ΕΡΓΑΣΙΑ 2: Απόδοση διοχέτευσης (pipeline) χρησιμοποιώντας τον προσομοιωτή WinMips64

ΜΕΡΟΣ Α: Γνωριμία με τον προσομοιωτή WinMips64

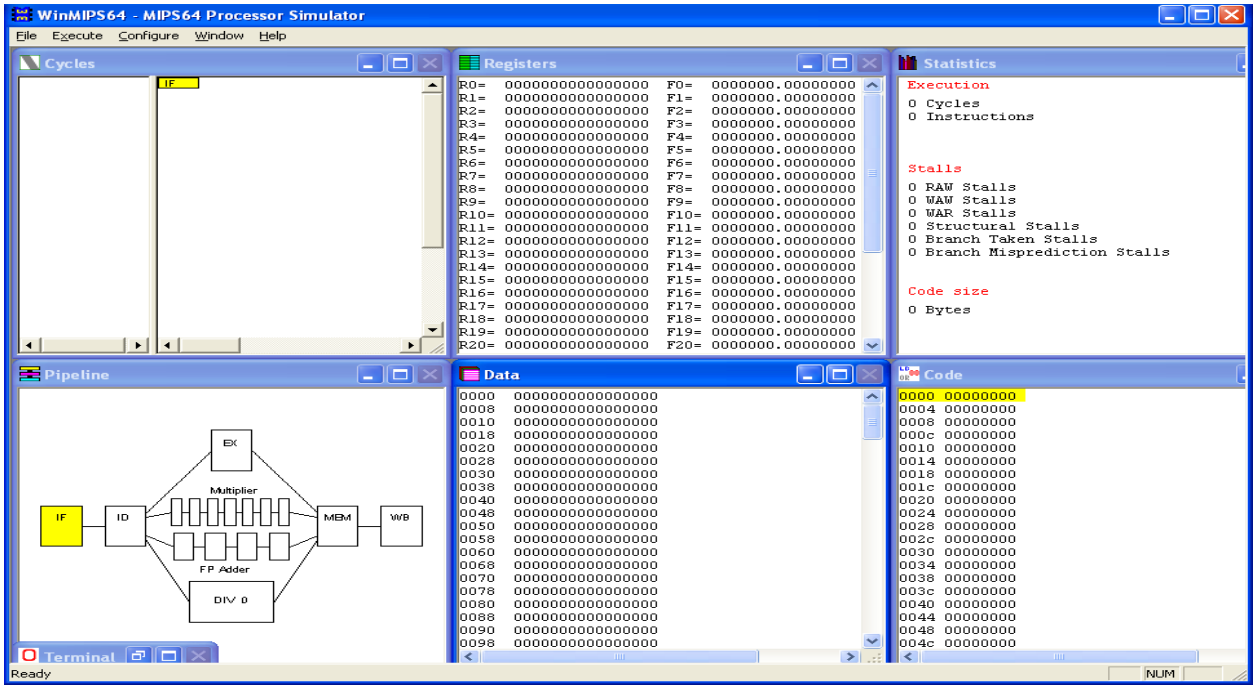
- Είναι ένας παραθυρικός προσομοιωτής για τη μελέτη του μηχανισμού της διοχέτευσης (pipelining) του 64-bit επεξεργαστή MIPS64
- Βασίζεται στο βιβλίο: «Computer Architecture - a Quantitative Approach, by Hennessy & Patterson, 5th edition»
- Μπορείτε να τον κατεβάσετε από την ιστοσελίδα: http://indigo.ie/~mscott/

Το περιβάλλον του WinMIPS64:

✓ Μπάρα των μενού
✓ Ένα κεντρικό παράθυρο με 7 εσωτερικά υποπαράθυρα:
- Pipeline (Διοχέτευση 5 σταδίων)
- Code (Κώδικας)
- Data (Δεδομένα)
- Registers (Καταχωρητές)
- Statistics (Στατιστικά)
- Cycles (Διάγραμμα κύκλων ρολογιού)
- Terminal (Κονσόλα-Τερματικό)

✓ Status line (Γραμμή κατάστασης)

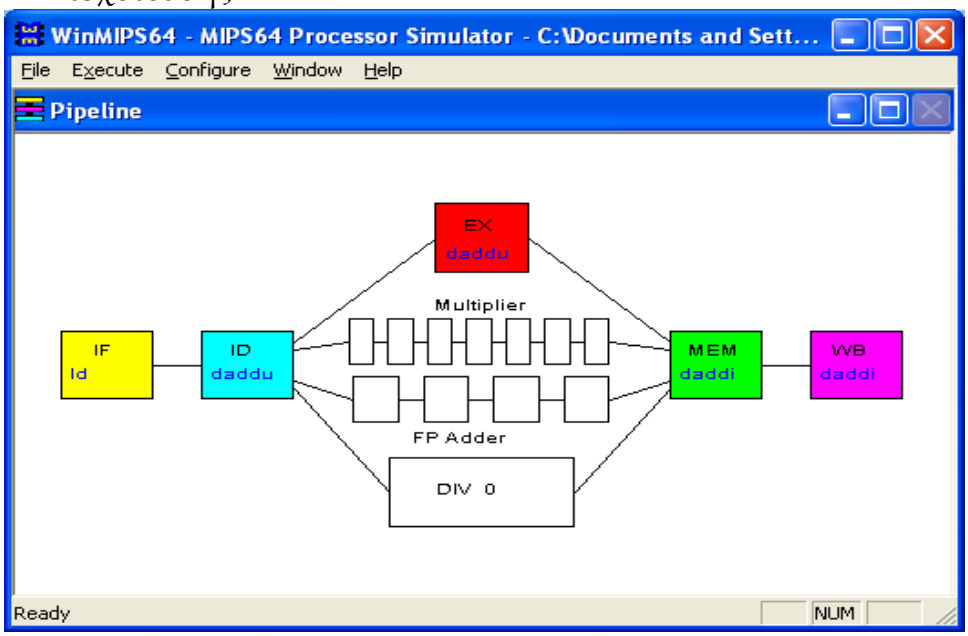
WinMIPS64 – Screenshot:



Το παράθυρο: Pipeline:

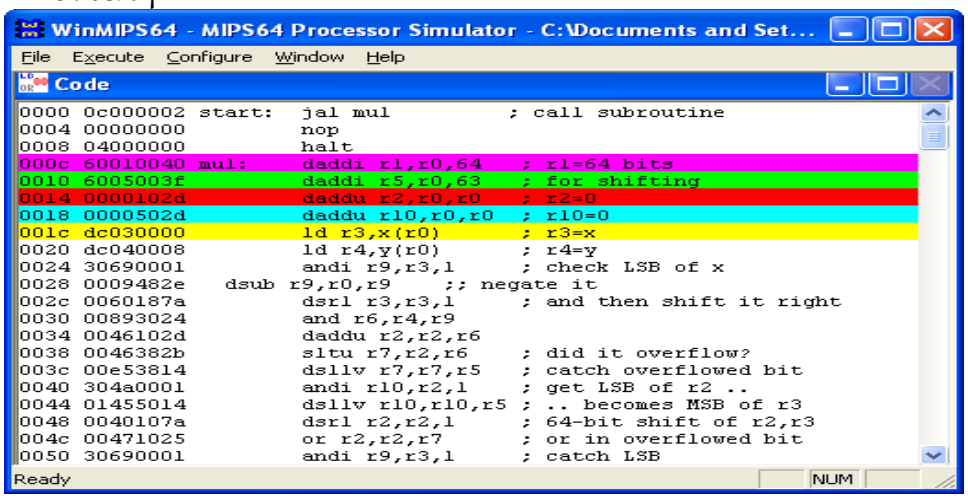
Δείχνει:

- Σχηματική αναπαράσταση των 5 σταδίων της διοχέτευσης του επεξεργαστή MIPS64 και των μονάδων για λειτουργίες κινητής υποδιαστολής (πρόσθεση/αφαίρεση, πολλαπλασιασμός και διαίρεση)
- Με διαφορετικό χρώμα, ποια εντολή βρίσκεται σε κάθε στάδιο της διοχέτευσης



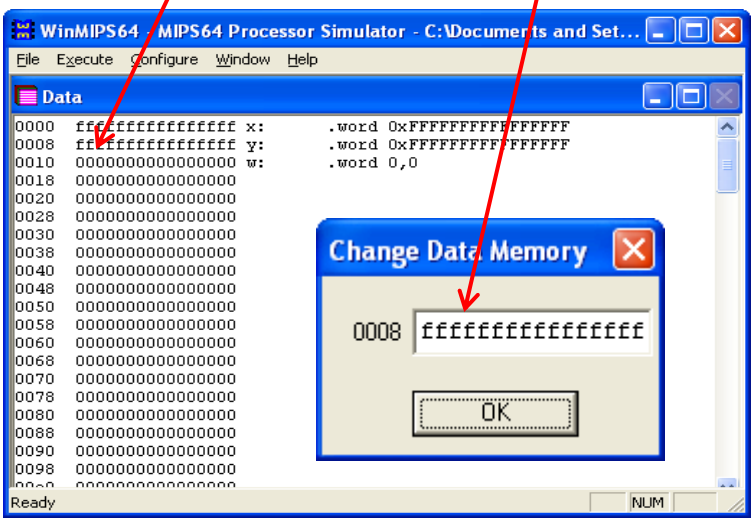
Το παράθυρο: Code:

- ✓ Αναπαράσταση της μνήμης εντολών σε 3 στήλες, δείχνοντας από αριστερά προς τα δεξιά:
  - την διεύθυνση byte
  - την αναπαράσταση της εντολής σε κώδικα μηχανής (hex-4 byte)
  - Την εντολή assembly
- ✓ Διπλό αριστερό click σε μια εντολή, θέτει & αναιρεί σημείο διακοπής (breakpoint)
- ✓ Τα χρώματα δείχνουν το στάδιο της διοχέτευσης που βρίσκεται η κάθε εντολή



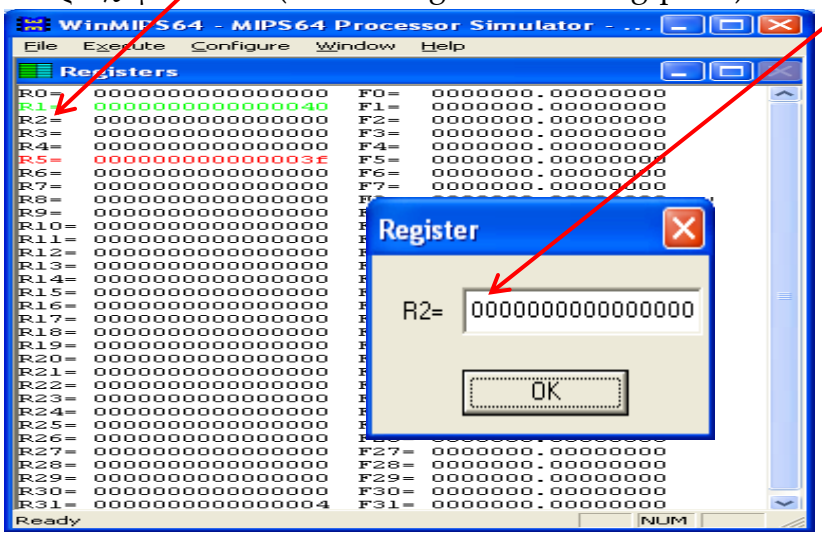
Το παράθυρο: Data:

- ✓ Αναπαράσταση της μνήμης δεδομένων σε 3 στήλες, δείχνοντας από αριστερά προς τα δεξιά:
  - την διεύθυνση byte
  - την αναπαράσταση των δεδομένων σε κώδικα μηχανής (hex-8 byte)
  - τις αντίστοιχες οδηγίες (directives)
- ✓ Διπλό αριστερό (δεξί) click εμφανίζει και δίνει τη δυνατότητα επεξεργασίας μιας ακεραίας (τιμής κινητής υποδιαστολής) τιμής



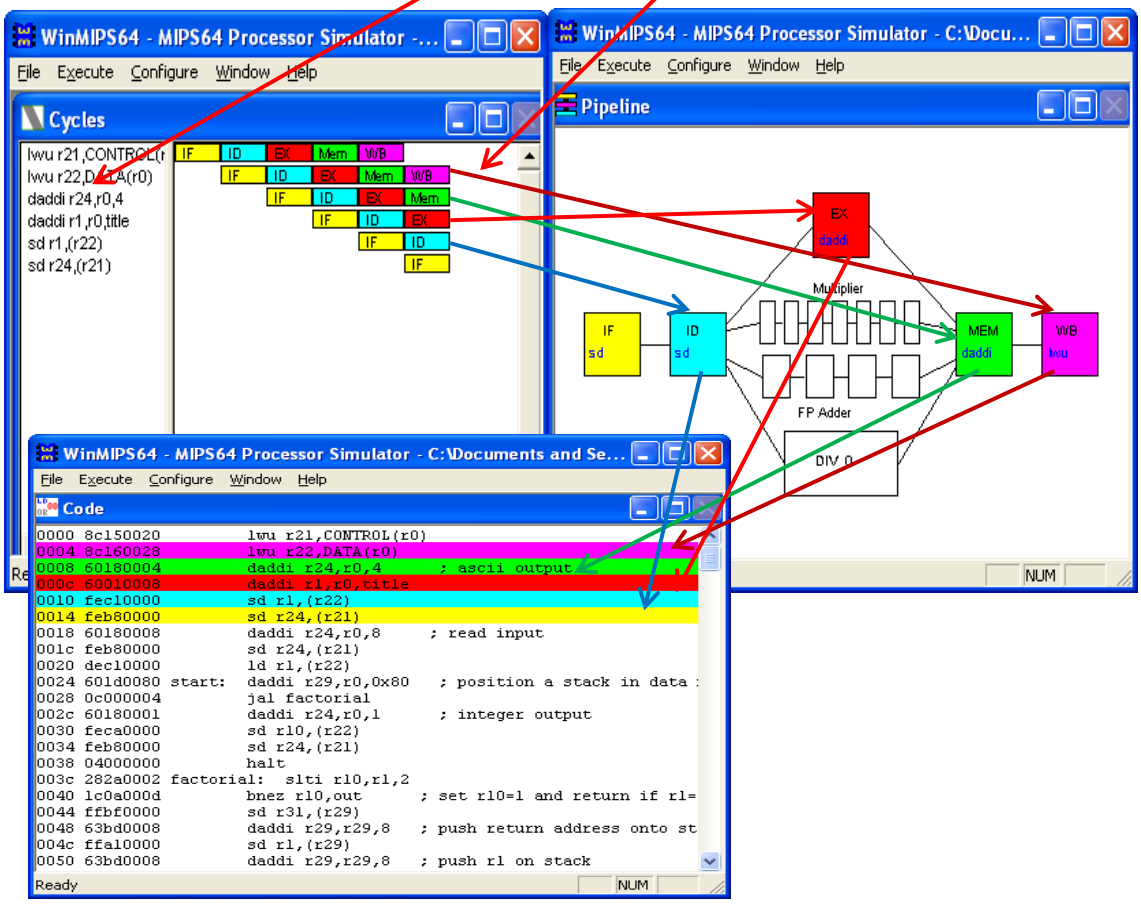
Το παράθυρο: Registers:

- ✓ Δείχνει τα ονόματα των καταχωρητών και τα περιεχόμενά τους.
- ✓ Ανάλογα με τον χρωματισμό τους:
  - **Γκρι**: κάποια εντολή πρόκειται να εγγράψει σε αυτόν τον καταχωρητή
  - Οποιοδήποτε άλλο χρώμα (**κόκκινο, πράσινο** κτλ): το χρώμα υποδεικνύει το στάδιο της διοχέτευσης στο οποίο η τιμή είναι διαθέσιμη για προώθηση (forwarding)
- ✓ Με διπλό αριστερό click πάνω σε κάποιον καταχωρητή,που δεν είναι σε διαδικασία να εγγραφεί ή προώθησης, μπορούμε να αλλάξουμε το περιεχόμενό του (64-bit integer και floating-point)

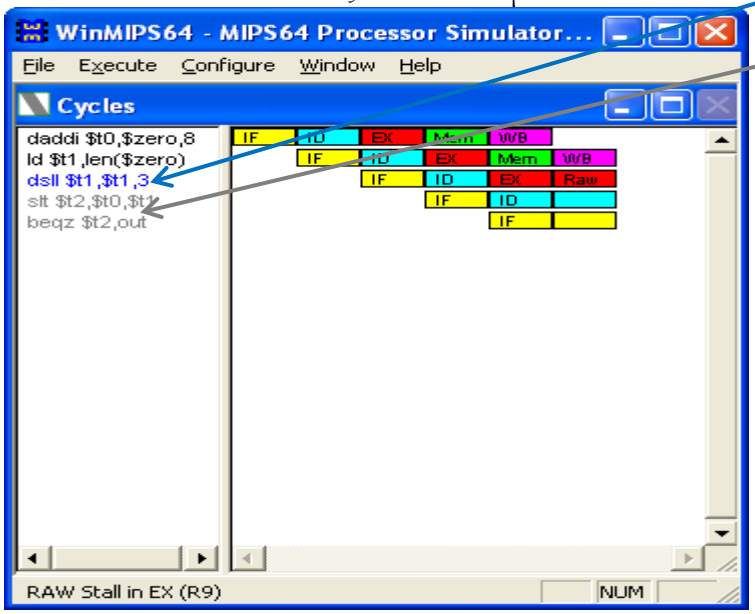


Το παράθυρο: Cycles:

Αποτελείται από 2 υποπαράθυρα (εντολές & διοχέτευση). Σε συνδυασμό με τα υπόλοιπα παράθυρα αναπαριστά την χρονική συμπεριφορά της διοχέτευσης.



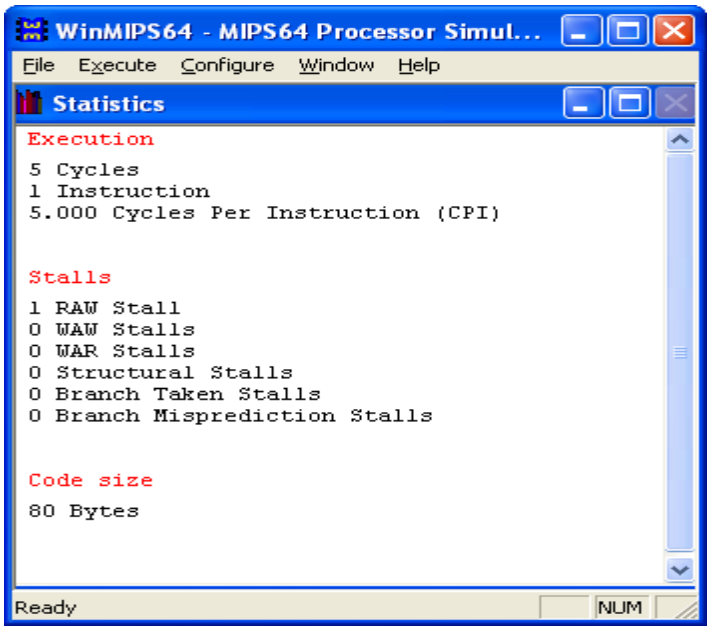
Όταν μία εντολή προκαλεί πάγωμα (stall) χρωματίζεται μπλε. Εντολές που δεν εκτελούνται ως αποτέλεσμα κάποιου stall χρωματίζονται γκρι.



Το παράθυρο: Statistics:

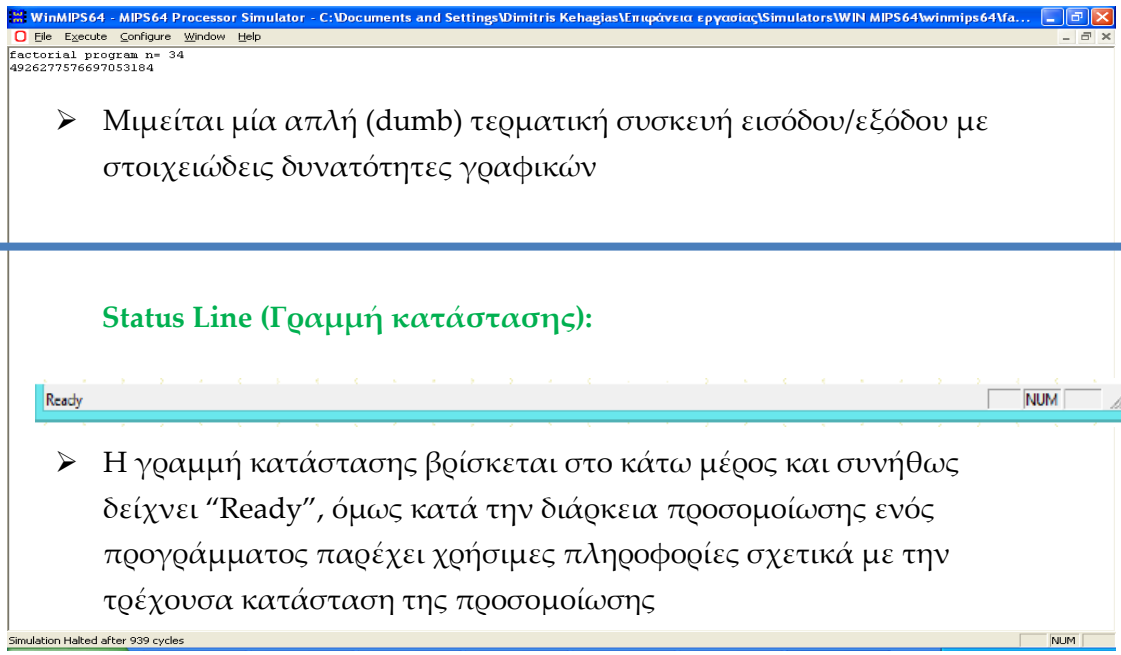
✓ Παρέχει στατιστικά σχετικά με:

- τον αριθμό των κύκλων προσομοίωσης
- τις εντολές
- το μέσο αριθμό κύκλων ανά εντολή (CPI)
- τους τύπους & τον αριθμό των stalls
- τον αριθμό των υπό συνθήκη διακλαδώσεων και των εντολών Load/Store



Το παράθυρο: Terminal:

Μιμείται μία απλή (dumb) τερματική συσκευή εισόδου/εξόδου με στοιχειώδεις δυνατότητες γραφικών.



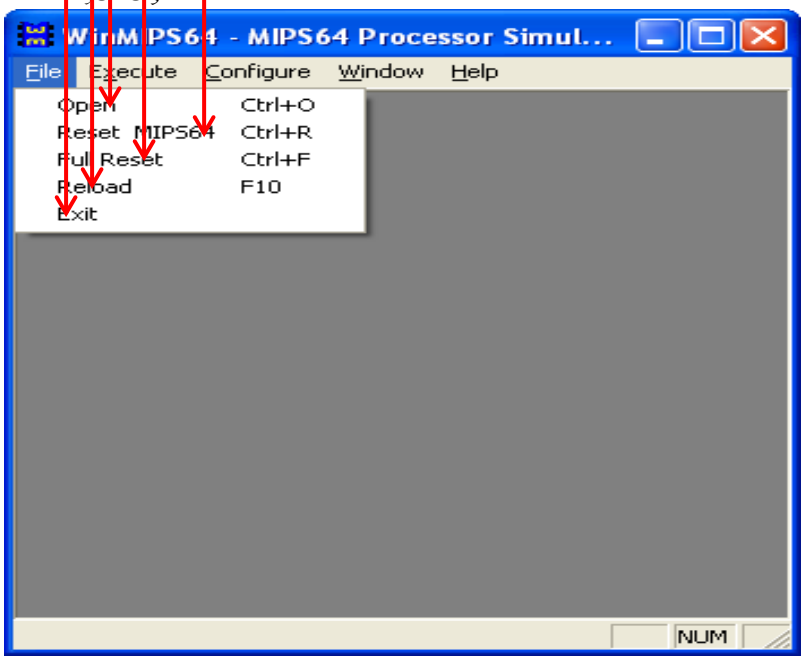
➢ Μιμείται μία απλή (dumb) τερματική συσκευή εισόδου/εξόδου με στοιχειώδεις δυνατότητες γραφικών

Status Line (Γραμμή κατάστασης):

➢ Η γραμμή κατάστασης βρίσκεται στο κάτω μέρος και συνήθως δείχνει "Ready", όμως κατά την διάρκεια προσομοίωσης ενός προγράμματος παρέχει χρήσιμες πληροφορίες σχετικά με την τρέχουσα κατάσταση της προσομοίωσης

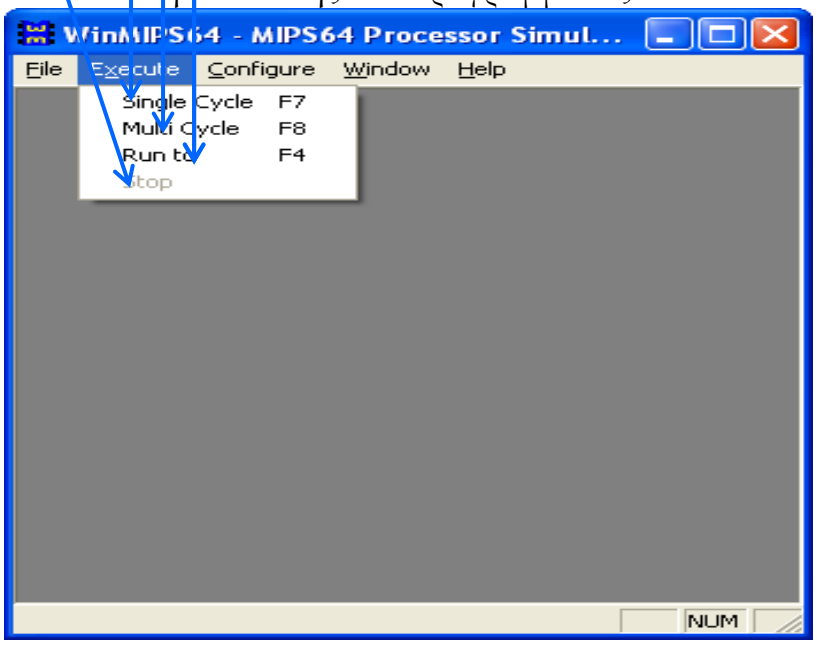
Το μενού: File:

- Φόρτωση αρχείου (επιλέγουμε το αρχείο με επέκταση.s)
- Αρχικοποίηση καταχωρητών, διοχέτευσης, στατιστικών και του περιεχομένου του PC στην πρώτη εντολή του προγράμματος
- Αρχικοποίηση όλων των ανωτέρω και της μνήμης δεδομένων
- Επαναφόρτωση αρχείου
- Έξοδος



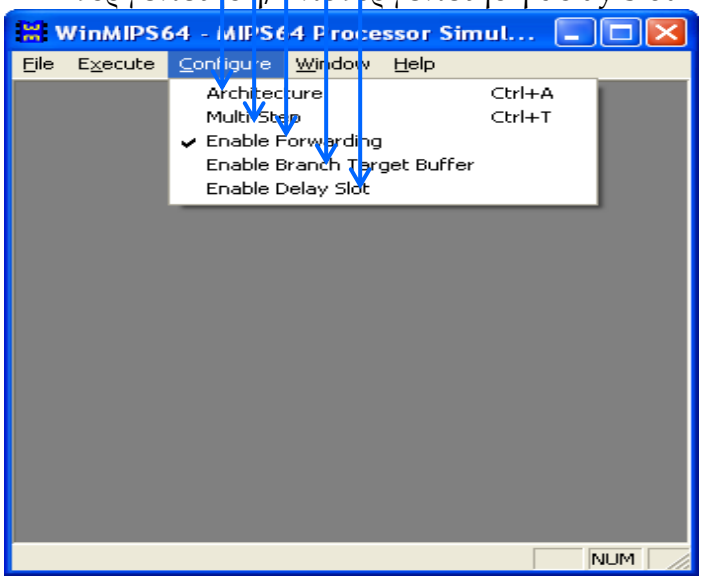
Το μενού: Execute:

- Εκτέλεση του προγράμματος κατά ένα κύκλο ρολογιού
- Εκτέλεση του προγράμματος κατά συγκεκριμένο αριθμό κύκλων ρολογιού (όσοι έχουν οριστεί στο multy-step-δείτε επόμενη διαφάνεια)
- Εκτέλεση όλου του προγράμματος
- Διακοπή εκτέλεσης του προγράμματος



Το μενού: Configure:

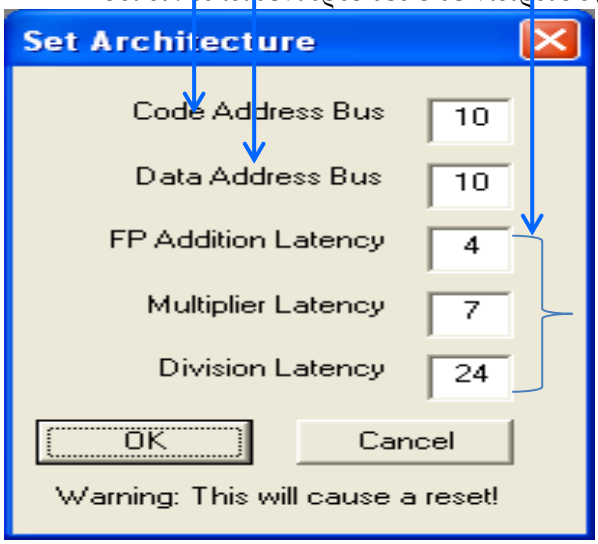
- Υπομενού επιλογών αρχιτεκτονικής (δείτε το μεθεπόμενο σχήμα)
- Υπομενού καθορισμού αριθμού πολλαπλών βημάτων εκτέλεσης (δείτε την μεθεπόμενη διαφάνεια)
- Ενεργοποίηση/Απενεργοποίηση προώθησης (forwarding)
- Ενεργοποίηση/Απενεργοποίηση πρόβλεψη διακλάδωσης
- Ενεργοποίηση/Απενεργοποίηση delay slot



Το Υπομενού: Architecture:

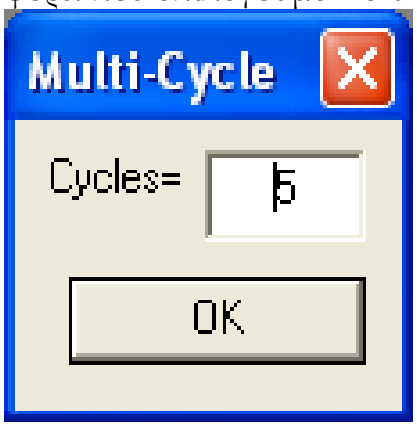
✓ Μπορείτε να αλλάξετε τη δομή και τους χρονικούς περιορισμούς της διοχέτευσης κινητής υποδιαστολής και των μεγεθών της μνήμης δεδομένων και της μνήμης εντολών:

- Αφορά το μέγεθος της μνήμης εντολών. Το 10 αντιστοιχεί σε 210=1024 byte μνήμης εντολών
- Αντίστοιχα το "data address bus"αφορά το μέγεθος της μνήμης δεδομένων
- Οποιαδήποτε αλλαγή στις καθυστερήσεις κινητής υποδιαστολής θα αντικατοπτριστεί στο παράθυρο της διοχέτευσης (Pipeline)



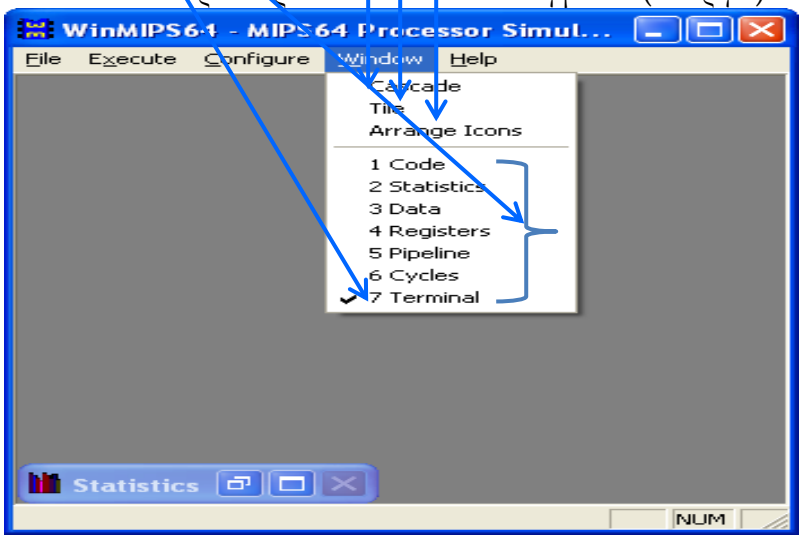
Το Υπομενού: Multi-Cycle:

Καθορίζουμε τον αριθμό των κύκλων του ρολογιού που εκτελούνται κάθε φορά που επιλέγουμε multi-cycle από το μενού execute.



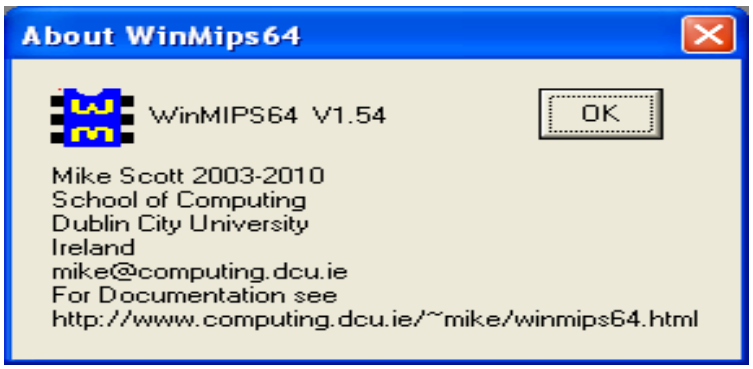
Το μενού: Window:

- Εμφανίζει τα 7 υποπαράθυρα το ένα πίσω από το άλλο
- Εμφανίζει τα 7 υποπαράθυρα ώστε να καλύπτουν πλήρως το κεντρικό παράθυρο
- Ταξινομεί τα υποπαράθυρα όταν είναι ελαχιστοποιημένα στο κάτω μέρος του κεντρικού παραθύρου
- Τα 7 υποπαράθυρα
- Το υποπαράθυρο που είναι επιλεγμένο (ενεργό)



Το μενού: Help:

Εμφανίζεται το διπλανό παράθυρο που περιέχει πληροφορίες σχετικές με τον WinMIPS64



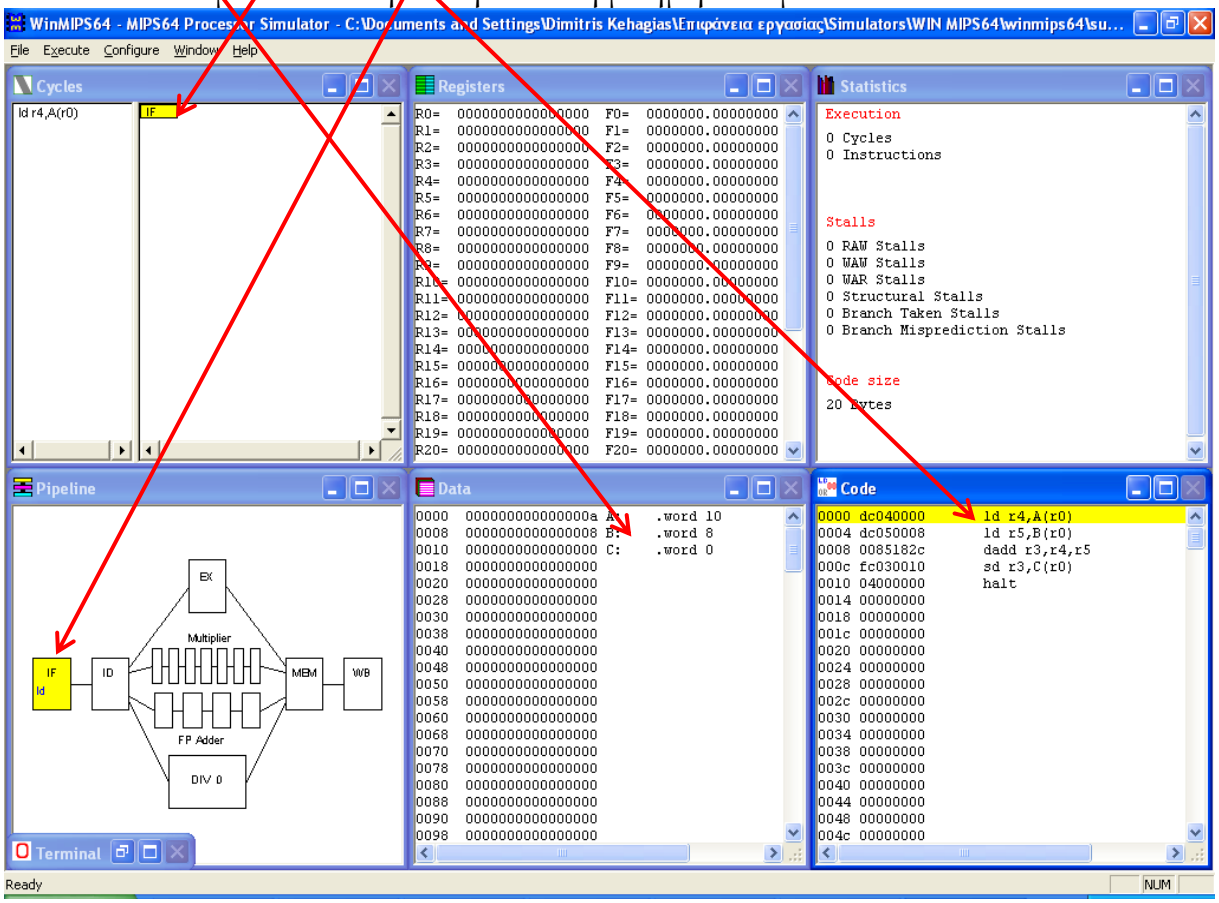
Παράδειγμα: Πρόγραμμα sum.s:

Χρησιμοποιείστε έναν text editor (π.χ. Notepad) και δημιουργείστε το αρχείο sum.s με το εξής περιεχόμενο:

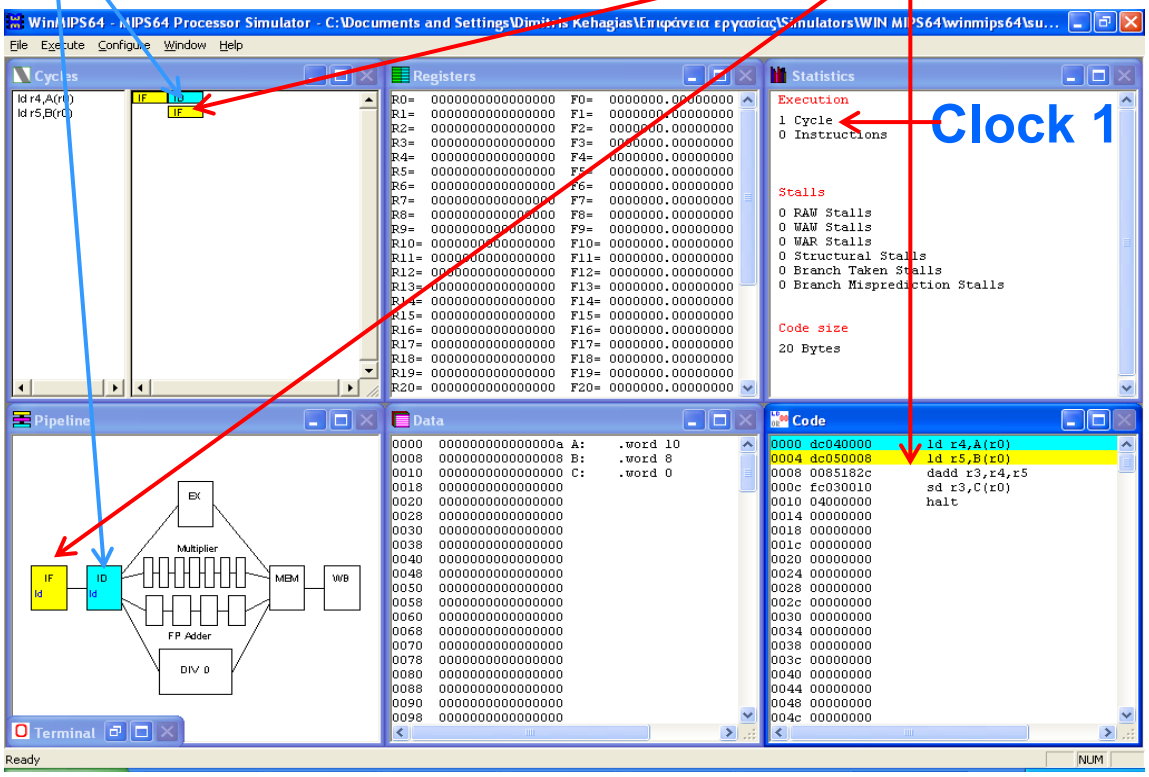
```
        .data
A:   .word 10
B:   .word 8
C:   .word 0
     .text
main:
     ld r4,A(r0)
     ld r5,B(r0)
     dadd r3,r4,r5
     sd r3,C(r0)
     halt
```

- Το πρόγραμμα αυτό υπολογίζει το άθροισμα δύο ακεραίων A και B από τη μνήμη και αποθηκεύει το αποτέλεσμα στη θέση μνήμης C
- Φορτώστε το πρόγραμμα sum.s στον προσομοιωτή WinMips64 και τρέξτε το με την επιλογή "Single Cycle"

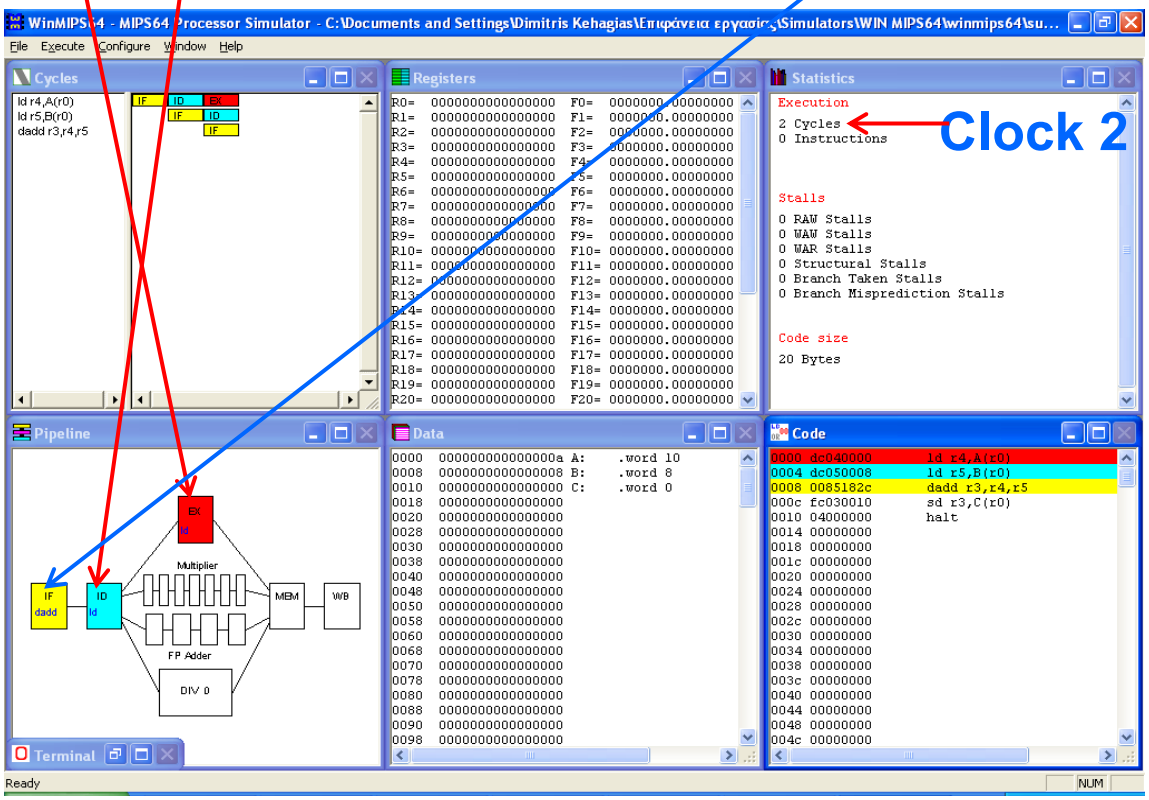
  - Η πρώτη εντολή, ld r4,A(r0), είναι έτοιμη να εκτελεστεί και βρίσκεται στο στάδιο IF
  - Τα δεδομένα αποθηκεύτηκαν στη μνήμη δεδομένων



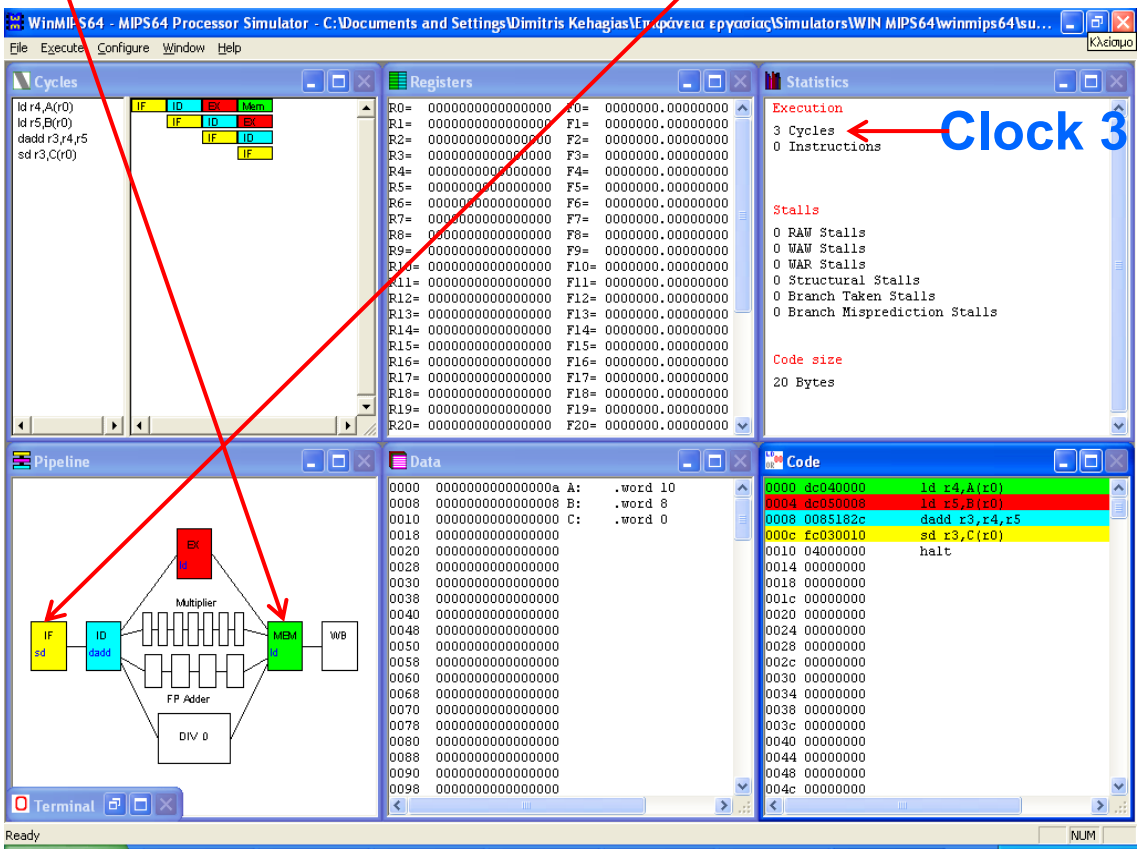
Η 1η εντολή είναι έτοιμη να εκτελεστεί στο στάδιο ID & η 2η εντολή είναι έτοιμη να εκτελεστεί στο στάδιο IF



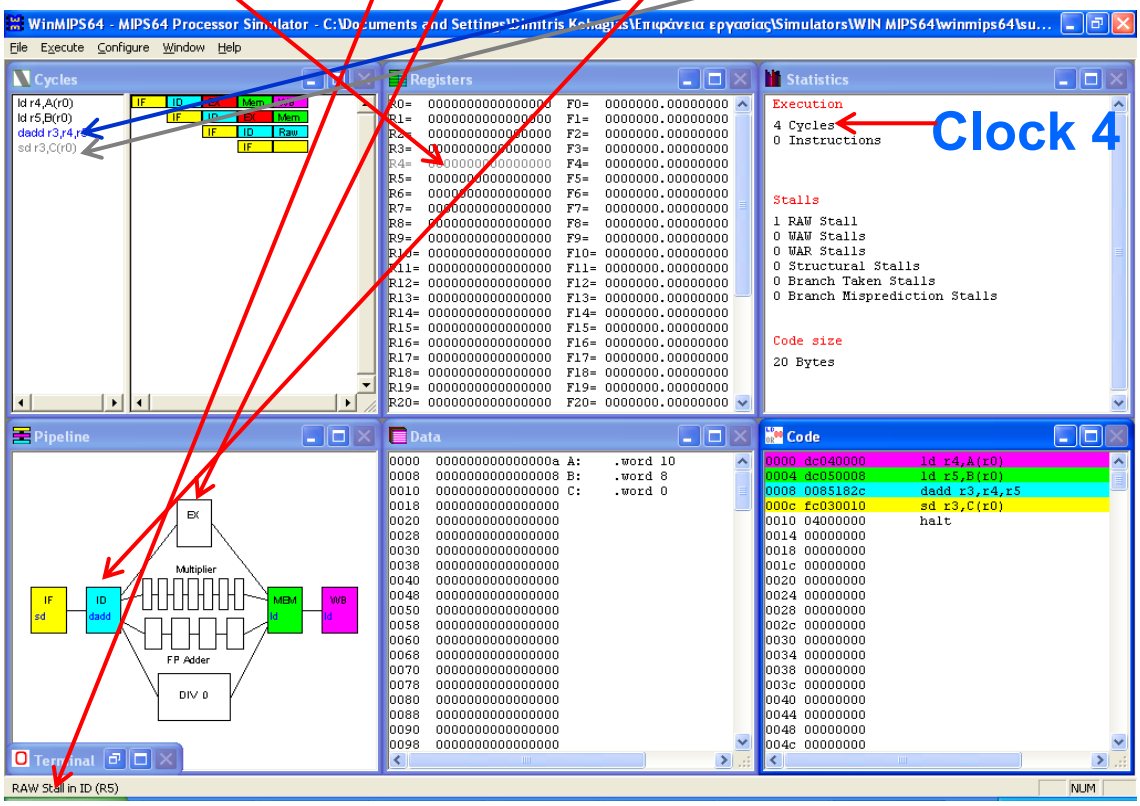
Η 1η & η 2η εντολή προχώρησαν κατά ένα στάδιο & η 3η εντολή είναι έτοιμη να εκτελεστεί



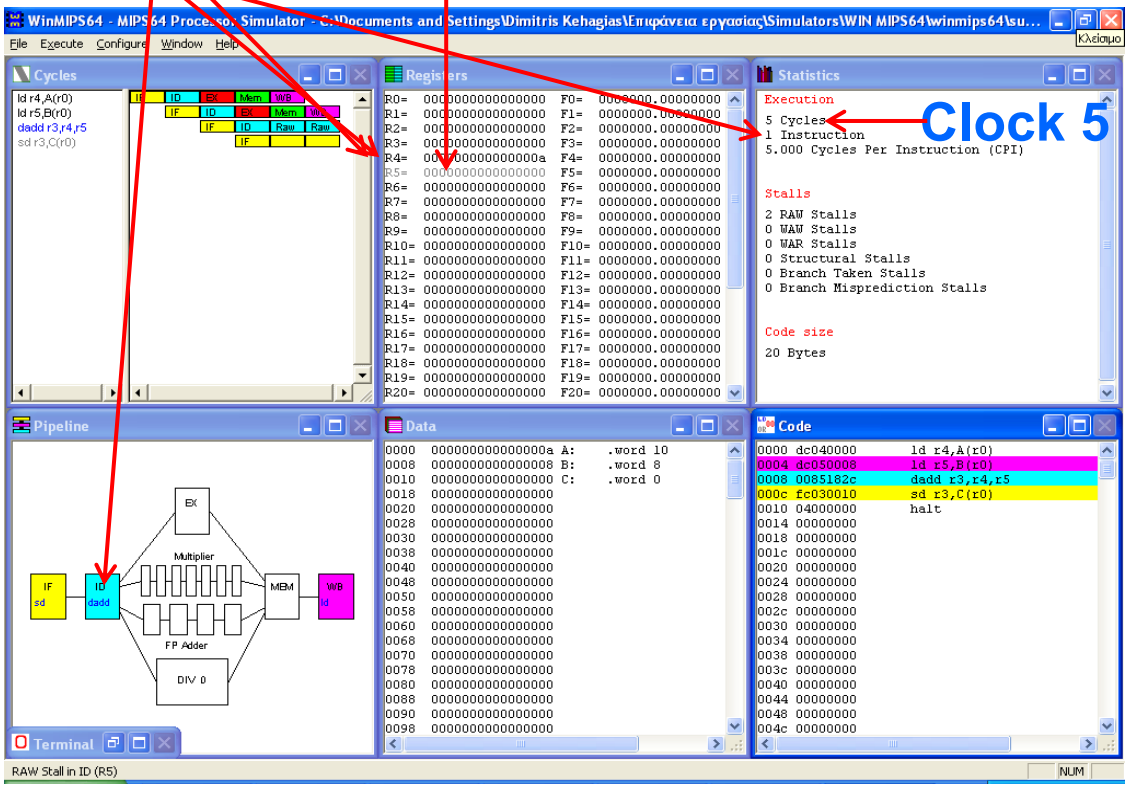
Η 1η εντολή βρίσκεται στο στάδιο MEM & η 4η εντολή είναι έτοιμη να εκτελεστεί.



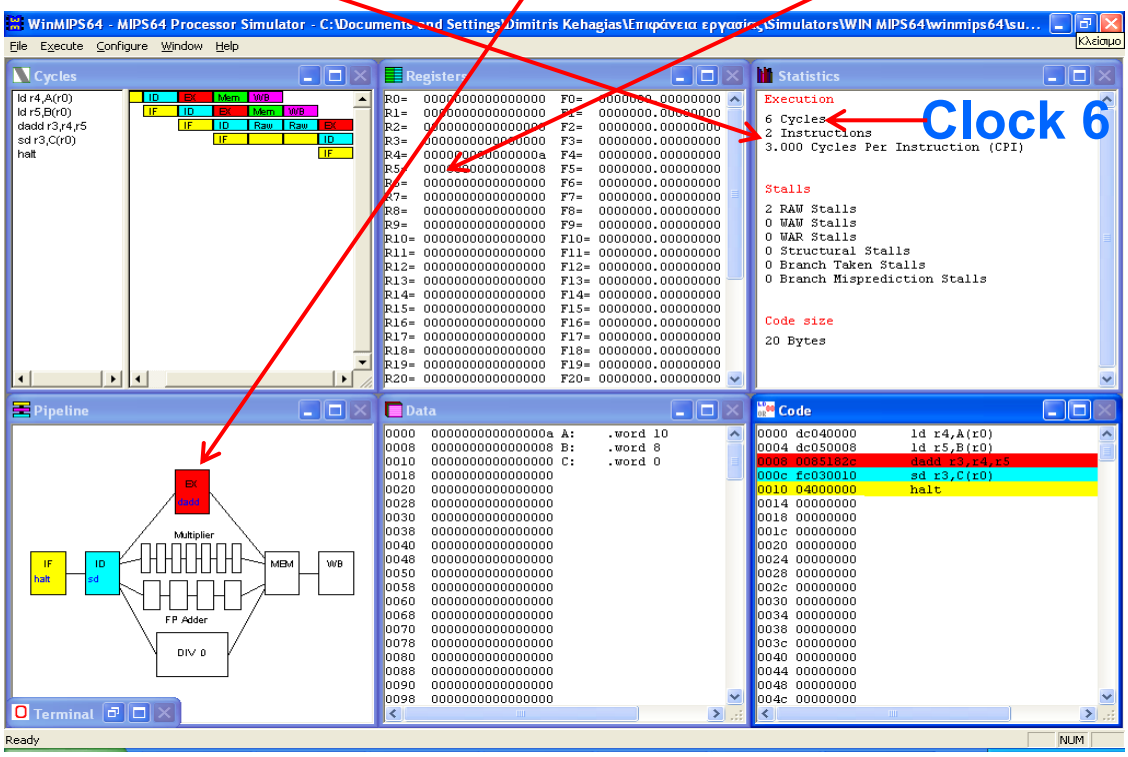
Η 3η εντολή προκαλεί RAW Stall στο στάδιο ID. Άλλο χρώμα.
Έχει άλλο χρώμα γιατί αναμένει κάποια τιμή.



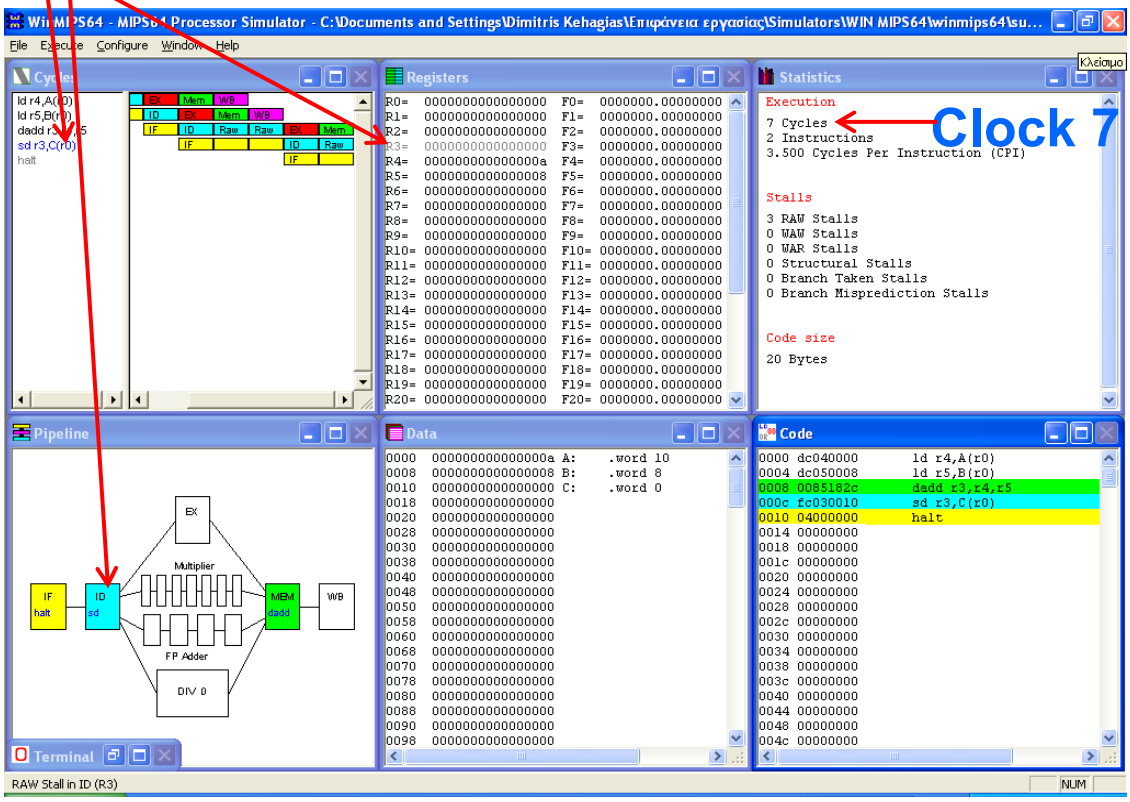
Δεύτερο stall της 3ης εντολής για τον R5.
Ο R4 ενημερώθηκε, ενώ ο R5 έχει άλλο χρώμα γιατί αναμένει κάποια τιμή.
Ολοκληρώθηκε η 1η εντολή.



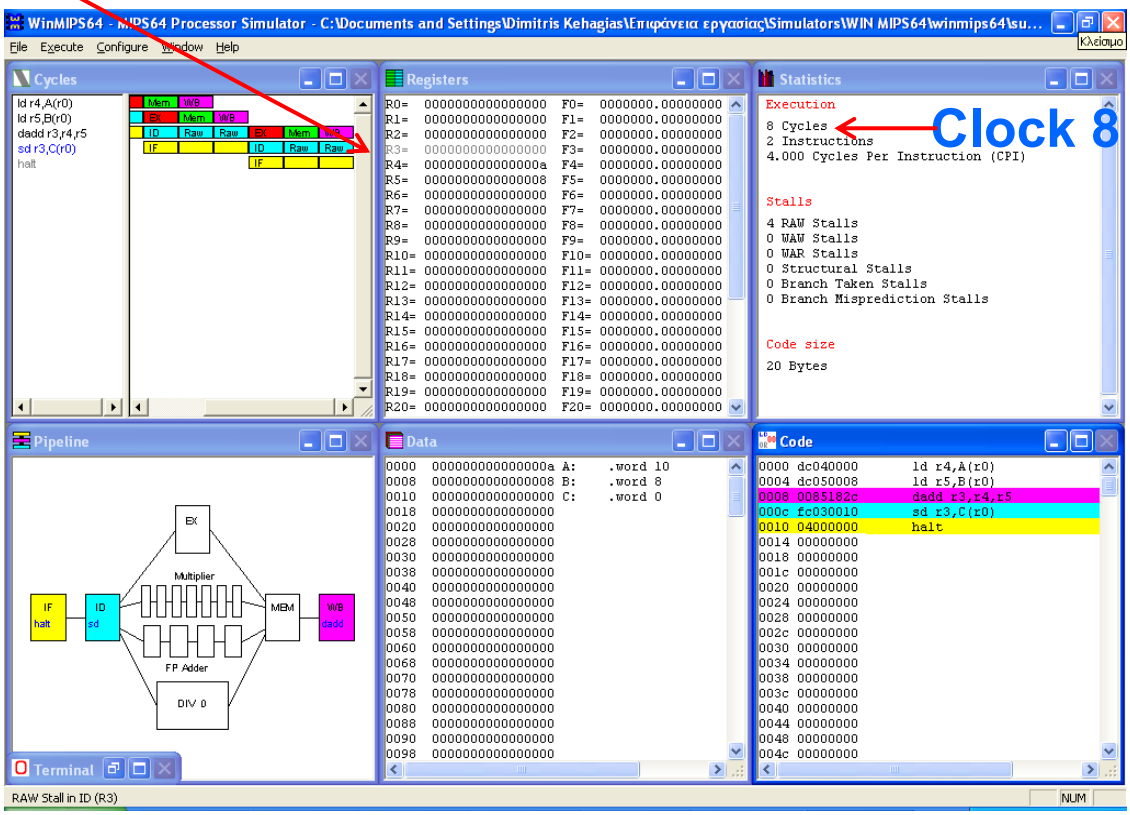
Η 3η εντολή προχώρησε στο στάδιο EX. Ενημερώθηκε και ο R5.
Ολοκληρώθηκαν οι 2 πρώτες εντολές.



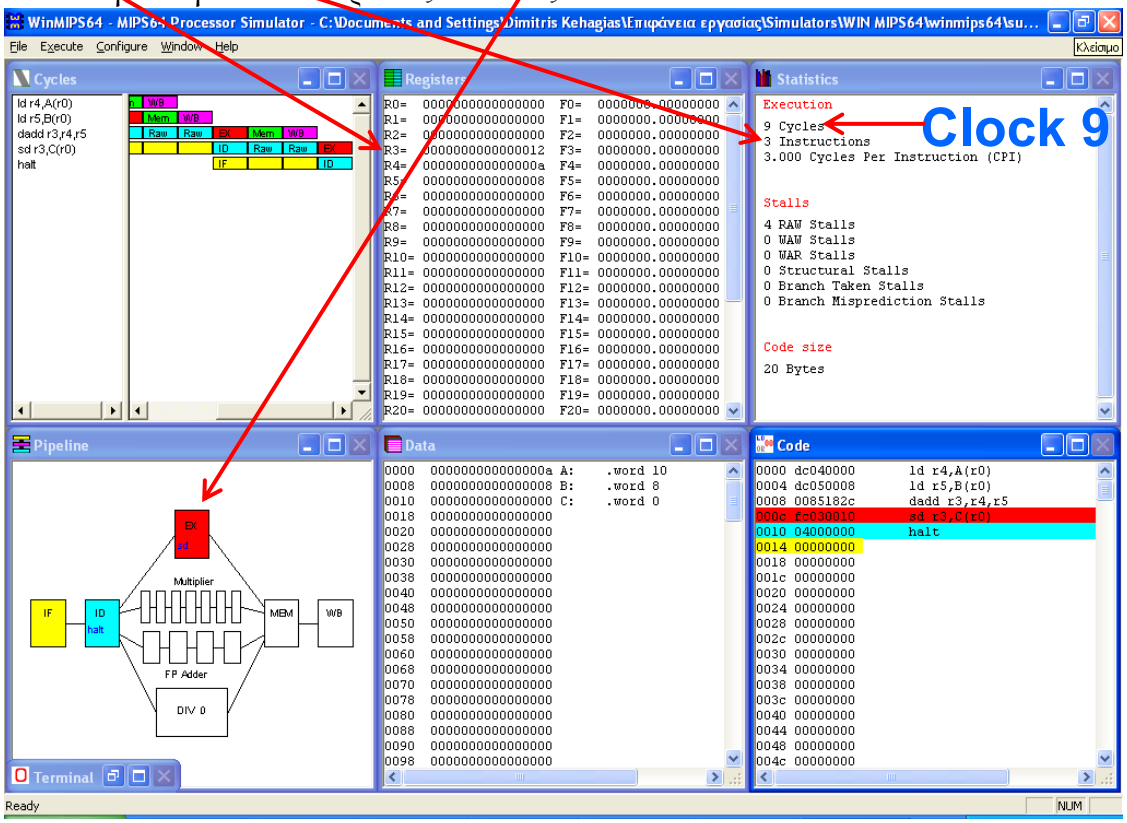
Η 4η εντολή προκαλεί stall εξαιτίας του R3.
Ο R3 αναμένει κάποια τιμή.



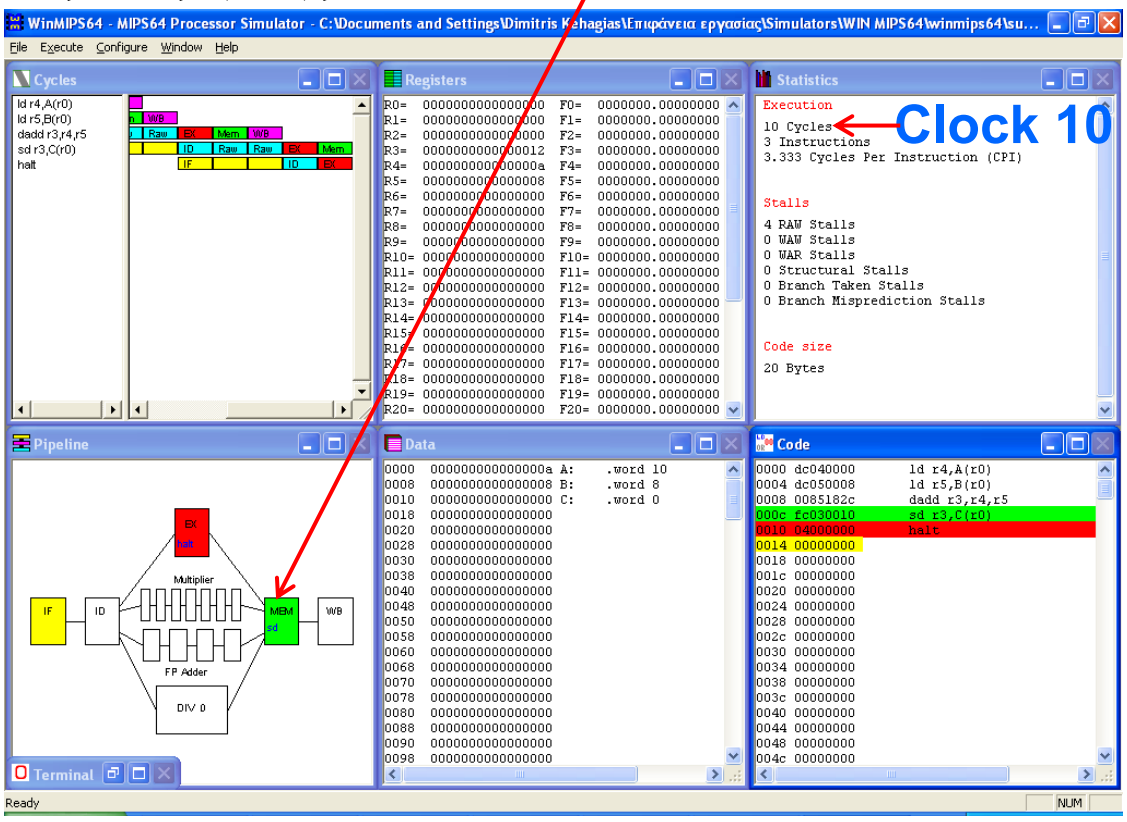
Η 4η εντολή προκαλεί και άλλο stall εξαιτίας του R3.
Ο R3 συνεχίζει να αναμένει κάποια τιμή εξαιτίας του νέου stall.



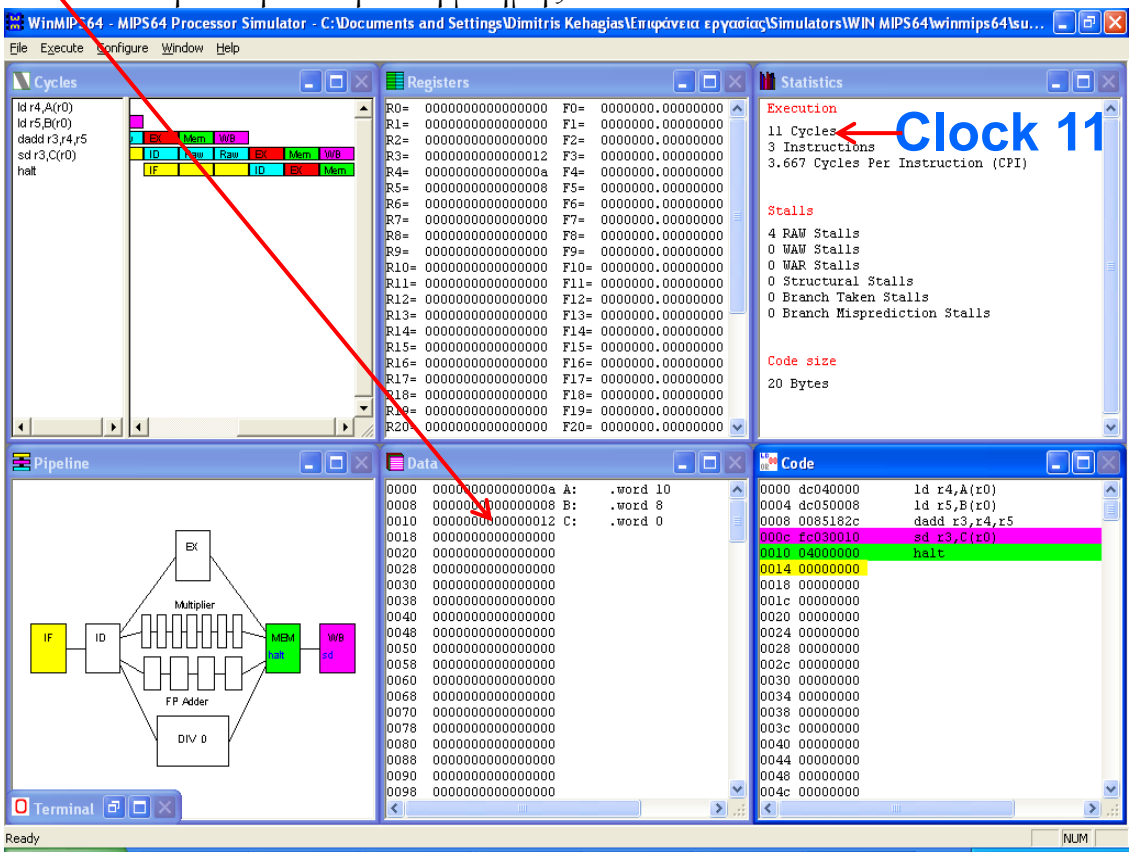
Η 4η εντολή προχώρησε στο στάδιο EX.
Ο R3 ενημερώθηκε με το αποτέλεσμα της πρόσθεσης.
Ολοκληρώθηκαν οι 3 πρώτες εντολές.



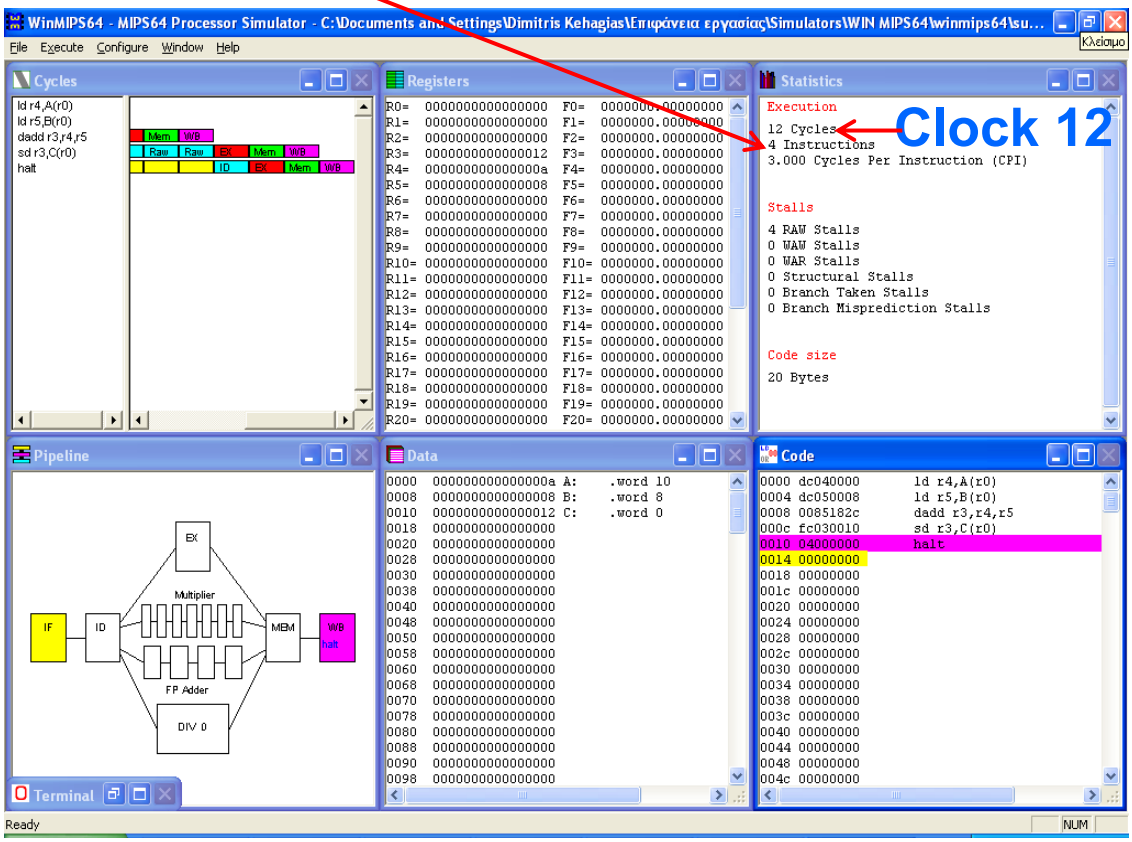
Η 4η εντολή προχώρησε στο στάδιο MEM.



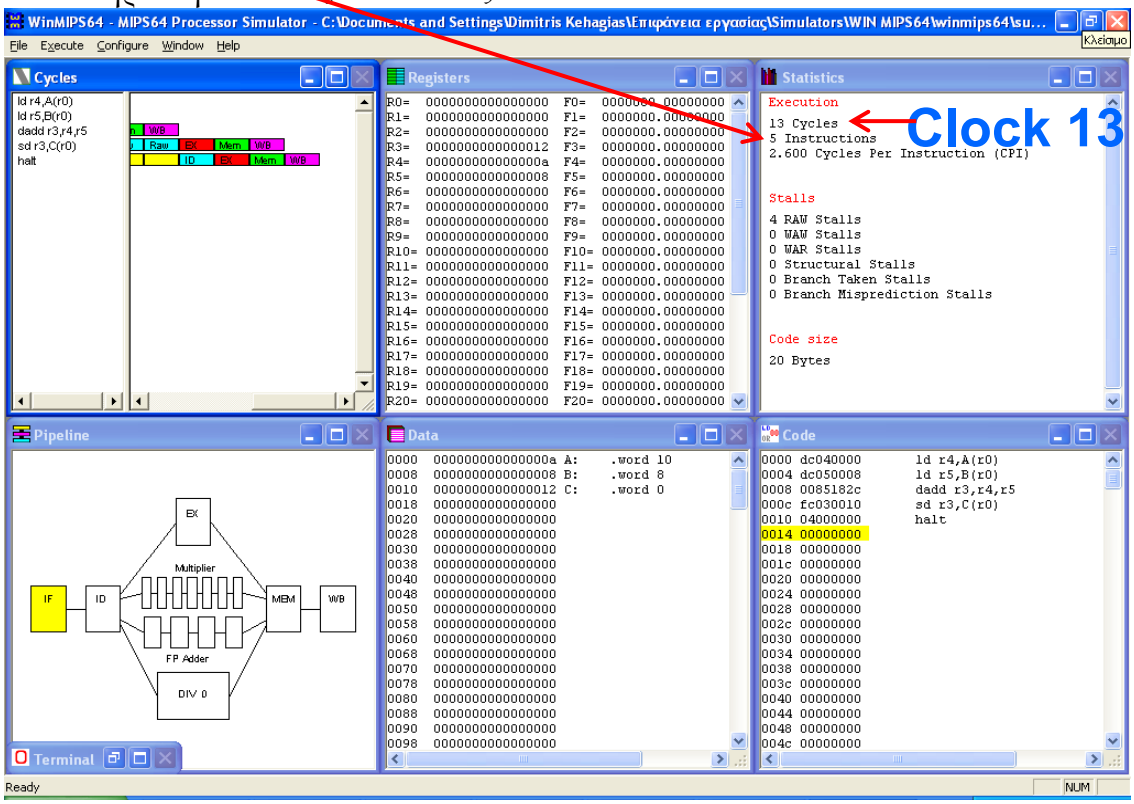
Ο R3 αποθηκεύτηκε στη θέση μνήμης C.



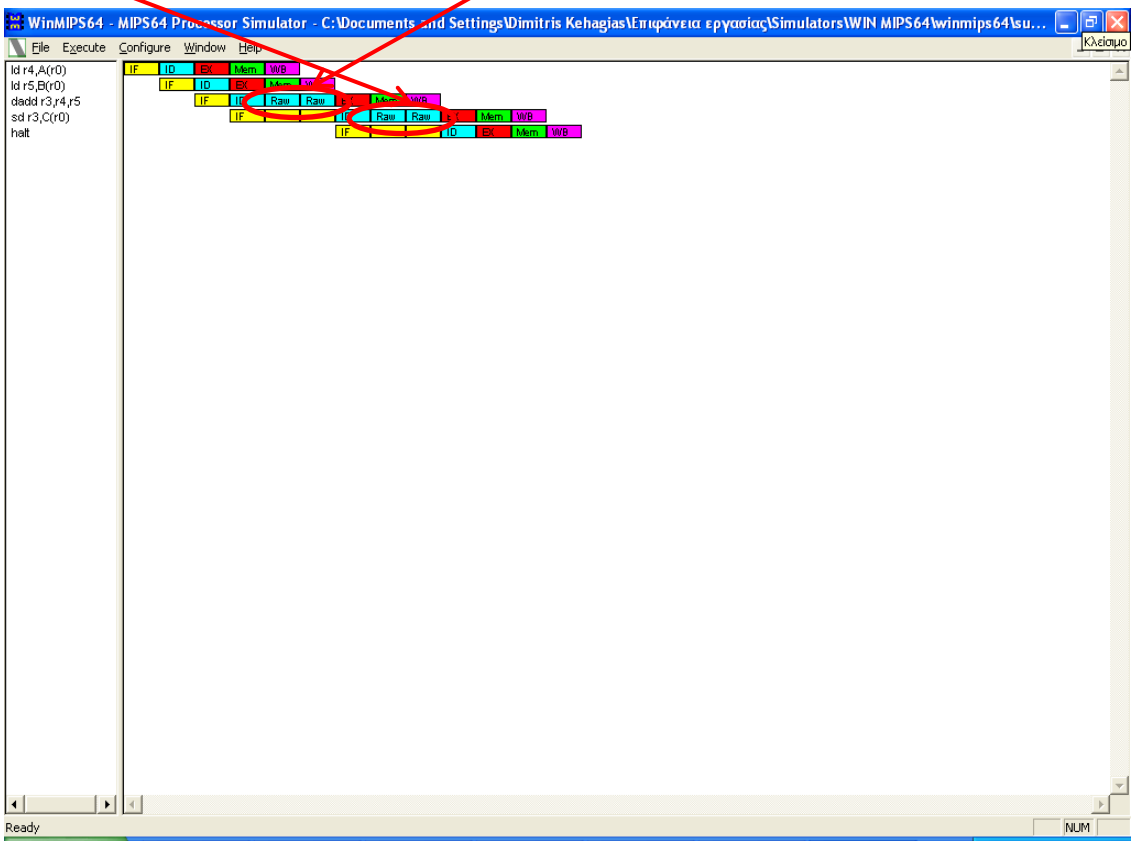
Ολοκληρώθηκαν οι 4 πρώτες εντολές.



Ολοκληρώθηκαν όλες οι εντολές.



Τα 4 stalls: 2 εξαιτίας της εντολής dadd r3,r4,r5 και 2 εξαιτίας της εντολής sd r3,C(r0).



Οδηγίες (Directives)

Οι οδηγίες προς το συμβολομεταφραστή έχουν ως στόχο να τον πληροφορήσουν για τον τρόπο με τον οποίο θα μεταφράσει ένα πρόγραμμα. Οι οδηγίες δεν είναι εντολές. Κυριότερες οδηγίες:

- **.text** ή **.code** - Η αρχή του τμήματος του κώδικα προσδιορίζεται από μία εκ των οδηγιών αυτών
- **.data** - προσδιορίζει την αρχή του τμήματος των δεδομένων
- **.org n** - ορίζει τη διεύθυνση **n** της μνήμης δεδομένων από την οποία μπορούν να αποθηκεύονται στη συνέχεια τα δεδομένα
- **.space n** – Δεσμεύει **n** διαδοχικά byte στη μνήμη δεδομένων
- **.asciiz str** - Αποθηκεύει τη συμβολοσειρά **str** στη μνήμη και την τερματίζει με τον κενό χαρακτήρα (μηδέν)
- **.ascii str** - Αποθηκεύει τη συμβολοσειρά **str** στη μνήμη
- **.align n** – Ευθυγραμμίζει στο n-byte όριο
- **.byte b1,...,bn** - Αποθηκεύει **n** τιμές σε διαδοχικά byte στη μνήμη
- **.word w1,...,wn** - Αποθηκεύει **n** τιμές των 64 bit σε διαδοχικές λέξεις μνήμης
- **.word16 w1,...,wn** - Αποθηκεύει **n** τιμές των 16 bit σε διαδοχικές θέσεις μνήμης
- **.word32 w1,...,wn** - Αποθηκεύει **n** τιμές των 32 bit σε διαδοχικές θέσεις μνήμης
- **.double w1,...,wn** - Αποθηκεύει **n** αριθμούς κινητής υποδιαστολής διπλής ακρίβειας σε διαδοχικές θέσεις μνήμης (8 byte για κάθε αριθμό).

Ρεπερτόριο Εντολών

Ακολουθούν οι εντολές που υποστηρίζει ο Winmips64:
Όπου:

- **reg** είναι ένας ακέραιος καταχωρητής
  - μπορούν να αναφέρονται ως r0-r31 ή R0-R31 ή $0-$31 ή με τα ψευδοονόματα του mips: $zero για τον r0, $t0…$t7 για τους r8….r15, $s0….$s7 για τους r16….r23
  - στο υποπαράθυρο Registers θα εμφανίζονται ως R0-R31
- **freg** είναι ένας καταχωρητής κινητής υποδιαστολής
  - στο υποπαράθυρο Registers θα εμφανίζονται ως F0-F31
- **imm** είναι μία σταθερή τιμή (μέχρι 16 bit)

lb reg,imm(reg) - load byte
lbu reg,imm(reg) - load byte unsigned
sb reg,imm(reg) - store byte
lh reg,imm(reg) - load 16-bit half-word
lhu reg,imm(reg) - load 16-bit half word unsigned
sh reg,imm(reg) - store 16-bit half-word

lw reg,imm(reg) - load 32-bit word
lwu reg,imm(reg) - load 32-bit word unsigned
sw reg,imm(reg) - store 32-bit word
ld reg,imm(reg) - load 64-bit double-word
sd reg,imm(reg) - store 64-bit double-word
l.d freg,imm(reg) - load 64-bit floating-point
s.d freg,imm(reg) - store 64-bit floating-point
halt - stops the program
daddi reg,reg,imm - add immediate
daddui reg,reg,imm - add immediate unsigned
andi reg,reg,imm - logical and immediate
ori reg,reg,imm - logical or immediate
xori reg,reg,imm - exclusive or immediate
lui reg,imm - load upper half of register immediate
slti reg,reg,imm - set if less than immediate
sltiu reg,reg,imm - set if less than immediate unsigned
beq reg,reg,imm - branch if pair of registers are equal
bne reg,reg,imm - branch if pair of registers are not equal
beqz reg,imm - branch if register is equal to zero
bnez reg,imm - branch if register is not equal to zero
j imm - jump to address
jr reg - jump to address in register
jal imm - jump and link to address (call subroutine)
jalr reg - jump and link to address in register
dsll reg,reg,imm - shift left logical
dsrl reg,reg,imm - shift right logical
dsra reg,reg,imm - shift right arithmetic
dsllv reg,reg,reg - shift left logical by variable amount
dsrlv reg,reg,reg - shift right logical by variable amount
dsrav reg,reg,reg - shift right arithmetic by variable amount
movz reg,reg,reg - move if register equals zero
movn reg,reg,reg - move if register not equal to zero
nop - no operation
and reg,reg,reg - logical and
or reg,reg,reg - logical or
xor reg,reg,reg - logical xor
slt reg,reg,reg - set if less than
sltu reg,reg,reg - set if less than unsigned
dadd reg,reg,reg - add integers
daddu reg,reg,reg - add integers unsigned
dsub reg,reg,reg - subtract integers
dsubu reg,reg,reg - subtract integers unsigned
dmul reg,reg,reg - signed integer multiplication

dmulu reg,reg,reg - unsigned integer multiplication
ddiv reg,reg,reg - signed integer division
ddivu reg,reg,reg - unsigned integer division
add.d freg,freg,freg - add floating-point
sub.d freg,freg,freg - subtract floating-point
mul.d freg,freg,freg - multiply floating-point
div.d freg,freg,freg - divide floating-point
mov.d freg,freg - move floating-point
cvt.d.l freg,freg - convert 64-bit integer to a double FP format
cvt.l.d freg,freg - convert double FP to a 64-bit integer format
c.lt.d freg,freg - set FP flag if less than
c.le.d freg,freg - set FP flag if less than or equal to
c.eq.d freg,freg - set FP flag if equal to
bc1f imm - branch to address if FP flag is FALSE
bc1t imm - branch to address if FP flag is TRUE
mtc1 reg,freg - move data from integer register to FP regist.
mfc1 reg,freg - move data from FP register to integer regist.

ΜΕΡΟΣ Β: ΑΣΚΗΣΗ

Δίνεται ο παρακάτω κώδικας. Απαντήστε στις ερωτήσεις που ακολουθούν χρησιμοποιώντας τον προσομοιωτή WINMIPS64, με: Multiplier Latency=4, FP Addition Latency=2, Division Latency=10.

```
.data
X: .double 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48
Z: .double 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
Y: .double 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
.text
MAIN:   daddi R1,R0,24
        daddi R2,R0,X
        daddi R3,R0,Z
        daddi R4,R0,Y
loop:   l.d F1,0(R2)
        l.d F2,0(R3)
        mul.d F3,F1,F1
        div.d F4,F1,F2
        add.d F5,F3,F4
        s.d F5,0(R4)
        daddi R2,R2,8
        daddi R3,R3,8
        daddi R4,R4,8
        daddi R1,R1,-1
        bnez R1,loop
        halt
```

Ερωτήσεις:

1. Ενεργοποιείστε την προώθηση (Forwarding enabled), απενεργοποιείστε Delay Slot, Branch Target Buffer και εκτελέστε τον ανωτέρω κώδικα.
   (α) Υπολογίστε το συνολικό αριθμό κύκλων εκτέλεσης του προγράμματος και το CPI.
   (β) Σχολιάστε τις καθυστερήσεις (RAW stalls, structural stalls, Branch stalls) στην εκτέλεση του προγράμματος (κυρίως τις αιτίες που τις προκαλούν).

2. Με ενεργοποιημένα μόνο την προώθηση (Forwarding enabled) και το Branch Target Buffer εκτελέστε τον κώδικα.

(α) Υπολογίστε το συνολικό αριθμό κύκλων εκτέλεσης, το CPI και την αύξηση της απόδοσης που επιτύχατε (έναντι του ερωτήματος 1(α)).
(β) Εξηγήστε σε ποιες περιπτώσεις συμβαίνουν καθυστερήσεις λόγω misspredict̲ion.

3. Με ενεργοποιημένα μόνο την προώθηση (Forwarding enabled) και το Enable Delay Slot εκτελέστε τον κώδικα. Πρέπει να τροποποιήσετε τον κώδικα ώστε να τοποθετήσετε μια εντολή στην υποδοχή «καθυστέρησης διακλάδωσης» (branch delay slot).
   (α) Υπολογίστε το συνολικό αριθμό κύκλων εκτέλεσης, το CPI και την αύξηση της απόδοσης που επιτύχατε (έναντι του ερωτήματος 1(α)).
   (β) Ποιος μηχανισμός είναι πιο αποδοτικός, Branch Target Buffer ή branch Delay Slot; Σχολιάστε την απάντησή σας.

4. Αναδιατάξτε τον κώδικα με σκοπό να αποφύγετε καθυστερήσεις (stalls). Με ενεργοποιημένη μόνο την προώθηση (Forwarding enabled) εκτελέστε τον κώδικα.
   (α) Υπολογίστε το συνολικό αριθμό κύκλων εκτέλεσης, το CPI και την αύξηση της απόδοσης που επιτύχατε (έναντι του ερωτήματος 1(α)).

5. Τροποποιήστε το πρόγραμμά σας με στόχο τον ελάχιστο χρόνο εκτέλεσης, χρησιμοποιώντας την τεχνική loop unrolling:
   Προσπαθήστε να αποφύγετε τους κύκλους καθυστέρησης λόγω εξάρτησης δεδομένων αναδιατάσσοντας τις εντολές (4 αναδιπλώσεις). Ενεργοποιήστε ένα από τα Delay Slot ή Branch Target Buffer (αυτό που σας δίνει την καλύτερη απόδοση) και την προώθηση (Forwarding enabled).
   Υπολογίστε το συνολικό αριθμό κύκλων εκτέλεσης, το CPI και την αύξηση της απόδοσης που επιτύχατε (έναντι του ερωτήματος 1(α)). Αναλύστε τους κύκλους καθυστέρησης που αποφύγατε με την τροποποίηση του προγράμματος.