModelSim η προσομοίωση τερματίζεται.

**Σημείες:** Ενεργοποιείται η σημαία H.

**Παράδειγμα:** HLT τερματίζει την εκτέλεση του προγράμματος απαιτώντας reset.

Η HLT τερματίζει την εκτέλεση στον κύκλο που εκτελείται αυτή. Ενεργοποιώντας την σημαία Halt μέσω άλλων εντολών (πχ. OR FLAGS, 0b00001000) επιτυγχάνεται το ίδιο αποτέλεσμα αλλά στον επόμενο κύκλο.

Ο κωδικός εντολής της HLT έχει τεθεί στο 00000000 έτσι ώστε κατά την μετατροπή σε μορφή BIT (βλ. Ενότητα 5.2, σ. 54) να αντιστοιχεί με την τιμή 'UUUUUUUU' (undefined) στα GHDL και ModelSim αλλά και στις FPGA που δοκιμάστηκαν. Δεδομένου ότι η μνήμη αρχικοποιείται σε τιμές 'U', αυτό σημαίνει ότι η απόπειρα εκτέλεσης μη-αρχικοποιημένης διεύθυνσης κλειδώνει άμεσα την CPU στην τρέχουσα εντολή. Το στοιχείο αυτό κάνει την HLT ιδιαίτερα χρήσιμη στον εντοπισμό λαθών.

## 4.11. Φυλλάδιο αρχιτεκτονικής εντολών του E80

Για την παρουσίαση του φυλλαδίου στην επιλέχθηκε η μορφή που έχει καταγραφεί από τον Bowen (1985) η οποία χρησιμοποιείται και στο εγχειρίδιο Μικροεπεξεργαστών του ΕΑΠ (2008) αλλά και ως τυπολόγιο σε εξετάσεις. Η χρήση Αγγλικής γλώσσας επιτρέπει τον εύκολο συσχετισμό γνώσεων με την υπάρχουσα βιβλιογραφία σε γλώσσες Assembly.

	Instr1	Instr2	Hex	Mnemonic	Description	CZSV
1	00000000		00	HLT	$H \leftarrow 1, PC \leftarrow PC$	
2	00000001		01	NOP		
3	00000010	nnnnnnnn	02 nn	JMP n	$PC \leftarrow n$	
4	00000011	00000rrr	03 0r	JMP r	$PC \leftarrow r$	
5	00000100	nnnnnnnn	04 nn	JC n	if C=1, $PC \leftarrow n$	
6	00000101	nnnnnnnn	05 nn	JNC n	if C=0, $PC \leftarrow n$	
7	00000110	nnnnnnnn	06 nn	JZ n	if Z=1, $PC \leftarrow n$	
8	00000111	nnnnnnnn	07 nn	JNZ n	if Z=0, $PC \leftarrow n$	
9	00001010	nnnnnnnn	0A nn	JS n	if S=1, $PC \leftarrow n$	
10	00001011	nnnnnnnn	0B nn	JNS n	if S=0, $PC \leftarrow n$	
11	00001100	nnnnnnnn	0C nn	JV n	if V=1, $PC \leftarrow n$	
12	00001101	nnnnnnnn	0D nn	JNV n	if V=0, $PC \leftarrow n$	
13	00001110	nnnnnnnn	0E nn	CALL n	$PC+2 \rightarrow [--SP]; PC \leftarrow n$	
14	00001111		0F	RETURN	$PC \leftarrow [SP++]$	
15	00010rrr	nnnnnnnn	1r nn	MOV r,n	$r \leftarrow n$	**
16	00011000	0rrr0rrr	18 rr	MOV r1,r2	$r1 \leftarrow r2$	**
17	00100rrr	nnnnnnnn	2r nn	ADD r,n	$r \leftarrow r+n$	****
18	00101000	0rrr0rrr	28 rr	ADD r1,r2	$r1 \leftarrow r1+r2$	****
19	00110rrr	nnnnnnnn	3r nn	SUB r,n	$r \leftarrow r+(\sim n)+1$	****
20	00111000	0rrr0rrr	38 rr	SUB r1,r2	$r1 \leftarrow r1+(\sim r2)+1$	****
21	01000rrr	nnnnnnnn	4r nn	ROR r,n	$r \gg n$ ( $r \ll 8-n$ )	**
22	01001000	0rrr0rrr	48 rr	ROR r1,r2	$r1 \gg r2$ ( $r1 \ll 8-r2$ )	**
23	01010rrr	nnnnnnnn	5r nn	AND r,n	$r \leftarrow r \& n$	**
24	01011000	0rrr0rrr	58 rr	AND r1,r2	$r1 \leftarrow r1 \& r2$	**
25	01100rrr	nnnnnnnn	6r nn	OR r,n	$r \leftarrow r   n$	**
26	01101000	0rrr0rrr	68 rr	OR r1,r2	$r1 \leftarrow r1   r2$	**
27	01110rrr	nnnnnnnn	7r nn	XOR r,n	$r \leftarrow r \wedge n$	**
28	01111000	0rrr0rrr	78 rr	XOR r1,r2	$r1 \leftarrow r1 \wedge r2$	**
29	10000rrr	nnnnnnnn	8r nn	STORE r,[n]	$r \rightarrow [n]$	
30	10001000	0rrr0rrr	88 rr	STORE r1,[r2]	$r1 \rightarrow [r2]$	
31	10010rrr	nnnnnnnn	9r nn	LOAD r,[n]	$r \leftarrow [n]$	**
32	10011000	0rrr0rrr	98 rr	LOAD r1,[r2]	$r1 \leftarrow [r2]$	**
33	10100rrr		Ar	RSHIFT r	$(r,C) \gg 1; V \leftarrow S \text{ flip}$	****
34	10110rrr	nnnnnnnn	Br nn	CMP r,n	SUB, discard result	****
35	10111000	0rrr0rrr	B8 rr	CMP r1,r2	SUB, discard result	****
36	11000rrr		Cr	LSHIFT r	$(C,r) \ll 1; V \leftarrow S \text{ flip}$	****
37	11010rrr	nnnnnnnn	Dr nn	BIT r,n	AND, discard result	**
38	11100rrr		Er	PUSH r	$r \rightarrow [--SP]$	
39	11110rrr		Fr	POP r	$r \leftarrow [SP++]$	

← or → : data transfer, takes effect on the next cycle  
n : 8-bit immediate value or memory address  
r,r1,r2 : 3-bit register address (R0 to R7)  
[x] : memory at 8-bit address x < 255, [255] = DIP input  
PC : program counter, initialized to 0 on reset  
SP : register R7, initialized to 255 on reset  
--SP : decrease SP by 1, and then read it  
SP++ : read SP, and then increase it by 1  
Flags : register R6 = [CZSVH--], \* = flag affected, space = flag unaffected  
C : carry flag, C=1 in SUB/CMP A,B  $\Leftrightarrow$  unsigned A  $\geq$  unsigned B  
Z : zero flag, ALU result = 0  
S : sign flag, ALU result MSB  
V : signed overflow flag, V=1 in L/RSHIFT  $\Leftrightarrow$  Sign bit flipped  
H : halt flag (set by HLT), PC freezes

## 5. Σχεδιασμός και υλοποίηση του E80 σε VHDL

### 5.1. Σχεδιαστικές επιλογές και περιορισμοί

Προκειμένου να επιτευχθούν οι στόχοι που έχουν καθοριστεί στην Εισαγωγή, η σχεδίαση του E80 οριοθετήθηκε ως εξής:

#### 5.1.1. Ανάπτυξη χωρίς αριθμητικές βιβλιοθήκες

Σύμφωνα με το επιχείρημα που αναπτύχθηκε στην Ενότητα 3.4 (σ. 25), ως προς τον στόχο της **Συνέχειας**, ο E80 σχεδιάστηκε αποκλειστικά με την βασική βιβλιοθήκη Std\_logic\_1164. Οι πράξεις πρόσθεσης, αφαίρεσης και σύγκρισης υλοποιήθηκαν μέσω ενός απλού κυματικού αθροιστή όπως παρουσιάζεται στην σχετική θεωρία ψηφιακής σχεδίασης.

#### 5.1.2. Χρήση της PROCESS αποκλειστικά στην κατασκευή του D flip flop

Το πρακτικό πλεονέκτημα της αφαιρετικότητας της VHDL ως προς την δημιουργία ακολουθιακών κυκλωμάτων μέσω της δήλωσης PROCESS, αποτελεί ταυτόχρονα και μειονέκτημα ως προς τους εκπαιδευτικούς σκοπούς του E80 δεδομένου ότι αποκρύπτει την γνώση που έχει αποκτηθεί στην ψηφιακή σχεδίαση, σύμφωνα με την αναφορά που έγινε στο Υπόβαθρο (Ενότητα 2.3.2, σ. 10). Επομένως, και πάλι με γνώμονα τον στόχο της **Συνέχειας**, η PROCESS χρησιμοποιήθηκε μόνο στο σημείο όπου είναι αντικειμενικά υποχρεωτική, δηλαδή στην κατασκευή του D flip flop.

Σε συνδυασμό με την αποκλειστική χρήση της βιβλιοθήκης Std\_logic\_1164, ο περιορισμός αυτός εξασφαλίζει ότι, με εξαίρεση το D flip-flop, όλα τα υποσυστήματα του E80 οπτικοποιούνται με την θεωρία της ψηφιακής σχεδίασης, όπως φαίνεται στα διαγράμματα του Παραρτήματος Γ (σ. 166).

#### 5.1.3. Συμβατότητα με το υποσύνολο της VHDL 2008 που υποστηρίζει το Quartus

Η συμβατότητα με το Quartus Prime Lite ήτανε απαραίτητη για την κάλυψη του στόχου της **Υλοποίησης** στην πλακέτα DSD-i1. Η συγκεκριμένη πλακέτα χρησιμοποιεί το FPGA Cyclone IV της Altera / Intel που υποστηρίζεται από το Quartus Prime Lite και χρησιμοποιείται στο Ψηφιακό Εργαστήριο του ΕΑΠ.

Δυστυχώς, το Quartus Prime Lite πληροί μόνο ένα υποσύνολο της VHDL 2008 όπως έχει καταγραφεί στην Ενότητα 2.6.2 (σ. 15). Αυτό αποτέλεσε την μεγαλύτερη τροχοπέδη στην εργασία αυτή, καθώς η VHDL 2008 προσφέρει σημαντικά πλεονεκτήματα ως προς την αναγνωσιμότητα του κώδικα όπως σημειώθηκε στην Ενότητα 2.3 (σ. 8).

Στην προσπάθεια να αντιμετωπιστεί αυτό το πρόβλημα, κατασκευάστηκε μια βιβλιοθήκη (Ενότητα 5.2, σ. 54) που προσέφερε κάποιες δυνατότητες της VHDL 2008 διατηρώντας την συμβατότητα με το Quartus.

Σημειώνεται ότι ο συγκεκριμένος περιορισμός εξασφάλισε συμβατότητα με Gowin, GHDL, ModelSim και Active-HDL τα οποία προσφέρουν ισχυρότερη υποστήριξη της VHDL από το Quartus. Επίσης οι περιορισμοί που τέθηκαν ως τώρα (βιβλιοθήκες, PROCESS, Quartus) δεν αφορούν στα testbench τα οποία εκτελούνται μόνο σε προσομοιωτές και δεν αποτελούν εγγενές τμήμα του σχεδιασμού του E80.

#### 5.1.4. Άμεση χρήση στιγμιοτύπων αντί για δηλώσεις συστατικών

Ο πιο συνηθισμένος τρόπος χρήσης συστατικών σε κώδικα VHDL, είναι η δήλωση του συστατικού και η μετέπειτα χρήση του στην αρχιτεκτονική. Υπάρχει όμως και η δυνατότητα άμεσης χρήσης ενός στιγμιοτύπου χωρίς πρότερη δήλωσή του, το οποίο προσφέρει πλεονεκτήματα αναγνωσιμότητας κατά τους Bergé (1993) και Jensen (2020). Αυτό επιβεβαιώθηκε στην πράξη, καθώς η άμεση χρήση στιγμιοτύπων των συστατικών μείωσε τις πλεονάζουσες δηλώσεις, βελτιώνοντας την αναγνωσιμότητα του τμήματος δηλώσεων.

#### 5.1.5. Ένας κύκλος ανα εντολή

Όπως τονίστηκε στο Υπόβαθρο (Ενότητα 2.2.3, σ. 6), οι επεξεργαστές ενός κύκλου είναι απλούστεροι αλλά πιο αργοί και χρειάζονται περισσότερα υποσυστήματα.

Το μειονέκτημα της ταχύτητας είναι αδιάφορο ως προς τους στόχους του E80. Το μειονέκτημα των υποσυστημάτων είναι σημαντικό. Η χρήση πολλαπλών κύκλων θα επέτρεπε την προσάυξηση του μετρητή προγράμματος με την υπάρχουσα ALU χωρίς να χρειαστεί προσθήκη ενός επιπλέον αθροιστή.

Το πλεονέκτημα της απλότητας είναι όμως ακόμα πιο σημαντικό, λαμβάνοντας υπόψιν ότι στην προσομοίωση ο ένας κύκλος ανα εντολή επιτρέπει στο ρολόι να χρησιμοποιείται για μέτρηση βημάτων εκτέλεσης. Το συγκεκριμένο πλεονέκτημα συσχετίζεται άμεσα με το χαρακτηριστικό του χαμηλού κατωφλίου που έχουν οι εκπαιδευτικοί μικρόκοσμοι κατα Πάπερτ (Ενότητα 2.7.1, 17).

Το μοναδικό πρόβλημα που παρουσιάστηκε με τον περιορισμό του ενός κύκλου ανα εντολή, ήταν η υλοποίηση της εντολής POP η οποία γράφει ταυτόχρονα σε δύο καταχωρητές και διαβάζει μια λέξη από την μνήμη. Για την υλοποίησή της δοκιμάστηκαν προσεγγίσεις με πολλαπλούς κύκλους αλλά απορρίφθηκαν λόγω της πολυπλοκότητάς τους. Τελικά αναπτύχθηκε η μέθοδος που καταγράφεται στην Ενότητα 5.9.9 (σ. 80) και επομένως ο περιορισμός του ενός κύκλου ανα εντολή διατηρήθηκε.

#### 5.1.6. Περιορισμός της ενθυλάκωσης υποσυστημάτων

Υλοποιήσεις που εντοπίζονται στην βιβλιογραφία όπως σε Harris & Harris (2022, σ. 458-459), κατασκευάζουν ξεχωριστά και διακριτά υποσυστήματα διόδου δεδομένων και ελέγχου. Αυτό έχει θεωρητικά πλεονεκτήματα (δόμηση, διαίρεση σε απλούστερα τμήματα) αλλά στην πράξη διαπιστώθηκε ότι αυτή η μέθοδος μειονεκτούσε ως προς την αναγνωσιμότητα της συνολικής λειτουργίας του E80. Για παράδειγμα, η κατανόηση της εκτέλεσης της στοίβας απαιτούσε επαναλαμβανόμενες αλλαγές του συστατικού προς ανάγνωση. Το πρόβλημα αυτό δεν γίνεται αντιληπτό στην υλοποίηση της Harris η οποία όμως περιορίζεται σε λίγες και απλές λειτουργίες.

Κατά τον σχεδιασμό του E80, λήφθηκε η συνειδητή απόφαση να δοθεί προτεραιότητα στην αναγνωσιμότητα του κώδικα VHDL και στην ολιστική κατανόηση του συστήματος, έναντι μιας αυστηρής διάσπασης του σχεδίου σε πολλά μικρά συστατικά. Αυτό πρακτικά σημαίνει ότι η δίοδος δεδομένων και ο έλεγχος βρίσκονται σε μια ενιαία αρχιτεκτονική στο αρχείο CPU.vhd όπως καταγράφεται στο Υπόβαθρο (Ενότητες 2.2.2 & 2.2.1, σ. 5). Ταυτόχρονα

έγινε προσπάθεια ώστε τα υποσυστήματα να βρίσκονται σε διακριτές (και σχολιασμένες) ενότητες στον κώδικα διατηρώντας έτσι τα πλεονεκτήματα της δομημένης προσέγγισης χωρίς τα μειονεκτήματα της ενθυλάκωσης.

Το μοναδικό μειονέκτημα της προσέγγισης αυτής εντοπίστηκε στην αναγνωσιμότητα των κυκλωματικών διαγραμμάτων του RTL Viewer του Quartus, τα οποία είναι, στην καλύτερη περίπτωση, δυσανάγνωστα, όπως φαίνεται στο Παράρτημα Γ (σ. 166). Το πρόβλημα αυτό αντιμετωπίστηκε με την κατασκευή πιο αφηρημένων διαγραμμάτων με την εφαρμογή QElectroTech, όπως θα φανεί στην συνέχεια του κεφαλαίου.

## 5.2. Βιβλιοθήκη υποστήριξης

Για την αντιμετώπιση της μερικής ασυμβατότητας του Quartus Lite με την VHDL2008 αλλά και για την βελτίωση της αναγνωσιμότητας του κώδικα, κατασκευάστηκε η βιβλιοθήκη `work.support` με τον πλήρη κώδικά της να καταγράφεται στο Παράρτημα B.2 (σ. 154).

Η βιβλιοθήκη ορίζει τους εξής τύπους δεδομένων:

- `WORD`, που εκφράζει διάνυσμα οκτάμπιτης λέξης ή διεύθυνση μνήμης
- `WORDx8`, για τα περιεχόμενα του αρχείου καταχωρητών
- `WORDx256`, για τα περιεχόμενα της μνήμης
- `REG_ADDR`, για την διεύθυνση καταχωρητή
- `DECIHERTZ`, για την προγραμματιζόμενη συχνότητα λειτουργίας του επεξεργαστή

Επίσης ορίζει τις εξής συναρτήσεις:

- `int(arg)`, που μετατρέπει τον δυαδικό `arg` σε φυσικό αριθμό
- `match(arg1, arg2)`, που συγκρίνει τα `arg1` και `arg2`, υποστηρίζοντας αδιάφορα σήματα ('-') όπως ο τελεστής `?` = ο οποίος δεν υποστηρίζεται στο Quartus Lite

Η `match` εξάγει αποτέλεσμα τύπου `Std_logic` ή `Boolean` ανάλογα με την έκφραση στην οποία χρησιμοποιείται. Αυτό γίνεται μέσω υπερφόρτωσης (overloading) η οποία υποστηρίζεται από την VHDL. Επίσης μπορεί να χρησιμοποιηθεί ακόμα και αν το μέτρο του `arg2` είναι μικρότερο από του `arg1`. Στην περίπτωση αυτή ο έλεγχος ξεκινά από το πιο σημαντικό ψηφίο. Για παράδειγμα η `match("1010", "1-1")` αποτιμάται ως Αληθής.

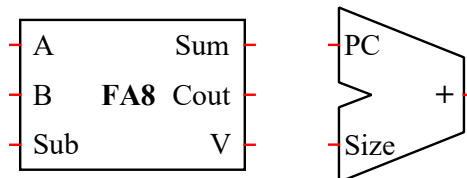
Εφόσον ένα σήμα δεν έχει οριστεί ως αδιάφορο τότε η `match` το συγκρίνει μετατρέποντας το σε τύπο `BIT` όπως φαίνεται στο παρακάτω τμήμα κώδικα:

```
-- compare as bit to avoid std_logic's matching table
ELSIF To_bit(v1(I)) XOR To_bit(v2(I)) THEN
    RETURN FALSE;
END IF;
```

Η μετατροπή αυτή γίνεται μέσω της συνάρτησης `To_bit`, που υλοποιείται στην βιβλιοθήκη `Std_logic_1164` (IEEE, [1993](#), σ. 10), στην οποία τα απροσδιόριστα σήματα ('U') θεωρούνται εκ-προεπιλογής ίσα με το '0'. Αυτό συνεπάγεται ότι το διάνυσμα 'UUUUUUUU', στο οποίο αρχικοποιείται η μνήμη του E80, ισοδυναμεί με το διάνυσμα '00000000' το οποίο είναι ο κωδικός της HLT. Άρα, η εκτέλεση μη-αρχικοποιημένου περιεχομένου θα τερματίζει την προσομοίωση και την εκτέλεση σε FPGA όπως θα φανεί στα Κεφάλαια 8 και 9.

### 5.3. Πλήρης αθροιστής

Για τις πράξεις πρόσθεσης, αφαίρεσης και σύγκρισης σχεδιάστηκε ο πλήρης αθροιστής που παρουσιάζεται στο Παράρτημα B.12 (σ. 164) σύμφωνα με την θεωρία του εγχειριδίου ψηφιακής σχεδίασης της ΠΛΗ21 (Λιναρδής, 2008, σ. 276).



Εικόνα 11: Πλήρης αθροιστής 8-bit και ένα στιγμιότυπό του (δεξιά)

Ο αθροιστής δέχεται διανύσματα A και B και το σήμα Sub για επιλογή πρόσθεσης (Sub=0) ή αφαίρεσης (Sub=1). Το αποτέλεσμα εξάγεται στο διάνυσμα Sum, ενώ το κρατούμενο και η υπερχειλίση προσημασμένων στα Cout και V. Στο δεξί σχήμα της Εικόνας 11 καταγράφεται το στιγμιότυπο του αθροιστή που χρησιμοποιείται στην δίοδο δεδομένων για τον υπολογισμό της διεύθυνσης της παρακείμενης εντολής. Στο στιγμιότυπο αυτό έχει οριστεί Sub=0 ενώ τα σήματα κρατουμένου και υπερχειλίσης αγνοούνται.

Ο αθροιστής υλοποιείται μέσω ενός εσωτερικού συστατικού πλήρους αθροιστού του 1 bit:

```
X <= A XOR B;
S <= X XOR Cin;
Cout <= (A AND B) OR (X AND Cin);
```

και μετά γίνεται παράταξη 8 τέτοιων αθροιστών με την δήλωση FOR:

```
C(0) <= Sub;
FA_Array : FOR i IN 0 TO 7 GENERATE
  FA: ENTITY work.FA PORT MAP(
    A(i),
    B(i) XOR Sub,
    C(i), -- Cin
    Sum(i),
    C(i+1)); -- Cout
END GENERATE;
Cout <= C(8);
V <= C(8) XOR C(7);
```

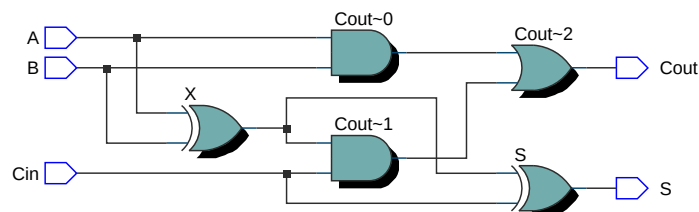
Αν Sub=0 τότε τα σήματα B(i) μένουν ως έχουν και πραγματοποιείται πρόσθεση. Αν Sub=1 τότε τα σήματα B(i) συμπληρώνονται, ενώ με την πρόσθεση του 1 μέσω του C(0)=Sub γίνεται αφαίρεση μέσω συμπληρώματος ως προς 2.

Στην περίπτωση που οι A και B θεωρηθούν προσημασμένοι, η υπερχειλίση ορίζεται από την έξοδο V. Στην περίπτωση που θεωρηθούν απρόσημοι, τότε:

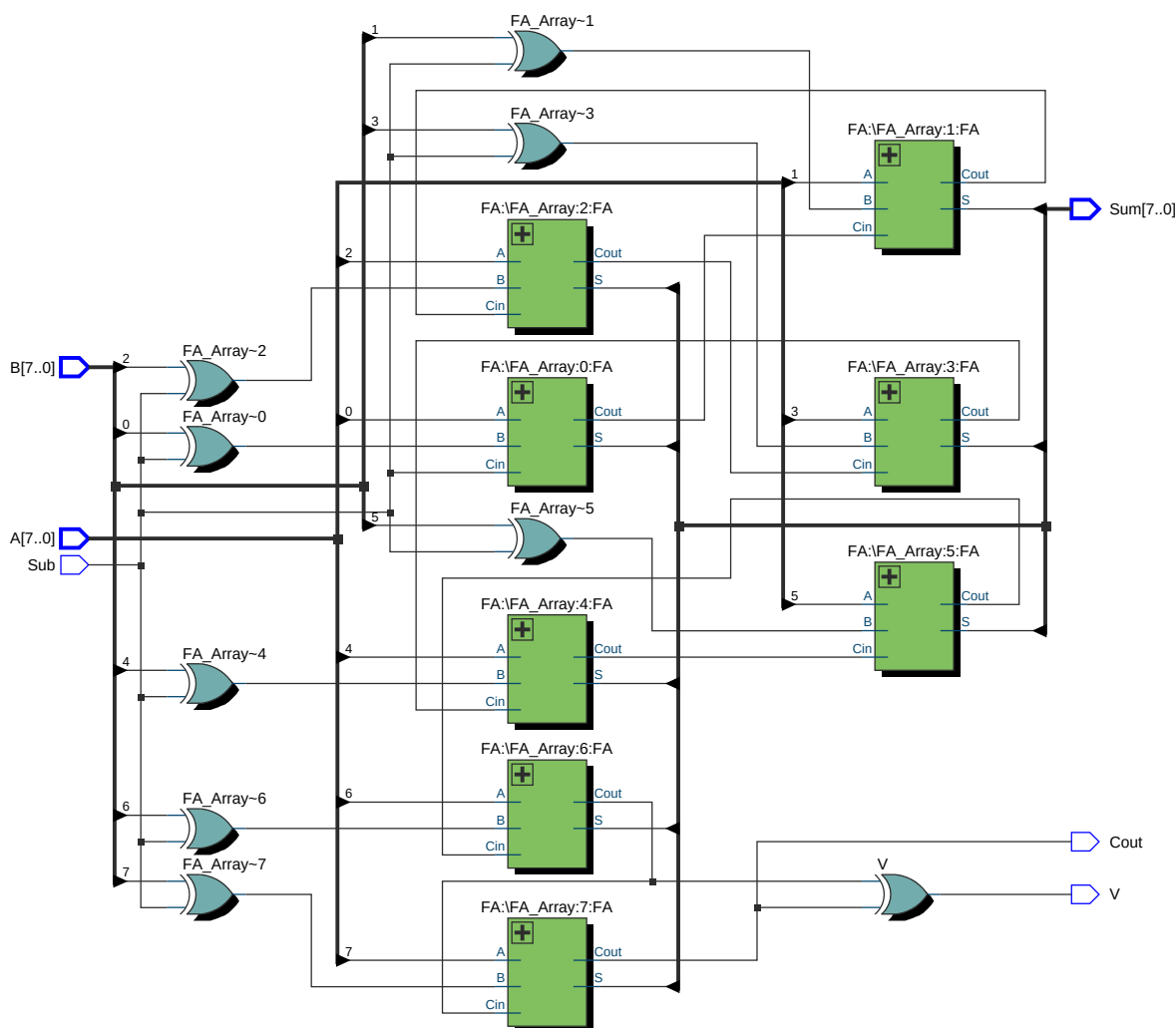
- Στην πρόσθεση η υπερχειλίση αντιστοιχεί με Cout=1,
- Στην αφαίρεση η υπερχειλίση αντιστοιχεί με Cout=0.

Δεδομένης της σημασίας των παραπάνω, όπως αυτή έχει αναλυθεί διεξοδικά στην Ενότητα 3.5 (σ. 28), τα σήματα αυτά εξάγονται ως σημαίες και υποστηρίζονται από τις εντολές άλματος JC/JNC/JV/JNV.

Χρησιμοποιώντας τον RTL Viewer του Quartus απο το μενού Compile > Tools > Netlist Viewer > RTL Viewer, εξήχθησαν τα κυκλωματικά διαγράμματα στις Εικόνες 12 και 13 που ακολουθούν. Η ομοιότητα με τον αθροιστή της άσκησης που παρουσιάστηκε στην Εικόνα 9 (σ. 28) είναι προφανής και επαληθεύει το επιχείρημα που αναπτύχθηκε στην εισαγωγή του κεφαλαίου: περιορίζοντας τον κώδικα στην Std\_logic\_1164, το κύκλωμα που παράγεται γίνεται εύκολα κατανοητό με στοιχειώδη θεωρία ψηφιακής σχεδίασης.



Εικόνα 12: Κυκλωματικό διάγραμμα 1-bit full adder απο τον RTL viewer του Quartus



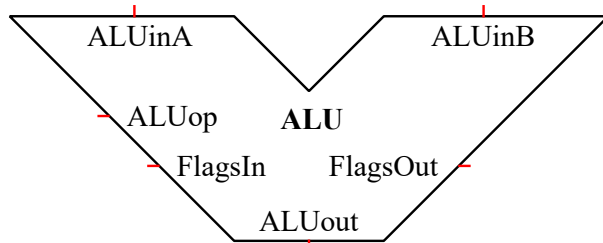
Εικόνα 13: Κυκλωματικό διάγραμμα 8-bit full adder απο τον RTL viewer του Quartus

Η σχετικά καλή αναγνωσιμότητα των παραπάνω διαγραμμάτων οφείλεται στην ενθυλάκωση του συστατικού του αθροιστού 1 bit. Αυτή είναι η μοναδική περίπτωση όπου παραβιάστηκε ο περιορισμός που αναφέρθηκε στην Υποενότητα 5.1.6 (σ. 53). Στα υπόλοιπα κυκλώματα χρησιμοποιούνται διαγράμματα που έχουν σχεδιαστεί “στο χέρι”, στοχευμένα για άμεσο συσχετισμό με τον κώδικα VHDL.



## 5.4. Αριθμητική-Λογική Μονάδα (ALU)

Η υλοποιηθείσα ALU καταγράφεται με πλήρη σχόλια εντός του κώδικα στο Παράρτημα B.10 (σ. 162). Ακολουθεί την αρχιτεκτονική που περιγράφηκε στο Υπόβαθρο (Ενότητα 2.1.4, σ. 4), προσθέτοντας ειδικές λειτουργίες για την υποστήριξη των εντολών του E80.



Εικόνα 14: Η ALU του E80

Όπως φαίνεται στην Εικόνα 14, η ALU δέχεται τελούμενα ALUinA και ALUinB, την επιλογή της πράξης ALUop, και τις τρέχουσες σημαίες κατάστασης FlagsIn. Τα αποτελέσματα εξάγονται ως ALUout και οι σημαίες εξόδου ως FlagsOut. Όλες οι είσοδοι και εξοδοι είναι διανύσματα λέξεων εκτός από το ALUop που είναι διάνυσμα 4 bit.

Η λειτουργία της ALU υλοποιείται μέσω διακριτών τμημάτων κώδικα, τα οποία αναλύονται παρακάτω.

### 5.4.1. Αποκωδικοποίηση (ALUop Decoder)

Ο αποκωδικοποιητής της ALU μεταφράζει το διάνυσμα επιλογής ALUop σε σήματα μορφής *isX* όπου *X* η πράξη που εκτελείται. Στην ενότητα του αποκωδικοποιητή ορίζονται επίσης τα σήματα FullFlags, DiscardFlags, και DiscardResult τα οποία επιτρέπουν την διατήρηση ή την απόρριψη του αποτελέσματος και των σημαιών:

-- ALUop Decoder	ALUout	CZSV
isBypass <= match(ALUop, "-000");	--  A (J*, STORE, CALL, etc)	**
isAssign <= match(ALUop, "-001");	--  B (MOV, LOAD)	****
isADD <= match(ALUop, "0010");	--  A + B	****
isSUB <= match(ALUop, "-011");	--  A - B (includes CMP)	**
isROR <= match(ALUop, "0100");	--  A rotated by B mod 8 bits	**
isAND <= match(ALUop, "-101");	--  A AND B (includes BIT)	**
isOR <= match(ALUop, "0110");	--  A OR B	**
isXOR <= match(ALUop, "0111");	--  A XOR B	****
isRSHIFT <= match(ALUop, "1010");	--  A >> 1, C ← A(0), V ← S flip	****
isCMP <= match(ALUop, "1011");	--  SUB, discard result	****
isLSHIFT <= match(ALUop, "1100");	--  A << 1, C ← A(7), V ← S flip	**
isBIT <= match(ALUop, "1101");	--  AND, discard result	**
isDCR <= match(ALUop, "1110");	--  A - 1 (POP, RETURN)	**
isINR <= match(ALUop, "1111");	--  A + 1 (PUSH, CALL)	**
FullFlags <= isADD OR isSUB OR isRSHIFT OR isLSHIFT;		
DiscardFlags <= isBypass OR isINR OR isDCR;		
DiscardResult <= isBypass OR isCMP OR isBIT;		

Απο τα σχόλια του αποκωδικοποιητή προκύπτουν οι διαθέσιμες λειτουργίες της ALU. Όλες αυτές αντιστοιχούν άμεσα σε εντολές, εκτός από τις isDCR/isINR που χρησιμοποιούνται για αυξομείωση του δείκτη στοίβας κατά 1. Σε αυτές τις δυο λειτουργίες δεν μεταβάλλονται οι σημαίες (DiscardFlags=1) όπως απαιτείται από τις σχετικές σχεδιαστικές επιλογές της αρχιτεκτονικής (Ενότητα 4.2, σ. 35).

### 5.4.2. Προσθαφαίρεση (Full Adder / Subtractor)

Το στιγμιότυπο του πλήρους αθροιστού που περιγράφηκε στην προηγούμενη ενότητα χρησιμοποιεί ως εισόδους τα διανύσματα ALUinA και B. Το B τίθεται στο ALUinB για όλες τις πράξεις εκτός από τις DCR/INR για τις οποίες τίθεται στο 1:

```
-----
-- Full Adder / Subtractor
-----
B <= x"01" WHEN isDCR OR isINR ELSE ALUinB;
ALU_Adder : ENTITY work.FA8 PORT MAP(
    A,
    B,
    isSUB OR isDCR, -- 1 = subtraction (includes CMP)
    Sum,
    Sum_C,
    Sum_V);
```

Στην περίπτωση που ισχύει isSUB OR isDCR, σύμφωνα με την προηγούμενη ενότητα, ο αθροιστής εφαρμόζει πρόσθεση με συμπλήρωμα ως προς 2, δηλαδή αφαίρεση. Τα σήματα Sum\_C και Sum\_V εκφράζουν κρατούμενο και υπερχείλιση προσημασμένων.

### 5.4.3. Κυκλικός ολισθητής (Barrel shifter)

Ο κυκλικός ολισθητής χρησιμοποιείται για την πράξη της περιστροφής. Το υποσύστημα αυτό κατασκευάστηκε ως εναλλακτικό του τελεστή ROR της VHDL 2008, εξαιτίας των περιορισμών του Quartus Lite.

```
-----
-- Barrel shifter
-----
-- Rotation is determined by the 3 LSBs of operand B.
-- It had to be performed manually because Quartus Lite
-- doesn't support VHDL 2008 SRL/SLL/ROR/ROL operators.
WITH B(2 DOWNTO 0) SELECT Rotated <=
    A(0 DOWNTO 0) & A(7 DOWNTO 1) WHEN "001", -- right 1, left 7
    A(1 DOWNTO 0) & A(7 DOWNTO 2) WHEN "010", -- right 2, left 6
    A(2 DOWNTO 0) & A(7 DOWNTO 3) WHEN "011", -- right 3, left 5
    A(3 DOWNTO 0) & A(7 DOWNTO 4) WHEN "100", -- right 4, left 4
    A(4 DOWNTO 0) & A(7 DOWNTO 5) WHEN "101", -- right 5, left 3
    A(5 DOWNTO 0) & A(7 DOWNTO 6) WHEN "110", -- right 6, left 2
    A(6 DOWNTO 0) & A(7 DOWNTO 7) WHEN "111", -- right 7, left 1
    A                                WHEN OTHERS; -- no rotation
```

Τα 3 ΛΣΨ του B χρησιμοποιούνται για να ορίσουν το πλήθος των ψηφίων του A που θα περιστραφούν κυκλικά προς τα δεξιά. Δεδομένου ότι η περιστροφή είναι συμμετρική πράξη, σε μια λέξη 8 bit, περιστροφή B ψηφίων προς τα δεξιά ισοδυναμεί με 8-B ψηφία προς τα αριστερά. Ο πίνακας αντιστοίχισης καταγράφεται στα σχόλια του κώδικα.

#### 5.4.4. Υπολογισμός αποτελέσματος και σημαίων

Στο σημείο αυτό υλοποιούνται οι λογικές πράξεις και οι πράξεις ολίσθησης ενός ψηφίου. Το αποτέλεσμα υπολογίζεται ακόμα και στην περίπτωση που θα απορριφθεί (όπως στην εντολή σύγκρισης) γιατί μέσω αυτού εξάγονται οι σημαίες.

```
-----
-- Result & Flags
-----
-- Result needs to be calculated, even if discarded later, for flags-only
-- operations. For shift operations, the carry bit holds the shifted
-- bit while the overflow bit is set if the sign bit was flipped.
Result <=
  B                WHEN isAssign      ELSE
  Rotated          WHEN isROR         ELSE
  A AND B          WHEN isAND         ELSE
  A OR B           WHEN isOR          ELSE
  A XOR B          WHEN isXOR         ELSE
  "0" & A(7 DOWNT0 1) WHEN isRSHIFT   ELSE
  A(6 DOWNT0 0) & "0" WHEN isLSHIFT   ELSE
  Sum;
C <= A(0) WHEN isRSHIFT ELSE A(7) WHEN isLSHIFT ELSE Sum_C;
Z <= match(Result, "00000000");
S <= Result(7);
V <= A(7) XOR S WHEN isRSHIFT OR isLSHIFT ELSE Sum_V;
```

Οι σημαίες μηδενικού αποτελέσματος και προσήμου υπολογίζονται με προφανή τρόπο.

Στην περίπτωση απλής ολίσθησης (RSHIFT/LSHIFT) το ολισθημένο ψηφίο αποθηκεύεται στο κρατούμενο (C). Αν η ολίσθηση οδήγησε σε μεταβολή του προσήμου, τότε ενεργοποιείται και η σημαία υπερχείλισης (V). Η δυνατότητα αυτή βασίστηκε στον τρόπο λειτουργίας της ολίσθησης του 6800 (Motorola, [1976](#), σ. Α-48). Σε περίπτωση που δεν υπήρχε πράξη ολίσθησης, οι C και V ενημερώνονται από τις εξόδους Sum\_C και Sum\_V του αθροιστή.

#### 5.4.5. Έξοδος ή απόρριψη (διατήρηση προηγούμενου) αποτελέσματος

Στο τέλος το αποτέλεσμα και οι σημαίες εξάγονται ή απορρίπτονται ανάλογα με την επιθυμητή λειτουργία όπως είχε καθοριστεί στα σήματα FullFlags, DiscardFlags, και DiscardResult.

```
-----
-- Final output
-----
FlagsOut <=
  FlagsIn
  C & Z & S & V & FlagsIn(3 DOWNT0 0) WHEN DiscardFlags ELSE
  FlagsIn(7) & Z & S & FlagsIn(4 DOWNT0 0);
ALUout <= A WHEN DiscardResult ELSE Result;
```

Τα bits που αντιστοιχούν στην σημαία Αλτ και στις μη-χρησιμοποιούμενες θέσεις [2:0] δεν μεταβάλλονται από την ALU, οπότε επιστρέφονται στο αποτέλεσμα FlagsOut όπως ελήφθησαν από την είσοδο FlagsIn.

### 5.4.6. ALU testbench

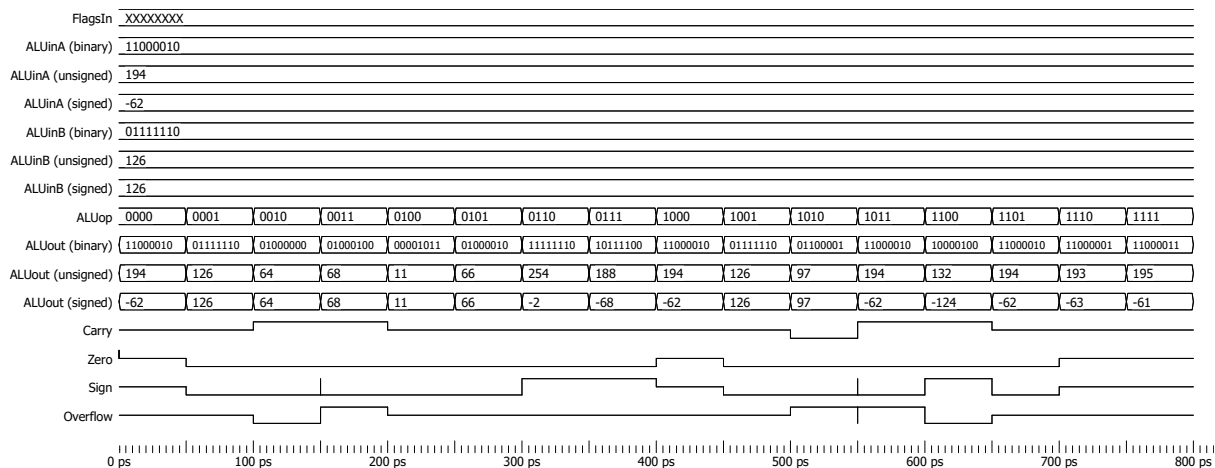
Για τις δοκιμές της ALU και του αθροιστή, δημιουργήθηκε το παρακάτω testbench (καταγράφεται πλήρως στο Παράρτημα B.11, σ. 164):

```
SIGNAL ALUinA : WORD := "11000010";
SIGNAL ALUinB : WORD := "01111110";
SIGNAL FlagsIn : WORD := "UUUUUUUU";
SIGNAL ALUop : STD_LOGIC_VECTOR(3 DOWNT0 0) := "0000";
SIGNAL ALUout, FlagsOut : WORD;
```

```
ALUop <= STD_LOGIC_VECTOR(UNSIGNED(ALUop) + 1) AFTER 50 ps;
ALU : ENTITY work.ALU PORT MAP(ALUop, ALUinA, ALUinB, FlagsIn, ALUout, FlagsOut);
```

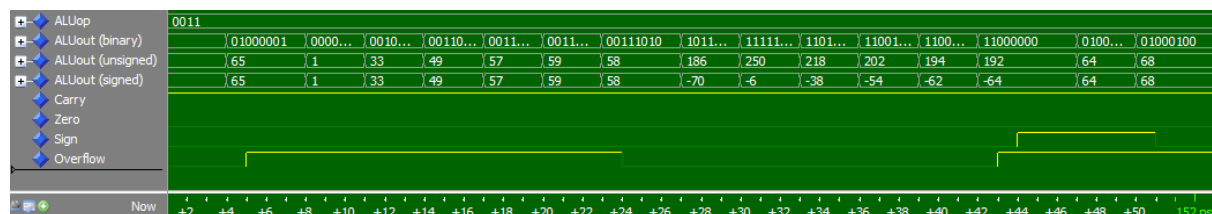
Το testbench δοκιμάζει όλες τις λειτουργίες της ALU διαδοχικά, καθώς προσauζάνει το σήμα ελέγχου ALUop κατα ένα. Οι είσοδοι αρχικοποιούνται σε δοκιμαστικές τιμές και οι σημαίες σε τιμές απροσδιοριστίας (U). Όταν εκτελούνται πράξεις που απορρίπτουν σημαίες (FullFlags=0) τότε οι απορριφθείσες σημαίες θα πρέπει να διατηρούν την τιμή του FlagsIn και συνεπώς θα πρέπει να εξάγονται (FlagsOut) ως U.

Δοκιμάζοντας το testbench στο GHDL ή στο ModelSim εξάγονται οι κυματομορφές της Εικόνας 15 που επιβεβαιώνουν την ορθή λειτουργία του ως προς τα διανύσματα ελέγχου.



Εικόνα 15: Κυματομορφές προσομοίωσης δοκιμών της ALU

Ενδιαφέρον παρουσιάζουν οι στιγμιαίες διαταραχές (hazards) που εμφανίζονται ενδιάμεσα σε κάποιες πράξεις. Χρησιμοποιώντας τα εργαλεία ελέγχου του ModelSim, εστιάζουμε στην πράξη της αφαίρεσης (ALUop=0011) στο χρονικό διάστημα του hazard στην Εικόνα 16:

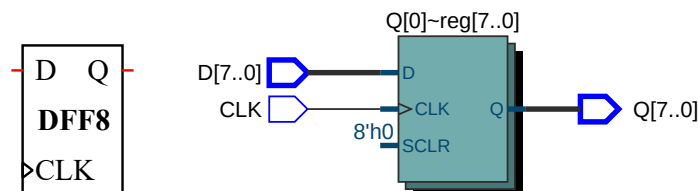


Εικόνα 16: Εστίαση σε στιγμιαία διαταραχή (hazard) στην αφαίρεση της ALU

Μπορούμε να παρατηρήσουμε τον τρόπο με τον οποίο ο κυματικός αθροιστής εξάγει διαδοχικά τα αποτελέσματα και να επιβεβαιώσουμε την αργή εκτέλεσή του. Ως προς τους εκπαιδευτικούς σκοπούς του E80, αυτό είναι ένα σημαντικό πλεονέκτημα καθώς μπορεί να χρησιμοποιηθεί ως βάση για να αιτιολογήσει βελτιωμένες μεθόδους υπολογισμού αθροίσματος ή μια αρχιτεκτονική πολλαπλών κύκλων.

## 5.5. D flip-flop

Το απλό D flip-flop των 8-bit όπως καταγράφεται στο Παράρτημα B.9 (σ. 162) χρησιμοποιείται για την αποθήκευση του μετρητή προγράμματος, τους καταχωρητές και τις κυψελίδες μνήμης. Είναι το μοναδικό συστατικό που κάνει χρήση δήλωσης PROCESS στον σχεδιασμό του E80. Στην Εικόνα 17 καταγράφεται η συμβολική μορφή του και η οπτικοποίησή του (ως “μαύρο κουτί”) στον RTL viewer του Quartus.



Εικόνα 17: Το 8-bit D flip-flop του E80 και η μορφή του στον RTL viewer του Quartus

Η είσοδος D αποθηκεύεται στην μνήμη στις θετικές ακμές του ρολογιού (RISING\_EDGE) και επομένως το D flip-flop είναι θετικά ακμοπυροδότητο.

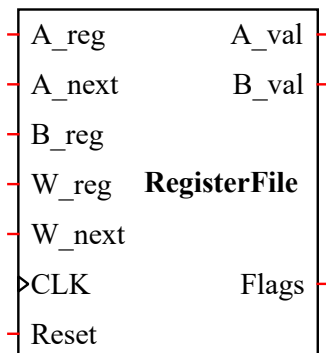
```
PROCESS (CLK) BEGIN
    IF RISING_EDGE(CLK) THEN
        Q <= D;
    END IF;
END PROCESS;
```

Η υλοποίηση του DFF έγινε με σύγχρονο Reset. Η απόφαση που οδήγησε σε αυτή την επιλογή αφορά στις λειτουργίες τριών υποσυστημάτων και θα αναλυθεί στην Ενότητα 5.10 (σ. 82) στο τέλος αυτού του κεφαλαίου.

## 5.6. Αρχείο καταχωρητών

Εφαρμόζοντας την θεωρία που αναπτύχθηκε στο Υπόβαθρο (Ενότητα 2.1.3, σ. 4), το αρχείο καταχωρητών του E80 υλοποιεί ένα πλέγμα 8 D flip-flop των 8-bit με ονόματα R0-R7 που πηγάζουν απο την Assembly του ARM. Οι καταχωρητές R0-R5 είναι γενικού σκοπού, ο R6/FLAGS αποθηκεύει τις σημαίες, και ο R7/SP είναι ο δείκτης στοίβας.

Οι είσοδοι και έξοδοι του αρχείου καταχωρητών παρουσιάζονται στο σύμβολο του συστατικού RegisterFile στην Εικόνα 18 που ακολουθεί. Τα ονόματα A, B και W προκύπτουν απο τον ρόλο των καταχωρητών: οι A και B συνήθως χρησιμοποιούνται ως τελούμενα στην ALU ενώ ο W χρησιμοποιείται για εγγραφή. Στις περισσότερες εντολές, ο W δέχεται το διάνυσμα FlagsOut απο την ALU προς ενημέρωση των σημαιών.



Εικόνα 18: Το αρχείο καταχωρητών (Register File) του E80

Πιο αναλυτικά, και εξαιρώντας τις προφανείς εισόδους CLK και Reset, το αρχείο καταχωρητών δέχεται:

- A\_reg: διεύθυνση του καταχωρητή A (3 bits)
- A\_next: τιμή του καταχωρητή A στον επόμενο κύκλο (λέξη)
- B\_reg: διεύθυνση του καταχωρητή B (3 bits)
- W\_reg: διεύθυνση του καταχωρητή W (3 bits)
- W\_next: τιμή του καταχωρητή W στον επόμενο κύκλο (λέξη)

Και εξάγει:

- Την τιμή του καταχωρητή A (λέξη)
- Την τιμή του καταχωρητή B (λέξη)
- Την τιμή του καταχωρητή σημαιών (λέξη)

Επομένως το αρχείο υποστηρίζει:

- Ανάγνωση και εγγραφή στον επιλεγμένο καταχωρητή A\_reg.
- Μόνο ανάγνωση στον επιλεγμένο καταχωρητή B\_reg.
- Μόνο εγγραφή στον επιλεγμένο καταχωρητή W\_reg.
- Ανάγνωση του καταχωρητή σημαιών.

Ο κώδικας του αρχείου καταχωρητών παρουσιάζεται στο Παράρτημα B.8 (σ. 161). Το κεντρικό σημείο του είναι η δημιουργία της διάταξης των 8 D flip flop με την δήλωση FOR:

```
Reg: FOR i IN 0 TO 7 GENERATE
  DFF8 : ENTITY work.DFF8 PORT MAP (CLK, Rnext(i), R(i));
  Rnext(i) <=
    Init(i) WHEN Reset ELSE
    A_next WHEN i = a ELSE
    W_next WHEN i = w ELSE
    R(i);
```

Σε περίπτωση σύγκρουσης (όταν A\_reg και W\_reg δείχνουν τον ίδιο καταχωρητή), προτεραιότητα έχει η τιμή A\_next. Αυτό γίνεται για να εξασφαλιστεί η σωστή λειτουργία εντολών που εσκεμμένα μεταβάλλουν τον καταχωρητή σημαιών. Για παράδειγμα, στις δύο παρακάτω εντολές, ισχύει W\_reg ← 110 (καταχωρητής σημαιών) και W\_next ← FlagsOut (απο ALU).

```
MOV R0,0
MOV FLAGS,0
```

Και στις δυο περιπτώσεις, η ALU επιστρέφει τις σημαίες που διάβασε χωρίς αλλαγή, άρα θα ισχύει W\_next ← Flags.

Στην MOV R0,0 θα ισχύει A\_reg ← 000 και επομένως δεν υπάρχει σύγκρουση μεταξύ A\_reg και W\_reg. Σύμφωνα με την αρχιτεκτονική των εντολών ως προς την MOV (Ενότητα 4.4.1, σ. 37), ο καταχωρητής R0 θα μηδενιστεί και η σημαία Z θα γίνει 1.

Στην δεύτερη περίπτωση όμως ο προγραμματιστής θέλει να μηδενίσει τις σημαίες. Επειδή ισχύει A\_reg = FLAGS = 110, θα ισχύει a=w=6 άρα θα υπήρχε σύγκρουση στην εγγραφή των A\_reg και W\_reg. Όμως η WHEN δίνει προτεραιότητα στην εγγραφή του A\_next άρα ο καταχωρητής των σημαιών θα μηδενιστεί απο την MOV και η λέξη W\_next θα αγνοηθεί.

### 5.6.1. Αρχικοποίηση σε απροσδιοριστία όταν γίνεται Reset

Ενδιαφέρον παρουσιάζει επίσης και η αρχικοποίηση των καταχωρητών όταν γίνεται Reset:

```
CONSTANT Init : WORDx8 := (6 => "UUUUUUUU", 7 => x"FF", OTHERS => x"UU");
```

Σύμφωνα με την Ενότητα 3.9 (σ. 33) ο δείκτης στοίβας αρχικοποιείται στο 255, ενώ οι υπόλοιποι καταχωρητές σε τιμές απροσδιοριστίας (undefined - "U") για δύο λόγους:

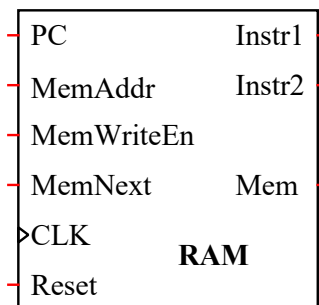
- αφενός για να διακρίνεται εύκολα το σημείο στο οποίο ένας καταχωρητής αρχικοποιείται στην προσομοίωση,
- αφετέρου για να ενισχυθεί η καλή πρακτική της αρχικοποίησης στον προγραμματισμό, δεδομένου το εκπαιδευτικού στόχου του E80.

Προφανώς, η σημαία Αλτ (R6[3]) τίθεται στο 0 για να ξεκινήσει και πάλι η λειτουργία του προγράμματος μετά το Reset.

Για την υποστήριξη της αποσφαλμάτωσης και της παρατήρησης σε πραγματικό χρόνο, το αρχείο καταχωρητών διαθέτει μια παράλληλη έξοδο όλων των τιμών του. Η έξοδος αυτή δεν αποτελεί μέρος της εσωτερικής διόδου δεδομένων της CPU, αλλά χρησιμοποιείται αποκλειστικά για την οδήγηση των ενδεικτικών LED στην πλακέτα FPGA.

### 5.7. Μνήμη

Το υποσύστημα της μνήμης του E80, όπως συμβολίζεται στην κάτωθι Εικόνα 19, δέχεται διευθύνσεις μνήμης PC και MemAddr και επιστρέφει τις λέξεις Instr1 και Instr2 που βρίσκονται στις διευθύνσεις PC και PC+1 αντίστοιχα, καθώς και την λέξη Mem που βρίσκεται στην θέση MemAddr. Αν MemWriteEn=1, τότε στην θέση MemAddr θα γραφτεί η λέξη MemNext στον επόμενο κύκλο.



Εικόνα 19: Το υποσύστημα μνήμης του E80

Χρησιμοποιώντας τους συμβολισμούς του φυλλαδίου της Αρχιτεκτονικής Εντολών του E80, ισχύει:

```
Instr1 = [PC]
Instr2 = [PC+1]
Mem = [MemAddr]
Αν MemWriteEn = 1, τότε [MemAddr] ← MemNext
```

Οι Instr1 και Instr2 εκφράζουν τα δυο τμήματα της εντολής που εκτελείται στον τρέχοντα κύκλο. Αν η εντολή έχει μέγεθος μιας λέξης, τότε η Instr2 θα αγνοηθεί.

Ακολουθώντας την αρχιτεκτονική Neumann, οι λέξεις Instr1, Instr2 και Mem βρίσκονται στον ίδιο χώρο διευθύνσεων.



Ο κώδικας του υποσυστήματος καταγράφεται στο Παράρτημα B.5 (σ. 156). Για την αποθήκευση των δεδομένων χρησιμοποιείται μια διάταξη 256 flip flop η οποία δημιουργείται με την δήλωση FOR, σύμφωνα με τον κώδικα που έχει προταθεί από τον Bhasker (1998, σ. 279).

Η συγκεκριμένη μέθοδος έχει σημαντικές διαφορές από την μέθοδο που εμφανίζεται συχνά στην βιβλιογραφία όπως ο κώδικας της Harris (2022, σ. 273) που αντιπαραβάλλεται παρακάτω:

```
----- Harris & Harris (2022, σ. 273) -----
a <= int(MemAddr)
PROCESS(CLK) BEGIN
    IF RISING_EDGE(CLK) THEN
        IF MemWriteEn THEN
            RAM(a) <= MemNext;
        END IF;
    END IF;
END PROCESS;
Mem <= RAM(a);
----- E80 -----
a <= int(MemAddr);
DFF_Array: FOR i IN 0 TO 255 GENERATE
    DFF8 : ENTITY work.DFF8 PORT MAP(CLK, RAMnext(i), RAM(i));
    RAMnext(i) <= MemNext WHEN MemWriteEn = '1' AND i = a ELSE RAM(i);
END GENERATE;
Mem <= RAM(a);
```

Ο κώδικας της Harris βασίζεται στο αφηρημένο συμπεριφορικό μοντέλο όπως καταγράφεται στο ανώτατο στάδιο της Εικόνας 4 (σ. 9) στο Υπόβαθρο. Ο κώδικας του E80 βασίζεται στο υβριδικό δομικό-συμπεριφορικό μοντέλο που καταγράφεται στο ενδιάμεσο στάδιο της ίδιας εικόνας.

Σε πρακτικές υλοποιήσεις η προσέγγιση της Harris έχει σημαντικά πλεονεκτήματα: όχι μόνο είναι απλούστερη, αλλά βασίζεται και στα πρότυπα υλοποίησης που επιτρέπουν στον compiler να χρησιμοποιήσει εσωτερικά στοιχεία μνήμης αντί για flip flop (Altera, 2013, σ. 818). Τα πλεονεκτήματα αυτά όμως δεν έχουν σημασία ως προς τους στόχους του E80. Για παράδειγμα, μια ερώτηση που θα μπορούσε να κάνει ο εκπαιδευόμενος που έχει ολοκληρώσει τις ενότητες στοιχειώδους ψηφιακής σχεδίασης θα μπορούσε να είναι:

*“Η ανάθεση στη RAM(a) γίνεται αν MemWriteEn=1. Από πού θα πάρει το RAM(a) τιμή αν δεν ισχύει MemWriteEn=1 ; Θα κρατήσει την παλιά του τιμή ; Που φαίνεται αυτό ;”*

Η δομή των flip flop του E80 αποκαλύπτει άμεσα τον μηχανισμό διατήρησης κατάστασης καθώς η ανάθεση στο RAMnext(i) γίνεται μέσω πολυπλέκτη (WHEN) 2 σε 1 και η αποθήκευση γίνεται στην οικεία δομή του flip flop.

Αυτό δεν σημαίνει ότι ο κώδικας του E80 είναι απόλυτα δομικός· η δημιουργία των flip flop και η αναφορά σε αυτά μέσω του ακεραίου δείκτη a, είναι συμπεριφορική. Μια καθαρά δομική προσέγγιση θα ήταν η συγγραφή μιας λίστας 256 flip flop και η επιλογή τους με πολυπλέκτες. Κάτι τέτοιο θα οδηγούσε σε εξαιρετικά μακροσκελή και δυσνόητο κώδικα.

### 5.7.1. Φόρτωμα προγράμματος στην μνήμη μέσω Reset

Όπως φαίνεται στο παρακάτω απόσπασμα του κώδικα της μνήμης, όταν γίνεται Reset τα δεδομένα απο την σταθερά Firmware φορτώνονται στα flip flop:

```
USE ieee.std_logic_1164.ALL, work.support.ALL, work.firmware.ALL;
...
DFF_Array: FOR i IN 0 TO 255 GENERATE
  DFF8 : ENTITY work.DFF8 PORT MAP(CLK, RAMnext(i), RAM(i));
  RAMnext(i) <=
    Firmware(i) WHEN Reset = '1'
    ELSE
```

Η σταθερά αυτή είναι πίνακας 256 λέξεων και βρίσκεται στην βιβλιοθήκη Firmware.vhd. Αυτή η μέθοδος μειονεκτεί ως προς την άμεση αρχικοποίηση του προγράμματος στην μνήμη κατα την έναρξη της εκτέλεσης επειδή απαιτεί ένα υποχρεωτικό Reset μετά την εκκίνηση. Προσφέρει όμως τρία σημαντικά πλεονεκτήματα:

- Επιτρέπει την επανεκτέλεση του προγράμματος με διαδοχικά Reset για πολλαπλές δοκιμές.
- Διαχωρίζει τον κώδικα του προγράμματος απο τον κώδικα της μνήμης, με αποτέλεσμα να είναι πιο εύκολη η δημιουργία του απο τον συμβολομεταφραστή.
- Το ξεχωριστό αρχείο Firmware/προγράμματος είναι πιο ευανάγνωστο γιατί περιέχει αποκλειστικά τα δεδομένα του προγράμματος και η κατασκευή του ως βιβλιοθήκη (αντί για οντότητα) επιτρέπει την απαλοιφή επιπλέον δηλώσεων.

Η δημιουργία του Firmware γίνεται απο τον συμβολομεταφραστή του E80 και παρουσιάζεται αναλυτικά στην Ενότητα 7.4.7 (σ. 93).

### 5.7.2. Διεύθυνση εισόδου δεδομένων (DIP input)

Αν και η μνήμη περιέχει 256 λέξεις, η διεύθυνση 0xFF είναι δεσμευμένη για την είσοδο απο τους 8 διακόπτες DIP, όπως φαίνεται στην κάτωθι δήλωση του αρχείου Computer.vhd:

```
Data <= DIPinput WHEN match(MemAddr,x"FF") ELSE Mem;
```

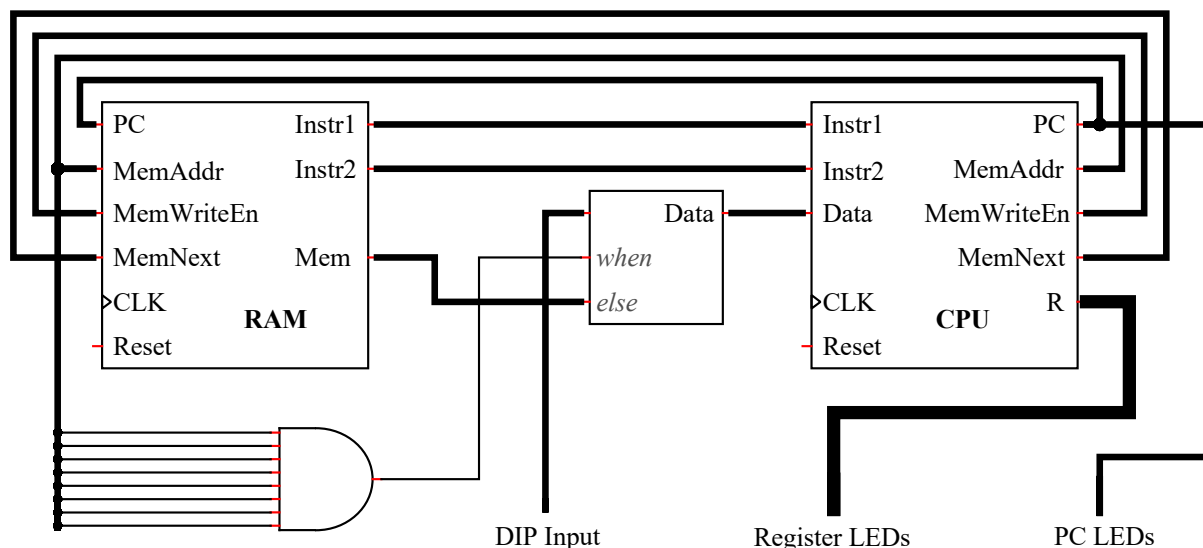
Αυτό σημαίνει οτι το τελευταίο flip flop της διάταξης είναι πλεονάζον καθώς μπορεί να γραφτεί αλλά δεν διαβάζεται. Η αφαίρεσή του και ο περιορισμός της μνήμης σε 255 λέξεις αύξησε σημαντικά την πολυπλοκότητα του κώδικα, με επιπλέον συνθήκες διευθυνσιοδότησης, και τελικά επιλέχθηκε η διατήρηση της πλεονάζουσας κυψελίδας.

## 5.8. Υπολογιστής

Πριν αναλυθεί η κεντρική μονάδα επεξεργασίας, θα είναι χρήσιμο να παρουσιαστεί η διασύνδεσή της με την μνήμη και την είσοδο δεδομένων σε έναν Υπολογιστή. Ο πλήρης κώδικας καταγράφεται στο Παράρτημα B.3 (σ. 155) και το κυκλωματικό διάγραμμά του απο το Quartus στην Εικόνα 67 του Παραρτήματος Γ (σ. 166). Πιο χρήσιμο είναι το διάγραμμα που έχει σχεδιαστεί στην Εικόνα 20 που ακολουθεί.

Όπως προκύπτει απο το διάγραμμα, ο E80 ακολουθεί την αρχιτεκτονική Neumann (Ενότητα 2.2.1, σ. 5) με διακριτό υποσύστημα μνήμης, είσοδο, έξοδο, και ένα ενοποιημένο τμήμα CPU που συνδυάζει τα Central Control και Central Arithmetic.

*Σημειώνεται οτι η γραμμή R εξάγει το πλήρες αρχείο καταχωρητών προς τα LED της FPGA.*



Εικόνα 20: Υπολογιστής E80

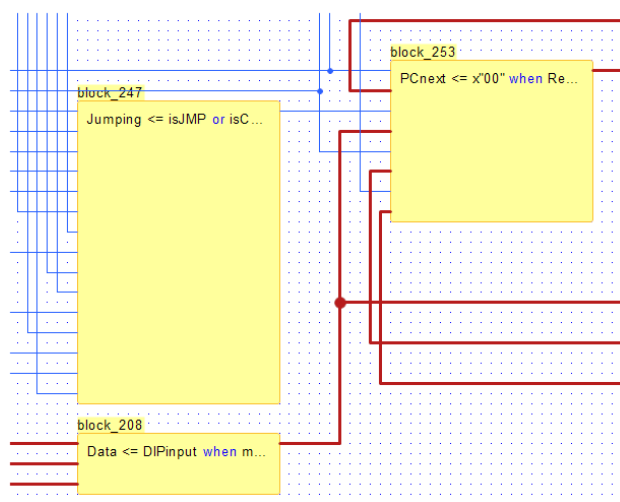
Η επιλογή της εισόδου δεδομένων προς την CPU γίνεται μέσω πολυπλέκτη και πύλης AND έτσι ώστε η διεύθυνση 255 να αντιστοιχεί με τους 8 διακόπτες του DIP. Ο αντίστοιχος κώδικας VHDL είναι ο εξής:

```
Data <= DIPinput WHEN match(MemAddr,x"FF") ELSE Mem;
```

### 5.8.1. Διαγραμματική αναπαράσταση της επιλογής WHEN

Όπως σημειώθηκε προηγουμένως, η παρουσίαση της διόδου δεδομένων και του ελέγχου του E80 θα γίνει μέσω διαγραμμάτων σε διανυσματική μορφή (για απεριόριστο zoom) που κατασκευάστηκαν με το “χέρι” στο QElectroTech<sup>1</sup> όπως η Εικόνα 20.

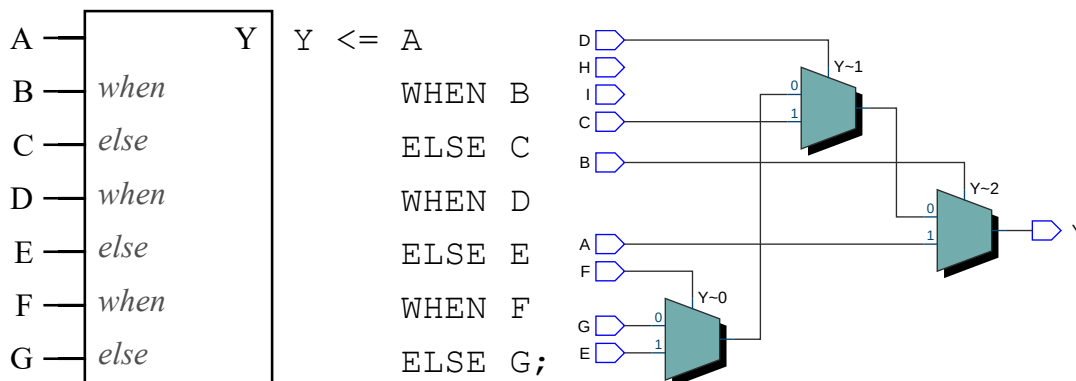
Ειδικά οι δηλώσεις WHEN, φαίνονται απλές στην VHDL αλλά μετατρέπονται σε πλήθος πολυπλεκτών στον RTL Viewer. Το Active HDL της Aldec προσφέρει μια ενδιαφέρουσα ιδέα καθώς παρουσιάζει ένα πιο ευανάγνωστο διάγραμμα, στο οποίο οι WHEN ενθυλακώνονται σε τμήματα όπως φαίνεται στην Εικόνα 21:



Εικόνα 21: Μέρος του διαγράμματος του E80 στο Active HDL της Aldec

<sup>1</sup> Ελεύθερο λογισμικό για την δημιουργία τεχνικών διαγραμμάτων, διαθέσιμο στο [qelectrotech.org](http://qelectrotech.org)

Η ιδέα αυτή υιοθετήθηκε στην παρούσα εργασία με την κατασκευή συμβόλου συστατικού στο QElectroTech που αντιστοιχεί σε μια δήλωση WHEN με ακροδέκτες όπως φαίνεται αριστερά στην Εικόνα 22. Στην εικόνα παρατίθεται ο αντίστοιχος κώδικας VHDL και δεξιά το κύκλωμα όπως έχει υλοποιηθεί με πολυπλέκτες απο το Quartus.



Εικόνα 22: Σύμβολο επιλογής WHEN (αριστερά) και η υλοποίησή του με πολυπλέκτες

Απο την σύγκριση των δύο σχημάτων φαίνεται το πλεονέκτημα που έχει (ως προς την αναγνωσιμότητα) η πιο αφηρημένη προσέγγιση που υλοποιείται στην εργασία αυτή.

## 5.9. CPU

Ο κώδικας της κεντρικής μονάδας επεξεργασίας που συνδυάζει τα προηγούμενα υποσυστήματα καταγράφεται στο Παράρτημα B.7 (σ. 158) και έχει εξέχουσα σημασία στην εργασία αυτή. Για την παρουσίαση της λειτουργίας του θα αναλυθεί το διάγραμμα της διόδου δεδομένων και του ελέγχου ως προς την εκτέλεση αντιπροσωπευτικών εντολών.

### 5.9.1. Αποκωδικοποίηση (Instruction Decoder)

Ο αποκωδικοποιητής των εντολών του E80 όπως φαίνεται στο παρακάτω τμήμα, αντιστοιχεί σήματα τύπου *isX* για όλες τις εντολές εξαιρώντας τις εντολές που έχουν δύο τελούμενα και εκτελούν μια τυπική λειτουργία της ALU.

```
isHLT <= match(Instr1,"00000000");
isNOP <= match(Instr1,"00000001");
isJMP <= match(Instr1,"0000001-");
isJMPPr <= match(Instr1,"00000011"); -- JMP reg
isJC <= match(Instr1,"00000100");
isJNC <= match(Instr1,"00000101");
isJZ <= match(Instr1,"00000110");
isJNZ <= match(Instr1,"00000111");
isJS <= match(Instr1,"00001010");
isJNS <= match(Instr1,"00001011");
isJV <= match(Instr1,"00001100");
isJNV <= match(Instr1,"00001101");
isCALL <= match(Instr1,"00001110");
isRETURN <= match(Instr1,"00001111");
isSTORE <= match(Instr1,"1000----");
isSTOREr <= match(Instr1,"10001000"); -- STORE reg1,reg2
isLOAD <= match(Instr1,"1001----");
isLOADr <= match(Instr1,"10011000"); -- LOAD reg1,reg2
isSHIFT <= match(Instr1,"10100---") OR -- RSHIFT
            match(Instr1,"11000---"); -- LSHIFT
isPUSH <= match(Instr1,"11100---");
isPOP <= match(Instr1,"11110---");
isStack <= isPUSH OR isCALL OR isPOP OR isRETURN;
```

Οι εντολές που δεν αποκωδικοποιούνται είναι οι MOV, ADD, SUB, ROR, AND, OR, XOR, CMP, BIT. Οι εντολές αυτές λειτουργούν με την προεπιλεγμένη δίοδο δεδομένων και η κωδικοποίησή τους συμπεριλαμβάνει τον κωδικό λειτουργίας της ALU, άρα δεν χρειάζονται ειδική μέριμνα. Αυτό θα φανεί στην παρουσίαση της εντολής ADD που θα ακολουθήσει.

Σημειώνεται ότι τα σήματα isJMP/isJMP<sub>r</sub>, isSTORE/isSTORE<sub>r</sub> και isLOAD/isLOAD<sub>r</sub> αφορούν στις περιπτώσεις που το ευέλικτο 2ο τελούμενο (op2) των εν λόγω εντολών είναι άμεση τιμή ή καταχωρητής.

### 5.9.2. Σήματα ελέγχου

Η ανάθεση των σημάτων ελέγχου γίνεται σύμφωνα την κωδικοποίησή τους όπως έχει οριστεί στην Ενότητα 4.3 (σ. 36).

```

ALIAS op2isReg    : STD_LOGIC IS Instr1(3);
ALIAS Instr1Reg   : REG_ADDR IS Instr1(2 DOWNTO 0);
ALIAS Instr2Reg1  : REG_ADDR IS Instr2(6 DOWNTO 4);
ALIAS Instr2Reg2  : REG_ADDR IS Instr2(2 DOWNTO 0);
CONSTANT FlagsRegister : REG_ADDR := "110"; -- R6
CONSTANT StackPointer : REG_ADDR := "111"; -- R7
ATTRIBUTE syn_keep : BOOLEAN; -- FPGA fix, see MemAddr assignment
ATTRIBUTE syn_keep OF ALUinB : SIGNAL IS TRUE;
SIGNAL Size        : WORD;
SIGNAL Adjacent    : WORD;
SIGNAL Jumping     : STD_LOGIC;
SIGNAL PCnext      : WORD;

```

Ακολουθεί μια επιπλέον ερμηνεία σε επιλεγμένα σήματα:

- Ψευδώνυμο **op2isReg**: Είναι το 4ο ΛΣΨ του τμήματος Instr1. Αφορά στις εντολές που δεν αποκωδικοποιούνται στον Instruction Decoder και τις LOAD και STORE. Αν op2isReg=1, τότε το 2ο τελούμενο είναι καταχωρητής, αλλιώς είναι άμεση τιμή ή κατευθείαν διεύθυνση.
- Ψευδώνυμο **Instr1Reg**: Διεύθυνση καταχωρητή που βρίσκεται στα 3 ΛΣΨ του τμήματος Instr1. Για παράδειγμα στις εντολές PUSH R3 και ADD R3, 10, ισχύει Instr1Reg=011.
- Ψευδώνυμα **Instr2Reg1**, **Instr2Reg2**: Οι καταχωρητές που βρίσκονται στο 2ο μέρος της εντολής. Για παράδειγμα στην εντολή ADD R3, R4 ισχύει Instr2Reg1=011 και Instr2Reg2=100.
- Σταθερές **FlagsRegister**, **StackPointer**: Διευθύνσεις 110 (R6) και 111 (R7).
- Ρύθμιση **syn\_keep**: Εμποδίζει τους μεταγλωττιστές VHDL να βελτιστοποιήσουν την διέλευση του σήματος ALUinB όπως εξηγείται στην Ενότητα 9.9 (σ. 127).
- Λέξη **Size**: Μέγεθος της τρέχουσας εντολής, 0 (για την HLT), 1 ή 2.
- Λέξη **Adjacent**: Η διεύθυνση της παρακείμενης εντολής: Adjacent=PC+Size.
- Σήμα **Jumping**: Εκφράζει αν γίνεται άλμα (Jumping=1) ή όχι.
- Λέξη **PCnext**: Η διεύθυνση της εντολής που θα εκτελεστεί στον επόμενο κύκλο. Αν Jumping=0 τότε PCnext=Adjacent.

### 5.9.3. Έλεγχος ροής και ο μετρητής προγράμματος

Ο έλεγχος της ροής του προγράμματος παρουσιάζεται συνοπτικά στην Εικόνα 23 και πιο αναλυτικά στον σχολιασμένο κώδικα που ακολουθεί.

Απο τα σήματα του αποκωδικοποιητή επιλέγεται η τιμή 1 ή 2 ως μέγεθος της τρέχουσας εντολής ως λέξη Size. Η λέξη αυτή δίνεται ως είσοδος στο στιγμιότυπο του αθροιστή που παρουσιάστηκε στην Ενότητα 5.3 (σ. 55). Απο τον αθροιστή υπολογίζεται η διεύθυνση της παρακείμενης εντολής Adjacent = PC+Size σύμφωνα με τις δηλώσεις:

```
Size <=
  x"01" WHEN isHLT OR isNOP OR isRETURN OR isSHIFT OR isPUSH OR isPOP ELSE
  x"02";
PC_Adder : ENTITY work.FA8 PORT MAP(PC, Size, '0', Adjacent);
```

Η ύπαρξη άλματος ορίζεται απο τα σήματα του αποκωδικοποιητή και τις σημαίες. Ο τρόπος γραφής του κώδικα μεταφράζεται άμεσα σε φυσική γλώσσα και δεν χρειάζεται ερμηνεία:

```
Jumping <=
  isJMP OR isCALL OR isRETURN OR
  (isJC AND Carry) OR (isJNC AND NOT Carry) OR
  (isJZ AND Zero) OR (isJNZ AND NOT Zero) OR
  (isJS AND Sign) OR (isJNS AND NOT Sign) OR
  (isJV AND Overflow) OR (isJNV AND NOT Overflow);
```

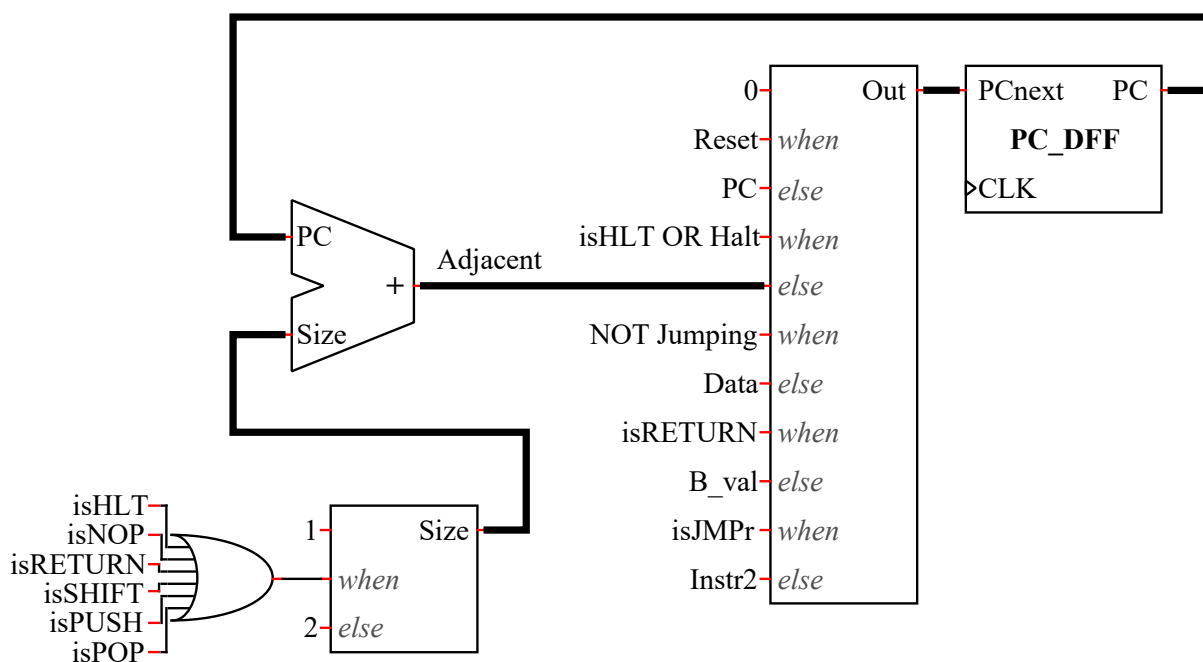
Η τιμή του μετρητή προγράμματος για τον επόμενο κύκλο (PCnext) επιλέγεται απο τις παρακάτω περιπτώσεις και αποθηκεύεται σε D flip flop:

```
PCnext <=
  x"00"      WHEN Reset      ELSE
  PC         WHEN isHLT OR Halt ELSE -- HLT works on the current cycle
  Adjacent   WHEN NOT Jumping ELSE
  Data       WHEN isRETURN   ELSE
  B_val      WHEN isJMPr      ELSE
  Instr2;
PC_DFF : ENTITY work.DFF8 PORT MAP(CLK, PCnext, PC);
```

Αναλυτικά ο υπολογισμός του PCnext γίνεται ως εξής:

- Τίθεται στο 0 αν υπάρχει Reset, οπότε το πρόγραμμα ξεκινά απο την αρχή.
- Διατηρεί την τρέχουσα τιμή του μετρητή προγράμματος αν εκτελείται εντολή HLT ή είχε ενεργοποιηθεί η σημαία Αλτ, οπότε το πρόγραμμα μπαίνει σε ατέρμονα βρόχο.
- Προχωρά στην παρακείμενη διεύθυνση που υπολογίστηκε απο τον αθροιστή, αν δεν υπάρχει άλμα οπότε γίνεται ακολουθιακή εκτέλεση. Αυτή είναι πιο συνηθισμένη περίπτωση.
- Κάνει άλμα στην λέξη Data στην οποία έχει γίνει απόθεση απο την στοίβα αν εκτελείται RETURN. Άρα επιστρέφει στην διεύθυνση που είχε αποθηκευτεί με CALL.
- Κάνει άλμα στην λέξη B\_val που είναι η τιμή του καταχωρητή - ορίσματος της JMP με έμμεση διεθυνσιοδότηση. Στο παράδειγμα της JMP R0 της Ενότητας 4.7.1 (σ. 42) θα ίσχυε B\_val=90.
- Σε κάθε άλλη περίπτωση κάνει άλμα στην κατευθείαν διεύθυνση που βρίσκεται στο 2ο τμήμα (Instr2) της εντολής που εκτελείται.

Οι λέξεις Data και Instr2 εξάγονται απο την μνήμη όπως παρουσιάστηκε στην Εικόνα 20 (σ. 66), ενώ η B\_val απο το αρχείο καταχωρητών όπως παρουσιάστηκε στην Εικόνα 18 (σ. 61).



Εικόνα 23: Διάγραμμα του ελέγχου ροής προγράμματος

#### 5.9.4. Σενάριο εκτέλεσης της NOP

Όπως τονίστηκε στην τεκμηρίωση της NOP στην Ενότητα 4.10.1 (σ. 49), η συγκεκριμένη εντολή θα χρησιμοποιηθεί για την εισαγωγή στον τρόπο λειτουργίας του E80.

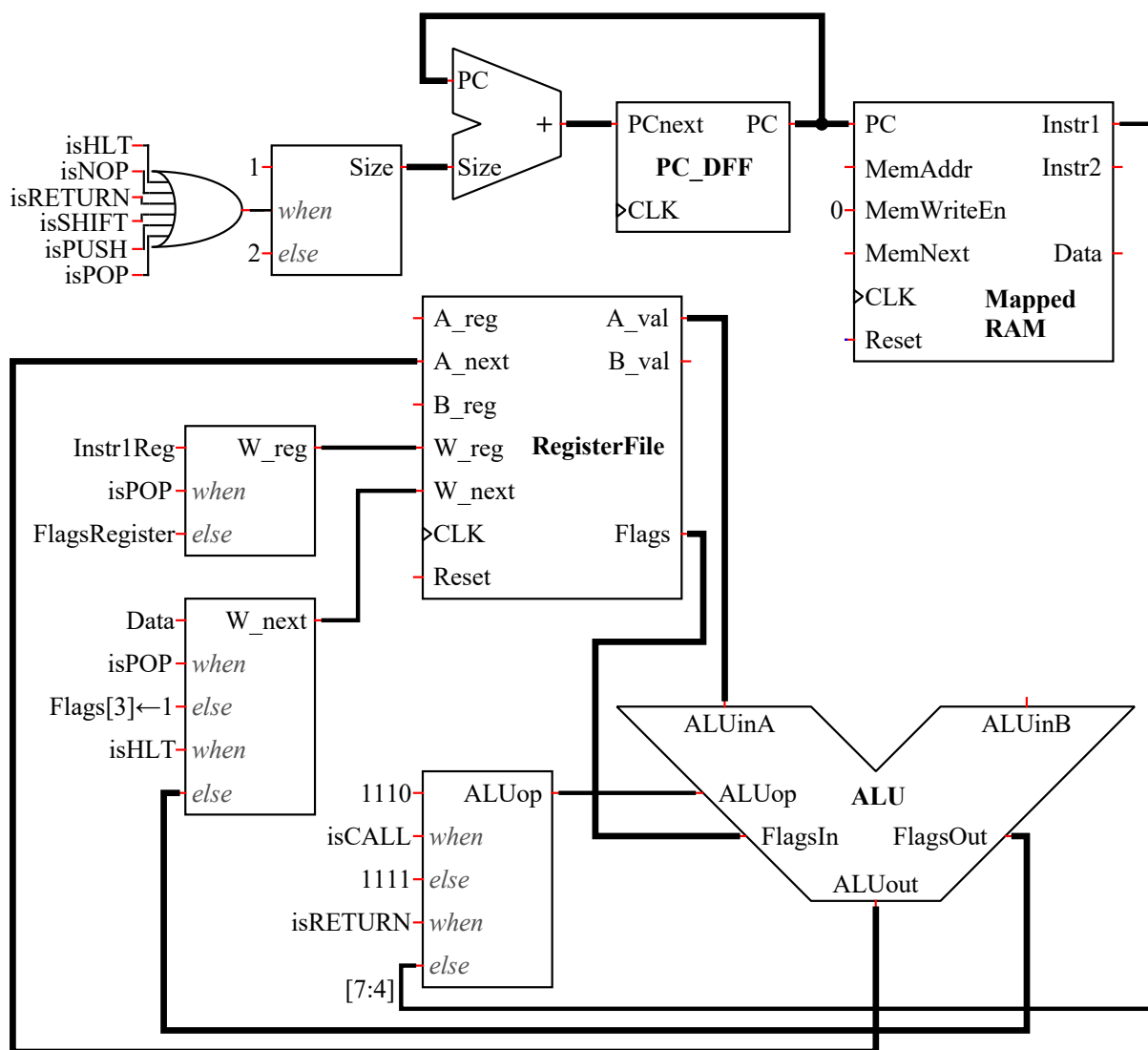
Στην Εικόνα 24 παρουσιάζεται η δίοδος δεδομένων και ο έλεγχος που πραγματοποιείται στην εκτέλεσή της. Τα αδιάφορα σήματα, όπως για παράδειγμα το Instr2 στην RAM, δεν εμφανίζονται συνδεδεμένα.

Έστω ότι στην θέση μνήμης 100 υπάρχει η λέξη “00000001” που είναι ο κωδικός της NOP.

1. Απο την μνήμη ισχύει  $[PC] = [100] = Instr1 = 00000001$ . Η Instr2 είναι αδιάφορη.
2. Απο τον αποκωδικοποιητή εντολών, ισχύει  $isNOP = 1$ .
3. Δεν ισχύει  $isCALL/isRETURN$  άρα  $ALUop = Instr[7:4] = 0000$ . Επομένως ισχύει  $isBypass = 1$  και η είσοδος ανακατευθύνεται στην έξοδο. Επομένως:
  - $ALUout = A\_val$
  - $FlagsOut = FlagsIn$
4. Οπότε δεν ενδιαφέρει η διεύθυνση του A\_reg, αφού ο εν λόγω καταχωρητής θα λάβει πίσω την τρέχουσα τιμή του.
5. Όπως φαίνεται απο τις επιλογές W\_reg & W\_next θα ισχύει  $W\_reg = FlagsRegister$  και  $W\_next = FlagsOut$ · επομένως και ο καταχωρητής σημαιών θα λάβει πίσω την τρέχουσα τιμή του.
6. Απο την επιλογή μεγέθους προκύπτει  $Size = 1$ , και επομένως  $PCnext = PC + 1 = 101$ . Άρα στον επόμενο κύκλο θα εκτελεστεί η παρακείμενη εντολή της τρέχουσας NOP.

Επομένως η NOP δεν επέφερε καμιά αλλαγή και απλά εκτελείται η επόμενη εντολή.





Εικόνα 24: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της NOP

Σημειώνεται ότι το συστατικό της μνήμης στην Εικόνα 24 καταγράφεται ως “Mapped RAM” υπονοώντας ότι έχει γίνει η πολυπλεξία της εξόδου της μνήμης με την είσοδο από τους διακόπτες DIP στο ενιαίο σύστημα Υπολογιστή όπως φάνηκε στην Εικόνα 20 (σ. 66).

Άρα στα διαγράμματα που θα ακολουθήσουν η λέξη Data θα είναι:

- Δεδομένο μνήμης αν MemAddr < 0xFF
- Είσοδος από χρήστη αν MemAddr = 0xFF

### 5.9.5. Η εκτέλεση της ADD (καταχωρητή-καταχωρητή)

Έχοντας αναλύσει την εκτέλεση της NOP, η ADD θα φανεί αρκετά πιο εύκολη. Έστω  $R1=100$ ,  $R2=200$  και εκτελείται η εντολή ADD R1,R2. Απο την ISA (Ενότητα 4.5.1, σ. 38) προκύπτει η κωδικοποίηση 00101000 00010010.

Όπως φαίνεται στην Εικόνα 25, οι αλλαγές σε σχέση με τη NOP αφορούν στα εξής:

1. Σύμφωνα με την κωδικοποίηση της ADD με κατευθείαν διευθυνσιοδότηση καταχωρητή-καταχωρητή, ισχύει:

- $[PC] = Instr1 = 00101000$
- $[PC+1] = Instr2 = 00010010$
- $op2isReg = Instr1[3] = 1$ . Σημειώνεται ότι  $[0] = ΛΣΨ$  και  $[7] = ΠΣΨ$ .
- $Instr2Reg1 = Instr2[6:4] = 001$  (R1)
- $Instr2Reg2 = Instr2[2:0] = 010$  (R2)

Επομένως,  $A\_reg = 001$ ,  $B\_reg = 010$

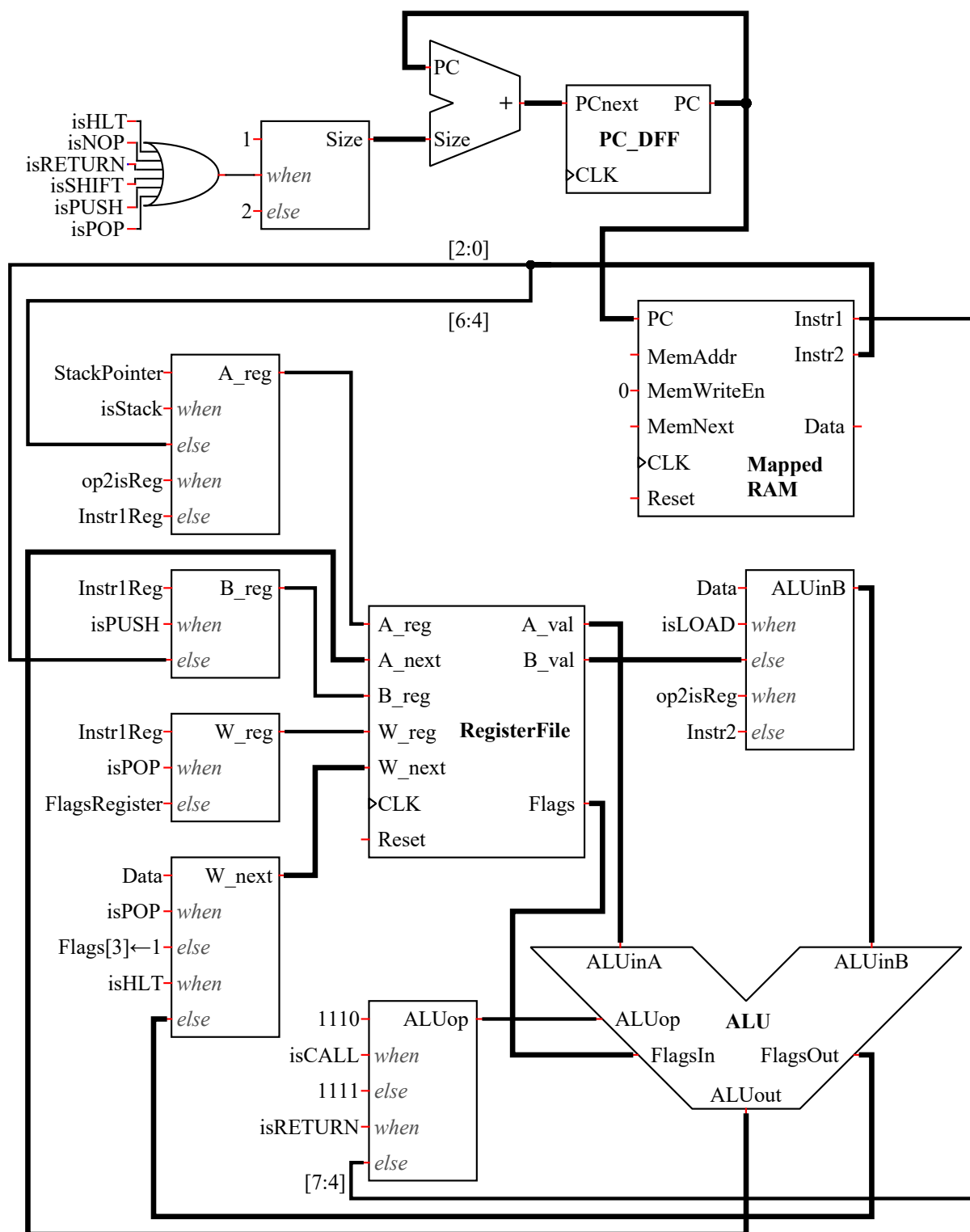
2. Όλες οι συνθήκες  $isX...$  του αποκωδικοποιητή είναι ψευδείς. Αυτό σημαίνει ότι στις περισσότερες περιπτώσεις επιλογών WHEN θα ισχύουν οι τελικές *else*, δηλαδή:

- $A\_reg = Instr2Reg1 = 001$  (R1)
- $A\_val = \text{τιμή του } R1 = 100$
- $B\_reg = Instr2Reg2 = 010$  (R2)
- $B\_val = \text{τιμή του } R2 = 200$
- $ALUop = Instr1[7:4] = 0010$

Το συγκεκριμένο διάνυσμα  $ALUop$  αντιστοιχεί με  $isADD$  στον αποκωδικοποιητή της ALU, οπότε υπολογίζεται  $ALUout = R1 + R2 = 300 \bmod 2^8 = 44$  με κρατούμενο  $C=1$  που εξάγεται στην λέξη  $FlagsOut$ .

3.  $A\_next = ALUout$ , άρα η επόμενη τιμή του R1 θα είναι 44.
4.  $W\_reg = 110$  (R6 / FlagsRegister) και  $W\_next = FlagsOut$ , άρα οι σημαίες θα ενημερωθούν με  $CZSV = 1000$ .
5. Απο την επιλογή μεγέθους, ισχύει  $Size = 2$ . Άρα  $PCnext = PC+2$ , οπότε στον επόμενο κύκλο ο Program Counter θα δείχνει στην παρακείμενη της ADD.

Επομένως έγινε η πρόθεση του R2 στον R1, ενεργοποιήθηκε η σημαία κρατουμένου, μηδενίστηκαν οι υπόλοιπες σημαίες και η εκτέλεση συνέχισε ακολουθιακά.



Εικόνα 25: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της ADD (register-register)

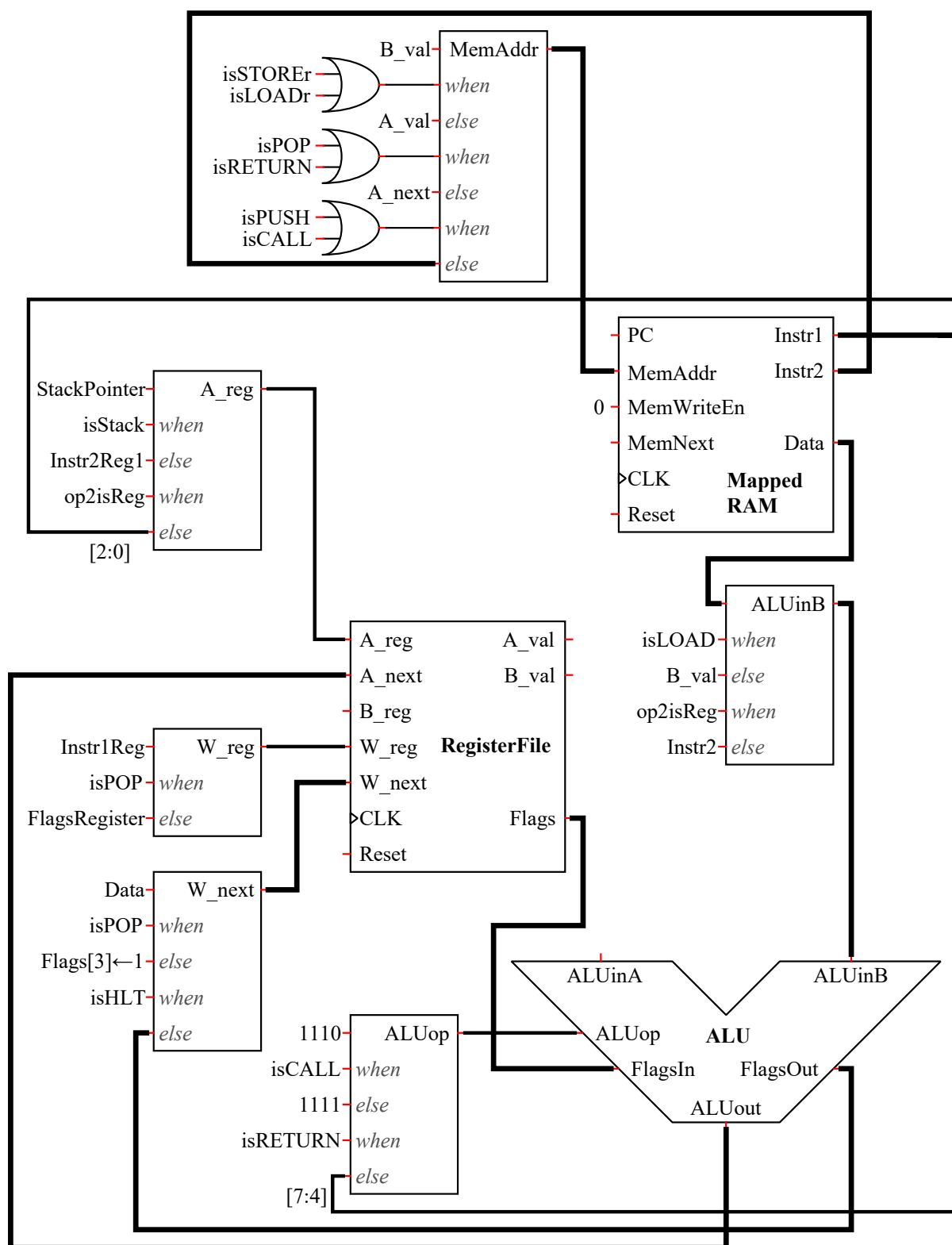
### 5.9.6. Η εκτέλεση της LOAD με κατευθείαν διεύθυνση

Έστω ότι εκτελείται LOAD με κατευθείαν διεύθυνση, για παράδειγμα LOAD R5, [131] και [131]=10101010. Θα μελετηθούν τα σήματα ελέγχου και η δίοδος δεδομένων όπως καταγράφονται στην Εικόνα 26. Ο μετρητής προγράμματος δεν έχει συμπεριληφθεί αφού λειτουργεί ακριβώς όπως στην ADD που αναλύθηκε προηγουμένως.

1. Απο την κωδικοποίηση της LOAD με κατευθείαν διεύθυνση (Ενότητα 4.4.2, σ. 37), ισχύει:
  - [PC] = Instr1 = 10010101
  - [PC+1] = Instr2 = 10000011
  - op2isReg = Instr1[3] = 0
  - Instr1Reg = Instr1[2:0] = 101 (R5)
2. Απο τον αποκωδικοποιητή τίθεται isLOAD=1 (αλλά isLOADr=0).
3. Απο την επιλογή διεύθυνσης μνήμης (MemAddr) ισχύει η τελική περίπτωση *else*, άρα MemAddr = Instr2 = 131.
4. Απο την μνήμη εξάγεται Data = [MemAddr] = [131] = 10101010.
5. Δοθέντος του isLOAD, ισχύει ALUinB = Data = 10101010.
6. Στις επιλογές A\_reg, και ALUop ισχύουν οι τελικές *else*, οπότε:
  - A\_reg = Instr1[2:0] = 101
  - ALUop = Instr1[7:4] = 1001

Απο το διάνυσμα ALUop, ο αποκωδικοποιητής της ALU θέτει isAssign = 1. Άρα το ALUinA είναι αδιάφορο και ισχύει ALUout = ALUinB = 10101010 και οι ενημερωμένες σημαίες Z=0 και S=1 εξάγονται στην FlagsOut.
7. A\_next = ALUout, στον R5 φορτώνεται η 10101010.
8. Στις επιλογές W\_reg και W\_next ισχύουν και πάλι οι τελικές *else*, οπότε:
  - W\_reg = FlagsRegister = 110
  - W\_next = FlagsOut άρα στις σημαίες ZS αποθηκεύεται 01
9. Απο την επιλογή μεγέθους, ισχύει Size = 2. Άρα PCnext = PC+2, οπότε στον επόμενο κύκλο ο Program Counter θα δείχνει στην παρακείμενη εντολή.

Επομένως στον καταχωρητή R5 φορτώθηκε η λέξη 10101010 απο την διεύθυνση 131. Οι σημαίες Z και S τέθηκαν στο 0 και 1 αντίστοιχα γιατί ο αριθμός που φορτώθηκε είναι μη-μηδενικός με αρνητικό πρόσημο.



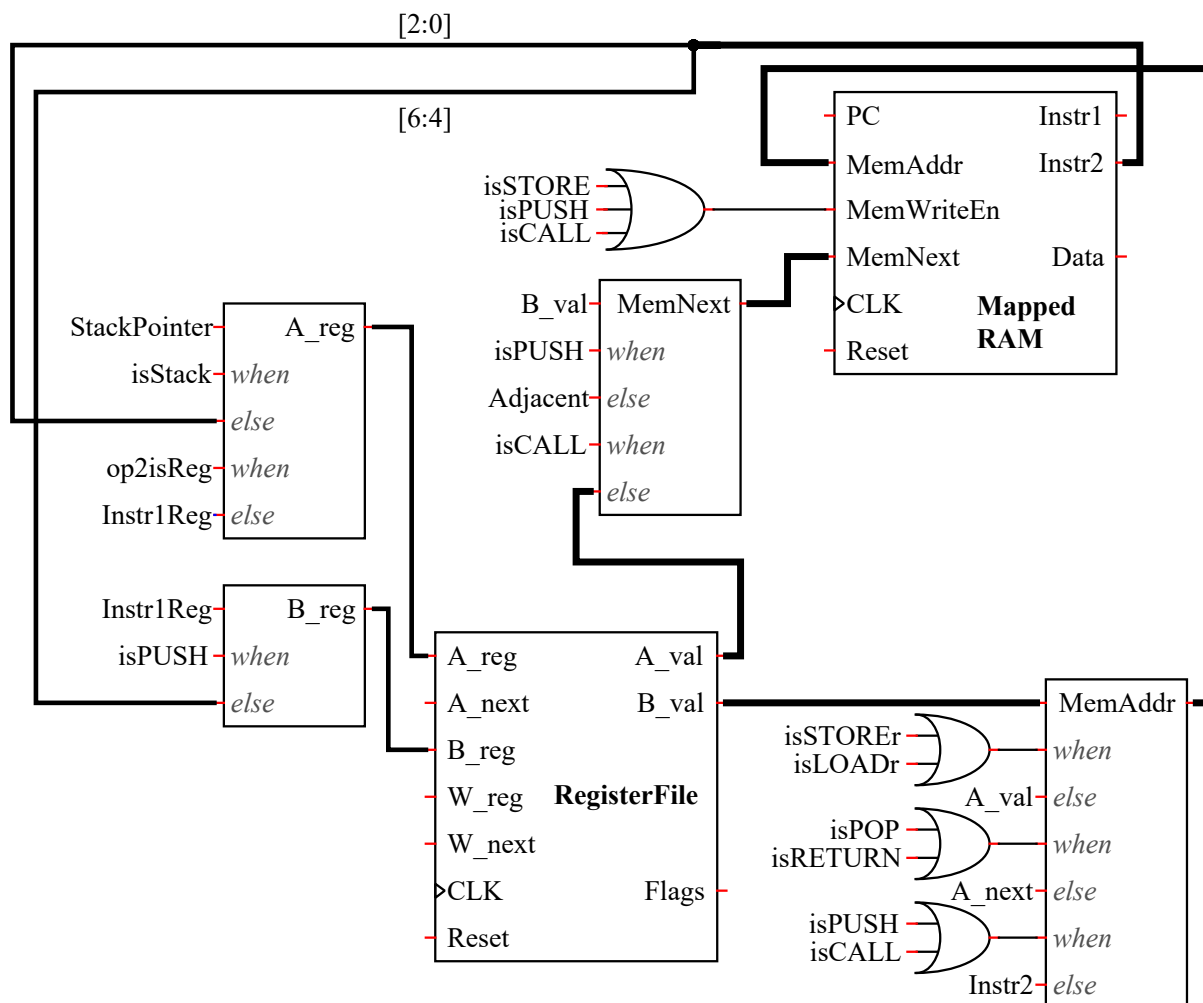
Εικόνα 26: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της **LOAD (register-direct)**

### 5.9.7. Η εκτέλεση της STORE με έμμεση διεύθυνση με χρήση καταχωρητή

Έστω  $R5=10$ ,  $R2=100$  και εκτελείται η εντολή `STORE R5, [R2]`. Θα μελετηθούν τα σήματα ελέγχου και η δίοδος δεδομένων όπως καταγράφονται στην Εικόνα 27. Η ALU δεν έχει συμπεριληφθεί στην εικόνα επειδή πρακτικά παρακάμπτεται όπως είχε γίνει στην `NOP`.

1. Απο την κωδικοποίηση της `STORE` με έμμεση διεύθυνση με χρήση καταχωρητή (Ενότητα 4.4.3, σ. 38), ισχύει:
  - $[PC] = Instr1 = 10001000$
  - $[PC+1] = Instr2 = 01010010$
  - $op2isReg = Instr1[3] = 1$
  - $Instr2Reg1 = Instr2[6:4] = 101$
  - $Instr2Reg2 = Instr2[2:0] = 010$
2. Απο τον αποκωδικοποιητή τίθεται `isSTORE` και `isSTOREr`.
3. Ισχύει  $ALUop = Instr1[7:4] = 1000$ . Άρα στον αποκωδικοποιητή της ALU ισχύει  $isBypass = 1$  οπότε δεν γίνεται μεταβολή καταχωρητή ή σημαιών. Επομένως τα διανύσματα που αφορούν σε υπολογισμούς και αποτελέσματα της ALU δεν θα μας απασχολήσουν.
4. Δοθέντος του  $op2isReg = 1$ , ισχύει  $A\_reg = Instr2Reg1 = 101$  ( $R5$ ), άρα  $A\_val = 10$ .
5. Στην επιλογή `MemNext` ισχύει η τελική *else*, άρα  $MemNext = A\_val = 10$ .
6. Στην επιλογή  $B\_reg$  ισχύει η *else*, άρα  $B\_reg = Instr2Reg2 = 010$  ( $R2$ ). Άρα  $B\_val = 100$ .
7. Δοθέντος του  $isSTOREr = 1$ , ισχύει  $MemAddr = B\_val = 100$ .
8. Στην είσοδο `MemWriteEn` τίθεται 1 αφού ισχύει `isSTORE`. Άρα στην διεύθυνση `MemAddr` γράφεται η τιμή `MemNext`, δηλαδή  $[100] = 10$ .
9. Απο την επιλογή μεγέθους, ισχύει  $Size = 2$ . Άρα  $PCnext = PC+2$ , οπότε στον επόμενο κύκλο ο Program Counter θα δείχνει στην παρακείμενη εντολή.

Επομένως έχει αποθηκευτεί η τιμή του καταχωρητή  $R5$  (10) στην διεύθυνση που ορίστηκε έμμεσα απο τον καταχωρητή  $R2$  (100) και η εκτέλεση συνεχίζει στην επόμενη εντολή.



Εικόνα 27: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της STORE (register indirect)

Παρατίθεται και το σχετικό τμήμα κώδικα που καταγράφει τις αναθέσεις που αφορούν στην προσπέλαση της μνήμης:

```
MemAddr <=
    B_val    WHEN isSTOREr OR isLOADr ELSE -- Instr2Reg2
    A_val    WHEN isPOP OR isRETURN  ELSE -- SP before increase
    A_next   WHEN isPUSH OR isCALL    ELSE -- SP after decrease
    Instr2;  -- STORE / LOAD direct
-- set MemWriteEn for all "> [...]" operations in the ISA Cheatsheet
MemWriteEn <= isSTORE OR isPUSH OR isCALL;
-- if MemWriteEn, [MemAddr] ← MemNext
MemNext <=
    B_val    WHEN isPUSH ELSE -- push the value of B_reg
    Adjacent WHEN isCALL ELSE -- push the RETURN address
    A_val;
```



### 5.9.8. Η εκτέλεση της CALL

Έστω ότι εκτελείται η εντολή CALL 0xA1 από την διεύθυνση μνήμης 100 και ο δείκτης στοίβας (R7) έχει τιμή 200. Θα μελετηθούν τα σήματα ελέγχου και η δίοδος δεδομένων όπως καταγράφονται στην Εικόνα 28.

1. Από την κωδικοποίηση της CALL (Ενότητα 4.7.10, σ. 46) ισχύουν:
  - $[PC] = [100] = Instr1 = 00001110$
  - $[PC+1] = [101] = Instr2 = 0xA1$
2. Από τον αποκωδικοποιητή τίθεται  $isCALL = 1$  και  $isStack = 1$ .
3. Εφόσον  $isStack = 1$ , θα ισχύει  $A\_reg = StackPointer = 111$  (R7). Άρα  $A\_val = 200$ .
4. Εφόσον  $isCALL = 1$ , θα ισχύει  $ALUop = 1110$  που αντιστοιχεί με την πράξη  $isDCR$ , η οποία εφαρμόζει μείωση κατά 1 χωρίς μεταβολή σημαιών.
5. Άρα  $ALUout = A\_val - 1 = 199$ .
6. Ισχύει  $A\_next = ALUout$ , άρα αποθηκεύεται το 199 στον R7.
7. Εφόσον  $isCALL = 1$  θα ισχύουν:
  - $MemAddr = ALUout = 199$
  - $MemWriteEn = 1$
  - $MemNext = Adjacent = PC + Size = PC + 2 = 102$ .

Άρα ωθείται στην θέση μνήμης 199 η λέξη 102 που είναι διεύθυνση της παρακαίμενης εντολής της CALL.

8. Εφόσον  $isCALL = 1$ , από τον παρακάτω κώδικα θα ισχύει  $Jumping = 1$ :

```
Jumping <=
  isJMP OR isCALL OR isRETURN OR
  (isJC AND Carry) OR (isJNC AND NOT Carry) OR
  (isJZ AND Zero) OR (isJNZ AND NOT Zero) OR
  (isJS AND Sign) OR (isJNS AND NOT Sign) OR
  (isJV AND Overflow) OR (isJNV AND NOT Overflow);
```

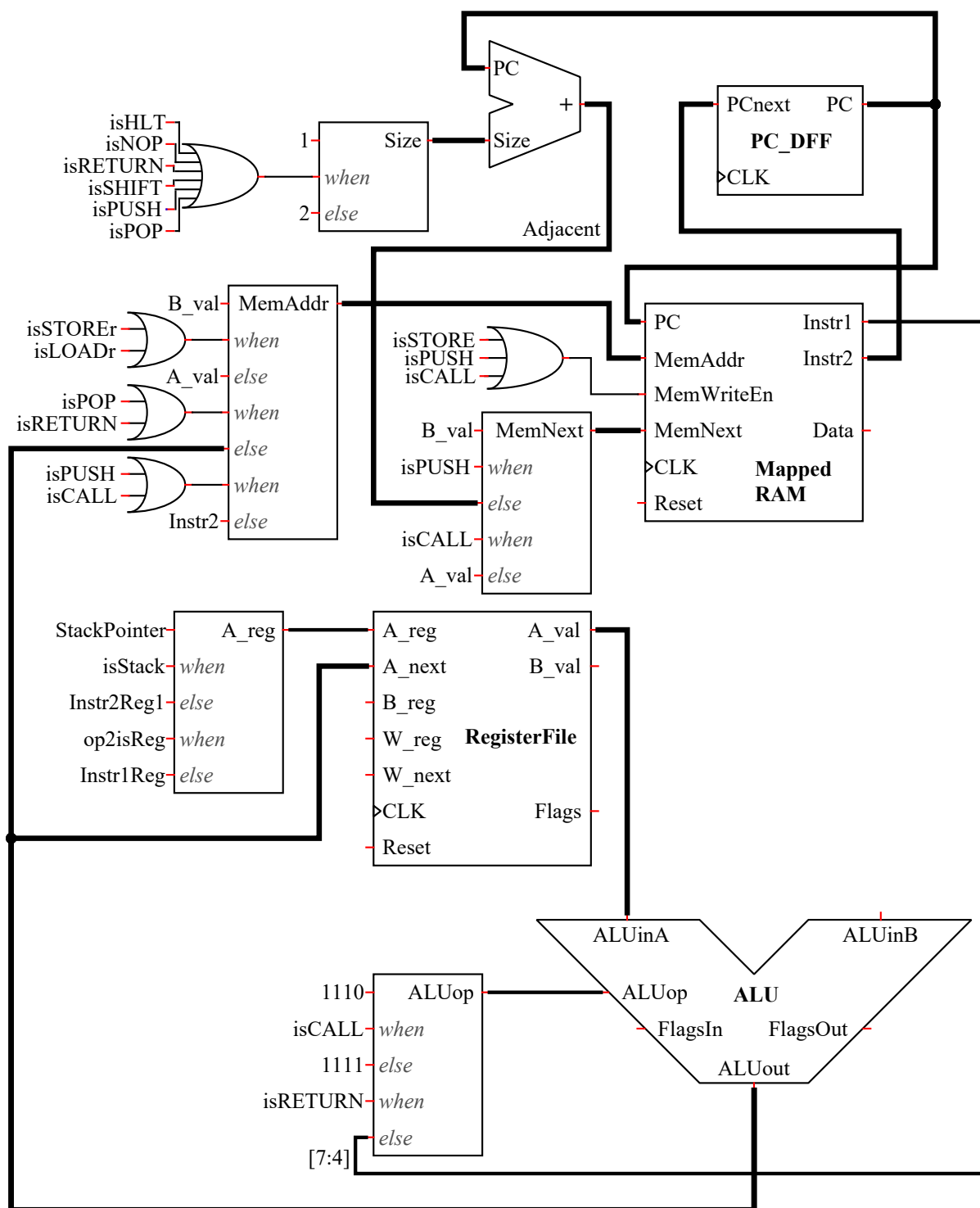
9. Επομένως, σύμφωνα με την περιγραφή του μετρητή προγράμματος στην Ενότητα 5.9.3 (σ. 69) θα ισχύσει  $PCnext = Instr2 = 0xA1$ .

Άρα, ο δείκτης στοίβας μειώθηκε κατά ένα, έγινε ώθηση της διεύθυνσης 102 που είναι παρακαίμενη στην CALL, και η εκτέλεση προχώρησε στην διεύθυνση κλήσεως 0xA1.

Αυτό σημαίνει ότι αν εκτελεστεί εντολή RETURN και η στοίβα έχει την τρέχουσα μορφή:

1. Θα γίνει απώθηση του 102 από την θέση 199 της στοίβας
2. Θα ανατεθεί το 102 στον μετρητή προγράμματος
3. Θα συνεχιστεί η εκτέλεση από την διεύθυνση 102, αμέσως μετά την CALL.

Η επιλογή μεταξύ των λέξεων  $A\_val$  και  $ALUout$  συσχετίζεται με το αν θέλουμε να χρησιμοποιήσουμε την τρέχουσα ή την μειωμένη κατά 1 τιμή του Stack Pointer. Χρησιμοποιώντας τους συμβολισμούς της C, η  $A\_val$  αντιστοιχεί με  $SP--$ , ενώ η  $ALUout$  με  $--SP$ .



Εικόνα 28: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της CALL

### 5.9.9. Η εκτέλεση της POP

Η εντολή POP είναι η πιο σύνθετη του παρόντος σχεδιασμού, καθώς απαιτεί:

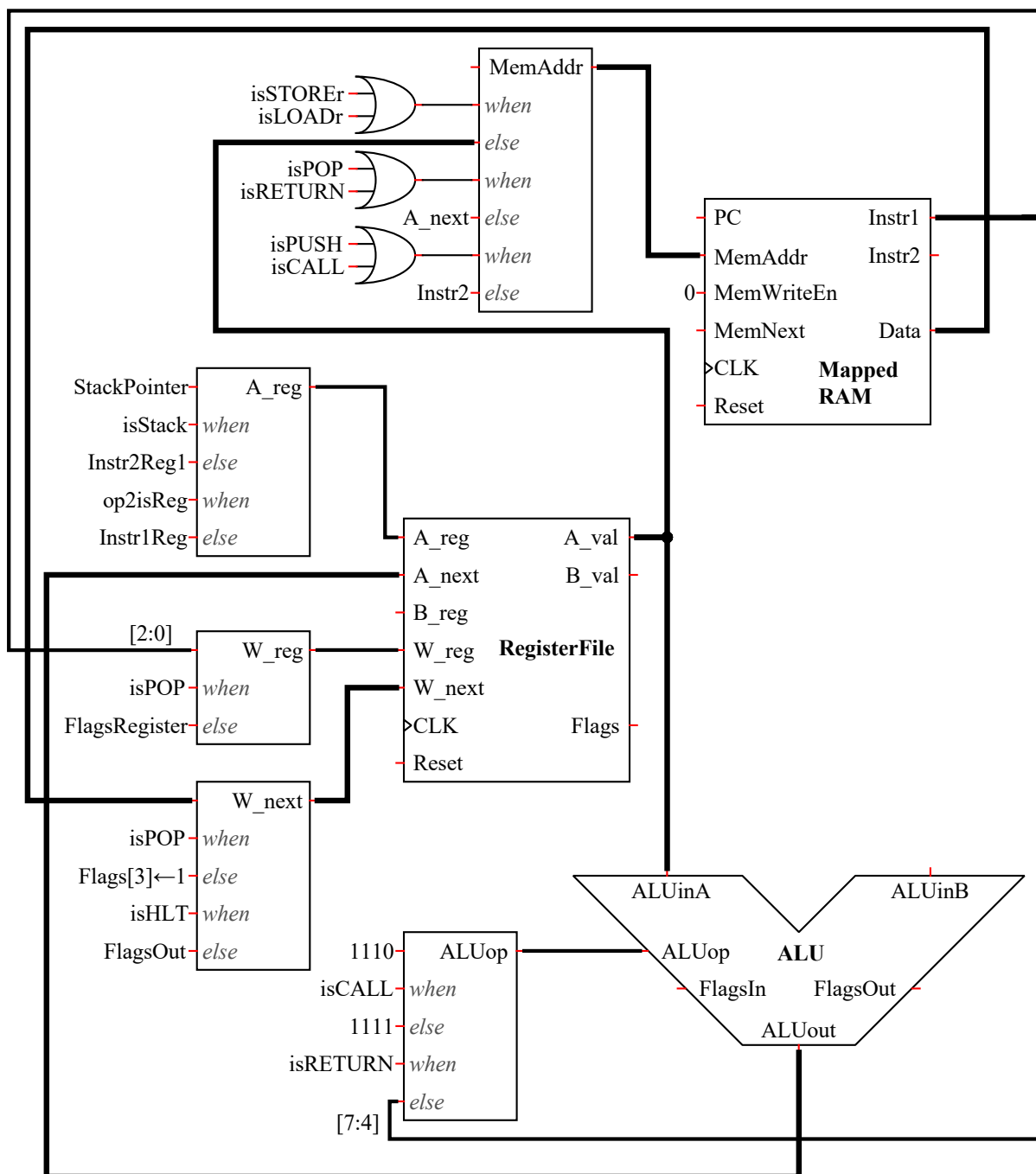
- Ανάγνωση του SP (R7)
- Προσέλαση της διεύθυνσης που περιέχει ο R7
- Εγγραφή της τιμής [R7] στον καταχωρητή - όρισμα της POP.
- Προσαύξηση του R7

Έστω ότι εκτελείται η εντολή POP R5 και τη δεδομένη στιγμή ισχύουν: R7 (SP) = 250 και [250]=15. Η διαδικασία εκτέλεσης, όπως φαίνεται στην Εικόνα 29, εξελίσσεται ως εξής:

1. Απο την κωδικοποίηση της POP (Ενότητα 4.8.2, σ. 47) ισχύουν:
  - [PC] = Instr1 = 11110101
  - Instr1Reg = Instr1[2:0] = 101
2. Απο τον αποκωδικοποιητή τίθεται isPOP = 1 και isStack = 1.
3. Αφού isStack = 1, θα ισχύει,
  - A\_reg = StackPointer = 111
  - A\_val = 250
4. Αφού isPOP=1, θα ισχύει,
  - MemAddr = A\_val = 250
  - Επομένως Data = [MemAddr] = [250] = 15.
  - W\_reg = Instr1[2:0] = Instr1Reg = 101
  - W\_next = Data = 15

Άρα θα γίνει απόθεση της τιμής 15 στον καταχωρητή R5
5. Ισχύει ALUop = Instr1[7:4] = 1111. Άρα η ALU εκτελεί αύξηση κατα 1. Επομένως ALUout = 250 + 1 = 251.
6. Δεδομένου του A\_reg = 111 (R7) και A\_next = 251, η νέα τιμή του δείκτη στοίβας θα είναι 251.

Άρα στον R5 καταχωρήθηκε το περιεχόμενο της μνήμης που έδειχνε ο δείκτης στοίβας και μετά ο δείκτης αυξήθηκε κατα ένα σύμφωνα με την οργάνωση της πλήρους καθοδικής στοίβας.



Εικόνα 29: Έλεγχος και δίοδος δεδομένων στην εκτέλεση της POP

### 5.10. Ο σχεδιασμός του D flip flop με σύγχρονο Reset χωρίς είσοδο επίτρεψης

Όπως συζητήθηκε στην Ενότητα 5.5, το θέμα του Reset έπρεπε να αναλυθεί στο τέλος του κεφαλαίου, δεδομένης της επιρροής του σε τρία υποσυστήματα. Συγκεκριμένα, όταν γίνεται Reset:

- Αντιγράφονται τα δεδομένα της βιβλιοθήκης Firmware.vhd στην μνήμη.
- Ο Program Counter τίθεται στο 0 και ο Stack Pointer στο 255.
- Η σημαία Αλτ τίθεται στο 0.

Σημειώνεται ότι οι αρχικοποιήσεις αυτές δεν γίνονται αυτόματα στην εκκίνηση της λειτουργίας του Υπολογιστή, και επομένως το χειροκίνητο Reset είναι υποχρεωτικό για να ξεκινήσει η εκτέλεση ενός προγράμματος.

Η αρχική υλοποίηση του D flip-flop έγινε με είσοδο ασύγχρονου Reset. Αυτό οδήγησε σε απρόβλεπτες επανεκκινήσεις κατά την εκτέλεση σε FPGA, επιβεβαιώνοντας την σχετική παραπομπή που έχει καταγραφεί στο Υπόβαθρο (Ενότητα 2.1.2, σ.3).

Το πρόβλημα αυτό οδήγησε στον επανασχεδιασμό του DFF αντικαθιστώντας τις εισόδους Reset και Enable με ένα σύγχρονο μηχανισμό αρχικοποίησης / επίτρεψης επι της εισόδου D με τα εξής πλεονεκτήματα:

1. Το Reset υλοποιείται σύγχρονα μέσω της εισόδου D. Η αρχικοποίηση των flip flop γίνεται με διαφορετικές τιμές ανάλογα με την περίπτωση. Για παράδειγμα σε κάποιες περιπτώσεις τα flip flop αρχικοποιούνται στο 0, σε κάποιες περιπτώσεις σε τιμές απροσδιοριστίας, ενώ στην περίπτωση του Stack Pointer στο 0xFF. Αυτό σημαίνει ότι δεν υπάρχει μια συγκεκριμένη τιμή Reset.
2. Στην ALU η έννοια της απόρριψης του αποτελέσματος εκφράζεται από την εξαγωγή του αρχικού σήματος. Αντίστοιχα, και οργανικά εντός του σχεδιασμού του E80, η λειτουργία της μη-επίτρεψης στο DFF γίνεται θέτοντας  $D \leftarrow Q$ .

Για παράδειγμα, ακολουθεί ο κώδικας του D flip flop του Program Counter:

```
PCnext <=
  x"00"    WHEN Reset      ELSE
  PC       WHEN isHLT OR Halt ELSE -- HLT works on the current cycle
  Adjacent WHEN NOT Jumping ELSE
  Data     WHEN isRETURN   ELSE
  B_val    WHEN isJMPr     ELSE
  Instr2;
PC_DFF : ENTITY work.DFF8 PORT MAP(CLK, PCnext, PC);
```

Στην περίπτωση αρχικοποίησης / Reset ισχύει  $PCnext \leftarrow 0$ , ενώ στην περίπτωση του Αλτ ισχύει  $PCnext \leftarrow PC$  το οποίο ισοδυναμεί με μη-επίτρεψη. Σε κάθε άλλη περίπτωση, το PCnext περιέχει την διεύθυνση της επόμενης εντολής προς εκτέλεση.

Επομένως, ορίζοντας το PCnext σε κάθε ενδεχόμενο καλύπτονται και τα θέματα αρχικοποίησης αλλά και της επίτρεψης σε μια και μόνο δήλωση του κώδικα VHDL χωρίς να χρειάζεται να ελέγχονται διαφορετικά σημεία του κώδικα.

## 6. Η συμβολική γλώσσα του E80

Η σχεδίαση μιας συμβολικής γλώσσας που θα καλύψει τους στόχους **Χρησιμότητας**, **Συνέχειας** και **Εφαρμογής** όπως διατυπώνονται στην εισαγωγή της εργασίας αυτής, απαιτεί περισσότερα από την βασική σύνταξη εντολών που έγινε στην Ενότητα 3.7 (σ. 30).

Όπως καταδεικνύει ο MacKenzie (1988), η χρήση συμβολικών στοιχείων, όπως οι ετικέτες και οι σταθερές, είναι χρήσιμες για τη γεφύρωση του χάσματος μεταξύ του ψευδοκώδικα και της υλοποίησής του σε χαμηλό επίπεδο.

Οι δυνατότητες και η σύνταξη της Assembly του E80 βασίστηκαν στην διερεύνηση της Ενότητας 3.7 και στις πηγές των πρωτογενών δεδομένων του Παραρτήματος Α (σ. 146). Ιδιαίτερη σημασία ως προς την κάλυψη του στόχου της **Συνέχειας**, είχαν τα παραδείγματα κώδικα MIPS που καταγράφονται στην “Εισαγωγή στη γλώσσα Assembly του επεξεργαστή MIPS” της ενότητας ΠΛΗ10 (“*Week 3-Επεξεργαστής MIPS\_June 2015.doc*”) καθώς και ο “ψευδοκώδικας” assembly που χρησιμοποιείται από τους Hayes (1998, σ. 195), τις ασκήσεις των “20” επι της αρχιτεκτονικής υπολογιστών (2013α) και τις ασκήσεις του DSMC Lab (Βασσάλος, κ.α., 2023).

### 6.1. Μορφή αριθμών

Οι αριθμοί που εκφράζουν λέξεις και διευθύνσεις στον E80 περιορίζονται στο πεδίο 0 ως 255. Χρησιμοποιείται η μορφή απροσήμεων αριθμών του GNU Assembler<sup>1</sup> με τους εξής κανόνες:

- Οι αριθμοί στο δεκαεξαδικό σύστημα ξεκινούν με πρόθεμα 0x, πχ. 0xA3
- Οι αριθμοί στο δυαδικό σύστημα ξεκινούν με πρόθεμα 0b, πχ 0b001010
- Οι αριθμοί στο δεκαδικό σύστημα γράφονται χωρίς πρόθεμα αλλά δεν μπορούν να έχουν προπορευόμενα μηδενικά. Για παράδειγμα το 01 δεν επιτρέπεται.

Δεν γίνεται χρήση των ειδικών κανόνων κεφαλαίου/πεζού προθέματος του GNU Assembler.

### 6.2. Ετικέτες

Οι ετικέτες είναι ένα βασικό χαρακτηριστικό των γλωσσών Assembly, και χρησιμοποιούνται ακόμα και στα εισαγωγικά παραδείγματα MIPS Assembly της ΠΛΗ10. Μια ετικέτα εκπροσωπεί την διεύθυνση της εντολής που ακολουθεί. Στο παράδειγμα που ακολουθεί, οι ετικέτες 1 και 2 εκφράζουν τις διευθύνσεις των αντίστοιχων εντολών, ενώ η ετικέτα 3 εκφράζει την διεύθυνση που βρίσκεται αμέσως μετά την εντολή 2:

```
label1: instruction1 ; label1 = address of instruction1
label2:                ; label2 = address of instruction2
        instruction2 ; (indentation optional)
label3:                ; label3 = address after instruction2
```

Ακολουθώντας την τυπική σύμβαση των ονομάτων στην VHDL, οι ετικέτες θα πρέπει να ξεκινάνε με γράμμα, περιέχουν γράμματα, αριθμούς και κάτω παύλα. Η δήλωση μιας ετικέτας τερματίζεται με άνω κάτω τελεία αλλά η αναφορά στις ετικέτες γίνεται **χωρίς** άνω κάτω τελεία.

<sup>1</sup> <https://sourceware.org/binutils/docs/as/Integers.html>

### 6.3. Στοίχιση και σχόλια

Η στοίχιση του κώδικα Assembly τονίζεται εμφατικά στα παραδείγματα που δίνει ο Hyde (2010) και στις ασκήσεις του DSMC lab από Βασσάλο κ.α.. Από την άλλη πλευρά όμως, τα μικρά παραδείγματα κώδικα που παρατίθενται στις εισαγωγικές ενότητες αρχιτεκτονικής, δεν έχουν στοίχιση. Για τον λόγο αυτό η στοίχιση θα υποστηριχθεί χωρίς να είναι υποχρεωτική.

Σύμφωνα με τις πηγές που αναφέρθηκαν στην εισαγωγή της ενότητας, θα υποστηριχθούν σχόλια που ξεκινούν με το σύμβολο του ελληνικού ερωτηματικού.

### 6.4. Οδηγίες

Όλες οι οδηγίες (directives) του Assembler, είναι προαιρετικές ώστε να επιτρέπονται μικρά προγράμματα που έχουν μόνο εντολές. Η δυνατότητα της άμεσης συγγραφής προγραμμάτων χωρίς την χρήση κώδικα “boiler plate” είναι χαρακτηριστικό των περισσότερων παραδειγμάτων που χρησιμοποιούνται στις πηγές που αναφέρθηκαν στην αρχή του κεφαλαίου.

#### 6.4.1. Όνομα προγράμματος .TITLE

Κατά την διάρκεια των δοκιμών φάνηκε χρήσιμο να ορίζεται ένα αναγνωριστικό στον εξαγόμενο κώδικα μηχανής σε VHDL. Η οδηγία .TITLE ακολουθούμενη από ένα τίτλο σε εισαγωγικά, κατευθύνει τον Assembler να προσθέσει τον τίτλο στην επικεφαλίδα του κώδικα VHDL. Για παράδειγμα η οδηγία:

```
.TITLE "Multiplication and division with subroutines (divmul.asm)"
```

Μετατρέπεται στην εξής επικεφαλίδα στον εξαγόμενο κώδικα VHDL:

```
-----  
-- Multiplication and division with subroutines (divmul.asm)  
-----
```

Σημειώνεται ότι στις συμβολοσειρές επιτρέπεται η χρήση εισαγωγικών με διαφυγή (escaped quotes) μέσω ανάστροφης καθέτου (\). Για παράδειγμα η συμβολοσειρά "a\ "b" θα πρέπει εσωτερικά να μεταφράζεται ως a"b.

#### 6.4.2. Δήλωση συμβολικής σταθερής .NAME

Η δυνατότητα ονομασίας αριθμών με συμβολικά ονόματα μπορεί να βελτιώσει την αναγνωσιμότητα του προγράμματος και επιτρέπει την πιο εύκολη συντήρησή του.

Στο ακόλουθο παράδειγμα, τα ονόματα dividend και divisor μπορούν να αντικαταστήσουν τις τιμές 179 και 12 σε διάφορα σημεία του προγράμματος.

```
.NAME dividend 179  
.NAME divisor 12
```

#### 6.4.3. Αρχικοποίηση περιοχών μνήμης .DATA

Για την μαζική αποθήκευση δεδομένων σε συγκεκριμένες διευθύνσεις της μνήμης θα δοθεί η οδηγία .DATA ακολουθούμενη από μια διεύθυνση, κενό, και τιμές που διαχωρίζονται με κόμματα. Οι τιμές μπορούν να είναι αριθμοί ή συμβολοσειρές. Οι συμβολοσειρές εκπροσωπούνται με τους αντίστοιχους κωδικούς ASCII. Για παράδειγμα, με την ακόλουθη οδηγία ο Assembler θα αποθηκεύσει τα δεδομένα 1,97,98,99,0 ξεκινώντας από την διεύθυνση 100.

```
.DATA 100 1,"abc",0
```



#### 6.4.4. Προεπιλεγμένη συχνότητα CPU .FREQUENCY

Η οδηγία αυτή αναπτύχθηκε μετά απο την ολοκλήρωση της υλοποίησης σε FPGA. Κατα την εκτέλεση με πραγματικό υλικό έγινε αντιληπτό οτι ο επεξεργαστής έπρεπε να υποστηρίξει μεταβλητή συχνότητα ανάλογα με το εκπαιδευτικό σενάριο. Η δυνατότητα αυτή αναλύεται στην Ενότητα 9.6 (σ. 111).

Η .FREQUENCY ακολουθείται απο ένα αριθμό απο 1 ως 1000 που εκφράζει deciHertz. Για παράδειγμα η παρακάτω οδηγία ξεκινά τον E80 με συχνότητα 10 Hz:

```
.FREQUENCY 100
```

Η προεπιλεγμένη συχνότητα είναι 15, δηλαδή 1.5 Hz. Αυτό επιτρέπει σχετικά εύκολη παρακολούθηση των μεταβολών στα LED της πλακέτας.

#### 6.4.5. Είσοδος DIP στην προσομοίωση .SIMDIP

Η υλοποίηση του E80 σε FPGA συμπεριλαμβάνει 8 διακόπτες σε ένα DIP για είσοδο δεδομένων απο τον χρήστη. Για να μπορεί να προσομοιωθεί η δυνατότητα αυτή σε GHDL και ModelSim, αποφασίστηκε η προσθήκη της οδηγίας .SIMDIP που ορίζει την είσοδο αυτή στο testbench που χρησιμοποιείται για την εκτέλεση του κώδικα.

Η .SIMDIP ακολουθείται απο αριθμό ή συμβολικό όνομα. Για παράδειγμα η δήλωση:

```
.SIMDIP 0b10000001
```

Θα επιτρέψει την χρήση μιας εντολής όπως η LOAD R0, [0xFF] που θα δώσει στον R0 την τιμή 10000001 στην προσομοίωση. Το αναμενόμενο είναι οτι κατα την εκτέλεση σε FPGA ο χρήστης θα ρυθμίσει τους διακόπτες στην τιμή αυτή.

#### 6.5. Ενοποίηση συμβολικών σταθερών και ετικετών

Οι ετικέτες και οι σταθερές θα θεωρηθούν ως ενιαία συμβολικά ονόματα και θα μπορεί να γίνει αναφορά τους με ενιαίο τρόπο σε οποιοδήποτε σημείο του προγράμματος. Για παράδειγμα, οι ετικέτες θα μπορούν να χρησιμοποιηθούν για αποθήκευση δεδομένων μετά απο το τέλος του προγράμματος, όπως φαίνεται ακολούθως:

```
.DATA dat 0xFF, 0xFE, 0xFD
    MOV R0, dat
    HLT
dat:
```

Η ετικέτα dat περιέχει την διεύθυνση μνήμης που ακολουθεί την HLT, και επομένως οι λέξεις 0xFF, 0xFE, 0xFD θα αποθηκευτούν αμέσως μετά την HLT. Η εν λόγω διεύθυνση μνήμης αποθηκεύεται και στον καταχωρητή R0. Υποθέτοντας οτι η HLT (η οποία έχει μέγεθος 1) βρίσκεται στην θέση μνήμης 100, θα ισχύει dat = 101 και επομένως οι τιμές 0xFF, 0xFE, 0xFD θα αποθηκευτούν στις θέσεις 101,102,103 ενώ στον R0 θα τοποθετηθεί η τιμή 101.

Αυτό επιτρέπει την εύκολη αρχικοποίηση δεδομένων στο τέλος του προγράμματος χωρίς να χρειάζεται η μέτρηση του μεγέθους του.

Σημειώνεται οτι ο συμβολομεταφραστής θα πρέπει να αναγνωρίζει την εγγραφή δεδομένων επι των εκτελέσιμων εντολών και να μην την επιτρέπει.

## 6.6. Διατύπωση με ασυμφραστική γραμματική

Απο την συγγραφή της γραμματικής για την συμβολική γλώσσα του E80 ξεκινά η κάλυψη του στόχου της **Εφαρμογής** της ύλης του προγράμματος σπουδών ΠΛΗ του ΕΑΠ.

Η συμβολική γλώσσα του E80 σχεδιάστηκε σε ασυμφραστική γραμματική τύπου BNF η οποία παρουσιάστηκε στο Υπόβαθρο (Ενότητα 2.4.1, σ. 12), και σύμφωνα με την σχετική θεωρία των ενοτήτων ΠΛΗ24 και ΠΛΗ30. Η προσέγγιση αυτή αποδείχτηκε εξαιρετικά χρήσιμη καθώς επέτρεψε την δυνατότητα της εύκολης δοκιμής διαφόρων γραμματικών προσεγγίσεων, καθώς και την οργάνωση της συντακτικής ανάλυσης που έγινε στον Assembler όπως θα παρουσιαστεί στο επόμενο κεφάλαιο.

Πιο συγκεκριμένα, οι διάφορες σχεδιαστικές προσεγγίσεις δοκιμάστηκαν αρχικά στην μορφή BNF, χρησιμοποιώντας το εργαλείο CFG-Tester<sup>1</sup> επιτρέποντας την ταχεία αξιολόγηση και επιλογή τους. Ταυτόχρονα η διατύπωση της γλώσσας σε ασυμφραστική γραμματική έδωσε την δυνατότητα να διορθωθούν λογικά λάθη που οδηγούσαν σε διφορούμενες γραμματικές. Αυτό επέτρεψε την κατασκευή μιας απλής αλλά ταυτόχρονα στιβαρής γλώσσας.

Όπως θα φανεί στο επόμενο κεφάλαιο, η διατύπωση σε ασυμφραστική γραμματική δεν χρησιμοποιήθηκε για αυτοματισμό της συντακτικής ανάλυσης με εργαλεία όπως Yacc & Bison διότι ο σκοπός ήταν να γίνει χρήση βασικών βιβλιοθηκών της C σύμφωνα με την ΠΛΗ10, για τον ίδιο λόγο που δεν χρησιμοποιήθηκαν αριθμητικές βιβλιοθήκες στην VHDL.

Επομένως, δυνατότητες και χαρακτηριστικά όπως σχόλια και εισαγωγικά εντός εισαγωγικών (escaped quotes) υποστηρίζονται απο τον συμβολομεταφραστή αλλά δεν εκφράζονται στην BNF καθώς η πλήρης μοντελοποίησή τους θα αύξανε σημαντικά την πολυπλοκότητα χωρίς να προσφέρει πλεονεκτήματα στην ανάπτυξη. Για παράδειγμα, ένας κανόνας για σχόλια θα απαιτούσε την καταγραφή όλων των χαρακτήρων του συνόλου ASCII δεδομένου ότι η BNF δεν επιτρέπει κλάσεις χαρακτήρων.

Ακολουθεί η τελική μορφή της γραμματικής στην διπλανή σελίδα.

<sup>1</sup> <https://cpiber.github.io/CFG-Tester> ο σύνδεσμος παραπέμπει σε ένα μεγάλο URL που συμπεριλαμβάνει την τελική μορφή της γραμματικής καθώς και ένα δοκιμαστικό πρόγραμμα με συντακτικές ιδιαιτερότητες που έπρεπε να υποστηριχθούν

```

<start> ::= <[directives]> <[codelines]>
<[directives]> ::= <directive> | <directive> <nl+> <[directives]> | <[\n]>
<directive> ::= ".TITLE" <s+> <quoted_string>
               | ".NAME" <s+> <identifier> <s+> <number>
               | ".FREQUENCY" <s+> <dec+>
               | ".SIMDIP" <s+> <n>
               | ".DATA" <s+> <n> <s+> <array>
<[codelines]> ::= <codeline> | <codeline> <nl+> <[codelines]> | <[\n]>
<codeline> ::= <label> | <instruction> | <label> <[\n]> <instruction>
<label> ::= <identifier> <s*> ":"
<instruction> ::= <instr_noarg>
                | <instr_reg> <s+> <reg>
                | <instr_n> <s+> <n>
                | <instr_op1> <s+> <op1>
                | <instr_reg_op2> <s+> <reg> <,> <op2>
                | <instr_ldst> <s+> <reg> <,> <bracket_op2>
                | <instr_reg_n> <s+> <reg> <,> <n>
<instr_noarg> ::= "HLT" | "NOP" | "RETURN"
<instr_reg> ::= "RSHIFT" | "LSHIFT" | "PUSH" | "POP"
<instr_n> ::= "JC" | "JNC" | "JZ" | "JNZ" | "JS" | "JNS" | "JV" | "JNV"
            | "CALL"
<instr_op1> ::= "JMP"
<instr_reg_op2> ::= "MOV" | "ADD" | "ROR" | "SUB" | "CMP" | "AND" | "OR" | "XOR"
<instr_ldst> ::= "LOAD" | "STORE"
<instr_reg_n> ::= "BIT"
<op1> ::= <op2>
<bracket_op2> ::= "[" <op2> "]"
<op2> ::= <reg> | <n>
<n> ::= <number> | <identifier>
<reg> ::= "R0" | "R1" | "R2" | "R3" | "R4" | "R5" | "R6" | "SP" | "R7"
        | "FLAGS"
<array> ::= <array_element> | <array_element> <,> <array>
<array_element> ::= <number> | <quoted_string>
<quoted_string> ::= "\"" <char+> "\""
<,> ::= <s*> "," <s*>
<identifier> ::= <letter> <name_char*>
<name_char*> ::= <name_char> <name_char*> | ""
<name_char> ::= <letter> | <dec> | "_"
<number> ::= "0x" <hex+> | "0b" <bit+> | <dec+>
<hex+> ::= <hex> | <hex> <hex+>
<dec+> ::= <dec> | <dec> <dec+>
<bit+> ::= <bit> | <bit> <bit+>
<hex> ::= <dec> | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c"
        | "d" | "e" | "f"
<dec> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<bit> ::= "0" | "1"
<char+> ::= <char> | <char> <char+>
<char> ::= <letter> | <dec> | " "
<[\n]> ::= <nl+> | <s*>
<nl+> ::= <nl> | <nl> <nl+>
<nl> ::= <s*> "\n" <s*>
<s*> ::= <s+> | ""
<s+> ::= <s> | <s> <s+>
<s> ::= " " | "\t"
<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
            | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T"
            | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d"
            | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n"
            | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x"
            | "y" | "z"

```



## 7. Υλοποίηση του συμβολομεταφραστή σε C

Ο συμβολομεταφραστής του E80 σχεδιάστηκε και υλοποιήθηκε στο πρότυπο C99 της γλώσσα προγραμματισμού C, χωρίς χρήση εξωτερικών βιβλιοθηκών με στόχο, αφενός την διαλειτουργικότητα, και αφετέρου την **Εφαρμογή** των βασικών τεχνικών και γλωσσών προγραμματισμού που παρουσιάζονται στην σχετική ύλη της ΠΛΗ10 (Καμέας, [2008](#), και Θραμπουλίδης, [2000](#)).

### 7.1. Μεταγλωττιστής και προγραμματιστικό περιβάλλον

Για τη συγγραφή και μεταγλώττιση του συμβολομεταφραστή χρησιμοποιήθηκε η συλλογή μεταγλωττιστών MinGW (Minimalist GNU for Windows), η οποία παρέχει μια μεταφορά της GNU Compiler Collection (GCC) για το περιβάλλον των Windows.

Ως ολοκληρωμένο προγραμματιστικό περιβάλλον, επιλέχθηκε η φορητή έκδοση του RedPanda C++<sup>1</sup>. Η επιλογή αυτή έγινε ώστε να υπάρχει συνέχεια με το κλασικό Dev-C++ που χρησιμοποιήθηκε στην ΠΛΗ10 στο έτος 2019-2020. Το Red Panda αποτελεί μια υποστηριζόμενη διακλάδωση του Dev-C++ με πληθώρα νέων χαρακτηριστικών.

### 7.2. Είσοδος/Εξόδος μέσω Stdin/Stdout

Ο Assembler κατασκευάστηκε έτσι ώστε να διαβάζει και να εξάγει δεδομένα αποκλειστικά μέσω της πρότυπης εισόδου/εξόδου (stdin/stdout). Για παράδειγμα η εντολή:

```
C:\E80\Assembler>e80asm < uppercase.asm > ..\VHDL\Firmware.vhd
```

Δέχεται το αρχείο uppercase.asm με ανακατεύθυνση εισόδου "<", παρουσιάζει το μήνυμα που καταγράφεται στην Εικόνα 30, και εξάγει τον κώδικα μηχανής στο αρχείο Firmware.vhd μέσω της ανακατεύθυνσης ">". Το ίδιο το μήνυμα δεν έχει ανακατευθυνθεί διότι εμφανίστηκε μέσω της πρότυπης εξόδου stderr.

```
E80 CPU Assembler v1.0 - July 2025, Panos Stokas
Translates an E80-assembly program to firmware VHDL code.
E80ASM [/Q]

/Q          Silent mode, hides this message.

I/O is handled via stdin/stdout. Eg. to read 'program.asm'
and write the result to 'firmware.vhd', type:

e80asm < program.asm > firmware.vhd

You can also paste your code here and then press
Ctrl-D & [Enter] to translate it, or Ctrl-C to exit.

Collecting symbols... None.
Parsing directives... OK.
Parsing instructions... OK.
```

Εικόνα 30: Εκτέλεση του Assembler με ανακατεύθυνση εισόδου/εξόδου

Η επιλογή της stdin έγινε ακολουθώντας την κλασική φιλοσοφία σχεδιασμού του Unix για τη δημιουργία απλών, εστιασμένων και διαλειτουργικών εργαλείων. Το πλεονέκτημά της είναι ότι μπορεί να ενσωματωθεί σε αρχεία δεσμίδας και να συνδεθεί με άλλα προγράμματα γραμμής εντολών μέσω ανακατεύθυνσης και διοχέτευσης. Αυτό χρησιμοποιήθηκε πρακτικά στην Ενότητα 8.5 “Απο την Assembly ως τις κυματομορφές με ένα πλήκτρο” (σ. 104).

<sup>1</sup> <https://sourceforge.net/projects/redpanda-cpp>

### 7.3. Δομές δεδομένων

Για την αποθήκευση και την προσπέλαση των δεδομένων στα στάδια της μετατροπής, χρησιμοποιήθηκε η σχετική θεωρία δομών δεδομένων από την ΠΛΗ10 (Χατζηλυγερούδης, [2008](#)). Οι δομές που κατασκευάστηκαν είναι οι εξής:

- Λίστα κόμβων `ListNode`: Χρησιμοποιείται για την αποθήκευση των γραμμών του κώδικα `Assembly`. Επιλέχθηκε η δομή της συνδεδεμένης λίστας επειδή οι γραμμές προσπελούνται μόνο σειριακά και το πλήθος τους δεν είναι γνωστό εκ των προτέρων. Η λίστα χρησιμοποιεί οργάνωση `FIFO` με δείκτες κεφαλής και ουράς.
- Πίνακας δεικτών σε κόμβους `SymbolElement`: Οι κόμβοι περιέχουν τα ζευγάρια ονομάτων/τιμών των συμβόλων που συλλέγονται κατά το πρώτο πέρασμα. Στο τέλος της συλλογής, οι δείκτες ταξινομούνται ως προς τα ονόματα των συμβόλων, επιτρέποντας δυαδική αναζήτηση στο δεύτερο πέρασμα για την διευθυνσιοδότηση των αναφορών.
- Πίνακες χαρακτήρων `ram` και `comment`: Αυτές οι δομές αποθηκεύουν τα περιεχόμενα της μνήμης (τον τελικό δυαδικό κώδικα μηχανής) και τα αντίστοιχα σχόλια που παράγονται στο τελικό στάδιο μετατροπής σε κώδικα `VHDL`.

### 7.4. Ροή λειτουργίας του Assembler

Ως προς τα βήματα λεκτικής και συντακτικής ανάλυσης, ακολουθείται η πορεία που παρουσιάστηκε στο Υπόβαθρο (Ενότητα 2.4, σ. 12).

#### 7.4.1. Ανάγνωση εισόδου και αποθήκευση γραμμών σε λίστα

Τα δεδομένα διαβάζονται από την `stdin`, γραμμή-προς-γραμμή. Αφαιρούνται τα σχόλια καθώς και τα κενά αριστερά και δεξιά από τις γραμμές. Στην περίπτωση που μια γραμμή υπερβεί το όριο `MAX_LINE_LENGTH` (150) εμφανίζεται μήνυμα σφάλματος. Η κάθε καθαρισμένη γραμμή εισάγεται ως `ListNode` στον δείκτη ουράς.

#### 7.4.2. Λεκτική ανάλυση

Η εξαγωγή των λεκτικών μονάδων γίνεται με την συνάρτηση `nexttoken()` η οποία σε κάθε κλήση της επιστρέφει ένα δείκτη στην επόμενη λεκτική μονάδα του πηγαίου κώδικα `Assembly`. Τα διαχωριστικά των λεκτικών μονάδων όπως προκύπτει από την ασυμφραστική γραμματική που παρουσιάστηκε στην Ενότητα 6.6 (σ. 86) είναι οι χαρακτήρες [ " ] , : space, tab, αλλαγή γραμμής κλπ καθώς και ο τερματικός χαρακτήρας `NULL`.

### 7.4.3. Συλλογή συμβόλων

Χρησιμοποιώντας την συνάρτηση nexttoken(), ο συμβολομεταφραστής διαβάζει διαδοχικά τις λεκτικές μονάδες για να εντοπίσει και να αναθέσει τιμές στα σύμβολα ως εξής:

- Αν η λεκτική μονάδα είναι εντολή, τότε προστίθεται το μέγεθός της στην τρέχουσα διεύθυνση,
- Αλλιώς αν είναι οδηγία .NAME τότε εισάγεται η συμβολική σταθερή και η τιμή της στον πίνακα συμβόλων,
- Αλλιώς αν είναι ετικέτα που ακολουθείται από άνω κάτω τελεία, εισάγεται το όνομά της και η τρέχουσα διεύθυνση στον πίνακα των συμβόλων.

### 7.4.4. Συντακτική ανάλυση οδηγιών και εγγραφή .DATA στην μνήμη

Στην φάση αυτή ελέγχονται οι οδηγίες και παράγονται τα σχετικά δεδομένα:

- Οδηγία .TITLE: Αναγνωρίζεται η οδηγία .TITLE και αποθηκεύεται ο τίτλος του προγράμματος. Ελέγχεται επίσης για διπλότυπες δηλώσεις τίτλου.
- Οδηγία .DATA: Ο Assembler διαβάζει τη διεύθυνση έναρξης και μια λίστα στοιχείων που έχουν διαχωριστεί με κόμματα. Ο κάθε αριθμός και ο κάθε χαρακτήρας συμβολοσειράς μετατρέπεται σε δυαδική μορφή και αποθηκεύεται στον πίνακα της μνήμης, ενώ παράλληλα καταγράφεται η τιμή ως σχόλιο που θα εξαχθεί στην VHDL.
- Οδηγία .FREQUENCY: Ορίζει τη συχνότητα λειτουργίας του CPU σε deciHertz και αποθηκεύεται η τιμή αυτή. Γίνεται έλεγχος για το επιτρεπτό εύρος της τιμής.
- Οδηγία .SIMDIP: Διαβάζεται και καταγράφεται η τιμή που την ακολουθεί.
- Οδηγία .NAME: Δεδομένου ότι οι συμβολικές σταθερές αναλύθηκαν στην προηγούμενη φάση, η οδηγία αυτή παρακάμπτεται.

Όταν εντοπιστεί μια λεκτική μονάδα που δεν είναι οδηγία, τερματίζεται η συντακτική ανάλυση οδηγιών. Δεδομένης της ασυμφραστικής γραμματικής, η μονάδα αυτή θα είναι υποχρεωτικά εντολή ή ετικέτα, οπότε και θα αναλυθεί στην επόμενη φάση.

### 7.4.5. Συντακτική ανάλυση εντολών και παραγωγή κώδικα μηχανής

Το στάδιο αυτό είναι το κύριο μέρος του κώδικα, όπου οι συμβολικές εντολές και τα τελούμενά τους μετατρέπονται σε δυαδική μορφή, αξιοποιώντας τον πίνακα συμβόλων.

Δεδομένου ότι η πρώτη συμβολική εντολή του E80 βρίσκεται στην διεύθυνση 0, ο δείκτης της θέσης μνήμης αρχικοποιείται ξανά στο 0 και ξεκινά η ανάλυση των εντολών, σύμφωνα με την ασυμφραστική γραμματική. Αυτό σημαίνει ότι η κάθε εντολή ελέγχεται αν ικανοποιεί τον κανόνα <instruction> σε μια δομή πολλαπλής επιλογής.

Εφόσον έχει εντοπιστεί μια εντολή, αναγνωρίζονται τα τελούμενα που πρέπει να την ακολουθούν. Τα τελούμενα γράφονται στην μνήμη μέσω της διαδικασίας bitcopy η οποία κατασκευάστηκε για να παρομοιάζει την σειρά DOWNT0 που χρησιμοποιείται στα διανύσματα του E80. Για παράδειγμα η εντολή bitcopy(RAM, reg, 7, 4) καταγράφει την μεταβλητή reg στις θέσεις 7 (ΠΣΨ) ως 4 (ΛΣΨ) στην τρέχουσα θέση της RAM.



Στην περίπτωση που γίνεται χρήση συμβόλων, λαμβάνεται η τιμή τους από τον πίνακα συμβόλων με δυαδική αναζήτηση. Αν η αναζήτηση δεν είναι επιτυχής, σημαίνει ότι γίνεται αναφορά ανύπαρκτου συμβόλου.

Ταυτόχρονα με την μετατροπή σε κώδικα μηχανής, το μνημονικό της τρέχουσας εντολής καταγράφεται και στην αντίστοιχη θέση στον πίνακα των σχολίων. Το σχόλιο γράφεται αντίστοιχα με την τελευταία λέξη κάθε εντολής, εξασφαλίζοντας έτσι την ομοιόμορφη στοίχιση των σχολίων σε εντολές μίας ή δύο λέξεων στην τελική έξοδο VHDL.

#### 7.4.6. Έλεγχοι σφαλμάτων

Δεδομένου ότι ο E80 απευθύνεται σε επίπεδο αρχαρίων, η παροχή ευανάγνωστων και εύστοχων μηνυμάτων αποσφαλμάτωσης έχει κρίσιμη σημασία.

Σημαντικό πλεονέκτημα που προέκυψε από την συγγραφή του Assembler χωρίς την χρήση έτοιμων πακέτων συντακτικής ανάλυσης, ήταν η δυνατότητα καλύτερης στόχευσης του ελέγχου λαθών με προτροπές πιο χρήσιμες από γενικά μηνύματα όπως “Άγνωστο αναγνωριστικό”. Για παράδειγμα, στην απόπειρα μεταγλώττισης της γραμμής

```
STORE R1, [R8]
```

Ο Assembler παρουσιάζει το μήνυμα της Εικόνας 31:

```
Collecting symbols... None.
Parsing directives... OK.
Parsing instructions...
*****
Error in line 1 : STORE R1, [R8]
'R8' is not number or symbol or register.
*****
```

Εικόνα 31: Μήνυμα σφάλματος σε όρισμα

Το μήνυμα αυτό επιτρέπει στον εκπαιδευόμενο να αντιληφθεί την σωστή σύνταξη της γλώσσας χωρίς να χρειάζεται να γνωρίζει την γενική έννοια του “αναγνωριστικού”.

Οι έλεγχοι σφαλμάτων πραγματοποιούνται με την διαδικασία `error()` η οποία δέχεται μια παράμετρο `errorlevel` τύπου `enum` που προσδιορίζει το είδος του λάθους. Παράλληλα, παρέχεται πρόσβαση στο τρέχον λεκτικό σύμβολο που παρουσίασε το σφάλμα, αλλά και στο προηγούμενο. Για παράδειγμα ο παρακάτω έλεγχος στον `error_handler.c`,

```
case RIGHTBRACKET:
    fprintf(stderr, "LOAD/STORE requires a right bracket "
               "after '%s'", PREVIOUS);
```

Επιτρέπει την εξαγωγή του επεξηγηματικού μηνύματος σφάλματος, όπως για παράδειγμα στην Εικόνα 32 που ακολουθεί:

```
*****
Error in line 1 : STORE R1, [R2
LOAD/STORE requires a right bracket after 'R2'
*****
```

Εικόνα 32: Μήνυμα συντακτικού σφάλματος

Η `exit(errorlevel)` τερματίζει την εκτέλεση και επιστρέφει τον αριθμό σφάλματος ως κωδικό εξόδου. Αυτό ενισχύει την δυνατότητα της χρήσης του συμβολομεταφραστή σε ένα ολοκληρωμένο προγραμματιστικό περιβάλλον, όπως γίνεται στο παράδειγμα του Notepad++ στην Ενότητα 8.5 (σ. 104).

#### 7.4.7. Τελική παραγωγή εξόδου σε VHDL

Αφού συμπληρωθούν οι πίνακες ram και comment, ο συμβολομεταφραστής προχωρά στην τελική παραγωγή του αρχείου VHDL. Αρχικά διαβάζει το προκαθορισμένο πρότυπο αρχείο (Template.vhd) και αναζητά συγκεκριμένες κρατήσεις θέσεων στον κώδικα (placeholders) στις οποίες θα τοποθετηθούν:

- Το όνομα του προγράμματος (προαιρετικά).
- Η τιμή της λέξης SimDIP που θα ανατεθεί στην είσοδο DIPinput στο testbench.
- Η προεπιλεγμένη συχνότητα λειτουργίας του επεξεργαστή σε deciHertz.
- Ο κώδικας με τα σχόλια.

Η διαδικασία παραγωγής του τελικού κώδικα μηχανής σε μορφή VHDL δεν περιορίστηκε στη δημιουργία μιας απλής λίστας διευθύνσεων/δεδομένων, αλλά επικεντρώθηκε στην παραγωγή ενός στοιχισμένου και ευανάγνωστου αρχείου. Η έμφαση στην αισθητική αρτιότητα ευθυγραμμίζεται με σύγχρονα ακαδημαϊκά ευρήματα (Maikantis κ.α., [2025](#)) που τεκμηριώνουν την συσχέτισή της με την τεχνική ποιότητα του κώδικα.

Για την κατασκευή των σχολίων έπρεπε να ληφθεί υπόψιν ότι ο E80 υποστηρίζει εντολές μεταβλητού μεγέθους. Όπως σημειώθηκε στη φάση της συντακτικής ανάλυσης, το στοιχείο του πίνακα comment που αντιστοιχεί στο πρώτο τμήμα μιας εντολής δύο λέξεων παραμένει σκόπιμα κενό. Κατά συνέπεια, όταν ο αλγόριθμος παραγωγής της εξόδου εντοπίζει μια λέξη με κενό σχόλιο, παραμένει στην τρέχουσα γραμμή και συνεχίζει με την επόμενη θέση μνήμης. Όταν υπάρχει σχόλιο, αυτό καταγράφεται στοιχισμένο με στηλοθέτη και εισάγεται κωδικός αλλαγής γραμμής (\n).

Παράλληλα, σχολιάζονται και τα δεδομένα που προκύπτουν από τις οδηγίες .DATA: οι αριθμητικές τιμές καταγράφονται στο δεκαδικό σύστημα, ενώ για τους χαρακτήρες συμβολοσειρών παρατίθεται ο ίδιος ο χαρακτήρας και, εντός παρενθέσεως, η τιμή του κατά ASCII. Η τελική στοίχιση επιτυγχάνεται μέσω του δυναμικού υπολογισμού των απαιτούμενων κενών διαστημάτων (padding) με χρήση της συνάρτησης sprintf, εξασφαλίζοντας ένα αποτέλεσμα το οποίο είναι ευανάγνωστο και κατάλληλο για την μελέτη της γλώσσας μηχανής.

#### 7.5. Παράδειγμα μετατροπής ιδιωματικού κώδικα

Το κάτωθι πρόγραμμα assembly μετατρέπει τους πεζούς λατινικούς χαρακτήρες της συμβολοσειράς `az{"0 σε κεφαλαίους. Ακολουθεί το τερματικό NULL (0). Τα δεδομένα αυτά γράφονται μετά απο την τελευταία εντολή, σύμφωνα με την δυνατότητα που προσφέρει η Ενοποίηση των συμβολικών ονομάτων που παρουσιάστηκε στην Ενότητα 6.5 (σ. 85).

```
.TITLE "Uppercase"
.DATA string "`az{"0,0 ; written at the address of the "string" label
    MOV R0, string ; R0 = address of the first character ("`")
loop:
    LOAD R1, [R0] ; R1 = ASCII value of current character
    CMP R1, 0 ; if R1 = 0 (terminal)
    JZ finish ; goto finish.
    CMP R1, 97 ; else if R1-97
    JNC next ; is < 0 (thus R1 < 97), goto next.
    CMP R1, 123 ; else if R1 - 123
    JC next ; is ≥ 0 (thus R1 > 122), goto next.
```

```

SUB R1, 32          ; else, R1 ← R1 - 32 (change to uppercase)
STORE R1, [R0]      ; write character back to RAM
next:
ADD R0, 1           ; advance to the next memory address
JMP loop            ; repeat loop
finish:
HLT                 ; stop execution & simulation
string:             ; memory address after HLT

```

Εκτελώντας την εντολή `e80asm < uppercase.asm`, χωρίς ανακατεύθυνση προς αρχείο εξόδου, τα αποτελέσματα καταγράφονται στην οθόνη στην Εικόνα 33. Έχουν παραληφθεί οι αρχικές γραμμές που περιγράφουν τις βασικές οδηγίες χρήσης του εκτελέσιμου `e80asm.exe`:

```

Collecting symbols...
- loop = 2
- next = 20
- finish = 24
- string = 25
Parsing directives... OK.
Parsing instructions... OK.

-----
-- Uppercase
-----

LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT SimDIP : WORD := "00000000"; -- DIP input for testbench only
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT Firmware : WORDx256 := (
0  => "00010000", 1  => "00011001", -- MOV R0, 25
2  => "10011000", 3  => "00010000", -- LOAD R1, [R0]
4  => "10110001", 5  => "00000000", -- CMP R1, 0
6  => "00000110", 7  => "00011000", -- JZ 24
8  => "10110001", 9  => "01100001", -- CMP R1, 97
10 => "00000101", 11 => "00010100", -- JNC 20
12 => "10110001", 13 => "01111011", -- CMP R1, 123
14 => "00000100", 15 => "00010100", -- JC 20
16 => "00110001", 17 => "00100000", -- SUB R1, 32
18 => "10001000", 19 => "00010000", -- STORE R1, [R0]
20 => "00100000", 21 => "00000001", -- ADD R0, 1
22 => "00000010", 23 => "00000010", -- JMP 2
24 => "00000000", -- HLT
25 => "01100000", -- '' (96)
26 => "01100001", -- 'a' (97)
27 => "01111010", -- 'z' (122)
28 => "01111011", -- '{' (123)
29 => "00100010", -- ''' (34)
30 => "00110000", -- '0' (48)
31 => "00000000", -- 0
OTHERS => "UUUUUUUU");END;

```

Εικόνα 33: Μετατροπή προγράμματος σε κώδικα VHDL απο τον Assembler

Όπως φαίνεται απο τον κώδικα, η κάθε εντολή καταγράφεται σε μια γραμμή και ακολουθείται απο το μνημονικό της σε σχόλιο. Προφανώς τα συμβολικά ονόματα δεν εμφανίζονται στα σχόλια δεν εμφανίζονται αφού έχουν μετατραπεί σε τιμές ή διευθύνσεις.

Οι χαρακτήρες που γράφτηκαν μέσω της οδηγίας `.DATA` εμφανίζονται μαζί με τον κωδικό ASCII που τους αντιστοιχεί στο τέλος του προγράμματος. Εφόσον δεν είχε οριστεί συχνότητα, η σταθερή `DefaultFrequency` ορίζεται στο 1.5 Hz το οποίο κατα την εκτέλεση πολλαπλών προγραμμάτων ορίστηκε ως η “χρυσή τομή” ανάμεσα στην ταχύτητα και την δυνατότητα παρατήρησης της εκτέλεσης.

Τέλος, η μνήμη που δεν γράφεται απο τον Assembler ορίζεται σε απροσδιόριστες τιμές (“U”) μέσω της ανάθεσης `OTHERS`, για τον λόγο που έχει γραφτεί στην Ενότητα 5.6.1 (σ. 63).

## 8. Λειτουργία του E80 σε προσομοίωση

Το κεφάλαιο αυτό είναι πιο πρακτικό από τα προηγούμενα και δεν αφορά μόνο στην παρουσίαση της προσομοίωσης αλλά και στην εγκατάσταση λογισμικού, σε ρυθμίσεις και εκτέλεση εντολών. Για τις οδηγίες αυτές, βασική απαίτηση είναι η αποσυμπίεση φακέλων και αρχείων του έργου <https://github.com/Stokpan/E80> σε ένα φάκελο, πχ:

```
C:\E80\Assembler\  
C:\E80\GHDL\  
C:\E80\ModelSim\  
C:\E80\VHDL\  
C:\E80\README.md
```

Στο υπόλοιπο μέρος του κεφαλαίου θα χρησιμοποιούνται οι παραπάνω διαδρομές, αλλά δόθηκε η απαραίτητη μέριμνα ώστε όλες οι εφαρμογές να λειτουργούν ανεξάρτητα από την επιλογή βασικού φακέλου αποσυμπίεσης.

Έχοντας ολοκληρώσει τον Assembler, ήταν πλέον εφικτή η προσομοίωση της λειτουργίας του Υπολογιστή E80. Αρχικά γράφτηκε το ακόλουθο testbench:

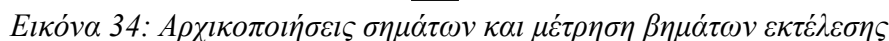
```
-- E80 Computer test bench  
LIBRARY ieee, work;  
USE ieee.std_logic_1164.ALL, work.support.ALL, work.firmware.ALL;  
ENTITY Computer_TB IS END;  
ARCHITECTURE a1 OF Computer_TB IS  
    SIGNAL CLK      : STD_LOGIC := '1';  
    SIGNAL Reset     : STD_LOGIC := '1';  
    SIGNAL DIPinput  : WORD := SimDIP;  
    SIGNAL PC        : WORD;  
    SIGNAL R         : WORDx8;  
    SIGNAL Halt      : STD_LOGIC;  
BEGIN  
    Halt <= R(6)(3);  
    -- if Halt=1, CLK stops pulsing => GHDL simulation ends  
    CLK <= '0' AFTER 50 ps WHEN CLK OR Halt ELSE '1' AFTER 50 ps;  
    Reset <= '0' AFTER 120 ps;  
    Computer : ENTITY work.Computer PORT MAP(CLK, Reset, DIPinput, PC, R);  
END;
```

Στο testbench αρχικοποιούνται το ρολόι, το Reset και η είσοδος DIP η οποία λαμβάνει τιμή από το Firmware σύμφωνα με την οδηγία .SIMDIP (Ενότητα 6.4.5, σ. 85).

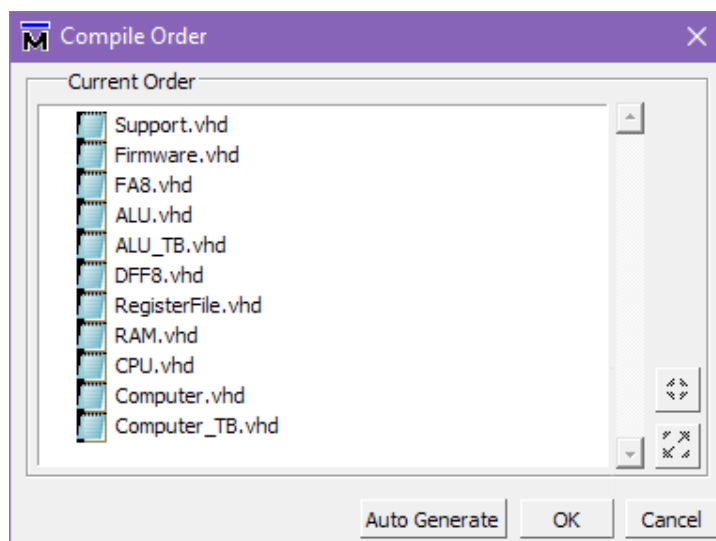
Το ρολόι έχει περίοδο 100 ps. Άρα με το Reset πιεσμένο, δοθέντος του σύγχρονου σχεδιασμού του και των θετικά ακμοπυροδότητων flip flop που χρησιμοποιούνται, η λειτουργία του υπολογιστή ξεκινά στο 100ο picosecond. Η συγκεκριμένη ανάθεση έγινε έτσι ώστε ο αριθμός του τρέχοντος βήματος εκτέλεσης να ορίζεται από την χρονική στιγμή div 100. Για παράδειγμα, στην Εικόνα 34, ο κέρσορας βρίσκεται στο 570 και συνεπώς δείχνει στο 5ο βήμα της εκτέλεσης.

Η εκτέλεση τερματίζεται σύμφωνα με την σημαία Halt με δυο διαφορετικούς τρόπους:

- Στο GHDL ο τερματισμός γίνεται μέσω της παύσης του παλμού του ρολογιού. Το GHDL αντιλαμβάνεται ότι δεν υπάρχει μεταβολή σημάτων και τερματίζει την εκτέλεση αυτόματα.
- Στο ModelSim ο τερματισμός γίνεται βάσει συνθήκης Halt=1 που έχει προγραμματιστεί στο σενάριο “c.do”.

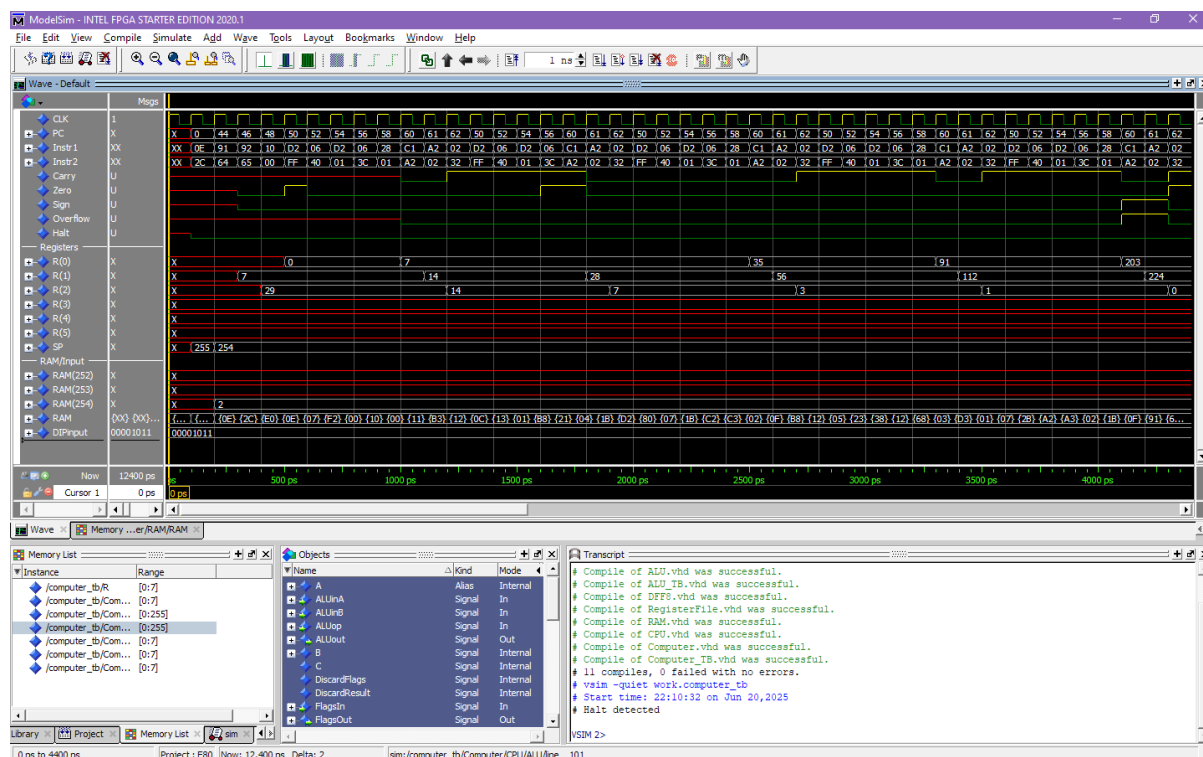


Για να ανοίξουμε το έργο με τις ρυθμίσεις όπως καταγράφονται στο Παράρτημα Ε.4 (σ. 178), επιλέγουμε File > Open και ως File name δίνουμε C:\E80\ModelSim\E80.mpf. Οι ρυθμίσεις έχουν συμπεριλάβει και την κατάλληλη σειρά μεταγλώττισης σύμφωνα με την Εικόνα 36.



Εικόνα 36: Σειρά μεταγλώττισης των συστατικών

Δεν χρειάζεται να προβούμε σε μεταγλώττιση των αρχείων. Αυτή θα γίνει αυτόματα με την εκτέλεση του σεναρίου “c.do” που έχει προετοιμαστεί για την εργασία. Τα αρχεία .do εκτελούνται μέσω της εντολής **do** στο Transcript. Επομένως **πληκτρολογούμε do c.do στο παράθυρο Transcript**, γίνεται αυτόματα μεταγλώττιση και εμφανίζονται οι κυματομορφές και τα αντικείμενα της Εικόνας 37 που ακολουθεί:



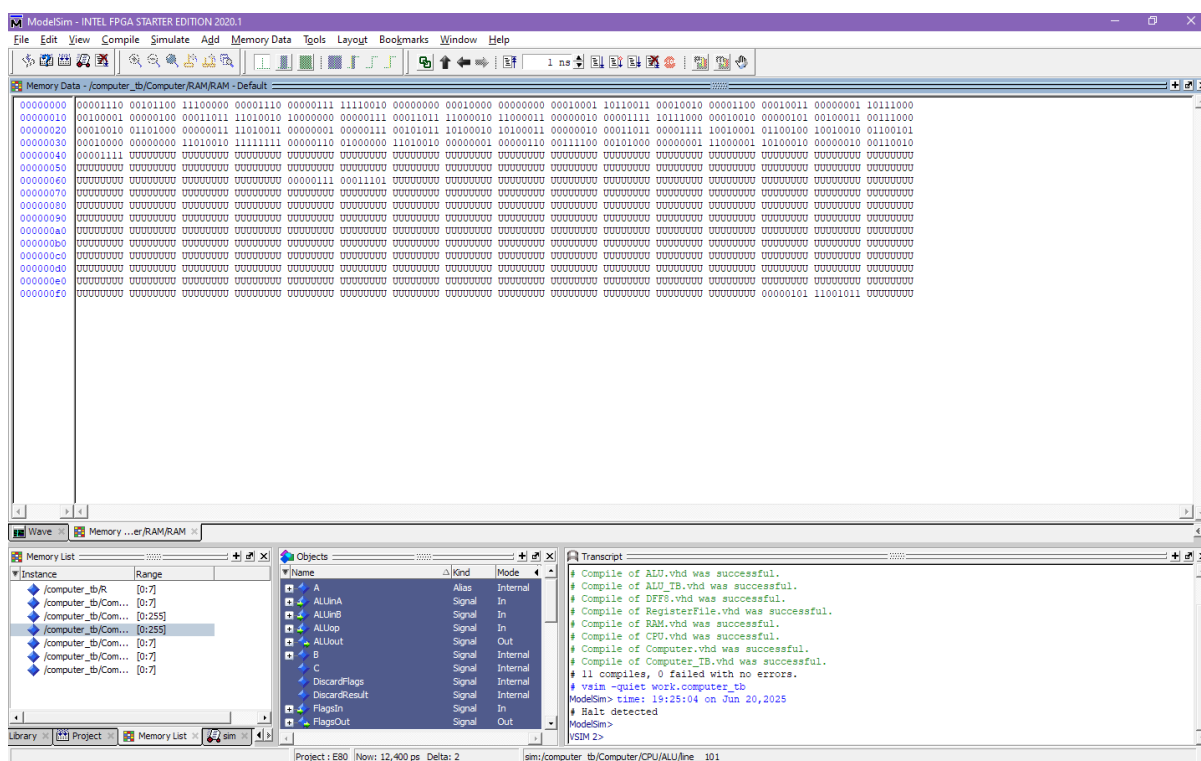
Εικόνα 37: Εκτέλεση σεναρίου προσομοίωσης μέσω εντολής “do c.do”



Απο την μελέτη του κώδικά του, και απο την Εικόνα 37 στην προηγούμενη σελίδα, προκύπτει οτι έχουν ομαδοποιηθεί τα σημαντικότερα σήματα και διανύσματα ως εξής:

- Βασικά σήματα: Ρολόι (CLK), μετρητής προγράμματος (PC), τρέχουσα εντολή που εκτελείται σε τμήματα 1 και 2 (Instr1 & Instr2), και σημαίες.
- Καταχωρητές γενικού σκοπού R0 - R5 και δείκτης στοίβας (SP).
- Μνήμη: Καταγράφονται οι τρεις τελευταίες θέσεις μνήμης για έλεγχο λειτουργίας της στοίβας, ολόκληρη η μνήμη και η είσοδος από τους διακόπτες DIP (διεύθυνση 0xFF).

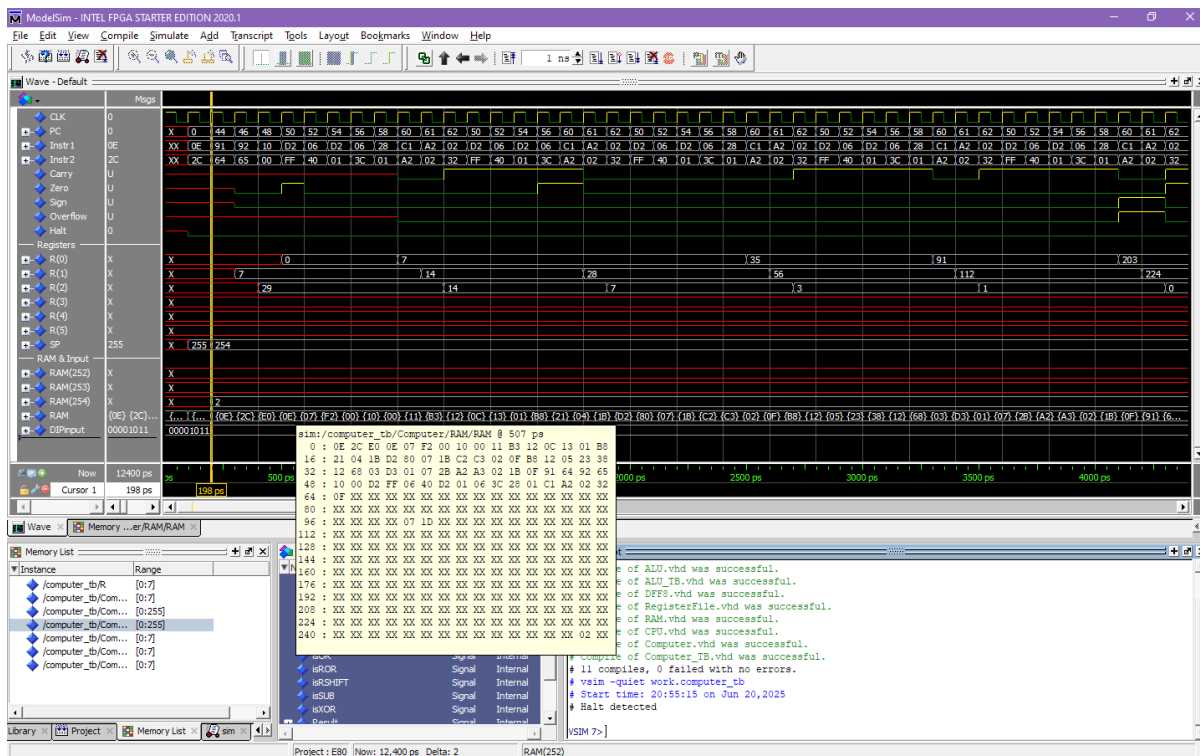
Επίσης απο το αρχείο c.do έχει δοθεί οδηγία να ανοίξει ένα παράθυρο με την κατάσταση της μνήμης στο τέλος της εκτέλεσης, όπως φαίνεται στην Εικόνα 38.



Εικόνα 38: Περιεχόμενα μνήμης στο τέλος της προσομοίωσης μέσω εντολής “do c.do”

Τα δεδομένα της μνήμης μπορούν να ελεγχθούν πιο άμεσα αφήνοντας τον δείκτη του ποντικιού πάνω στην γραμμή RAM σε ένα συγκεκριμένο βήμα εκτέλεσης. Τότε εμφανίζεται ένα αναδυόμενο πλαίσιο με τα περιεχόμενα όπως φαίνεται στην Εικόνα 39.





Εικόνα 39: Περιεχόμενα μνήμης σε σημείο εκτέλεσης μέσω “mouse hover”

### 8.1.2. Το σφάλμα με τα άγκιστρα στο ModelSim

Στο σημείο αυτό απαιτείται προσοχή από τον χρήστη: εάν το διάλυσμα RAM ρυθμιστεί σε radix ASCII και η μνήμη περιέχει τις λέξεις 123 ή 125 που αντιστοιχούν στα σύμβολα των αγκιστρών, τότε αφήνοντας τον δείκτη του ποντικιού στο συγκεκριμένο σημείο θα εμφανίσει το μήνυμα “Error in Tcl Script: unmatched open brace in list”. Ένα παράδειγμα που παρουσιάζει αυτό το πρόβλημα είναι το πρόγραμμα της Ενότητας 7.5 (σ. 93).

Ο μοναδικός τρόπος για να ελεγχθούν τα περιεχόμενα της μνήμης σε μορφή ASCII είναι το ξεδίπλωμά της (επέκταση / expand) μέσω του συμβόλου + αριστερά της RAM και η επισκόπηση των επιμέρους κυψελίδων στις κυματομορφές.

### 8.1.3. Αυτόματος τερματισμός προσομοίωσης μέσω Halt flag

Η ιδέα για τον αυτόματο τερματισμό προέκυψε από τις πολλαπλές δοκιμές προγραμμάτων κατά την διάρκεια του ελέγχου του E80. Η εκτέλεση σε μεγάλο χρονικό μήκος εμπόδιζε την επισκόπηση μικρών προγραμμάτων με πλήρες ζουμ, ενώ η προσομοίωση προγραμμάτων με πολλά βήματα απαιτούσε συνεχείς επαναλήψεις της λειτουργίας Run. Αυτό οδήγησε στην υλοποίηση της εντολής και σημαίας Αλτ. Συγκεκριμένα, οι εντολές στο σενάριο c.do:

```
when {/computer_tb/Halt=='1'} {
    stop
    echo "Halt detected"
}
run 100ns
```

Πραγματοποιούν προσομοίωση για 100 nanosecond το πολύ και τερματίζουν άμεσα όταν ενεργοποιηθεί η σημαία Αλτ επιτρέποντας την αποτελεσματική χρήση του πλήρους ζουμ για μικρά προγράμματα και την άμεση ολοκλήρωση πολύπλοκων προγραμμάτων.

## 8.2. Ένα σενάριο χρήσης του E80 με το ModelSim

Ξεκινάμε την εκτέλεση του Assembler (e80asm.exe) χωρίς ανακατεύθυνση:

```
C:\E80\Assembler>e80asm
```

Εφόσον δεν υπάρχει ανακατεύθυνση, ο Assembler περιμένει εντολές απο την κονσόλα:

```
E80 CPU Assembler v1.0 - July 2025, Panos Stokas
Translates an E80-assembly program to firmware VHDL code.
E80ASM [/Q]

/Q          Silent mode, hides this message.

I/O is handled via stdin/stdout. Eg. to read 'program.asm'
and write the result to 'firmware.vhd', type:

e80asm < program.asm > firmware.vhd

You can also paste your code here and then press
Ctrl-D & [Enter] to translate it, or Ctrl-C to exit.
```

Πληκτρολογούμε τις παρακάτω εντολές και στο τέλος πιέζουμε Ctrl-D και Enter:

```
MOV R0,0b1011
ADD R0,0xA
HLT^D
```

Ο Assembler εξάγει τον κώδικα μηχανής σε VHDL:

```
Collecting symbols... None.
Parsing directives... OK.
Parsing instructions... OK.

-----
-- E80 Firmware, generated by the E80ASM Assembler
-----

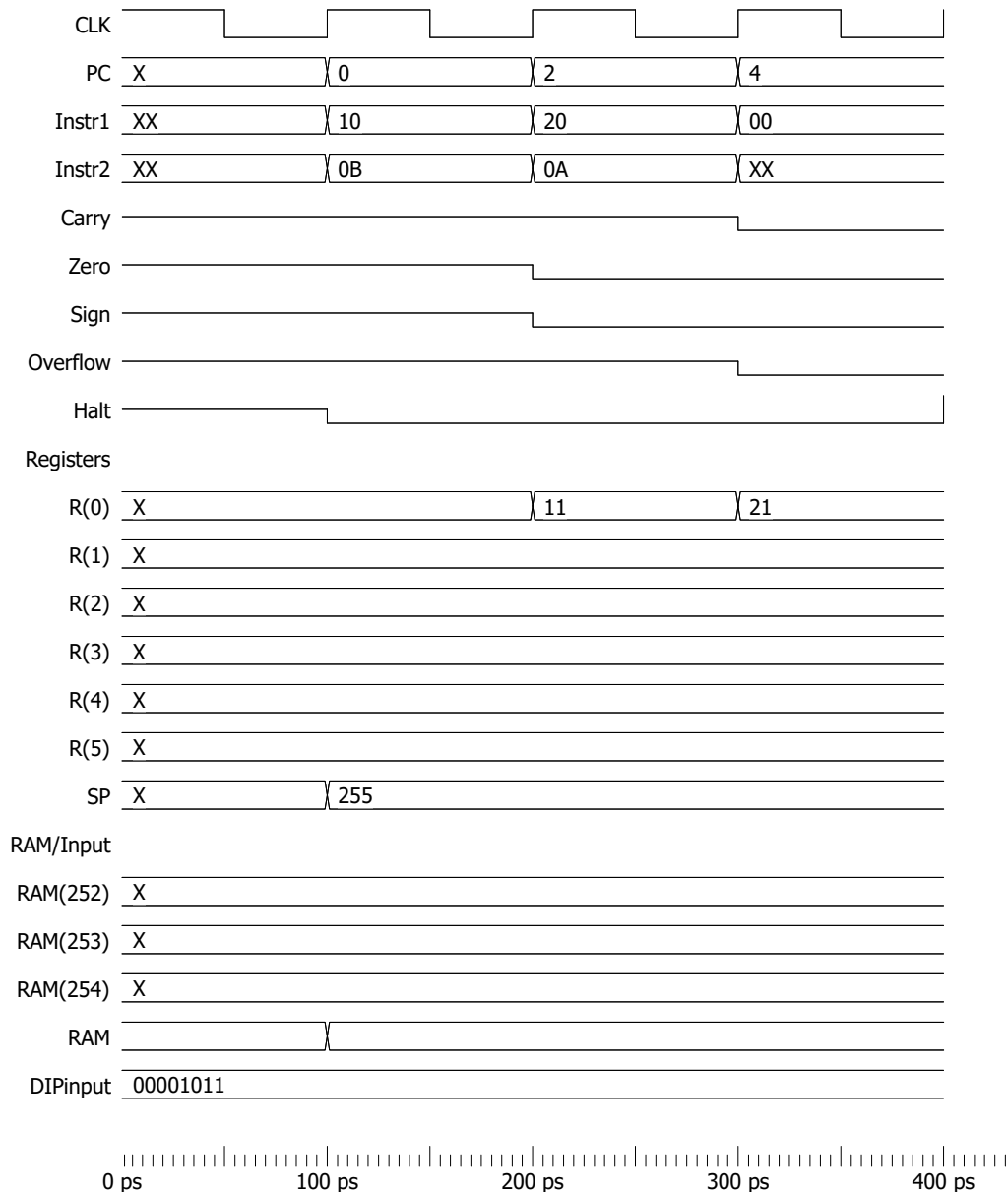
LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT SimDIP : WORD := "00000000"; -- DIP input for testbench only
CONSTANT Firmware : WORDx256 := (
0 => "00010000", 1 => "00001011", -- MOV R0, 11
2 => "00100000", 3 => "00001010", -- ADD R0, 10
4 => "00000000", -- HLT
OTHERS => "UUUUUUUU");END;
```

Τον επικολλούμε στο αρχείο VHDL/Firmware.vhd και εκτελούμε do c.do στο ModelSim. Τα αποτελέσματα λαμβάνονται όπως φαίνονται στην Εικόνα 40.

Τα δεδομένα του Firmware έχουν φορτωθεί στην μνήμη μετά απο Reset. Η εκτέλεση των εντολών εμφανίζεται απόλυτα σύμφωνη με την λογική του προγράμματος σε σχέση με την assembly, και τον κώδικα μηχανής σε σχέση με την αρχιτεκτονική εντολών:

- Οι διαδοχικές τιμές του Program Counter δείχνουν τις εντολές που εκτελούνται όπως έχουν καταγραφεί σε σχολιασμένη γλώσσα μηχανής στο αρχείο Firmware.vhd.
- Οι εντολές (Instr1, Instr2) που εκτελούνται ακολουθούν την μορφή Τύπου 5 σύμφωνα με την Ενότητα 4.3 (σ. 36).
- Οι τιμές των καταχωρητών ενημερώνονται στους επόμενους κύκλους απο την εκτελεσθείσα εντολή, όπως είναι αναμενόμενο σε flip flop.
- Οι σημαίες έχουν παραμείνει σε κατάσταση απροσδιοριστίας έως το σημείο της ενημέρωσής τους.

- Οι σημαίες έχουν ενημερωθεί σύμφωνα με το φυλλάδιο ISA στην Ενότητα 4.11 (σ. 51). Η MOV επηρεάζει μόνο τις σημαίες μηδενικού και προσήμου, ενώ η ADD επηρεάζει όλες τις σημαίες.
- Η ενεργοποίηση της σημαίας Αλτ έχει οδηγήσει στον τερματισμό της προσομοίωσης.



Εικόνα 40: Αποτέλεσμα εκτέλεσης απλού-δοκιμαστικού προγράμματος

Όλα τα παραπάνω μπορούν να αναλυθούν παράλληλα με την δίοδο δεδομένων και τον έλεγχο στον επεξεργαστή όπως παρουσιάστηκαν στην Ενότητα 5 (σ. 52).

### 8.3. GHDL & GTKWave

Ο συνδυασμός των GHDL & GTKWave προϋποθέτει την εγκατάσταση των δύο πακέτων από τους συνδέσμους που έχουν παρατεθεί στο Υπόβαθρο (Ενότητα 2.5.2, σ. 14). Μετά την εγκατάσταση, ο υποφάκελος `.bin` του κάθε πακέτου θα πρέπει να προστεθεί στην μεταβλητή περιβάλλοντος `PATH` του συστήματος, για παράδειγμα:

```
C:\GHDL\bin;C:\GTKWave\bin;
```

Τα πακέτα αυτά εκτελούνται μέσω γραμμής εντολών και οι οδηγίες τους είναι δυσνόητες, ειδικά στη περίπτωση του GHDL. Για τον λόγο αυτό, έχουν δημιουργηθεί αρχεία δεσμίδας (`.bat`) που απλοποιούν τις διαδικασίες και επιτρέπουν προσομοίωση που είναι πιο γρήγορη και εύκολη από το ModelSim.

Συγκεκριμένα το μοναδικό που απαιτείται από τον χρήστη είναι η πλοήγηση στον φάκελο `C:\E80\GHDL` και η εκτέλεση του `alu_tb.bat` για την προσομοίωση του testbench της ALU, ή του `computer_tb.bat` για την προσομοίωση του testbench του Υπολογιστή E80.

Ας εξετάσουμε αρχικά την προσομοίωση του testbench της ALU και ας αναλύσουμε τις εντολές που εκτελούνται. Το αρχείο `alu_tb.bat` περιέχει την εξής μοναδική εντολή:

```
g alu_tb 800ps
```

Η εκτέλεση της εντολής αυτής ορίζει την επιθυμητή μονάδα ανωτάτου επιπέδου που θα προσομοιωθεί, καθώς και την χρονική διάρκεια της προσομοίωσης. Με την εκτέλεσή της εμφανίζεται το στιγμιότυπο του τερματικού που φαίνεται στην Εικόνα 41.

```
C:\E80\GHDL>g alu_tb 800ps
E80 parametric GHDL-GTKwave simulation batch file
-----
1. Import and parse all VHDL files into the workspace :
ghdl -i --std=08 ..\VHDL\*.vhd
-----
2. Make the design with alu_tb as top unit :
ghdl -m --std=08 -Wno-hide alu_tb
-----
3. Run (simulate) the design for 800ps :
ghdl -r --std=08 alu_tb --stop-time=800ps --wave=alu_tb.ghw
ghdl:info: simulation stopped by --stop-time 0800ps
-----
4. Opening GTKWave config alu_tb.gtkw :
gtkwave alu_tb.gtkw

GTKWave Analyzer v3.3.90 (w)1999-2018 BSI

RCUAR : 'hide_sst on' FOUND
[0] start time.
[800000] end time.
```

Εικόνα 41: Εκτέλεση δεσμίδας `alu_tb.bat` για προσομοίωση GHDL/GTKWave

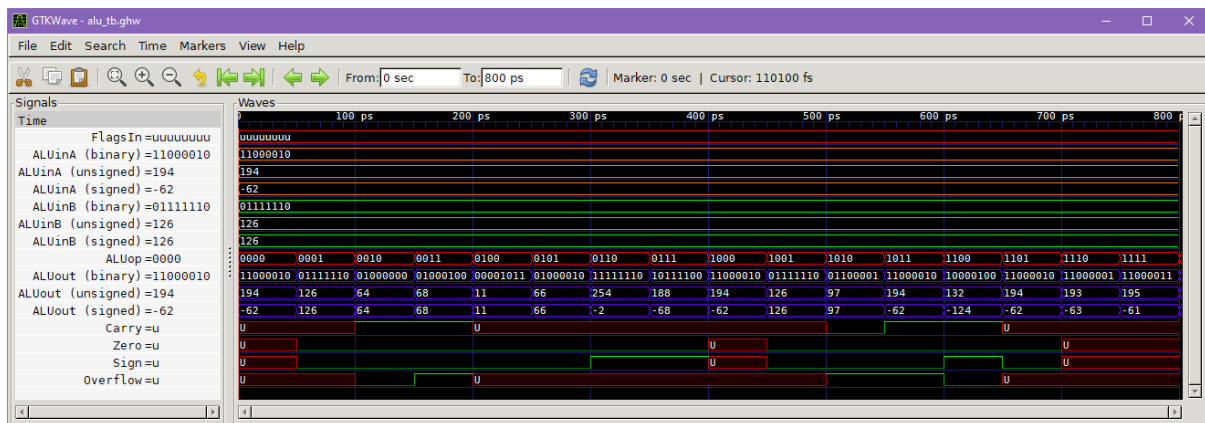
Η 1η εντολή (`ghdl -i`) αναλύει συντακτικά όλα τα αρχεία VHDL από τον αντίστοιχο φάκελο του έργου. Σκοπός της είναι ο εντοπισμός των συστατικών στα αρχεία, ώστε η επόμενη εντολή να μπορεί να δομήσει την ιεραρχία του σχεδιασμού.

Η 2η εντολή (`ghdl -m`) μεταγλωττίζει την VHDL, ορίζοντας ως μονάδα ανώτατου επιπέδου το `alu_tb`. Το GHDL πραγματοποιεί μεταγλώττιση χωρίς να χρειάζεται ορισμό της σειράς των αρχείων όπως απαιτούσε το ModelSim.

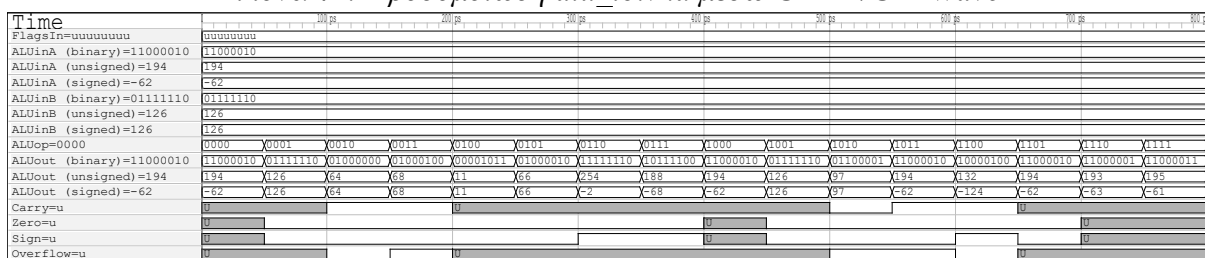
Στην συνέχεια, εκτελείται η προσομοίωση του μεταγλωττισμένου κώδικα για 800 picosecond, όπως είχε γίνει και στο ModelSim. Οι κυματομορφές που προκύπτουν αποθηκεύονται στο αρχείο `alu_tb.ghw`.

Τέλος, το αρχείο .ghw ανοίγει με το GTKWave όπως φαίνεται στην Εικόνα 42, με τις προκαθορισμένες ρυθμίσεις από το συνοδευτικό αρχείο alu\_tb.gtkw. Οι ρυθμίσεις αυτές έχουν πραγματοποιηθεί ως μέρος της παρούσας εργασίας στο Παράρτημα Δ.2 (σ. 172) και καθορίζουν το “ποια σήματα” και το “πώς” θα εμφανιστούν.

Το GTKWave δίνει δυνατότητα εκτύπωσης σε διανυσματική μορφή Postscript (και PDF) από το μενού File > Print to File, και το αποτέλεσμα φαίνεται στην Εικόνα 43. Το πλεονέκτημα των γραφημάτων σε διανυσματική μορφή είναι ότι επιτρέπουν απεριόριστο zoom και προσφέρουν επαγγελματική ποιότητα στην εκτύπωση σε αντίθεση με τα στιγμιότυπα σε μορφή bitmap. Το χαρακτηριστικό αυτό έχει χρησιμοποιηθεί εκτενώς στην εργασία αυτή.



Εικόνα 42: Προσομοίωση alu\_tb.vhd μέσω GHDL/GTKWave



Εικόνα 43: Εκτύπωση κυματομορφών σε Postscript από το GTKWave

## 8.4. Σενάριο προσομοίωσης του E80 με τα GHDL/GTKWave

Θα παρουσιαστεί η εκτέλεση του κώδικα που χρησιμοποιήθηκε στο σενάριο προσομοίωσης του E80 στο ModelSim:

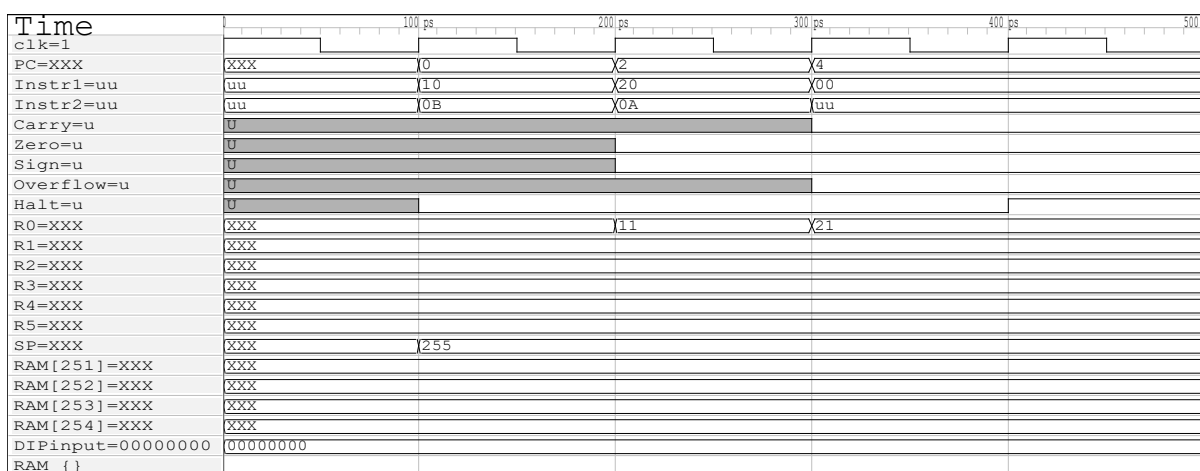
```
-- E80 Firmware, generated by the E80ASM Assembler

LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT SimDIP : WORD := "00000000"; -- DIP input for testbench only
CONSTANT Firmware : WORDx256 := (
0 => "00010000", 1 => "00001011", -- MOV R0, 11
2 => "00100000", 3 => "00001010", -- ADD R0, 10
4 => "00000000", -- HLT
OTHERS => "UUUUUUUU");END;
```

Αρχικά αντιγράφουμε τον κώδικα στο Firmware.vhd και μετά εκτελούμε το αρχείο computer\_tb.bat από τον φάκελο C:\E80\GHDL. Εμφανίζεται άμεσα το στιγμιότυπο με τις κυματομορφές στην Εικόνα 44 οι οποίες είναι ισοδύναμες με αυτές του ModelSim.

Η εντολή `computer_tb.bat` είναι μια συντόμευση της εντολής `g computer_tb 100ns`. Σύμφωνα με τον τρόπο λειτουργίας που αναλύθηκε στην προηγούμενη υποενότητα, θα προσομοιωθεί η αρχιτεκτονική `computer_tb` για 100 nanosecond. Αυτό σημαίνει ότι θα παρουσιαστούν μέχρι και 1000 βήματα εκτέλεσης.

Στο παράδειγμα αυτό η προσομοίωση τερμάτισε στο 4ο βήμα. Σε αντίθεση με το ModelSim όπου έπρεπε να αναγνωριστεί η εκτέλεση της HLT μέσω του ελέγχου της σημαίας Halt, το GHDL τερματίζει αυτόματα την εκτέλεση όταν τα σήματα σταματήσουν να μεταβάλλονται. Αυτός είναι και ο λόγος που το testbench του Υπολογιστή σταματά τους παλμούς του ρολογιού όταν `Halt=1` με το σχόλιο “if Halt=1, CLK stops pulsing ⇒ GHDL simulation ends”.



Εικόνα 44: Προσομοίωση `computer_tb.vhd` μέσω GHDL/GTKWave

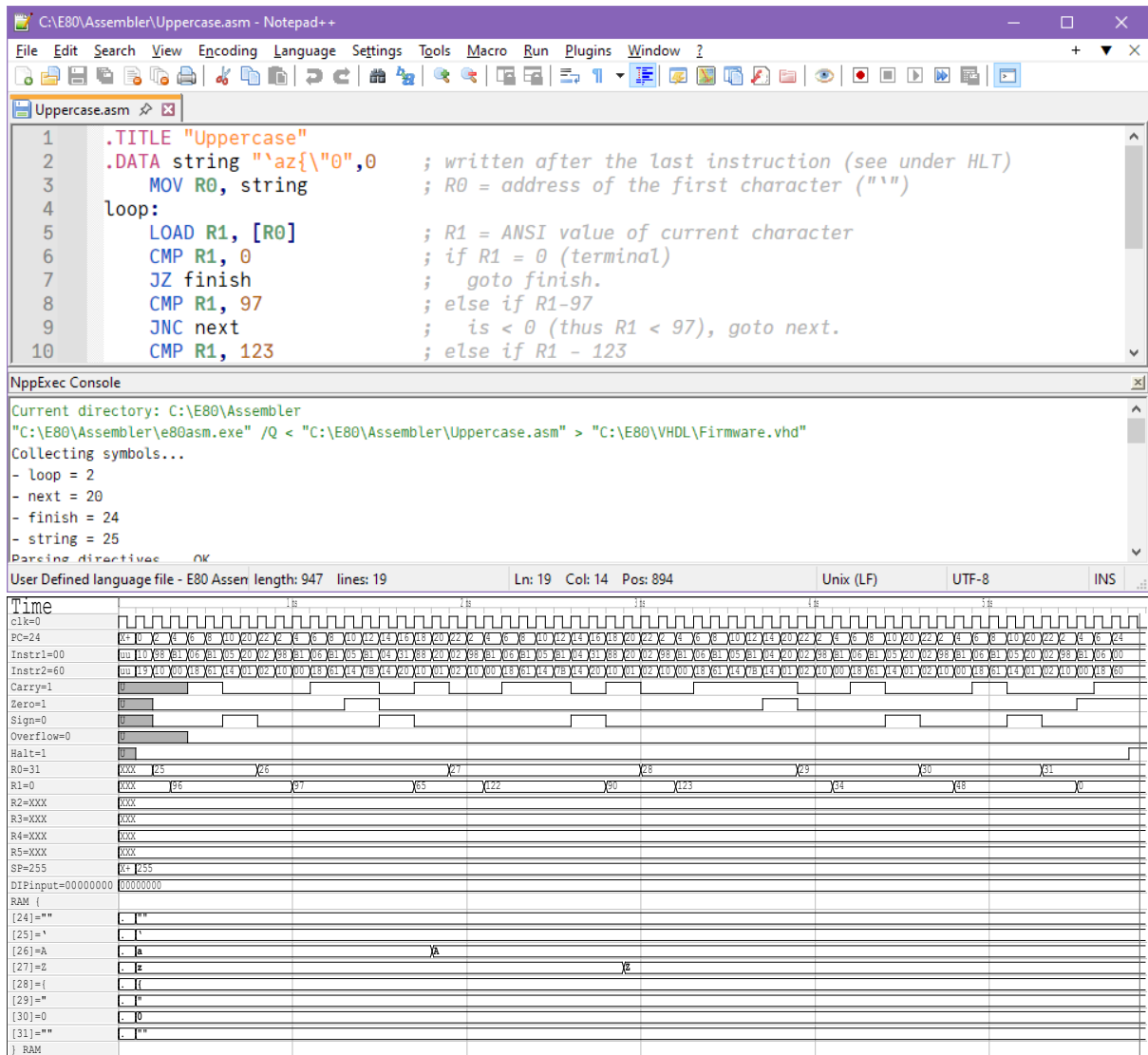
## 8.5. Απο την Assembly ως τις κυματομορφές με ένα πλήκτρο

Δεδομένου ότι ο συμβολομεταφραστής του E80 λειτουργεί με τυποποιημένη είσοδο και έξοδο (stdin, stdout, stderr) και εξάγει κωδικούς σφάλματος στην λήξη της εκτέλεσής του, η διαδικασία της συγγραφής της συμβολικής γλώσσας, του ελέγχου λαθών, της μετάφρασης, της εξαγωγής σε αρχείο VHDL και της εμφάνισης των κυματομορφών ολοκληρώθηκε μέσα από το περιβάλλον του επεξεργαστή Notepad++.

Το σενάριο που δημιουργήθηκε υποστηρίζεται από το πρόσθετο NppExec που επιτρέπει εκτέλεση προγραμμάτων που χρησιμοποιούν ως δεδομένο το τρέχον αρχείο του Notepad++. Το σενάριο καταγράφεται στο Παράρτημα Θ.1 (σ. 208) με τις κατάλληλες οδηγίες, και συνοδεύεται από μια περιγραφή της Assembly του E80 σε αρχείο XML (Παράρτημα Θ.2, σ. 209) με χρωματικούς κωδικούς για τον εύκολο έλεγχο συντακτικών λαθών.

Στην Εικόνα 45 καταγράφεται η συγγραφή του προγράμματος της Ενότητας 7.5 (σ. 93) που μετατρέπει το κείμενο “az{\'0” σε κεφαλαία. Η μετατροπή που φαίνεται στις κυματομορφές του GHDL στο κάτω μέρος της εικόνας πραγματοποιήθηκε πιέζοντας μονάχα ένα πλήκτρο.

Δεδομένου ότι η διαδικασία αυτή συμπεριλαμβάνει και την δημιουργία του κατάλληλου Firmware.vhd, μπορεί να χρησιμοποιηθεί και για την παράγωγή του κώδικα που θα μεταγλωττιστεί από τα Quartus/Gowin όπως θα παρουσιαστούν στο επόμενο Κεφάλαιο.

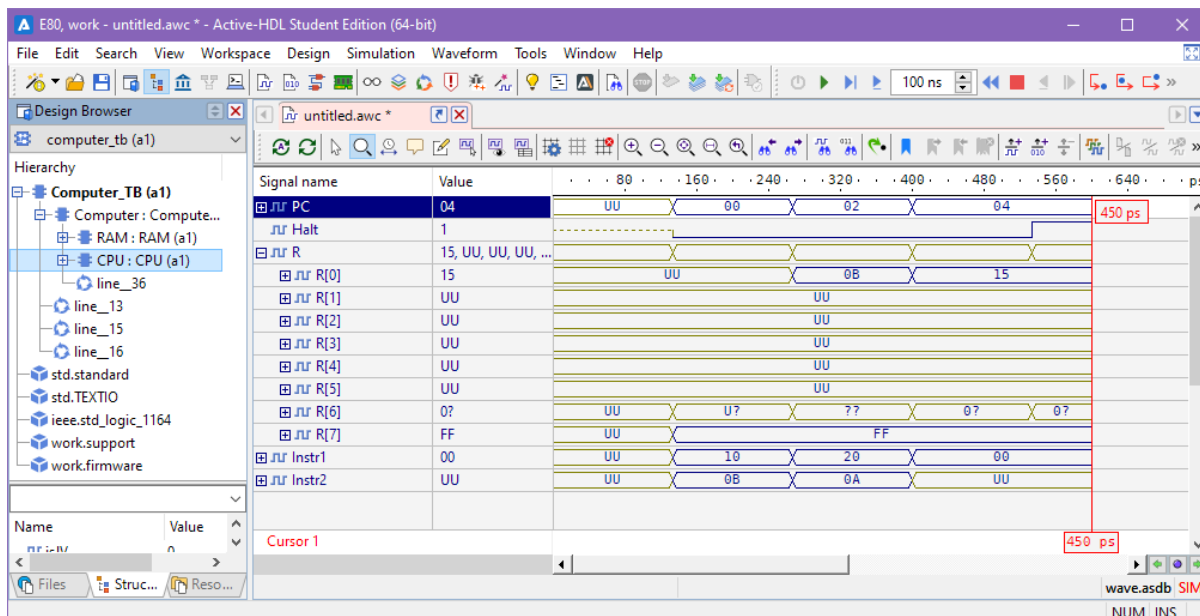


Εικόνα 45: Σγγραφή, μετάφραση, προσομοίωση με ένα πλήκτρο



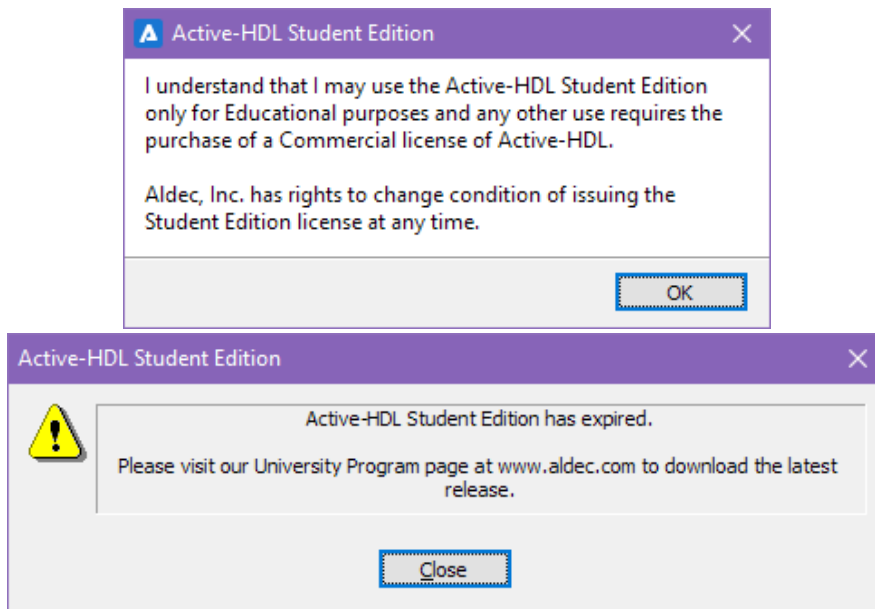
## 8.6. Aldec Active-HDL

Στην Εικόνα 46 καταγράφεται η προσομοίωση του προγράμματος που χρησιμοποιήθηκε στα ModelSim και GHDL/GTKWave με χρήση του Aldec Active-HDL.



Εικόνα 46: Προσομοίωση *computer\_tb.vhd* μέσω Aldec Active-HDL

Το Active-HDL δοκιμάστηκε για να καταγραφεί η συμβατότητα του σχεδιασμού της VHDL με διαφορετικές εφαρμογές. Κατά την έναρξη της εκτέλεσης εμφανίζει το μήνυμα που φαίνεται στο επάνω μέρος της Εικόνας 47· θέτοντας την ημερομηνία στο επόμενο έτος, το Active-HDL έπαψε να λειτουργεί όπως φαίνεται στο κάτω μέρος της εικόνας.



Εικόνα 47: Περιορισμός της άδειας χρήσης του Active-HDL

Η δυνατότητα ανάκλησης της άδειας χρήσης της φοιτητικής έκδοσης είναι ένα σοβαρό μειονέκτημα, όπως αναλύθηκε στο Υπόβαθρο (Ενότητα 2.5.1, σ. 13), και για τον λόγο αυτό δεν δόθηκε περισσότερος χρόνος στην δοκιμή των δυνατοτήτων του Active-HDL.

## 9. Εκτέλεση του E80 σε FPGA

Για την υλοποίηση του E80 σε FPGA χρησιμοποιήθηκαν οι πλακέτες DSD-i1 και Tang Primer 25K που αναφέρονται στο Υπόβαθρο. Δεδομένου ότι και οι δυο πλακέτες προσφέρουν ρολόι 50 MHz, χρησιμοποιήθηκε ο ίδιος διαιρέτης ρολογιού και δεν χρειάστηκε καμιά μεταβολή στην VHDL. Το μόνο που απαιτήθηκε, ήταν ο ορισμός των ακροδεκτών εισόδου/εξόδου και η καλωδίωση με τα απαιτούμενα εξαρτήματα σε breadboard.

Για την DSD-i1 χρησιμοποιήθηκαν:

- Η breadboard που παρέχει το ψηφιακό εργαστήριο.
- Ένα Joystick τύπου “Five Direction Navigation Button” με επιπλέον κουμπιά SET και RST.
- Ένα DIP με 8 διακόπτες.
- 16 αντιστάτες του 1 KΩ.

Για την Tang Primer 25K χρησιμοποιήθηκαν:

- Μια breadboard.
- Joystick, DIP και αντιστάτες όπως στην DSD-i1.
- 24 LED από τα οποία τα 13 κόκκινα και τα 11 πορτοκαλί.

### 9.1. Δοκιμαστικό πρόγραμμα

Για τον έλεγχο των δυνατοτήτων που θα παρουσιαστούν στις επόμενες ενότητες, κατασκευάστηκε το ακόλουθο πρόγραμμα που πραγματοποιεί 256 δεξιές περιστροφές του ενός bit στον R0. Το πρόγραμμα θέτει προσομοίωση εισόδου από τους διακόπτες DIP την λέξη “00000010”. Κατά την εκτέλεση σε FPGA οι διακόπτες DIP θα ορίζονται με αυτή την τιμή.

```
; minimal program to test the FPGA implementation and joystick control
.TITLE "256 ROR"
.SIMDIP 0b00000010      ; simulated DIP input
    LOAD R0, [0xFF]      ; load R0 with the DIP input
    MOV R1, 0
loop:
    ROR R0, 1
    ADD R1, 1
    JNC loop              ; stop after 256 iterations
    HLT
```

Εφεξής το πρόγραμμα θα ονομάζεται “256 ROR”. Ακολουθεί το αποτέλεσμα της μεταγλώττισης του στον Assembler. Το πρόγραμμα αυτό επικολλήθηκε στο αρχείο Firmware.vhd.

```
-----
-- 256 ROR
-----
LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT SimDIP : WORD := "00000010"; -- DIP input for testbench only
CONSTANT Firmware : WORDx256 := (
0  => "10010000", 1  => "11111111", -- LOAD R0, [255]
2  => "00010001", 3  => "00000000", -- MOV R1, 0
4  => "01000000", 5  => "00000001", -- ROR R0, 1
6  => "00100001", 7  => "00000001", -- ADD R1, 1
8  => "00000101", 9  => "00000100", -- JNC 4
10 => "00000000", -- HLT
```

[illegible]

## 9.2. DIP Input

Όπως αναφέρθηκε στην Ενότητα 5.8 (σ. 65), η είσοδος δεδομένων γίνεται μέσω συστοιχίας 8 διακοπών (DIP). Αυτό δίνει την δυνατότητα ανάγνωσης μιας λέξης κατά την διάρκεια του προγράμματος μέσω της εντολής `LOAD reg, [0xFF]`. Η υλοποίηση της εισόδου δεδομένων γίνεται μέσω της αντιστοίχισης ακίδων στα bits του διανύσματος `DIPinput`. Η αντιστοίχιση αυτή θα τεκμηριωθεί ξεχωριστά για τις 2 πλακέτες.

### 9.3. LED

Κατα την εκτέλεση εμφανίζονται αποτελέσματα σε 3 σειρές των 8 LED σύμφωνα με τον παρακάτω κώδικα:

```
LED_rowA(7 DOWNT0 4) <= R(6)(7 DOWNT0 4); -- CZSV flags on R6 register
WITH reg SELECT LED_rowA(3 DOWNT0 1) <=
    "000" WHEN 0, "001" WHEN 1, "010" WHEN 2, "011" WHEN 3,
    "100" WHEN 4, "101" WHEN 5, "110" WHEN 6, "111" WHEN 7;
LED_rowB <= DIPinput WHEN Reset ELSE R(reg);
LED_rowC <= PC;
```

Στην γραμμή Α εμφανίζονται απο αριστερά προς τα δεξιά:

- 4 LED για τις σημαίες CZSV
- 3 LED για την διεύθυνση του “**επιλεγμένου καταχωρητή**”. Ο όρος αυτός θα χρησιμοποιείται για τον καταχωρητή που εμφανίζεται στην σειρά B.
- 1 LED για το ρολόι. Ο σχετικός κώδικας θα παρουσιαστεί στην Ενότητα 9.5 (σ. 110).

Στην γραμμή Β εμφανίζεται ο επιλεγμένος καταχωρητής.

Στην γραμμή C εμφανίζεται ο μετρητής προγράμματος.

Όταν πιέζεται το Reset, τότε στην σειρά B εμφανίζεται η τιμή που αντιστοιχεί στους 8 διακόπτες του DIP.

## 9.4. Γεννήτρια αργού ρολογιού

Ο εκπαιδευτικός σκοπός του E80 επιβάλλει ένα αργό ρολόι που να επιτρέπει την παρακολούθηση της μεταβολής των σημάτων κατά την εκτέλεση των εντολών. Παρακάτω θα χρησιμοποιηθεί το όνομα “κανονικό” για το ρολόι των 50 MHz των FPGA και “αργό” για το ρολόι που οδηγεί τον E80 και είναι δυναμικό από 1 ως 1000 deciHertz (δηλαδή 0.1 ως 100 Hz).

Η δημιουργία του αργού ρολογιού γίνεται με ακολουθιακές εντολές που βασίζονται στο πλήθος των θετικών ακμών (Tick20ns) του κανονικού ρολογιού σύμφωνα με τον παρακάτω κώδικα:

```
PROCESS (CLK50MHz)
    VARIABLE Tick20ns : NATURAL RANGE 0 TO 249999999 := 0;
    VARIABLE Frequency : DECIHERTZ := DefaultFrequency; -- see Firmware.vhd
BEGIN
    IF RISING_EDGE(CLK50MHz) THEN
        IF NOT Pause THEN
            Tick20ns := Tick20ns + 1;
            IF Tick20ns * Frequency >= 249999999 THEN
                Tick20ns := 0;
                CLK <= NOT CLK;
            END IF;
        END IF;
    END IF;
```

Τα 50 MHz του κανονικού ρολογιού έχουν περίοδο

$$\frac{1}{50 \times 10^6} \text{ sec} = \frac{2}{10^8} \text{ sec} = 20 \text{ ns}$$

Άρα το Tick20ns αυξάνεται ανα 20 ns. Για την δημιουργία ρολογιού 1 deciHertz χρειάζεται περίοδος 10 sec. Άρα χρειάζονται

$$10 \text{ sec} = 5 \times 10^8 \times \frac{2}{10^8} \text{ sec} = 5 \times 10^8 \times 20 \text{ ns} = 5 \times 10^8 \text{ Tick20ns}$$

Επειδή όμως σε κάθε περίοδο υπάρχουν 2 παλμοί του αργού ρολογιού (δηλ. η CLK <= NOT CLK πρέπει να εκτελείται 2 φορές), η μεταβολή του θα γίνεται κάθε  $2.5 \times 10^8 \text{ Tick20ns}$ . Και επειδή η μέτρηση των Tick20ns ξεκινά από το 0, θα χρειαστούν

$$2.5 \times 10^8 - 1 = 249999999 \text{ Tick20ns}$$

Διαιρώντας τον αριθμό αυτό με την συχνότητα που θα ορίζεται από το joystick, θα έχουμε την επιθυμητή συχνότητα του αργού ρολογιού.

### 9.4.1. Παύση ρολογιού

Πιέζοντας το κουμπί SET στο Joystick, ενεργοποιείται το σήμα Pause και επομένως όλες οι αναθέσεις που αναφέρθηκαν προηγουμένως παύουν να εκτελούνται. Σημειώνεται ότι η παύση αφορά στην γεννήτρια του αργού ρολογιού και όχι στο κανονικό ρολόι. Επομένως οι κινήσεις του Joystick που αφορούν στην επιλογή του καταχωρητή και την ρύθμιση συχνότητας του αργού ρολογιού μπορούν να εκτελούνται κανονικά κατά την παύση.

## 9.5. LED ρολογιού

Το LED του ρολογιού είναι το δεξιότερο LED της σειράς A, δηλ. το LED\_rowA(0). Το LED αυτό δεν περιορίζεται στην ένδειξη του παλμού του ρολογιού· συνδυάζοντας μεταβολή φωτεινότητας με PWM και παύση παλμικής λειτουργίας οπτικοποιεί τρεις διαφορετικές καταστάσεις:

- Τον τερματισμό εκτέλεσης λόγω σημαίας Αλτ με την ανάθεση του LED σε σταθερό και κανονικό φωτισμό.
- Την ολοκλήρωση ενός σύγχρονου Reset με παλμικό αλλά κανονικό φωτισμό.
- Την κανονική λειτουργία του ρολογιού με παλμικό και αχνό φωτισμό.

Οι διαφοροποιήσεις αυτές κρίθηκαν απαραίτητες κατά τις δοκιμές προγραμμάτων σε FPGA διότι:

- Η σημαία Αλτ μπορεί να ενεργοποιηθεί από εκτέλεση μη-προβλεπόμενου κώδικα λόγω εσφαλμένου άλματος σε μη-αρχικοποιημένη διεύθυνση μνήμης. Άρα δεν είναι το ίδιο με ένα ατέρμονα βρόχο.
- Το Reset είναι προϋπόθεση για το φόρτωμα του Firmware στην μνήμη. Επειδή όμως έχει σχεδιαστεί ως σύγχρονο (βλέπε Ενότητα 5.10, σ. 82) με το αργό ρολόι της τάξεως deciHertz, δεν συνάδει με την εμπειρία που έχει ένας χρήστης για την λειτουργία του Reset σε κανονικές συσκευές. Πρέπει επομένως να φαίνεται η επιτυχής ολοκλήρωση του Reset.
- Η παλμική λειτουργία του ρολογιού με κανονικό φωτισμό ήτανε ιδιαίτερα ενοχλητική κατά την μελέτη των σημάτων στην εκτέλεση κώδικα.

Για τους λόγους αυτούς υλοποιήθηκε ο παρακάτω κώδικας που διαχειρίζεται το LED του ρολογιού:

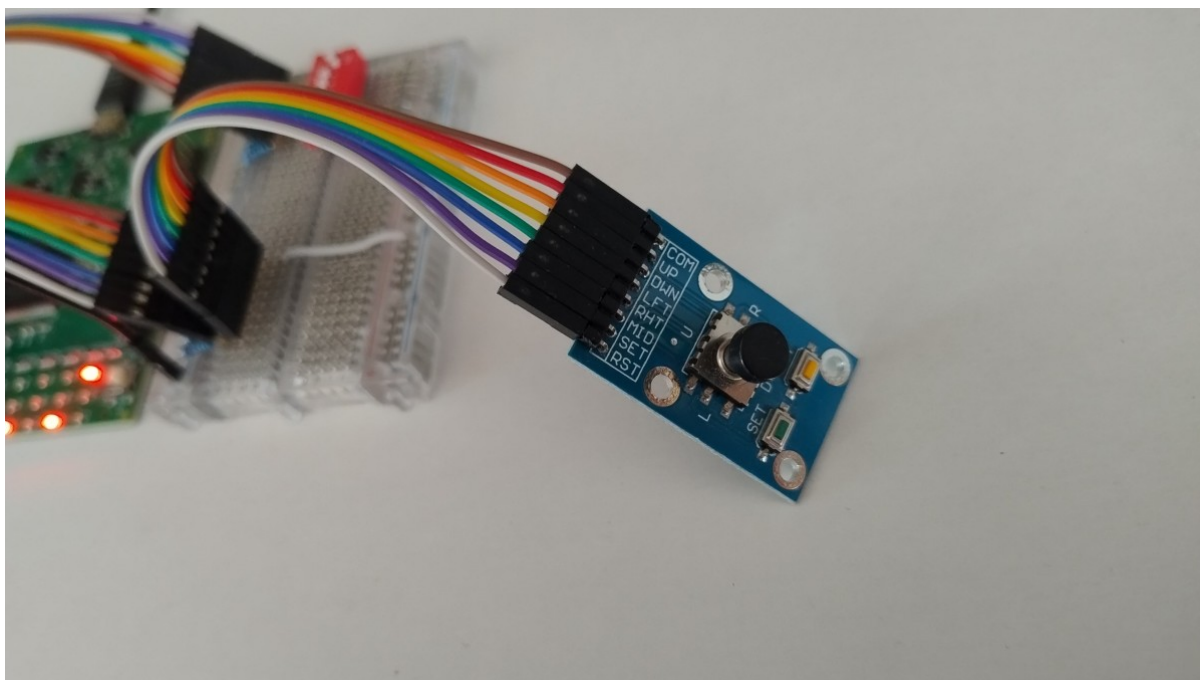
```
PROCESS (CLK50MHz)
    IF RISING_EDGE(CLK50MHz) THEN
        IF NOT Pause THEN
            IF Tick20ns * Frequency >= 249999999 THEN
                Tick20ns := 0;
                CLK <= NOT CLK;
                ResetComplete <= CLK AND Reset;
            ...
            LED_rowA(0) <=
                '1'      WHEN Halt AND NOT Reset ELSE -- solid, bright
                '1'      WHEN ResetComplete      ELSE -- pulse, bright
                CLK50MHz WHEN CLK                  ELSE -- pulse, dim
                '0';
```

Απο την μελέτη του ακολουθιακού κώδικα στην PROCESS προκύπτει ότι το σήμα ResetComplete εκφράζει ότι έχει γίνει ένα σύγχρονο reset. Και αυτό διότι σύμφωνα με τον κώδικα, η ανάθεση CLK AND Reset προϋποθέτει ύπαρξη ακμής του αργού ρολογιού λόγω της συνθήκης Tick20ns \* Frequency >= 249999999 που αναλύθηκε στην Ενότητα 9.4. Άρα το ResetComplete εκφράζει ότι το κουμπί Reset βρίσκεται πιεσμένο στην στιγμή μιας θετικής ακμής του αργού ρολογιού. Επομένως:

- Όταν έχει ενεργοποιηθεί η σημαία Halt και δεν πιέζεται το κουμπί Reset, το LED του ρολογιού παραμένει σταθερό σε κανονικό (έντονο) φωτισμό.
- Όταν το αργό ρολόι είναι στο 1 και έχει ολοκληρωθεί ένα σύγχρονο Reset, το LED του ρολογιού ακολουθεί το ResetComplete και επομένως αναβοσβήνει με κανονικό (έντονο) φωτισμό.
- Σε αντίθετη περίπτωση, όταν το αργό ρολόι είναι στο 1, το LED πάλλεται με ρυθμό 50 MHz, και επομένως με χαμηλή φωτεινότητα 50% μέσω PWM.

## 9.6. Joystick

Για τον έλεγχο της ταχύτητας και των ενδείξεων του Υπολογιστή E80 χρησιμοποιήθηκε το Joystick που φαίνεται στην Εικόνα 49 με 5 κατευθύνσεις και κουμπιά SET και RST. Το Joystick έφερε την εμπορική ονομασία “Five Direction Navigation Button Module 5D Rocker Joystick” και το κόστος ήταν αμελητέο.



Εικόνα 49: Joystick ελέγχου ταχύτητας, ενδείξεων και Reset

Η λειτουργία του Joystick ελέγχεται από τις εξής ακολουθιακές εντολές:

```
SIGNAL reg: NATURAL RANGE 0 TO 7 := 0; -- current register address
SIGNAL ResetComplete : STD_LOGIC := '0';
...
PROCESS (CLK50MHz)
    CONSTANT RepeatRate : NATURAL := 18000000; -- x 2/10^8 = 0.36 sec
    VARIABLE Delay : NATURAL RANGE 0 TO RepeatRate := 0;
...
    -- handle joystick inputs with a delay to prevent rapid toggling
    IF Delay < RepeatRate THEN
        Delay := Delay + 1;
    ELSIF Up THEN
        reg <= reg + 1;
        Delay := 0;
    ELSIF Down THEN
        reg <= reg - 1;
        Delay := 0;
    ELSIF Right THEN -- increase CLK speed
```



```

IF Frequency > 850 THEN
    Frequency := 1000; -- ceiling
ELSE
    Frequency := Frequency + Frequency/8 + 2;
END IF;
Delay := 0;
ELSIF Left THEN -- decrease CLK speed
    IF Frequency < 3 THEN
        Frequency := 1; -- floor
    ELSE
        Frequency := Frequency - Frequency/8 - 2;
    END IF;
    Delay := 0;
ELSIF Mid THEN
    Frequency := DefaultFrequency;
    Delay := 0;
END IF;

```

Οι εντολές αυτές μεταφράζονται στις εξής δυνατότητες:

- Κατεύθυνση πάνω: η διεύθυνση του επιλεγμένου καταχωρητή (reg) αυξάνεται κατά 1. Αν για παράδειγμα ο επιλεγμένος καταχωρητής είναι ο R6, τότε μια πίεση προς τα πάνω θα ενεργοποιήσει τον R7 και διατηρώντας την, θα ενεργοποιηθεί ο R0.
- Κατεύθυνση κάτω: η διεύθυνση του επιλεγμένου καταχωρητή (reg) μειώνεται κατά 1. Αν για παράδειγμα ο επιλεγμένος καταχωρητής είναι ο R1, τότε μια πίεση προς τα κάτω θα ενεργοποιήσει τον R0 και διατηρώντας την, θα ενεργοποιηθεί ο R7.
- Κατευθύνσεις αριστερά/δεξιά: μείωση/αύξηση συχνότητας επεξεργαστή. Η αυξομείωση της συχνότητας δεν γίνεται με σταθερό βήμα, αλλά σύμφωνα με την τρέχουσα συχνότητα. Αυτό εξασφαλίζει ότι ο χρόνος που χρειάζεται για την αύξηση στο μέγιστο (ή την μείωση στο ελάχιστο) είναι μικρός.
- Κεντρική κατεύθυνση (πίεση του μοχλού απο πάνω): επαναφορά προεπιλεγμένης συχνότητας.
- κουμπί SET: Παύση ρολογιού.
- κουμπί RST: Reset.

Κατα την δοκιμή του συστήματος προέκυψε ότι η χρήση του Joystick ήτανε πρακτικά αδύνατη επειδή ο έλεγχός του γίνεται στο κανονικό ρολόι των 50 MHz: το παραμικρό κλικ έστελνε χιλιάδες μεταβολές. Για τον λόγο αυτό χρησιμοποιήθηκε η προσέγγιση του “ρυθμού επανάληψης πληκτρολογίου” που χρησιμοποιείται σε κανονικούς υπολογιστές, μέσω της σταθεράς RepeatRate και της μεταβλητής Delay.

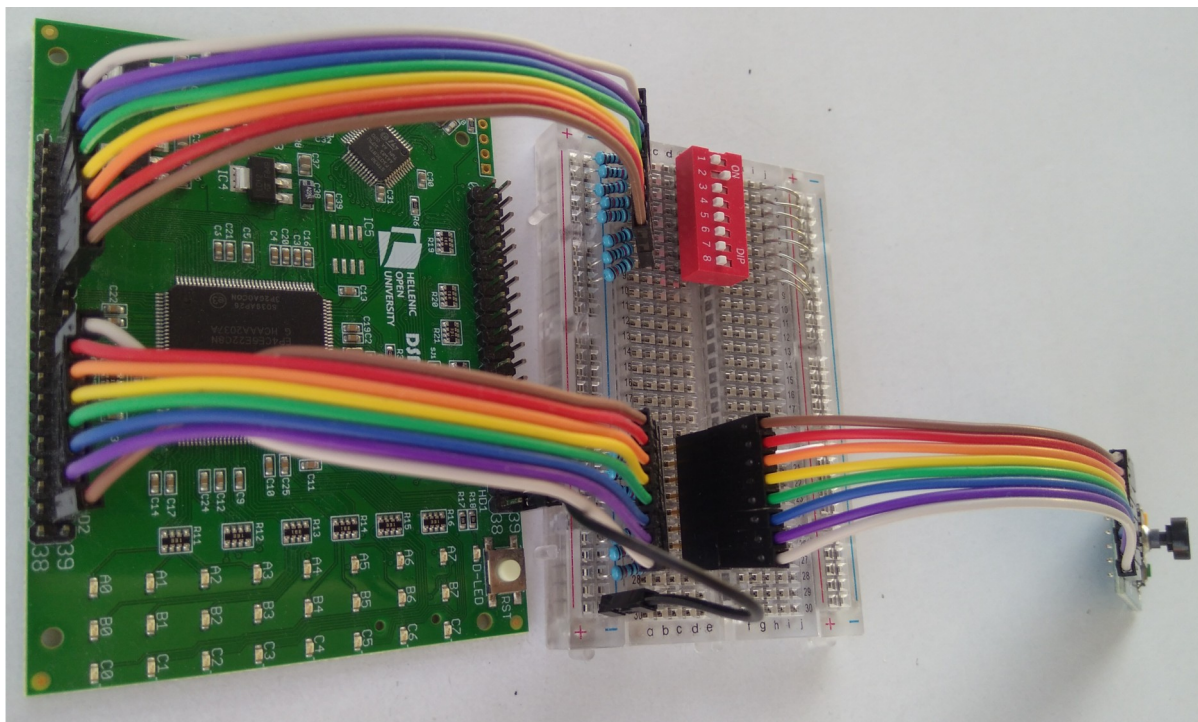
Η RepeatRate εκφράζει την ελάχιστη χρονική περίοδο που πρέπει να μεσολαβήσει μεταξύ δύο διαδοχικών ενεργειών του joystick. Η Delay λειτουργεί ως χρονόμετρο που, μετά από κάθε έγκυρη ενέργεια, μηδενίζεται και πρέπει να φτάσει ξανά την τιμή της RepeatRate πριν επιτραπεί η επόμενη ενέργεια. Μετά απο δοκιμές (και σε χρήστες ηλικίας απο 15 ως 65 ετών) κρίθηκε ότι η χρυσή τομή για το RepeatRate βρίσκεται γύρω στα 0.3 - 0.4 sec.



## 9.7. Quartus & DSD-i1

Η σύνδεση της DSD-i1 με τα εξαρτήματα της breadboard του ΕΑΠ έγινε με αποκλειστικό γνώμονα την απλότητα της καλωδίωσης. Για τον λόγο αυτό χρησιμοποιήθηκαν ακροδέκτες που έχουν διπλή λειτουργία<sup>1</sup> και χρειάστηκε η ρύθμιση *Assignments > Device > Device and Pin Options > Dual-Purpose Pins > nCEO > Use as regular I/O* που επέτρεψε την λειτουργία τους για είσοδο/έξοδο.

Στην Εικόνα 50 παρουσιάζεται η μορφή του συστήματος και οι καλωδιώσεις.



Εικόνα 50: Σύνδεση εξαρτημάτων στην DSD-i1

Ένα πλεονέκτημα της DSD-i1 σε σχέση με την Tang Primer 25K είναι η ύπαρξη των 24+1 ενσωματωμένων LED. Αφενός τα LED μείωσαν το πλήθος των απαιτούμενων καλωδιώσεων, αφετέρου στην πράξη φάνηκαν πολύ πιο ευκρινή σε σχέση με τα LED που χρησιμοποιήθηκαν στην Tang Primer 25K.

Το σημείο αυτό είχε εξέχουσα σημασία στον συγγραφέα της εργασίας αυτής, λόγω δυσχρωματοψίας που τον ανάγκασε να λειτουργήσει τα LED της Tang Primer χωρίς αντιστάτες για μπορεί να τα βλέπει.

<sup>1</sup> <https://community.intel.com/t5/Programmable-Devices/nCEO-as-general-purpose-I-O/td-p/53206>

### 9.7.1. Ανάθεση ακροδεκτών

Η ανάθεση των ακροδεκτών της DSD-i1 αρχικά έγινε στον Pin Planner του Quartus όπως φαίνεται στην Εικόνα 51 αλλά τελικά φάνηκε ευκολότερη η επεξεργασία του αρχείου ρυθμίσεων E80.qsf που καταγράφεται στο Παράρτημα ΣΤ.1 (σ. 181). Παρουσιάζονται συνοπτικά οι αναθέσεις σημάτων εισόδου / εξόδου στους Πίνακες 10 και 11 που ακολουθούν.

Pin Planner - C:/E80/Quartus/E80 - E80

File Edit View Processing Tools Window Help

Search altera.com

Groups

Named: \*

Node Name Direction

in DIPinput[7..0] Input

out LED\_rowA[7..0] Output

out LED\_rowB[7..0] Output

Groups Report

Tasks

Early Pin Planning

Early Pin Planning...

Run I/O Assignmer

Top View

Wire Bond, with Exposed Pad

Cyclone IV E

EP4CE6E22C8

Pin Legend

Symbol Pin Type

User I/O

User assigned I...

Fitter assigned I...

Unbonded pad

Reserved pin

Other configura...

DEV\_OE

DEV\_CLR

DIFF\_n

DIFF\_p

Named: \* Edit: CLK50MHz Filter: Pins: all

Node Name	Direction	Location	Fitter Location	I/O Standard	Current Strength	Slew Rate
in CLK50MHz	Input	PIN_23	PIN_23	3.3-V LVTTTL	8mA (default)	
in DIPinput[7]	Input	PIN_73	PIN_73	3.3-V LVTTTL	8mA (default)	
in DIPinput[6]	Input	PIN_71	PIN_71	3.3-V LVTTTL	8mA (default)	
in DIPinput[5]	Input	PIN_69	PIN_69	3.3-V LVTTTL	8mA (default)	
in DIPinput[4]	Input	PIN_67	PIN_67	3.3-V LVTTTL	8mA (default)	
in DIPinput[3]	Input	PIN_65	PIN_65	3.3-V LVTTTL	8mA (default)	
in DIPinput[2]	Input	PIN_60	PIN_60	3.3-V LVTTTL	8mA (default)	
in DIPinput[1]	Input	PIN_58	PIN_58	3.3-V LVTTTL	8mA (default)	
in DIPinput[0]	Input	PIN_54	PIN_54	3.3-V LVTTTL	8mA (default)	
in Down	Input	PIN_89	PIN_89	3.3-V LVTTTL	8mA (default)	
out LED_rowA[7]	Output	PIN_110	PIN_110	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[6]	Output	PIN_111	PIN_111	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[5]	Output	PIN_112	PIN_112	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[4]	Output	PIN_113	PIN_113	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[3]	Output	PIN_114	PIN_114	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[2]	Output	PIN_115	PIN_115	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[1]	Output	PIN_119	PIN_119	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowA[0]	Output	PIN_120	PIN_120	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowB[7]	Output	PIN_121	PIN_121	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowB[6]	Output	PIN_124	PIN_124	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowB[5]	Output	PIN_125	PIN_125	3.3-V LVTTTL	8mA (default)	2 (default)
out LED_rowB[4]	Output	PIN_126	PIN_126	3.3-V LVTTTL	8mA (default)	2 (default)

0% 00:00:00

Εικόνα 51: Ανάθεση ακροδεκτών μέσω του Pin Planner στο Quartus

Ακροδέκτης	Σήμα	Παρατηρήσεις
23	CLK50MHz	Εσωτερικό ρολόι 50 MHz
85	Reset	Joystick RST active High, pull down 10 kΩ
87	Up	Joystick UP active High, pull down 10 kΩ
89	Down	Joystick DWN active High, pull down 10 kΩ
91	Left	Joystick LFT active High, pull down 10 kΩ
99	Right	Joystick RHT active High, pull down 10 kΩ
101	Mid	Joystick MID active High, pull down 10 kΩ
104	Pause	Joystick SET active High, pull down 10 kΩ
3.3V		Joystick COM
54	DIPinput[0]	Active High, pull down 10 kΩ
58	DIPinput[1]	Active High, pull down 10 kΩ
60	DIPinput[2]	Active High, pull down 10 kΩ
65	DIPinput[3]	Active High, pull down 10 kΩ
67	DIPinput[4]	Active High, pull down 10 kΩ
69	DIPinput[5]	Active High, pull down 10 kΩ
71	DIPinput[6]	Active High, pull down 10 kΩ
73	DIPinput[7]	Active High, pull down 10 kΩ

Πίνακας 10: Αναθέσεις σημάτων εισόδου στους ακροδέκτες της DSD-il

Ακροδέκτης	Σήμα	Παρατηρήσεις
110	LED_rowA[7]	Carry
111	LED_rowA[6]	Zero
112	LED_rowA[5]	Sign
113	LED_rowA[4]	Overflow
114	LED_rowA[3]	ΠΣΨ αριθμού/διεύθυνσης επιλεγμένου καταχωρητή
115	LED_rowA[2]	
119	LED_rowA[1]	ΛΣΨ αριθμού/διεύθυνσης επιλεγμένου καταχωρητή
120	LED_rowA[0]	Ένδειξη αργού ρολογιού
121	LED_rowB[7]	ΠΣΨ τιμής επιλεγμένου καταχωρητή
124	LED_rowB[6]	
125	LED_rowB[5]	
126	LED_rowB[4]	
127	LED_rowB[3]	
128	LED_rowB[2]	
129	LED_rowB[1]	
132	LED_rowB[0]	ΛΣΨ τιμής επιλεγμένου καταχωρητή
133	LED_rowC[7]	ΠΣΨ μετρητή προγράμματος
135	LED_rowC[6]	
136	LED_rowC[5]	
137	LED_rowC[4]	
138	LED_rowC[3]	
141	LED_rowC[2]	
142	LED_rowC[1]	
143	LED_rowC[0]	ΛΣΨ μετρητή προγράμματος

Πίνακας 11: Αναθέσεις σημάτων εξόδου στους ακροδέκτες της DSD-i1

### 9.7.2. Οδηγίες σύνθεσης και εκτέλεσης

Εφόσον έχουν πραγματοποιηθεί οι απαραίτητες συνδέσεις, η σύνθεση του έργου στο Quartus και η εκτέλεσή του στην πλακέτα DSD-i1 θα γίνει εύκολα. Θεωρώντας ότι έχουν αποσυμπίεστεί τα αρχεία σύμφωνα με τις οδηγίες που είχανε δοθεί στην εισαγωγή της Ενότητας 8 (σ. 95), αρκεί να ανοίξουμε το Quartus και μετά μέσω File > Open Project να ανοίξουμε το αρχείο C:\E80\Quartus\E80.qpf. Όπως έχει σημειωθεί στην Ενότητα 8, όλα τα αρχεία σε αυτή την εργασία θα λειτουργήσουν ανεξαρτήτως επιλεγμένου φακέλου αποσυμπίεσης του έργου.

Κάνουμε Compile και μετά ανοίγουμε τον X2 Loader για να ανεβάσουμε το αρχείο .RBF στην πλακέτα.

Αρχικά θέτουμε τους διακόπτες DIP στις θέσεις 00000010 έτσι ώστε η εκτέλεση να είναι σε συμφωνία με την οδηγία .SIMDIP που χρησιμοποιήθηκε στην προσομοίωση.

Μόλις ανέβει το έργο παρατηρούμε ότι το LED A7 που εκφράζει το ρολόι παραμένει σταθερό φωτεινό. Και αυτό διότι το Firmware του προγράμματος που προετοιμάστηκε στην Ενότητα 9.1 (σ. 107) δεν έχει ακόμα φορτωθεί, οπότε τα flip flop της μνήμης είναι σε απροσδιόριστη κατάσταση η οποία αντιστοιχίζεται από το Quartus στο 0. Επομένως, με τον Program Counter να έχει ξεκινήσει από 0, εκτελέστηκε η εντολή "00000000" που αντιστοιχεί στην HLT. Άρα το LED A7 δείχνει κατάσταση Αλτ.

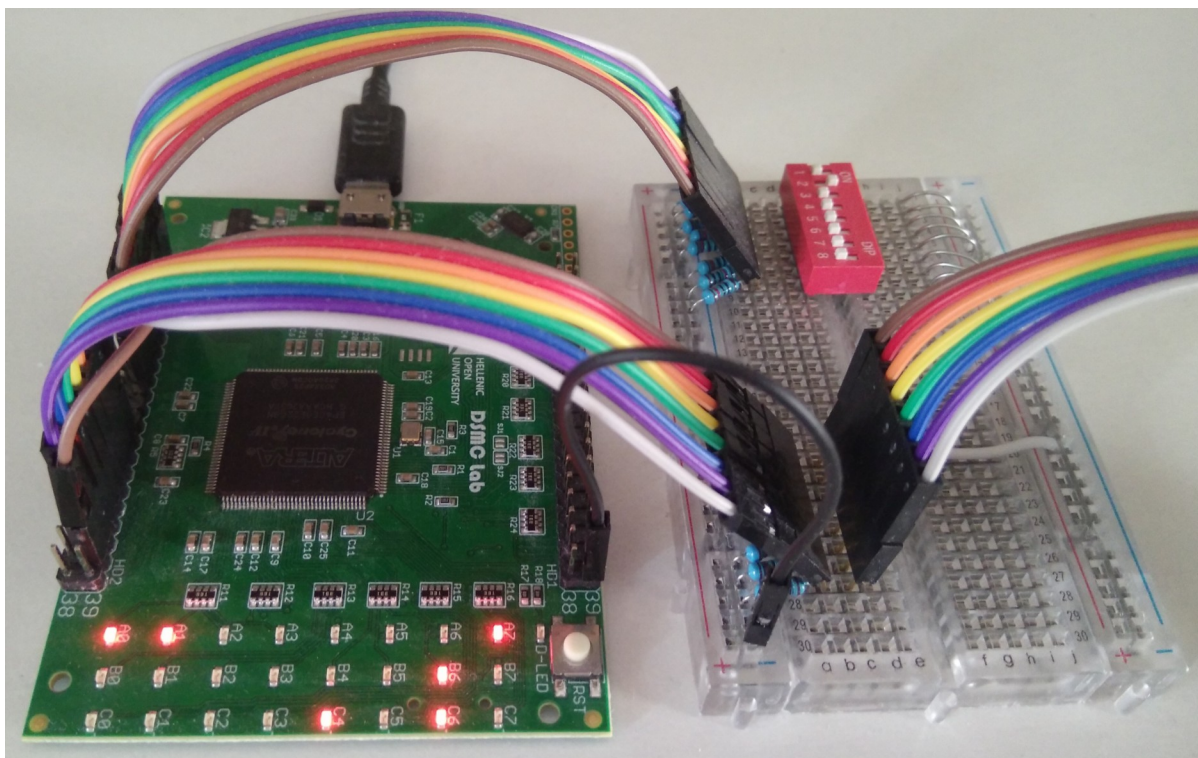
Για να φορτώσουμε το Firmware πιέζουμε το Reset και παρατηρούμε ότι το LED A7 αναβοσβήνει ζωηρά. Επομένως το Firmware φορτώθηκε. Αφήνουμε το Reset και το πρόγραμμα ξεκινά να λειτουργεί. Κατά την εκτέλεση μπορούμε να πιέσουμε το joystick δεξιά για να αυξήσουμε την συχνότητα του επεξεργαστή.

Το πρόγραμμα θα ολοκληρώσει την εκτέλεσή του, και μετά από αρκετές περιστροφές το LED A7 θα μείνει σταθερά φωτισμένο. Η φάση αυτή καταγράφεται στην Εικόνα 52.

- Τα τέσσερα αριστερά LED στην γραμμή A καταγράφουν τις σημαίες, άρα C=1, Z=1, S=0, V=0. Τα επόμενα τρία LED καταγράφουν τον επιλεγμένο καταχωρητή, επομένως τον R0.
- Στην γραμμή B καταγράφεται η τιμή του επιλεγμένου καταχωρητή. Παρατηρούμε ότι είναι ίδια με την αρχική που είχε ληφθεί από τους διακόπτες DIP, δεδομένου ότι το πλήθος των περιστροφών (256) είναι πολλαπλάσιο του 8.
- Ο μετρητής προγράμματος έχει τιμή 10 το οποίο επιβεβαιώνει την γραμμή "10 => "00000000", -- HLT" στον κώδικα μηχανής σε VHDL.

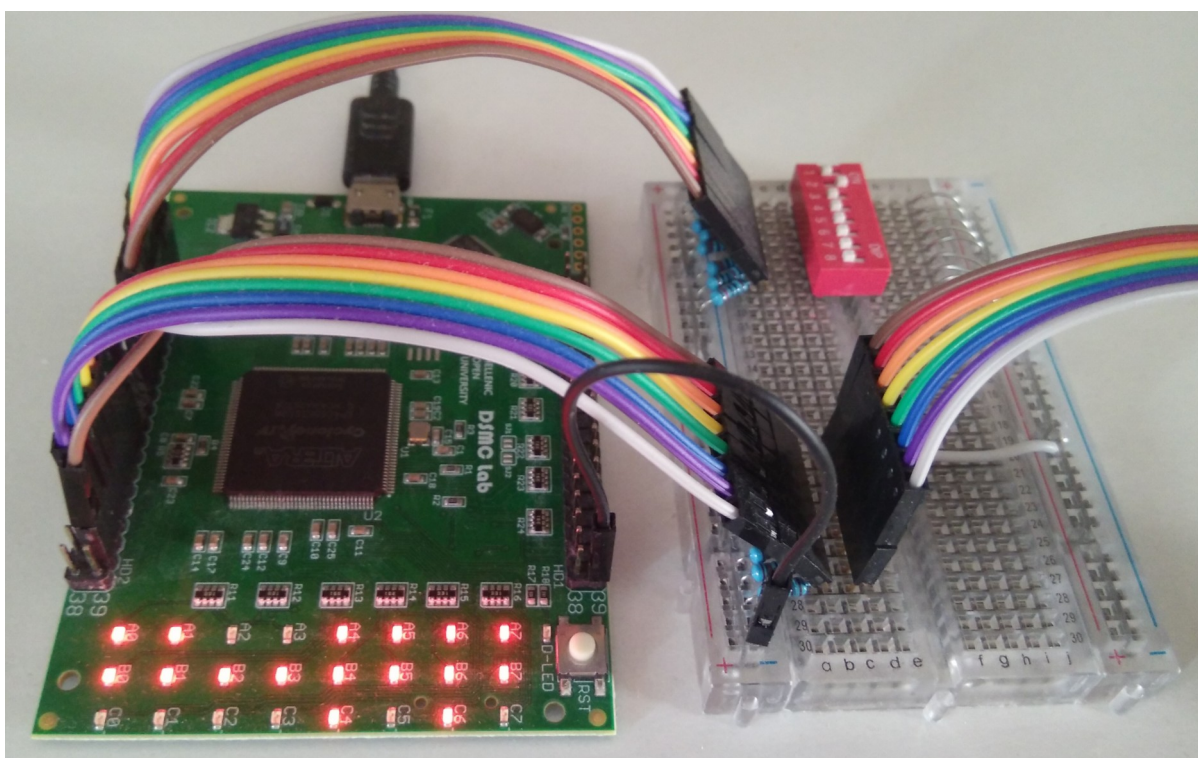
Όλα τα παραπάνω επιβεβαιώνονται από τα αποτελέσματα της προσομοίωσης στην Εικόνα 48 (σ. 108).





Εικόνα 52: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής R0

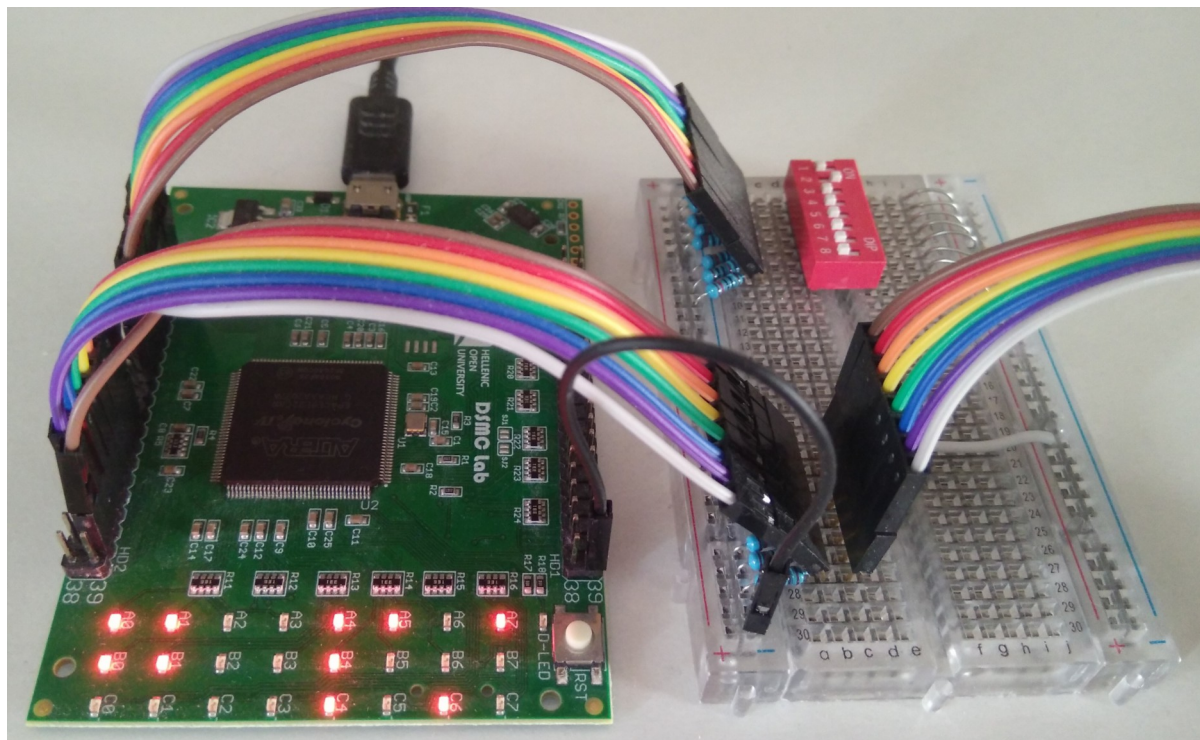
Πιέζουμε το joystick προς τα κάτω για να επιλέξουμε τον καταχωρητή R7 ο οποίος είναι ο Stack Pointer. Όπως εμφανίζεται στην Εικόνα 53, έχει τιμή 255 άρα, σύμφωνα με την τεκμηρίωση της Ενότητας 3.9 (σ. 33), η στοίβα είναι άδεια.



Εικόνα 53: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής SP (R7)

Τέλος, πιέζουμε το joystick και πάλι προς τα κάτω, και επιλέγεται ο R6, που είναι ο καταχωρητής των σημαιών. Δεδομένης της σειράς των σημαιών στον R6 (C,Z,S,V,H) επαληθεύεται η αντιστοιχία του με τα τέσσερα αριστερά LED της γραμμής A που εκφράζουν τις ίδιες σημαίες.

Στον R6 επίσης φαίνεται και η σημαία Halt που έχει ενεργοποιηθεί λόγω εκτέλεσης εντολής HLT.

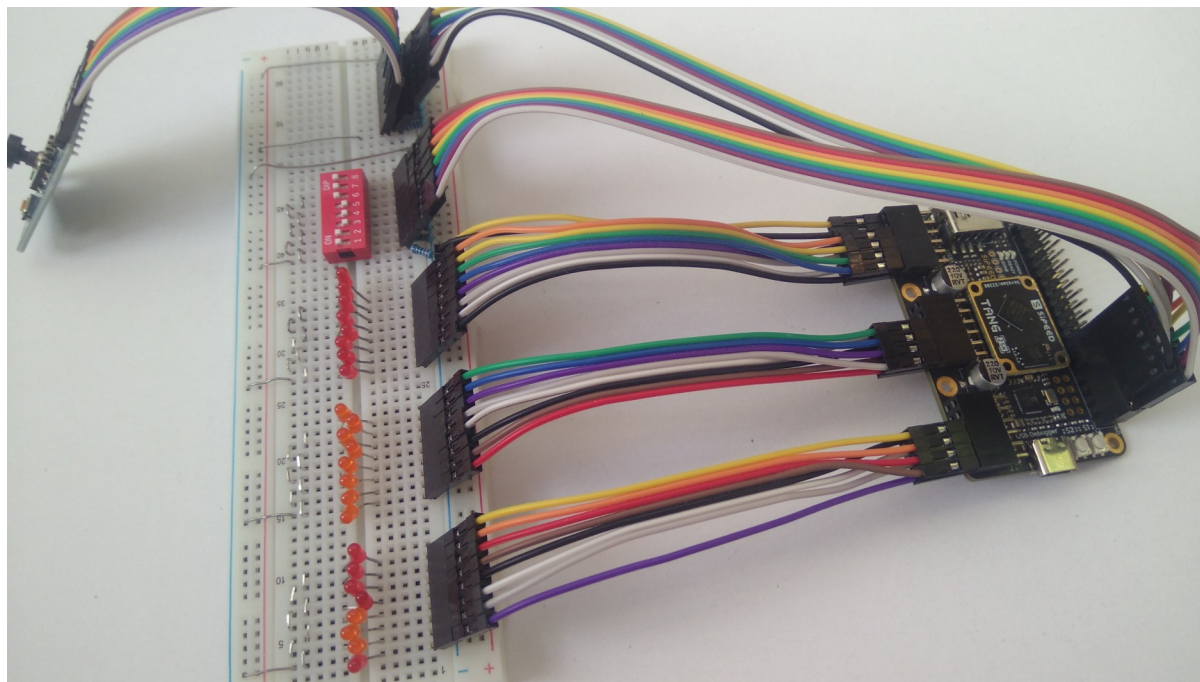


Εικόνα 54: Ολοκλήρωση “256 ROR” στην DSD-i1, καταχωρητής FLAGS (R6)



## 9.8. Gowin & Tang Primer 25K

Όπως και στην DSD-i1, η σύνδεση της Tang Primer 25K έγινε με γνώμονα την απλότητα στην καλωδίωση.



Εικόνα 55: Σύνδεση εξαρτημάτων στην Tang Primer 25K

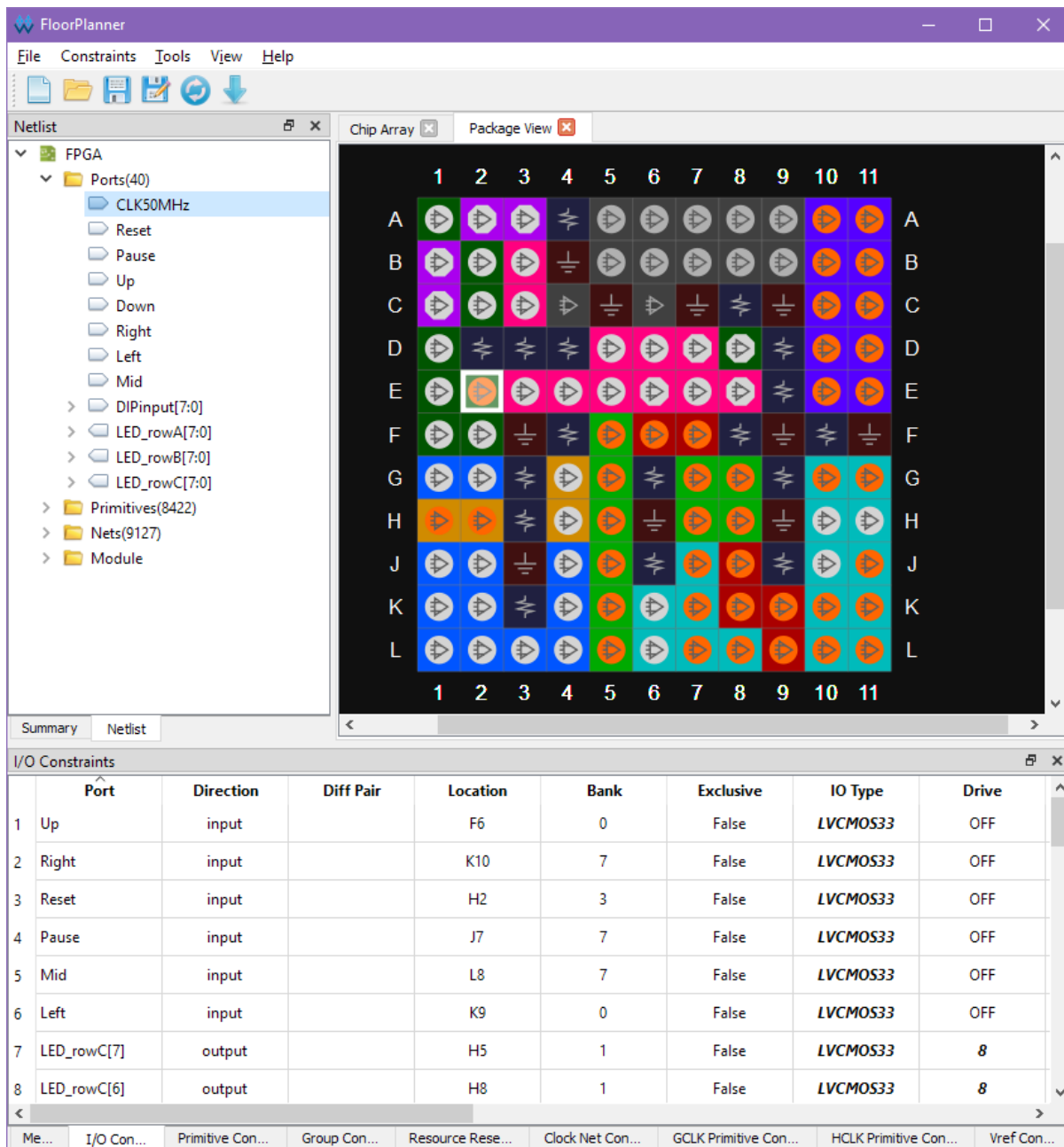
Στην Εικόνα 55 καταγράφονται οι ομάδες των LED από πάνω προς τα κάτω:

- Στην πάνω ομάδα, ακριβώς μετά τους διακόπτες DIP, είναι τα 8 κόκκινα LED του μετρητή προγράμματος.
- Στην μεσαία ομάδα καταγράφεται η τιμή του επιλεγμένου καταχωρητή με 8 πορτοκαλί LED.
- Στην κάτω ομάδα υπάρχουν 8 LED:
  - τα πάνω τέσσερα κόκκινα LED είναι οι σημαίες CZSV,
  - τα τρία πορτοκαλί είναι αριθμός/διεύθυνση του επιλεγμένου καταχωρητή,
  - ενώ το κατώτατο κόκκινο LED είναι το ρολόι.

Σε αντίθεση με τις συνδέσεις του joystick και του DIP, δεν έγινε χρήση αντιστατών στα LED. Αν και αυτό δεν είναι σωστή πρακτική, αποφασίστηκε κατά την διάρκεια των δοκιμών όπου τα LED ήτανε δυσδιάκριτα, λόγω δυσχρωματοψίας του συγγραφέα της εργασίας, ακόμα και με αντιστάσεις λίγων Ωμ.

### 9.8.1. Ανάθεση ακροδεκτών

Όπως και στο Quartus/DSD-i1, η ανάθεση ακροδεκτών της Tang Primer στο Gowin ξεκίνησε απο την λειτουργία Tools > Floor Planner όπως καταγράφεται στην Εικόνα 56 αλλά ολοκληρώθηκε με την επεξεργασία του αρχείου Gowin.sdc (Παράρτημα Z.4, σ. 186).



Εικόνα 56: Ανάθεση ακροδεκτών μέσω του Floor Planner στο Gowin

Παρουσιάζονται συνοπτικά οι αναθέσεις σημάτων εισόδου / εξόδου στους Πίνακες 12 και 13 που ακολουθούν. Σημειώνεται οτι οι αναθέσεις αυτές έχουν γίνει στο αρχείο Gowin.sdc και επομένως αρκεί το άνοιγμα του project C:\E80\Gowin\Gowin.gprj για άμεση σύνθεση και εκτέλεση.

Ακροδέκτης	Σήμα	Παρατηρήσεις
E2	CLK50MHz	Εσωτερικό ρολόι 50 MHz
H2	Reset	Joystick RST active High, pull down 10 kΩ
F6	Up	Joystick UP active High, pull down 10 kΩ
K8	Down	Joystick DWN active High, pull down 10 kΩ
K9	Left	Joystick LFT active High, pull down 10 kΩ
K10	Right	Joystick RHT active High, pull down 10 kΩ
L8	Mid	Joystick MID active High, pull down 10 kΩ
J7	Pause	Joystick SET active High, pull down 10 kΩ
3.3V		Joystick COM
J11	DIPinput[0]	Active High, pull down 10 kΩ
F7	DIPinput[1]	Active High, pull down 10 kΩ
J8	DIPinput[2]	Active High, pull down 10 kΩ
L9	DIPinput[3]	Active High, pull down 10 kΩ
L10	DIPinput[4]	Active High, pull down 10 kΩ
L7	DIPinput[5]	Active High, pull down 10 kΩ
K7	DIPinput[6]	Active High, pull down 10 kΩ
H1	DIPinput[7]	Active High, pull down 10 kΩ

Πίνακας 12: Αναθέσεις σημάτων εισόδου στους ακροδέκτες της Tang Primer 25K

Ακροδέκτης	Σήμα	Παρατηρήσεις
C11	LED_rowA[7]	Carry
B11	LED_rowA[6]	Zero
D11	LED_rowA[5]	Sign
G11	LED_rowA[4]	Overflow
C10	LED_rowA[3]	ΠΣΨ αριθμού/διεύθυνσης επιλεγμένου καταχωρητή
B10	LED_rowA[2]	
D10	LED_rowA[1]	ΛΣΨ αριθμού/διεύθυνσης επιλεγμένου καταχωρητή
G10	LED_rowA[0]	Ένδειξη αργού ρολογιού
L5	LED_rowB[7]	ΠΣΨ τιμής επιλεγμένου καταχωρητή
K11	LED_rowB[6]	
E11	LED_rowB[5]	
A11	LED_rowB[4]	
K5	LED_rowB[3]	
L11	LED_rowB[2]	
E10	LED_rowB[1]	
A10	LED_rowB[0]	ΛΣΨ τιμής επιλεγμένου καταχωρητή
H5	LED_rowC[7]	ΠΣΨ μετρητή προγράμματος
H8	LED_rowC[6]	
G7	LED_rowC[5]	
F5	LED_rowC[4]	
J5	LED_rowC[3]	
H7	LED_rowC[2]	
G8	LED_rowC[1]	
G5	LED_rowC[0]	ΛΣΨ μετρητή προγράμματος

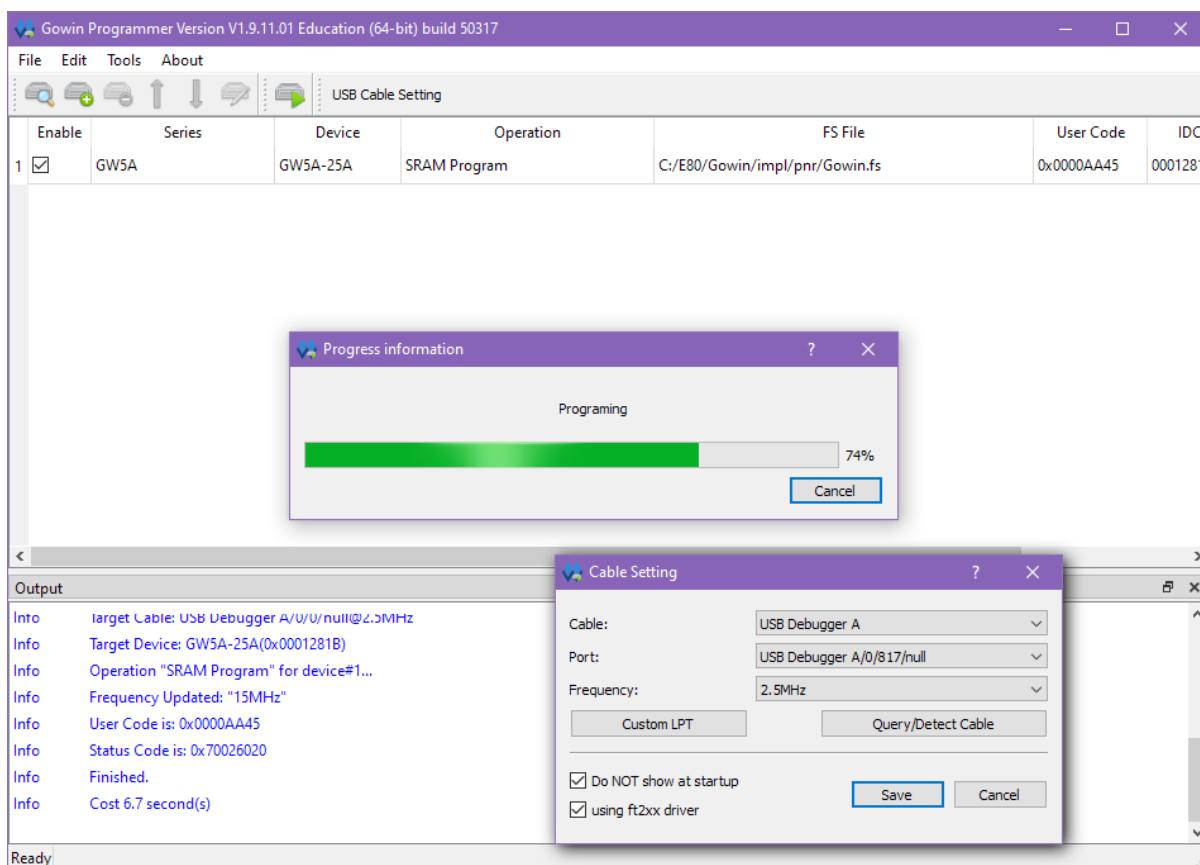
Πίνακας 13: Αναθέσεις σημάτων εξόδου στους ακροδέκτες της Tang Primer 25K

### 9.8.2. Σύνθεση και εκτέλεση

Εφόσον έχουν πραγματοποιηθεί οι απαραίτητες συνδέσεις και καλωδιώσεις, η σύνθεση του έργου στο Gowin και η εκτέλεσή του στην πλακέτα Tang Primer 25K είναι το ίδιο εύκολη με τα Quartus/DSD-i1. Θεωρώντας ότι έχουν αποσυμπίεστεί τα αρχεία σύμφωνα με τις οδηγίες που είχαν δοθεί στην εισαγωγή της Ενότητας 8 (σ. 95), αρκεί να ανοίξουμε το Gowin και μετά μέσω File > Open... να ανοίξουμε το αρχείο C:\E80\Gowin\Gowin.gprj.

Κάνουμε κλικ στο Run All και όταν ολοκληρωθεί η σύνθεση από την καρτέλα Process, πηγαίνουμε στο Tools > Programmer.

Έχοντας συνδέσει την πλακέτα με το παρεχόμενο καλώδιο USB στον υπολογιστή μας, μπορούμε να επιλέξουμε USB Cable Setting και Query/Detect Cable. Θα εντοπιστεί το καλώδιο USB Debugger A. Μπορούμε να αποθηκεύσουμε την ρύθμιση και να προχωρήσουμε στο ανέβασμα όπως φαίνεται στην Εικόνα 57 με το εικονίδιο Program Configure.



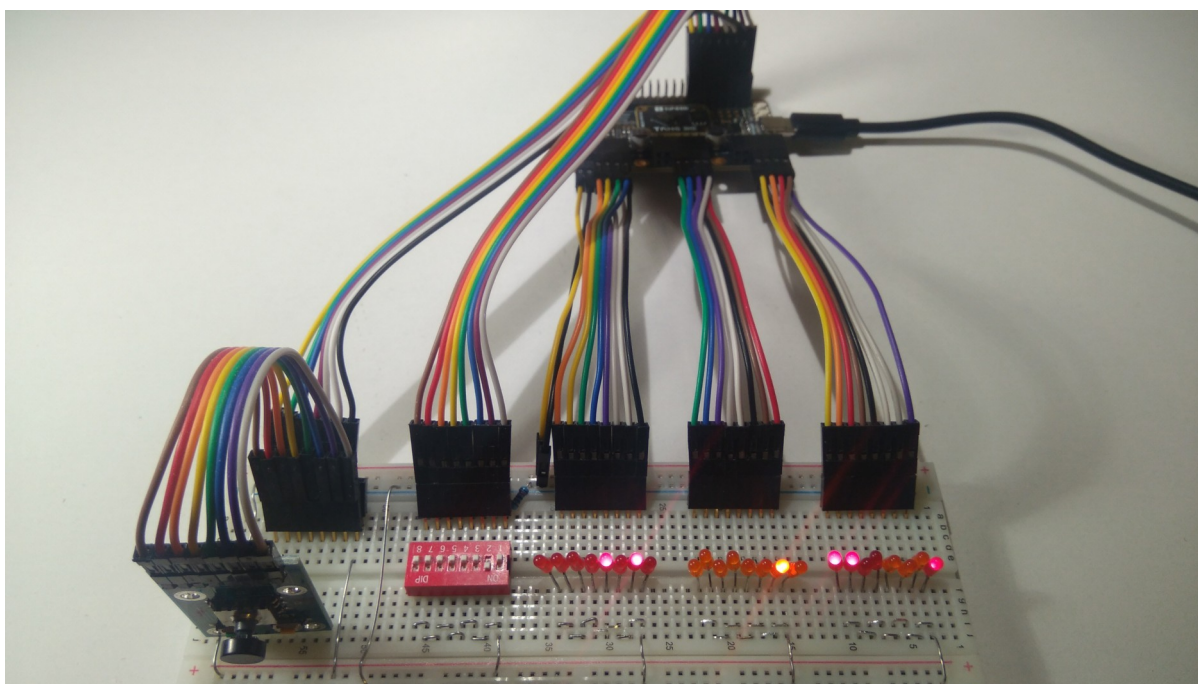
Εικόνα 57: Προγραμματισμός της Tang Primer 25K μέσω USB

Η πλακέτα έχει πλέον προγραμματιστεί και ξεκινά η λειτουργία της. Θέτουμε τους διακόπτες DIP στις θέσεις 00000010, όπως και στην DSD-i1, και κάνουμε Reset. Όπως και στην DSD-i1, μπορούμε να χρησιμοποιήσουμε το joystick για να μεταβάλλουμε την ταχύτητα ή το κουμπί SET για προσωρινή παύση του ρολογιού.

Το πρόγραμμα θα ολοκληρώσει την εκτέλεσή του και το LED του ρολογιού (τέρμα δεξιά) θα μείνει σταθερά φωτισμένο. Η φάση αυτή καταγράφεται στην Εικόνα 58.



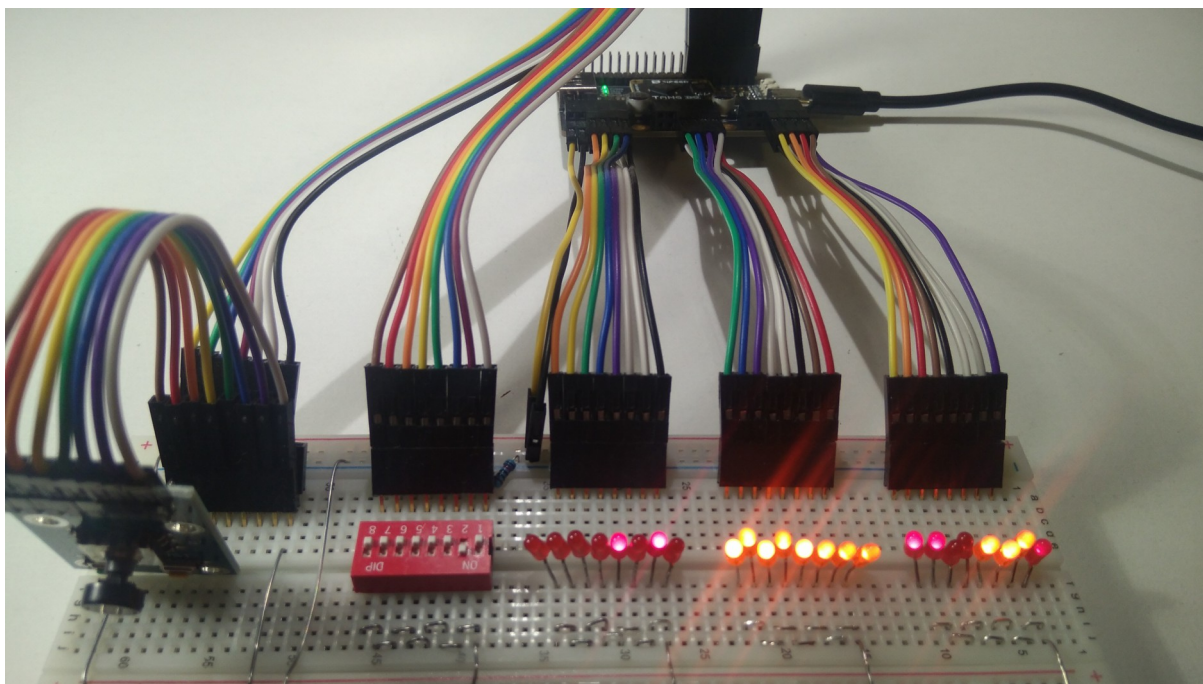
- Τα τέσσερα αριστερά LED στην δεξιά ομάδα καταγράφουν τις σημαίες, άρα  $C=1$ ,  $Z=1$ ,  $S=0$ ,  $V=0$ . Τα επόμενα τρία LED καταγράφουν τον επιλεγμένο καταχωρητή, επομένως τον R0.
- Στην μεσαία ομάδα καταγράφεται η τιμή του επιλεγμένου καταχωρητή, σε αντιστοιχία με την προσομοίωση και την εκτέλεση στην DSD-i1.
- Στην αριστερή ομάδα εμφανίζεται ο μετρητής προγράμματος, με τιμή 10 όπως και στην DSD-i1 και στην προσομοίωση στην Εικόνα 48 (σ. 108).



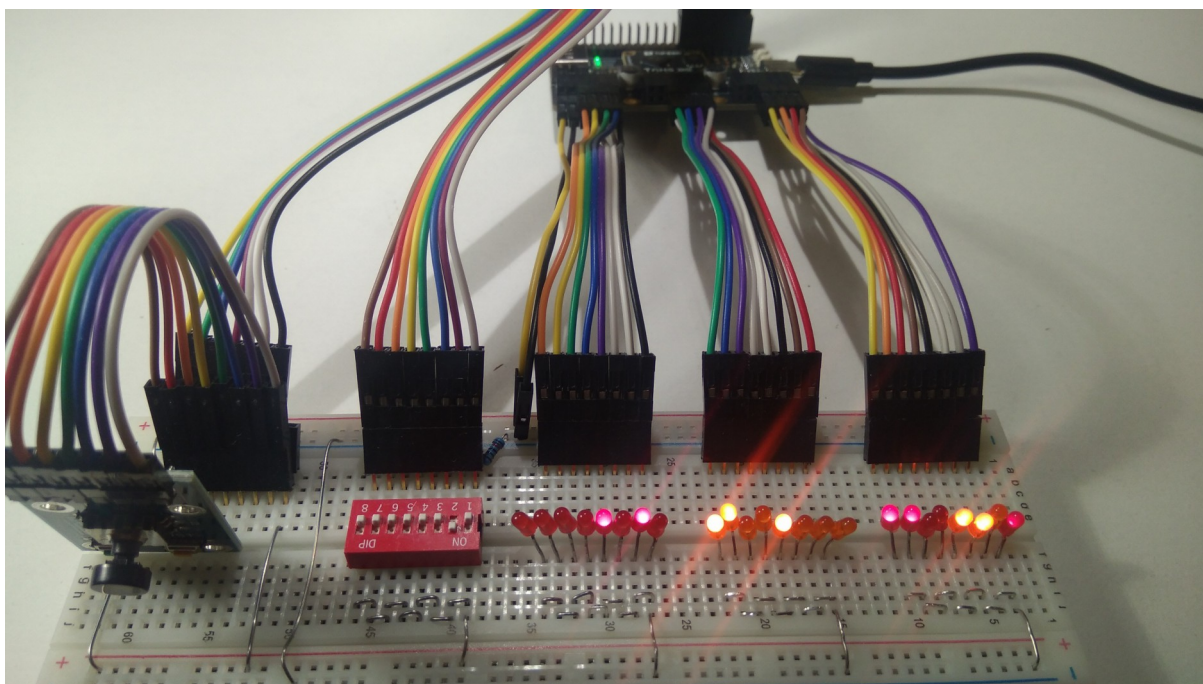
Εικόνα 58: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής R0

Πιέζοντας το joystick προς τα κάτω επιλέγεται ο καταχωρητής R7 (Stack Pointer) με τιμή 255 όπως φαίνεται στην Εικόνα 59. Με μια ακόμα πίεση προς τα κάτω, επιλέγεται ο καταχωρητής R6 (Flags) με τιμή 0b11001000 όπως φαίνεται στην Εικόνα 60.

Τα αποτελέσματα αυτά επικυρώνουν την λειτουργία της σύνθεσης σε διαφορετικό υλικό χωρίς μεταβολές στον κώδικα της VHDL.



Εικόνα 59: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής SP (R7)



Εικόνα 60: Ολοκλήρωση “256 ROR” στην Tang Primer 25K, καταχωρητής FLAGS (R6)



## 9.9. Αναφορές συνδυαστικών βρόχων

Κατα την διάρκεια της μεταγλώττισης το Quartus προβαίνει σε μεγάλο πλήθος προειδοποιήσεων για συνδυαστικούς βρόχους (combinational loop). Μέσω trial & error το πρόβλημα εντοπίστηκε στην ανάθεση της διεύθυνσης μνήμης για την ώθηση λέξης απο καταχωρητή μέσω των PUSH και CALL. Η ανάθεση αυτή γίνεται με την δήλωση

```
MemAddr <=
  B_val  WHEN isSTOREr OR isLOADr ELSE -- Instr2Reg2
  A_val  WHEN isPOP OR isRETURN  ELSE -- SP before increase
  A_next WHEN isPUSH OR isCALL    ELSE -- SP after decrease
  Instr2;                               -- STORE / LOAD direct
```

Ως προς τις PUSH/CALL η δήλωση αυτή εξασφαλίζει οτι η διεύθυνση μνήμης θα προσπελαστεί αφού πρώτα μειωθεί κατα 1. Στις δοκιμές που έγιναν προέκυψε το πρόβλημα λυνόταν χρησιμοποιώντας υπολογισμό απο την ieee.numeric\_std αντί της ALU. Η προσέγγιση αυτή θα απαιτούσε εξάρτηση απο την βιβλιοθήκη και για τον λόγο αυτό απορρίφθηκε καθώς θα παραβίαζε τον περιορισμό 5.1.1: “Ανάπτυξη χωρίς αριθμητικές βιβλιοθήκες” (σ. 52).

Απο την ανάλυση της διαδρομής των σημάτων προέκυψε οτι δεν υπάρχει συνδυαστικός βρόχος στον σχεδιασμό του E80 και το πρόβλημα που αναφέρει το Quartus οφείλεται στις εσωτερικές βελτιστοποιήσεις του.

Και αυτό γιατί το ίδιο ακριβώς πρόβλημα **λύθηκε** στο Gowin χρησιμοποιώντας την ιδιότητα syn\_keep που εμποδίζει τις βελτιστοποιήσεις αυτές (Gowin, [2025](#), σ. 47-83):

```
ATTRIBUTE syn_keep : BOOLEAN; -- FPGA fix, see MemAddr assignment
ATTRIBUTE syn_keep OF ALUinB : SIGNAL IS TRUE;
```

Το Quartus Prime Lite φαίνεται να μην υποστηρίζει<sup>1</sup> τέτοιες ιδιότητες και συνεπώς το πρόβλημα δεν μπορεί να λυθεί ικανοποιητικά.

Αυτό πάντως δεν επηρεάζει τίποτε άλλο στην υλοποίηση· είναι απλά ένας μεγάλος αριθμός απο ενοχλητικά προειδοποιητικά μηνύματα.

## 9.10. Μετατροπή σε VHDL-93 για συμβατότητα με οποιαδήποτε FPGA

Κατα την διάρκεια της εκπόνησης της εργασίας, δοκιμάστηκε μια παλαιά έκδοση του Xilinx που δεν υποστήριζε το πρότυπο VHDL 2008. Για τον λόγο αυτό χρησιμοποιήθηκε η δυνατότητα που παρέχει το GHDL για σύνθεση όλου του έργου σε ένα ενιαίο αρχείο μορφής VHDL-93 το οποίο αναγνωρίστηκε κανονικά απο το Xilinx.

Αν και η διερεύνηση της συμβατότητας με το Xilinx δεν ενσωματώθηκε στην εργασία λόγω προτίμησης της Tang Primer 25K, η δυνατότητα του GHDL για μετατροπή του έργου σε πιο συμβατή μορφή μπορεί να είναι χρήσιμη για εκτέλεση σε διαφορετικό υλικό. Το σχετικό αρχείο δεσμίδας διατίθεται στο Παράρτημα Δ.4 (σ. 175) και εκτελείται μέσω της εντολής:

```
FPGA synthesis.bat
```

Το αρχείο “FPGA.GHDL.vhdl” που εξάγεται δεν είναι σε μορφή κατάλληλη για ανάγνωση απο άνθρωπο, αλλά διαβάζεται και μετατρέπεται κανονικά απο μεταγλωττιστές.

<sup>1</sup> <https://community.intel.com/t5/Intel-Quartus-Prime-Software/Quartus-Prime-Lite-keep-hierarchy/td-p/1308469>

## 10. Μια νέα CPU για εκπαιδευτικούς σκοπούς

Έχοντας παρουσιάσει το πλαίσιο του οικοδομισμού στην Ενότητα 2.7 (σ. 16) και την τεχνική υλοποίηση του E80 έως το στάδιο της προσομοίωσης, μπορεί να γίνει πλέον η σύνδεση της θεωρίας με την πράξη μέσω της μελέτης μιας περίπτωσης που καταδεικνύει πώς το οικοσύστημα του E80 λειτουργεί ως ένας μικρόκοσμος για μάθηση και ανακάλυψη.

### 10.1. Η ιδέα για ένα αλγόριθμο διαίρεσης

Μετά την ολοκλήρωση του σχεδιασμού του E80 και των βασικών εργαλείων του (assembler, προσομοιωτής), η επαλήθευση της ορθής λειτουργίας του απαιτούσε κάτι περισσότερο από απλές εντολές που παρουσιάστηκαν στο προηγούμενο κεφάλαιο. Για να ελεγχθεί η πληρότητα της αρχιτεκτονικής των εντολών, κρίθηκε απαραίτητη η ανάπτυξη και εκτέλεση ενός μη τετριμμένου αλγορίθμου.

Ο πολλαπλασιασμός και η διαίρεση αποτελεί ένα παράδειγμα μιας τέτοιας διαδικασίας. Σε αντίθεση με την προσθαφαίρεση, ο πολλαπλασιασμός και η διαίρεση δεν παρουσιάζεται στην ύλη της στοιχειώδους ψηφιακής σχεδίασης και ούτε υποστηρίζεται στο βασικό σύνολο εντολών του RISC-V (Waterman, 2006, σ. 31) για τον ίδιο πρακτικό σκοπό που έχει εφαρμοστεί στον E80: απλοποίηση της υλοποίησης για περιπτώσεις όπου οι πράξεις αυτές γίνονται σπάνια. Αλλά ενώ στην περίπτωση του RISC-V οι πράξεις αυτές υλοποιούνται αποεπεκτάσεις, στην περίπτωση του E80 θα εκτελεστούν αλγοριθμικά.

Αντί για την απλή αντιγραφή ενός έτοιμου αλγορίθμου από τη βιβλιογραφία, επιλέχθηκε η οδός της ανάπτυξης ενός νέου αλγορίθμου, **χωρίς διερεύνηση αλγορίθμων βελτιστοποιημένης διαίρεσης**, για να δοκιμαστεί στην πράξη ο μικρόκοσμος του E80.

### 10.2. Απο την χειρόγραφη ιδέα στον αλγόριθμο

Ως χαμηλό κατώφλι χρησιμοποιήθηκε ο τετριμμένος αλγόριθμος των επαναληπτικών αφαιρέσεων του διαιρέτη από τον διαιρετέο:

-----  
Αλγόριθμος: διαίρεση με επαναληπτικές αφαιρέσεις  
-----

Είσοδος: r1 (διαιρετέος), r2 (διαιρέτης)

```
1.  r0 ← 0
2.  Επανάλαβε ώσπου r1 < r2 {
3.      r1 -= r2
4.      r0++
5.  }
```

Εξοδος: r0 (πηλίκo), r1 (υπόλοιπο)  
-----

Ο αλγόριθμος αυτός έχει πολυπλοκότητα  $\Theta(r1/r2)$ , καθώς απαιτεί μια (αργή) αφαίρεση για κάθε μονάδα του πηλίκου. Το ερώτημα που τέθηκε ήταν αν θα μπορούσε να εφαρμοστεί η προσέγγιση του πολλαπλασιασμού αλα ρωσικά, και γενικά η θεωρία της αλγοριθμικής πολυπλοκότητας (ΠΛΗ30) στην διαίρεση.

Μετά απο κάποιες δοκιμές γράφτηκε το χειρόγραφο για την διαίρεση 179 προς 13 που φαίνεται στην Εικόνα 61. Στο χειρόγραφο οι R1 και R2 εκφράζουν διαιρετέο και διαιρέτη και στο τέλος των υπολογισμών ο R0 περιέχει το πηλίκo ενώ ο R1 το υπόλοιπο.

Στην αρχική φάση της “ανάβασης” να διπλασιάζεται ο διαιρέτης R2 και ταυτόχρονα να υπολογίζεται στον R3 η δύναμη στην οποία έχει υψωθεί ο R2. Η επανάληψη αυτή θα τερματίσει όταν ο R2 γίνει μεγαλύτερος ή ίσος του R1. Αυτό φαίνεται στο χειρόγραφο από τις διαδοχικές τιμές του R2 από 13 ως 208 και τις αντίστοιχες τιμές του R3 από 1 ως 16.

Στο σημείο αυτό δημιουργήθηκε η εξής ιδέα: Η τρέχουσα τιμή του R3 εκφράζει πόσα “δεκατριάρια” αντιστοιχούν στην τρέχουσα τιμή του R2. Άρα αν ο R2 χωράει στον R1, τότε τον αφαιρούμε από τον R1 και προσθέτουμε στο πηλίκο το πλήθος των “δεκατριών” που αφαιρέθηκαν. Άρα θα γίνει μια αφαίρεση μόνο για κάθε bit του πηλίκου (στην χειρότερη περίπτωση όπου το πηλίκο είναι πχ. 1111), μειώνοντας δραστικά την πολυπλοκότητα στο  $\Theta(\log(r1/r2))$ . Από την ιδέα αυτή προκύπτει η αντίστροφη διαδικασία της “κατάβασης”:

- Αν ο R2 χωράει στον διαιρετέο R1, τότε:
  - Θα αφαιρείται ο R2 από τον R1
  - Το πλήθος που εκφράζει ο R3 θα προστίθεται στο πηλίκο R0
- Θα υποδιπλασιάζονται οι R2 και R3 ώσπου ο R3 να γίνει 0.

R0	R1	R2	R3
0	179	13	1
8	75	26	2
12	23	52	4
13	10	104	8
		208	16
		104	8
		52	4
		26	2
		13	1

Εικόνα 61: Το αρχικό χειρόγραφο της ιδέας του αλγορίθμου “ανέβα-κατέβα”

Επομένως, με τις τρέχουσες τιμές R0=0, R1=179, R2=208, R3=16 θα ισχύουν τα εξής:

- ο R2 (208) δεν χωράει στον R1 (179)  
υποδιπλασιάζεται ο R2 σε 104 και ο R3 σε 8
- ο R2 (104) χωράει στον R1, οπότε  $R1 = R1 - R2 = 179 - 104 = 75$ , και  $R0 = R0 + R3 = 0 + 8 = 8$   
υποδιπλασιάζεται ο R2 σε 52 και ο R3 σε 4
- ο R2 (52) χωράει στον R1, οπότε  $R1 = R1 - R2 = 75 - 52 = 23$ , και  $R0 = R0 + R3 = 8 + 4 = 12$   
υποδιπλασιάζεται ο R2 σε 26 και ο R3 σε 2
- ο R2 (26) δεν χωράει στον R1 (23)  
υποδιπλασιάζεται ο R2 σε 13 και ο R3 σε 1
- ο R2 (13) χωράει στον R1, οπότε  $R1 = R1 - R2 = 23 - 13 = 10$ , και  $R0 = R0 + R3 = 12 + 1 = 13$   
υποδιπλασιάζεται ο R2 σε αδιάφορο και ο R3 σε 0

Η επανάληψη τερματίζεται αφού R3 = 0. Πηλίκο = R0 = 13 και υπόλοιπο = R1 = 10.

Αρχικά η ιδέα γράφτηκε ως αλγόριθμος σε ψευδοκώδικα:

```
-----
Αλγόριθμος: διαίρεση ανέβα-κατέβα
-----
Είσοδος: r1 (διαιρετέος), r2 (διαιρέτης)
1.  r0 ← 0
2.  r3 ← 1
3.  Επανάλαβε ώσπου r2 ≥ r1 {
4.      Διπλασίασε r2
5.      Διπλασίασε r3
6.  }
7.  Επανάλαβε ώσπου r3 = 0 {
8.      Αν r2 ≤ r1 {
9.          r1 -= r2
10.         r0 += r3
11.     }
12.     Υποδιπλασίασε r2
13.     Υποδιπλασίασε r3
14. }
Εξοδος: r0 (πηλίκο), r1 (υπόλοιπο)
-----
```

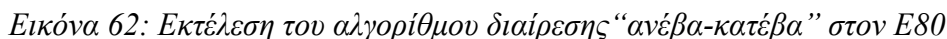
Έπειτα μετατράπηκε “κατα γράμμα” σε συμβολική γλώσσα του E80:

```
.NAME dividend 179
.NAME divisor 13

    MOV R1, dividend
    MOV R2, divisor
    MOV R0, 0
    MOV R3, 1
ascend:
    CMP R2, R1
    JC descend    ; ώσπου r2 ≥ r1
    LSHIFT R2
    LSHIFT R3
    JMP ascend    ; επανάλαβε
descend:
    CMP R3, 0
    JZ fin        ; ώσπου r3 = 0
    CMP R1, R2
    JNC halve     ; αν r2 > r1, παράκαμψε τις προσθαφαιρέσεις
    SUB R1, R2
    ADD R0, R3
halve:
    RSHIFT R2
    RSHIFT R3
    JMP descend
fin:
    HLT
```

Και τέλος εκτελέστηκε σύμφωνα με την διαδικασία που περιγράφεται στην Ενότητα 8.3 (σ. 102).

Απο το GHDL/GTKWave παρήχθησαν οι κυματομορφές που καταγράφονται στα δυο τμήματα της Εικόνας 62.



### 10.3. Βελτιστοποιημένη υλοποίηση συμβολικού προγράμματος

Δεύτερον, η πρόσθεση του R3 στο R0 μπορεί να βελτιωθεί, δεδομένου ότι το μοναδικό bit του R3 ολισθαίνει συνεχώς, άρα αντιστοιχεί πάντα με 0 στον R0. Επομένως η πρόσθεση μπορεί να αντικατασταθεί με λογική διάζευξη η οποία, σε επεξεργαστές πολλών κύκλων, είναι ταχύτερη. Αυτό καταδεικνύει το μειονέκτημα του σχεδιασμού εντολών ενός κύκλου: μια τέτοια βελτιστοποίηση δεν έχει πρακτικό αποτέλεσμα.

Απο τα παραπάνω προκύπτει η βελτιστοποιημένη υλοποίηση του αλγορίθμου σε συμβολική γλώσσα του E80, με τις εξής αλλαγές και προσθήκες:

- Γίνεται πρόβλεψη υπερχειρίσης ολίσθησης με το έλεγχο του ΠΣΨ μέσω της BIT R2,0b10000000.
- Αντικαθίσταται η πρόσθεσης ADD R0, R3 με την λογική διάζευξη OR R0, R3.
- Ο έλεγχος μηδενικού R3 μετακινείται μετά απο την δεξιά ολίσθησή του, χωρίς CMP.

```
.NAME dividend 179
.NAME divisor 13

MOV R1, dividend      ; διαιρετέος / υπόλοιπο
MOV R2, divisor
MOV R0, 0              ; πηλίκο
MOV R3, 1              ; κλιμακούμενο πηλίκο
ascend:
  CMP R2, R1
  JC descend           ; ανέβασμα τέλος όταν R2 ≥ R1 (φτιάσαμε στην κορυφή)
  BIT R2, 0b10000000
  JNZ descend          ; ή όταν R2 ≥ 128 αφού μέγιστος διαιρέτης σε 8bit = 127
  LSHIFT R2
  LSHIFT R3
  JMP ascend
descend:
  CMP R1, R2
  JNC halve            ; αν R2 > R1, παράκαμψε τις προσθαφαιρέσεις
  SUB R1, R2
  OR R0, R3             ; R0 += R3, δεδομένου ότι το R3 ολισθαίνει συνεχώς
halve:
  RSHIFT R2
  RSHIFT R3
  JZ fin               ; ώπου R3 = 0
  JMP descend
fin:
  HLT                  ; R0 = πηλίκο, R1 = υπόλοιπο
```

Μετατρέποντας το πρόγραμμα σε γλώσσα μηχανής και προσομοιώνοντας το με το GHDL/GTKWave μέσω του computer\_tb.bat, παίρνουμε τα αποτελέσματα που εμφανίζονται στην Εικόνα 63 με τα οποία επιβεβαιώνεται η σωστή λειτουργία του στην περίπτωση χρήσης.



Εικόνα 63: Βελτιωμένος αλγόριθμος διαίρεσης “ανέβα-κατέβα”



## 10.4. Συνδυασμός με πολλαπλασιασμό a la russe και υπορουτίνες

Για την ολοκλήρωση της δυνατότητας διαίρεσης και πολλαπλασιασμού με αποτελεσματικούς αλγορίθμους, θα χρησιμοποιηθεί ο πολλαπλασιασμός a la russe που βασίζεται στην απλή λογική:

Αν R2 άρτιος, τότε  $R1 \times R2 = (R1 \times 2) \times (R2 \div 2)$   
Αν R2 περιττός, τότε  $R1 \times R2 = (R1 \times 2) \times (R2 \div 2) + R1$

και επομένως διπλασιάζει τον R1 και υποδιπλασιάζει τον R2 προσθέτοντας την τιμή του R1 στο γινόμενο όποτε ο R2 είναι περιττός. Ο πολλαπλασιασμός a la russe χρησιμοποιείται σήμερα στο hardware των υπολογιστών (Φωτάκης & Σπυράκης, [2001](#), σ. 19) και είναι αρκετά γνωστός οπότε δεν χρειάζεται να αναλυθεί η μορφή του σε assembly.

Για να συνδυαστούν οι δυο αλγόριθμοι μπορούν να κατασκευαστούν υπορουτίνες ως εξής:

```
CALL multiply      ; R0 = 7*29 = 203
PUSH R0
CALL division     ; R0 = 179 div 13 = 14, R1 = 179 mod 13 = 11
POP R2
```

Η υπορουτίνα multiply εξάγει το αποτέλεσμα του γινομένου στον καταχωρητή R0 ενώ αυτός χρησιμοποιείται εντός της υπορουτίνας division. Όμως η τιμή του R0 ωθείται στην στοίβα, και επομένως μετά την ολοκλήρωση της division, η τιμή απωθείται στον R2. Άρα στο τέλος του προγράμματος, οι R0 και R1 θα περιέχουν πηλίκο και υπόλοιπο, όπως παρουσιάστηκε προηγουμένως, ενώ ο R2 θα περιέχει το γινόμενο.

```
; This program is for testing and showcasing various features of E80
; It calculates 179 div 13 = 14 into R0,
;               179 mod 13 = 11 into R1,
;               7 mul 29 = 203 into R2.

.TITLE "Multiplication and division with subroutines (divmul.asm)"
.DATA 100 7,29          ; multiplicand = [100] = 7, multiplier = [101] = 29
.NAME dividend 179
.NAME divisor 12

    CALL multiply      ; R0 = 7*29 = 203
    PUSH R0
    CALL division     ; R0 = 179 div 13 = 14, R1 = 179 mod 13 = 11
    POP R2
    HLT

division:                ; "Ascend-descend" division algorithm by Panos Stokas
    MOV R1, dividend    ; remainder (starts as dividend)
    MOV R2, divisor     ; scaled divisor, goes up to dividend and back down
    MOV R0, 0           ; quotient
    MOV R3, 1           ; quotient bit, R3 * original divisor = R2
ascend:                  ; double divisor until ≥ dividend or 128
    CMP R2, R1
    JC descend          ; if R2 ≥ R1
    BIT R2, 0b10000000
    JNZ descend        ; no divisor ≥ 128 for 8 bits (also prevent overflow)
    LSHIFT R2           ; scale divisor
    LSHIFT R3           ; move quotient bit to the left
    JMP ascend

descend:                 ; subtract and halve divisor until we're back
    CMP R1, R2
    JNC halve_divisor   ; if R2 > R1
    SUB R1, R2          ; remainder -= scaled divisor
    OR R0, R3           ; quotient += quotient bit
halve_divisor:
    RSHIFT R2           ; halve scaled divisor
```

```

    RSHIFT R3                ; move quotient bit to the right
    JZ division_done         ; stop when the quotient bit is dropped
    JMP descend
division_done:
    RETURN

multiply:
    ; Russian Peasant multiplication algorithm
    ; multiplicand at memory address 100
    LOAD R1, [100]
    ; multiplier at memory address 101
    LOAD R2, [101]
    ; accumulated product
    MOV R0, 0
multiply_loop:
    BIT R2, 0xFF
    JZ multiply_done         ; stop when multiplier = 0
    BIT R2, 1
    JZ skip_add              ; even R2 ⇒ R1×R2 = (R1×2)×(R2 div 2) ⇒ skip add R1
    ADD R0, R1               ; odd R2 ⇒ R1×R2 = (R1×2)×(R2 div 2)+R1 ⇒ add R1
skip_add:
    LSHIFT R1                ; R1×2
    RSHIFT R2                ; R2 div 2
    JMP multiply_loop
multiply_done:
    RETURN

```

## 10.5. Μεταγλώττιση, προσομοίωση και υλοποίηση

Το πρόγραμμα μεταγλωττίζεται από τον Assembler στον παρακάτω κώδικα μηχανής:

```

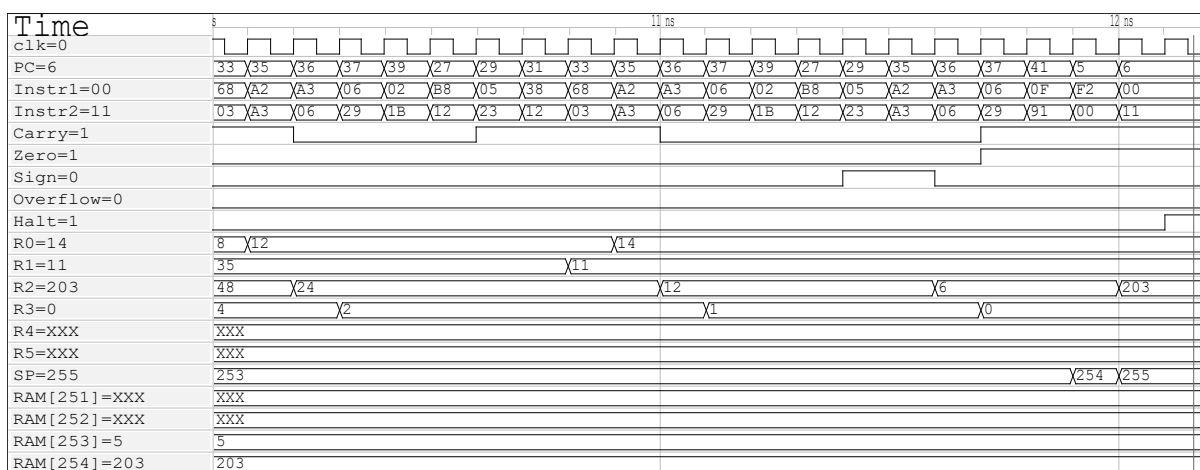
-----
-- Multiplication and division with subroutines (divmul.asm)
-----

LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT SimDIP : WORD := "00000000"; -- DIP input for testbench only
CONSTANT Firmware : WORDx256 := (
0  => "00001110", 1  => "00101010", -- CALL 42
2  => "11100000", -- PUSH R0
3  => "00001110", 4  => "00000111", -- CALL 7
5  => "11110010", -- POP R2
6  => "00000000", -- HLT
7  => "00010001", 8  => "10110011", -- MOV R1, 179
9  => "00010010", 10 => "00001100", -- MOV R2, 12
11 => "00010000", 12 => "00000000", -- MOV R0, 0
13 => "00010011", 14 => "00000001", -- MOV R3, 1
15 => "10111000", 16 => "00100001", -- CMP R2, R1
17 => "00000100", 18 => "00011011", -- JC 27
19 => "11010010", 20 => "10000000", -- BIT R2, 128
21 => "00000111", 22 => "00011011", -- JNZ 27
23 => "11000010", -- LSHIFT R2
24 => "11000011", -- LSHIFT R3
25 => "00000010", 26 => "00001111", -- JMP 15
27 => "10111000", 28 => "00010010", -- CMP R1, R2
29 => "00000101", 30 => "00100011", -- JNC 35
31 => "00111000", 32 => "00010010", -- SUB R1, R2
33 => "01101000", 34 => "00000011", -- OR R0, R3
35 => "10100010", -- RSHIFT R2
36 => "10100011", -- RSHIFT R3
37 => "00000110", 38 => "00101001", -- JZ 41
39 => "00000010", 40 => "00011011", -- JMP 27
41 => "00001111", -- RETURN
42 => "10010001", 43 => "01100100", -- LOAD R1, [100]
44 => "10010010", 45 => "01100101", -- LOAD R2, [101]
46 => "00010000", 47 => "00000000", -- MOV R0, 0
48 => "11010010", 49 => "11111111", -- BIT R2, 255
50 => "00000110", 51 => "00111110", -- JZ 62
52 => "11010010", 53 => "00000001", -- BIT R2, 1
54 => "00000110", 55 => "00111010", -- JZ 58
56 => "00101000", 57 => "00000001", -- ADD R0, R1

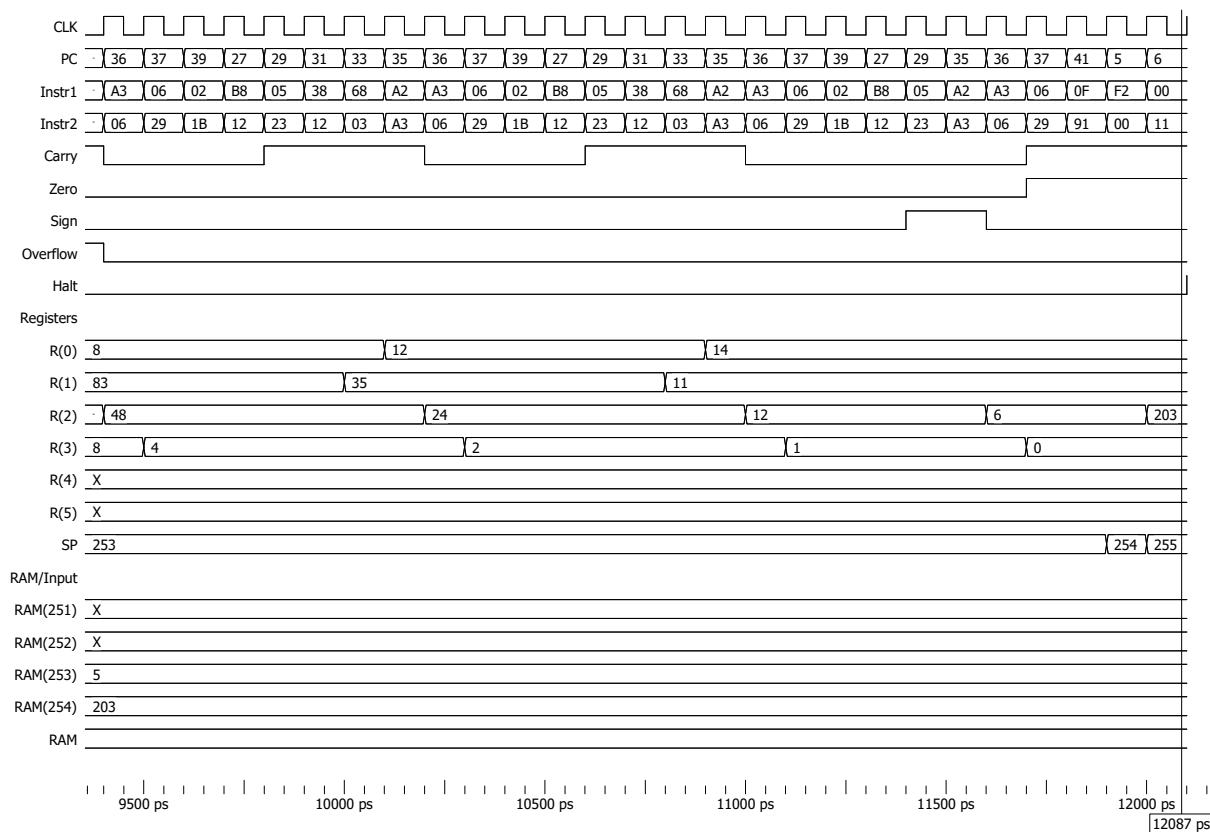
```

```
58 => "11000001", -- LSHIFT R1
59 => "10100010", -- RSHIFT R2
60 => "00000010", 61 => "00110000", -- JMP 48
62 => "00001111", -- RETURN
100 => "00000111", -- 7
101 => "00011101", -- 29
OTHERS => "UUUUUUUU");END;
```

Προσομοιώνεται σε GHDL/GTKWave με απλή εκτέλεσης του αρχείου C:\E80\GHDL\computer\_tb.bat και τα αποτελέσματα καταγράφονται στην Εικόνα 64. Επίσης προσομοιώνεται σε ModelSim γράφοντας την εντολή do c.do στο παράθυρο Transcript. Τα αποτελέσματα καταγράφονται στην Εικόνα 65. Και στις δυο περιπτώσεις επιβεβαιώνεται η ορθή λειτουργία του προγράμματος και του E80.



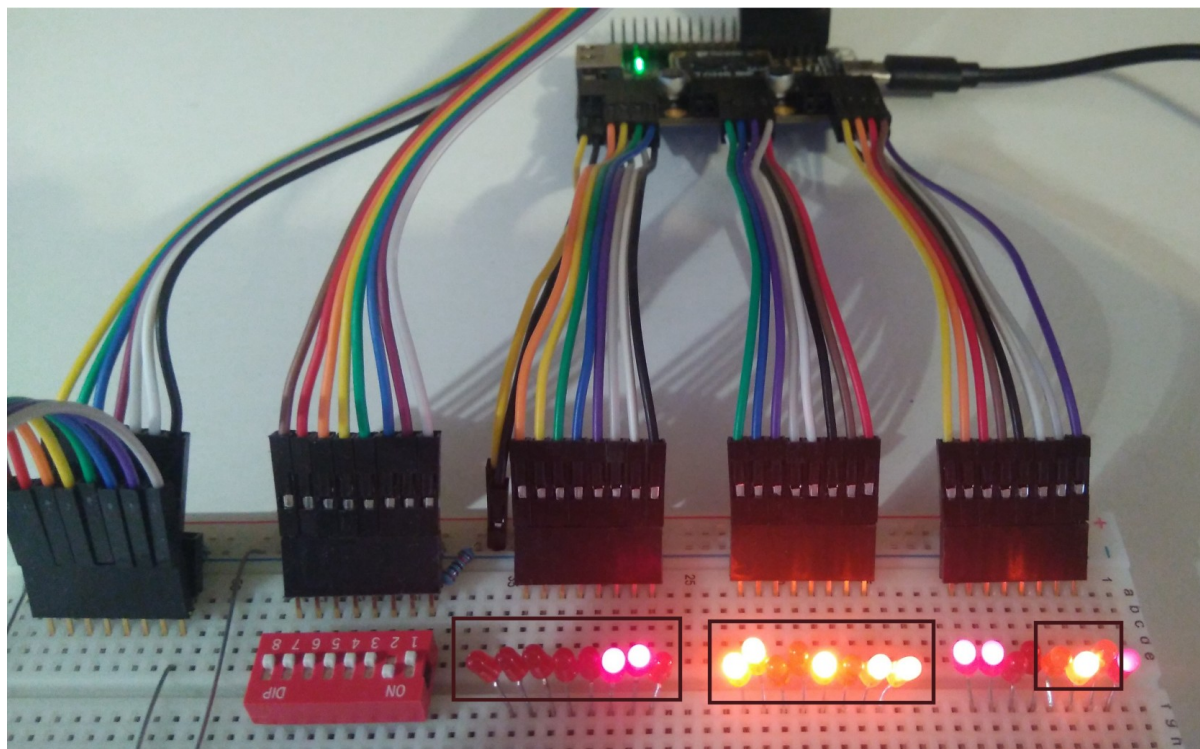
Εικόνα 64: Προσομοίωση εκτέλεσης divmul.asm στον E80 (GHDL/GTKWave)



Εικόνα 65: Προσομοίωση εκτέλεσης divmul.asm στον E80 (ModelSim)

Ολοκληρώνοντας την υλοποίηση του προγράμματος σε πραγματικό υλικό, ανοίγουμε το Gowin ή το Quartus και προχωράμε σε σύνθεση του FPGA.vhd. Ανεβάζουμε την μεταγλωττισμένη διαμόρφωση του υλικού στην πλακέτα, και φορτώνουμε το πρόγραμμα πιέζοντας Reset. Αυξάνουμε αν θέλουμε την συχνότητα με το joystick ώσπου να γίνει Αλτ και να παραμείνει σταθερό το LED του ρολογιού. Τα αποτελέσματα εξάγονται στους R0, R1, R2.

Στην Εικόνα 66 το πρόγραμμα έχει εκτελεστεί στον Υπολογιστή E80, στην πλατφόρμα Tang Primer 25K. Έχει επιλεγθεί ο καταχωρητής R2 όπως φαίνεται απο την ομάδα των 3 LED δεξιά, και η τιμή του είναι 11001011 (203) όπως φαίνεται απο την μεσαία ομάδα των 8 LED.



Εικόνα 66: Αποτέλεσμα εκτέλεσης *divmul.asm* στον E80 (Tang Primer 25K)

Παρατηρούμε επίσης οτι ο μετρητής προγράμματος (αριστερά) έχει τιμή 6, που αντιστοιχεί στην θέση της HLT στο πρόγραμμα. Η ρύθμιση των αντίστοιχων διανυσμάτων σε δυαδική μορφή στο GHDL/ModelSim μπορεί διευκολύνει την επαλήθευση της εκτέλεσης.

## 10.6. Ο εκπαιδευτικός μικρόκοσμος του E80

Η διαδικασία ανάπτυξης και βελτιστοποίησης του αλγορίθμου διαίρεσης “ανέβα-κατέβα” λειτουργεί ως μελέτη περίπτωσης για το πώς το ολοκληρωμένο οικοσύστημα του E80 (αρχιτεκτονική, συμβολική γλώσσα, προσομοίωση, υλοποίηση) μπορεί να λειτουργήσει ως ένας μικρόκοσμος μάθησης, σύμφωνα με τις αρχές του Πάπερτ. Εντός ενός τέτοιου πλαισίου, ο εκπαιδευόμενος ενθαρρύνεται να μεταβεί από την παθητική πρόσληψη γνώσης στην ενεργό οικοδόμησή της, μέσω πειραματισμού, εντοπισμού σφαλμάτων και δημιουργίας.

Ο σχεδιασμός του συστήματος E80 επιχειρεί να ευθυγραμμιστεί με τις αρχικές σχεδιασμού ενός μικρόκοσμου, όπως καθορίστηκαν στο Υπόβαθρο (Ενότητα 2.7.1, σ. 17):

- *Χαμηλό κατώφλι:* Η αρχική αλληλεπίδραση με το περιβάλλον του E80 έχει σχεδιαστεί με γνώμονα την προσβασιμότητα. Ο εκπαιδευόμενος μπορεί να αρχίσει με θεμε-

λιώδεις συμβολικές εντολές, να παρακολουθήσει τη βηματική εκτέλεσή τους στον προσομοιωτή και να παρατηρήσει την επίδραση του κώδικά του στην κατάσταση των καταχωρητών και της μνήμης όπως μεταβάλλονται σε ένα κύκλο του ρολογιού για κάθε εντολή. Αυτός ο μηχανισμός άμεσης ανατροφοδότησης μπορεί να λειτουργήσει ως κίνητρο για περαιτέρω ενασχόληση.

- *Υψηλό ταβάνι*: Το περιβάλλον δεν περιορίζεται σε στοιχειώδεις ασκήσεις. Η δυνατότητα υλοποίησης σύνθετων αλγορίθμων, σε συνδυασμό με την υποστήριξη υπορουτινών και στοίβας, παρέχει το πλαίσιο για την ενασχόληση με πιο απαιτητικά έργα και την εξερεύνηση των αρχών του δομημένου προγραμματισμού. Επιπλέον, η υλοποίηση του επεξεργαστή σε πραγματικό υλικό (FPGA) καταδεικνύει τη σύνδεση μεταξύ του αφηρημένου κώδικα και της φυσικής του εκτέλεσης, διευρύνοντας το πεδίο των πιθανών διερευνήσεων.
- *Ευρείς τοίχοι*: Το περιβάλλον είναι σχεδιασμένο ώστε να υποστηρίζει ένα φάσμα έργων, επιτρέποντας στον χρήστη να εστιάσει, μεταξύ άλλων, στη συγγραφή λογισμικού, στη μελέτη της αρχιτεκτονικής, στην κατανόηση της μεταγλώττισης ή στη σχέση υλικού-λογισμικού. Η εκπαιδευτική του αξία ενισχύεται από το γεγονός ότι ο E80 έχει κατασκευαστεί με πλήρη διαφάνεια, χωρίς “μαύρα κουτιά”, καθιστώντας τον ίδιο τον επεξεργαστή ως προσβάσιμο αντικείμενο μελέτης.

Η ενασχόληση με την ανάπτυξη του αλγορίθμου διαίρεσης προσέφερε την ευκαιρία για επαφή με μία “ισχυρή ιδέα”. Στη συγκεκριμένη περίπτωση, αυτή η ιδέα ήταν η αλγοριθμική αποσύνθεση και η βελτιστοποίηση μέσω δυαδικών χειρισμών.

Η αρχική προσέγγιση, που βασίζεται σε επαναληπτικές αφαιρέσεις, είναι απλή στη σύλληψή της, αλλά αναποτελεσματική. Η αναζήτηση μιας βελτιωμένης λύσης ωθεί στη θεώρηση της δομής των αριθμών στο δυαδικό σύστημα και τελικά στην αναγωγή των αριθμητικών πράξεων σε λογικές πύλες.

Παράλληλα, η χρήση υπορουτινών για τον πολλαπλασιασμό και τη διαίρεση εισάγει μια δεύτερη, εξίσου ισχυρή ιδέα: αυτή του δομημένου προγραμματισμού. Με την χρήση των CALL και RETURN ο εκπαιδευόμενος μπορεί να διασπάσει ένα σύνθετο πρόβλημα σε ανεξάρτητες και επαναχρησιμοποιήσιμες ενότητες, παρατηρώντας ταυτόχρονα τον τρόπο με τον οποίο λειτουργεί η κλήση τους μέσω του δείκτη στοίβας και των αποθηκευμένων διευθύνσεων στις υψηλότερες θέσεις μνήμης.

Η αντικατάσταση της εντολής ADD από την εντολή OR για την ενημέρωση του πηλίκου αποτελεί ένα παράδειγμα Παπερτιανής ισχυρής ιδέας: η συνειδητοποίηση ότι οι αριθμητικές πράξεις μπορούν, υπό συνθήκες, να αντιστοιχιστούν σε απλούστερες και ταχύτερες λογικές πράξεις, αποτελεί έναν ακρογωνιαίο λίθο της υπολογιστικής σκέψης και του σχεδιασμού αποδοτικών συστημάτων.



## 11. Συμπεράσματα

Σκοπός της παρούσας εργασίας ήταν ο σχεδιασμός και η υλοποίηση του E80, ενός απλού, οκτάμπιτου επεξεργαστή, με στόχο τη δημιουργία ενός ολοκληρωμένου και προσβάσιμου εκπαιδευτικού οικοσυστήματος. Το εγχείρημα δεν περιορίστηκε μόνο στη σχεδίαση του υλικού σε VHDL, αλλά επεκτάθηκε στην ανάπτυξη ενός πλήρους συνόλου εργαλείων, συμπεριλαμβανομένου ενός συμβολομεταφραστή σε C και σεναρίων για προσομοίωση και υλοποίηση σε πλατφόρμες FPGA.

Η μεθοδολογία που ακολουθήθηκε επέτρεψε την επίτευξη των τεσσάρων θεμελιωδών στόχων που τέθηκαν στην Εισαγωγή:

**Χρησιμότητα:** Το τελικό σύστημα του E80, επιχειρεί να προσφέρει χαρακτηριστικά ενός Μικρόκοσμου επιτρέποντας προγραμματισμό σε μια απλή αρχιτεκτονική που σχεδιάστηκε για να προσφέρει ένα πλήρες σετ εντολών για εκπαιδευτικούς σκοπούς. Τα εκτενή μηνύματα επεξήγησης σφαλμάτων, και η εξαγωγή μορφοποιημένου κώδικα μηχανής απο τον συμβολομεταφραστή, καθώς και η προετοιμασία των προσομοιωτών για εύκολη παρακολούθηση της εκτέλεσης με ένα κύκλο ανα εντολή πλαισιώνουν τον στόχο αυτό.

**Συνέχεια:** Ο σχεδιασμός του υλικού έγινε με σχεδόν αποκλειστική χρήση των αρχών της στοιχειώδους ψηφιακής σχεδίασης. Η αποφυγή έτοιμων αριθμητικών βιβλιοθηκών και η περιορισμένη χρήση της δήλωσης PROCESS, όπως αναλύθηκε στην Ενότητα 5.1, καθιστούν τον κώδικα του E80 προσβάσιμο και κατανοητό σε κάποιον που έχει ολοκληρώσει τις βασικές ενότητες της ψηφιακής σχεδίασης και σκοπεύει να συνεχίσει με την αρχιτεκτονική υπολογιστών.

**Εφαρμογή:** Για την ανάπτυξη του συμβολομεταφραστή αξιοποιήθηκαν οι θεωρητικές αρχές των μεταγλωττιστών, των ασυμφραστικών γραμματικών, τεχνικών προγραμματισμού C και η εκτίμηση της πολυπλοκότητας. Το έργο υλοποιήθηκε μέσω συστήματος ελέγχου εκδόσεων με την παρακολούθηση του επιβλέποντα, εφαρμόζοντας στην πράξη τις αρχές τεχνολογίας λογισμικού. Εφαρμόστηκε επομένως μια ευρεία γκάμα γνώσεων που αποκτήθηκαν στις ενότητες του προγράμματος ΠΛΗ του ΕΑΠ.

**Υλοποίηση:** Ο σχεδιασμός αποδείχθηκε απόλυτα μεταφέρσιμος, καθώς υλοποιήθηκε με επιτυχία και χωρίς καμία τροποποίηση στον κώδικα VHDL σε δύο εντελώς διαφορετικές πλατφόρμες FPGA (DSD-i1 με Quartus και Tang Primer 25K με Gowin), επικυρώνοντας τη στιβαρότητα και την ανεξαρτησία του από συγκεκριμένα εργαλεία.

### 11.1. Περιορισμοί και κριτική

Αν και οι ενδείξεις από την δημιουργία και την χρήση του E80 ως Μικρόκοσμου είναι ενθαρρυντικές, δεν παύουν να ανήκουν στην σφαίρα της εύλογης υπόθεσης, καθώς η τεκμηρίωσή τους απαιτεί εμπειρική έρευνα με πραγματικούς εκπαιδευόμενους και τη διερεύνηση πολλών, ποικιλόμορφων έργων. Επομένως δεν μπορεί να υποστηριχθεί ότι ο E80 είναι Μικρόκοσμος, αλλά ότι έχει σχεδιαστεί με τέτοιο σκοπό.

Η αρχιτεκτονική ενός κύκλου που έγινε για την επίτευξη του στόχου της Χρησιμότητας, αποτέλεσε διδακτικό εμπόδιο στην περίπτωση της βελτιστοποίησης του αλγορίθμου διαί-

ρησης με την αντικατάσταση της ADD από την OR (Ενότητα 10.3, σ. 131). Το όφελος της “ισχυρής ιδέας” δεν μπορούσε να γίνει αντιληπτό καθώς οι απλές λογικές πράξεις και οι αριθμητικές που εκτελούνται από τον αργό κυματικό αθροιστή θεωρούνται ισοδύναμες.

Επιπλέον, η επιλογή για σχεδίαση χωρίς αριθμητικές βιβλιοθήκες, εξυπηρετεί μεν τον στόχο της Συνέχειας αλλά ταυτόχρονα οδηγεί σε έναν σχεδιασμό που δεν επιτρέπει στον μεταγλωττιστή (Quartus/Gowin) να αξιοποιήσει τα εσωτερικά δομικά στοιχεία του FPGA. Αυτό συσχετίζεται και με το πρακτικό πρόβλημα που προέκυψε κατά την εκτέλεση προγραμμάτων σε FPGA: δηλαδή στην απαίτηση για μεταγλώττιση του έργου σε Quartus/Gowin για τον προγραμματισμό της πλακέτας, μετά από την οποιαδήποτε μεταβολή στο πηγαίο πρόγραμμα. Η μεταγλώττιση αυτή είναι πολύ αργή και καθιστά δύσκολο τον πειραματισμό στο υλικό.

## 11.2. Μελλοντικές επεκτάσεις

Οι παραπάνω περιορισμοί ανοίγουν τον δρόμο για μελλοντικές επεκτάσεις του έργου με την εξής σειρά προτεραιότητας:

**Σειριακή είσοδος/έξοδος δεδομένων:** Θα μπορούσε να διερευνηθεί η δυνατότητα ανάγνωσης και εγγραφής δεδομένων σειριακά μέσω USB ώστε αφενός να επιτρέπεται το φόρτωμα προγραμμάτων χωρίς την χρονοβόρα μεταγλώττιση όλου του συστήματος σε Quartus/Gowin, και αφετέρου για να ολοκληρωθεί η πλήρης μορφή της αρχιτεκτονικής Neumann (Ενότητα 2.2.1, σ. 5) μέσω της υλοποίησης του υποσυστήματος αποθήκευσης R.

**Οθόνη ένδειξης περιεχομένων:** Η δυνατότητα της ένδειξης περιεχομένων σε μια μικρή οθόνη LCD σε πραγματικό χρόνο θα αναβάθμιζε την εποπτεία των καταχωρητών και της μνήμης, ενισχύοντας την άμεση αλληλεπίδραση που είναι απαραίτητη στον Παπερτιανό Μικρόκοσμο.

**Ανάλυση επιδόσεων:** Η αξιοποίηση των εργαλείων ανάλυσης χρονισμού των Quartus και Gowin για την μέτρηση της μέγιστης συχνότητας λειτουργίας του E80 θα μπορούσε να αποτελέσει την βάση για μια ρεαλιστική αιτιολόγηση των πλεονεκτημάτων μιας αρχιτεκτονικής πολλαπλών κύκλων με διοχέτευση (pipelining), επιτρέποντας τη διερεύνηση των σχεδιαστικών συμβιβασμών μεταξύ απλότητας και απόδοσης.

**Ανάπτυξη μεταγλωττιστή:** Το οικοσύστημα θα μπορούσε να ολοκληρωθεί με την ανάπτυξη ενός μεταγλωττιστή που θα δέχεται ένα υποσύνολο της C και θα το μετατρέπει σε συμβολική γλώσσα του E80, καλύπτοντας ολόκληρη την πορεία από τον πηγαίο κώδικα υψηλού επιπέδου μέχρι την εκτέλεση σε υλικό, όπως έχει γίνει στην TINYCPU (Ενότητα 3.2.5, σ. 23).



## Βιβλιογραφία

- Αλεξίου, Γ. (2001). *Μικροεπεξεργαστές*, Ελληνικό Ανοικτό Πανεπιστήμιο.
- Αλεξίου, Γ., Βέργος, Χ., Ευσταθίου, Κ., Θεοδωρίδης, Γ., Καβουσιανός, Χ., Κουφοπαύλου, Ο., Λαμπρινουδάκης, Κ., Λιοτόπουλος, Φ., Μόσχοβος Α., Μπακάλης, Δ., Μπεκάκος, Μ., Νικολαΐδης, Σ., Νικολός, Δ., Παλιουράς, Β., Παπαευσταθίου, Ι., Παπακώστας, Δ., Στουραΐτης Α., Σκόδρας, Α., Φωτόπουλος, Β., Χατζόπουλος, Α. (2013α). *Αρχιτεκτονική Υπολογιστών: Ασκήσεις γραπτών εργασιών & Θεμάτων εξετάσεων*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Αλεξίου, Γ., Βέργος, Χ., Ευσταθίου, Κ., Θεοδωρίδης, Γ., Καβουσιανός, Χ., Κουφοπαύλου, Ο., Λαμπρινουδάκης, Κ., Λιοτόπουλος, Φ., Μόσχοβος Α., Μπακάλης, Δ., Μπεκάκος, Μ., Νικολαΐδης, Σ., Νικολός, Δ., Παλιουράς, Β., Παπαευσταθίου, Ι., Παπακώστας, Δ., Στουραΐτης Α., Σκόδρας, Α., Φωτόπουλος, Β., Χατζόπουλος, Α. (2013β). *Μικροεπεξεργαστές: Ασκήσεις γραπτών εργασιών & Θεμάτων εξετάσεων*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Βασάλος, Ε., Γιαννακόπουλος, Κ., Κακαρούντας, Α., Κεραμίδας, Γ., Κίτσος, Π., Μάχντι, Α., Τοπάλης, Ε., & Φωτόπουλος, Β. (2023). *Ψηφιακά Συστήματα II: Εργαστηριακές Ασκήσεις*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Βέργος, Χ. Θ. (2022). *Βασικές αρχές οργάνωσης & λειτουργίας υπολογιστικών συστημάτων*. Διαφάνειες Αμφιθεάτρου. Τμήμα Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών. <https://eclass.upatras.gr/modules/document/file.php/CEID1244/CS.pdf>
- Βέργος, Χ. Θ. (2020). *Λογική Σχεδίαση II*. Τμήμα Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών. [http://pc-vlsi18.ceid.upatras.gr/files/slides\\_ld2.pdf](http://pc-vlsi18.ceid.upatras.gr/files/slides_ld2.pdf)
- Βέργος, Χ. Θ. (2007). *Πανεπιστημιακές Παραδόσεις στην Εισαγωγή στα Συστήματα Υπολογιστών*. Τμήμα Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών. [https://eclass.upatras.gr/modules/document/file.php/CEID1244/Notes\\_CS\\_v3.1\\_2006-2007.pdf](https://eclass.upatras.gr/modules/document/file.php/CEID1244/Notes_CS_v3.1_2006-2007.pdf)
- Δασυγένης, Μ. (Μετ.). (2013). *Οδηγός Εκμάθησης στην Assembly 8086*. Πανεπιστήμιο Δυτικής Μακεδονίας, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. (Μετάφραση του *8086 Assembler Tutorial for Beginners* του E.G. Carati, 2008). [https://arch.ece.uowm.gr/courses/arch/oc\\_Assembly.pdf](https://arch.ece.uowm.gr/courses/arch/oc_Assembly.pdf)
- Δασυγένης, Μ. (2021). *Αρχιτεκτονική Υπολογιστών - Ενότητα 10: Πέρασμα Παραμέτρων σε Διαδικασίες*, Πανεπιστήμιο Δυτικής Μακεδονίας. [https://web.archive.org/web/202210-24112919/http://arch.ict.e.uowm.gr/courses/arch/oc\\_archlab-theory-10.pdf](https://web.archive.org/web/202210-24112919/http://arch.ict.e.uowm.gr/courses/arch/oc_archlab-theory-10.pdf)
- Θραμπουλίδης, Κ. (2000). *Εισαγωγή στην Πληροφορική - Τόμος Γ': Γλώσσες Προγραμματισμού*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Καμέας, Α. Δ. (2008). *Εισαγωγή στην Πληροφορική - Τόμος Β': Τεχνικές Προγραμματισμού (Β' Έκδ.)*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Κόμης, Β. (2001). *Πληροφορική και Εκπαίδευση - Τόμος Α': Διδακτική της Πληροφορικής*. Ελληνικό Ανοικτό Πανεπιστήμιο.

- Λιναρδής, Π. (2008). *Ψηφιακά Συστήματα Τόμος Α' - Ψηφιακή Σχεδίαση Ι*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Νικολός, Δ. (2008). *Αρχιτεκτονική Υπολογιστών Ι (Β' έκδοση)*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Νικολός, Δ. (2010). *Ασκήσεις Αρχιτεκτονικής Υπολογιστών*. Γκιούρδας.
- Στουρνάρας, Α. (2022). *Υλοποίηση εφαρμογής γραφικών σε FPGA με χρήση παράλληλων postfix CPUs και διαμοιραζόμενη μνήμη* (Διπλωματική εργασία, Πανεπιστήμιο Πατρών). Επιβλέπων: Χ. Βέργος. <http://doi.org/10889/16364>
- Φράγκου, Σ., & Παπανικολάου, Κ. (2010). Εκπαιδευτική αξιοποίηση συστημάτων ρομποτικής. *Συνέδρια της Ελληνικής Επιστημονικής Ένωσης Τεχνολογιών Πληροφορίας & Επικοινωνιών στην Εκπαίδευση*, 149-151. <https://eproceedings.epublishing.ekt.gr/index.php/cetpe/article/view/5126>
- Φωτάκης, Δ., & Σπυράκης, Π. (2001). *Θεμελιώσεις Επιστήμης Η/Υ - Τόμος Α' - Αλγόριθμοι και Πολυπλοκότητα*. Ελληνικό Ανοικτό Πανεπιστήμιο.
- Χατζηλυγερούδης, Ι. (2008). *Εισαγωγή στην Πληροφορική - Τόμος Γ': Δομές Δεδομένων*. (Β' έκδ.). ΕΑΠ.
- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, techniques, and tools* (2nd ed.). Addison-Wesley.
- Altera. (2013). *Quartus II Handbook Version 13.1*. <https://cdrdv2.intel.com/v1/dl/getContent/653796>
- Ashenden, P. J. (2008). *The Designer's Guide to VHDL (3rd ed.)*. Morgan Kaufmann.
- Aylor, J. H., Waxman, R., & Scarratt, C. (1986). "VHDL - Feature Description And Analysis". *IEEE Design & Test of Computers*, 3(2), 17-27. <http://doi.org/10.1109/mdt.1986.294899>
- Bergé, J. M., Fonkoua, A., Maginot, S., & Rouillard, J. (1993). Direct instantiation. In *VHDL '92* (Vol. 229). Springer. [https://doi.org/10.1007/978-1-4615-3246-0\\_5](https://doi.org/10.1007/978-1-4615-3246-0_5)
- Bhardwaj, P., & Murugesan, S. (2016). *Design & simulation of a 32-bit RISC based MIPS processor using Verilog*. *International Journal of Research in Engineering and Technology*, 5(11), 166-172. <http://dx.doi.org/10.15623/ijret.2016.0511030>
- Bhasker, J. (1998). *A VHDL primer* (3rd ed.). Prentice Hall. <https://archive.org/details/vhdl-primer0000bhas>
- Bowen, J. (1985). *8085A MICROPROCESSOR Instruction Set Summary, Issue 1.1*. Oxford University Computing Laboratory. <http://archive.org/details/ProcessorInstructionSet8085>
- Calazans, N. L. V., & Moraes, F. G. (2001). Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses. *IEEE Transactions on Education*, 44(2), 109-119. <https://doi.org/10.1109/13.925805>

- Clements, A. (1999). Selecting a processor for teaching computer architecture. *Microprocessors and Microsystems*, 23(5), 281-290. [https://doi.org/10.1016/S0141-9331\(99\)00049-6](https://doi.org/10.1016/S0141-9331(99)00049-6)
- Coldwind, G., & Murray, D. (2024). *Top 100 assembly instructions (x86-64)*. HexArcana. [http://hexarcana.ch/b/2024-08-12-top-100-x86\\_64-asm-instructions](http://hexarcana.ch/b/2024-08-12-top-100-x86_64-asm-instructions)
- Coulter, N. S., & Kelly, N. H. (1986). Computer instruction set usage by programmers: An empirical investigation. *Communications of the ACM*, 29(7), 643-647. <https://doi.org/10.1145/325694.325727>
- Cummings, C. E., & Mills, D. (2002). Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use?. In *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*.
- Fanariotis, A., Orphanoudakis, T., Fotopoulos, V., & Kitsos, P. (2019). *DSD-i1: A Mixed Functionality Development Board Geared Towards Digital Systems Design Education*. 22nd Euromicro Conference on Digital System Design. <http://doi.org/10.1109/dsd.2019.00032>
- Freeman, M. (2019). *Simple CPU*. [http://www.simplecpudesign.com/simple\\_cpu\\_v1](http://www.simplecpudesign.com/simple_cpu_v1)
- Gray, J. (2000). *Hands-on computer architecture: teaching processor and integrated systems design with FPGAs*. WCAE '00: Proceedings of the 2000 workshop on Computer architecture education. <https://doi.org/10.1145/1275240.1275262>
- Gowin Semiconductor Corporation. (2025, April 30). Gowin synthesis user guide. <https://cdn.gowinsemi.com.cn/SUG550E.pdf>
- Guccione, S. (1994). *The Hardware Guy's FPGA Page*. [https://web.archive.org/web/19961228020332/http://www.io.com/~guccione/HW\\_list.html](https://web.archive.org/web/19961228020332/http://www.io.com/~guccione/HW_list.html)
- Haigh, T., Priestley, M., & Rope, C. (2016). *ENIAC in action: Making and remaking the modern computer*. The MIT Press. <https://archive.org/details/eniacinactionmak0000thom>
- Hamblen, J. O., Hall, T. S., & Furman, M. D. (2008). *Rapid Prototyping of Digital Systems: SOPC Edition*. Springer. <https://doi.org/10.1007/978-0-387-72671-7>
- Harris, S., & Harris, D. (2022). *Digital Design and Computer Architecture, RISC-V Edition*. Morgan Kaufmann. <http://doi.org/10.1016/C2019-0-00213-0>
- Hayes, J. P. (1998). *Computer architecture and organization* (3rd ed.). WCB/McGraw-Hill.
- Hennessy, J., & Patterson, D. (2012). *Computer architecture: A quantitative approach* (5th ed.). Morgan Kaufmann.
- Herman, G. L., Zilles, C., & Loui, M. C. (2011). How do students misunderstand number representations?. *Computer Science Education*, 21(3), 289-312. <http://doi.org/10.1080/08993408.2011.611712>
- Hyde, R. (2010). *The art of assembly language* (2nd ed.). No Starch Press.
- Institute of Electrical and Electronics Engineers. (1993). *IEEE standard multivalued logic system for VHDL model interoperability (Std\_logic\_1164) (IEEE Std 1164-1993)*. <https://doi.org/10.1109/IEEESTD.1993.115571>

- Institute of Electrical and Electronics Engineers. (1995). *IEEE Standard Glossary of Computer Hardware Terminology (IEEE Std 610.10-1994)*. <https://doi.org/10.1109/IEEESTD.1995.79522>
- Jensen, J. J. (2020). *Entity instantiation and component instantiation*. VHDLwhiz. <https://vhdlwhiz.com/entity-instantiation-and-component-instantiation>
- Kautz, W. H. (1970). *The Necessity of Closed Circuit Loops in Minimal Combinational Circuits*. IEEE Transactions on Computers, C-19(2), 162–164. <http://doi.org/10.1109/t-c.1970.222884>
- Leligkou, E., Voliotis, S., & Kakarountas, A. (2016). *Η Λογική Σχεδίαση στο Εργαστήριο*. Kallipos, Open Academic Editions. <https://dx.doi.org/10.57713/kallipos-691>
- Lewis, J. (2013). *VHDL-2008, The End of Verbosity!*. SynthWorks. [http://www.synthworks.com/papers/VHDL\\_2008\\_end\\_of\\_verbosity\\_2013.pdf](http://www.synthworks.com/papers/VHDL_2008_end_of_verbosity_2013.pdf)
- Li, Y., & Chu, W. (1996). Aizup - A pipelined processor design and implementation on Xilinx FPGA chip. *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, 98–106. IEEE. <https://doi.org/10.1109/FPGA.1996.564755>
- MacKenzie, S. (1988). A structured approach to assembly language programming. *IEEE Transactions on Education*, 31(2), 123–128. <https://doi.org/10.1109/13.2296>
- Maikantis, T., Natsiou, I., Volioti, C., Arvanitou, E.-M., Ampatzoglou, A., Mittas, N., Chatzigeorgiou, A., & Xinogalos, S. (2025). Code beauty is in the eye of the beholder: Exploring the relation between code beauty and quality. *Journal of Systems and Software*, 229, 112494. <https://doi.org/10.1016/j.jss.2025.112494>
- McLoughlin, I. (2017). *Computer Architecture: An Embedded Approach*. McGraw-Hill Education. <https://archive.org/details/computerarchitec0000mclo/page/n11/mode/2up>
- Microchip Technology Inc. (2016). *ATmega88/ATmega168: High Temperature Automotive Microcontroller*. [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-9365-Automotive-Microcontrollers-ATmega88-ATmega168\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-9365-Automotive-Microcontrollers-ATmega88-ATmega168_Datasheet.pdf)
- MOS Technology, Inc. (1976). *MCS6500 Microcomputer Family Programming Manual*. [https://archive.org/details/6500-50a\\_mcs6500pgmmanjan76](https://archive.org/details/6500-50a_mcs6500pgmmanjan76)
- Morris Mano, M. (1993). *Computer system architecture* (3rd ed.). Prentice-Hall. <https://archive.org/details/computer-system-architecture-morris-mano-third-edition>
- Morris Mano, M., Kime, C. R., & Martin, T. (2015). *Logic and Computer Design Fundamentals* (5th ed.). Pearson
- Motorola. (1976). *M6800 Programming Reference Manual*. Motorola Inc.
- Motorola. (1992). *M68000 family programmer's reference manual*. Motorola Inc.
- Nakano, K., & Ito, Y. (2008). Processor, assembler, and compiler design education using an FPGA. *14th IEEE International Conference on Parallel and Distributed Systems*, 723–728. IEEE. <https://doi.org/10.1109/ICPADS.2008.71>
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.



- Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, 39(3&4), 720–729 <https://blog.paperstatic.com/wp-content/uploads/2015/12/Papert-Big-Idea.pdf>
- Patterson, D., & Hennessy, L. (2014). *Computer organization and design: The hardware/software interface* (5th ed.). Morgan Kaufmann.
- Patti, D., Spadaccini, A., Palesi, M., Fazzino, F., & Catania, V. (2012). Supporting Undergraduate Computer Architecture Students Using a Visual MIPS64 CPU Simulator. *IEEE Transactions on Education*, 55(3). <https://doi.org/10.1109/TE.2011.2180530>
- Pedroni, V. A. (2020). *Circuit design with VHDL* (3rd ed.). MIT Press.
- Pesler, J. L., & Shoulders, D. (1982). On the F-16 MIL-STD-1750A Microprocessor and the F-16 MIL-STD-1589B Compiler. *Proceedings of the 3rd Technical Forum at Wright-Patterson AFB*. <http://apps.dtic.mil/sti/citations/ADA150583>
- Pinault, P. (2006). *Simple-CPU Instruction Set SC91-A*. The Simple CPU project. <http://www.simple-cpu.com/doc/sc91a-instruction-set-en.pdf>
- Psenka, C. E., Kim, K.-Y., Okudan Kremer, G. E., Haapala, K. R., & Jackson, K. L. (2017). Translating constructionist learning to engineering design education. *Journal of Integrated Design and Process Science*, 21(2), 3–20. <https://doi.org/10.3233/jid-2017-0004>
- Reaz, M. B. I., Islam, M. S., & Sulaiman, M. S. (2002). A single clock cycle MIPS RISC processor design using VHDL. *ICSE 2002 Proceedings*, pp. 199–203. <https://doi.org/10.1109/SMELEC.2002.1217806>
- Rojprasert, S., Neanchaleay, J., & Boonlue, S. (2013). A synthesis of self-directed learning design model with constructionism in the environment of new media in Thai higher education. *Review of Higher Education and Self-Learning*, 6(18), 157–165. <https://www.researchgate.net/publication/307204979>
- Sacristán, A. I. (2017). Constructionist computer programming for the teaching and learning of mathematical ideas at university level. In R. Göller, R. Biehler, R. Hochmuth, & H.-G. Rück (Eds.), *Didactics of mathematics in higher education as a scientific discipline* (khdm-Report 17-05, pp. 124–131). Universitätsbibliothek Kassel. <https://www.researchgate.net/publication/315098547>
- Salazar, C. & Birrer, B. (2020). Instrumentation and Extension of reduced, simulated Single Cycle MIPS architecture to improve Student Comprehension. *IEEE Frontiers in Education Conference (FIE)*, pp. 1–5. <http://doi.org/10.1109/FIE44824.2020.9273938>
- Solanki, M. S., & Sharma, A. (2021). A review paper on the difference between single-cycle and multi-cycle processor. *International Journal of Innovative Research in Computer Science and Technology*, 9(6), 86–90. <https://doi.org/10.55524/ijircst.2021.9.6.20>
- Stager, G. (2012). *Friends of Papertian constructionism*. In *Proceedings of the Constructionism 2012 Conference*, Athens, Greece. <http://constructingmodernknowledge.com/wp-content/uploads/2012/10/StagerConstructionism2012.pdf>

- Stallings, W. (2015). *Computer organization and architecture: Designing for performance* (10th ed.). Pearson.
- Stanley, T., Chetty, V., Styles, M., Jung, S.Y., Duarte, F., Lee, T.W.J., Gunter, M. & Fife, L. (2012). Teaching computer architecture through simulation: (a brief evaluation of CPU simulators). *Journal of Computing Sciences in Colleges*, 27(4), 37-44. <https://dl.acm.org/doi/10.5555/2167431.2167439>
- Tanenbaum, A., & Austin, T. (2013). *Structured Computer Organization* (6th ed.). Pearson Higher Education.
- Thind, V., Pandey, N., Pandey, B., & Hussain, D. M. A. (2017). Stack memory implementation and analysis of timing constraint, power and memory using FPGA. In *9th International Conference on Computational Intelligence and Communication Networks (CICN)* (pp. 215–220). IEEE. <https://doi.org/10.1109/CICN.2017.8319388>
- Vahid, F. (2011). *Digital design with RTL design, VHDL, and Verilog* (2nd ed.). Wiley.
- VHDL Analysis and Standardization Group. (2024). *IEEE 1076: VHDL Packages*. <https://opensource.ieee.org/vasg/Packages/-/tree/release/ieee>
- Neumann, J. (1945). *First draft of a report on the EDVAC*. United States Army Ordnance Department & University of Pennsylvania. <https://web.mit.edu/sts.035/www/PDFs/edvac.pdf>
- Waterman, A., Lee, Y., Patterson, D. A., & Asanović, K. (2016, May 31). *The RISC-V instruction set manual, Volume I: User-level ISA* (Version 2.1) (Technical Report No. UCB/EECS-2016-118). University of California at Berkeley, Department of Electrical Engineering and Computer Sciences. <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html>
- Yıldız, A., Ugurdag, H. F., Aktemur, B., İskender, D., & Gören, S. (2018). CPU design simplified. In *3rd International Conference on Computer Science and Engineering (UBMK)*, 630–632. <https://doi.org/10.1109/UBMK.2018.8566475>



## Παράρτημα Α: Πρωτογενή δεδομένα

Τα παρακάτω δεδομένα αφορούν στην συχνότητα χρήσης συγκεκριμένων εντολών σε εκπαιδευτικό υλικό εισαγωγής στην Assembly πανεπιστημιακής βαθμίδας. Συλλέχθηκαν και επεξεργάστηκαν με σκοπό να δοθεί μια απάντηση στο ερώτημα “ποιό είναι το ελάχιστο σύνολο εντολών που πρέπει να υποστηριχθεί στον E80;”

Χρησιμοποιώντας μηχανές αναζήτησης, εντοπίστηκαν πηγές που αφορούν σε ασκήσεις εισαγωγικών μαθημάτων assembly, στοχεύοντας κυρίως σε γενική ύλη αρχιτεκτονικής. Απο τις πηγές αυτές μετρήθηκαν οι εντολές που αναφέρονται στα παραδείγματα κώδικα και για κάθε εντολή καταγράφηκε η συχνότητα και το ποσοστό της εμφάνισής της. Η συχνότητα μετρήθηκε σύμφωνα με τα τμήματα στα οποία εμφανίζεται μια εντολή και όχι στο εύρος της χρήσης της σε κάθε τμήμα. Κάθε εντολή συνοδεύεται από την λειτουργία της η οποία έχει τυποποιηθεί έτσι ώστε να επιτρέπει σύγκριση με άλλους παρόμοιους πίνακες για την εξαγωγή του τελικού πίνακα 1 (σ. 19) με τα κανονικοποιημένα ποσοστά.

Στον πίνακα 14 καταγράφεται η συχνότητα των εντολών στην “Εισαγωγή στη γλώσσα Assembly του επεξεργαστή MIPS” της ενότητας ΠΛΗ10 (συγγραφείς: Ιωάννης Βογιατζής, Γιώργος Μανής, και Γιώργος Γιαγλής). Το υλικό στοχεύει στην ανάπτυξη δεξιοτήτων για τη δημιουργία απλών προγραμμάτων assembly, την κατανόηση βασικών εννοιών του MIPS και την περιγραφή των καταχωρητών και του μετρητή προγράμματος.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
LI	12	20,00%	Φόρτωση άμεσης τιμής
LW/LB	8	13,33%	Φόρτωση από μνήμη
ADD	7	11,67%	Πρόσθεση
SW	7	11,67%	Αποθήκευση στην μνήμη
ADDI	5	8,33%	Πρόσθεση άμεσης τιμής
MOVE	4	6,67%	Μεταφορά
BNE	3	5,00%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
SLL	3	5,00%	Αριστερή ολίσθηση
BGE	2	3,33%	Άλμα αν μεγαλύτερο ή ίσο
SUB	2	3,33%	Αφαίρεση
AND	1	1,67%	AND
BEQ	1	1,67%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
J	1	1,67%	Άλμα χωρίς συνθήκη
NOT	1	1,67%	NOT
OR	1	1,67%	OR
SRL	1	1,67%	Δεξιά ολίσθηση
XOR	1	1,67%	XOR

Πίνακας 14: Συχνότητα εντολών σε παραδείγματα MIPS (ΠΛΗ10, ΕΑΠ)

Στον πίνακα 15 καταγράφεται η συχνότητα εντολών απο τις ασκήσεις αρχιτεκτονικής υπολογιστών των “20” (2013α) για την θεματική ενότητα ΠΛΗ21. Το υλικό περιλαμβάνει ασκήσεις σε κωδικοποιήσεις, απόδοση, αρχιτεκτονικές επεξεργαστών, μνήμη, είσοδο-έξοδο κλπ. Αρκετές απο τις ασκήσεις χρησιμοποιούν εντολές για μηχανισμό στοίβας και έτσι αιτιολογείται η υψηλή συχνότητα αναφορών σε εντολές στοίβας.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
LOAD/LD	39	15,12%	Φόρτωση απο μνήμη ή άμεσης τιμής
ADD	36	13,95%	Πρόσθεση
STORE/ST	35	13,57%	Αποθήκευση στην μνήμη
SUB	31	12,02%	Αφαίρεση
MUL	30	11,63%	Πολλαπλασιασμός
DIV	22	8,53%	Διαίρεση
POP	21	8,14%	Απόθεση
PUSH	21	8,14%	Ωθηση
ADDI	4	1,55%	Πρόσθεση άμεσης τιμής
MOV	4	1,55%	Μεταφορά
AND	3	1,16%	AND
OR/ORI	3	1,16%	OR
NAND	2	0,78%	NAND
NOR	2	0,78%	NOR
DCR	1	0,39%	Μείωση κατα 1
INR	1	0,39%	Αύξηση κατα 1
SLL	1	0,39%	Αριστερή ολίσθηση
SLR	1	0,39%	Δεξιά ολίσθηση
XOR	1	0,39%	XOR

Πίνακας 15: Συχνότητα εντολών σε ασκήσεις αρχιτεκτονικής (ΠΛΗ21, ΕΑΠ)

Στον πίνακα 16 καταγράφεται η συχνότητα εντολών από τις ασκήσεις Μικροεπεξεργαστών των “20” (2013β) και Αλεξίου (2001) για την θεματική ενότητα ΠΛΗ21. Το υλικό περιλαμβάνει θέματα σχεδίασης συστήματος μνήμης, προγραμματισμού assembly του επεξεργαστή 8085, και διασύνδεσης περιφερειακών συσκευών.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
HLT	50	15,15%	Παύση εκτέλεσης
MOV	32	9,70%	Μεταφορά
ADD	28	8,48%	Πρόσθεση
LDA	23	6,97%	Φόρτωση από μνήμη
MVI	20	6,06%	Φόρτωση άμεσης τιμής
STA	19	5,76%	Αποθήκευση στην μνήμη
JNZ	18	5,45%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
SUB	17	5,15%	Αφαίρεση
INR	15	4,55%	Αύξηση κατά 1
DCR	13	3,94%	Μείωση κατά 1
JMP	13	3,94%	Άλμα χωρίς συνθήκη
LXI	12	3,64%	Φόρτωση διεύθυνσης σε ζεύγος καταχωρητών
CMP	10	3,03%	Σύγκριση
INX	9	2,73%	Αύξηση ζεύγους καταχωρητών κατά 1
JNC	7	2,12%	Άλμα αν δεν υπάρχει κρατούμενο
JC	6	1,82%	Άλμα αν υπάρχει κρατούμενο
ANI	5	1,52%	AND
JZ	5	1,52%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
RLC	4	1,21%	Αριστερή περιστροφή
STAX	4	1,21%	Αποθήκευση μέσω ζεύγους καταχωρητών
CPI	3	0,91%	Σύγκριση με άμεση τιμή
XRA	3	0,91%	XOR
ADC	2	0,61%	Πρόσθεση με κρατούμενο
CMA	2	0,61%	NOT
LDAX	2	0,61%	Φόρτωση μέσω ζεύγους καταχωρητών
RAR	2	0,61%	Δεξιά ολίσθηση μέσω κρατούμενου
RRC	2	0,61%	Δεξιά περιστροφή
SUI	2	0,61%	Αφαίρεση άμεσης τιμής
XCHG	2	0,61%	Ανταλλαγή τιμών

Πίνακας 16: Συχνότητα εντολών σε ασκήσεις μικροεπεξεργαστών (ΠΛΗ21, ΕΑΠ)

Στον πίνακα 17 καταγράφεται η συχνότητα εντολών απο ασκήσεις του μαθήματος CS 61 του Harvard για τη βελτίωση της κατανόησης υπολογιστικών συστημάτων. Το υλικό περιλαμβάνει κώδικα για την αρχιτεκτονική x86-64 με έμφαση στη σχέση της assembly με την C.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
MOV	50	16,78%	Μεταφορά
RET	39	13,09%	Επιστροφή απο υπορουτίνα
CMP	25	8,39%	Σύγκριση
ADD	20	6,71%	Πρόσθεση
JE/JZ	20	6,71%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
CALL	14	4,70%	Κλήση υπορουτίνας
JMP	14	4,70%	Άλμα χωρίς συνθήκη
INC	12	4,03%	Αύξηση κατα 1
JNE/JNZ	12	4,03%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
PUSH	12	4,03%	Ωθηση
XOR	12	4,03%	XOR
TEST	11	3,69%	Σύγκριση με AND
SUB	10	3,36%	Αφαίρεση
AND	8	2,68%	AND
POP	8	2,68%	Απόθεση
DEC	7	2,35%	Μείωση κατα 1
IMUL	7	2,35%	Πολλαπλασιασμός με πρόσημο
LEA	7	2,35%	Φόρτωση διεύθυνσης με υπολογισμούς
OR	3	1,01%	OR
XCHG	3	1,01%	Ανταλλαγή τιμών
SHL	2	0,67%	Αριστερή ολίσθηση
SHR	2	0,67%	Δεξιά ολίσθηση

Πίνακας 17: Συχνότητα εντολών σε ασκήσεις x86-64 (CS61, Harvard)

Στον πίνακα 18 καταγράφεται η συχνότητα εντολών απο το βιβλίο Ασκήσεις Αρχιτεκτονικής Υπολογιστών (Νικολός, 2010). Περιλαμβάνονται αναλυτικά λυμένες ασκήσεις, καλύπτοντας θέματα όπως δομή και λειτουργία υπολογιστών, οργάνωση και μεταφορά πληροφορίας, σχεδίαση επεξεργαστών, συστήματα μνήμης, κ.α.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
ADD	24	21,82%	Πρόσθεση
LOAD	20	18,18%	Φόρτωση απο μνήμη ή άμεσης τιμής
STORE	16	14,55%	Αποθήκευση στην μνήμη
SUB	10	9,09%	Αφαίρεση
BRNE/JNZ	6	5,45%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
NOP	6	5,45%	Καμία λειτουργία
AND	5	4,55%	AND
BRE	5	4,55%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
END	4	3,64%	Παύση εκτέλεσης
CLEAR	3	2,73%	Εκκαθάριση καταχωρητή
POP	3	2,73%	Απόθεση
PUSH	3	2,73%	Ωθηση
INC	2	1,82%	Αύξηση κατα 1
MUL	2	1,82%	Πολλαπλασιασμός
MOVE	1	0,91%	Μεταφορά

Πίνακας 18: Συχνότητα εντολών σε ασκήσεις αρχιτεκτονικής (Νικολός, ΤΜΗΥΠ)

Στον πίνακα 19 καταγράφεται η συχνότητα εντολών απο τον οδηγό εκμάθησης στην assembly του 8086 που χρησιμοποιείται στο Πανεπιστήμιο Δυτικής Μακεδονίας (Δασυγένης, 2013). Ο οδηγός ενθαρρύνει την χρήση του προσομοιωτή emu8086, και καλύπτει θέματα όπως πρόσβαση στη μνήμη, αριθμητικές εντολές, ροή προγράμματος, διαδικασίες, στοίβα και μακροεντολές. Ο οδηγός δεν έχει ασκήσεις· η συχνότητα υπολογίστηκε απο τον αριθμό των παραδειγμάτων κώδικα στα οποία εμφανίζεται μια εντολή.

Εντολή	Συχνότητα	Ποσοστό	Λειτουργία
MOV	23	33,33%	Μεταφορά
RET	14	20,29%	Επιστροφή απο υπορουτίνα
JMP	5	7,25%	Άλμα χωρίς συνθήκη
CALL	4	5,80%	Κλήση υπορουτίνας
ADD	3	4,35%	Πρόσθεση
CMP	3	4,35%	Σύγκριση
JE	3	4,35%	Άλμα αν ισότητα / μηδενικό αποτέλεσμα
JNE	2	2,90%	Άλμα αν ανισότητα / μη-μηδενικό αποτέλεσμα
MUL	2	2,90%	Πολλαπλασιασμός
POP	2	2,90%	Απόθεση
PUSH	2	2,90%	Ωθηση
XOR	2	2,90%	XOR
DEC	1	1,45%	Μείωση κατα 1
DIV	1	1,45%	Διαίρεση
INC	1	1,45%	Αύξηση κατα 1
LEA	1	1,45%	Φόρτωση διεύθυνσης με υπολογισμούς

Πίνακας 19: Συχνότητα εντολών στον οδηγό εκμάθησης 8086 (Δασυγένης, Π.Δ.Μ.)



## Παράρτημα Β: Ο κώδικας του E80 σε VHDL

### B.1 FPGA.vhd - FPGA implementation

```

-----
-- E80 FPGA implementation
-- Converts the fast 50 MHz clock to a slow deciHertz-class CLK.
-- Reads the DIPinput and Reset signals.
-- Runs the E80 CPU with the converted CLK, DIPinput and Reset.
-- Displays the flags, one register, the CLK, and the PC on the FPGA LED.
-- Allows reset, pause, speed control, and register selection via Joystick.
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL, work.firmware.ALL;
ENTITY FPGA IS PORT (
    CLK50MHz : IN STD_LOGIC; -- converted to slow CLK
    Reset     : IN STD_LOGIC; -- resets the PC & SP and uploads the firmware
    Pause     : IN STD_LOGIC; -- pauses the CLK
    Up        : IN STD_LOGIC; -- shows the next register on row B
    Down      : IN STD_LOGIC; -- shows the previous register on row B
    Right     : IN STD_LOGIC; -- increases CLK frequency
    Left      : IN STD_LOGIC; -- decreases CLK frequency
    Mid       : IN STD_LOGIC; -- resets CLK to the firmware-specified frequency
    DIPinput  : IN WORD;      -- 8-pin DIP switch input
    LED_rowA  : OUT WORD;
    LED_rowB  : OUT WORD;
    LED_rowC  : OUT WORD);
END;
ARCHITECTURE a1 OF FPGA IS
    SIGNAL CLK : STD_LOGIC; -- slow clock (deciHertz-class)
    -- clock control and display signals
    SIGNAL PC : WORD;
    SIGNAL R : WORDx8;
    SIGNAL reg: NATURAL RANGE 0 TO 7 := 0; -- current register address
    SIGNAL ResetComplete : STD_LOGIC := '0';
    ALIAS Halt : STD_LOGIC IS R(6)(3);
BEGIN
    -----
    -- E80 Computer instantiation
    -----
    Computer: ENTITY work.Computer PORT MAP(
        CLK,
        Reset,
        DIPinput,
        PC,
        R);
    -----
    -- Clock conversion and joystick input handling
    -----
    PROCESS (CLK50MHz)
        -- Tick20ns: 50 MHz counter
        -- RepeatRate (in 20ns): minimum time between joystick signals
        -- Delay: 50 MHz joystick signal repeat rate counter
        -- Frequency: initialized to firmware's DefaultFrequency value
        VARIABLE Tick20ns : NATURAL RANGE 0 TO 249999999 := 0;
        CONSTANT RepeatRate : NATURAL := 18000000; -- x 2/10^8 = 0.36 sec
        VARIABLE Delay : NATURAL RANGE 0 TO RepeatRate := 0;
        VARIABLE Frequency : DECIHERTZ := DefaultFrequency; -- see Firmware.vhd
    BEGIN
        IF RISING_EDGE(CLK50MHz) THEN
            IF NOT Pause THEN -- freeze CLK while Pause is being pressed
                -- CLK50MHz to deciHertz CLK conversion:
                -- 50MHz frequency = 50x10^6 cycles/sec => Period =
                -- 1/(50x10^6) sec/cycle = 2/10^8 sec = 20ns (1 tick).
                -- For a 1 deciHertz clock we need a 10 second period:
                -- 10sec = 5x10^8 x 2/10^8 sec = 5x10^8 * 20ns = 5x10^8 ticks.
            END IF
        END IF
    END PROCESS

```

```

-- CLK <= NOT CLK needs to be executed twice to make a period,
-- therefore 2.5×108 - 1 = 249999999 ticks, because the tick
-- count starts from zero.
Tick20ns := Tick20ns + 1;
IF Tick20ns * Frequency >= 249999999 THEN
    Tick20ns := 0;
    CLK <= NOT CLK;
    -- Signify a completed reset (may take a few seconds
    -- due to synchronization with the slow CLK).
    ResetComplete <= CLK AND Reset;
END IF;
END IF;
-- handle joystick inputs with a delay to prevent rapid toggling
IF Delay < RepeatRate THEN
    Delay := Delay + 1;
ELSIF Up THEN
    reg <= reg + 1;
    Delay := 0;
ELSIF Down THEN
    reg <= reg - 1;
    Delay := 0;
ELSIF Right THEN -- increase CLK speed
    IF Frequency > 850 THEN
        Frequency := 1000; -- ceiling
    ELSE
        Frequency := Frequency + Frequency/8 + 2;
    END IF;
    Delay := 0;
ELSIF Left THEN -- decrease CLK speed
    IF Frequency < 3 THEN
        Frequency := 1; -- floor
    ELSE
        Frequency := Frequency - Frequency/8 - 2;
    END IF;
    Delay := 0;
ELSIF Mid THEN
    Frequency := DefaultFrequency;
    Delay := 0;
END IF;
END IF;
END PROCESS;

-----
-- LED display
-----

-- Row A: status flags, selected register address and clock
-- [7]Carry [6]Zero [5]Sign [4]Overflow [3][2][1]Register Address [0]CLK
LED_rowA(7 DOWNTO 4) <= R(6) (7 DOWNTO 4); -- CZSV flags on R6 register
WITH reg SELECT LED_rowA(3 DOWNTO 1) <=
    "000" WHEN 0, "001" WHEN 1, "010" WHEN 2, "011" WHEN 3,
    "100" WHEN 4, "101" WHEN 5, "110" WHEN 6, "111" WHEN 7;
LED_rowA(0) <=
    '1'      WHEN Halt AND NOT Reset ELSE -- solid, bright
    '1'      WHEN ResetComplete         ELSE -- pulse, bright
    CLK50MHz WHEN CLK                   ELSE -- pulse, dim
    '0';
-- Row B: selected register value or DIP input during reset
LED_rowB <= DIPinput WHEN Reset ELSE R(reg);
-- Row C: program counter
LED_rowC <= PC;
END;
```

## B.2 Support.vhd - support library

```

-----
-- E80 Support Library
-- Provides types and a few functions to allow for cleaner code that's
-- compatible with Quartus Lite which doesn't fully support VHDL2008.
-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
PACKAGE support IS
    SUBTYPE WORD IS STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE WORDx8 IS ARRAY (0 TO 7) OF WORD; -- 8 registers
    TYPE WORDx256 IS ARRAY (0 TO 255) OF WORD; -- RAM signals
    SUBTYPE REG_ADDR IS STD_LOGIC_VECTOR(2 DOWNTO 0);
    SUBTYPE DECIHERTZ IS NATURAL RANGE 1 TO 1000;
    FUNCTION int(arg : STD_LOGIC_VECTOR) RETURN NATURAL;
    FUNCTION match(arg1, arg2 : STD_LOGIC_VECTOR) RETURN BOOLEAN;
    FUNCTION match(arg1, arg2 : STD_LOGIC_VECTOR) RETURN STD_LOGIC;
END;
PACKAGE BODY support IS
    -- Equivalent to TO_INTEGER with a logic vector argument to be used
    -- for indexing purposes where logic vectors are considered unsigned.
    FUNCTION int(arg : STD_LOGIC_VECTOR) RETURN NATURAL IS
        VARIABLE result : NATURAL := 0;
    BEGIN
        FOR I IN arg'RANGE LOOP
            result := 2*result;
            IF arg(I) = '1' THEN
                result := result + 1;
            END IF;
        END LOOP;
        RETURN result;
    END;
    -- Simplified version of STD_MATCH that can be used in both boolean and
    -- std_logic expressions to substitute "?=" matching and unary logic
    -- operators which Quartus Lite doesn't support.
    FUNCTION match(arg1, arg2 : STD_LOGIC_VECTOR) RETURN BOOLEAN IS
        -- reorder to make DOWNTOs compatible with FOR I IN RANGE
        ALIAS v1 : STD_LOGIC_VECTOR(1 TO arg1'LENGTH) IS arg1;
        ALIAS v2 : STD_LOGIC_VECTOR(1 TO arg2'LENGTH) IS arg2;
    BEGIN
        FOR I IN v2'RANGE LOOP -- match("abc","a") = true
            IF v1(I) = '-' OR v2(I) = '-' THEN -- skip don't cares
                NEXT;
            -- compare as bit to avoid std_logic's matching table
            ELSIF To_bit(v1(I)) XOR To_bit(v2(I)) THEN
                RETURN FALSE;
            END IF;
        END LOOP;
        RETURN TRUE;
    END;
    -- overloaded to allow matching in STD_LOGIC context
    FUNCTION match(arg1, arg2 : STD_LOGIC_VECTOR) RETURN STD_LOGIC IS
    BEGIN
        IF match(arg1, arg2) THEN
            RETURN '1';
        ELSE
            RETURN '0';
        END IF;
    END;
END;

```

### B.3 Computer.vhd - E80 Computer

```

-----
-- E80 Computer
-- Interconnects the CPU with RAM for instruction/data access.
-- Routes DIPinput to CPU when MemAddr=0xFF (memory-mapped I/O).
-- Outputs PC and registers for LED display on the FPGA.
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY Computer IS PORT (
    CLK      : IN STD_LOGIC;
    Reset    : IN STD_LOGIC;
    DIPinput  : IN WORD;      -- 8-pin DIP switch user input
    PC       : BUFFER WORD;
    R        : OUT WORDx8);  -- Passthrough for FPGA LED
END;
ARCHITECTURE a1 OF Computer IS
    SIGNAL Instr1, Instr2 : WORD;
    SIGNAL MemAddr : WORD;
    SIGNAL MemWriteEn : STD_LOGIC;
    SIGNAL MemNext : WORD;
    SIGNAL Mem : WORD;
    SIGNAL Data : WORD; -- [MemAddr] if MemAddr<0xFF, else DIP input
BEGIN
    RAM : ENTITY work.RAM PORT MAP(
        CLK,
        Reset,
        PC,      -- current instruction address
        MemAddr, -- memory address to be read or written
        MemWriteEn, -- write enable for MemAddr
        MemNext, -- next cycle value of [MemAddr]
        Instr1,  -- [PC] first part of current instruction
        Instr2,  -- [PC+1] 2nd part (ignored for 1-word instructions)
        Mem);    -- [MemAddr] (RAM output)

    Data <= DIPinput WHEN match(MemAddr,x"FF") ELSE Mem;

    CPU : ENTITY work.CPU PORT MAP(
        CLK,
        Reset,
        Instr1,
        Instr2,
        Data,
        PC,
        MemAddr,
        MemWriteEn,
        MemNext,
        R);
END;

```

## B.4 Computer\_TB.vhd - E80 Computer test bench

```
-- E80 Computer test bench
LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL, work.firmware.ALL;
ENTITY Computer_TB IS END;
ARCHITECTURE a1 OF Computer_TB IS
    SIGNAL CLK      : STD_LOGIC := '1';
    SIGNAL Reset     : STD_LOGIC := '1';
    SIGNAL DIPinput  : WORD := SimDIP;
    SIGNAL PC        : WORD;
    SIGNAL R         : WORDx8;
    SIGNAL Halt      : STD_LOGIC;
BEGIN
    Halt <= R(6)(3);
    -- if Halt=1, CLK stops pulsing => GHDL simulation ends
    CLK <= '0' AFTER 50 ps WHEN CLK OR Halt ELSE '1' AFTER 50 ps;
    Reset <= '0' AFTER 120 ps;
    Computer : ENTITY work.Computer PORT MAP(CLK, Reset, DIPinput, PC, R);
END;
```

## B.5 RAM.vhd - 256x8 RAM

```
-----
-- E80 256x8 RAM
-- Stores 256 words in 8-bit flip flop.
-- Reads a two-word instruction at PC and PC+1 addresses, and the Mem
-- word at MemAddr; updates the Mem word to MemNext if MemWriteEn=1.
-- Uploads the firmware to the RAM upon a synchronous reset.
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL, work.firmware.ALL;
ENTITY RAM IS PORT (
    CLK      : IN STD_LOGIC;
    Reset    : IN STD_LOGIC;
    PC       : IN WORD;          -- current instruction address
    MemAddr  : IN WORD;          -- address for Mem and MemNext
    MemWriteEn : IN STD_LOGIC;  -- write enable
    MemNext  : IN WORD;          -- if MemWriteEn, MemNext -> [MemAddr]
    Instr1   : OUT WORD;         -- [PC]
    Instr2   : OUT WORD;         -- [PC+1]
    Mem      : OUT WORD);        -- [MemAddr]
END;
ARCHITECTURE a1 OF RAM IS
    SIGNAL RAMnext, RAM : WORDx256;
    SIGNAL i1, i2, a : NATURAL RANGE 0 TO 255;
BEGIN
    i1 <= int(PC);
    i2 <= i1+1;
    a <= int(MemAddr);
    DFF_Array: FOR i IN 0 TO 255 GENERATE
        DFF8 : ENTITY work.DFF8 PORT MAP(CLK, RAMnext(i), RAM(i));
        RAMnext(i) <=
            Firmware(i) WHEN Reset = '1' ELSE
            MemNext      WHEN MemWriteEn = '1' AND i = a ELSE
            RAM(i);
    END GENERATE;
    Instr1 <= RAM(i1);
    Instr2 <= RAM(i2);
    Mem <= RAM(a);
END;
```

## B.6 Firmware.vhd - multiplication and division with subroutines

```
-----
-- Multiplication and division with subroutines (divmul.asm)
-----

LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
CONSTANT DefaultFrequency : DECIHERTZ := 15; -- 1 to 1000
CONSTANT SimDIP : WORD := "00000000"; -- DIP input for testbench only
CONSTANT Firmware : WORDx256 := (
0  => "00001110", 1  => "00101010", -- CALL 42
2  => "11100000", -- PUSH R0
3  => "00001110", 4  => "00000111", -- CALL 7
5  => "11110010", -- POP R2
6  => "00000000", -- HLT
7  => "00010001", 8  => "10110011", -- MOV R1, 179
9  => "00010010", 10 => "00001100", -- MOV R2, 12
11 => "00010000", 12 => "00000000", -- MOV R0, 0
13 => "00010011", 14 => "00000001", -- MOV R3, 1
15 => "10111000", 16 => "00100001", -- CMP R2, R1
17 => "00000100", 18 => "00011011", -- JC 27
19 => "11010010", 20 => "10000000", -- BIT R2, 128
21 => "00000111", 22 => "00011011", -- JNZ 27
23 => "11000010", -- LSHIFT R2
24 => "11000011", -- LSHIFT R3
25 => "00000010", 26 => "00001111", -- JMP 15
27 => "10111000", 28 => "00010010", -- CMP R1, R2
29 => "00000101", 30 => "00100011", -- JNC 35
31 => "00111000", 32 => "00010010", -- SUB R1, R2
33 => "01101000", 34 => "00000011", -- OR R0, R3
35 => "10100010", -- RSHIFT R2
36 => "10100011", -- RSHIFT R3
37 => "00000110", 38 => "00101001", -- JZ 41
39 => "00000010", 40 => "00011011", -- JMP 27
41 => "00001111", -- RETURN
42 => "10010001", 43 => "01100100", -- LOAD R1, [100]
44 => "10010010", 45 => "01100101", -- LOAD R2, [101]
46 => "00010000", 47 => "00000000", -- MOV R0, 0
48 => "11010010", 49 => "11111111", -- BIT R2, 255
50 => "00000110", 51 => "00111110", -- JZ 62
52 => "11010010", 53 => "00000001", -- BIT R2, 1
54 => "00000110", 55 => "00111010", -- JZ 58
56 => "00101000", 57 => "00000001", -- ADD R0, R1
58 => "11000001", -- LSHIFT R1
59 => "10100010", -- RSHIFT R2
60 => "00000010", 61 => "00110000", -- JMP 48
62 => "00001111", -- RETURN
100 => "00000111", -- 7
101 => "00011101", -- 29
OTHERS => "UUUUUUUU");END;
```



## B.7 CPU.vhd - E80 CPU

```

-----
-- E80 CPU
-- Fetches instruction and data from the RAM and the DIP input.
-- Decodes the instruction to get the control signals, addresses and values.
-- Reads registers from the Register Array.
-- Assigns input values to the ALU.
-- Writes the result of the ALU to the registers and the RAM.
-- Advances to the next instruction.
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY CPU IS PORT (
    CLK          : IN STD_LOGIC;
    Reset        : IN STD_LOGIC;
    Instr1       : IN WORD;      -- [PC] first part of current instruction
    Instr2       : IN WORD;      -- [PC+1] 2nd part (ignored for 1-word instr.)
    Data         : IN WORD;      -- [MemAddr] if MemAddr<0xFF, else DIP input
    PC           : BUFFER WORD;  -- current instruction address
    MemAddr      : OUT WORD;     -- memory address to be read or written
    MemWriteEn   : OUT STD_LOGIC; -- write enable for [MemAddr]
    MemNext      : OUT WORD;     -- next cycle value of [MemAddr]
    R            : OUT WORDx8);  -- FPGA LED output
END;

ARCHITECTURE a1 OF CPU IS
    -- Instruction format signal aliases
    --
    -- Instr1
    -- 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
    -- +-----+-----+
    -- Type 1 | opcode |
    -- +-----+-----+
    -- +-----+-----+
    -- Type 2 | opcode | 0 | reg |
    -- +-----+-----+
    -- +-----+-----+
    -- Type 3 | opcode | direct |
    -- +-----+-----+
    -- +-----+-----+
    -- Type 4 | opcode | 1 | 0 | 0 | 0 | 0 | reg1 | 0 | reg2 |
    -- +-----+-----+
    -- +-----+-----+
    -- Type 5 | opcode | 0 | reg | immediate or direct |
    -- +-----+-----+
    -- op2isReg discerns between types 4 and 5 to check if the flexible 2nd
    -- operand is a register (op2isReg=1) or an immediate/direct (op2isReg=0)
    -- eg. in ADD R0,R1 op2isReg=1 whereas ADD R0,10 op2isReg=0.
    ALIAS op2isReg : STD_LOGIC IS Instr1(3);
    ALIAS Instr1Reg : REG_ADDR IS Instr1(2 DOWNTO 0);
    ALIAS Instr2Reg1 : REG_ADDR IS Instr2(6 DOWNTO 4);
    ALIAS Instr2Reg2 : REG_ADDR IS Instr2(2 DOWNTO 0);
    -- Instruction Decoder output
    SIGNAL isHLT, isNOP, isJMP, isJMPr, isJC, isJNC, isJZ, isJNZ, isJV, isJNV,
        isJS, isJNS, isCALL, isRETURN, isSTORE, isSTOREr, isLOAD, isLOADr,
        isSHIFT, isPUSH, isPOP, isStack : STD_LOGIC;
    -- Register signals
    SIGNAL A_reg : REG_ADDR; -- accumulator address (usually 1st operand)
    SIGNAL A_val : WORD;     -- current value of A_reg
    SIGNAL A_next : WORD;    -- next cycle value of A_reg
    SIGNAL B_reg : REG_ADDR; -- read register address (usually 2nd operand)
    SIGNAL B_val : WORD;     -- current value of B_reg
    SIGNAL W_reg : REG_ADDR; -- write register address (usually R6)
    SIGNAL W_next : WORD;    -- next cycle value of W_reg
    SIGNAL Flags : WORD;     -- current value of Flags Register
    CONSTANT FlagsRegister : REG_ADDR := "110"; -- R6
    CONSTANT StackPointer : REG_ADDR := "111"; -- R7

```

```
-- Flags
ALIAS Carry      : STD_LOGIC IS Flags(7);
ALIAS Zero       : STD_LOGIC IS Flags(6);
ALIAS Sign       : STD_LOGIC IS Flags(5);
ALIAS Overflow   : STD_LOGIC IS Flags(4);
ALIAS Halt       : STD_LOGIC IS Flags(3);

-- ALU signals
SIGNAL ALUop : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL ALUinB : WORD;
SIGNAL FlagsOut : WORD;
ATTRIBUTE syn_keep : BOOLEAN; -- FPGA fix, see MemAddr assignment
ATTRIBUTE syn_keep OF ALUinB : SIGNAL IS TRUE;

-- Execution control flow signals
-- if Jumping=0, PCnext ← Adjacent
-- if Jumping=1, PCnext ← jump, call, or return address
SIGNAL Size      : WORD;      -- current instruction size
SIGNAL Adjacent  : WORD;      -- address of adjacent instruction (PC + Size)
SIGNAL Jumping   : STD_LOGIC; -- are we jumping ?
SIGNAL PCnext    : WORD;      -- address of the next instruction to execute

BEGIN

-----
-- Instruction Decoder
-----

-- By default, all instructions are assumed to be types 5 and 6 with
-- opcodes matching their required ALUop. The following instructions are
-- either exceptions or need custom handling.
isHLT    <= match(Instr1,"00000000");
isNOP    <= match(Instr1,"00000001");
isJMP    <= match(Instr1,"0000001-");
isJMPPr  <= match(Instr1,"00000011");    -- JMP reg
isJC     <= match(Instr1,"00000100");
isJNC    <= match(Instr1,"00000101");
isJZ     <= match(Instr1,"00000110");
isJNZ    <= match(Instr1,"00000111");
isJS     <= match(Instr1,"00001010");
isJNS    <= match(Instr1,"00001011");
isJV     <= match(Instr1,"00001100");
isJNV    <= match(Instr1,"00001101");
isCALL   <= match(Instr1,"00001110");
isRETURN <= match(Instr1,"00001111");
isSTORE  <= match(Instr1,"1000----");
isSTOREr <= match(Instr1,"10001000");    -- STORE reg1,reg2
isLOAD   <= match(Instr1,"1001----");
isLOADr  <= match(Instr1,"10011000");    -- LOAD reg1,reg2
isSHIFT  <= match(Instr1,"10100---") OR -- RSHIFT
            match(Instr1,"11000---");    -- LSHIFT
isPUSH   <= match(Instr1,"11100---");
isPOP    <= match(Instr1,"11110---");
isStack  <= isPUSH OR isCALL OR isPOP OR isRETURN;

-----
-- Arithmetic Logical Unit
-----

-- All instructions start with their ALU opcode, except for CALL & RETURN.
-- ALUinA/ALUout are assigned to A_reg's current A_val and A_next values.
-- ALUinB is assigned to either an immediate value (Instr2), a register
-- value (B_val), or data from the RAM or DIP input (Data).
ALU : ENTITY work.ALU PORT MAP(
    ALUop,
    A_val,    -- ALUinA
    ALUinB,
    Flags,
    A_next,   -- ALUout, results are accumulated on A_reg
    FlagsOut);

ALUop <=
    "1110" WHEN isCALL ELSE -- push PC
    "1111" WHEN isRETURN ELSE -- pop PC
    Instr1(7 DOWNTO 4);
ALUinB <=
```

```

Data      WHEN isLOAD  ELSE -- RAM or DIP input
B_val     WHEN op2isReg ELSE
Instr2;

-----
-- Registers
-----

RegisterFile : ENTITY work.RegisterFile PORT MAP(
    CLK,
    Reset,
    A_reg,  -- accumulator (usually 1st operand) address
    A_next, -- next cycle value of accumulator = ALUout
    B_reg,  -- read register (usually 2nd operand) address
    W_reg,  -- write register (flags or POP reg) address
    W_next, -- next cycle value of W_reg
    A_val,  -- current value of A_reg
    B_val,  -- current value of B_reg
    Flags,  -- current value of Flags Register
    R);     -- FPGA LED output (not accessible on the CPU)
-- A_reg, the ALU accumulator, is set to the first operand in all
-- instruction types, except for stack operations where it's set to the
-- stack pointer to be increased or decreased by the ALU.
A_reg <=
    StackPointer WHEN isStack ELSE
    Instr2Reg1   WHEN op2isReg ELSE -- type 4
    Instr1Reg;   -- type 2 or 5
-- B_reg's value is used in ALUinB, or as a memory address for LOAD/STORE.
-- B_reg is almost exclusively set to Instr2Reg2, except for PUSH where
-- A_reg is assigned to the stack pointer and Instr1Reg's value needs to
-- be read through B_reg / B_val to be stored in the RAM.
B_reg <= Instr1Reg WHEN isPUSH ELSE Instr2Reg2;
-- W_reg is the write register. It's almost exclusively set to the
-- flags register, except for POP which writes on its Instr1Reg.
W_reg <= Instr1Reg WHEN isPOP ELSE FlagsRegister;
W_next <=
    Data      WHEN isPOP ELSE
    Flags OR "00001000" WHEN isHLT ELSE -- HLT sets the Halt flag
    FlagsOut; -- ALU flags output
-----
-- Memory access
-----
-- All operations that use memory addressing, specified with [...]
-- brackets in the ISA, assign MemAddr to the address in the bracket.
-- Assigning MemAddr to A_next (ALUout) causes conditional loop warnings
-- in Quartus & Gowin. That's due to their optimizations which cause an
-- "A_next → MemAddr → Data → ALUinB → A_next" loop; they somehow make
-- ALUinB depend on Data for Stack operations. By replacing A_next with
-- STD_LOGIC_VECTOR(UNSIGNED(A_val)-1) the problem is "fixed" but this
-- adds the undesirable dependency on numeric_std. Thankfully, Gowin
-- supports the syn_keep attribute to disable these optimizations, thus
-- eliminating the problem.
MemAddr <=
    B_val  WHEN isSTOREr OR isLOADr ELSE -- Instr2Reg2
    A_val  WHEN isPOP OR isRETURN  ELSE -- SP before increase
    A_next WHEN isPUSH OR isCALL  ELSE -- SP after decrease
    Instr2; -- STORE / LOAD direct
-- set MemWriteEn for all "→ [...]" operations in the ISA Cheatsheet
MemWriteEn <= isSTORE OR isPUSH OR isCALL;
-- if MemWriteEn, [MemAddr] ← MemNext
MemNext <=
    B_val  WHEN isPUSH ELSE -- push the value of B_reg
    Adjacent WHEN isCALL ELSE -- push the RETURN address
    A_val; -- store the value of Instr1Reg
-----
-- Program flow control
-----
-- Instructions have a variable Size of 1 or 2 words.
-- Adjacent, the address of the following instruction, is set to PC+Size
-- by using an 8-bit full adder (PC_Adder).

```

```
-- If a jump is occurring, Adjacent is ignored and the PC is set to the
-- target address which is either the Instr2 argument of the jump/call
-- instruction, or memory data from the stack (in case of RETURN).
-- The program counter is stored in an 8-bit D flip-flop.
Size <=
    x"01" WHEN isHLT OR isNOP OR isRETURN OR isSHIFT OR isPUSH OR isPOP ELSE
    x"02";
PC_Adder : ENTITY work.FA8 PORT MAP(PC, Size, '0', Adjacent);
Jumping <=
    isJMP OR isCALL OR isRETURN OR
    (isJC AND Carry) OR (isJNC AND NOT Carry) OR
    (isJZ AND Zero) OR (isJNZ AND NOT Zero) OR
    (isJS AND Sign) OR (isJNS AND NOT Sign) OR
    (isJV AND Overflow) OR (isJNV AND NOT Overflow);
PCnext <=
    x"00"      WHEN Reset          ELSE
    PC         WHEN isHLT OR Halt  ELSE -- HLT works on the current cycle
    Adjacent   WHEN NOT Jumping    ELSE
    Data       WHEN isRETURN       ELSE
    B_val      WHEN isJMPPr        ELSE
    Instr2;
PC_DFF : ENTITY work.DFF8 PORT MAP(CLK, PCnext, PC);
END;
```

## B.8 RegisterFile.vhd - 8x8 register file

```
-----
-- E80 8x8 Register File
-- Reads A_reg, B_reg, and Flags; writes to A_reg and W_reg. Reset is
-- synchronous (to ensure a full first cycle) and clears only the SP and
-- the Halt flag, leaving the rest to undefined.
-- The R-array (FPGA LED output) is passed to the final FPGA component for
-- display and should *not* be accessible by the CPU.
-- R0-R5: General-purpose registers, R6: Flags register, R7: Stack pointer
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY RegisterFile IS PORT (
    CLK      : IN STD_LOGIC;
    Reset    : IN STD_LOGIC;
    A_reg    : IN REG_ADDR;    -- read/write register
    A_next   : IN WORD;        -- next cycle value of A_reg
    B_reg    : IN REG_ADDR;    -- read only register
    W_reg    : IN REG_ADDR;    -- write only register
    W_next   : IN WORD;        -- next cycle value of W_reg
    A_val    : OUT WORD;       -- current value of A_reg
    B_val    : OUT WORD;       -- current value of B_reg
    Flags    : OUT WORD;       -- current value of Flags Register
    R        : OUT WORDx8);    -- all current register values for FPGA LED output
END;
ARCHITECTURE a1 OF RegisterFile IS
    SIGNAL Rnext : WORDx8; -- stored values
    SIGNAL a, b, w : NATURAL RANGE 0 TO 7; -- indexes
    -- Clear the Halt flag and reset the Stack Pointer to 255 to reserve this
    -- address for DIP input. Everything else is set to undefined for easier
    -- inspection in ModelSim/GHDL and to enforce good programming practices.
    CONSTANT Init : WORDx8 := (6 => "UUUUUUUU", 7 => x"FF", OTHERS => x"UU");
BEGIN
    a <= int(A_reg);
    b <= int(B_reg);
    w <= int(W_reg);
    DFF_Array: FOR i IN 0 TO 7 GENERATE
        DFF8 : ENTITY work.DFF8 PORT MAP(CLK, Rnext(i), R(i));
        Rnext(i) <=
            Init(i) WHEN Reset ELSE
            -- Typically, a=w when trying to modify the FLAGS register,
            -- eg. by OR FLAGS, 0b10000000 which would set the Carry flag.
    END GENERATE;
```

```

-- In these cases, it's essential to pass the data from ALUresult
-- (A_next) instead of the normal ALU Flags output (W_next).
A_next WHEN i = a ELSE -- higher priority for A_next when a=w
W_next WHEN i = w ELSE
R(i);
END GENERATE;
A_val <= R(a);
B_val <= R(b);
Flags <= R(6);
END;

```

## B.9 DFF8.vhd - 8-bit D flip-flop

```

-----
-- E80 8-bit D flip-flop
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY DFF8 IS PORT (
    CLK    : IN STD_LOGIC;
    D      : IN WORD;
    Q      : OUT WORD);
END;
ARCHITECTURE a1 OF DFF8 IS
BEGIN
    PROCESS (CLK) BEGIN
        IF RISING_EDGE (CLK) THEN
            Q <= D;
        END IF;
    END PROCESS;
END;

```

## B.10 ALU.vhd - arithmetic logic unit

```

-----
-- E80 Arithmetic Logic Unit
-- Performs addition, subtraction, rotation, and logical operations.
-- The calculated result and/or flags is discarded in some operations.
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY ALU IS PORT (
    ALUop    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    ALUinA   : IN WORD;
    ALUinB   : IN WORD;
    FlagsIn  : IN WORD; -- Carry, Zero, Sign, Overflow, Halt
    ALUout   : OUT WORD;
    FlagsOut : OUT WORD);
END;
ARCHITECTURE a1 OF ALU IS
    ALIAS A : WORD IS ALUinA;
    SIGNAL B : WORD; -- ALUinB or 1 for INR/DCR
    -- ALU Decoder output
    SIGNAL isBypass, isAssign, isADD, isSUB, isROR, isAND, isOR, isXOR,
        isRSHIFT, isCMP, isLSHIFT, isBIT, isDCR, isINR,
        FullFlags, DiscardFlags, DiscardResult : STD_LOGIC;
    -- Barrel shifter result
    SIGNAL Rotated : WORD;
    -- Adder / subtractor output
    SIGNAL Sum_C, Sum_V : STD_LOGIC;
    SIGNAL Sum : WORD;
    -- Result signals (assigned to ALUout/FlagsOut or ignored)
    SIGNAL Result : WORD;
    SIGNAL C, Z, S, V : STD_LOGIC;
BEGIN
    -----+-----+-----+
    -- ALUop Decoder                                | ALUout          | CZSV |

```

```

-----+-----+-----+
isBypass <= match(ALUop,"-000"); --| A (J*, STORE, CALL, etc) | ** |
isAssign <= match(ALUop,"-001"); --| B (MOV, LOAD) | ** |
isADD <= match(ALUop,"0010"); --| A + B | **** |
isSUB <= match(ALUop,"-011"); --| A - B (includes CMP) | **** |
isROR <= match(ALUop,"0100"); --| A rotated by B mod 8 bits | ** |
isAND <= match(ALUop,"-101"); --| A AND B (includes BIT) | ** |
isOR <= match(ALUop,"0110"); --| A OR B | ** |
isXOR <= match(ALUop,"0111"); --| A XOR B | ** |
isRSHIFT <= match(ALUop,"1010"); --| A >> 1, C ← A(0), V ← S flip | **** |
isCMP <= match(ALUop,"1011"); --| SUB, discard result | **** |
isLSHIFT <= match(ALUop,"1100"); --| A << 1, C ← A(7), V ← S flip | **** |
isBIT <= match(ALUop,"1101"); --| AND, discard result | ** |
isDCR <= match(ALUop,"1110"); --| A - 1 (PUSH, CALL) | ** |
isINR <= match(ALUop,"1111"); --| A + 1 (POP, RETURN) | ** |
FullFlags <= isADD OR isSUB OR isRSHIFT OR isLSHIFT;
DiscardFlags <= isBypass OR isINR OR isDCR;
DiscardResult <= isBypass OR isCMP OR isBIT;
-----

-- Full Adder / Subtractor
-----

B <= x"01" WHEN isDCR OR isINR ELSE ALUinB;
ALU_Adder : ENTITY work.FA8 PORT MAP(
    A,
    B,
    isSUB OR isDCR, -- 1 = subtraction (includes CMP)
    Sum,
    Sum_C,
    Sum_V);
-----

-- Barrel shifter
-----

-- Rotation is determined by the 3 LSBs of operand B.
-- It had to be performed manually because Quartus Lite
-- doesn't support VHDL 2008 SRL/SLL/ROR/ROL operators.
WITH B(2 DOWNTO 0) SELECT Rotated <=
    A(0 DOWNTO 0) & A(7 DOWNTO 1) WHEN "001", -- right 1, left 7
    A(1 DOWNTO 0) & A(7 DOWNTO 2) WHEN "010", -- right 2, left 6
    A(2 DOWNTO 0) & A(7 DOWNTO 3) WHEN "011", -- right 3, left 5
    A(3 DOWNTO 0) & A(7 DOWNTO 4) WHEN "100", -- right 4, left 4
    A(4 DOWNTO 0) & A(7 DOWNTO 5) WHEN "101", -- right 5, left 3
    A(5 DOWNTO 0) & A(7 DOWNTO 6) WHEN "110", -- right 6, left 2
    A(6 DOWNTO 0) & A(7 DOWNTO 7) WHEN "111", -- right 7, left 1
    A WHEN OTHERS; -- no rotation
-----

-- Result & Flags
-----

-- Result needs to be calculated, even if discarded later, for flags-only
-- operations. For shift operations, the carry bit holds the shifted
-- bit while the overflow bit is set if the sign bit was flipped.
Result <=
    B WHEN isAssign ELSE
    Rotated WHEN isROR ELSE
    A AND B WHEN isAND ELSE
    A OR B WHEN isOR ELSE
    A XOR B WHEN isXOR ELSE
    "0" & A(7 DOWNTO 1) WHEN isRSHIFT ELSE
    A(6 DOWNTO 0) & "0" WHEN isLSHIFT ELSE
    Sum;
C <= A(0) WHEN isRSHIFT ELSE A(7) WHEN isLSHIFT ELSE Sum_C;
Z <= match(Result,"00000000");
S <= Result(7);
V <= A(7) XOR S WHEN isRSHIFT OR isLSHIFT ELSE Sum_V;
-----

-- Final output
-----

FlagsOut <=
    FlagsIn
    WHEN DiscardFlags ELSE

```

```

        C & Z & S & V & FlagsIn(3 DOWNT0 0)          WHEN FullFlags      ELSE
        FlagsIn(7) & Z & S & FlagsIn(4 DOWNT0 0);
    ALUout <= A WHEN DiscardResult ELSE Result;
END;
```

## B.11 ALU\_TB.vhd - ALU test bench

```

-- E80 ALU test bench
LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, ieee.numeric_std.ALL, work.support.ALL;
ENTITY ALU_TB IS END;
ARCHITECTURE a1 OF ALU_TB IS
    SIGNAL ALUinA : WORD := "11000010";
    SIGNAL ALUinB : WORD := "01111110";
    SIGNAL FlagsIn : WORD := "UUUUUUUU";
    SIGNAL ALUop : STD_LOGIC_VECTOR(3 DOWNT0 0) := "0000";
    SIGNAL ALUout, FlagsOut : WORD;
BEGIN
    ALUop <= STD_LOGIC_VECTOR(UNSIGNED(ALUop) + 1) AFTER 50 ps;
    ALU : ENTITY work.ALU PORT MAP(
        ALUop,
        ALUinA,
        ALUinB,
        FlagsIn,
        ALUout,
        FlagsOut);
END;
```

## B.12 FA8.vhd - 8-bit ripple carry full adder / subtractor

```

-----
-- E80 8-bit Full Adder
-- Performs textbook ripple-carry addition or subtraction
-----

-----
-- 1-bit full adder
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY FA IS PORT (
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    Cin : IN STD_LOGIC;
    S : OUT STD_LOGIC;
    Cout : OUT STD_LOGIC);
END;
ARCHITECTURE a1 OF FA IS
    SIGNAL X : STD_LOGIC;
BEGIN
    X <= A XOR B;
    S <= X XOR Cin;
    Cout <= (A AND B) OR (X AND Cin);
END;

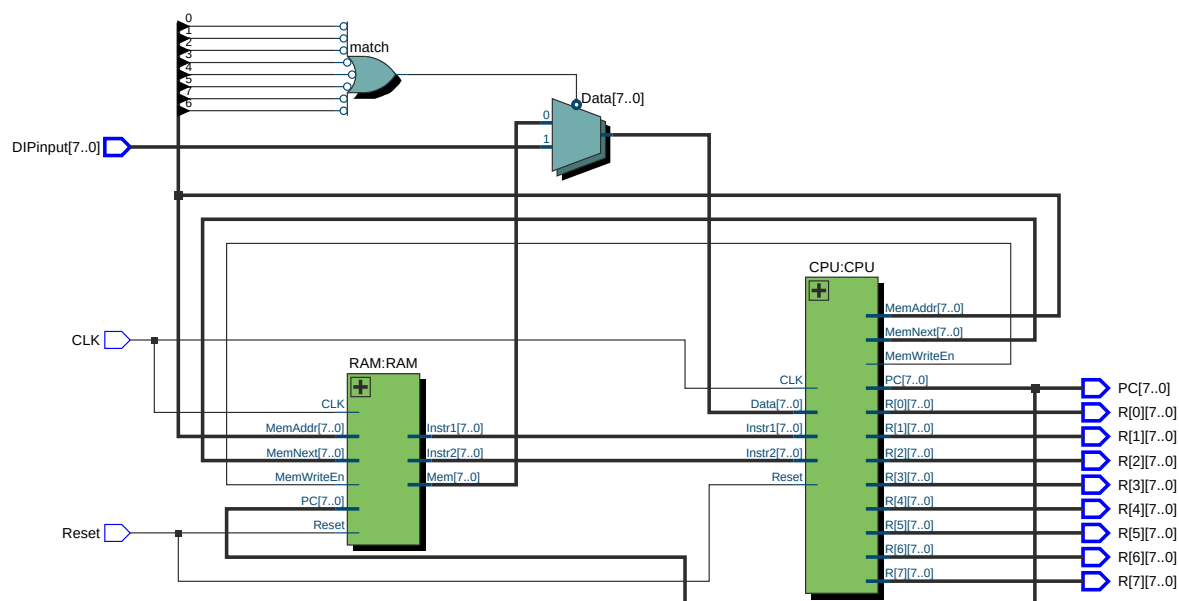
-----
-- 8-bit full adder
-----

LIBRARY ieee, work;
USE ieee.std_logic_1164.ALL, work.support.ALL;
ENTITY FA8 IS PORT (
    A : IN WORD;
    B : IN WORD;
    Sub : IN STD_LOGIC; -- 0=addition, 1=subtraction
    Sum : OUT WORD;
    Cout : OUT STD_LOGIC;
    V : OUT STD_LOGIC); -- overflow
END;
ARCHITECTURE a1 OF FA8 IS
```

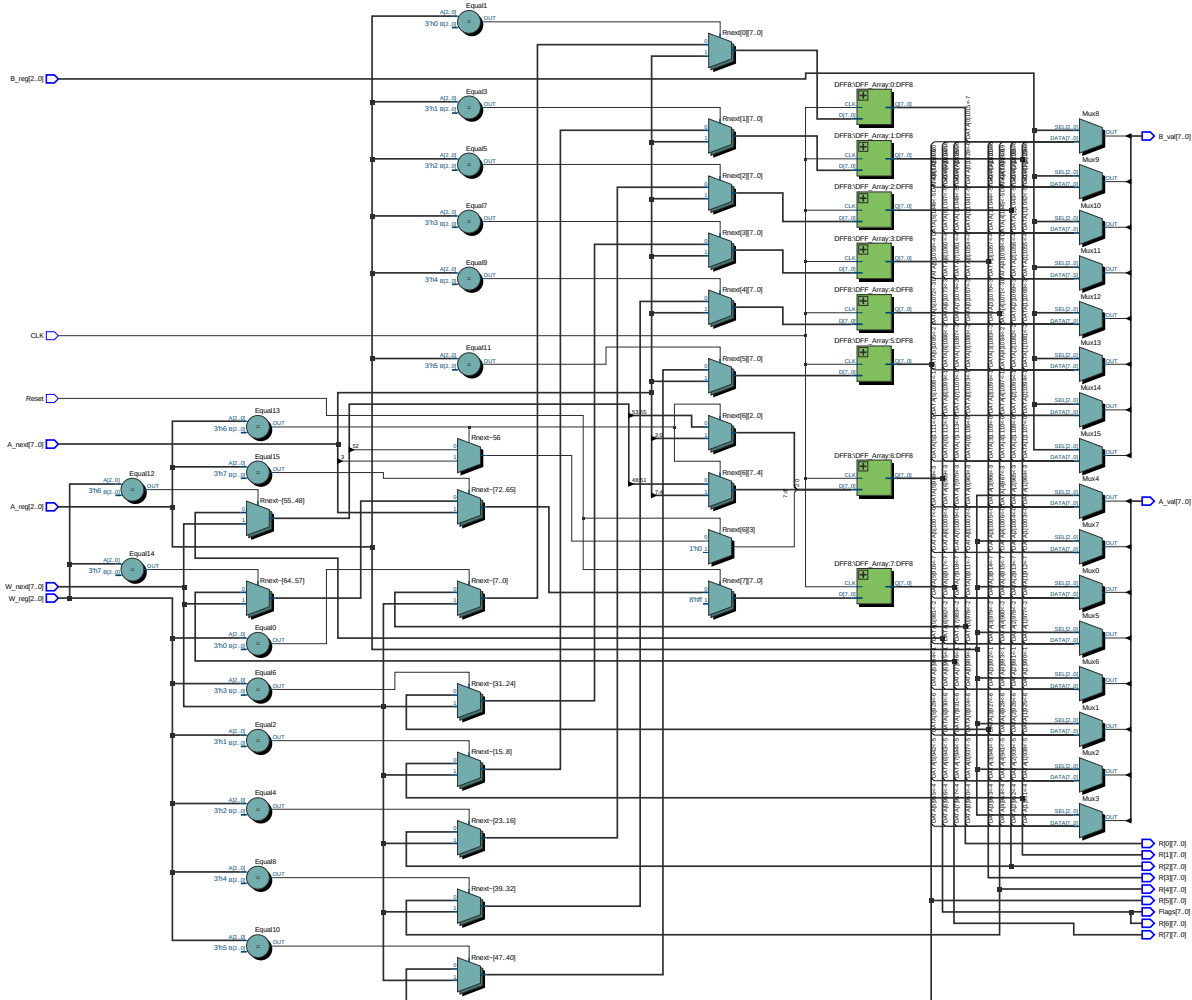


```
SIGNAL C : STD_LOGIC_VECTOR(8 DOWNT0 0); -- C(i) = CarryIn for bit i
BEGIN
-- Sum = A+(B XOR Sub)+C(0) = A + B + 0      = A+B (if Sub=0)
--                               A + NOT B + 1 = A-B (if Sub=1)
C(0) <= Sub;
FA_Array : FOR i IN 0 TO 7 GENERATE
    FA: ENTITY work.FA PORT MAP(
        A(i),
        B(i) XOR Sub,
        C(i),      -- Cin
        Sum(i),
        C(i+1)); -- Cout
    END GENERATE;
Cout <= C(8);
V <= C(8) XOR C(7);
END;
```

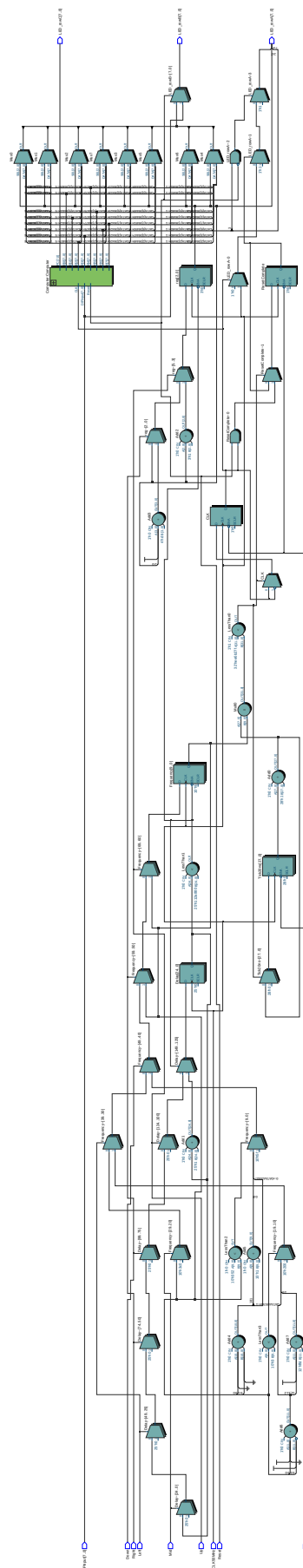
## Παράρτημα Γ: Κυκλωματικά διαγράμματα RTL απο Quartus



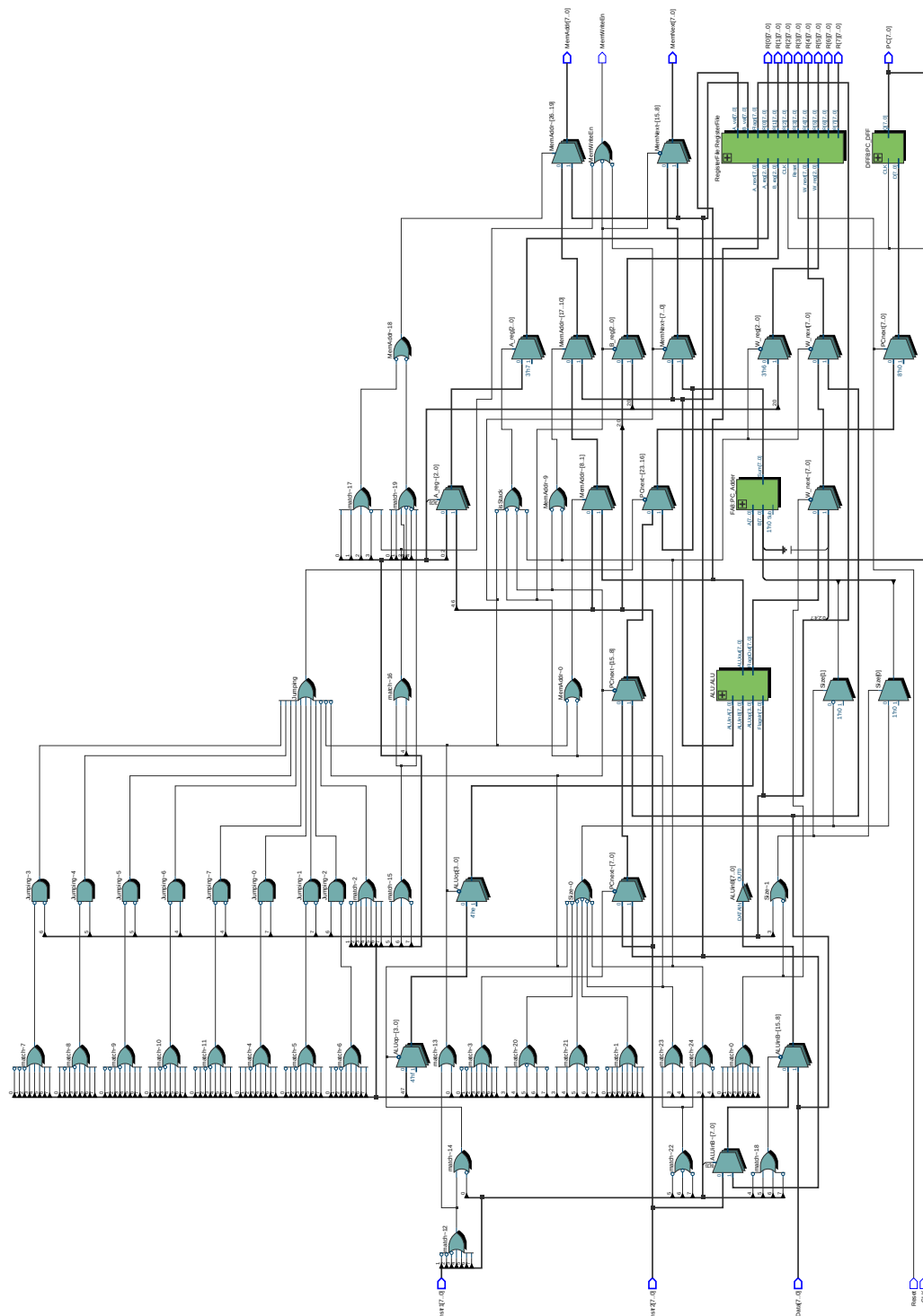
Εικόνα 67: Διάγραμμα Computer (RTL Viewer / Quartus)



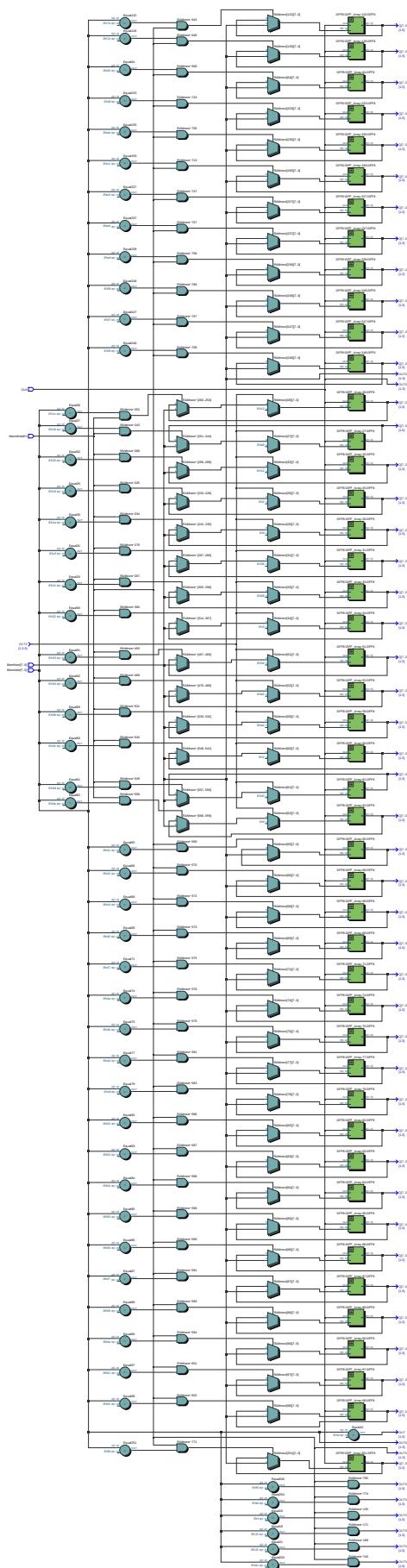
Εικόνα 68: Διάγραμμα RegisterFile (RTL Viewer / Quartus)



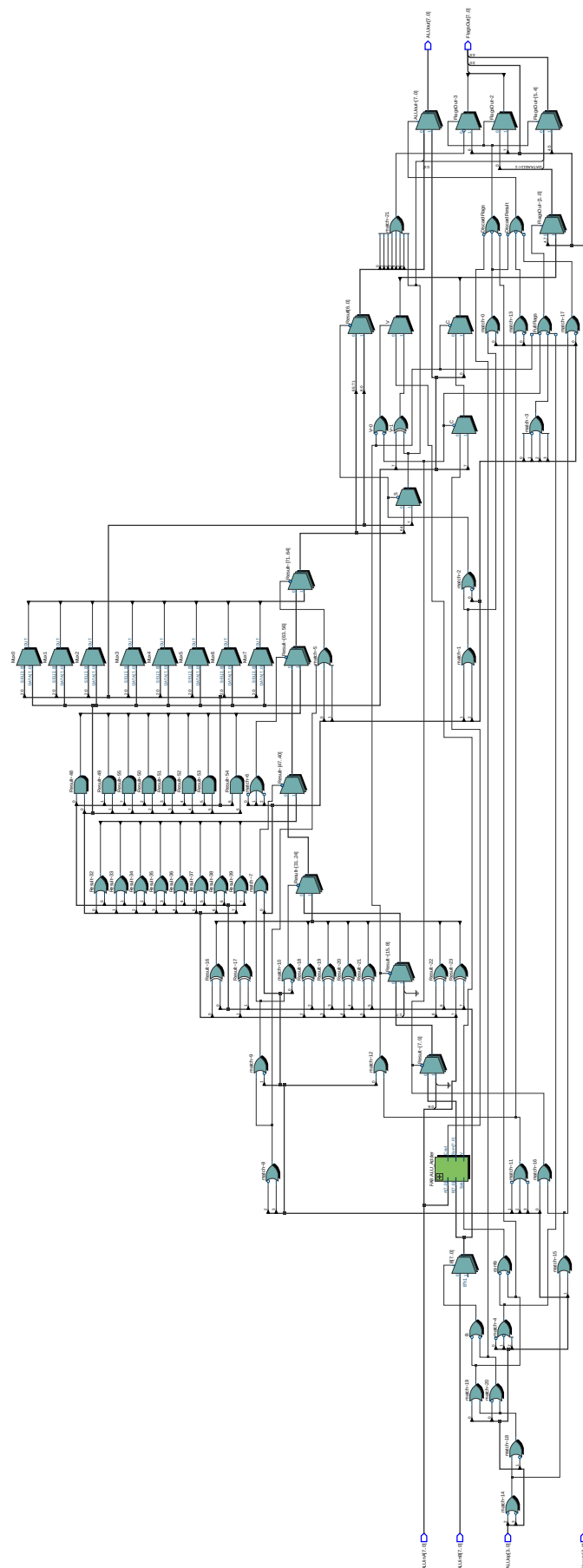
Εικόνα 69: Διάγραμμα FPGA (RTL Viewer / Quartus)



Εικόνα 70: Διάγραμμα CPU (RTL Viewer / Quartus)



Εικόνα 71: Διάγραμμα RAM (RTL Viewer / Quartus)



Εικόνα 72: Διάγραμμα ALU (RTL Viewer / Quartus)

## Παράρτημα Δ: Σενάρια και ρυθμίσεις GHDL/GTKWave

### Δ.1 g.bat - Βασικό αρχείο δεσμίδας GHDL-GTKWave

Το αρχείο g.bat περιέχει την διαδικασία προσομοίωσης με έλεγχο λαθών (πχ. μη-εγκατάσταση GHDL/GTKWave). Η κατασκευή του αρχείου αυτού ήταν αρκετά δύσκολη γιατί η τεκμηρίωση του GHDL δεν ήταν αρκετά προφανής και ο οδηγός “Quick Start” αναφέρεται σε μοναδικά αρχεία VHDL και όχι σε έργα πολλαπλών αρχείων όπως ο E80. Έγινε λοιπόν προσπάθεια για να απλοποιηθεί η απαιτούμενη διαδικασία από μέρους του χρήστη.

Το αρχείο μπορεί να προσομοιώσει μια αρχιτεκτονική, και να ορίσει τον μέγιστο χρόνο προσομοίωσης, σύμφωνα με τον τρόπο που χρησιμοποιείται από τα δοσμένα alu\_tb.bat και computer\_tb.bat. Σημειώνεται ότι στο g.bat η εκτέλεση του GTKWave γίνεται με χρήση της επιλογής --rcvar "hide\_sst on" για την απόκρυψη της μπάρας του δένδρου των σημάτων. Αυτό γίνεται για να δοθεί όσο το δυνατόν περισσότερο πλάτος για τις κυματομορφές, το οποίο είναι απαραίτητο για την μελέτη πολύπλοκων προγραμμάτων. Η προσθήκη σημάτων μπορεί να γίνει μέσω της λειτουργίας Search > Signal Search Tree οπότε η μπάρα είναι περιττή.

```
@echo off

echo E80 parametric GHDL-GTKWave simulation batch file

:ghdl_test
ghdl > NUL 2>&1
if %errorlevel% NEQ 9009 goto :gtkwave_test
echo Install GHDL and add ghdl\bin to your path
goto :error

:gtkwave_test
gtkwave -h > NUL 2>&1
if %errorlevel% NEQ 9009 goto :parameter1_test
echo Install GTKWave and add gtkwave\bin to your path
goto :error

:parameter1_test
if "%~1" NEQ "" goto :parameter2_test
echo Missing top unit on 1st parameter
goto :error

:parameter2_test
if "%~2" NEQ "" goto :exec
echo Missing duration on 2nd parameter
goto :error

:exec
echo -----
echo 1. Import and parse all VHDL files into the workspace :
echo ghdl -i --std=08 ..\VHDL\*.vhd
ghdl -i --std=08 ..\VHDL\*.vhd
if %errorlevel% NEQ 0 goto :error
echo -----
echo 2. Make the design with %1 as top unit :
echo ghdl -m --std=08 -Wno-hide %1
ghdl -m --std=08 -Wno-hide %1
if %errorlevel% NEQ 0 goto :error
echo -----
echo 3. Run (simulate) the design for %2 :
echo ghdl -r --std=08 %1 --stop-time=%2 --wave=%1.ghw
```



```
ghdl -r --std=08 %1 --stop-time=%2 --wave=%1.ghw
if %errorlevel% NEQ 0 goto :error
echo -----
echo 4. Opening GTKWave config %1.gtkw :
echo   gtkwave %1.gtkw
gtkwave %1.gtkw --rcvar "hide_sst on"
if %errorlevel% NEQ 0 (
echo -----
    echo 5. File not found; opening wave without config :
    echo   gtkwave %1.ghw
    gtkwave %1.ghw
)
echo -----

goto :end

:error
pause

:end
```

## Δ.2 alu\_tb.gtkw - GTKWave E80 ALU preset

Εντολή για χρήση του preset: g alu\_tb 800ps

```
[*]
[*] GTKWave Analyzer v3.3.100 (w)1999-2019 BSI
[*] Thu Feb 06 15:20:13 2025
[*]
[dumpfile] "alu_tb.ghw"
[dumpfile_mtime] "Thu Feb 06 15:19:00 2025"
[dumpfile_size] 2784
[savefile] "alu_tb.gtkw"
[timestart] 0
[size] 1861 1177
[pos] -1 -1
*-16.677013 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1
[treeopen] top.
[treeopen] top.alu_tb.
[treeopen] top.alu_tb.alu.
[sst_width] 43
[signals_width] 239
[sst_expanded] 0
[sst_vpaned_height] 357
@28
[color] 1
+{FlagsIn} #{top.alu_tb.flagsin[7:0]} top.alu_tb.flagsin[7] top.alu_tb.flagsin[6]
top.alu_tb.flagsin[5] top.alu_tb.flagsin[4] top.alu_tb.flagsin[3]
top.alu_tb.flagsin[2] top.alu_tb.flagsin[1] top.alu_tb.flagsin[0]
[color] 2
+{ALUinA (binary)} #{top.alu_tb.aluina[7:0]} top.alu_tb.aluina[7]
top.alu_tb.aluina[6] top.alu_tb.aluina[5] top.alu_tb.aluina[4] top.alu_tb.aluina[3]
top.alu_tb.aluina[2] top.alu_tb.aluina[1] top.alu_tb.aluina[0]
@24
[color] 2
+{ALUinA (unsigned)} #{top.alu_tb.aluina[7:0]} top.alu_tb.aluina[7]
top.alu_tb.aluina[6] top.alu_tb.aluina[5] top.alu_tb.aluina[4] top.alu_tb.aluina[3]
top.alu_tb.aluina[2] top.alu_tb.aluina[1] top.alu_tb.aluina[0]
@420
[color] 2
+{ALUinA (signed)} #{top.alu_tb.aluina[7:0]} top.alu_tb.aluina[7]
top.alu_tb.aluina[6] top.alu_tb.aluina[5] top.alu_tb.aluina[4] top.alu_tb.aluina[3]
top.alu_tb.aluina[2] top.alu_tb.aluina[1] top.alu_tb.aluina[0]
@28
[color] 4
+{ALUinB (binary)} #{top.alu_tb.aluinb[7:0]} top.alu_tb.aluinb[7]
top.alu_tb.aluinb[6] top.alu_tb.aluinb[5] top.alu_tb.aluinb[4] top.alu_tb.aluinb[3]
```

```
top.alu_tb.aluinb[2] top.alu_tb.aluinb[1] top.alu_tb.aluinb[0]
@24
[color] 4
+{ALUinB (unsigned)} #{top.alu_tb.aluinb[7:0]} top.alu_tb.aluinb[7]
top.alu_tb.aluinb[6] top.alu_tb.aluinb[5] top.alu_tb.aluinb[4] top.alu_tb.aluinb[3]
top.alu_tb.aluinb[2] top.alu_tb.aluinb[1] top.alu_tb.aluinb[0]
@420
[color] 4
+{ALUinB (signed)} #{top.alu_tb.aluinb[7:0]} top.alu_tb.aluinb[7]
top.alu_tb.aluinb[6] top.alu_tb.aluinb[5] top.alu_tb.aluinb[4] top.alu_tb.aluinb[3]
top.alu_tb.aluinb[2] top.alu_tb.aluinb[1] top.alu_tb.aluinb[0]
@28
[color] 1
+{ALUop} #{top.alu_tb.aluop[3:0]} top.alu_tb.aluop[3] top.alu_tb.aluop[2]
top.alu_tb.aluop[1] top.alu_tb.aluop[0]
[color] 6
+{ALUout (binary)} #{top.alu_tb.aluout[7:0]} top.alu_tb.aluout[7]
top.alu_tb.aluout[6] top.alu_tb.aluout[5] top.alu_tb.aluout[4] top.alu_tb.aluout[3]
top.alu_tb.aluout[2] top.alu_tb.aluout[1] top.alu_tb.aluout[0]
@24
[color] 6
+{ALUout (unsigned)} #{top.alu_tb.aluout[7:0]} top.alu_tb.aluout[7]
top.alu_tb.aluout[6] top.alu_tb.aluout[5] top.alu_tb.aluout[4] top.alu_tb.aluout[3]
top.alu_tb.aluout[2] top.alu_tb.aluout[1] top.alu_tb.aluout[0]
@420
[color] 6
+{ALUout (signed)} #{top.alu_tb.aluout[7:0]} top.alu_tb.aluout[7]
top.alu_tb.aluout[6] top.alu_tb.aluout[5] top.alu_tb.aluout[4] top.alu_tb.aluout[3]
top.alu_tb.aluout[2] top.alu_tb.aluout[1] top.alu_tb.aluout[0]
@28
+{Carry} top.alu_tb.flagsout[7]
+{Zero} top.alu_tb.flagsout[6]
+{Sign} top.alu_tb.flagsout[5]
+{Overflow} top.alu_tb.flagsout[4]
```

### Δ.3 computer\_tb.gtkw - GTKWave E80 Computer preset

Εντολή για χρήση του preset: g computer\_tb 100ns

```
[*]
[*] GTKWave Analyzer v3.3.100 (w)1999-2019 BSI
[*] Mon May 05 12:10:30 2025
[*]
[dumpfile] "computer_tb.ghw"
[dumpfile_mtime] "Mon May 05 12:03:14 2025"
[dumpfile_size] 72698
[savefile] "computer_tb.gtkw"
[timestart] 0
[size] 1861 1177
[pos] -1 -1
*-19.258013 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1
[treeopen] top.
[treeopen] top.computer_tb.
[treeopen] top.computer_tb.computer.
[treeopen] top.computer_tb.computer.ram.ram.
[treeopen] top.computer_tb.r.
[sst_width] 43
[signals_width] 169
[sst_expanded] 0
[sst_vpaned_height] 357
@24
[color] 1
top.computer_tb.clk
@4
[color] 4
+{PC} #{top.computer_tb.computer.pc[7:0]} top.computer_tb.computer.pc[7]
top.computer_tb.computer.pc[6] top.computer_tb.computer.pc[5]
```

```

top.computer_tb.computer.pc[4] top.computer_tb.computer.pc[3]
top.computer_tb.computer.pc[2] top.computer_tb.computer.pc[1]
top.computer_tb.computer.pc[0]
@22
+{Instr1} #{top.computer_tb.computer.instr1[7:0]}
top.computer_tb.computer.instr1[7] top.computer_tb.computer.instr1[6]
top.computer_tb.computer.instr1[5] top.computer_tb.computer.instr1[4]
top.computer_tb.computer.instr1[3] top.computer_tb.computer.instr1[2]
top.computer_tb.computer.instr1[1] top.computer_tb.computer.instr1[0]
+{Instr2} #{top.computer_tb.computer.instr2[7:0]}
top.computer_tb.computer.instr2[7] top.computer_tb.computer.instr2[6]
top.computer_tb.computer.instr2[5] top.computer_tb.computer.instr2[4]
top.computer_tb.computer.instr2[3] top.computer_tb.computer.instr2[2]
top.computer_tb.computer.instr2[1] top.computer_tb.computer.instr2[0]
@28
+{Carry} top.computer_tb.r[6][7]
+{Zero} top.computer_tb.r[6][6]
+{Sign} top.computer_tb.r[6][5]
+{Overflow} top.computer_tb.r[6][4]
+{Halt} top.computer_tb.r[6][3]
@24
[color] 6
+{R0} #{top.computer_tb.r[0][7:0]} top.computer_tb.r[0][7] top.computer_tb.r[0][6]
top.computer_tb.r[0][5] top.computer_tb.r[0][4] top.computer_tb.r[0][3]
top.computer_tb.r[0][2] top.computer_tb.r[0][1] top.computer_tb.r[0][0]
[color] 6
+{R1} #{top.computer_tb.r[1][7:0]} top.computer_tb.r[1][7] top.computer_tb.r[1][6]
top.computer_tb.r[1][5] top.computer_tb.r[1][4] top.computer_tb.r[1][3]
top.computer_tb.r[1][2] top.computer_tb.r[1][1] top.computer_tb.r[1][0]
[color] 6
+{R2} #{top.computer_tb.r[2][7:0]} top.computer_tb.r[2][7] top.computer_tb.r[2][6]
top.computer_tb.r[2][5] top.computer_tb.r[2][4] top.computer_tb.r[2][3]
top.computer_tb.r[2][2] top.computer_tb.r[2][1] top.computer_tb.r[2][0]
[color] 6
+{R3} #{top.computer_tb.r[3][7:0]} top.computer_tb.r[3][7] top.computer_tb.r[3][6]
top.computer_tb.r[3][5] top.computer_tb.r[3][4] top.computer_tb.r[3][3]
top.computer_tb.r[3][2] top.computer_tb.r[3][1] top.computer_tb.r[3][0]
[color] 6
+{R4} #{top.computer_tb.r[4][7:0]} top.computer_tb.r[4][7] top.computer_tb.r[4][6]
top.computer_tb.r[4][5] top.computer_tb.r[4][4] top.computer_tb.r[4][3]
top.computer_tb.r[4][2] top.computer_tb.r[4][1] top.computer_tb.r[4][0]
[color] 6
+{R5} #{top.computer_tb.r[5][7:0]} top.computer_tb.r[5][7] top.computer_tb.r[5][6]
top.computer_tb.r[5][5] top.computer_tb.r[5][4] top.computer_tb.r[5][3]
top.computer_tb.r[5][2] top.computer_tb.r[5][1] top.computer_tb.r[5][0]
[color] 6
+{SP} #{top.computer_tb.r[7][7:0]} top.computer_tb.r[7][7] top.computer_tb.r[7][6]
top.computer_tb.r[7][5] top.computer_tb.r[7][4] top.computer_tb.r[7][3]
top.computer_tb.r[7][2] top.computer_tb.r[7][1] top.computer_tb.r[7][0]
[color] 2
+{RAM[251]} #{top.computer_tb.computer.ram.ram[251][7:0]}
top.computer_tb.computer.ram.ram[251][7] top.computer_tb.computer.ram.ram[251][6]
top.computer_tb.computer.ram.ram[251][5] top.computer_tb.computer.ram.ram[251][4]
top.computer_tb.computer.ram.ram[251][3] top.computer_tb.computer.ram.ram[251][2]
top.computer_tb.computer.ram.ram[251][1] top.computer_tb.computer.ram.ram[251][0]
[color] 2
+{RAM[252]} #{top.computer_tb.computer.ram.ram[252][7:0]}
top.computer_tb.computer.ram.ram[252][7] top.computer_tb.computer.ram.ram[252][6]
top.computer_tb.computer.ram.ram[252][5] top.computer_tb.computer.ram.ram[252][4]
top.computer_tb.computer.ram.ram[252][3] top.computer_tb.computer.ram.ram[252][2]
top.computer_tb.computer.ram.ram[252][1] top.computer_tb.computer.ram.ram[252][0]
[color] 2
+{RAM[253]} #{top.computer_tb.computer.ram.ram[253][7:0]}
top.computer_tb.computer.ram.ram[253][7] top.computer_tb.computer.ram.ram[253][6]
top.computer_tb.computer.ram.ram[253][5] top.computer_tb.computer.ram.ram[253][4]
top.computer_tb.computer.ram.ram[253][3] top.computer_tb.computer.ram.ram[253][2]
top.computer_tb.computer.ram.ram[253][1] top.computer_tb.computer.ram.ram[253][0]
[color] 2

```

```
+{RAM[254]} #{top.computer_tb.computer.ram.ram[254][7:0]}
top.computer_tb.computer.ram.ram[254][7] top.computer_tb.computer.ram.ram[254][6]
top.computer_tb.computer.ram.ram[254][5] top.computer_tb.computer.ram.ram[254][4]
top.computer_tb.computer.ram.ram[254][3] top.computer_tb.computer.ram.ram[254][2]
top.computer_tb.computer.ram.ram[254][1] top.computer_tb.computer.ram.ram[254][0]
@28
+{DIPinput} #{top.computer_tb.dipinput[7:0]} top.computer_tb.dipinput[7]
top.computer_tb.dipinput[6] top.computer_tb.dipinput[5] top.computer_tb.dipinput[4]
top.computer_tb.dipinput[3] top.computer_tb.dipinput[2] top.computer_tb.dipinput[1]
top.computer_tb.dipinput[0]
@c00200
[pattern_trace] 1
[pattern_trace] 0
```

#### Δ.4 CPU\_DSDi1\_synthesis.bat - GHDL all-in-one synthesis

```
@echo off

echo E80 GHDL synthesis script for the FPGA implementation

echo -----
echo 1. Import and parse all VHDL files into the workspace :
echo   ghdl -i --std=08 ..\VHDL\*.vhd
ghdl -i --std=08 ..\VHDL\*.vhd
if %errorlevel% NEQ 0 pause
echo -----
echo 2. Make the design with FPGA as top unit :
echo   ghdl -m --std=08 -Wno-hide FPGA
ghdl -m --std=08 -Wno-hide FPGA
if %errorlevel% NEQ 0 pause
echo -----
echo 3. Synthesise the design into a single vhd file:
echo   ghdl --synth --std=08 FPGA > FPGA.GHDL.vhdl
ghdl --synth --std=08 FPGA > FPGA.GHDL.vhdl
@if %errorlevel% NEQ 0 pause
echo -----
echo Done. You can add FPGA.GHDL.vhdl into a project with
echo support.vhd and firmware.vhd
```

## Παράρτημα Ε: Σενάρια και ρυθμίσεις ModelSim

### Ε.1 a.do - σενάριο προσομοίωσης της ALU

```
quit -sim
wave zoom full
.main clear
vsim work.alu_tb -quiet
onerror {resume}

quietly WaveActivateNextPane {} 0
add wave -noupdate -radix binary /alu_tb/FlagsIn
add wave -noupdate -label {ALUinA (binary)} -radix binary /alu_tb/ALUinA
add wave -noupdate -label {ALUinA (unsigned)} -radix unsigned /alu_tb/ALUinA
add wave -noupdate -label {ALUinA (signed)} -radix decimal /alu_tb/ALUinA
add wave -noupdate -label {ALUinB (binary)} -radix binary /alu_tb/ALUinB
add wave -noupdate -label {ALUinB (unsigned)} -radix unsigned /alu_tb/ALUinB
add wave -noupdate -label {ALUinB (signed)} -radix decimal /alu_tb/ALUinB
add wave -noupdate /alu_tb/ALUop
add wave -noupdate -label {ALUout (binary)} -radix binary /alu_tb/ALUout
add wave -noupdate -label {ALUout (unsigned)} -radix unsigned /alu_tb/ALUout
add wave -noupdate -label {ALUout (signed)} -radix decimal /alu_tb/ALUout
add wave -noupdate -label Carry /alu_tb/FlagsOut(7)
add wave -noupdate -label Zero /alu_tb/FlagsOut(6)
add wave -noupdate -label Sign /alu_tb/FlagsOut(5)
add wave -noupdate -label Overflow /alu_tb/FlagsOut(4)

TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {0 ps} 0}
quietly wave cursor active 0
configure wave -namecolwidth 137
configure wave -valuecolwidth 56
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps

run 800
wave zoom full
```

### Ε.2 c.do - σενάριο προσομοίωσης του E80 Computer

```
quit -sim
wave zoom full
.main clear
quietly project compileall
vsim -quiet work.computer_tb
onerror {resume}
quietly WaveActivateNextPane {} 0

add wave -noupdate /computer_tb/CLK
add wave -radix unsigned /computer_tb/PC
add wave -noupdate -radix hexadecimal /computer_tb/Computer/Instr1
add wave -noupdate -radix hexadecimal /computer_tb/Computer/Instr2
add wave -noupdate /computer_tb/Computer/CPU/Carry
add wave -noupdate /computer_tb/Computer/CPU/Zero
add wave -noupdate /computer_tb/Computer/CPU/Sign
add wave -noupdate /computer_tb/Computer/CPU/Overflow
add wave -noupdate /computer_tb/Halt
```

```

add wave -noupdate -divider {Registers}
add wave -noupdate -radix unsigned /computer_tb/R(0)
add wave -noupdate -radix unsigned /computer_tb/R(1)
add wave -noupdate -radix unsigned /computer_tb/R(2)
add wave -noupdate -radix unsigned /computer_tb/R(3)
add wave -noupdate -radix unsigned /computer_tb/R(4)
add wave -noupdate -radix unsigned /computer_tb/R(5)
add wave -noupdate -label SP -radix unsigned /computer_tb/R(7)

add wave -noupdate -divider {RAM/Input}
add wave -noupdate -radix unsigned /computer_tb/Computer/RAM/RAM(251)
add wave -noupdate -radix unsigned /computer_tb/Computer/RAM/RAM(252)
add wave -noupdate -radix unsigned /computer_tb/Computer/RAM/RAM(253)
add wave -noupdate -radix unsigned /computer_tb/Computer/RAM/RAM(254)
add wave -noupdate -radix hexadecimal /computer_tb/Computer/RAM/RAM
add wave -noupdate /computer_tb/DIPinput

TreeUpdate [SetDefaultTree]
configure wave -namecolwidth 104
configure wave -valuecolwidth 64
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -rowmargin 4
configure wave -gridperiod 100
configure wave -timelineunits ps
quietly set PrefSource(OpenOnBreak) 0

when {/computer_tb/Halt=='1'} {
    stop
    echo "Halt detected"
}
run 100ns
add mem /computer_tb/Computer/RAM/RAM -a hexadecimal -d symbolic -wo 16
# zoom is fine-tuned to fit 3 digits in a cycle
quietly wave zoom range 0 4400
quietly view wave

```

### E.3 LayoutE80.reg - ρυθμίσεις παραθύρων ModelSim

Windows Registry Editor Version 5.00

```

[HKEY_CURRENT_USER\SOFTWARE\Model Technology Incorporated\ModelSim]
"LayoutForDefault"="E80"
"LayoutForLoad"="E80"
"LayoutForLoadCov"="E80"
"LayoutV5%2EE80"="vertical {{{{{.main_pane.process {-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na {{ .main_pane.locals {-height 600 -hide 1 -minsize
50 -stretch always -width 600} 0 na {{ .main_pane.details {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 1 na {{ {{-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na paned {{.main_pane.wave {-height 556 -hide 0
-minsize 50 -stretch always -width 1440} 533 na {{ .main_pane.memdata {-height 600
-hide 1 -minsize 50 -stretch always -width 600} 0 na {{ .main_pane.source1 {-height
600 -hide 1 -minsize 50 -width 600} 0 na {{ .main_pane.source2 {-height 600 -hide 1
-minsize 50 -width 600} 0 na {{ .main_pane.source3 {-height 600 -hide 1 -minsize 50
-width 600} 0 na {{ .main_pane.source4 {-height 600 -hide 1 -minsize 50 -width 600}
0 na {{ .main_pane.source5 {-height 600 -hide 1 -minsize 50 -width 600} 0 na
{{ .main_pane.source {-height 600 -hide 1 -minsize 50 -stretch always -width 600} 0
na {{ .main_pane.dataflow {-height 600 -hide 1 -minsize 50 -stretch always -width
600} 0 na {{ .main_pane.list {-height 600 -hide 1 -minsize 50 -stretch always
-width 600} 0 na {{ .main_pane.fsmview {-height 600 -hide 1 -minsize 50 -stretch
always -width 600} 0 na {{ .main_pane.msgviewer {-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na {{ .main_pane.triageviewer {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 0 na {{ .main_pane.atv {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 0 na {{ .main_pane.schematic {-height 600
-hide 1 -minsize 50 -stretch always -width 600} 0 na {{ .main_pane.tracker {-height
600 -hide 1 -minsize 50 -stretch always -width 600} 0 na {{ .main_pane.browser {-
height 600 -hide 1 -minsize 50 -stretch always -width 600} 0 na
{{ .main_pane.canalysis {-height 600 -hide 1 -minsize 50 -stretch always -width

```

```
600} 0 na {} .main_pane.duranked {-height 600 -hide 1 -minsize 50 -stretch always
-width 600} 0 na {} .main_pane.watch {-height 600 -hide 1 -minsize 50 -stretch
always -width 600} 0 na {} .main_pane.ranked {-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na {} .main_pane.calltree {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 0 na {} .main_pane.structural {-height 600
-hide 1 -minsize 50 -stretch always -width 600} 0 na {} .main_pane.profiledetails
{-height 600 -hide 1 -minsize 50 -stretch always -width 600} 0 na
{} .main_pane.assertions {-height 600 -hide 1 -minsize 50 -stretch always -width
600} 0 na {} .main_pane.fcovers {-height 600 -hide 1 -minsize 50 -stretch always
-width 600} 0 na {} .main_pane.covergroups {-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na {} .main_pane.classtree {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 0 na {} .main_pane.classgraph {-height 600
-hide 1 -minsize 50 -stretch always -width 600} 0 na {} .main_pane.trender {-height
600 -hide 1 -minsize 50 -stretch always -width 600} 0 na {} .main_pane.capacity {-
height 600 -hide 1 -minsize 50 -stretch always -width 600} 0 na
{} .main_pane.memdata1 {-height 600 -hide 1 -minsize 50 -width 600} 556 na {} {-
height 556 -hide 0 -minsize 50 -stretch always -width 1440} 1440 na tabbed} {-
height 556 -hide 0 -minsize 50 -stretch always -width 1440} 556 na paned
{{.main_pane.library {-height 215 -hide 0 -minsize 50 -stretch always -width 317}
215 na {} .main_pane.project {-height 215 -hide 0 -minsize 50 -stretch always
-width 317} 256 na {} .main_pane.memory {-height 600 -hide 1 -minsize 50 -stretch
always -width 600} 0 na {} .main_pane.structure {-height 600 -hide 1 -minsize 50
-stretch always -width 600} 0 na {} .main_pane.files {-height 600 -hide 1 -minsize
50 -stretch always -width 600} 0 na {} .main_pane.fsmlist {-height 600 -hide 1
-minsize 50 -stretch always -width 600} 0 na {} .main_pane.powerstatelist {-height
600 -hide 1 -minsize 50 -stretch always -width 600} 0 na {} .main_pane.stackview {-
height 600 -hide 1 -minsize 50 -stretch always -width 600} 0 na
{} .main_pane.instance {-height 600 -hide 1 -minsize 50 -stretch always -width 600}
238 na {} {-height 238 -hide 0 -minsize 50 -stretch always -width 317} 317 na
tabbed {.main_pane.objects {-height 238 -hide 0 -minsize 50 -width 336} 238 na {}
{-height 238 -hide 0 -minsize 50 -stretch always -width 336} 659 na paned
{.main_pane.transcript {-height 238 -hide 0 -minsize 50 -stretch always -width 775}
238 na {} {-height 238 -hide 0 -minsize 50 -width 775} 1440 na tabbed} {-height
238 -hide 0 -minsize 50 -width 1440} 800 na paned} {-height 800 -hide 0 -minsize 50
-stretch always -width 1440} 1440 na paned} {-height 800 -hide 0 -minsize 200
-stretch always -width 1440} 800 na paned}"
```

## E.4 E80.mpf - αρχείο ρυθμίσεων έργου

```

; -----
; E80 Computer - ModelSim Project File
; -----

[Library]
std = $MODEL_TECH/./std
ieee = $MODEL_TECH/./ieee
work = work

[vcom]
VHDL93 = 2008
Explicit = 1

[vsim]
Resolution = ps
UserTimeUnit = default
RunLength = 1 ns
IterationLimit = 5000
BreakOnAssertion = 2
DefaultRadix = symbolic
TranscriptFile = transcript
PathSeparator = /
DatasetSeparator = :
UnbufferedOutput = 0
ConcurrentFileLimit = 40

[Project]
Project_Version = 6
Project_DefaultLib = work

```



```

Project_SortMethod = unused
Project_Files_Count = 11
Project_File_0 = ../VHDL/FA8.vhd
Project_File_P_0 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort 0 compile_to work
compile_order 2 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_File_1 = ../VHDL/ALU_TB.vhd
Project_File_P_1 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 cover_excludedefault 0 vhdl_vital 0
vhdl_warn1 1 vhdl_showsource 0 vhdl_explicit 1 vhdl_warn2 1 vhdl_0InOptions {}
cover_covercells 0 vhdl_warn3 1 vhdl_options {} cover_optlevel 3 voptflow 1
vhdl_warn4 1 ood 0 toggle - vhdl_warn5 1 compile_to work cover_noshort 0
compile_order 4 dont_compile 0 cover_nosub 0 vhdl_use93 2008
Project_File_2 = ../VHDL/Computer.vhd
Project_File_P_2 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 cover_excludedefault 0 vhdl_vital 0
vhdl_warn1 1 vhdl_showsource 0 vhdl_explicit 1 vhdl_warn2 1 vhdl_0InOptions {}
cover_covercells 0 vhdl_warn3 1 vhdl_options {} cover_optlevel 3 voptflow 1
vhdl_warn4 1 ood 0 toggle - vhdl_warn5 1 compile_to work cover_noshort 0
compile_order 9 dont_compile 0 cover_nosub 0 vhdl_use93 2008
Project_File_3 = ../VHDL/RAM.vhd
Project_File_P_3 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 cover_excludedefault 0 vhdl_vital 0
vhdl_warn1 1 vhdl_showsource 0 vhdl_explicit 1 vhdl_warn2 1 vhdl_0InOptions {}
cover_covercells 0 vhdl_warn3 1 vhdl_options {} cover_optlevel 3 voptflow 1
vhdl_warn4 1 ood 0 toggle - vhdl_warn5 1 compile_to work cover_noshort 0
compile_order 7 dont_compile 0 cover_nosub 0 vhdl_use93 2008
Project_File_4 = ../VHDL/DFE8.vhd
Project_File_P_4 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort - ood 0 compile_to work
compile_order 5 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_File_5 = ../VHDL/CPU.vhd
Project_File_P_5 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 cover_excludedefault 0 vhdl_vital 0
vhdl_warn1 1 vhdl_showsource 0 vhdl_explicit 1 vhdl_warn2 1 vhdl_0InOptions {}
cover_covercells 0 vhdl_warn3 1 vhdl_options {} cover_optlevel 3 voptflow 1
vhdl_warn4 1 ood 0 toggle - vhdl_warn5 1 compile_to work cover_noshort 0
compile_order 8 dont_compile 0 cover_nosub 0 vhdl_use93 2008
Project_File_6 = ../VHDL/Computer_TB.vhd
Project_File_P_6 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort 0 compile_to work
compile_order 10 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_File_7 = ../VHDL/RegisterFile.vhd
Project_File_P_7 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 cover_excludedefault 0 vhdl_vital 0
vhdl_warn1 1 vhdl_showsource 0 vhdl_explicit 1 vhdl_warn2 1 vhdl_0InOptions {}
cover_covercells 0 vhdl_warn3 1 vhdl_options {} cover_optlevel 3 voptflow 1
vhdl_warn4 1 ood 0 toggle - vhdl_warn5 1 compile_to work cover_noshort 0
compile_order 6 dont_compile 0 cover_nosub 0 vhdl_use93 2008
Project_File_8 = ../VHDL/Support.vhd
Project_File_P_8 = vhdl novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0

```

```
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort 0 compile_to work
compile_order 0 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_File_9 = ../VHDL/Firmware.vhd
Project_File_P_9 = vhdl_novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort 0 compile_to work
compile_order 1 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_File_10 = ../VHDL/ALU.vhd
Project_File_P_10 = vhdl_novitalcheck 0 file_type vhdl group_id 0 cover_nofec 0
vhdl_nodebug 0 vhdl_1164 1 vhdl_noload 0 vhdl_synth 0 vhdl_enable0In 0 folder {Top
Level} last_compile 0 vhdl_disableopt 0 vhdl_vital 0 cover_excludedefault 0
vhdl_warn1 1 vhdl_warn2 1 vhdl_explicit 1 vhdl_showsource 0 vhdl_warn3 1
cover_covercells 0 vhdl_0InOptions {} vhdl_warn4 1 voptflow 1 cover_optlevel 3
vhdl_options {} vhdl_warn5 1 toggle - ood 0 cover_noshort 0 compile_to work
compile_order 3 cover_nosub 0 dont_compile 0 vhdl_use93 2008
Project_Sim_Count = 0
Project_Folder_Count = 0
Echo_Compile_Output = 0
Save_Compile_Report = 0
Project_Opt_Count = 0
ForceSoftPaths = 0
Project_Major_Version = 2020
Project_Minor_Version = 1
```

## Παράρτημα ΣΤ: Αρχεία ρυθμίσεων Quartus

### ΣΤ.1 E80.qsf - ρυθμίσεις και αναθέσεις ακροδεκτών

```
# ----- #
# E80 Computer - Settings and Pin Assignments for the DSD-il
# ----- #

set_global_assignment -name FAMILY "Cyclone IV E"
set_global_assignment -name DEVICE EP4CE6E22C8
set_global_assignment -name TOP_LEVEL_ENTITY FPGA
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 20.1.1
set_global_assignment -name PROJECT_CREATION_TIME_DATE "22:18:50 NOVEMBER 26, 2024"
set_global_assignment -name LAST_QUARTUS_VERSION "20.1.1 Lite Edition"
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 1
set_global_assignment -name NOMINAL_CORE_SUPPLY_VOLTAGE 1.2V
set_global_assignment -name EDA_SIMULATION_TOOL "ModelSim-Altera (VHDL)"
set_global_assignment -name EDA_TIME_SCALE "1 ps" -section_id eda_simulation
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT VHDL -section_id eda_simulation
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id
eda_board_design_timing
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id
eda_board_design_symbol
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id
eda_board_design_signal_integrity
set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST OFF -section_id
eda_board_design_boundary_scan
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200
LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name VHDL_INPUT_VERSION VHDL_2008
set_global_assignment -name VHDL_SHOW_LMF_MAPPING_MESSAGES OFF
set_global_assignment -name ENABLE_OCT_DONE OFF
set_global_assignment -name USE_CONFIGURATION_DEVICE OFF
set_global_assignment -name GENERATE_RBF_FILE ON
set_global_assignment -name CRC_ERROR_OPEN_DRAIN OFF
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
set_global_assignment -name RESERVE_ALL_UNUSED_PINS_WEAK_PULLUP "AS OUTPUT DRIVING
GROUND"
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTTL"
set_global_assignment -name OUTPUT_IO_TIMING_NEAR_END_VMEAS "HALF VCCIO" -rise
set_global_assignment -name OUTPUT_IO_TIMING_NEAR_END_VMEAS "HALF VCCIO" -fall
set_global_assignment -name OUTPUT_IO_TIMING_FAR_END_VMEAS "HALF SIGNAL SWING"
-rise
set_global_assignment -name OUTPUT_IO_TIMING_FAR_END_VMEAS "HALF SIGNAL SWING"
-fall
set_global_assignment -name NUM_PARALLEL_PROCESSORS 3
set_global_assignment -name TIMING_ANALYZER_MULTICORNER_ANALYSIS ON
set_global_assignment -name SMART_RECOMPILE ON
set_global_assignment -name EDA_RUN_TOOL_AUTOMATICALLY OFF -section_id
eda_simulation
set_global_assignment -name CYCLONEII_OPTIMIZATION_TECHNIQUE SPEED
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC ON
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION ON
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING ON
set_global_assignment -name ROUTER_LCELL_INSERTION_AND_LOGIC_DUPLICATION ON
set_global_assignment -name ROUTER_TIMING_OPTIMIZATION_LEVEL_MAXIMUM
set_global_assignment -name QII_AUTO_PACKED_REGISTERS NORMAL
set_global_assignment -name ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP ON
set_global_assignment -name ALLOW_REGISTER_RETIMING ON
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -section_id Top
```

```

set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
set_global_assignment -name FITTER_EFFORT "AUTO FIT"
set_global_assignment -name ENABLE_CONFIGURATION_PINS OFF
set_global_assignment -name ENABLE_BOOT_SEL_PIN OFF
set_global_assignment -name CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS
REGULAR IO"
set_global_assignment -name SDC_FILE E80.out.sdc
set_global_assignment -name VHDL_FILE ../VHDL/FPGA.vhd
set_global_assignment -name VHDL_FILE ../VHDL/Support.vhd
set_global_assignment -name VHDL_FILE ../VHDL/Computer.vhd
set_global_assignment -name VHDL_FILE ../VHDL/RAM.vhd
set_global_assignment -name VHDL_FILE ../VHDL/Firmware.vhd
set_global_assignment -name VHDL_FILE ../VHDL/CPU.vhd
set_global_assignment -name VHDL_FILE ../VHDL/RegisterFile.vhd
set_global_assignment -name VHDL_FILE ../VHDL/DFF8.vhd
set_global_assignment -name VHDL_FILE ../VHDL/ALU.vhd
set_global_assignment -name VHDL_FILE ../VHDL/FA8.vhd

set_location_assignment PIN_23 -to CLK50MHz

set_location_assignment PIN_85 -to Reset
set_location_assignment PIN_87 -to Up
set_location_assignment PIN_89 -to Down
set_location_assignment PIN_91 -to Left
set_location_assignment PIN_99 -to Right
set_location_assignment PIN_101 -to Mid
set_location_assignment PIN_104 -to Pause

set_location_assignment PIN_54 -to DIPinput[0]
set_location_assignment PIN_58 -to DIPinput[1]
set_location_assignment PIN_60 -to DIPinput[2]
set_location_assignment PIN_65 -to DIPinput[3]
set_location_assignment PIN_67 -to DIPinput[4]
set_location_assignment PIN_69 -to DIPinput[5]
set_location_assignment PIN_71 -to DIPinput[6]
set_location_assignment PIN_73 -to DIPinput[7]

set_location_assignment PIN_110 -to LED_rowA[7] # Carry
set_location_assignment PIN_111 -to LED_rowA[6] # Zero
set_location_assignment PIN_112 -to LED_rowA[5] # Sign
set_location_assignment PIN_113 -to LED_rowA[4] # Overflow
set_location_assignment PIN_114 -to LED_rowA[3] # Register address [2]
set_location_assignment PIN_115 -to LED_rowA[2] # Register address [1]
set_location_assignment PIN_119 -to LED_rowA[1] # Register address [0]
set_location_assignment PIN_120 -to LED_rowA[0] # CLK

set_location_assignment PIN_121 -to LED_rowB[7] # Register value [7]
set_location_assignment PIN_124 -to LED_rowB[6] # Register value [6]
set_location_assignment PIN_125 -to LED_rowB[5] # Register value [5]
set_location_assignment PIN_126 -to LED_rowB[4] # Register value [4]
set_location_assignment PIN_127 -to LED_rowB[3] # Register value [3]
set_location_assignment PIN_128 -to LED_rowB[2] # Register value [2]
set_location_assignment PIN_129 -to LED_rowB[1] # Register value [1]
set_location_assignment PIN_132 -to LED_rowB[0] # Register value [0]

set_location_assignment PIN_133 -to LED_rowC[7] # Program counter [7]
set_location_assignment PIN_135 -to LED_rowC[6] # Program counter [6]
set_location_assignment PIN_136 -to LED_rowC[5] # Program counter [5]
set_location_assignment PIN_137 -to LED_rowC[4] # Program counter [4]
set_location_assignment PIN_138 -to LED_rowC[3] # Program counter [3]
set_location_assignment PIN_141 -to LED_rowC[2] # Program counter [2]
set_location_assignment PIN_142 -to LED_rowC[1] # Program counter [1]
set_location_assignment PIN_143 -to LED_rowC[0] # Program counter [0]

set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id
Top

```

## ΣΤ.2 E80.out.sdc - ρυθμίσεις χρονικών περιορισμών

```
#####
## E80 Computer - Quartus Timing Constraints for the DSD-il
## Device Cyclone IV E - "EP4CE6E22C8"
#####

set_time_format -unit ns -decimal_places 3
create_clock -name {CLK50MHz} -period 20.000 -waveform { 0.000 10.000 } [get_ports
{CLK50MHz}]
create_clock -name {CLK} -period 1000000.000 -waveform { 0.000 500000.000 }
[get_registers {CLK}]
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK50MHz}] -setup 1
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK50MHz}] -hold 0.020
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK50MHz}] -setup 1
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK50MHz}] -hold 0.020
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK}] 0.030
set_clock_uncertainty -rise_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK}] 0.030
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK50MHz}] -setup 1
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK50MHz}] -hold 0.020
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK50MHz}] -setup 1
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK50MHz}] -hold 0.020
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -rise_to [get_clocks
{CLK}] 0.030
set_clock_uncertainty -fall_from [get_clocks {CLK50MHz}] -fall_to [get_clocks
{CLK}] 0.030
set_clock_uncertainty -rise_from [get_clocks {CLK}] -rise_to [get_clocks
{CLK50MHz}] 0.030
set_clock_uncertainty -rise_from [get_clocks {CLK}] -fall_to [get_clocks
{CLK50MHz}] 0.030
set_clock_uncertainty -rise_from [get_clocks {CLK}] -rise_to [get_clocks {CLK}]
0.020
set_clock_uncertainty -rise_from [get_clocks {CLK}] -fall_to [get_clocks {CLK}]
0.020
set_clock_uncertainty -fall_from [get_clocks {CLK}] -rise_to [get_clocks
{CLK50MHz}] 0.030
set_clock_uncertainty -fall_from [get_clocks {CLK}] -fall_to [get_clocks
{CLK50MHz}] 0.030
set_clock_uncertainty -fall_from [get_clocks {CLK}] -rise_to [get_clocks {CLK}]
0.020
set_clock_uncertainty -fall_from [get_clocks {CLK}] -fall_to [get_clocks {CLK}]
0.020
set_input_delay -add_delay -clock [get_clocks {CLK}] 20.000 [get_ports {Reset}]
set_max_delay -from [get_clocks *] -through [get_pins -compatibility_mode *] -to
[get_clocks *] 1000.000
```

## Παράρτημα Z: Αρχεία ρυθμίσεων Gowin

### Z.1 Gowin.gprj - FPGA project

```
<?xml version="1" encoding="UTF-8"?>
<!DOCTYPE gowin-fpga-project>
<Project>
  <Template>FPGA</Template>
  <Version>5</Version>
  <Device name="GW5A-25A" pn="GW5A-LV25MG121NC1/I0">gw5a25a-002</Device>
  <FileList>
    <File path="../../VHDL/FPGA.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/Support.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/Computer.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/RAM.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/Firmware.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/CPU.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/RegisterFile.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/DFF8.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/ALU.vhd" type="file.vhdl" enable="1"/>
    <File path="../../VHDL/FA8.vhd" type="file.vhdl" enable="1"/>
    <File path="src/Gowin.cst" type="file.cst" enable="1"/>
    <File path="src/Gowin.sdc" type="file.sdc" enable="1"/>
  </FileList>
</Project>
```

### Z.2 Gowin\_process\_config.json - ρυθμίσεις

```
{
  "BACKGROUND_PROGRAMMING" : "off",
  "COMPRESS" : false,
  "CPU" : true,
  "CRC_CHECK" : true,
  "Clock_Route_Order" : 0,
  "Convert_SDP32_36_to_SDP16_18" : true,
  "Correct_Hold_Violation" : true,
  "DONE" : false,
  "DOWNLOAD_SPEED" : "default",
  "Disable_Insert_Pad" : false,
  "ENABLE_CTP" : false,
  "ENABLE_MERGE_MODE" : false,
  "ENCRYPTION_KEY" : false,
  "ENCRYPTION_KEY_TEXT" : "00000000000000000000000000000000",
  "ERROR_DECTION_AND_CORRECTION" : false,
  "ERROR_DECTION_ONLY" : false,
  "ERROR_INJECTION" : false,
  "EXTERNAL_MASTER_CONFIG_CLOCK" : false,
  "Enable_DSRM" : false,
  "FORMAT" : "binary",
  "FREQUENCY_DIVIDER" : "1",
  "Generate_Constraint_File_of_Ports" : false,
  "Generate_IBIS_File" : false,
  "Generate_Plain_Text_Timing_Report" : false,
  "Generate_Post_PNR_Simulation_Model_File" : false,
  "Generate_Post_Place_File" : false,
  "Generate_SDF_File" : false,
  "Generate_VHDL_Post_PNR_Simulation_Model_File" : false,
  "Global_Freq" : "default",
  "GwSyn_Loop_Limit" : 2000,
  "HOTBOOT" : false,
  "I2C" : false,
  "I2C_SLAVE_ADDR" : "00",
  "INCREMENTAL_PLACE_AND_ROUTING" : "0",
  "INCREMENTAL_PLACE_ONLY" : "0",
  "IncludePath" : [
  ],
}
```

```
"Incremental_Compile" : "",
"Initialize_Primitives" : false,
"JTAG" : false,
"MODE_IO" : false,
"MSPI" : false,
"MSPI_JUMP" : false,
"MULTIBOOT_ADDRESS_WIDTH" : "24",
"MULTIBOOT_MODE" : "Single",
"MULTIBOOT_SPI_FLASH_ADDRESS" : "000000",
"MULTIJUMP_ADDRESS_WIDTH" : "24",
"MULTIJUMP_MODE" : "Single",
"MULTIJUMP_SPI_FLASH_ADDRESS" : "000000",
"Multi_Boot" : false,
"OUTPUT_BASE_NAME" : "Gowin",
"POWER_ON_RESET_MONITOR" : true,
"PRINT_BSRAM_VALUE" : true,
"PROGRAM_DONE_BYPASS" : false,
"PlaceInRegToIob" : true,
"PlaceIoRegToIob" : true,
"PlaceOutRegToIob" : true,
"Place_Option" : "0",
"Process_Configuration_Verion" : "1.0",
"Promote_Physical_Constraint_Warning_to_Error" : true,
"READY" : false,
"RECONFIG_N" : false,
"Ram_RW_Check" : false,
"Replicate_Resources" : false,
"Report_Auto-Placed_Io_Information" : false,
"Route_Maxfan" : 23,
"Route_Option" : "0",
"Run_Timing_Driven" : true,
"SECURE_MODE" : false,
"SECURITY_BIT" : true,
"SEU_HANDLER" : false,
"SEU_HANDLER_CHECKSUM" : false,
"SEU_HANDLER_MODE" : "auto",
"SSPI" : true,
"STOP_SEU_HANDLER" : false,
"Show_All_Warnings" : false,
"Synthesize_tool" : "GowinSyn",
"TclPre" : "",
"TopModule" : "FPGA",
"USERCODE" : "default",
"Unused_Pin" : "As_input_tri_stated_with_pull_up",
"VCC" : "0.9",
"VCCAUX" : "3.3",
"VCCX" : "3.3",
"VHDL_Standard" : "VHDL_Std_2008",
"Verilog_Standard" : "Vlg_Std_2001",
"WAKE_UP" : "0",
"show_all_warnings" : false,
"turn_off_bg" : false
}
```



## Z.3 Gowin.sdc - χρονισμός ρολογιού

```
// E80 Computer - Timing Constraints for the Tang Primer 25K
// Gowin V1.9.11.01 Education (64-bit)
// FPGA: GW5A-25 Version A (GW5A-LV25MG121NC1/I0)

create_clock -name Clock50Mhz -period 20 -waveform {0 10} [get_ports {CLK50MHz}]
```

## Z.4 Gowin.cst - ακροδέκτες

```
// E80 Computer - Physical Constraints file
// Gowin V1.9.11.01 Education (64-bit)
// FPGA: GW5A-25 Version A (GW5A-LV25MG121NC1/I0)

IO_LOC "LED_rowC[7]" H5;
IO_PORT "LED_rowC[7]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[6]" H8;
IO_PORT "LED_rowC[6]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[5]" G7;
IO_PORT "LED_rowC[5]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[4]" F5;
IO_PORT "LED_rowC[4]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[3]" J5;
IO_PORT "LED_rowC[3]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[2]" H7;
IO_PORT "LED_rowC[2]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[1]" G8;
IO_PORT "LED_rowC[1]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowC[0]" G5;
IO_PORT "LED_rowC[0]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[7]" L5;
IO_PORT "LED_rowB[7]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[6]" K11;
IO_PORT "LED_rowB[6]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[5]" E11;
IO_PORT "LED_rowB[5]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[4]" A11;
IO_PORT "LED_rowB[4]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[3]" K5;
IO_PORT "LED_rowB[3]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[2]" L11;
IO_PORT "LED_rowB[2]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[1]" E10;
IO_PORT "LED_rowB[1]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowB[0]" A10;
IO_PORT "LED_rowB[0]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[7]" C11;
IO_PORT "LED_rowA[7]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[6]" B11;
IO_PORT "LED_rowA[6]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[5]" D11;
IO_PORT "LED_rowA[5]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[4]" G11;
IO_PORT "LED_rowA[4]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[3]" C10;
IO_PORT "LED_rowA[3]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[2]" B10;
IO_PORT "LED_rowA[2]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[1]" D10;
IO_PORT "LED_rowA[1]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "LED_rowA[0]" G10;
IO_PORT "LED_rowA[0]" IO_TYPE=LVC MOS33 PULL_MODE=NONE DRIVE=8 BANK_VCCIO=3.3;
IO_LOC "DIPinput[7]" H1;
IO_PORT "DIPinput[7]" IO_TYPE=LVC MOS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[6]" K7;
IO_PORT "DIPinput[6]" IO_TYPE=LVC MOS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[5]" L7;
```

```
IO_PORT "DIPinput[5]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[4]" L10;
IO_PORT "DIPinput[4]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[3]" L9;
IO_PORT "DIPinput[3]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[2]" J8;
IO_PORT "DIPinput[2]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[1]" F7;
IO_PORT "DIPinput[1]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "DIPinput[0]" J11;
IO_PORT "DIPinput[0]" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Mid" L8;
IO_PORT "Mid" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Left" K9;
IO_PORT "Left" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Right" K10;
IO_PORT "Right" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Down" K8;
IO_PORT "Down" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Up" F6;
IO_PORT "Up" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Pause" J7;
IO_PORT "Pause" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "Reset" H2;
IO_PORT "Reset" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
IO_LOC "CLK50MHz" E2;
IO_PORT "CLK50MHz" IO_TYPE=LVCMS33 PULL_MODE=NONE BANK_VCCIO=3.3;
```

## Παράρτημα Η: Κώδικας συμβολομεταφραστή E80ASM σε C

### Η.1 main.c - συντακτική ανάλυση και μετάφραση

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "error_handler.h"
#include "data_structures.h"
#include "parse_functions.h"

int main(void)
{
    char str[MAX_LINE_LENGTH] = {0}; // scratchpad string
    char instr[MAX_LINE_LENGTH] = {0}; // current instruction
    char title[MAX_LINE_LENGTH] = {0}; // .TITLE string (optional)
    int frequency = DEFAULT_FREQ; // .FREQUENCY value
    char sim DIP[9] = "00000000"; // .SIMDIP value
    int reg, reg2; // register address
    int n; // scratchpad index or value
    int data_space; // address after the last instruction
    int len, spaces; // formatting helpers
    char* CtrlD = NULL; //
    FILE* asm_input = stdin; // fopen("test.asm", "r");
    FILE* vhd_template = fopen(TEMPLATE, "r");
    if (!vhd_template) error(OPEN_TEMPLATE);

    /* Starting message */
    fprintf(stderr,
        "E80 CPU Assembler - April 2025, Panos Stokas\n\n"
        "Translates an assembly program into E80-firmware VHDL code.\n\n"
        "I/O is handled via stdin/stdout. Eg. to read 'program.asm'\n"
        "and write the result to 'firmware.vhd', type:\n\n"
        "e80asm < program.asm > firmware.vhd\n\n"
        "You can also paste your code here and then press\n"
        "Ctrl-D & [Enter] to translate it, or Ctrl-C to exit.\n\n");

    /* Read lines from stdin until EOF or end-of-transmit (Ctrl-D). */
    while (!CtrlD && fgets(str, MAX_LINE_LENGTH, asm_input)) {
        if ((CtrlD = strchr(str, 4))) { // end of transmit found
            *CtrlD = 0; // replace end of transmit with terminal
        } else if (!strchr(str, '\n') && !feof(asm_input)) {
            // a line without a newline character, is either the last line or
            // it exceeds the maximum supported size.
            error(MAX_LENGTH_EXCEEDED);
        }
        trim(str); // trim whitespace and comments
        enqueue(str); // store the line in the global "In" structure
    }

    /* Collect symbols (names and labels).
    Symbol/value pairs are added to the "Out" structure. Error checking is
    minimal in this stage. */
    fprintf(stderr, "Collecting symbols... ");
    Out.addr = 0; // current memory address in the global "Out" structure
    firstline(); // go to the first token of the queued code
    while (In.current) { // read until the last line
        if (eq(TOKEN, ".NAME")) {
            // <directive> ::= ".NAME" <s+> <identifier> <s+> <number>
            strcpy(str, nexttoken()); // <identifier>
            if (!identifier(str)) error(IDENTIFIER);
            n = number(nexttoken()); // <number>
            if (n < 0) error(NUMBER); // error codes = negative values
            addsymbol(str, n); // includes a check for duplicates
        } else if (instr_size1(TOKEN)) {
```

```

        nextaddr(); // combines Out.addr++ and ram limit check
    } else if (instr_size2(TOKEN)) {
        nextaddr();
        nextaddr(); // two-word instructions
    } else if (identifier(TOKEN)) {
        // <label> ::= <identifier> <s*> ":"
        strcpy(str, TOKEN);
        if (eq(nexttoken(), ":")) {
            addsymbol(str, Out.addr);
            // check the next token instead of the next line to allow
            // for <codeline> ::= <[label]> <[\n]> <[instruction]>
            nexttoken();
            continue;
        }
    }
    nextline();
}
data_space = Out.addr; // for checking collisions with program code

// print symbol-value pairs prior to sorting
if (Out.symbols == 0) fprintf(stderr, "None.");
for (n = 0; n < Out.symbols; n++) {
    fprintf(stderr,
        "\n- %s = %d", Out.symbol[n].name, Out.symbol[n].val);
}
fprintf(stderr, "\n");
sortsymbols(); // sort by name for binary search on symbolvalue()

/* Parse directives.
E80 is designed according to the Neumann model where machine code and data
are stored in the same area. However, .DATA arrays are checked against
overwriting program code. */
fprintf(stderr, "Parsing directives... ");
firstline();
while (In.current) {
    if (eq(TOKEN, ".TITLE")) {
        // <directive> ::= ".TITLE" <s*> <quoted_string>
        if (title[0]) error(DUPLICATE_TITLE); // previously set
        nexttoken();
        if (TOKEN[0] != '"') error(UNQUOTED_TITLE);
        strncpy(title, TOKEN + 1, strlen(TOKEN) - 2); // unquote
    } else if (eq(TOKEN, ".DATA")) {
        // <directive> ::= ".DATA" <s*> <value> <s*> <array>
        n = value(nexttoken());
        if (n < 0) error(DATA_ADDRESS);
        if (n < data_space) error(DATA_SPACE);
        // write data on the RAM starting from address n
        Out.addr = n;
        do {
            if (!array_element(nexttoken())) error ARRAY_ELEMENT);
            // <array_element> ::= <number> | <quoted_string>
            if (TOKEN[0] != '"') {
                // <number>, write on the RAM as 8 bits
                bitcopy(RAM, value(TOKEN), 7, 0);
                // add the original number as a comment
                sprintf(COMMENT, "%s", TOKEN);
                nextaddr();
            } else {
                // <quoted_string> ::= "\"" <char*> "\""
                // skip quotes and write each character's ASCII
                // value on the RAM as 8 bits
                for (unsigned int i = 1; i < strlen(TOKEN) - 1; i++) {
                    bitcopy(RAM, (int)TOKEN[i], 7, 0);
                    // add each character as a comment
                    sprintf(COMMENT, "'%c' (%d)", TOKEN[i], TOKEN[i]);
                    nextaddr();
                }
            }
        }
    }
}

```

```

        // <array> ::= <array_element> | <array_element> <,> <array>
    } while (eq(nexttoken(), ","));
    if (!eq(TOKEN, ",")) error(COMMA);
} else if (eq(TOKEN, ".FREQUENCY")) {
    // <directive> ::= ".FREQUENCY" <s+> <number>
    // <number> is an exception here, it's not restricted to 1 byte
    frequency = (int)strtol(nexttoken(), NULL, 10);
    if (frequency < MIN_FREQ || frequency > MAX_FREQ) error(FREQUENCY);
} else if (eq(TOKEN, ".SIMDIP")) {
    // <directive> ::= ".SIMDIP" <s+> <value>
    bitcopy(simdip, value(nexttoken()), 7, 0);
} else if (eq(TOKEN, ".NAME")) {
    // already processed during symbol collection
    nexttoken();
    nexttoken();
} else if (!eq(TOKEN, ",")) {
    // a non empty token which is not a directive ⇒ end of directives
    break;
}
if (nexttoken()) error(EXTRANEIOUS);
nextline();
}
fprintf(stderr, "OK.\n");

/* Parse instructions according to the BNF syntax rules.
The parser functions (instr_argumentless, instr_n, etc) handle syntax
checking, translation and write the opcode to the "Out" structure's array.
The remaining bits are filled by the code below. The RAM and COMMENT
macros specify a string element at Out.addr. Each binary instruction is
followed by a comment with its assembly mnemonic. For two-word instructions
the first part will not have a comment. Comments therefore are used to
differentiate between one and two word instructions and allows to create
well-formatted VHDL code where each instruction is written in one line. */
fprintf(stderr, "Parsing instructions... ");
Out.addr = 0;
while (In.current) {
    if ((instr_noarg(TOKEN)) {
        // <[instruction]> ::= <instr_noarg>
        sprintf(COMMENT, "%s", TOKEN);
        nextaddr();
    } else if (instr_reg(TOKEN)) {
        // <[instruction]> ::= <instr_reg> <s+> <reg>
        strcpy(instr, TOKEN);
        reg = regnum(nexttoken());
        if (reg < 0) error(REGISTER);
        bitcopy(RAM, reg, 2, 0); // <reg> in Instr1[2:0]
        sprintf(COMMENT, "%s R%d", instr, reg);
        nextaddr();
    } else if (instr_n(TOKEN)) {
        // <[instruction]> ::= <instr_n> <s+> <n>
        strcpy(instr, TOKEN);
        n = value(nexttoken());
        if (n < 0) error(VALUE);
        nextaddr();
        bitcopy(RAM, n, 7, 0); // <n>
        sprintf(COMMENT, "%s %d", instr, n);
        nextaddr();
    } else if (instr_op1(TOKEN)) {
        // <[instruction]> ::= <instr_op1> <s+> <op>
        strcpy(instr, TOKEN);
        nexttoken();
        n = value(TOKEN);
        reg = regnum(TOKEN);
        if (n >= 0) {
            // op1 is a direct n address
            strcpy(&RAM[7], "0");
            nextaddr();
            bitcopy(RAM, n, 7, 0); // <n> in Instr2

```

```

        sprintf(COMMENT, "%s %d", instr, n);
        nextaddr();
    } else if (reg >= 0) {
        // op1 is a register address
        strcpy(&RAM[7], "1");
        nextaddr();
        strcpy(&RAM[0], "00000");
        bitcopy(RAM, reg, 3, 0); // <reg> in Instr2[2:0]
        sprintf(COMMENT, "%s R%d", instr, reg);
        nextaddr();
    } else {
        error(OP);
    }
} else if (instr_reg_op2(TOKEN)) {
    // <[instruction]> ::= <instr_reg_op2> <s+> <reg> <,> <op2>
    char bracket_op2 = load_store(TOKEN); // op2 must be bracketed
    strcpy(instr, TOKEN);
    reg = regnum(nexttoken());
    if (reg < 0) error(REGISTER);
    if (!eq(nexttoken(), ",")) error(COMMA);
    str[0] = 0; // clear scratchpad string
    sprintf(str, "%s R%d", instr, reg);
    nexttoken();
    if (bracket_op2) {
        if (!eq(TOKEN, "[")) error(LEFTBRACKET);
        sprintf(str+strlen(str), "[");
        nexttoken();
    }
    n = value(TOKEN);
    reg2 = regnum(TOKEN);
    if (n >= 0) {
        // op2 is an immediate n value
        bitcopy(RAM, reg, 3, 0); // <reg> in Instr1[3:0]
        nextaddr();
        bitcopy(RAM, n, 7, 0); // <n> in Instr2
        sprintf(COMMENT, "%s%d", str, n);
    } else if (reg2 >= 0) {
        // op2 is a register address (reg2)
        strcpy(&RAM[4], "1000");
        nextaddr();
        bitcopy(RAM, reg, 7, 4); // <reg> in Instr2[7:4]
        bitcopy(RAM, reg2, 3, 0); // <reg2> in Instr2[3:0]
        sprintf(COMMENT, "%sR%d", str, reg2);
    } else {
        error(OP);
    }
    if (bracket_op2) {
        if (!eq(nexttoken(), "]")) error(RIGHTBRACKET);
        sprintf(COMMENT+strlen(COMMENT), "]);");
    }
    nextaddr();
} else if (instr_reg_n(TOKEN)) {
    // <[instruction]> ::= <instr_reg_n> <s+> <reg> <,> <n>
    strcpy(instr, TOKEN);
    reg = regnum(nexttoken());
    if (reg < 0) error(REGISTER);
    if (!eq(nexttoken(), ",")) error(COMMA);
    nexttoken();
    n = value(TOKEN);
    if (n < 0) error(VALUE);
    bitcopy(RAM, reg, 2, 0);
    nextaddr();
    bitcopy(RAM, n, 7, 0);
    sprintf(COMMENT, "%s R%d, %d", instr, reg, n);
    nextaddr();
} else if (symbolvalue(TOKEN) >= 0) {
    // <[label]> ::= <identifier> <s*> ":" | ""
    if (!eq(nexttoken(), ":")) error(COLON);

```

```

        nexttoken();
        continue;
    } else if (!eq(TOKEN, "")) {
        error(UNKNOWN);
    }
    if (nexttoken()) error(EXTRANEIOUS);
    nextline();
}
fprintf(stderr, "OK.\n\n");

/* Print the converted VHDL code using the template file.
Each instruction reserves one line, followed by a comment specifying the
original mnemonic (in which references to symbols have been translated
to specific values or addresses). Title and frequency are also printed
on their specific placeholders. */
while (fgets(str, MAX_LINE_LENGTH, vhdl_template) != NULL) {
    if (strstr(str, "E80 Firmware")) {
        if (!eq(title, "")) {
            printf("-- %s\n", title);
        } else {
            printf(str);
        }
    } else if (strstr(str, "DefaultFrequency")) {
        printf(str, frequency); // template contains %d specifier
    } else if (strstr(str, "SimDIP")) {
        printf(str, simdip); // template contains %s specifier
    } else if (strstr(str, "MACHINE_CODE_PLACEHOLDER")) {
        str[0] = 0; // clear scratchpad string
        for (Out.addr = 0; Out.addr < RAM_SIZE; Out.addr++) {
            if (eq(RAM, "")) continue; // handled by OTHERS in VHDL
            len = strlen(str);
            /* Write the instruction address in the end of the current
            line; this allows for two-word instructions to have
            both parts in the same line, such as:
            addr => "instr1", addr+1 => "instr2" -- comment
            or, for single-word instructions:
            addr => "instr1", -- comment. */
            sprintf(str + len, "%d", Out.addr);
            len = strlen(str);
            if (len < 15) {
                // space after the address in the 1st part of the line
                // this allows for 1-3 address digits
                spaces = 4 - len;
            } else {
                // space after the address in the 2nd part of the line
                spaces = 23 - len;
            }
            // write the VHDL assignment of the word after the address
            sprintf(str + len, "%*c=> \"%s\"", " ", spaces, ' ', RAM);
            if (!eq(COMMENT, "")) {
                // comments are written after single-word instructions
                // or after the 2nd part of two-word instructions
                len = strlen(str);
                // streamline comments for both instruction types
                spaces = 39 - len;
                sprintf(str + len, "%*c-- %s", spaces, ' ', COMMENT);
                puts(str);
                str[0] = 0; // prepare for new line
            }
        }
    } else {
        printf(str); // unmodified template lines
    }
}

return NO_ERROR;
}

```



## H.2 error\_handler.h - επικεφαλίδα χειρισμού σφαλμάτων

```
#ifndef ERROR_HANDLER_H
#define ERROR_HANDLER_H

enum ErrorCode {
    NO_ERROR,
    OPEN_TEMPLATE,
    MAX_LENGTH_EXCEEDED,
    MEMORY_ALLOCATION_ERROR,
    FEW_ARGUMENTS,
    TOO_MANY_ARGUMENTS,
    HEX_NUMBER_MALFORMED,
    IDENTIFIER,
    EMPTY_STRING,
    UNCLOSED_STRING,
    ARRAY_ELEMENT,
    FREQUENCY,
    NUMBER,
    MANY_SYMBOLS,
    DUPLICATE_SYMBOL,
    EXTRANEOUS,
    UNKNOWN,
    RESERVED,
    COLON,
    REGISTER,
    VALUE,
    COMMA,
    LEFTBRACKET,
    RIGHTBRACKET,
    OP,
    DATA_ADDRESS,
    DATA_SPACE,
    RAM_LIMIT,
    UNQUOTED_TITLE,
    DUPLICATE_TITLE
};

enum NumErrorCode {
    HEX_ERROR = -16,
    BIN_ERROR = -2,
    NUMBER_ERROR = -1,
    OCTAL_ERROR = -8,
    RANGE_ERROR = -255
};

void error(enum ErrorCode errorlevel);

#endif
```

## H.3 error\_handler.c - μηνύματα σφαλμάτων

```
#include <stdio.h>
#include <stdlib.h>
#include "error_handler.h"
#include "data_structures.h"
#include "parse_functions.h"

void printf_number_format_help(void) {
    fprintf(stderr,
        "Numbers can either be:\n"
        "1) Hexadecimal preceded by 0x, up to 2 digits (eg. 0x0F)\n"
        "2) Binary preceded by 0b, up to 8 digits (eg. 0b00001111)\n"
        "3) Decimal 0-255 with no leading zeroes (eg. 15)\n");
}

void printf_val_help(void) {
    if (number(TOKEN) == HEX_ERROR) {
```

```

        fprintf(stderr,
            "Hexadecimals are limited to 2 digits (eg. 0xF or 0x1A)\n");
    } else if (number(TOKEN) == BIN_ERROR) {
        fprintf(stderr,
            "Binary numbers are limited to 8 digits (eg. 0b00101011)\n");
    } else if (number(TOKEN) == OCTAL_ERROR) {
        fprintf(stderr, "Leading zeroes are not allowed on decimal numbers\n");
    } else if (number(TOKEN) == RANGE_ERROR) {
        fprintf(stderr, "Decimal numbers are limited to unsigned 0-255");
    } else if (!identifier(TOKEN)) {
        fprintf(stderr,
            "It contains letters, but names must start with a letter and"
            " followed by letters, numbers and underscores");
    } else {
        fprintf(stderr,
            "Symbolic names must be defined in .NAME directives"
            " or as labels in the code\n");
    }
}

/* Terminates execution, printing a message and returns an error code. */
void error(enum ErrorCode errorlevel)
{
    fprintf(stderr,
        "\n*****\n");
    if (In.line_number) {
        fprintf(stderr,
            "Error in line %d : %s\n", In.line_number, In.current->line);
    }

    switch (errorlevel) {
    case OPEN_TEMPLATE:
        fprintf(stderr, "Error! Can't open the template file '%s'", TEMPLATE);
        break;
    case MAX_LENGTH_EXCEEDED:
        fprintf(stderr, "Line exceeds maximum %d characters.", MAX_LINE_LENGTH);
        break;
    case IDENTIFIER:
        fprintf(stderr, "'%s' is not a valid identifier.", TOKEN);
        break;
    case EMPTY_STRING:
        fprintf(stderr, "Empty strings are not permitted.");
        break;
    case UNCLOSED_STRING:
        fprintf(stderr, "Quote expected after string '%s'.", TOKEN);
        break;
    case ARRAY_ELEMENT:
        if (eq(TOKEN, "")) {
            fprintf(stderr, "Expected an array element.\n");
        } else {
            fprintf(stderr, "'%s' is not a literal.\n", TOKEN);
        }
        fprintf(stderr,
            "Example of an array: .DATA 100 12, \"abc\", 0xAF, 0b1011\n"
            "Quotes can be escaped in strings, eg: \"a\\\"b\\\"\"");
        printf_number_format_help();
        break;
    case FREQUENCY:
        fprintf(stderr,
            "Frequency must be a number between '%d' and '%d' deciHertz.\n",
            MIN_FREQ, MAX_FREQ);
        break;
    case NUMBER:
        fprintf(stderr, "'%s' is not a number between 0 and 255.\n", TOKEN);
        printf_number_format_help();
        break;
    case MANY_SYMBOLS:
        puts("Maximum number of symbols reached");
    }
}

```

```

        break;
    case DUPLICATE_SYMBOL:
        fprintf(stderr, "This name or label has been set in a previous line.");
        break;
    case MEMORY_ALLOCATION_ERROR:
        puts("Memory allocation error!");
        break;
    case EXTRANEOUS:
        fprintf(stderr, "'%s' was unexpected", TOKEN);
        break;
    case UNKNOWN:
        fprintf(stderr, "'%s' is an unknown instruction or directive", TOKEN);
        break;
    case RESERVED:
        fprintf(stderr, "'%s' is reserved and cannot be a name or label", TOKEN);
        break;
    case COLON:
        fprintf(stderr, "Colon expected after label");
        break;
    case REGISTER:
        fprintf(stderr,
            "'%s' is not a register; allowed registers are R0-R7", TOKEN);
        break;
    case VALUE:
        fprintf(stderr, "'%s' is not a number or symbol", TOKEN);
        break;
    case COMMA:
        fprintf(stderr, "Comma expected after '%s'", PREVIOUS);
        break;
    case LEFTBRACKET:
        fprintf(stderr, "LOAD/STORE requires a left bracket before '%s'", TOKEN);
        break;
    case RIGHTBRACKET:
        fprintf(stderr, "LOAD/STORE requires a right bracket"
            " after '%s'", PREVIOUS);
        break;
    case OP:
        fprintf(stderr, "'%s' is not number or symbol or register.", TOKEN);
        break;
    case DATA_ADDRESS:
        fprintf(stderr, "'%s' is not a valid address or symbol.", TOKEN);
        printf_val_help();
        break;
    case DATA_SPACE:
        fprintf(stderr, "'%s' is an address in program code.", TOKEN);
        break;
    case RAM_LIMIT:
        fprintf(stderr, "%d-byte RAM limit exceeded.", RAM_SIZE);
        break;
    case UNQUOTED_TITLE:
        fprintf(stderr, "Quoted title string expected.");
        break;
    case DUPLICATE_TITLE:
        fprintf(stderr, "Only one .TITLE directive is allowed.");
        break;
    default:
        break;
}
fprintf(stderr, "\n");
fprintf(stderr, "*****\n");

exit(errorlevel);
}

```

## H.4 parse\_functions.h - επικεφαλίδα συντακτικών εργαλείων

```
#ifndef PARSE_FUNCTIONS_H
#define PARSE_FUNCTIONS_H

/* Converts s to a number according to this rule:
<number> ::= "0x" <hex+> | "0b" <bit+> | <dec+>
Strings correspond to unsigned numbers, so a negative return value signifies
an error or a non-number */
int number(const char *s);

/* Trims leading and trailing trimmable characters from s. */
void trim(char *s);

/* Compares two strings case-insensitively, taking into account NULL pointers.
Returns 1 if the strings are equal, 0 otherwise. */
char eq(const char *s1, const char *s2);

char identifier(const char *str);
char array_element(const char *str);

/* Returns 1 if s is a single-word instruction */
char instr_size1(const char *s);

/* Returns 1 if the parameter is a two-word instruction */
char instr_size2(const char *s);

/* "HLT" | "NOP" | "RETURN" */
char instr_noarg(const char *s);

/* "RSHIFT" | "LSHIFT" | "PUSH" | "POP" */
char instr_reg(const char *s);

/* "JC" | "JNC" | "JZ" | "JNZ" | "JS" | "JNS" | "JV" | "JNV" | "CALL" */
char instr_n(const char *s);

/* "JMP" */
char instr_op1(const char *s);

/* "MOV" | "ADD" | "ROR" | "SUB" | "CMP" | "AND" | "OR" | "XOR" |
"LOAD" | "STORE" */
char instr_reg_op2(const char *s);

/* "LOAD" | "STORE" */
char load_store(const char *s);

/* "BIT" */
char instr_reg_n(const char *s);

/* Returns the address of the register string parameter according to:
<reg> ::= "R0" | "R1" | "R2" | "R3" | "R4" | "R5" | "R6" | "SP" | "R7" | "FLAGS" */
int regnum(const char *s);

/* Returns the value of the operand parameter according to:
<n> ::= <number> | <identifier>
if the parameter is an identifier, it gets its value from the symbols table. */
int value(const char *s);

/* Converts num to bits, in Little Endian order to match VHDL's DOWNTO */
void bitcopy(char *dest, int num, int high, int low);

#endif
```

## H.5 parse\_functions.c - εργαλεία συντακτικής ανάλυσης

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "parse_functions.h"
#include "error_handler.h"
#include "data_structures.h"
#include "config.h"

/* Converts a string to uppercase, excluding quoted parts
taking into account escaped quotes. */
void uppcase(char *s)
{
    if (!s || !*s) return; // ignore NULL or empty strings
    *s = (char)toupper(*s); // first character
    char quoted = 0; // flag
    for (s++; *s; s++) {
        if (s[-1] != '\\' && s[0] == '"') { // \" = escaped quote
            quoted = !quoted;
        }
        if (!quoted) *s = (char)toupper(*s);
    }
}

int number(const char *s)
{
    if (!s) return -1;
    int n; // converted value (or negative)
    char valid[MAX_LINE_LENGTH]; // for sscanf masking -- valid digits
    char invalid[MAX_LINE_LENGTH]; // for sscanf masking -- invalid characters
    char S[MAX_LINE_LENGTH]; // uppercase version of the parameter
    strcpy(S, s);
    uppcase(S);
    // extract the valid digits of the parameter and convert it
    // according to its prefix
    if (!strcmp(S, "0X", 2)) {
        // Hexadecimal format, up to 2 digits
        if (sscanf(S, "0X%2[0-9A-F]%^\\n", valid, invalid) != 1) {
            return HEX_ERROR;
        }
        n = (int)strtol(valid, NULL, 16);
    } else if (!strcmp(S, "0B", 2)) {
        // Binary format, up to 8 digits
        if (sscanf(S, "0B%8[01]%^\\n", valid, invalid) != 1) {
            return BIN_ERROR;
        }
        n = (int)strtol(valid, NULL, 2);
    } else {
        // Decimal format
        if (sscanf(S, "%[0-9]%^0-9]", valid, invalid) != 1) {
            return NUMBER_ERROR;
        }
        if (S[0] == '0' && strlen(S) > 1) {
            // trailing zero is not accepted because it signifies
            // octal numbers in GNU-assembly
            return OCTAL_ERROR;
        }
        n = (int)strtol(S, NULL, 10);
        if (n > 255 || n < 0) return RANGE_ERROR; // unsigned 8 bit
    }
    return n;
}

void trim(char *s)
{

```

```

    if (!s || !*s) return;
    char *start = s;
    char *end = s; // end of string
    char quoted = 0;
    while (*end) { // find the terminal or an unquoted semicolon
        if (*end == '"' && end[-1] != '\\') quoted = !quoted; // \" = escaped
        if (!quoted && *end == ';') break;
        end++;
    }
    end--; // terminal = [end+1]
    while (isspace(*start)) start++; // first non-trimmable up to terminal
    if (start < end) {
        while (isspace(*end)) end--; // last non-trimmable
    }
    end[1] = '\\0'; // terminate after the last non-trimmable
    // shift the trimmed content to the begin of the string,
    // +1 for a single character, and +1 for the added terminator
    memmove(s, start, end - start + 2);
}

char eq(const char *s1, const char *s2)
{
    if (s1 == s2) return 1; // both NULL or point to the same string
    if (!s1 || !s2) return 0; // only one is NULL
    // compare each character case-insensitively until a terminator is reached
    // in either string
    while (*s1 && *s2 && toupper(*s1) == toupper(*s2)) {
        s1++;
        s2++;
    }
    // if both terminators have been reached, they are equal
    return !*s1 && !*s2;
}

char instr_size1(const char *s) {
    // search string: space + str + space
    char search_str[strlen(s)+3];
    sprintf(search_str, " %s ", s);
    uppercase(search_str);
    return (strstr(" HLT NOP RETURN RSHIFT LSHIFT PUSH POP ", search_str) != 0);
}

char instr_size2(const char *s) {
    // search string: space + str + space
    char search_str[strlen(s)+3];
    sprintf(search_str, " %s ", s);
    uppercase(search_str);
    return (strstr(" JMP JC JNC JZ JNZ JS JNS JV JNV CALL MOV ADD SUB ROR AND"
        " OR XOR STORE LOAD CMP BIT ", search_str) != 0);
}

char reserved(const char *s) {
    if (instr_size1(s)) return 1;
    if (instr_size2(s)) return 1;
    // search string: space + s + space
    char search_str[strlen(s)+3];
    sprintf(search_str, " %s ", s);
    uppercase(search_str);
    return (strstr(" R0 R1 R2 R3 R4 R5 R6 R7 SP FLAGS ", search_str) != 0);
}

char instr_noarg(const char *s)
{
    if (eq(s, "HLT")) strcpy(RAM, "00000000");
    else if (eq(s, "NOP")) strcpy(RAM, "00000001");
    else if (eq(s, "RETURN")) strcpy(RAM, "00001111");
    else return 0;
    return 1;
}

```

```

}

char instr_reg(const char *s)
{
    if      (eq(s, "RSHIFT")) strcpy(RAM, "10100");
    else if (eq(s, "LSHIFT")) strcpy(RAM, "11000");
    else if (eq(s, "PUSH"))   strcpy(RAM, "11100");
    else if (eq(s, "POP"))    strcpy(RAM, "11110");
    else return 0;
    return 1;
}

char instr_n(const char *s)
{
    if      (eq(s, "JC"))      strcpy(RAM, "00000100");
    else if (eq(s, "JNC"))     strcpy(RAM, "00000101");
    else if (eq(s, "JZ"))      strcpy(RAM, "00000110");
    else if (eq(s, "JNZ"))     strcpy(RAM, "00000111");
    else if (eq(s, "JS"))      strcpy(RAM, "00001010");
    else if (eq(s, "JNS"))     strcpy(RAM, "00001011");
    else if (eq(s, "JV"))      strcpy(RAM, "00001100");
    else if (eq(s, "JNV"))     strcpy(RAM, "00001101");
    else if (eq(s, "CALL"))    strcpy(RAM, "00001110");
    else return 0;
    return 2;
}

char instr_op1(const char *s)
{
    if      (eq(s, "JMP"))     strcpy(RAM, "0000001"); // ≠ NOP
    else return 0;
    return 2;
}

char instr_reg_op2(const char *s)
{
    if      (eq(s, "MOV"))     strcpy(RAM, "0001");
    else if (eq(s, "ADD"))     strcpy(RAM, "0010");
    else if (eq(s, "SUB"))     strcpy(RAM, "0011");
    else if (eq(s, "ROR"))     strcpy(RAM, "0100");
    else if (eq(s, "AND"))     strcpy(RAM, "0101");
    else if (eq(s, "OR"))      strcpy(RAM, "0110");
    else if (eq(s, "XOR"))     strcpy(RAM, "0111");
    else if (eq(s, "STORE"))   strcpy(RAM, "1000");
    else if (eq(s, "LOAD"))    strcpy(RAM, "1001");
    else if (eq(s, "CMP"))     strcpy(RAM, "1011");
    else return 0;
    return 2;
}

char load_store(const char *s)
{
    if      (eq(s, "STORE"))   strcpy(RAM, "1000");
    else if (eq(s, "LOAD"))    strcpy(RAM, "1001");
    else return 0;
    return 2;
}

char instr_reg_n(const char *s)
{
    if (eq(s, "BIT")) strcpy(RAM, "11010");
    else return 0;
    return 1;
}

/* <name_char> ::= <letter> | <dec> | "_" */
char name_char(const char c)
{

```



```

    return isalpha(c) || isdigit(c) || c == '_';
}

/* <identifier> ::= <letter> <name_char*> */
char identifier(const char *str)
{
    if (!isalpha(str[0])) return 0;
    for (int i = 1; str[i] != '\0'; i++) {
        if (!name_char(str[i])) return 0;
    }
    // don't allow identifiers to use reserved words
    if (reserved(str)) error(RESERVED);
    return 1;
}

/* <array_element> ::= <number> | <quoted_string> */
char array_element(const char *str)
{
    int len = strlen(str);
    if (str[0] == '"') {
        // <quoted_string> ::= "\"" <char+> "\""
        if (len < 3) error(EMPTY_STRING);
        // the closing quote is handled at nexttoken in data_structures.c
    } else if (number(str) < 0) {
        error(ARRAY_ELEMENT);
    }
    return 1;
}

int regnum(const char *s)
{
    if (eq(s, "R0")) return 0;
    if (eq(s, "R1")) return 1;
    if (eq(s, "R2")) return 2;
    if (eq(s, "R3")) return 3;
    if (eq(s, "R4")) return 4;
    if (eq(s, "R5")) return 5;
    if (eq(s, "R6") || eq(s, "FLAGS")) return 6;
    if (eq(s, "R7") || eq(s, "SP")) return 7;
    return -1;
}

int value(const char *s)
{
    int n = number(s);
    if (n < 0) n = symbolvalue(s); // if it's not a number, search symbols
    return n;
}

void bitcopy(char *dest, int num, int high, int low)
{
    // convert VHDL [7 DOWNT0 0] to array order MSB=0, LSB=7
    int MSB = 7 - high;
    int LSB = 7 - low;
    for (int i = LSB; i >= MSB; i--) { // from LSB to MSB
        // num & 1 = num's LSB (bitwise AND)
        dest[i] = (num & 1) ? '1' : '0';
        num >>= 1;
    }
    if (LSB == 7) dest[8] = '\0';
}

```

## H.6 data\_structures.h - επικεφαλίδα δομών δεδομένων

```
#ifndef DATA_STRUCTURE_H
#define DATA_STRUCTURE_H

#include "config.h"

#define SINGLE_CHAR_DELIMITERS "[\],: " // ["],:
#define ALL_DELIMITERS "[\],: \t\n\r\f\v\0" // above + whitespace & terminal

/* Stores a line from the assembly input */
struct LineNode {
    char line[MAX_LINE_LENGTH];
    struct LineNode *next;
};

/* Stores the pointers to the LineNode list, and related variables for
processing assembly input. */
struct InputHeader {
    struct LineNode *front;
    struct LineNode *rear;
    struct LineNode *current;
    char *chr; // tokenization candidate character at current->line
    char token[MAX_LINE_LENGTH];
    char previous[MAX_LINE_LENGTH]; // previous token for error context
    int line_number;
};

/* Symbol/value pair, indexed at the SymbolElement. */
struct SymbolElement {
    char *name;
    unsigned char val;
};

/* Stores an array of pointers to SymbolElements; this array is sorted after
the Symbol collection stage at main() to allow fast binary search. This header
includes the final ram/comment arrays where the translated code is printed. */
struct OutputHeader {
    unsigned char symbols; // number of stored symbols
    struct SymbolElement symbol[MAX_SYMBOLS];
    unsigned int addr; // current instruction address
    char ram[255][9];
    char comment[255][MAX_LINE_LENGTH];
};

extern struct InputHeader In; // global input data structure
extern struct OutputHeader Out; // global output data structure
#define TOKEN In.token
#define PREVIOUS In.previous
#define RAM Out.ram[Out.addr]
#define COMMENT Out.comment[Out.addr] // advances to next comment with nextaddr()

/* Inserts s in the LineNode list, after the last node. */
void enqueue(const char *s);

/* Resets tokenization variables, moves the current pointer to the
first line, copies the first token to the TOKEN scratchpad and returns
a pointer to the first line. */
char* firstline();

/* Moves the current pointer to the next line, copies the first token to the
TOKEN scratchpad, and returns a pointer to the current line string. After the
last node, returns NULL and requires a call to firstline() to restart. */
char* nextline(void);

/* Moves to the next token in the current line, copies its characters to the
TOKEN scratchpad and returns a pointer to it. After the last token it returns
NULL. */
```

```
char* nexttoken(void);

/* Allocates memory of the symbol/name element and points the next index of
the Out.symbol array to it. Returns the current number of symbols. */
int addsymbol(const char* name, int value);

/* Sorts the symbols array according their linked SymbolElement names. */
void sortsymbols();

/* Searches name on the names linked on the symbols array and returns its
SymbolElement value. */
int symbolvalue(const char* name);

/* Moves to the next RAM address, checking for out-of-bounds error. */
void nextaddr();

#endif
```

## H.7 data\_structures.c - λειτουργίες δομών δεδομένων

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#include "config.h"
#include "error_handler.h"
#include "data_structures.h"

struct InputHeader In = {0};
struct OutputHeader Out = {0};

void enqueue(const char* s)
{
    struct LineNode* new = malloc(sizeof(*new));
    if (!new) error(MEMORY_ALLOCATION_ERROR);
    // copy the string parameter to the reserved memory space
    strncpy(new->line, s, MAX_LINE_LENGTH);
    new->line[MAX_LINE_LENGTH - 1] = '\0'; // terminate, just to be sure
    new->next = NULL;
    if (!In.front) {
        In.front = new;
    } else {
        In.rear->next = new;
    }
    In.rear = new;
}

char* firstline(void)
{
    In.line_number = 0;
    In.chr = NULL;
    In.token[0] = '\0';
    return nextline(); // process the first line (and get the first token)
}

char* nextline(void)
{
    if (In.line_number == 0) {
        In.current = In.front;
    } else if (In.line_number > 0) {
        In.current = In.current->next;
    }

    if (In.current == NULL) {
        In.line_number = -1; // require restart by firstline() at this point
        In.chr = NULL;
    } else {
```

```

        In.line_number++;
        In.chr = In.current->line;
    }
    nexttoken();
    return In.chr;
}

char* nexttoken(void)
{
    // In.chr++ ⇒ advances to next character
    // *In.chr == '\\' && In.chr[1] == '"' ⇒ escaped quoted \" found
    // In.token[i++] = *In.chr++ ⇒ copies current character, and then advances

    strcpy(In.previous, In.token); // useful for error message context
    /* Token character index; declared as unsigned char because of its
    practical 255 limit. */
    unsigned char i = 0; // token character index
    In.token[0] = '\0';
    // all lines/tokens were processed or line is empty
    if (In.chr == NULL) return NULL;
    if (*In.chr == '\0') return NULL;
    while (isspace(*In.chr)) In.chr++; // skip leading whitespace
    if (*In.chr == '"') {
        // copy all quoted text, including the quotes
        In.token[i++] = *In.chr++; // opening quote
        while (*In.chr && *In.chr != '"') { // until closing quote or terminal
            if (*In.chr == '\\' && In.chr[1] == '"') In.chr++; // escaped quote
            In.token[i++] = *In.chr++;
        }
        if (*In.chr == '\0') error(UNCLOSED_STRING); // no closing quote found
        In.token[i++] = *In.chr++; // closing quote
    } else if (strchr(SINGLE_CHAR_DELIMITERS, *In.chr)) {
        // copy a single-character delimiter
        In.token[i++] = *In.chr++;
    } else {
        // copy all characters until hitting a delimiter or terminal
        while (!strchr(ALL_DELIMITERS, *In.chr)) In.token[i++] = *In.chr++;
    }
    In.token[i] = '\0'; // i++ was performed after the last copy
    return In.token;
}

int addsymbol(const char* name, int value)
{
    if (Out.symbols >= MAX_SYMBOLS) error(MANY_SYMBOLS);
    if (symbolvalue(name) != -1) error(DUPLICATE_SYMBOL);
    Out.symbol[Out.symbols].name = malloc(strlen(name) + 1); // +1 = terminator
    if (!Out.symbol[Out.symbols].name) error(MEMORY_ALLOCATION_ERROR);
    strcpy(Out.symbol[Out.symbols].name, name);
    Out.symbol[Out.symbols].val = (unsigned char)value;
    Out.symbols++;
    return Out.symbols;
}

/* Compares the name field of SymbolElement pointers a and b.
nameA > nameB ≡ 1, nameA < nameB ≡ -1, nameA = nameB ≡ 0. */
int comparesymbols(const void* a, const void* b)
{
    char* nameA = ((struct SymbolElement*)a)->name;
    char* nameB = ((struct SymbolElement*)b)->name;
    return strcmp(nameA, nameB);
}

void sortsymbols(void)
{
    qsort(
        Out.symbol, Out.symbols, sizeof(struct SymbolElement), comparesymbols);
}

```

```
int symbolvalue(const char* name)
{
    struct SymbolElement key = { .name = (char*)name };
    struct SymbolElement* found = (struct SymbolElement*) bsearch(
        &key, Out.symbol, Out.symbols, sizeof(Out.symbol[0]), comparesymbols);
    if (!found) return -1;
    return (int)found->val;
}

void nextaddr(void)
{
    Out.addr++;
    if (Out.addr > RAM_SIZE) error(RAM_LIMIT);
}
```

## H.8 template.vhd - πρότυπο αρχείο VHDL firmware

```
-----
-- E80 Firmware, generated by the E80ASM Assembler
-----

LIBRARY ieee, work; USE ieee.std_logic_1164.ALL, work.support.ALL;
PACKAGE firmware IS
    CONSTANT DefaultFrequency : DECIHERTZ := %d; -- 1 to 1000
    CONSTANT SimDIP : WORD := "%s"; -- DIP input for testbench only
    CONSTANT Firmware : WORDx256 := (
        MACHINE_CODE_PLACEHOLDER
        OTHERS => "UUUUUUUU");END;
```

## H.9 E80ASM.dev - ρυθμίσεις Red Panda Cpp & Dev Cpp

```
[Project]
FileName = E80ASM.dev
Name = E80ASM
Type = 1
Ver = 3
ObjFiles =
Includes =
Libs =
PrivateResource =
ResourceIncludes =
MakeIncludes =
Compiler =
CppCompiler =
Linker =
IsCpp = 0
Icon =
ExeOutput =
ObjectOutput =
LogOutput =
LogOutputEnabled = 0
OverrideOutput = 0
OverrideOutputName =
HostApplication =
UseCustomMakefile = 0
CustomMakefile =
CommandLine =
Folders =
IncludeVersionInfo = 0
SupportXPThemes = 0
CompilerSet = 2
UnitCount = 8
Bins =
ResourceCommand =
UsePrecompiledHeader = 0
PrecompiledHeader =
StaticLink = 1
AddCharset = 1
```

```
ExecEncoding = SYSTEM
Encoding = UTF-8
ModelType = 1
ClassBrowserType = 0
AllowParallelBuilding = false
ParallelBuildingJobs = 0
UseUTF8 = 1

[VersionInfo]
Major = 1
Minor = 0
Release = 0
Build = 0
LanguageID = 1033
CharsetID = 1252
CompanyName =
FileVersion = 1.0.0.0
FileDescription = Developed using the Dev-C++ IDE
InternalName =
LegalCopyright =
LegalTrademarks =
OriginalFilename = E80ASM.exe
ProductName = E80ASM
ProductVersion = 1.0.0.0
AutoIncBuildNr = 0
SyncProduct = 1

[Unit1]
FileName = error_handler.h
CompileCpp = 0
Folder =
Compile = 0
Link = 0
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit2]
FileName = config.h
CompileCpp = 0
Folder =
Compile = 0
Link = 0
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit3]
FileName = parse_functions.c
CompileCpp = 0
Folder =
Compile = 1
Link = 1
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8
```

```
[Unit4]
FileName = data_structures.c
CompileCpp = 0
Folder =
Compile = 1
Link = 1
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit5]
FileName = main.c
CompileCpp = 0
Folder =
Compile = 1
Link = 1
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit6]
FileName = parse_functions.h
CompileCpp = 0
Folder =
Compile = 0
Link = 0
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit7]
FileName = data_structures.h
CompileCpp = 0
Folder =
Compile = 0
Link = 0
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = UTF-8

[Unit8]
FileName = error_handler.c
CompileCpp = 0
Folder =
Compile = 1
Link = 1
Priority = 1000
OverrideBuildCmd = 0
BuildCmd =
FileEncoding = PROJECT
RealEncoding = ASCII

[Unit9]
FileName = syntax_functions.h
CompileCpp = 0
```



```
Folder =  
Compile = 0  
Link = 0  
Priority = 1000  
OverrideBuildCmd = 0  
BuildCmd =  
FileEncoding = PROJECT  
RealEncoding = UTF-8  
  
[Unit10]  
FileName = main.c  
CompileCpp = 0  
Folder =  
Compile = 1  
Link = 1  
Priority = 1000  
OverrideBuildCmd = 0  
BuildCmd =  
FileEncoding = PROJECT  
RealEncoding = UTF-8  
  
[CompilerSettings]  
c_cmd_opt_std = c99  
cc_cmd_opt_check_iso_conformance = on  
cc_cmd_opt_stack_protector = protector-all  
cc_cmd_opt_use_pipe = on  
cc_cmd_opt_warning_all = on  
cc_cmd_opt_warning_extra = on  
link_cmd_opt_stack_size = 12  
link_cmd_opt_strip_exe = on
```

## Παράρτημα Θ: Σενάρια αυτοματοποίησης του Notepad++

### Θ.1 Ανάθεση πλήκτρου συντόμευσης μεταγλώττισης & προσομοίωσης

Απο το μενού *Plugins > Plugins Admin*, εγκαθιστούμε το NppExec. Απο το μενού *Plugins > NppExec > Execute NppExec Script* μας δίνεται η δυνατότητα να επικολλήσουμε το παρακάτω σενάριο και να το αποθηκεύσουμε (Save...) με ένα όνομα, έστω "E80 Assembler".

Η μεταβλητή project πρέπει να περιέχει τον φάκελο οπου αποσυμπιέσαμε το όλο έργο.

```
// hide console messages
npp_console local -
npe_console local v+ m-
npp_console local +

// save current assembly
npp_save

// set local paths to local variables
set local project = C:\E80 // change to the folder of your choice!
set local asm = "$(FULL_CURRENT_PATH)"
set local vhd = "$(project)\VHDL\Firmware.vhd"
set local assembler = "$(project)\Assembler\e80asm.exe"

// change directory (and drive) to the E80ASM location
cd "$(project)\Assembler"
// run and show the assembler command line
set local commandline = $(assembler) /Q < $(asm) > $(vhd)
echo $(commandline)
cmd /c " $(commandline) "

if $(EXITCODE) == 0 then
    // assembly successful, run GHDL/GTKWave
    cd "$(project)\GHDL"
    set local commandline = "computer_tb.bat"
    echo $(commandline)
    cmd /c " $(commandline) "
else
    // assembly error
    // find the word line before the number (format being "line (number) :")
    set local line_start ~ strfind "$(OUTPUT)" "line"
    if $(line_start) > 0 then // the error message contains a line
        // mark the start of the number after the word "line "
        set local line_start ~ $(line_start) + 5
        // find the colon marking the end of the line number
        set local line_end ~ strfind "$(OUTPUT)" ":"
        // calculate the length of the line number (# of digits)
        set local line_len ~ $(line_end) - $(line_start)
        // get the line number from the start to the length
        set local line_num ~ substr $(line_start) $(line_len) "$(OUTPUT)"
        // subtract one because Notepad++/Scintilla starts from 0
        set local line_num ~ $(line_num) - 1
        // move the active line to the erroneous line
        sci_sendmsg SCI_GOTOLINE $(line_num)
    endif
endif
```

Πηγαίνουμε στο μενού *Plugins > NppExec > Advanced Options...* επιλέγουμε *Associated script > E80 Assembler > Add/Modify* και κάνουμε κλικ στο *Place to the Macros submenu*. Δίνουμε OK και επανεκκινούμε το Notepad++. Παρατηρούμε οτι η λειτουργία E80 Assembler έχει περαστεί στο μενού Macro. Απο *Settings > Shortcut mapper* θα την εντοπίσουμε στην καρτέλα *Plugin commands* και μπορούμε πλέον να της αναθέσουμε το επιθυμητό πλήκτρο συντόμευσης.

## Θ.2 Χρώματα σύνταξης

Απο το μενού *Language > User Defined Language > Define your language* επιλέγουμε *Import...* και ανοίγουμε το αρχείο “E80 Assembly Syntax Coloring.xml” που περιέχει τον εξής κώδικα:

```
<NotepadPlus> <UserLang name="E80 Assembly" ext="asm" udlVersion="2.1">
<Settings>
  <Global caseIgnored="yes" allowFoldOfComments="no" foldCompact="no"
forcePureLC="0" decimalSeparator="0" />
  <Prefix Keywords1="no" Keywords2="no" Keywords3="no" Keywords4="no"
Keywords5="no" Keywords6="no" Keywords7="yes" Keywords8="no" />
</Settings>
<KeywordLists>
  <Keywords name="Comments">00; 01 02 03 04</Keywords>
  <Keywords name="Numbers, prefix1">0b $</Keywords>
  <Keywords name="Numbers, prefix2">0x</Keywords>
  <Keywords name="Numbers, extras1">A B C D E F a b c d e f</Keywords>
  <Keywords name="Operators1">,>, : [ ]</Keywords>
  <Keywords name="Operators2"><</Keywords>
  <Keywords name="Keywords1">HLT NOP RETURN RSHIFT LSHIFT PUSH POP JC JNC JZ JNZ
JS JNS JV JNV CALL JMP MOV ADD ROR SUB CMP AND OR XOR LOAD STORE BIT</Keywords>
  <Keywords name="Keywords2"> .TITLE .NAME .FREQUENCY .SIMDIP .DATA</Keywords>
  <Keywords name="Keywords3">R0 R1 R2 R3 R4 R5 R6 R7 SP FLAGS</Keywords>
  <Keywords name="Delimiters">00&quot; 01&quot; 02&quot; 03&apos; 04&apos;
05&apos; 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23</Keywords>
</KeywordLists>
<Styles>
  <WordsStyle name="DEFAULT" fgColor="000000" bgColor="FFFFFF" fontStyle="0"
nesting="0" />
  <WordsStyle name="COMMENTS" fgColor="008000" bgColor="FFFFFF" fontStyle="2"
nesting="0" />
  <WordsStyle name="LINE COMMENTS" fgColor="B0B0B0" bgColor="FFFFFF"
fontStyle="2" nesting="0" />
  <WordsStyle name="NUMBERS" fgColor="B05800" bgColor="FFFFFF" fontStyle="0"
nesting="0" />
  <WordsStyle name="KEYWORDS1" fgColor="0C52C7" bgColor="FFFFFF" fontStyle="0"
nesting="0" />
  <WordsStyle name="KEYWORDS2" fgColor="C6397C" bgColor="FFFFFF" fontStyle="0"
nesting="0" />
  <WordsStyle name="KEYWORDS3" fgColor="5A9A6A" bgColor="FFFFFF" fontStyle="1"
nesting="0" />
  <WordsStyle name="OPERATORS" fgColor="000080" bgColor="FFFFFF" fontStyle="1"
nesting="0" />
  <WordsStyle name="FOLDER IN CODE1" fgColor="000000" bgColor="FFFFFF"
fontStyle="0" nesting="0" />
  <WordsStyle name="FOLDER IN CODE2" fgColor="000000" bgColor="FFFFFF"
fontStyle="0" nesting="0" />
  <WordsStyle name="FOLDER IN COMMENT" fgColor="000000" bgColor="FFFFFF"
fontStyle="0" nesting="0" />
  <WordsStyle name="DELIMITERS1" fgColor="FF8D1C" bgColor="FFFFFF" fontStyle="0"
nesting="0" />
</Styles>
</UserLang> </NotepadPlus>
```

Σημειώνεται ότι τα χρώματα έχουν διαμορφωθεί σύμφωνα με τις ανάγκες δυσχρωματοψίας του συγγραφέα. Αλλάζουν εύκολα μέσω της λειτουργίας *Define your language*.

## Παράρτημα Ι: Ιστορικό αποθετηρίου [github.com/Stokpan/E80](https://github.com/Stokpan/E80)

```
> git log --all --reverse
```

```
commit 1e8f0af34022baecb9baf304d00f77162d4587e
Author: Stokpan <std145225@ac.eap.gr>
Date: Sun Jan 26 09:11:14 2025 +0200
```

Initial commit

```
commit a4831b49fa16a673b71ada33ccddcd261dd7f1956
Author: Stokpan <std145225@ac.eap.gr>
Date: Mon Jan 27 20:13:53 2025 +0200
```

Version 1.2 - GHDL and DSD-i1 LEDs

- \* Included GHDL batch files
- \* Updated test benches
- \* Made full use of the DSD-i1 LEDs

```
commit 5d645d205a257f17deb896b063745f43272f540b
Author: Stokpan <std145225@ac.eap.gr>
Date: Wed Jan 29 21:43:53 2025 +0200
```

Version 1.5 - 8-bit DIP input

- \* 8-bit DIP input in FF address
- \* SP initialized to FF on reset
- \* A single g.bat file handles the GHDL simulation
- \* Added error handling on the g.bat file
- \* Modified to test bench batch files to use the g.bat instead
- \* Improved the CPU c.do (ModelSim) and cpu\_tb.gtkw (GHDL) wave configs with better labelling and to include the DIP Input
- \* Added a short explanation of the clock conversion math in the ClockConverter
- \* Renamed CPU\_Quartus to CPI\_DSDi1 to correctly reflect its purpose
- \* Pressing the reset button will now display the DIPinput on row C
- \* Removed the 2's complement of the SP from the top CPU entity; it's no longer necessary since the SP initializes to FF and therefore it's easy to read in reverse LEDs

```
commit 637ac429590acc56f8b9f83473bacdd11fa22d3d
Author: Stokpan <std145225@ac.eap.gr>
Date: Mon Feb 3 21:16:59 2025 +0200
```

Version 2.0 - Redundancy cleanup

- \* Relocated the Flags Register to the Register File
- \* An extra port for POP is no longer necessary on the Register File
- \* Removed the redundant Datapath & ControlUnit components
- \* All the logic of removed components has been relocated to the CPU
- \* All components and logic in the CPU has been divided in clear sections
- \* Signal naming has been improved and streamlined everywhere for readability.
- \* Updated GHDL/GTKWave config files with labels and separators
- \* Re-organized the DSD-i1 LED assignment
- \* Block comments have been replaced with ieee-styled line comments

```
commit a4f4d4610e6d6d8f1bf055ac92ef35c1315e5b74
Author: Stokpan <std145225@ac.eap.gr>
Date: Wed Feb 5 21:10:48 2025 +0200
```

Version 2.4 - 5D Joystick & better LED display

- \* 5D joystick board support
- \* Reset button implemented at the 5D joystick board
- \* Implemented a Pause function with the 5D board SET button
- \* LED row B shows the current register value

- \* Up & Down joystick buttons control the current register address
- \* Current register address displayed on row A LEDs instead of the SP (which can be displayed with the Joystick)
- \* The clock led will be shown in full brightness after Reset has been completed to signify that the firmware has been uploaded
- \* Implemented a Repeat Delay for the Joystick buttons
- \* Modified the timings file to reflect the new clock-driven signals (repeat delay and reset complete)
- \* Integrated the Clock converter into the DSDi1 CPU entity
- \* The entire register file is now passed for LED output; this simplifies the RegisterFile and CPU code too

commit 57b3973c54a16b0c197b0ac0c5a3af0960f7fb55

Author: Stokpan <std145225@ac.eap.gr>

Date: Fri Feb 7 01:30:26 2025 +0200

#### Version 2.7 - Sign, Overflow and speed control

- \* Added support for the Sign and Overflow flags at the Full Adder, the CPU and the display LEDs
- \* Added JV, JNV, JS, JNS instructions to support the new flags
- \* Simplified the Flags update logic at the ALU: either all flags are set, or remain unchanged
- \* Updated testbenches to include the new flags
- \* Added support for frequency control via the Joystick; all Joystick pins are now mapped to the DSD-il
- \* Improved the register display control via the Joystick
- \* Set nCEO as regular I/O pin instead of programming to make it available for the Joystick pin header
- \* Fixed the Joystick repeat rate at the DSD-il implementation

commit e8f01783226488303a5ad296f39f4ce3fc890b55

Author: Stokpan <std145225@ac.eap.gr>

Date: Sat Feb 8 13:42:02 2025 +0200

#### Version 2.8 - Improved flags and removed buffers

- \* Flags resetting better matches 8085 & 6502 ISA
- \* Shift instructions set the overflow flag according to 68000
- \* Thanks to 2.0 redundancy cleanups, buffers can be reverted to output signals
- \* Spellchecked all comments
- \* Added a Readme with the finalized ISA

commit 64009df0ba99083229594e0d4c56eeced13a1cdb

Author: Stokpan <std145225@ac.eap.gr>

Date: Sun Feb 9 20:38:16 2025 +0200

#### Version 3.0 - Final Touches

- \* Vastly improved comments for accuracy and readability
- \* Added a batch file for GHDL Synthesis; very useful for Xilinx
- \* Improved the GHDL simulation batch files
- \* Slightly improved performance of the ALU
- \* Removed ButtonDown constant because it would complicate the Joystick and DIP assignments; active High is assumed everywhere
- \* Removed 1-bit full adder; it's now generated in the 8-bit full adder
- \* Renamed some signals in the ALU to make their roles clear

commit flcf04ad6820e74e3a2de210f3e3caffec8bffa

Author: Stokpan <std145225@ac.eap.gr>

Date: Tue Feb 11 14:07:19 2025 +0200

#### Version 3.1 - Improved GHDL batch and readability

- \* Additional error checks in g.bat
- \* Improved comments
- \* Better explanation for the write resolution when A\_reg = W\_reg
- \* Corrected CMP flags in the readme

- \* Switched to hexadecimal system when it improves readability
- \* Rewrote the LED register address using WITH/SELECT (easier to read and improves performance in Quartus)

commit 97dc0b904ccd41bc87b4359d4ealf15a95910f9d  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Tue Mar 4 11:12:55 2025 +0200

#### Version 4.0 - Assembler

- \* Reads test.asm
- \* Collects symbols
- \* Syntax check according to bnf.txt
- \* Generates machine code
- \* Pretty prints output

#### Remaining

- \* Output in file
- \* Support frequency
- \* Testing

commit be3fc88e1abd5c01df3ca6310441fbb0cb6e9b93  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Tue Mar 18 23:54:41 2025 +0200

#### Version 5.0 - feature complete

- \* Reviewed, reformatted and added comments to C code
- \* Improved the BNF syntax to remove many redundant rules
- \* Stdin/out is fully supported
- \* Cleaned up error messages
- \* Remove the context parameter from the error handler function
- \* Added a PREVIOUS string to the output data structure to provide context for some errors
- \* Added welcome message with special directives for Windows & Unix
- \* Renamed Frequency to deciHertz
- \* Cleaned up comments on some VHDL files
- \* Increased simulation duration on the cpu\_tb GHDL batch
- \* Relocated function descriptions to the header files

commit 08663d98be2d3fdc8af9bc102e8fdee69c428f18  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Wed Mar 19 14:43:08 2025 +0200

#### Version 5.1 - final cleanups

- \* Corrected a possible undefined behavior in uppercase()
- \* Corrected type casting on addsymbol
- \* Removed toupper/isspace macros
- \* Fixed old-style C declarations
- \* Fixed typos

commit a8d08b5133bd2b15721ea71f730bbd35072012e2  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Sat Mar 22 23:54:56 2025 +0200

#### Version 5.12 - Minor improvements

- \* Ctrl-D for end of transmit
- \* Symbol collection report

commit c9cf8fa499ff800d3704fcd5a0fa80e8b38e658  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Mon Mar 24 01:26:50 2025 +0200

#### Version 5.13 - Testing and fixes

- \* fixed SP & FLAGS aliases

```
* Dev Cpp pass
* Red Panda pass
* Division test pass
* Number conversions pass
* Template pass
* Git reset pass
```

```
commit 323b75f9dd5430d8240cf6650b6d63460aa26e40
Author: Stokpan <std145225@ac.eap.gr>
Date: Thu Apr 3 20:32:44 2025 +0300
```

Version 6.0 - Gowin - Tang Primer 25K support

```
* Added a Gowin project; no VHDL changes!
* Added support for the Sipeed Tang Primer 25K
* GTKWave save file includes complete (folded) RAM dump
```

```
commit 13e5bea7f4569d7d8041378a8848be9c6dfff6a04
Author: Stokpan <std145225@ac.eap.gr>
Date: Sat Apr 12 00:37:22 2025 +0300
```

Version 6.1 - Tang Primer 25K DIP input mappings

```
commit 79dd3c2d4a70487433ab01a0316cb6da9b64a34f
Author: Stokpan <std145225@ac.eap.gr>
Date: Sun Apr 20 21:24:19 2025 +0300
```

Version 6.2 - Frequency naming fixes

```
* Streamlined the Frequency name conventions
* Renamed the .DECIHERTZ directive to .FREQUENCY
* Improved the frequency limit enforcing at the VHDL code
```

```
commit d2ca41dc985cd5d6a60dc5553600b1e60b2bf71c
Author: Stokpan <std145225@ac.eap.gr>
Date: Tue Apr 22 00:58:31 2025 +0300
```

Version 6.3 - Replaced ADC with ROR

```
* ROR was found to be much more frequently used in education & in practice
* Applying rotation by shifting is much more of a hassle than applying ADC with
  ADD & JNC
```

```
commit 2aaedd8e966491ca2a8ea9e5186e9a573d625294
Author: Stokpan <std145225@ac.eap.gr>
Date: Thu Apr 24 01:36:41 2025 +0300
```

Version 6.4 - Improved FA, added Barrel shifter

```
* Removing ADC means that FA can now do subtraction since Cin is now used
  exclusively for selection. ALU is slightly simplified with this because it no
  longer needs to handle complement logic by its own.
* Streamlined the rotation logic of the ALU into a clean separate Barrel
  shifter.
```

```
commit 986cee4bf5bd94dfcaa2be03746b901e57f82634
Author: Stokpan <std145225@ac.eap.gr>
Date: Thu Apr 24 11:28:04 2025 +0300
```

Version 6.5 - Completed Readme and revised FA8

```
* Completed the Readme with full descriptions and cheatsheets
* Added subtraction logic to the FA8
* Relocated subtraction logic from the ALU to the new FA8
* Updated comments to better explain ROR logic
* Improved the comments on the ALUop decoder
```

```
commit 2e12bdf1c433d75875272fccdc7f09b02e64f219
```



Author: Stokpan <std145225@ac.eap.gr>  
Date: Sun Apr 27 01:43:11 2025 +0300

Version 6.8 - Ascend-descend division & sequential logic elimination (except for DFFs)

- \* Removed all sequential PROCESS blocks from the VHDL except for the DFF. All storage in the CPU (Registers, Program Counter, RAM) is implemented by the simple DFF
- \* Replaced the old assembly test file with a new one that combines multiplication and division, using a division algorithm of my own... devise (as far as I can tell).
- \* Reset values are set to Undefined instead of Don't Cares; Undefined matches 0 and translates to NOP instead of causing random jumps when execution lands in Don't Care space
- \* Fixed OR size in parse\_functions.c
- \* Improved the Readme to add some tips on the Assembly Cheatsheet
- \* Removed fluff from template.vhd
- \* Replaced "DSDi1" references to "FPGA". The Tang Primer 25K has been fully tested and my design no longer targets a specific FPGA
- \* Added registers R3-R5 to ModelSim c.do batch file

commit d818dc93096ec79ae2caf6a585b24f0bae5be50d  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Sun Apr 27 21:21:49 2025 +0300

Version 7.0 - HLT implemented

- \* Added the HLT instruction to BNF, Assembler and VHDL; HLT pauses execution, stops flashing the CLK LED, and finishes simulation on GHDL and ModelSim
- \* Improved the cpu testbench to stop simulation when HLT is executed, which makes waveform inspection much easier (eg. zoom to fit now works for the whole execution instead of an arbitrary duration)
- \* Added a 5th flag, "H"
- \* Fixed max frequency in the assembler
- \* Fixed .FREQUENCY directive in BNF.txt
- \* Fixed BIT belonging to two <instr> groups in BNF.txt
- \* Reordered <instr> groups in BNF and VHDL by order of cardinality
- \* Started work on a diagnostics program; so far so good
- \* Started removing special customizations for the DSDi1

commit 6500bdb4a84956b2fd12aed19a1e8bdbdf28051a  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Mon Apr 28 00:05:03 2025 +0300

Version 7.1 - Removed DSDi1-specific edits

- \* All edits targeting DSDi1 have been removed; all VHDL is now FPGA-neutral
- \* A few edits on the pinouts of both Tang Primer 25K and DSD-i1 were necessary to support this change

commit 77f30e06b0bccaa878502cad283154afd5b8d74c  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Mon Apr 28 00:11:22 2025 +0300

Version 7.12 - Minor edits while writing Thesis

- \* divmul.asm improved in both implementation and comments
- \* Readme fixes and inclusion of Flags cheat sheets
- \* Simplified the method in which HLT freezes the PC

commit b0ce04f30e99b96f9cf0378ee70ee537ea4afa33  
Author: Stokpan <std145225@ac.eap.gr>  
Date: Tue Apr 29 13:33:52 2025 +0300

Version 7.5 - JMP reg implemented; RAM defaults to HLT

- \* Set HLT to 0b00000000 to take advantage of RAM defaulting to zeroes, thereby causing execution of uninitialized addresses to freeze
- \* Implemented JMP reg (indirect addressing)
- \* NOP is set to 0b00000001 and some Jx instructions reordered as well
- \* Improved assembler's bitcopy to match VHDL's DOWNT0
- \* In match(a,b) use b to define range; this helped remove some redundant checks in the CPU code
- \* Simplified some parts of the VHDL code
- \* Fixed H bit not Halting
- \* Replaced don't cares with zeroes on the Readme
- \* Added an FAQ section on the Readme

commit 58ba72318a30e0e6bde57dc96b632930fd5f5cad

Author: Stokpan <std145225@ac.eap.gr>

Date: Fri May 2 15:57:50 2025 +0300

#### Version 7.9 - Flags, syn\_keep, and FA8 componentization

- \* Flags now follow a 6502-inspired rule: unchanged for non-register-modifying ops (including stack ops used mid-sequence), always Z/S, with C/V only on arithmetic/shifts (except ROR, which preserves C/V)
- \* Added syn\_keep attribute for ALUinB to fix the false logical loop positive in Gowin. Quartus cannot be fixed without introducing a redundant dependency.
- \* FA8 1-bit adder part has been split to a separate entity on the same file. This greatly improves the RTL viewer diagram output.
- \* Removed redundant aliases from CPU.vhd
- \* Streamlined addressing modes
- \* Streamlined flag descriptions
- \* RegisterArray renamed back to RegisterFile because the latter is far more common
- \* Comments updated to reflect on Thesis

commit e8bca244965829870398413475c5d74d72af9fd3

Author: Stokpan <std145225@ac.eap.gr>

Date: Mon May 5 09:07:04 2025 +0300

#### Version 8.0 - Thesis-related changes

- \* Componentized instruction decoder in CPU
- \* Removed MEM\_ADDR subtype, it's really redundant since |MEM\_ADDR|=|WORD|
- \* Added Instr1 & Instr2 to the GTKWave preset, matches with ModelSim
- \* Replaced DOWNT0 comments with Little Endian
- \* Renamed test to DUT in testbenches (Harris&Harris)
- \* After simulation finishes, ModelSim opens RAM tab and then switches to the wave tab

commit 6fe59f87392a8c4f5a54a9839f48cbce402c339b

Author: Stokpan <std145225@ac.eap.gr>

Date: Fri May 9 01:39:13 2025 +0300

#### Version 8.5 - QElectroTech, Componentization Revert, & Cleanups

- \* Added QElectroTech designs in folios
- \* Reverted Thesis-related componentization (makes VHDL harder to read)
- \* Overflow renamed to V
- \* Finely tuned zoom in c.do (ModelSim)
- \* Clarified FA8 output in ALU
- \* Improved Halt flag setting
- \* Removed JMP-exclusive format; fits perfectly in current types
- \* Removed redundant library from Assembler
- \* Removed ALU\_OPCODE type, cleaned up the logic of match()
- \* All flags are reset to undefined except for Halt which is zeroed

commit 3868919d7af3a938077003e7aab1977694c3c775

Author: Stokpan <std145225@ac.eap.gr>

Date: Sun Jun 8 00:13:41 2025 +0300

#### Version 8.8 - Computer.vhd

- \* Separated RAM from CPU
- \* Interconnected RAM with CPU on Computer.vhd
- \* Relocated Mem/DIP selection on Computer.vhd
- \* Greatly improved QElectroTech diagrams
- \* ModelSim no longer opens a vhd code tab after detecting Halt

commit 29cec75a09d8176768113227595f7d553670967f

Author: Stokpan <std145225@ac.eap.gr>

Date: Tue Jun 17 17:55:49 2025 +0300

#### Version 9.1 - Final Presentation Version

- \* Added brackets on LOAD/STORE
- \* Added .SIMDIP directive for setting DIP input on the testbench
- \* Set a bus width on QET diagrams
- \* Renamed AdjInstr to Adjacent and isReg1Reg2 to op2isReg (easier to understand)
- \* .DATA character comments now include ANSI value
- \* Improved clock logic in testbench to allow clock to match current step
- \* Set Halt to undefined prior to Reset in the testbench
- \* Added compileall to ModelSim c.do script
- \* Fixed reversed ALU isDCR/isINR comments
- \* Reformatted VHDL Firmware template
- \* Integrated FAQ into the Readme; FAQ section removed
- \* Improved LOAD/STORE description on the Assembly cheatsheet
- \* Removed redundant Gowin files
- \* Streamlined comments with documentation
- \* Added Uppercase.asm example
- \* Set the title placeholder on top
- \* Made the Template title default if no .TITLE directive is provided
- \* Added a check against writing .DATA in program space
- \* Removed redundant entries from error\_handler
- \* Cleaned up ModelSim project file, ready for publishing
- \* Added RAM row in ModelSim (mouse hover)
- \* Improved Ascend/Descend algorithm
- \* Removed SST bar from GTKWave (--rcvar "hide\_sst on")
- \* Fixed Assembler error when parsing a comment after directives
- \* Added ror.asm for FPGA testing

commit 46e4e29779feb4f1f934c5de1e7288da4800dd

Author: Stokpan <std145225@ac.eap.gr>

Date: Sun Jun 22 21:13:43 2025 +0300

#### Version 9.2 - Notepad++ integration just before the presentation

- \* Added an NppExec script for Notepad++ to allow assembly & simulation via a hotkey
- \* Added a syntax coloring XML declaration for Notepad++
- \* Added /Q switch to E80ASM to support NppExec
- \* HLT size is now properly set to 1 instead of 0; it was a hack
- \* HLT / Halt freezes the PC at the PCnext selection
- \* QelectroTech diagrams corrected to reflect on the improved HLT logic
- \* .SIMDIP additions to Readme
- \* Fixed Mhz to MHz
- \* Removed Reset signal from computer\_tb.gtkw to match with ModelSim
- \* Updated compiler set to MinGW-w64 GCC 11.5.0 32-bit release

### Υπεύθυνη Δήλωση Συγγραφέα

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.