Αρχιτεκτονική Υπολογιστών

Ενότητα 14: Σχεδιασμός μιας απλής CPU

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.ece.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός ενότητας

- Η αναλυτική παρουσίαση των βημάτων σχεδίασης μιας κεντρικής μονάδας επεξεργασίας.
- Η παρουσίαση των βελτιστοποιήσεων και η δικαιολόγηση των σχεδιαστικών αποφάσεων, κατά το σχεδιασμό της CPU.

Βασικές Λειτουργικές Μονάδες CPU

- Αριθμητική Λογική Μονάδα (Arithmetical Logical Unit, ALU).
- Μονάδα Ελέγχου (Control unit, CU).
- Συστοιχία Καταχωρητών (register file).

Λειτουργία CPU

- Μεταφορά εντολής στον επεξεργαστή (fetch).
- Αποκωδικοποίηση εντολής (decode).
- Εκτέλεση Εντολής (execute).
- Αποθήκευση αποτελεσμάτων (store).

Σχεδιασμός CPU

- Αναλύονται οι εφαρμογές που θα εκτελεστούν στη CPU (απλές, σύνθετες, ακέραιες τιμές, πραγματικές, κτλ).
- Αποφασίζονται οι εντολές που θα υποστηριχθούν (ISA).
- Αποφασίζονται οι ορατοί και οι κρυφοί καταχωρητές της αρχιτεκτονικής.
- Κατασκευάζεται το διάγραμμα καταστάσεων για κάθε εντολή της ISA, καθώς και τις μικρο-λειτουργίες που πρέπει να γίνονται σε κάθε στάδιο για κάθε συγκεκριμένη εντολή.
- Καθορίζονται οι εσωτερικοί δίαυλοι και τα σήματα ελέγχου.
- Όταν έχουν οριστεί όλα τα παραπάνω, σχεδιάζεται η μονάδα ελέγχου.

Καθορισμός της CPU

- Θα παρουσιάσουμε το σχεδιασμό μιας πολύ απλής CPU, π.χ. για τον έλεγχο ενός πλυντηρίου.
- Ονομάζεται **Very Simple CPU (VSC)**, και δεν αντιστοιχεί σε πραγματικό επεξεργαστή.
- Αποφασίζεται ότι θα σχεδιαστεί ένας επεξεργαστής 4bit, επειδή καλύπτει τις επεξεργαστικές απαιτήσεις μας.
- Στη συνέχεια θα:
  - Καθορίσουμε τις εντολές που θα υποστηρίζονται.
  - Καθορίζεται το διάγραμμα καταστάσεων, δηλαδή οι μικρο-λειτουργίες που απαιτούνται για τη μεταγωγή από τη μια κατάσταση στην άλλη, για κάθε εντολή.

Γενικό Διάγραμμα Καταστάσεων

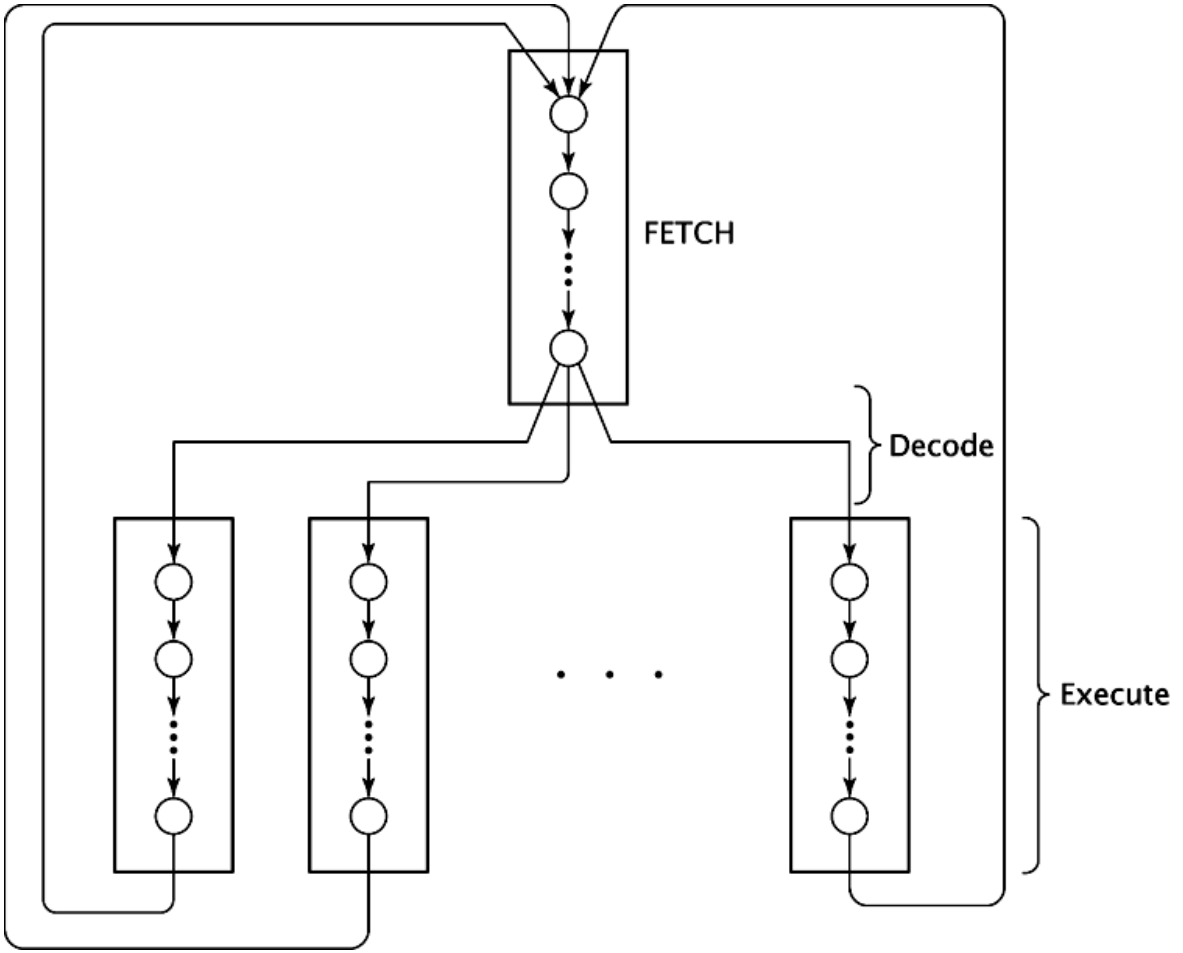
Διακρίνονται:

Fetch

Decode

execute

Στο σχήμα, το decode δεν έχει κάποιο στάδιο, επειδή η λειτουργία είναι οι πολλαπλές ροές εκτέλεσης μετά το στάδιο fetch, κατά το στάδιο execute.



Καθορισμός προδιαγραφών

- Διευθυνσιοδότηση 64 Bytes μνήμης.
- Κάθε Byte είναι 8 bit.
- Ο δίαυλος διευθύνσεων είναι 6 bit (000000 – 111111).
- Οι διευθύνσεις τοποθετούνται στα pin A[5..0].
- Ο δίαυλος δεδομένων είναι 8 bit (pin D[7..0]).
- Υπάρχει μόνο ένας καταχωρητής 8 bit που ονομάζεται AC (accumulator).
- Ο επεξεργαστής υποστηρίζει μόνο 4 εντολές (επόμενη διαφάνεια).

Οι εντολές του VSC

Εντολή	Κώδικας Μηχανής	Λειτουργία
ADD	00AAAAAA	AC <- AC + M[AAAAAA]
AND	01AAAAAA	AC <- AC ^ M[AAAAAA]
JMP	10AAAAAA	GOTO AAAAAA
INC	11XXXXXX	AC <- AC + 1

- Οι 4 εντολές είναι για **πρόσθεση**, λογική πράξη **ΚΑΙ**, **αλλαγή ροής εκτέλεσης**, και **αύξησης κατά 1**.
- Στον κώδικα μηχανής, χρησιμοποιούνται τα 2 πρώτα MSB για να κωδικοποιήσουν τις 4 αυτές λειτουργίες, ενώ τα υπόλοιπα 6 bit του κώδικα μηχανής, χρησιμοποιούνται για την παράμετρο της εντολής.
- Στη στήλη λειτουργία, αναλύεται η εντολή.

Επιπρόσθετοι Καταχωρητές

- Εκτός από τον καταχωρητή AC που είναι ορατός στο χρήστη, υπάρχουν και άλλοι καταχωρητές που χρησιμοποιούνται σε κάθε CPU για την ομαλή λειτουργία:
  - Καταχωρητής διευθύνσεων (address register, AR), που παρέχει μια διεύθυνση στη μνήμη. Στο παράδειγμά μας είναι 6 bit, λόγω του μεγέθους της μνήμης.
  - Καταχωρητής προγράμματος (program counter, PC), που περιέχει τη διεύθυνση της επόμενης εντολής προς εκτέλεση.
  - Καταχωρητής δεδομένων (data register, DR), που λαμβάνει τις εντολές και τα δεδομένα από τη μνήμη D[7..0].
  - Καταχωρητής εντολών (instruction register, IR), που αποθηκεύει το τμήμα op-code της εντολής (τα 2 πρώτα MSB) .

Στάδιο μεταφοράς εντολής από τη μνήμη 1

- Πριν την εκτέλεση της εντολής, πρέπει να μεταφερθεί αυτή από τη μνήμη στον επεξεργαστή.
- Βήματα:
  - Αποστολή της διεύθυνσης μέσω των pin A[5..0].
  - Μετά από συγκεκριμένους κύκλους που απαιτούνται ώστε η μνήμη να βρει το συγκεκριμένο κελί, αποστολή κατάλληλου σήματος για μεταφορά των 8βιτ δεδομένων στο CPU, μέσω D[7..0].
  - Η διεύθυνση της εντολής αποθηκεύεται στο μετρητή προγράμματος. Οπότε, το πρώτο βήμα είναι να γίνει η μεταφορά από το PC στο AR:

FETCH1: AR <- PC

Στάδιο μεταφοράς εντολής από τη μνήμη 2

- Στη συνέχεια, η CPU διαβάσει τα δεδομένα. Για να γίνει αυτό, θα πρέπει να στείλει ένα σήμα READ (ανάγνωσης), ώστε η μνήμη να τοποθετήσει τα δεδομένα στο δίαυλο (D[7..0]).
- Ταυτόχρονα, η CPU πρέπει να διαβάσει τα δεδομένα και να τα τοποθετήσει στο DR, που είναι ο καταχωρητής πρόσβασης στη μνήμη.
- Επίσης, σε αυτό το στάδιο γίνεται η αύξηση του μετρητής προγράμματος κατά 1.
- Αυτές οι λειτουργίες γίνονται στην κατάσταση FETCH2, επειδή η μνήμη καθυστερεί στην πρόσβαση.
  - **FETCH2: DR <- M, PC <-PC + 1**

Στάδιο μεταφοράς εντολής από τη μνήμη 3

- Αφού μεταφερθούν τα 8bit δεδομένα στο DR, στο στάδιο FETCH ο επεξεργαστής θα κάνει κάτι ακόμη:
  - Θα μεταφέρει τα 2 MSB που υποδηλώνουν τη λειτουργία στο IR.
  - Θα μεταφέρει τα 6 LSB που υποδηλώνουν την παράμετρο στο AR (αν και δε χρησιμοποιούν όλες οι εντολές το AR).

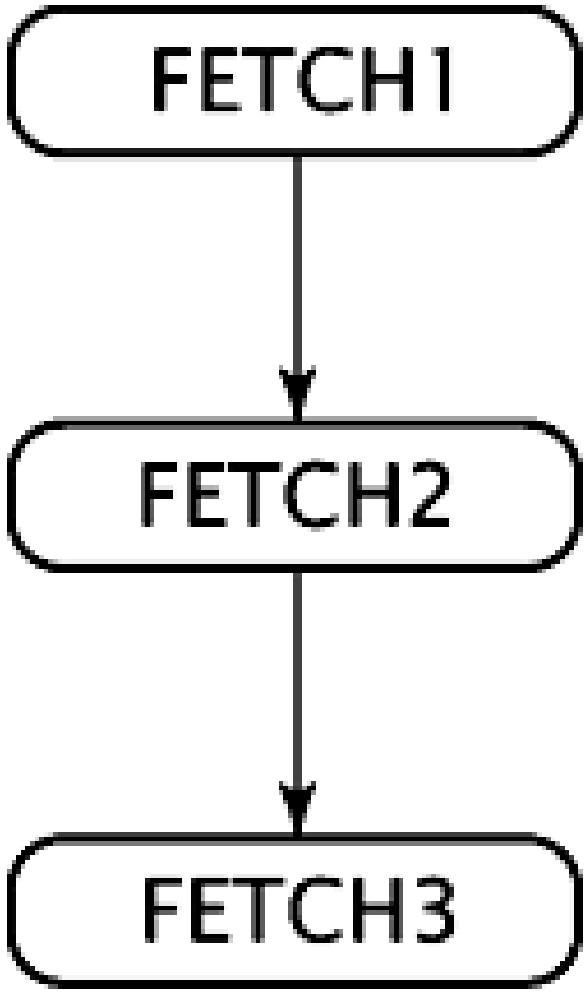
  FETCH3: IR <- DR[7..6], AR <- DR[5..0]
- Κατά τη σχεδίαση ενός επεξεργαστή, είναι προτιμότερο να τοποθετείται μια λειτουργία όσο πιο νωρίς γίνεται. Για παράδειγμα, η αύξηση του IR γίνεται στο FETCH2 (θα μπορούσε να γίνεται στο EXECUTE), ενώ η μεταφορά στο AR γίνεται στο FETCH3 (θα μπορούσε να γίνεται στο EXECUTE). Επίσης, το στάδιο EXECUTE είναι πολύ πολύπλοκο και η μεταφορά λειτουργιών σε άλλα στάδια είναι προτιμητέα.

Στάδιο μεταφοράς εντολής – Διάγραμμα Καταστάσεων

FETCH1: AR <-PC

FETCH2: DR <-M, PC <- PC + 1

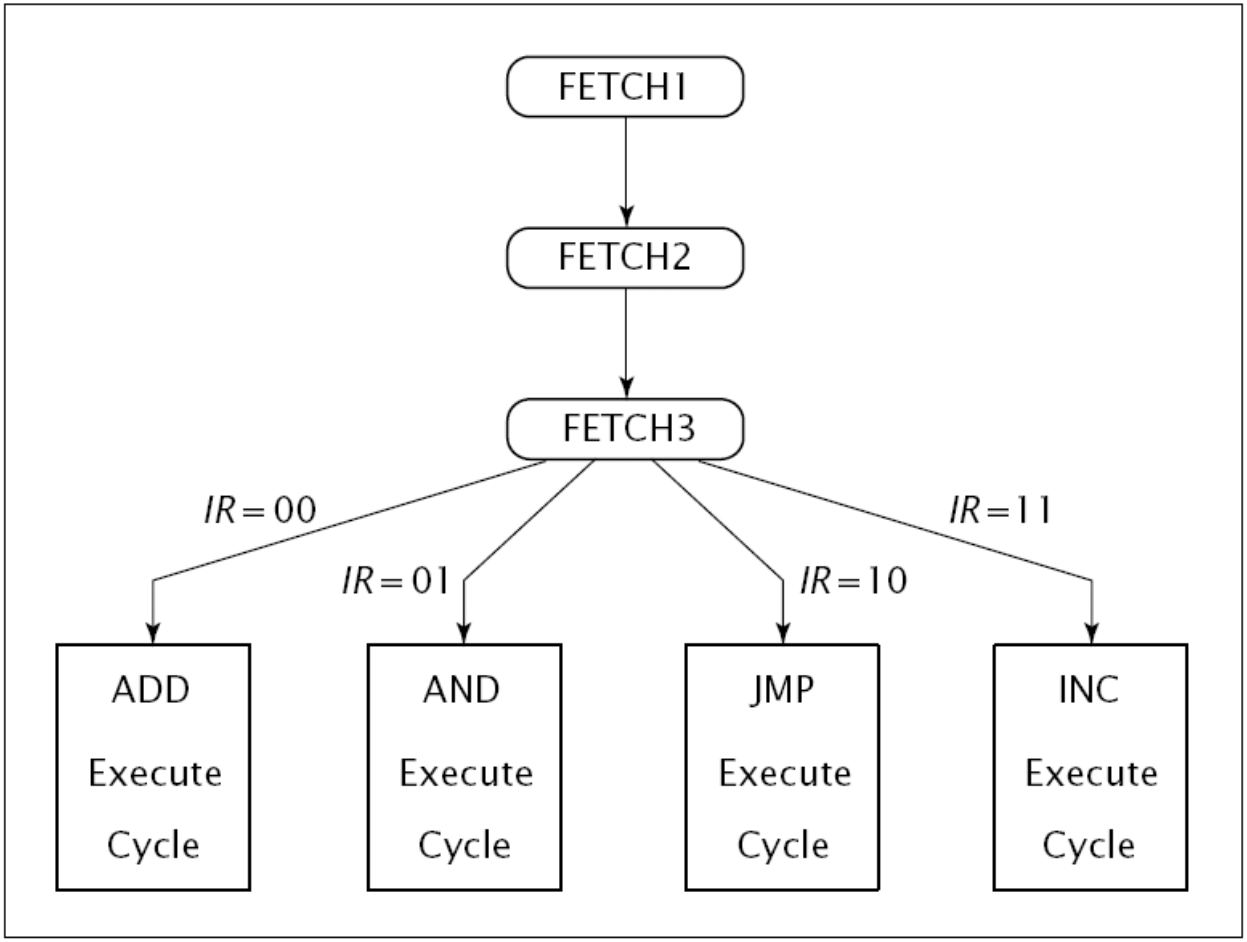
FETCH3: IR <- DR[7..6], AR <- DR[5..0]



Αποκωδικοποίηση εντολής

- Το επόμενο στάδιο είναι η αποκωδικοποίηση.
- Απαιτείται να προσδιοριστεί ποια λειτουργία θα εκτελεστεί, προκειμένου να δημιουργηθούν τα κατάλληλα σήματα ελέγχου.
- Στο VSC υπάρχουν μόνο 4 εντολές, οπότε 4 είναι τα πιθανά μονοπάτια.
- Για την εύρεση του μονοπατιού χρησιμοποιείται το IR, που κωδικοποιεί τις 4 εντολές (00 – 11).
- Το διάγραμμα καταστάσεων για την αποκωδικοποίηση εμφανίζεται στην επόμενη διαφάνεια.

Αποκωδικοποίηση εντολής – Διάγραμμα Καταστάσεων



Εκτέλεση Εντολής

- Το επόμενο στάδιο είναι η εκτέλεση της εντολής. Κάθε μονοπάτι, έχει τα δικά του σήματα ελέγχου.

Εκτέλεση Εντολής - ADD

- Η εντολή ADD, απαιτεί:
  - Τη μεταφορά της 2ης παραμέτρου από τη μνήμη.
    - Απαιτείται η διεύθυνση να τοποθετηθεί στο A[5..0] μέσω του AR (αυτό έχει γίνει προηγουμένως στο FETCH3).
    - Απαιτείται η μεταφορά των δεδομένων από τη μνήμη στον επεξεργαστή:

ADD1: DR <- M

  - Την εκτέλεση της άθροισης της τιμής του AC και της τιμής που μόλις ήρθε από τη μνήμη.
  - Την αποθήκευση του αποτελέσματος στο AC.

ADD2: AC <- AC + DR

Εκτέλεση Εντολής - AND

- Η εντολή AND, απαιτεί:
  - Τη μεταφορά της 2ης παραμέτρου από τη μνήμη.
  - Απαιτείται η διεύθυνση να τοποθετηθεί στο A[5..0] μέσω του AR (αυτό έχει γίνει προηγουμένως στο FETCH3).
  - Απαιτείται η μεταφορά των δεδομένων από τη μνήμη στον επεξεργαστή:

    AND1: DR <- M
  - Την εκτέλεση της λογικής πράξης AND του AC και της τιμής που μόλις ήρθε από τη μνήμη.
  - Την αποθήκευση του αποτελέσματος στο AC.

    AND2: AC <- AC ^ DR

Εκτέλεση Εντολής - JMP

- Η εντολή JMP, απαιτεί:
  - Τη μεταφορά της παραμέτρου, που ήδη έχει αποθηκευτεί στο DR[5..0] στο PC:

    JMP1: PC <- DR[5..0]
  - Εναλλακτικά, η μεταφορά μπορεί να γίνει από τον AR, στον οποίο έχουμε αποθηκεύσει την παράμετρο στο FETCH.

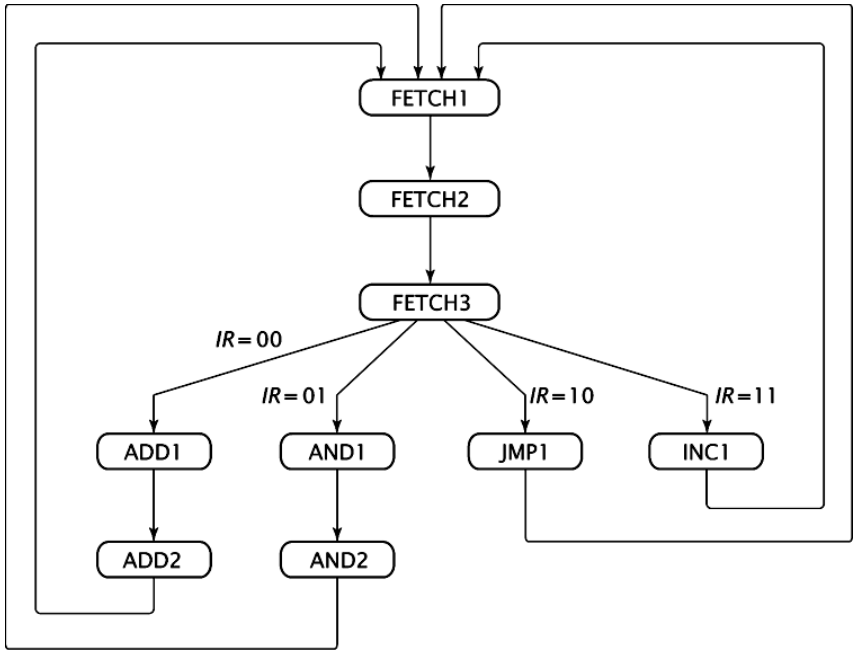
Εκτέλεση Εντολής - INC

- Η εντολή INC, απαιτεί:
  - Την προσθήκη 1 στο AC.
  - Τη μεταφορά του αποτελέσματος στο AC.

INC1: AC <-AC + 1

Καθορισμός διαύλων & διάγραμμα καταστάσεων

- Το επόμενο βήμα, είναι να καθοριστούν οι δίαυλοι του επεξεργαστή.
- Κοιτάμε όλες τις μικρο-λειτουργίες που συμβαίνουν σε όλες τις εντολές, ώστε να καθορίσουμε τι στοιχεία μετακινούνται προς ποια κατεύθυνση.



Επιλογή αποκλειστικών διαύλων 1-προς-1 ή κοινής χρήσης

- Τα pin της διεύθυνσης A[5..0] δέχονται δεδομένα από τον καταχωρητή διευθύνσεων AR, οπότε τα συνδέει ένα μονοπάτι .
- Η μνήμη παρέχει τα δεδομένα μέσω των pin D[7..0], οπότε από αυτά θα ξεκινάνε ένα ή περισσότερα μονοπάτια.
- Μπορεί να χρησιμοποιηθούν δίαυλοι 1-προς-1, ή ένας μεγάλος κοινός δίαυλος με πολυπλέκτες, απομονωτές και στοιχεία 3 καταστάσεων, ώστε καταχωρητές να δέχονται είσοδο από πολλές πηγές.
- Η λύση που **επιλέγεται είναι ένας κοινός δίαυλος**, λόγω χαμηλότερου κόστους.

Καθορισμός των λειτουργιών

- Οι λειτουργίες συνοπτικά:

FETCH1: AR <-PC
FETCH2: DR <- M, PC <- PC + 1
FETCH3: IR <- DR[7..6], AR <- DR[5..0]
ADD1: DR <- M
ADD2: AC <- AC + DR
AND1: DR <- M
AND2: AC <- AC ^ DR
JMP1: PC <- DR[5..0]
INC1: AC <- AC + 1

Το επόμενο βήμα είναι η προσεχτική μελέτη κάθε μικρο-λειτουργίας, για να βρεθούν οι λειτουργίες που συμβαίνουν σε κάθε συστατικό (π.χ. σε κάθε καταχωρητή).

Δημιουργία διαύλων

- Αναδιάταξη των λειτουργιών, και τακτοποίηση ως προς το πρώτο συστατικό:

  AR: AR <- PC; AR <- DR[5..0]

  PC: PC <- PC + 1; PC <- DR[5..0]

  DR: DR <- M

  IR: IR <- DR[7..6]

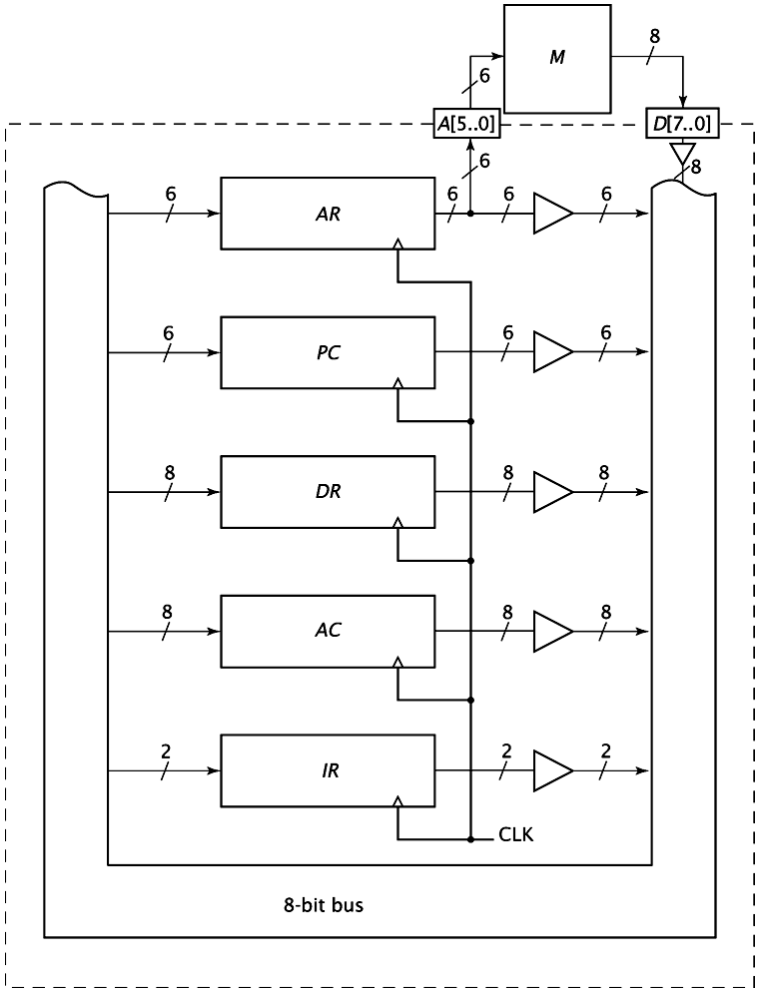
  AC: AC <- AC + DR; AC <- AC ^ DR; AC <- AC + 1

  - Τα AR, DR, IR πάντα φορτώνουν δεδομένα από κάποιο άλλο στοιχείο από το δίαυλο, οπότε απαιτείται η δυνατότητα παράλληλης φόρτωσης (parallel load).
  - Τα PC,AC φορτώνουν τιμές από εξωτερικά στοιχεία, αλλά απαιτείται και η αύξηση της τιμής τους (θα μπορούσε να τοποθετηθεί ειδικό υλικό, αλλά θα χρησιμοποιήσουμε καταχωρητές παράλληλης φόρτωσης).
  - Συνδέουμε όλα τα στοιχεία στο δίαυλο του συστήματος.

Πρώτη έκδοση του αρχείου καταχωρητών

Παρατηρήσεις:

- Το AR παρέχει πάντα πληροφορίες για τη μνήμη. Δε χρειάζεται να είναι συνδεδεμένος στο δίαυλο.
- Το IR παρέχει μόνο πληροφορίες για τη μονάδα ελέγχου, οπότε δε χρειάζεται να συνδέεται με άλλα στοιχεία.
- Το AC δεν παρέχει δεδομένα στα υπόλοιπα στοιχεία, μπορεί να απομακρυνθεί η σύνδεση.
- Ο δίαυλος είναι 8bit, αλλά δεν είναι όλες οι μεταφορές 8bit. Πρέπει να οριστούν ποια bit χρησιμοποιούνται. (π.χ. AR, PC bus bits[5..0], IR bus bits [7..6].
- Πρέπει να υπάρχει μια ALU με τις πράξεις ADD,AND



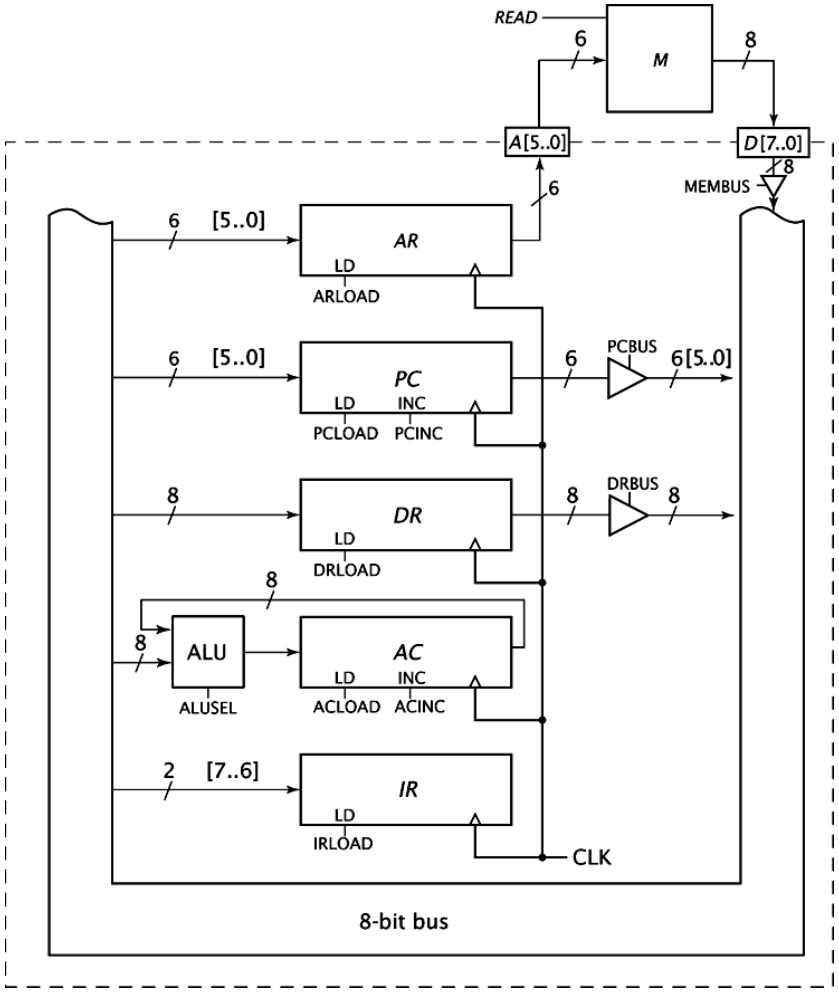
Προσθήκη ALU

- Απαιτούνται οι πράξεις: **ADD** και η **AND**.
- Η ALU θα μπορεί να δέχεται ως είσοδο το AC και το DR και θα στέλνει την έξοδο στο AC.
- Η πιο εύκολη λύση είναι το AC να είναι μόνιμα συνδεδεμένο σε δυο σημεία:
  - στην 1η είσοδο της ALU,
  - και στην έξοδο της ALU.

Έλεγχος παράλληλης εκτέλεσης

- Πρέπει να διασφαλιστεί ότι οι λειτουργίες που συμβαίνουν στο ίδιο στάδιο, μπορούν να συμβούν παράλληλα (δηλαδή, υπάρχουν αρκετοί πόροι για να καλύψουν τις λειτουργίες).
- Π.χ. αν στο ίδιο στάδιο χρησιμοποιείται ταυτόχρονα ο δίαυλος, τότε υπάρχει πρόβλημα (μόνο μια μεταφορά επιτρέπεται, αφού είναι κοινός ο δίαυλος).
- Για αυτό το λόγο η αύξηση του PC γίνεται καλύτερα με ειδικό υλικό μετρητή, χωρίς να χρησιμοποιείται ο δίαυλος.
- Εφόσον, σε κάθε στάδιο υπάρχει μόνο μια μεταφορά με δίαυλο, δεν υπάρχει πρόβλημα.

Δεύτερη έκδοση του αρχείου καταχωρητών



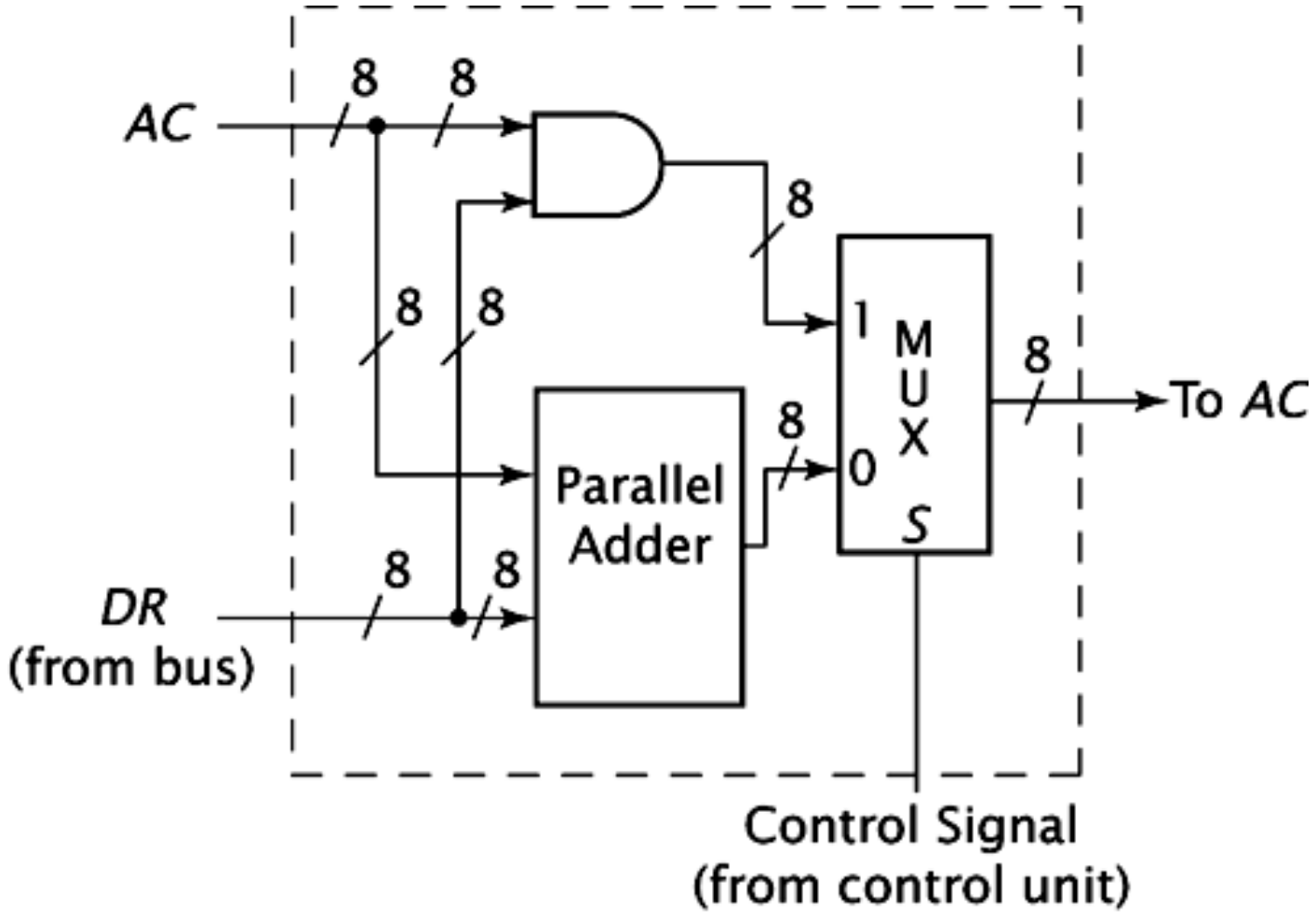
Αλλαγές:

- AR
- AC
- IR
- προσθήκη ALU

Σχεδιασμός της ALU

- Η ALU εκτελεί μόνο 2 λειτουργίες (AND και ADD).
- Ο πιο εύκολος τρόπος σχεδιασμού είναι να σχεδιαστεί με ξεχωριστό υλικό για κάθε λειτουργία και στη συνέχεια να συνδεθεί με ένα πολυπλέκτη για να επιλεχθεί είτε το πρώτο είτε το δεύτερο αποτέλεσμα.
- Η πρόσθεση γίνεται με ένα τυπικό αθροιστή 8bit ριπής (ripple carry adder).
- Η πράξη ΚΑΙ γίνεται με 8 πύλες AND.
- Χρησιμοποιείται ένας πολυπλέκτης 8bit 2 σε 1. Η είσοδος επιλογής ονομάζεται S (από το Select).

Διάγραμμα της ALU



Σχεδιασμός της μονάδας ελέγχου: Καλωδιωμένη Λογική

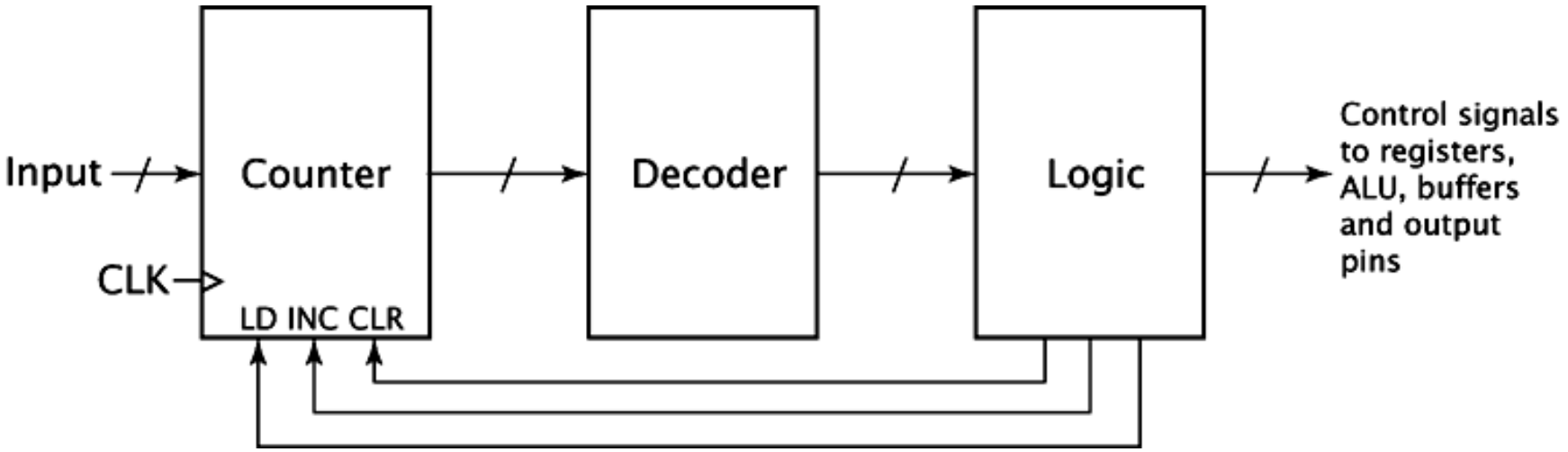
- Το επόμενο βήμα είναι ο σχεδιασμός του κυκλώματος, που θα δημιουργεί όλα τα σήματα ελέγχου προκειμένου όλες οι λειτουργίες να συμβούν με την κατάλληλη ακολουθία.
- Το κύκλωμα αυτό ονομάζεται **Μονάδα Ελέγχου**.
- Υπάρχουν 2 μεθοδολογίες σχεδιασμού της μονάδας ελέγχου:
  - **Καλωδιωμένη** (hardwired) μεθοδολογία.
  - **Μικρο-προγραμματισμένη** (microsequenced) μεθοδολογία.
- Επιλέγεται η καλωδιωμένη μεθοδολογία, γιατί δεν έχουμε πολύπλοκες απαιτήσεις από τη CU.

Καλωδιωμένη Μονάδα Ελέγχου

Η πιο απλή καλωδιωμένη μονάδα ελέγχου αποτελείται από:

- Μετρητή, που φέρει την τρέχουσα κατάσταση.
- Αποκωδικοποιητή, που δημιουργεί τα κατάλληλα σήματα κατάστασης, αναλόγως της τιμής του μετρητή.
- Συνδυαστική λογική, που λαμβάνει τα σήματα κατάστασης και δημιουργεί τα σήματα ελέγχου για κάθε συστατικό, όπως και το κατάλληλο σήμα προς το μετρητή. Με αυτά τα σήματα, υλοποιείται το διάγραμμα καταστάσεων και η σωστή μετάβαση από τη μια κατάσταση στην επόμενη.

Καλωδιωμένη Μονάδα Ελέγχου: Γενικό Κύκλωμα



Σήματα μονάδας ελέγχου - 1

- Υπάρχουν 9 καταστάσεις
  - Απαιτείται μετρητής καταστάσεων 4bit,
  - Οπότε, και ένας αποκωδικοποιητής 4 σε 16bit, από τις οποίες θα χρησιμοποιηθούν μόνο οι 9 έξοδοι (7 έξοδοι του αποκωδικοποιητή δε θα χρησιμοποιηθούν).
- Γενικές Οδηγίες:
  - Η κατάσταση FETCH1 θα τοποθετηθεί στη τιμή 0 του μετρητή. Με το CLR θα μηδενίζεται ο μετρητής. Η κατάσταση FETCH1 μπορεί να συμβεί από 4 άλλες καταστάσεις (σε αντίθεση με τις επόμενες καταστάσεις που συμβαίνουν μόνο μετά από μια κατάσταση).
  - Με το σήμα CLR λοιπόν εύκολα από τις 4 καταστάσεις θα καταλήγουμε στη FETCH1.

Σήματα μονάδας ελέγχου - 2

- Οι καταστάσεις που ακολουθεί η μια την άλλη, θα έχουν διαδοχικές τιμές, προκειμένου να χρειάζεται μια απλή λειτουργία αύξησης του μετρητή κατά 1 για να διέλθει από όλες αυτές το CU. Παράδειγμα:
  - FETCH3, FETCH2, FETCH1 συνεχόμενες τιμές μετρητή.
  - ADD1, ADD2 συνεχόμενες τιμές μετρητή.
  - AND1, AND2 συνεχόμενες τιμές μετρητή.

Σήματα μονάδας ελέγχου - 3

- Με βάση το opcode της εντολής και το μέγιστο πλήθος μικρο-λειτουργιών, να τοποθετηθεί η πρώτη μικρο-λειτουργία ενός σταδίου, να γίνει εκχώρηση των τιμών του μετρητή. Τα opcodes θα είναι είσοδοι στο μετρητή και στην είσοδο LD (load).
- Με αυτόν τον τρόπο γίνεται μια ευθεία λειτουργία αποκωδικοποίησης (δε χρειάζεται άλλο ειδικό κύκλωμα).
- Αυτό θα πρέπει να γίνει στο τελευταίο στάδιο της κατάστασης FETCH, προκειμένου να συνεχίσει ο επεξεργαστής προς την αποκωδικοποίηση και εκτέλεση.

Αντιστοίχηση Διευθύνσεων στο μετρητή σε μικρολειτουργίες 1

- Απαιτείται η αντιστοίχηση κάθε τιμής του μετρητή σε μια μικρολειτουργία (αρκετά δύσκολο).
- Ως προς την υλοποίηση, θα πρέπει να φορτωθεί η τιμή στο μετρητή με:
  - Τοποθέτηση της τιμής στην παράλληλη είσοδο του μετρητή.
  - Ανύψωση του σήματος LD, για παράλληλη φόρτωση (εύκολο να υλοποιηθεί, αφού θα συμβαίνει αυτόματα κάθε φορά στο τελευταίο στάδιο του FETCH, δηλαδή στο FETCH3).

Αντιστοίχηση Διευθύνσεων στο μετρητή σε μικρολειτουργίες 2

Instruction	First State	*IR*
ADD	ADD1	00
AND	AND1	01
JMP	JMP1	10
INC	INC1	11

Ο μετρητής είναι 4bit

Το πρώτο βήμα είναι:

- Να καταγραφούν οι εντολές.
- Η πρώτη κατάσταση της εντολής.
- Το machine code για αυτή την εντολή.

ΣΚΟΠΟΣ: Να χρησιμοποιήσουμε τα IR αυτούσια στην τιμή του μετρητή. π.χ. μπορούμε να έχουμε τα 2 bit 10 και μετά τα 2 bit του IR (δεν είναι καλή κωδικοποίηση).

Αντιστοίχηση Διευθύνσεων στο μετρητή σε μικρολειτουργίες 3

IR[1..0]	Counter Value	State
00	1000 (8)	ADD1
01	1001 (9)	AND1
10	1010 (10)	JMP1
11	1011 (11)	INC1

- Χρησιμοποιώντας αυτή την κωδικοποίηση, παρουσιάζεται το πρόβλημα ότι δε μπορούμε να εκμεταλλευτούμε την οδηγία “όλες οι μικρολειτουργίες του ίδιου σταδίου να έχουν συνεχόμενες τιμές”.
- **ΔΕΝ είναι αποδεκτή κωδικοποίηση.**

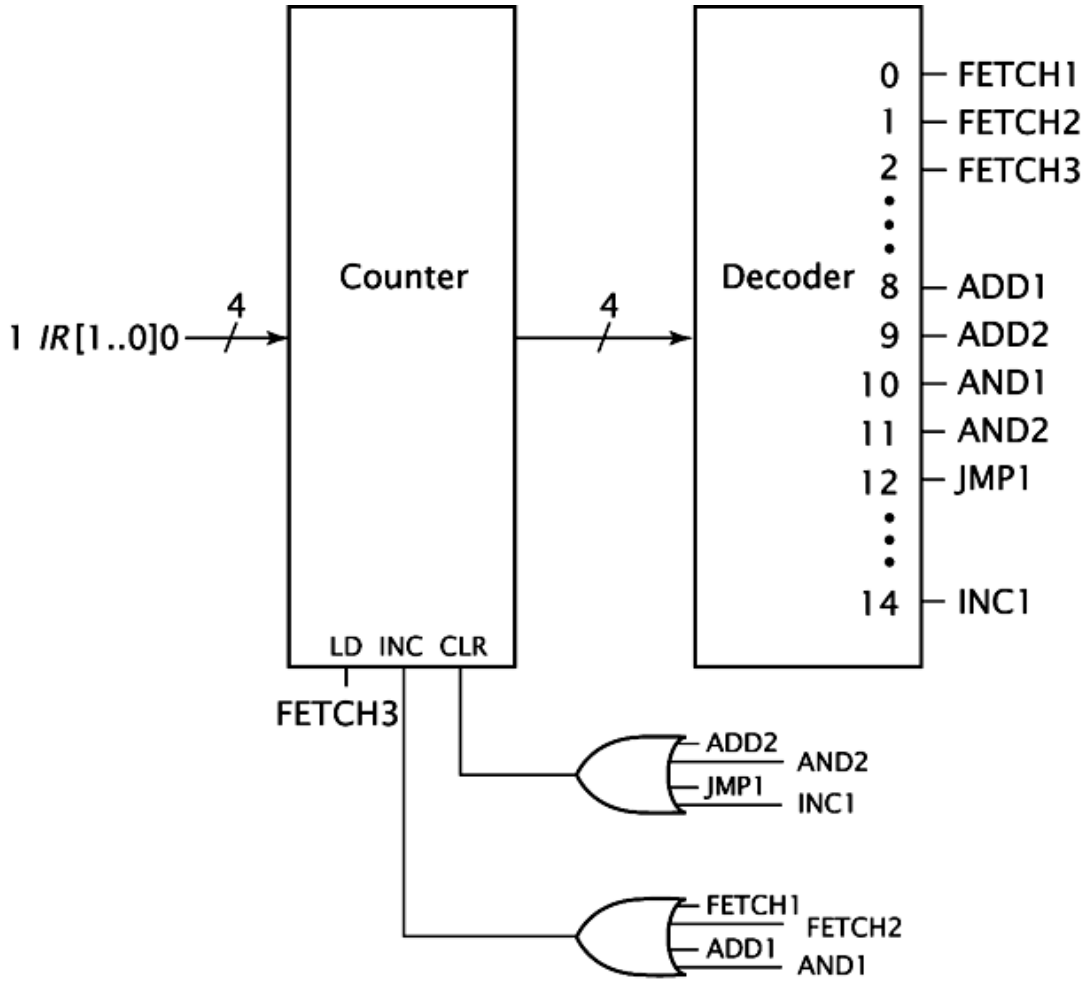
Αντιστοίχηση Διευθύνσεων στο μετρητή σε μικρολειτουργίες 4

- Παρατηρούμε ότι ο μέγιστος αριθμός σταδίων σε όλες τις εντολές είναι 2. Δηλαδή, αρκεί να βρούμε μια κωδικοποίηση που επιτρέπει να έχουμε τουλάχιστον 2 θέσεις κενές.
- Αυτή είναι η: 1 IR[1..0] 0 , δηλαδή, το πρώτο bit 1, ακολουθούν τα 2 bit του IR, και τέλος το bit 0.
- Ως εκ τούτου έχουμε τις αντιστοιχήσεις:
  - ADD1 (00) => 1000 (=8)
  - AND1 (01) => 1010 (=10)
  - JMP1 (10) => 1100 (=12)
  - INC1 (11) => 1110 (=14)

Αντιστοίχηση Διευθύνσεων στο μετρητή σε μικρολειτουργίες 5

- Από τη στιγμή που έχουμε καθορίσει τις αντιστοιχήσεις διευθύνσεων, μπορούμε να δημιουργήσουμε τα κατάλληλα σήματα ελέγχου.
- Τα σήματα ελέγχου για το μετρητή (**μόνο**) είναι:
  - **INC** (αύξησης κατά 1),
    - Χρησιμοποιείται στις συνεχόμενες καταστάσεις (δηλαδή, ακολουθεί στάδιο της ίδιας κατάστασης).
    - Δηλαδή: FETCH1,FETCH2,ADD1,AND1.
  - **CLR** (καθαρισμού),
    - Χρησιμοποιείται στο τέλος της κατάστασης εκτέλεσης, για να μηδενιστεί ο μετρητής, και να αρχίσει πάλι από το στάδιο FETCH1.
    - Δηλαδή: ADD2, AND2, JMP1, INC1
  - **LD** (φόρτωσης),
    - Χρησιμοποιείται για τη φόρτωση τιμής στο μετρητή, στο τέλος του FETCH.
    - Δηλαδή: FETCH3.

Μονάδα ελέγχου με καλωδιωμένη λογική



Υπόλοιπα σήματα ελέγχου (καταχωρητές) 1

- Εκτός από τα σήματα ελέγχου του μετρητή, απαιτούνται και σήματα ελέγχου για τις υπόλοιπες μονάδες (AR,PC,DR,IR,M,ALU,buffers).
- Τα σήματα αυτά προκύπτουν με συνδυασμό καταστάσεων του μετρητή.
- Παράδειγμα AR (**AR: AR <- PC; AR <- DR[5..0]**).
  - Χρησιμοποιείται στις καταστάσεις FETCH1 και FETCH3.
  - Αρκεί να γίνει η πράξη OR ανάμεσα στις καταστάσεις FETCH1 (έξοδος αποκωδικοποιητή 0) και FETCH3 (έξοδος αποκωδικοποιητή 2) για να δημιουργηθεί το σήμα ARLOAD για τη φόρτωση του AR με δεδομένα.
  - Ασφαλώς, με άλλα σήματα θα φροντίσουμε τα δεδομένα που θα φορτωθούν να είναι αυτά που πρέπει (είτε το PC, είτε το DR[5..0]).

Υπόλοιπα σήματα ελέγχου (καταχωρητές) 2

- Με παρόμοιο τρόπο δημιουργούνται τα παρακάτω σήματα ελέγχου:
  - PCLOAD = JMP1 (η τιμή του PC αλλάζει, στο στάδιο JMP1).
  - PCINC = FETCH2 (η τιμή του PC αυξάνει κατά 1, στο στάδιο FETCH2).
  - DRLOAD = FETCH1 **OR** ADD1 **OR** AND1 (ο DR φορτώνει δεδομένα, είτε στο στάδιο FETCH1, είτε στο στάδιο ADD1, είτε στο στάδιο AND1).
  - ACLOAD = ADD2 **OR** AND2 (ο AC φορτώνει δεδομένα, είτε στο στάδιο ADD2, είτε στο στάδιο AND2).
  - ACINC = INC1 (η τιμή του AC αυξάνει κατά 1, στο στάδιο INC1).
  - IRLOAD = FETCH3 (γράφονται τα 2 bit του IR στο στάδιο FETCH3).

Υπόλοιπα σήματα ελέγχου (ALU)

- Η ALU έχει ένα σήμα ελέγχου ALUSEL:
  - Αν το ALUSEL είναι 0, τότε η ALU εκτελεί ADD.
  - Αν το ALUSEL είναι 1, τότε η ALU εκτελεί AND.
- Ως εκ τούτου αν θέσουμε ALUSEL = AND2, τότε επιτελείται αυτή η λειτουργία όταν η ALU χρησιμοποιείται για AND ή ADD.

=> Με την παραπάνω σύνδεση, απλοποιείται το κύκλωμα, αλλά αυτό σημαίνει ότι η ALU θα χρησιμοποιείται σε κάθε στάδιο, απλώς δε θα αποθηκεύεται η τιμή στο AC σε κάθε άλλο στάδιο (δε θα υπάρχει το σήμα ACLOAD).

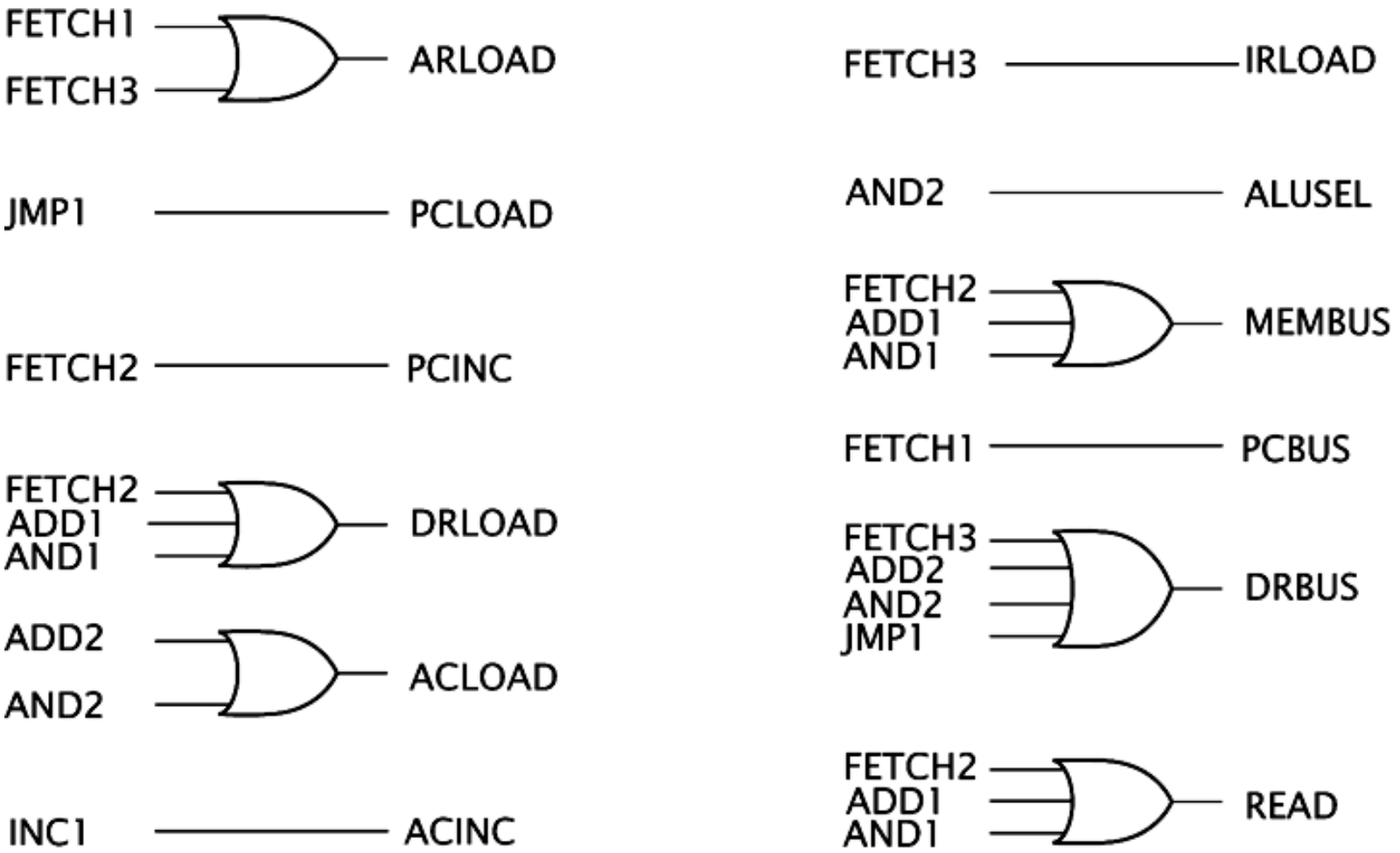
Υπόλοιπα σήματα ελέγχου (δίαυλος) 1

- Πολλές λειτουργίες χρησιμοποιούν σήματα από το εσωτερικό δίαυλο συστήματος.
- Πρέπει να δημιουργηθούν τα κατάλληλα σήματα ελέγχου, ώστε να τοποθετηθούν τα σωστά δεδομένα στο δίαυλο την κατάλληλη στιγμή.
- Το DR πρέπει να τοποθετηθεί στο δίαυλο στα στάδια: FETCH3, ADD2, AND2, JMP1. Με τη λογική πράξη OR ανάμεσα σε αυτές τις εξόδους του αποκωδικοποιητή δημιουργείται το σήμα DRLOAD (φόρτωση στον καταχωρητή DR).
- DRLOAD = FETCH3 **OR** ADD2 **OR** AND2 **OR** JMP1.

Υπόλοιπα σήματα ελέγχου (δίαυλος) 2

- Ομοίως, δημιουργούνται τα σήματα, MEMBUS (μεταφορά τον DR στο δίαυλο συστήματος), και PCBUS (μεταφορά από το PC στο δίαυλο συστήματος):
  - MEMBUS = FETCH2 **OR** ADD1 **OR** AND1.
  - PCBUS = FETCH1.
- Επίσης, πρέπει να δημιουργείται και το σήμα READ, ώστε η μνήμη να τοποθετήσει τα δεδομένα στο D[7..0]:
  - READ = FETCH2 **OR** ADD1 **OR** AND1.

Δημιουργία σημάτων ελέγχου VSC



Επιβεβαίωση ορθής λειτουργίας 1

- Μετά το σχεδιασμό, πρέπει να επιβεβαιωθεί η ορθή λειτουργία. Για να γίνει αυτό, εξετάζεται η ακολουθία κάθε εντολής στα στάδια fetch-decode-execute.
- Έστω για παράδειγμα, ότι έχουμε το παρακάτω κομμάτι κώδικα (πρώτη στήλη διεύθυνση μνήμης):
  - 0 ADD 4
  - 1 AND 5
  - 2 INC
  - 3 JMP 0
  - 4 27H
  - 5 39H

Επιβεβαίωση ορθής λειτουργίας 2

- Ο επεξεργαστής, εκτελεί τα fetch/decode/execute για κάθε εντολή:
  - ADD 4: FETCH1->FETCH2->FETCH3->ADD1->ADD2
  - AND 5: FETCH1->FETCH2->FETCH3->AND1->AND2
  - INC: FETCH1->FETCH2->FETCH3->INC1
  - JMP 0: FETCH1->FETCH2->FETCH3->JMP1
- Μπορούμε να δούμε το ίχνος εκτέλεσης και δημιουργίας σημάτων στις επόμενες διαφάνειες.

Επιβεβαίωση μέσω ίχνους εκτέλεσης 1

Instruction	State	Active Signals	Operations Performed	Next State
ADD 4	FETCH1	PCBUS, ARLOAD	*AR*←0	FETCH2
	FETCH2	READ, MEMBUS, DRLOAD, PCINC	*DR*←04H, *PC*←1	FETCH3
	FETCH3	DRBUS, ARLOAD, IRLOAD	*IR*←00, *AR*←04H	ADD1
	ADD1	READ, MEMBUS, DRLOAD	*DR*←27H	ADD2
	ADD2	DRBUS, ACLOAD	*AC*←0 + 27H = 27H	FETCH1
AND 5	FETCH1	PCBUS, ARLOAD	*AR*←1	FETCH2
	FETCH2	READ, MEMBUS, DRLOAD, PCINC	*DR*←45H, *PC*←2	FETCH3
	FETCH3	DRBUS, ARLOAD, IRLOAD	*IR*←01, *AR*←05H	AND1
	AND1	READ, MEMBUS, DRLOAD	*DR*←39H	AND2
	AND2	DRBUS, ALUSEL, ACLOAD	*AC*←27H ^ 39H = 31H	FETCH1

Επιβεβαίωση μέσω ίχνους εκτέλεσης 2

INC	FETCH1	PCBUS, ARLOAD	*AR*←2	FETCH2
	FETCH2	READ, MEMBUS, DRLOAD, PCINC	*DR*←C0H, *PC*←3	FETCH3
	FETCH3	DRBUS, ARLOAD, IRLOAD	*IR*←11, *AR*←00H	INC1
	INC1	ACINC	*AC*←21H + 1 = 22H	FETCH1
JMP 0	FETCH1	PCBUS, ARLOAD	*AR*←3	FETCH2
	FETCH2	READ, MEMBUS, DRLOAD, PCINC	*DR*←80H, *PC*←4	FETCH3
	FETCH3	DRBUS, ARLOAD, IRLOAD	*IR*←10, *AR*←00H	JMP1
	JMP1	DRBUS, PCLOAD	*PC*←0	FETCH1

Μια πιο σύνθετη σχεδίαση CPU

Μια πιο σύνθετη σχεδίαση: Relatively Simple CPU

- Η προηγούμενη σχεδίαση ήταν μια πολύ απλή CPU.
- Θα εξετάσουμε τη σχεδίαση μιας πιο σύνθετης CPU (αλλά σχετικά απλής).
- Ονομάζεται Relatively Simple CPU (RSC).
- Αυτή η CPU, έχει περισσότερες εντολές, καταχωρητές, και διαθέσιμη μνήμη.

Προδιαγραφές RSC 1

- 64KB μνήμης των 8 bit.
- 8 bit δίαυλο.
- Pin διευθύνσεων A[15..0].
- Pin δεδομένων διπλής κατεύθυνσης D[7..0].
- 3 καταχωρητές διαθέσιμοι στο χρήστη:
  - AC, 8 bit.
  - R, 8 bit ως δεύτερη παράμετρος.
  - Z, 1 bit (σημαία) τίθεται μετά από αριθμητική/λογική πράξη.

Προδιαγραφές RSC 2

- Το AC, χρησιμοποιείται πάντα ως η πρώτη είσοδος στο ALU σε διμελείς πράξεις, και είναι η έξοδος της ALU.
- Όταν μεταφέρεται ένα Byte από τη μνήμη, φορτώνεται πάντα στο AC.
- Όταν μεταφέρεται ένα Byte στη μνήμη, διαβάζεται πάντα από το AC.
- Ο καταχωρητής R, είναι γενικού τύπου και χρησιμοποιείται στις διμελείς πράξεις. Μπορεί να χρησιμοποιηθεί και για την προσωρινή αποθήκευση δεδομένων.

Εντολές του RSC

Instruction	Instruction Code	Operation
NOP	0000 0000	No Operation
LDAC	0000 0001 Γ	AC ,-- M[Γ]
STAC	0000 0010 Γ	M[Γ] <- AC
MVAC	0000 0011	R <- AC
MOVR	0000 0100	AC <- R
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF(Z=1) THEN GOTO Γ
JPNZ	0000 0111 Γ	IF(Z=0) THEN GOTO Γ
ADD	0000 1000	AC <-AC + R, IF(AC + R = 0) THEN Z <-1 ELSE Z <- 0
SUB	0000 1001	AC <- AC - R, IF(AC - R = 0) THEN Z <- 1 ELSE Z <- 0
INAC	0000 1010	AC<- AC + R, IF(AC + R = 0) THEN Z <- 1 ELSE Z <- 0
CLAC	0000 1011	AC <- 0, Z <- 1
AND	0000 1100	AC <- AC ^ R, IF(AC ^ R = 0) THEN Z <- 1 ELSE Z <- 0
OR	0000 1101	AC<- AC | R, IF(AC | R = 0) THEN Z <-1 ELSE Z <- 0
XOR	0000 1110	AC <- AC -| R, IF(AC -| R = 0) THEN Z <- 1 ELSE Z <- 0
NOT	0000 1111	AC <- AC`, IF(AC` = 0) THEN Z <- 1 ELSE Z <- 0

Επιπρόσθετοι Καταχωρητές

- Εκτός από τους καταχωρητές που είναι ορατοί στο χρήστη, υπάρχουν και άλλοι καταχωρητές.
  - Καταχωρητής διευθύνσεων AR 16bit, που γράφει στη μνήμη A[15..0].
  - Μετρητής προγράμματος PC 16bit, που φέρει είτε τη διεύθυνση της επόμενης προς εκτέλεση εντολή, είτε τη διεύθυνση της επόμενης παραμέτρου (αν απαιτείται).
  - Καταχωρητής δεδομένων DR 8bit, που είτε γράφει, είτε διαβάζει διαμέσου των pins D[7..0].
  - Καταχωρητής εντολών IR 8bit, που αποθηκεύει το opcode της τρέχουσας εντολής.
  - Προσωρινής καταχωρητής TR 8bit, που αποθηκεύει προσωρινά δεδομένα κατά την εκτέλεση.

Βελτιώσεις του επεξεργαστή RSC

- Παρατηρούμε ότι εκτός από τις αλλαγές στα μεγέθη των καταχωρητών, υπάρχουν και άλλες αλλαγές, που τις **συναντάμε και σε πραγματικούς επεξεργαστές**:
  - Αμφίδρομη μεταφορά δεδομένων), που απαιτούνται για την αυξημένη πολυπλοκότητα.
  - Ο PC μεταφέρει και διευθύνσεις προς παραμέτρους και όχι μόνο προς εντολές (γιατί, ενώ έχουμε 8bit, η διεύθυνση μιας θέσης μνήμης είναι 16bit, οπότε δε χωράει εντολή + διεύθυνση σε 8bit, αν είχαμε 24bit τότε δε θα γίνονται καλή χρήση γιατί υπάρχουν εντολές που δεν κάνουν πρόσβαση στη μνήμη).
  - Επιπρόσθετο καταχωρητή R (για μείωση των προσβάσεων στη μνήμη).
  - Προσωρινή καταχωρητή TR (δεν είναι προσβάσιμος από το χρήστη, χρησιμοποιείται μόνο από τη CPU).
  - Σημαία Z (zero flag), που τροποποιείται μόνο από συγκεκριμένες εντολές, κυρίως για εντολές αλλαγής ροής εκτέλεσης.

Στάδιο FETCH & DECODE

- Το στάδιο FETCH μοιάζει με το αντίστοιχο του VSC.

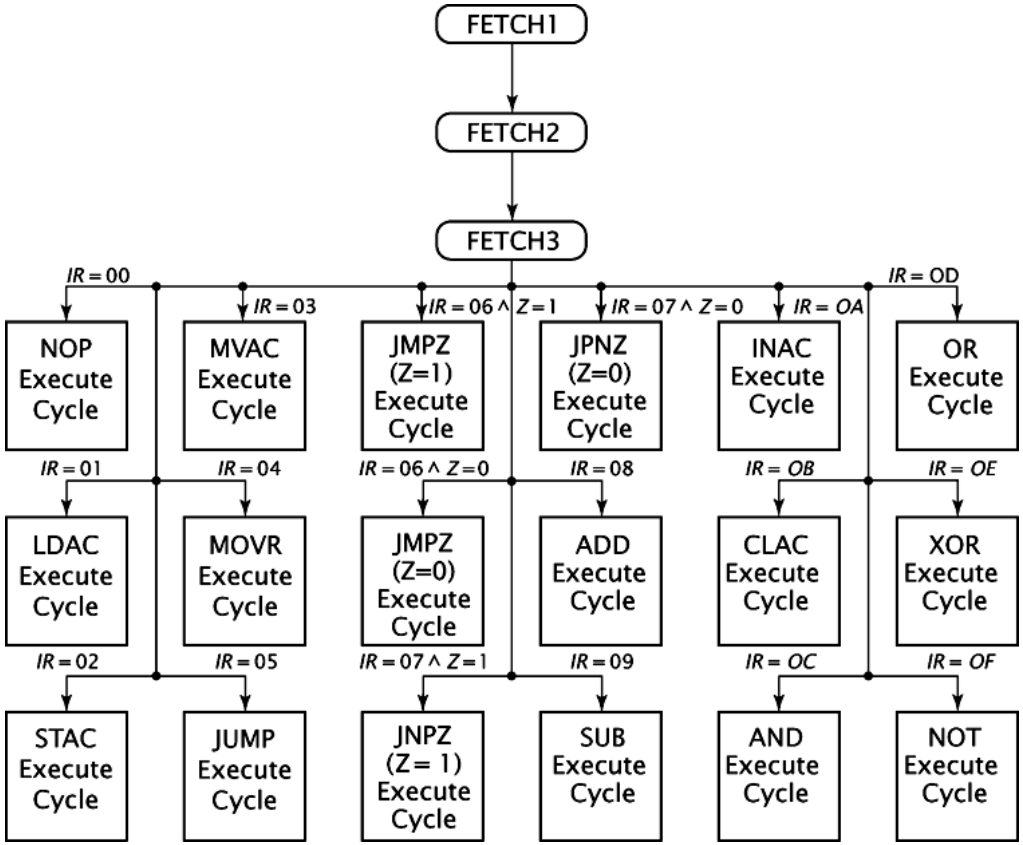
  FETCH1: AR <- PC

  FETCH2: DR <- M, PC <- PC <- PC + 1

  FETCH3: IR <- DR, AR <- PC

- Όλα τα 8bit του DR μεταφέρονται στο IR.
- Το PC μεταφέρεται στο AR, γιατί θα είναι η επόμενη διεύθυνση που θα γίνει πρόσβαση.
- Το στάδιο DECODE είναι το ίδιο.

Διάγραμμα καταστάσεων για Fetch και Decode



Οι εντολές JMPZ, JPNZ για το ίδιο opcode, έχουν δυο διαφορετικά μονοπάτια, αναλόγως της τιμής Z.

Εκτέλεση των εντολών: NOP

- Το τελικό στάδιο, είναι για κάθε εντολή να αναπτυχθεί το διάγραμμα εκτέλεσης.
- **NOP:**
  - Είναι η πιο εύκολη εντολή. Καμία λειτουργία.
  - Μπορεί να γίνει είτε με άμεση μεταφορά στο FETCH1, είτε με τη δημιουργία μιας κατάστασης NOP1 για το στάδιο της εκτέλεσης, κάτι που είναι προτιμητέο, ώστε κάθε εντολή να έχει στάδιο εκτέλεσης.

Εκτέλεση των εντολών: LDAC

- Εντολή πολλών Byte (3 byte).
- Απαιτείται το opcode, το χαμηλό Byte και το υψηλό Byte της διεύθυνσης.
- Κατά το στάδιο της εκτέλεσης, θα πρέπει να πάρει τη διεύθυνση από τη μνήμη, στη συνέχεια να φέρει τα δεδομένα από αυτή τη διεύθυνση και να τα φορτώσει στον AC.
- Ο PC κάθε φορά δείχνει στο επόμενο Byte (που συνήθως είναι εντολή). Αν είναι εντολή με πολλά Byte, τότε το PC δείχνει στο χαμηλό Byte της διεύθυνσης και στην επόμενη αύξηση, δείχνει στο υψηλό Byte της διεύθυνσης.

Εκτέλεση των εντολών: LDAC1

- Αρχικά, πρέπει να βρεθεί η διεύθυνση της παραμέτρου.
- Το PC δείχνει στο χαμηλό Byte της διεύθυνσης.
- Το PC έχει αντιγραφεί στο AR (έχει αυξηθεί στο FETCH3).
- Για το υψηλό Byte, απαιτείται η αύξηση του PC και του AR κατά 1. Θα μπορούσε να γίνει μόνο αύξηση του PC, και μεταφορά στο AR, αλλά αυτό θα απαιτούσε την προσθήκη μιας ακόμη κατάστασης εκτέλεσης.

LDAC1: DR <-M, PC <- PC + 1, AR <- AR + 1

Εκτέλεση των εντολών: LDAC2

- Το επόμενο βήμα είναι η μεταφορά του υψηλού Byte.
- Πρέπει να αποθηκευτεί το χαμηλό Byte στον προσωρινό καταχωρητή (γιατί θα σβηστεί με την εγγραφή του νέου Byte).
- Επίσης, το PC πρέπει να αυξηθεί κατά 1, ώστε να δείχνει στην επόμενη προς εκτέλεση εντολή.

LDAC2: TR <- DR, DR <- M, PC <- PC + 1

Εκτέλεση των εντολών: LDAC3,4,5

- Σε αυτό το στάδιο χρησιμοποιούνται τα 2 byte (υψηλό + χαμηλό) ως διεύθυνση για να ζητηθεί από τη μνήμη η μεταφορά των αντίστοιχων δεδομένων.
- Ο CPU πρέπει να αντιγράψει τη διεύθυνση στο AR.

LDAC3: AR <- DR, TR

- Πρέπει να αντιγραφούν τα δεδομένα από τη μνήμη στο DR.

LDAC4: DR <- M

- Τέλος, αντιγράφονται τα δεδομένα από το DR στο AC.

LDAC5: AC <- DR

Εκτέλεση των εντολών: STAC

- Εντολή πολλών Byte (3 byte).
- Η εντολή αυτή αποθηκεύει το AC στη διεύθυνση μνήμης που ορίζεται από το επόμενο Byte (χαμηλό Byte διεύθυνσης) και το μεθεπόμενο byte (υψηλό Byte διεύθυνσης).
- Αντίστροφη της LDAC.
- Χρησιμοποιεί παρόμοια στάδια:
  - **STAC1: DR <- M, PC <- PC + 1, AR <- AR + 1**
  - **STAC2: R <- DR, DR <- M, PC <- PC + 1**
  - **STAC3: AR <- DR, TR**
  - **SRTAC4: DR <- AC**
  - **STAC5: M <- DR**

Τα STAC3,STAC4 δε μπορούν να συγχωνευτούν, γιατί μοιράζονται τον ίδιο εσωτερικό δίαυλο.

Εκτέλεση των εντολών: MVAC,MOVR

- Η εντολή MVAC μεταφέρει την τιμή από το AC στο R. Απαιτεί μια κατάσταση μόνο:
  - **MVAC1: R <- AC**
- Η εντολή MOVR μεταφέρει την τιμή από το R στο AC. Απαιτεί μια κατάσταση μόνο:
  - **MOVR1: AC <- R**

Εκτέλεση των εντολών: JUMP

- Εντολή πολλών Byte (3 byte).
- Αλλάζει η ροή εκτέλεσης στη διεύθυνση που ακολουθεί στο 2ο και 3ο byte.
- Μεταφέρεται στο DR το πρώτο byte της διεύθυνσης, και αυξάνει το AR κατά 1, για να δείξει στο επόμενο byte.
  - **JUMP1: DR <- M, AR <- AR + 1**
    - Ομοίως με τις εντολές LDAC,STAC μεταφέρεται το 2ο στο TR και το 3ο byte στο DR.
  - **JUMP2: TR <- DR, DR <- M**
    - Στη συνέχεια αντιγράφονται και τα 2 byte στο PC, ώστε να δείχνει στη διεύθυνση της νέας εντολής.
  - **JUMP3: PC <- DR, TR**

Εκτέλεση των εντολών: JMPZ,JPNZ 1 (1/2)

- Εντολές πολλών Byte (3 byte).
- Οι εντολές αυτές μπορούν να τροποποιήσουν τη ροή εκτέλεσης, αναλόγως της σημαίας Z.
- Αν πρόκειται να γίνει αλλαγή ροής εκτέλεσης, τότε το στάδιο της εκτέλεσης εκτελεί τις λειτουργίες της εντολής JUMP.
- Αν δεν πρόκειται να γίνει αλλαγή ροής εκτέλεσης, τότε θα πρέπει να αυξηθεί το PC κατά 2, γιατί τα επόμενα 2 byte είναι η διεύθυνση που θα γίνονταν η αλλαγή αν το επέτρεπε το Z.
- Η διαφορά του JMPZN και του JMPZY είναι ότι τα JMPZY εκτελούνται αν Z=1, ενώ τα ZMPZN αν Z=0.
- Ως προς τα JPNZ αυτά είναι παρόμοια, αλλά χρησιμοποιούνται με τις αντίστροφες συνθήκες. Δηλαδή, τα JPNZY εκτελούνται αν Z=0, ενώ το JPNZN αν Z=0 .

Εκτέλεση των εντολών: JMPZ,JPNZ 1 (2/2)

JMPZY1: DR <- M, AR <- AR + 1

JMPZY2: TR <- DR, DR <- M

JMPZY3:PC <- DR, TR

JMPZN1: PC <- PC + 1

JMPZN2: PC <- PC + 1

JPNZY1: DR <- M, AR <- AR + 1

JPNZY2: TR <- DR, DR <- M

JPNZY3:PC <- DR, TR

JPNZN1: PC <- PC + 1

JPNZN2: PC <- PC + 1

Εκτέλεση των εντολών: υπόλοιπες εντολές 1

- Οι υπόλοιπες εντολές εκτελούνται σε ένα στάδιο.
- Για κάθε εντολή, συμβαίνουν τα εξής:
  - Υπολογίζεται η τιμή, αναλόγως της πράξης, και τοποθετείται στο AC.
  - Αν το αποτέλεσμα είναι 0, τότε το Z γίνεται 1, διαφορετικά γίνεται 0. Αυτό γίνεται ταυτόχρονα για εξοικονόμηση σταδίων.

Εκτέλεση των εντολών: υπόλοιπες εντολές 2

ADD1: AC <- AC + R, IF (AC + R = 0) THEN Z <- 1 ELSE Z <- 0

SUB1: AC <- AC - R, IF (AC - R = 0) THEN Z <- 1 ELSE Z <- 0

INAC1: AC <- AC + R, IF (AC + R = 0) THEN Z <- 1 ELSE Z <- 0

CLAC1: AC <- 0, Z <- 1

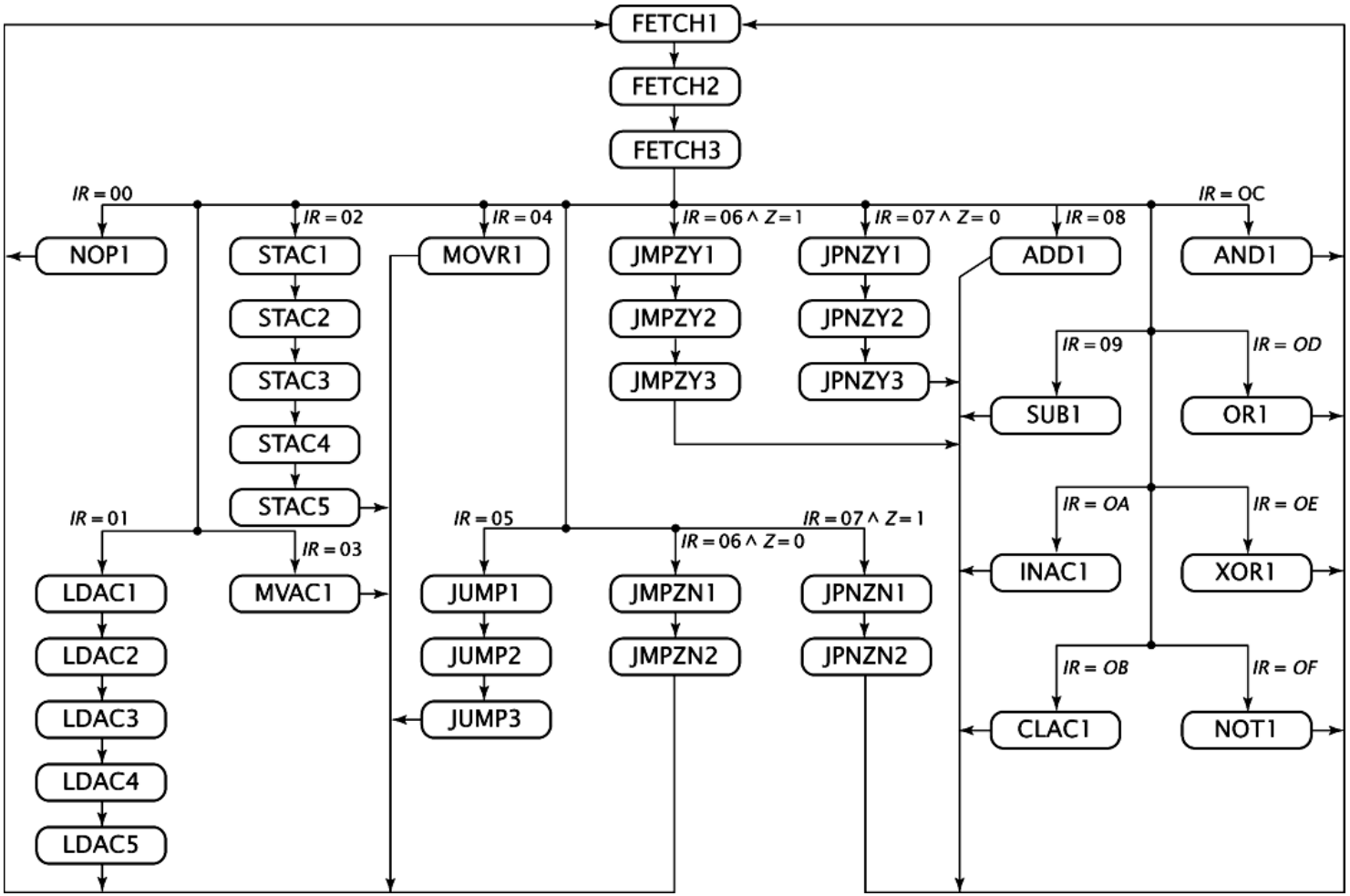
AND1: AC <- AC ^ R, IF (AC ^ R = 0) THEN Z <- 1 ELSE Z <- 0

OR1: AC <- AC | R, IF (AC | R = 0) THEN Z <- 1 ELSE Z <- 0

XOR1: AC <- AC +| R, IF (AC +| R = 0) THEN Z <- 1 ELSE Z <- 0

NOT1: AC <- AC`IF (AC` = 0) THEN Z <- 1 ELSE Z <- 0

Πλήρες διάγραμμα καταστάσεων



Δημιουργία των διαύλων

- Χρησιμοποιείται ένας εσωτερικός δίαυλος για τη μεταφορά δεδομένων.
- Αρχικά, γίνεται η ομαδοποίηση (ταξινόμηση) ως προς τον καταχωρητή που καταλήγουν τα δεδομένα.
- Για να γίνει αυτό, έχουμε συγκεντρώσει όλες τις λειτουργίες και τις εξετάζουμε μια προς μια.
- Η ταξινόμηση βρίσκεται στην επόμενη διαφάνεια.

Ομαδοποίηση ενεργειών 1

AR: AR <- PC; AR <- AR + 1; AR <- DR, TR

PC: PC <- PC + 1; PC <- DR, TR

DR: DR <- M, DR <- AC

IR: IR <- DR

R: R <- AC

TR: TR <- DR

AC: AC <- DR; AC <- R; AC <- AC + R; AC <- R;

AC <- AC + 1; AC <- 0; AC <- AC ^ r; AC <- AC | r;

AC <- AC +| R; AC <- AC`

Z: Z <- 0 (both conditional)

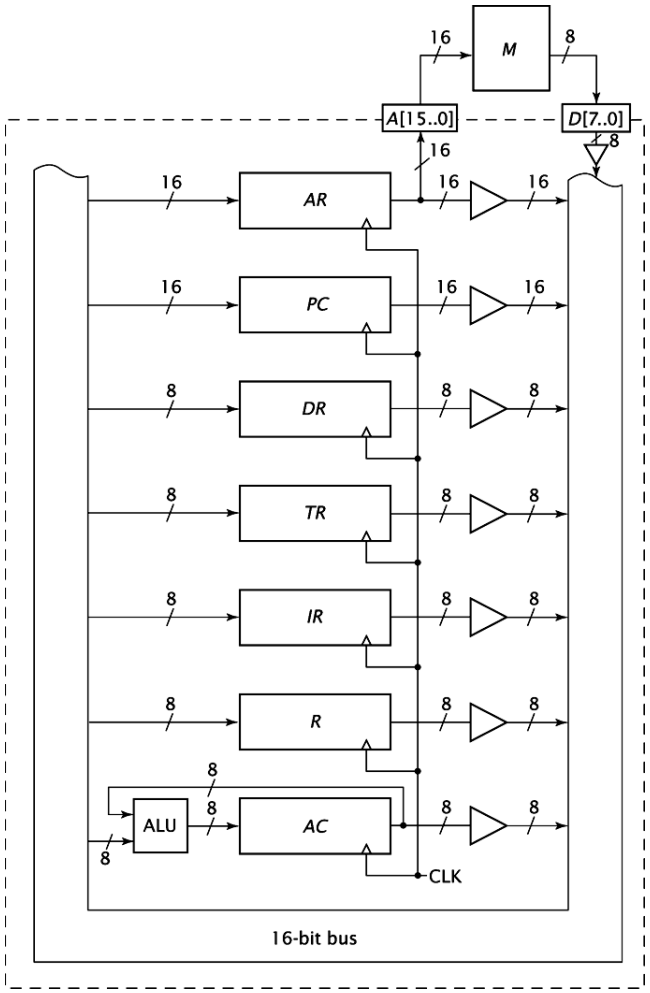
Ομαδοποίηση ενεργειών 2

- Οι καταχωρητές AR και PC, πρέπει να μπορούν να έχουν:
  - παράλληλη φόρτωση.
  - Αύξηση κατά 1.
- Οι καταχωρητές DR,IR,R,TR, πρέπει να μπορούν να έχουν παράλληλη φόρτωση. Μπορεί να χρησιμοποιηθεί είτε ο κοινός δίαυλος, είτε να προστεθούν αποκλειστικές συνδέσεις (αναλόγως το χρονισμό).
- Η ALU χρησιμοποιεί στη μια είσοδο το AC και την τιμή που φέρει ο δίαυλος για τη 2η είσοδο.
- Το AC θα παίρνει τιμή από το ALU.
- Η CPU χρησιμοποιεί την τιμή του AC για το Z.

Ομαδοποίηση ενεργειών 3

- Στο AC χρησιμοποιείται η αύξηση κατά 1. Αυτό μπορεί να γίνει μέσω ειδικής διάταξης στον καταχωρητή AC (όπως σε άλλους καταχωρητές) ή μέσω της ALU (επιλέγεται η ALU, για να μπορεί να υπολογιστεί και το Z).
- Το Z είναι καταχωρητής 1bit.
- Συνδέονται τα pins της διεύθυνσης A[15..0], και των αμφίδρομων δεδομένων D[7..0].
- Η πρώτη έκδοση του διαγράμματος που συνδέει όλα τα στοιχεία φαίνεται στην επόμενη διαφάνεια.

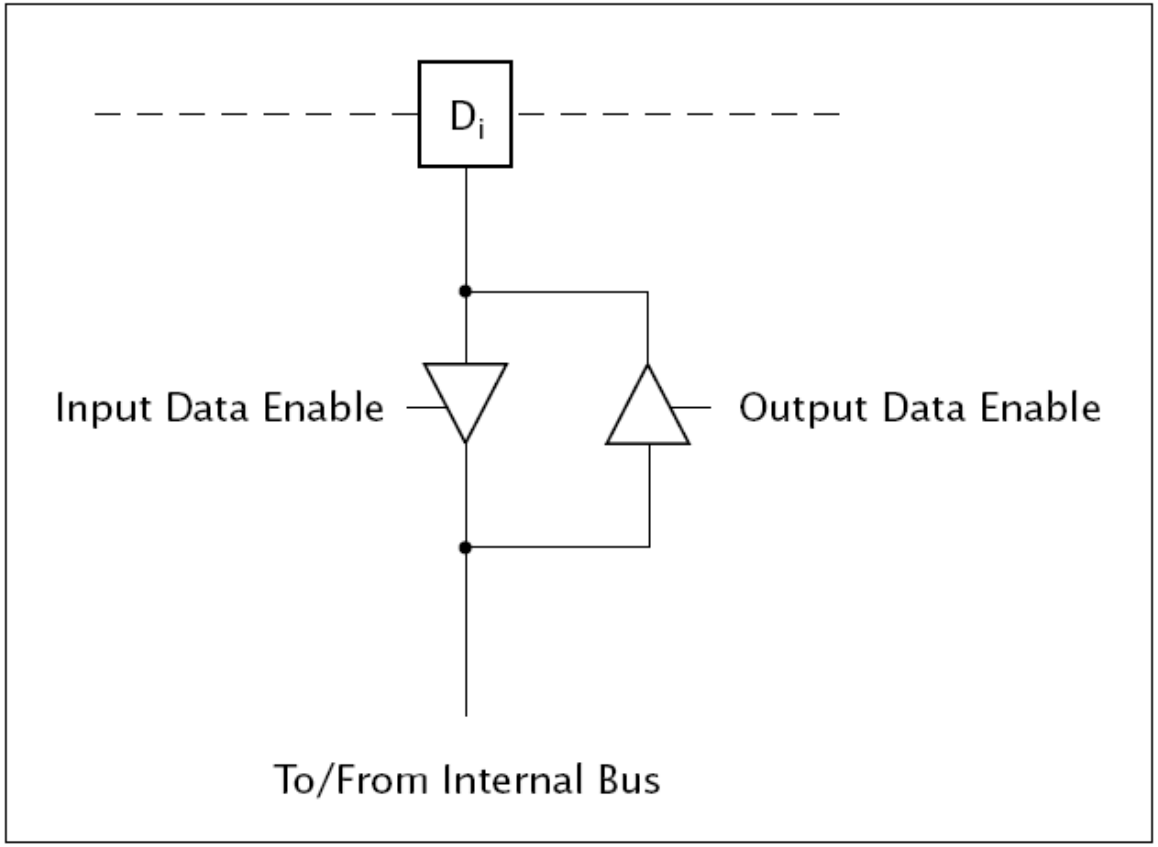
Πρώτη έκδοση κυκλωματικού διαγράμματος



Βελτιώσεις:

- Τα AR,IR δε παρέχουν δεδομένα αλλού, μπορούν να απομακρυνθούν οι συνδέσεις.
- Τα D[7..0] δεν έχουν ακόμη τροποποιηθεί για αμφίδρομη λειτουργία, απαιτούνται buffers για κάθε κατεύθυνση.
- Κάποιοι καταχωρητές δε χρησιμοποιούν το δίαυλο 16bit. Πρέπει να οριστούν ποια bit του διαύλου συνδέονται σε ποιο καταχωρητή.
- Ο καταχωρητής Z δεν έχει συνδεθεί.

Γενικό διάγραμμα αμφίδρομης λειτουργίας



Δεν επιτρέπεται ποτέ να είναι ενεργοποιημένα και τα δυο data enable.

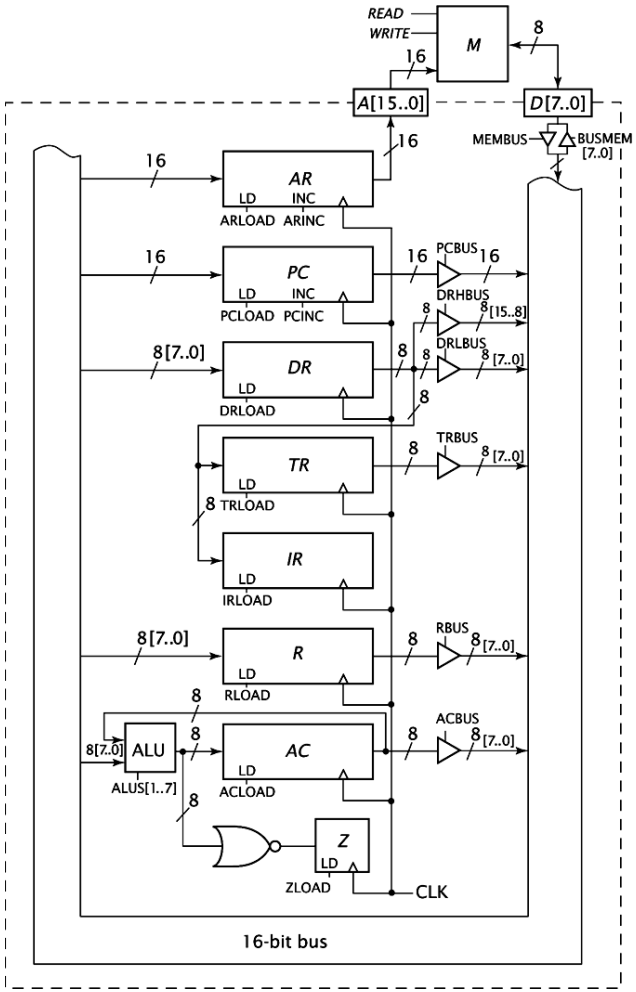
Μεταφορές από το δίαυλο (16 bit) σε καταχωρητές (16 bit και 8 bit) 1

- Το AR,PC είναι 16bit, οπότε τα bit του διαύλου συνδέονται άμεσα σε αυτούς τους καταχωρητές.
- Οι υπόλοιποι καταχωρητές είναι 8bit, οπότε μια υλοποίηση είναι η σύνδεση τους με τα bit [7..0]. Όμως, μπορεί να βελτιωθεί:
  - Κατά το FETCH3 απαιτείται η μεταφορά 2 στοιχείων στους καταχωρητές (**IR <- DR, AR <- PC**). ή θα προστεθεί ένα ακόμη στάδιο, ή μια μεταφορά θα γίνει εκτός διαύλου. Επιλέγουμε να συνδέσουμε το IR στο DR, επειδή το IR μόνο από το DR δέχεται δεδομένα. Ως εκ τούτου το IR μπορεί να αποσυνδεθεί από το κοινό δίαυλο.

Μεταφορές από το δίαυλο (16 bit) σε καταχωρητές (16 bit και 8 bit) 2

- Κατά το LDAC2 (**TR <- DR, DR <- M**), καθώς και σε άλλα στάδια, απαιτείται η ταυτόχρονη μεταφορά 2 στοιχείων από το δίαυλο. Επίσης, το TR δέχεται δεδομένα μόνο από το DR, οπότε μπορεί να υπάρχει μια αποκλειστική σύνδεση του TR με το DR. Το TR μπορεί να αποσυνδεθεί λοιπόν από το δίαυλο.
- Στο LDAC3 (**AR <- DR, TR**), πρέπει να γράψουμε ταυτόχρονα 2 8bit λέξεις στο 16bit δίαυλο. Μπορούμε να τοποθετήσουμε το DR στα [15..8], αλλά θα δημιουργηθεί πρόβλημα σε επόμενες εντολές (π.χ. **LDAC5: AC <- DR**). Μπορούμε να χρησιμοποιήσουμε απομονωτές και να στέλνουμε το DR είτε στα [15..8], είτε στα [7..0] αναλόγως της λειτουργίας.
- Το Z υπολογίζεται μόνο όταν υπάρχει μια λειτουργία ALU. Μπορεί να χρησιμοποιηθεί μια πράξη NOR. Για αυτό το λόγο το CLEAR και INC στο AC γίνεται μέσω της ALU, για να ενημερώνεται άμεσα και το Z.

Τελική έκδοση κυκλωματικού διαγράμματος



Βελτιώσεις:

- Αμφίδρομο D[7..0].
- Υπολογισμός Z.
- AR, IR χωρίς σύνδεση στο δίαυλο.
- DR με σύνδεση είτε [15..8] είτε [7..0].
- TR δεδομένα από DR.
- IR δεδομένα από DR.

Σχεδιασμός της ALU 1

- Όλα τα δεδομένα που καταλήγουν στο AC πρέπει να διέλθουν μέσω της ALU.
- Συνοψίζονται οι ενέργειες που τροποποιούν το AC.

LDAC5: AC <- DR
MOVR1: AC <- R
ADD1: AC <- AC + R
SUB1: AC <- AC - R
INAC1: AC <- AC + 1
CLAC1: AC <- 0
AND1: AC <- AC ^ R
OR1: AC <- AC | R
XOR1: AC <- AC +| R
NOT1: AC <- AC

Σχεδιασμός της ALU 2

- Οι ενέργειες αυτές θα πρέπει να χωριστούν σε αριθμητικές και λογικές λειτουργίες. Θα σχεδιαστεί πρώτα το αριθμητικό κομμάτι της ALU και μετά το λογικό.
- Κατά τη λειτουργία της ALU θα υπολογίζεται μια τιμή τόσο από το αριθμητικό κομμάτι τόσο από το λογικό.
- Με έναν πολυπλέκτη θα γίνεται η επιλογή του κατάλληλου αποτελέσματος και θα συνδέεται στην έξοδο.

Σχεδιασμός της ALU: Αριθμητικές λειτουργίες 14

- Ξεχωρίζονται οι αριθμητικές λειτουργίες.

  LDAC5: AC <- DR

  MOVR1: AC <- R

  ADD1: AC <- AC + R

  SUB1: AC <- AC - R

  INAC1: AC <- AC + 1

  CLAC1: AC <- 0

- Το DR έχει ως πηγή το δίαυλο (BUS).
- Το R έχει ως πηγή το δίαυλο (BUS).

Σχεδιασμός της ALU: Αριθμητικές λειτουργίες 2

- Τροποποιούνται οι αριθμητικές λειτουργίες, ώστε να χρησιμοποιούν έναν παράλληλο αθροιστή ριπής μαζί με είσοδο κρατουμένου.
- Προσθέτονται: α είσοδος, β είσοδος και κρατούμενο.

  LDAC5: AC<-0 + BUS + 0

  MOVR1: AC <- 0 + BUS + 0

  ADD1: AC <- AC + BUS + 0

  SUB1: AC <- AC +BUS' + 1 (πρόσθεση με συμπλήρωμα ως προς 2)

  INAC1: AC <- AC + 0 + 1

  CLAC1: AC <- 0 + 0 + 0
- Παρατηρούμε ότι:
  - Η α είσοδος είναι 0 ή AC (χρήση πολυπλέκτη).
  - Η β είσοδος είναι BUS ή BUS' ή 0 (χρήση πολυπλέκτη).
  - Το κρατούμενο είναι 0 ή 1 (σήμα από τη μονάδα ελέγχου).

Σχεδιασμός της ALU : Λογικές λειτουργίες 1

- Ξεχωρίζονται οι λογικές λειτουργίες:

  AND1: AC <- AC ^ R

  OR1: AC <- AC | R

  XOR1: AC <- AC +| R

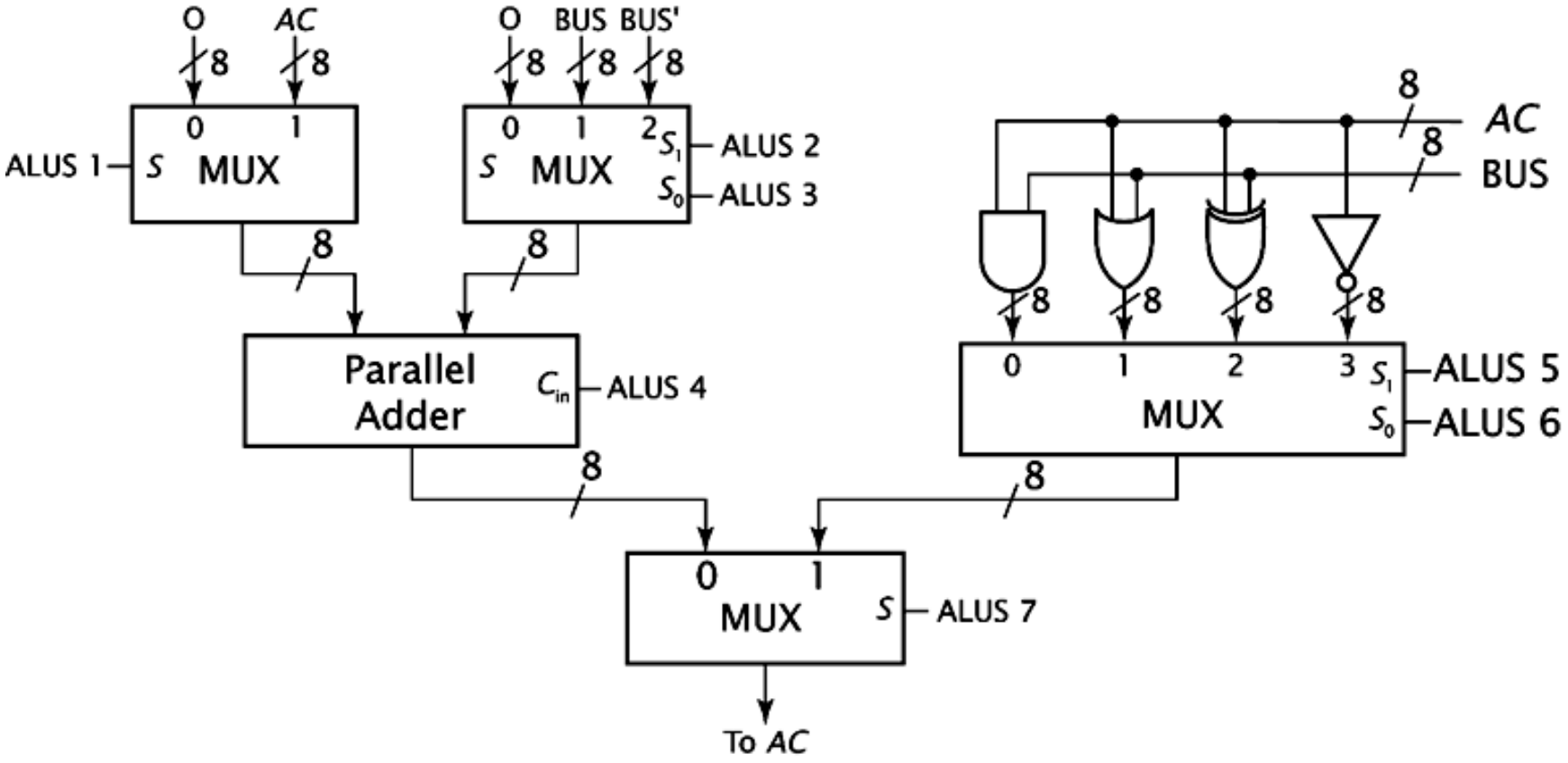
  NOT1: AC <- AC

- Το AC έχει ως πηγή το δίαυλο (BUS).
- Το R έχει ως πηγή το δίαυλο (BUS).

Σχεδιασμός της ALU: Λογικές λειτουργίες 2

- Υπάρχουν 4 λειτουργίες.
- Κάθε λειτουργία εξάγει ένα αποτέλεσμα 8bit.
- Μπορεί να χρησιμοποιηθεί ένας πολυπλέκτης 8bit 4 σε 1, ώστε να επιλέγεται κάθε φορά η κατάλληλη έξοδος.
- Το μειονέκτημα είναι ότι ξοδεύεται ενέργεια γιατί συνεχώς υπολογίζονται όλα τα αποτελέσματα, αλλά επιλέγεται ένα κάθε φορά. Επίσης, ακόμη και όταν δε λειτουργεί η ALU, όλα τα κυκλώματα της λειτουργούν.
  - Η 1η είσοδος του πολυπλέκτη (00): **AND1: AC <- AC ^ BUS**
  - Η 2η είσοδος του πολυπλέκτη (01): **OR1: AC <- AC | BUS**
  - Η 3η είσοδος του πολυπλέκτη (10): **XOR1: AC <- AC +| BUS**
  - Η 4η είσοδος του πολυπλέκτη (11): **NOT1: AC <- AC'**

Σχεδιασμός της ALU 3



Σχεδιασμός της μονάδας ελέγχου 1

- Υπάρχει αυξημένη πολυπλοκότητα και έτσι χρησιμοποιούνται δυο καταχωρητές μέσα στη CU (αντί για ένα που είχε η VSC).
  - Ο πρώτος καταχωρητής φέρει τον 8bit op-code.
  - Ο δεύτερος καταχωρητής φέρει τον αριθμό του σταδίου (κατάσταση) που βρίσκεται κάθε φορά ο επεξεργαστής.
- Ο καταχωρητής που φέρει το op-code της τρέχουσας εντολής είναι εύκολο να σχεδιαστεί. Όλα τα opcodes είναι της μορφής 0000XXXX, αρκεί να γίνει αποκωδικοποίηση των τελευταίων 4 bit. Δηλαδή, ως είσοδο στον αποκωδικοποιητή εισάγονται τα IR[3..0].

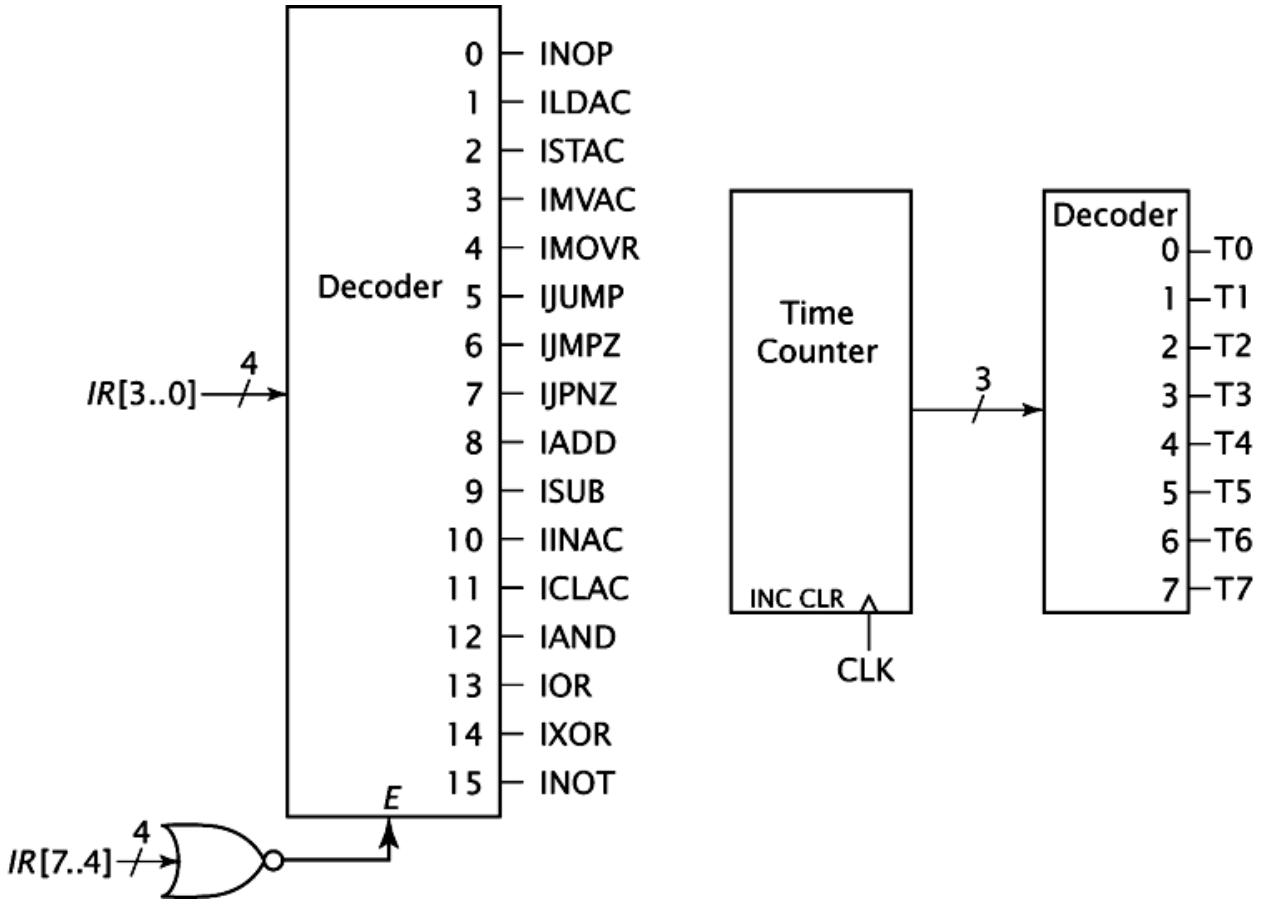
Σχεδιασμός της μονάδας ελέγχου 2

- Προκειμένου να ενεργοποιηθεί η αποκωδικοποίηση, αρκεί να γίνει NOR των 4 υψηλών bit IR[7..4], ώστε να υπάρξει αποτέλεσμα 1. Αυτό το αποτέλεσμα τοποθετείται στην είσοδο E (enable) του αποκωδικοποιητή.
- Ο μετρητής θα πρέπει να μπορεί να αυξηθεί κατά 1 και να μπορεί να καθαριστεί (δε χρειάζεται παράλληλη φόρτωση).

Τα opcodes και το κυκλωματικό διάγραμμα (1/2)

Instruction	Instruction Code
NOP	0000 0000
LDAC	0000 0001 Γ
STAC	0000 0010 Γ
MVAC	0000 0011
MOVR	0000 0100
JUMP	0000 0101 Γ
JMPZ	0000 0110 Γ
JPNZ	0000 0111 Γ
ADD	0000 1000
SUB	0000 1001
INAC	0000 1010
CLAC	0000 1011
AND	0000 1100
OR	0000 1101
XOR	0000 1110
NOT	0000 1111

Τα opcodes και το κυκλωματικό διάγραμμα (2/2)



Κωδικοποίηση καταστάσεων

- Η FETCH δε χρησιμοποιεί κάποια τιμή από τον αποκωδικοποιητή εντολών.
- Το FETCH1 για τους ίδιους λόγους με το VSC το τοποθετούμε στο T0, ενώ τα FETCH2,FETCH3 στα T1,T2 αντίστοιχα.
- Τοποθετούμε τα υπόλοιπα στάδια των εντολών στις καταστάσεις T3,T4.
- Με τη λογική πράξη **AND** ανάμεσα στην έξοδο του αποκωδικοποιητή και του χρονισμού T0-T7,δημιουργούμε τα κατάλληλα σήματα ελέγχου.
- π.χ. **LDAC1 = ILDAC ^ T3** , **LDAC2 = IDAC ^ T4**

Δημιουργία Σημάτων

- Μετά τη δημιουργία των καταστάσεων, πρέπει να δημιουργηθούν τα σήματα CLR,INC για τον μετρητή χρονισμού (T0,..,T7).
- **Σήμα CLR**. Ο μετρητής μηδενίζεται κάθε φορά στο τελευταίο στάδιο εκτέλεσης της κάθε εντολής. Για αυτό το λόγο, θα κάνουμε την πράξη OR με την τελευταία κατάσταση εκτέλεσης.
- **Σήμα INC**. Το σήμα INC πρέπει να είναι ενεργό σε κάθε άλλη στιγμή, εκτός από το τελευταίο στάδιο εκτέλεσης. Για αυτό το λόγο, θα κάνουμε πράξη OR με όλες τις υπόλοιπες καταστάσεις (εκτός από την τελευταία).

Διάγραμμα καταστάσεων

State	Function	State	Function
FETCH1	T0	JMPZY1	IJMPZ ^ Z ^ T3
FETCH2	T1	JMPZY2	IJMPZ ^ Z ^ T4
FETCH3	T2	JMPZY3	IJMPZ ^ Z ^ T5
NOP1	INOP ^ T3	JMPZN1	IJMPZ ^ Z' ^ T3
LDAC1	ILDAC ^ T3	JMPZN2	IJMPZ ^ Z' ^ T4
LDAC2	ILDAC ^ T4	JPNZY1	IJPNZ ^ Z ^ T3
LDAC3	ILDAC ^ T5	JPNZY2	IJPNZ ^ Z ^ T4
LDAC4	ILDAC ^ T6	JPNZY3	IJPNZ ^ Z ^ T5
LDAC5	ILDAC ^ T7	JPNZN1	IJPNZ ^ Z' ^ T3
STAC1	ISTAC ^ T3	JPNZN2	IJPNZ ^ Z' ^ T4
STAC2	ISTAC ^ T4	ADD1	IADD ^ T3
STAC3	ISTAC ^ T5	SUB1	ISUB ^ T3
STAC4	ISTAC ^ T6	INAC1	IINAC ^ T3
STAC5	ISTAC ^ T7	CLAC1	ICLAC ^ T3
MVAC1	IMVAC ^ T3	AND1	IAND ^ T3
MOVR1	IMOVR ^ T3	OR1	IOR ^ T3
JUMP1	IJUMP ^ T3	XOR1	IXOR ^ T3
JUMP2	IJUMP ^ T4	NOT1	INOT ^ T3
JUMP3	IJUMP ^ T5		

Σήματα ALU 1 (1/2)

Τα σήματα της ALU (ALUS1 – ALUS7) δημιουργούνται με παρόμοιο τρόπο με το VSC.

- ALUS1 (επιλογή 0 ή AC) = ADD1 **OR** SUB1 **OR** INAC1.
- ALUS2 (επιλογή BUS' ή BUS ή 0)=SUB1.
- ALUS3 (επιλογή BUS' ή BUS ή 0) =LDAC5 **OR** MOVR1 **OR** ADD1.
- ALUS4 (επιλογή Cin)= SUB1 **OR** INAC1.

ADD1: $AC \leftarrow AC + BUS$
SUB1: $AC \leftarrow AC - BUS$
INAC1: $AC \leftarrow AC + 1$

SUB1: $AC \leftarrow AC + BUS' + 1$

Σήματα ALU 1 (2/2)

- Τα σήματα της ALU (ALUS1 – ALUS7) δημιουργούνται με παρόμοιο τρόπο με το VSC.
- ALUS1 (επιλογή 0 ή AC) = ADD1 **OR** SUB1 **OR** INAC1.
- ALUS2 (επιλογή BUS' ή BUS ή 0)=SUB1.
- ALUS3 (επιλογή BUS' ή BUS ή 0) =LDAC5 **OR** MOVR1 **OR** ADD1.
- ALUS4 (επιλογή Cin)= SUB1 **OR** INAC1.

LDAC5: $AC \leftarrow 0 + BUS + 0$
MOVR1: $AC \leftarrow 0 + BUS + 0$
ADD1: $AC \leftarrow AC + BUS + 0$

SUB1: $AC \leftarrow AC + BUS' + 1$
INAC1: $AC \leftarrow AC + 0 + 1$

Σήματα ALU 2

- ALUS5 (επιλογή λογικής πράξης) =XOR1 **OR** NOT1.
- ALUS6 (επιλογή λογικής πράξης) =OR1 **OR** NOT1.
- ALUS7 (επιλογή αποτελέσματος, αν 0 τότε αριθμητικό αποτέλεσμα, αν 1 τότε λογικό) =AND1 **OR** OR1 **OR** XOR1 **OR** NOT1.

XOR1: $AC \leftarrow AC \oplus R$

NOT1: $AC \leftarrow AC'$

OR1: $AC \leftarrow AC \vee R$ NOT1: $AC \leftarrow AC'$

Για να βγάλουμε τα σήματα, κοιτάμε:

1) το κυκλωματικό διάγραμμα της ALU.

2) τις καταστάσεις της ALU.

Σήματα απομονωτών και διαύλου

Signal	Value
PCBUS	FETCH1 ∨ FETCH3
DRHBUS	LDAC3 ∨ STAC3 ∨ JUMP3 ∨ JMPZY3 ∨ JPNZY3
DRLBUS	LDAC5 ∨ STAC5
TRBUS	LDAC3 ∨ STAC3 ∨ JUMP3 ∨ JMPZY3 ∨ JPNZY3
RBUS	MOVR1 ∨ ADD1 ∨ SUB1 ∨ AND1 ∨ OR1 ∨ XOR1
ACBUS	STAC4 ∨ MVAC1
MEMBUS	FETCH2 ∨ LDAC1 ∨ LDAC2 ∨ LDAC4 ∨ STAC1 ∨ STAC2 ∨ JUMP1 ∨ JUMP2 ∨ JMPZY1 ∨ JMPZY2 ∨ JPNZY1 ∨ JPNZY2
BUSMEM	STAC5
ARLOAD	FETCH1 ∨ FETCH3 ∨ LDAC3 ∨ STAC3
ARINC	LDAC1 ∨ STAC1 ∨ JUMP1 ∨ JMPZY1 ∨ JPNZY1

Με παρόμοιο τρόπο, δημιουργούνται και τα υπόλοιπα σήματα.

Επιβεβαίωση Σχεδίασης

- Απαιτείται να ετοιμαστεί ένα ίχνος εκτέλεσης εντολών, και να εξεταστούν όλα τα μονοπάτια εκτέλεσης.

- Μπορεί να χρησιμοποιηθεί ο προσομοιωτής RSCPU.

Επεκτάσεις / Βελτιώσεις 1

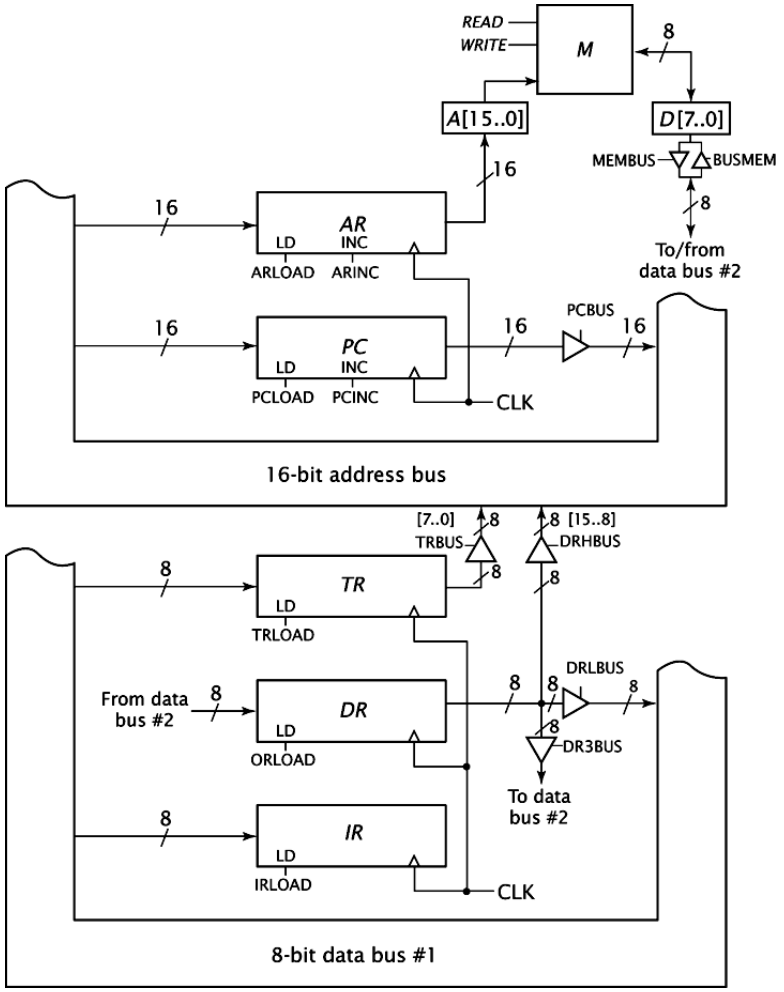
Ο επεξεργαστής μπορεί να επεκταθεί, ώστε να πλησιάσει του πραγματικούς επεξεργαστές.

- Προσθήκη περισσότερων καταχωρητών και κρυφής μνήμης (μείωση εξωτερικού αριθμού προσβάσεων).
- Περισσότεροι καταχωρητές, επιτρέπουν και την εκτέλεση ρουτίνων. Θα μπορούσε η επικοινωνία ανάμεσα στο κυρίως πρόγραμμα και την ρουτίνα, να γίνεται μέσω καταχωρητών και όχι μέσω της κύριας μνήμης (παράμετροι κλήσης, ενδιάμεσα αποτελέσματα, τελικό αποτέλεσμα).

Επεξεργαστές Intel & καταχωρητές

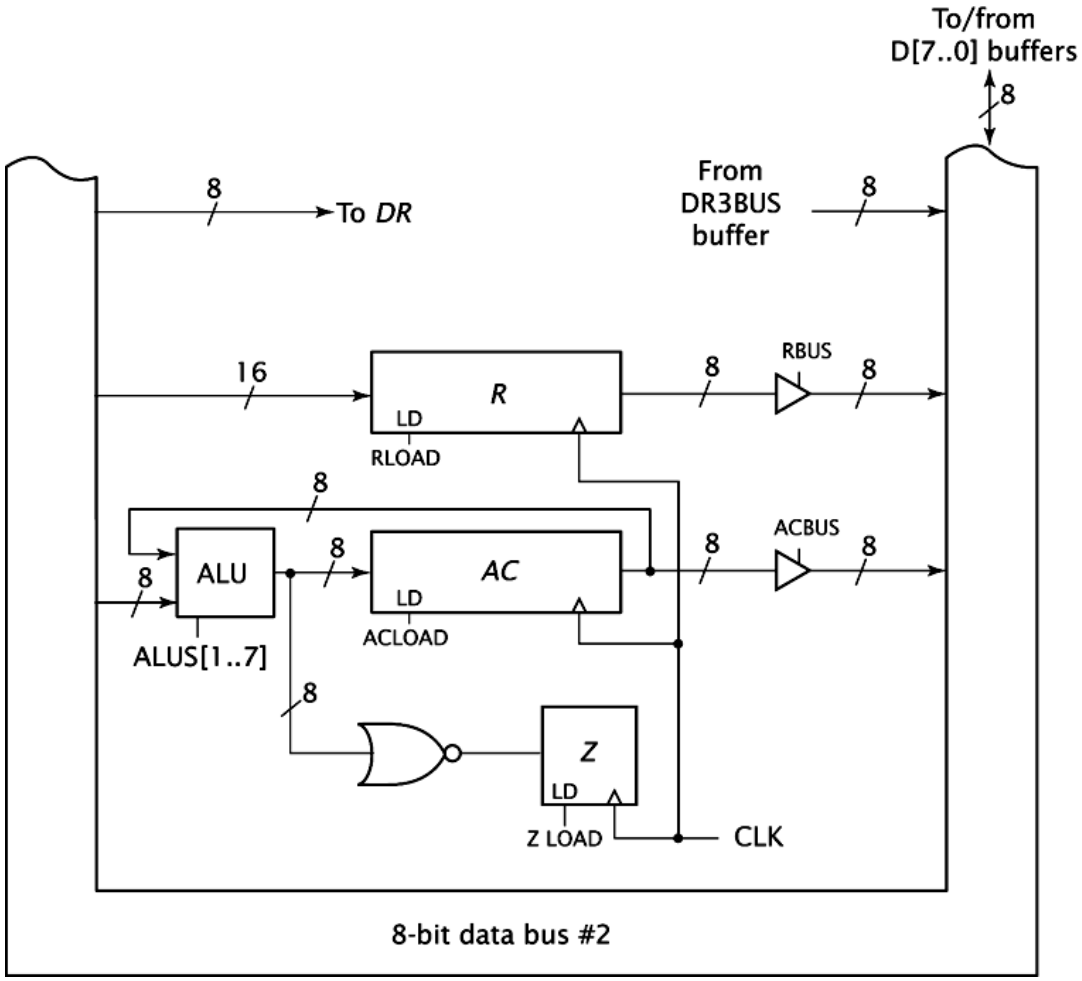
- 4004, καθόλου καταχωρητές.
- 8008, 8080, 8085 έξι καταχωρητές και ένα ACC.
- 8086, 80286, 80386, 80486 8 καταχωρητές.
- Pentium 8 καταχωρητές των 32bit,32KB Cache.
- Itanium 128 καταχωρητές ακεραίων, 128 καταχωρητές πραγματικών, 4MB Cache.

Επεκτάσεις / Βελτιώσεις 2



- Πολλαπλοί δίαυλοι, ώστε να επιτρέπονται πολλαπλές παράλληλες μεταφορές.
- Επίσης, δεν υπάρχει ανάγκη για αποκλειστικές συνδέσεις (π.χ. TR).

Επεκτάσεις / Βελτιώσεις 2 (συνέχεια)



Επεκτάσεις / Βελτιώσεις 3

- Χρήση διασωλήνωσης:

Στον επεξεργαστή που σχεδιάσαμε, πάντα βρίσκονταν μόνο σε μια κατάσταση (οι άλλες καταστάσεις δεν έκαναν τίποτα χρήσιμο). Κατά τη διασωλήνωση, όλα τα στάδια χρησιμοποιούνται, και έτσι ενώ κάτι βρίσκεται στο FETCH, μια άλλη εντολή είναι στην κατάσταση DECODE, και μια άλλη είναι στην κατάσταση EXECUTE.

  - Αυξάνεται ο ρυθμός απόδοσης.
  - Η κάθε εντολή απαιτεί το ίδιο χρόνο εκτέλεσης.
  - Η διασωλήνωση αναλύεται σε άλλη διάλεξη.

Επεκτάσεις / Βελτιώσεις 4

- Μεγαλύτερα σετ εντολών. Ο επεξεργαστής που σχεδιάσαμε έχει μόνο κάτι απλές εντολές.

π.χ. αν ο επεξεργαστής μας κάνει μόνο την πράξη συμπλήρωμα ως προς 2 και την πράξη AND, τότε για τη λογική πράξη A OR B θα έπρεπε να έχουμε:

  - Συμπλήρωμα ως προς 2 του A
  - Συμπλήρωμα ως προς 2 του A
  - Πράξη AND A και B

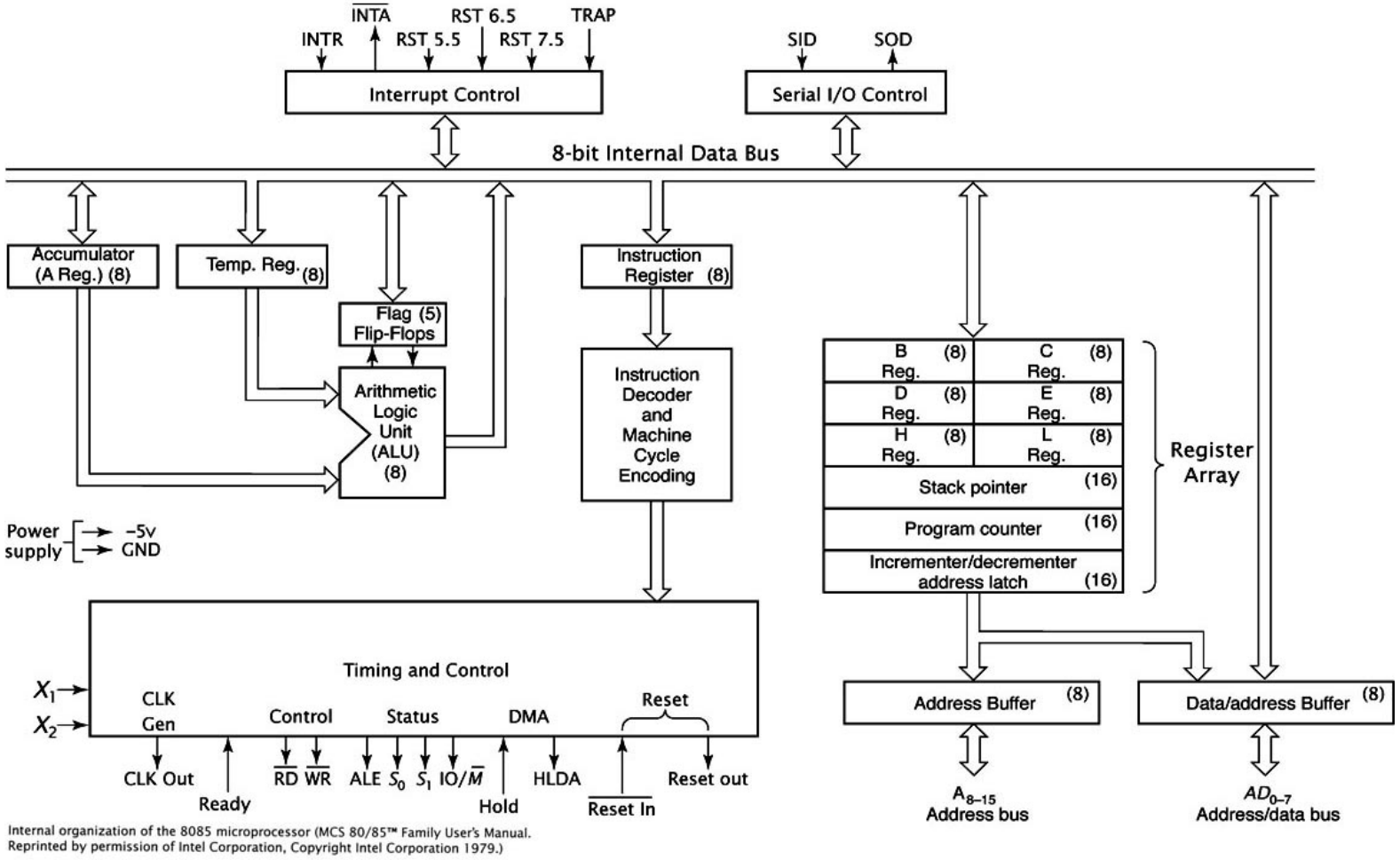
Συμπλήρωμα ως προς 2 του αποτελέσματος:

- Αν αντιθέτως υποστηρίζονταν η πράξη OR τότε θα είχαμε μόνο μια εντολή.
- Αν υπάρχουν πολλές εντολές, τότε γίνεται πολύπλοκη η αποκωδικοποίηση.

Επεκτάσεις / Βελτιώσεις 5

- Υπορουτίνες και διακοπές. Για να υποστηρίζονται υπορουτίνες πρέπει να υπάρχει ένας δείκτης στοίβας, ώστε κάθε φορά να αποθηκεύεται η διεύθυνση επιστροφής από τη ρουτίνα.
- Οι διακοπές είναι σήματα από συσκευές Ε/Ε για να ενημερώσουν τον επεξεργαστή για κάποιο θέμα. Οι διακοπές επιτρέπουν τον επεξεργαστή να διακόψει προσωρινά την εκτέλεση και να εκτελέσει ένα άλλο κομμάτι κώδικα που ονομάζεται “κώδικας ή συνάρτηση εξυπηρέτησης διακοπής”.
- Οι διακοπές αναλύονται σε άλλη διάλεξη.

Ο επεξεργαστής 8085 (1/2)



Internal organization of the 8085 microprocessor (MCS 80/85™ Family User's Manual. Reprinted by permission of Intel Corporation, Copyright Intel Corporation 1979.)

Ο επεξεργαστής 8085 (2/2)

- Ο επεξεργαστής υποστηρίζει διακοπές και I/O.
- Το ALU εκτελεί εντολές αριθμητικές, λογικές και ολίσθησης.
- Υπάρχουν οι καταχωρητές: A, B, C, D, E, H, L, SP, Flags, IR, PC, TR, AR.
- Χρησιμοποιούνται απομονωτές τριών καταστάσεων.
- Υπάρχει η μονάδα ελέγχου που δημιουργεί όλα τα σήματα χρονισμού.
- Η αποκωδικοποίηση της εντολής δημιουργεί όλα τα σήματα κατάστασης που συνδέονται στο block χρονισμού.
- Οι διακοπές ελέγχουν αν είναι ενεργοποιημένες και στέλνουν σήμα στη μονάδα ελέγχου.

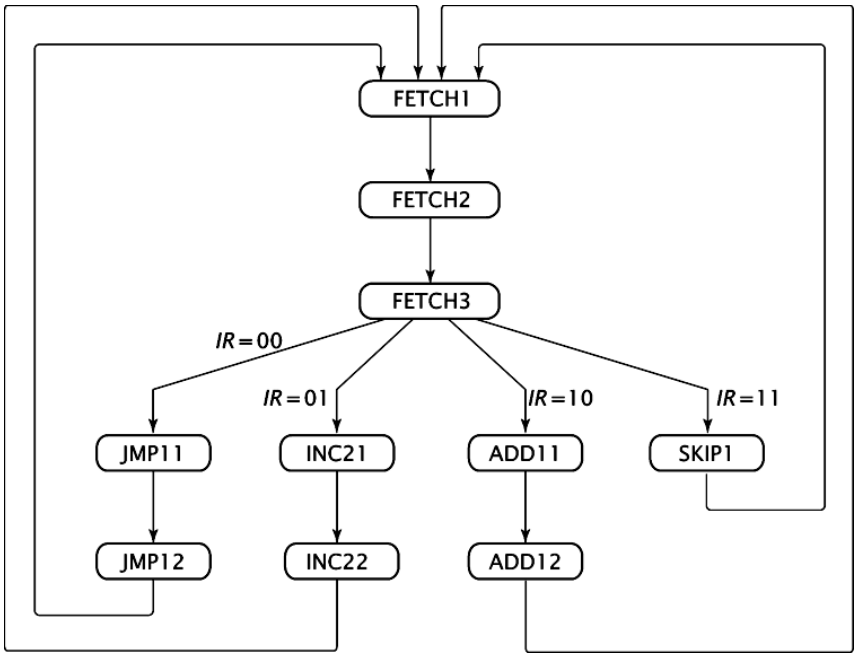
Σύνοψη Σχεδιασμού CPU

- Δημιουργείται το ISA του επεξεργαστή (εντολές και ο αριθμός - ονόματα των καταχωρητών).
- Δημιουργείται μετά το διάγραμμα καταστάσεων και ορίζονται οι μικρο-λειτουργίες για κάθε εντολή.
- Στη συνέχεια αναπτύσσεται η περιγραφή υλικού (RTL).
- Η CPU έχει τρία βασικά τμήματα:
  - Αρχείο καταχωρητών (ορατοί και μη).
  - ALU.
  - Μονάδα ελέγχου.
- Οι μικρο-λειτουργίες ορίζουν το τι πρέπει να γίνει και πότε. Χρησιμοποιούν μόνο συνδυαστική λογική γιατί πρέπει να ολοκληρώνονται σε 1 περίοδο ρολογιού.
- Η μονάδα ελέγχου δημιουργεί πληθώρα σημάτων για κάθε στοιχείο πάνω στο chip .

Πρόβλημα 1

Ένας επεξεργαστής παρόμοιος με το VSC, έχει το παρακάτω σετ εντολών και το διάγραμμα καταστάσεων. Να δώσετε το RTL.

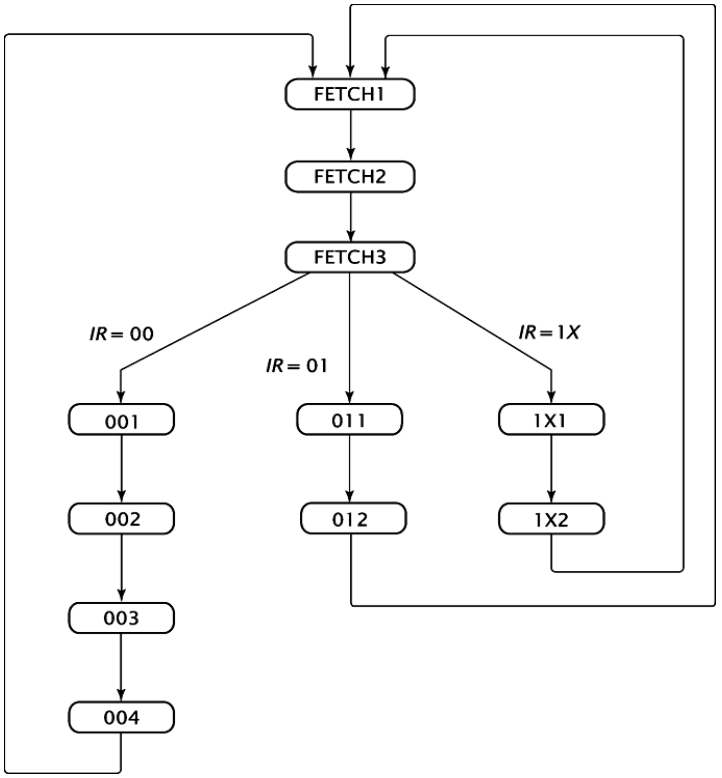
Instruction	Instruction Code	Operation
JMP1	00AAAAAA	*PC*←AAAAAA + 1
INC2	01XXXXXX	*AC*←*AC* + 2
ADD1	10AAAAAA	*AC*←*AC* + *M*[AAAAAA] + 1
SKIP	11XXXXXX	*PC*←*PC* + 1



Πρόβλημα 2

Ένας επεξεργαστής παρόμοιος με το VSC έχει το παρακάτω διάγραμμα καταστάσεων, και το RTL κώδικα. Να δώσετε το instruction set.

FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR\ [7..6]$, $AR \leftarrow DR[5..0]$
001: $DR \leftarrow M$, $AR \leftarrow AR + 1$
002: $AC \leftarrow AC + DR$
003: $DR \leftarrow M$
004: $AC \leftarrow AC + DR$
011: $DR \leftarrow M$, $PC \leftarrow PC + 1$
012: $AC \leftarrow AC \wedge DR$
1X1: $AC \leftarrow AC + 1$, $DR \leftarrow M$
1X2: $AC \leftarrow AC \wedge DR$

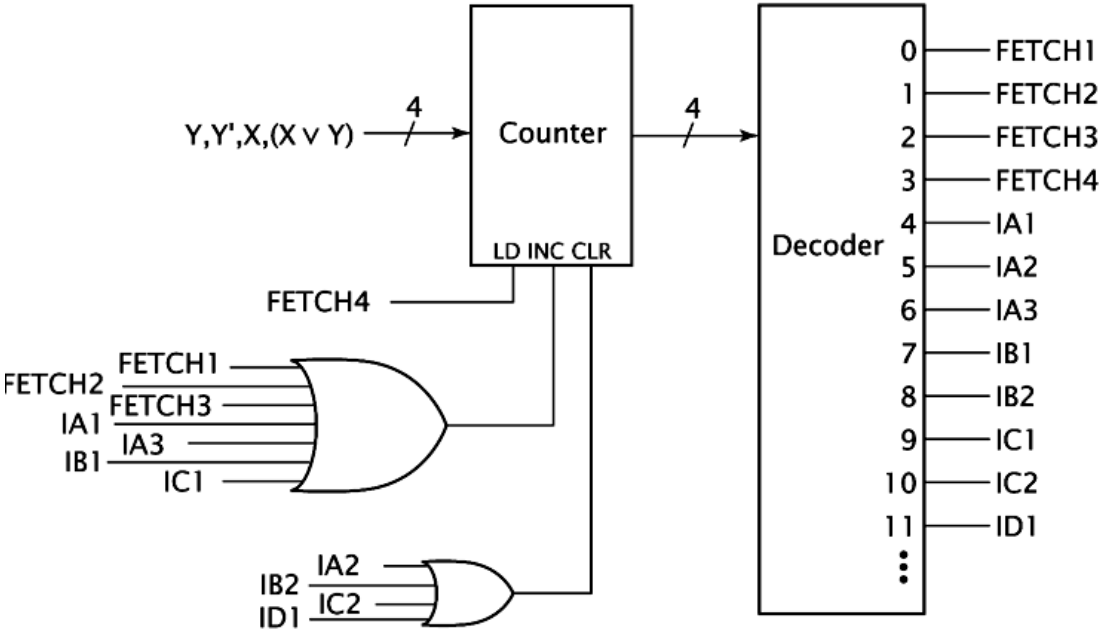


Πρόβλημα 3

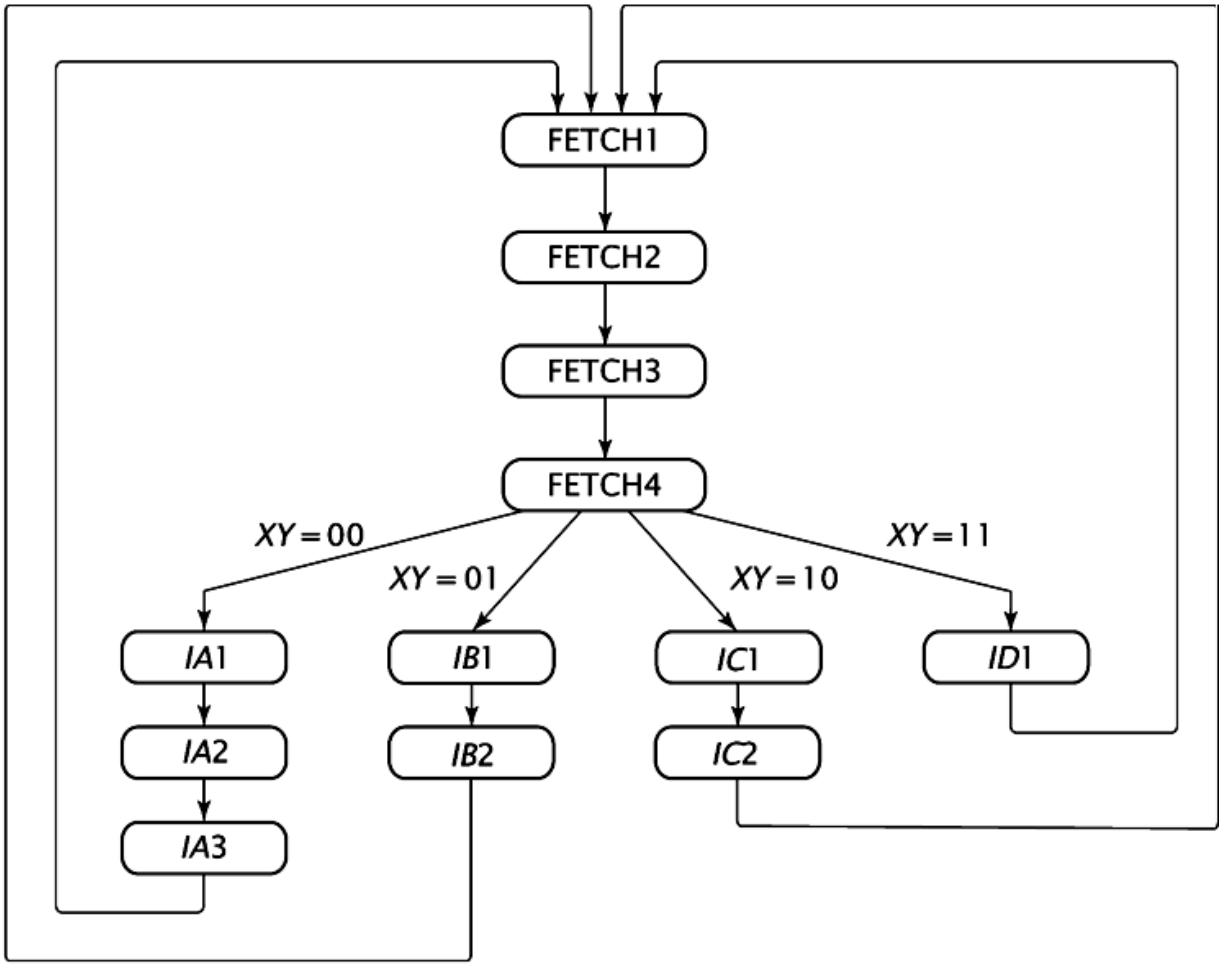
- Να δημιουργήσετε τη μονάδα ελέγχου για το πρόβλημα 2.

Πρόβλημα 4 (1ο μέρος)

- Η παρακάτω μονάδα ελέγχου υποτίθεται ότι υλοποιεί το διάγραμμα καταστάσεων που φαίνεται στο επόμενο slide. Ισχύει αυτό; Αν όχι ποιο διάγραμμα καταστάσεων υλοποιεί;



Πρόβλημα 4 (2ο μέρος)



Βιβλιογραφία

- Η παρουσίαση βασίζεται στο κεφάλαιο 6 – CPU Design, από το βιβλίο.

- Carpinelli, “Computer Systems Organization & Architecture”, Pearson Education, 2001, ISBN 8177587676, 978817758767 aw-bc.com/info/carpinelli/

Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης