ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Διάρθρωση

1. Μικροπρογραμματιζόμενη μονάδα ελέγχου
2. Διαδρομή δεδομένων
3. Χρονισμός διαδρομής ελέγχου
4. Επικοινωνία με την κύρια μνήμη
5. Αρχιτεκτονική Mic-1
6. Ακολουθίες μικροεντολών
7. Συμβολισμός μικροεντολών (MAL)

μΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

Κύκλος Εντολής

- Όπως έχουμε αναφέρει η εκτέλεση μιας εντολής γίνεται με τα ακόλουθα βήματα:
  1. *Προσκόμιση* (*fetch*) της εντολής από τη μνήμη (τη διεύθυνση μνήμης την οποία υποδεικνύει ο μετρητής προγράμματος – PC) στον καταχωρητή εντολών IR .
  2. Μεταβολή του PC ώστε να δείχνει στην επόμενη θέση εντολής στη μνήμη.
  3. *Αποκωδικοποίηση* (*decode*) της εντολής ώστε να ενεργοποιηθούν τα κατάλληλα σήματα ελέγχου.
  4. *Προσκόμιση δεδομένων* (τελούμενων – operands) που ενδεχόμενα χρειάζεται η εντολή από την κύρια μνήμη ή τους καταχωρητές.
  5. *Εκτέλεση* (*execute*) της εντολής.
  6. *Αποθήκευση* (*store*) του αποτελέσματος.
  7. Επιστροφή στο 1ο βήμα για την εκτέλεση της επόμενης εντολής.
- Τα βήματα αυτά αποτελούν τον *κύκλο εντολής*.
- Ο κύκλος εντολής μπορεί να χωριστεί σε 2 φάσεις:
  - τη φάση της προσκόμισης (βήματα 1-2) που είναι σταθερά και δεν εξαρτώνται από το είδος της εντολής
  - τη φάση της εκτέλεσης (βήματα 3-6) που εξαρτώνται από την κάθε εντολή.

Μονάδα Ελέγχου

- Η μονάδα ελέγχου παράγει σε κάθε κύκλο εντολής ένα σύνολο από κατάλληλα σήματα (σήματα ελέγχου) για την εκτέλεση της εντολής.
  - Κατά τη φάση προσκόμισης, παράγει σήματα που επιφέρουν την προσκόμιση της εντολής (*εδώ επιλέγεται η σειρά εκτέλεσης των εντολών*).
  - Κατά τη φάση εκτέλεσης, αποκωδικοποιεί την εντολή και παράγει σήματα που κατευθύνουν τα δεδομένα στις κατάλληλες μονάδες και επιλέγουν τις λειτουργίες που θα πρέπει να εκτελεστούν σε συγκεκριμένες χρονικές περιόδους (*εδώ ερμηνεύονται η εντολές*).
- Η σειρά εκτέλεσης των εντολών βασίζεται στη χρήση του *μετρητή προγράμματος*, (*program counter – PC*) ο οποίος περιέχει τη διεύθυνση της επόμενης προς εκτέλεση εντολής. Οι εντολές εκτελούνται συνήθως σειριακά.
- Υπάρχουν περιπτώσεις αλλαγής της σειράς εκτέλεσης:
  1. Η εκτελούμενη εντολή είναι εντολή *άλματος* (*jump*) ή *διακλάδωσης* (*branch*).
  2. Η εκτελούμενη εντολή είναι εντολή κλήσεως υποπρογράμματος.
  3. Έχει συμβεί μια ειδική περίπτωση (exception), π.χ. υπερχείλιση, διαίρεση με 0 ...
  4. Έχει ληφθεί σήμα διακοπής (interrupt) για την εξυπηρέτηση κάποιας μονάδας.

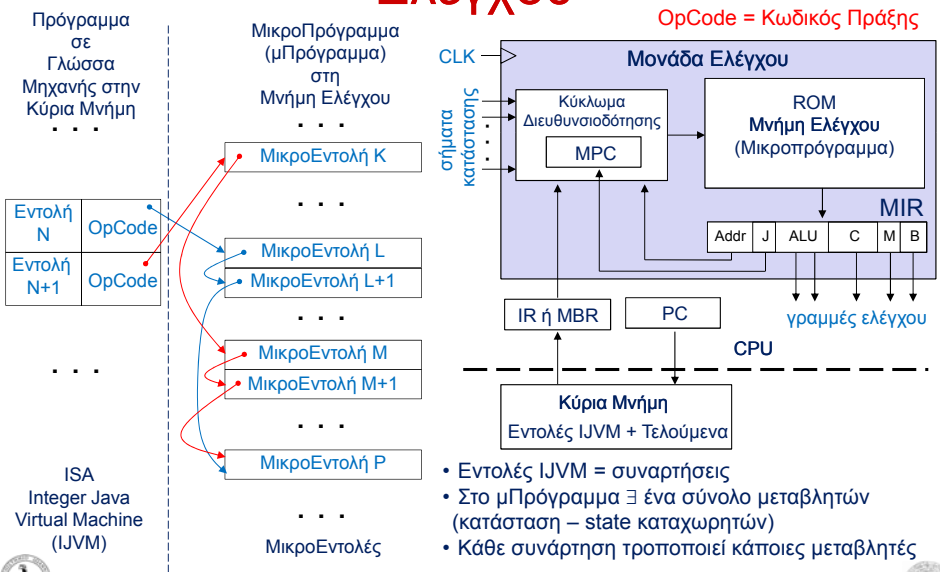
Συνήθη εσωγενή αίτια

Συνήθη εξωγενή αίτια

Μικροαρχιτεκτονική

- Ως Αρχιτεκτονική Συνόλου Εντολών (Instruction Set Architecture – ISA) θα θεωρήσουμε αυτή της *IJVM* (*Integer Java Virtual Machine*).
- Η μικροαρχιτεκτονική αποτελείται από:
  - Το υλικό και τις διασυνδέσεις του
    - τη διαδρομή δεδομένων
    - τη μονάδα ελέγχου
    - τους εσωτερικούς διαύλους και
    - τη βοηθητική λογική
  - Το *μικροπρόγραμμα* (*μΠρόγραμμα*) είναι αποθηκευμένο σε μνήμη ROM (*Μνήμη Ελέγχου*), και αποτελείται από *μικροεντολές* (*μΕντολές*).
- Κάθε εντολή της IJVM αντιστοιχεί σε μια συνάρτηση στο μΠρόγραμμα, δηλ. σε μια αλληλουχία μΕντολών στη ROM. Οι εντολές καλούνται από το κυρίως πρόγραμμα το οποίο είναι ένας ατέρμον βρόχος που προσδιορίζει πια εντολή-συνάρτηση θα κληθεί και την καλεί.
- Οι καταχωρητές της CPU χρησιμεύουν ως μεταβλητές του μΠρογράμματος.
- Κάθε εντολή τροποποιεί μέρος των καταχωρητών-μεταβλητών.

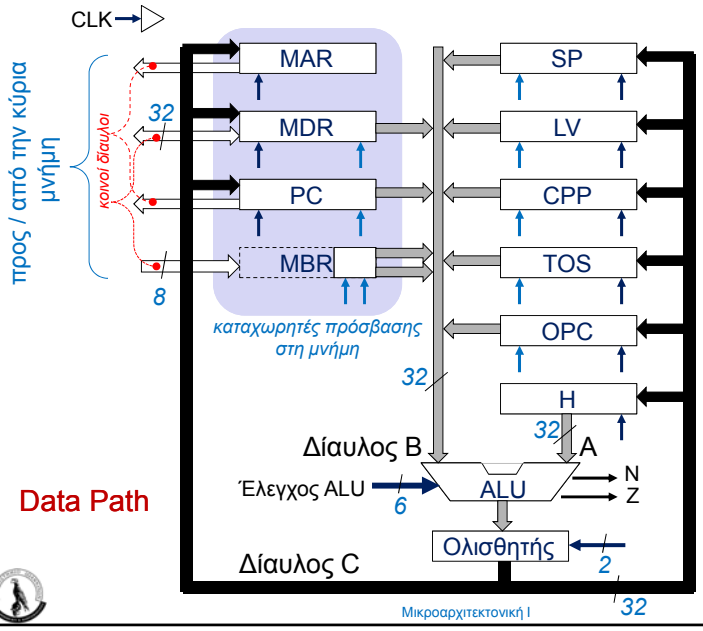
Μικροπρογραμματιζόμενη Μονάδα Ελέγχου



- Εντολές IJVM = συναρτήσεις
- Στο μΠρόγραμμα ∃ ένα σύνολο μεταβλητών (κατάσταση – state καταχωρητών)
- Κάθε συνάρτηση τροποποιεί κάποιες μεταβλητές

ΔΙΑΔΡΟΜΗ ΔΕΔΟΜΕΝΩΝ (DATA PATH)

Διαδρομή Δεδομένων



MAR:
κατ. διεύθυνσης μνήμης

MDR:
κατ. δεδομένων μνήμης

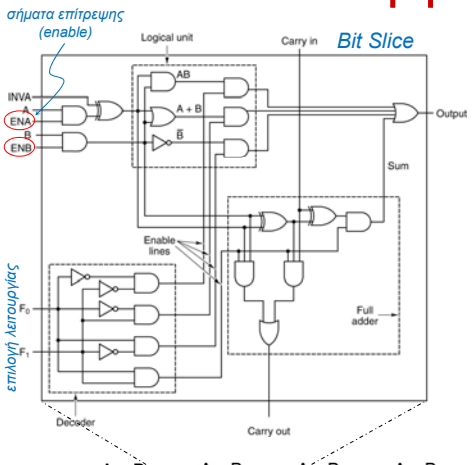
PC:
μετρητής προγράμματος
(έχει τη δ/νση του
επόμενου byte εντολής)

MBR:
κατ. εντολών
(περιέχει το τρέχων byte
της εντολής)

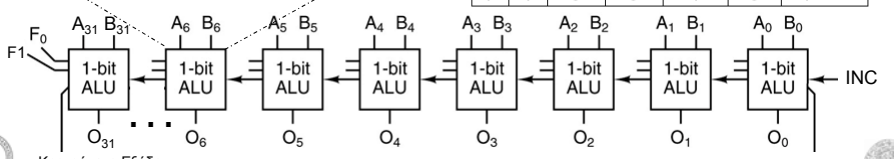
H (hold):
κατ. κατακράτησης

OPC:
κατ. προσωρινής
αποθήκευσης

Λειτουργία ALU 1/3



F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



Λειτουργία ALU 2/3

- Ο έλεγχος της ALU γίνεται με 6 σήματα και συνεπώς ∃ 2^6=64 συνδυασμοί. Οι περισσότεροι δεν χρησιμοποιούνται!
- Τα σήματα επίτρεψης ENA/ENB στο "0" απενεργοποιούν τις αντίστοιχες εισόδους και μηδενίζουν την σχετική τιμή.
- Κάποιες διεργασίες εκτελούνται με περισσότερους από έναν τρόπους:

  B or 0 ≡ B + 0

  <F0,F1,ENA,ENB,INVA,INC>: <011100> και <110100>
- Η λειτουργία (B−A) έχει συνδυασμό κλήσης <111111>. Συνεπώς στην πράξη εκτελείτε η ακόλουθη λειτουργία:

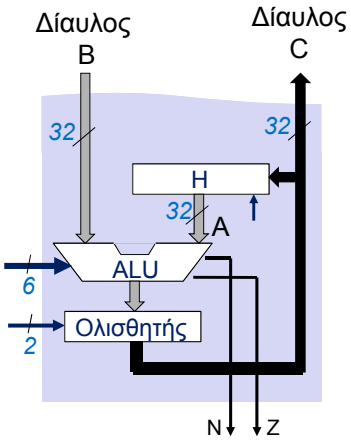
  $B + \overline{A} + 1$

  που είναι εξ ορισμού η πράξη της αφαίρεσης στο συμπλήρωμα ως προς 2.
  Η συγκεκριμένη ALU χρησιμοποιεί το συμπλήρωμα ως προς 2.

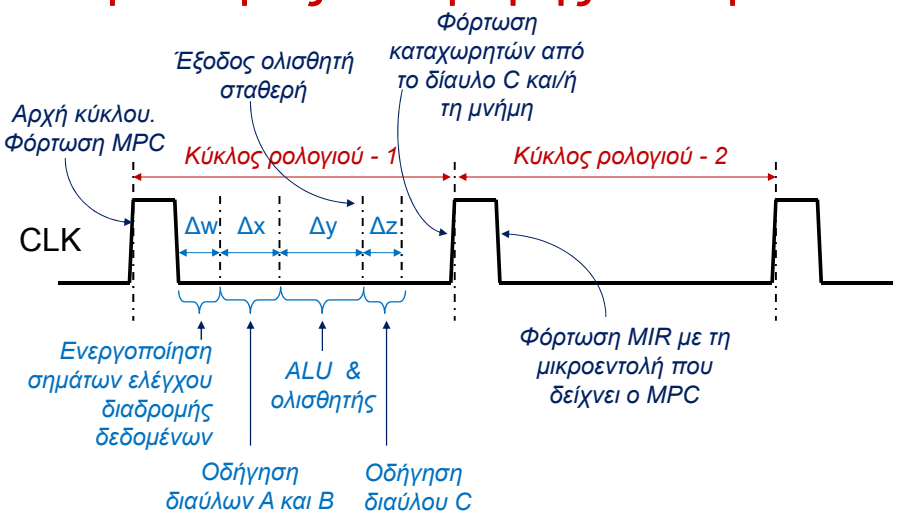
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

Λειτουργία ALU 3/3

- Το σήμα N της ALU προκύπτει από το σημαντικότερο (MSB) bit του αποτελέσματος. Στις αριθμητικές πράξεις είναι το πρόσημο του αποτελέσματος (N="1" ⇒ το αποτέλεσμα είναι αρνητικός αριθμός - negative)
- Το σήμα Z της ALU είναι "1" όταν το αποτέλεσμα είναι μηδέν (zero).
- Η ALU έχει 2 εισόδους των 32-bit. Την A οδηγεί μόνο ο καταχωρητής H. Την B οδηγούν όλοι οι υπόλοιποι καταχωρητές εκτός των H και MAR.
- Αριθμητικές και λογικές πράξεις ALU (F1F0):
  - 00 → AND 10 → invert B
  - 01 → OR 11 → plus (+)
- Ο Ολισθητής έχει 2 σήματα ελέγχου:
  - SLL8: Shift Logical Left 8 bits
  - SRA1: Shift Right Arithmetic 1 bit



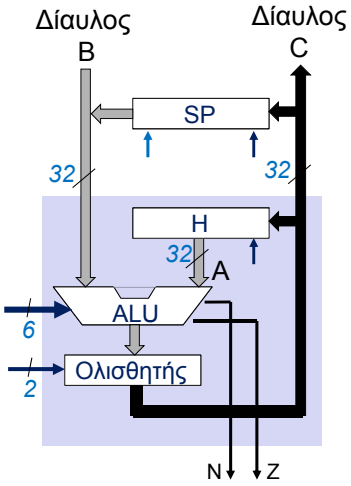
Χρονισμός Διαδρομής Δεδομένων



Παράδειγμα Διεργασίας στην ALU

- Διεργασία: SP = SP + 1

SP → B δίαυλο
ALU: F0F1 = 11 (plus)
ENA = INVA = 0
ENB = 1
INC = 1
Ολισθητής: SLL8 = SRA1 = 0
C δίαυλος → SP



Επικοινωνία με την Κύρια Μνήμη 1/2

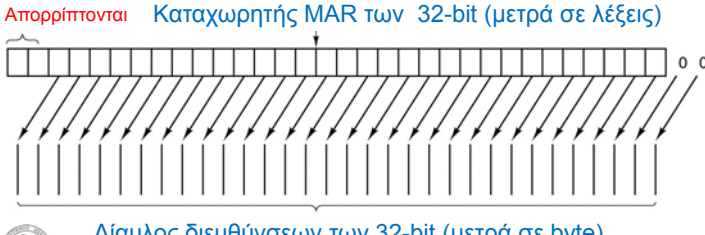
- Δύο τρόποι επικοινωνίας με τη μνήμη:
  - Μέσω μιας θύρας μνήμης των 32-bit, προσπελάσιμη ανά λέξη (MAR/MDR).
  - Μέσω μιας θύρας μνήμης των 8-bit, προσπελάσιμη ανά byte (PC/MBR).
- Ο καταχωρητής MAR χρησιμοποιείται για τη διευθυνσιοδότηση της μνήμης με σκοπό τη μεταφορά δεδομένων των 32-bit (λέξη). *Στη/από-τη* διεύθυνση που υποδεικνύει ο MAR *εγγράφονται/διαβάζονται* δεδομένα μέσω του καταχωρητή MDR.
- Ο καταχωρητής PC χρησιμοποιείται για τη διευθυνσιοδότηση της μνήμης με σκοπό την προσκόμιση εντολών των 8-bit. Από την διεύθυνση που υποδεικνύει ο PC προσκομίζονται στον MBR το byte της επόμενης προς εκτέλεση εντολής.
  - Ο MBR ποτέ δεν φορτώνεται από το δίαυλο C και συνεπώς δεν μπορούμε να εγγράψουμε δεδομένα στη μνήμη με τη χρήση αυτού του καταχωρητή.

Διεύθυνση στον PC	Byte στη μνήμη
0	0
1	1
2	2
3	3

Διεύθυνση στον MAR	Byte στη μνήμη
0	0 – 3
1	4 – 7
2	8 – 11
3	12 – 15

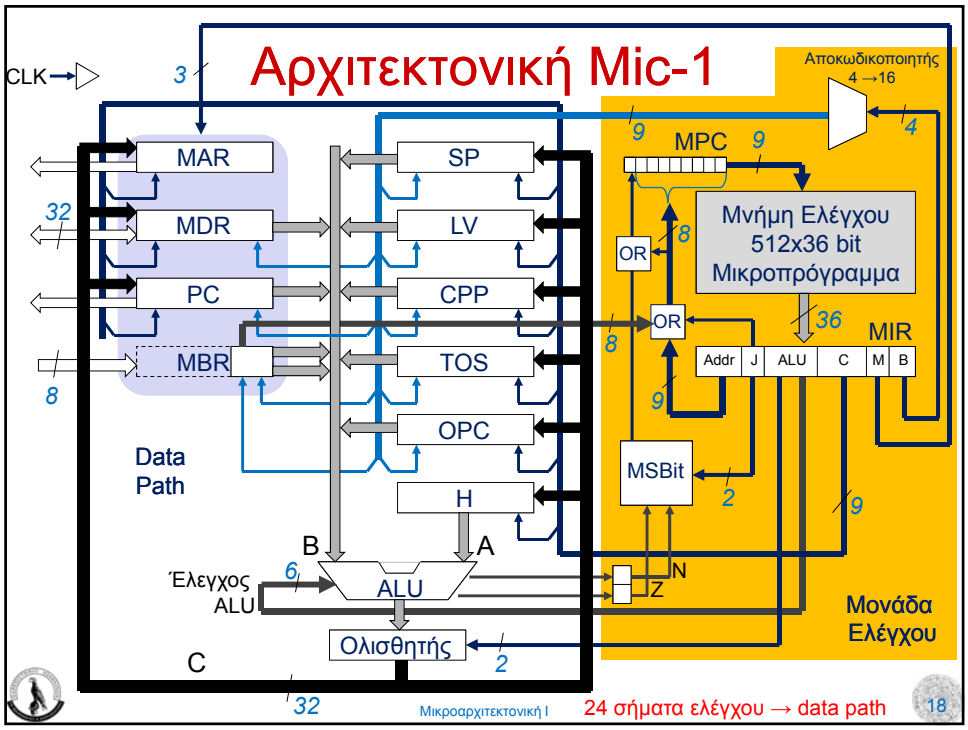
Επικοινωνία με την Κύρια Μνήμη 2/2

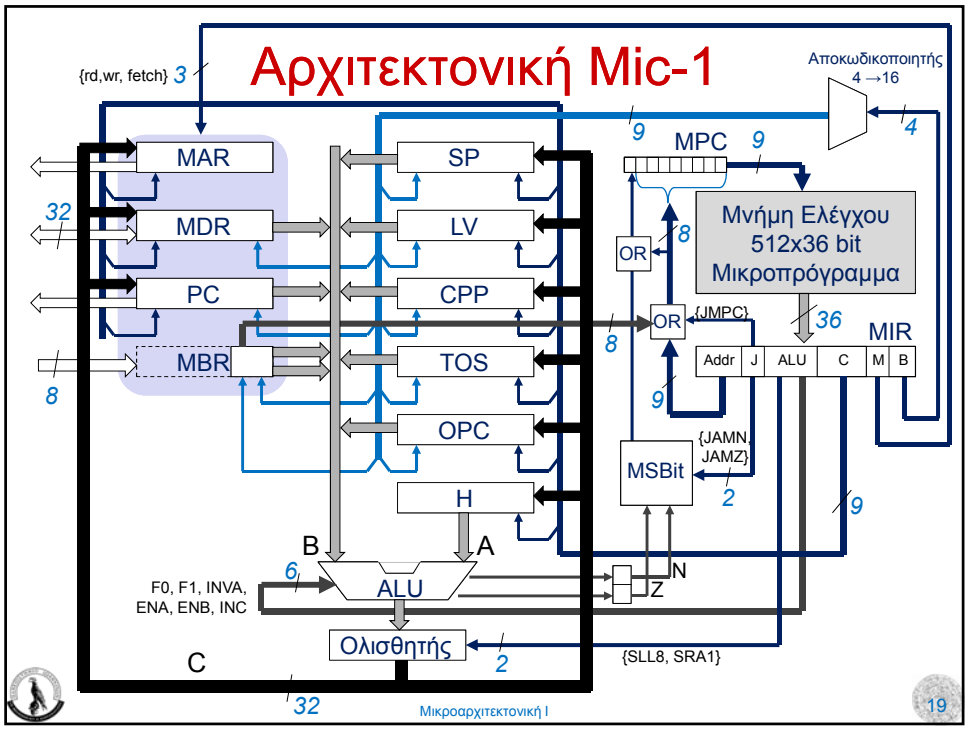
- Η μέγιστη διευθυνσιοδοτούμενη μνήμη είναι 2^{32}B=4GB και είναι οργανωμένη με θέσεις του 1 byte.
- Ο καταχωρητής MAR διευθυνσιοδοτεί στη μνήμη λέξεις των 32-bit. Συνεπώς σε bytes δείχνει πάντα σε πολλαπλάσια του 4.
- Για να επιτευχθεί αυτό τα δύο περισσότερο σημαντικά ψηφία του καταχωρητή απορρίπτονται με ολίσθησή του κατά δύο θέσεις αριστερά και προστίθενται μηδενικά “0” στις θέσεις των δύο λιγότερο σημαντικών ψηφίων (δηλ. πολλαπλασιάζεται η τιμή ×4).
  - π.χ. ο MAR όταν φορτώνεται με τη διεύθυνση 0 δείχνει τη διεύθυνση 0, με τη διεύθυνση 1 δείχνει τη διεύθυνση 4, ενώ με τη διεύθυνση 2 δείχνει τη διεύθυνση 8 κ.ο.κ.

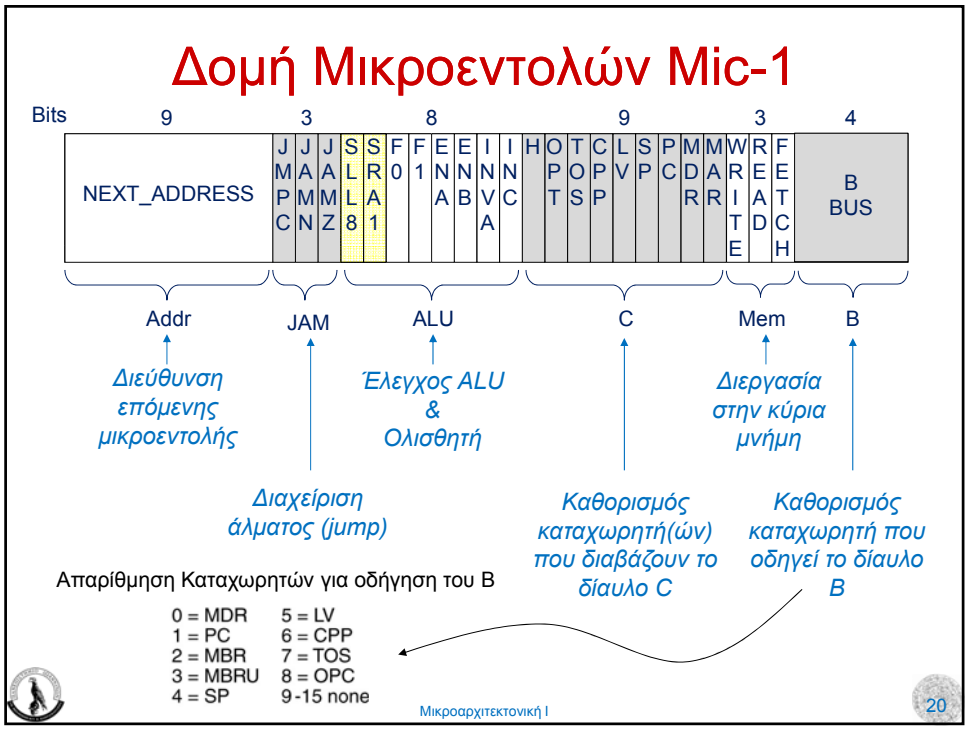


Διεύθυνση στον MAR	Byte στη μνήμη
0	0 – 3
1	4 – 7
2	8 – 11
3	12 – 15

μΑΡΧΙΤΕΚΤΟΝΙΚΗ Mic-1

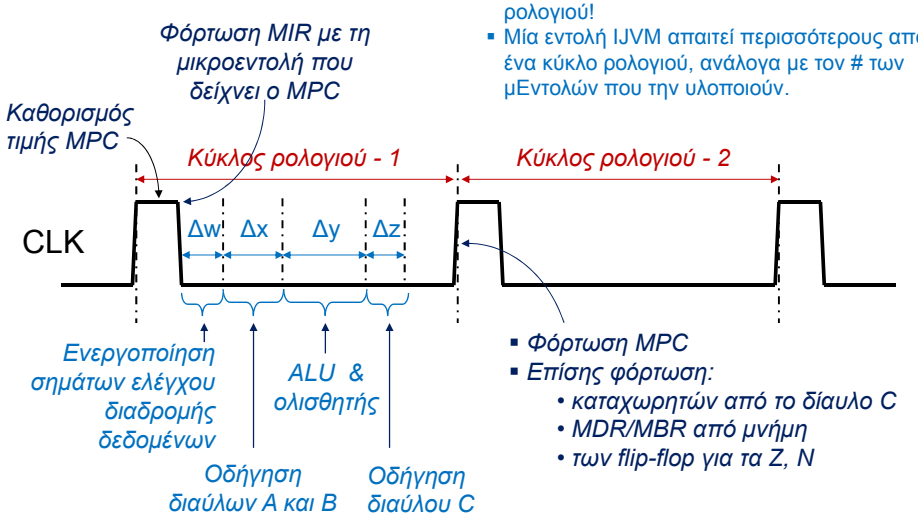



Αρχιτεκτονική Mic-1
{rd,wr, fetch} 3
Αποκωδικοποιητής 4 →16
MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
MPC
9
4
32
8
OR
Μνήμη Ελέγχου 512x36 bit Μικροπρόγραμμα
{JMPC}
36
MIR
Addr
J
ALU
C
M
B
MSBit
{JAMN, JAMZ}
2
B
A
6
F0, F1, INVA, ENA, ENB, INC
ALU
N
Z
Ολισθητής
{SLL8, SRA1}
C
32
Μικροαρχιτεκτονική I
19

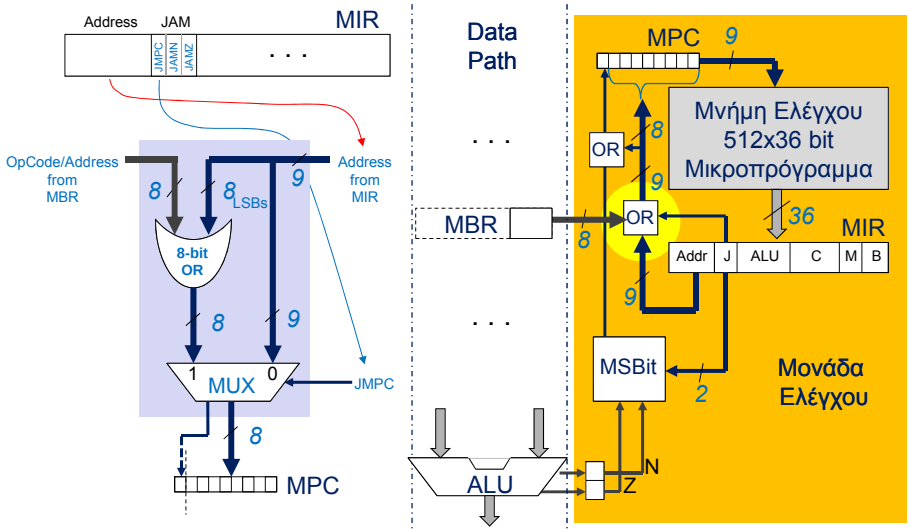

Δομή Μικροεντολών Mic-1
Bits
9
3
8
9
3
4
NEXT_ADDRESS
JMPC
JAMN
JAMZ
SLL8
SRA1
F0
F1
ENA
ENB
INVA
INC
H
OPT
TOS
CPP
LV
SP
PC
MDR
MAR
WRITE
READ
FETCH
B BUS
Addr
JAM
ALU
C
Mem
B
Διεύθυνση επόμενης μικροεντολής
Διαχείριση άλματος (jump)
Έλεγχος ALU & Ολισθητή
Καθορισμός καταχωρητή(ών) που διαβάζουν το δίαυλο C
Διεργασία στην κύρια μνήμη
Καθορισμός καταχωρητή που οδηγεί το δίαυλο B
Απαρίθμηση Καταχωρητών για οδήγηση του B
0 = MDR
1 = PC
2 = MBR
3 = MBRU
4 = SP
5 = LV
6 = CPP
7 = TOS
8 = OPC
9-15 none
Μικροαρχιτεκτονική I
20

Χρονισμός Mic-1

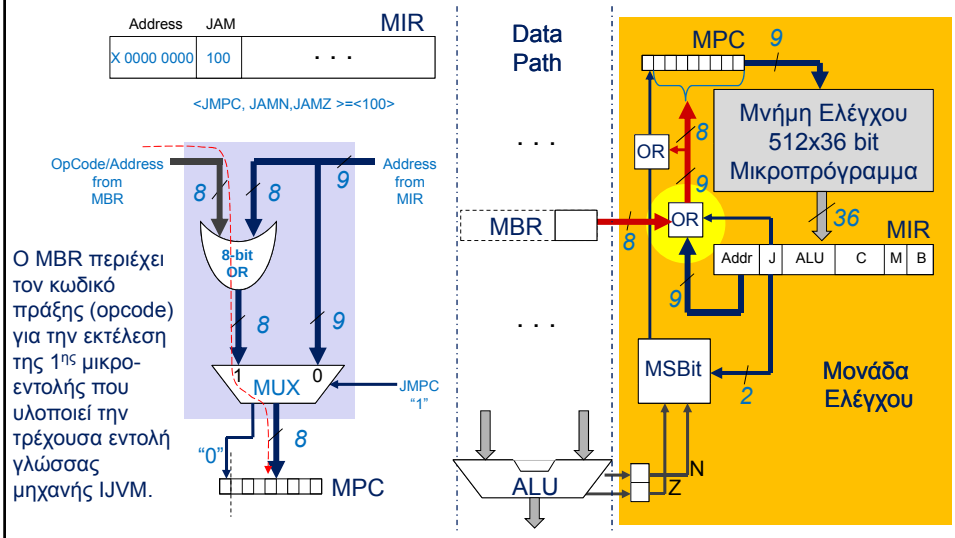
- Μία μΕντολή εκτελείται μέσα σε ένα κύκλο ρολογιού!
- Μία εντολή IJVM απαιτεί περισσότερους από ένα κύκλο ρολογιού, ανάλογα με τον # των μΕντολών που την υλοποιούν.



Ακολουθίες Μικροεντολών 1/5



Ακολουθίες Μικροεντολών 2/5

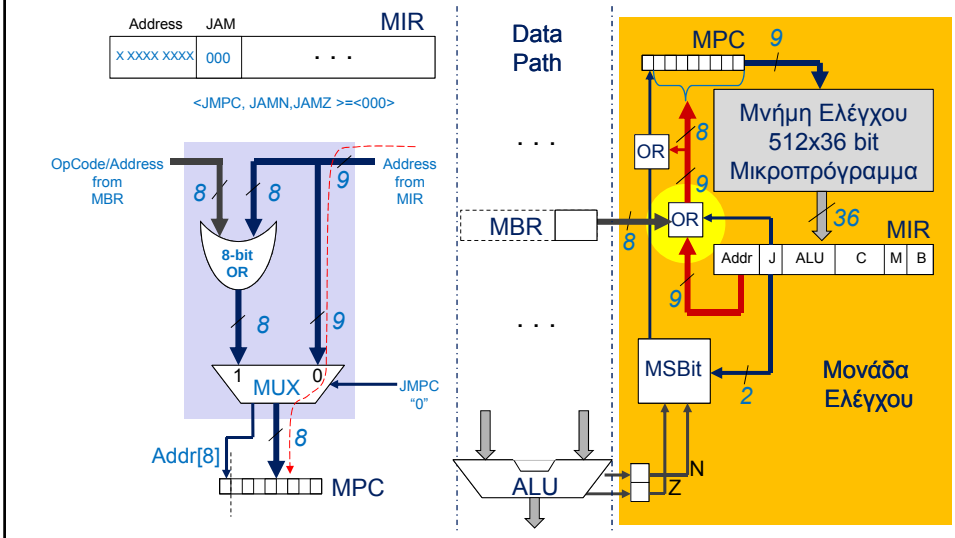


Ο MBR περιέχει τον κωδικό πράξης (opcode) για την εκτέλεση της 1ης μικρο-εντολής που υλοποιεί την τρέχουσα εντολή γλώσσας μηχανής IJVM.

MPC[7:0] = MBR[7:0]

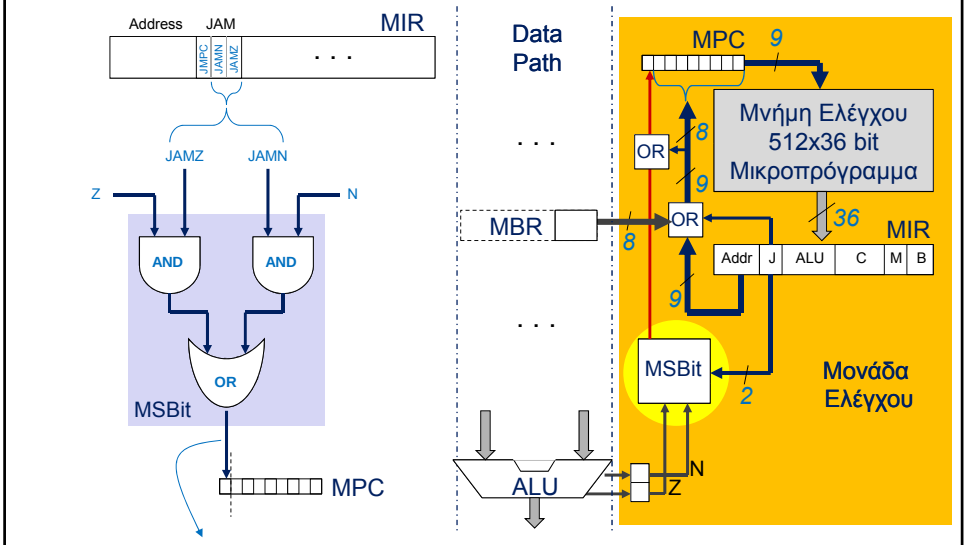
Ο MBR μπορεί να διευθυνσιοδοτήσει οποιαδήποτε από τις 256 (2^8) αρχικές δ/σεις της Μνήμης Ελέγχου.

Ακολουθίες Μικροεντολών 3/5



MPC[8:0] = Addr[8:0]

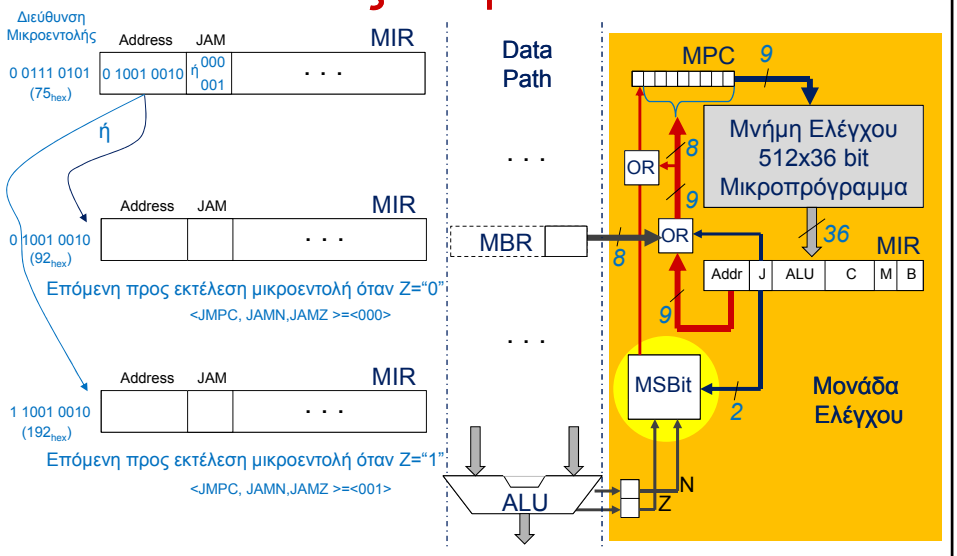
Ακολουθίες Μικροεντολών 4/5



MPC[8] = (JAMZ AND Z) OR (JAMN AND N)

MPC[7:0] = Addr[7:0]

Ακολουθίες Μικροεντολών 5/5

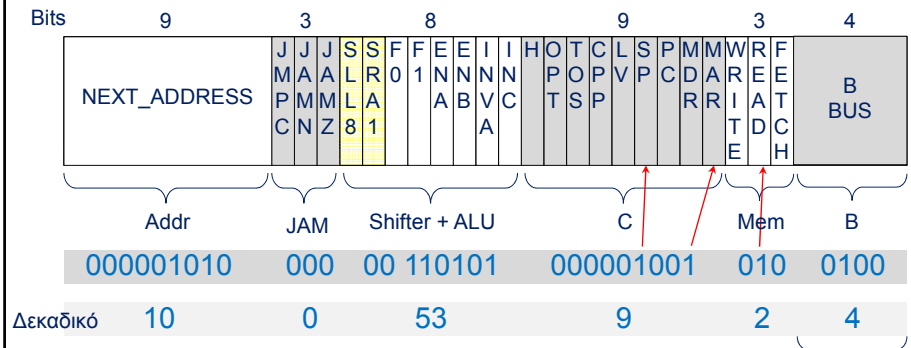


Επισημάνσεις

- Αρχικά για την τρέχουσα εντολή της γλώσσας μηχανής IJVM, ο MBR περιέχει τον *κωδικό πράξης* (*opcode*) για την εκτέλεση της 1ης μΕντολής που την υλοποιεί. Δηλ. το πεδίο opcode μιας εντολής είναι μια διεύθυνση της Μνήμης Ελέγχου στην οποία βρίσκεται η πρώτη μΕντολή από την ομάδα των μΕντολών που υλοποιούν την IJVM εντολή.
- Η ομάδα των μΕντολών δεν είναι σε διαδοχικές διευθύνσεις μνήμης στη Μνήμη Ελέγχου. Κάθε μΕντολή προσδιορίζει την επόμενη προς εκτέλεση μΕντολή και συνεπώς η ομάδα των μΕντολών είναι πάντα προσδιορισμένη.
- Η τελευταία μΕντολή της ομάδας έχει το πεδίο Addr[8:0] = 000000000 & JMPC = "1" έτσι ώστε στον MPC να φορτωθεί ο MBR για να ξεκινήσει η εκτέλεση της επόμενης εντολής IJVM.
- Τα σήματα JAMZ και JAMN βοηθούν στο να δημιουργηθούν υπό συνθήκη άλματα στο μΠρόγραμμα (κάτι σαν "IF"). Η διεύθυνση του άλματος είναι στο δεύτερο μισό της Μνήμης Ελέγχου ($\geq 256_{10}$). Έτσι μια εντολή μπορεί να εκτελεστεί με περισσότερους από έναν τρόπους ανάλογα με το αποτέλεσμα της πράξης στην ALU (μηδέν ή αρνητικός αριθμός), π.χ.:
  - if Z=0 then goto Addr[7:0] else goto Addr[8:0] ≡ 256+Addr[7:0] ή
  - if N=0 then goto Addr[7:0] else goto Addr[8:0] ≡ 256+Addr[7:0].

ΣΥΜΒΟΛΙΚΗ ΓΛΩΣΣΑ μΕΝΤΟΛΩΝ (μASSEMBLY LANGUAGE)

Συμβολισμός Σημάτων μΕντολής 1/2



Απλούστερη αναπαράσταση σημάτων:

B=4; MEM=2; C=9; ALU=53; JAM=0; Addr=10

- Φόρτωση του διαύλου B από τον καταχωρητή SP (B=4)
- Η ALU αυξάνει την είσοδο B κατά 1 – Shifter τίποτε (ALU=53)
- Φόρτωση καταχωρητών SP & MAR από το δίαυλο C (C=9)
- Εκτέλεση ανάγνωσης (READ) στην Κύρια Μνήμη (MEM=2)
- Επόμενη μΕντολή προς εκτέλεση η $10^{η}$ (Addr=10)

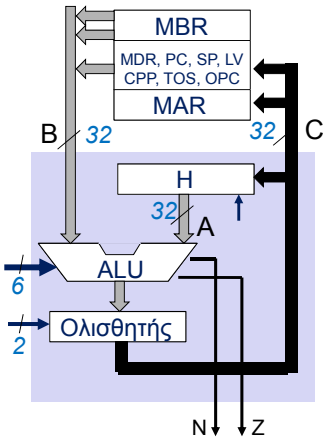
0 = MDR
1 = PC
2 = MBR
3 = MBRU
4 = SP
5 = LV
6 = CPP
7 = TOS
8 = OPC
9-15 none

Συμβολισμός Σημάτων μΕντολής 2/2

- Η προηγούμενη αναπαράσταση των σημάτων:
  B=4; MEM=2; C=9; ALU=53; JAM=0; Addr=10
- Πιο κατανοητή αναπαράσταση των σημάτων:
  B=SP; Read; WC(SP, MAR); INC; NextAddr=10
- Συμβολική αναπαράσταση των σημάτων:
  MAR = SP = SP+1; Read; goto 10
- MAL: *Συμβολική Γλώσσα μΕντολών* (*MicroAssembly Language*)
  - Τρόπος αναπαράστασης των 36 σημάτων της κάθε μΕντολής.
  - Αναπαράσταση κοντινή στη φυσική γλώσσα για διευκόλυνση του χρήστη. Στη Μνήμη Ελέγχου υπάρχουν 36 bits.

Συμβολική Γλώσσα μΕντολών – MAL

- Η MAL είναι μια γλώσσα που αντανακλά τα χαρακτηριστικά της συγκεκριμένης μΑρχιτεκτονικής.
- Ένα από τα δύο ορίσματα της πράξης στην ALU είναι: ο Η, το 0, το 1 ή το -1.
- Δεν χρησιμοποιούνται άλλοι αριθμοί εκτός από το 0, 1, -1
  - π.χ. SP = SP + 2 είναι μη επιτρεπτό!
- Από το δίαυλο C μπορεί να φορτωθεί ένας ή περισσότεροι καταχωρητές
  - MDR = SP ή MAR = SP = H = SP+1
- Δεν επιτρέπονται:
  - H = H – SP
  - MBR = H
  - H = MAR + H
  - MDR = MDR + SP (δεν εκτελείται σε ένα κύκλο)



Επιτρεπτές Ενέργειες

- Ο διπλανός πίνακας δίνει τις επιτρεπτές ενέργειες που υποστηρίζει η MAL.
- Στη θέση του ορίσματος προορισμού DEST, μέσω του διαύλου C, μπορεί να είναι οποιοσδήποτε από τους καταχωρητές: MAR, MDR, PC, SP, LV, CPP, TOS, H.
- Στη θέση του ορίσματος πηγής SOURCE, μέσω του διαύλου B, μπορεί να είναι ένας από τους καταχωρητές MDR, PC, MBR, SP, LV, CPP, TOS.

DEST = H
DEST = SOURCE
DEST = $\overline{H}$
DEST = $\overline{SOURCE}$
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE – H
DEST = SOURCE – 1
DEST = –H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = –1

Κλήση Επόμενης μΕντολής 1/2

- Η διεύθυνση της επόμενης μΕντολής προς εκτέλεση δηλώνεται με το “goto” στη θέση μνήμης μέσα στην Μνήμη Ελέγχου, π.χ.:
  - MAR = H + CPP; rd; goto 10
- Είναι πιο κατανοητό να χρησιμοποιήσουμε *ετικέτες* (*labels*) για τον προσδιορισμό της διεύθυνσης της επόμενης μΕντολής, π.χ.:
  - MAR = H + CPP; rd; goto label1
- Ως ετικέτα χρησιμοποιείται το όνομα της διευθυνσιοδοτούμενης μΕντολής, π.χ.:
  - MAR = H + CPP; rd; goto iload3
- Αν η επόμενη μΕντολή είναι στην αμέσως επόμενη διεύθυνση, τότε στη MAL μπορεί να παραληφθεί το πεδίο “goto” το οποίο υπονοείτε, π.χ.
  - SP = MAR = SP + 1

Κλήση Επόμενης μΕντολής 2/2

- Τα bits του πεδίου JAM μπορούν να αλλάξουν τη διεύθυνση της επόμενης μΕντολής.
- Η υπόδειξη της υπό συνθήκη διακλάδωσης δηλώνεται ως ακολούθως:
  - if (Z) goto L1; else goto L2 (JAMZ=“1”)
  - if (N) goto L1; else goto L2 (JAMN=“1”)
  - goto (MBR) (JMPC=“1”)
- Παράδειγμα:
  - Z = OPC; if (Z) goto T; else goto F
  - Οι μΕντολές T και F απέχουν μεταξύ τους στη Μνήμη Ελέγχου κατά 256 θέσεις καθώς η διεύθυνσή τους διαφέρει μόνο στο MSBit.

Επικοινωνία με τη Κύρια Μνήμη

- Η ενέργειες των μΕντολών για πρόσβαση στην Κύρια Μνήμη καθορίζονται από τα 3 bits (rd, wr και fetch) του πεδίου Μ της μΕντολής, π.χ.:
  - MAR = SP; rd — δεδομένα προσκομίζονται στον MDR
  - MAR = SP; wr — δεδομένα αποστέλλονται από τον MDR
  - PC = PC + 1; fetch — το επόμενο byte εντολής μεταφέρεται στον MBR
- Απαιτείται προσοχή σε θέματα χρονισμού της μνήμης και διαθεσιμότητας των δεδομένων στους καταχωρητές, π.χ.:
  - MAR = SP; rd;

    MDR = H — διπλή ανάθεση στον MDR μέσα σε ένα κύκλο
  - MAR = SP; rd;

    TOS = MDR — ο MDR δεν έχει διαθέσιμα τα δεδομένα στον αμέσως επόμενο κύκλο ρολογιού

Το μΠρόγραμμα 1/5

Label	Operations	Comments
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch
nop1	goto Main1	Do nothing
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack
isub1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR – H; wr; goto Main1	Do subtraction; write to top of stack
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack
ior1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto Main1	Do OR; write to new top of stack
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR
dup2	MDR = TOS; wr; goto Main1	Write new stack word
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
swap1	MAR = SP – 1; rd	Set MAR to SP – 1; read 2nd word from stack
swap2	MAR = SP	Set MAR to top word
swap3	H = MDR; wr	Save TOS in H; write 2nd word to top of stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Set MAR to SP – 1; write as 2nd word on stack
swap6	TOS = H; goto Main1	Update TOS

Το μΠρόγραμμα 2/5

bipush1	SP = MAR = SP + 1	MBR = the byte to push onto stack
bipush2	PC = PC + 1; fetch	Increment PC, fetch next opcode
bipush3	MDR = TOS = MBR; wr; goto Main1	Sign-extend constant and push on stack
iload1	H = LV	MBR contains index; copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
istore1	H = LV	MBR contains index; Copy LV to H
istore2	MAR = MBRU + H	MAR = address of local variable to store into
istore3	MDR = TOS; wr	Copy TOS to MDR; write word
istore4	SP = MAR = SP – 1; rd	Read in next-to-top word on stack
istore5	PC = PC + 1; fetch	Increment PC; fetch next opcode
istore6	TOS = MDR; goto Main1	Update TOS
wide1	PC = PC + 1; fetch;	Fetch operand byte or next opcode
wide2	goto (MBR OR 0x100)	Multiway branch with high bit set
wide_iload1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
wide_iload2	H = MBRU << 8	H = 1st index byte shifted left 8 bits
wide_iload3	H = MBRU OR H	H = 16-bit index of local variable
wide_iload4	MAR = LV + H; rd; goto iload3	MAR = address of local variable to push
wide_istore1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
wide_istore2	H = MBRU << 8	H = 1st index byte shifted left 8 bits
wide_istore3	H = MBRU OR H	H = 16-bit index of local variable
wide_istore4	MAR = LV + H; goto istore3	MAR = address of local variable to store into
ldc_w1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
ldc_w2	H = MBRU << 8	H = 1st index byte << 8
ldc_w3	H = MBRU OR H	H = 16-bit index into constant pool
ldc_w4	MAR = H + CPP; rd; goto iload3	MAR = address of constant in pool

Το μΠρόγραμμα 3/5

Label	Operations	Comments
iinc1	H = LV	MBR contains index; Copy LV to H
iinc2	MAR = MBRU + H; rd	Copy LV + index to MAR; Read variable
iinc3	PC = PC + 1; fetch	Fetch constant
iinc4	H = MDR	Copy variable to H
iinc5	PC = PC + 1; fetch	Fetch next opcode
iinc6	MDR = MBR + H; wr; goto Main1	Put sum in MDR; update variable
goto1	OPC = PC – 1	Save address of opcode.
goto2	PC = PC + 1; fetch	MBR = 1st byte of offset; fetch 2nd byte
goto3	H = MBR << 8	Shift and save signed first byte in H
goto4	H = MBRU OR H	H = 16-bit branch offset
goto5	PC = OPC + H; fetch	Add offset to OPC
goto6	goto Main1	Wait for fetch of next opcode
iflt1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iflt2	OPC = TOS	Save TOS in OPC temporarily
iflt3	TOS = MDR	Put new top of stack in TOS
iflt4	N = OPC; if (N) goto T; else goto F	Branch on N bit
ifeq1	MAR = SP = SP – 1; rd	Read in next-to-top word of stack
ifeq2	OPC = TOS	Save TOS in OPC temporarily
ifeq3	TOS = MDR	Put new top of stack in TOS
ifeq4	Z = OPC; if (Z) goto T; else goto F	Branch on Z bit

Το μΠρόγραμμα 4/5

if_icmpeq1	MAR = SP = SP – 1; rd	Read in next-to-top word of stack
if_icmpeq2	MAR = SP = SP – 1	Set MAR to read in new top-of-stack
if_icmpeq3	H = MDR; rd	Copy second stack word to H
if_icmpeq4	OPC = TOS	Save TOS in OPC temporarily
if_icmpeq5	TOS = MDR	Put new top of stack in TOS
if_icmpeq6	Z = OPC – H; if (Z) goto T; else goto F	If top 2 words are equal, goto T, else goto F
T	OPC = PC – 1; goto goto2	Same as goto1; needed for target address
F	PC = PC + 1	Skip first offset byte
F2	PC = PC + 1; fetch	PC now points to next opcode
F3	goto Main1	Wait for fetch of opcode
invokevirtual1	PC = PC + 1; fetch	MBR = index byte 1; inc. PC, get 2nd byte
invokevirtual2	H = MBRU << 8	Shift and save first byte in H
invokevirtual3	H = MBRU OR H	H = offset of method pointer from CPP
invokevirtual4	MAR = CPP + H; rd	Get pointer to method from CPP area
invokevirtual5	OPC = PC + 1	Save Return PC in OPC temporarily
invokevirtual6	PC = MDR; fetch	PC points to new method; get param count
invokevirtual7	PC = PC + 1; fetch	Fetch 2nd byte of parameter count
invokevirtual8	H = MBRU << 8	Shift and save first byte in H
invokevirtual9	H = MBRU OR H	H = number of parameters
invokevirtual10	PC = PC + 1; fetch	Fetch first byte of # locals
invokevirtual11	TOS = SP – H	TOS = address of OBJREF – 1
invokevirtual12	TOS = MAR = TOS + 1	TOS = address of OBJREF (new LV)
invokevirtual13	PC = PC + 1; fetch	Fetch second byte of # locals
invokevirtual14	H = MBRU << 8	Shift and save first byte in H
invokevirtual15	H = MBRU OR H	H = # locals

Το μΠρόγραμμα 5/5

Label	Operations	Comments
invokevirtual16	MDR = SP + H + 1; wr	Overwrite OBJREF with link pointer
invokevirtual17	MAR = SP = MDR;	Set SP, MAR to location to hold old PC
invokevirtual18	MDR = OPC; wr	Save old PC above the local variables
invokevirtual19	MAR = SP = SP + 1	SP points to location to hold old LV
invokevirtual20	MDR = LV; wr	Save old LV above saved PC
invokevirtual21	PC = PC + 1; fetch	Fetch first opcode of new method.
invokevirtual22	LV = TOS; goto Main1	Set LV to point to LV Frame
ireturn1	MAR = SP = LV; rd	Reset SP, MAR to get link pointer
ireturn2		Wait for read
ireturn3	LV = MAR = MDR; rd	Set LV to link ptr; get old PC
ireturn4	MAR = LV + 1	Set MAR to read old LV
ireturn5	PC = MDR; rd; fetch	Restore PC; fetch next opcode
ireturn6	MAR = SP	Set MAR to write TOS
ireturn7	LV = MDR	Restore LV
ireturn8	MDR = TOS; wr; goto Main1	Save return value on original top of stack

112 μΕντολές!