Απλός επεξεργαστής (Επανάληψη)

Διάδρομος δεδομένων και μονάδα ελέγχου

4ο κεφάλαιο

Ο επεξεργαστής : Διάδρομος Δεδομένων και Έλεγχος

• Σε αυτό το κεφάλαιο θα μελετήσουμε την υλοποίηση του διαδρόμου δεδομένων και μονάδας ελέγχου για εκτέλεση διαφόρων τύπων πράξεων για τις βασικές εντολές της Αρχιτεκτονικής Συνόλου Εντολών MIPS

Πολλές πιθανές διαφορετικές υλοποιήσεις

ΑΣΕ (ISA)

- ΑΣΕ:
  - σύνολο εντολών
  - τελεστές (add,sub,shift, mul, branches, load, store..)
  - τελεσταίοι (καταχωρητές, μνήμη και άμεσες τιμές)
  - σύνταξη εντολών και κωδικοποίηση στο δυαδικό
  - σημασιολογία εντολών
  - Endianess
  - Αριθμός καταχωρητών, μέγεθος καταχωρητών
  - Μέγεθος μνήμης
  - Από ποια διεύθυνση ξεκινά η εκτέλεση η πρώτη εντολή
  - …
  - Ένα μεγάλο document

- Πχ ΑΣΕ: x86, ARM, Power, MIPS
  - Κάθε 1-2 χρόνια εξελίσσονται

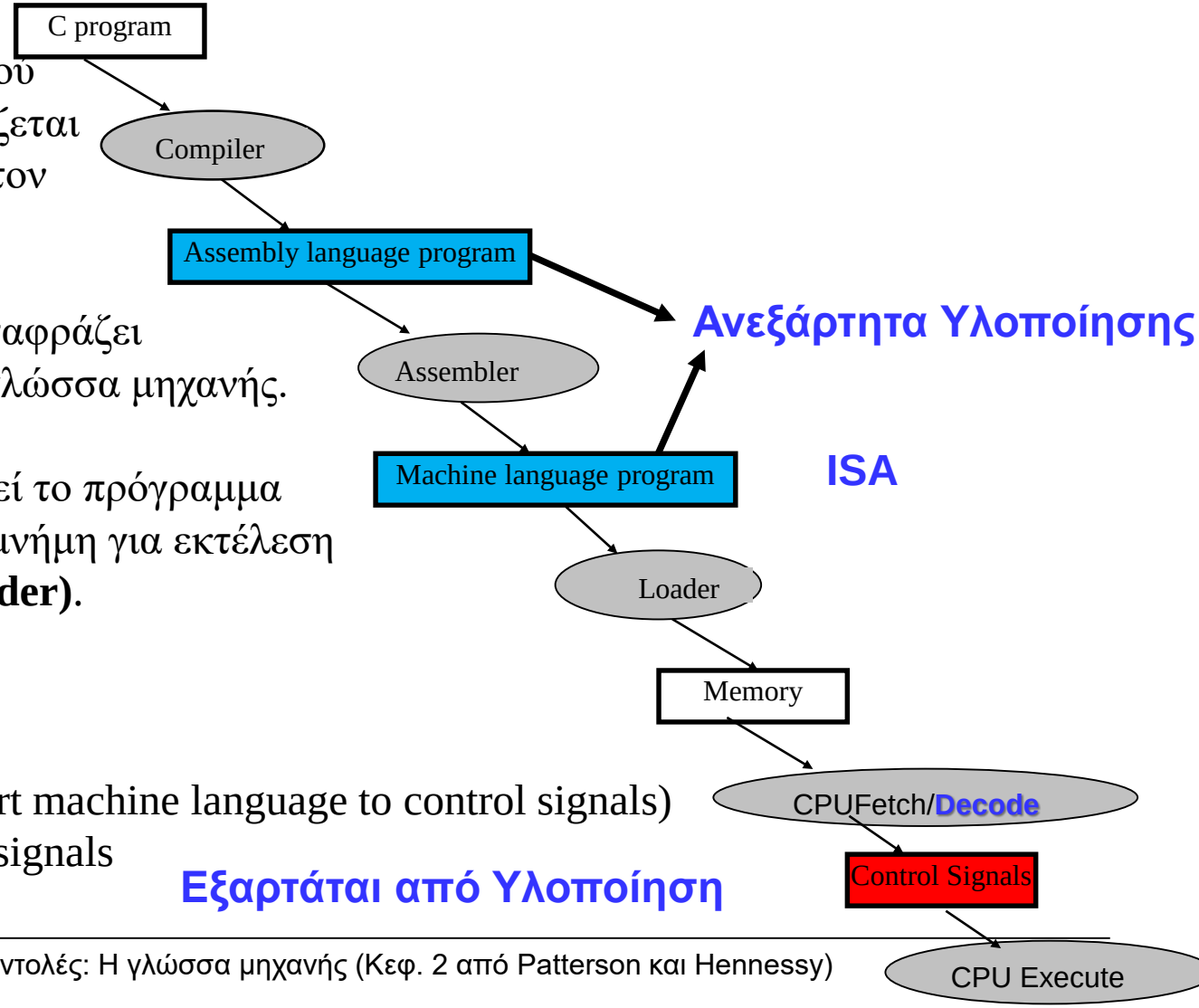
Μεγάλη Εικόνα: Διασύνδεση Λογισμικού/Υλικού

- Η ιεραρχία μετάφρασης για ένα πρόγραμμα από υψηλού επιπέδου γλώσσα μέχρι τη γλώσσα μηχανής.

- ένα πρόγραμμα σε C (υψηλού επιπέδου γλώσσα) μεταφράζεται σε συμβολική γλώσσα από τον μεταγλωττιστή

- ο συμβολομεταφραστής μεταφράζει την συμβολική γλώσσα σε γλώσσα μηχανής.

- Το πρόγραμμα που τοποθετεί το πρόγραμμα της γλώσσας μηχανής στη μνήμη για εκτέλεση ονομάζεται **φορτωτής (loader)**.



- Fetch and Decode (convert machine language to control signals)
- Execute decoded control signals

Γλώσσα μηχανής

Αρχιτεκτονική Συνόλου Εντολών (ΑΣΕ)

Instruction Set Architecture (ISA)

- **ανεξάρτητη της υλοποίησης του διαδρόμου δεδομένων**
  - **Ο διάδρομος δεδομένων περιλαμβάνει μονάδα ελέγχου (αποκωδικοποίηση γλώσσα μηχανής σε σήματα ελέγχου)**
  - **Αλλαγή στο υλικό δεν χρειάζεται να παραχθεί καινούργιο binary**
  - **Επέκταση της γλώσσας μηχανής (καινούργιες εντολές): προηγούμενα εκτελέσιμα τρέχουν σε καινούργιο υλικό που υλοποιεί νέες εντολές**

Μεγάλη Εικόνα

Table[2]=Table[0] + Table[1]

SW Program

```
lw      $1,0($29)
lw      $2,4($29)
add     $3,$1,$2
sw      $3,8($29)
```

SW Assembly

```
100011 11101 00001 0000 0000 0000 0000
100011 11101 00010 0000 0000 0000 0100
000000 00001 00010 00011  00000 100000
101011 11101 00011 0000 0000 0000 1000
```

SW Machine Language

```
11101 00000 00001 1 010 1 0000 0000 0000 0000 10 0
11101 00000 00010 1 010 1 0000 0000 0000 0100 10 0
00001 00010 00011 1 010 0 0000 0000 0000 0000 00 1
[11101  00011 00000 0 010 1 0000 0000 0000 1000 01 0
```

HW Decoder Control Signals for datapath

Ο επεξεργαστής : Διάδρομος Δεδομένων και Έλεγχος

• Σε αυτό το κεφάλαιο θα μελετήσουμε την υλοποίηση του διαδρόμου δεδομένων και της μονάδας ελέγχου για το σύνολο εντολών της μηχανής MIPS.

• Μία υλοποίηση για τις βασικές εντολές του MIPS:

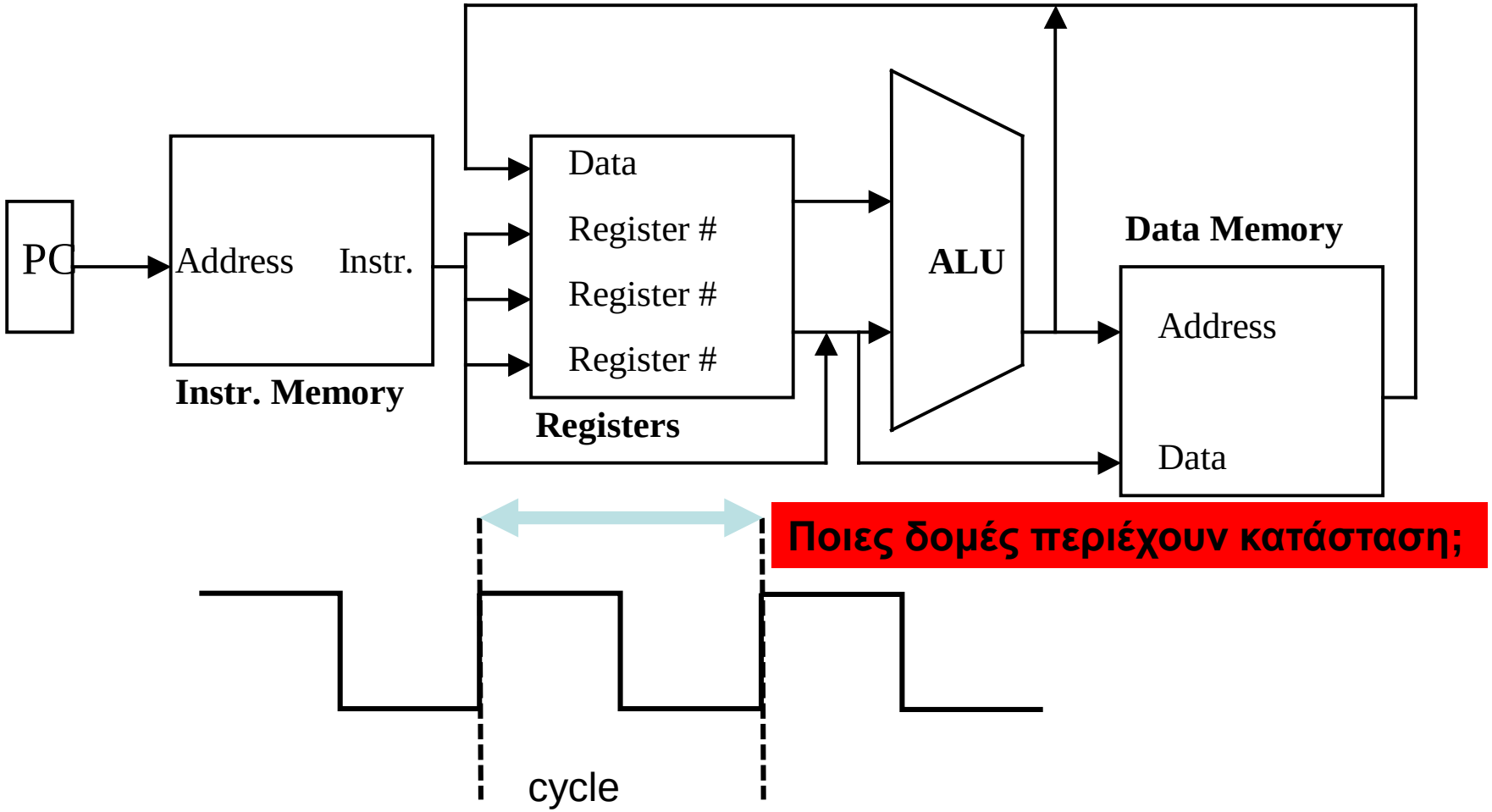
- Εντολές μνήμης (memory reference), δηλ. την load word(lw) και store word(sw).

- Αριθμητικές και λογικές εντολές, δηλ. add, sub, and, or και slt.

- Εντολή σύγκρισης branch equal(beq) και τέλος

- Εντολή jump(j).

- 32 integer registers
- Memory (instruction and data)



- Σε κάθε κύκλο ξεκινά και τελειώνει μια εντολή
- Κατάσταση αλλάζει στο τέλος του κύκλου και παραμένει σταθερή για ένα κύκλο

Πέντε Στάδια για την εκτέλεση μιας Εντολής (Fetch-Execute Cycle)

1. Προσκόμιση Εντολής (ifetch)

2. Αποκωδικοποίηση, Πρόσβαση στο Αρχείο Καταχωρητών (decode and register read)

3. Εκτέλεση (Execution)
   - alu, υπολογισμός διεύθυνσης, branch-completion, jump

4. Πρόσβαση στη μνήμη ή R-type πρόσβαση σε καταχωρητές (Memory Access and R-type completion)

5. Εντολές μνήμης-- αποθήκευση σε καταχωρητή (Memory read completion)

Μια περίληψη της υλοποίησης

• Ο μετρητής προγράμματος (PC-Program Counter) είναι ένας 32-bit καταχωρητής ο οποίος έχει αποθηκευμένη τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί.

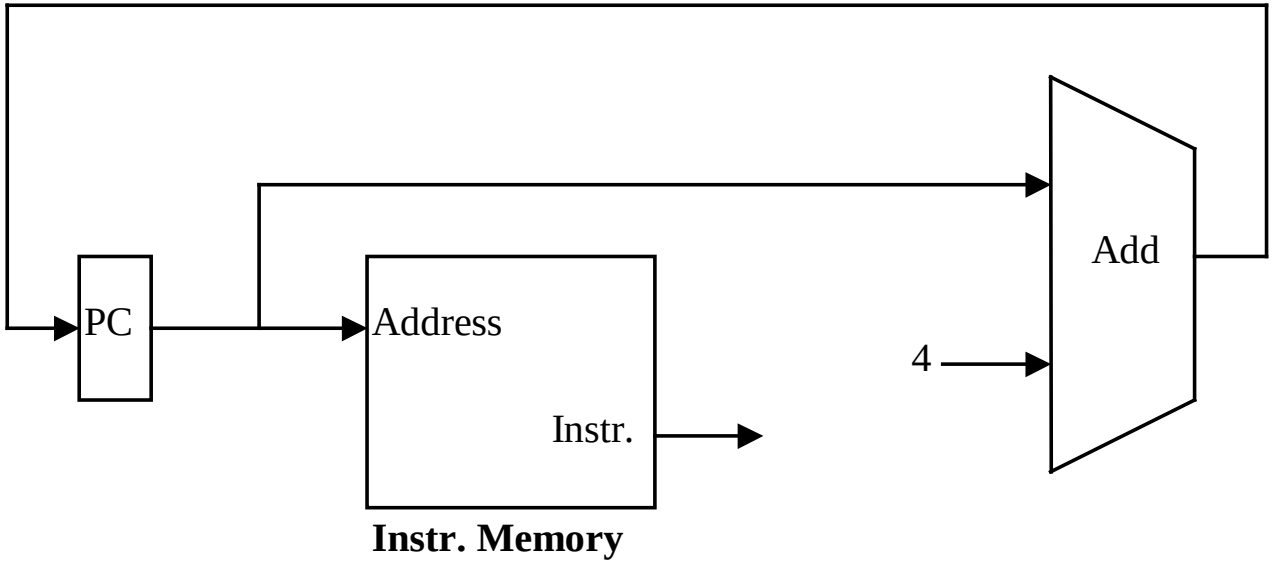
• Για κάθε εντολή τα πρώτα δύο βήματα είναι τα ίδια :

1. Στέλλεται το PC στη μνήμη που περιέχει τον κώδικα
   - προσκομίζεται (fetch) η εντολή από τη μνήμη.

2. Διαβάζεται ένας ή δύο καταχωρητές
   - χρησιμοποιώντας τα πεδία της εντολής, για να εντοπιστούν οι καταχωρητές που θα διαβαστούν τα δεδομένα τους,
   - π.χ. για την εντολή load χρειάζεται να διαβαστεί μόνο ένας καταχωρητής αλλά για άλλες εντολές πρέπει να διαβαστούν δύο καταχωρητές.

• Μετά από αυτά τα βήματα, οι ενέργειες που θα ακολουθήσουν για την ολοκλήρωση της εκτέλεσης της κάθε εντολής εξαρτώνται από τον τύπο της.

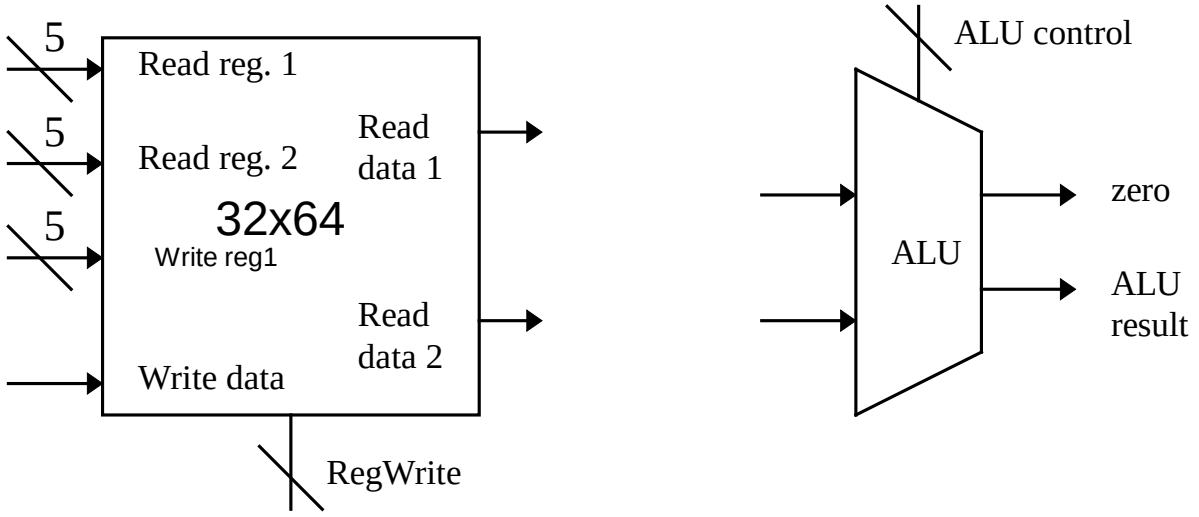
- Τμήμα του διαδρόμου δεδομένων που χρησιμοποιείται για:
  - προσκόμιση των εντολών από τη μνήμη και την
  - αύξηση του μετρητή προγράμματος(PC) κατά 4 bytes.



Αυτό γίνεται κάθε κύκλο (δεν υπάρχει σήμα ελέγχου)

Εντολές τύπου R - Αριθμητικές και Λογικές Εντολές

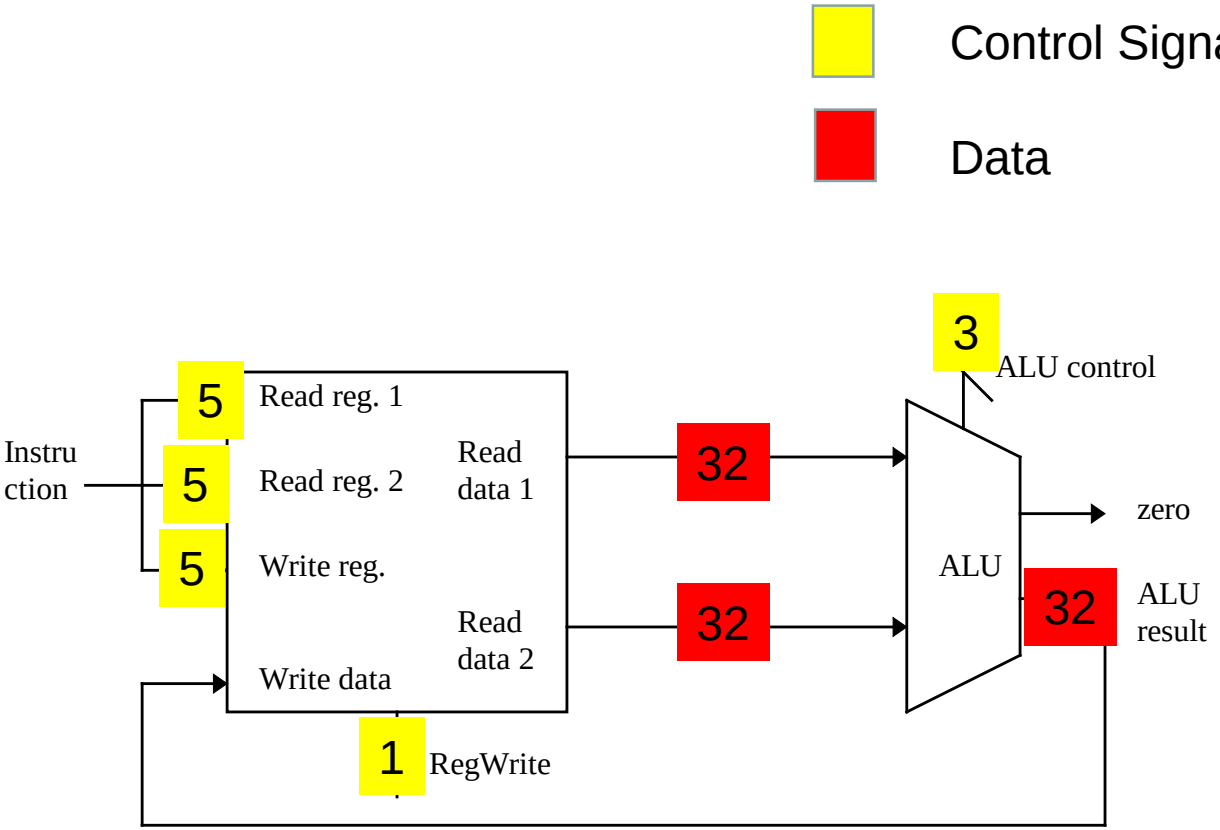
- Οι αριθμητικές και λογικές εντολές (πχ add, sub και slt)
  - διαβάζουν δύο καταχωρητές (2 πορτές για διάβασμα),
  - εκτελούν μια λειτουργία ALU και
  - γράφουν το αποτέλεσμα σε ένα καταχωρητή (1 πόρτα για γράψιμο.



a. Registers

b. ALU

•

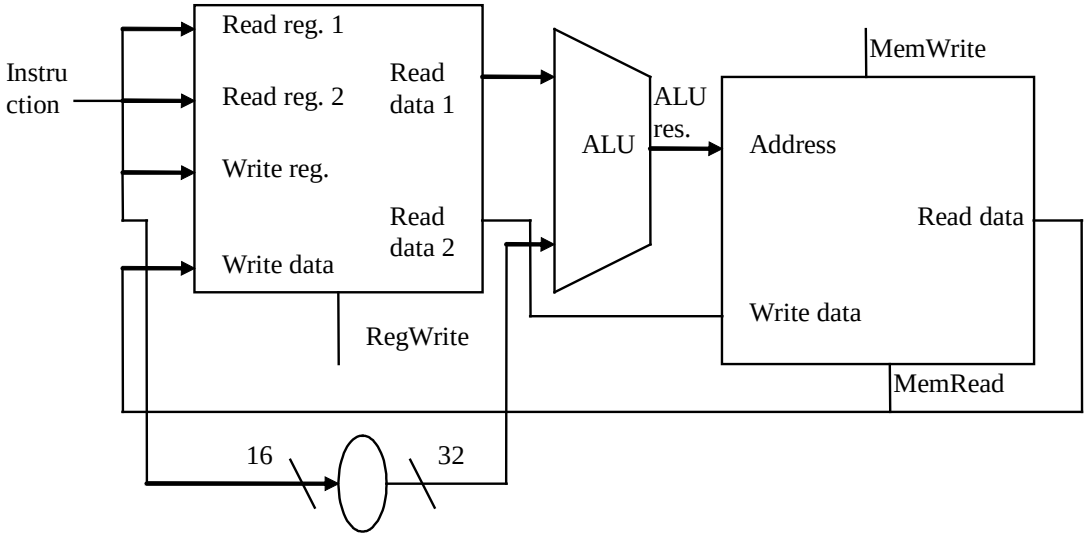


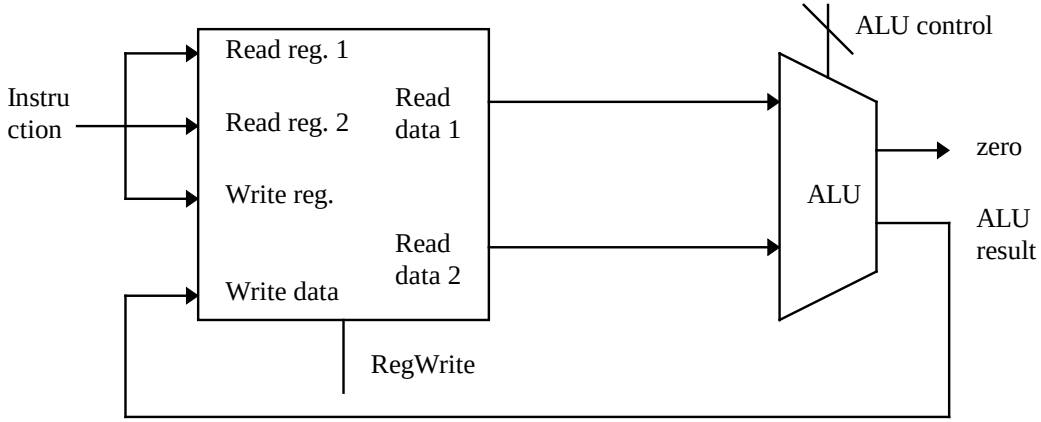
Ποιοι καταχωρήτες διαβάζονται, ποιος γράφεται, ποια πράξη εκτελείται και εάν θα αποθηκευτεί το αποτέλεσμα? Εξαρτάται από τα σήματα ελέγχου

Εντολές Μνήμης - load και store Instructions

- lw $t, offset($s)
- sw $t, offset($s)

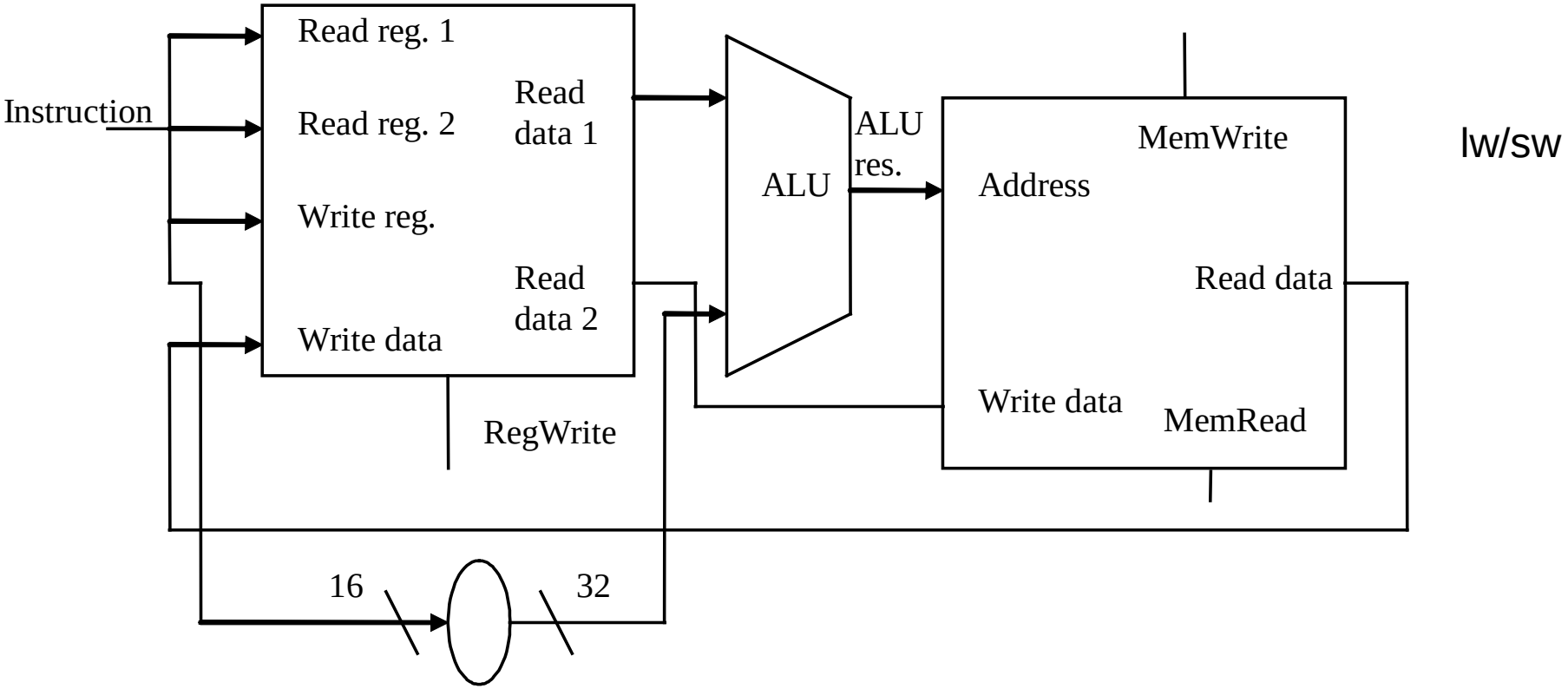
- Offset: sign-extended 16-bit value
  - 0x7a3f (0111 1010 0011 1111) => 0x00007a3f
  - 0x8a3f (1000 1010 0011 1111) => 0xffff8a3f





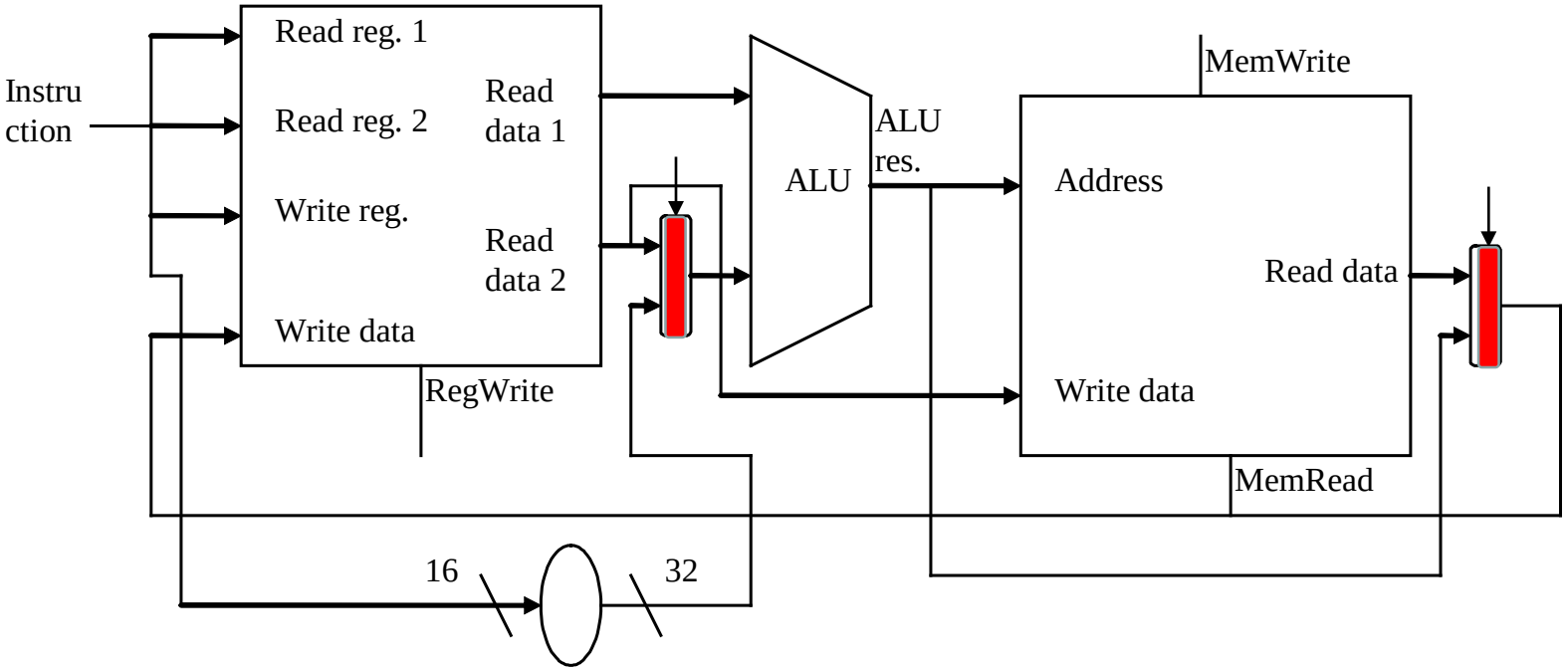
Combine R-type and I-type

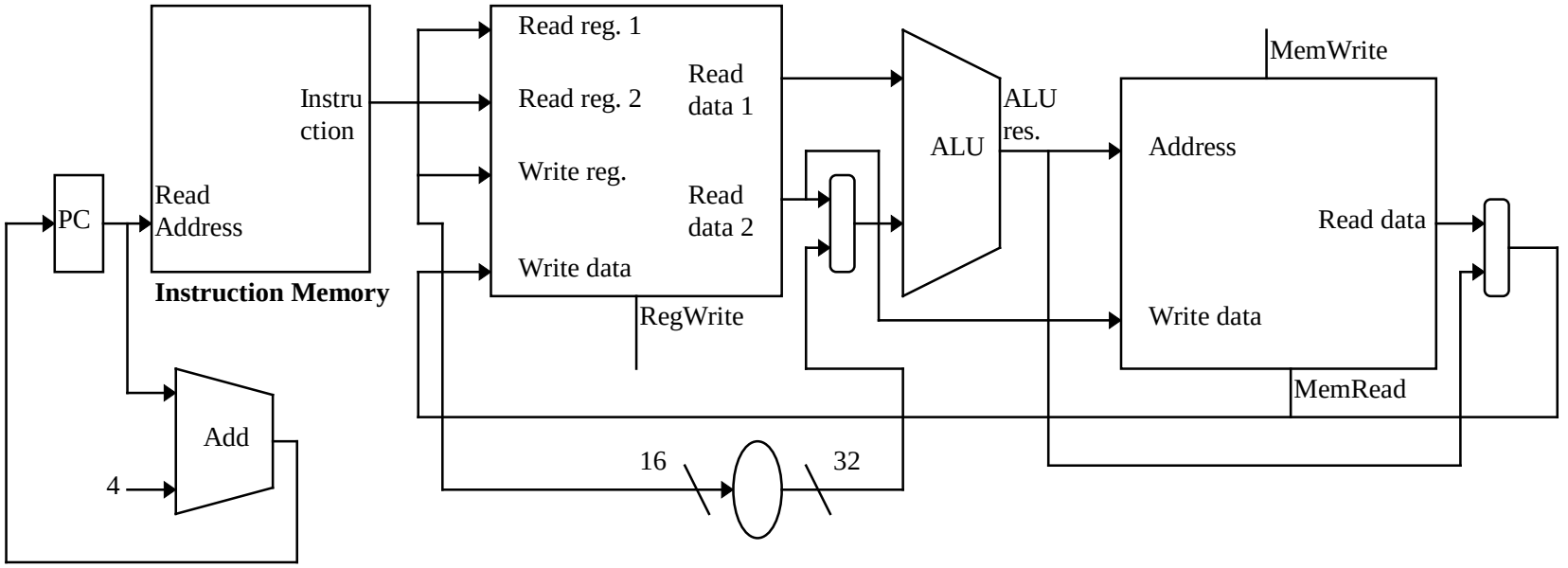
- two different sources for second ALU input???
- Two different sources for write-data in RFILE???



• Ένας πολυπλέκτης τοποθετείται στην είσοδο της ALU και

• Ένας πολυπλέκτης για τα δεδομένα εισόδου του αρχείου των καταχωρητών.




PC
Read
Address
Instru
ction
Instruction Memory
Add
4
Read reg. 1
Read reg. 2
Write reg.
Write data
Read
data 1
Read
data 2
RegWrite
16
32
ALU
ALU
res.
MemWrite
Address
Read data
Write data
MemRead

Εντολή Σύγκρισης - Branch Equal Instruction

• Για παράδειγμα η εντολή: beq $s, $t, offset έχει τρεις τελεστές:

- δύο καταχωρητές που συγκρίνονται για ισότητα

- ένα 16-bit offset για υπολογισμό της διεύθυνσης-στόχου (branch target address) που θα κατευθυνθεί η εκτέλεση του προγράμματος.

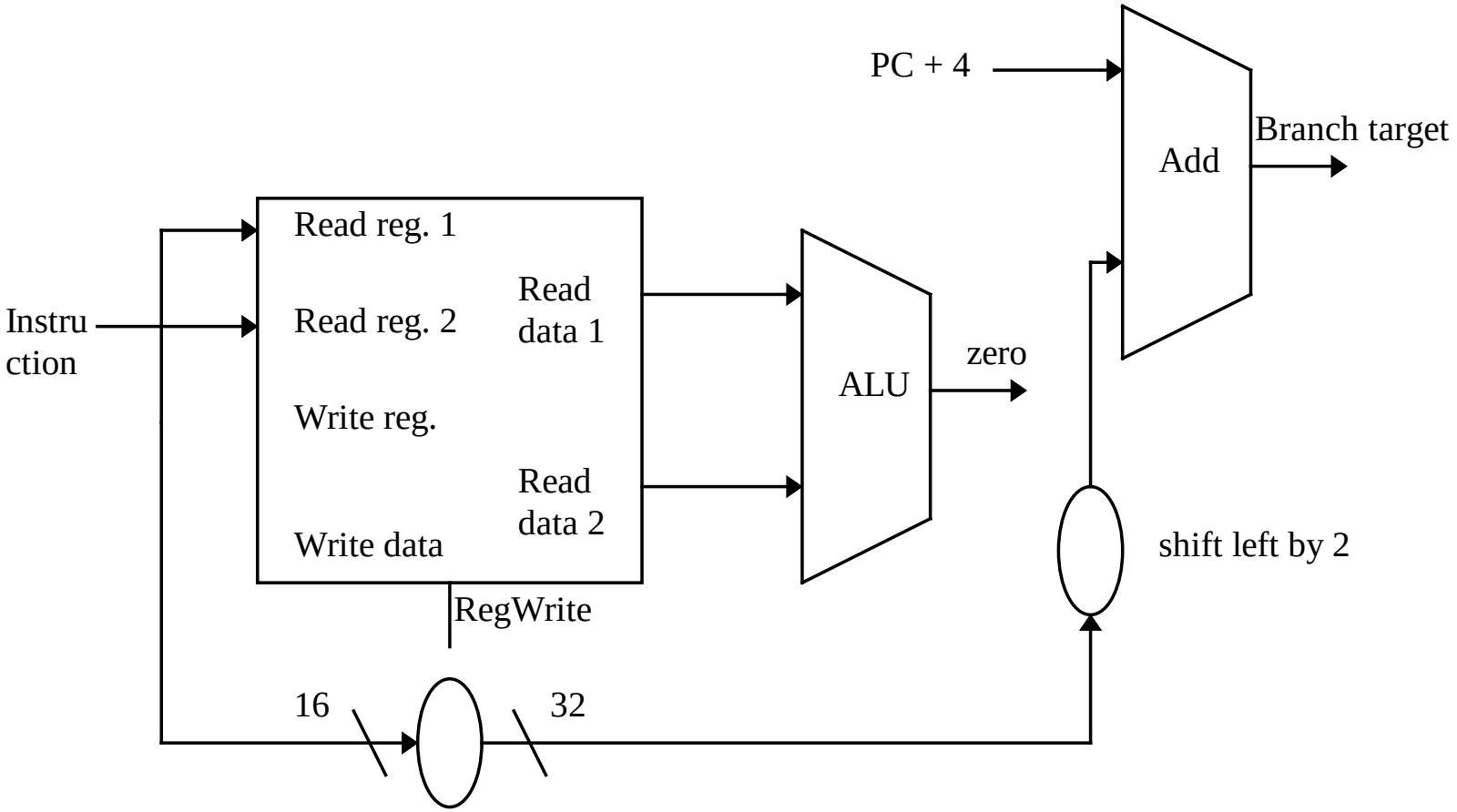
if ($s == $t)

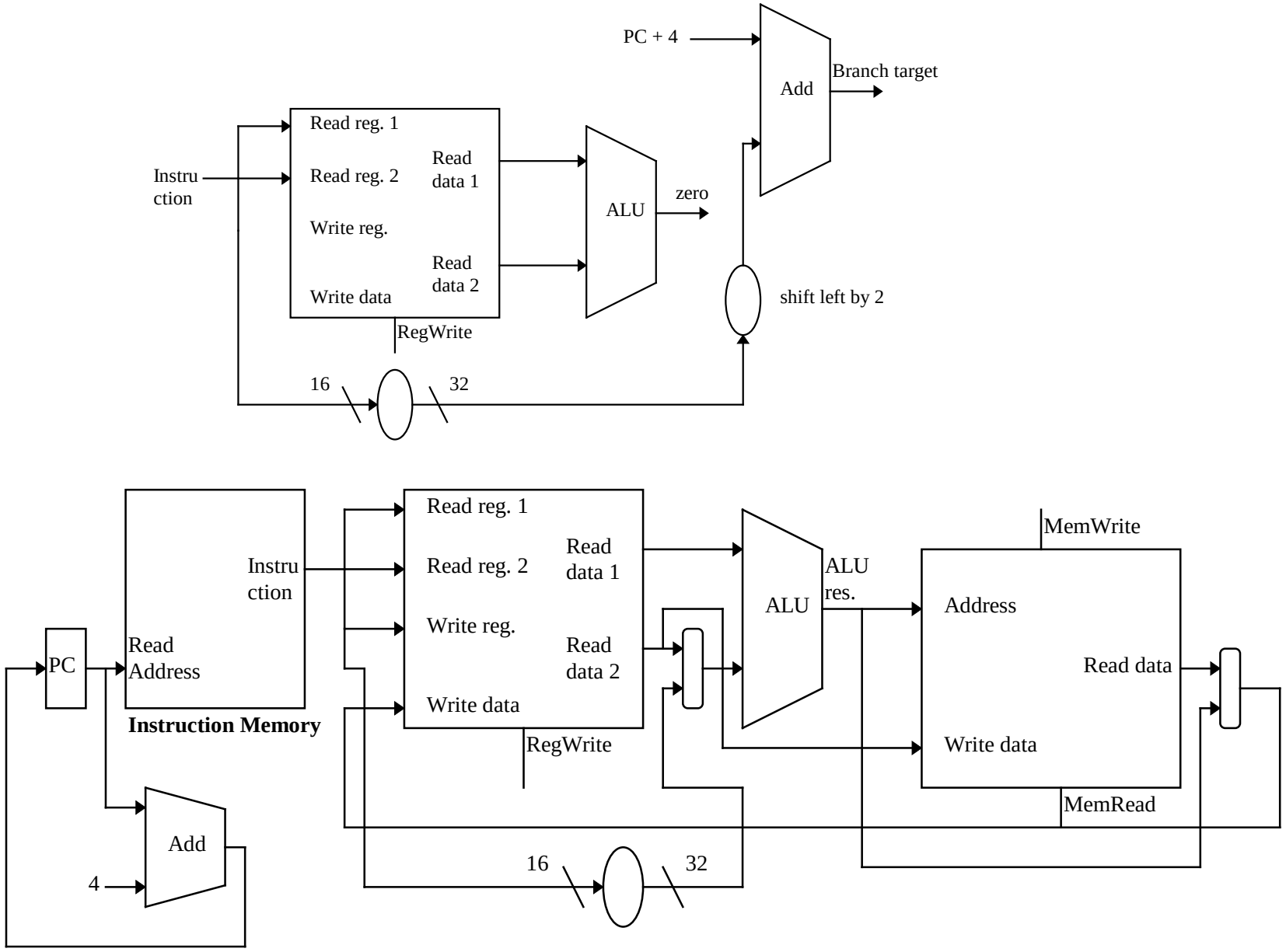
PC = PC + 4 + sign-extension(offset) x 4

Else

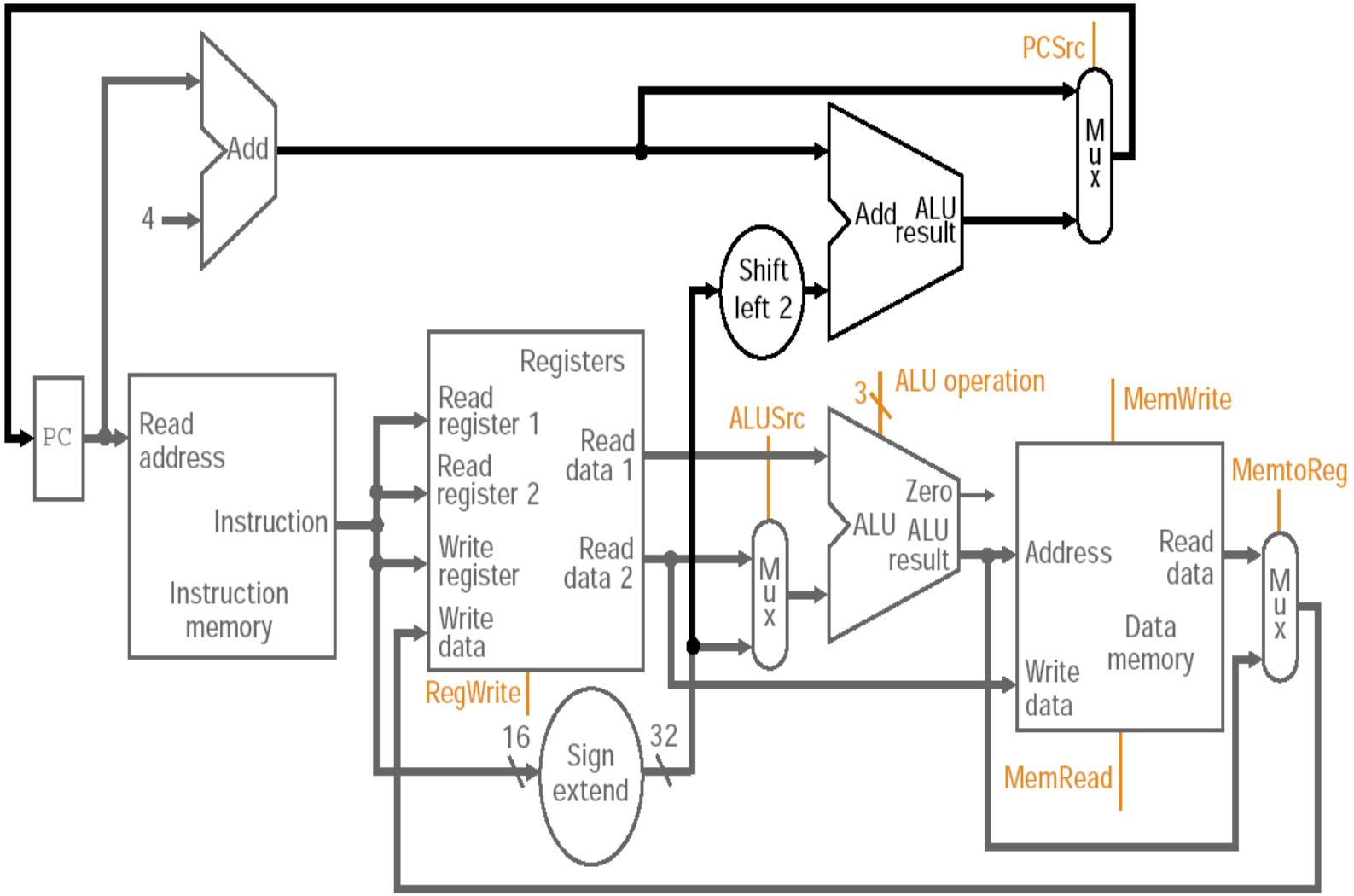
PC = PC + 4

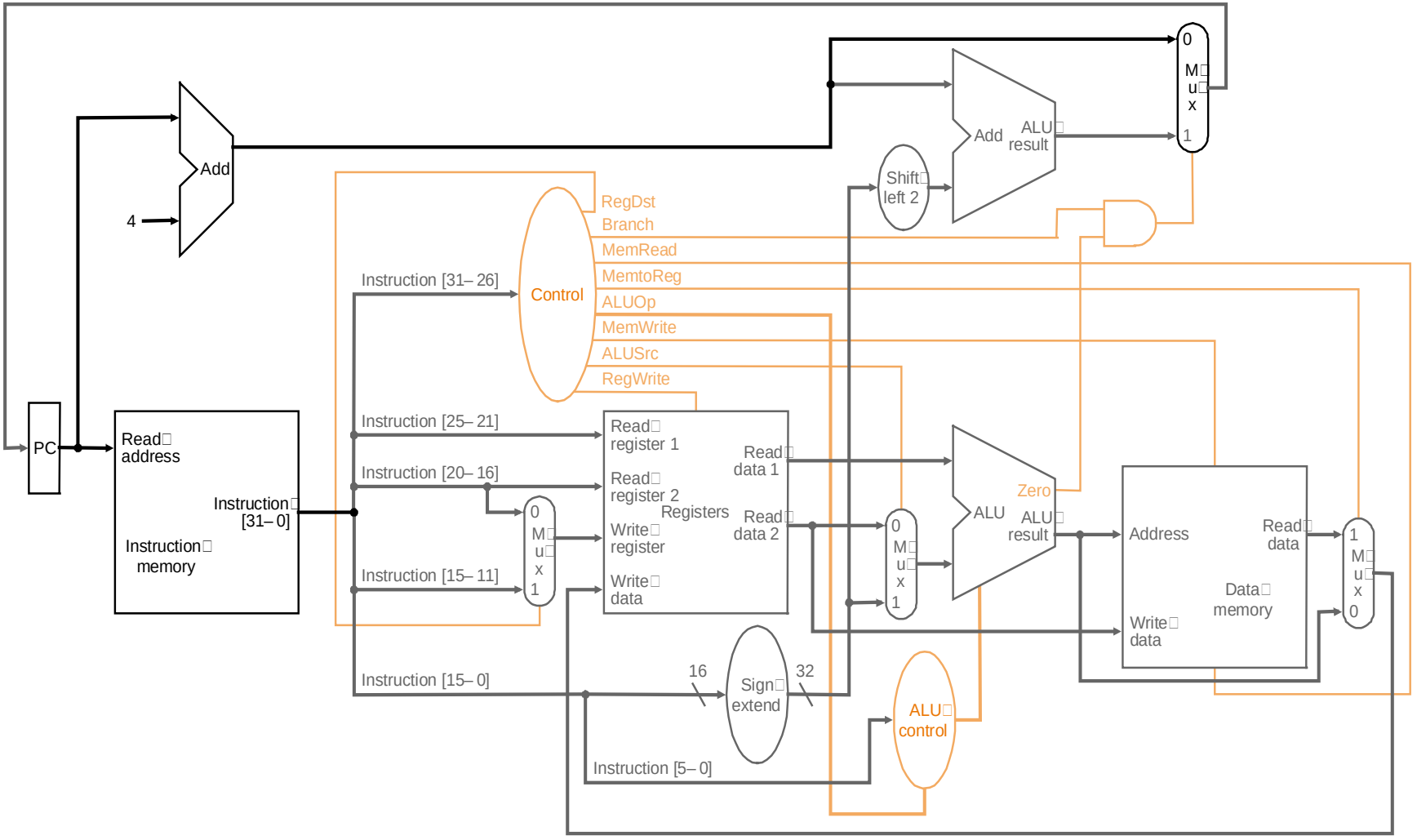
• Επίσης το πεδίο offset μετακινείται 2 bits αριστερά για να είναι word offset. Αυτή η μετακίνηση είναι βοηθητική επειδή αυξάνεται η αποτελεσματικότητα του πεδίου offset κατά ένα παράγοντα του 4.




PC + 4
Add
Branch target
Read reg. 1
Read reg. 2
Write reg.
Write data
Read data 1
Read data 2
Instru ction
ALU
zero
RegWrite
shift left by 2
16
32
Instruction Memory
Instru ction
Read Address
PC
Add
4
Read reg. 1
Read reg. 2
Write reg.
Write data
Read data 1
Read data 2
RegWrite
ALU
ALU res.
MemWrite
Address
Read data
Write data
MemRead
16
32

Διάδρομος δεδομένων για τις βασικές εντολές του MIPS: load/store, ALU, branch (ένα κύκλο ρολογιού.)




Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0
M
u
x
1
Shift left 2
Add
ALU result
16
Sign extend
32
ALU control
ALU
Zero
Address
Read data
Data memory
Write data

Signal	Effect when deasserted 0	Effect when asserted 1
MemRead	**None**	**Data memory contents at the read address are put on read data output.**
MemWrite	**None**	**Data memory contents at address given by write address is replaced by value on write data input.**
ALUSrc	**The second ALU operand comes from the second register file output.**	**The second ALU operand is the sign-extended lower 16-bits of the instruction.**
RegDst	**The register destination number for the Write register comes from the rt field.**	**The register destination number for the Write register comes from the rd field.**
RegWrite	**None**	**The register on the Write register input is written into with the value on the write data input.**
PCSrc	**The PC is replaced by the output of the adder that computes the value of PC+4.**	**The PC is replaced by the output of the adder that computes the branch target.**
MemtoReg	**The value fed to the register write data input comes from the ALU.**	**The value fed to the register write data input comes from the data memory.**

Μονάδα Έλεγχου της Αριθμητικής και Λογικής Μονάδας (ALU control)

Η ALU έχει τρεις εισόδους ελέγχου, αλλά μόνο πέντε από τους οκτώ (2^3) συνδυασμούς χρησιμοποιούνται

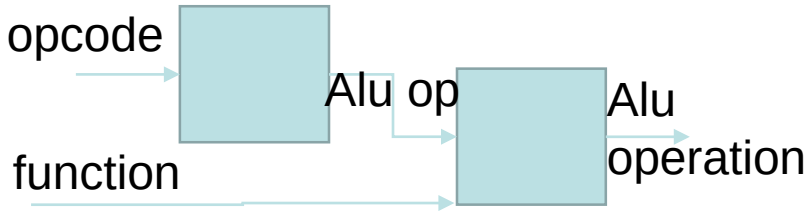
ALU control Input	Function
000	AND
001	OR
010	ADD
110	Subtract
111	Set-on-less-than

- lw,sw: add

- R-type: η ALU χρειάζεται να εκτελέσει μια από τις πέντε πράξεις ανάλογα με την τιμή του 6-bit πεδίου της εντολή (function field).

- beq: subtract.

• Μπορούμε να παράξουμε τα 3-bit ελέγχου της ALU (ALU control inputs) χρησιμοποιώντας *μια μικρή συνδυαστική μονάδα ελέγχου* που έχει ως εισόδους το **πεδίο function** της εντολής και ένα **πεδίο ελέγχου 2-bit που θα ονομάζουμε ALUOp**.



• Το ALUOp καθορίζεται από το opcode πεδίο της εντολής και προσδιορίζει τη πράξη που πρέπει να εκτελεστεί, π.χ.

> 00 - πρόσθεση για την εντολή load και store,

> 01 - αφαίρεση για την εντολή beq,

> 10 - λειτουργία που προσδιορίζεται από το πεδίο function της εντολής.

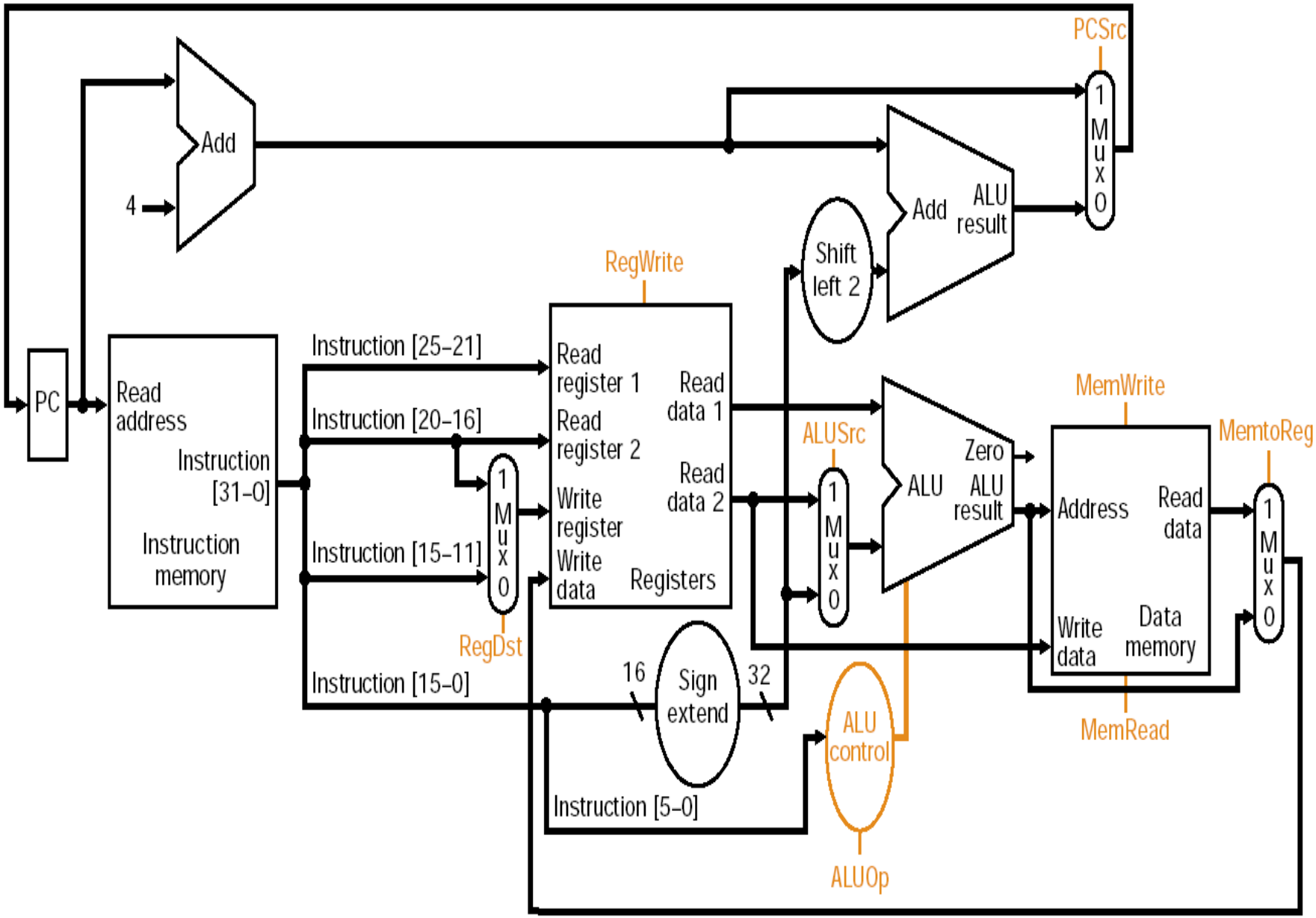
Opcode	ALU Op1	ALU Op2	
000000	1	0	Function field
100011	0	0	Add
101011	0	0	Add
000100	0	1	Subtract

Η παραγωγή των 3-bit ελέγχου της ALU βασίζεται στο 2-bit πεδίο ALUOp και στο 6-bit πεδίο function.

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control Input
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch equal	01	branch equal	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set-on less-than	101010	set-on-less-than	111

Ο πίνακας αληθείας για τα 3 bits ελέγχου του ALU

ALU OP1	ALU Op0	F5	F4	F3	F2	F1	F0	Operation
0	0	x	x	x	x	x	x	010
x	1	x	x	x	x	x	x	110
1	x	x	x	0	0	0	0	010
1	x	x	x	0	0	1	0	110
1	x	x	x	0	1	0	0	000
1	x	x	x	0	1	0	1	001
1	x	x	x	1	0	1	0	111



Κεφάλαιο 4: Ο επεξεργαστής: Διάδρομος δεδομένων και μονάδα έλεγχου (Κεφ. 4 από Patterson και Hennessy)

Σχεδιασμός του Κύριου Μέρους της Μονάδας Ελέγχου

- R-type

0	rs	rt	rd	shamt	funct
31 - 26	25 - 21	20 - 16	15 - 11	10 - 6	5 - 0

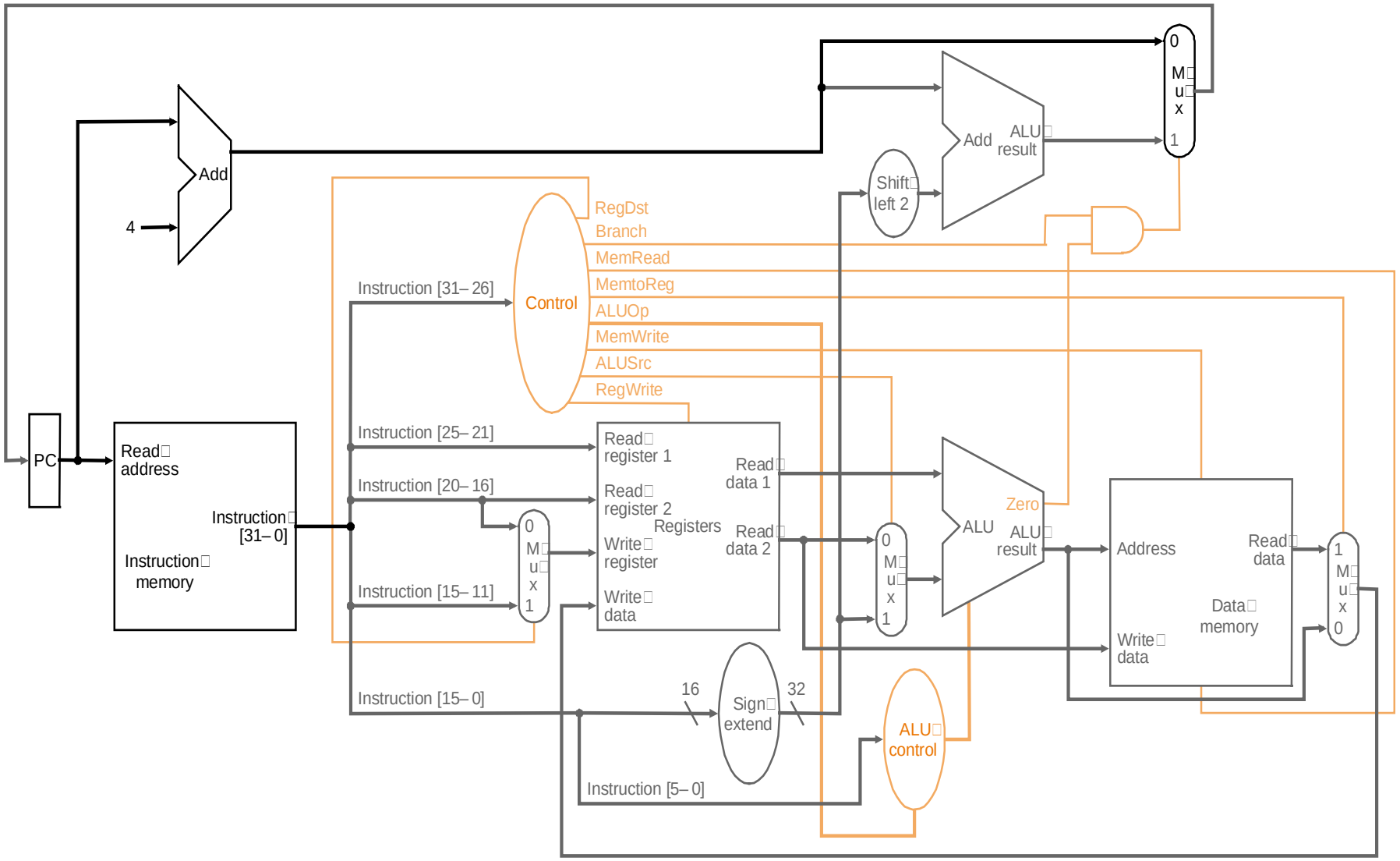
- Load & Store

35 or 43	rs	rt	address
31 – 26	25 - 21	20 - 16	15 - 0

-Branch

4	rs	rt	address
31 - 26	25 - 21	20 - 16	15 - 0

- **Το opcode → bits 31-26 Op[5-0].**
- **rs και rt → bits 25-21 και 20-16, αντίστοιχα.**
- **Ο καταχωρητής βάσης (base register) → 25-21 (rs).**
- **Το 16-bit πεδίο offset (branch equal, load και store) → bits 15-0.**
- **Ο καταχωρητής προορισμού (Χρήση Πολυπλέκτη)**
  - **είναι στη θέση 20-16 (rt) για την εντολή load.**
  - **Για τις εντολές R-type βρίσκεται στη θέση 15-11 bit**



Κεφάλαιο 4: Ο επεξεργαστής: Διάδρομος δεδομένων και μονάδα έλεγχου (Κεφ. 4 από Patterson και Hennessy)

• Η λειτουργία της μονάδας ελέγχου καθορίζεται πλήρως από το πίνακα.

Instruct opcode (31:25)	RegDst	ALUSrc	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-type									
lw									
sw									
beq									

Reg Dst	ALUSrc	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0	Jump		

• Η λειτουργία της μονάδας ελέγχου καθορίζεται πλήρως από το πίνακα.

Instruct	RegDst	ALUSrc	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	x	1	x	0	0	1	0	0	0
beq	x	0	x	0	0	0	1	0	1

• Κωδικοί εντολών

Name	Opcode in decimal	Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0	0
lw	35	1	0	0	0	1	1
sw	43	1	0	1	0	1	1
beq	4	0	0	0	1	0	0

Αληθοπίνακας της λειτουργίας της μονάδας ελέγχου

		R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	Regdst	1	0	x	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

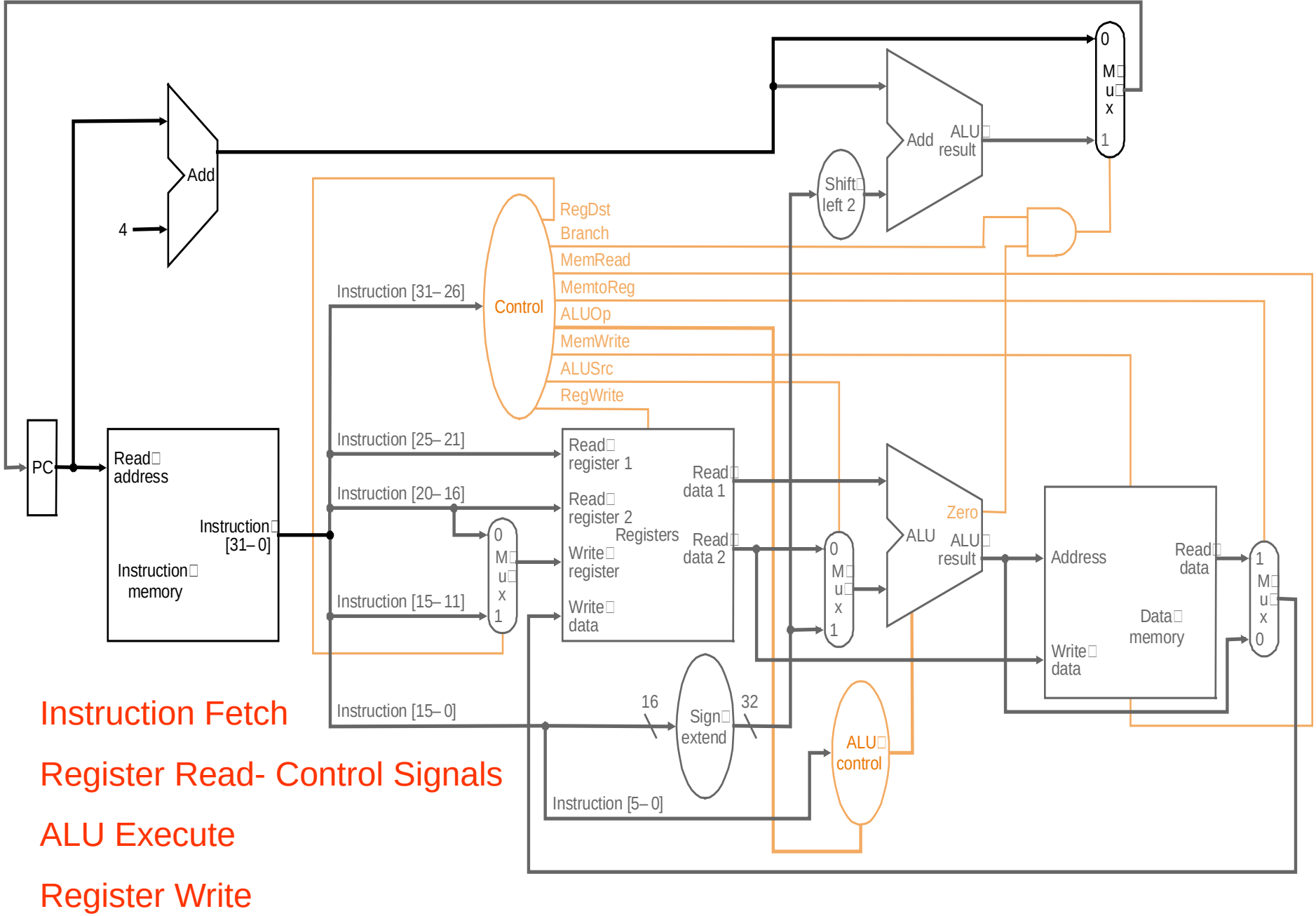
Εκτέλεση της εντολής R-type--add $x,$y,$z.

1. Μια εντολή προσκομίζεται από τη μνήμη εντολών το PC αυξάνεται

2. Οι καταχωρητές $y και $z διαβάζονται από το αρχείο των καταχωρητών. Επίσης η κύρια μονάδα ελέγχου υπολογίζει τις τιμές για τις γραμμές ελέγχου.

3. Το ALU λειτουργεί πάνω στα δεδομένα που διαβάστηκαν από το αρχείο των καταχωρητών χρησιμοποιώντας το function code της εντολής, για να επιλέξει την λειτουργία της ALU.

4. Το αποτέλεσμα από την ALU γράφεται στο αρχείο των καταχωρητών χρησιμοποιώντας bits 15-11 από την εντολή για να επιλέξει τον καταχωρητή προορισμού (εξόδου) $x.


Add
4
0
M
u
x
1
Add
ALU
result
Shift
left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
PC
Read
address
Instruction
[31–0]
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
ALU
Zero
ALU
result
Address
Read
data
Data
memory
Write
data
16
Sign
extend
32
ALU
control
Instruction Fetch
Register Read- Control Signals
ALU Execute
Register Write

Εκτέλεση της εντολής load-- lw $x,offset($y).

1. Η εντολή προσκομίζεται από τη μνήμη εντολών και αυξάνεται η τιμή του PC.

2. Ο καταχωρητής ($y) διαβάζεται από το αρχείο των καταχωρητών.

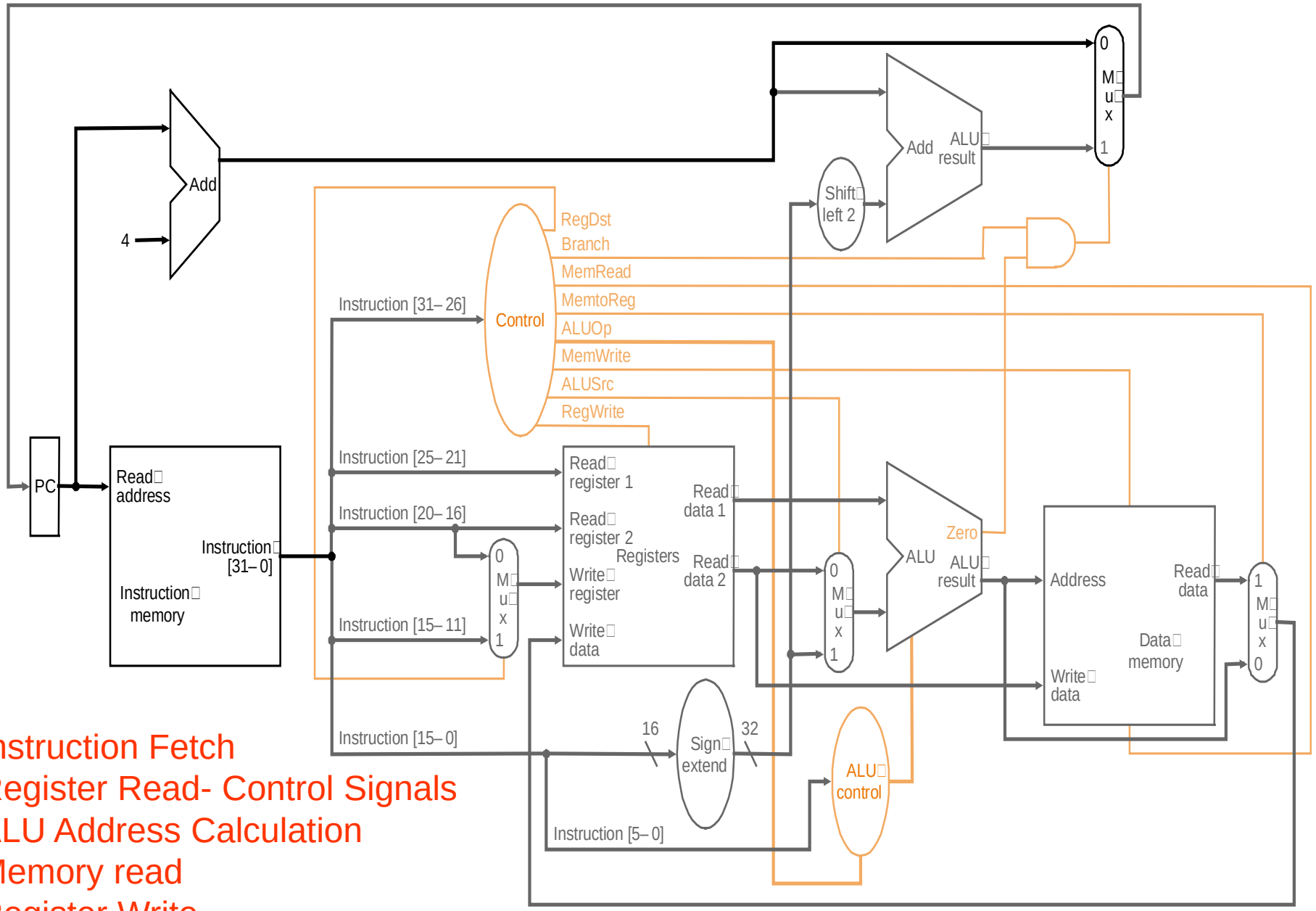
3. Η ALU υπολογίζει το άθροισμα της τιμής που διαβάστηκε από το αρχείο των καταχωρητών και την σταθερά (προέκταση 16bits της εντολής).

4. Το άθροισμα από το ALU χρησιμοποιείται ως η διεύθυνση για τη μνήμη των δεδομένων.

5. Τα δεδομένα από τη μονάδα μνήμης γράφονται στο αρχείο του καταχωρητή. Ο καταχωρητής προορισμού δίνεται από τα 20-16 bits της εντολής.

• Για την εντολή store η κυριότερη διαφορά είναι στο έλεγχο μνήμης όπου πρέπει να προσδιοριστεί έλεγχος γραφής και όχι διαβάσματος.

- Η λειτουργία γραφής των δεδομένων της μνήμης στο αρχείο των καταχωρητών δεν θα υπάρχει.



Instruction Fetch
Register Read- Control Signals
ALU Address Calculation
Memory read
Register Write

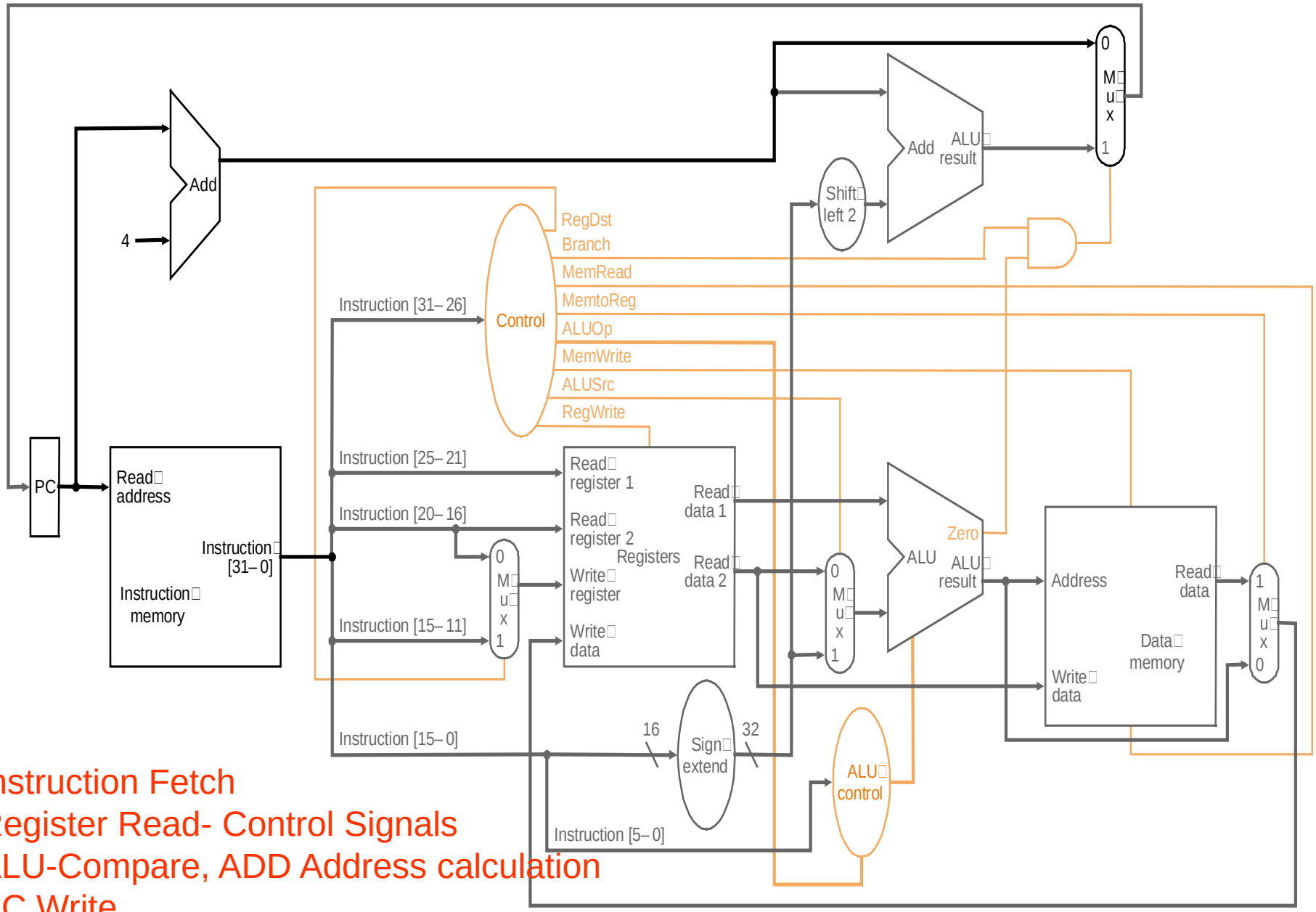
Εκτέλεση της εντολής branch-on-equal (υπό συνθήκη)—beq $x,$y,offset.

1. Η εντολή προσκομίζεται από τη μνήμη εντολών και αυξάνεται η τιμή του PC.

2. Δύο καταχωρητές ($x,$y) διαβάζονται από το αρχείο των καταχωρητών.

3. Η ALU αφαιρεί τις τιμές των δεδομένων που διάβασε από το αρχείο των καταχωρητών.
   - Η τιμή του PC+4 προστίθεται στο σταθερά (sign-extended lower 16 bits της εντολής (offset)),
   - το αποτέλεσμα είναι η διεύθυνση - στόχος του branch.

4. Το αποτέλεσμα μηδέν από την ALU χρησιμοποιείται για να επιλέξουμε το αποτέλεσμα του αθροιστή που θα αποθηκεύσουμε στο PC.

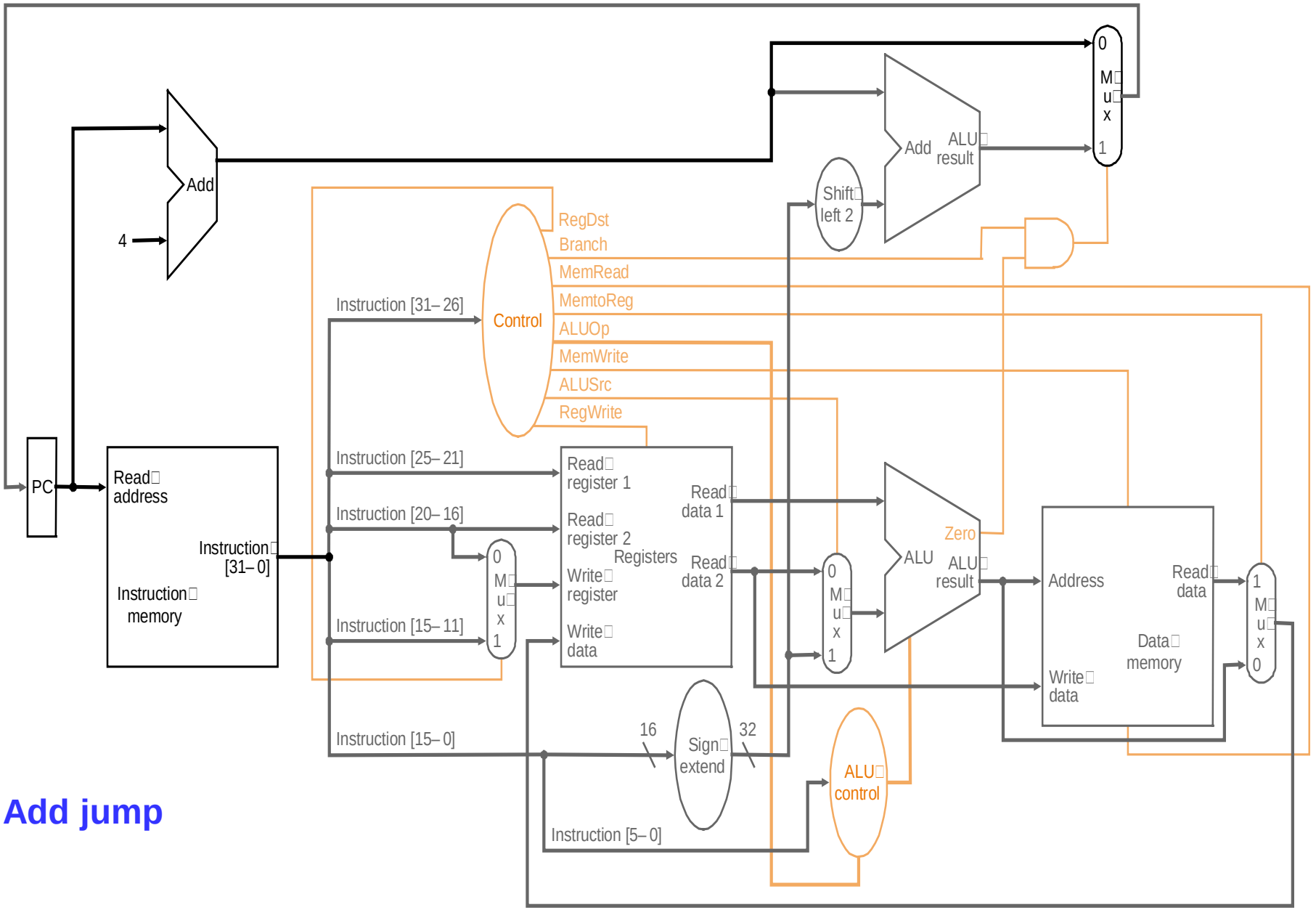


Instruction Fetch

Register Read- Control Signals

ALU-Compare, ADD Address calculation

PC Write


Add
4
M
u
x
0
1
Add
ALU
result
Shift
left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
PC
Read
address
Instruction
[31–0]
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
Zero
ALU
ALU
result
Address
Read
data
Data
memory
Write
data
16
Sign
extend
32
ALU
control

Add jump

Υλοποίηση της Εντολής Άλμα (jump)

–Πώς μπορεί να επεκταθεί η υλοποίηση ώστε να περιέχει και την εντολή jump?

–Πως θα δοθούν (αρχικές) τιμές στις νέες γραμμές ελέγχου?

• Η εντολή jump μοιάζει με την εντολή branch,

–αλλά υπολογίζει διαφορετικά τη διεύθυνση-στόχος PC

–και εκτελείται πάντα.

• Όπως στη branch, τα low order 2 bits της διεύθυνσης (της jump) είναι πάντοτε 00.

• Τα υπόλοιπα lower 26 bits από αυτά της 32 bits διεύθυνσης προέρχονται από το 26 bit immediate πεδίο της εντολής.

• Τα upper 4 bits της διεύθυνσης προέρχονται από το τρέχον PC.

• Έτσι μπορούμε να υλοποιήσουμε την jump αποθηκεύοντας στο PC την ένωση των:

- upper four bits του τρέχον PC (bits 31-28)
- τα 26 bits του πεδίου immediate της εντολής jump.
- τα bits 00

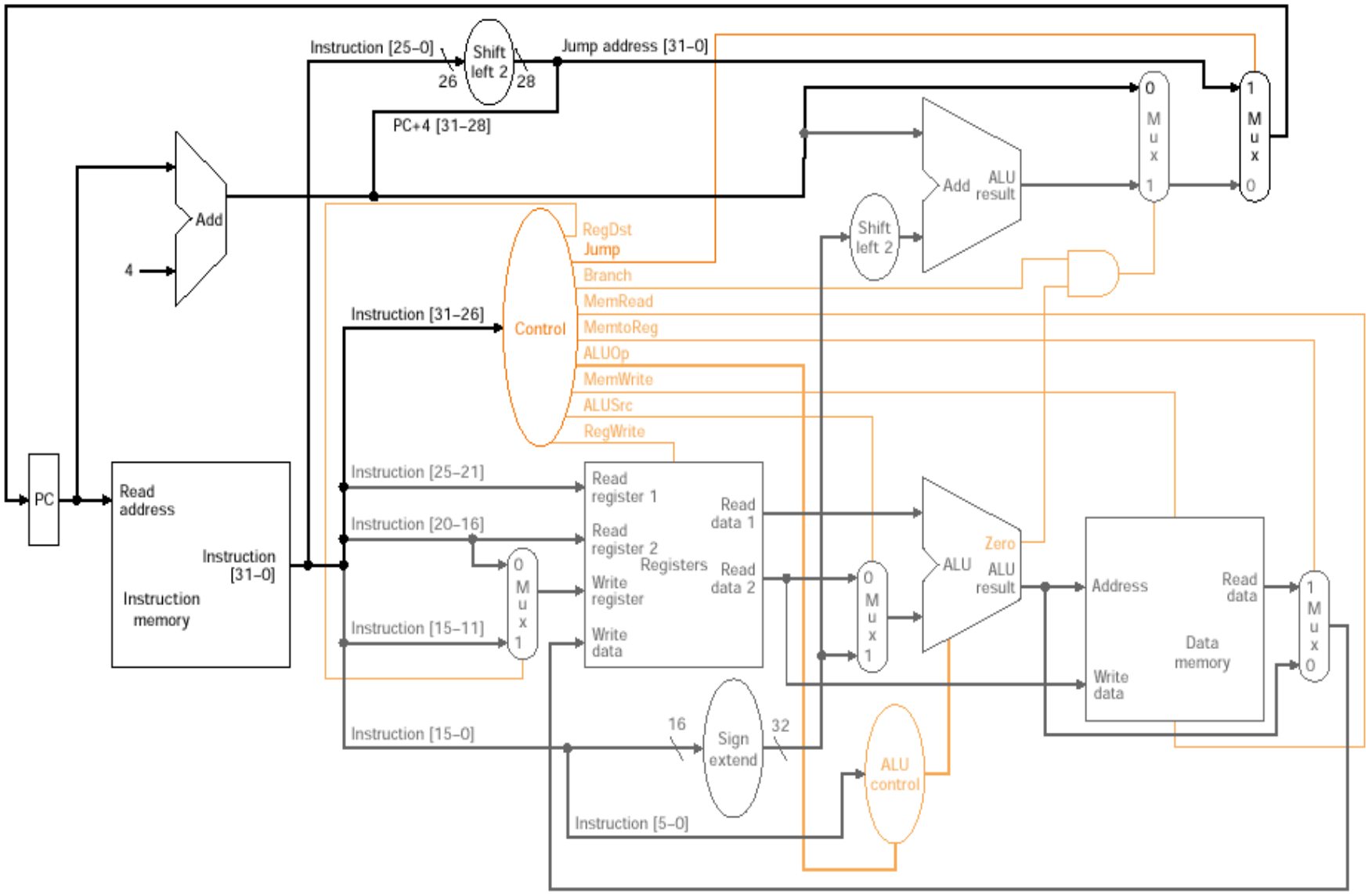
Προσθήκες στο βασικό σχεδιασμό (Σχήμα 5.19) για την υλοποίηση της Εντολής Άλμα (jump)

- Προσθέτουμε έναν επιπλέον πολυπλέκτη για επιλογή της πηγής για τη καινούργια τιμή του PC
  - –που θα είναι είτε PC+4 ,
  - –είτε branch target PC,
  - –είτε jump target PC.

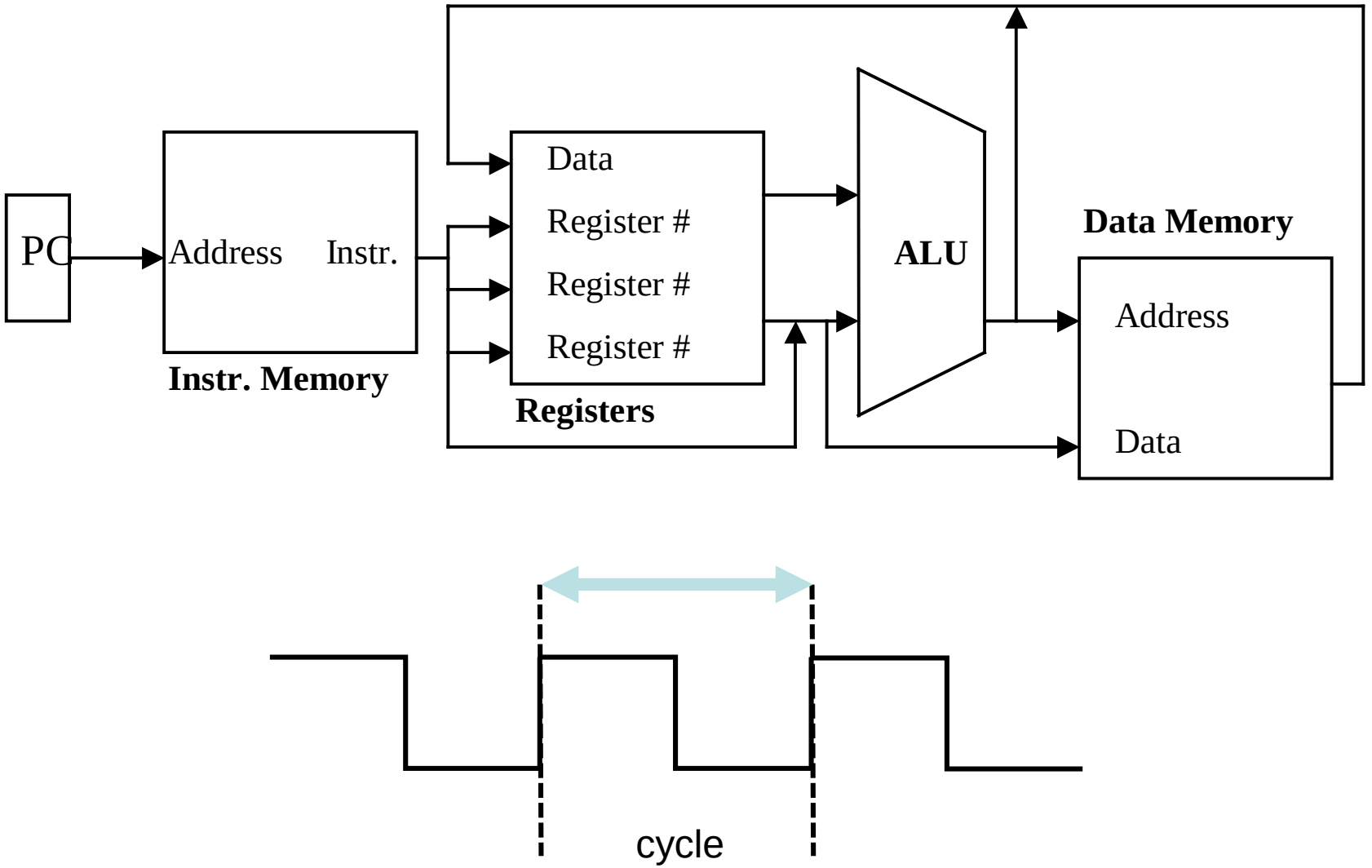
- Προσθέτουμε ένα επιπλέον σήμα ελέγχου χρειάζεται για τον επιπρόσθετο πολυπλέκτη.
  - –Το σήμα ελέγχου ονομάζεται Jump που ενεργοποιείτε όταν η εντολή είναι jump, δηλ. όταν το opcode είναι 2.

Αληθοπίνακας της λειτουργίας της μονάδας ελέγχου

		R-format	lw	sw	beq	Jump
Inputs	Op5	0	1	1	0	
	Op4	0	0	0	0	
	Op3	0	0	1	0	
	Op2	0	0	0	1	
	Op1	0	1	1	0	
	Op0	0	1	1	0	
Outputs	Regdst	1	0	x	X	
	ALUSrc	0	1	1	0	
	MemtoReg	0	1	x	X	
	RegWrite	1	1	0	0	
	MemRead	0	1	0	0	
	MemWrite	0	0	1	0	
	Branch	0	0	0	1	
	ALUOp1	1	0	0	0	
	ALUOp0	0	0	0	1	
	JUMP	0	0	0	0	1



Κεφάλαιο 4: Ο επεξεργαστής: Διάδρομος δεδομένων και μονάδα έλεγχου (Κεφ. 4 από Patterson και Hennessy)


PC
Address
Instr.
Instr. Memory
Data
Register #
Register #
Register #
Registers
ALU
Data Memory
Address
Data
cycle