ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και υλοποίηση μίας Σχετικά απλής CPU με χρήση της Verilog

Ζαφειριάδης Αντώνης ΑΜ: 7132

Σταθόπουλος Γιώργος ΑΜ: 7198

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΚΑΡΕΛΗΣ ΔΗΜΗΤΡΙΟΣ

ΠΑΤΡΑ 2023

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Σε αυτή την πτυχιακή, αρχικά παρέχουμε μια επισκόπηση της αρχιτεκτονικής υπολογιστών και της σχεδίασης μιας CPU χρησιμοποιόντας την Verilog, συμπεριλαμβανομένων των βασικών συστατικών και λειτουργιών μιας CPU και του ρόλου της ISA (αρχιτεκτονικής συνόλου εντολών) στον καθορισμό των δυνατοτήτων της.

Στη συνέχεια περιγράφουμε τη σχεδίαση και την υλοποίηση της απλής μας CPU με τη χρήση Verilog, συμπεριλαμβανομένου του σχεδιασμού του συνόλου εντολών και των διαφόρων στοιχείων της όπως ο κύκλος ανάκτησης-κωδικοποίησης-εκτέλεσης, το αρχείο καταχωρητών και η μονάδα ελέγχου.

Τέλος, παρουσιάζουμε τα αποτελέσματα της προσομοίωσης και της σύνθεσής μας με τη βοήθεια του λογισμικού Quartus της ALTERA, συμπεριλαμβανομένης της ανάλυσης χρονισμού και ισχύος, και συζητάμε τις πιθανές εφαρμογές και τους περιορισμούς της απλής μας cpu.

Συνολικά, η παρούσα διατριβή αποσκοπεί στην παροχή μιας ολοκληρωμένης κατανόησης του σχεδιασμού και της υλοποίησης της CPU με ιδιαίτερη έμφαση στο σχεδιασμό και την υλοποίηση μιας πολύ απλής.

Με την επεξεργασία της σχεδίασης και της υλοποίησης της απλής μας CPU, ελπίζουμε να παρουσιάσουμε τις αρχές και τις τεχνικές που εμπλέκονται στη σχεδίαση και να παράσχουμε μια βάση για περαιτέρω μελέτη και εξερεύνηση αυτού του συναρπαστικού και απαιτητικού πεδίου.

1. Κεντρική μονάδα επεξεργασίας (CPU) και η γλώσσα Verilog.

Η κεντρική μονάδα επεξεργασίας (CPU) είναι ο εγκέφαλος ενός υπολογιστή, υπεύθυνος για την εκτέλεση εντολών και την εκτέλεση υπολογισμών.

Είναι ένα πολύπλοκο και κρίσιμο στοιχείο κάθε συστήματος υπολογιστή και ο σχεδιασμός και η υλοποίησή της απαιτεί βαθιά κατανόηση της αρχιτεκτονικής υπολογιστών και του σχεδιασμού υλικού.

Στην παρούσα διπλωματική εργασία παρουσιάζουμε τη σχεδίαση και υλοποίηση μιας πολύ απλής CPU με τη χρήση της γλώσσας περιγραφής υλικού Verilog.

Η Verilog είναι μια τυποποιημένη γλώσσα περιγραφής υλικού (HDL) που χρησιμοποιείται για τη μοντελοποίηση, προσομοίωση και σύνθεση ψηφιακών λογικών κυκλωμάτων. Χρησιμοποιείται ευρέως στον τομέα της ψηφιακής σχεδίασης και είναι ιδιαίτερα κατάλληλη για τη σχεδίαση CPU, καθώς επιτρέπει τη δημιουργία ιεραρχικών και αρθρωτών σχεδίων που μπορούν εύκολα να προσομοιωθούν και να συντεθούν.

1.1 Η ISA της σχετικάς απλής CPU

Η αρχιτεκτονική συνόλου εντολών (ISA) είναι το σύνολο των προδιαγραφών για έναν υπολογιστή και περγράφουν το σύνολο και την μορφή των εντολών που μπορεί να εκτελέση ένας επεξεργαστής.

Η ISA καθορίζει την σύνταξη και την σημασιολογία των εντολών καθώς και τη συμπεριφορά εισόδου/εξόδου του επεξεργαστή.

Υπάρχουν πολλές διαφορετικές ISA που χρησιμοποιούνται σήμερα, από απλές αρχιτεκτονικές μειωμένου συνόλου εντολών (RISC) έως πιο σύνθετες αρχιτεκτονικές σύνθετου συνόλου εντολών (CISC).

Η επιλογή της ISA εξαρτάται από την προοριζόμενη εφαρμογή της CPU και τους περιορισμούς και συμβιβασμούς σχεδιασμού.

1.2 Επισκόπηση του συνόλου των εντολών

Οι βασικές εντολές σε μια CPU (κεντρική μονάδα επεξεργασίας) είναι οι θεμελιώδεις λειτουργίες που μπορεί να εκτελέσει μια CPU.

Αυτές οι εντολές κωδικοποιούνται συνήθως ως κώδικας μηχανής, ο οποίος είναι μια σειρά δυαδικών τιμών που η CPU μπορεί να ερμηνεύσει και να εκτελέσει.

Ορισμένοι συνήθεις τύποι βασικών εντολών που μπορεί να υποστηρίζει μια CPU περιλαμβάνουν:

- Αριθμητικές εντολές: Αυτές οι εντολές εκτελούν βασικές αριθμητικές πράξεις, όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση.
- Λογικές εντολές: Αυτές οι εντολές εκτελούν λογικές πράξεις όπως AND, OR, XOR και NOT.

- Εντολές μεταφοράς δεδομένων: Αυτές οι εντολές μεταφέρουν δεδομένα μεταξύ της ΚΜΕ και της μνήμης ή μεταξύ διαφορετικών καταχωρητών εντός της ΚΜΕ.
- Εντολές ροής ελέγχου: Αυτές οι εντολές μεταβάλλουν τη ροή εκτέλεσης ενός προγράμματος με διακλάδωση, βρόχο ή άλμα σε διαφορετική θέση του προγράμματος.
- Εντολές εισόδου/εξόδου: Αυτές οι εντολές επιτρέπουν στην ΚΜΕ να επικοινωνεί με συσκευές όπως πληκτρολόγια, ποντίκια και οθόνες.

Η απλή μας CPU περιλαμβάνει τις ακόλουθες εντολές:

- LOAD: Φορτώνει μια τιμή από τη μνήμη σε έναν καταχωρητή.
- STORE: Αποθηκεύει μια τιμή από έναν καταχωρητή στη μνήμη.
- ADD: Προσθέτει δύο τιμές και αποθηκεύει το αποτέλεσμα σε έναν καταχωρητή.
- BRANCH: Μεταβαίνει άνευ όρων σε μια καθορισμένη διεύθυνση στη μνήμη.

Κάθε εντολή κωδικοποιείται χρησιμοποιώντας μια λέξη 16 bit, με τον κωδικό λειτουργίας να αποθηκεύεται στα πρώτα 4 bit και τους τελεστές στα υπόλοιπα 12 bit.

Οι τελεστές κωδικοποιούνται ως εξής:

Οι εντολές LOAD και STORE χρησιμοποιούν έναν προσδιορισμό καταχωρητή 3 bit και μια διεύθυνση μνήμης 12 bit.

Οι εντολές ADD χρησιμοποιούν έναν προσδιορισμό καταχωρητή προορισμού 3-bit, έναν προσδιορισμό καταχωρητή πηγής 3-bit και μια άμεση τιμή 6-bit.

Οι εντολές BRANCH χρησιμοποιούν διεύθυνση μνήμης 12-bit.

Mnemonic	Instruction
ADD	Add
SUB	Subtract
STA	Store
LDA	Load
BRA	Branch always
BRZ	Branch if zero
BRP	Branch if positive
INP	Input
OUT	Output
HLT	End program
DAT	Data location

Σύνολο εντολών της σχετικά απλής cpu

Οι βασικές εντολές συνήθως συνδυάζονται με συγκεκριμένη σειρά για να σχηματίσουν πιο σύνθετα προγράμματα, τα οποία μπορούν να εκτελέσουν ένα ευρύ φάσμα εργασιών.

1.2.1 Οι καταχωρητές

Υπάρχει ένας αριθμός διαφορετικών καταχωρητών που μπορεί να χρειάζονται στη σχεδίαση μιας CPU.
Ορισμένοι συνηθισμένοι τύποι καταχωρητών περιλαμβάνουν:

- Καταχωρητές εντολών: Αυτοί οι καταχωρητές αποθηκεύουν τις οδηγίες που εκτελούνται αυτήν τη στιγμή από τη CPU.
- Καταχωρητές δεδομένων: Αυτοί οι καταχωρητές αποθηκεύουν τα δεδομένα που επεξεργάζεται η ΚΜΕ.

- Καταχωρητές διευθύνσεων: Αυτοί οι καταχωρητές αποθηκεύουν διευθύνσεις μνήμης στις οποίες έχει πρόσβαση ο επεξεργαστής.

- Καταχωρητές μετρητών προγραμμάτων (PC): Αυτοί οι καταχωρητές αποθηκεύουν τη διεύθυνση της επόμενης εντολής που εκτελείται από τον επεξεργαστή.

- Δείκτες στοίβας: Αυτοί οι καταχωρητές αποθηκεύουν στη μνήμη τη θέση της κορυφής της στοίβας.

- Καταχωρητές κατάστασης: Αυτοί οι καταχωρητές αποθηκεύουν πληροφορίες σχετικά με την τρέχουσα κατάσταση της CPU, όπως αν μια αριθμητική λειτουργία είχε ως αποτέλεσμα υπερχείλιση ή αν έχει ληφθεί διακοπή.

1.2.2 Ο Κύκλος Εντολής

Ο κύκλος εντολών, επίσης γνωστός ως κύκλος fetch-execute, είναι η διαδικασία που ακολουθεί η CPU για την εκτέλεση εντολών σε ένα πρόγραμμα.

Ο κύκλος διδασκαλίας αποτελείται από τα ακόλουθα βήματα:

- Fetch: Η CPU ανακτά την επόμενη οδηγία που εκτελείται από τη μνήμη.

- Αποκωδικοποίηση: Η CPU αποκωδικοποιεί την οδηγία για τον προσδιορισμό της λειτουργίας που καθορίζει.

- Εκτέλεση: Η CPU εκτελεί τη λειτουργία που καθορίζεται από την οδηγία.

- WriteBack: Αν η εντολή παράγει αποτέλεσμα, η CPU αποθηκεύει το αποτέλεσμα σε έναμητρώο ή μια θέση μνήμης.

Αυτά τα βήματα επαναλαμβάνονται για κάθε εντολή σε ένα πρόγραμμα, μέχρι το πρόγραμμα να ολοκληρώσει την εκτέλεση.

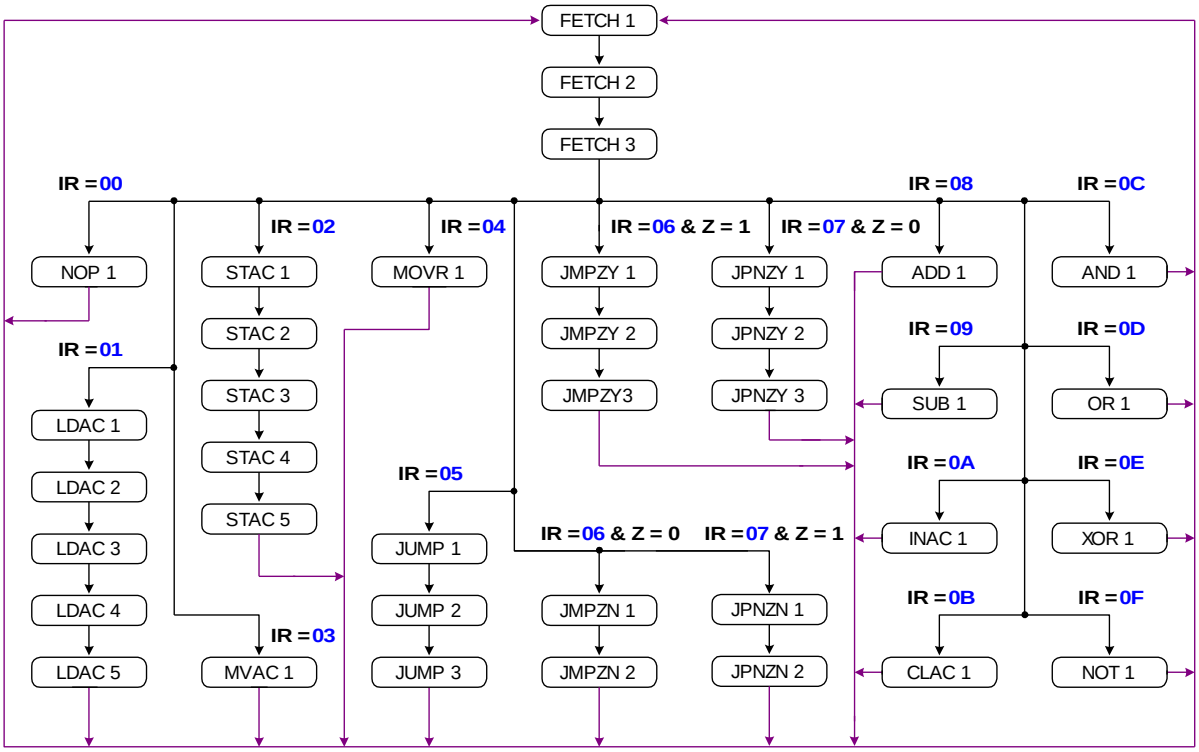
Ο κύκλος λειτουργίας είναι μια σημαντική έννοια στην αρχιτεκτονική του υπολογιστή, καθώς καθορίζει τον τρόπο με τον οποίο η CPU επεξεργάζεται τις οδηγίες και εκτελεί τις λειτουργίες.

Η ταχύτητα και η αποδοτικότητα του κύκλου λειτουργίας μπορεί να έχει σημαντική επίδραση στη συνολική απόδοση ενός υπολογιστή.

Η κατάσταση FETCH αντιπροσωπεύει το βήμα στον κύκλο εντολών, όπου η CPU ανακτά την επόμενη οδηγία από τη μνήμη.

Η κατάσταση DECODE αντιπροσωπεύει το βήμα στο οποίο η CPU αποκωδικοποιεί την οδηγία για τον προσδιορισμό της λειτουργίας που πρέπει να εκτελεστεί.
Η κατάσταση EXECUTE αντιπροσωπεύει το βήμα όπου ο CPU εκτελεί τη λειτουργία που καθορίζεται από την οδηγία.
Η κατάσταση WRITEBACK αντιπροσωπεύει το βήμα στο οποίο η CPU αποθηκεύει το αποτέλεσμα της λειτουργίας, αν ισχύει.
Η κατάσταση IDLE αντιπροσωπεύει μια αδρανή κατάσταση όπου η CPU δεν εκτελεί οδηγίες.

Για παράδειγμα, ο κύκλος εξελίσσεται από την κατάσταση FETCH στην κατάσταση DECODE μετά την ανάκτηση της εντολής από τη μνήμη.



Πλήρες διάγραμμα καταστάσεων.

1.2.3 Η Εσωτερική οργάνωση της CPU και τα στοιχεία της

Η εσωτερική οργάνωση μιας CPU (κεντρική μονάδα επεξεργασίας) αναφέρεται στον τρόπο με τον οποίο τα διάφορα στοιχεία της CPU είναι διατεταγμένα και πώς αλληλεπιδρούν μεταξύ τους. Η συγκεκριμένη εσωτερική οργάνωση μιας CPU μπορεί να διαφέρει ανάλογα με το σχεδιασμό και την αρχιτεκτονική της CPU.

Μερικά κοινά στοιχεία μιας CPU που συμβάλλουν στην εσωτερική της οργάνωση περιλαμβάνουν:

- Καταχωρητές: Αυτές είναι μικρές θέσεις αποθήκευσης εντός της CPU που χρησιμοποιούνται για την προσωρινή αποθήκευση δεδομένων και οδηγιών.

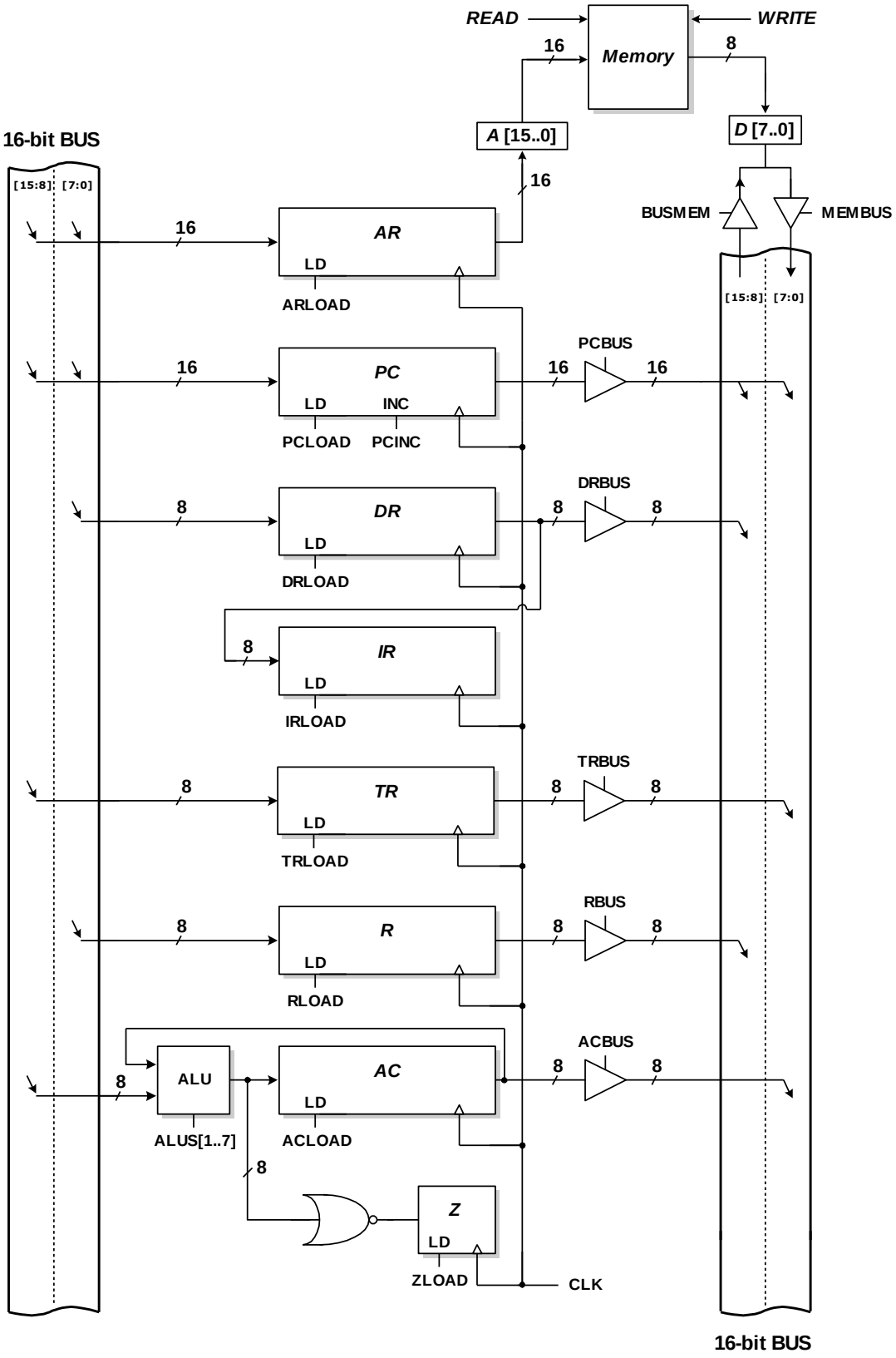
- Αριθμητική λογική μονάδα (ALU): Είναι ένα στοιχείο της CPU που εκτελεί αριθμητικές και λογικές λειτουργίες στα δεδομένα.

- Μονάδα ελέγχου: Αυτή είναι μια συνιστώσα της CPU που ελέγχει τη ροή των δεδομένων και των οδηγιών εντός της CPU και συντονίζει τη λειτουργία των άλλων συστατικών.

- Αποκωδικοποιητής εντολών: Αυτό είναι ένα στοιχείο της CPU που αποκωδικοποιεί τις οδηγίες και καθορίζει τη λειτουργία που πρέπει να εκτελέσει η CPU.

- Μνήμη cache: Αυτή είναι μια μικρή ποσότητα μνήμης υψηλής ταχύτητας που χρησιμοποιείται για την αποθήκευση δεδομένων και οδηγιών με συχνή πρόσβαση, για τη βελτίωση της απόδοσης της CPU.

Η εσωτερική οργάνωση μιας CPU μπορεί επίσης να περιλαμβάνει άλλα στοιχεία, όπως μονάδες διαχείρισης μνήμης, ελεγκτές διακοπών και διασυνδέσεις διαύλου, τα οποία χρησιμοποιούνται για τη διαχείριση της επικοινωνίας και της αλληλεπίδρασης της CPU με άλλα στοιχεία του υπολογιστή.



Η εσωτερική οργάνωση (data path) για την σχετικά απλή cpu

1.2.4 Η Αριθμητική/Λογική Μονάδα ALU

Η αριθμητική λογική μονάδα (ALU) είναι ένα στοιχείο της CPU που εκτελεί αριθμητικές και λογικές λειτουργίες στα δεδομένα.
Η ALU είναι ένα σημαντικό μέρος της CPU, καθώς είναι υπεύθυνη για την εκτέλεση πολλών από τις λειτουργίες που καθορίζονται από τις οδηγίες σε ένα πρόγραμμα.

Εδώ είναι μερικά βασικά πράγματα που πρέπει να ξέρετε για την ALU:

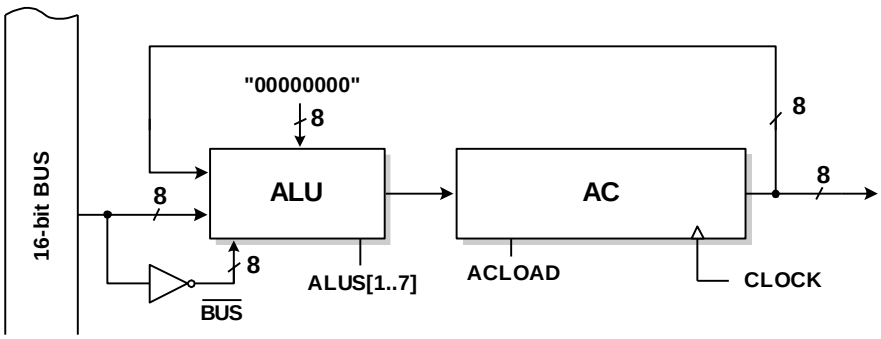
- Η ALU εκτελεί βασικές αριθμητικές πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση. Μπορεί επίσης να εκτελέσει λογικές πράξεις όπως AND, OR, XOR και NOT.

- Η ALU συνήθως ελέγχεται από τη μονάδα ελέγχου της CPU, η οποία καθορίζει ποια λειτουργία θα πρέπει να εκτελέσει η ALU με βάση την εντολή που εκτελείται.

- Η ALU μπορεί να περιλαμβάνει μια σειρά από καταχωρητές ή άλλους τόπους αποθήκευσης για την τήρηση ενδιάμεσων αποτελεσμάτων και την εκτέλεση των εργασιών σε αυτά.

Η απόδοση της ALU μπορεί να έχει σημαντικό αντίκτυπο στη συνολική απόδοση της CPU.

Ταχύτερες και πιο αποδοτικές ALUs επιτρέπουν στον επεξεργαστή να εκτελεί εντολές πιο γρήγορα.



Διασύνδεση Αριθμητικής/Λογικής μονάδας

1.2.5 Δίαυλος Δεδομένων

Στην κεντρική μονάδα επεξεργασίας (CPU) ενός υπολογιστή, ο δίαυλος δεδομένων αναφέρεται στις διαδρομές που χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ διαφορετικών στοιχείων της CPU και άλλων συσκευών του υπολογιστή.
Αυτές οι οδοί συνήθως περιλαμβάνουν διαδρόμους και καταχωρητές.

Οι δίαυλοι χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ διαφορετικών στοιχείων εντός της CPU, όπως μεταξύ της ALU και των καταχωρητών, ή μεταξύ των καταχωρητών και της μνήμης.

Σε μια CPU, οι δίαυλοι είναι ένα σύνολο καλωδίων ή ιχνών που χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ των διαφόρων στοιχείων της CPU. Συνήθως υπάρχουν τρεις τύποι διαύλων σε μια ΚΜΕ:

- Δίαυλος δεδομένων: είναι ένας αμφίδρομος δίαυλος που χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ των καταχωρητών της ΚΜΕ και της μνήμης, καθώς και μεταξύ των διαφόρων λειτουργικών μονάδων της ΚΜΕ (π.χ. της ALU και του αρχείου καταχωρητών). Το πλάτος του διαύλου δεδομένων καθορίζει τον όγκο των δεδομένων που μπορούν να μεταφερθούν κάθε φορά. Για παράδειγμα, ένας δίαυλος δεδομένων 8 bit μπορεί να μεταφέρει 8 bit δεδομένων κάθε φορά, ενώ ένας δίαυλος δεδομένων 16 bit μπορεί να μεταφέρει 16 bit δεδομένων κάθε φορά.

- Δίαυλος διευθύνσεων: Πρόκειται για ένα μονόδρομο που χρησιμοποιείται για τον προσδιορισμό της θέσης μνήμης ή του καταχωρητή στον οποίο γίνεται πρόσβαση. Το πλάτος του διαύλου διευθύνσεων καθορίζει τη μέγιστη ποσότητα μνήμης που μπορεί να προσπελάσει η ΚΜΕ. Για παράδειγμα, ένας δίαυλος διευθύνσεων 8 bit μπορεί να απευθύνεται σε μνήμη έως 256 bytes (2^8), ενώ ένας δίαυλος διευθύνσεων 16 bit μπορεί να απευθύνεται σε μνήμη έως 64K bytes (2^16).

- Δίαυλος ελέγχου: Πρόκειται για ένα σύνολο σημάτων που χρησιμοποιείται για τον έλεγχο της λειτουργίας των διαφόρων στοιχείων της ΚΜΕ. Ο δίαυλος ελέγχου περιλαμβάνει σήματα όπως το ρολόι, η επαναφορά και η ενεργοποίηση, τα οποία χρησιμοποιούνται για το συγχρονισμό της λειτουργίας της ΚΜΕ και των εξαρτημάτων της.

Μαζί, αυτοί οι δίαυλοι παρέχουν ένα μέσο μεταφοράς δεδομένων μεταξύ των διαφόρων συστατικών της ΚΜΕ, επιτρέποντας στην ΚΜΕ να εκτελεί εντολές και να εκτελεί λειτουργίες σε δεδομένα. Η συγκεκριμένη υλοποίηση των διαύλων εξαρτάται από την αρχιτεκτονική της ΚΜΕ και τις ειδικές απαιτήσεις της σχεδίασης.

Οι καταχωρητές είναι μικρές θέσεις μνήμης εντός της CPU που χρησιμοποιούνται για την προσωρινή αποθήκευση δεδομένων κατά την εκτέλεση των οδηγιών.

Υπάρχουν πολλοί διαφορετικοί τύποι καταχωρητών, συμπεριλαμβανομένων των καταχωρητών γενικής χρήσης, των καταχωρητών κινητής υποδιαστολής και των καταχωρητών ειδικής χρήσης, που χρησιμοποιούνται για διάφορους σκοπούς, όπως η αποθήκευση τιμών μετρητή προγραμμάτων, η αποθήκευση των αποτελεσμάτων των αριθμητικών πράξεων ή η διατήρηση δεδομένων που μεταφέρονται μεταξύ της CPU και άλλων συσκευών.

Εκτός από το κανάλι δεδομένων εντός της CPU, υπάρχει επίσης ένα κανάλι δεδομένων μεταξύ της CPU και άλλων συσκευών στον υπολογιστή, όπως η μνήμη, οι συσκευές I/O και η εξωτερική αποθήκευση.
Αυτό το κανάλι δεδομένων χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ της CPU και αυτών των συσκευών, όπως απαιτείται.

Συνολικά, ο δίαυλος δεδομένων παίζει κρίσιμο ρόλο στη λειτουργία ενός επεξεργαστή, καθώς επιτρέπει τη μεταφορά δεδομένων μεταξύ διαφορετικών στοιχείων και συσκευών και επιτρέπει στον επεξεργαστή να έχει πρόσβαση και να επεξεργάζεται τα δεδομένα που απαιτούνται για την εκτέλεση εντολών.

1.2.6 Μικροπρογραμματισμένη CPU και Σχεδίαση της Μονάδας Έλεγχου

Σε μια μικροπρογραμματισμένη CPU, η λογική ελέγχου που κατευθύνει τις λειτουργίες του επεξεργαστή υλοποιείται με τη χρήση μικροκώδικα αποθηκευμένου σε μνήμη μόνο για ανάγνωση (ROM) ή σε προγραμματιζόμενη μνήμη μόνο για ανάγνωση (PROM).

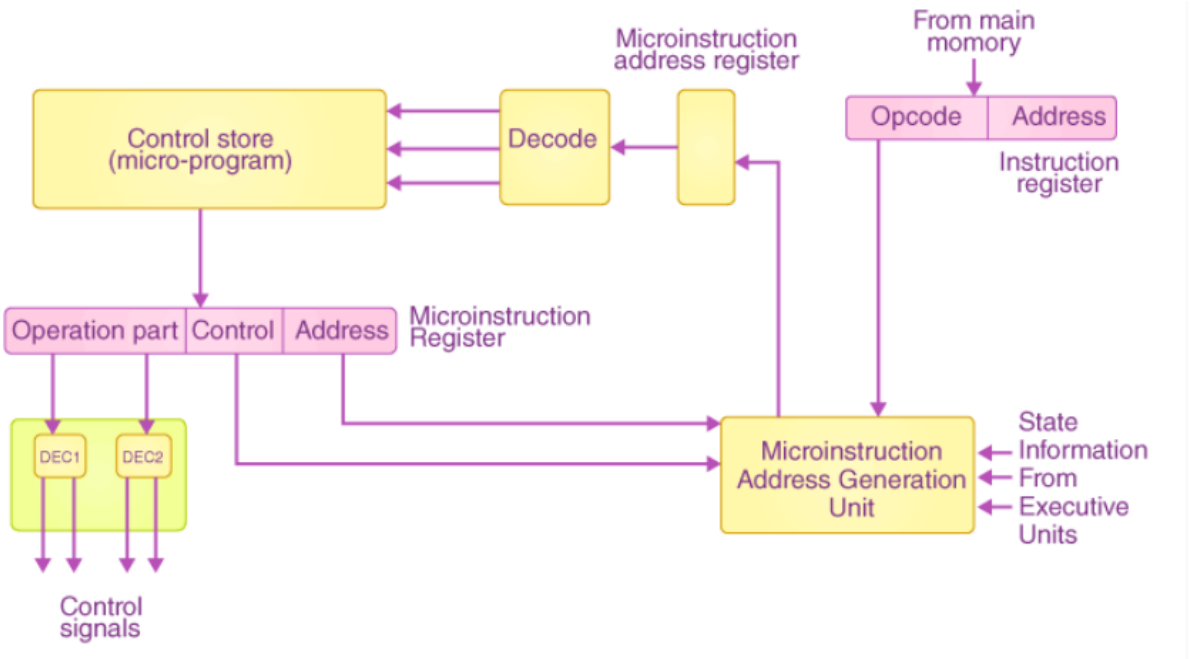
Αυτός ο μικροκώδικας είναι ένα σύνολο εντολών χαμηλού επιπέδου που χρησιμοποιεί η CPU για την εκτέλεση πιο σύνθετων εντολών υψηλότερου επιπέδου.

Μια μικροπρογραμματισμένη CPU έχει δύο κύρια στοιχεία: τη μονάδα ελέγχου και τη μονάδα εκτέλεσης.

- Η μονάδα ελέγχου περιέχει τη μνήμη ROM μικροκώδικα και έναν μικροακολουθητή, ο οποίος είναι υπεύθυνος για την ανάκτηση και αποκωδικοποίηση των εντολών μικροκώδικα.
- Η μονάδα εκτέλεσης είναι υπεύθυνη για την εκτέλεση των πράξεων που καθορίζονται από τον μικροκώδικα.

Όταν λαμβάνεται μια εντολή υψηλού επιπέδου από την CPU, η μονάδα ελέγχου ανακτά τον αντίστοιχο μικροκώδικα από τη ROM και τον αποστέλλει στον μικροακολουθητή για αποκωδικοποίηση.

Στη συνέχεια, ο μικροακολουθητής παράγει μια σειρά σημάτων ελέγχου που χρησιμοποιούνται για να κατευθύνουν τη μονάδα εκτέλεσης να εκτελέσει τις απαραίτητες λειτουργίες για την εκτέλεση της εντολής.



microprogrammed control unit1

Ένα από τα πλεονεκτήματα της χρήσης μικροκώδικα είναι ότι επιτρέπει την εύκολη τροποποίηση της λογικής ελέγχου του επεξεργαστή χωρίς να απαιτείται ο επανασχεδιασμός ολόκληρου του επεξεργαστή.

Αυτό καθιστά εύκολη την προσθήκη νέων εντολών στον επεξεργαστή ή την πραγματοποίηση άλλων αλλαγών στο σύνολο εντολών.

Ο μικροκώδικας διευκολύνει επίσης τον χειρισμό των λειτουργιών χαμηλού επιπέδου για την αρχιτεκτονική, τον χειρισμό διαφορετικής μορφής εντολών, την ανάλυση τους και στη συνέχεια τη μετάφρασή τους σε μια ακολουθία λειτουργιών χαμηλού επιπέδου που μπορούν να εκτελεστούν από τη μονάδα εκτέλεσης.

Από την άλλη πλευρά, ένα από τα μειονεκτήματα της χρήσης μικροκώδικα είναι ότι μπορεί να επιβραδύνει το χρόνο επεξεργασίας, δεδομένου ότι η μονάδα ελέγχου πρέπει να ανακτήσει και να αποκωδικοποιήσει τις εντολές μικροκώδικα πριν η μονάδα εκτέλεσης εκτελέσει τη λειτουργία που καθορίζεται από την εντολή υψηλού επιπέδου.

[1] Πηγή: https://byjus.com/gate/microprogrammed-control-unit-notes/

Σήμερα, οι περισσότερες CPU υλοποιούν μια σκληρά συνδεδεμένη μονάδα ελέγχου που καθορίζει τη λειτουργία τους. Έτσι, η αρχιτεκτονική τους δεν βασίζεται σε μικροκώδικα αλλά σε λογικές πύλες και flip-flop.

1.2.7 Σχεδίαση της μονάδας ελέγχου με μικροπρογραμματιζόμενη λογική

Η μονάδα ελέγχου είναι υπεύθυνη για το συντονισμό των δραστηριοτήτων της CPU, συμπεριλαμβανομένης της ανάκτησης εντολών από τη μνήμη, της αποκωδικοποίησής τους και του ελέγχου της εκτέλεσης αυτών των εντολών.

Σε μια μικροπρογραμματισμένη σχεδίαση, η μονάδα ελέγχου αποτελείται από μια προγραμματιζόμενη μνήμη ελέγχου και ένα σύνολο σημάτων ελέγχου που παράγονται με βάση τα περιεχόμενα της μνήμης ελέγχου.

Η μνήμη ελέγχου είναι μια μνήμη ROM (μνήμη μόνο για ανάγνωση) που αποθηκεύει μια σειρά μικροεντολών, καθεμία από τις οποίες καθορίζει μια ακολουθία σημάτων ελέγχου που πρέπει να παραχθούν.

Οι μικροεντολές εκτελούνται διαδοχικά, με τα σήματα ελέγχου που παράγονται σε κάθε βήμα να χρησιμοποιούνται για τον έλεγχο της λειτουργίας της CPU.

Για να υλοποιήσουμε τη μονάδα ελέγχου της απλής μας CPU χρησιμοποιώντας μικροπρογραμματισμένη λογική, θα ορίζαμε πρώτα το σύνολο των σημάτων ελέγχου που απαιτούνται για την εκτέλεση κάθε εντολής και την ακολουθία των σημάτων ελέγχου που απαιτούνται για κάθε βήμα του κύκλου ανάκτησης-κωδικοποίησης-εκτέλεσης.

Στη συνέχεια θα κωδικοποιούσαμε αυτές τις ακολουθίες ως μικροεντολές στη μνήμη ελέγχου, με κάθε μικροεντολή να καθορίζει τις τιμές που πρέπει να εξάγονται στα σήματα ελέγχου σε κάθε βήμα.

Για παράδειγμα, για την εκτέλεση μιας εντολής LOAD, η μονάδα ελέγχου θα παρήγαγε την ακόλουθη ακολουθία σημάτων ελέγχου:

- Λήψη της εντολής από τη μνήμη και αποθήκευση στην IR.
- Αποκωδικοποίηση του opcode και των τελεστών της εντολής.
- Ανάγνωση του καθορισμένου καταχωρητή από το αρχείο καταχωρητών.
- Ανάγνωση της καθορισμένης διεύθυνσης μνήμης από τη μνήμη.

- Αποθήκευση της τιμής που διαβάστηκε από τη μνήμη στον καθορισμένο καταχωρητή.
- Ενημέρωση του PC ώστε να δείχνει την επόμενη εντολή.

Αυτά τα βήματα θα μπορούσαν να κωδικοποιηθούν ως μία ενιαία μικροεντολή στη μνήμη ελέγχου, με τα σήματα ελέγχου για κάθε βήμα να καθορίζονται ως διανύσματα bit.

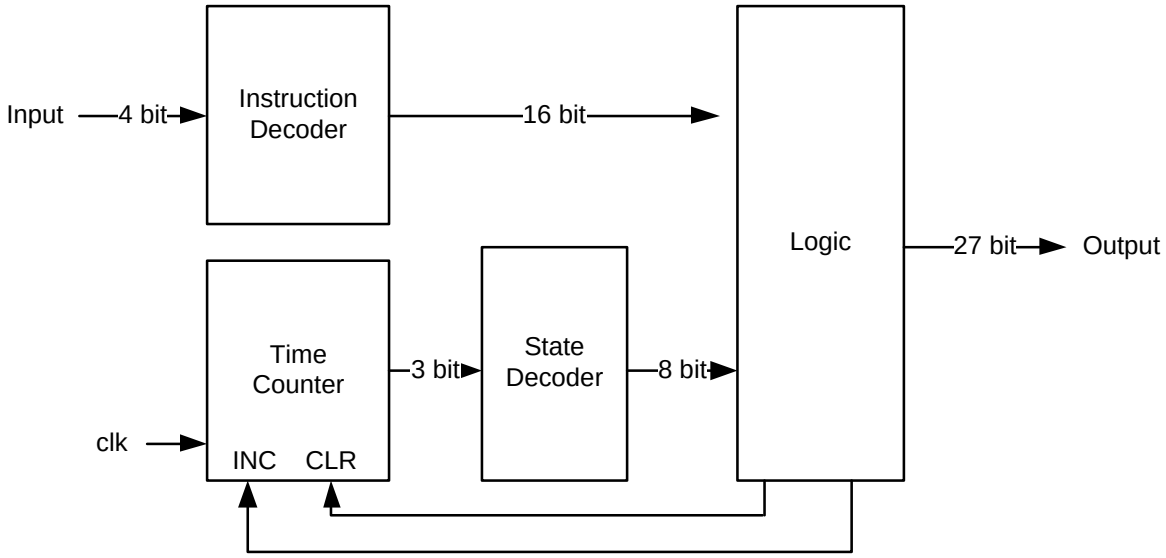
Για την εκτέλεση μιας συγκεκριμένης εντολής, η μονάδα ελέγχου θα έπαιρνε την κατάλληλη μικροεντολή από τη μνήμη ελέγχου και θα έβγαζε τα σήματα ελέγχου που καθορίζονται από τη μικροεντολή.

Τα σήματα ελέγχου θα χρησιμοποιούνταν στη συνέχεια για τον έλεγχο της λειτουργίας της ΚΜΕ, με την ALU, το αρχείο καταχωρητών και άλλα στοιχεία της CPU να ανταποκρίνονται στα σήματα ελέγχου όπως απαιτείται για την εκτέλεση της επιθυμητής λειτουργίας.

1.2.8 Σχεδίαση της μονάδας ελέγχου με hardwired λογική

Στην αρχιτεκτονική των υπολογιστών, η μονάδα ελέγχου είναι υπεύθυνη για τον έλεγχο της ροής δεδομένων και εντολών μεταξύ των διαφόρων στοιχείων της CPU. Σε μια σhardwired μονάδα ελέγχου, η λογική ελέγχου (control logic) υλοποιείται με τη χρήση ενός συνδυασμού πυλών, flip-flops και άλλων ψηφιακών στοιχείων.

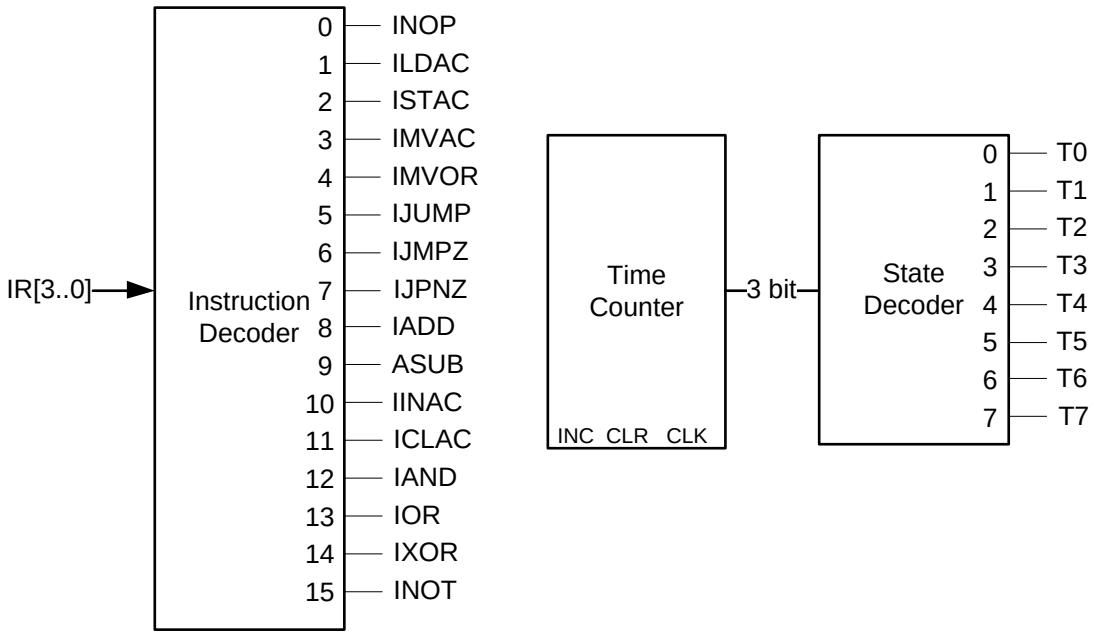
Ο σχεδιασμός μιας hardwired μονάδας ελέγχου περιλαμβάνει την ανάλυση του συνόλου εντολών και τον προσδιορισμό των σημάτων ελέγχου που απαιτούνται για την εκτέλεση κάθε εντολής. Κάθε εντολή τυπικά απαιτεί μια μοναδική ακολουθία σημάτων ελέγχου που πρέπει να παραχθεί προκειμένου να εκτελεστεί η απαιτούμενη λειτουργία.



Γενική διάταξη του τύπου της μονάδας ελέγχου

Η μονάδα ελέγχου υλοποιείται συνήθως ως μηχανή πεπερασμένων καταστάσεων, με κάθε κατάσταση να αντιπροσωπεύει ένα συγκεκριμένο στάδιο εκτέλεσης μιας εντολής. Τα σήματα ελέγχου για κάθε εντολή παράγονται από ένα συνδυασμό συνδυαστικής λογικής (π.χ. πύλες) και ακολουθιακής λογικής (π.χ. flip-flops) που είναι διαμορφωμένα έτσι ώστε να παράγουν την απαιτούμενη ακολουθία σημάτων ελέγχου για κάθε εντολή.

Το πλεονέκτημα μιας hardwired μονάδας ελέγχου είναι ότι μπορεί να σχεδιαστεί έτσι ώστε να είναι πολύ γρήγορη και αποδοτική, δεδομένου ότι τα σήματα ελέγχου παράγονται απευθείας από το σύνολο εντολών και δεν υπάρχει ανάγκη άντλησης εντολών ελέγχου από τη μνήμη. Ωστόσο, η σχεδίαση μιας σκληρά συνδεδεμένης μονάδας ελέγχου μπορεί να είναι πολύπλοκη, δεδομένου ότι κάθε εντολή απαιτεί μια μοναδική ακολουθία σημάτων ελέγχου και η σχεδίαση πρέπει να είναι σε θέση να χειριστεί όλους τους πιθανούς συνδυασμούς εντολών.



Block διάγραμμα της hardwired μονάδας ελέγχου

2 Υλοποίηση του κώδικα της σχετικά απλής CPU.

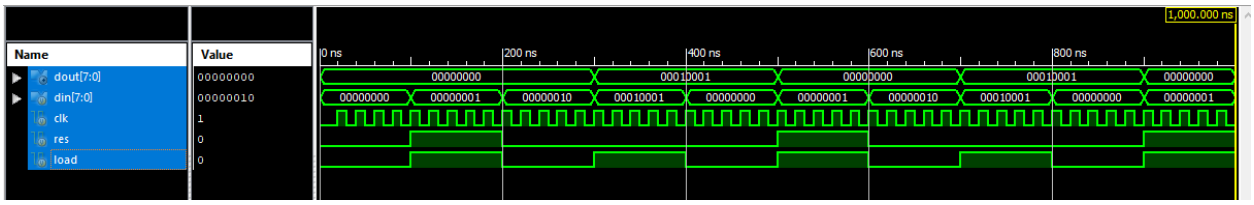
2.1 Οι καταχωρητές τύπου A και B.

Πρόκειται για καταχωρητές γενικής χρήσης που μπορούν να χρησιμοποιηθούν για την αποθήκευση δεδομένων που επεξεργάζεται ο επεξεργαστής.

Χρησιμοποιούνται συχνά για την αποθήκευση τελεστών που χρησιμοποιούνται ως είσοδος σε μια εντολή ή ως προσωρινή αποθήκευση ενδιάμεσων αποτελεσμάτων.

2.2 καταχωρητής τύπου A

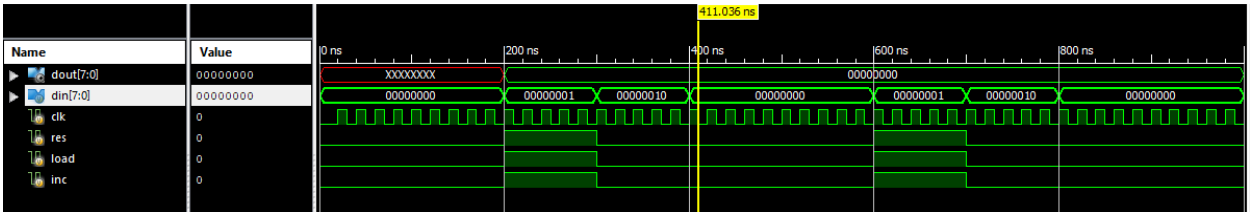
```
module rega (
input wire [n - 1:0] din,
input wire clk,
input wire res,
input wire load,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;

reg [n - 1:0] temp=0;

always @(posedge clk, posedge res) begin
    if (res == 1'b1) begin
       temp <= {(((n - 1))-((0)) +1) {1'b0}};
    end else begin
    if (load == 1'b1) begin
       temp <= din;
    end
    end
end

assign dout = temp;
endmodule
```



Σχ.1 Κυματομορφές εξομοίωσης για καταχωρητή τύπου A (rega)

2.3 Ο καταχωρητής τύπου B

```
module regb (
input wire [n - 1:0] din,
input wire clk,
input wire res,
input wire load,
input wire inc,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;



reg [n - 1:0] temp;

always @(posedge clk, posedge res) begin
    if (res == 1'b1) begin
temp <= {(((n - 1))-((0))+1){1'b0}};
end else begin
      if (load == 1'b1) begin
temp <= din;
end
else if (inc == 1'b1) begin
temp <= temp + 1;
end
end
end

assign dout = temp;
endmodule
```



Σχ. Κυματομορφές εξομοίωσης καταχωρητή τύπου B (regb)

2.4 Ο καταχωρητής σημαίας (Z).

Ένας καταχωρητής σημαιών σε μια CPU είναι ένας καταχωρητής ειδικού σκοπού που περιέχει ένα σύνολο σημαιών ή bits κατάστασης που υποδεικνύουν την κατάσταση διαφόρων συνθηκών.

Οι σημαίες ή τα bits κατάστασης σε έναν καταχωρητή σημαίας συνήθως τίθενται ή διαγράφονται από την KME κατά την εκτέλεση εντολών και μπορούν να χρησιμοποιηθούν για τον έλεγχο της ροής της εκτέλεσης ή για την ένδειξη της κατάστασης του συστήματος.

Ακολουθούν μερικά παραδείγματα των τύπων συνθηκών που μπορεί να υποδεικνύει ένας καταχωρητής σημαιών:

- Αριθμητικές σημαίες: Όπως η σημαία μεταφοράς (CY), η σημαία υπερχείλισης (OF) που υποδεικνύει το αποτέλεσμα μιας αριθμητικής πράξης.
- Σημαίες σύγκρισης: Όπως σημαία μηδενισμού (ZF), σημαία προσήμου (SF) που υποδεικνύει το αποτέλεσμα της σύγκρισης μεταξύ δύο αριθμών.
- Σημαίες ελέγχου: όπως σημαία ενεργοποίησης διακοπής (IE), σημαία επιπέδου προνομίων (PL) που υποδεικνύει την κατάσταση του συστήματος.

Ένας από τους κύριους σκοπούς αυτών των σημαιών είναι να επιτρέπουν στην KME να αναλαμβάνει διαφορετικές ενέργειες ανάλογα με το αποτέλεσμα μιας λειτουργίας.

Για παράδειγμα, μια εντολή διακλάδωσης που μπορεί να χρησιμοποιηθεί για άλμα σε διαφορετική θέση στον κώδικα με βάση την κατάσταση μιας σημαίας.

Όταν μια CPU εκτελεί μια αριθμητική πράξη, θέτει τις σημαίες στον καταχωρητή σημαιών για να υποδείξει το αποτέλεσμα της πράξης.

Για παράδειγμα, εάν μια αριθμητική πράξη παράγει μια μεταφορά, η σημαία μεταφοράς (CY) τίθεται.

Αυτό μπορεί να χρησιμοποιηθεί για τον έλεγχο υπερχείλισης και άλλων σφαλμάτων στην επόμενη εκτέλεση εντολής.

Μια άλλη χρήση του καταχωρητή σημαιών είναι ο έλεγχος της κατάστασης του συστήματος, όπως το αν το σύστημα βρίσκεται σε προνομιακή λειτουργία, αν είναι ενεργοποιημένες οι διακοπές ή όχι, κ.ο.κ.

Είναι επίσης σύνηθες να χρησιμοποιούνται οι σημαίες στον καταχωρητή σημαιών για τον έλεγχο της ροής της εκτέλεσης.

Για παράδειγμα, μια εντολή διακλάδωσης μπορεί να χρησιμοποιηθεί για άλμα σε διαφορετική θέση στον κώδικα με βάση την κατάσταση μιας σημαίας.

Αυτό επιτρέπει στην ΚΜΕ να λαμβάνει αποφάσεις με βάση το αποτέλεσμα μιας λειτουργίας και να χειρίζεται σφάλματα και άλλες εξαιρετικές συνθήκες με πιο ευέλικτο τρόπο.

Ο καταχωρητής σημαίας συνδέεται συνήθως με την αριθμητική και λογική μονάδα (ALU) σε μια CPU.

Η ALU είναι υπεύθυνη για την εκτέλεση αριθμητικών και λογικών πράξεων σε δεδομένα και παράγει αποτελέσματα και θέτει σημαίες στον καταχωρητή σημαίας για να υποδείξει το αποτέλεσμα των πράξεων.

Για παράδειγμα, όταν η ALU εκτελεί μια αριθμητική πράξη, όπως η πρόσθεση, θέτει τη σημαία μεταφοράς (CY) στον καταχωρητή σημαίας εάν υπάρχει μεταφορά από το πιο σημαντικό bit. Ομοίως, θέτει τη σημαία υπερχείλισης (OF) στον καταχωρητή σημαίας εάν υπάρχει υπερχείλιση από το αποτέλεσμα της πράξης.

Επιπλέον, η ALU θέτει και άλλες σημαίες, όπως τη σημαία μηδέν (ZF) εάν το αποτέλεσμα της πράξης είναι μηδέν και τη σημαία προσήμου (SF) εάν το πιο σημαντικό bit είναι ρυθμισμένο υποδεικνύοντας ότι ο αριθμός είναι αρνητικός.

Οι σημαίες στον καταχωρητή σημαιών χρησιμοποιούνται από τη μονάδα ελέγχου της CPU για τον προσδιορισμό της επόμενης εντολής προς εκτέλεση.

Για παράδειγμα, εάν η σημαία μηδέν είναι ρυθμισμένη, τότε η μονάδα ελέγχου μπορεί να εκτελέσει μια εντολή διακλάδωσης που παρακάμπτει ένα συγκεκριμένο μπλοκ κώδικα ή μια εντολή άλματος υπό συνθήκη με βάση την τιμή της σημαίας μηδέν, αυτό επιτρέπει στην CPU να λαμβάνει αποφάσεις με βάση το αποτέλεσμα μιας λειτουργίας και να χειρίζεται σφάλματα και άλλες εξαιρετικές συνθήκες με πιο ευέλικτο τρόπο.

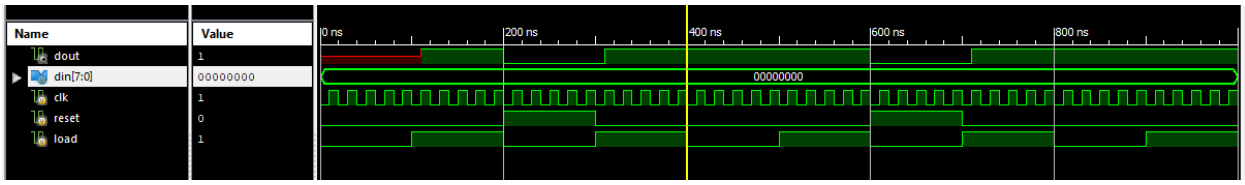
Είναι επίσης σημαντικό να σημειωθεί ότι ενώ ο καταχωρητής σημαιών συνδέεται συχνά με την ALU, ανάλογα με την αρχιτεκτονική, ορισμένες σημαίες μπορεί να παράγονται από άλλες μονάδες και στοιχεία της CPU.

2.5 Ο καταχωρητής σημαίας

```
module regz (
input wire [n - 1:0] din,
input wire clk,
input wire reset,
input wire load,
output reg dout
);

parameter [31:0] n= 8;



always @(posedge clk, posedge reset) begin
    if (reset == 1'b1) begin
      dout <= 1'b0;
end else begin
      if (load == 1'b1) begin
        if (din <= 2'b00) begin
          dout <= 1'b1;
end
else begin
          dout <= 1'b0;
end
end
end
end
endmodule
```

Αυτός ο κώδικας ορίζει μια ενότητα καταχωρητών με μια παράμετρο "n" για τον καθορισμό του μεγέθους των δεδομένων εισόδου.
Η μονάδα έχει μια είσοδο "din" που είναι τα δεδομένα που πρέπει να αποθηκευτούν στον καταχωρητή, μια είσοδο ρολογιού "clk" για να ελέγχει πότε αποθηκεύονται τα δεδομένα, μια είσοδο επαναφοράς "reset" για να επαναφέρει τον καταχωρητή στο 0 και μια είσοδο φόρτωσης "load" για να φορτώσει τα δεδομένα εισόδου στον καταχωρητή.
Η έξοδος είναι ένας καταχωρητής "dout" που περιέχει τα αποθηκευμένα δεδομένα.

Το μπλοκ πάντα ενεργοποιείται από μια θετική ακμή είτε του σήματος ρολογιού είτε του σήματος επαναφοράς.
Εάν η είσοδος reset είναι υψηλή, το dout τίθεται στο 0. Εάν η reset δεν είναι υψηλή και η είσοδος load είναι υψηλή, τα δεδομένα εισόδου "din" αξιολογούνται.
Εάν τα δεδομένα εισόδου είναι μικρότερα ή ίσα με 2'b00, το dout τίθεται σε 1, διαφορετικά τίθεται σε 0. Εάν η είσοδος φορτίου δεν είναι υψηλή, η έξοδος δεν αλλάζει.

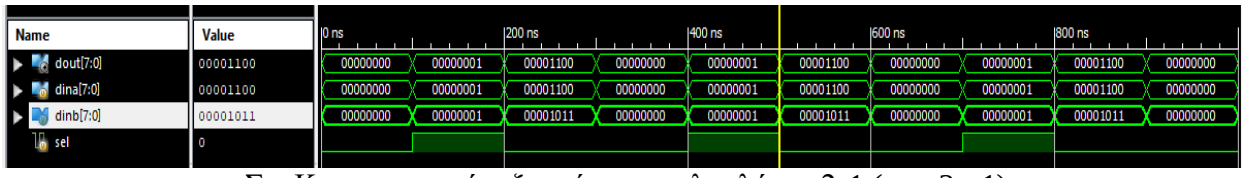


Σχ. Κυματομορφές εξομοίωσης καταχωρητή σημαίας Z (regz)

2.6 Οι πολυπλέκτες της ALU

2.6.1 Πολυπλέκτης 2 σε 1

```
module mux2to1(
input wire [n - 1:0] dina,
input wire [n - 1:0] dinb,
input wire sel,
output wire [n - 1:0] dout
);
parameter [31:0] n= 8;
assign dout = sel == 1'b0? dina: dinb;
endmodule
```



Σχ. Κυματομορφές εξομοίωσης πολυπλέκτη 2-1 (mux2to1)

Πρόκειται για έναν κώδικα Verilog που υλοποιεί έναν πολυπλέκτη 2 προς 1 (mux).
Ένας πολυπλέκτης είναι ένα ψηφιακό κύκλωμα που επιλέγει ένα από πολλά σήματα εισόδου και προωθεί το επιλεγμένο σήμα στην έξοδό του.

Ακολουθεί η ανάλυση του κώδικα:

- Η μονάδα ονομάζεται "mux2to1" και διαθέτει τέσσερις θύρες εισόδου: "dina", "dinb", "sel", και μία θύρα εξόδου "dout".

- Η παράμετρος n έχει οριστεί σε 8, που σημαίνει ότι το πλάτος των σημάτων εισόδου και εξόδου είναι 8 bit, το πλάτος των θυρών εισόδου και εξόδου ορίζεται από την παράμετρο n-1:0.

- Η εντολή assign αναθέτει την τιμή της dout με βάση την τιμή του σήματος επιλογής sel και την τιμή των εισόδων dina, dinb .

- Η τιμή του sel χρησιμοποιείται ως σήμα ελέγχου για τον πολυπλέκτη,ο τεταρτοβάθμιος τελεστής (?:) ελέγχει την τιμή του sel,εάν το sel είναι ίσο με 0, στην έξοδο dout αποδίδεται η τιμή του dina,διαφορετικά του αποδίδεται η τιμή του dinb.

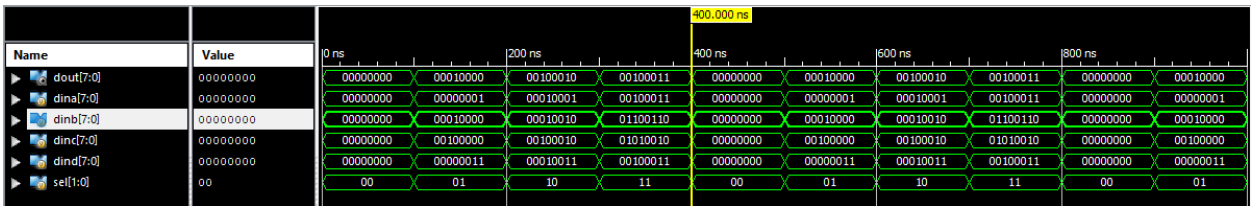
Έτσι αυτό το κύκλωμα μας επιτρέπει να επιλέξουμε μεταξύ δύο εισόδων dina και dinb με βάση την τιμή του sel.
Και να προωθεί το επιλεγμένο σήμα στη θύρα dout.
Το κύκλωμα παραμετροποιείται από τον αριθμό των bits, ώστε να μπορείτε εύκολα να το αλλάξετε ανάλογα με τις ανάγκες σας.

Είναι μια απλή υλοποίηση, αλλά μπορεί να είναι χρήσιμη όταν χρησιμοποιείται σε μια μεγαλύτερη σχεδίαση ως δομικό στοιχείο.
Επίσης, αξίζει να σημειωθεί ότι αυτή η υλοποίηση είναι συνδυαστική και δεν υπάρχει σήμα ρολογιού σε αυτήν, οπότε το σήμα dout θα αλλάζει ταυτόχρονα με την αλλαγή του sel.

2.6.2 Ο Πολυπλέκτης 4 σε 1

```
module mux4to1(
input wire [n - 1:0] dina,
input wire [n - 1:0] dinb,
input wire [n - 1:0] dinc,
input wire [n - 1:0] dind,
input wire [1:0] sel,
output reg [n - 1:0] dout
);

parameter [31:0] n= 8;


always @( *) begin
case (sel)
2'b00 : dout <= dina ;
2'b01 : dout <= dinb ;
2'b10 : dout <= dinc ;
      default : dout <= dina ;
    endcase
```

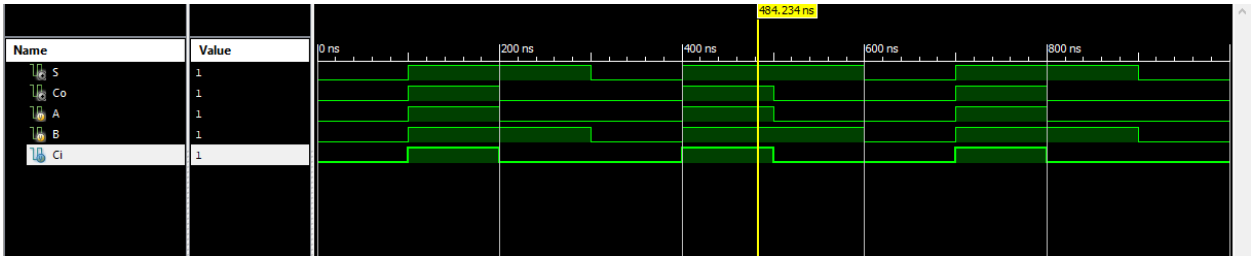
```
  end
endmodule
```



Σχ. Κυματομορφές εξομοίωσης πολυπλέκτη 4-1 (mux4to1)

2.7 Ο παράλληλος αθροιστής των 8-bit.

```
module FA (
input wire A,
input wire B,
input wire Ci,
output wire S,
output wire Co
);


assign S = A ^ B ^ Ci;
assign Co = (A & B) | (Ci & A) | (Ci & B);

endmodule
```



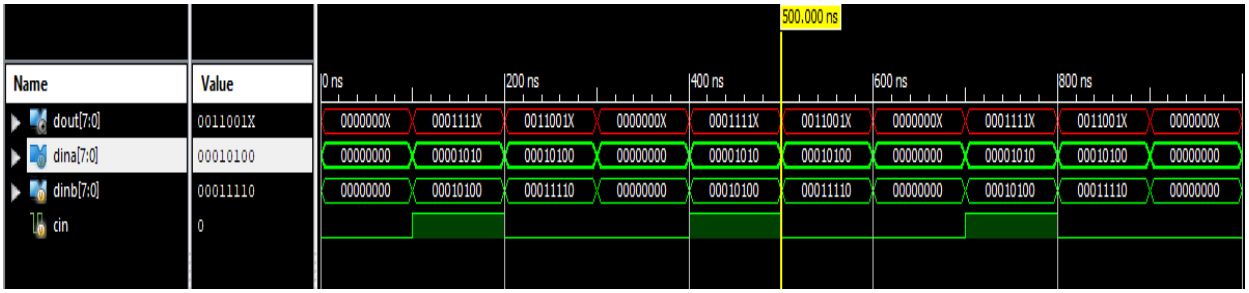
Σχ. Κυματομορφές εξομοίωσης παράλληλου αθροιστή (FA)

Αυτός είναι ένας κώδικας Verilog για μια μονάδα πλήρους αθροιστή (FA). Ο πλήρης αθροιστής είναι ένα ψηφιακό λογικό κύκλωμα που εκτελεί την αριθμητική πράξη της πρόσθεσης, συγκεκριμένα προσθέτει τρεις δυαδικούς αριθμούς ενός bit (A, B και Ci) και εξάγει δύο δυαδικούς αριθμούς ενός bit, S και Co.

Η πρώτη γραμμή του κώδικα, "module FA (καλώδιο εισόδου A, καλώδιο εισόδου B, καλώδιο εισόδου Ci, καλώδιο εξόδου S, καλώδιο εξόδου Co);", ορίζει το module και τα καλώδια εισόδου και εξόδου του. Τα ονόματα των καλωδίων εισόδου είναι A, B και Ci και τα ονόματα των καλωδίων εξόδου είναι S και Co.

Οι επόμενες δύο γραμμές του κώδικα, "assign S = A ^ B ^ Ci;" και "assign Co = (A & B) | (Ci & A) | (Ci & B);", χρησιμοποιούν τη λέξη κλειδί "assign" για να ορίσουν τις λογικές πράξεις που παράγουν την έξοδο του πλήρους αθροιστή. Η πρώτη γραμμή υπολογίζει το άθροισμα των τριών εισόδων χρησιμοποιώντας μια πράξη αποκλειστικού ή (^), ενώ η δεύτερη γραμμή υπολογίζει την έξοδο (Co) χρησιμοποιώντας έναν συνδυασμό των πράξεων and (&) και or (|).

```
module adder8bit(
input wire [7:0] dina,
input wire [7:0] dinb,
input wire cin,
output wire [7:0] dout
);


wire [7:0] C;

FA FA_1(
      dina [ 0],
    dinb [ 0],
    cyn,
    dout [ 0],
    C [ 0]);

  genvar i;
generate for (i =1; i <= 7; i = i + 1)
begin: p1
FA FA_2(
          dina [ i ],
      dinb [ i ],
      C [ i - 1],
      dout [ i ],
C [ i ] );

end
  endgenerate

endmodule
```



Σχ. Κυματομορφές εξομοίωσης αθροιστή 8-bit (Adder8Bit)

Αυτός είναι ένας κώδικας Verilog για μια μονάδα αθροιστή 8-bit, η οποία κατασκευάζεται χρησιμοποιώντας πολλαπλούς πλήρεις αθροιστές 1-bit. Έχει σχεδιαστεί για να προσθέτει δύο δυαδικούς αριθμούς 8-bit, dina και dinb, και ένα πρόσθετο carry-in bit, cin. Η έξοδος της πρόσθεσης είναι ένα άθροισμα 8 bit, dout.

Η πρώτη γραμμή "module adder8bit(input wire [7:0] dina, input wire [7:0] dinb, input wire cin, output wire [7:0] dout);" ορίζει τη μονάδα και τις εισόδους και εξόδους της. Τα ονόματα των καλωδίων εισόδου είναι dina, dinb και cin και το όνομα του καλωδίου εξόδου είναι dout. Τα καλώδια εισόδου dina και dinb έχουν πλάτος 8 bit, ενώ το cin έχει πλάτος 1 bit. Το dout είναι επίσης έξοδος πλάτους 8 bit.

Η επόμενη γραμμή του κώδικα, "wire [7:0] C;", ορίζει έναν πίνακα συρμάτων που ονομάζεται C και χρησιμοποιείται για την αποθήκευση των bits μεταφοράς από κάθε έναν από τους πλήρεις αθροιστές 1 bit.

Οι επόμενες γραμμές "FA FA_1(dina[0], dinb[0], cin, dout[0], C[0]);" ενσαρκώνουν τον πρώτο πλήρη αθροιστή 1 bit και συνδέουν τα αντίστοιχα καλώδια εισόδου και εξόδου. Ο πρώτος πλήρης αθροιστής 1 bit λαμβάνει τα λιγότερο σημαντικά bits (LSB) των dina, dinb και cin ως εισόδους και παράγει το LSB της εξόδου dout και το carry out C[0] ως εξόδους.

Στη συνέχεια, ένα μπλοκ παραγωγής χρησιμοποιείται για την ενσάρκωση πρόσθετων πλήρων αθροιστών 1 bit χρησιμοποιώντας έναν βρόχο for. Αυτός ο βρόχος επαναλαμβάνεται 7 φορές για να δημιουργήσει πλήρεις αθροιστές 1 bit που αριθμούνται από FA_2 έως FA_8. Κάθε ένας από αυτούς τους πλήρεις αθροιστές παίρνει ως είσοδο τα αντίστοιχα bit εισόδου από τα dina, dinb και το carry από την έξοδο του προηγούμενου πλήρους αθροιστή και παράγει το αντίστοιχο bit εξόδου του dout και το carry από το C[i] για κάθε επανάληψη του βρόχου.

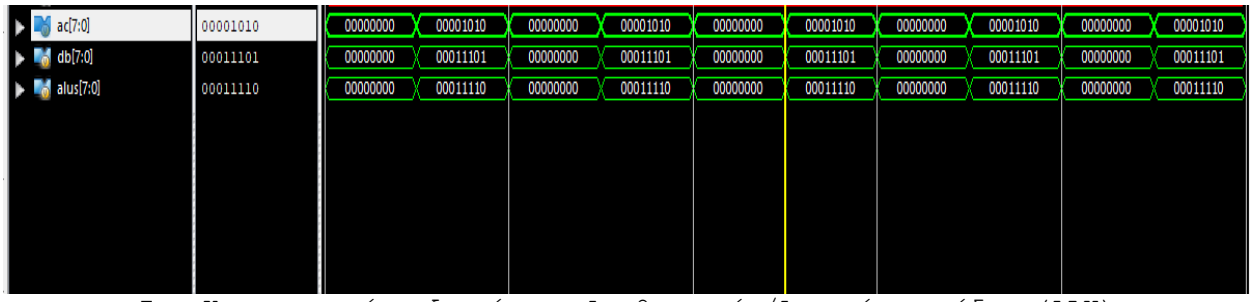
Οι γραμμές endgenerate και endmodule σηματοδοτούν το τέλος του μπλοκ παραγωγής και το τέλος του ορισμού της μονάδας αντίστοιχα.

Με αυτόν τον τρόπο, αυτή η μονάδα αθροιστή 8-bit είναι σε θέση να προσθέσει δύο δυαδικούς αριθμούς 8-bit, dina και dinb, και ένα πρόσθετο bit μεταφοράς, cin, συνδέοντας 8 περιπτώσεις πλήρους αθροιστή 1-bit και παράγοντας 8-bit έξοδο dout.

2.8 Συνολικό κύκλωμα της Αριθμητικής/Λογικής Μονάδας (ALU).

```
module alu (
input wire [n - 1 : 0] ac,
input wire [n - 1 : 0] db,
input wire [7 : 1] alus,
output wire [7 : 0] dout
);

parameter [31 : 0] n = 8;

wire [n - 1 : 0] f1;
wire [n - 1 : 0] f2;
wire [n - 1 : 0] f3;
wire [n - 1 : 0] f4;
wire [n - 1 : 0] zero;
wire [n - 1 : 0] notdb;
wire [n - 1 : 0] ANDout;
wire [n - 1 : 0] ORout;
wire [n - 1 : 0] XORout;
wire [n - 1 : 0] NOTout;

assign zero = {(((n - 1)) - ((0)) + 1){1'b0}};
assign ANDout = ac & db;
assign ORout = ac | db;
assign XORout = ac ^ db;
assign NOTout = ~ (ac);
assign notdb = ~ (db);

mux2to1 MUX_1(
ZERO,
AC,
Alus [ 1],
f1);

mux4to1 MUX_2(
ZERO,
db,
NOTDB,
ZERO,
Alus [ 2],
alus [ 3]);

mux4to1 MUX_3(
ANDout,
ORout,
XORout,
NOTout,
Alus [ 5],
Alus [ 6]);

adder8bit PADDR(
f1,
```

```
f2,
alus [ 4],
f3);

mux2to1 MUX_4(
f3,
f4,
alus [ 7],
dout);
endmodule
```



Σχ. Κυματομορφές εξομοίωσης Αριθμητικής/Λογικής μονάδας (ALU)

Αυτός είναι ένας κώδικας Verilog για μια μονάδα ALU (Arithmetic Logic Unit).

Σε αυτόν τον κώδικα, το πλάτος των εισόδων ac και db καθορίζεται από την παράμετρο "n", η οποία έχει οριστεί σε 8 bit.

Ο κώδικας ορίζει διάφορες συρμάτινες μεταβλητές f1, f2, f3, f4, zero, notdb, ANDout, ORout, XORout και NOTout, οι οποίες χρησιμοποιούνται για την αποθήκευση των ενδιάμεσων αποτελεσμάτων των διαφόρων πράξεων που εκτελούνται από την ALU.

Οι επόμενες γραμμές, "assign zero = {(((n - 1))-((0))+1){1'b0}};", "assign ANDout = ac & db ;", "assign ORout = ac | db ;" κ.λπ. χρησιμοποιούν τη λέξη κλειδί "assign" για να εκτελέσουν βασικές λογικές πράξεις bitwise στις εισόδους ac και db και να αποθηκεύσουν τα αποτελέσματα στις μεταβλητές wire.

Στη συνέχεια, χρησιμοποιούνται πολλαπλά muxes για την επιλογή ενός από τα ενδιάμεσα αποτελέσματα που θα περάσουν ως είσοδοι στο επόμενο βήμα λειτουργίας.

Αυτοί οι muxes ενσαρκώνονται ως "MUX_1", "MUX_2" ... "MUX_4" και παίρνουν την είσοδο alus για να επιλέξουν τη συγκεκριμένη είσοδο για την επόμενη λειτουργία.

Το "adder8bit PADDR(f1,f2,alus[4],f3)" ενσαρκώνει μια μονάδα αθροιστή 8 bit και χρησιμοποιεί τα f1, f2 ως είσοδο και το alus[4] ως carry-in bit, το f3 ως έξοδο.

Τέλος, το "mux2to1 MUX_4(f3,f4,alus[7],dout)" χρησιμοποιείται για να επιλέξει μία από τις δύο εξόδους ως τελικό αποτέλεσμα και το στέλνει ως dout.

Η μονάδα ALU μπορεί να εκτελέσει διαφορετικές λειτουργίες ανάλογα με την τιμή της εισόδου alus.

Λαμβάνει δύο εισόδους 8-bit ac, db και με βάση την είσοδο alus των 7-bit εκτελεί διάφορες πράξεις και στέλνει την έξοδο των 8-bit dout.

2.9 Κύκλωμα παραγωγής των σημάτων ελέγχου της ALU

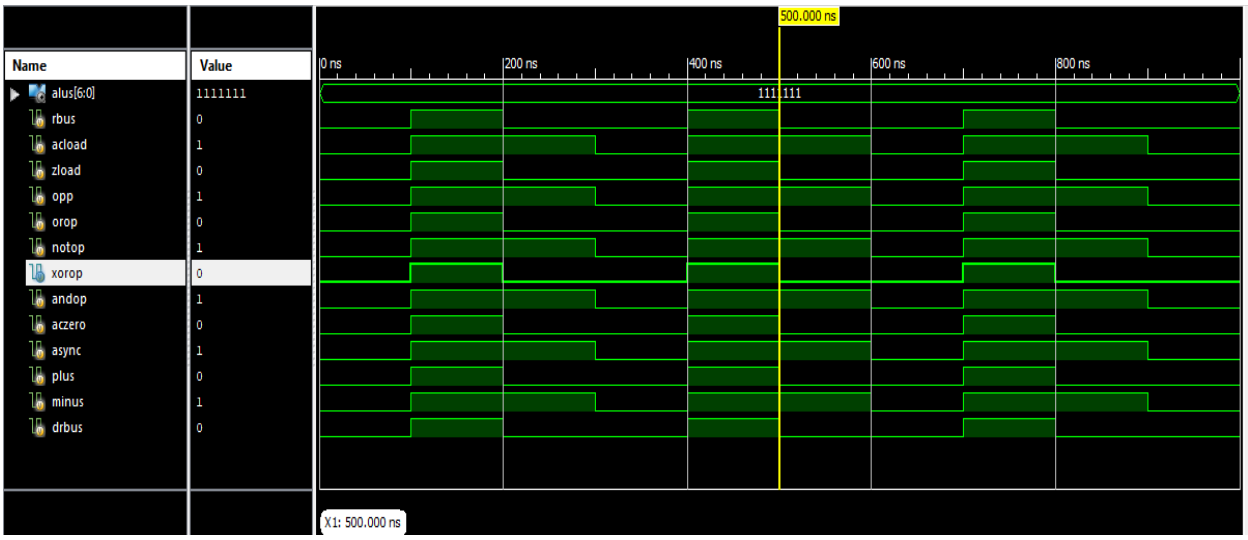
Αυτός είναι ένας κώδικας Verilog για μια μονάδα ελέγχου ALU, η οποία παράγει τα σήματα ελέγχου που χρησιμοποιούνται για τον έλεγχο της λειτουργίας μιας ALU.

Λαμβάνει πολλαπλά σήματα εισόδου όπως rbus, acload, zload, opp, orop, notop, xorop, andop, aczero, async, plus και minus και τα χρησιμοποιεί για να παράγει μια έξοδο 7-bit, alus.

```
module alus (
input wire rbus,
input wire acload,
input wire zload,
input wire opp,
input wire orop,
input wire notop,
input wire xorop,
input wire andop,
input wire aczero,
input wire async,
input wire plus,
input wire minus,
input wire drbus,
output reg [6:0] alus
);
```

```
wire [11:0] control;

assign control = { rbus,drbus
,acload,zload,andop,orop,notop,xorop,aczero,async,plus,minus};
always @(control) begin
case(control)
12'b101110000000 : begin
      alus <= 7'b1000000;
// AND
end
12'b 101101000000 : begin
```

```
      alus <= 7'b1100000;
// OR
end
12'b 001100100000 : begin
      alus <= 7'b1110000;
// NOT
end
12'b 101100010000 : begin
      alus <= 7'b1010000;
// XOR
end
12'b 001100001000 : begin
      alus <= 7'b0000000;
// CLAC
end
12'b 001100000100 : begin
      alus <= 7'b0001001;
// INAC
end
12'b 101100000010 : begin
      alus <= 7'b0000101;
// ADD
end
12'b 101100000001 : begin
      alus <= 7'b0001011;
// SUB
end
12'b 101100000000 : begin
      alus <= 7'b0000100;
// MOVR
end
12'b 011100000000 : begin
      alus <= 7'b0000100;
// LDAC5
end
    default : begin
      alus <= 7'b1111111;
// NO OPERATION
end
    endcase
end


endmodule
```



Σχ. Κυματομορφές εξομοίωσης ALUS

Τα σήματα εισόδου αντιπροσωπεύουν διάφορες επιλογές ελέγχου που μπορούν να περάσουν στη μονάδα ALU για να τη διαμορφώσουν ώστε να εκτελεί διάφορες λειτουργίες.

Για παράδειγμα, το "rbus" αντιπροσωπεύει την ανάγνωση από το δίαυλο, το "acload" αντιπροσωπεύει τη φόρτωση μιας τιμής στο συσσωρευτή, το "zload" αντιπροσωπεύει τη φόρτωση ενός μηδενός σε έναν καταχωρητή, το "opp" αντιπροσωπεύει τη λειτουργία που πρέπει να εκτελεστεί, το "orop" αντιπροσωπεύει την εκτέλεση της λειτουργίας OR, το "notop" αντιπροσωπεύει την εκτέλεση της λειτουργίας NOT, το "xorop" αντιπροσωπεύει την εκτέλεση της λειτουργίας XOR, το "andop" αντιπροσωπεύει την εκτέλεση της λειτουργίας AND, το "aczero" αντιπροσωπεύει τη μηδενισμό του συσσωρευτή, το "async" αντιπροσωπεύει την ασύγχρονη φόρτωση, το "plus" αντιπροσωπεύει τη λειτουργία πρόσθεσης και το "minus" αντιπροσωπεύει τη λειτουργία αφαίρεσης.

Ορίζεται μια συρμάτινη μεταβλητή "control" στην οποία αποδίδεται η τιμή όλων των σημάτων εισόδου. Για την αποκωδικοποίηση του σήματος ελέγχου εισόδου χρησιμοποιείται ένα μπλοκ always με λίστα ευαισθησίας στη μεταβλητή control και με βάση την τιμή του σήματος ελέγχου επιλέγεται η κατάλληλη περίπτωση και εκχωρείται η αντίστοιχη τιμή του alus.

Η δήλωση case ελέγχει την τιμή της μεταβλητής "control" και αναθέτει μια συγκεκριμένη τιμή 7 bit στην έξοδο "alus", με βάση τον συγκεκριμένο συνδυασμό σημάτων εισόδου.

Κάθε δήλωση περίπτωσης αντιστοιχεί σε μια διαφορετική λειτουργία που μπορεί να εκτελέσει η ALU.

Για παράδειγμα, όταν η τιμή της μεταβλητής "control" είναι "12'b101110000000", η ALU είναι ρυθμισμένη να εκτελεί μια πράξη AND και η έξοδος "alus" τίθεται στην τιμή "7'b1000000000".

Η προεπιλεγμένη περίπτωση είναι για το σενάριο όταν καμία από τις δηλώσεις περίπτωσης δεν ταιριάζει, οπότε η έξοδος "alus" ορίζεται σε "7'b1111111", η οποία θεωρείται ως NO-OPERATION.

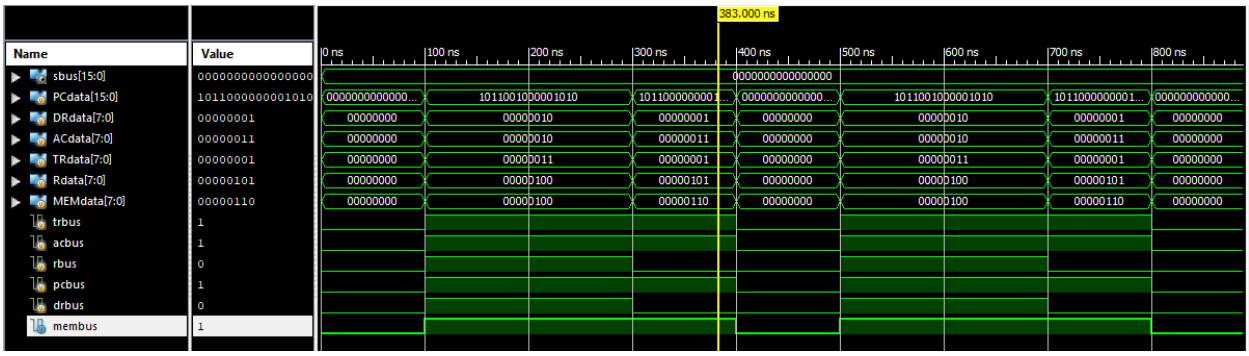
Με αυτόν τον τρόπο, αυτή η μονάδα ελέγχου ALU παράγει τα σήματα ελέγχου που χρησιμοποιούνται για τη διαμόρφωση της λειτουργίας μιας ALU με βάση τις τιμές των σημάτων ελέγχου εισόδου.

Το alus εξόδου των 7 bit διαβιβάζεται στην κύρια μονάδα ALU για τη διαμόρφωση της λειτουργίας της.

2.10 Ο δίαυλος δεδομένων.

```
module sys_bus (
inout wire [15:0] sbus,
input wire [15:0] PCdata,
input wire [7:0] DRdata,
input wire [7:0] ACdata,
input wire [7:0] TRdata,
input wire [7:0] Rdata,
input wire [7:0] MEMdata,
input wire trbus,
input wire acbus,
input wire rbus,
input wire pcbus,
input wire drbus,
input wire membus
);

wire [5:0] control;

assign control = {membus ,pcbus ,drbus ,trbus ,rbus ,acbus };
assign sbus = control == 6'b 100000 ?
{8'h 00,MEMdata } : control == 6'b010000 ?
PCdata : control == 6'b001000 ?
{DRdata,8'h00 } : control == 6'b000100 ?
{8'h 00,TRdata } : control == 6'b000010 ?
{Rdata,8'h00 } : control == 6'b000001 ?
```

```
{8'h 00,ACdata } : control == 6'b101000 ?
{ DRdata,MEMdata } : control == 6'b001100 ?
{ DRdata , TRdata } : {16{1'b 0}};

endmodule
```



Σχ. Κυματομορφές εξομοίωσης δίαυλου δεδομένων (sys_bus)

Πρόκειται για μια ενότητα γραμμένη σε Verilog, μια γλώσσα περιγραφής υλικού που χρησιμοποιείται για τη μοντελοποίηση ψηφιακών λογικών συστημάτων.

Η μονάδα, που ονομάζεται "sys_bus", διαθέτει ένα καλώδιο εισόδου 16 bit που ονομάζεται "sbus" και διάφορα καλώδια εισόδου για σήματα δεδομένων και ελέγχου.

Η μονάδα διαθέτει επίσης ένα καλώδιο με όνομα "control" που έχει πλάτος 6 bit.

Αυτό το καλώδιο ανατίθεται στη συνένωση των σημάτων "membus", "pcbus", "drbus", "trbus", "rbus" και "acbus".

Η τιμή αυτού του καλωδίου "control" καθορίζει ποια είσοδος δεδομένων θα περάσει στο καλώδιο εξόδου "sbus" μέσω μιας σειράς δηλώσεων υπό όρους.

Η εντολή assign που ακολουθεί τον ορισμό control είναι ουσιαστικά μια αλυσίδα τριμερών τελεστών όπου ελέγχει την τιμή του καλωδίου control και με βάση αυτή αναθέτει τιμή στο sbus.

Αυτή η ενότητα λειτουργεί ουσιαστικά ως πολυπλέκτης, επιλέγοντας μία από τις διάφορες εισόδους δεδομένων που θα περάσουν στο καλώδιο εξόδου "sbus" με βάση την τιμή του καλωδίου "control".

Όταν το καλώδιο "control" έχει την τιμή 6'b100000, η είσοδος "MEMdata" περνάει στο "sbus". Όταν το καλώδιο "control" έχει την τιμή 6'b010000, η είσοδος "PCdata" περνάει στο "sbus", και ούτω καθεξής για τις άλλες πιθανές τιμές του "control".

Σε περίπτωση που το control δεν ταιριάζει σε καμία από τις συνθήκες αναθέτει το '{16{1'b 0}}' (το οποίο οδηγεί σε δίαυλο πλάτους 16 bit με όλα τα 0)

Μοιάζει αρκετά με την εντολή switch, αλλά στη verilog, ονομάζεται casez.

3 ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

3.1 Η microprogrammed μονάδα ελέγχου

Η μονάδα μικροπρογραμματισμένου ελέγχου (MCU) είναι μια μονάδα σε μια κεντρική μονάδα επεξεργασίας (CPU) που ελέγχει τις λειτουργίες της CPU.

Στις μικροπρογραμματισμένες ΚΜΕ, η μονάδα ελέγχου υλοποιείται με τη χρήση μιας ακολουθίας μικροεντολών που αποθηκεύονται σε μια μνήμη μόνο για ανάγνωση (ROM) ή σε μια μνήμη ελέγχου.

Αυτές οι μικροεντολές καθορίζουν τις ενέργειες που πρέπει να γίνουν για κάθε εντολή που λαμβάνεται από την κύρια μνήμη και καθορίζουν τη ροή ελέγχου εντός της ΚΜΕ.

Μια μικροπρογραμματισμένη μονάδα ελέγχου λειτουργεί εκτελώντας διαδοχικά μια σειρά μικροεντολών, αντί να εκτελεί απευθείας εντολές.

Οι μικροεντολές καθορίζουν τα σήματα ελέγχου και τις λειτουργίες που εκτελεί η ΚΜΕ σε απάντηση σε κάθε εντολή.

Αυτό καθιστά τη σχεδίαση μιας μικροπρογραμματισμένης μονάδας ελέγχου πιο ευέλικτη, καθώς οι μικροεντολές μπορούν εύκολα να αλλάξουν για να τροποποιήσουν τη συμπεριφορά της ΚΜΕ.

Το κύριο πλεονέκτημα μιας μικροπρογραμματισμένης μονάδας ελέγχου είναι ότι επιτρέπει την εύκολη τροποποίηση της αρχιτεκτονικής συνόλου εντολών (ISA) της ΚΜΕ, καθώς οι μικροεντολές μπορούν εύκολα να τροποποιηθούν ώστε να αντικατοπτρίζουν τις αλλαγές στην ISA.

Καθιστά επίσης δυνατή την ενσωμάτωση νέων εντολών στην ISA της ΚΜΕ, καθώς οι μικροεντολές μπορούν να γραφούν για να υποστηρίξουν αυτές τις νέες εντολές.

3.2 Ο πολυπλέκτης 4-σε-1

```
module mux4to1(
input wire [n - 1:0] dina,
input wire [n - 1:0] dinb,
input wire [n - 1:0] dinc,
input wire [n - 1:0] dind,
input wire [1:0] sel,
output reg [n - 1:0] dout
);

parameter [31:0] n= 8;


always @( *) begin
case ( sel )
2'b00 : dout <= dina ;
2'b01 : dout <= dinb ;
2'b10 : dout <= dinc ;
      default : dout <= dina ;
    endcase
  end
endmodule
```

Αυτός είναι ο κώδικας Verilog για έναν πολυπλέκτη 4 προς 1 (mux4to1).

Η μονάδα δέχεται τέσσερις εισόδους δεδομένων (dina, dinb, dinc και dind), μια είσοδο επιλογής 2 bit (sel) και παράγει μια μοναδική έξοδο δεδομένων (dout).
Το μέγεθος των εισόδων και των εξόδων καθορίζεται από την παράμετρο "n", η οποία σε αυτή την περίπτωση έχει οριστεί σε 8.

Το μπλοκ always με τη λίστα ευαισθησίας "@(*)" καθορίζει ότι το μπλοκ θα πρέπει να εκτελείται κάθε φορά που αλλάζει κάποια από τις εισόδους του.
Σε αυτό το μπλοκ, χρησιμοποιείται μια δήλωση case για να καθοριστεί ποια είσοδος θα περάσει στην έξοδο με βάση την τιμή της εισόδου sel. Εάν η sel είναι 00, τότε η dout τίθεται στην dina. Εάν η sel είναι 01, τότε η dout τίθεται σε dinb, και ούτω καθεξής. Εάν το sel δεν είναι 00, 01 ή 10, τότε η προεπιλεγμένη περίπτωση θέτει το dout σε dina.

Σημειώστε ότι η έξοδος dout δηλώνεται ως reg, που σημαίνει ότι μπορεί να κρατάει μια τιμή και να ενημερώνεται με την πάροδο του χρόνου κατά την προσομοίωση.

3.3 Ο Καταχωρητής αποθήκευσης

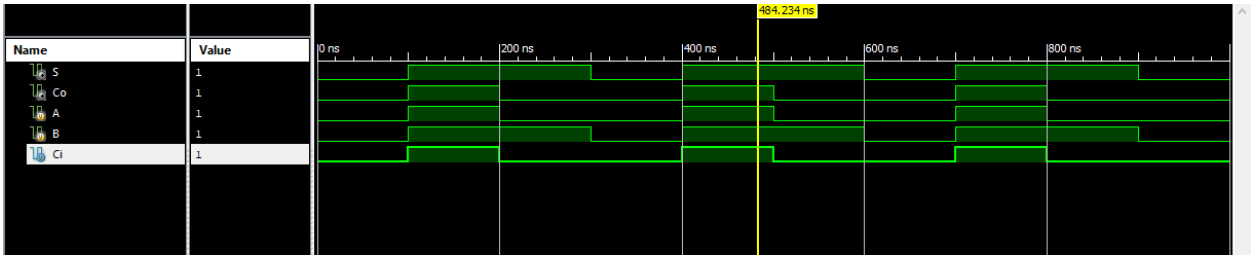
```
module FA (
input wire A,
input wire B,
input wire Ci,
output wire S,
output wire Co
);


assign S = A ^ B ^ Ci;
assign Co = (A & B) | (Ci & A) | (Ci & B );

endmodule
```

Αυτός είναι ο κώδικας Verilog για έναν πλήρη αθροιστή (FA).

Η μονάδα δέχεται τρεις εισόδους (A, B και Ci) και παράγει δύο εξόδους (S και Co).
Οι έξοδοι υπολογίζονται χρησιμοποιώντας συνδυαστική λογική εκφρασμένη ως αναθέσεις.

Η πρώτη εντολή ανάθεσης υπολογίζει το άθροισμα των εισόδων A, B και Ci και το αναθέτει στην έξοδο S.
Η δεύτερη εντολή ανάθεσης υπολογίζει τη μεταφορά από τον πλήρη αθροιστή, με βάση τις εισόδους A, B και Ci, και την αναθέτει στην έξοδο Co.

Σημειώστε ότι οι είσοδοι και οι έξοδοι δηλώνονται ως καλώδια, που σημαίνει ότι αντιπροσωπεύουν συνδέσεις με άλλες μονάδες ή σήματα στη σχεδίαση.
Τα σύρματα μπορούν να λάβουν μια τιμή μόνο μία φορά και χρησιμοποιούνται για τη μοντελοποίηση της συνδυαστικής λογικής.



Σχ. Κυματομορφές εξομοίωσης καταχωρητή αποθήκευσης (FA)

3.4 Η Λογική καθορισμού των σημάτων S1 και S0

```
module mseq_logic (
input wire bt,
input wire [1:0] cond,
input wire z,
output wire s1,
output wire s0
);

reg muxout ;
wire notz ;

assign notz = ~ (z);
always @( *) begin
case ( cond )
2'b 00 : muxout <= 1'b1;
2'b 01 : muxout <= z;
2'b 10 : muxout <= notz ;
      default : muxout <= 1'b0;
    endcase
end

assign s1 = bt ;
assign s0 = ~ ( muxout );

endmodule
```

Αυτός είναι ο κώδικας Verilog για μια μονάδα ακολουθιακής λογικής με το όνομα "mseq_logic".

Η μονάδα δέχεται τρεις εισόδους: "bt", "cond" και "z".
Το "bt" είναι μια δυαδική είσοδος, το "cond" είναι μια είσοδος 2 bit και το "z" είναι μια άλλη δυαδική είσοδος.
Η μονάδα παράγει επίσης δύο εξόδους "s1" και "s0".

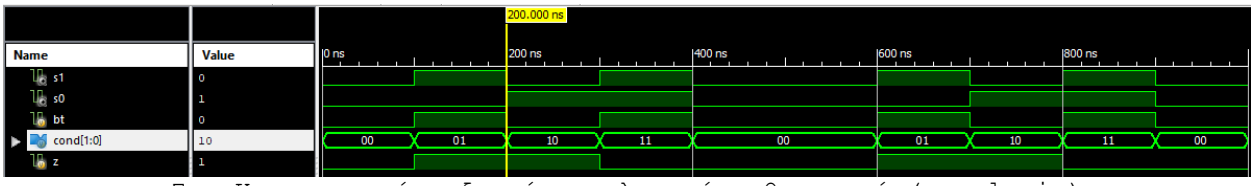
Η μονάδα χρησιμοποιεί ένα ενδιάμεσο σήμα "muxout", το οποίο δηλώνεται ως reg και χρησιμοποιείται για την αποθήκευση της εξόδου ενός πολυπλέκτη 4 προς 1.
Ο πολυπλέκτης υλοποιείται χρησιμοποιώντας μια δήλωση case σε ένα μπλοκ always με λίστα ευαισθησίας "@(*)", πράγμα που σημαίνει ότι θα εκτελείται κάθε φορά που αλλάζει κάποια από τις εισόδους του.
Ο πολυπλέκτης επιλέγει μία από τις εισόδους του με βάση την τιμή του "cond". Εάν η "cond" είναι 00, τότε η "muxout" τίθεται σε 1.

Εάν η "cond" είναι 01, τότε η "muxout" τίθεται σε "z". Εάν η "cond" είναι 10, τότε η "muxout" τίθεται στην άρνηση της "z". Εάν το "cond" δεν είναι 00, 01 ή 10, τότε η προεπιλεγμένη περίπτωση θέτει το "muxout" σε 0.

Το σήμα "notz" είναι ένα ενδιάμεσο σήμα που κρατά την άρνηση της εισόδου "z".

Στην έξοδο "s1" αποδίδεται η τιμή της εισόδου "bt". Στην έξοδο "s0" αποδίδεται η άρνηση του σήματος "muxout".

Σημειώστε ότι τα ενδιάμεσα σήματα "muxout" και "notz" δηλώνονται ως reg, δηλαδή μπορούν να κρατούν μια τιμή και να ενημερώνονται με την πάροδο του χρόνου κατά την προσομοίωση. Οι έξοδοι "s1" και "s0" δηλώνονται ως wires, που σημαίνει ότι αντιπροσωπεύουν συνδέσεις με άλλες μονάδες ή σήματα στη σχεδίαση και μπορούν να λάβουν τιμή μόνο μία φορά.



Σχ. Κυματομορφές εξομοίωσης λογική καθορισμού (seq_logic)

3.5 Η Μνήμη μικροκώδικα

```
module micro_ROM(
input wire [5 : 0] address,
input wire clock,
output wire [35 : 0] q
);
wire [35 : 0] sub_wire0;

assign q = sub_wire0[35 : 0];
altsyncram #(
clock_enable_input_a(6'bBYPASS),
clock_enable_output_a(6'bBYPASS),
init_file(16'brs_microcode.mif),
intended_device_family(10'BCYCLONE II),
lpm_hint(21'bENABLE_RUNTIME_MOD = NO),
lpm_type(10'baltsyncram),
numwords_a(64),
operation_mode(3'bROM),
outdata_aclr_a(4'bNONE),
outdata_reg_a(12'bUNREGISTERED),
widthad_a(6),
width_a(36),
width_byteena_a(1))
altsyncram_component(
```

```
clock0(clock),
address_a(address),
q_a(sub_wire0));
endmodule
```

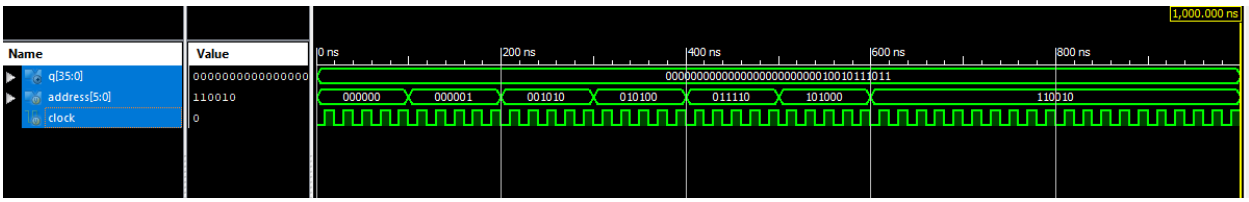
Πρόκειται για έναν κώδικα Verilog που υλοποιεί ένα micro-ROM. Η μονάδα έχει είσοδο διεύθυνσης 6-bit, είσοδο ρολογιού και έξοδο "q" 36-bit.

Η μονάδα χρησιμοποιεί μια περίπτωση του στοιχείου "altsyncram" με προκαθορισμένες παραμέτρους.
Το συστατικό είναι ένας τύπος RAM που υλοποιείται χρησιμοποιώντας μια συγκεκριμένη τεχνολογία, στην προκειμένη περίπτωση την οικογένεια συσκευών Cyclone II.

Το στοιχείο διαθέτει μια είσοδο διεύθυνσης 6 bit "address_a", μια είσοδο ρολογιού "clock0" και μια έξοδο 36 bit "q_a".
Η έξοδος του στοιχείου εκχωρείται στο ενδιάμεσο καλώδιο "sub_wire0".
Η τελική έξοδος του στοιχείου ανατίθεται στο "q", το οποίο είναι συνδεδεμένο στο "sub_wire0".

Το δομοστοιχείο αρχικοποιείται με την παράμετρο ".init_file" να έχει οριστεί σε "rs_microcode.mif", το οποίο είναι πιθανότατα ένα αρχείο αρχικοποίησης μνήμης.
Το εξάρτημα λειτουργεί σε κατάσταση λειτουργίας μόνο για ανάγνωση, όπως ορίζεται από την παράμετρο ".operation_mode" που έχει οριστεί σε "3'bROM".

Το micro-ROM χρονίζεται από την είσοδο "clock" και τα περιεχόμενά του μπορούν να διαβαστούν καθορίζοντας την επιθυμητή διεύθυνση στην είσοδο "address".
Τα περιεχόμενα των 36 bit στην καθορισμένη διεύθυνση θα είναι διαθέσιμα στην έξοδο "q".



Σχ.2 Κυματομορφές εξομοίωσης micro_ROM

3.6 O Microsequencer

```
module mseq (
input wire [3:0] ir,
input wire clock,
input wire reset,
input wire z,
output wire [35:0] code,
output wire [26:0] mOPs
);

wire [5:0] addr ;
wire [5:0] MUXout ; wire [5:0] REGout ;
reg [5:0] mux0; reg [5:0] mux1;
reg s1; reg s0;
wire [5:0] irmap ;
wire bt ;
wire [1:0] cond ; wire [1:0] control; wire [1:0] selmux ;
wire [35:0] microcode;
wire sel1; wire sel0;

assign irmap = {ir,2'b00 };
always @( REGout, addr , sel1, sel0, reset) begin
    if (reset == 1'b1) begin
s1 <= 1'b0;
s0 <= 1'b0;
mux0 <= 6'b000000;
mux1 <= 6'b000000;
end
else begin
s1 <= sel1;
s0 <= sel0;
mux0 <= REGout + 1;
mux1 <= addr ;
end
end

// MUXmap    : mseq_map port map (irmap, ir);
  mseq_logic MUXlogic (
      btw,
    cond,
z,
sel1,
sel0 );

mux4to1 #(
      .n (6))
  MUXproc (
mux0,
mux1,
    irmap,

s1,
```

```
s0,
    MUXout );

  rega #(
      .n (6))
  REGproc (
      MUXout,
clock,
reset,
1'b1,
    REGout );

  micro_ROM MEMproc (
      MUXout,
clock,
microcode );

  genvar j;
generate for (j=0; j <= 1; j = j + 1) begin: g1
assign cond [j] = microcode [ j + 34 ];
end
  endgenerate
assign bt = microcode [33];
  genvar m;
generate for (m=0;m<= 5; m=m+ 1) begin: g2
assign addr[m]= microcode[m];
end
  endgenerate
  genvar v;
generate for (v=0;v<= 26; v= v+ 1) begin: g3
assign mOPs [v]= microcode [ v+ 6 ];
end
  endgenerate
endmodule
```

Αυτός είναι ένας κώδικας Verilog για μια μονάδα που ονομάζεται "mseq" και υλοποιεί έναν μικροακολουθητή.
Ο μικροακολουθητής είναι ένα ακολουθιακό ψηφιακό κύκλωμα που χρησιμοποιείται για τον έλεγχο της λειτουργίας ενός μικροεπεξεργαστή.

Οι είσοδοι της μονάδας είναι οι εξής:

- ir: ένα καλώδιο 4 bit που αντιπροσωπεύει τον καταχωρητή εντολών του μικροεπεξεργαστή.
- clock: ένα καλώδιο ενός bit που αντιπροσωπεύει το σήμα ρολογιού.
- reset: καλώδιο ενός bit που αντιπροσωπεύει το σήμα επαναφοράς.

- z: ένα καλώδιο ενός bit που αντιπροσωπεύει μια είσοδο.

Οι έξοδοι της μονάδας είναι οι εξής:

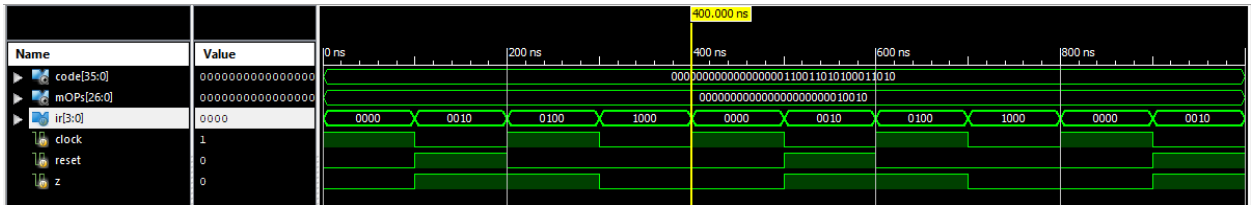
- code: ένα καλώδιο 36 bit που αντιπροσωπεύει την έξοδο της ROM μικροκώδικα.
- mOPs: ένα καλώδιο 27 bit που αντιπροσωπεύει τα σήματα μικρολειτουργίας.

Η μονάδα διαθέτει διάφορες υπομονάδες, μεταξύ των οποίων:

- mseq_logic: υπομονάδα που υλοποιεί τη λογική για την επιλογή της διεύθυνσης της ROM μικροκώδικα.
- altsyncram: υπομονάδα που υλοποιεί τον μικροκώδικα ROM.
- mux4to1: υπομονάδα που υλοποιεί πολυπλέκτη 4 προς 1.
- rega: υπομονάδα που υλοποιεί καταχωρητή 6 bit.

Η μονάδα υλοποιεί έναν σύνθετο μηχανισμό ελέγχου για έναν μικροεπεξεργαστή, όπου ο καταχωρητής εντολών (ir) χρησιμοποιείται για την πρόσβαση στον μικροκώδικα στη ROM μικροκώδικα και η έξοδος της ROM μικροκώδικα χρησιμοποιείται για τον έλεγχο των λειτουργιών του μικροεπεξεργαστή.
Η μονάδα υλοποιεί επίσης έναν πολυπλέκτη και έναν καταχωρητή για την αποθήκευση ενδιάμεσων αποτελεσμάτων.

Ο κώδικας παράγει τη διεύθυνση για τη ROM μικροκώδικα, τα σήματα ελέγχου για τον πολυπλέκτη και τον καταχωρητή και τα σήματα μικρολειτουργίας για τον μικροεπεξεργαστή.
Η τιμή της εξόδου "κωδικός" λαμβάνεται από τη ROM μικροκώδικα.
Η τιμή της εξόδου "mOPs" λαμβάνεται από την έξοδο "code" επιλέγοντας τα bits 6 έως 32.



Σχ. Κυματομορφές εξομοίωσης microsequencer (mseq)

4 Η Hardwired μονάδα ελέγχου

Η hardwired μονάδα ελέγχου είναι ένας τύπος που χρησιμοποιεί ενσύρματα ψηφιακά λογικά κυκλώματα για την υλοποίηση της λογικής ελέγχου της κεντρικής μονάδας επεξεργασίας (CPU) ενός υπολογιστή.

Χρησιμοποιεί σταθερές φυσικές συνδέσεις μεταξύ των εξαρτημάτων για την εκτέλεση των λειτουργιών που καθορίζονται από την αρχιτεκτονική συνόλου εντολών (ISA) της ΚΜΕ.

Η λογική ελέγχου είναι σταθερή και δεν μπορεί να αλλάξει ή να τροποποιηθεί.

Συγκριτικά, μια μικροπρογραμματισμένη μονάδα ελέγχου χρησιμοποιεί ένα μικροπρόγραμμα, το οποίο είναι ένα μικρό πρόγραμμα αποθηκευμένο σε μνήμη μόνο για ανάγνωση (ROM), για να υλοποιήσει τη λογική ελέγχου της ΚΜΕ.

Το μικροπρόγραμμα μπορεί να τροποποιηθεί για να αλλάξει η λογική ελέγχου της ΚΜΕ.

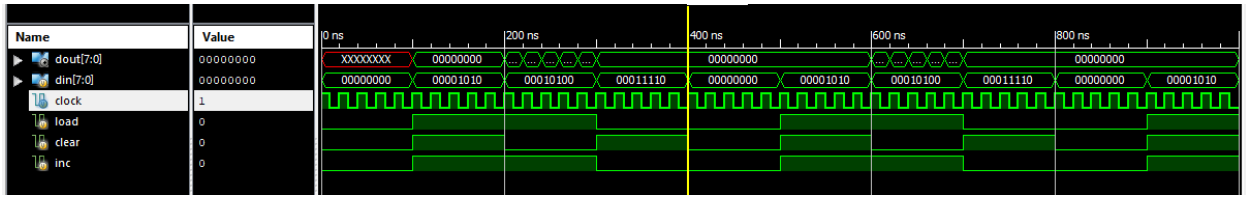
Οι μονάδες ελέγχου με μικροπρογραμματισμό είναι πιο ευέλικτες και ευκολότερα τροποποιήσιμες από τις hardwired, αλλά μπορεί επίσης να είναι πιο αργές.

4.1 Ο απαριθμητής κατάστασης

```
module count(
input wire [n - 1:0] din,
input wire clock,
input wire load,
input wire clear,
input wire inc,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;
reg [n-1:0] temp;
```

```
always @( posedge clock) begin : P1

    if (clear == 1'b1) begin
temp = {1{1'b0} };
end
else if (inc == 1'b1) begin
temp = temp + 1;
end
else if (load == 1'b1) begin
temp = din;
end
end

assign dout = temp;
endmodule
```

Ο κώδικας υλοποιεί έναν απλό ψηφιακό μετρητή.
Διαθέτει μια είσοδο din που μπορεί να φορτωθεί στον εσωτερικό καταχωρητή temp θέτοντας την είσοδο load στο 1'b1.
Ο μετρητής μπορεί να αυξηθεί θέτοντας την είσοδο inc σε 1'b1.
Η είσοδος clear μπορεί να χρησιμοποιηθεί για την εκκαθάριση του εσωτερικού καταχωρητή σε όλα τα μηδενικά.
Η τιμή του μετρητή εξάγεται μέσω της εξόδου dout.
Το μέγεθος του μετρητή καθορίζεται από την παράμετρο n.



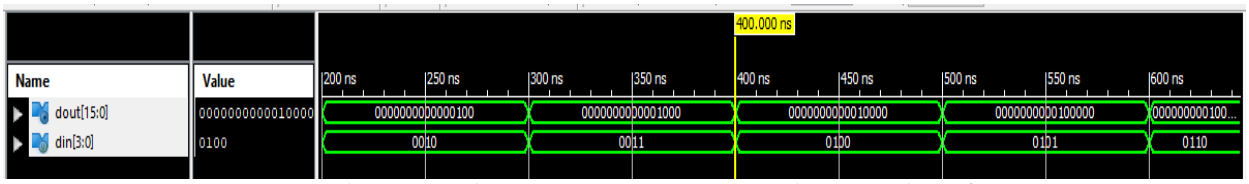
Σχ. Κυματομορφές εξομοίωσης απαριθμητή κατάστασης (COUNT)

4.2 Ο αποκωδικοποιητής εντολών

```
module ir_decoder (
input wire [3:0] din,
output wire [15:0] dout
);

assign dout = din == 8'h 00 ? 16'b 0000000000000001 : din == 8'h01 ? 16'b
0000000000000010 : din == 8'h02 ? 16'b 0000000000000100 : din == 8'h03 ? 16'b
0000000000001000 : din == 8'h04 ? 16'b 0000000000010000 : din == 8'h05 ? 16'b
0000000000100000 : din == 8'h06 ? 16'b 0000000001000000 : din == 8'h07 ? 16'b
0000000010000000 : din == 8'h08 ? 16'b 0000000100000000 : din == 8'h09 ? 16'b
```

```
0000001000000000 : din == 8'h0A ? 16'b 0000010000000000 : din == 8'h0B ? 16'b
0000100000000000 : din == 8'h0C ? 16'b 0001000000000000 : din == 8'h0D ? 16'b
0010000000000000 : din == 8'h0E ? 16'b 0100000000000000 : din == 8'h0F ? 16'b
1000000000000000 : {16{1'b0}};

endmodule
```

Πρόκειται για την υλοποίηση ενός αποκωδικοποιητή καταχωρητή εντολών (IR) σε Verilog.
Ο αποκωδικοποιητής IR λαμβάνει μια τιμή εισόδου 4 bit, din, και την αντιστοιχίζει σε μια έξοδο 16 bit, dout.
Η αντιστοίχιση της τιμής εισόδου στην έξοδο καθορίζεται χρησιμοποιώντας μια σειρά από δηλώσεις υπό όρους (; : τελεστής), όπου η τιμή εισόδου συγκρίνεται με κάθε μια από τις πιθανές τιμές (0x00, 0x01, 0x02, κ.λπ.) και η αντίστοιχη τιμή εξόδου εκχωρείται στο dout.
Εάν η τιμή εισόδου δεν ταιριάζει με καμία από τις καθορισμένες περιπτώσεις, τότε στο dout εκχωρείται μια προεπιλεγμένη τιμή 16 μηδενικών.
Αυτός ο αποκωδικοποιητής χρησιμοποιείται συνήθως στη μονάδα ελέγχου ενός επεξεργαστή για τον προσδιορισμό της επόμενης λειτουργίας που πρέπει να εκτελεστεί με βάση την τρέχουσα εντολή που είναι αποθηκευμένη στον καταχωρητή εντολών.

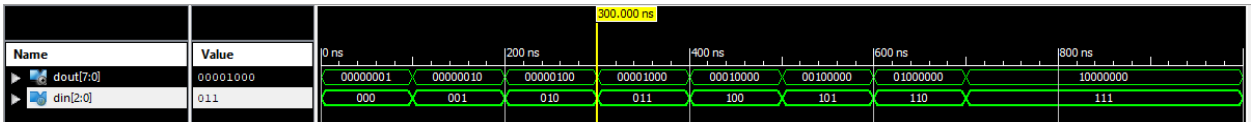


Σχ. Κυματομορφές εξομοίωσης αποκωδικοποιητή εντολών (ir_DECODER)

4.3 Ο αποκωδικοποιητής κατάστασης

```
module st_decoder (
input wire [2:0] din,
inout wire [7:0] dout
);

assign dout = din == 4'b 0000 ? 8'b 00000001 :
din == 4'b 0001 ? 8'b 00000010 :
din == 4'b 0010 ? 8'b 00000100 :
din == 4'b 0011 ? 8'b 00001000 :
din == 4'b 0100 ? 8'b 00010000 :
din == 4'b 0101 ? 8'b 00100000 :
din == 4'b 0110 ? 8'b 01000000 :
din == 4'b 0111 ? 8'b 10000000 :
{8{1'b0} };
endmodule
```

Η μονάδα st_decoder είναι ένας αποκωδικοποιητής που λαμβάνει ένα σήμα εισόδου 3-bit din και εξάγει ένα σήμα 8-bit dout.
Το σήμα εξόδου καθορίζεται από την τιμή του σήματος εισόδου.
Εάν το din είναι ίσο με μια συγκεκριμένη τιμή, το αντίστοιχο bit στο dout θα τεθεί σε 1, ενώ όλα τα υπόλοιπα θα τεθούν σε 0.
Οι πιθανές τιμές του din και οι αντίστοιχες τιμές τους στο dout καθορίζονται στην εντολή assign.



Σχ. Κυματομορφές εξομοίωσης αποκωδικοποιητή κατάστασης (st_DECODER)

4.4 Η υλοποίηση της hardwired μονάδας ελέγχου

```
module hardwired (
input wire [3:0] ir,
input wire clock,
input wire reset,
input wire z,
output wire [26:0] mOPs
);



wire FETCH1; wire FETCH2; wire FETCH3; wire NOP1;
wire LDAC1; wire LDAC2; wire LDAC3; wire LDAC4; wire LDAC5;
wire STAC1; wire STAC2; wire STAC3; wire STAC4; wire STAC5;
wire MVAC1; wire MOVR1; wire JUMP1; wire JUMP2; wire JUMP3;
wire JMPZ1; wire JMPZY1; wire JMPZY2; wire JMPZY3; wire JMPZN1; wire JMPZN2;
wire JPNZ1; wire JPNZY1; wire JPNZY2; wire JPNZY3; wire JPNZN1; wire JPNZN2;
wire ADD1; wire SUB1; wire INAC1; wire CLAC1; wire AND1; wire OR1; wire XOR1; wire
NOT1;
wire [2:0] cdata;
wire [7:0] T;
wire [15:0] I;
wire notz;
wire inc;
wire clear;

assign notz = ~ (z);
count #(
      .n (3))
```

```
  CA(
3'b000,
clock,
1'b0,
clear,
    inc,
    cdata );

  st_decoder DE(
      cdata,
T );

  ir_decoder ID(
      ir,
I );

assign inc = FETCH1 | FETCH2 | FETCH3 | LDAC1 | LDAC2 | LDAC3 | LDAC4 | STAC1 | STAC2
| STAC3 | STAC4 | JUMP1 | JUMP2 | JMPZY1 | JMPZY2 | JMPZN1 | JPNZY1 | JPNZY2 |
JPNZN1;
assign clear = NOP1 | LDAC5 | STAC5 | MVAC1 | MOVR1 | JUMP3 | JMPZY3 | JMPZN2 |
JPNZY3 | JPNZN2 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 | NOT1;
assign FETCH1 = T[ 0 ];
assign FETCH2 = T[ 1 ];
assign FETCH3 = T[ 2 ];
assign NOP1 = I[ 0] & T[3];
assign LDAC1 = I[ 1] & T[3];
assign LDAC2 = I[ 1] & T[4];
assign LDAC3 = I[ 1] & T[5];
assign LDAC4 = I[ 1] & T[6];
assign LDAC5 = I[ 1] & T[7];
assign STAC1 = I[ 2] & T[3];
assign STAC2 = I[ 2] & T[4];
assign STAC3 = I[ 2] & T[5];
assign STAC4 = I[ 2] & T[6];
assign STAC5 = I[ 2] & T[7];
assign MVAC1 = I[ 3] & T[3];
assign MOVR1 = I[ 4] & T[3];
assign JUMP1 = I[ 5] & T[3];
assign JUMP2 = I[ 5] & T[4];
assign JUMP3 = I[ 5] & T[5];
assign JMPZY1 = I[ 6] & z & T[3];
assign JMPZY2 = I[ 6] & z & T[4];
assign JMPZY3 = I[ 6] & z & T[5];
assign JMPZN1 = I[ 6] & notz & T[3];
assign JMPZN2 = I[ 6] & notz & T[4];
assign JPNZY1 = I[ 7] & notz & T[3];
assign JPNZY2 = I[ 7] & notz & T[4];
assign JPNZY3 = I[ 7] & notz & T[5];
assign JPNZN1 = I[ 7] & z & T[3];
assign JPNZN2 = I[ 7] & z & T[4];
assign ADD1 = I[ 8] & T[3];
assign SUB1 = I[ 9] & T[3];
assign INAC1 = I[ 10] & T[3];
assign CLAC1 = I[ 11] & T[3];
```

```
assign AND1 = I[ 12] & T[3];
assign OR1 = I[ 13] & T[3];
assign XOR1 = I[ 14] & T[3];
assign NOT1 = I[ 15] & T[3];
assign mOPs [ 0 ] = SUB1;
//MINUS
assign mOPs [ 1 ] = ADD1;
//PLUS
assign mOPs [ + 1] = CLAC1;
//ACZERO
assign mOPs [ + 1] = INAC1;
//ACINC
assign mOPs [ + 1] = NOT1;
//NOTOP
assign mOPs [ + 1] = XOR1;
//XOROP
assign mOPs [ + 1] = OR1;
//OROP
assign mOPs [ + 1] = AND1;
//ANDOP
assign mOPs [ 8 ] = STAC4 | MVAC1;
//ACBUS
assign mOPs [ 9 ] = MOVR1 | ADD1 | SUB1 | AND1 | OR1 | XOR1;
//RBUS
assign mOPs [ 10 ] = LDAC3 | STAC3 | JUMP3 | JMPZY3 | JPNZY3;
//TRBUS
assign mOPs [ 11] = LDAC2 | LDAC3 | LDAC5 | STAC2 | STAC3 | STAC5 | JUMP2 | JUMP3 |
JMPZY2 | JMPZY3 | JPNZY2 | JPNZY3;
//DRBUS
assign mOPs [ 12 ] = FETCH1 | FETCH3;
//PCBUS
assign mOPs [ 13] = STAC5;
//BUSMEM
assign mOPs [ 14] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | JUMP1 | JUMP2 |
JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
//MEMBUS
assign mOPs [ 15 ] = STAC5 ;
//wren
assign mOPs [ 16] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | JUMP1 | JUMP2 |
JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
// rden
assign mOPs [ 17] = LDAC5 | MOVR1 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 |
NOT1;
//ZLOAD
assign mOPs [ 18] = LDAC5 | MOVR1 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 |
NOT1;
//ACLOAD
assign mOPs [ 19 ] = MVAC1;
//RLOAD
assign mOPs [ 20 ] = FETCH3;
//IRLOAD
assign mOPs [ 21] = LDAC2 | STAC2 | JUMP2 | JMPZY2 | JPNZY2;
//TRLOAD
```

```
assign mOPs [ 22 ] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | STAC4 | JUMP1 |
JUMP2 | JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
//DRLOAD
assign mOPs [ 23 ] = FETCH2 | LDAC1 | LDAC2 | STAC1 | STAC2 | JMPZN1 | JMPZN2 |
JPNZN1 | JPNZN2;
//PCINC
assign mOPs [ 24] = JUMP3 | JMPZY3 | JPNZY3;
//PCLOAD
assign mOPs [ 25 ] = LDAC1 | STAC1 | JUMP1 | JMPZY1 | JPNZY1;
//ARINC
assign mOPs [ 26 ] = FETCH1 | FETCH3 | LDAC3 | STAC3;
    //ARLOAD
endmodule
```

Αυτός ο κώδικας περιγράφει ένα ψηφιακό κύκλωμα που δέχεται εισόδους εντολής 4 bit (ir), σήμα ρολογιού (clock), σήμα επαναφοράς (reset) και μια είσοδο ενός bit (z) και εξάγει ένα αποτέλεσμα 27 bit (mOPs).

Σε αυτόν τον κώδικα, η είσοδος ir περνάει από ένα στοιχείο ir_decoder (ID) για να παράγει ένα ενδιάμεσο σήμα 16-bit I.

Ένα άλλο στοιχείο st_decoder (DE) λαμβάνει το I και παράγει μια έξοδο 8-bit T.

Το στοιχείο count (CA) λαμβάνει ως εισόδους τα σήματα clock, reset και cdata και παράγει τα σήματα inc και clear.

Στη συνέχεια, ο κώδικας χρησιμοποιεί μια σειρά από εντολές assign για να δημιουργήσει μια σειρά από καλωδιακά σήματα που αντιπροσωπεύουν τις διάφορες καταστάσεις και λειτουργίες του FSM.

Αυτά τα ενδιάμεσα σήματα δημιουργούνται με βάση τις τιμές των T, I, z και notz (που είναι η άρνηση του z).

Τέλος, η έξοδος 27-bit mOPs δημιουργείται με βάση τις τιμές αυτών των ενδιάμεσων σημάτων και αντιπροσωπεύει το αποτέλεσμα των πράξεων που εκτελούνται από το FSM.

Είναι σημαντικό να σημειωθεί ότι αυτός ο κώδικας περιγράφει μόνο τη δομή και τη συμπεριφορά του κυκλώματος και θα πρέπει να μεταφραστεί σε μια φυσική υλοποίηση για την πραγματική εκτέλεση των περιγραφόμενων πράξεων.

4.5 Η εξωτερική μνήμη

```
module external_RAM (
input wire [7 : 0] address,
input wire clock,
```

```
input wire [7 : 0] data,
input wire wren,
output wire [7 : 0] q
);
wire [7 : 0] sub_wire0;

assign q = sub_wire0[7 : 0 ];
endmodule
```

Αυτός ο κώδικας περιγράφει μια ψηφιακή μονάδα που υλοποιεί εξωτερική μνήμη τυχαίας προσπέλασης (RAM). Λαμβάνει διάφορες εισόδους:

- διεύθυνση: ένας δίαυλος διευθύνσεων πλάτους 8 bit για τον προσδιορισμό της θέσης μνήμης στην οποία θα γίνει πρόσβαση.
- ρολόι: ένα σήμα ρολογιού που καθορίζει πότε μπορούν να διαβαστούν ή να εγγραφούν δεδομένα από τη μνήμη RAM
- data: δίαυλος δεδομένων πλάτους 8 bit που μεταφέρει τα δεδομένα που πρόκειται να εγγραφούν στη RAM ή να διαβαστούν από τη RAM.
- wren: ένα σήμα ενεργοποίησης εγγραφής που καθορίζει εάν τα δεδομένα εγγράφονται στη RAM ή διαβάζονται από αυτήν.

Διαθέτει επίσης μια έξοδο q: ένας δίαυλος δεδομένων πλάτους 8 bit που μεταφέρει τα δεδομένα που διαβάζονται από τη RAM.

Η μονάδα χρησιμοποιεί ένα μόνο καλώδιο, το sub_wire0, για την αποθήκευση των δεδομένων που διαβάζονται από τη RAM.

Η τιμή της εξόδου q ανατίθεται στη συνέχεια στα 8 λιγότερο σημαντικά bit του sub_wire0. Ωστόσο, αυτή η ενότητα είναι ελλιπής, καθώς δεν καθορίζει τον τρόπο εγγραφής των δεδομένων στη RAM ή τον τρόπο πρόσβασης στη μνήμη.

4.6 Η συνολική υλοποίηση της σχετικά απλής CPU

```
module rs_cpu (
inout wire [15:0] ARdata,
inout wire [15:0] PCdata,
inout wire [7:0] DRdata,
inout wire [7:0] ACdata,
inout wire [7:0] IRdata,
```

```
inout wire [7:0] TRdata,
inout wire [7:0] RRdata,
inout wire ZRdata,
input wire clock,
input wire reset,
inout wire [26:0] mOP,
inout wire [15:0] addressBus,
inout wire [7:0] dataBus
);

wire ARload ;
wire DRload ;
wire PCload ;
wire TRload ; wire IRload ; wire Rload ;
wire ACload ; wire Zload ; wire ACbus ; wire ARinc ;
wire PCbus ; wire PCinc ; wire DRbus ; wire TRbus ;
wire Rbus ; wire MEMbus ; wire busMEM ; wire wren;
wire [15:0] ARin ; wire [15:0] ARout ; wire [15:0] PCin ;
wire [15:0] PCout ; wire [15:0] BUSout ;
wire [7:0] Drin; wire [7:0] DRout ; wire [7:0] TRin ; wire [7:0] TRout ;
wire [7:0] IRin,DRin ; wire [7:0] IRout ; wire [7:0] Rin; wire [7:0] Rout;
wire [7:0] MEMin ; wire [7:0] MEMout ; wire [7:0] MEMdata ;
wire [7:0] ACin ; wire [7:0] ACout ; wire [7:0] ALUin ;
wire Zout ;
wire [6:0] alusel ;
wire [7:0] addr8bit;
wire [35:0] opcode;
wire [26:0] control;
wire [3:0] IRsignal ;

assign IRin = DRout ;
// Control Unit Selection
  genvar j;
generate for (j=0; j <= 3; j = j + 1) begin: p1
assign IRsignal [j] = IRin [j ];
// IR_signal to microsequencer
end
  endgenerate

  mseq_logic CUnit(
      IRsignal ,
clock,
reset,
opcode,
control );

// CUnit : hardwired port map ( IRsignal,clock ,reset,Zout,control );
assign ARload = control[ 26 ];
assign ARinc = control[ 25 ];
assign PCload = control[ 24 ];
assign PCinc = control[ 23 ];
assign DRload = control[ 22 ];
assign TRload = control[ 21 ];
assign IRload = control[ 20 ];
```

```
assign Rload = control[ 19 ];
assign ACload = control[ 18 ];
assign Zload = control[ 17 ];
assign wren = control[ 15 ];
assign MEMbus = control[ 14 ];
assign busMEM = control[ 13 ];
assign PCbus = control[ 12 ];
assign DRbus = control[ 11 ];
assign TRbus = control[ 10 ];
assign Rbus = control[ 9 ];
assign ACbus = control[ 8 ];
assign ARin = BUSout ;
//??
assign PCin = BUSout ;
//??
  genvar k;
generate for (k=0; k <= 7; k = k + 1) begin: Reg8
assign DRin [k] = BUSout [k ];
//??
assign Rin[k] = BUSout [k ];
assign TRin [k] = BUSout [ k + 8 ];
assign ALUin [k] = BUSout [ k + 8 ];
assign MEMin [k] = BUSout [ k + 8 ];
end
  endgenerate
mux2to1 #(
      .n (8))
  MBMUX(
8'b00000000,
    MEmin ,
    busMEM ,
    MEMdata );

  sys_bus SYSBUS(
      BUSout ,
    pcout ,
    DRout ,
    ACout ,
    TRout ,
rout,
    MEMout ,
    TRbus ,
    ACbus ,
    Rbus ,
    pcbus ,
    DRbus ,
    MEMbus );

  genvar J;
generate for (J=0; J <= 7; J = J + 1) begin: Addr8
assign addr8bit [J] = ARout [J ];
end
  endgenerate
```

```
  external_RAM RAM(
addr8bit,
clock,
    MEMdata ,
wren,
    MEMout );

  regb #(
      .n (16))
  ARregister (
      ARin ,
clock,
reset,
    ARload ,
    ARinc ,
    ARout );

  regb #(
      .n (16))
  PCregister (
      pcin ,
clock,
reset,
    pcload ,
    pcinc ,
    PCout );

  rega #(
      .n (8))
  DRregister (
      Drin ,
clock,
reset,
    DRload ,
    DRout );

  rega #(
      .n (8))
  TRregister (
      TRin ,
clock,
reset,
    TRload ,
    TRout );

  rega #(
      .n (8))
  IRregister (
      IRin ,
clock,
reset,
    IRload,
    IRout );
```

```
  rega #(
      .n(8))

  Rregister (
Rin,
clock,
reset,
    rload ,
Rout );

  rega #(
      .n (8))

  ACregister (
      ACin ,
clock,
reset,
    ACload ,
    ACout);

  regz #(
      .n (8))
  Zregister (
      ACout ,
clock,
reset,
1'b1,
    Zout );

  alu ALUnit (
      ACout ,
    aluminum ,
    alusel ,
    ACin );

  alus ALUnitS (
      Rbus ,
    ACload ,
    zload ,
    control[ 7],
    control[ 6],
    control[ 4 ],
    control[ 5],
    control[ 2 ],
    control[ 3 ],
    control[ 1 ],
    control[ 0 ],
    DRbus ,
    alusel );

assign ARdata = ARout ;
assign PCdata = PCout ;
assign DRdata = DRout ;
assign RRdata = Rout;
```

```
assign ACdata = ACout ;
assign IRdata = IRout ;
assign TRdata = TRout ;
assign ZRdata = Zout ;
assign mOP = control;
assign addressBus = ARout ;
assign dataBus = MEMdata ;

endmodule



module rega (
input wire [n - 1:0] din,
input wire clk ,
input wire res,
input wire load,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;



reg [n - 1:0] temp=0;

always @( posedge clk , posedge res) begin
    if( res == 1'b1) begin
temp <= {(((n - 1 ))-( (0))+1){1'b0}};
end else begin
      if( load == 1'b1) begin
temp <= din;
end
end
end

assign dout = temp;

endmodule



module regb (
input wire [n - 1:0] din,
input wire clk ,
input wire res,
input wire load,
input wire inc ,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;
```

```
reg [n - 1:0] temp;

always @( posedge clk , posedge res) begin
    if( res == 1'b1) begin
temp <= {(((n - 1 ))-( (0))+1){1'b0}};
end else begin
      if( load == 1'b1) begin
temp <= din;
end
else if( inc == 1'b1) begin
temp <= temp + 1;
end
end
end

assign dout = temp;

endmodule



module regz (
input wire [n - 1:0] din,
input wire clk ,
input wire reset,
input wire load,
output reg dout
);

parameter [31:0] n= 8;




always @( posedge clk , posedge reset) begin
    if( reset == 1'b1) begin
      dout <= 1'b0;
end else begin
      if( load == 1'b1) begin
        if( din <= 2'b00) begin
          dout <= 1'b1;
end
else begin
          dout <= 1'b0;
end
end
end
end
endmodule



module mux2to1(
```

```
input wire [n - 1:0] dina ,
input wire [n - 1:0] dinb ,
input wire sel ,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;



assign dout = sel == 1'b0 ? dina : dinb ;

endmodule



module mux4to1(
input wire [n - 1:0] dina ,
input wire [n - 1:0] dinb ,
input wire [n - 1:0] dinc ,
input wire [n - 1:0] dind ,
input wire [1:0] sel ,
output reg [n - 1:0] dout
);

parameter [31:0] n= 8;


always @( *) begin
case( sel )
2'b00 : dout <= dina ;
2'b01 : dout <= dinb ;
2'b10 : dout <= dinc ;
      default : dout <= dina ;
    endcase
  end
endmodule



module FA (
input wire A,
input wire B,
input wire Ci,
output wire S,
output wire Co
);


assign S = A ^ B ^ Ci;
assign Co = (A & B) | (Ci & A) | (Ci & B );
```

```
endmodule
```

```
module adder8bit(
input wire [7:0] dina ,
input wire [7:0] dinb ,
input wire cin ,
output wire [7:0] dout
);


wire [7:0] C;

FA FA_1(
      dina [ 0],
    dinb [ 0],
    cyn ,
    dout [ 0],
    C[ 0 ]);

  genvar i;
generate for ( i =1; i <= 7; i = i + 1)
begin: p1
FA FA_2(
          dina [ i ],
      dinb [ i ],
      C[ i - 1],
      dout [ i ],
C[ i ] );

end
  endgenerate

endmodule
```

```
module alu (
input wire [n - 1:0] ac,
input wire [n - 1:0] db ,
input wire [7:1] alus ,
output wire [7:0] dout
);

parameter [31:0] n= 8;

wire [n - 1:0] f1; wire [n - 1:0] f2; wire [n - 1:0] f3; wire [n - 1:0] f4;
wire [n - 1:0] zero; wire [n - 1:0] notdb ;
```

```
wire [n - 1:0] ANDout ; wire [n - 1:0] ORout ; wire [n - 1:0] XORout ; wire [n - 1:0]
NOTout ;

assign zero = {(((n - 1 ))-( (0))+1){1'b0}};
assign ANDout = ac & db ;
assign ORout = ac | db ;
assign XORout = ac ^ db ;
assign NOTout = ~ (ac);
assign notdb = ~ ( db );

mux2to1 MUX_1(
ZERO,
AC,
 alus[ 1],
f1 );

mux4to1 MUX_2(
ZERO,
    db ,
NOTDB,
ZERO,
    alus[ 2],
    alus [ 3] );

mux4to1 MUX_3(
      ANDout ,
    ORout ,
    XORout ,
    NOTout ,
    alus[ 5],
    alus[ 6]);

adder8bit PADDR(
f1,
f2,
    alus[ 4],
f3 );

mux2to1 MUX_4(
f3,
f4,
    alus[ 7],
    dout  );


endmodule




module alus (
input wire rbus ,
```

```
input wire acload ,
input wire zload ,
input wire opp ,
input wire orop ,
input wire notop ,
input wire xorop ,
input wire andop ,
input wire aczero ,
input wire async ,
input wire plus,
input wire minus,
input wire drbus ,
output reg [6:0] alus
);



wire [11:0] control;

assign control = { rbus,drbus
,acload,zload,andop,orop,notop,xorop,aczero,async,plus,minus};
always @(control) begin
case(control)
12'b101110000000 : begin
      alus <= 7'b1000000;
// AND
end
12'b 101101000000 : begin
      alus <= 7'b1100000;
// OR
end
12'b 001100100000 : begin
      alus <= 7'b1110000;
// NOT
end
12'b 101100010000 : begin
      alus <= 7'b1010000;
// XOR
end
12'b 001100001000 : begin
      alus <= 7'b0000000;
// CLAC
end
12'b 001100000100 : begin
      alus <= 7'b0001001;
// INAC
end
12'b 101100000010 : begin
      alus <= 7'b0000101;
// ADD
end
12'b 101100000001 : begin
      alus <= 7'b0001011;
```

```
// SUB
end
12'b 101100000000 : begin
      alus <= 7'b0000100;
// MOVR
end
12'b 011100000000 : begin
      alus <= 7'b0000100;
// LDAC5
end
    default : begin
      alus <= 7'b1111111;
// NO OPERATION
end
    endcase
end


endmodule
```

```
module sys_bus (
inout wire [15:0] sbus ,
input wire [15:0] PCdata ,
input wire [7:0] DRdata ,
input wire [7:0] ACdata ,
input wire [7:0] TRdata ,
input wire [7:0] Rdata ,
input wire [7:0] MEMdata ,
input wire trbus ,
input wire acbus,
input wire rbus ,
input wire pcbus  ,
input wire drbus ,
input wire membus
);

wire [5:0] control;

assign control = { membus,pcbus ,drbus,trbus,rbus,acbus };
assign sbus = control == 6'b 100000 ?
{8'h 00,MEMdata } : control == 6'b010000 ?
PCdata : control == 6'b001000 ?
{DRdata,8'h00 } : control == 6'b000100 ?
{8'h 00,TRdata } : control == 6'b000010 ?
{Rdata,8'h00 } : control == 6'b000001 ?
{8'h 00,ACdata } : control == 6'b101000 ?
{ DRdata,MEMdata } : control == 6'b001100 ?
{ DRdata , TRdata } : {16{1'b 0}};
```

```
endmodule
```

```
module mseq_logic (
input wire bt ,
input wire [1:0] cond ,
input wire z,
output wire s1,
output wire s0
);

reg muxout ;
wire notz ;

assign notz = ~ (z);
always @( *) begin
case( cond )
2'b 00 : muxout <= 1'b1;
2'b 01 : muxout <= z;
2'b 10 : muxout <= notz ;
      default : muxout <= 1'b0;
    endcase
end

assign s1 = bt ;
assign s0 = ~ ( muxout );

endmodule
```

```
module count(
input wire [n - 1:0] din,
input wire clock,
input wire load,
input wire clear,
input wire inc ,
output wire [n - 1:0] dout
);

parameter [31:0] n= 8;
reg [n-1:0] temp;

always @( posedge clock) begin : P1

    if( clear == 1'b1) begin
```

```
temp = {1{1'b0} };
end
else if( inc == 1'b1) begin
temp = temp + 1;
end
else if( load == 1'b1) begin
temp = din;
end
end

assign dout = temp;

endmodule
```

```
module ir_decoder (
input wire [3:0] din,
output wire [15:0] dout
);


assign dout = din == 8'h 00 ? 16'b 0000000000000001 : din == 8'h01 ? 16'b
0000000000000010 : din == 8'h02 ? 16'b 0000000000000100 : din == 8'h03 ? 16'b
0000000000001000 : din == 8'h04 ? 16'b 0000000000010000 : din == 8'h05 ? 16'b
0000000000100000 : din == 8'h06 ? 16'b 0000000001000000 : din == 8'h07 ? 16'b
0000000010000000 : din == 8'h08 ? 16'b 0000000100000000 : din == 8'h09 ? 16'b
0000001000000000 : din == 8'h0A ? 16'b 0000010000000000 : din == 8'h0B ? 16'b
0000100000000000 : din == 8'h0C ? 16'b 0001000000000000 : din == 8'h0D ? 16'b
0010000000000000 : din == 8'h0E ? 16'b 0100000000000000 : din == 8'h0F ? 16'b
1000000000000000 : {16{1'b0}};

endmodule

module st_decoder (
input wire [2:0] din,
inout wire [7:0] dout
);

assign dout = din == 4'b 0000 ? 8'b 00000001 :
din == 4'b 0001 ? 8'b 00000010 :
din == 4'b 0010 ? 8'b 00000100 :
din == 4'b 0011 ? 8'b 00001000 :
din == 4'b 0100 ? 8'b 00010000 :
din == 4'b 0101 ? 8'b 00100000 :
din == 4'b 0110 ? 8'b 01000000 :
din == 4'b 0111 ? 8'b 10000000 :
{8{1'b0} };

endmodule
```

```
module hardwired (
input wire [3:0] ir ,
input wire clock,
input wire reset,
input wire z,
output wire [26:0] mOPs
);



wire FETCH1; wire FETCH2; wire FETCH3; wire NOP1;
wire LDAC1; wire LDAC2; wire LDAC3; wire LDAC4; wire LDAC5;
wire STAC1; wire STAC2; wire STAC3; wire STAC4; wire STAC5;
wire MVAC1; wire MOVR1; wire JUMP1; wire JUMP2; wire JUMP3;
wire JMPZ1; wire JMPZY1; wire JMPZY2; wire JMPZY3; wire JMPZN1; wire JMPZN2;
wire JPNZ1; wire JPNZY1; wire JPNZY2; wire JPNZY3; wire JPNZN1; wire JPNZN2;
wire ADD1; wire SUB1; wire INAC1; wire CLAC1; wire AND1; wire OR1; wire XOR1; wire
NOT1;
wire [2:0] cdata ;
wire [7:0] T;
wire [15:0] I;
wire notz ;
wire inc ;
wire clear;

assign notz = ~ (z);
count #(
      .n (3))
  CA(
3'b000,
clock,
1'b0,
clear,
    inc ,
    cdata );

  st_decoder DE(
      cdata ,
T );

  ir_decoder ID(
      ir ,
I );

assign inc = FETCH1 | FETCH2 | FETCH3 | LDAC1 | LDAC2 | LDAC3 | LDAC4 | STAC1 | STAC2
| STAC3 | STAC4 | JUMP1 | JUMP2 | JMPZY1 | JMPZY2 | JMPZN1 | JPNZY1 | JPNZY2 |
JPNZN1;
```

```
assign clear = NOP1 | LDAC5 | STAC5 | MVAC1 | MOVR1 | JUMP3 | JMPZY3 | JMPZN2 |
JPNZY3 | JPNZN2 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 | NOT1;
assign FETCH1 = T[ 0 ];
assign FETCH2 = T[ 1 ];
assign FETCH3 = T[ 2 ];
assign NOP1 = I[ 0] & T[3];
assign LDAC1 = I[ 1] & T[3];
assign LDAC2 = I[ 1] & T[4];
assign LDAC3 = I[ 1] & T[5];
assign LDAC4 = I[ 1] & T[6];
assign LDAC5 = I[ 1] & T[7];
assign STAC1 = I[ 2] & T[3];
assign STAC2 = I[ 2] & T[4];
assign STAC3 = I[ 2] & T[5];
assign STAC4 = I[ 2] & T[6];
assign STAC5 = I[ 2] & T[7];
assign MVAC1 = I[ 3] & T[3];
assign MOVR1 = I[ 4] & T[3];
assign JUMP1 = I[ 5] & T[3];
assign JUMP2 = I[ 5] & T[4];
assign JUMP3 = I[ 5] & T[5];
assign JMPZY1 = I[ 6] & z & T[3];
assign JMPZY2 = I[ 6] & z & T[4];
assign JMPZY3 = I[ 6] & z & T[5];
assign JMPZN1 = I[ 6] & notz & T[3];
assign JMPZN2 = I[ 6] & notz & T[4];
assign JPNZY1 = I[ 7] & notz & T[3];
assign JPNZY2 = I[ 7] & notz & T[4];
assign JPNZY3 = I[ 7] & notz & T[5];
assign JPNZN1 = I[ 7] & z & T[3];
assign JPNZN2 = I[ 7] & z & T[4];
assign ADD1 = I[ 8] & T[3];
assign SUB1 = I[ 9] & T[3];
assign INAC1 = I[ 10] & T[3];
assign CLAC1 = I[ 11] & T[3];
assign AND1 = I[ 12] & T[3];
assign OR1 = I[ 13] & T[3];
assign XOR1 = I[ 14] & T[3];
assign NOT1 = I[ 15] & T[3];
assign mOPs [ 0 ] = SUB1 ;
//MINUS
assign mOPs [ 1 ] = ADD1 ;
//PLUS
assign mOPs [ + 1] = CLAC1;
//ACZERO
assign mOPs [ + 1] = INAC1;
//ACINC
assign mOPs [ + 1] = NOT1;
//NOTOP
assign mOPs [ + 1] = XOR1;
//XOROP
assign mOPs [ + 1] = OR1;
//OROP
assign mOPs [ + 1] = AND1;
```

```
//ANDOP
assign mOPs [ 8 ] = STAC4 | MVAC1;
//ACBUS
assign mOPs [ 9 ] = MOVR1 | ADD1 | SUB1 | AND1 | OR1 | XOR1;
//RBUS
assign mOPs [ 10 ] = LDAC3 | STAC3 | JUMP3 | JMPZY3 | JPNZY3;
//TRBUS
assign mOPs [ 11] = LDAC2 | LDAC3 | LDAC5 | STAC2 | STAC3 | STAC5 | JUMP2 | JUMP3 |
JMPZY2 | JMPZY3 | JPNZY2 | JPNZY3;
//DRBUS
assign mOPs [ 12 ] = FETCH1 | FETCH3;
//PCBUS
assign mOPs [ 13] = STAC5;
//BUSMEM
assign mOPs [ 14] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | JUMP1 | JUMP2 |
JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
//MEMBUS
assign mOPs [ 15 ] = STAC5 ;
//wren
assign mOPs [ 16] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | JUMP1 | JUMP2 |
JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
// rden
assign mOPs [ 17] = LDAC5 | MOVR1 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 |
NOT1;
//ZLOAD
assign mOPs [ 18] = LDAC5 | MOVR1 | ADD1 | SUB1 | INAC1 | CLAC1 | AND1 | OR1 | XOR1 |
NOT1;
//ACLOAD
assign mOPs [ 19 ] = MVAC1 ;
//RLOAD
assign mOPs [ 20 ] = FETCH3 ;
//IRLOAD
assign mOPs [ 21] = LDAC2 | STAC2 | JUMP2 | JMPZY2 | JPNZY2;
//TRLOAD
assign mOPs [ 22 ] = FETCH2 | LDAC1 | LDAC2 | LDAC4 | STAC1 | STAC2 | STAC4 | JUMP1 |
JUMP2 | JMPZY1 | JMPZY2 | JPNZY1 | JPNZY2;
//DRLOAD
assign mOPs [ 23 ] = FETCH2 | LDAC1 | LDAC2 | STAC1 | STAC2 | JMPZN1 | JMPZN2 |
JPNZN1 | JPNZN2;
//PCINC
assign mOPs [ 24] = JUMP3 | JMPZY3 | JPNZY3;
//PCLOAD
assign mOPs [ 25 ] = LDAC1 | STAC1 | JUMP1 | JMPZY1 | JPNZY1;
//ARINC
assign mOPs [ 26 ] = FETCH1 | FETCH3 | LDAC3 | STAC3;
    //ARLOAD
endmodule



module external_RAM (
input wire [7:0] address,
```

```
input wire clock,
input wire [7:0] data,
input wire wren,
output wire [7:0] q
);


wire [7:0] sub_wire0;

assign q = sub_wire0[7:0 ];


endmodule


module st_decoder (
input wire [2:0] din,
inout wire [7:0] dout
);




assign dout = din == 4'b 0000 ? 8'b 00000001 :
din == 4'b 0001 ? 8'b 00000010 :
din == 4'b 0010 ? 8'b 00000100 :
din == 4'b 0011 ? 8'b 00001000 :
din == 4'b 0100 ? 8'b 00010000 :
din == 4'b 0101 ? 8'b 00100000 :
din == 4'b 0110 ? 8'b 01000000 :
din == 4'b 0111 ? 8'b 10000000 :
{8{1'b0} };

endmodule


module mseq (
input wire [3:0] ir ,
input wire clock,
input wire reset,
input wire z,
output wire [35:0] code,
output wire [26:0] mOPs
);




wire [5:0] addr ;
wire [5:0] MUXout ; wire [5:0] REGout ;
reg[5:0] mux0; reg[5:0] mux1;
reg s1; reg s0;
wire [5:0] irmap ;
```

```
wire bt ;
wire [1:0] cond ; wire [1:0] control; wire [1:0] selmux ;
wire [35:0] microcode;
wire sel1; wire sel0;

assign irmap = {ir,2'b00 };
always @( REGout, addr , sel1, sel0, reset) begin
    if( reset == 1'b1) begin
s1 <= 1'b0;
s0 <= 1'b0;
mux0 <= 6'b000000;
mux1 <= 6'b000000;
end
else begin
s1 <= sel1;
s0 <= sel0;
mux0 <= REGout + 1;
mux1 <= addr ;
end
end

// MUXmap    : mseq_map port map ( irmap,ir );
  mseq_logic MUXlogic (
      btw ,
    cond ,
z,
sel1,
sel0 );

mux4to1 #(
      .n (6))
  MUXproc (
mux0,
mux1,
    irmap ,

s1,
s0,
    MUXout );

  rega #(
      .n (6))
  REGproc (
      MUXout ,
clock,
reset,
1'b1,
    REGout );

  micro_ROM MEMproc (
      MUXout ,
clock,
microcode );
```

```
  genvar j;
generate for (j=0; j <= 1; j = j + 1) begin: g1
assign cond [j] = microcode[ j + 34 ];
end
  endgenerate
assign bt = microcode[33];
  genvar m;
generate for (m=0;m<= 5; m=m+ 1) begin: g2
assign addr[m]= microcode[m];
end
  endgenerate
  genvar v;
generate for (v=0;v<= 26; v= v+ 1) begin: g3
assign mOPs [v]= microcode[ v+ 6 ];
end
  endgenerate
endmodule


module micro_ROM (
input wire [5:0] address,
input wire clock,
output wire [35:0] q
);
wire [35:0] sub_wire0=1211;

assign q = sub_wire0[35:0 ];
endmodule
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. COMPUTER SYSTEMS ORGANISATION AND ARCHITECTURE “JOHN D CAPRINELLI”
2. Youtube video (titles): 8 bit CPU designVERILOG MODELLING OF THE PROCESSOR (ALL PARTS) The best way to start learning Verilog
3. Udemy: Verilog HDL: VLSI Hardware Design Comprehensive Masterclass