Αρχιτεκτονική Υπολογιστών

Κεφάλαιο #4 (α)

Single Cycle Datapath Design

Διονύσης Πνευματικάτος

pnevmati@cslab.ece.ntua.gr

5ο εξάμηνο ΣΗΜΜΥ – Ακαδημαϊκό Έτος: 2019-20

Τμήμα 3 (ΠΑΠΑΔ-Ω)

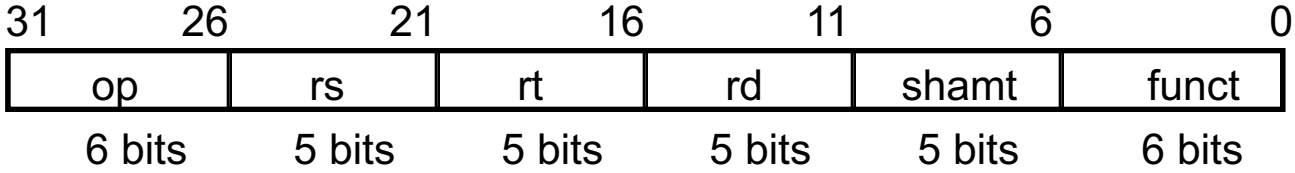
http://www.cslab.ece.ntua.gr/courses/comparch/

Εισαγωγή

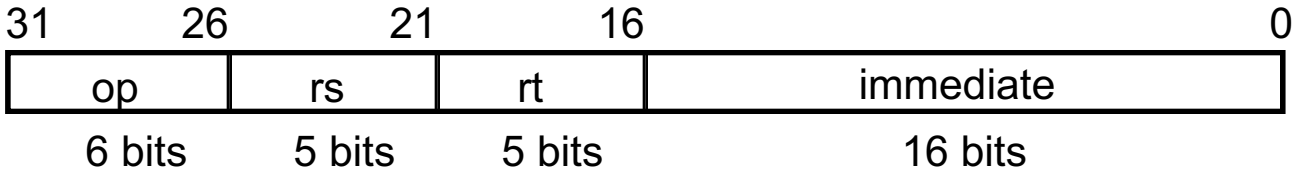
- Παράγοντες απόδοσης της CPU
  - Πλήθος εντολών
    - Καθορίζεται από την αρχιτεκτονική συνόλου εντολών και το μεταγλωττιστή
  - CPI και Χρόνος κύκλου
    - Καθορίζεται από το υλικό της CPU
- Θα εξετάσουμε δύο υλοποιήσεις του MIPS
  - Μια απλουστευμένη έκδοση
  - Μια πιο ρεαλιστική έκδοση με διοχέτευση (pipeline)
- Απλό υποσύνολο, δείχνει τις περισσότερες πτυχές
  - Αναφορά μνήμης: `lw`, `sw`
  - Αριθμητικές/λογικές: `add`, `sub`, `and`, `or`, `slt`
  - Μεταφοράς ελέγχου: `beq`, `j`

The MIPS Instruction Formats

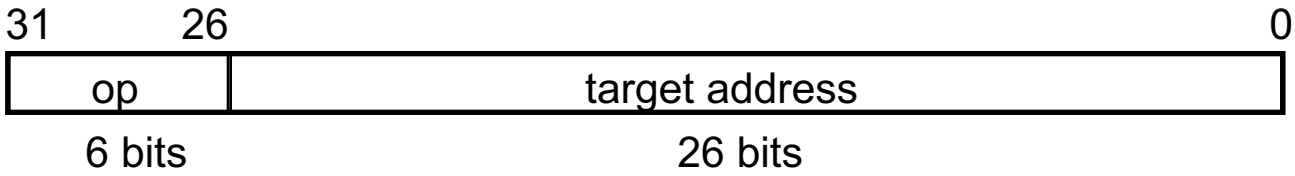
- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type



  - I-type



  - J-type

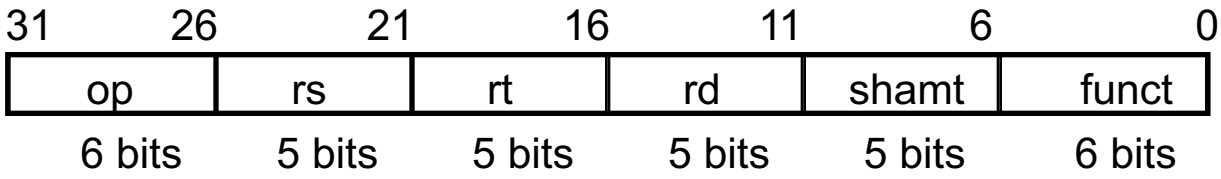


- The fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS Subset for today

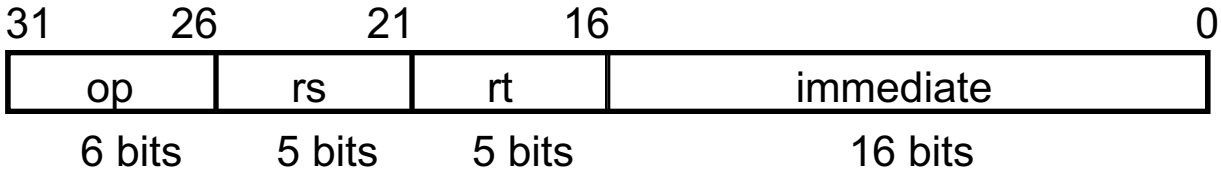
ADD and SUB

- addU rd, rs, rt
- subU rd, rs, rt



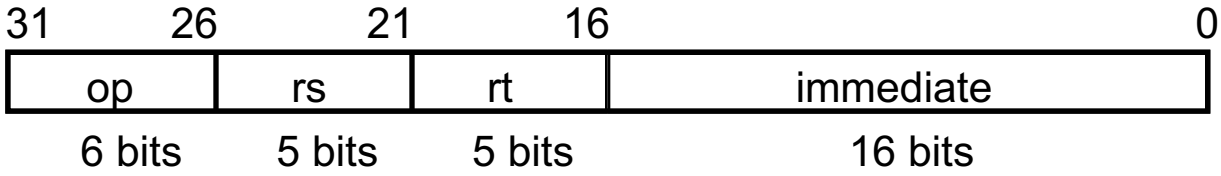
OR Immediate:

- ori rt, rs, imm16



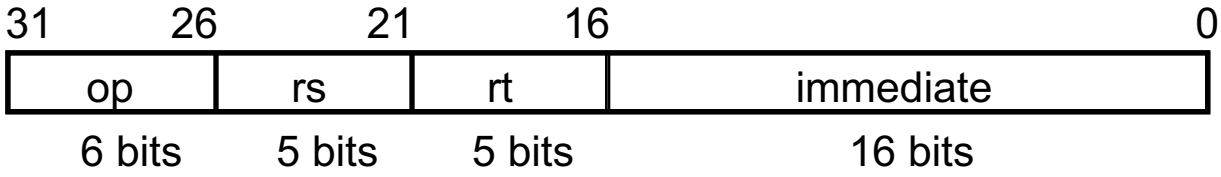
LOAD and STORE Word

- lw rt, rs, imm16
- sw rt, rs, imm16



BRANCH:

- beq rs, rt, imm16



Εκτέλεση εντολής

- PC → μνήμη εντολών, προσκόμιση (fetch) εντολής
- Αριθμοί καταχωρητών → αρχείο καταχωρητών (register file), ανάγνωση καταχωρητών
- Ανάλογα με τη κατηγορία της εντολής
  - Χρήση της ALU για τον υπολογισμό
    - Αριθμητικού αποτελέσματος
    - Διεύθυνσης μνήμης για εντολές load/store
    - Διεύθυνση προορισμού διακλάδωσης
  - Προσπέλαση μνήμης δεδομένων για load/store
  - PC ← διεύθυνση προορισμού ή PC + 4

Logical Register Transfers

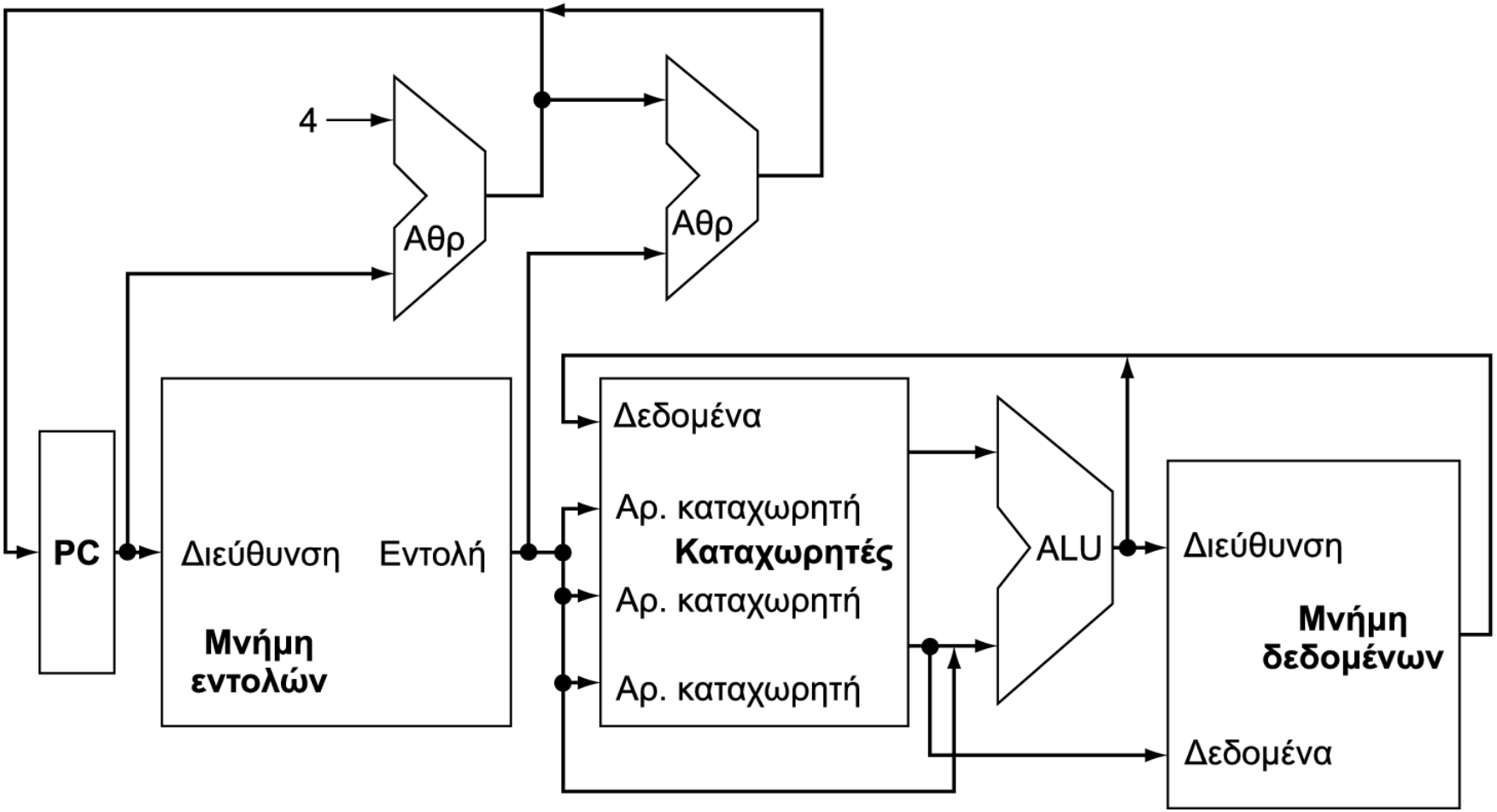
- RTL gives the meaning of the instructions
- All start by fetching the instruction

op | rs | rt | rd | shamt | funct = MEM[PC]

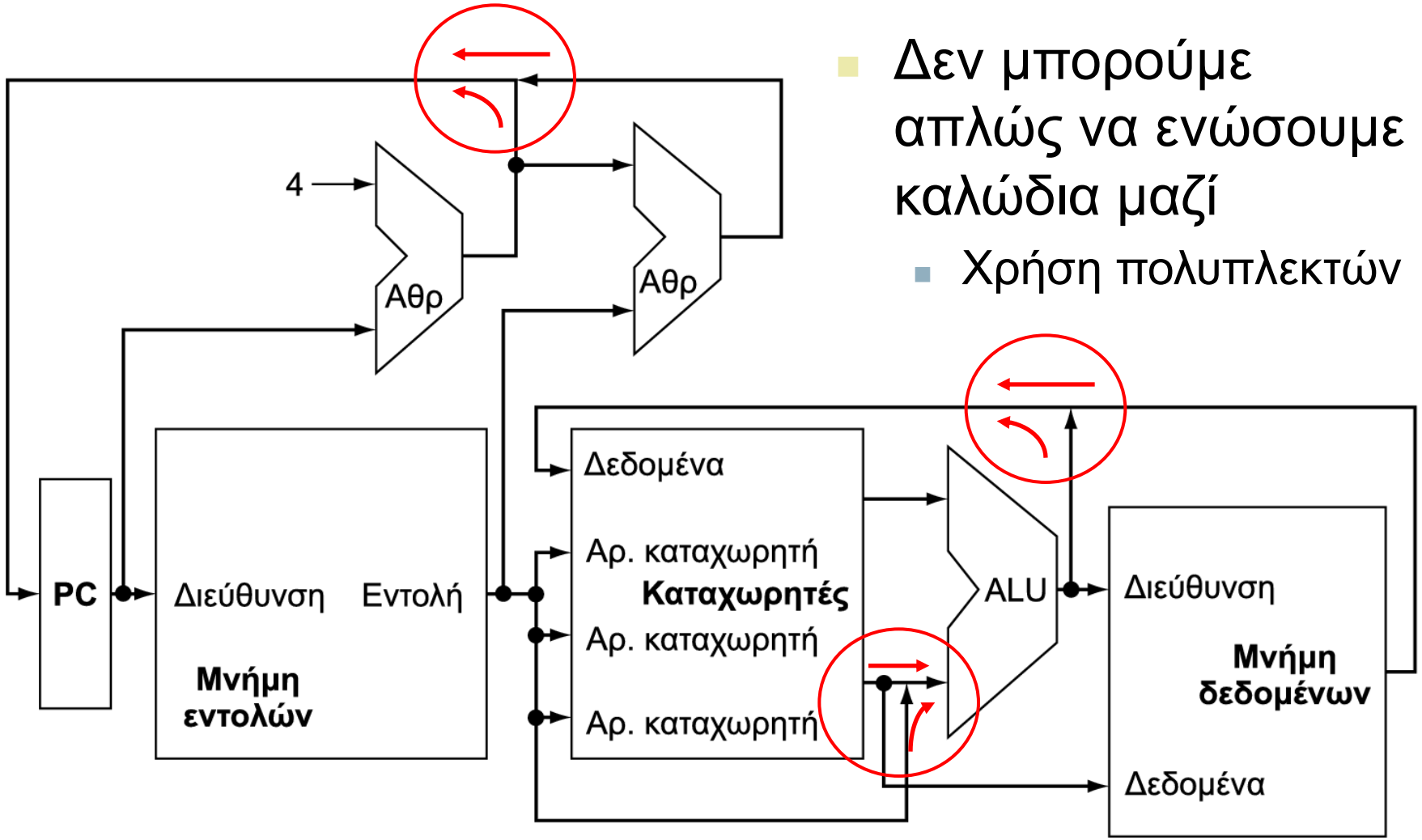
op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] | zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then PC <– PC + 4 + sign_ext(Imm16)] || 00 else PC <– PC + 4	

Επισκόπηση της CPU

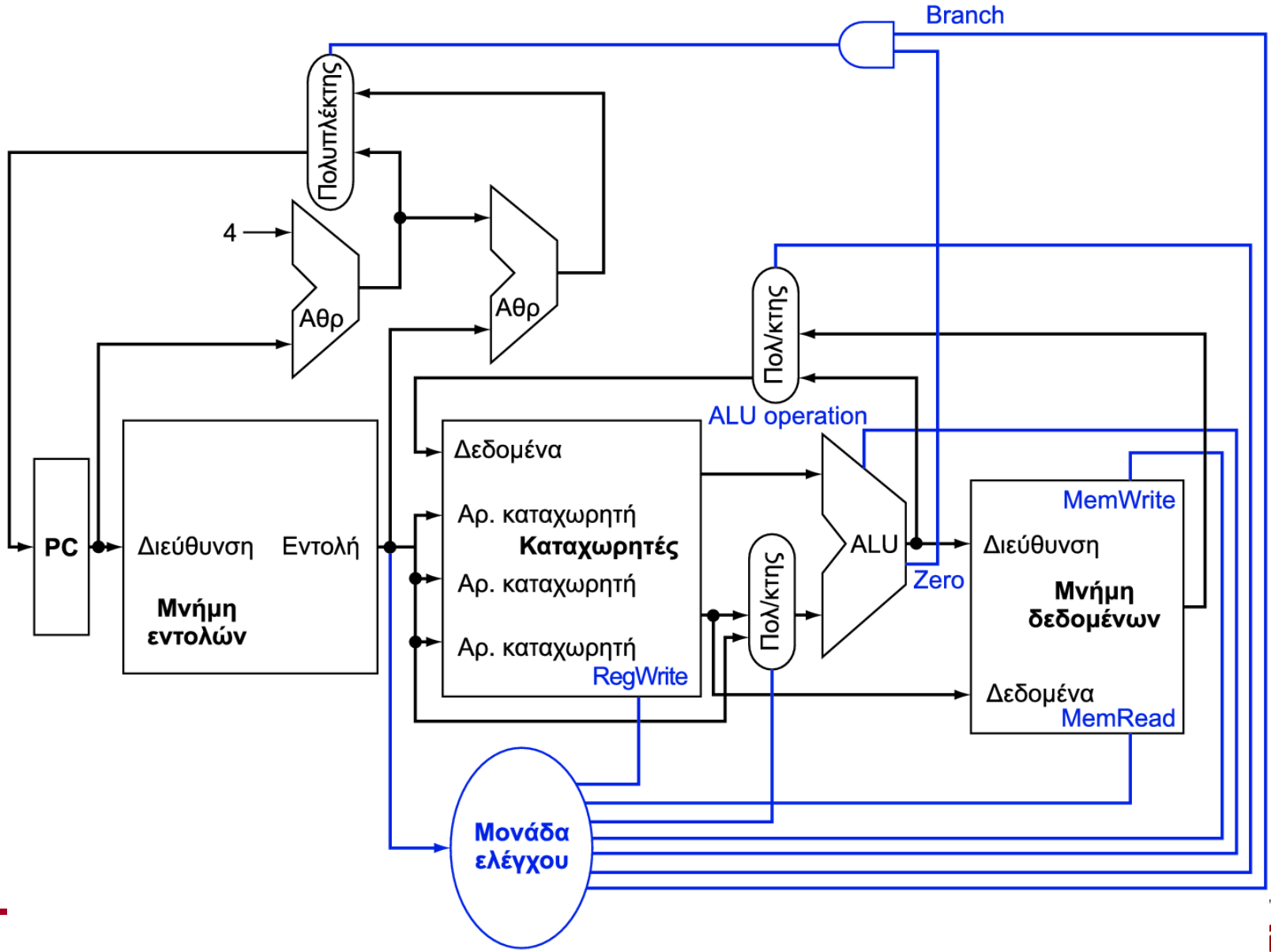


Πολυπλέκτες



- Δεν μπορούμε απλώς να ενώσουμε καλώδια μαζί
  - Χρήση πολυπλεκτών

Έλεγχος

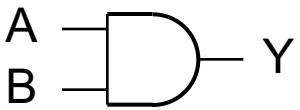

Branch
Πολυπλέκτης
4
Αθρ
Αθρ
Πολ/κτης
ALU operation
PC
Διεύθυνση
Εντολή
Μνήμη εντολών
Δεδομένα
Αρ. καταχωρητή
Καταχωρητές
Αρ. καταχωρητή
Αρ. καταχωρητή
RegWrite
Πολ/κτης
ALU
Zero
MemWrite
Διεύθυνση
Μνήμη δεδομένων
Δεδομένα
MemRead
Μονάδα ελέγχου

Βασικά λογικής σχεδίασης

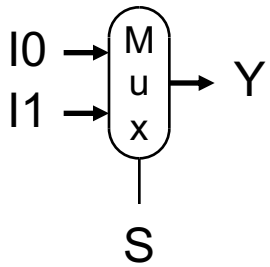
- Η πληροφορία κωδικοποιείται δυαδικά
  - Χαμηλή τάση = 0, Υψηλή τάση = 1
  - Ένα καλώδιο ανά bit
  - Δεδομένα πολλών bit κωδικοποιούνται με διαύλους πολλών καλωδίων
- Συνδυαστικό στοιχείο
  - Επενεργεί σε δεδομένα
  - Η έξοδος είναι συνάρτηση της εισόδου
- Στοιχεία κατάστασης (ακολουθιακά)
  - Αποθηκεύουν πληροφορίες

Συνδυαστικά στοιχεία

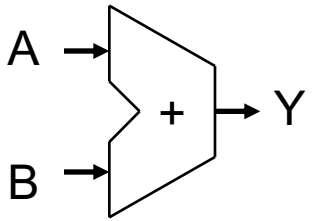
- Πύλη AND
  - Y = A & B



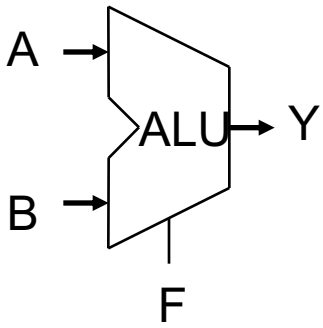
- Πολυπλέκτης
  - Y = S ? I1 : I0



- Αθροιστής
  - Y = A + B

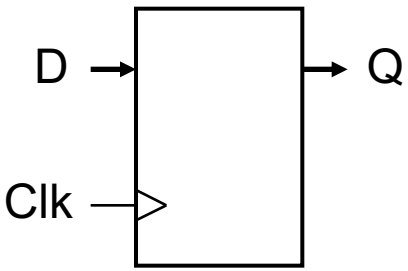


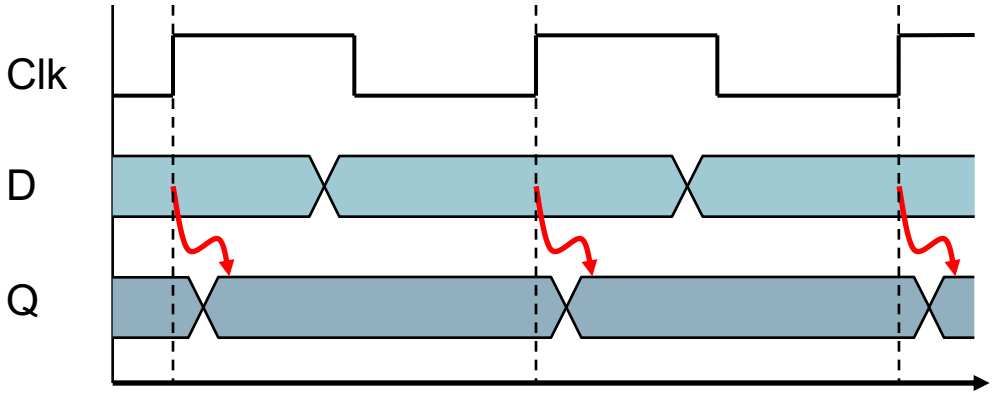
- Αριθμητική/Λογική Μονάδα
  - Y = F(A, B)



Ακολουθιακά στοιχεία

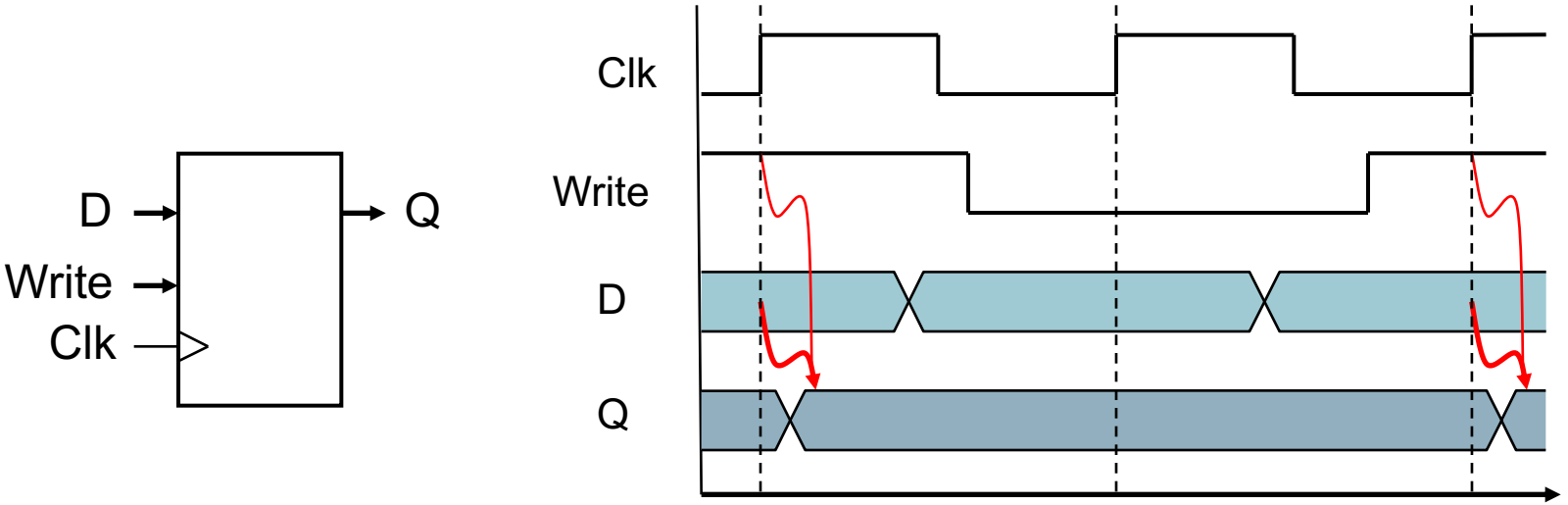
- Καταχωρητής: αποθηκεύει δεδομένα σε ένα κύκλωμα
  - Χρησιμοποιεί σήμα ρολογιού για να καθορίσει πότε ενημερώνεται η αποθηκευμένη τιμή
  - Ακμοπυροδοτούμενη: ενημέρωση όταν το Clk αλλάζει από 0 σε 1





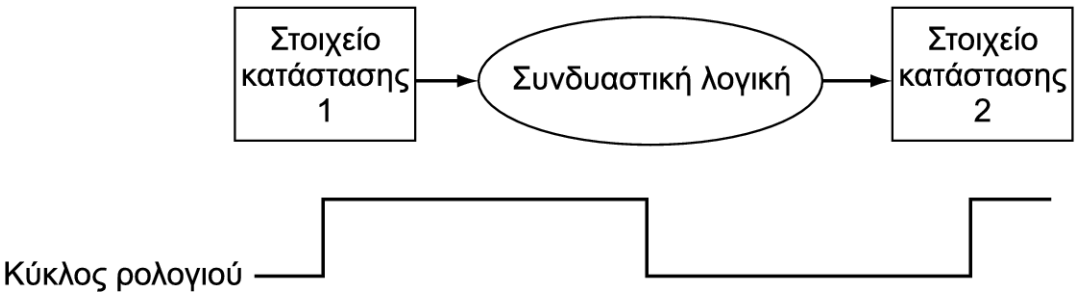
Ακολουθιακά στοιχεία

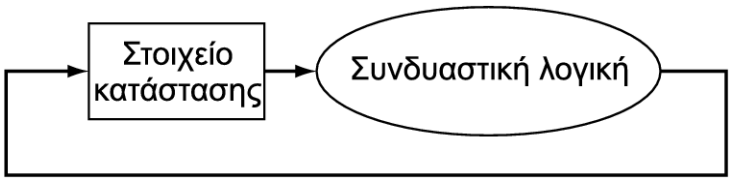
- Καταχωρητής με έλεγχο εγγραφής
  - Ενημερώνει στην ακμή του ρολογιού μόνο όταν η είσοδος ελέγχου εγγραφής είναι 1
  - Χρησιμοποιείται όταν η αποθηκευμένη τιμή απαιτείται αργότερα



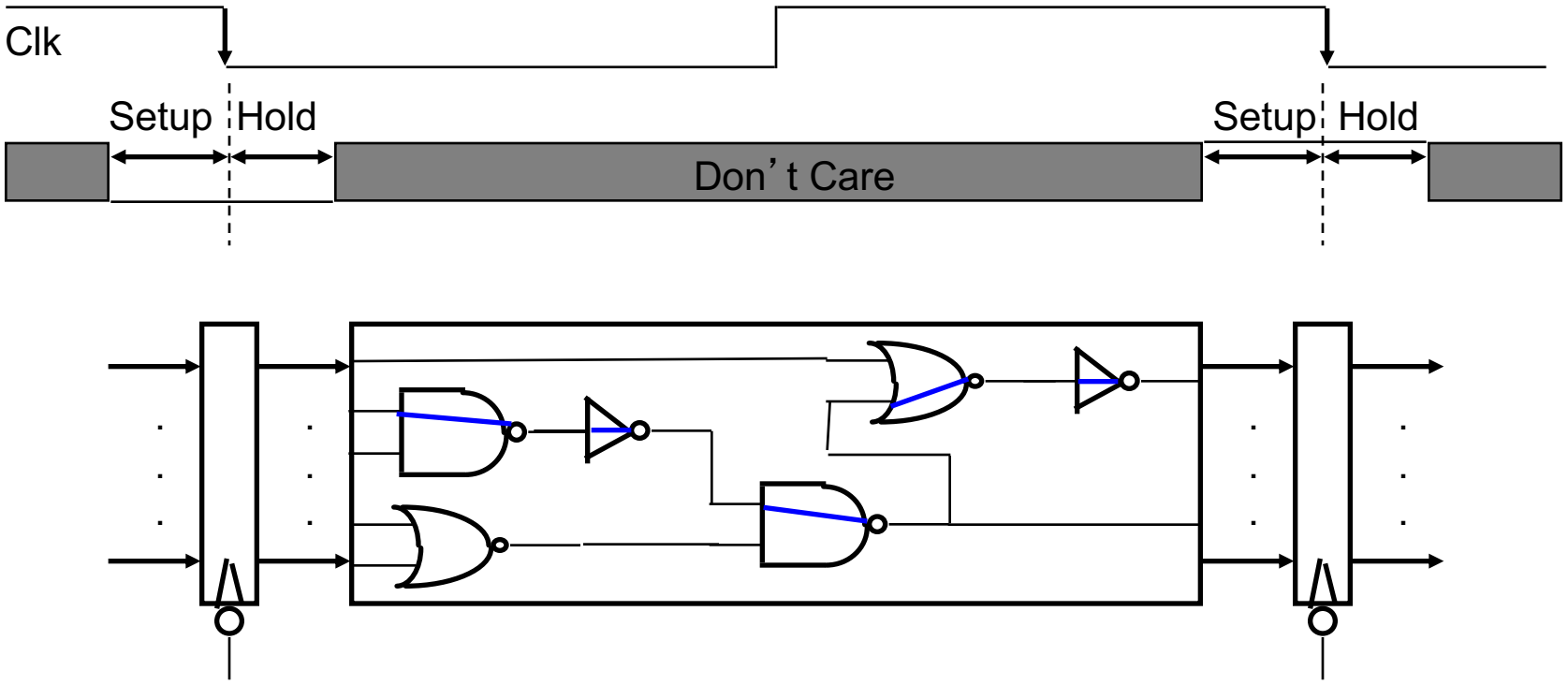
Μεθοδολογία χρονισμού

- Η συνδυαστική λογική μετασχηματίζει τα δεδομένα στη διάρκεια των κύκλων ρολογιού
  - Μεταξύ ακμών ρολογιού
  - Είσοδος από στοιχεία κατάστασης, έξοδος σε στοιχεία κατάστασης
  - Η μεγαλύτερη καθυστέρηση καθορίζει την περίοδο του ρολογιού





Clocking Methodology

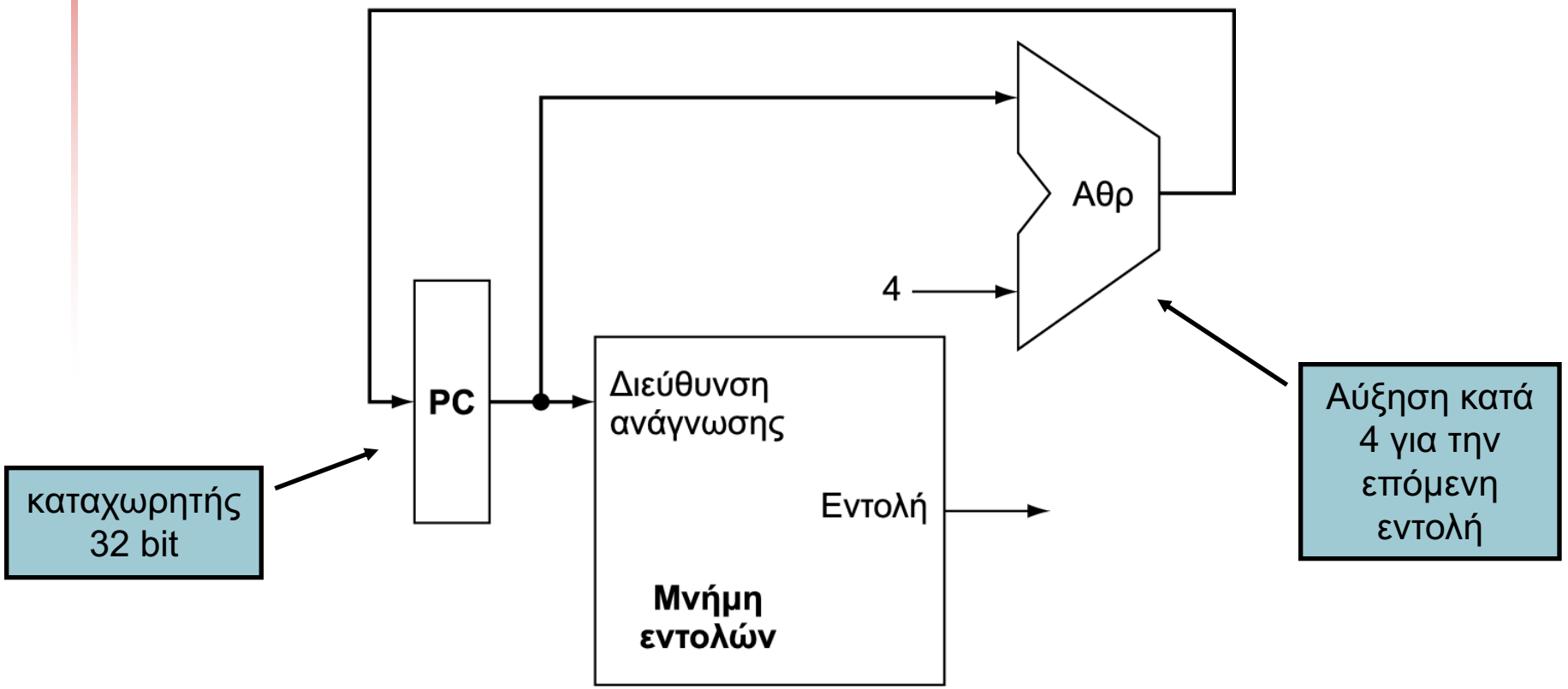


- Όλοι οι καταχωρητές έχουν το ίδιο ρολόι
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Κατασκευή διαδρομής δεδομένων

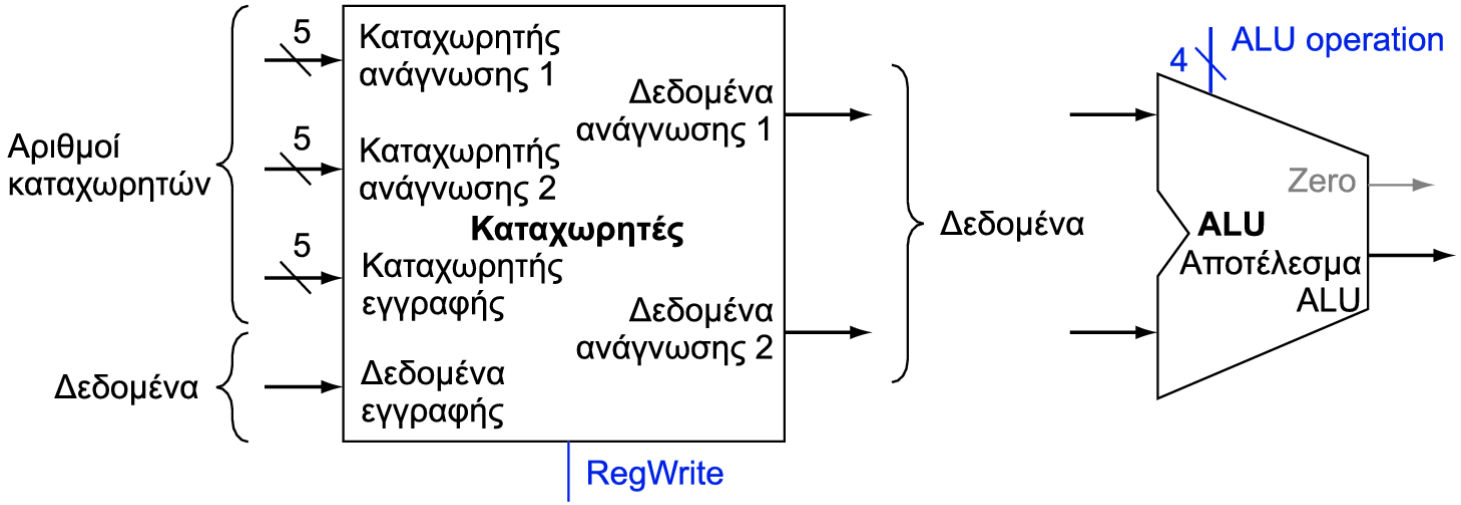
- Διαδρομή δεδομένων (datapath)
  - Στοιχεία που επεξεργάζονται δεδομένα και διευθύνσεις στη CPU
    - Καταχωρητές, ALU, πολυπλέκτες, μνήμες, …
- Θα κατασκευάσουμε μια διαδρομή δεδομένων MIPS με διαδοχικά βήματα
  - Θα κάνουμε πιο αναλυτικό το συνολικό σχέδιο

Προσκόμιση εντολής (Instruction Fetch)



Εντολές μορφής R

- Ανάγνωση δύο τελεστέων καταχωρητών
- Εκτέλεση αριθμητικής/λογικής λειτουργίας
- Εγγραφή αποτελέσματος σε καταχωρητή

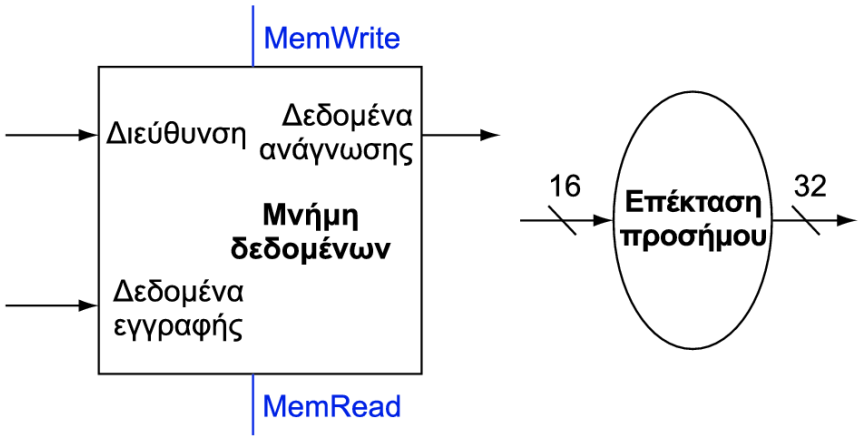


α. Καταχωρητές

β. ALU

Εντολές Load/Store

- Ανάγνωση τελεστέων καταχωρητών
- Υπολογισμός διεύθυνσης με χρήση της σχετικής απόστασης (offset) των 16 bit
  - Χρήση της ALU, αλλά με επέκταση προσήμου του offset
- Φόρτωση (Load): ανάγνωση μνήμης και ενημέρωση καταχωρητή
- Αποθήκευση (Store): εγγραφής τιμής καταχωρητή στη μνήμη



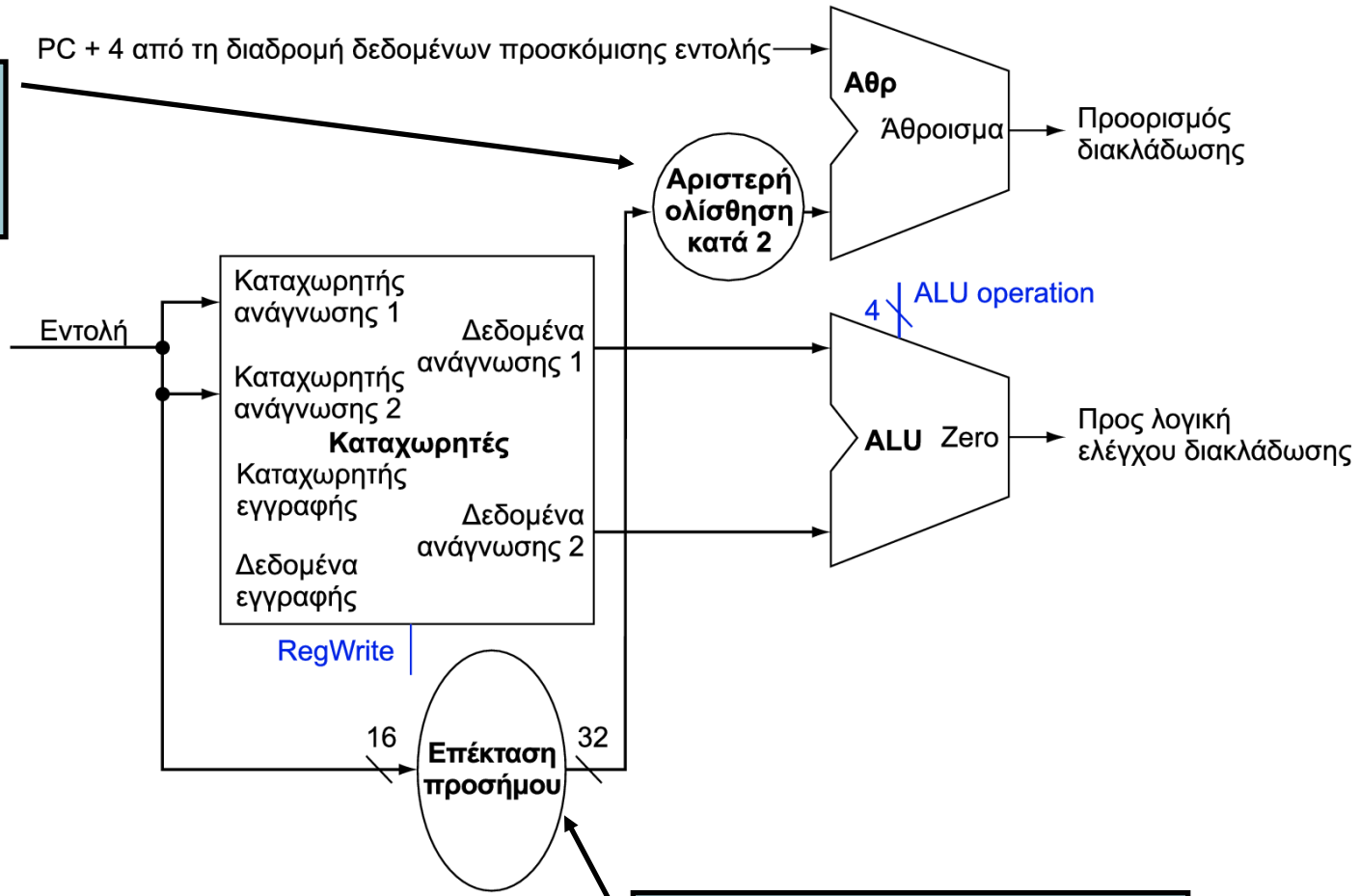
α. Μονάδα μνήμης δεδομένων β. Μονάδα επέκτασης προσήμου

Εντολές διακλάδωσης (branch)

- Ανάγνωση τελεστέων καταχωρητών
- Σύγκριση τελεστέων
  - Χρήση ALU, αφαίρεση και έλεγχος της εξόδου Zero
- Υπολογισμός διεύθυνσης προορισμού
  - Επέκταση προσήμου της μετατόπισης (displacement)
  - Αριστερή ολίσθηση κατά 2 θέσεις (μετατόπιση λέξης)
  - Πρόσθεση στο PC + 4
    - Έχει ήδη υπολογιστεί από την προσκόμιση εντολής

Εντολές διακλάδωσης

Απλώς επαναδρομολογεί τις εγγραφές

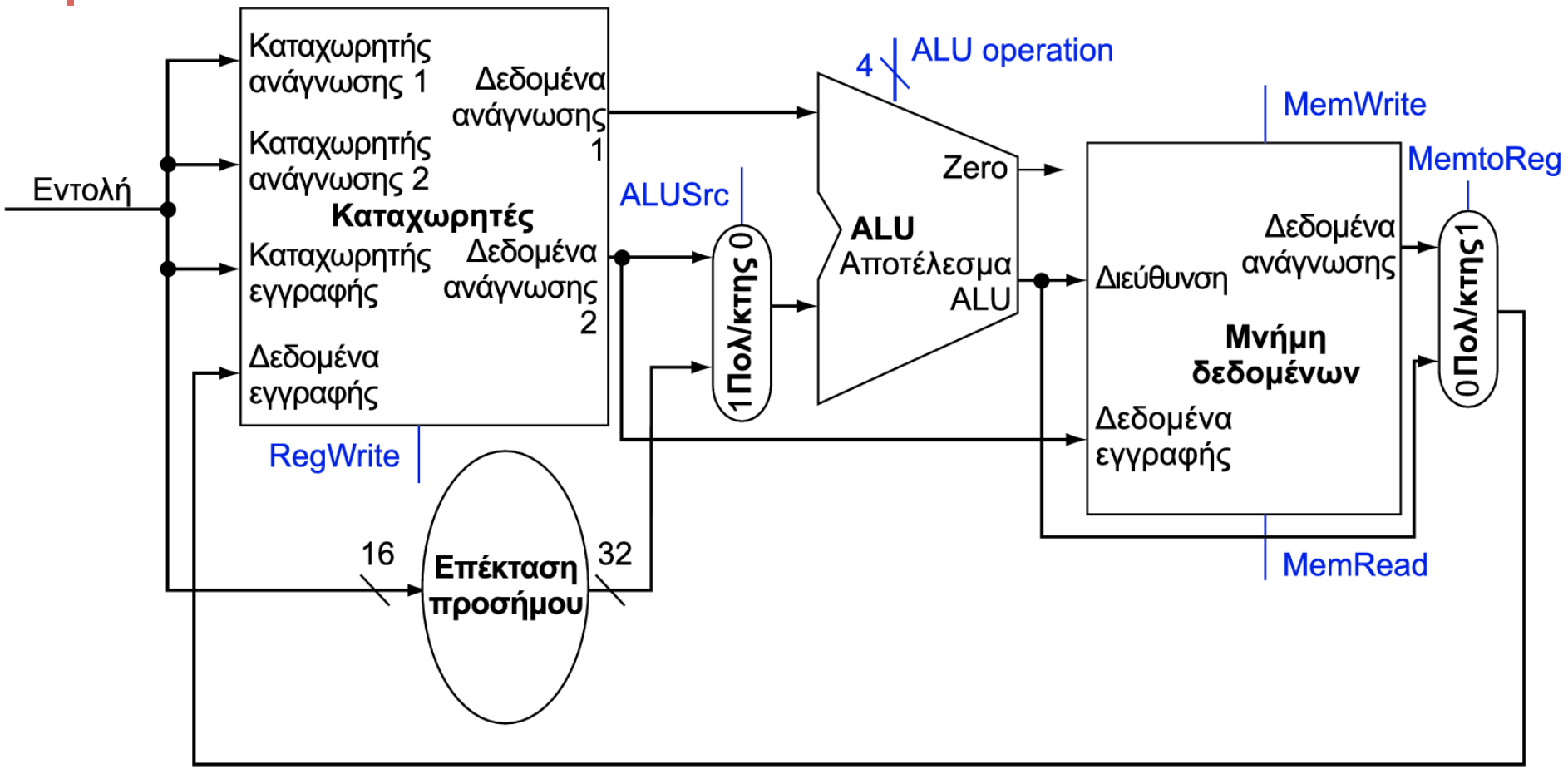


Επαναλαμβάνεται το καλώδιο του bit προσήμου

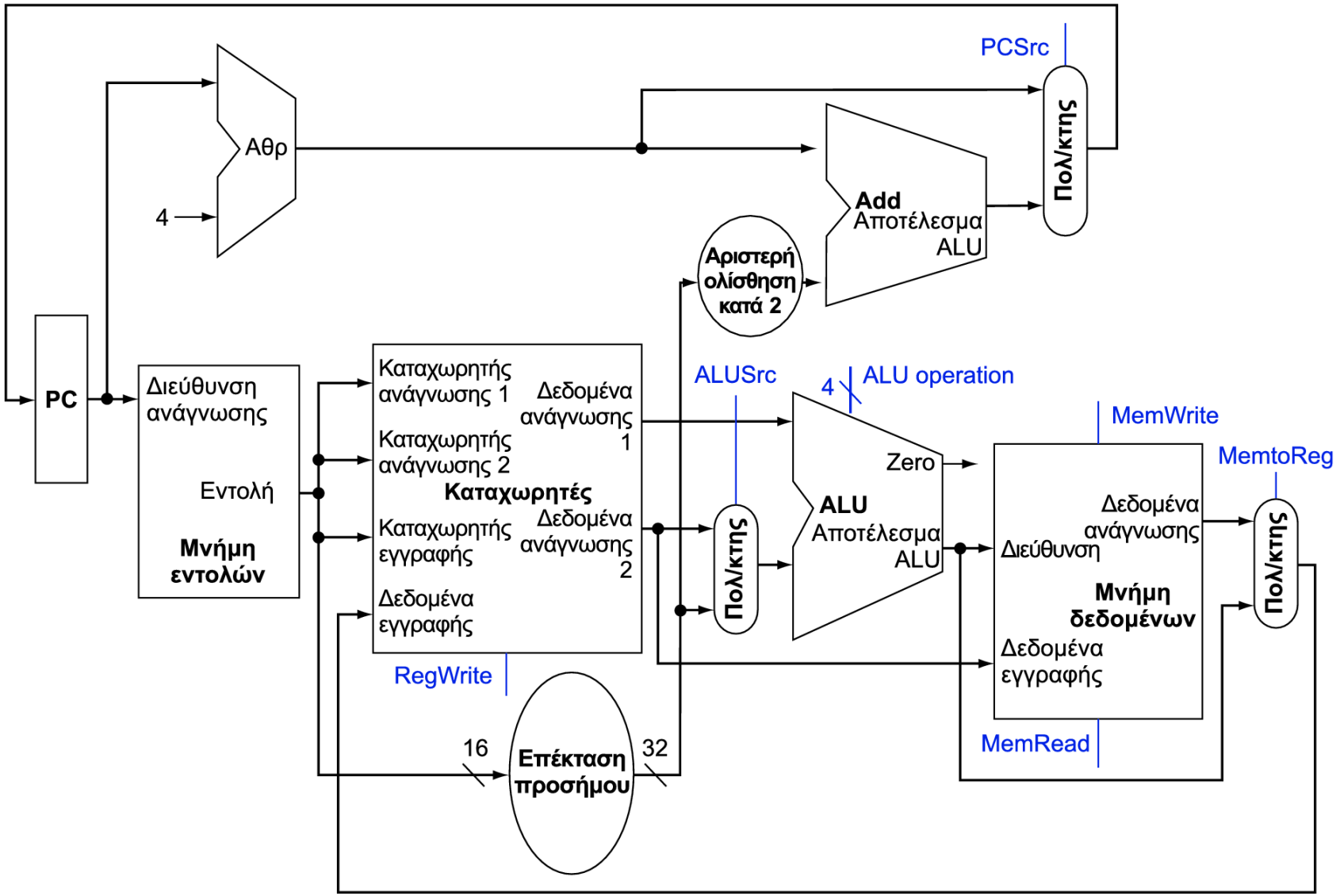
Δημιουργία των στοιχείων

- Μια πρώτη διαδρομή δεδομένων (data path) εκτελεί μία εντολή σε έναν κύκλο ρολογιού
  - Κάθε στοιχείο της διαδρομής δεδομένων κάνει μόνο μία συνάρτηση κάθε φορά
  - Έτσι, χρειαζόμαστε ξεχωριστές μνήμες εντολών και δεδομένων
- Χρήση πολυπλεκτών όταν χρησιμοποιούνται διαφορετικές προελεύσεις δεδομένων σε διαφορετικές εντολές

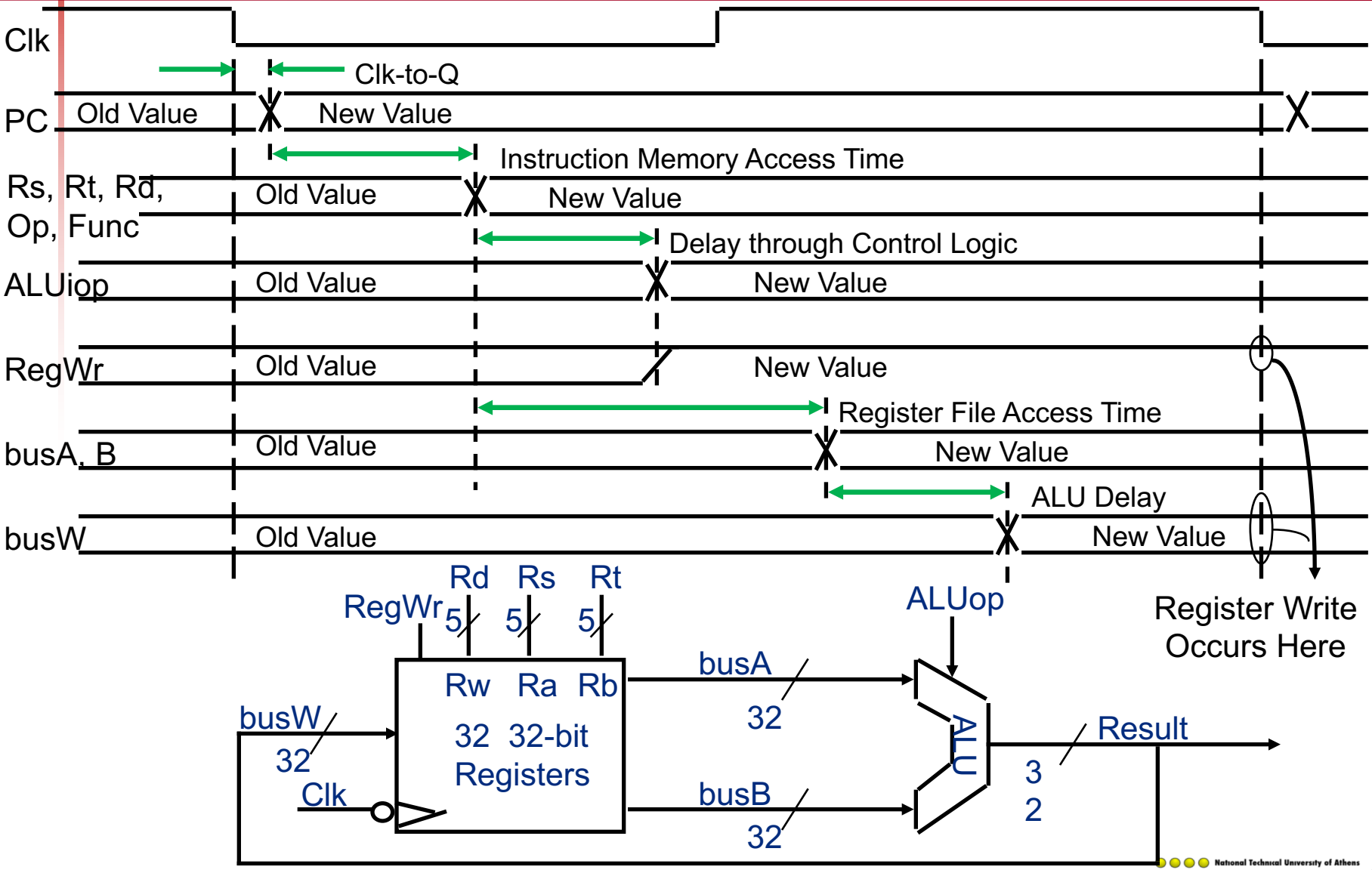
Διαδρομή δεδομένων για Τύπο R/Load/Store



Πλήρης διαδρομή δεδομένων



Register-Register Timing: A complete cycle



Έλεγχος ALU

- Η ALU χρησιμοποιείται για
  - Load/Store: λειτουργία = add
  - Branch: λειτουργία = subtract
  - Τύπου R: λειτουργία εξαρτάται από το πεδίο funct

Έλεγχος ALU	Λειτουργία
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

Έλεγχος ALU

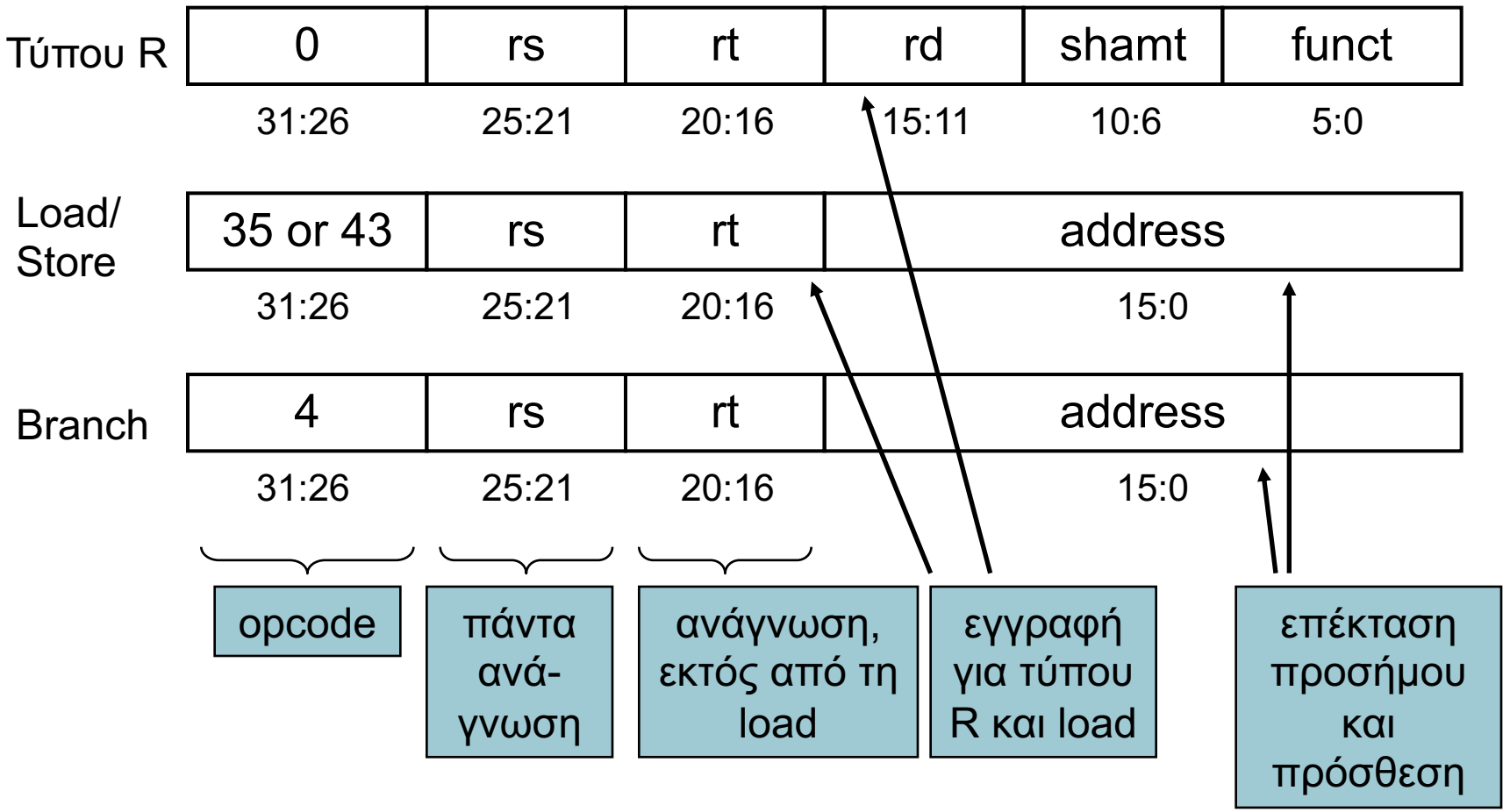
- Υποθέτουμε ότι ένα πεδίο 2 bit ALUOp εξάγεται από το opcode
  - Συνδυαστική λογική εξάγει τον έλεγχο της ALU

opcode	ALUOp	Λειτουργία	funct	ALU function	ALUcontrol
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
ori	11	OR immediate	XXXXXX	or	0001
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

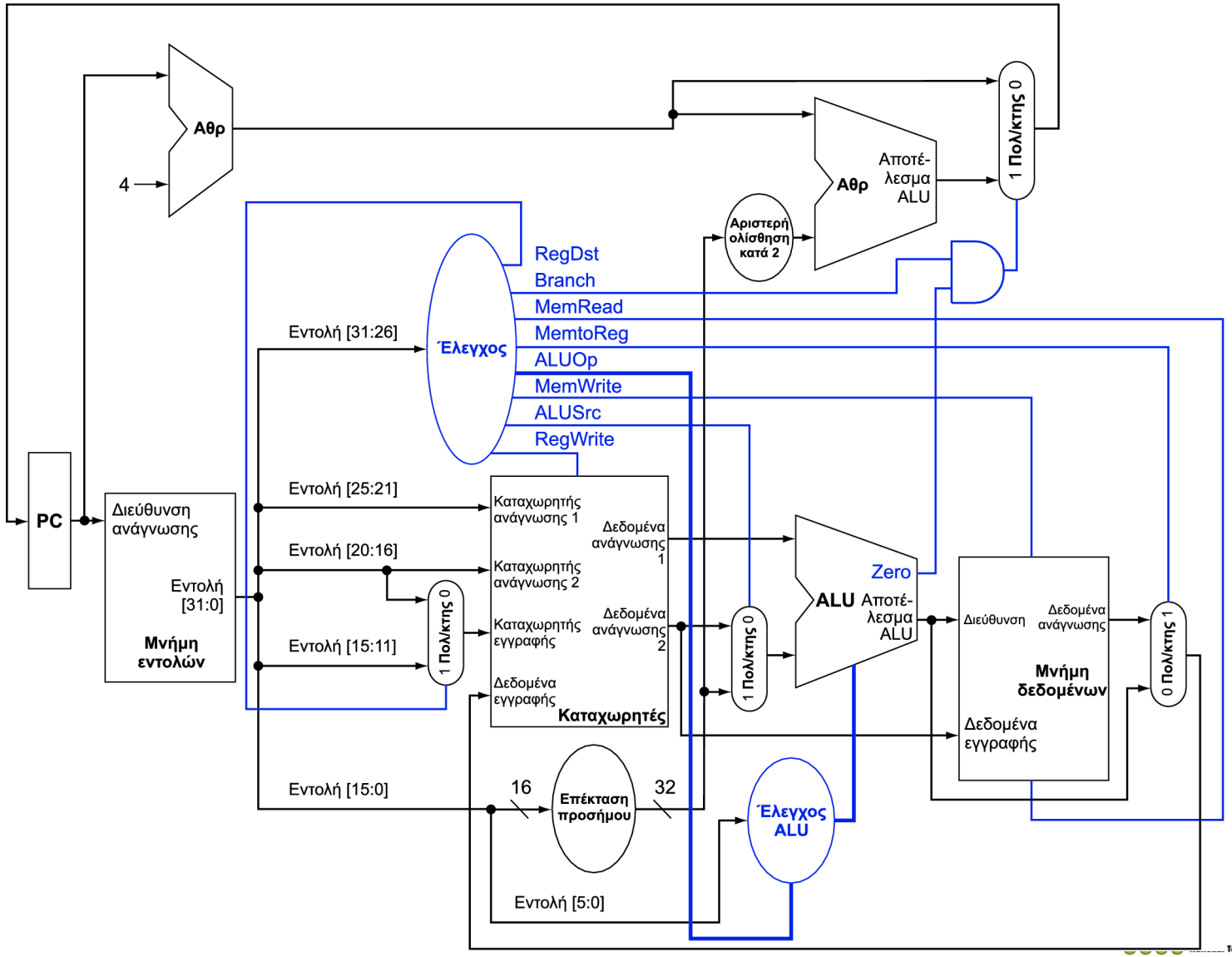
Δεν υπάρχει στο παράδειγμα του βιβλίου

Η κύρια μονάδα ελέγχου

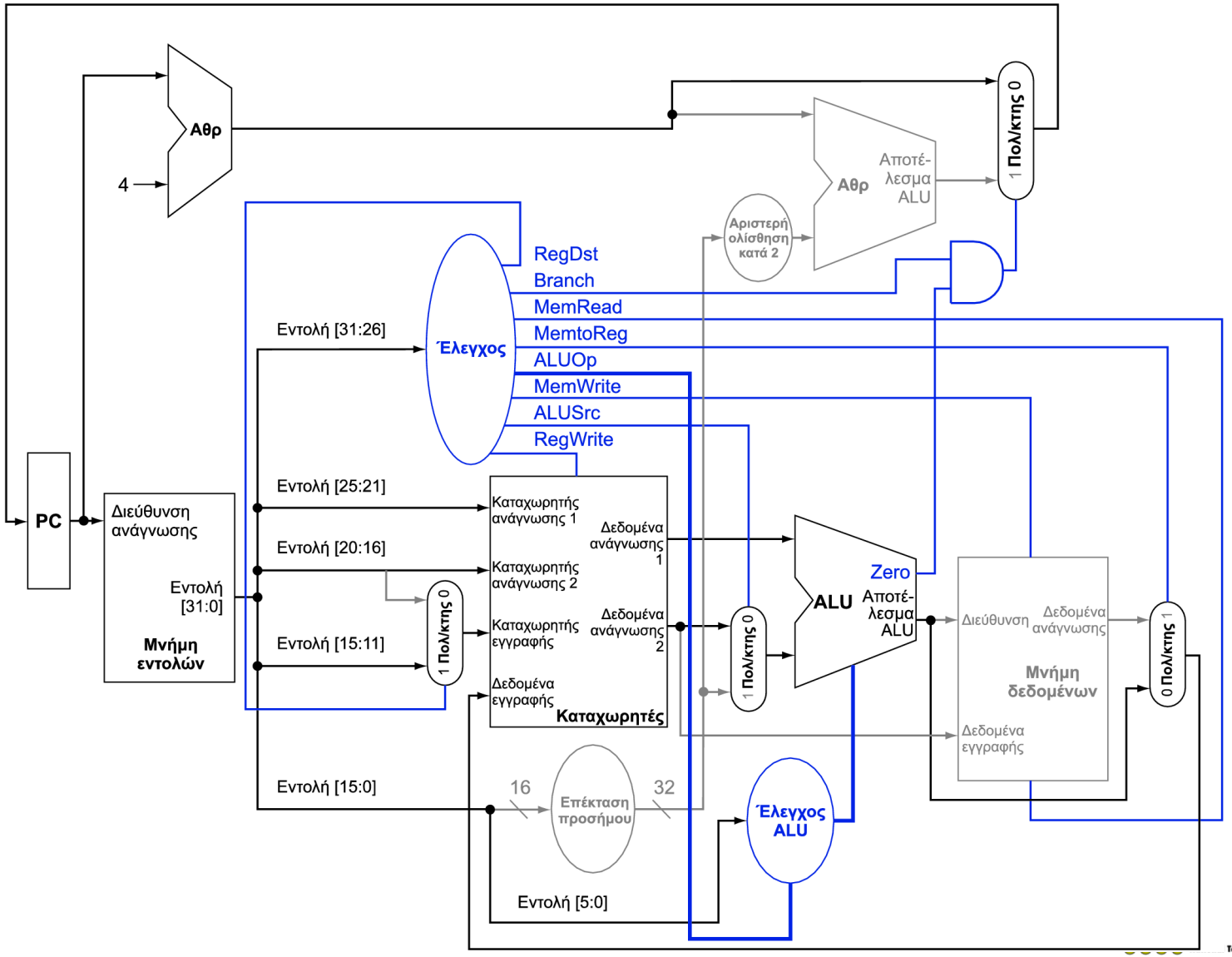
- Σήματα ελέγχου που εξάγονται από εντολή



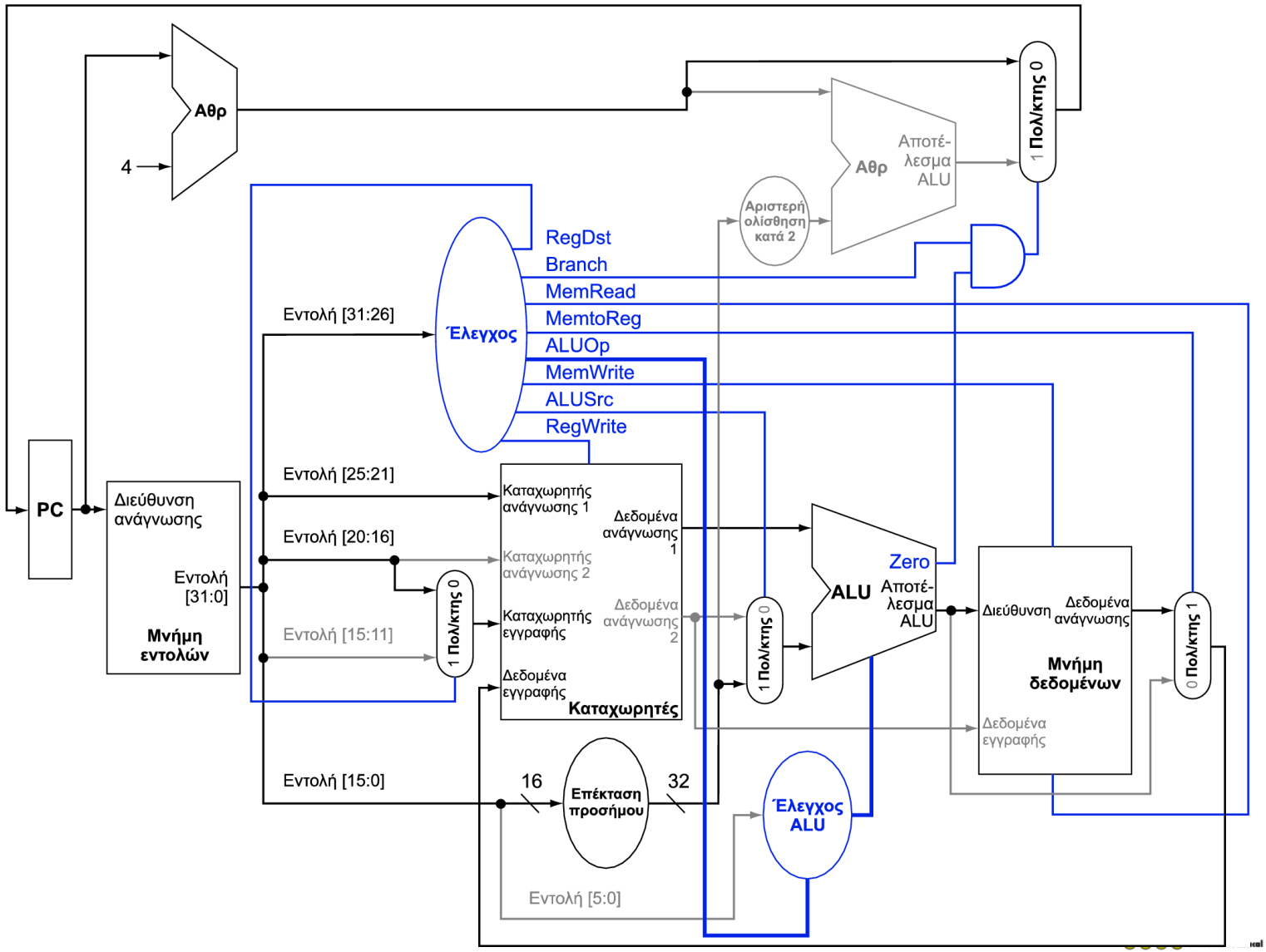
Διαδρομή δεδομένων και έλεγχος



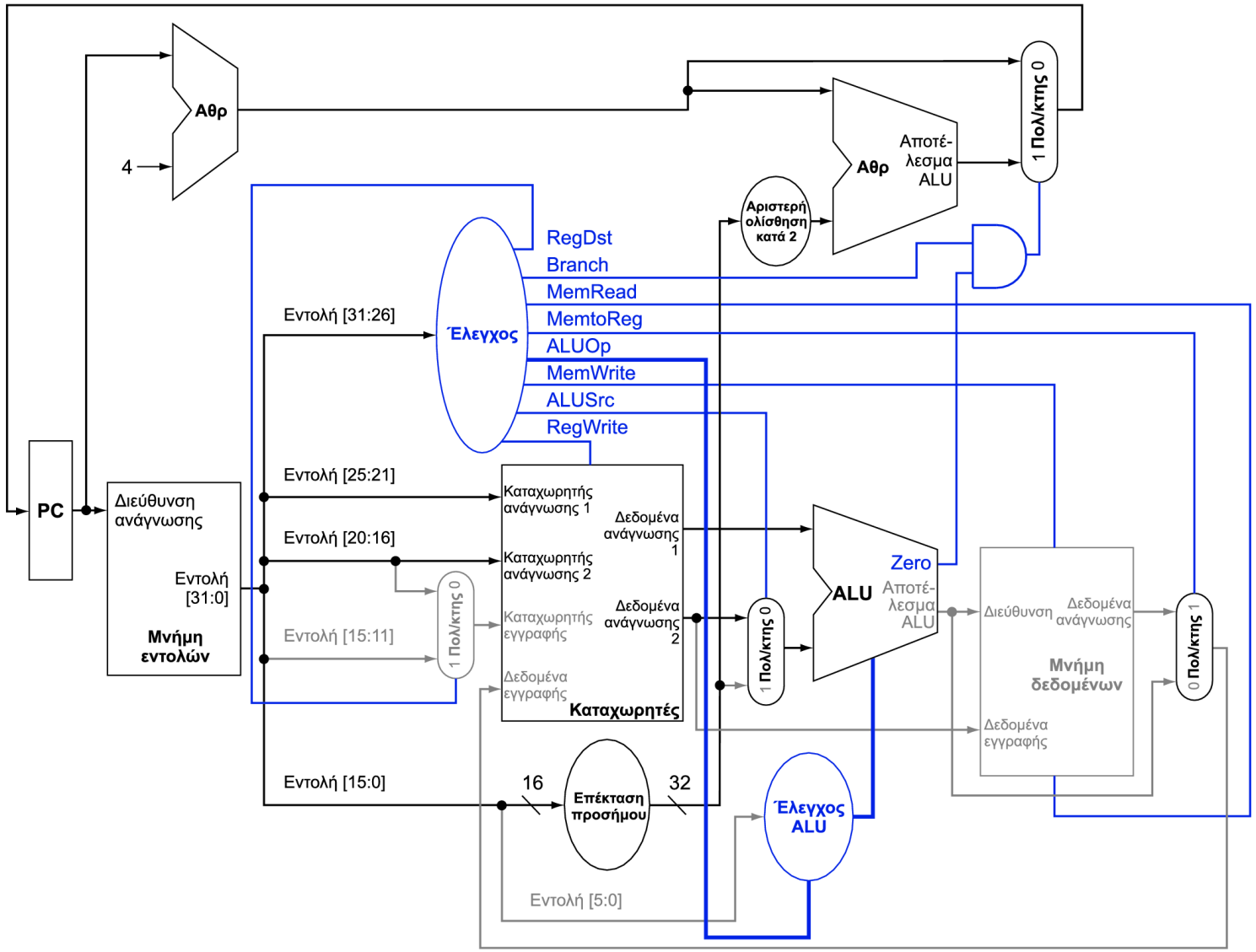
Εντολή τύπου R



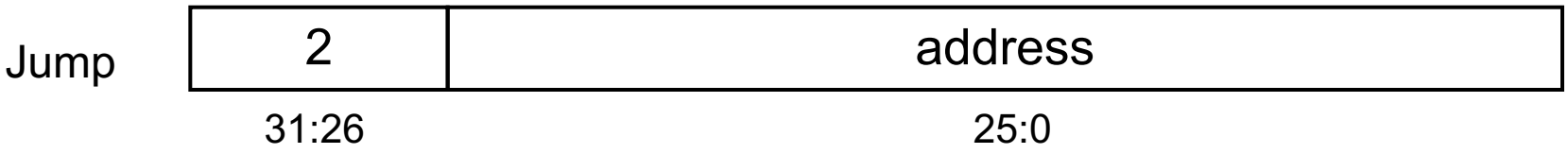
Εντολή Load


Αθρ
4
Αθρ
Αποτέ-
λεσμα
ALU
1 Πολ/κτης 0
Αριστερή
ολίσθηση
κατά 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Έλεγχος
Εντολή [31:26]
Εντολή [25:21]
Εντολή [20:16]
Εντολή [15:11]
Εντολή [15:0]
PC
Διεύθυνση
ανάγνωσης
Εντολή
[31:0]
Μνήμη
εντολών
1 Πολ/κτης 0
Καταχωρητής
ανάγνωσης 1
Καταχωρητής
ανάγνωσης 2
Καταχωρητής
εγγραφής
Δεδομένα
εγγραφής
Δεδομένα
ανάγνωσης
1
Δεδομένα
ανάγνωσης
2
Καταχωρητές
1 Πολ/κτης 0
ALU
Zero
Αποτέ-
λεσμα
ALU
Διεύθυνση
Δεδομένα
ανάγνωσης
Μνήμη
δεδομένων
Δεδομένα
εγγραφής
0 Πολ/κτης 1
16
Επέκταση
προσήμου
32
Έλεγχος
ALU
Εντολή [5:0]

Εντολή Branch-on-Equal

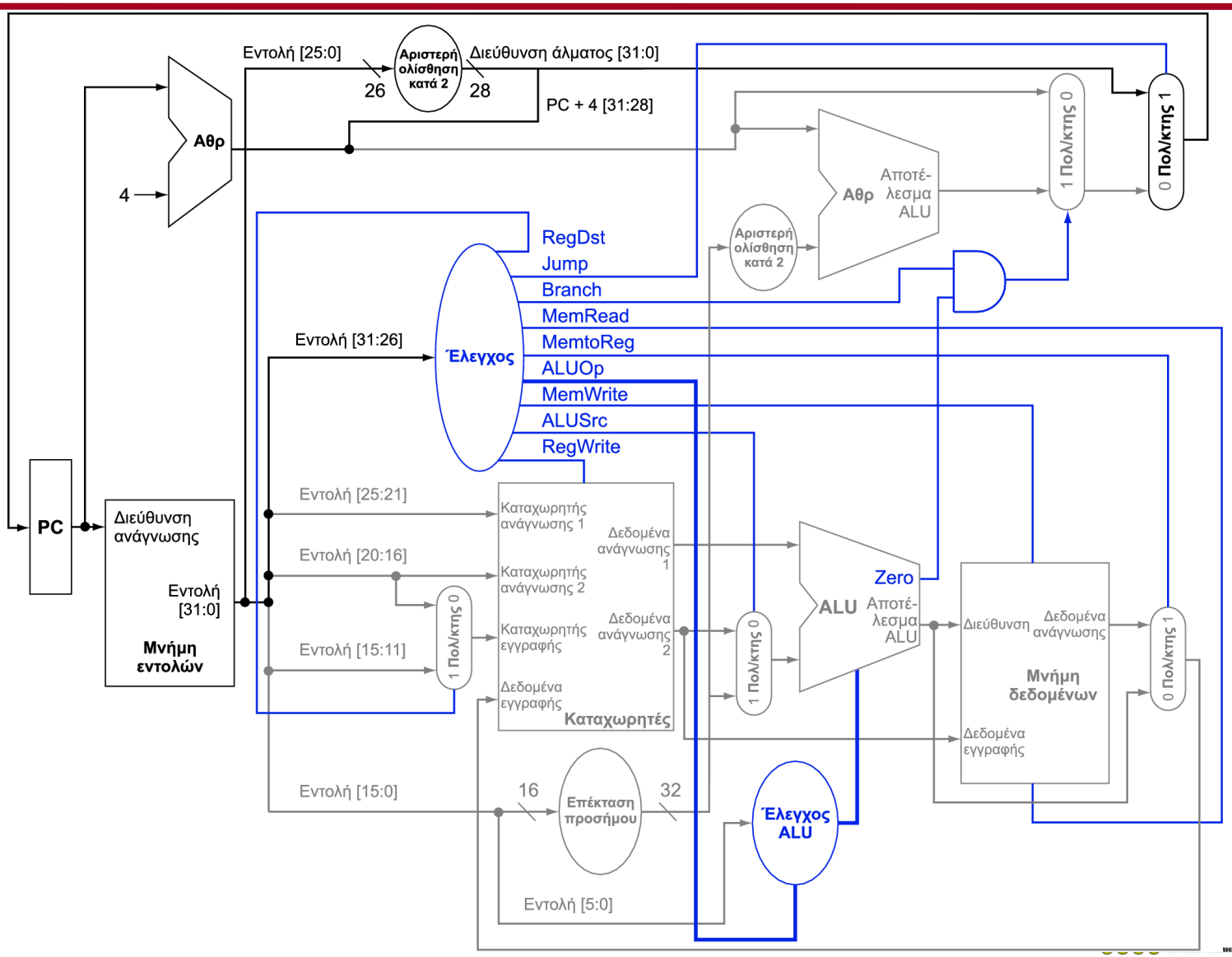


Υλοποίηση αλμάτων



- Η Jump χρησιμοποιεί διεύθυνση λέξης
- Ενημέρωση του PC με συνένωση των
  - Υψηλότερων 4 bit του παλιού PC
  - Διεύθυνσης άλματος των 26 bit
  - 00
- Χρειάζεται ένα επιπλέον σήμα ελέγχου από την αποκωδικοποίηση του opcode

Διαδρομή δεδ. με προσθήκη αλμάτων



Σύνοψη Σημάτων Ελέγχου

Σήματα που δεν εμφανίζονται είναι 0 («απενεργοποιημένα»)

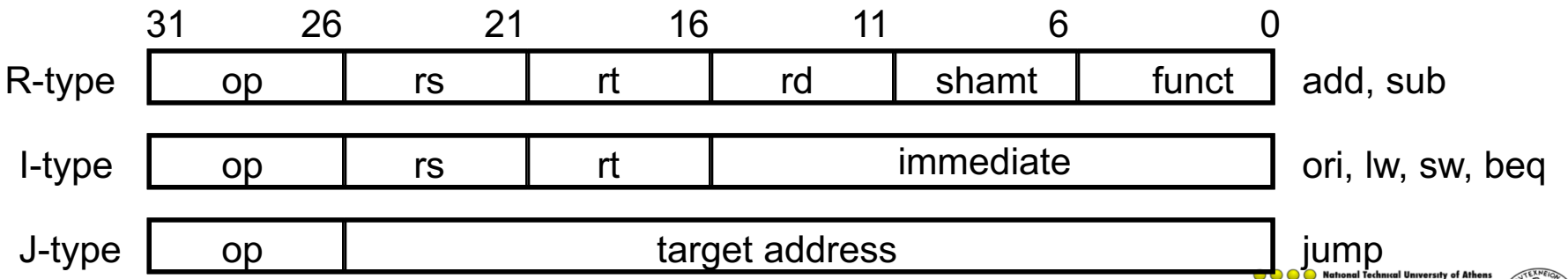
inst	Register Transfer	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “add”, RegDst = rd, RegWr, nPC_sel = “+4”	
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
	ALUsrc = RegB, ALUctr = “sub”, RegDst = rd, RegWr, nPC_sel = “+4”	
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
	ALUsrc = Im, Extop = “Z”, ALUctr = “or”, RegDst = rt, RegWr nPC_sel=“+4”	
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemtoReg, RegDst = rt RegWr, nPC_sel = “+4”	
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUctr = “add”, MemWr, nPC_sel = “+4”	
BEQ	if (R[rs] == R[rt]) {PC <– PC + sign_ext(Imm16)] || 00} else PC <– PC + 4	
	nPC_sel = “Br”, ALUctr = “sub”	

Σύνοψη Σημάτων Ελέγχου

Δεν υπάρχει στο παράδειγμα του βιβλίου

See Appendix A

	func	10 0000	10 0010	We Don't Care :-)				
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		add	sub	ori	lw	sw	beq	jump
RegDst		1	1	0	0	x	x	x
ALUSrc		0	0	1	1	1	0	x
MemtoReg		0	0	0	1	x	x	x
RegWrite		1	1	1	1	0	0	0
MemWrite		0	0	0	0	1	0	0
nPCsel		0	0	0	0	0	1	0
Jump		0	0	0	0	0	0	1
ExtOp		x	x	0	1	1	x	x
ALUctr<1:0>		R	R	Or	Mem	Mem	Br	xxx



Υλοποίηση λογικής ελέγχου σε HDL

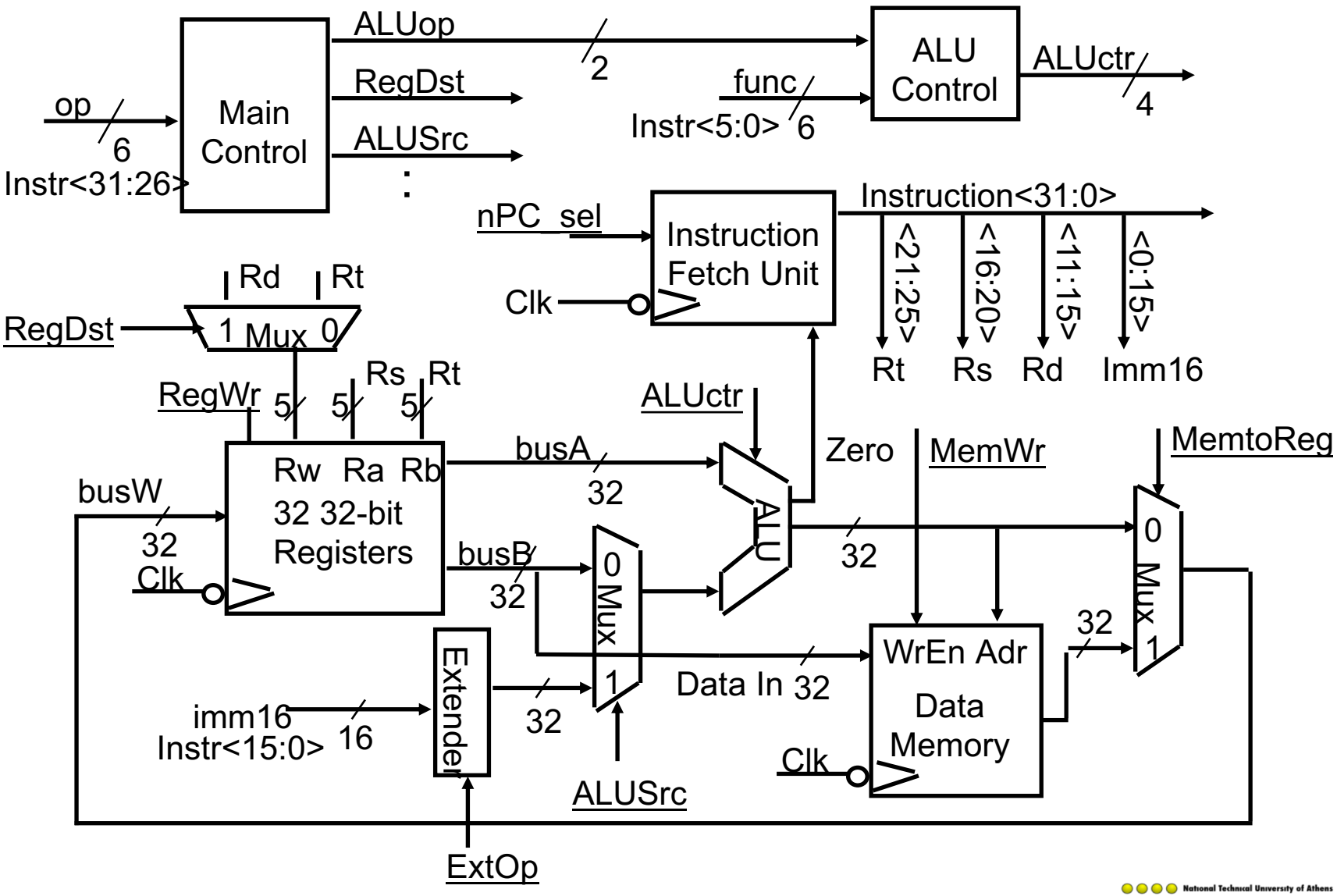
- nPC_sel <= if (OP == BEQ) then “Br” else “+4”
- ALUsrc <= if (OP == “Rtype”) then “regB” else “immed”
- ALUctr <= if (OP == “Rtype”) then funct
  elseif (OP == ORi) then “OR”
  elseif (OP == BEQ) then “sub”
  else “add”
- ExtOp <= _____
- MemWr <= _____
- MemtoReg<= _____
- RegWr: <= _____
- RegDst: <= _____

This is also 2-level decoding: if branch, check additional signals to decide

Υλοποίηση λογικής ελέγχου σε HDL

- nPC_sel <= if (OP == BEQ) then “Br” else “+4”
- ALUsrc <= if (OP == “Rtype”) then “regB” else “immed”
- ALUctr <= if (OP == “Rtype”) then funct
  elseif (OP == ORi) then “OR”
  elseif (OP == BEQ) then “sub”
  else “add”
- ExtOp <= if (OP == ORi) then “zero” else “sign”
- MemWr <= (OP == Store)
- MemtoReg<= (OP == Load)
- RegWr: <= if ((OP == Store) || (OP == BEQ)) then 0 else 1
- RegDst: <= if ((OP == Load) || (OP == ORi)) then 0 else 1

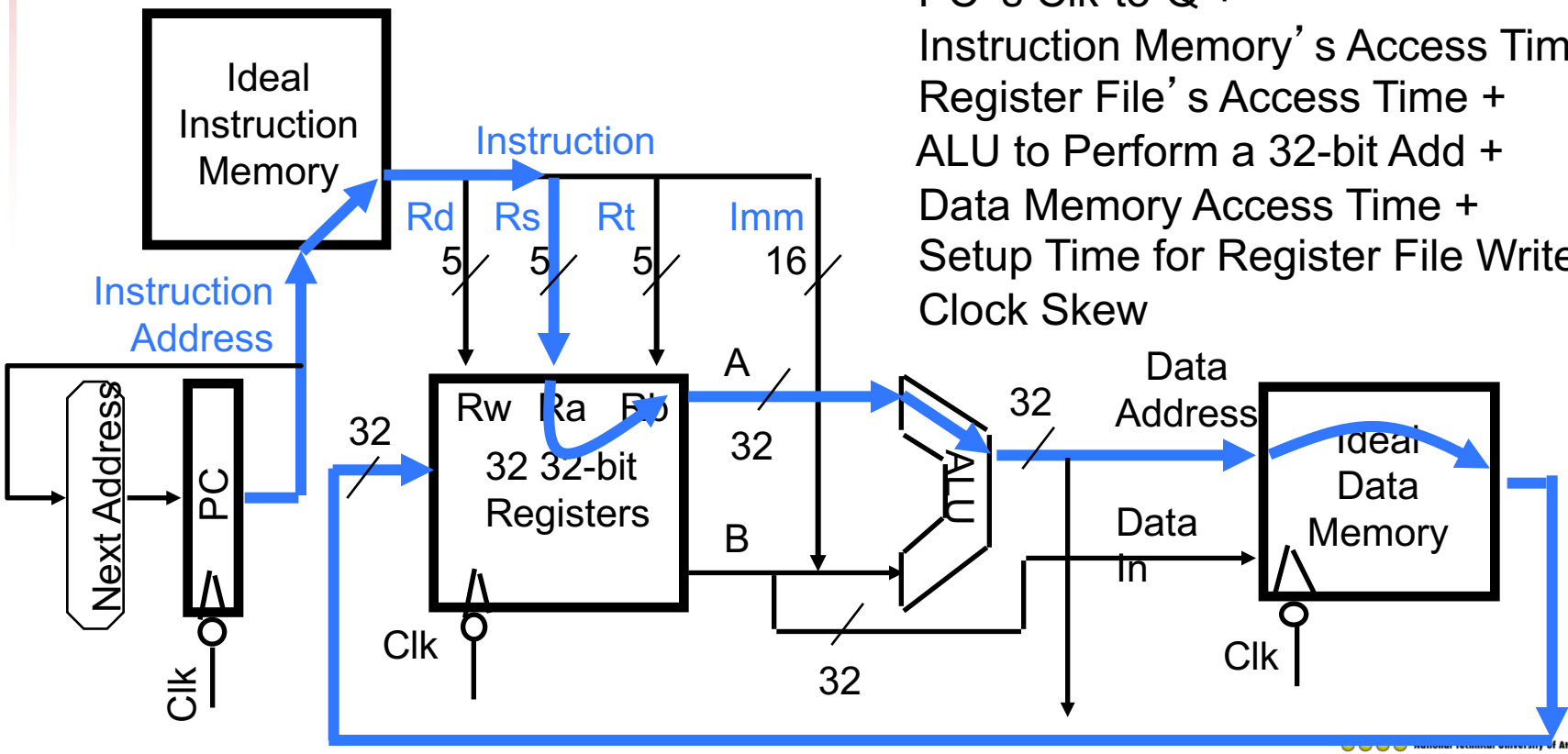
Putting it All Together: A Single Cycle Processor



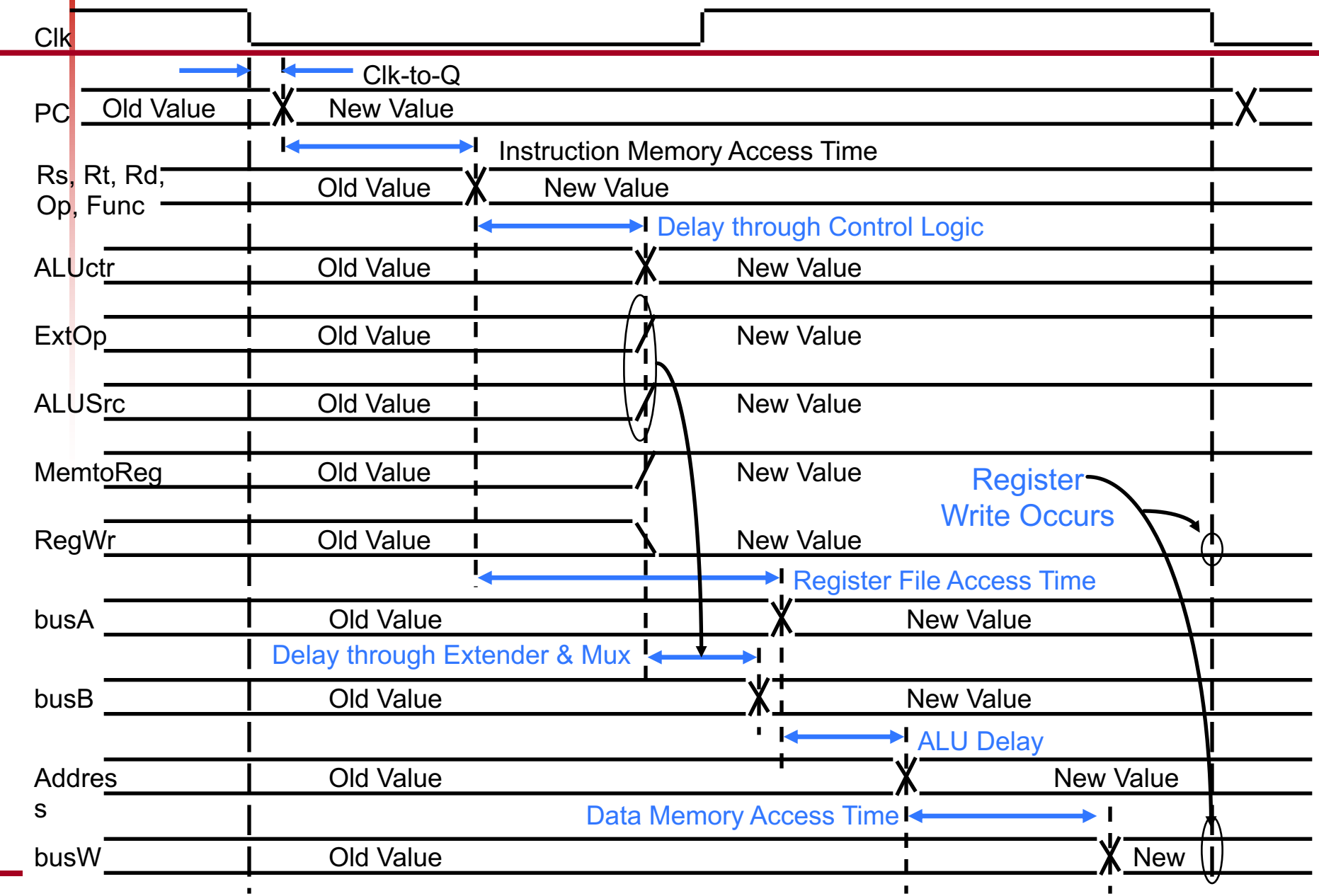
An Abstract View of the Critical Path (Load)

- Register file & ideal memory: το ρολόι επηρεάζει μόνο την εγγραφή!
- Η ανάγνωση γίνεται συνδυαστικά: Address valid => Output valid μετά από τον χρόνο πρόσβασης (access time)

Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew



Worst Case Timing (Load)



Ζητήματα απόδοσης

- CPI = 1 αλλά ένας μεγάλος κύκλος
- Η μεγαλύτερη καθυστέρηση καθορίζει την περίοδο ρολογιού
  - Κρίσιμη διαδρομή (critical path): εντολή load
  - Μνήμη εντολών → αρχείο καταχωρητών → ALU → μνήμη δεδομένων → αρχείο καταχωρητών
- Δεν είναι εφικτή διαφορετική περίοδος για διαφορετικές εντολές
- Παραβιάζει τη σχεδιαστική αρχή: «Κάνε τη συνηθισμένη περίπτωση γρήγορη»
- Θα βελτιώσουμε την απόδοση με τη διοχέτευση (pipelining)