Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Μηχανικών Πληροφορικής & Τηλεπικοινωνιών
(Κοζάνη)

Εισαγωγή στην Επιστήμη των Υπολογιστών

- Εργαστήριο 5 -

Διδάσκων: Παντελής Αγγελίδης

Χειμερινό εξάμηνο 2008

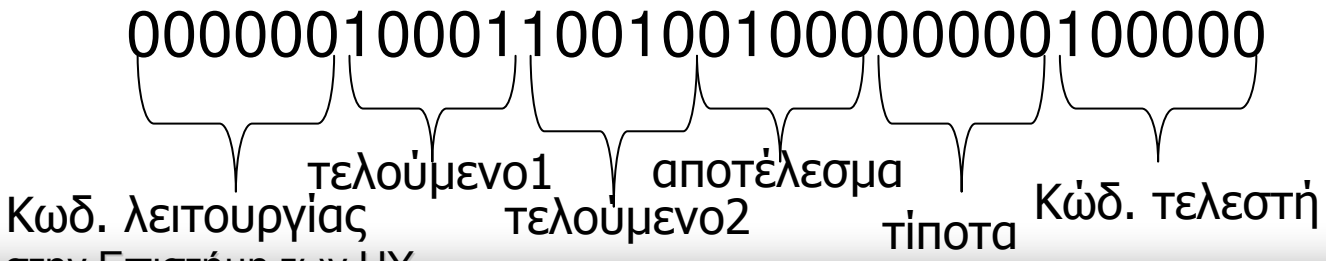
Συμβολική Γλώσσα
Assembly

Βασικές έννοιες

- **Πηγαία γλώσσα (source language):**η γλώσσα στην οποία είναι γραμμένο το αρχικό πρόγραμμα.
- **Γλώσσα προορισμού (target language)**: η γλώσσα στην οποία μετατρέπεται το αρχικό πρόγραμμα.
- **Μεταφραστές (translators):** τα προγράμματα που μετατρέπουν ένα πρόγραμμα χρήστη από τη γλώσσα που είναι γραμμένο σε κάποια άλλη γλώσσα
- **Αντικειμενικό πρόγραμμα (object program)** ή εκτελέσιμο δυαδικό πρόγραμμα (executable binary program): το πρόγραμμα που προκύπτει από τη μετάφραση
- **Συμβολομεταφραστής (assembler):** ο μεταφραστής του προγράμματος που είναι γραμμένο σε συμβολική γλώσσα (assembly language)
- **Μεταγλωτιστής (compiler):** ο μεταφραστής του προγράμματος που είναι γραμμένο σε γλώσσα υψηλού επιπέδου

Γλώσσα μηχανής

- Το σύνολο των δυαδικών αριθμών που ο επεξεργαστής αντιλαμβάνεται ως εντολές για:
  - Την εκτέλεση κάποιων πράξεων,
  - Τη μεταφορά πληροφορίας μεταξύ υπο-μονάδων του, ή μεταξύ αυτού και του συστήματος μνήμης ή των συσκευών Ε/Ε
  - την εκτέλεση άλλων λειτουργιών
- Διαφορετική για κάθε επεξεργαστή
  - Συμβατότητα μεταξύ επεξεργαστών
- Κάθε εντολή της γλώσσας μηχανής είναι κωδικοποιημένη με συγκεκριμένο τα τρόπο μέσα σε ένα δυαδικό αριθμό



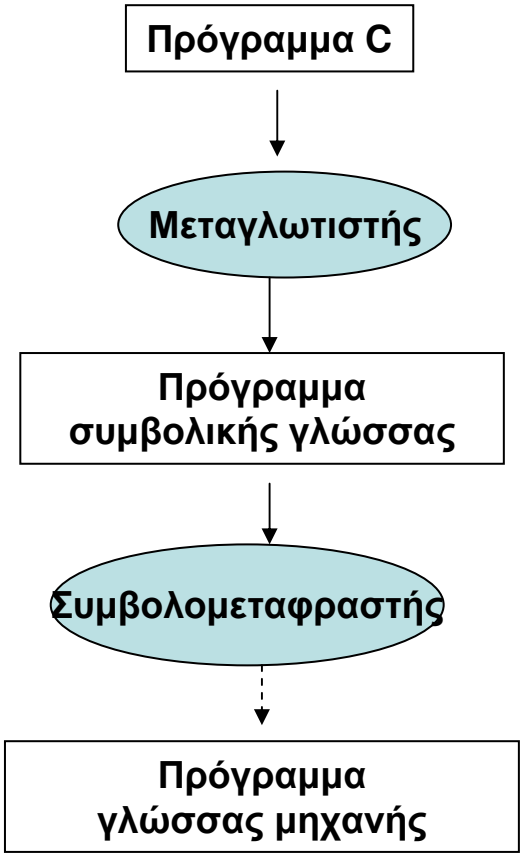
assembly..

- Πιο εύκολος ο προγραμματισμός σε συμβολική γλώσσα από ότι σε γλώσσα μηχανής
- Πλήρης ισοδυναμία ανάμεσα σε εντολές και υλικό: κάθε συμβολική πρόταση παράγει ακριβώς μια εντολή μηχανής
- Παρέχει πρόσβαση σε όλες τις δυνατότητες και εντολές που είναι διαθέσιμες στην μηχανή προορισμού
- Ένα πρόγραμμα σε συμβολική γλώσσα μπορεί να εκτελεί όλες τις εντολές του συνόλου εντολών της μηχανής προορισμού
- Ένα πρόγραμμα σε συμβολική γλώσσα μπορεί να εκτελείται μόνο σε μια οικογένεια μηχανών

assembly..

- Ο προγραμματισμός σε συμβολική γλώσσα είναι δυσκολότερος και χρειάζεται πολύ περισσότερο χρόνο από οτι για το γράψιμο του ίδιου προγράμματος σε γλώσσα υψηλού επιπέδου.

Ιδεατά Επίπεδα




```
A=B+C;
```

```
LDAC B
MVAC
LDAC C
ADD
STAC A
END
```

```
0000 0001 0000 1110 0000 0000 0000 0011
0000 0001 0000 1111 0000 0000 0000 1000
0000 0010 0001 0000 0000 0000 1111 1111
0000 0000 0000 0000 0000 0001 0000 1001
0000 1010 0000 0000 0000 0000 0000 0000
```

Γιατί assembly..?

- Λόγω απόδοσης
- Λόγω πρόσβασης στη μηχανή

Γνωριμία με τον Relatively Simple CPU Simulator

Ο προσομοιωτής Relatively Simple CPU Simulator

- **Τι είναι προσομοιωτές;**

  Οι προσομοιωτές είναι προγράμματα υπολογιστή που προσπαθούν να συμπεριφέρονται όσο το δυνατόν πιο όμοια γίνεται με ένα φυσικό σύστημα.

- **Τι είναι ο Relatively Simple CPU Simulator**

  Ο Relatively Simple CPU Simulator είναι προσομοιωτής για εκπαιδευτικούς σκοπούς. Επιτρέπει τον χρήστη να προσομοιώσει την ροή δεδομένων στην CPU όταν αυτή ανακαλεί κωδικοποιεί και εκτελεί εντολές. Η προσομοίωση μπορεί να γίνει είτε σε hard-wired control unit ή σε microcoded control unit.

Γνωριμία με τον Relatively Simple CPU Simulator

- **Τι κάνει ο Relatively Simple CPU Simulator .**

  Ο Relatively Simple CPU Simulator μας δίνει πληροφορίες για τα περιεχόμενα των καταχωρητών και της μνήμης μετά την εκτέλεση κάθε εντολής. Μπορεί να έχει πρόσβαση σε 64K bytes μνήμης, μέσω των address pins A[15..0] και των data pins D[7..0]. Αυτή η CPU έχει τους εξής καταχωρητές: τον 8-bit συσσωρευτή AC, τον 8-bit γενικού σκοπού καταχωρητή R.Επίσης έχει 1-bit ενδείκτη μηδενικού αποτελέσματος Z.

- **Τι δεν κάνει ο Relatively Simple CPU Simulator .**

  Ο Relatively Simple CPU Simulator δεν δίνει καμιά πληροφορία, π.χ. για το χρόνο εκτέλεσης της κάθε εντολής (ποιες εντολές εκτελούνται γρήγορα και ποιες αργά), όπως και για άλλες απόψεις του φυσικού συστήματος.

Καταχωρητές..

Carpinelli κεφ. 6

PC μετρητής προγράμματος-program counter (16 bits)
- περιέχει την διεύθυνση της επόμενης εντολής προς εκτέλεση

AC συσσωρευτής-accumulator (8 bits)
- «ενδιάμεσος» για την εκτέλεση των λογικών και αριθμητικών πράξεων από την Αριθμητική – Λογική Μονάδα (ALU)
- περιέχει ένα τμήμα δεδομένων που πρόκειται να επεξεργαστεί η ALU και είναι αποδέκτης του αποτελέσματος της αριθμητικής ή λογικής πράξης

R καταχωρητής γενικού σκοπού-general purpose register (8 bits)
- Για την αποθήκευση οποιονδήποτε δεδομένων του προγράμματος ή διευθύνσεων

Z ενδείκτης μηδενικού αποτελέσματος-zero register (1 bit)
- γίνεται 1 όταν η εντολή που εκτελείται έχει σαν αποτέλεσμα το μηδενισμό του περιεχομένου του συσσωρευτή

Επιπλέον εσωτερικοί καταχωρητές

AR Address Register (16-bit)
- παρέχει μια διεύθυνση στη μνήμη μέσω A[15..0]

DR Data Register (8-bit)
- λαμβάνει εντολές και δεδομένα από την μνήμη μέσω D[7..0]

IR Instruction Register (8-bit):
- αποθηκεύει το τμήμα opcode του κώδικα εντολής που ήρθε από την μνήμη

TR Temporary Register (8-bit)
- αποθηκεύει προσωρινά τα δεδομένα κατά την εκτέλεση εντολών

Το Σύνολο Εντολών (Instruction Set)

- 16 εντολές, 8-bit ο Κώδικας Λειτουργίας (opcode)
- Γ είναι ένας 16-bit τελεστέος (operand) που αναπαριστά μια διεύθυνση μνήμης

NOP	καμιά ενέργεια
LDAC Γ	φόρτωσε στον συσσωρευτή το περιεχόμενο της διεύθ. μνήμης Γ
STAC Γ	αποθήκευσε τα περιεχόμενα του συσσωρευτή στη διεύθ. μνήμη Γ
MVAC	Βάλε τα περιεχόμενα του συσσωρευτή στον R
MOVR	Βάλε τα περιεχόμενα του R στον συσσωρευτή
JUMP Γ	κάνε άλμα χωρίς συνθήκη στη διεύθ. μνήμης Γ
JMPZ Γ	αν Z=0, θέσε το PC στη διεύθ. μνήμη Γ
JPNZ Γ	αν Z=1, θέσε το PC στη διεύθ. μνήμη Γ
ADD	πρόσθεσε AC+R και βάλε το άθροισμα στο AC, ενημέρωσε το Z

Το Σύνολο Εντολών (Instruction Set)

SUB αφαίρεσε AC-R και βάλε το αποτέλεσμα στο AC, ενημέρωσε το Z

INAC Αύξησε περιεχόμενο AC κατά 1 , ενημέρωσε το Z

CLAC Μηδένισε το AC, ενημέρωσε το Z

AND κάνε λογικό AND μεταξύ των bit του AC και του R (bitwise) και βάλε το αποτέλεσμα στο AC, ενημέρωσε το Z

OR κάνε λογικό OR μεταξύ των bit του AC και του R (bitwise) και βάλε το αποτέλεσμα στο AC, ενημέρωσε το Z

XOR κάνε λογικό XOR μεταξύ των bit του AC και του R (bitwise) και βάλε το αποτέλεσμα στο AC, ενημέρωσε το Z

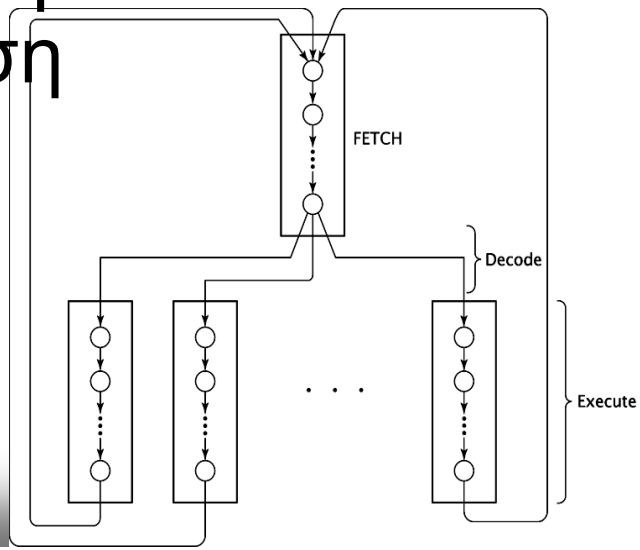
NOT κάνε λογικό NOT μεταξύ των bit του AC και του R (bitwise) και βάλε το αποτέλεσμα στο AC, ενημέρωσε το Z

Προσδιοριστές (directives) και Σχόλια

- Ο RSCPU προσομοιωτής αναγνωρίζει τους παρακάτω directives :
  - ORG **G**
    - Ξεκίνησε την εκτέλεση του προγράμματος στη διευθυνση **G**
  - DB **b**
    - Γράψε την 8-bit (byte) τιμή **b** στην μνήμη
  - DB **w**
    - Γράψε την 16-bit (word) τιμή **w** στη μνήμη
- Το κομμάτι κάθε γραμμής μετά το ; είναι σχόλια

fetch, decode, execute

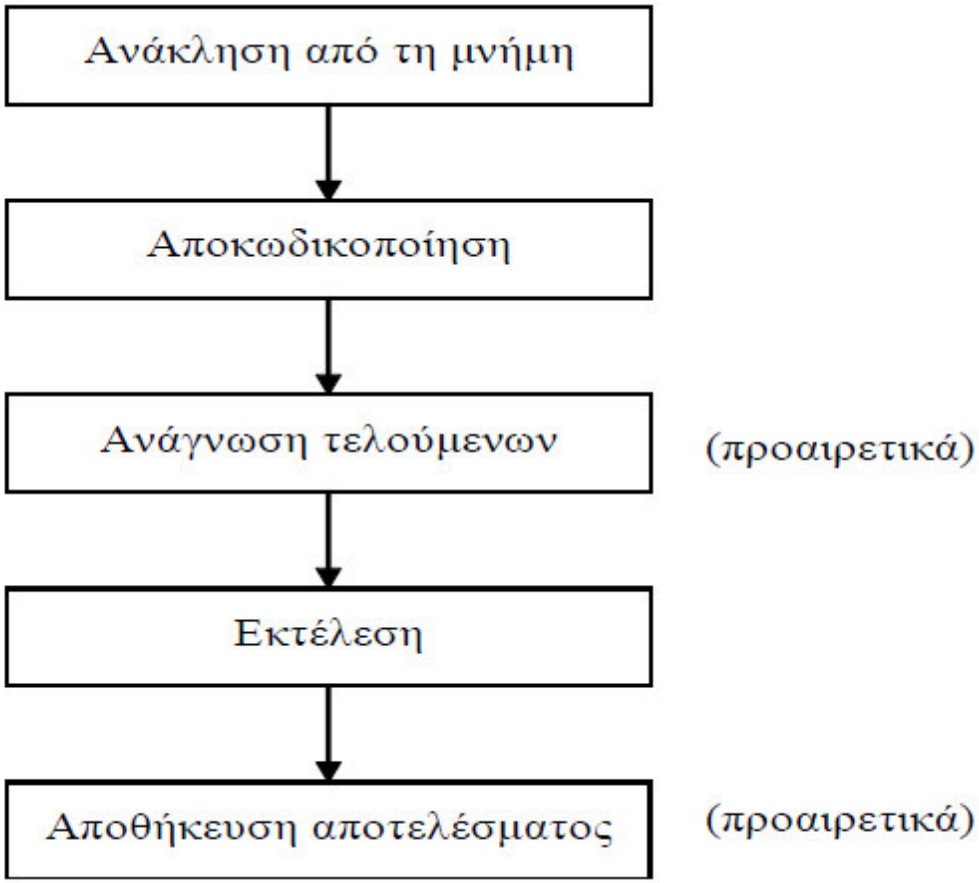
- fetch cycle
  - Αναζήτησε και επέστρεψε την εντολή από την μνήμη, αποθήκευσέ την σε instruction register
- decode cycle
  - Ο instruction register οδηγείτε στον decoder για να ενεργοποιήσει την αντίστοιχη εντολή προς εκτέλεση
- execute cycle
  - Εκτέλεση της εντολής
- … έπειτα fetch την επόμενη



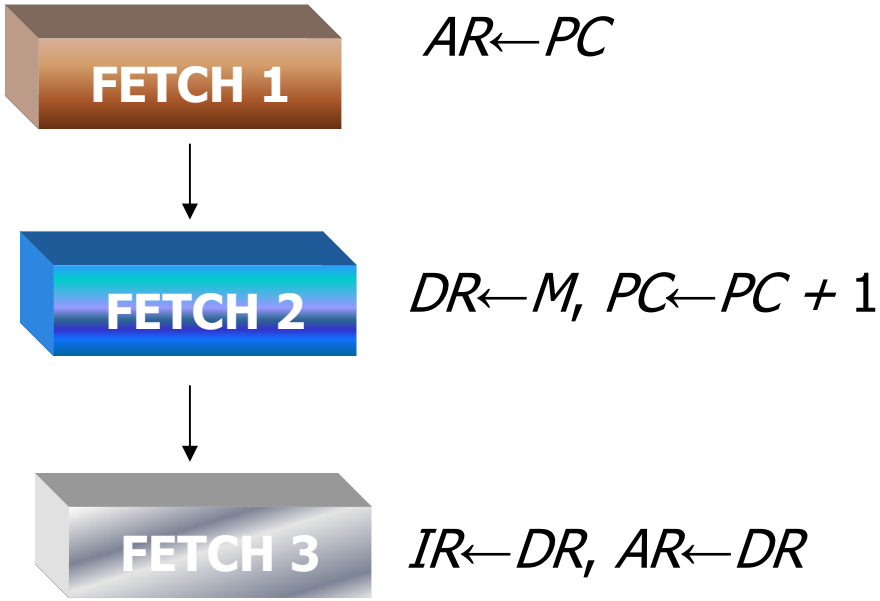
fetch, decode, execute

1. Προσκομίζεται (fetch) η επόμενη εντολή από τη μνήμη στον καταχωρητή εντολών (instruction -> IR).
2. Μεταβάλλεται ο μετρητής προγράμματος, ώστε να δείχνει στην επόμενη εντολή (increment PC).
3. Προσδιορίζεται ο τύπος της εντολής που προσκομίστηκε (decode).
4. Εάν η εντολή ακολουθείτε από ορίσματα προσδιορίζεται η θέση τους στη μνήμη (Address-AR) και προσκομίζονται από τη μνήμη σε ένα καταχωρητή της ΚΜΕ (Data-DT).
5. Εκτελείται η εντολή (execute).
6. Επιστροφή στο βήμα 1, για να αρχίσει η εκτέλεση της επόμενης εντολής. (ατέρμον βρόγχος)

fetch, decode, execute



Fetch cycle: Αναζήτησε και επέστρεψε την εντολή



Fetch cycle: Αναζήτησε και επέστρεψε την εντολή

- Τι είναι μια εντολή;
- Πού είναι η εντολή;
- Τι χρειάζεται για να επιστρέψει;
- Πως ξέρει ο ΗΥ που βρίσκετε;
- Που είναι τοποθετημένη η εντολή;

Opcode	Operand(s)

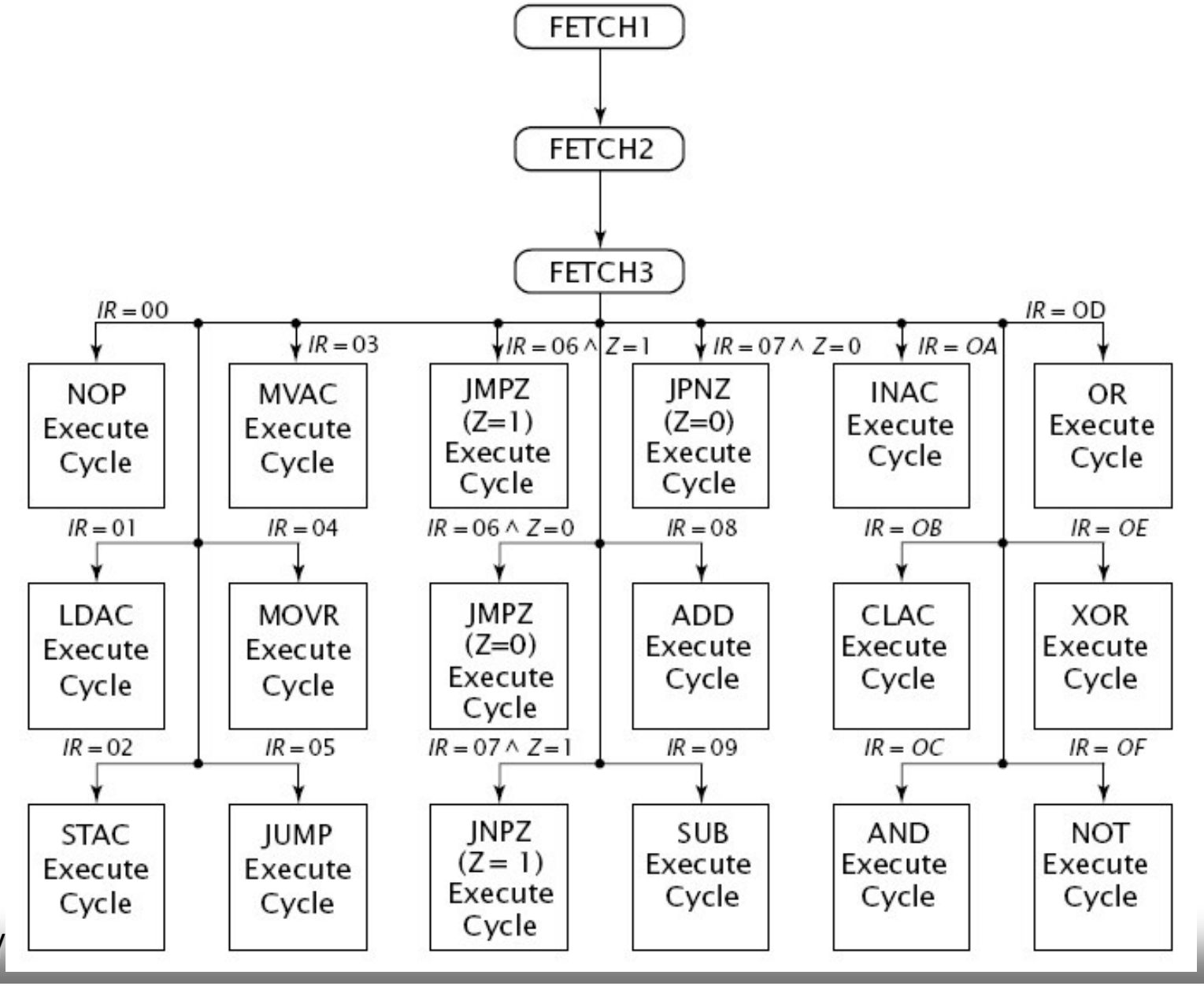
Στη μνήμη

Ταχύτητα, απαιτείται να προσπελάσει κάθε μέρος της εντολής

Program Counter (PC)

Instruction Register (IR)

Fetch and decode cycles



Executing Instructions

LDAC

LDAC1: *DR*←*M*, *PC*←*PC* + 1, *AR*←*AR* + 1
LDAC2: TR←DR, DR←M, PC←PC + 1
LDAC3: AR←DR,TR
LDAC4: DR←M
LDAC5: AC←DR

STAC

STAC1: *DR*←*M*, *PC*←*PC* + 1, *AR*←*AR* + 1
STAC2: *TR*←*DR*, *DR*←*M*, *PC*←*PC* + 1
STAC3: *AR*←*DR*,*TR*
STAC4: *DR*←*AC*
STAC5: *M*←*DR*

Executing Instructions

NOP

NOP1: (No operation)

MVAC και MOVR

MVAC1: *R*←*AC*

MOVR1: *AC*←*R*

JUMP

JUMP1: *DR*←*M*, *AR*←*AR*+1

JUMP2: *TR*←*DR*, *DR*←*M*

JUMP3: *PC*←*DR*,*TR*

Executing Instructions

JMPZ και JPNZ

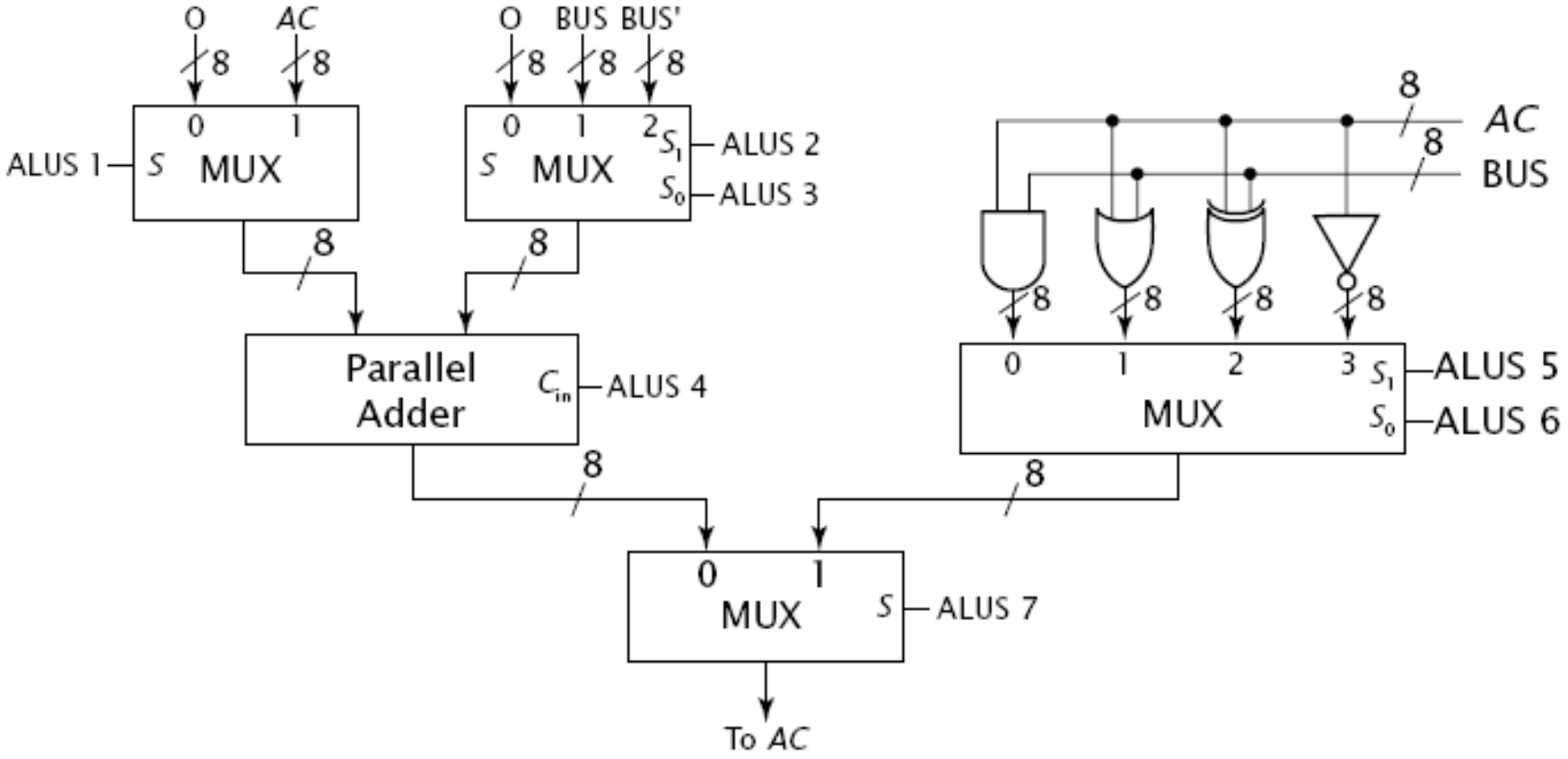
JMPZY1: *DR*←*M*, *AR*←*AR* + 1
JMPZY2: *TR*←*DR*, *DR*←*M*
JMPZY3: *PC*←*DR*, *TR*
JMPZN1: *PC*←*PC* + 1
JMPZN2: *PC*←*PC* + 1

JPNZY1: *DR*←*M*, *AR*←*AR* +1
JPNZY2: *TR*←*DR*, *DR*←*M*
JPNZY3: *PC*←*DR*, *TR*
JPNZN1: *PC*←*PC* + 1
JPNZN2: *PC*←*PC* + 1

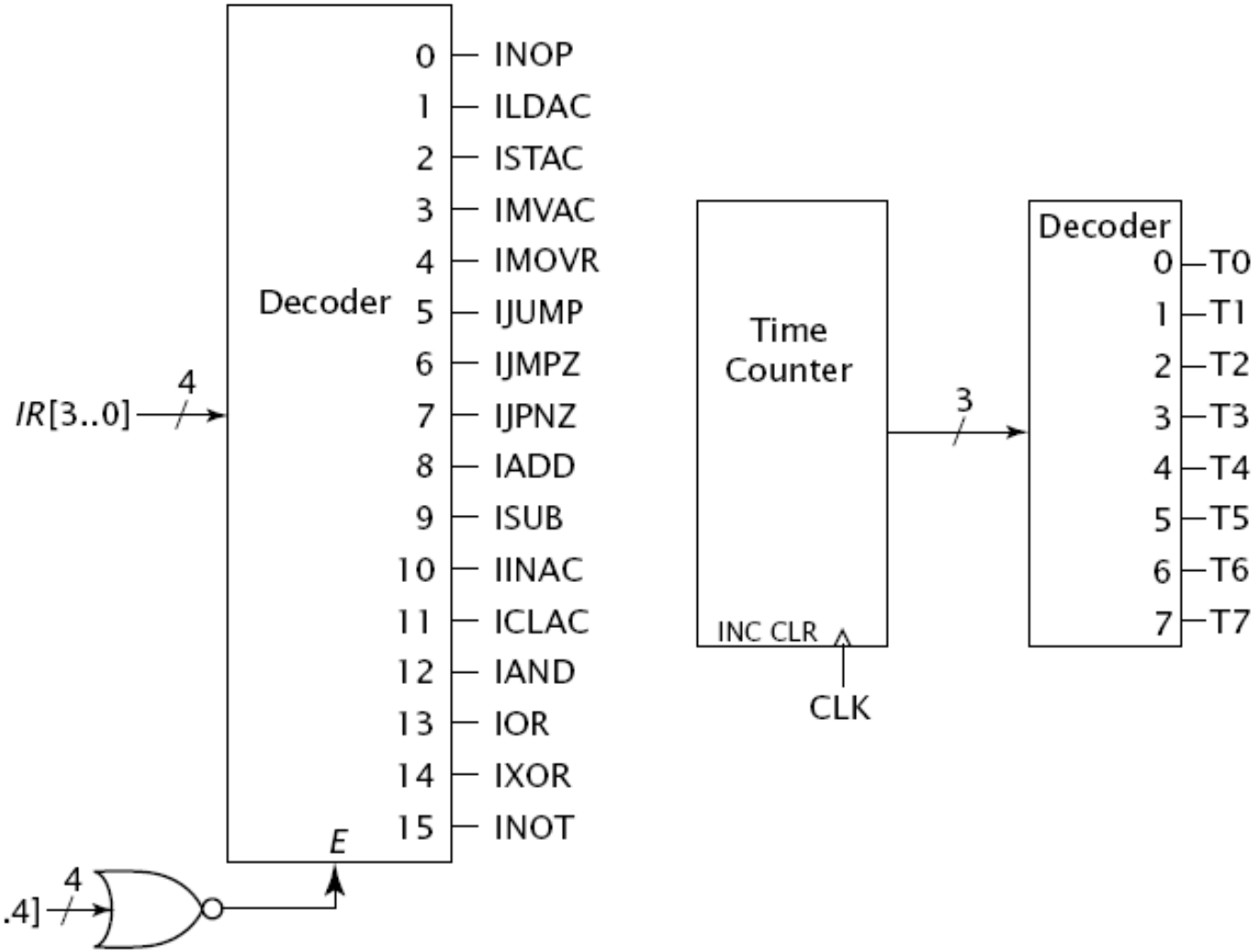
Οι υπόλοιπες εντολές

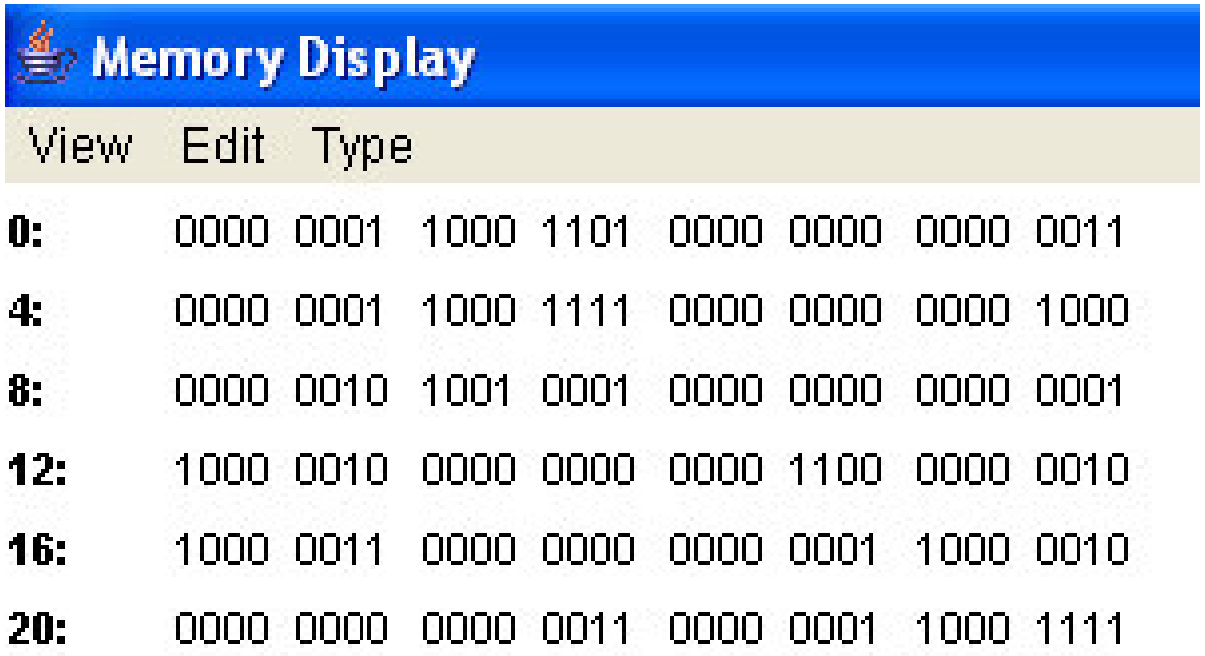
ADD1: $AC \leftarrow AC + R$, IF $(AC + R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
SUB1: $AC \leftarrow AC - R$, IF $(AC - R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
INAC1: $AC \leftarrow AC + 1$, IF $(AC + 1 = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
CLAC1: $AC \leftarrow 0$, $Z \leftarrow 1$
AND1: $AC \leftarrow AC \wedge R$, IF $(AC \wedge R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
OR1: $AC \leftarrow AC \vee R$, IF $(AC \vee R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
XOR1: $AC \leftarrow AC \oplus R$, IF $(AC \oplus R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
NOT1: $AC \leftarrow AC'$, IF $(AC' = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$

A Relatively Simple ALU



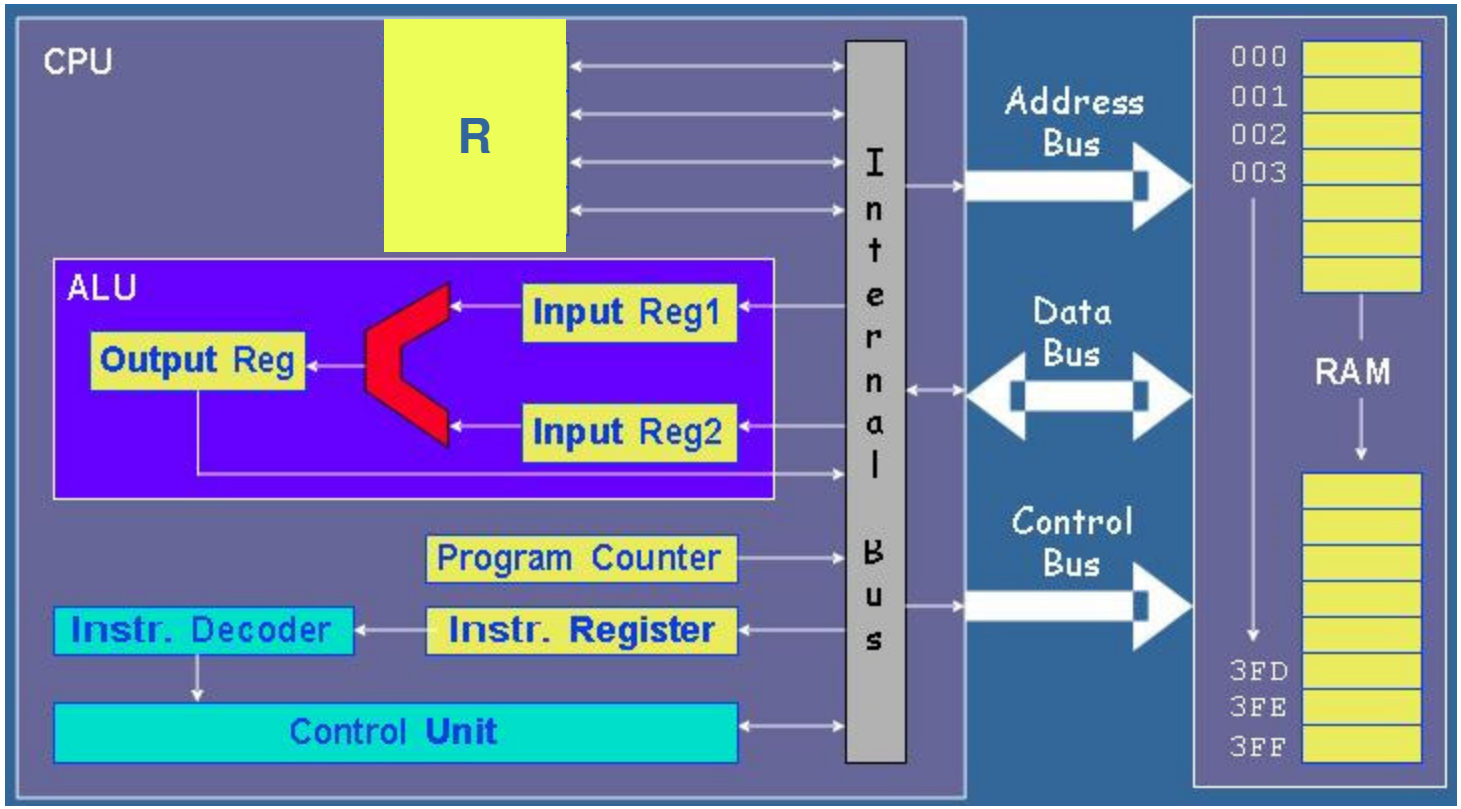
Hardwired control unit for the Relatively Simple CPU

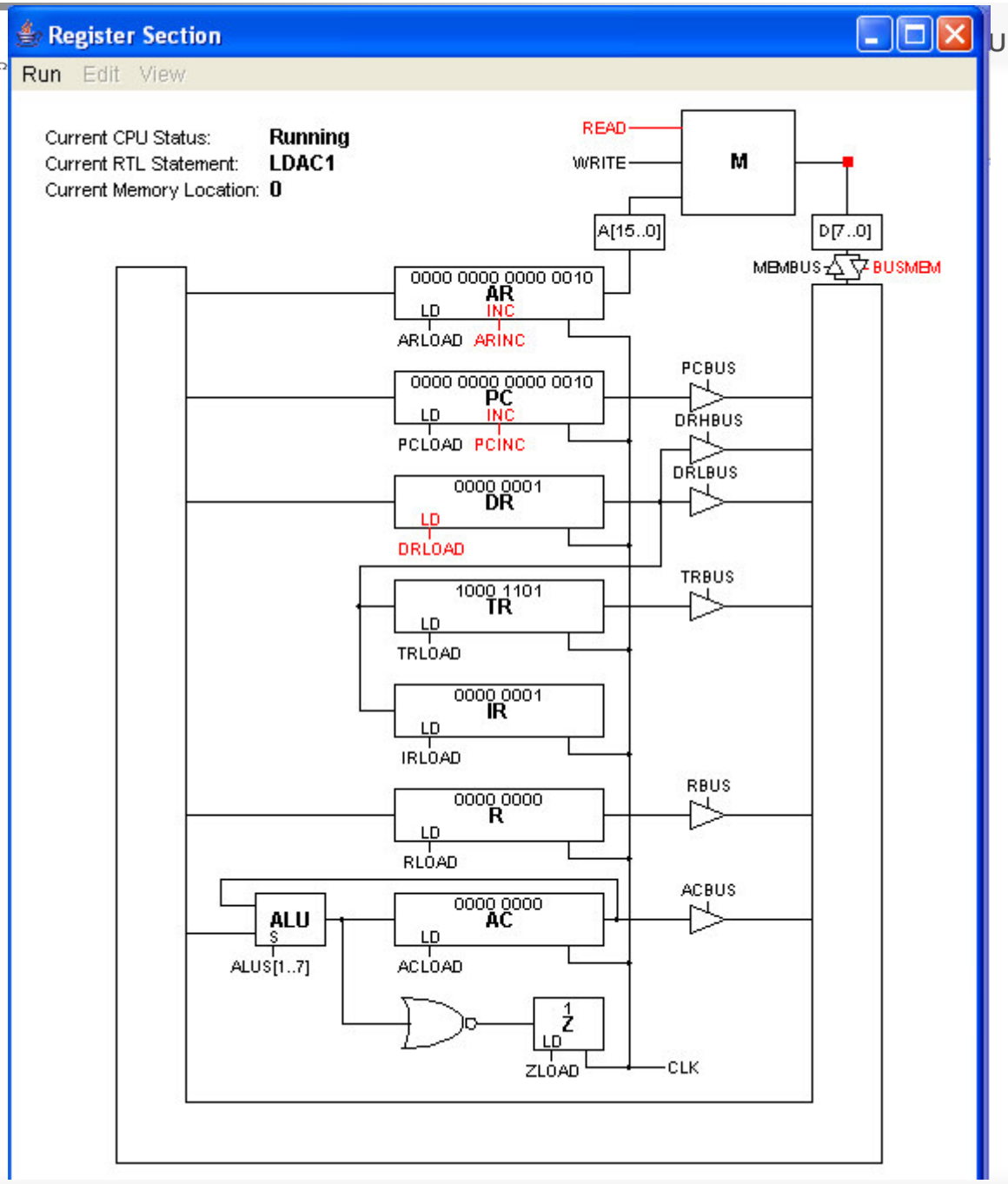


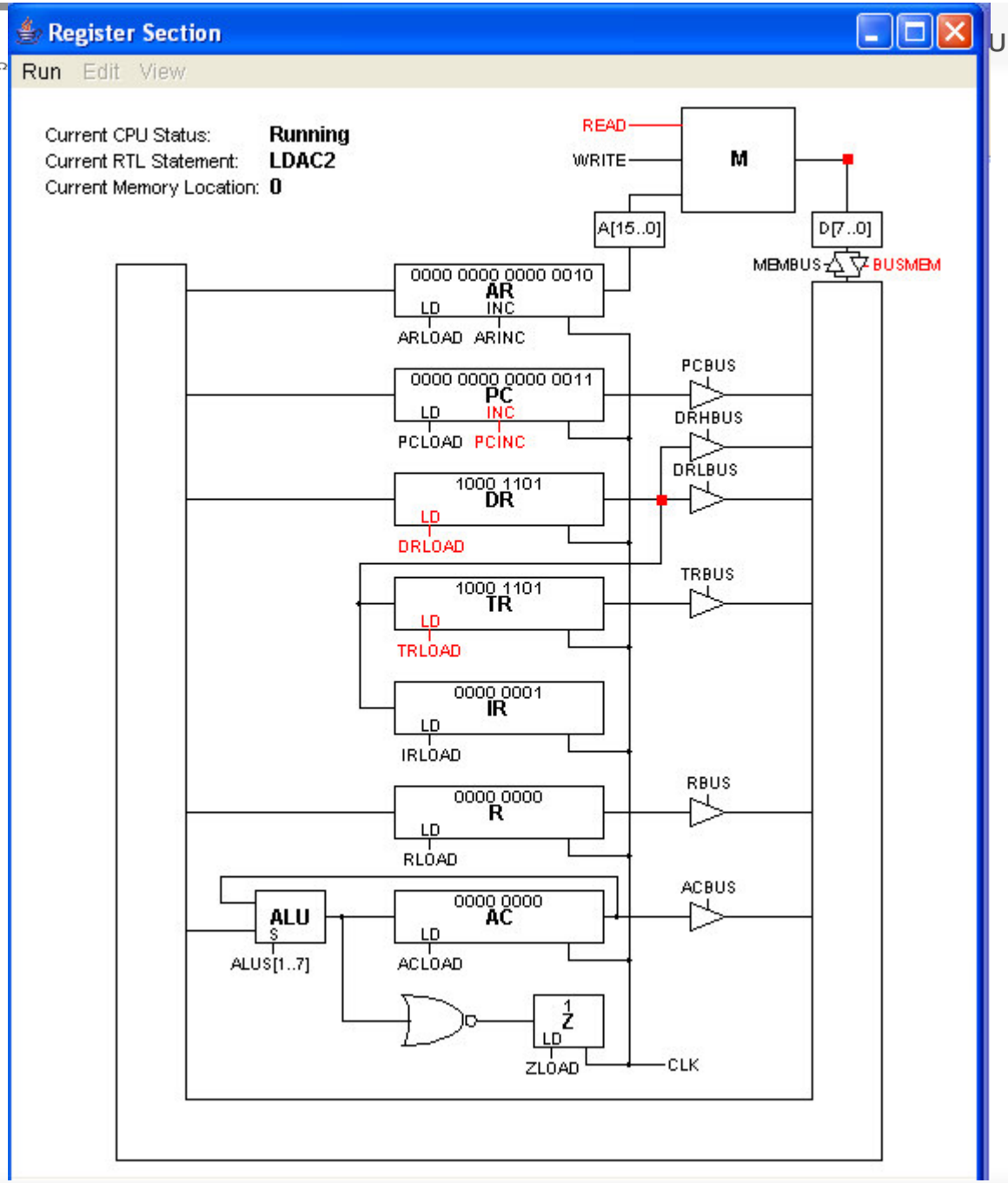


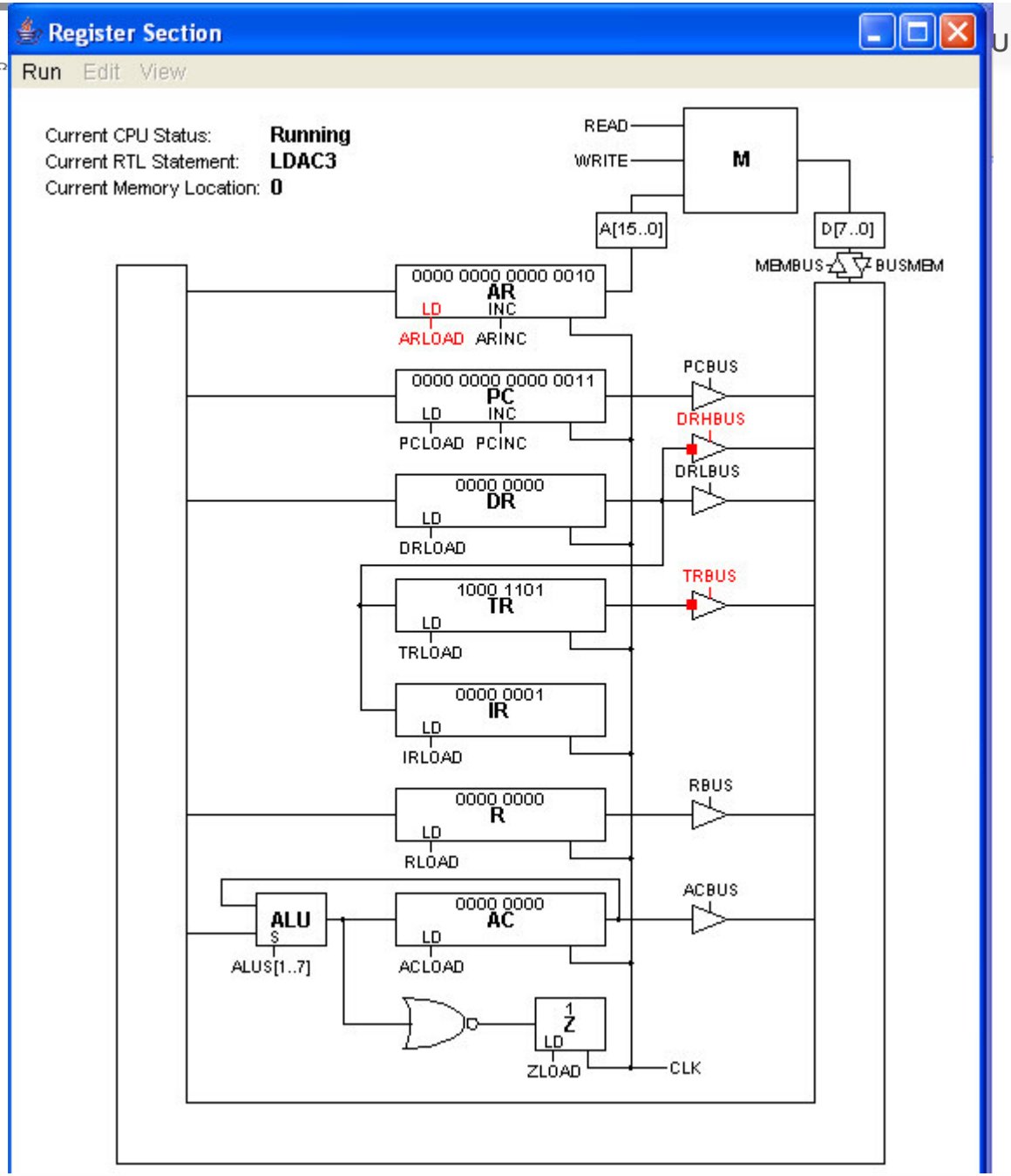
LDAC 141
MVAC
LDAC 143
ADD
STAC 145
END

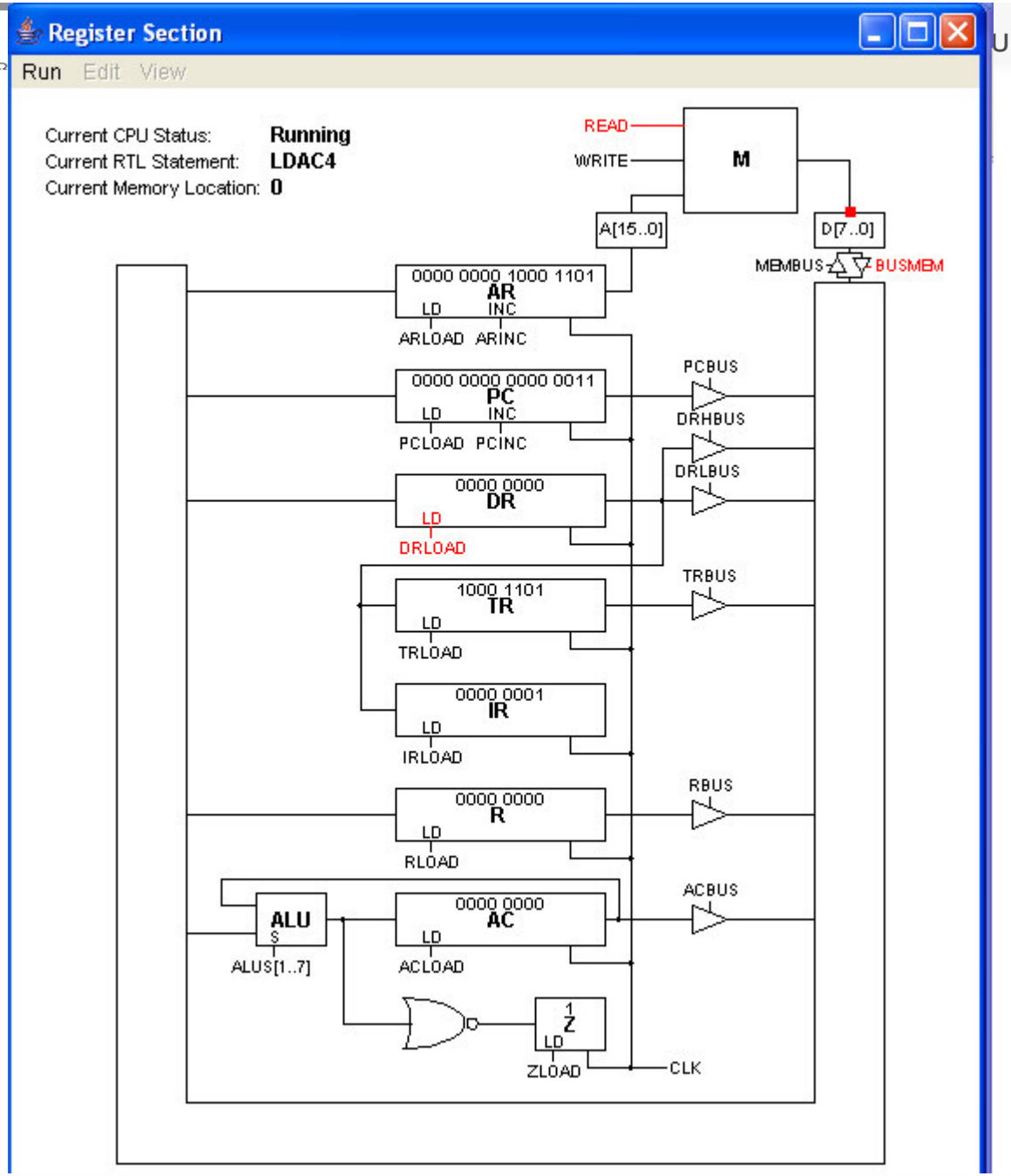
Οργάνωση CPU

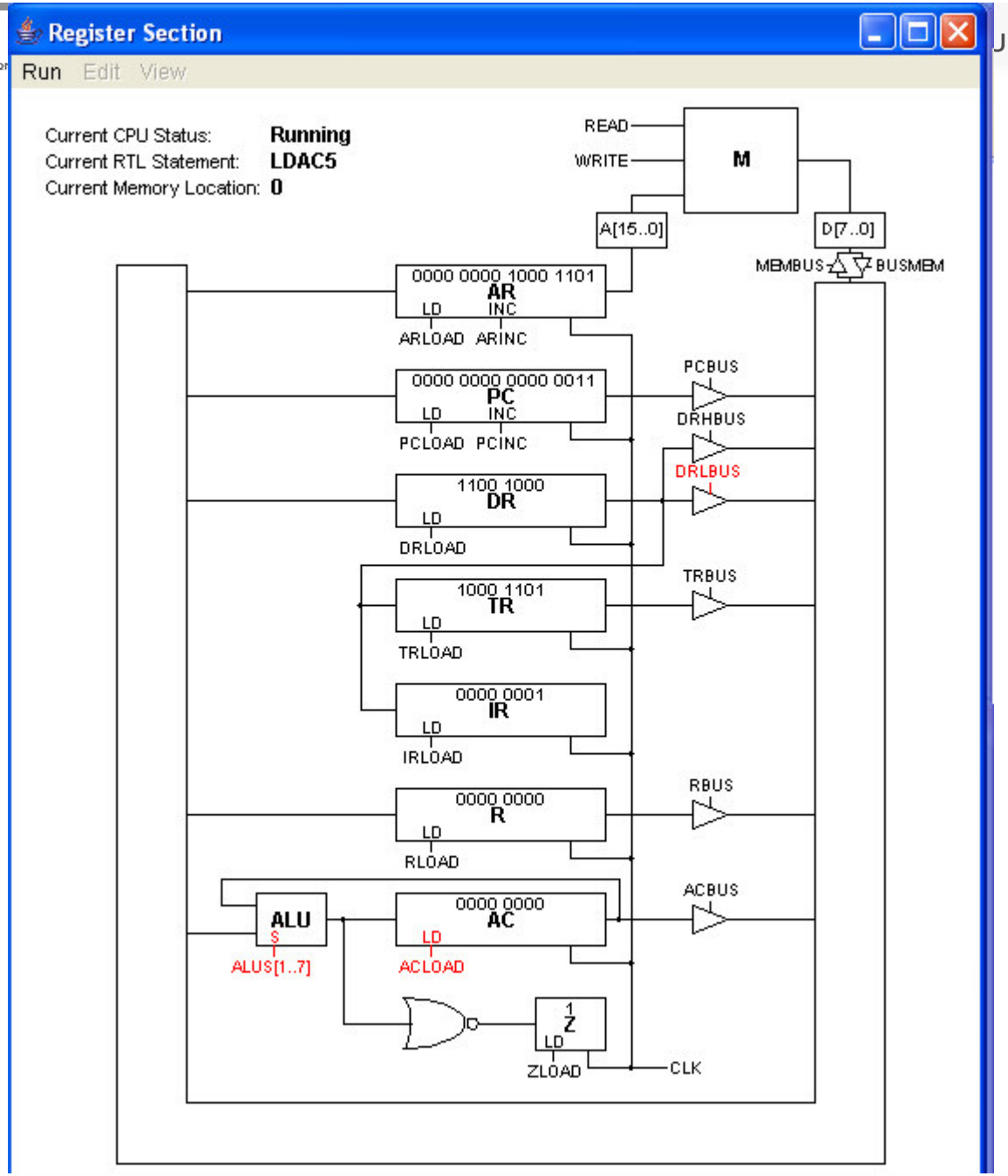


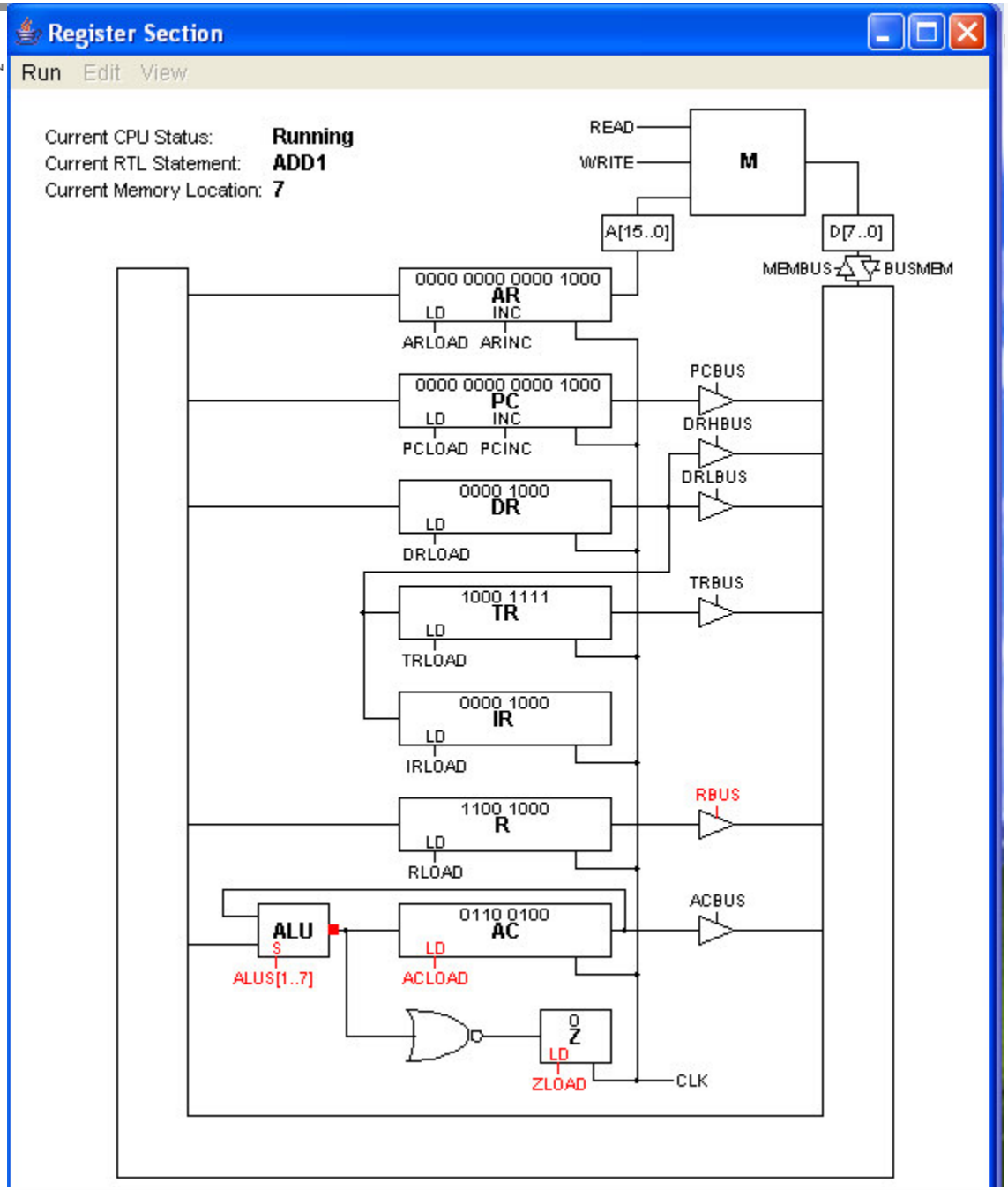

Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: LDAC1
Current Memory Location: 0
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 0000 0010
AR
LD
INC
ARLOAD
ARINC
0000 0000 0000 0010
PC
LD
INC
PCLOAD
PCINC
0000 0001
DR
LD
DRLOAD
1000 1101
TR
LD
TRLOAD
0000 0001
IR
LD
IRLOAD
0000 0000
R
LD
RLOAD
ALU
S
ALUS[1..7]
0000 0000
AC
LD
ACLOAD
1
Z
LD
ZLOAD
CLK
PCBUS
DRHBUS
DRLBUS
TRBUS
RBUS
ACBUS

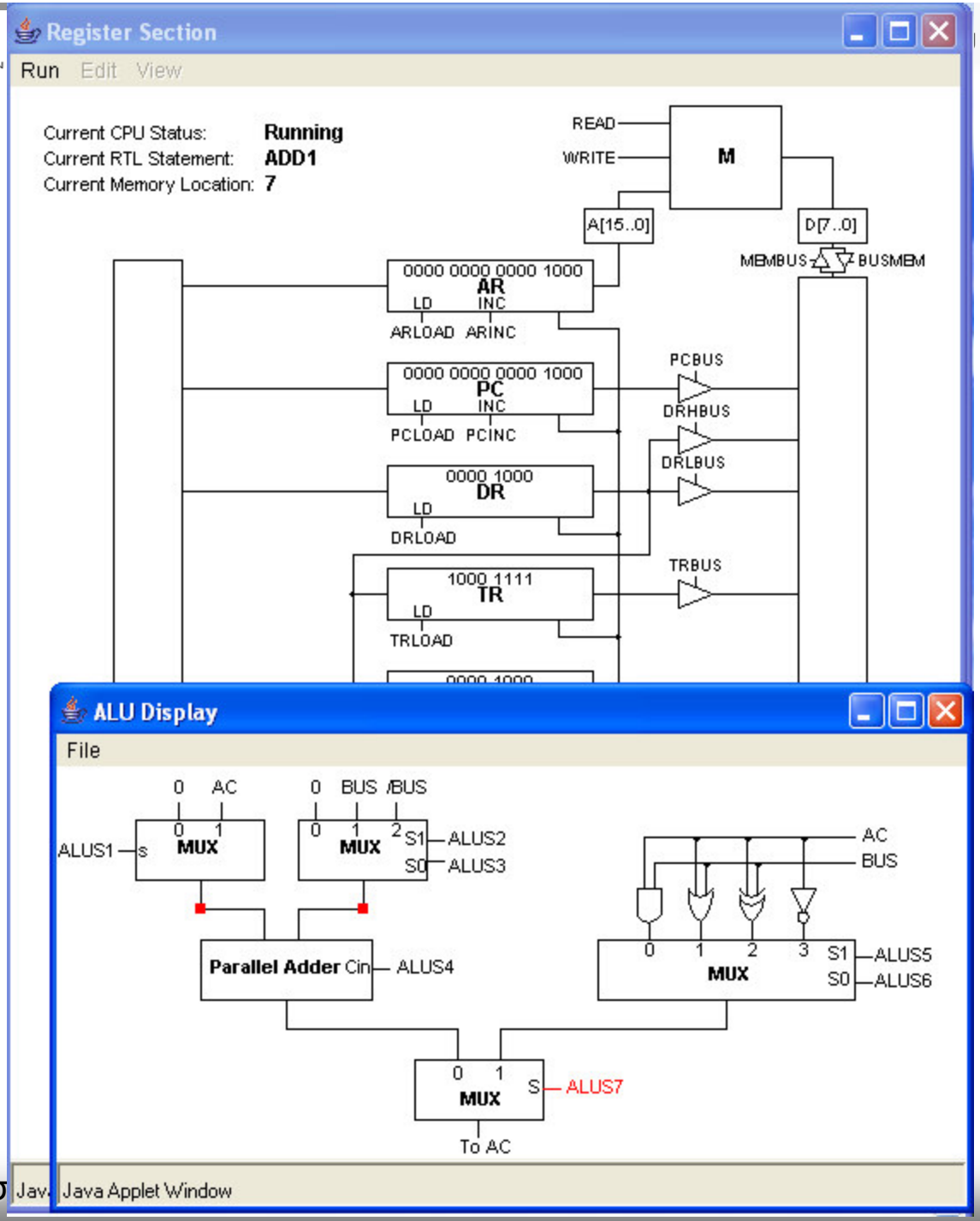

Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: LDAC2
Current Memory Location: 0
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 0000 0010
AR
LD INC
ARLOAD ARINC
0000 0000 0000 0011
PC
LD INC
PCLOAD PCINC
1000 1101
DR
LD
DRLOAD
1000 1101
TR
LD
TRLOAD
0000 0001
IR
LD
IRLOAD
0000 0000
R
LD
RLOAD
ALU
S
ALUS[1..7]
0000 0000
AC
LD
ACLOAD
1
Z
LD
ZLOAD
CLK
PCBUS
DRHBUS
DRLBUS
TRBUS
RBUS
ACBUS


Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: LDAC3
Current Memory Location: 0
READ
WRITE
M
A[15..0]
D[7..0]
MBMBUS
BUSMBM
0000 0000 0000 0010
AR
LD INC
ARLOAD ARINC
PCBUS
0000 0000 0000 0011
PC
LD INC
PCLOAD PCINC
DRHBUS
DRLBUS
0000 0000
DR
LD
DRLOAD
TRBUS
1000 1101
TR
LD
TRLOAD
0000 0001
IR
LD
IRLOAD
RBUS
0000 0000
R
LD
RLOAD
ACBUS
ALU
S
ALUS[1..7]
0000 0000
AC
LD
ACLOAD
1
Z
LD
ZLOAD
CLK

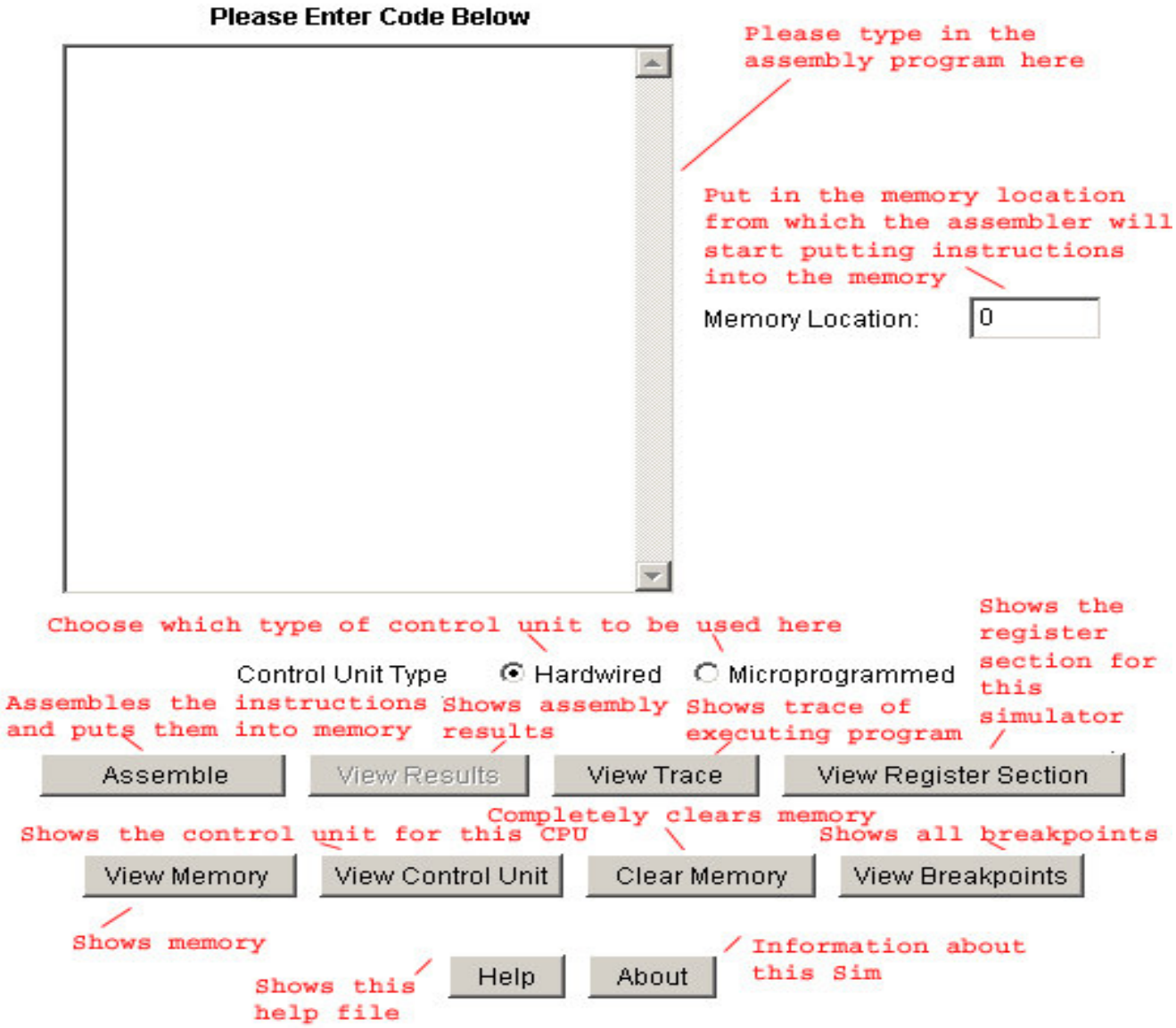

Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: LDAC4
Current Memory Location: 0
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 1000 1101
AR
LD INC
ARLOAD ARINC
0000 0000 0000 0011
PC
LD INC
PCLOAD PCINC
0000 0000
DR
LD
DRLOAD
1000 1101
TR
LD
TRLOAD
0000 0001
IR
LD
IRLOAD
0000 0000
R
LD
RLOAD
ALU
S
ALUS[1..7]
0000 0000
AC
LD
ACLOAD
1
Z
LD
ZLOAD
CLK
PCBUS
DRHBUS
DRLBUS
TRBUS
RBUS
ACBUS


Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: LDAC5
Current Memory Location: 0
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 1000 1101
AR
LD INC
ARLOAD ARINC
PCBUS
0000 0000 0000 0011
PC
LD INC
PCLOAD PCINC
DRHBUS
DRLBUS
1100 1000
DR
LD
DRLOAD
TRBUS
1000 1101
TR
LD
TRLOAD
0000 0001
IR
LD
IRLOAD
RBUS
0000 0000
R
LD
RLOAD
ACBUS
ALU
S
ALUS[1..7]
0000 0000
AC
LD
ACLOAD
1
Z
LD
ZLOAD
CLK


Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: ADD1
Current Memory Location: 7
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 0000 1000
AR
LD INC
ARLOAD ARINC
0000 0000 0000 1000
PC
LD INC
PCLOAD PCINC
0000 1000
DR
LD
DRLOAD
1000 1111
TR
LD
TRLOAD
0000 1000
IR
LD
IRLOAD
1100 1000
R
LD
RLOAD
ALU
S
ALUS[1..7]
0110 0100
AC
LD
ACLOAD
0
Z
LD
ZLOAD
CLK
PCBUS
DRHBUS
DRLBUS
TRBUS
RBUS
ACBUS


Register Section
Run Edit View
Current CPU Status: Running
Current RTL Statement: ADD1
Current Memory Location: 7
READ
WRITE
M
A[15..0]
D[7..0]
MEMBUS
BUSMEM
0000 0000 0000 1000
AR
LD INC
ARLOAD ARINC
0000 0000 0000 1000
PC
LD INC
PCLOAD PCINC
0000 1000
DR
LD
DRLOAD
1000 1111
TR
LD
TRLOAD
PCBUS
DRHBUS
DRLBUS
TRBUS
ALU Display
File
0 AC
0 BUS /BUS
MUX
ALUS1
ALUS2
ALUS3
Parallel Adder Cin
ALUS4
AC
BUS
ALUS5
ALUS6
ALUS7
To AC
Java Applet Window

Relatively Simple CPU Simulator



http://media.pearsoncmg.com/aw/aw_carpinel_compsys_1/rscpu/web.html

Κώδικας γνωριμίας

Πρόβλημα: Πρόσθεσε τα περιεχόμενα των διευθύνσεων μνήμης 20 και 21, αποθήκευσε τα αποτελέσματα στην διεύθυνση μνήμης 22
Λάβετε υπόψιν ότι:

- Μπορεί και προσθέτει αριθμούς που είναι στον AC και στον R;Τα αποτελέσματα καταλήγουν στον AC
- Περιεχόμενα μνήμης μπορούν να μεταφερθούν μόνο στον AC, όχι άμεσα στον R, αλλά μπορούμε να μεταφέρουμε τιμές από και προς μεταξύ των AC και R
- Μόνο η τιμή του AC μπορεί να μεταφερθεί στην μνήμη

Επομένως, μπορούμε να γράψουμε τον πρόγραμμά μας κάπως έτσι :

```
LDAC 20
MVAC
LDAC 21
ADD   ; m[20]+m[21]
STAC 22
JUMP 65535 ; stop
```

Και μπορούμε να αποθηκεύσουμε test values στην μνήμη κάπως έτσι:

```
ORG 20
DB 6
DB 9
```

Λίγο πιο δύσκολος κώδικας …

Πρόβλημα: δίνοντας έναν ακέραιο θετικό αριθμό n, υπολογίστε το άθροισμα όλων των θετικών ακεραίων που είναι μικρότεροι ή ίσοι του n

π.χ,. δίνοντας 4, compute 4+3+2+1

Αρχικός αλγόριθμος

```
1  sum = 0
2  counter = 4
3  sum = sum + counter
4  counter = counter - 1
5  if counter != 0: go to line 3
6  stop
```

Τι γίνετε βήμα-βήμα..

```
1  sum = 0
2  counter = 4
3  sum = sum + counter
          a. load AC with counter
          b. copy AC to R
          c. load AC with sum
          d. add
          e. store AC in sum
4  counter = counter - 1
          a. load AC with a memory location containing a 1
          b. copy AC to R
          c. load AC with counter
          d. subtract
          e. store AC in counter
5  if counter != 0: go to line 3 (3b)
6  stop
```

σε Assemply..

```
ORG 36     ; DATA STARTS at location 36
DB 4       ; counter (location 36)
DB 1       ; decrement “memory location containing a 1” (37)
DB 0       ; sum (38)
ORG 0      ; PROGRAM STARTS at location 0
LDAC 36    ; load AC with counter
MVAC       ; copy AC to R (top of loop, location 3)
LDAC 38    ; load AC with sum
ADD        ; add AC and R (sum+counter), result in AC
STAC 38    ; store AC in sum
LDAC 37    ; load AC with decrement
MVAC       ; copy AC to R
LDAC 36    ; load AC with counter
SUB        ; subtract AC from R (counter -1), result in AC
STAC 36    ; store AC in counter
JPNZ 3      ; if AC not zero, to go location 3
JUMP 65535 ; otherwise, stop
```


CAN'T YOU DO
ANYTHING RIGHT?
GLASBERGEN
Copyright © Randy Glasbergen. www.glasbergen.com