Υλοποίηση μικροπρογραμματιζόμενου Συστήματος σε VHDL

Κούτρας Απόστολος
Επιβλέπων : Καβουσιανός Χρυσοβαλάντης

11 Δεκεμβρίου 2012

Περίληψη

Το αντικείμενο της παρούσας πτυχιακής εργασίας είναι η υλοποίηση σε VHDL ενός σχετικά απλού μικροπρογραμματιζόμενου συστήματος.

Μέσα απο το σύστημα αυτό θα τεθούν οι βάσεις για την διασαφήνιση βασικών εννοιών της αρχιτεκτονικής υπολογιστών όπως microprocessor, microcode, microprogramming, microsequencer. Κυρίως θα μελετηθεί ο μικροπρογραμματισμός αυτός καθέ αυτός από την οπτική γωνία του προγραμματιστή καθώς και απο την οπτική γωνία του μηχανικού των υπολογιστών.

Το σύστημα αυτό είναι αρκέτα εύκολο να κατανοηθεί αναλύοντας τον κώδικα της VHDL που το συνθέτει, έτσι ώστε να αποτελέσει βάση για επόμενα projects καθώς και να χρήσιμοποιηθεί αυτό καθέ αυτο ως εκπαιδευτικη πλατφορμα για τους ενδιαφερόμενους φοιτητές της πληροφορικής στον τομέα του μικροπρογραμματισμού.

Το παρών συστημα βασίζεται σε πρωτότυπο σύστημα, που υλοποιήθηκε σε τυπωμένο κύκλωμα και περιγράφεται στην διπλωματική εργασία των κυρίων Αγγελή Δημήτρη, Καλαματιανού Γιάννη και Καλαμπούκα Λάμπρου υπό την επιβλεψη του Κυρίου Δημήτρη Νικολού στο τμήμα Μηχανικών και Πληροφορικής της Πολυτεχνικής σχολής Πατρών. Στην δεδομένη διπλωματική εργασία εγινε σχεδιασμός μικροπρογραμματιζόμενου συστήματος με δομικά κυκλώματα τα "2901 ALU" και "2909 Microsequencer" της Texas instruments.

Η αρχική αρχιτεκντονική του συστήματος διατηρήθηκε όσο το δυνατόν πιστότερα και παράλληλα έγιναν παραδοχές και τροποποιήσεις όπου κρίνονταν αναγκαίο.

Η σχεδίαση και υλοποίηση του συστήματος έγινε στην εκπαιδευτική πλακέτα FPGA, DE2 της οικογένειας Cyclone 2 της Altera.

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

Σκοπός του κεφαλαίου αυτού είναι να κάνει μια εισαγωγή βασικα στοιχεία αρχιτεκτονικής και να εισαγάγει τον αναγνώστη στην φιλοσοφία των υπολογιστών. Επίσης αναφέρει τάσεις και έννοιες που ώθησαν στην χρήση μικροπρογραμματισμού και τον θεμελίωσαν ως τεχνική κατά την κατασκευή των μικροεπεξεργαστών.

1.1 Αρχιτεκτονική

Η αρχιτεκτονική είναι η πρακτική τέχνη του καθορισμού της δομής και της αλληλεπίδρασης των υπομονάδων που απαρτίζουν έναν υπολογιστή.

Ο ηλεκτρονικός υπολογιστής είναι μια συσκευή γενικής χρήσης ικάνη στο να επεξεργάζεται και να αποθηκεύει δεδομένα σε δυαδική μορφή δοσμένα υπο την μορφή προγράμματος κάθε φορά διαφορετικού ανάλογα με την εφαρμογή. Ένας υπολογιστής έχει την δυνατότητα να διαχειρίζεται πληροφορία απο περιφερειακά τόσο στην αποθήκευση όσο και στην ανάκτησή της.

Αρχικά θα δώσουμε μια βασική περιγραφή της αρχιτεκτονικής ξεκινώντας απο την ευρύτερη κατηγορία στην οποία ανήκει το σύστημά μας. Αυτή ειναι ο Stored Program Computer.

1.1.1 **Stored Program Computer**

Stored Program Computer είναι μια μηχανή ικανή να εκτελέσει εντολές προγράμματος οι οποίες είναι αποθηκεύμενες σε ηλεκτρονική μνήμη.

Ιστορικά στοιχεία

Οι υπολογιστές που μπορούν να διαχειριστούν αποθηκευμένο πρόγραμμα γνωστοποιήθηκαν απο την δεκαετία του 40 και επέφεραν επανάσταση αφού ήταν

και οι πρώτοι οι οποίοι μπορούσαν να διαχειριστούν real-time δεδομένα.

Χαρακτηριστικό παράδειγμα ο MIT Whirlwind ο οποίος λειτούργησε για στρατιωτική χρήση ως flight simulator και έθεσε έμμεσα τις βάσεις για την μετέπειτα πορέια των υπολογιστών. Ηταν ο πρώτος ο οποίος δεν ήταν ηλεκτρονική εκδοχή των προγενέστερων μηχανικών συστήματων.

Επίσης αποτελούσε εξαίρεση των υπολογιστικών συστήματων της εποχής καθώς διαχειριζόταν bit δεδομένων με παράλληλο τρόπο και όχι σειριακά δηλαδή τροφοδοτώντας bit προς bit για τους υπολογισμούς. Επιπλέον έκανε χρήση της control store μιάς δισδιάστατης δομής για την αποθήκευση των control signals (σημάτων ελέγχου).

1.1.2 Αρχιτεκτονική **Von Neuman**

Στον παραπάνω ορισμό του Stored Program Computer δεν αναφέρεται ρητά αν τα δεδομένα αποθηκεόνταιν στην ίδια μνήμη με το πρόγραμμα παρόλο που στις ημέρες μας κάτι τέτοιο θεωρείται αυτονόητο.

Ενα υποσύνολο της αρχιτεκτονικής Stored Program Computer ειναι η αρχιτεκτονική Von Neuman σύμφωνα με την οποία και τα δεδομένα είναι αποθηκευμένα στην μνήμη.

Κατά σύμβαση, ένας ηλεκτρονικός υπολογιστής διαθέτει κατ΄ ελάχιστο, μια κεντρική μονάδα επεξεργασιασίας CPU (Central Processing Unit),μνήμη, καθώς και μονάδες εισόδου εξόδου (I/O devices) για την επικοινωνία με τον εξωτερικό κόσμο.

Η CPU χωρίζεται λειτουργικά σε δψο τμήματα, στο Operative/Processing Unit και στο Control Unit .

Το operative unit τμήμα είναι υπεύθυνο για τον χειρισμό των δεδομένων, δηλαδή την ανάκτηση καθώς και για τον μετασχηματισμό τους και τέλος την αποθέκευσή τος.

Το Control unit οδηγεί το Operative Part ρυθμίζοντας την ροή των εντολών που θα εκτελεστούν και ενεργοποιώντας κατάλληλα τις γραμμές ελέγχου της. Τροφοδοτείται τόσο απο το execution part χρησιμοποιώντας το αποτέλεσμα του μετασχηματισμού όσο και από την αποθηκευμένη πληροφορία.

1.2 **Processing Unit**

Η μονάδα επεξεργάσιας, εκτελεί τις αριθμητικές και λογικές πραξεις, συγχρονιζόμενη απο το τμήμα που ρυθμίζει την αλληλουχία εντολών που εκτέλουνται που καλείται control unit . Η Control Unit δύναται να μεταβάλλει την σείρα των εκτελούμενων εντολών βάση της αποθηκευμένης πληροφορίας.

Το αποτέλεσμα των πράξεων δύναται να αποθηκευτεί στην μνήμη η στους καταχωρητές από όπου μπορεί να ανασυρθεί αργότερα. Ως μνήμη ορίζεται η φυσική μονδάδα για την αποθήκευσης των προγραμμάτων. Μνήμη μπορει να ονομαστούν η κύρια μνήμη, βοηθητική μνήμη, εξωτερικά μέσα αποθέκευσης κ.α. Αποτελείται από κύτταρα, ικάνα να αποθηκεύσουν το 1 η το 0 και συχνά υλοποιούνται με flip-flops.

Το μικρότερο τμήμα πληροφορίας που μπορεί να διαχειριστεί η CPU είναι η λέξη, η οποία είναι ένα σύνολο bit.

Η μνήμη επίσης οργανώνεται σε λέξεις με κάθε λέξη να είναι μια ομαδοποίηση bit με μήκος δύναμη του δυο. Συνήθεις τιμές μήκους λέξης είναι απο 8 ως 128 bit.

Η CPU έχει την δυνατότητα να επικοινωνεί με την μνήμη ανταλάσσοντας λέξεις. Στην απολούστερη υλοποίηση μία λέξη ανα εγγραφή η ανάγνωση.

Η CPU , συνήθως διαβάζει η γράφει μία λέξη κάθε φορά μόλις καθορίσει την διεύθυνσή της στην μνήμη. Για τον λόγο αυτόν χρησιμοποιεί δύο καταχωρητές, έναν καταχωρτητή διεύθυνσης και έναν καταχωρητή δεδομένων.

Ο καταχωρητής διεύθυνσης η αλλιώς MAR (Memory Address Register) περιέχει την τρέχουσα διεύθυνση μνήμης που επεξεργάζεται η CPU και ο καταχωρητής δεδομένων η αλλίως MDR (Memory Data Register) περιέχει τα δεδομένα που βρίσκονται στην δεδομένη διεύθυνση μνήμης.

Για την ανάγνωση η CPU αρχικοποιεί τον MAR με την διεύθυνση της μνήμης από την οποιία θα γίνει ανάγνωση. Στην περίπτωση που ο MAR είναι 8 bit t'ote sthn CPU είναι διαθέσιμες 256 θέσεις μνήμης.

Αφού τεθεί ο MAR, η CPU θα πρέπει να περιμένει εως ότου η μηήμη ΄προλάβει΄ να προσκομίσει τα δεδομένα που της ζητήθηκαν. Συνήθως το σύστημα είναι χρονισμένο ώστε στον επόμενο κύκλο ρολογιού τα δεδομένα να είναι διαθέσιμα.

Τα διαθέσιμα δεδομένα αυτά, δηλαδή το περιεχόμενο της διεύθυνσης μνήμης στην οποία υποδείκνυε ο MAR θα αποθηκευτούν στον καταχωρητη δεδομένων MDR.

Στην Εγγραφή ακολουθείται ανίστροφη διαδικασία, δηλαδή η CPU θέτει τον MDR με τα δεδομένα προς εγγραφή. Σε επόμενο βήμα τίθεται και ο MAR υποδεικνύοντας σε ποια διεύθυνση θα γίνει η εγγραφή.

Συνήθως η CPU ενεργοποιεί και ένα επιπλέον σήμα επίτρεπσης γνωστό ως WE (Write Enable) το οποίο ενεργοποιεί την εγγραφή.

Γενικά το προιον της επεξεργασίας της operation unit εκτός απο την μνήμη, μπορεί να αποθηκεύτει και εσωτερικά της Operation Unit στους καταχωρητές.

Πολλε΄ς φορές μεταξύ εξόδου της execution unit και καταχωρητών μεσολαβούν κυκλωματικά στοιχεία τα οποία παρέχουν ένα προαιρετικό επίπεδο μετασχηματισμο΄ψ των δεδομένων.Για παράδειγμα στα δεδομένα μπορει να πραγματοποιηθεί δεξία η αριστερή ολίσθηση (shift), κάτι που ισοδυναμεί με διαίρεση και πολλαπλασιασμό αντίστοιχα με το 2.

Παραπάνω περιγράψαμε πως επικοινωνεί η CPU με την μνήμη

Η μονάδα επεξεργασίας δεδομένων και η μονάδα ελέγχου και συνήθως κάποια βοηθητικά κυκλώματα εισόδου και εξόδου ομαδοποιούνται στην λεγόμενη CPU (Central Processing Unit). Στην σημερινή εποχή η CPU αποτελείται απο ένα τυπωμένο κύκλωμα που καλείται μικροεπεξεργαστής/microprocessor.

Η CPU με την σειρά της διαχωρίζεται στην *Control Unit* και *Execution/Processing Unit*. Η Execution Unit ή Μονάδα Επεξεργασίας περιλαμβάνει την alu (arithmetic logic unit) και τους καταχωρητές. Η Control unit ή μονάδα ελέγχου, είναι μια μονάδα συγρονισμού τον επιμέρους μερών του υπολογιστή. Περιέχει τον instruction register και τον program counter

1.3 **Control Unit**

Όπως είδαμε παραπάνω, η Execution Unit πραγματοποιεί τον μετασχηματισμό των δεδομένων. Χρειάζεται όμως επιπλέον μια μονάδα γνωστή ως *Control Unit,*που θα αναλαμβάνει να προσκομίσει την επόμενη εντολή προγράμματος απο την μνήμη, κατευθύνοντας την εγγραφή του MAR , να την αποκωδικοποιήσει και να διαχειριστεί τα δεδομένα και τέλος να τα αποθηκεύσει. Επίσης σε κάποιες αρχιτεκτονικές μεταφράζει κάθε εντολή σε έναν κατάλληλο αριθμό βημάτων, τις μίκροεντολές και αναλαμβάνει την δρομολόγηση τους με την σωστή σειρά προς την μονάδα επεξεργασίας.

Αναλαμβάνει τόσο τον συγχρονισμό των υποσυστήματων του επεξεργαστή όσο και τροφοδότηση των εισόδων τους με τα κατάλληλα σήματα. Επίσης, καθώς η ροή των προγραμμάτων περιλαμβάνει συχνά διακλαδώσεις, τα λεγόμενα Jump η Branch η Control Unit μπορεί να αλλάξει την αλληλουχία των εντολών βασιζόμενη στην αποθηκεύμενη πληροφορία.

Διαβάζει την επόμενη εντολή προς εκτέλεση, την αποκωδικοποιεί (interprets) τις εντολές του προγράμματος μετατρεποντας τα σε σήματα ελέγχου που ενεργοποιούν άλλα μέρη του υπολογιστή. Όλα τα μέρη του υπολογιστή συγχρονίζονται βάση της control unit. Το κύκλωμα αυτό ελέγχει την ροή των δεδομένων προς την μονάδα επεξεργασίας.

Ο τρόπος που διαχειρίζεται ένα μικροπρογραμματιζόμενο σύστημα τις ακολουθίες σημάτων ελέγχου είναι παρόμοιος με τον τρόπο που λειτουργεί εκτελώντας ένα κανονικό πρόγραμμα. Η διαφορά είναι οτι το αποθηκεύμενο μικροπρόγραμμα το οποίο αντιπροσωπεύει πλέον τις ακολουθίες των σημάτων ελέγχου (control signals)

Κεφάλαιο 2

Execution Unit

Μονάδα Εκτέλεσης είναι το κύκλωμα που προσκομίζει τα δεδομένα, εφαρμόζει τους αριθμητικούς και λογικούς τελεστές πάνω σε αυτά και αποθηκέυει το αποτέλεσμα στους διαθέσιμους καταχωρητές και την κύρια μνήμη του συστήματος. Οι εντολές που εκτελεί η μονάδα επεξεργασίας είναι γενικά απλές, πρόσθεσε ε΄ναν καταχωρητή με κάποιον άλλο η με μια σταθερά, μετακίνησε δεδομένα απο μια τοποθεσία της μνήμης σε μία άλλη ή και τέλος αποθήκευσε το αποτέλεσμα σε έναν καταχωρητή και επίσης σε μια θέση της κύριας μνήμης.

ALU

Η *alu* είναι ένα ψηφιακό κύκλωμα που πραγματοποιεί τον μετασχηματισμό των δεδομένων, εκτελώντας αριθμητικές και λογικές λειτουργίες. Είναι η θεμέλια υπομονάδα της **execution unit** . Οι είσοδοι της **alu** είναι τα δεδομένα πάνω στα οποία θα εφαρμοστεί η αριθμητική πράξη, καθώς και επιπλέον κώδικας απο την **control unit** η οποία υποδεικνύει ποια λειτουργία θα εφαρμοστεί. Η έξοδός της είναι το αποτέλεσμα της λειτουργίας.

Σε πολλές υλοποιήσεις η **alu** παράγει επιπλέον κώδικα από και πρός έναν καταχωρητη, τον **status register** (καταχωρητή κατάστασης). Αυ΄τος ο καταχωρητής χρησιμοποιείται για να υποδεικνύει τις περιπτώσεις **carry-in, carry-out, zero, overflow, sign** δηλαδή κρατουμένου εισόδου, κρατουμένου εξόδου, μηδενικού αποτελέσματος, υπερχείλισης/υπεκχείλισης και αρνητικού αριθμού.

Καταχωρητές

Εν γένει η προσπέλαση της κύριας μνήμης είναι χρονοβόρα διαδικασία και ειδικότερα η εγγραφή καθώς χρειάζεται η ενημέρωση και των δυο καταχωρητών που συχνά γίνεται σε δύο στάδια όταν υπάρχει μόνο μία δίαυλος προς την κύρια μνήμη. Για τον λόγο αυτόν υπάρχουν θέσεις μέσα στην **CPU** που μπορούν

να προσπελαστούν ταχύτατα. Αυτές οι θέσεις είναι οι καταχωρτήτες η αλλιώς Working registers.

Επιπλέον πολλές ALU είναι καλωδιωμένες με τέτοιον τρόπο ωστε να εκμεταλλέυονται κάποιους συγκεκριμένους καταχωρητες που βρίσκονται στην CPU. Ένας απο αυτούς είναι ο συσσωρευτής η accumulator ο οποίος δύναται να χρησιμοποιηθεί τόσο ώς προέλευσι όσο και ως ώς προορισμός του αποτελέσματος και συχνά περιέχονται εντολές γλώσσας μηχανής που εκμεταλλέυονται την ύπαρξη αυτου του καταχωρητη.

Επιπλέον του παραπάνω καταχωρητή υπάρχουν και κάποιοι άλλοι ειδικής σημασίας που αξίζει να αναφερθούν. Αυτοί είναι οι PC και ο X. Ο PC/Program Counter είναι δείκτης στην διεύθυνση της επόμενης εντολής προς εκτέλεση.

Ο καταχωρητής X αποτελεί βοηθητικό καταχωρητή ο οποίος χρησιμοποιείται κυρίως στον υπολογισμό διευθύνσεων με το να λειτουργεί ως offset και σαν μετρητής στην περίπτωση loop, χωρίς αυτό να απαγορέυει και τιν χρήση του ως απλού καταχωρητή.

Ένα βασικό στοιχείο κοινό σε όλες τις CPU είναι ο *program counter* (PC), ένας ειδικού σκοπού καταχωρητής ο οποίος διατηρεί την θέση μνήμης απο την οποία θα διαβαστεί η επόμενη εντολή, Η αλληλουχία των συμβάντων στην control unit κατά την διερμηνεία μιας εντολής, αποτελεί στην ουσία ένα είδος προγράμματος και σε μερικές cpu υπάρχει μια μονάδα ο microsequencer ο οποίος εκτελεί πρόγραμμα σε μικροκώδικα για να προκειμένου να συμβούν τα γεγονότα.

2.0.1 Γλώσσα Μηχανής

2.0.2 Διευθυνσιοδότηση

2.0.3 Μικροπρογραμματισμός

Μικροπρογραμματισμός είναι η διαδικασία συγγραφής μικροκώδικα για έναν μικροεπεξεργαστή.(BRITTANICA). Ο μικροκώδικας είναι χαμηλού επιπέδου κώδικας που καθορίζει την συμπεριφορά ενός μικροεπεξεραγαστή όταν εκτελεί εντολές γλώσσας μηχανής (machine language) . Τυπικά μια εντολή γλώσσας μηχανής μεταφράζεται σε κάποιες εντολές μικροκώδικα, τις μικροεντολές.

Ο Μικροκώδικας παρέχει ένα επίπεδο διαστρωμάτωσης (layer) καθώς παρεμβάλεται μεταξύ του hardware και των εντολών γλώσσας μηχανής. Βοηθά στην απόκρυψη των κυκλωματικών στοιχείων και έτσι ψπάρχει μεγαλύτερη ελευθερία στην σχεδίαση των εντολών. Η αποθήκευση του μικροκώδικα γίνεται σε αφιερωμένη μνήμη ROM ,την μικρομνήη. Για τον λόγο αυτό υπάρχει εξαρχής δομημένη οργάνωση, επομένως ευκολότερη διάγνωση σφαλμάτων.

Μικροεντολή Είναι μια ακολουθία **bit** που καθορίζουν την επακριβή συμπεριφορά των δομικών μονάδων του συστήματος δηλαδή τον τρόπο που λειτουργούν οι καταχωρητές, οι διάυλοι,η εξωτερική μνήμη, και η **ALU**. Επίσης, κάθε μικροεντολή καθορίζει την πηγή της επόμενής μικροεντολής, καθώς δεν είναι δεσμευτικό να εκτελεστούν με την σειρά που είναι αποθηκεύμενες στην μικρομνήμη.

Εκτενέστερη χρήση μικροκώδικα επιτρέπει σε μια υπάρχουσα αρχικεκτονική να προσομοιώση άλλες πολυπλοκότερες με μεγαλύτερο μήκος λέξης και πολυπλοκότερη μονάδα εκτέλεσης.

2.0.4 Δυνατότητες

2.0.5 Λειτουργικό τμήμα

Αλλαγή ροής

Υπορουτίνες

Παρόλο που με εντολές **loop** ο προγραμματιστής δύναται να εκτελέσει την ίδια ακολουθία κώδικα πολλε΄ς φορές, στην περίπτωση που σε ένα πρόγραμμα εμφανίζεται μεθοδικά η ίδια ακολουθία κώδικα, μπορούμε να τον ενσωματώσουμε σε υπορουτίνα/**subroutine**.Σε αυτήν μεταβαίνουμε με κατάλληλη εντολή **Jump-to-subroutine** καθώς και επιστρέφουμε στην κανονική εκτέλεση του προγράμματος με την **Return-from-subroutine**. Με την τελευταία εντολή επιστρέφουμε στην κανονική εκτέλεση του προγράμματος με το να εκτελεστεί η επόμενη εντολή της εντολής κλήσης της υπορουτίνας.

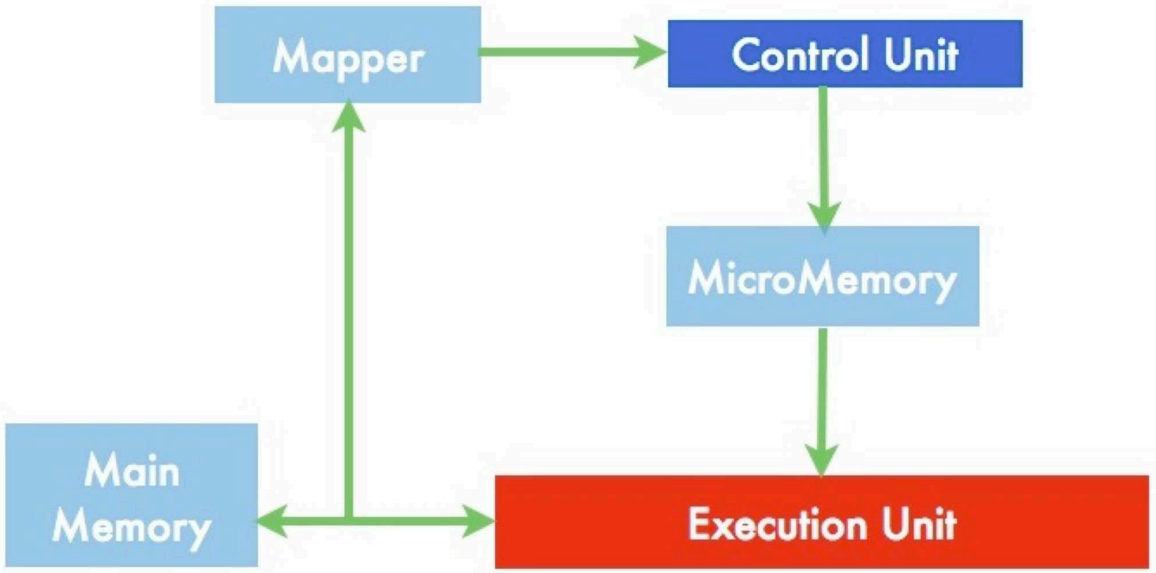
2.0.6 **VHDL**

Η VHDL είναι μια γλώσσα περιγραφής κυκλωμάτων, κυρίως ψηφιακών. Μοίαζει με μια γλώσσα προγραμματισμού αλλά χρησιμοποιείται για να περιγράψει ένα κύκλωμα και όχι πρόγραμμα που θα εκτελεστεί στον υπολογιστή.Περιγράφει πρωτίστος την δομή του σχεδίου του κυκλώματος, δηλαδή των τμημάτων το οποίο αυτό αποτελείται, καθώς και την διασύνδεση των επιμέρους τμημάτων αυτών στο.

Η νέα γλώσσα αυτή επιτάσσει μια νέα μεθοδολογία σχεδιάσης. Αρχικά δημιουργείται σχέδιο του συστήματος είται σχηματικά είται με μορφή κώδικα.Στην συνέχεια συντίθεται. Έπειτα γίνεται έλεγχος του σχεδίου και της λειτουργικότητας του είτε με γραφική αναπαράσταση με κυματομορφές (**waveform**) είτε με χρήση κώδικα που τροφοδοτεί το εκάστοτε κύκλωμα με εισόδους και επιβεβαιώνοντας την έξοδό του με την αναμενόμενη. Τέλος ένα κύκλωμα μπορεί να εξεταστεί με απλά πρακτικα τέστ για την ορθότητα του αν αυτό είναι εφικτό.

Κεφάλαιο 3

ΤΟ ΣΥΣΤΗΜΑ



Σχήμα 3.1: ΤΟ ΣΥΣΤΗΜΑ

3.1 Γενική Αρχιτεκτονική

Ολο το σύστημα αποτελεί μικροεπεξεργαστή των 8 bit.Οι κυρίαρχες δομικές μονάδες του συστήματός μας είναι δύο, η Μονάδα Ελέγχου/*Control Unit* και η Μονάδα Επεξεργασίας/*Execution Unit*. Τόσο η Μονάδα Ελεγχου όσο και η Μονάδα Εκτέλεσης έχουν δυνατότητα επικοινωνίας με την Κύρια Μνήμη/*Macromemory*. Επιπλέον της Macromemory, το σύστημα διαθέτει επιπλέον δύο μνήμες, την Μικρομνήμη (microMemory) και μνήμη αντιστοίχισης της μακροεντολής/mapper στην διεύθυνση της πρώτης εντολής που την διερμνημεύει. Όλες αυτές οι μνήμες χρησιμοποιούν δίαυλο διεύθυνσης των 8 bit, και η έξοδος τους είναι επίσης 8 bit με εξαίρεση την μικρομνήμη που εξάγει 40.

Μία ορθή χρονική αλληλουχία συμβάντων είναι η εξής. Αρχικά ο ακολουθητής προγράμματος (program counter) αρχικοποιείται με μια διεύθυνση της μακρομνήμης. Το περιεχόμενο αυτής της θέσης μνήμης αποτελεί και το opcode της δεδομένης εντολής. Αυτό δίνεται με την σειρά του ως όρισμα στον mapper απο τον οποίο εξάγεται σε επόμενο κύκλο ρολογιού οι πρωτή διεύθυνση της μικροεντολής που αντιστοιχεί στο δεδομενο opcode. Είναι ευθύνη πλέον της Control Unit να βρίσκει κάθε φορά ποια θα είναι η διεύθυνση της επόμενης μικροεντολής. Παράλληλα, κάθε νέα μικροεντολή ενεργοποιεί και την execution

unit η οποία αναλαμβάνει την προσκόμιση, επεξεργασία και αποθήκευση των δεδομένων με τον τρόπο που ορίζει η εκάστοτε μικροεντολή.

Να σημειωθεί οτι η περιγραφή των δύο οντοτήτων αυτών καθώς και ο διαμερισμός των κυκλωμάτων του συστήματος στις δύο αυτές μονάδες έγινε υπό την βλέψη του προγραμματισμού προκειμένει να γίνει ευκολότερη υλοποίηση. Αυτό φυσικά δεν αλλάζει την λειτουργικότητα του συστήματος ούτε τις γενικές αρχές που το διέπουν.

3.2 Μονάδα Ελέγχου (**Control Unit)**

Είναι η μονάδα που αναλαμβάνει επιλέξει την επόμενη μικροεντολή και να προσκομίσει τα δεδομένα της για επεξεργασία στην Execution Unit .

Περιλαμβάνει τα ακόλουθα συστατικά (components).

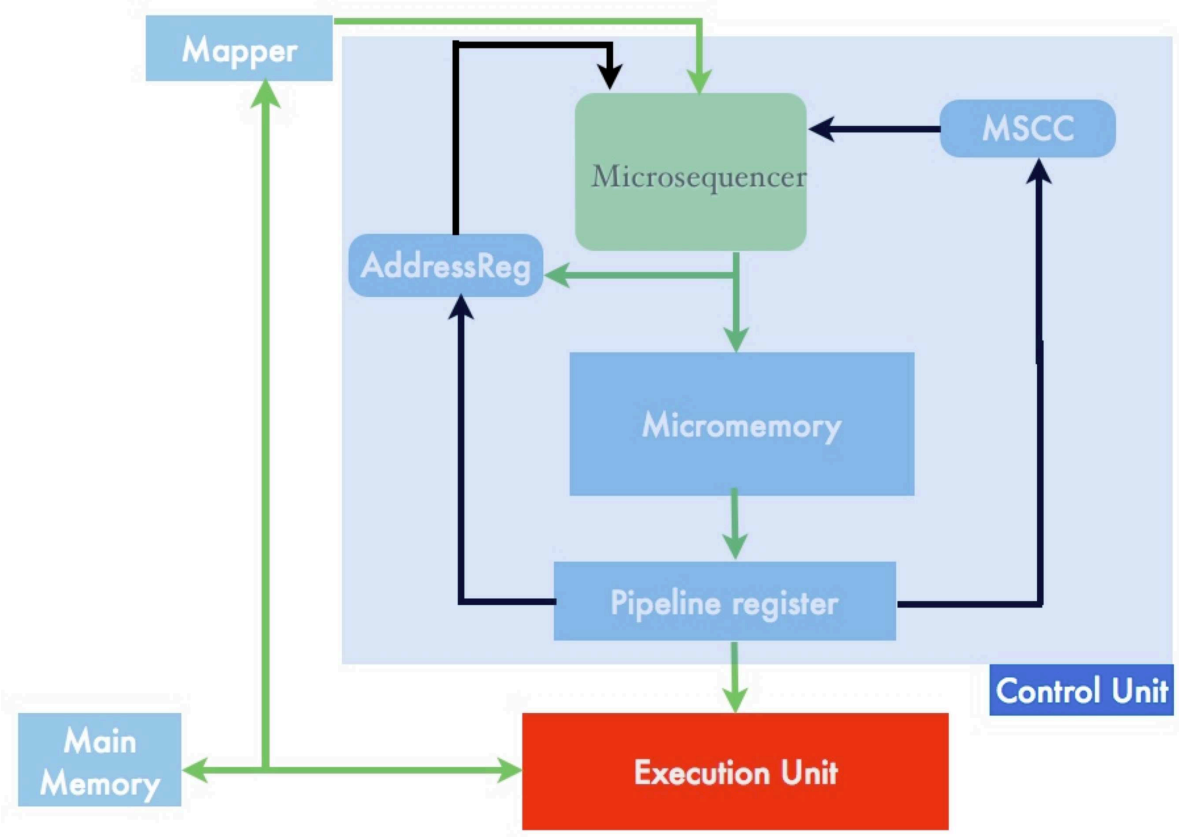
1. τον ακολουθητή μικροπρογράμματος (microsequencer) που εξαγει την διεύθυνση της επόμενης μικροεντολής προς εκτέλεση. Ο ακολουθητής είναι το κυρίαρχο συστατικό της μονάδας ελέγχου.

2. το κύκλωμα ' καταχωρητη' που περιέχει την τρέχουσα μικροεντολή που εκτελείται (pipeline register).

3. το κύκλωμα οδηγησης του microsequencer, MSCC το οποίο αναλαμβάνει να προμηθεύσει τα σήματα για να λειτουργήσει, οδηγούμενο το ίδιο απο την τρέχουσα μικροεντολή.

4. Τη μικρομνήμη/Micromemory στην οποία βρίσκεται ο μικροκώδικας. Η διεύθυνση της τρέχουσας μικροεντολής επιλέγεται μέσω του microsequencer.

5. Έναν αθροιστή που αθροίζει την διεύθυνση της τρέχουσας μικροεντολής με το offset το οποίο περιέχεται ως δεδομένο μέσα στην τρέχουσα μικροεντολή και δίνει μια νέα διεύθυνση άλματος στον microsequencer.



Σχήμα 3.2: Σχέδιο Control Unit

3.3 Microsequencer

Ο microsequencer ή ακολουθητής μικροπρογράματος είναι το δεσπόζων κύκλωμα της μονάδας ελέγχου. Αποτελείται από

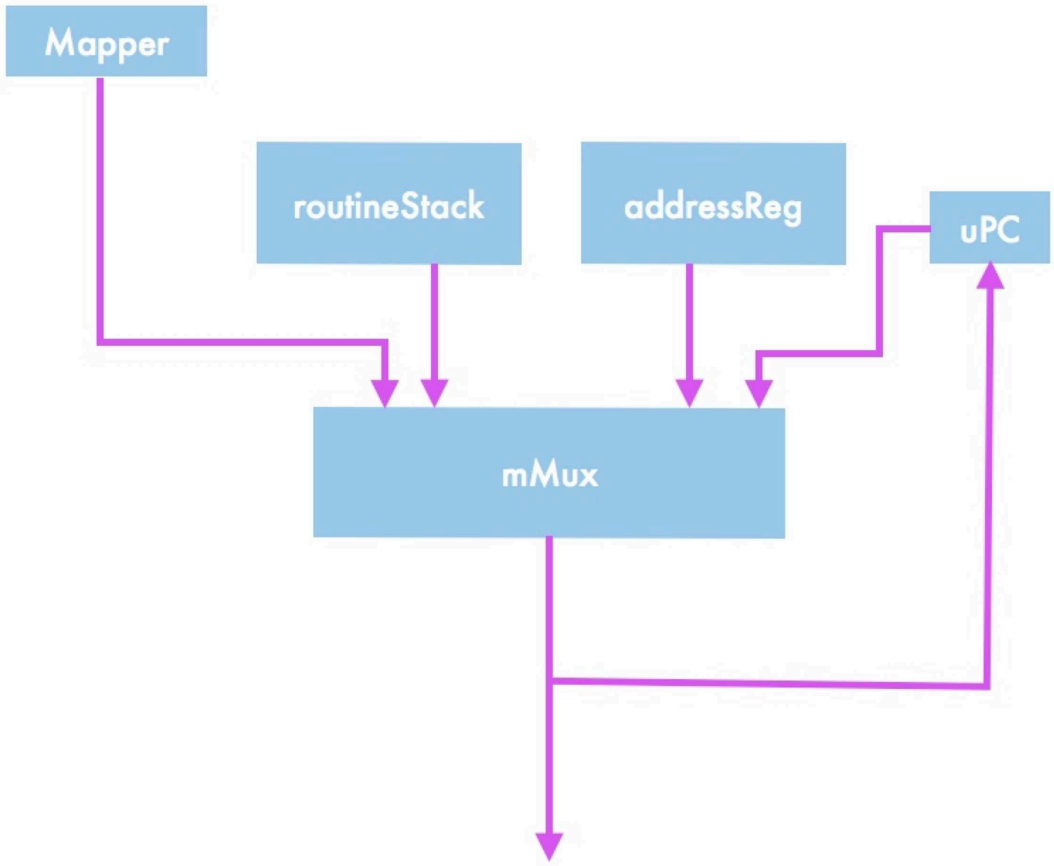
(α΄) τον πολυπλέκτη **mux** με τον οποίο επιλέγεται μία εκ των διευθύνσεων οι οποίοες προέρχονται απο :

- τον **mapper**
- καταχωρητή διεύθυνσης/**address register**,
- την στοίβα ρουτίνων
- τον **mPC**.

(β΄) Τον καταχωρητής διεύθυνσης **address register** με τον οποίο μπορούμε να κάνουμε άλματα σε διευθύνσεις σχετικές με την διεύθυνση της τρέχουσας μικροεντολής. Τα δεδομένα του καταχωρητή αυτού είναι αποτέλεσμα του αθροίσματος της τρέχουσας διεύθυνσης μικροεντολής με ένα **offset** το οποίο παρέχεται από την τρέχουσα μικροεντολή που εκτελείται.

(γ΄) Την στοίβα υπορουτίνων (**file/routine stack**), με την οποία παρέχεται στο σύστημα η διεύθυνση επιστροφής στην περίπτωση εκτέλεσης υπορουτίνας.

(δ΄) Τον **microprogam counter (mPC)** ο οποίος είναι ένας απλός καταχωρητής ο οποίος συνδέεται μέσω ενός αθροιστικού κυκλώματος με τον πολυπλέκτη επομένως διαθέτει κάθε φορά την επόμενη κατα σειρά μικροεντολή απο την τρέχουσα.



Σχήμα 3.3: Σχέδιο Microsequencer

3.4 Execution Unit

Ειναι η μονάδα που τροφοδοτείται απο την control Unit και αναλαμβάνει την επεξεργασία των δεδομένων.
[1]

Αρχικά, κάνει επιλογή των εντέλων, δηλαδή της προέλευσης των δεδομένων είται αυτά προέρχονται απο καταχωρητές είτε είναι εξωτερικά δεδομένα.

Στην συνέχεια, πάνω σε αυτά εφαρμόζεται ο κατάλληλος λογικός/αριθμητικός τελεστής.

Μετά το τέλος των υπολογισμών η Execution Unit θα επιλέξει αν το προιόν της θα αποθηκευτεί σε κάποιον απο τους εσωτερικούς καταχωρητές ή στην κυρία μνήμη(macromemory) ή αν ακόμα θα αποτελέσει διεύθυνση απο την οποία θα διαβαστούν δεδομένα στον επόμενο κύκλο ρολογιού, δηλαδη άν τεθει ο program counter.

Αποτελειται απο:

1. Την αριθμητική και λογική μονάδα (alu) η οποία έχει 2 εισόδους και δυνατότητα για 8 αριθμητικές και λογικές πράξεις με μορφή συμπληρώματος ώς προς 2.

2. Τον επιλογέα των έντελων (Source Selector) που τροφοδοτεί τις 2 εισόδους της alu με δοδομένα. Αυτά μπορει να προέρχονται είτε απο τους 16 καταχωρητές (Registers) είτε απο τον καταχωρητή Q είτε απο εξωτερικά δεδομένα. Τα εξωτερικά δεδομενα προέρχονται :

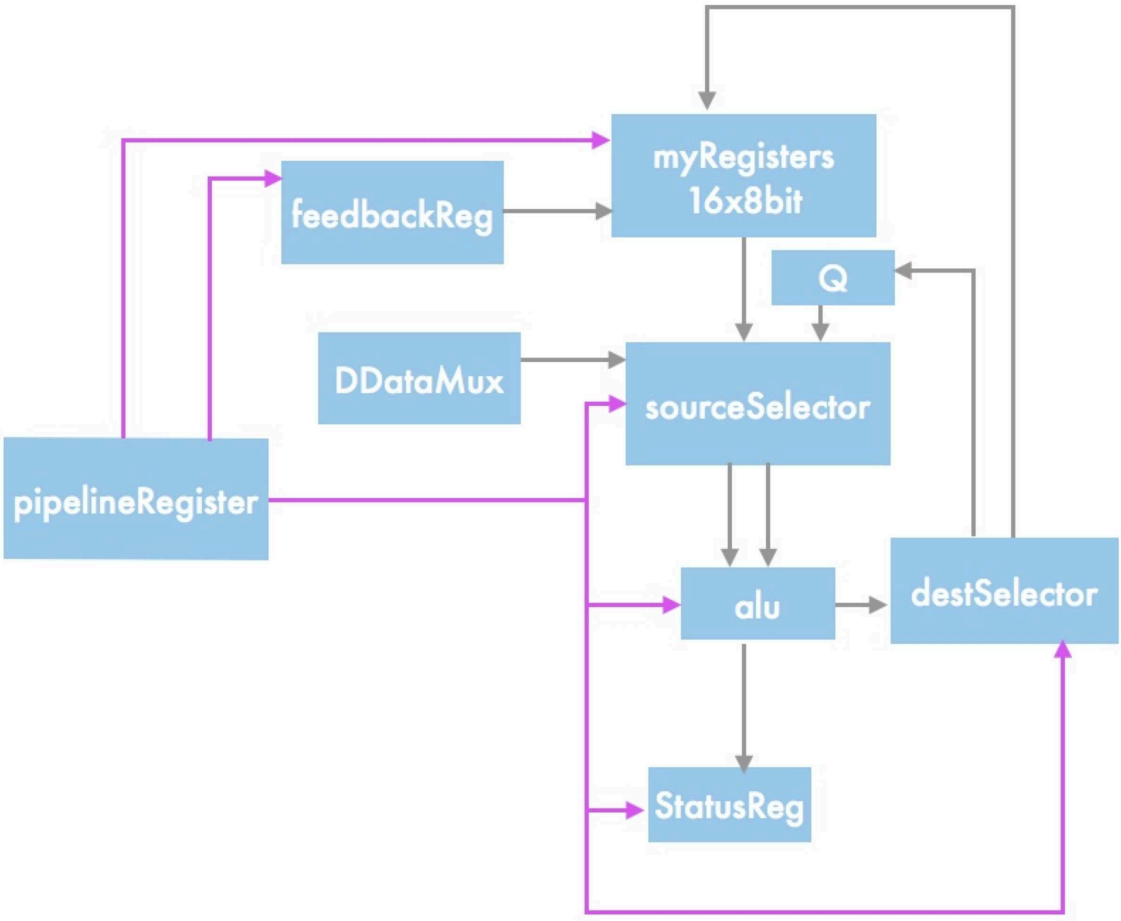
   - απο τα 2 bit τα οποία είναι διαθέσιμα απο την ίδια την μικροεντολή,
   - απο τα switches τα οποία χειρίζεται ο ίδιος ο χρήστης
   - την κύρια μνήμη
   - απλά το λογικό ′0′.

3. Τους 16 καταχωρητές Registers απο μία dual port μνήμη, που δέχεται 2 διευθύνσεις εισοδου a και b απο τις οποίες μπορούμε να διαβάσουμε ταυτόχρονα ή να γράψουμε δεδομένα στην διεύθυνση b.

4. Τον επιλογέα προορισμού (destSelector) με τον οποίο διαλέγουμε το που υα αποθηκευτεί το αποτέλεσμα, δηλαδή στους 16 καταχωρητές η και στον καταχωρητή Q . Επιπλέον μας δίνει δυνατότητα για shift.

5. Ένας επιπλεον καταχωρητης Q.

[1] Στο σχήμα οι μωβ συνδέσεις, εξωτερικές συνδέσεις.Η οδήγηση των κυκλωμάτων γίνεται απο τα bit της ίδιας της μικροεντολής. Γκρι συνδέσεις, εσωτερικές συνδέσεις μεταξύ των επιμέρους κυκλωμάτων της execution unit.



Σχήμα 3.4: Σχέδιο execution Unit

6. Το κύκλωμα (DDATAMUX) που διαχειρίζεται εξωτερικά/άμεσα δεδομένα (direct data)

7. Το κύκλωμα ανατροφοδότησης (feedback reg) του αποτελέσματος της alu ως διεύθυνση στην θύρα b . (Δέν φαίνεται η σύνδεση της εισόδου του στο σχήμα)

8. Τον καταχωρητή κατάστασης του συστήματος (status register) στον οποίο αποθηκέυονται τα *flags*

   - της υπερχείλισης/overflow
   - κρατουμενου/carry ,
   - προσήμου/sign
   - μηδενικου αποτελέσματος/zero.

3.5 ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΜΙΚΡΟΕΝΤΟΛΗΣ

Οι μικροεντολές που χρησιμοποιεί το σύστημα μας είναι εύρους 40 bit και είναι της μοφής

BRA	BIN	CON	I210	I543	I876	A	B	D	ControlBits

1. Το *BRA* είναι 5bit offset σε μορφή συμπληρώματος ως προς 2 με το οποίο το σύστημα μπορει να μεταβεί σε γειτονική διεύθυνση της τρέχουσας

2. Το *BIN* είναι ο τροπος με τον οποίο θα μεταβουμε στην επόμενη μικροεντολή. Με άλλα λόγια ποιά θα είναι η διευθυνση της επόμενης μικροεντολής. Αναλυτικότερα με

   - Cont , η διεύθυνση της επόμενης μικροεντολής είναι η επόμενη αριθμητικά διεύθυνση απο την τρέχουσα.
   - J2S ,πραγματοποιούμε άλμα στην μηδενική μικροεντολή (αρχικοποίηση του συστήματος) jump to start
   - UJMP , άλμα χωρίς συνθήκη/unconditional jump με διεύθυνση του address register.
   - CJMP , άλμα υπό συνθήκη /conditional jump .
   - UJSR , άλμα χωρίς συνθήκη σε υπορουτίνα.
   - CJSR , κλήση υπορουτίνας υπο συνθήκη.

- CRSR, ,επιστροφή απο υπορουτίνα υπο συνθήκη.

3. Με το *CON* επιλέγεται η ίδια η συνθήκη σύμφωνα με την οποία υα πραγματοποιηθεί άλμα υπο συνθήκη. Δηλαδή, η συνθήκη μπορεί να αφορά μία απο τις τιμές του κρατουμένου, μηδενικού αποτελέσματος, προσήμου η bit υπερχείλισης.

   Ακολουθεί μία ομάδα των 9 bit σε υποομάδες των 3 bit τα οποία καθορίζουν

   - *I210* , τα έντελα που θα τροφοδοτήσουν τις δυο θύρες της αριθμητικής λογικής μονάδας alu . Το αριστερό έντελο αντιστοιχεί στην θύρα R της alu και αντίστοιχα το δεξιο στην S.

     Αναλυτικότερα οι θύρες R και S οδηγούται απο τις εξής διαθέσιμες επιλογές

     - A/B που είναι οι καταχωρητές που επιλέγονται απο την dual port μνήμη της μονάδας επεξεργασίας
     - Q ο ανεξάρτητος καταχωρητης της μονάδας επεξεργασίας
     - D τα άμεσα δεδομένα τα οποία προέρχονται είτε απο την μικροεντολή (τμήμα D)
     - MDR απο την κύρια μνήμη
     - dip-switches απο το ίδο το board.
     - Z συμβολίζεται το λογικό ′0′.
   - *I543*, η πράξη που θα πραγματοποιηθεί μεταξύ των δεδομένων που προσκομίζονται μέσω των θυρών R και S της ALU.
   - *I876*, ο προορισμός του αποτελέσματος καθώς και αν θα πραγματοποιηθούν απλα σύνθετα shift
   - QREG Αποθήκευση στον καταχωρητή Q
   - NOP Δεν γίνεται αποθήκευση
   - RAMA Αποθήκευση στην RAM(macro) καθώς και έξοδος την A data port των καταχωρητών
   - RAMF Αποθήκευση στην RAM
   - RAMQD Αποθήκευση στην RAM και τον Q καθώς και shift down
   - RAMD Αποθήκευση στην RAM καθώς και shift down
   - RAMQU Αποθήκευση στην RAM και τον Q καθώς και shift down
   - RAMU Αποθήκευση στην RAM καθώς και shift up

bits	BIN	CON	I210	I543	I876
000	Con	C	AQ	R+S	QREG
001	J2S	O	AB	S-R	NOP
010	UJMP	N	ZQ	R-S	RAMA
011	CJMP	Z	ZB	RvS	RAMF
100	UJSR	mC	ZA	RŜ	RAMQD
101	CJSR	mO	DA	R Ŝ	RAMD
110	URSR	mN	DQ	RxorS	RAMQU
111	CRSR	mZ	DZ	(RxorS)	RAMU

4. *A* Η διεύθυνση ανάγνωσης της A port της μνήμης των καταχωρητών

5. *B* Η διεύθυνση ανάγνωσης/έγγραφής

6. *D* Δύο bit ως άμεσα δεδομένα

3.5.1 Control Bits

7. Τέλος έχουμε και μία ομάδα απο 10 bits τα οποία ελέγχουν 10 διαφορετικά σήματα, αυτα ειναι

   sh , το οποίο ενεργοποιεί κυκλικό shift με 1

   selb , Ενεργοποιεί ανατροφοδότηση των δεδομενων της execution unit της alu ως είσοδο στην b θύρα των 16 καταχωρητών με ΄1΄

   aluMacro ,Πραγματοποιεί γράθιμο στην κύρια μνήμη με δεδομένα το αποτέλεσμα της execution unit, δηλαδή θέτει τον MDR. Η διεύθυνση της κύριας μνήμης πρέπει να έχει αρχικοποιηθεί απο πριν.

   alumar , με ΄1΄ αρχικοποιεί την διευθυνση της κύριας μνήμης που θα διαβαστεί δηλαδή θέτει τον MAR (Memory address register).

   updateFlags , με ΄1΄ ενημερώνει τα flags που αντιστοιχούν στην τρέχουσα μαςροεντολή.

   mapperSeq , με ΄1΄ προκαλεί την διερμηνεία της πρώτης μικροεντολής που προέρχεται απο την μνήμη αντιστοίχισης δηλαδή του mapper.

   switchesAlu ,με ΄1΄ τροφοδοτεί την alu με άμεση είσοδο απο τα switches που βρίσκονται στο board.

   carryEnable , με ΄1΄ ενεργοποιεί την χρήση κρατουμένου στις αριθμητικές πράξεις.

mdrAlu , με ΄1΄ η alu τροφοδοτείται με άμεση είσοδο τον καταχωρητη MDR, πραγματοποιούμε δηλαδή διάβασμα απο την κύρια μνήμη.

ddataAlu , με ΄1΄ τροφοδοτεί την alu με άμεση είσοδο τα δύο βιτ της μικροεντολής.

3.6 ΠΑΡΑΔΕΙΓΜΑ

Η φιλοσοφία του μικροπρογραμματισμού διαφέρει αρκετά από αυτή του προγραμματισμού των εφαρμογών η και της assembly .

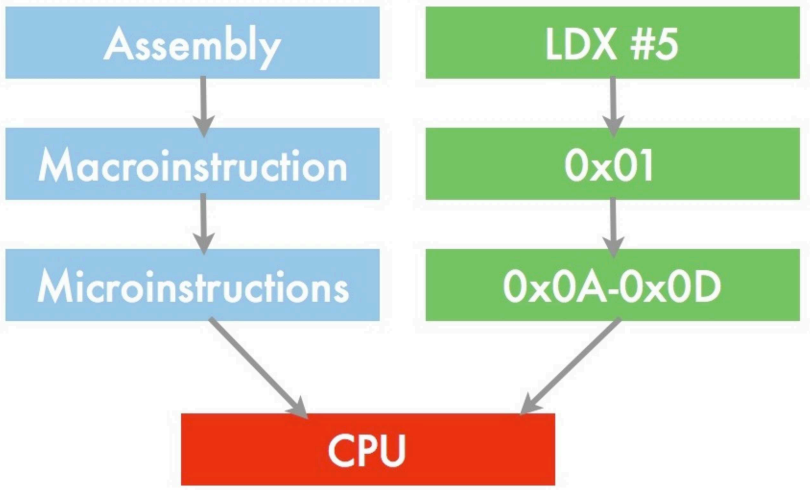
Σαν ένα παράδειγμα των παραπάνω ας δούμε τι γίνεται στην περίπτωση μιας συγκεκριμένης εντολής μηχανής. Ας υποθέσουμε οτι θέλουμε να εκτελέσουμε την εντολή LDX 5 με την οποία γίνεται φόρτωμα (LOAD) του καταχωρητή X της αριθμητικής λογικής μονάδας με την άμεση τιμή 5.

Η εντολή αυτή βρίσκεται αποθηκευμένη στην κύρια μνήμη του συστήματος με την μορφή opcode, ας υποθέσουμε με την τιμή "0x01" στο δεκαεξαδικό. Μολις Προσκομιστεί στο σύστημα συγκεκριμένη εντολή θα ακολουθηθεί η αποκωδικοποίησή της και η ¨διάσπασή' της σε μια ακολουθία μικροεντολών, έστω τεσσάρων με τις διευθύνσεις 0x0A-0x0D.

Το σύστημα πλέον είναι ικανό να τις εκτελέσει και έτσι εκτελείται η συγκεκριμένη εντολή. Κάθε μικροεντολή ορίζει ένα σύνολο ενεργειών οι οποίες εκτελούνται με παράλληλο τρόπο. Ως συνέχεια του παραδείγματος η πρώτη μικροεντολή της αλληλουχίας με διεύθυνση 0x0A αντιστοιχεί σε 40 bit "00000 000 000 101 000 011 0001 0001 01 01110 11110" τα οποία κάνουν τις ακόλουθες παράλληλες ενέργειες,

- ¨Συνέδεσε' τον καταχωρητή PC (program counter) στην θύρα R της ALU. [2]
- ¨Συνέδεσε' το λογικό ΄1΄ στην θύρα S της ALU .
- Εκτέλεσε την αριθμητική πράξη R+S.
- Τοποθέτησε το αποτέλεσμα της ALU στον δίαυλο διεύθυνσης της κύριας μνήμης (MAR) και στον μετρήτη προγράμματος (PC)
- Διάβασε την επόμενη μικροεντολή που βρίσκεται στην επόμενη διεύθυνση.

[2] Ο σκόπος του παραδείγματος στο δεδομένο σημείο είναι μία επιστκόπηση των λειτουργίων και όχι η επιμέρους τεχνικές λεπτομέρειες



Σχήμα 3.5: Παράδειγμα

Με άλλα λόγια δηλαδή αύξησε τον μετρητή προγράμματος κατά ένα ωστέ να δείξει την επόμενη κατα σειρα μικροεντολή και έπειτα προκάλεσε δίαβασμα από την κύρια μνήμη ωστέ να προσκομιστεί το **opcode** αυτής της εντολής.

3.7 ΕΠΙΣΚΟΠΗΣΗ

project Διασυνδέει τα επιμέρους υποσυστήματα

ControlUnit προσκόμιση της επόμενης προς εκτέλεση μικροεντολής

mseq Κάνει την επιλογή μεταξύ των διαθέσιμων διευθύνσεων μικροεντολών.

increg αποθηκεύει την είσοδό του αυξημένη κατα 1.

mMux πολυπλε΄κτης 4 σε μια.

pipelineReg διαμερίζει την τρέχουσα μικροεντολής στα επιμέρους τμήματα της.

mscc οδηγει τον microsequencerδεχομενο ως είσοδο τμήματα της τρέχουσας μικροεντολής.

addressReg Καταχωρητής στον οποίο αποθηκεύεται η διεύθυνση της τρέχουσας μικροεντολής προστιθέμενη με ένα δεδομένο offset με μορφί συμπληρώματος ως προς 2.

routineStack Κύκλωμα που υλοποιεί στοίβα.

Execution Unit Κύκλωμα που αναλαμβάνει την εκτέλεση της μικροεντολής

alu η μονάδα που εκτελεί τις αριθμητικές/λογικές πράξεις.

sourceSelector Αναλαμβάνει να προμηθεύσει την alu με τα έντελα.

myRegisters Dual port μνήμη

destSelecto επιλέγει τον προορισμό του αποτελέσματος

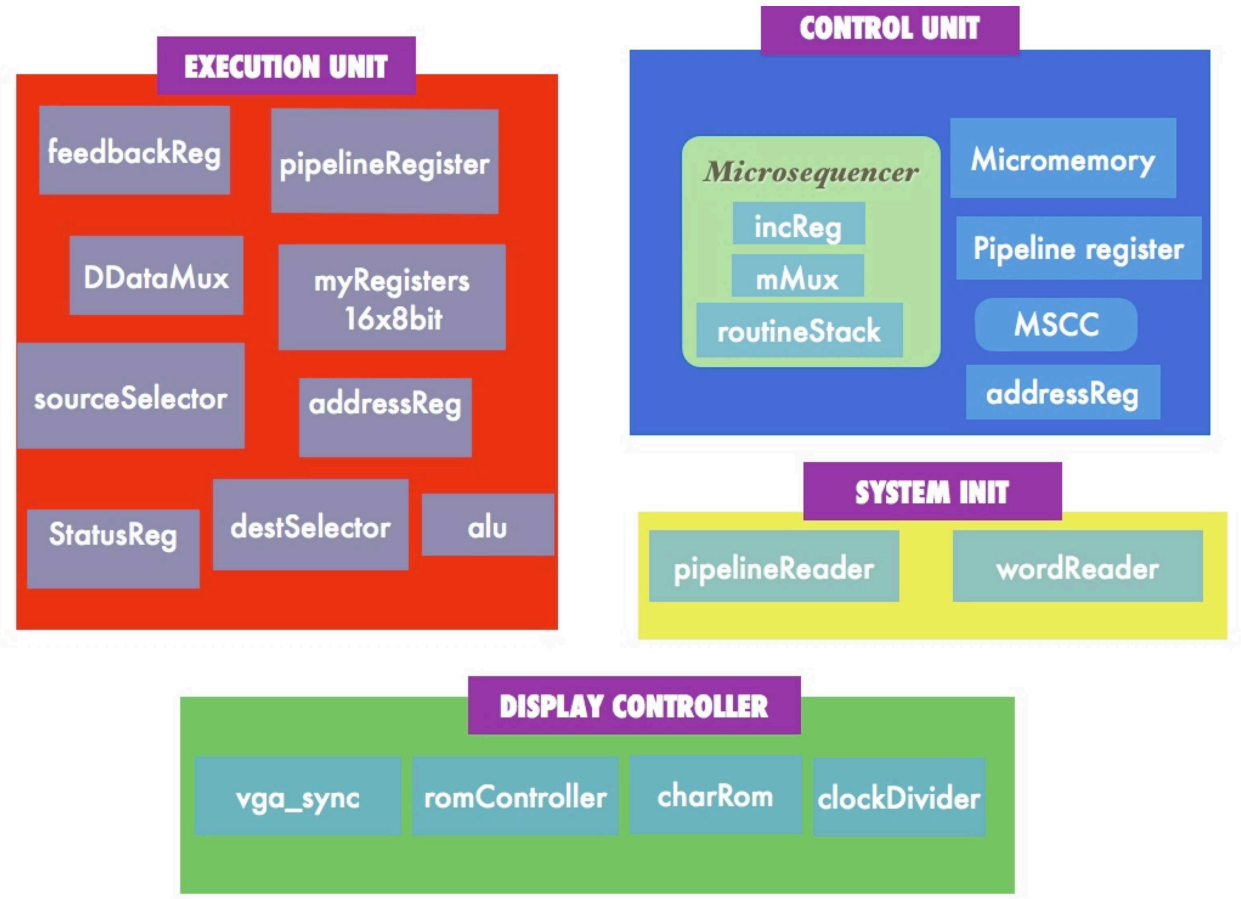
qReg Καταχωρητής

DDataMux επιλογεί απο τις διαθέσιμες άμεσες εισόδους προς τροφοδότηση της alu

feedbackReg ανατροφοδοτεί την είσοδο B με το αποτέλεσμα της alu.

statusRegister αποθηκεύει τα flags που θέτει η alu μετα την εκτέλεση μιας μικροεντολής.

SystemInit αναλαμβάνει την μεταβολή/αρχικοποίηση των μνημών του συστήματος



Σχήμα 3.6: ΙΕΡΑΡΧΙΑ

wordRead Διαβάζει 2 σέτ των 4 bit και επιστρέφει 8 bit .

pipelineReader Χρησιμοποιείται για την τμηματική εισαγωγή των 40 βιτ της μικροεντολής

displayController αναλαμβάνει την απεικόνιση των πληροφοριών του συστήματος στην οθόνη.

vga_sync Συγχρονίζει την οθόνη με το συστημά μας για απεικόνιση.

rom_Controller χρησιμοποιείται για να κάνει map τον font template στην οθόνη

charRom περιέχει font template τα οποια γίνονται μαπ στην οθόνη

clockDivider Στήν ουσία είναι ένα κύκλωμα που διαιρεί την συχνότητα του εισερχόμενου ρολογιου με 2.

vectorToSevengSeg Δέχεται ως είσοδο 4 bits και οδηγεί κατάλληλα ενα seven segment display.

clockSelector Έχει ως εισόδους δύο ρολόγια , με είσοδο επιλογής.

fiveToFive κάνει διαχωρισμό ενος bus 5 bit σε 5 εξοδους των 5 bit η καθεμία

sevenToSeven Διαχωρισμό bus σε 7 εξόδους

Κεφάλαιο 4

Σύνθεση **Control Unit**

4.1 Σύνθεση **Microsequencer**

4.1.1 **Inc Register**

Απλό κύκλωμα το οποίο δέχεται ως είσοδο τιν **inAddress** και την επιστρέφει αυξημένη κατά 1 ως **outAddress** στον επόμενο κύκλο ρολογιού.

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

entity incReg is
port(
inAddress : in std_logic_vector(7 downto 0);
outAddress : out std_logic_vector(7 downto 0);
```

Σχήμα 4.1: Inc Reg

```
clk : in std_logic);
end incReg;

architecture incReg of incReg is
begin
process(clk)
begin
if(clk'event and clk='1') then
outAddress <= inAddress + 1;
end if;
end process;
end incReg;
```

4.2 mMux

Πολυπλέκτης 4 εισόδων σε μία.

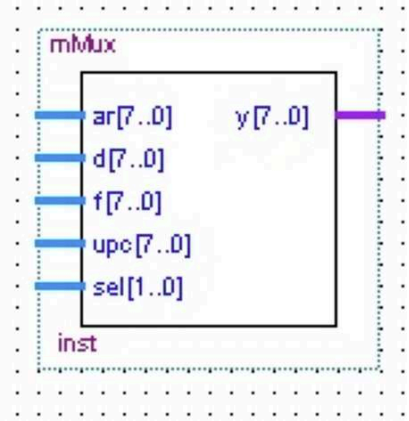
ar Address Register,για jump

d Direct Data, απο τον mapper.

f file/Stack στοίβα υπορουτίνων

upc microprogram counter, η επόμενη στη σειρά διεύθυνση.

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

entity mMux is
port(
ar, d, f, upc : in std_logic_vector(7 downto 0);
sel : in std_logic_vector(1 downto 0);
y: out std_logic_vector(7 downto 0));
end mMux;

architecture  mux1 of mMux is
begin
process(d,upc,ar,f,sel)
```



Σχήμα 4.2: mMux

```
begin
case sel is
when "00"=>y<=upc;
  when"01"=>y<=ar;
when "10"=>y<=f;
when "11"=>y<=d;

end case;
end process;
end mux1;
```

4.3 Routine Stack

Στοίβα που δίνει στο σύστημα την εκτέλεση υπορουτίνων.

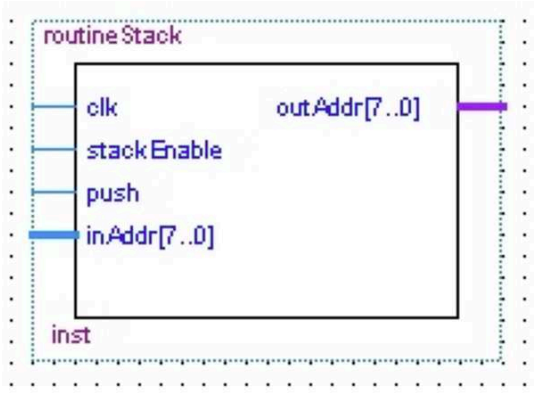
stackEnable Ενεργοποίηση της στοίβας με ΄1΄

push ώθηση με ΄1΄

inAddr εισερχόμενη διεύθυνση στην στοίβα

outAddr κορυφή της στοίβας.

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



Σχήμα 4.3: routineStack

```
entity routineStack is

generic (bits : integer:=8; -- # of bits per word
words : integer := 50); --# of words in stack
port(
--input ports
clk : in std_logic;
stackEnable :in std_logic; -- file enable
push :in std_logic;-- push with one
inAddr :in std_logic_vector(7 downto 0);--microcommand
--right through microprogram counter register

--output ports

outAddr :out std_logic_vector(7 downto 0)
-- top of the stack

);
end routineStack;

architecture routineStack of routineStack is
type vector_array is array(0 to words - 1) of
std_logic_vector(bits-1 downto 0);
signal stack: vector_array;
shared variable  top : natural range 0 to 50 :=0;
begin
```

```
process(clk)
--0 is considered the bottom of the stack
begin
if(clk'event and clk='0') then
if(stackEnable='1') then
if(push = '1') then
top:=top+1;
stack(top) <= inAddr;
else
top:=top-1;
end if;
end if;
end if;
end process;
outAddr <= stack(top);
end routineStack;
```

Η στοίβα υλοποιείται με πίνακα.Στην περίπτωση κλήσης υπορουτίνας γίνεται ώθηση, αυξάνεται ο δείκτης και ενημερώνεται η κορυφή της στοίβας. Στην επιστροφή της υπορουτίνας έχουμε απώθηση αλλά δεν μεταβάλλουμε τον καταχωρτητή της κορυφής της στοίβας για να υπάρχει η διεύθυνση επιστροφής στον επόμενο κύκλο ρολογιού.

4.4 mseq

Αφού πλέον έχουμε γνώση των οντοτήτων οι οποίες συνθέτουν τον microsequencer μπορούμε πλεον να περιγράψουμε και αυτον.

D σύνδεση με mapper

AR συνδεση με address Register

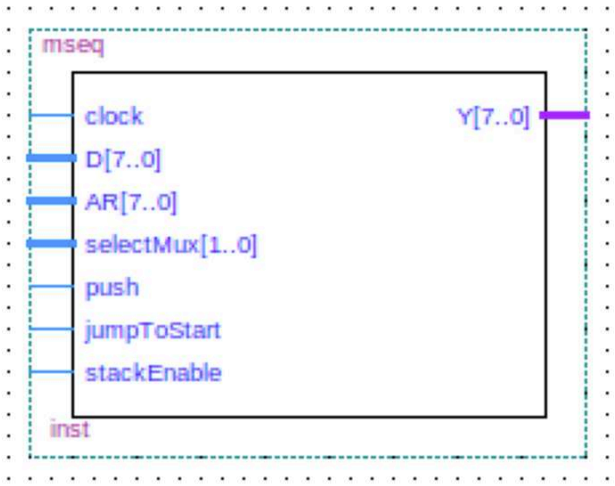
selectMux γραμμή επιλογής

push ενεργοποίηση της ώθησης

jumpToStart μετάβαση στην μηδενική διεύθυνση μικροεντολής.

stackEnable ενεργοποίηση της στοίβας υπορουτίνων

Y Η διεύθυνση της επόμενης μικροεντολής(έξοδος).



Σχήμα 4.4: mseq

```
library ieee;
use ieee.std_logic_1164.all;
entity mseq is
port
(
-- INPUT PORTS
clock : in std_logic;
--Inputs for MULTIPLEXER
D : in std_logic_vector(7 downto 0); --direct input
AR : in std_logic_vector(7 downto 0); --address register
-- signals driven by mscc
selectMux : in std_logic_vector(1 downto 0);
 -- select inputs
push : in std_logic;
jumpToStart : in std_logic;
stackEnable : in std_logic;
-- OUTPUT PORTS
Y : out  std_logic_vector(7 downto 0));
end mseq;

--------------------------------------------------------------------

architecture mseq of mseq is
component routineStack is -- the stack for routines
port(
```

```
--input ports
clk : in std_logic;
stackEnable :in std_logic; -- file enable
push :in std_logic;-- push with one
inAddr :in std_logic_vector(7 downto 0);--microcommand
--right through microprogram counter register
--output ports
outAddr :out std_logic_vector(7 downto 0)
-- top of the stack
);
end component;

component incReg is
port(
inAddress : in std_logic_vector(7 downto 0);
outAddress : out std_logic_vector(7 downto 0);
clk : in std_logic);
end component;

component mMux is
port(
ar, d, f, upc : in std_logic_vector(7 downto 0);
sel : in std_logic_vector(1 downto 0);
y: out std_logic_vector(7 downto 0)
);
end component;

signal X : std_logic_vector(7 downto 0); -- muxOutput;
signal F : std_logic_vector(7 downto 0); --file/stack
signal uPC :std_logic_vector(7 downto 0);
 --microProgramCounter


begin
c1: routineStack port map(stackEnable=>stackEnable,
outAddr=>F,
push=>push,inAddr=>uPC,clk=>clock);

c2: incReg port map(inAddress=>X,
 outAddress=>uPC, clk=>clock);
```

```
c3: mMux port map (ar=>AR,y=>X, upc=>uPC,
d=> D, f=>F, sel =>selectMux);

 process(X,jumpToStart)
begin
if(jumpToStart='0') then
Y<=X;
else
Y<=(others=>'0');
end if;
end process;
end mseq;
```

Η κύρια λειτουργία του **microsequencer** επιτελείται απο τον πολυπλέκτη **mMux**.Η πρώτη διεύθυνση μικροεντολής δίνεται εξωτερικά στον **mseq** από την άμεση είσοδο D.Αυτή περνάει κατευθείαν στην έξοδο του **mMux** και του κυκλώματος. Στην συνέχεια η νέα διεύθυνση της μικροεντολής και κάθε νεα διεύθυνση που εξάγει ο **mMux** περνάει απο το αυξητικό κύκλωμα **increg** και εισάγεται στον **uPC(microprogram counter)**. Απο εκεί είναι διαθέσιμη στον επόμενο κύκλο ρολογιού και με αυτό τον τρόπο υπάρχει διαθέσιμη η επόμενη σειριακά εντολή.

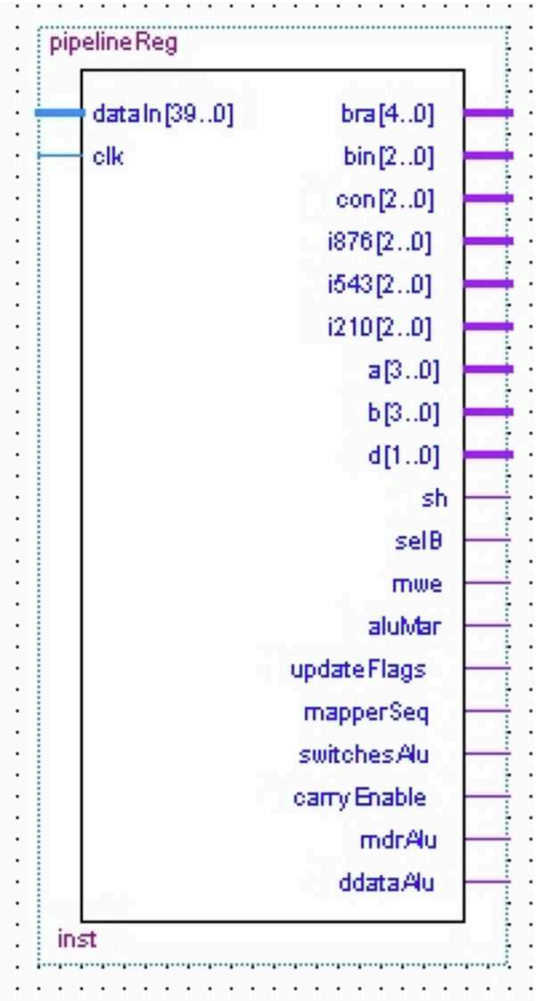
Το σύστημα υποστηρίζει άλματα και αυτό επιτυγχάνεται μέσω τής εισόδου **AR (Address Register)**. Ο AR όπως θά δούμε και κατα την μελέτη της **Control Unit** δέν είναι κάτι παραπάνω απο έναν ειδικό καταχωρητή στον οποίο αποθηκεύεται η διεύθυνση της μικροεντολής που μόλις εξήχθε απο τον **mMux** αθροισμένη με ένα συγκεκριμένο **offset** προκειμένου να επιτευχθούν σχετικά άλματα ως προς την εντολή αυτή. Χρησιμοποιώντας αυτή την διεύθυνση γίνεται άλμα.

Στην περίπτωση υπορουτίνων χρειάζεται όμως και μία διεύθυνση επιστροφής. Αυτή δίνεται απο την στοίβα **routineStack** . Έτσι, αφου ενημερωθεί το σύστημα οτι θα κληθεί υπορουτίνα στο επόμενο βήμα ωθείται(**push**) στην στοίβα υπορουτίνων ο **uPC** που περιέχει την επόμενη σειριακά μικροεντολή. Έπειτα επιλέγεται ο **Address Register** και μεταβαίνουμε στην διεύθυνση της υπορουτίνας. Μετά το πέρας της, γίνεται απώθηση(**pop**) και εξάγεται η διεύθυνση επιστροφής.

Τέλος υπάρχει και η είσοδος **jumpToStart** με την οποία οδηγούμε την έξοδο του **mseq** στο 0.

4.5 Pipeline Register

Κύκλωμα το οποίο διαμερίζει την μικροεντολή στα συστατικά της.



Σχήμα 4.5: pipelineReg

```
-- pipelineReg --
library ieee;
use ieee.std_logic_1164.all;

entity pipelineReg is
port (
--
dataIn : in std_logic_vector(39 downto 0);
clk : in std_logic;

-- the 40 bits broken tou subgroups--
bra : out std_logic_vector(4 downto 0);
bin : out std_logic_vector(2 downto 0);
con : out std_logic_vector(2 downto 0);

i876 : out std_logic_vector(2 downto 0);
i543 : out std_logic_vector(2 downto 0);
i210 : out std_logic_vector(2 downto 0);
a :out std_logic_vector(3 downto 0);
b : out std_logic_vector(3 downto 0);
d : out std_logic_vector(1 downto 0);
sh : out std_logic;
selB : out std_logic;
mwe : out std_logic;
aluMar : out std_logic;
updateFlags : out std_logic;
mapperSeq : out std_logic;
switchesAlu : out std_logic;
carryEnable : out std_logic;
mdrAlu : out std_logic;
ddataAlu : out std_logic

);
end pipelineReg;

architecture pipelineReg of pipelineReg is
begin
bra <= dataIn(39 downto 35);
 --branch offset
bin <= dataIn(34 downto 32);
```

```
 -- how to obtain next address
con <= dataIn(31 downto 29);
 -- how's the condition selected
a<= dataIn(19 downto 16);
-- A register address
b<= dataIn(15 downto 12);
-- B register address
i210 <= dataIn(28 downto 26);-- operands
i543 <= dataIn(25 downto 23);
-- arithmetic/logic operation
i876 <= dataIn(22 downto 20);-- destination of result
d<= dataIn(11 downto 10);-- 2 bit direct data
sh <= dataIn(9);-- circular or linear shift
selb<= dataIn(8);
-- feedback of alu output to register address
mwe <= dataIn(7);
-- enable of writing to macromemory
aluMar <=dataIn(6); -- place output of alu to MAR
updateFlags<= dataIn(5); --update alu flags
mapperSeq<= dataIn(4);
 -- enable mapper to read next macroaddress
switchesAlu <= dataIn(3);
 --direct data obtained for circuit switches
carryEnable <= dataIn(2); -- carry Enable
mdrAlu <= dataIn(1);
 -- place contents of MDR to ALU direct Input
ddataAlu <= dataIn(0);
-- enable the two bit direct data 'd'

end pipelineReg;
```

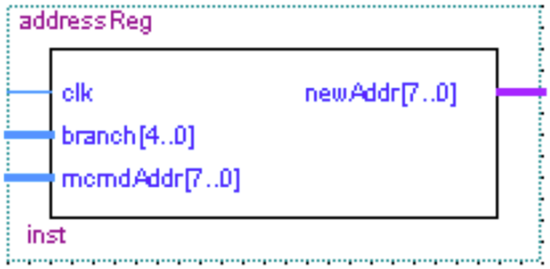
4.6 Address Register

branch το bra το οποίο αποτελεί offset

mcmdAddr Η διεύθυνση της μικροεντολής από τον mseq

newAddr Η προκύπτουσα διεύθυνση εξόδου

```
library ieee;
```



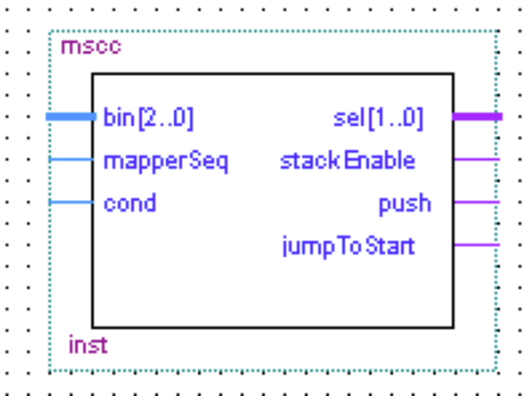
Σχήμα 4.6: pipelineReg

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

entity addressReg is
port (
clk : in std_logic;
branch : in std_logic_vector(4 downto 0);
mcmdAddr : in std_logic_vector(7 downto 0);
newAddr : out std_logic_vector(7 downto 0));
end addressReg;

architecture addressReg of addressReg is
begin
process(clk)
begin
if(clk'event and clk='1') then
newAddr<= mcmdAddr + ("000" & branch);
end if;
end process;
end addressReg;
```

Απλό κύκλωμα το οποίο εξάγει το αποτέλεσμα του αθροίσματος διεύθυνσης μικροεντολής και offset.

4.7 Microsequencer Control Circuit

Το κύκλωμα αυτό αναλαμβάνει να οδηγήσει τον microsequencer βάση των bin,mapperSeq τα οποία προμηθεύεται απο την τρέχουσα μικροεντολή και του cond το οποίο είναι το bit της τρέχουσας συνθήκης που έχει επιλεγεί.



Σχήμα 4.7: mscc

mnem	Bin(2:0)	Cond	mapperSeq	S1	S0	stackEnable	push	jumpToStart
cont	000b	x	0	0	0	0	x	1
J2S	001b	x	0	x	x	0	x	0
UJMP	010b	x	0	0	1	0	x	1
CJMP	011b	0	0	0	0	0	x	1
CJMP	011b	1	0	0	1	1	x	1
UJSR	100b	x	0	0	1	1	1	1
CJSR	101b	0	0	0	0	0	x	1
CJSR	101b	1	0	0	1	1	1	1
URSR	110b	x	1	1	0	1	0	1
CRSR	111b	1	0	1	0	1	0	1
CRSR	111b	0	0	0	0	0	x	1
xxx	xxx	x	1	1	1	0	x	1

CONT Continue, επιλέγεται η επόμενη σειρακά διεύθυνση

J2S Jump To Start, επιλέγεται η μηδενική διεύθυνση.

UJMP Unconditional Jump, επιλέγεται ο AddressReg που περιέχει την διεύθυνση άλματος

CJMP Conditional Jump, εξετάζεται η συνθήκη *COND*, (βλέπε μικροεντολή σελίδα 17).Εαν είναι αληθής η διεύθυνση επιλέγεται απο τον AddressReg ειδάλλως απο τον uPC που αποτελεί την επόμενη στη σειρά διεύθυνση.

CJSR Conditional Jump Subroutine, ομοίως με CJMP όμως επιπλέον στην περίπτωση που η συνθήκη είναι αληθής, το περιεχόμενο του uPC , δηλαδή

η επομένη διεύθυνση μικροεντολής ωθείται στην στοίβα έτσι ώστε να χρησιμοποιηθεί ως διεύθυνση επιστροφής.

CRSR Conditional Return Subroutine, σε περιπτωση που η συνθήκη είναι αληθής η διεύθυνση επιστροφής είναι αυτή που επιστρέφει η στοίβα, διαφορετικά η επόμενη σειριακά διεύθυνση.

```
---mscc microsequencer control circuit
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mscc is
port (
bin : in std_logic_vector(2 downto 0);
mapperSeq: in std_logic;--mapper seq
cond : in std_logic;  -- bit of selected condition

-- output ports
sel : out std_logic_vector(1 downto 0); --input of mux
-- uPC:0, AR:1, F:2, D:3
stackEnable : out std_logic;
push : out std_logic;
jumpToStart : out std_logic);
end mscc;

-- bin cond loadSeq load_dir_seq,s,fileEnable,push,jumpTostart
architecture mscc of mscc is



constant TRUE : std_logic :='1';
constant FALSE : std_logic:='0';
constant MPC : std_logic_vector:="00";
constant AR : std_logic_vector:="01";
constant STK : std_logic_vector:="10";
constant DDATA : std_logic_vector:="11";


--BIN
```

```
constant CONT : std_logic_vector(2 downto 0):="000";
constant J2S : std_logic_vector(2 downto 0):="001";
constant UJMP : std_logic_vector(2 downto 0) :="010";
constant CJMP : std_logic_vector(2 downto 0):="011";
constant UJSR : std_logic_vector(2 downto 0):="100";
constant CJSR : std_logic_vector(2 downto 0):="101";
constant URSR : std_logic_vector(2 downto 0):="110";
constant CRSR : std_logic_vector(2 downto 0):="111";


begin

process(bin,mapperSeq,cond)
begin
if(mapperSeq=TRUE) then
sel<=DDATA;
stackEnable<=FALSE;
push<='0';
jumpToStart<=FALSE;
else
case bin is
when CONT => sel<=MPC;
 stackEnable<=FALSE;
 push<='0';
 jumpToStart<=FALSE;

when J2S => sel<=(others=>'0');
jumpToStart<=TRUE;
stackEnable<=FALSE;
 push<='0';

when UJMP => sel<=AR;
 stackEnable<=FALSE;
 push<='0';
 jumpToStart<=FALSE;

when CJMP =>if(cond=TRUE) then
sel<=AR;
stackEnable<=FALSE;
 push<='0';
 jumpToStart<=FALSE;
```

```
else
sel<=MPC;
stackEnable<=FALSE;
 push<='0';
 jumpToStart<=FALSE;
end if;

when UJSR => sel<= AR;
 stackEnable<=TRUE;
 push<=TRUE;
 jumpToStart<=FALSE;

when CJSR =>if(cond=TRUE) then
sel<=AR;
stackEnable<=TRUE;
push<=TRUE;
jumpToStart<=FALSE;
else
sel<=MPC;
stackEnable<=FALSE;
push<='0';
jumpToStart<=FALSE;
end if;

when URSR => stackEnable<=TRUE;
push<=FALSE;
sel<=STK;
jumpToStart<=FALSE;

when CRSR => if(cond=TRUE) then
stackEnable<=TRUE;
push<=FALSE;
sel<=STK;
jumpToStart<=FALSE;
else
sel<=MPC;
stackEnable<=FALSE;
push<='0';
jumpToStart<=FALSE;
end if;
```

```
end case;
end if;
end process;

end mscc;
```

4.8 Control Unit

Το κύκλωμα το οποίο αναλαμβάνει τον προσδιορισμό της επόμενης προς εκτέλεσης μικροεντολής καθώς και την οδήγηση των υποκυκλωμάτων της Execution Unit

conditionBit Χρήση για conditional jump

firstMcmdAddr Η διεύθυνση της πρώτης μικροεντολής κατευθέιαν απο τον mapper

microQ Τα bit της μικροεντολής που διαβάζονται απο την μικρομνήμη

d 2 bit άμεσων δεδομένων.

A επιλογή της διεύθυνσης ανάγνωσης της A port των καταχωρητών.

B επιλογή της διεύθυνσης ανάγνωσης/εγγραφής της B port.

sh κυκλική περιστροφή με ′1′.

selB feedback με ′1′.

switchesAlu Επιτρέπει τα switches του κυκλώματος να χρησιμοποιηθούν ώς άμεσο όρισμα προς επεξεργασία απο την alu.

carryEnable επιτρέπει την χρήση κρατουμένου στις αριθμητικές πράξεις.

mapperSeq προκαλεί την διερμηνεία της επόμενης μακροεντολής.

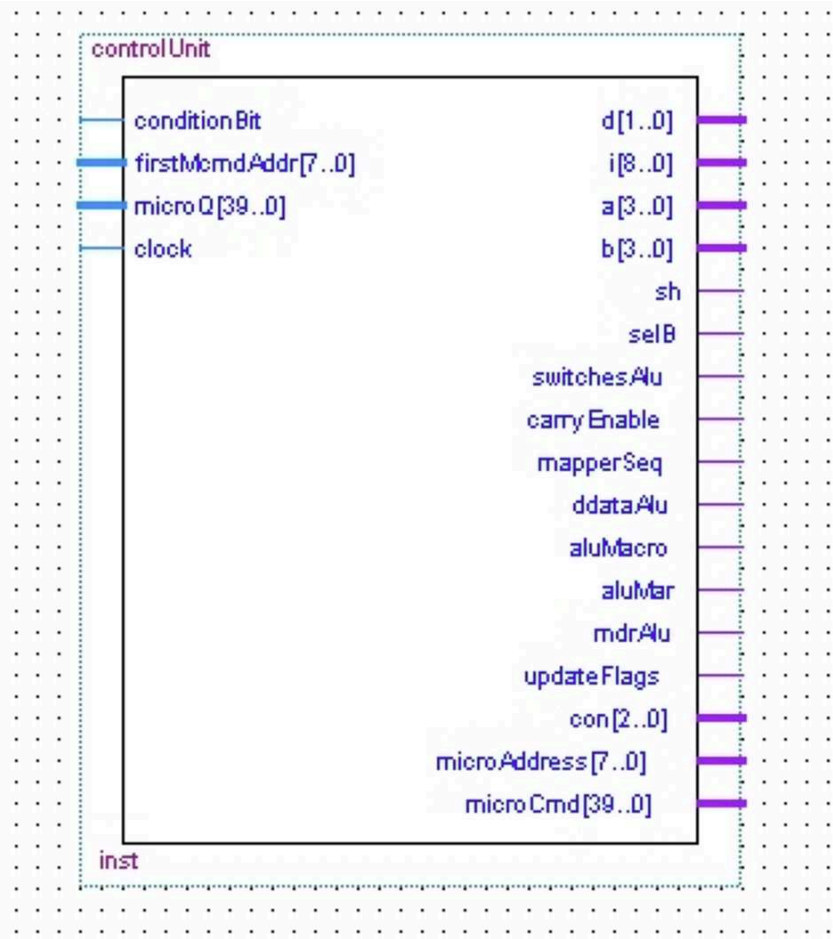
ddataAlu επιτρέπει την χρήση των 2 bit ως άμεσα δεδομένα προς επεξεργασία.

aluMacro επιτρέπει την εγγραφή στης κύρια μνήμη.

aluMar θέτει τον MAR(Memory Address Register) .

mdrAlu επιτρέπει το διάβασμα του MDR.

updateFlags θέτει τα flags της μακροεντολής.



Σχήμα 4.8: control Unit

con καθορίζει την συνθήκη υπεύθυνη για τα άλματα.

microAddress η διεύθυνση της τρέχουσας μικροεντολής.

microCmd τα 40 bit της μικροεντολής.

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

--diaferei apo tin control unit
--tis endiamesis epeidi den perilamvanei
--tin mnimi sto swma tis

entity controlUnit is
port
(
-- Input ports
conditionBit : in std_logic; --used for conditional jumps
firstMcmdAddr : in  std_logic_vector(7 downto 0);
--first microCommand Address
--retrieved from mapper

microQ : in std_logic_vector(39 downto 0);--microcmd data
-- signals to the execution unit
d : out std_logic_vector( 1 downto 0);--direct data
clock : in std_logic;

i : out std_logic_vector( 8 downto 0);
--operands/operation/destination

a : out std_logic_vector( 3 downto 0);
--register addr A

b : out std_logic_vector( 3 downto 0);
--register addr B

sh : out std_logic;--circular shift
selB : out std_logic; --feedback
switchesAlu : out std_logic;
--enable input from switches
```

```
carryEnable : out std_logic;
mapperSeq : out std_logic;--interpret new macro command
ddataAlu : out std_logic; --enable the two bit direct data
aluMacro : out std_logic; -- write enable
aluMar : out std_logic; --place output of alu to mar
mdrAlu : out std_logic;
--retrieve data from memory(MDR)

updateFlags : out std_logic;
 --update macroflags

con : out std_logic_vector(2 downto 0);
--condition for cond. jumps

microAddress : out std_logic_vector(7 downto 0);
--for display purposes

microCmd : out std_logic_vector( 39 downto 0)
);
end controlUnit;

architecture controlUnit of controlUnit is
component mseq is
port
(
-- INPUT PORTS

clock : in std_logic;
--Inputs for MULTIPLEXER
D : in std_logic_vector(7 downto 0); --direct input
AR : in std_logic_vector(7 downto 0); --address register

-- signals driven by mscc
selectMux : in std_logic_vector(1 downto 0); -- select inputs
push : in std_logic;
jumpToStart : in std_logic;
stackEnable : in std_logic;

-- OUTPUT PORTS
Y : out std_logic_vector(7 downto 0));
end component;
```

```
component pipelineReg is
port (
--
dataIn : in std_logic_vector(39 downto 0);
clk : in std_logic;

-- the 40 bits broken tou subgroups--
bra : out std_logic_vector(4 downto 0);
bin : out std_logic_vector(2 downto 0);
con : out std_logic_vector(2 downto 0);

i876 : out std_logic_vector(2 downto 0);
i543 : out std_logic_vector(2 downto 0);
i210 : out std_logic_vector(2 downto 0);
a :out std_logic_vector(3 downto 0);
b : out std_logic_vector(3 downto 0);
d : out std_logic_vector(1 downto 0);
sh : out std_logic;
selB : out std_logic;
mwe : out std_logic;
aluMar : out std_logic;
updateFlags : out std_logic;
mapperSeq : out std_logic;
switchesAlu : out std_logic;
carryEnable : out std_logic;
mdrAlu : out std_logic;
ddataAlu : out std_logic

);
end component;

component mscc is
port (
bin : in std_logic_vector(2 downto 0);
mapperSeq: in std_logic;--mapper seq
cond : in std_logic;  -- bit of selected condition

-- output ports
sel : out std_logic_vector(1 downto 0); --input of mux
-- uPC:0, AR:1, F:2, D:3
```

```
stackEnable : out std_logic;
push : out std_logic;
jumpToStart : out std_logic);
end component;

component addressReg is
 port (clk : in std_logic;
branch : in std_logic_vector(4 downto 0);
mcmdAddr : in std_logic_vector(7 downto 0);
newAddr : out std_logic_vector(7 downto 0));

end component;




component routineStack is

generic (bits : integer:=8; -- # of bits per word
words : integer := 4); --# of words in stack
port(
--input ports
clk : in std_logic;
stackEnable :in std_logic; -- file enable
push :in std_logic;-- push with one
inAddr :in std_logic_vector(7 downto 0);--microcommand
--right through microprogram counter register

--output ports
outAddr :out std_logic_vector(7 downto 0)-- top of the stack

);
end component;

--microsequencer
signal MSjumpToStart : std_logic;
signal MSAddrReg : std_logic_vector( 7 downto 0);
signal MSSelect : std_logic_vector( 1 downto 0);
signal MSpush : std_logic;
signal MSStackEnable : std_logic;
```

```
--mscc
signal MSCCBin : std_logic_vector(2 downto 0);
signal MSCCMapperSeq : std_logic;

--addressReg
signal ARBranch : std_logic_vector( 4 downto 0);
signal mcmdAddr: std_logic_vector( 7 downto 0);

--microMemory also input of addressReg
signal microAddr : std_logic_vector( 7 downto 0);

--pipelineRegister
signal pipelineDataIn : std_logic_vector(39 downto 0);
begin
c1: mscc port map(sel=>MSSelect,
stackEnable=>MSStackEnable, push=>MSPush,
jumpToStart=> MSjumpToStart,
bin=>MSCCBin, mapperSeq=>MSCCMapperSeq,
 cond=>conditionBit);

c2: mseq port map(D=> firstMcmdAddr,
 AR=> MSAddrReg,push=>MSpush,clock=>clock,
 jumpToStart=>MSjumpToStart,selectMux=> MSSelect,
 stackEnable=>MSStackEnable, Y=> microAddr);

c3 : pipelineReg port map(dataIn=> pipelineDataIn, clk=>clock,
bra => ARBranch, bin=> MSCCBin,
con=>con,i876=>i(8 downto 6), i543=>i(5 downto 3),
i210=> i(2 downto 0), a=>a, b=>b, d=>d,
sh=>sh, selB=>selB, mwe=>aluMacro, aluMar=>aluMar,
updateFlags=>updateFlags,
mapperSeq=>MSCCmapperSeq,switchesAlu=>switchesAlu,
carryEnable=>carryEnable,mdrAlu=>mdrAlu,
 ddataAlu=>ddataAlu);

c4 : addressReg port map(clk => clock, branch =>ARBranch,
mcmdAddr => microAddr,
newAddr =>MSAddrReg);
```

```
microAddress<=microAddr;

mapperSeq<=MSCCMapperSeq;
microCmd<=pipelineDataIn;
pipelineDataIn<=microQ;
microCmd<=microQ;
end controlUnit;
```

Κεφάλαιο 5

Συνθέτωντας την **Execution Unit**

5.1 Source Selector

Είναι το κύκλωμα υπέυθυνο για την τροφοδότηση των δύο port R, S της alu.

R αριστερό έντελο της alu

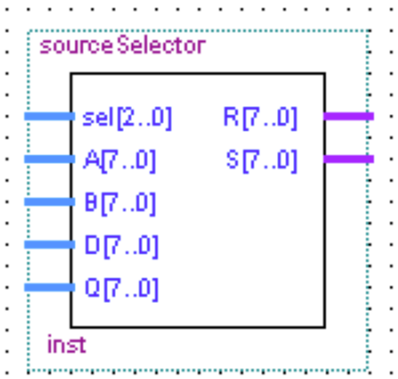
S δεξιό έντελο της alu

sel Βλέπε στήλη I210 του πίνακα σελίδα 18

A περιεχόμενο του καταχωρητη με την διεύθυνση A

B περιεχόμενο του καταχωρητη με την διεύθυνση B

D 2 bit απο την μικροεντολή



Σχήμα 5.1: sourceSelector

Q περιεχόμενο του καταχωρητή Q.

```
--source selector --
library ieee;
use ieee.std_logic_1164.all;

entity sourceSelector is
port (
 -- out
R : out std_logic_vector(7 downto 0);
S : out std_logic_vector(7 downto 0);
 -- in
sel : in std_logic_vector(2 downto 0);
A : in std_logic_vector(7 downto 0);
B : in std_logic_vector(7 downto 0);
D : in std_logic_vector(7 downto 0);
Q : in std_logic_vector(7 downto 0)
);

end sourceSelector;

architecture  sourceSelector of sourceSelector is
signal storeA,storeB,storeQ : std_logic_vector(7 downto 0);
begin
process(sel,storeA,storeB,D,storeQ)
begin
case sel is

 when "000"=>R <= storeA;S<=storeQ;
when "001"=>R<=storeA;S<=storeB;
when"010"=>R<="00000000";S<=storeQ;
when"011"=>R<="00000000";S<=storeB;

when"100"=>R<="00000000";S<=storeA;
when "101"=>R<=D ;S<=storeA;
 when "110"=>R<=D;S<=storeQ;
when "111"=>R<=D ;S<="00000000";
end case;
end process;
```

```
process(A,B,Q)
begin
storeA<=A;
storeQ<=Q;
storeB<=B;
end process;
end sourceSelector;
```

Το κύκλωμα θέτει της εισόδους της alu βάση του προαναφερθέντος πίνακα.

5.2 myRegisters

Οι 16 καταχωρητές του συστήματος

addr_a διεύθυνση για την θύρα a

addr_b διεύθυνση για την θύρα b

data_b δεδομένα προς εγγραφή για την θύρα b

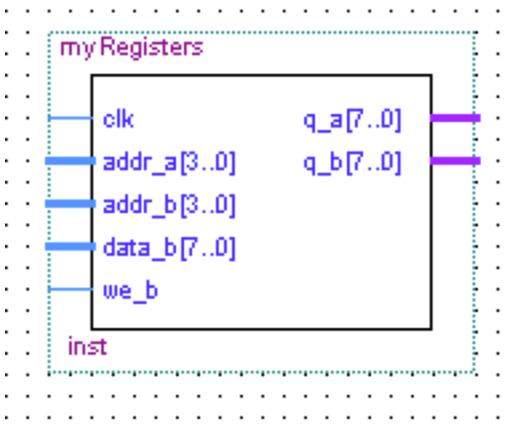
we_b επίτρεψη εγγραφής για την θύρα b

q_a το περιεχόμενο της διεύθυνσης addr_a

q_b το περιεχόμενο της διεύθυνσης addr_b

reg0-15 Το περιεχόμενο των 16 καταχωρητών


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

entity myRegisters is

port
(
clk : in std_logic;
addr_a : in  std_logic_vector(3 downto 0);
addr_b : in std_logic_vector(3 downto 0);
```



Σχήμα 5.2: myRegisters

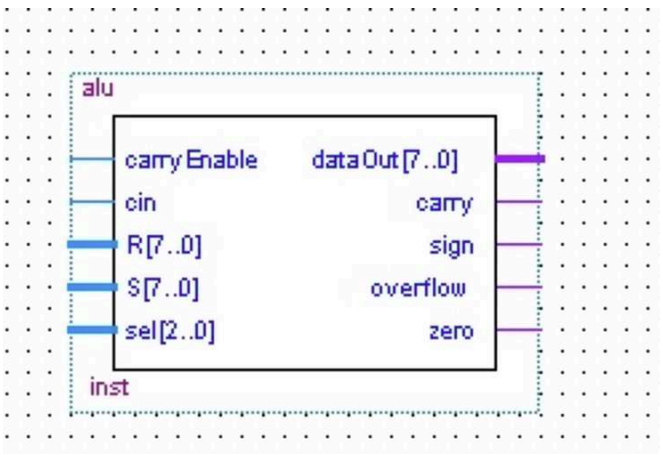
```
data_b : in std_logic_vector((8-1) downto 0);
we_b : in std_logic := '1';
q_a : out std_logic_vector((8 -1) downto 0);
q_b : out std_logic_vector((8 -1) downto 0);

reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,
reg14,reg15 : out std_logic_vector(7 downto 0)
);

end myRegisters;

architecture rtl of myRegisters is

-- Build a 2-D array type for the RAM
subtype word_t is std_logic_vector((8-1) downto 0);
type memory_t is array(2**4-1 downto 0) of word_t;

-- Declare the RAM
shared variable ram : memory_t;

begin
--process(inclock)
process(clk)
begin
if(clk'event and clk='0') then
```

```
if(we_b='1') then
ram(conv_integer(addr_b)):=data_b;
end if;
end if;
end process;

process(addr_a,addr_b,we_b)
begin
q_a<=ram(conv_integer(addr_a));
q_b<=ram(conv_integer(addr_b));
end process;

reg0<=ram(0);
reg1<=ram(1);
reg2<=ram(2);
reg3<=ram(3);
reg4<=ram(4);
reg5<=ram(5);
reg6<=ram(6);
reg7<=ram(7);
reg8<=ram(8);
reg9<=ram(9);
reg10<=ram(10);
reg11<=ram(11);
reg12<=ram(12);
reg13<=ram(13);
reg14<=ram(14);
reg15<=ram(15);
end rtl;
```

Η εγγραφή πραγματοποιείται σύγχρονα στην καθοδική ακμή όταν τα δεδομένα προς εγγραφή έχουν σταθεροποιηθεί, ενώ η ανάγνωση γίνεται ασύγχρονα και ενεργοποιείται με την μεταβολή των διευθύνσεων. Αυτό γίνεται γιατί η Control Unit που προμηθεύει τις διευθύνσεις λειτουργεί με θετικό παλμό οπότε σε μια σύγχρονη υλοποίηση οι καταχωρητες δεν θα προλάβαιναν να διαβαστούν από την Control Unit πριν ξεκινήσει την επεξεργασία τους.

5.3 **Alu**

Το κύκλωμα υπεύθυνο για τις αριθμητικές και λογικές πράξεις.



Σχήμα 5.3: alu

carryEnable Ενεργοποίηση του κρατουμένου στις αριθμητικές πράξεις

cin το κρατούμενο εισόδου(macrocarry)

R αριστερό operand.

S δεξιό operand

sel Η πράξη, βλέπε στήλη I543 του πίνακα σελίδα 18

dataOut Έξοδος της

carry Το κρατούμενο εξόδου

sign Το πρόσημο του αποτελέσματος

overflow Το bit υπερχείλισης/υπεκχείλισης

zero Η ύπαρξη μηδενικού αποτελέσματος.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity alu is
port(
```

```
 carryEnable : in std_logic;
 cin : in std_logic;
 R : in  std_logic_vector(7 downto 0);
 S : in std_logic_vector(7 downto 0);
 sel : in std_logic_vector(2 downto 0);
 dataOut : out std_logic_vector(7 downto 0);
 clk : in std_logic;
 carry : out std_logic;
 sign : out std_logic;
 overflow : out std_logic;
 zero : out std_logic

);

end alu;

architecture alu of alu is

signal result : std_logic_vector(8 downto 0);
signal c: std_logic;


begin
process(sel,R,S,c)
variable Rint : integer range -128 to 127;
variable Sint : integer range -128 to 127;
variable cInt : integer range 0 to 1;
variable tempOverflow : std_logic;
begin
Rint:=conv_integer(R);
Sint:=conv_integer(S);
cInt:=conv_integer(c);
tempOverflow:='0';
case sel is
when "000"=>
result <= '0'&(R + S +c);
if((Rint>0 and Sint>127-(Rint+cInt)) or
 (Rint<0 and Sint<-128-(Rint+cInt)))then
tempOverflow:='1';
end if;
```

```
when"001"=>result<='0'&(S-R+c) ;
if((-Rint>0 and Sint>127+(Rint+cInt)) or
 (-Rint<0 and Sint<-128+(Rint+cInt)))then
tempOverflow:='1';
end if;
when "010"=>result<='0'&(R-S +c) ;
if((Rint>0 and -Sint>127-(Rint+cInt)) or
 (Rint<0 and -Sint<-128-(Rint+cInt)))then
tempOverflow:='1';
end if;
when "011"=>result<='0'&(R or S);
when "100"=>result<='0'&(R and S);
when "101"=>result<='0'&((not R) and S) ;
when "110"=>result<='0'&(R xor S);
when "111"=>result<= '0'&(not(R xor S)) ;
end case;
overflow<=tempOverflow;

end process;

process(carryEnable,cin)
begin
if(carryEnable='1') then
c<=cin;
else
c<='0';
end if;

end process;

process(result)
begin
if (result="000000000") then
zero<='1';
else
zero<='0';
end if;
dataOut<=result(7 downto 0);
carry<=result(8);
sign<=result(7);
end process;
```

```
end alu;
```

Γενικά με την sel επιλέγεται η αριθμητική/λογική πράξη που θα εκτελεστεί

5.3.1 FLAGS

Oveflow Εφόσον έχουμε διαθέσιμα 8 bit και απεικόνιση των αριθμών με μορφή συμπληρώματος ως προς 2, τότε το διαθέσιμο εύρος είναι απο -128 εως 127. Κίνδυνος υπερχείλισης έχουμε είτε όταν προσθέτουμε θετικούς αριθμούς, είτε όταν προσθέτουμε αρνητικούς αριθμούς. Όταν το αποτέλεσμα είναι έξω απο το διαθέσιμο έυρος, τότε αυτό που συμβαίνει είναι το λεγόμενο wraparound σύμφωνα με το οποίο ο επόμενος αριθμός του 127 είναι το -128 και ο προηγούμενος του -128 το 127.

Για παράδειγμα, αν κάνουμε την πράξη 64+64 το αποτέλεσμα θα είναι -128. Χωρίς βλάβη της γενικότητας θα δείξουμε πως γίνεται να εντοπίσεις υπερχείλιση σε περίπτωση αθροίσματος. Τοτε αν ο R είναι θέτικός πρέπει ο S να μην είναι μεγαλύτερος απο τον αριθμό που απομένει αν αφαιρέσουμε τον R απο το 127 που ειναι ο μεγαλύτερος θετικός αριθμός που μπορούμε να αναπαραστήσουμε.

```
if((Rint>0 and Sint>127-(Rint+cInt))
```

Sign Το bit 7 (όγδοο).

Zero Το αποτέλεσμα ισούται με 0. Απλός έλεγχος ισότητας

Carry Το αποτέλεσμα των πράξεων αποθηκέυεται σε καταχωρητή των 9 bit οπέτε το κρατούμενο μετα το πέρας των πράξεων αποθηκεύεται στο πιο σηματνικό bit `carry<=result(8);`

5.4 Dest Selector

Το κύκλωμα αυτό ρυθμίζει τον προορισμό του αποτελέσματος καθώς και την ενδεχόμενη ολίσθηση είτε απλή/διπλή και είτε απλή/κυκλική.

sel Βλέπε στήλη I876 του πίνακα σελίδα 18

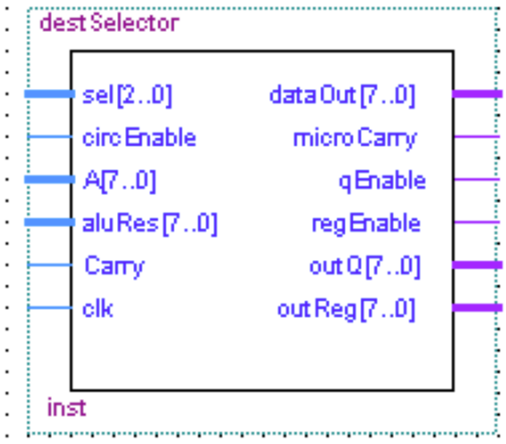
circEnable Επιτρέπει το κυκλικό shift

A Χρειάζεται σε περίπτωση που θα οδηγηθεί στην έξοδο

aluRes Το αποτέλεσμα της alu.

dataOut Η Έξοδος του κυκλώματος.

macroCarry Το Νέο macroCarry.



Σχήμα 5.4: destSelector

Carry Το κρατούμενο εισοδου (macroCarry).

qEnable Η επίτρεψη εγγραφής του Q.

regEnable Η επίτρεψη εγγραφής του καταχωρητή εκ των 16.

outQ Τα δεδομένα προς εγγραφη στον Q.

outReg Τα δεδομένα προς εγγραφη στον καταχωρητή.

```
--destSelector
library ieee;
use ieee.std_logic_1164.all;
entity destSelector is
port(
sel : in std_logic_vector(2 downto 0);
circEnable : in std_logic; -- enable carryIn
A : in std_logic_vector(7 downto 0);
--needed to forward to ouput
aluRes : in std_logic_vector(7 downto 0);
--alu result
dataOut : out std_logic_vector(7 downto 0);
--output
--signals from/to the status register
microCarry : out std_logic;
Carry : in std_logic;
```

```
--outputs to register memory and q register
qEnable : out std_logic;
regEnable :out std_logic;
outQ : out std_logic_vector(7 downto 0);
clk : in std_logic;
outReg : out std_logic_vector(7 downto 0)
);
end destSelector;

architecture destSelector of destSelector is

constant FALSE : std_logic:='0';
constant TRUE : std_logic:='1';

-- reg <---> Q (connected as shown )

begin
process(sel,aluRes,Carry,A,circEnable)
variable Q : std_logic_vector(7 downto 0);
variable reg : std_logic_vector(7 downto 0);
begin
case sel is
when "000" =>  --QREG
dataOut <= aluRes;
Q:=aluRes;
regEnable<=FALSE;
qEnable<=TRUE;
reg:="00000000";
microCarry<='0';
when "001" =>  --NOP
 dataOut <= aluRes;
regEnable<=FALSE;
qEnable<=FALSE;
Q:="00000000";
reg:="00000000";
microCarry<='0';

when "010" => --RAMA
dataOut <= A;
reg:=aluRes;
regEnable<=TRUE;
```

```
qEnable<=FALSE;
Q:="00000000";
microCarry<='0';

when "011" => --RAMF
dataOut <= aluRes;
reg:=aluRes;
regEnable<=TRUE;
qEnable<=FALSE;
Q:="00000000";
microCarry<='0';

when "100" =>
dataOut <= aluRes; --RAMQD
--double shift to the right
Q:=aluRes;
reg:=aluRes;
microCarry<=Q(0);
if(circEnable='1') then
reg(reg'high downto 0):=Carry & reg(reg'high downto 1);
else
reg(reg'high downto 0):='0' & reg(reg'high downto 1);
end if;
Q(Q'high downto 0):=reg(0) & Q(Q'high downto 1);
regEnable<=TRUE;
qEnable<=TRUE;

when "101" =>
dataOut <= aluRes;  --RAMD
reg:=aluRes;
microCarry<=reg(0);
if(circEnable=TRUE) then
reg(reg'high downto 0):=Carry & reg(reg'high downto 1);
else
reg(reg'high downto 0):='0' & reg(reg'high downto 1);
end if;
regEnable<=TRUE;
qEnable<=FALSE;
Q:="00000000";

when "110" =>
```

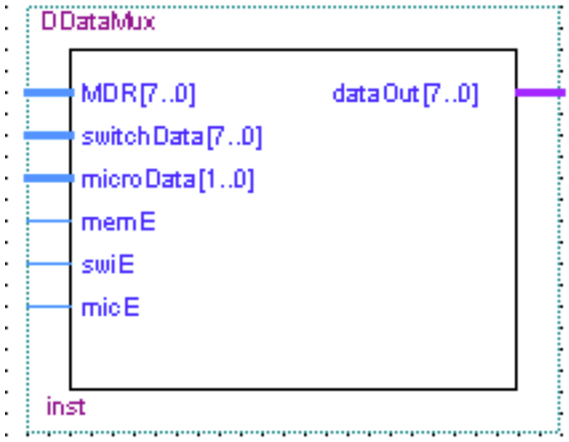
```
dataOut <= aluRes;  --RAMQU
reg:=aluRes;
Q:=aluRes;
microCarry<=reg(7);
reg(reg'high downto 0):=reg(reg'high downto 1)&Q(7);
if(circEnable=TRUE) then
Q(Q'high downto 0):=Q(Q'high -1 downto 0) & Carry;
else
Q(Q'high downto 0):=Q(Q'high -1 downto 0)& '0';
end if;
regEnable<=TRUE;
qEnable<=TRUE;

when "111" =>
dataOut <= aluRes; --RAMU
reg:=aluRes;
microCarry<=reg(7);
if(circEnable=TRUE) then
reg(reg'high downto 0):=reg(reg'high -1 downto 0)& Carry;
else
reg(reg'high downto 0):=reg(reg'high-1 downto 0) & '0';
end if;
regEnable<=TRUE;
qEnable<=FALSE;
Q:="00000000";

end case;
outQ <= Q;
outReg <= reg;
end process;
end destSelector;
```

Σε οποιαδήποτε ολίσθηση το εκτοπιζόμενο bit αποτελεί το microCarry της μικροεντολής.

Στην διπλή ολίσθηση θεωρείται ο 16 bit καταχωρητης που αποτελείται απο τον καταχωρητή reg στα πιο σηματικά bit και απο τον Q αντίστοιχα στα λιγότερο.

```
-- reg <---> Q (connected as shown )
```

Στη κυκλική ολίσθηση το εισερχόμενο bit είναι το κρατούμενο (macroCarry) από την προηγούμενη πράξη.



Σχήμα 5.5: DDataMux

5.5 DDataMux

Το κύκλωμα αυτό ειναι υπεύθυνο για την διαχείριση των άμεσων εξωτερικών δεδομένων.

MDR Διάβασμα δεδομένων απο μνήμη (Memory data register)

switchData Άμεσα δεδομένα απευθείας απο τα switches

microData Άμεσα 2 bit απο την μικροεντολή

memE ενεργοποίηση του διαβάσματος απο τη μνήμη

swiE ενεργοποίηση του διαβάσματος απο τα switches

micE ενεργοποίηση του διαβάσματος απο την μικροεντολή

dataOut έξοδος του κυκλώματος

```
--DDataMux
library ieee;
use ieee.std_logic_1164.all;

entity DDataMux is
port(
MDR : in std_logic_vector(7 downto 0);
```

```
switchData : in std_logic_vector( 7 downto 0);
microData : in std_logic_vector( 1 downto 0);

memE : in std_logic;
swiE : in std_logic;
micE : in std_logic;

dataOut : out std_logic_vector(7 downto 0)
);

end DDataMux;

architecture DDataMux of DDataMux is
begin
process(memE,swiE,micE,microData,MDR,switchData)
begin
if(memE='1') then
dataOut<=MDR;
elsif(swiE='1') then
dataOut<=switchData;
elsif(micE='1') then
dataOut<="000000"&microData;
else
dataout<="00000000";
end if;
end process;
end DDataMux;
```

Το memE αντιστοιχεί στο aluMacro. Το swiE αντιστοιχεί στο switchesAlu. Το micE αντιστοιχεί στο ddataAlu

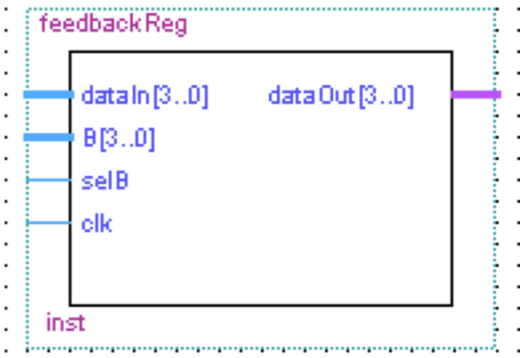
5.6 feedback Register

Το κύκλωμα αυτό ανατροφοδοτεί την έξοδο του destSelector και συγεκριμένα τα 4 λιγότερα σημαντικά bit, ως διεύθυνση στην b port των καταχωρητών.

dataIn τα δεδομένα προς ανατροφοδότηση

B Η διεύθυνση B όπως δίνεται από την μικροεντολή.

selB Με 1 επιλέγει το dataIn.



Σχήμα 5.6: feedbackReg

dataOut η έξοδος του κυκλώματος

```
-- feedbackReg--
library ieee;
use ieee.std_logic_1164.all;

entity feedbackReg is
port ( dataIn : in std_logic_vector(3 downto 0);
B : in std_logic_vector(3 downto 0);
selB : in std_logic;
clk : in std_logic;
-- out
dataOut : out std_logic_vector(3 downto 0)
);
end feedbackReg;

architecture feedbackReg of feedbackReg is
begin
process(selB,B,dataIn)
begin
if(selB='0') then
dataOut <= B;
else
dataOut <= dataIn;
end if;
end process;
end feedbackReg;
```

Στην ουσία το κύκλωμα αυτό υλοποιεί έναν πολυπλέκτη. Η ονομασία είναι αντίστροφη απο αυτή που θα έπρεπε να είναι αλλά αυτό

5.7 Status Register

Το κύκλωμα αυτό καταχωρεί τα microFlags δηλαδη την κατάσταση του συστήματος μετα το πέρας μιας μικροεντολής και τα macroFlags, δηλαδή τα flags που εκφ΄ραζουν την κατάσταση του συστήματος μετά την εκτέλεση μίας μακροεντολής. Στην ουσία δεν είναι τίποτε περισσότερο απο την επιλεκτική αποθήκευση των microFlags.

con επιλέγει την έξοδο Βλέπε στήλη ὃν του πίνακα σελίδα 18

updateFlags Ενημέρωση των macroFlags

y bit συνθήκης που έχει επιλεγεί.

carry κρατούμενο εισόδου

sign πρόσημο εισόδου

overflow υπερχείλιση εισόδου

zero μηδενικό εισόδου

macroCarryOut κρατούμενο μακροεντολής

macroSignOut πρόσημο μακροεντολής

macroOverflowOut bit υπερχείλισης μακροεντολής

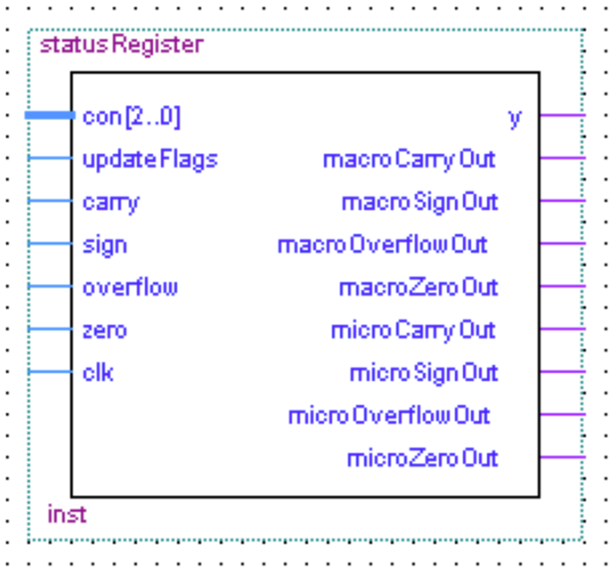
macroZeroOut bit μηδενισμού μακροεντολής

microCarryOut κρατούμενο μικροενολής

macroSignOut πρόσημο μικροεντολής

microOverflowOut bit υπερχείλισης μικροεντολής

microZeroOut bit μηδενισμού εισοδου μικροεντολής



Sq'hma 5.7: statusRegister

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

entity statusRegister is
port
( con : in std_logic_vector(2 downto 0); -- condition
updateFlags : in std_logic:='0'; --refresh data
y : out std_logic; -- output condition bit
carry : in std_logic;
sign : in std_logic;
overflow : in std_logic;
zero : in  std_logic;
clk : in std_logic;
macroCarryOut : out std_logic;
macroSignOut : out std_logic;
macroOverflowOut : out std_logic;
macroZeroOut : out std_logic;
microCarryOut : out std_logic;
microSignOut : out std_logic;
microOverflowOut : out std_logic;
microZeroOut : out std_logic
```

```
);
end statusRegister;

architecture statusRegister of statusRegister is
signal microCarry : std_logic;
signal macroCarry : std_logic;
signal microSign : std_logic;
signal macroSign : std_logic;
signal microOverflow : std_logic;
signal macroOverflow : std_logic;
signal microZero : std_logic;
signal macroZero : std_logic;
begin


--process(clk)
--begin
process(carry,sign,zero,overflow,clk)
begin
microCarry <= carry;
microSign <= sign;
microOverflow <= overflow;
microZero <= zero;
if(clk'event and clk='0') then
if(updateFlags = '1') then
macroCarry <= carry;
macroSign <= sign;
macroOverflow <= overflow;
macroZero <= zero;
end if;
end if;
end process;

with con select
y <= macroCarry when "000",
macroOverflow when "001",
macroSign when "010",
macroZero when "011",
microCarry when "100",
microOverflow when "101",
```

```
microSign when "110",
microZero when "111";


macroCarryOut<=macroCarry; --needed for rotate
--for display purposes
microCarryOut<=microCarry;
macroSignOut<=macroSign;
microSignOut<=microSign;
macroOverflowOut <=macroOverflow;
microOverflowOut<=microOverflow;
microCarryOut<= microCarry;
macroZeroOut<= macroZero;
microZeroOut<= microZero;

end statusRegister;
```

Απλό κύκλωμα το οποίο καταχωρεί μετα απο κάθε πράξη της alu τα flags της μικροενολής και επιλεκτίκα της μακροεντολής. Επίσης βάση της δεδομένης συνθήκης εισόδου που εκφράζεται στην τρέχουσα μικροεντολή επιλέγεται το bit συνθήκης εξόδου το οποίο προωθείται στην Control Unit προκειμένου να υλοποιηθούν τα άλματα υπό συνθήκη.

5.8 EXECUTION UNIT

Οι προαναφερθείσες οντότητες αποτελούν υποσυστήματα της Execution Unit

memAddr Τρέχουσα διεύθυνση της κύριας μνήμης

memData Τρέχουσα δεδομένα της κύριας μνήμης

memWe Επίτρεψη Εγγραφής στην κύρια μνήμη

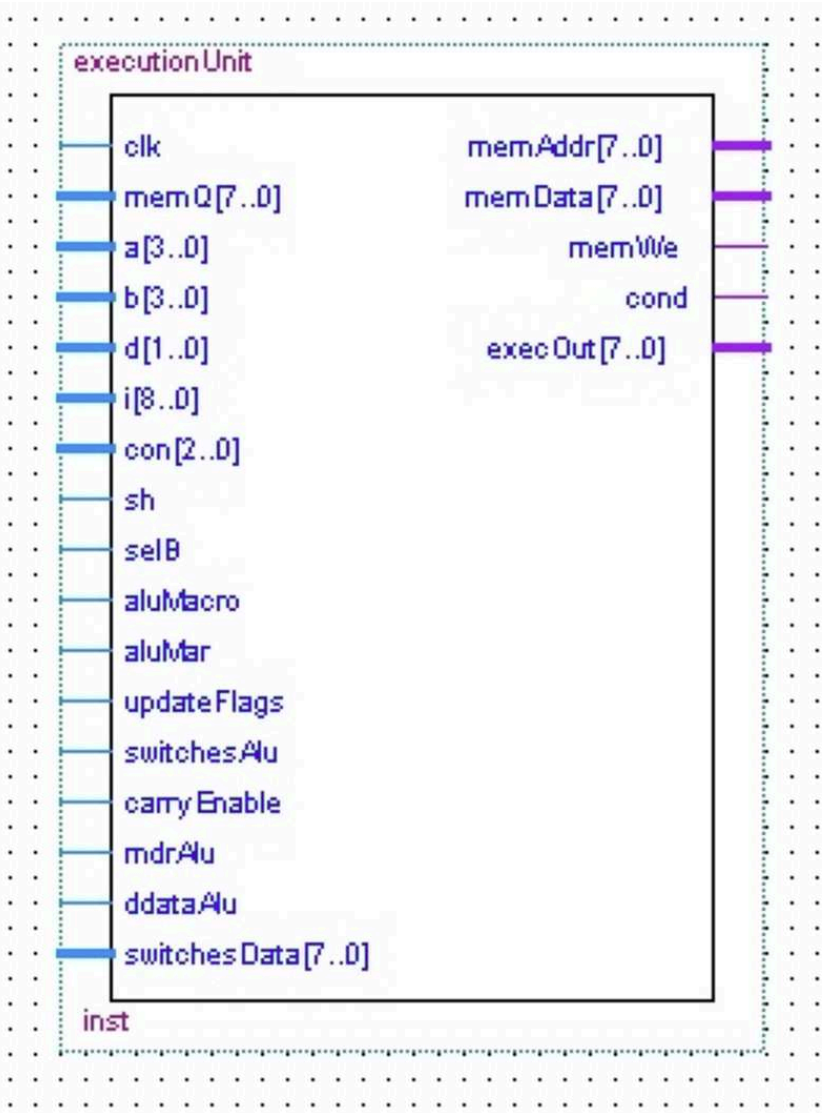
memQ Έξοδος της κύριας μνήμης

a Διεύθυνση της b port.

b Διεύθυνση της b port.

d 2 bit άμεσα δεδομένα

i έντελα,πράξη, προορισμός αποτελέσματος για την alu



Σχήμα 5.8: Execution Unit

con επιλογή flag bit

sh κυκλικό rotate

selB ανατροφοδότηση εξόδου της Execution Unit

aluMacro επίτρεψη εγγραφής για την maromn'hmh

aluMar πρόκληση ανάγνωσης απο την κύρια μνήμη.

updateFlags ενημέρωση των macroFlags.

switchesAlu Χρήση των switches ως άμεση είσοδο

carryEnable Ενεγοποίηση κρατουμένου εισόδου

mdrAlu Χρήση του MDR, διάβασμα απο την κύρια μνήμη

ddataAlu Χρήση των 2 bit ως άμεσων δεδομένων.

cond Το flag bit βάση του con.

switchesData Τα δεδομένα από τα switches

regAddr_aOut Η διεύθυνση του A register

regAddr_bOut Η διεύθυνση του B register

q_aOut Τα περιεχόμενα του A register

q_bOut Τα περιεχόμενα του A register

regData_bOut δεδομένα προς εγγραφή στην b port

regWe_bOut επίτρεψη εγγραφής για την b address.

macroCarryOut κρατούμενο μακροεντολής

macroSignOut πρόσημο μακροεντολής

macroOverflowOut bit υπερχείλισης μακροεντολής

macroZeroOut μηδενικό αποτέλεσμα μακροεντολής.

microCarryOut κρατούμενο μικροεντολής

microSignOut πρόσημο μικροεντολής

microOverflowOut bit υπερχείλισης μικροεντολής.

microZeroOUt μηδενικό αποτέλεσμα μικροεντολής.

aluOut Η έξοδος της alu.

reg0-15 Οι 16 καταχωρητές

MDRdisplay Τα περιεχόμενα του MDR προς απεικόνιση

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

entity executionUnit is
port(
clk : in std_logic;

--signals to memory
memAddr : out std_logic_vector(7 downto 0);
memData : out std_logic_vector(7 downto 0);
memWe : out std_logic;
memQ : in std_logic_vector(7 downto 0);

--signals from control Unit

a : in std_logic_vector(3 downto 0);
b : in std_logic_vector(3 downto 0);
d : in std_logic_vector(1 downto 0);

i : in std_logic_vector(8 downto 0);


con : in std_logic_vector(2 downto 0);

sh : in std_logic;
selB : in std_logic;
aluMacro : in std_logic;
aluMar : in std_logic;
updateFlags : in std_logic;
--mapperSeq : in std_logic;
switchesAlu : in std_logic;
carryEnable : in std_logic;
mdrAlu : in std_logic;
```

```
ddataAlu : in std_logic;

--signals to control Unit
cond : out std_logic;

--signals to DDataMux
switchesData : in std_logic_vector(7 downto 0);
execOut : out std_logic_vector(7 downto 0);--execution output
regAddr_aOut : out std_logic_vector(3 downto 0);
regAddr_bOut : out std_logic_vector(3 downto 0);
q_aOut : out std_logic_vector(7 downto 0);
q_bOut : out std_logic_vector(7 downto 0);
regData_bOut : out std_logic_vector(7 downto 0);
regWe_bOut : out std_logic;
macroCarryOut : out std_logic;
macroSignOut : out std_logic;
macroOverflowOut : out std_logic;
macroZeroOut : out std_logic;
microCarryOut : out std_logic;
microSignOut : out std_logic;
microOverflowOut : out std_logic;
aluOut : out std_logic_vector(7 downto 0);
microZeroOut : out std_logic;
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,
reg14,reg15 : out std_logic_vector(7 downto 0);
MDRdisplay : out std_logic_vector(7 downto 0)

);

end executionUnit;

architecture executionUnit of executionUnit is


component alu is
port(
 carryEnable : in std_logic;
 cin : in std_logic;
 R : in  std_logic_vector(7 downto 0);
 S : in std_logic_vector(7 downto 0);
```

```
 sel : in std_logic_vector(2 downto 0);
 dataOut : out std_logic_vector(7 downto 0);
 clk : in std_logic;
 carry : out std_logic;
 sign : out std_logic;
 overflow : out std_logic;
 zero : out std_logic

);
end component;


component sourceSelector is
port (
 -- out
R : out std_logic_vector(7 downto 0);
S : out std_logic_vector(7 downto 0);
 -- in
sel : in std_logic_vector(2 downto 0);
A : in std_logic_vector(7 downto 0);
B : in std_logic_vector(7 downto 0);
D : in std_logic_vector(7 downto 0);
Q : in std_logic_vector(7 downto 0)

);
end component;


component myRegisters is

port
(
clk : in std_logic;
addr_a : in  std_logic_vector(3 downto 0);
addr_b : in std_logic_vector(3 downto 0);
data_b : in std_logic_vector((8-1) downto 0);
we_b : in std_logic := '1';
q_a : out std_logic_vector((8 -1) downto 0);
q_b : out std_logic_vector((8 -1) downto 0);
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,
```

```
reg13,reg14,reg15 : out std_logic_vector(7 downto 0)
);

end component;

component destSelector is
port(
sel : in std_logic_vector(2 downto 0);
--out

circEnable : in std_logic; -- enable circular shift

A : in std_logic_vector(7 downto 0);--needed to forward to
-- to ouput

aluRes : in std_logic_vector(7 downto 0);--alu result
dataOut : out std_logic_vector(7 downto 0);--output
--signals from/to the status register
microCarry : out std_logic;
Carry : in std_logic;


--outputs to register memory and q register

qEnable : out std_logic;
regEnable :out std_logic;

outQ : out std_logic_vector(7 downto 0);
outReg : out std_logic_vector(7 downto 0)

);
end component;

component qReg is
port (
inData : in std_logic_vector(7 downto 0);
writeEnable : in std_logic;
clk : in std_logic;
outData : out std_logic_vector(7 downto 0)

);
```

```
end component;

component DDataMux is
port(
MDR : in std_logic_vector(7 downto 0);
switchData : in std_logic_vector( 7 downto 0);
microData : in std_logic_vector( 1 downto 0);

memE : in std_logic;
swiE : in std_logic;
micE : in std_logic;

dataOut : out std_logic_vector(7 downto 0)
);

end component;

component feedbackReg is
port ( dataIn : in std_logic_vector(3 downto 0);
B : in std_logic_vector(3 downto 0);
selB : in std_logic;
-- out
dataOut : out std_logic_vector(3 downto 0)
);
end component;

component statusRegister is
port
( con : in std_logic_vector(2 downto 0); -- condition
updateFlags : in std_logic; --refresh data
y : out std_logic; -- output condition bit
carry : in std_logic;
sign : in std_logic;
overflow : in std_logic;
zero : in  std_logic;
clk : in std_logic;
macroCarryOut :  out std_logic;
macroSignOut : out std_logic;
macroOverflowOut : out std_logic;
macroZeroOut : out std_logic;
microCarryOut : out std_logic;
```

```
microSignOut : out std_logic;
microOverflowOut : out std_logic;
microZeroOut : out std_logic
);
end component;
--alu
signal aluR : std_logic_vector(7 downto 0);
signal aluS : std_logic_vector(7 downto 0);
signal aluSel : std_logic_vector(2 downto 0);
signal aluDataOut : std_logic_vector(7 downto 0);


--registers
signal regAddr_b : std_logic_vector(3 downto 0);
signal regAddr_a : std_logic_vector(3 downto 0);
signal regWe_b : std_logic;
signal regData_b : std_logic_vector( 7 downto 0);
signal regQ_a : std_logic_vector(7 downto 0);
signal regQ_b : std_logic_vector(7 downto 0);


--source selector
signal sourceA: std_logic_vector(7 downto 0);
signal sourceB: std_logic_vector(7 downto 0);
signal sourceD: std_logic_vector(7 downto 0);
signal sourceQ: std_logic_vector(7 downto 0);
signal sourceSel : std_logic_vector( 2 downto 0);

--dest selector
signal destSel :  std_logic_vector(2 downto 0);
signal destCirc : std_logic;
signal destA : std_logic_vector(7 downto 0);
signal destAluRes :std_logic_vector(7 downto 0);
signal destCarry : std_logic;


--q register
signal qIN :std_logic_vector(7 downto 0);
signal qWriteEnable : std_logic;
--signal qOut : std_logic_vector(7 downto 0);
```

```
--DData Mux
signal DMDR : std_logic_vector(7 downto 0);
signal DswitchData : std_logic_vector( 7 downto 0);
signal DmicroData : std_logic_vector( 1 downto 0);
signal DmemE : std_logic;
signal DswiE : std_logic;
signal DmicE : std_logic;

--feedback reg
signal feedbackDataIn : std_logic_vector(3 downto 0);
signal feedbackOut : std_logic_vector(3 downto 0);

--status register
signal statusCon : std_logic_vector(2 downto 0);
signal statusUpdateFlags : std_logic;
signal statusCarry : std_logic;
signal statusCarryDest : std_logic;
signal statusCarryAlu : std_logic;
signal statusSign : std_logic;
signal statusOverflow : std_logic;
signal statusZero : std_logic;

--signals with many uses
signal dataOut : std_logic_vector(7 downto 0);

signal macroCarry : std_logic;

signal memAddrStore : std_logic_vector(7 downto 0);
signal memDataStore : std_logic_vector(7 downto 0);

begin

c1 : alu port map( carryEnable=>carryEnable,
 cin=>macroCarry, R=>aluR,
S=>aluS, sel=>aluSel, dataOut=>destAluRes,
carry=>statusCarryAlu,
sign=>statusSign,overflow=>statusOverflow,
zero=>statusZero,clk=>clk);


c2: myRegisters port map(addr_a=>regAddr_a,addr_b=>regAddr_b,
```

```
data_b=>regData_b,clk=>clk,
we_b=>regWe_b,q_a=>sourceA,q_b=>sourceB,
reg0=>reg0,reg1=>reg1,reg2=>reg2,reg3=>reg3,reg4=>reg4,
reg5=>reg5,reg6=>reg6,
reg7=>reg7,reg8=>reg8,reg9=>reg9,reg10=>reg10,reg11=>reg11,
reg12=>reg12,reg13=>reg13,
reg14=>reg14,reg15=>reg15);

c3 : sourceSelector port map(sel=>sourceSel,A=>sourceA,
B=>sourceB,d=>sourceD,
Q=>sourceQ,R=>aluR, S=>aluS);

c4 : destSelector port map(sel=>destSel, circEnable=>sh,
 A=>destA,
aluRes=>destAluRes, Carry=>macroCarry,
microCarry=>statusCarryDest,
dataOut=>dataOut,qEnable=>qWriteEnable,regEnable=>regWe_b,
outQ=>qIN,outReg=>regData_b);

c5: qReg port map(inData=>qIN,writeEnable=>qWriteEnable,
outData=>sourceQ,clk=>clk);

c6 : DDataMux port map(MDR=>memQ,switchData=>switchesData,
microData=>d, memE=>mdrAlu,swiE=>switchesAlu,micE=>ddataAlu
,dataOut=>sourceD);

c7 : feedbackReg port map(dataIn=>feedbackDataIn,
B=>b,selB=>selB,
dataOut=>feedBackOut);

c8 : statusRegister port map(con=>con,
updateFlags=>updateFlags,y=>cond,
carry=>statusCarry,sign=>statusSign,
overflow=>statusOverflow,zero=>statusZero,
clk=>clk,macroCarryOut=>macroCarry,
microCarryOut=>microCarryOut,
macroZeroOut=>macroZeroOut,macroOverflowOut=>macroOverflowOut,
microZeroOut=>microZeroOut,microOverflowOut=>microOverflowOut,
macroSignOut=>macroSignOut,microSignOut=>microSignOut);
```

```
regAddr_a<=a;
regAddr_b<=feedBackOut;

feedbackDataIn<=dataOut(3 downto 0);
 --feedback the least significant bits
destA<=sourceA;


--if shifts of registers are perfomed,
--destSelector updates the carry
--in other cases, alu

process(i,statusCarryDest,statusCarryAlu)
begin
if(i(7)='1') then
statusCarry<=statusCarryDest;
else
statusCarry<=statusCarryAlu;
end if;

sourceSel<=i(2 downto 0);
aluSel<=i(5 downto 3);
destSel<=i(8 downto 6);
end process;

process(clk)
begin
if(clk'event and clk='0') then --want to be steady
if(aluMar='1') then
memAddr<=dataOut;
end if;

if(aluMacro='1') then
memWe<='1';
memData<=dataOut;
MDRdisplay<=dataOut;
else
memWe<='0';
MDRdisplay<=memQ;
end if;
```

```
end if;
end process;

--process(alu,aluMacro) --refresh a virtual MDR
----used for display purposes
--begin
--if(aluMacro'event) then
--MDRdisplay<=dataOut;
--else
--MDRdisplay<=memQ;
--end if;
--end process;


aluOut<=destAluRes;
regAddr_aOut <=regAddr_a;
regAddr_bOut <=regAddr_b;
execOut <= dataOut;
q_aOut<=sourceA;
q_bOut<=sourceB;
regData_bOut <= regData_b;
regWe_bOut<= regWe_b;
macroCarryOut<=macroCarry;
end executionUnit;
```

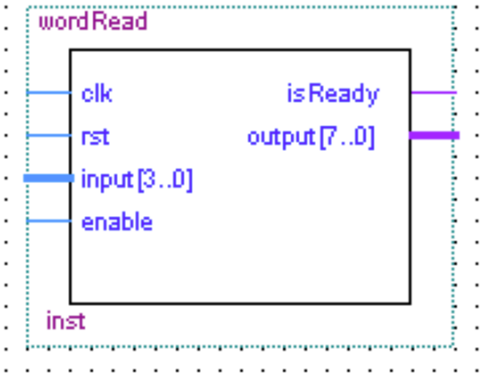
Γενικά η Execution Unit είναι ο συνδετικός ιστός των επιμέρους κυκλωμάτων που την αποτελούν και έχουν περιγραφει.

process(i,statusCarryDest,statusCarryAlu) .....

Στην περίπτωση που γινονται shift (βλ. αναλυτική περιγραφή μικροεντολής), Τυχαίνει να συμβαίνει όταν το πιο σηματικό bit των i876 έιναι ένα τότε το microCarry ανανενώνεται απο το κύκλωμα destSelectorαλλιώς κατευθείαν απο την alu.

if(clk'event and clk='0') then –want to be steady

Η Execution Unit προχωρα σε ενημερωση των MAR,MDR σε καθοδική ακμή ακμή ρολογιού. Αυτό γιατι η Control Unit λειτουργεί σε ανοδική ακμή ρολογιού , προσκομίζει τα δεδομενα προς εκτέλεση και η alu ασύγχρονα, όταν μεταβάλλονται τα δεδομένα των εισόδων της, οπότε με σιγουρια(δεδομενου οτι ο παλμός ρολογιου είναι επαρκής) η Execution Unit έχει τα ορθά δεδομένα στην καθοδική ακμή.



Σχήμα 5.9: wordRead

5.9 Σύνθεση του **SYSTEMINIT**

5.10 wordRead

Το σύστημα αυτό διαβάζει 2 τετράδες των 8 bit αρχίζοντας απο το σημαντικότερο τμήμα.

rst Μηδενισμός του αυτόματου, ανάγνωση απο την αρχή.

input 4 bit εισόδου

isReady 8 bit /διαβάστηκαν, η αναγνωση ολοκληρώθηκε.

enable ενεργοποίηση του αυτόματου που κάνει την ανάγνωση

output 8 bit εξόδου.

```
library ieee;
use  ieee.std_logic_1164.all;
entity wordRead is
port (
clk : in std_logic;
rst : in std_logic;
input : std_logic_vector(3 downto 0);
isReady : out std_logic;
enable : in std_logic;
output :  out std_logic_vector( 7 downto 0)
);
```

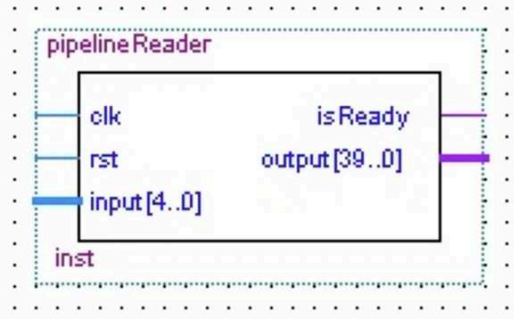
```
end wordRead;

architecture a of wordRead is

type state is(zero,one,final);
signal prState,nxState : state;

shared variable tempInput : std_logic_vector(7 downto 0);
begin
process(prState,input)
begin
case prState is
when zero=>tempInput(7 downto 4):=input;
nxState<=one;isReady<='0'; --first half (high part)
when one=>tempInput(3 downto  0):=input;
nxState<=final;isReady<='0'; --second half
when final=>nxState<=zero;isReady<='1';
end case;
end process;

process(clk,rst,enable)
begin
if(rst='1' or enable='0') then
prState<=zero;
elsif (clk'event and clk='1') then
prState<=nxState;
end if;
end process;
output<=tempInput;
end a;
```

Το σύστημα αυτό έχει τα στάδια *zero,one,final* οπου τα δύο πρώτα αφορούν διάβασμα 4 βιτ το καθένα και το τελευταίο ενεργοποιει το σήμα isFinal που είναι και το σήμα ενεργοποίησης εγγραφής για την μνήμη στην οποία αναφέρεται.

5.11 pipeline Reader

Σύστημα το οποίο πραγματοποιεί τμηματικό διάβασμα των 40 bit της μικροεντολής

rst Επαναφορά στην αρχική κατάσταση διαβάσματος



Σχήμα 5.10: pipelineReader

input 5 bit για τμηματική αναγνωση

isReady το διάβασμα της μικροεντολής ολοκληρώθηκε

enable ενεργοποίηση του αυτομάτου διαβάσματος

output 40 βιτ μικροεντολής

fiveLeds 5 led που υποδεικνύουν πόσα είναι τα διαθέσιμα switches για την εγγραφή.

left7 seven segment display1ο ψηφίο για υπόδειξη του τμήματος που διαβάζεται

middle7 2ο ψηφιό

right7 3ο ψηφίο

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

entity pipelineReader is
port (
clk : in std_logic;
rst : in std_logic;
input : std_logic_vector(4 downto 0);
isReady : out std_logic;
enable : in std_logic;
output :  out std_logic_vector( 39 downto 0);
fiveLeds : out std_logic_vector(4 downto 0);
```

```
left7 : out std_logic_vector(6 downto 0);
middle7 : out std_logic_vector(6 downto 0);
right7 : out std_logic_vector(6 downto 0)
);
end pipelineReader;

architecture a of pipelineReader is
type state is(zero,one,two,three,four,five,
six,seven,eight,nine,ten,final);
signal prState,nxState : state;

signal l7,m7,r7 : std_logic_vector(6 downto 0);
-- abcdefg
constant a7 : std_logic_vector(6 downto 0):= "1110111";
constant b7 : std_logic_vector(6 downto 0):= "0011111";
constant c7 : std_logic_vector(6 downto 0):= "1001110";
constant d7 : std_logic_vector(6 downto 0):= "0111101";
constant e7 : std_logic_vector(6 downto 0):= "1001111";
constant  i7 : std_logic_vector(6 downto 0):= "0110000";
constant n7 : std_logic_vector(6 downto 0):= "0010101";
constant o7 : std_logic_vector(6 downto 0):= "0011101";
constant  ar7 : std_logic_vector(6 downto 0):= "0000111";
constant zero7: std_logic_vector(6 downto 0):= "1111110";
constant two7: std_logic_vector(6 downto 0):= "1101101";
constant three7: std_logic_vector(6 downto 0):= "1111001";
constant four7  : std_logic_vector(6 downto 0):= "0110011";
constant five7: std_logic_vector(6 downto 0):= "1011011";
constant six7: std_logic_vector(6 downto 0):= "1011111";
constant eight7: std_logic_vector(6 downto 0):= "1111111";
constant nine7 : std_logic_vector(6 downto 0):="1111011";
constant nil : std_logic_vector(6 downto 0):= "0000000";

shared variable tempInput : std_logic_vector(39 downto 0);
begin
process(prState,input)
begin
case prState is
when zero=>
tempInput(39 downto 35):=input;
nxState<=one;isReady<='0';
fiveLeds<="11111";l7<=b7;m7<=ar7;r7<=a7;--bra read
```

```
when one=>
tempInput(34 downto 32):=input(2 downto 0);
nxState<=two;isReady<='0';
fiveLeds<="00111";
l7<=b7;m7<=i7;r7<=n7;--bin read
when two=>
tempInput(31 downto 29):=input(2 downto 0);
nxState<=three;isReady<='0';
fiveLeds<="00111";
l7<=c7;m7<=o7;r7<=n7;--con read
when three=>
tempInput(28 downto 26):=input(2 downto 0);
nxState<=four;isReady<='0';
fiveLeds<="00111";
l7<=i7;m7<=two7;r7<=zero7;--i210 read
when four=>
tempInput(25 downto 23):=input(2 downto 0);
nxState<=five;isReady<='0';
fiveLeds<="00111";l7<=i7;m7<=five7;r7<=three7;--i543 read;
when five=>
tempInput(22 downto 20):=input(2 downto 0);
nxState<=six;isReady<='0';
fiveLeds<="00111";
l7<=i7;m7<=eight7;r7<=six7;--i876 read;
when six=>
tempInput(19 downto 16):=input(3 downto 0);
 nxState<=seven;isReady<='0';
 fiveLeds<="01111" ;
 l7<=a7;m7<=nil;r7<=nil;--A read;
when seven=>
tempInput(15 downto 12):=input(3 downto 0);
 nxState<=eight;isReady<='0';
 fiveLeds<="01111";
 l7<=b7;m7<=nil;r7<=nil;--Bread;
when eight=>
tempInput(11 downto 10):=input(1 downto 0);
nxState<=nine;isReady<='0';
fiveLeds<="00011"; l7<=d7;m7<=nil;r7<=nil;--Dread
when nine=>
tempInput(9 downto 5):=input;
nxState<=ten;isReady<='0';
```

```
fiveLeds<="11111";
l7<=c7;m7<=nine7;r7<=five7;-- 5 bits of control read
when ten=>
tempInput(4 downto 0):=input;
nxState<=final;isReady<='0';
fiveLeds<="11111";
l7<=c7;m7<=four7;r7<=zero7;-- next five bits read
when final=>
nxState<=zero;isReady<='1';
fiveLeds<="00000";l7<=e7;m7<=n7;r7<=d7;
end case;
end process;

left7 <=not l7;
right7 <= not  r7;
middle7 <= not m7;

process(clk,rst,enable)
begin
if(rst='1' or enable='0') then
prState<=zero;
elsif (clk'event and clk='1') then
prState<=nxState;
end if;
end process;
output<=tempInput;
end a;
```

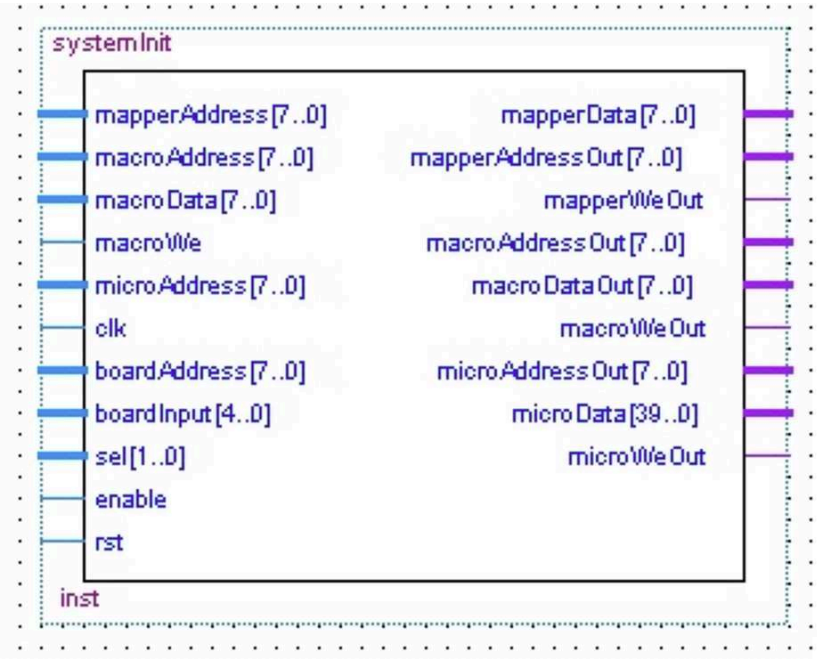
Το κύκλωμα αυτο υλοποιεί την ανάγνωση των μερών της μικροεντολής, τμηματικά με τμήματα που ταυτίζονται με τιν περιγραφή της μικροεντολής, της σελίδας 17, με την μόνη διάφορά οτι τα controlBits διαβάζονται σε 2 πεντάδες.

Επιπλέον το κύκλωμα δεσμεύει 3 ψηφία μέ seven segment displays, που υποδεικνύουν πιο τμήμα της μικροενολής διαβάζεται.

Τέλος leds ενεργοποιούνται κατάλληλα φωτίζοντας ποια dip-switches είναι διαθέσιμα.

5.12 System Init

Το συνολικό σύστημα με το οποίο να παρατηρούμε τα δεδομένα στις μνήμες του συστήματος καθώς και να τα μεταβάλλουμε χρησιμοποιώντας τα dip-switches



Σχήμα 5.11: systemInit

του συστήματος

mapperAddress Διεύθυνση εισόδου για mapper

macroAddress Διέυθυνση εισόδου για μακρομνήμης

macroData Δεδομένα προς εγγραφή της μακρομνήμης

macroWe Ενεργοποίηση εγγραφης της μακρομνήμης

microAddress Διεύθυνση εισόδου μικρομνήμης

boardAddress 7 dip-switches ως διεύθυνση

boardInput 5 dip-switches για εισαγωγή δεδομένων

sel επιλογή της μνήμης ανάγνωσης/τροποποίησης

enable ενεργοποίηση του συστήματος

rst επαναφορα στο αρχικό στάδιο τροποποίησης

enableWriting ενεργοποιεί την εγγραφή του συστήματος

externalReset μετάβαση στην μηδενική διεύθυνση μικροεντολής

mapperData δεδομένα προς εγγραφή στον mapper

mapperAddressOut διεύθυνση στον mapper

mapperWeOut ενεργοποίηση εγγραφής

macroAddressout διεύθυνση μακρομνήμης

macroDataOut δεδομόνα προς εγγραφη για την μακρομνήμη

macroWeOut ενεργοποίηση εγγραφής

microAddressOut διεύθυνση μικρομνήμης

microData δεδομένα προς εγγραφή στην μικρομνήμη

fiveLeds 5 leds που υποδεικνύουν ποιοι διακόπτες είναι διαθέσιμοι στην αρχικοποίηση

left7 3 seven segment displays ψπεύθυνα για την υπόδεικση του τμήματος της μικροεντολής

middle7

right7

microWeOut 40 βιτ της τρέχουσας μικροεντολής προς ανάγνωση/εγγραφή

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

entity systemInit is
port (
mapperAddress : in std_logic_vector(7 downto 0);

macroAddress : in std_logic_vector(7 downto 0);
macroData : in std_logic_vector(7 downto 0);
macroWe : in std_logic;

microAddress : in std_logic_vector(7 downto 0);
clk : in std_logic;
boardAddress : in std_logic_vector(7 downto 0);
```

```
boardInput : in std_logic_vector(4 downto 0);
sel : in std_logic_vector(1 downto 0);
enable : in std_logic;
rst : in std_logic;
enableWriting : in std_logic;
externalReset : in std_logic;
-------------------------------------------------
mapperData : out std_logic_vector(7 downto 0);
mapperAddressOut : out std_logic_vector(7 downto 0);
mapperWeOut : out std_logic;

macroAddressOut : out std_logic_vector(7 downto 0);
macroDataOut : out std_logic_vector(7 downto 0);
macroWeOut : out std_logic;

microAddressOut : out std_logic_vector(7 downto 0);
microData : out std_logic_vector(39 downto 0);
fiveLeds : out std_logic_vector(4 downto 0);
left7 : out std_logic_vector(6 downto 0);
middle7 : out std_logic_vector(6 downto 0);
right7 : out std_logic_vector(6 downto 0);
microWeOut : out std_logic
);
end systemInit;

architecture a of systemInit is

signal mapperAddressTemp : std_logic_vector(7 downto 0);

signal macroAddressTemp : std_logic_vector(7 downto 0);
signal macroDataTemp : std_logic_vector(7 downto 0);
signal macroWeTemp : std_logic;
signal macroDataInit : std_logic_vector(7 downto 0);
signal macroWeInit : std_logic;

signal microAddressTemp : std_logic_vector(7 downto 0);

signal mapReadEnable :std_logic;
signal microReadEnable : std_logic;
signal macroReadEnable : std_logic;
```

```
component wordRead is
port (
clk : in std_logic;
rst : in std_logic;
input : std_logic_vector(3 downto 0);
isReady : out std_logic;
enable : in std_logic;
output :  out std_logic_vector( 7 downto 0)
);
end component;

component pipelineReader is
port (
clk : in std_logic;
rst : in std_logic;
input : std_logic_vector(4 downto 0);
enable : in std_logic;
isReady : out std_logic;
fiveLeds : out std_logic_vector(4 downto 0);
left7 : out std_logic_vector(6 downto 0);
middle7 : out std_logic_vector(6 downto 0);
right7 : out std_logic_vector(6 downto 0);

output :  out std_logic_vector( 39 downto 0)
);
end component;

begin

pipelineRead : pipelineReader port map(clk=>clk,rst=>rst,
input=>boardInput(4 downto 0),
isReady=>microWeOut,output=>microData,
enable=>microReadEnable,fiveLeds=>fiveLeds,
middle7=>middle7,right7=>right7,left7=>left7);

macroWordRead :component wordRead port map(clk=>clk,rst=>rst,
input=>boardInput(3 downto 0),
isReady=>macroWeInit,output=>macroDataInit,
enable=>macroReadEnable);

mapperWordRead : component wordRead port map(clk=>clk,rst=>rst,
```

```
input=>boardInput(3 downto 0),
isReady=>mapperWeOut,output=>mapperData,
enable=>mapReadEnable);

process(enable,sel,mapperAddress,macroAddress,
microAddress,boardAddress,
macroData,macroWe,macroDataInit,macroWeInit,
mapReadEnable,macroReadEnable,
microReadEnable,enableWriting)
begin
if(enable='0') then
mapperAddressTemp<=mapperAddress;
macroAddressTemp<=macroAddress;
microAddressTemp<=microAddress;
macroDataTemp<=macroData;
macroWeTemp<=macroWe;

mapReadEnable<='0';
macroReadEnable<='0';
microReadEnable<='0';
else
case sel is
when "00"=> --arqikopoihsh mapper
mapperAddressTemp<=boardAddress;
 --change mapper
macroAddressTemp<=macroAddress;
microAddressTemp<=microAddress;
macroDataTemp<=macroData;
macroWeTemp<=macroWe;
if(enableWriting='1') then
mapReadEnable<='1';
else
mapReadEnable<='0';
end if;
macroReadEnable<='0';
microReadEnable<='0';
when "01"=> --arqikopoihsh macro
mapperAddressTemp<=mapperAddress;
--change macromemory
macroAddressTemp<=boardAddress;
microAddressTemp<=microAddress;
```

```
macroDataTemp<=macroDataInit;
macroWeTemp<=macroWeInit;
mapReadEnable<='0';
if(enableWriting='1') then
macroReadEnable<='1';
else
macroReadEnable<='0';
end if;
microReadEnable<='0';
when others=>  --arqikopoihsh micro
mapperAddressTemp<=mapperAddress;
-- change micromemory
macroAddressTemp<=macroAddress;
microAddressTemp<=boardAddress;
macroDataTemp<=macroData;
macroWeTemp<=macroWe;
mapReadEnable<='0';
macroReadEnable<='0';
if(enableWriting='1') then
microReadEnable<='1';
else
microReadEnable<='0';
end if;
end case;
end if;
end process;


mapperAddressOut<=mapperAddressTemp;

macroAddressOut<=macroAddressTemp;
macroDataOut<=macroDataTemp;
macroWeOut<=macroWeTemp;

process(externalReset,microAddressTemp)
begin
if(externalReset='1') then
microAddressOut<="00000000";
else
microAddressOut<=microAddressTemp;
end if;
```

```
end process;


end a;
```

Γενικά όταν το System Init δεν είναι ενεργοποιημένο, τοτε λειτουργεί σαν να είναι βραχυκυκλωμένο, οδηγώντας τις εισόδους στις εξόδους του. Οτάν ενεργοποιηθεί τότε ανάλογα με την επιλογή sel μπορούμε να επιλέξουμε την μνήμη που θα δούμε τα περιεχόμενά της και παράλληλα ενεργοποιώντας την εγγραφή (enableWriting) να την τροποποιήσουμε.

Τέλος μπορούμε να οδηγήσουμε την διεύθυνση της μικροεντολής στην μηδενική, οπότε να επαναφέρουμε το σύστημα στην αρχική του κατάσταση.

Με 00 επηρρεάζουμε τον mapper με 01 την μακρομνήμη και με 10 και 11 την μικρομνήμη

Κεφάλαιο 6

PROJECT

Η κύρια οντότητα του σύστήματος καθώς περιλαμ΄βανει την Control Unit, Execution Unit, System Init.

switchesData 8 bit dip-switches

initBoardInput 4 bit dip-switches διαθέσιμα για το συστήμα αρχικοποίησης

initSel επιλογή της μνήμης για το SystemInit

initEnable Ενεργοποιηση του SystemInit

initRst Μηδενισμός κατα τη διάρκεια εισαγωγης δεδομένων για το SystemInit

macroAddressOut Διεύθυνση της μακρομνήμης

microAddressOut Διεύθυνση της μικρομνήμης

execOut Έξοδος της Execution Unit

cFirstMcmdAddrOut Έξοδος του Mapper διύθυνση της πρώτης μικροεντολής

regAddr_aOut διεύθυνση του A καταχωρητή

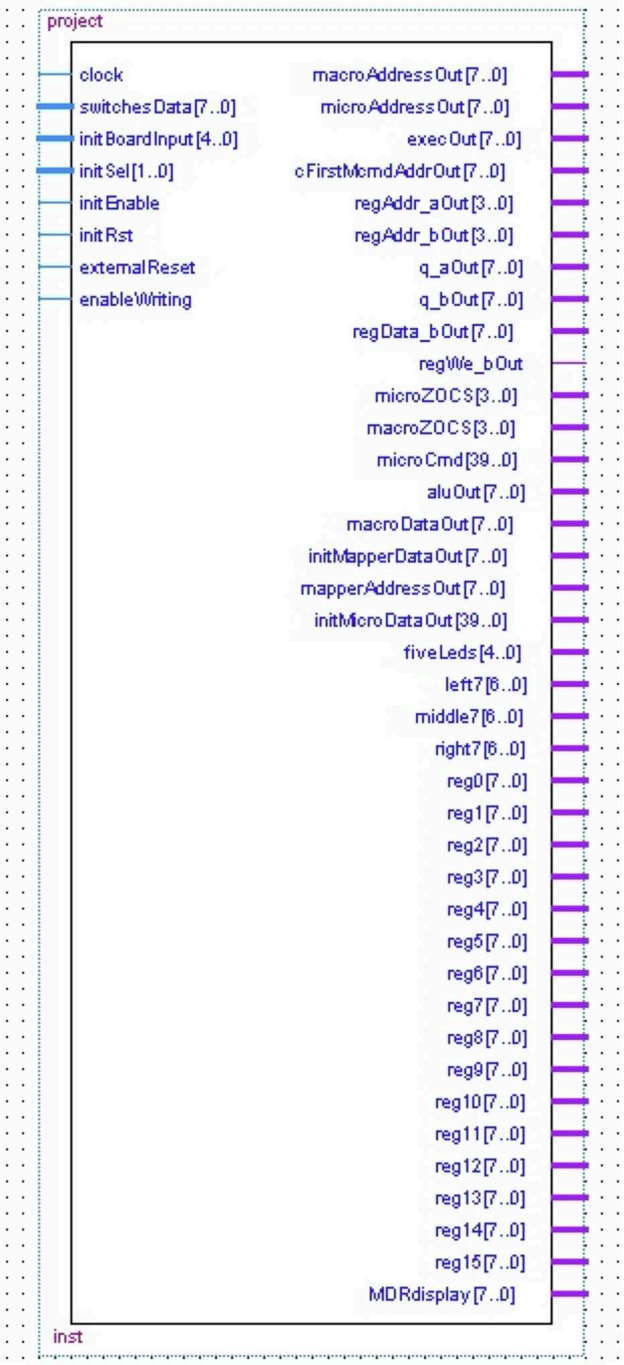
regAddr_bOut διεύθυνση του B καταχωρητή

q_aOut περιεχόμενα του A καταχωρητη

q_bOut περιεχόμεενα του B καταχωρητή

regData_bOut δεδομένα προς εγγραφή του B καταχωρητή

microZOCS microFlags(Zero,Overflow,Carry,Sign)



Σχήμα 6.1: Project

macroZOCS μαςροΦλαγς

microCmd 40 bitτρέχουσας μικροεντολής

aluOut έξοδος της alu

macroDataOut δεδομένα προς εγγραφή για την μακρομνήμη

externalReset επαναφορά του συστήματος στην αρχική κατάσταση

enableWriting επίτρεψη εγγραφής για ο System Init

initMapperDataOut Δεδομένα προς εγγραφή για mapper από System Init

mapperAddressOut Διεύθυνση του mapper

initMicroDataOut Δεδομένα προς εγγραφή για την μικρομνήμη απο το System Init

fiveLeds 5 ledsΓια χρήση απο το σύστημα SystemInit

left7 seven segment displayγια χρήση απο το SystemInit

middle7 seven segment display

right7 σεεν σεγμεντ δισπλαψ

reg0-15 16 καταχωρητές του συστήματος

MDRdisplay εικονικός MDR

```
--system
library ieee;
use ieee.std_logic_1164.all;

entity project is
port(
clock : in std_logic;
switchesData : in std_logic_vector(7 downto 0);
initBoardInput : in std_logic_vector(4 downto 0);
initSel : in std_logic_vector(1 downto 0);
initEnable : in std_logic;
initRst : in std_logic;
```

```
macroAddressOut : out std_logic_vector(7 downto 0);
microAddressOut : out std_logic_vector(7 downto 0);

execOut : out std_logic_vector(7 downto 0);

cFirstMcmdAddrOut : out std_logic_vector(7 downto 0);
regAddr_aOut : out std_logic_vector(3 downto 0);
regAddr_bOut : out std_logic_vector(3 downto 0);
q_aOut : out std_logic_vector(7 downto 0);
q_bOut :  out std_logic_vector(7 downto 0);
regData_bOut : out std_logic_vector(7 downto 0);
regWe_bOut : out std_logic;
microZOCS : out std_logic_vector(3 downto 0);
macroZOCS: out std_logic_vector(3 downto 0);
microCmd : out std_logic_vector(39 downto 0);
aluOut : out std_logic_vector(7 downto 0);
macroDataOut : out std_logic_vector(7 downto 0);

externalReset : in std_logic;
enableWriting : in std_logic;


initMapperDataOut : out std_logic_vector(7 downto 0);
mapperAddressOut : out std_logic_vector(7 downto 0);
initMicroDataOut : out std_logic_vector(39 downto 0);
fiveLeds : out std_logic_vector(4 downto 0);
left7 : out std_logic_vector(6 downto 0);
middle7 : out std_logic_vector(6 downto 0);
right7 : out std_logic_vector(6 downto 0);
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,reg14,
reg15 : out std_logic_vector(7 downto 0);
MDRdisplay : out std_logic_vector(7 downto 0)

);
end project;


architecture project of project is

component macro IS
```

```
PORT
(
address : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
clock : IN STD_LOGIC  := '1';
data : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
wren : IN STD_LOGIC ;
q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END component;



component mapperWritable IS
PORT
(
address : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
clock : IN STD_LOGIC  := '1';
data : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
wren : IN STD_LOGIC ;
q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END component;

component microWritable IS
PORT
(
address : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
clock : IN STD_LOGIC  := '1';
data : IN STD_LOGIC_VECTOR (39 DOWNTO 0);
wren : IN STD_LOGIC ;
q : OUT STD_LOGIC_VECTOR (39 DOWNTO 0)
);
END component;

component controlUnit is
port
(
conditionBit : in std_logic;
firstMcmdAddr : in  std_logic_vector(7 downto 0);

--first microCommand Address
```

```
--retrieved from mapper

microQ : in std_logic_vector(39 downto 0);
-- signal to dmux
d : out std_logic_vector( 1 downto 0);

clock : in std_logic;
-- signals to the execution unit
i : out std_logic_vector( 8 downto 0);
a : out std_logic_vector( 3 downto 0);
b : out std_logic_vector( 3 downto 0);
sh : out std_logic;
selB : out std_logic;
carryEnable : out std_logic; --enable carry
mapperSeq : out std_logic;
switchesAlu : out std_logic;
mdrAlu : out std_logic; --enable mdr as direct data
ddataAlu : out std_logic; --enable the two bit direct data


-- signals to main memory (macromemory)
aluMacro : out std_logic; -- write enable
aluMar : out std_logic; --place output of alu to mar

-- signals to status register
updateFlags : out std_logic;
con : out std_logic_vector(2 downto 0);
microCmd : out std_logic_vector(39 downto 0);
microAddress : out std_logic_vector(7 downto 0)
);
end component;

component executionUnit is
port(
clk : in std_logic;

--signals to memory
memAddr : out std_logic_vector(7 downto 0);
memData : out std_logic_vector(7 downto 0);
memWe : out std_logic;
memQ : in std_logic_vector(7 downto 0);
```

```
--signals from control Unit

a : in std_logic_vector(3 downto 0);
b : in std_logic_vector(3 downto 0);
d : in std_logic_vector(1 downto 0);

i : in std_logic_vector(8 downto 0);


con : in std_logic_vector(2 downto 0);

sh : in std_logic;
selB : in std_logic;
aluMacro : in std_logic;
aluMar : in std_logic;
updateFlags : in std_logic;
--mapperSeq : in std_logic;
switchesAlu : in std_logic;
carryEnable : in std_logic;
mdrAlu : in std_logic;
ddataAlu : in std_logic;

--signals to control Unit
cond : out std_logic;

--signals to DDataMux
switchesData : in std_logic_vector(7 downto 0);
execOut : out std_logic_vector(7 downto 0);
regAddr_aOut : out std_logic_vector(3 downto 0);
regAddr_bOut : out std_logic_vector(3 downto 0);
q_aOut : out std_logic_vector(7 downto 0);
q_bOut : out std_logic_vector(7 downto 0);
regData_bOut : out std_logic_vector(7 downto 0);
regWe_bOut : out std_logic;
macroCarryOut : out std_logic;
macroSignOut : out std_logic;
macroOverflowOut : out std_logic;
macroZeroOut : out std_logic;
microCarryOut : out std_logic;
microSignOut : out std_logic;
```

```
microOverflowOut : out std_logic;
aluOut : out std_logic_vector(7 downto 0);
microZeroOut : out std_logic;
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,
reg14,reg15 : out std_logic_vector(7 downto 0);
MDRdisplay : out std_logic_vector(7 downto 0)

);
--data directly from switches
end component;


component systemInit is
port (
mapperAddress : in std_logic_vector(7 downto 0);
macroAddress : in std_logic_vector(7 downto 0);
macroData : in std_logic_vector(7 downto 0);
macroWe : in std_logic;
microAddress : in std_logic_vector(7 downto 0);
clk : in std_logic;
boardAddress : in std_logic_vector(7 downto 0);
boardInput : in std_logic_vector(4 downto 0);
sel : in std_logic_vector(1 downto 0);
enable : in std_logic;
rst : in std_logic;
enableWriting : in std_logic;
externalReset : in std_logic;
fiveLeds : out std_logic_vector(4 downto 0);
left7 : out std_logic_vector(6 downto 0);
middle7 : out std_logic_vector(6 downto 0);
right7 : out std_logic_vector(6 downto 0);
-------------------------------------------------
mapperData : out std_logic_vector(7 downto 0);
mapperAddressOut : out std_logic_vector(7 downto 0);
mapperWeOut : out std_logic;

macroAddressOut : out std_logic_vector(7 downto 0);
macroDataOut : out std_logic_vector(7 downto 0);
macroWeOut : out std_logic;
```

```
microAddressOut : out std_logic_vector(7 downto 0);
microData : out std_logic_vector(39 downto 0);
microWeOut : out std_logic
);
end component;
```

```
--main memory
signal macroAddress : std_logic_vector(7 downto 0);
signal macroData : std_logic_vector(7 downto 0);
signal macroWE : std_logic;
signal macroQ : std_logic_vector(7 downto 0);
--mapper--
signal mapperAddress : std_logic_vector(7 downto 0);

--controlUnit
signal cConditionBit :  std_logic;
signal cFirstMcmdAddr : std_logic_vector(7 downto 0);
signal microQ : std_logic_vector(39 downto 0);
signal microAddress : std_logic_vector(7 downto 0);

--executionUnit
signal eA : std_logic_vector(3 downto 0);
signal eB : std_logic_vector(3 downto 0);
signal eD : std_logic_vector(1 downto 0);
signal eI : std_logic_vector(8 downto 0);
signal eCon : std_logic_vector( 2 downto 0);
signal eSh : std_logic;
signal eSelB : std_logic;
signal eAluMacro : std_logic;
signal eAluMar : std_logic;
signal eUpdateFlags : std_logic;
signal mapperSeq : std_logic;

signal eSwitchesAlu : std_logic;
signal eCarryEnable : std_logic;
signal eMdrAlu : std_logic;
signal eDdataAlu : std_logic;
```

```
signal eAPeek : std_logic_vector(3 downto 0);
signal eBPeek : std_logic_vector(3 downto 0);
signal eI210Peek : std_logic_vector(2 downto 0);
signal microZero : std_logic;
signal microCarry : std_logic;
signal microOverflow:  std_logic;
signal microSign : std_logic;
signal macroZero :  std_logic;
signal macroCarry : std_logic;
signal macroOverflow : std_logic;
signal macroSign : std_logic;

signal execOutTemp : std_logic_vector(7 downto 0);

--systemInit

signal initMapperAddress : std_logic_vector(7 downto 0);
signal initMapperData : std_logic_vector(7 downto 0);
signal initMapperWe: std_logic;

signal initMicroAddress : std_logic_vector(7 downto 0);
signal initMicroData : std_logic_vector(39 downto 0);
signal initMicroWe : std_logic;

signal initMacroAddress  : std_logic_vector(7 downto 0);
signal initMacroData : std_logic_vector(7 downto 0);
signal initMacroWe : std_logic;

signal initBoardAddress : std_logic_vector(7 downto 0);


begin
c1 : macro port map(clock=>clock,address=>initMacroAddress,
data=>initMacroData,wren=>initMacroWE,q=>macroQ);

c2: mapperWritable port map(clock=> not clock,
address=>initMapperAddress,data=>initMapperData,
wren=>initMapperWe,
q=>cFirstMcmdAddr);

c3 : controlUnit port map(clock=>clock,
```

```
conditionBit=>cConditionBit,
firstMcmdAddr=>cFirstMcmdAddr,d=>eD,i=>eI,a=>eA,
b=>eB,sh=>eSh,selB=>eSelB,carryEnable=>eCarryEnable,
mapperSeq=>mapperSeq,
switchesAlu=>eSwitchesAlu,
mdrAlu=>eMdrAlu,ddataAlu=>eDdataAlu,
aluMacro=>eAluMacro,aluMar=>eAluMar,
updateFlags=>eUpdateFlags,con=>eCon,
microAddress=>microAddress,microCmd=>microCmd,microQ=>microQ);

c4: executionUnit port map(clk=>clock,
memAddr=>macroAddress,
memData=>macroData,memWe=>macroWE,memQ=>macroQ,
a=>eA,b=>eB,d=>eD,
i=>eI,con=>eCon,sh=>eSh,selB=>eSelB,
aluMacro=>eAluMacro,aluMar=>eAluMar,
updateFlags=>eUpdateFlags,switchesAlu=>eSwitchesAlu,
carryEnable=>eCarryEnable,
mdrAlu=>eMdrAlu,ddataAlu=>eDdataAlu,
cond=>cConditionBit,switchesData=>switchesData,
execOut=>execOutTemp,regAddr_aOut=>regAddr_aOut,
regAddr_bOut=>regAddr_bOut,
q_aOut=>q_aOut,q_bOut=>q_bOut,
regData_bOut=>regData_bOut,regWe_bOut=>regWe_bOut,
macroCarryOut=>macroCarry,microCarryOut=>microCarry,
macroZeroOut=>macroZero,macroOverflowOut=>macroOverflow,
microZeroOut=>microZero,microOverflowOut=>microOverflow,
macroSignOut=>macroSign,microSignOut=>microSign,aluOut=>aluOut,
reg0=>reg0,reg1=>reg1,reg2=>reg2,reg3=>reg3,
reg4=>reg4,reg5=>reg5,reg6=>reg6,
reg7=>reg7,reg8=>reg8,reg9=>reg9,reg10=>reg10,
reg11=>reg11,reg12=>reg12,reg13=>reg13,
reg14=>reg14,reg15=>reg15,MDRdisplay=>MDRdisplay);

c5 : microWritable port map(clock=> clock,
address=> initMicroAddress, q=>microQ,data=>initMicroData,
wren=>initMicroWe);

c6 : systemInit port map(mapperAddress=>mapperAddress,
macroAddress=>macroAddress,microAddress=>microAddress,
clk=> clock,macroData=>macroData,macroWe=>macroWE,
```

```
boardAddress=>initBoardAddress,boardInput=>initBoardInput,
sel=>initSel,enable=>initEnable,rst=>not initRst,
   mapperData=>initMapperData,
mapperAddressOut=>initMapperAddress,
mapperWeOut=>initMapperWe,macroAddressOut=>initMacroAddress,
macroDataOut=>initMacroData,
macroWeOut=>initMacroWe,microAddressOut=>initMicroAddress,
microData=>initMicroData,microWeOut=>initMicroWe,
enableWriting=>enableWriting,externalReset=>externalReset,
fiveLeds=>fiveLeds,middle7=>middle7,right7=>right7,
left7=>left7);

process(macroQ)
begin
mapperAddress<=macroQ;
end process;


initBoardAddress<=switchesData;
--signals for display/debugging
macroAddressOut<=initMacroAddress;
microAddressOut<=initMicroAddress;




execOut<=  execOutTemp;
mapperAddressOut <=initMapperAddress;
cFirstMcmdAddrOut<=cFirstMcmdAddr;
macroZOCS<=macroZero&macroOverflow&macroCarry&macroSign;
microZOCS<=microZero&microOverflow&microCarry&microSign;
macroDataOut<=initMacroData;
initMapperDataOut<=initMapperData;
initMicroDataOut<=initMicroData;

end project;
```

Στο σύστημα συναντούμε επιπλέον των οντοτήτων που έχουν περιγραφεί τρεις μνήμες.

1. *macro*μακρομνήμη με διεύθυνση εισόδου 8 και έξοδο 8

2. *mapperWritable* μαππερ με διεύθυνση εισόδου 8 και εΐξοδο 8

3. *microWritable*μικρομνήμη με διεύθυνση εισόδου 8 και εΐξοδο 8

chapterAPEIKONISH DEDOMENWN STHN OJONH

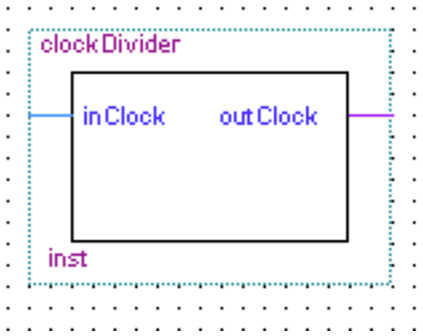
Το κεφάλαιο αυτό περιγράφει τα κυκλώματα σχετική με την απεικόνιση του συστήματος στην οθόνη.

Θα ξεκινήσουμε περιγράφοντας κάποια οιτηιτικά κυκλώματα

6.1 Clock Divider

Το κύκλωμα αυτό πραγματοποιεί διαίρεση με 2 της συχνότητας του εισερχόμενου ρολογιού.

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

entity clockDivider is
port ( inClock : in std_logic;
outClock : buffer std_logic);
end clockDivider;


architecture a of clockDivider is
begin
process(inClock)
begin
if(inClock'event and inClock='1') then
```



Σχήμα 6.2: clockDivider

```
outClock<=not outClock;
end if;
end process;
end a;
```

6.2 Rom Controller

Ο κώδικας παρατίθεται αυτούσιος απο το [8].

charRom Mia apl'i mn'mh 9 bit διεύθυνσης 8 bit εξόδου, η οποία περιλαμβάνει ξαρακτίρες με μορφί πινάκων 8X 8 με την ακόλουθη μορφή.

Ο Rom Controller επιτρέπει να διαχειριζόμαστε την συγκεκριμένη μνήμη και να τυπώνουμε το σωστό pixel για τον εκάστοτε χαρατκήρα.

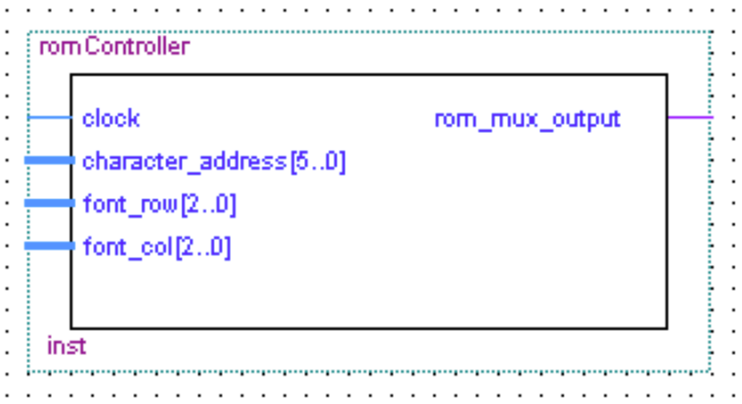
character_address διεύθυνση του ξαρακτήρα

font_row γραμμή του 8X8 πίνακα

font_ col στήλη του 8X8 πίνακα

rom_mux_output το bit της δεδομένης θέσης πίνακα

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY romController IS
PORT( clock : IN STD_LOGIC;
character_address : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
font_row, font_col: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
rom_mux_output : OUT STD_LOGIC);
END romController;

ARCHITECTURE a OF romController IS
SIGNAL rom_data: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL rom_address: STD_LOGIC_VECTOR(8 DOWNTO 0);

component charRom IS
PORT
```



Sq΄hma 6.3: romController

```
010  : 00011000 ; %      **       %
011  : 00111100 ; %    ****       %
012  : 01100110 ; %   **  **      %
013  : 01111110 ; %   ******      %
014  : 01100110 ; %   **  **      %
015  : 01100110 ; %   **  **      %
016  : 01100110 ; %   **  **      %
017  : 00000000 ; %               %
```

Σχήμα 6.4: Pattern 8 X 8

```
(
address : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
clock : IN STD_LOGIC  := '1';
q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END component;
BEGIN

c0 : charRom  port map(address=>rom_address,
clock=>clock,q=>rom_data);

rom_address <= character_address & font_row;
-- Mux to pick off correct rom data bit from 8-bit word
-- for on screen character generation
rom_mux_output <=
 rom_data((CONV_INTEGER(NOT font_col(2 downto 0))));

END a;
```

Με το να δώσουμε μία διεύθυνση των 9 bit στην μικρομνήμη, τότε αυτή μας επιστρέφει μια σειρά των 8 X 8 bit. Απο αυτή την σείρα επιλέγουμε το bit που μας ενδιαφέρει, επομένως και την στήλη.

6.3 VGA_SYNC

Το σύστημα για τον συγχρονισμό της οθόνης [8]

clock_25Mhz ρολόι εισόδου 25Mhz αρκετο για 640X480

red,green,blue είσοδοι 1 bit η κάθεμια

red_out,green_out,blue_out

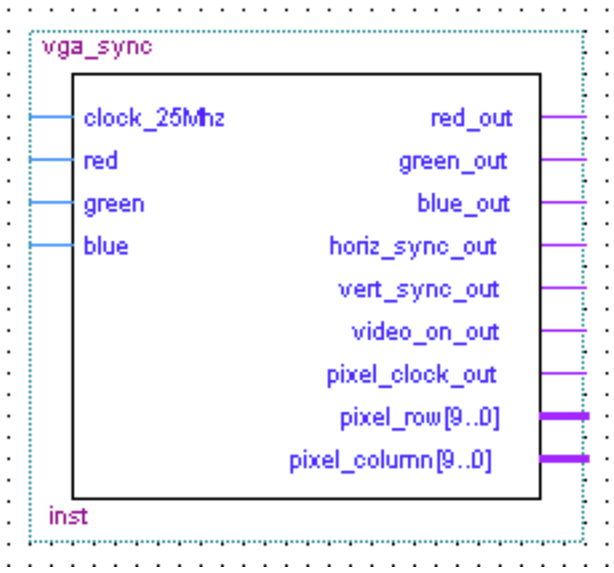
horiz_sync_out χρόνος για οριζόντιο συγχρονισμό

vert_sync_out χ΄ροξνος για κατακόρυφο συγχρονισμό

pixel_clock_out ρολόι της οθόνης

pixel_row γραμμή του ενεργού pixel

pixel_column στήλη του ενεργού pixel



Σχήμα 6.5: myRegisters

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


entity vga_sync is
port( clock_25Mhz, red, green,blue : in std_logic;
red_out, green_out,blue_out : out std_logic;
horiz_sync_out, vert_sync_out : out std_logic;
video_on_out :  out std_logic;
pixel_clock_out : out std_logic;
pixel_row, pixel_column : out std_logic_vector(9 downto 0));
end vga_sync;

architecture a of vga_sync is
signal horiz_sync, vert_sync : std_logic;
signal video_on, video_on_v, video_on_h : std_logic;
signal h_count, v_count : std_logic_vector(9 downto 0);


begin
```

```
video_on <= video_on_h and video_on_v;
pixel_clock_out<=clock_25Mhz;

process
begin
wait until( clock_25Mhz'event) and ( clock_25Mhz='1');

--Generate Horizontal and Vertical
-- Timing signals for video signal
--H_count counts pixels(650 + extra time
-- for sync signals)
--
-- Horiz_sync ------------------------_________------
-- H_count 0 640 659 755   799

if(h_count=799) then
h_count <= "0000000000";
else
h_count <=h_count + 1;
end if;

--Generate Horizontal Sync Signal using H_count

if (h_count <=755 and h_count >=659) then
horiz_sync<='0';
else
horiz_sync<='1';
end if;

--V_count counts row of pixels(480 + extra time
-- for sync signals)
--
-- Vert_sync ---------------------_________---------
-- V_count  0 480   493-494 524

if(v_count>=524 and h_count>=699) then
v_count <="0000000000";
elsif( h_count = 699) then
v_count <=v_count +1;
end if;
```

```
-- Generate Vertical Sync Signal using v_count

if(v_count<=494 and v_count >=493) then
vert_sync<='0';
else
vert_sync<='1';
end if;

--Generate Video on Screen Signals for pixel data

if (h_count<=639) then
video_on_h<='1';
pixel_column<=h_count;
else
video_on_h<='0';
end if;

if(v_count<=479) then
video_on_v<='1';
pixel_row <=v_count;
else
video_on_v<='0';
end if;

red_out <= red and video_on;
green_out <=green and video_on;
blue_out <=blue and video_on;
horiz_sync_out <= horiz_sync;
vert_sync_out <=vert_sync;
video_on_out <=   video_on;
end process;
end a;
```

Για να απεικονίσουμε στην οθόνη πρέπει να της παράσχουμε τα κατάλληλα σήματα για να λειτουργήσει. Αναλόγως την ανάλυση μεταβάλλονται και οι χρονισμοί. Επιλέχθηκε ανάλυση 640X 480 διότι μπορούμε να επιτύχουμε την απαιτούμενη συχνότητα (25.175 Mhz) με διαιρώντας την συχνότητα του διαθέσιμου ρολογιού συστήματος των 50Mhz.

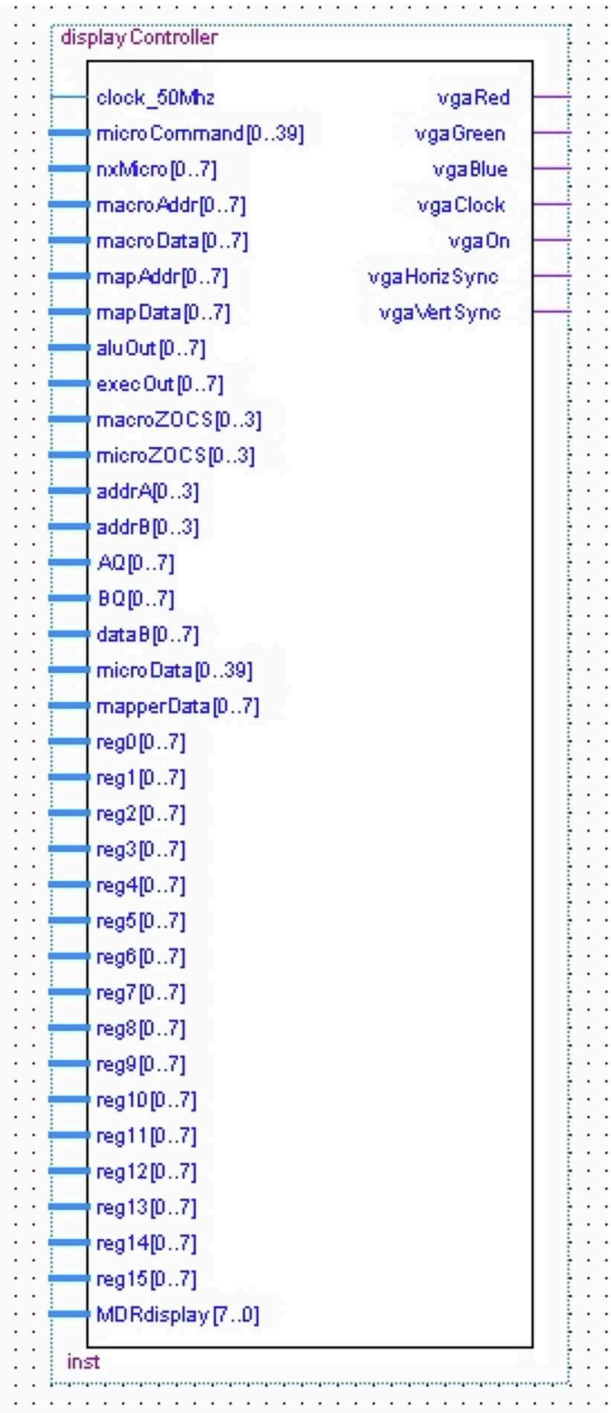
Γενικά, προκειμένου να απεικονίσουμε δεδομένα στην οθόνη πρεπει όχι μόνο να υπάρχει επαρκής χρόνος για τον χρωματισμό του κάθε pixel κατά τη διάρκεια

σχεδιασμού, αλλα πρέπει να δώσουμε ένα περιθώριο συγχρονισμού οπως επίσης και χρόνο για την αλλαγή γραμμής και επαναφορά στην αρχή για σχεδιασμό του νέου frame. Στον επιπλέον χρόνο το χρώμα οδηγείται στο μάυρο.

6.4 displayController

Κύκλωμα υπέυθυνο για την απεικόνιση των σταθερών είτε διανυσμάτων που συνεχώς μεταβάλλονται στον χρόνο.

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


entity displayController is
port(

clock_50Mhz  : in std_logic;
-- data to display
microCommand : in std_logic_vector(0 to 39);
nxMicro : in std_logic_vector(0 to 7);
macroAddr : in std_logic_vector(0 to 7);
macroData : in std_logic_vector(0 to 7);
mapAddr : in std_logic_vector(0 to 7);
mapData : in std_logic_vector(0 to 7);
aluOut : in std_logic_vector(0 to 7);
execOut : in std_logic_vector(0 to 7);
macroZOCS : in std_logic_vector(0 to 3);
microZOCS : in std_logic_vector(0 to 3);
addrA : in std_logic_vector(0 to 3);
addrB : in std_logic_vector(0 to 3);
AQ : in std_logic_vector(0 to 7);
BQ : in std_logic_vector(0 to 7);
dataB : in std_logic_vector(0 to 7);
microData : in std_logic_vector(0 to 39);
mapperData : in std_logic_vector(0 to 7);
reg0,reg1,reg2,reg3,reg4,reg5,
```



Sq'hma 6.6: displayController

```
reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,
reg14,reg15 : in std_logic_vector(0 to 7);

--to vga
vgaRed : out std_logic;
vgaGreen : out std_logic;
vgaBlue : out std_logic;
vgaClock :out std_logic;
vgaOn : out std_logic;
vgaHorizSync : out std_logic;
vgaVertSync : out std_logic


);
end displayController;

architecture a of displayController is

component vga_sync is
port( clock_25Mhz, red, green,blue : in std_logic;
red_out, green_out,blue_out : out std_logic;
horiz_sync_out, vert_sync_out : out std_logic;
video_on_out :  out std_logic;
pixel_clock_out : out std_logic;
pixel_row, pixel_column : out std_logic_vector(9 downto 0));
end component;

component romController IS
PORT( clock : IN STD_LOGIC;
character_address : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
font_row, font_col : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
rom_mux_output : OUT STD_LOGIC);
end component;

component clockDivider is
port ( inClock : in std_logic;
outClock : buffer std_logic);
end component;
```

```
signal clock_25Mhz : std_logic;
signal pixel_clock : std_logic;
signal red_data,green_data,blue_data : std_logic;
signal pixel_row,pixel_col : std_logic_vector(9 downto 0);
signal char_address : std_logic_vector(5 downto 0);



begin
c0 : vga_sync port map(clock_25Mhz=>clock_25Mhz,
red=>red_data,
green=>green_data,blue=>blue_data,
horiz_sync_out=>vgaHorizSync,
vert_sync_out=>vgaVertSync,video_on_out=>vgaOn,
pixel_clock_out=>pixel_clock,
pixel_row=>pixel_row,pixel_column=>pixel_col,
red_out=>vgaRed,blue_out=>vgaBlue,
green_out=>vgaGreen);


c1 : clockDivider port map(inClock=>clock_50Mhz,
outClock=>clock_25Mhz);

c2 : romController port map(clock=>pixel_clock,
character_address=>char_address,
font_row=>pixel_row(2 downto 0),
font_col=>pixel_col(2 downto 0),
rom_mux_output=>green_data);

process(pixel_col,pixel_row,microCommand,
nxMicro,macroAddr,
macroData,mapAddr,
mapData,aluOut,execOut,
macroZOCS,microZOCS,addrA,addrB,AQ,BQ,dataB,
microData,mapperData,
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8,reg9,reg10,
reg11,reg12,reg13,reg14,reg15)
variable i : integer;
begin
-- start displaying from second row
```

```
-- display data
if(pixel_row>=16 and pixel_row<24) then --3rd row
if(pixel_col>=128 and pixel_col<448) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"11"; --I
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; -- R
when "0100" =>char_address<=O"17"; -- O
when "0101" =>char_address<=O"03"; --C
when "0110" =>char_address<=O"15"; --M
when "0111" =>char_address<=O"04"; --D
when others => char_address<=O"40"; --D
end case;
else
char_address<=O"40";
end if;
-- display tags
elsif(pixel_row>=32 and pixel_row<40) then --4th row
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(nxMicro(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"16";  --N
when "0001" =>char_address<=O"30"; --X
when "0010" =>char_address<=O"24"; --T
when "0011" =>char_address<=O"40"; --"  "
when "0100" =>char_address<=O"15"; --M
when "0101" =>char_address<=O"03";--C
when "0110" =>char_address<=O"15";--M
when "0111" =>char_address<=O"04"; --D
when others => char_address<=O"40";--" "
end case;
```

```
else
 char_address<=O"40";
end if;

elsif(pixel_row>=48 and pixel_row<56) then --4th row
if(pixel_col>=128 and pixel_col<448) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microData(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"11"; --I
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; -- R
when "0100" =>char_address<=O"17"; -- O
when "0101" =>char_address<=O"17"; -- O
when "0110" =>char_address<=O"04";--D
when "0111" =>char_address<=O"01"; --A
when "1000" =>char_address<=O"24"; --T
when "1001" =>char_address<=O"01"; --A
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;




elsif(pixel_row>=64 and pixel_row<72) then --4th row
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(macroAddr(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
```

```
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"17"; --O
when "0101" =>char_address<=O"40";-- " "
when "0110" =>char_address<=O"01";--A
when "0111" =>char_address<=O"04"; --D
when "1000" =>char_address<=O"04"; --D
when "1001" =>char_address<=O"22"; --R
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;
elsif(pixel_row>=80 and pixel_row<88) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(macroData(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"17"; --O
when "0101" =>char_address<=O"40";-- " "
when "0110" =>char_address<=O"04";--D
when "0111" =>char_address<=O"01"; --A
when "1000" =>char_address<=O"24"; --T
when "1001" =>char_address<=O"01"; --A
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;

elsif(pixel_row>=96 and pixel_row<104) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
```

```
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(mapAddr(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"20"; --P
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"40";-- " "
when "0101" =>char_address<=O"01";--A
when "0110" =>char_address<=O"04"; --D
when "0111" =>char_address<=O"04"; --D
when "1000" =>char_address<=O"57";-- /
when "1001" =>char_address<=O"15";  --M
when "1010" =>char_address<=O"01"; --A
when "1011" =>char_address<=O"03"; --C
when "1100" =>char_address<=O"22"; --R
when "1101" =>char_address<=O"17"; --O
when "1110" =>char_address<=O"21";--Q

when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;

elsif(pixel_row>=112 and pixel_row<120) then --4th row
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(mapperData(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"20"; --P
when "0011" =>char_address<=O"20"; --P
when "0100" =>char_address<=O"22"; --R
when "0101" =>char_address<=O"40"; -- " "
```

```
when "0110" =>char_address<=O"04";--D
when "0111" =>char_address<=O"01"; --A
when "1000" =>char_address<=O"24"; --T
when "1001" =>char_address<=O"01"; --A
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"22";  --R
when "0001" =>char_address<=O"05"; --E
when "0010" =>char_address<=O"07"; --G
when "0011" =>char_address<=O"23"; --S
when "0100" =>char_address<=O"61"; --1
when "0101" =>char_address<=O"65"; -- 5
when "0110" =>char_address<=O"55";-- -
when "0111" =>char_address<=O"60"; --0
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;

elsif(pixel_row>=128 and pixel_row<136) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(mapData(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"20"; --P
when "0011" =>char_address<=O"20"; --P
when "0100" =>char_address<=O"22"; --R
when "0101" =>char_address<=O"40";-- " "
when "0110" =>char_address<=O"21";--Q
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
```

```
char_address<="110000"+
conv_std_logic_vector(reg15(i),6);
else
 char_address<=O"40";
end if;


elsif(pixel_row>=144and pixel_row<152) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(aluOut(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"01";  --A
when "0001" =>char_address<=O"14"; --L
when "0010" =>char_address<=O"25"; --U
when "0011" =>char_address<=O"40"; --"  "
when "0100" =>char_address<=O"17"; --O
when "0101" =>char_address<=O"25";-- U
when "0110" =>char_address<=O"24";--T
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg14(i),6);
else
 char_address<=O"40";
end if;


elsif(pixel_row>=160 and pixel_row<168) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(execOut(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
```

```
when "0000" =>char_address<=O"05";  --E
when "0001" =>char_address<=O"30"; --X
when "0010" =>char_address<=O"05"; --E
when "0011" =>char_address<=O"03"; --C
when "0100" =>char_address<=O"40"; --"  "
when "0101" =>char_address<=O"17"; --O
when "0110" =>char_address<=O"25";-- U
when "0111" =>char_address<=O"24";--T
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg13(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=176 and pixel_row<184) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(macroZOCS(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"17"; --O
when "0101" =>char_address<=O"32"; --Z
when "0110" =>char_address<=O"17";-- O
when "0111" =>char_address<=O"03";--C
when "1000" =>char_address<=O"23";--S
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg12(i),6);
```

```
else
 char_address<=O"40";
end if;

elsif(pixel_row>=192 and pixel_row<200) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microZOCS(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"11"; --I
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"17"; --O
when "0101" =>char_address<=O"32"; --Z
when "0110" =>char_address<=O"17";-- O
when "0111" =>char_address<=O"03";--C
when "1000" =>char_address<=O"23";--S
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg11(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=208 and pixel_row<216) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(addrA(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"01";  --A
when "0001" =>char_address<=O"04"; --D
```

```
when "0010" =>char_address<=O"04"; --D
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"01"; --A
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg10(i),6);
else
 char_address<=O"40";
end if;


elsif(pixel_row>=224 and pixel_row<232) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(addrB(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"01";  --A
when "0001" =>char_address<=O"04"; --D
when "0010" =>char_address<=O"04"; --D
when "0011" =>char_address<=O"22"; --R
when "0100" =>char_address<=O"02"; --B
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg9(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=240 and pixel_row<248) then
if(pixel_col>=128 and pixel_col<192) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
```

```
char_address<="110000"+
conv_std_logic_vector(dataB(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"04";  --D
when "0001" =>char_address<=O"01"; --A
when "0010" =>char_address<=O"24"; --T
when "0011" =>char_address<=O"01"; --A
when "0100" =>char_address<=O"02"; --B
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg8(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=256 and pixel_row<264) then
case pixel_col(6 downto 3) is
when others=> char_address<=O"55";
end case;

elsif(pixel_row>=272 and pixel_row<280) then
if(pixel_col>=128 and pixel_col<168) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"02";  --B
when "0001" =>char_address<=O"22"; --R
when "0010" =>char_address<=O"01"; --A
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg7(i),6);
```

```
else
 char_address<=O"40";
end if;


elsif(pixel_row>=288 and pixel_row<296) then
if(pixel_col>=128 and pixel_col<152) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+5),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"02";  --B
when "0001" =>char_address<=O"11"; --I
when "0010" =>char_address<=O"16"; --N
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg6(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=304 and pixel_row<312) then
if(pixel_col>=128 and pixel_col<152) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+8),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"03";  --C
when "0001" =>char_address<=O"17"; --O
when "0010" =>char_address<=O"16"; --N
when others => char_address<=O"40";--" "
end case;
```

```
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg5(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=320 and pixel_row<328) then
if(pixel_col>=128 and pixel_col<152) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+11),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"11";  --I
when "0001" =>char_address<=O"62"; --2
when "0010" =>char_address<=O"56"; --,
when "0011" => char_address<=O"60" ;--0
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg4(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=336 and pixel_row<344) then
if(pixel_col>=128 and pixel_col<152) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+14),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"11";  --I
when "0001" =>char_address<=O"65"; --5
when "0010" =>char_address<=O"56"; --,
```

```
when "0011" => char_address<=O"63" ;--3
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg3(i),6);
else
 char_address<=O"40";
end if;


elsif(pixel_row>=352 and pixel_row<360) then
if(pixel_col>=128 and pixel_col<152) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+17),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"11";  --I
when "0001" =>char_address<=O"70"; --8
when "0010" =>char_address<=O"56"; --,
when "0011" => char_address<=O"66" ;--6
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg2(i),6);
else
 char_address<=O"40";
end if;



elsif(pixel_row>=368 and pixel_row<376) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
```

```
conv_std_logic_vector(microCommand(i+20),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"01";  --A
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg1(i),6);
else
 char_address<=O"40";
end if;

elsif(pixel_row>=384 and pixel_row<392) then
if(pixel_col>=128 and pixel_col<160) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+24),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"02";  --B
when others => char_address<=O"40";--" "
end case;
elsif(pixel_col>=256 and pixel_col<320) then
i:=conv_integer(pixel_col( 8 downto 3)) - 32;
char_address<="110000"+
conv_std_logic_vector(reg0(i),6);
else
 char_address<=O"40";
end if;



elsif(pixel_row>=400 and pixel_row<408) then
if(pixel_col>=128 and pixel_col<144) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+28),6);
```

```
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"04";  --D
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;


elsif(pixel_row>=416 and pixel_row<424) then
if(pixel_col>=128 and pixel_col<168) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+30),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"03";  --C
when "0001" =>char_address<=O"71"; --9
when "0010" =>char_address<=O"56"; --,
when "0011" => char_address<=O"65" ;--5
when others => char_address<=O"40";--" "
end case;
else
 char_address<=O"40";
end if;

elsif(pixel_row>=432 and pixel_row<440) then
if(pixel_col>=128 and pixel_col<168) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i+35),6);
elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"03";  --C
when "0001" =>char_address<=O"64"; --5
when "0010" =>char_address<=O"56"; --,
when "0011" => char_address<=O"60" ;--0
when others => char_address<=O"40";--" "
```

```
end case;
else
 char_address<=O"40";
end if;


else
char_address<=O"40";
end if;
end process;

red_data<='0';
blue_data<='0';
vgaClock<=pixel_clock;
end a;
```

Γενικά το κύκλωμα αυτό παρακολουθεί τον σχεδιασμό των pixel τ στην οθόνη (Στην πραγματιότητα ενεργοποιείται ανα 8 pixel) και επεμβαίνει για τον σχεδιασμό των χαρακτήρων στα σωστά σημεία.

```
if(pixel_row>=16 and pixel_row<24) then --3rd row
if(pixel_col>=128 and pixel_col<448) then
--display the 40bits of the microcommand
i:=conv_integer(pixel_col( 8 downto 3)) - 16;
char_address<="110000"+
conv_std_logic_vector(microCommand(i),6);


elsif(pixel_col<128) then
case pixel_col(6 downto 3) is
when "0000" =>char_address<=O"15";  --M
when "0001" =>char_address<=O"11"; --I
when "0010" =>char_address<=O"03"; --C
when "0011" =>char_address<=O"22"; -- R
when "0100" =>char_address<=O"17"; -- O
when "0101" =>char_address<=O"03"; --C
when "0110" =>char_address<=O"15"; --M
when "0111" =>char_address<=O"04"; --D
when others => char_address<=O"40"; --D
end case;
else
```

```
char_address<=O"40";
end if;
```

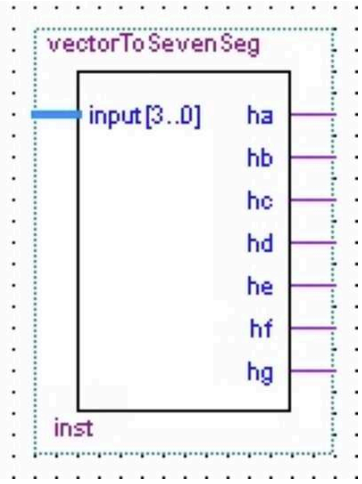
Κεφάλαιο 7

ΒΟΗΘΗΤΙΚΑ ΚΥΚΛΩΜΑΤΑ

7.1 **vectorToSevenSeg**

Κύκλωμα το οποίο δέχεται ως είσοδο ένα διάνυσμα των **4 bit** και οδηγεί κατάλληλα ένα **seven segment display** αναπαριστώντας δηλαδή όλα τα δεκαεξαδικά ψηφία.

```
library ieee;
use ieee.std_logic_1164.all;
--takes 4 bits and drives a seven segment
--outputing 0 to F
entity vectorToSevenSeg is
port(
input : in std_logic_vector(3 downto 0);
ha : out std_logic;
hb : out  std_logic;
hc : out  std_logic;
hd : out std_logic;
he : out std_logic;
hf : out std_logic;
hg : out std_logic);
end vectorToSevenSeg;

architecture a of vectorToSevenSeg is
signal a :std_logic;
signal b: std_logic;
signal c : std_logic;
signal d : std_logic;
```



Σχήμα 7.1: vectorToSevenSeg

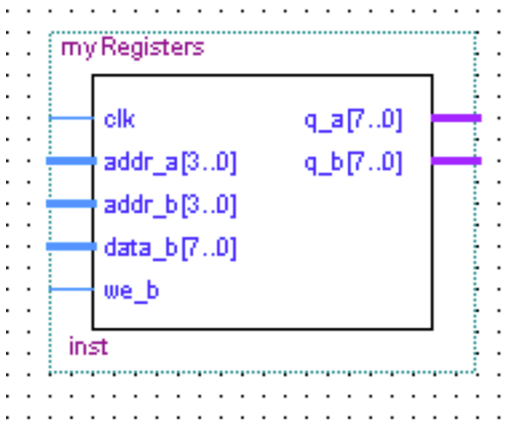
```
signal e : std_logic;
signal f : std_logic;
signal g : std_logic;

begin
process(input)
begin
case input is
when "0000" =>
 a<='1';b<='1';c<='1';d<='1';e<='1';f<='1';g<='0';
when "0001"=>
 a<='0';b<='1';c<='1';d<='0';e<='0';f<='0';g<='0';
when "0010"=>
 a<='1';b<='1';c<='0';d<='1';e<='1';f<='0';g<='1';
when "0011"=>
 a<='1';b<='1';c<='1';d<='1';e<='0';f<='0';g<='1';
when "0100"=>
 a<='0';b<='1';c<='1';d<='0';e<='0';f<='1';g<='1';
when "0101"=>
 a<='1';b<='0';c<='1';d<='1';e<='0';f<='1';g<='1';
when "0110"=>
 a<='1';b<='0';c<='1';d<='1';e<='1';f<='1';g<='1';
when "0111"=>
 a<='1';b<='1';c<='1';d<='0';e<='0';f<='0';g<='0';
when "1000"=>
```

```
 a<='1';b<='1';c<='1';d<='1';e<='1';f<='1';g<='1';
when "1001"=>
 a<='1';b<='1';c<='1';d<='1';e<='0';f<='1';g<='1';
when "1010"=>
 a<='1';b<='1';c<='1';d<='0';e<='1';f<='1';g<='1';
when "1011"=>
 a<='0';b<='0';c<='1';d<='1';e<='1';f<='1';g<='1';
when "1100"=>
 a<='1';b<='0';c<='0';d<='1';e<='1';f<='1';g<='0';
when "1101"=>
 a<='0';b<='1';c<='1';d<='1';e<='1';f<='0';g<='1';
when "1110"=>
 a<='1';b<='0';c<='0';d<='1';e<='1';f<='1';g<='1';
when "1111"=>
 a<='1';b<='0';c<='0';d<='0';e<='1';f<='1';g<='1';
end case;
end process;
ha<= not a;
hb<= not b;
hc<= not c;
hd<= not d;
he<= not e;
hf<= not f;
hg<= not g;
end a;
```

7.2 sevenToSeven - fiveToFive

Απλά κυκλωματα τα οποία διαμερίζουν ενα **bus** στα επιμέρους σήματα.

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

entity sevenToSeven is
port ( input  :  in std_logic_vector(6 downto 0);
a,b,c,d,e,f,g : out std_logic);
end sevenToSeven;
```



Σχήμα 7.2: myRegisters

```
architecture a of sevenToSeven is
begin
a<=input(6);
b<=input(5);
c<=input(4);
d<=input(3);
e<=input(2);
f<=input(1);
g<=input(0);
end  a;


library ieee;
use ieee.std_logic_1164.all;

entity fiveToFive is
port ( input  :  in std_logic_vector(4 downto 0);
out4,out3,out2,out1,out0 : out std_logic);
end fiveTofive;

architecture a of fiveToFive is
begin
out4<=input(4);
out3<=input(3);
out2<=input(2);
out1<=input(1);
out0<=input(0);
```

```
end  a;
```

7.3 clockSelector

Κύκλωμα επιλογέας ρολογιών - πολυπλέκτης.

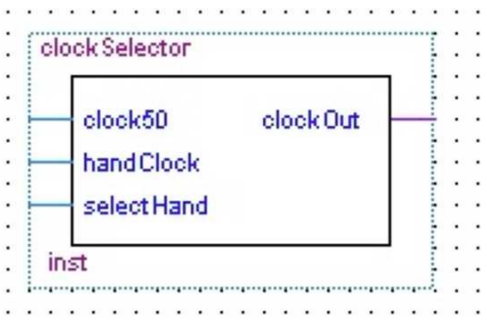
clock50 50Mhz ρολόι συστήματος

handClock χειροκίνητο ρολοί οδηγούμενο απο pushbutton

selectHand Γραμμή επιλογής, με ένα επιλέγεται το χειροκίνητο

clockOut ρολόι εισόδου.

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

entity clockSelector is
port(
clock50 : in std_logic;
handClock : in std_logic;
selectHand : in std_logic;
clockOut : out std_logic);
end clockSelector;

architecture a of clockSelector is
signal clock5 : std_logic;
begin
process(clock50)
```



Σχήμα 7.3: clockSelector

```
variable  count : integer range 0 to 10:=0;
begin
if(clock50'event and clock50='1') then
count:=count+1;
if(count=10) then
count:=0;
clock5<=not clock5;
end if;
end if;
end process;

process(selectHand,clock5,handClock)
begin
case selectHand is
when '1'=>clockOut<=handClock;
when '0' =>clockOut<=clock5;
end case;
end process;
end a;
```

Κεφάλαιο 8

DEFAULT ΑΡΧΙΚΟΠΟΙΗΣΗ

Παρακάτω δίνεται το αρχείο .mif με το οποίο γίνεται η default αρχικοποίηση του συστήματος όπως προλέπεται απο το εργαλείο της altera.

Στην αρχή του αρχείου παρατίθονται οι πιο συχνέχ 9bit ακολουθίες που εκφράζουν τα έντελα τιν πράξη και τον προορισμό του αποτελέσματος αν αυτός υπάρχει.

Επομένως το 001-000-011 εκφράζει οτι τα έντελα που θα προσκομιστούν ειναι τα A και B, η πράξη που θα εφαρμοστεί πάνω σε αυτά ειναι η πρόσθεση και οτι το αποτέλεσμα θα αποθηκευτεί στην μνήμη RAM (macromemory).

Παρακάτω δίνονται οι ακολουθίες τουν 40 bit για κάθε θέση της μικρομνήμης αντίστοιχα.Πανω απο τα 30 πρώτα bit αναγράφεται ο διαμερισμός τους στις ομάδες BRAN-BIN-CON-I20-I53-I86-APOR-BPOR-DD

Επίσης τα 10 τελευταία bit της μικροεντολής αφορουν στα λεγόμενα**control bits** οι ετικέτες τον οποίων διαβάζονται κατακόρυφα.

Αυτά ειναι

circle Η αλλιώς ση, επιτρέπει το κυκλικο rotate

selb ανατροφοδοτεί με 1

mwe οδηγεί την έξοδο της Execution Unit προς εγγραφή στην κύρια μνήμη (macromemory)

aluMar Θέται τον MAR με την εξοδο της Execution Unit.

update Ενημερώνει τα macroFlags

mapper Ξεκινά την διερμηνεία μιας νέας μακροεντολής

switch οδηγεί την είσοδο απο τα dip-switches της πλακέτας ως είσοδο στην alu

carrye Ενεργοποιεί την χρήση κρατουμένου στις αριθμητικές πράξεις.

mdralu Χρησιμοποιεί ως άμεσα δεδομένα το περιεχόμενο του ΜΔΡ

ddataA Χρησιμοποιεί ως άμεσα δεδομένα τα δυο bit της μικροεντολής.

Οι δύο πρώτες θέσεις μνήμης αφορουν τις δύο μικροεντολές για την αρχικοποίηση του συστήματος (bootstrap)

```
-- ta pio syxna I[8:0]
----------------------
-- 001 AB    000 R+S 001 NOP
-- 100 ZA            011 RAMF
-- 101 DA
-- 111 DZ
depth=256;
width=40;
data_radix=bin;
content
begin
--  csmaumscmd
--  iewlpawadd
--  rleudpirra
--  cb-maptrat
--  l--atecyla
--  *    *  *  *  *  *  *   *   * e00rerheuA
--  BRANCBINCONI20I53I86APORBPORDDContrSigna
--  PC<-switches,MAR<-PC
00:   0000000000011100001100000001000001001000 ;
--  MapperSeq
01:   0000000000010000000100010000000000010000 ;
```

Το bootstrap απαρτίζεται απο τις μικροεντολές στις διευθύνσεις 00 και 01.

Η πρώτη μικροεντολή αρχικοποιεί τον PC (Program Counter) ετσι ώστε να κάνουμε άλμα στην θέση της κύριας μνήμης που εμείς επιθυμούμε. Εξετάζοντας ποια control bits είναι ενεργοποιημένα, βλέπουμε οτι αυτά είναι δύο, το aluMar και το switch . Το switch επιτρέπει χρήση των διακοπτών, ως άμεσα δεδομένα, σε συνδιασμό με το 111 του I20 που σημαινει DZ , (αναλυτικότερα βλέπε σελίδα 18) δηλαδή ο αριστερός τελεστής της alu είναι το D απο direct data και ο δεύτερος το 0 (απο ZERO).

Μεταξύ των δύο αυτών τελεστών εκτελείται πρόσθεση R+S αφού το I543 που ορίζει την πράξη είναι το 000.

Τέλος το αποτέλεσματα αποθηκέυονται στην μνήμη σύμφωνα με τα 8

```
-- ta pio syxna I[8:0]
----------------------
-- 001 AB    000 R+S 001 NOP
-- 100 ZA            011 RAMF
-- 101 DA
-- 111 DZ
depth=256;
width=40;
data_radix=bin;
content
begin
--                                              csmaumscmd
--                                              iewlpawadd
--                                              rleudpirra
--                                              cb-maptrat
--                                              l--atecyla
--            *    *  *  *  *  *  *   *   * e00rerheuA
--            BRANCBINCONI20I53I86APORBPORDDContrSigna
--            PC<-switches,MAR<-PC
00:           0000000000011100001100000001000001001000 ;
--            MapperSeq
01:           0000000000010000000100010000000000010000 ;
--firewall hehe
02:           0000000000000000000000000000000000000000 ;
-- that was the bootstrap
-- LDA $K,X  Acc:=M[$K+X] opou $K to periexomeni tis addr K
--                                              csmaumscmd
--                                              iewlpawadd
--                                              rleudpirra
--                                              cb-maptrat
--                                              l--atecyla
--            *    *  *  *  *  *  *   *   * e00rerheuA
---           BRANCBINCONI20I53I86APORBPORDDContrSigna
--            mdr<-M[PC++]
03:           0000000000010100001100010001010001000001;
--            mar<-M[mdr+X]
```

```
04:           00000000000101000001001000000000001000010;
--             acc<-mdr;
05:           00000000000111000011000000000000000000010;
--            mar<-++pc
06:           00000000000101000011000100010100010000001;
--            mapperSeq
07:           00000000000100000001000000000000010000;

-------------------------------------------------------------------
08:           00000000000000000000000000000000000000;

-------------------------------------------------------------------
---LDX #K
--                                              csmaumscmd
--                                              iewlpawadd
--                                              rleudpirra
--                                              cb-maptrat
--                                              l--atecyla
--            *    *  *  *  *  *  *   *   * e00rerheuA
---           BRANCBINCONI20I53I86APORBPORDDContrSigna
--
--            mdr<-M[PC++]
09:           00000000000101000011000100010100010000001;

--            X<-MDR
0A:           00000000000111000011000000100000000000010;
--            mar<-++pc
0B:           00000000000101000011000100010100010000001;
--            mapperSeq
0C:           00000000000100000001000000000000010000;
-----------------------------------------------------------
--INX #K
--                                              csmaumscmd
--                                              iewlpawadd
--                                              rleudpirra
--                                              cb-maptrat
--                                              l--atecyla
--            *    *  *  *  *  *  *   *   * e00rerheuA
---           BRANCBINCONI20I53I86APORBPORDDContrSigna
--            X++
0D:           00000000000101000011001000100100000000001;
```

```
--            mar<-++pc
0E:           00000000000101000011000100010100010000001;
--            mapperSeq
0F:           00000000000100000001000000000000000010000;
-------------------------------------------------------
-- test microcommands
--                                              csmaumscmd
--                                              iewlpawadd
--                                              rleudpirra
--                                              cb-maptrat
--                                              l--atecyla
--            *    *  *  *  *  *  *   *   * e--rerheuA
---           BRANCBINCONI20I53I86APORBPORDDContrSigna
              -- unconditional jum to  subroutine in address 13
10:           00011100000100000001000000000000000000000;
--            mar<-++pc
11:           00000000000101000011000100010100010000001;
--            mapperSeq
12:           00000000000100000001000000000000000010000;
              --unconditional return from subroutine
13:           00000110000101000011000100010100010000001;
[14..FF] :    00000000000000000000000000000000000000000;
end;



depth=256;
width=8;
content
begin

00 : 00;
01 : 02; --LDX #07
02 : 07; --K
03 :        01; --LDA $K,X
04 : 10; --K
05 : 03; --INX
06 : 04; --test microcommands
07 : FF; --data
17 : AA;
end;
```

```
depth=256;
width=8;
content
begin
00 : 00;
01 : 03 ;--LDA $X,K
02 : 09 ;--LDX #K
03 : 0D ;--INX
04 : 10 ;--test microcommands
[05..FF] : 00;
end;
```

Κεφάλαιο 9

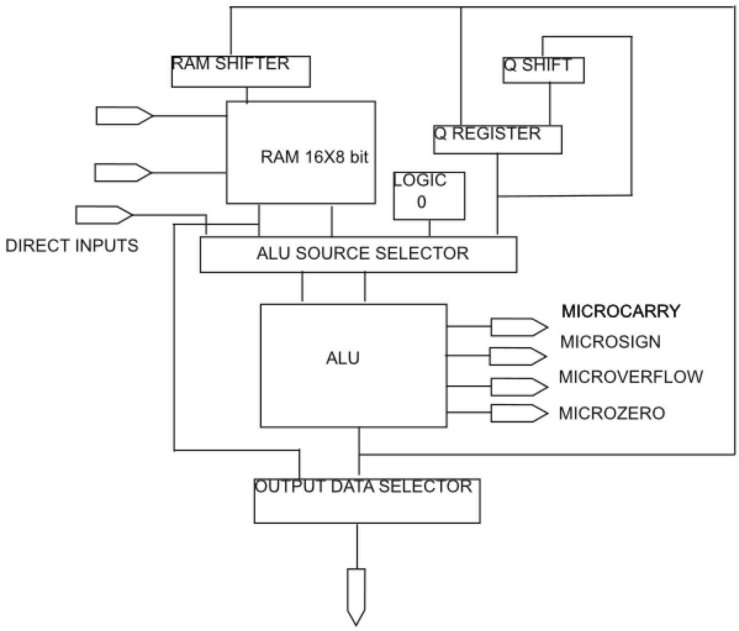
Χρήση του Συστήματος

Η πλακέτα της **altera** περιλαμβάνει 18 διακόπτες χωρις επαναφορά (SW[0-17]) καθώς και 4 διακόπτες με επαναφορά (**push buttons** KEY[0-3]) που είναι κλειστοί απο την κατασκεή τους (δινουν λογικό 1 όταν δεν εφαρμόζεται δύναμη πίεσης).

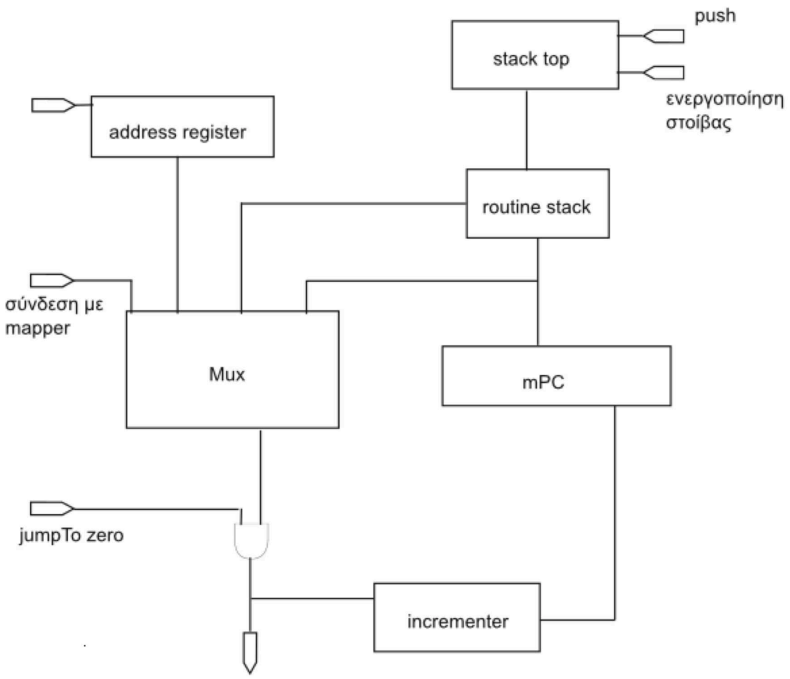
Κεφάλαιο 10

ΠΕΡΙΓΡΑΦΗ ΠΡΩΤΟΤΥΠΟΥ

Το πρωτότυπο βασιζεται στην αριθμητική λογική μονάδα 2901 καθώς και τον ακολουθητή μικροπρογραμματος 2909 της Texas Instruments αντίστοιχα.



Σχήμα 10.1: 2901



Σχήμα 10.2: 2909

Βιβλιογραφία

[1] Μια απλή επεξεργαστική μονάδα Χρ. Καβουσιανός.

[2] Εξομοιωτής Εκπαιδευτικού Μικροπρογραμματιζόμενου υπολογιστή Δ.Νικολός, Χρ. Καβουσιανός

[3] Σχεδίαση Ψηφιακών Συστημάτων με χρήση Η/Ψ Χρ.Καβουσιανός

[4] 6-7 Computer Systems Organizations & Architecture John D.Carpinelli

[5] Διπλωματική εργασία : Σχειδιασμός και ανάπτυξη Μικροπρογραμματιζόμενου Συστήματος, Αγγελής Δημήτρης, Καλαματιανός Γιάννης, Καλαμπούκας Λάμπτρος. Τμήμα Μηχανικών Η/Ψ & Πληροφορικής, Πολυτεχνική σχολή Πατρών.

[6] Μικροπρογραμματιζόμενος εκπαιδευτικός υπολογιστής ver 2.3 Δ.Νικολός, Πολυτεχνική σχολή Πατρών.

[7] Ψηφιακά Συστήματα ΕΑΠ Δ.Νικολός.

[8] Rapid Prototyping of Digital Systems by J.O Hamblen, T.S Hall and M.D Furman.