Ο **επεξεργαστής**: Η **δίοδος δεδομένων** (**datapath**) και η **μονάδα ελέγχου** (**control**)

Μέρος Α: single cycle υλοποίηση

Νεκ. Κοζύρης
nkoziris@cslab.ece.ntua.gr

-Figures are from COD2e/Patterson-Hennessy

-Some figures and slides are from CS61C Berkeley course

-Hand drawn figures from J. Wawrzynek notes on "UCB CS61c: State Elements: Circuits That Remember"

Ο **επεξεργαστής**: Η **δίοδος δεδομένων** (**datapath**) και η **μονάδα ελέγχου** (**control**)

Σχεδίαση datapath

4 κατηγορίες εντολών:

Αριθμητικές-λογικές εντολές (add, sub, slt κλπ) –R Type

Εντολές αναφοράς στη μνήμη (lw, sw) – I Type

Εντολές διακλάδωσης (branch beq, bne) – I Type

Εντολές άλματος (jump j) – J Type

…απλότητα στη σχεδίαση του ISA……

2 πρώτα βήματα κοινά σε κάθε εντολή (IF+ID):

– Στείλε το PC στη μνήμη (instruction memory) και φέρε (fetch) την εντολή (IF)

– Αποκωδικοποίησε την εντολή και διάβασε έναν ή δύο καταχωρητές-ορίσματα (ID-instruction decode+register file read)

Στη συνέχεια (EX):

–Οι arithmetic-logical χρησιμοποιούν την ALU για εκτέλεση της λειτουργίας του με βάση opcode και funct

–Οι memory-reference χρησιμοποιούν την ALU για υπολογισμό της τελικής δνσης του ορίσματος.

–Οι branch χρησιμοποιούν την ALU για σύγκριση

Κατόπιν (MEM-WB):

–Οι arithmetic-logical γράφουν το αποτέλεσμα της ALU πίσω σε ένα καταχωρητή του Register File

–Οι memory-reference διαβάζουν από τη μνήμη και γράφουν πίσω σε ένα καταχωρητή του Register File ή αποθηκεύουν στη μνήμη

–Οι branch αλλάζουν το περιεχόμενο του PC

IF-ID-EX-MEM-WB

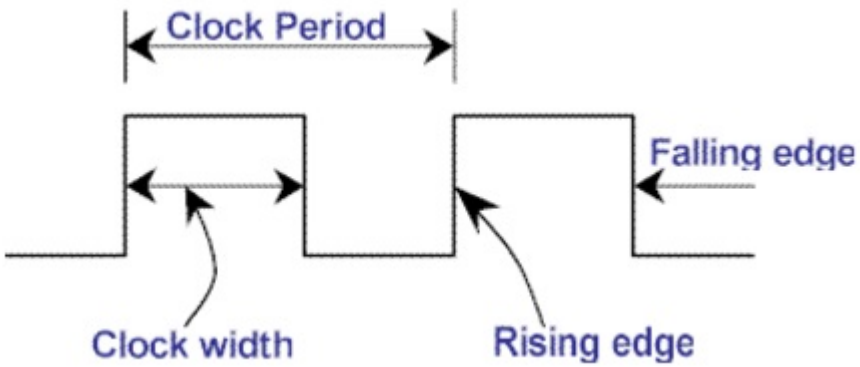
Επανάληψη λογικής σχεδίασης

- 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

Type of Circuits

- *Synchronous Digital Systems (SDS)* consist of two basic types of circuits:
  - Combinational Logic (CL) circuits
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that “remember” or store information
    - aka “State Elements”
    - E.g., memories and registers (Registers)

Edge triggered clocking

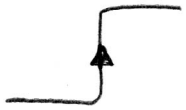


Edge triggered clocking methodology: All state changes occur on a clock edge (i.e. positive/rising clock edge)

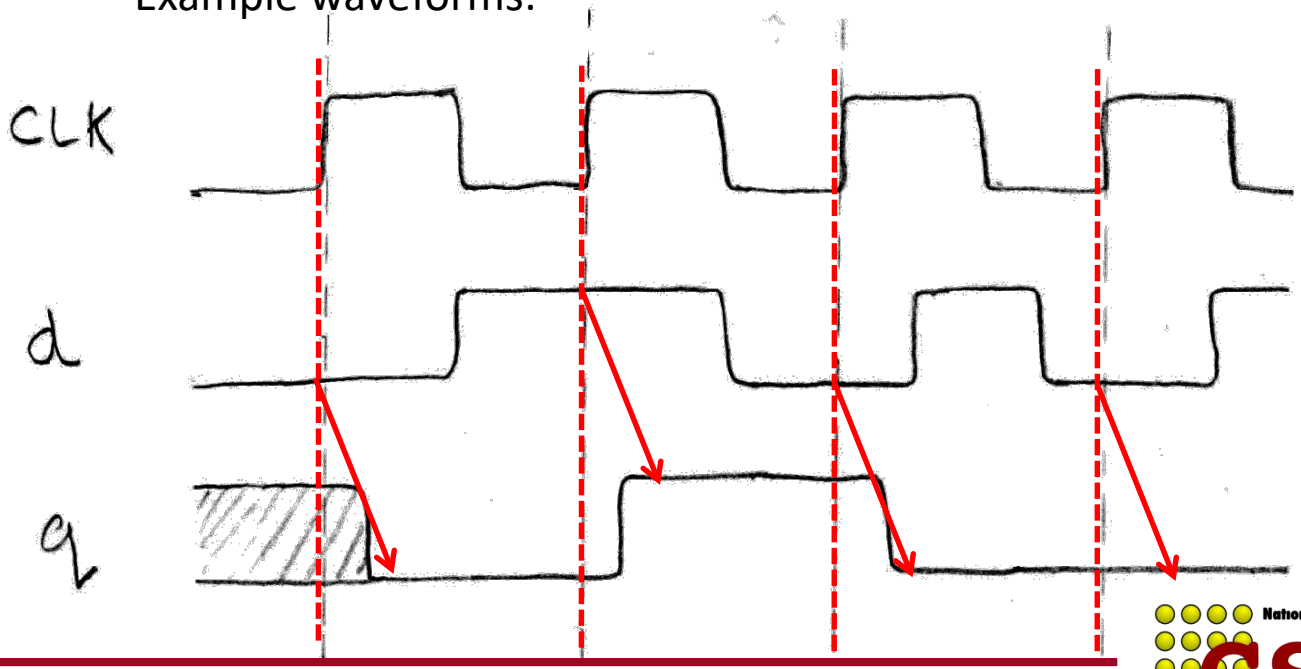
Clock edge acts like a **sampling** signal

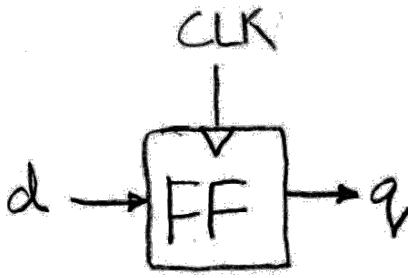
Flip-Flop Operation

- Edge-triggered d-type flip-flop
  - This one is “positive edge-triggered”



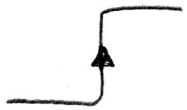
- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”
- Example waveforms:



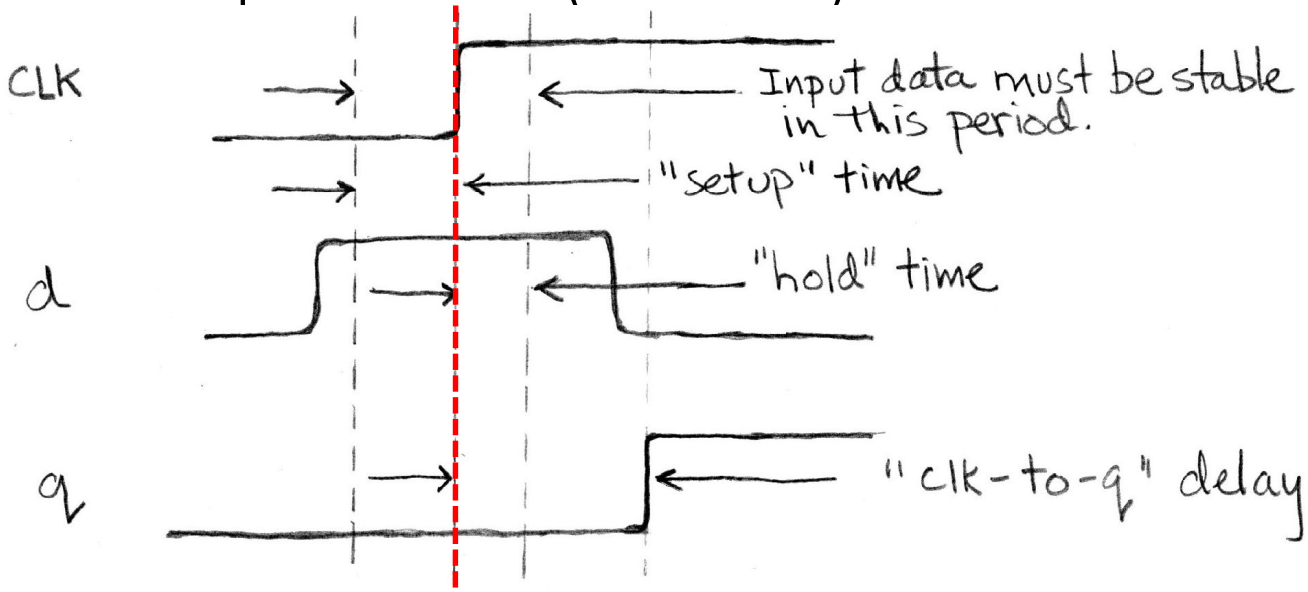


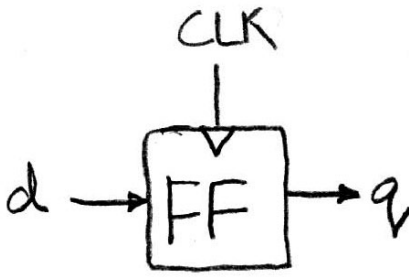
Flip-Flop Timing

- Edge-triggered d-type flip-flop
  - This one is “positive edge-triggered”



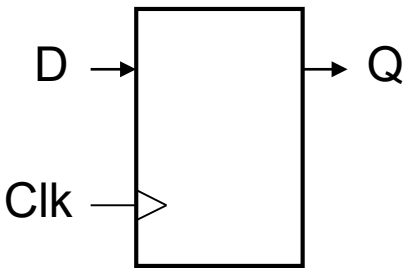
- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”
- Example waveforms (more detail):

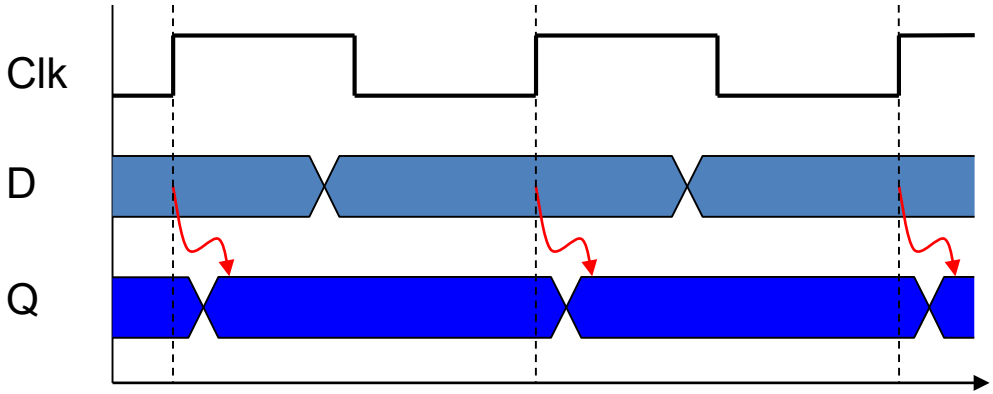




Ακολουθιακά στοιχεία

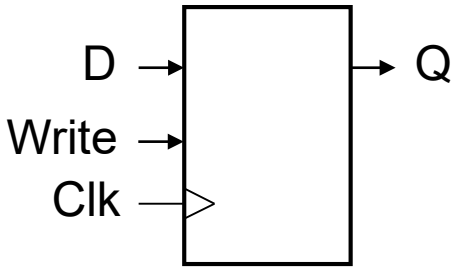
- Καταχωρητής: αποθηκεύει δεδομένα σε ένα κύκλωμα
  - Χρησιμοποιεί σήμα ρολογιού για να καθορίσει πότε ενημερώνεται η αποθηκευμένη τιμή
  - Ακμοπυροδοτούμενη: ενημέρωση όταν το Clk αλλάζει από 0 σε 1

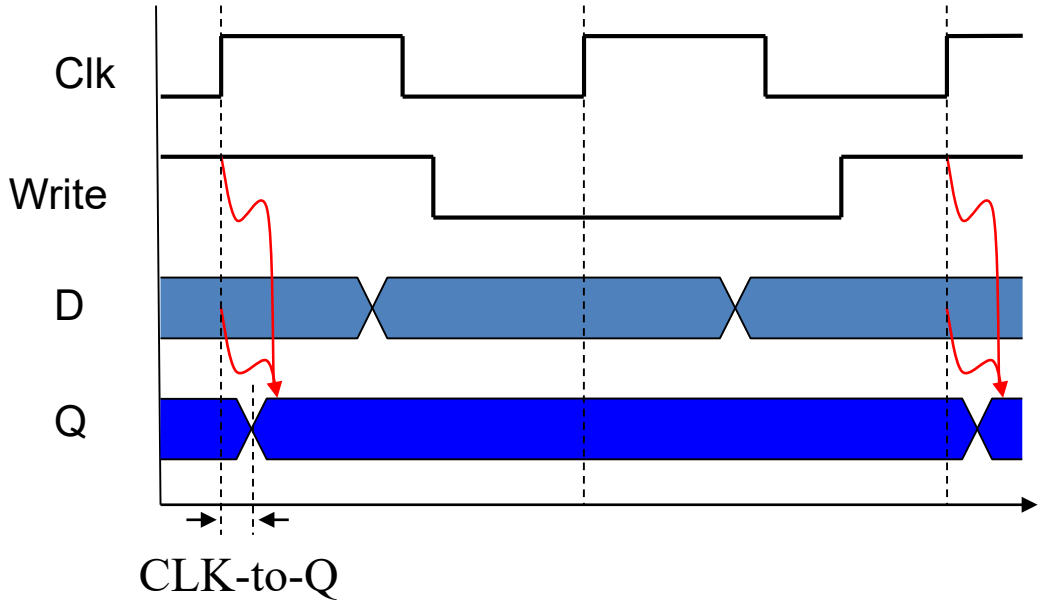




Ακολουθιακά στοιχεία

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





Camera Analogy Timing Terms

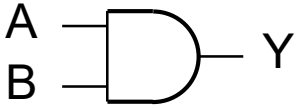
- Want to take a portrait – timing right before and after taking picture
- *Set up time* – don't move since about to take picture (open camera shutter)
- *Hold time* – need to hold still after shutter opens until camera shutter closes
- *Time click to data* – time from open shutter until can see image on output (viewscreen)

Hardware Timing Terms

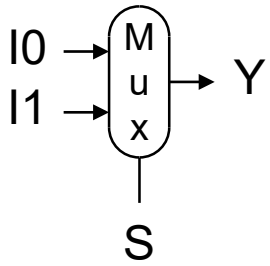
- Setup Time: when the input must be stable *before* the edge of the CLK
- Hold Time: when the input must be stable *after* the edge of the CLK
- "CLK-to-Q" Delay: how long it takes the output to change, measured from the edge of the CLK

Συνδυαστικά στοιχεία

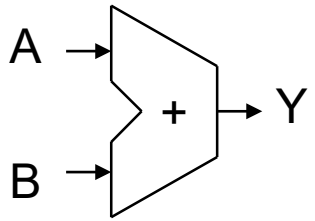
- Πύλη AND
  - Y = A & B



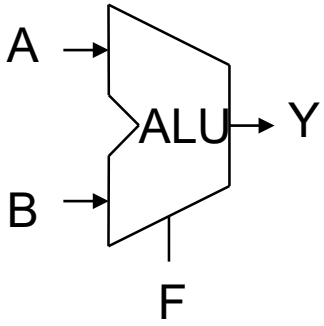
- Πολυπλέκτης
  - Y = S ? I1 : I0



- Αθροιστής
  - Y = A + B

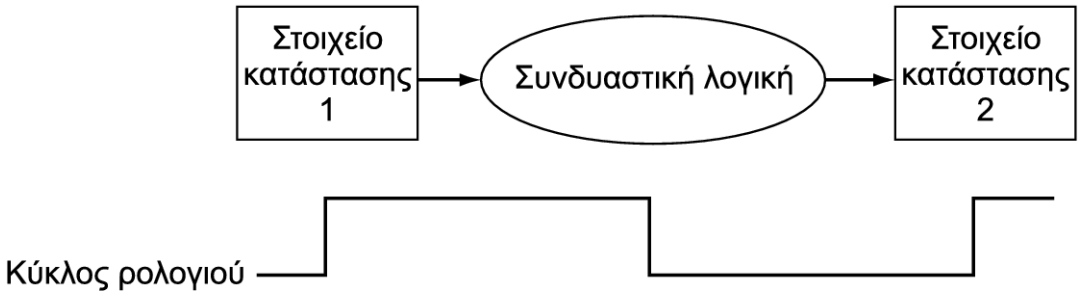


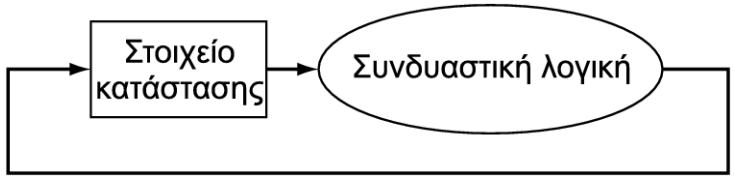
- Αριθμητική/Λογική Μονάδα
  - Y = F(A, B)



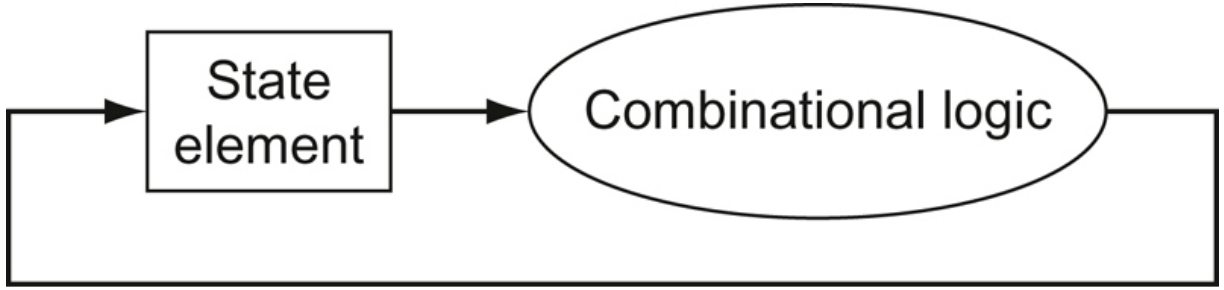
Μεθοδολογία χρονισμού

- 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





state elements and clocking

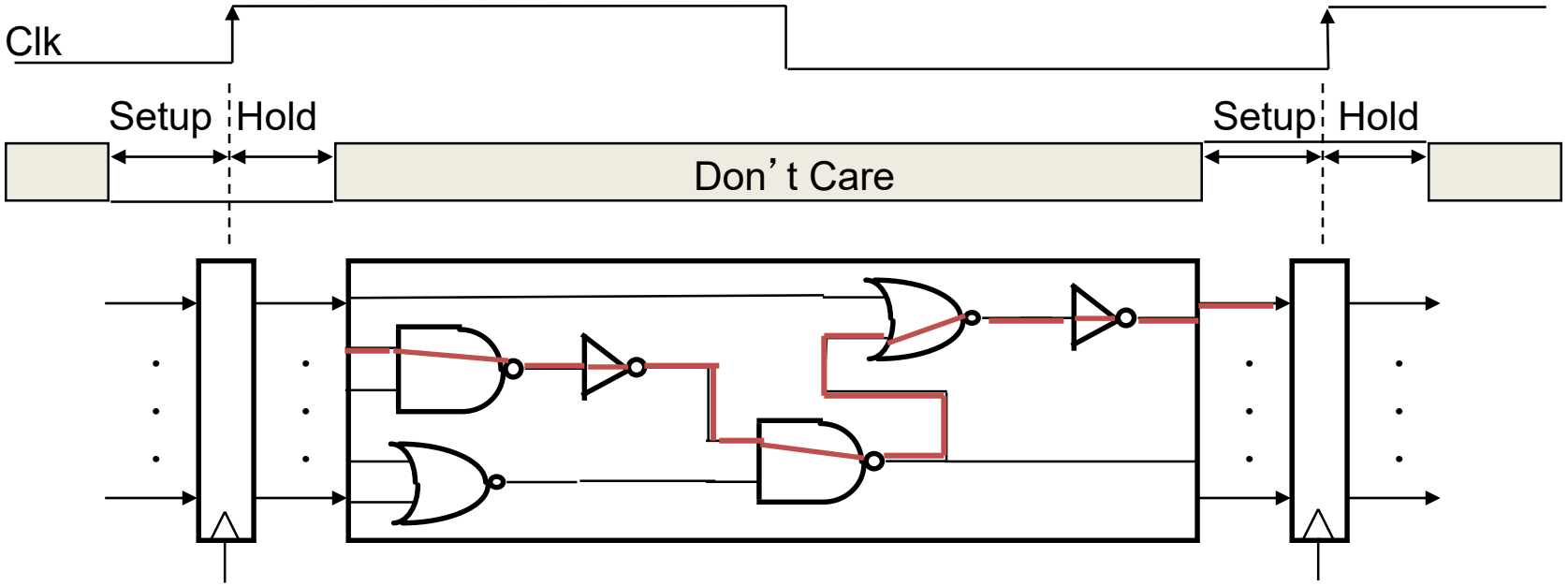


An edge triggered methodology allows a state element to be read and written in the same clock-cycle without creating a race that could lead to undermined data values

State elements are written on clock edge (under conditions met)

Clocking Methodology

"Critical path" (longest path through logic) determines length of clock period



- Registers (state elements) clocked by same edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time
- Flip-flops (FFs) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay

Uses for State Elements

- Place to store values for later re-use:
  - Register files (like $1-$31 in MIPS)
  - Memory (caches and main memory)
- *Help control flow of information between combinational logic blocks*
  - State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage

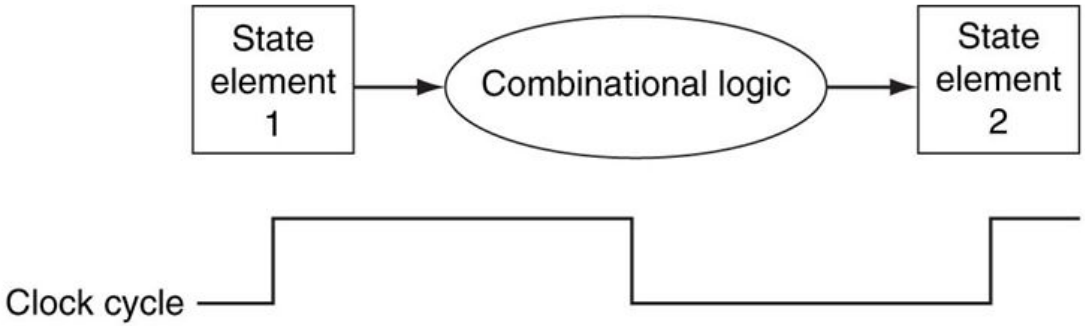
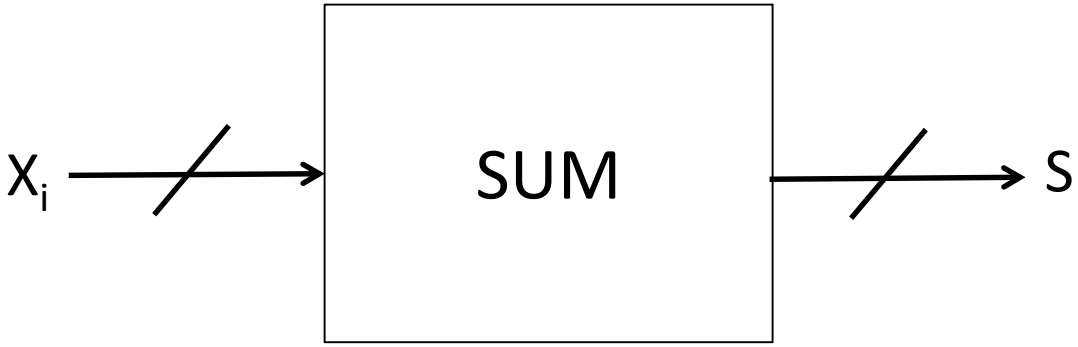


FIGURE 4.3 Combinational logic, state elements, and the clock are closely related. In a synchronous digital system, the clock determines when elements with state will write values into internal storage. Any inputs to a state element must reach a stable value (that is, have reached a value from which they will not change until after the clock edge) before the active clock edge causes the state to be updated. All state elements in this chapter, including memory, are assumed to be edge-triggered.

Accumulator Example

Why do we need to control the flow of information?



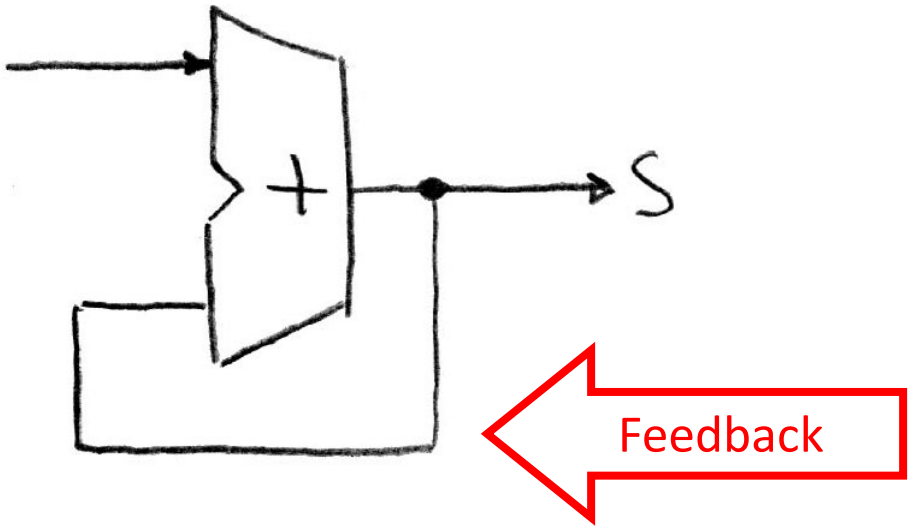
Want:

```
S=0;
for (i=0;i<n;i++)
        S = S + Xi
```

Assume:

- Each X value is applied in succession, one per cycle
- After n cycles the sum is present on S

First Try: Does this work?

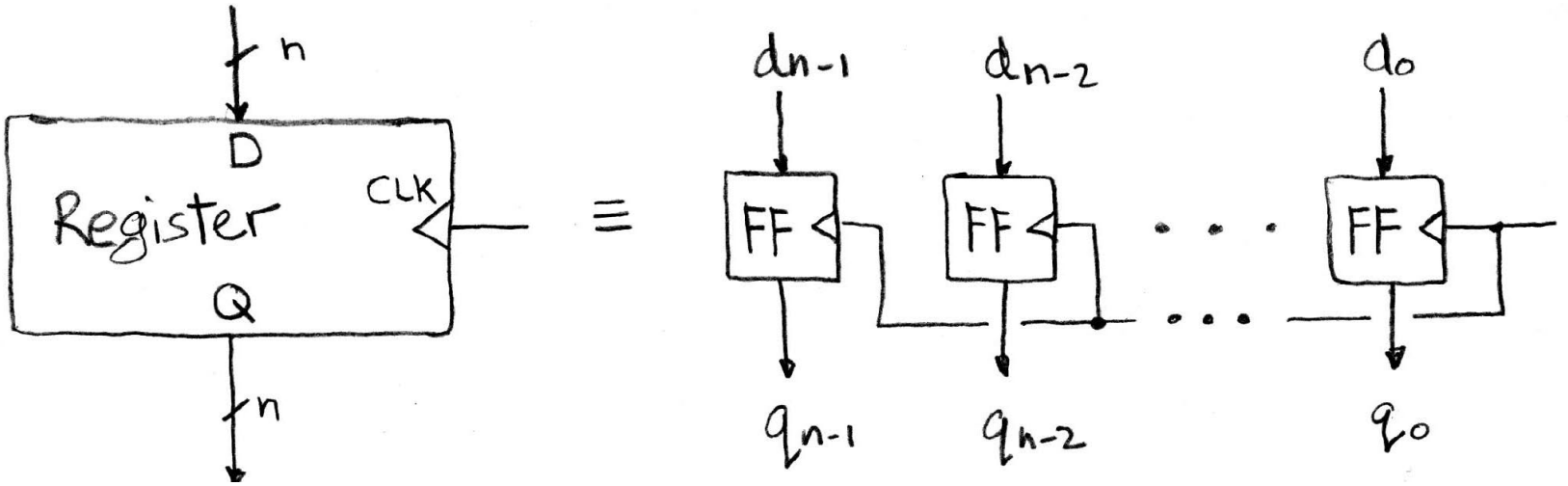


No!

Reason #1: How to control the next iteration of the 'for' loop?

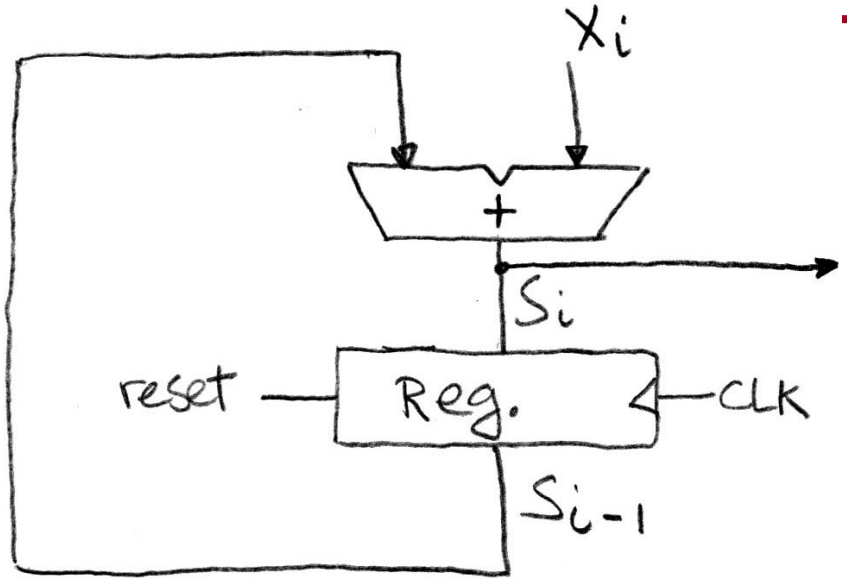
Reason #2: How do we say: 'S=0'?

Register Internals

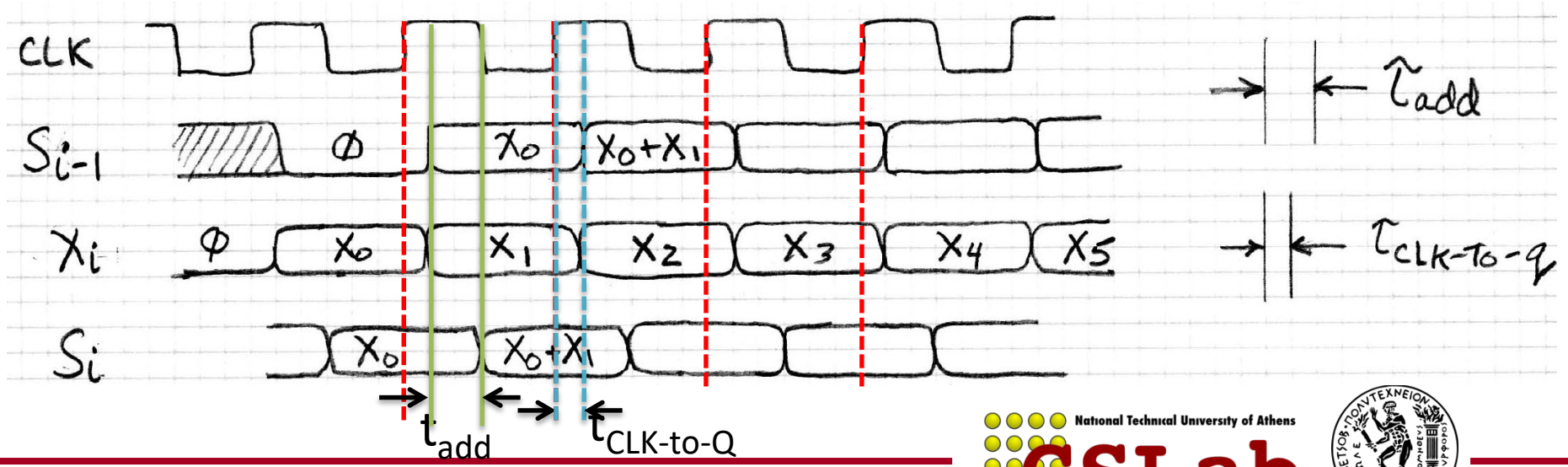


- n instances of a "Flip-Flop"
- Flip-flop name because the output flips and flops between 0 and 1
- D is "data input", Q is "data output"
- Also called "D-type Flip-Flop"

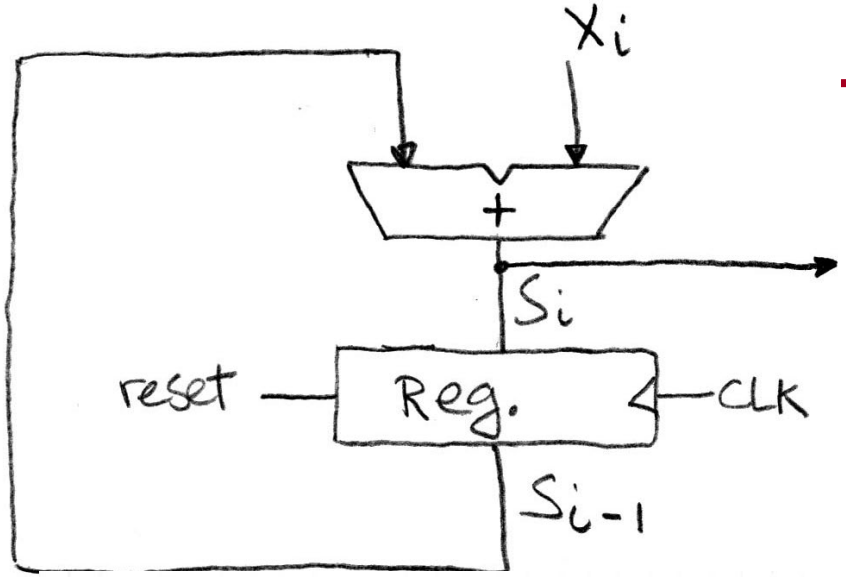
Accumulator Timing 1/2



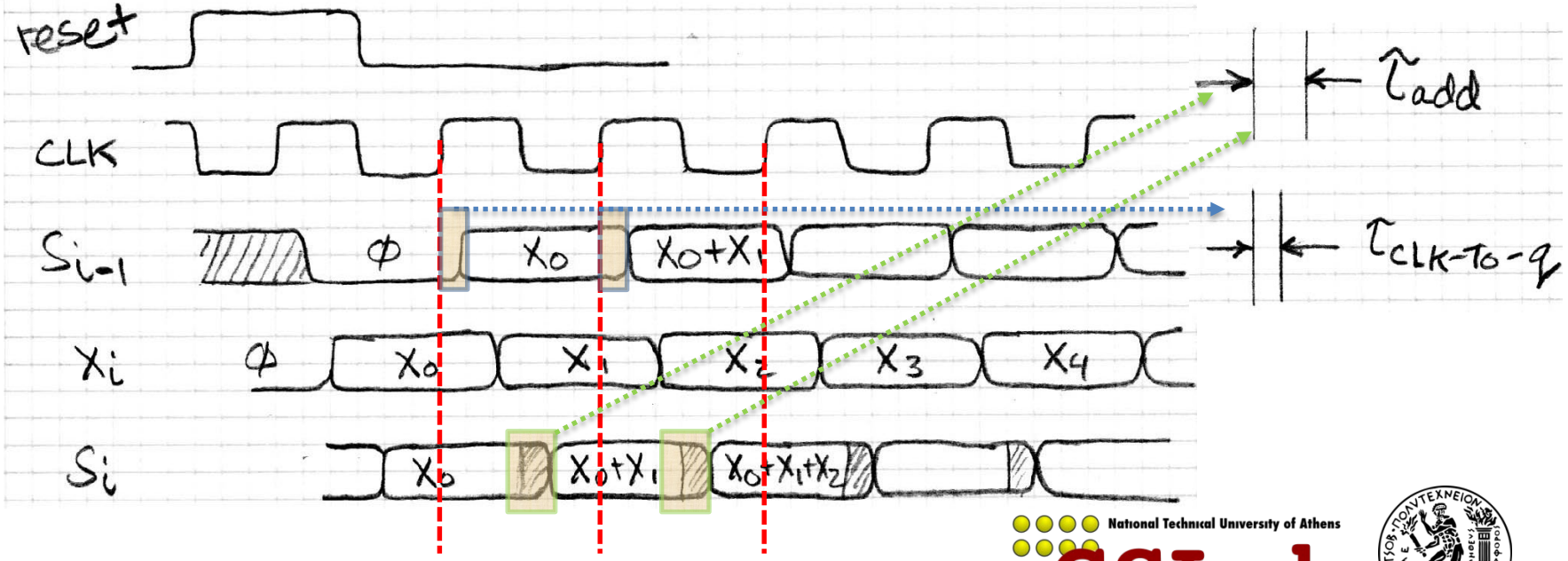
- Reset input to register is used to force it to all zeros (takes priority over D input).
- S_{i-1} holds the result of the i^{th}-1 iteration.
- Analyze circuit timing starting at the output of the register.



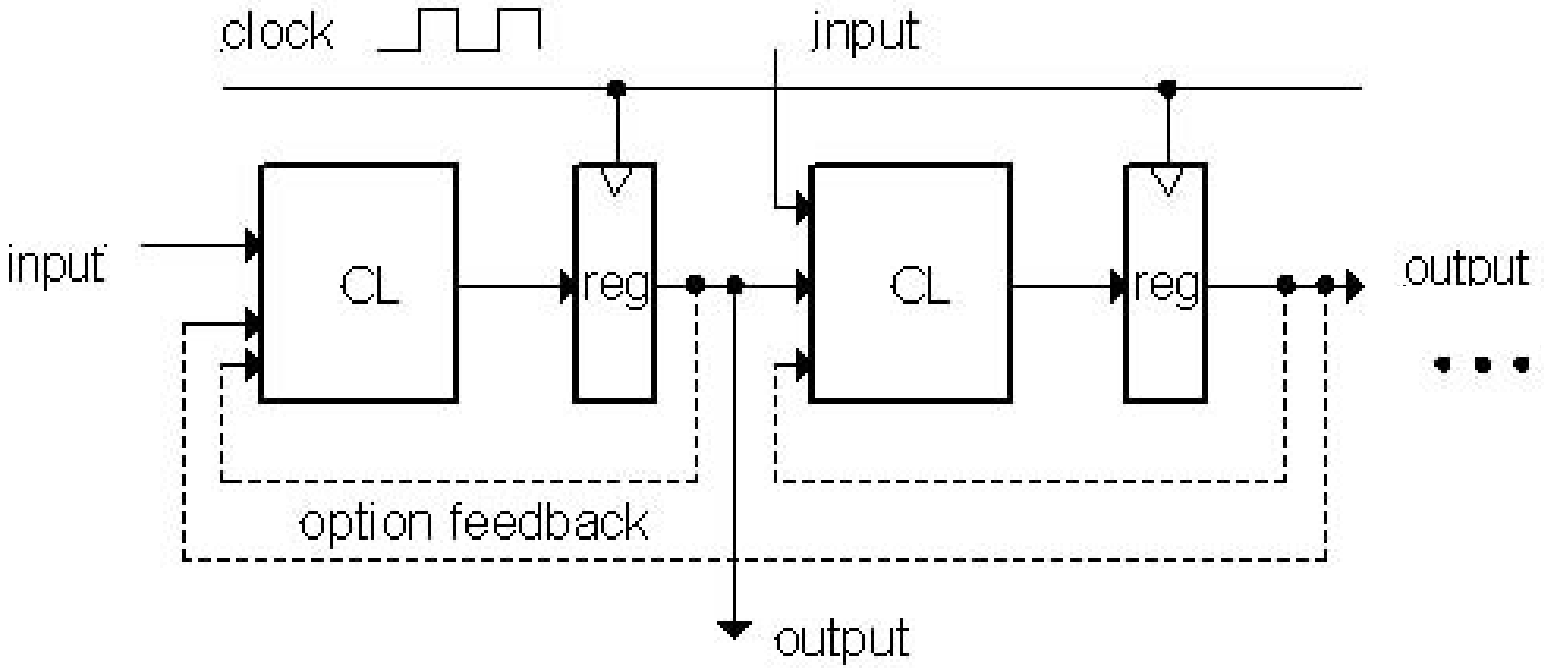
Accumulator Timing 2/2



- reset signal shown.
- Also, in practice X might not arrive to the adder at the same time as S_{i-1}
- S_i temporarily is wrong, but register always captures correct value.
- In good circuits, instability never happens around rising edge of clk.



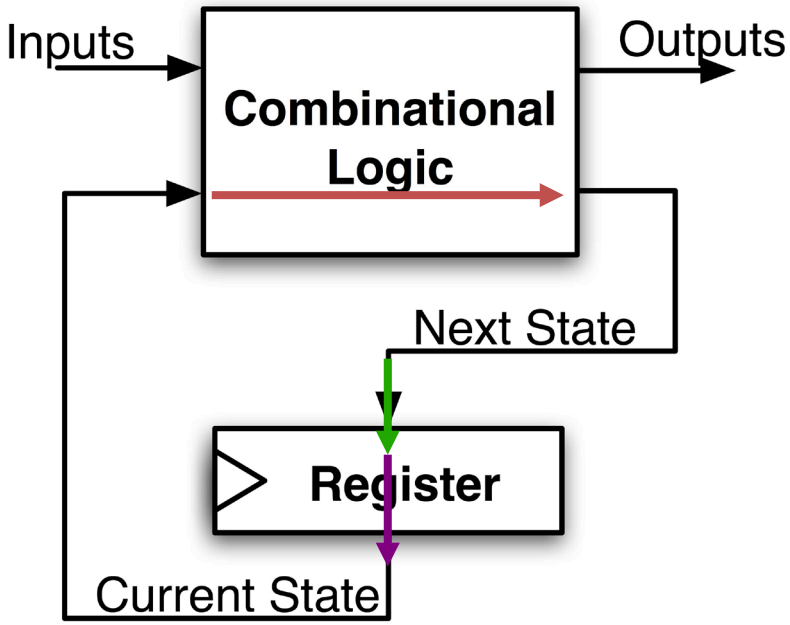
Model for Synchronous Systems



- Collection of Combinational Logic blocks separated by registers
- Feedback is optional
- Clock signal(s) connects only to clock input of registers
- Clock (CLK): steady square wave that synchronizes the system
- Register: several bits of state that samples on rising edge of CLK (positive edge-triggered) or falling edge (negative edge-triggered)

Maximum Clock Frequency

- What is the maximum frequency of this circuit?



Hint:
Frequency = 1/Period

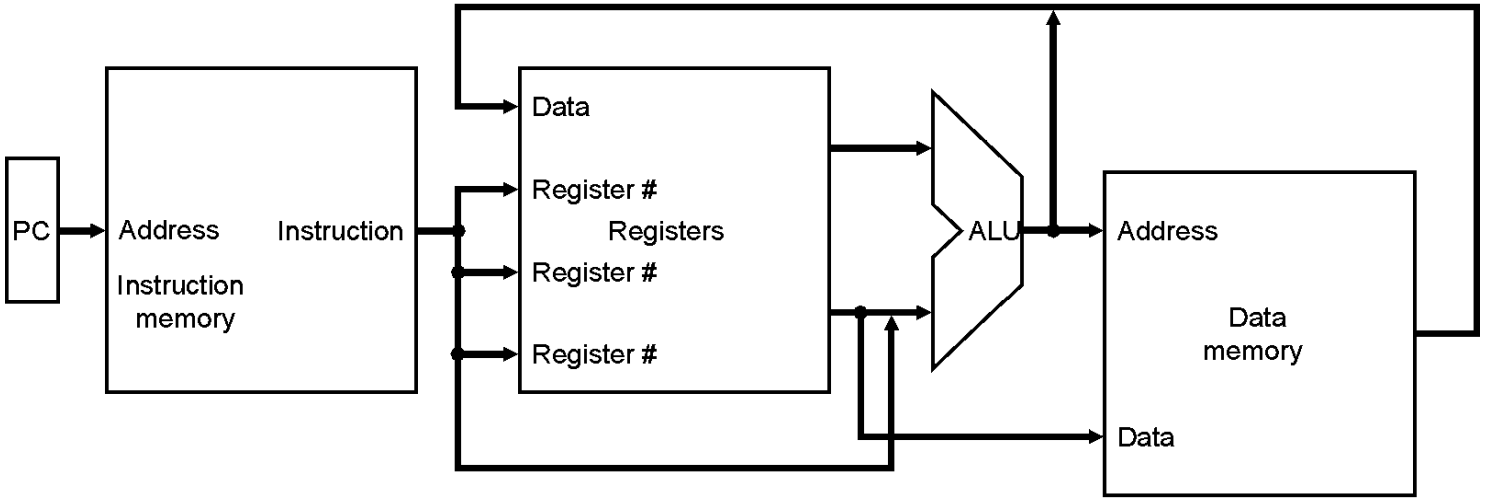
Period = Max Delay = CLK-to-Q Delay + CL Delay + Setup Time

Recap of Timing Terms

- Clock (CLK) - steady square wave that synchronizes system
- Setup Time - when the input must be stable before the rising edge of the CLK
- Hold Time - when the input must be stable after the rising edge of the CLK
- "CLK-to-Q" Delay - how long it takes the output to change, measured from the rising edge of the CLK

- Flip-flop - one bit of state that samples every rising edge of the CLK (positive edge-triggered)
- Register - several bits of state that samples on rising edge of CLK or on LOAD (positive edge-triggered)

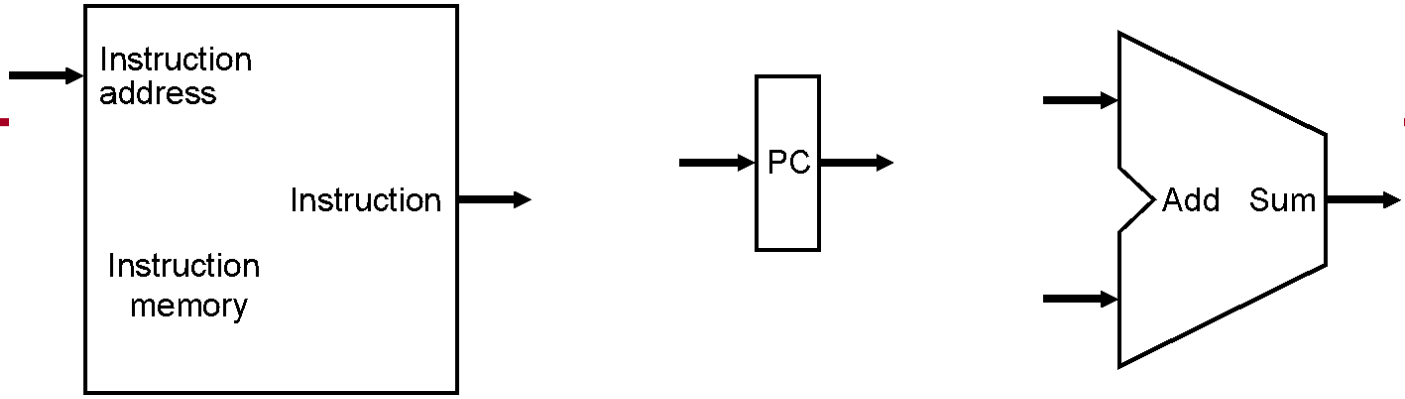
Απλή (αφαιρετική) μορφή ενός datapath:



Αρχικά, θα κάνουμε σχεδίαση ενός κύκλου (single cycle)!

(*κάθε εντολή διαρκεί έναν κύκλο ρολογιού*)

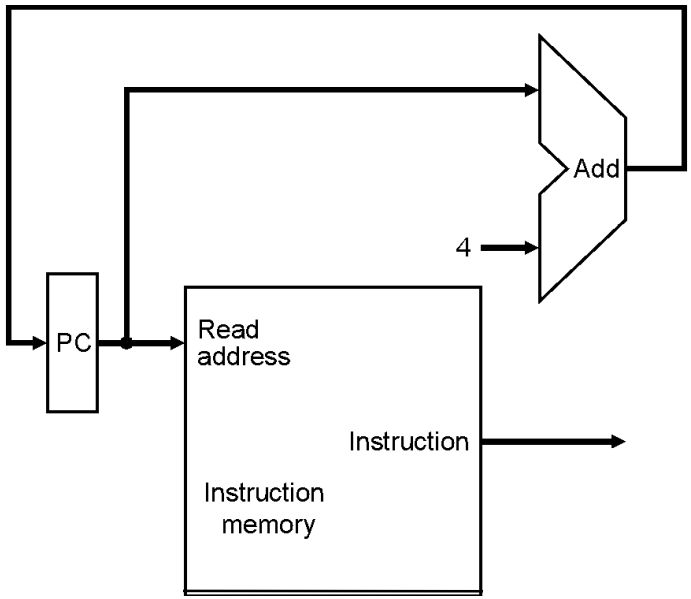
Στοιχεία για αποθήκευση και προσπέλαση εντολών:



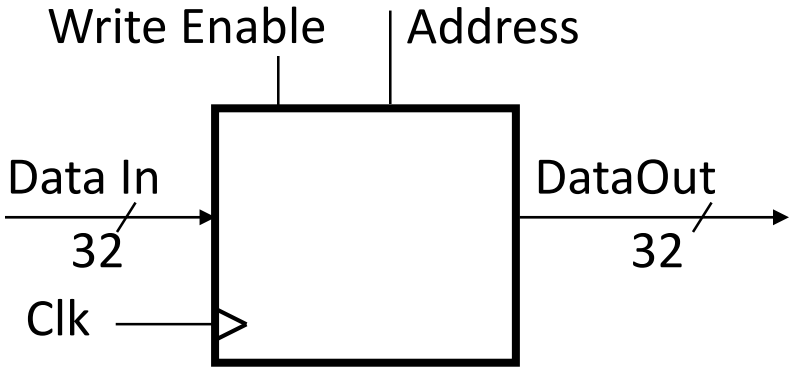
a. Instruction memory b. Program counter c. Adder

Αθροιστής για υπολογισμό PC+4 (επόμενη εντολή)

Fetching instructions and incrementing PC:

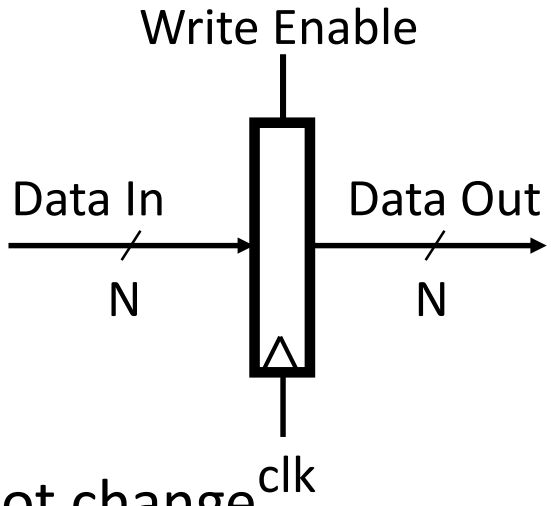


Storage Element: Idealized Memory



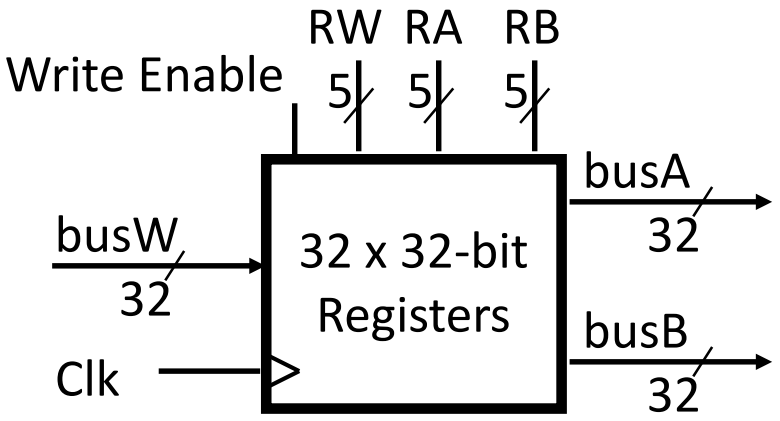
- "Magic" Memory
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - For Read: Address selects the word to put on Data Out
  - For Write: Set Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block: Address valid $\Rightarrow$ Data Out valid after "access time"

Storage Element: Register (Building Block)



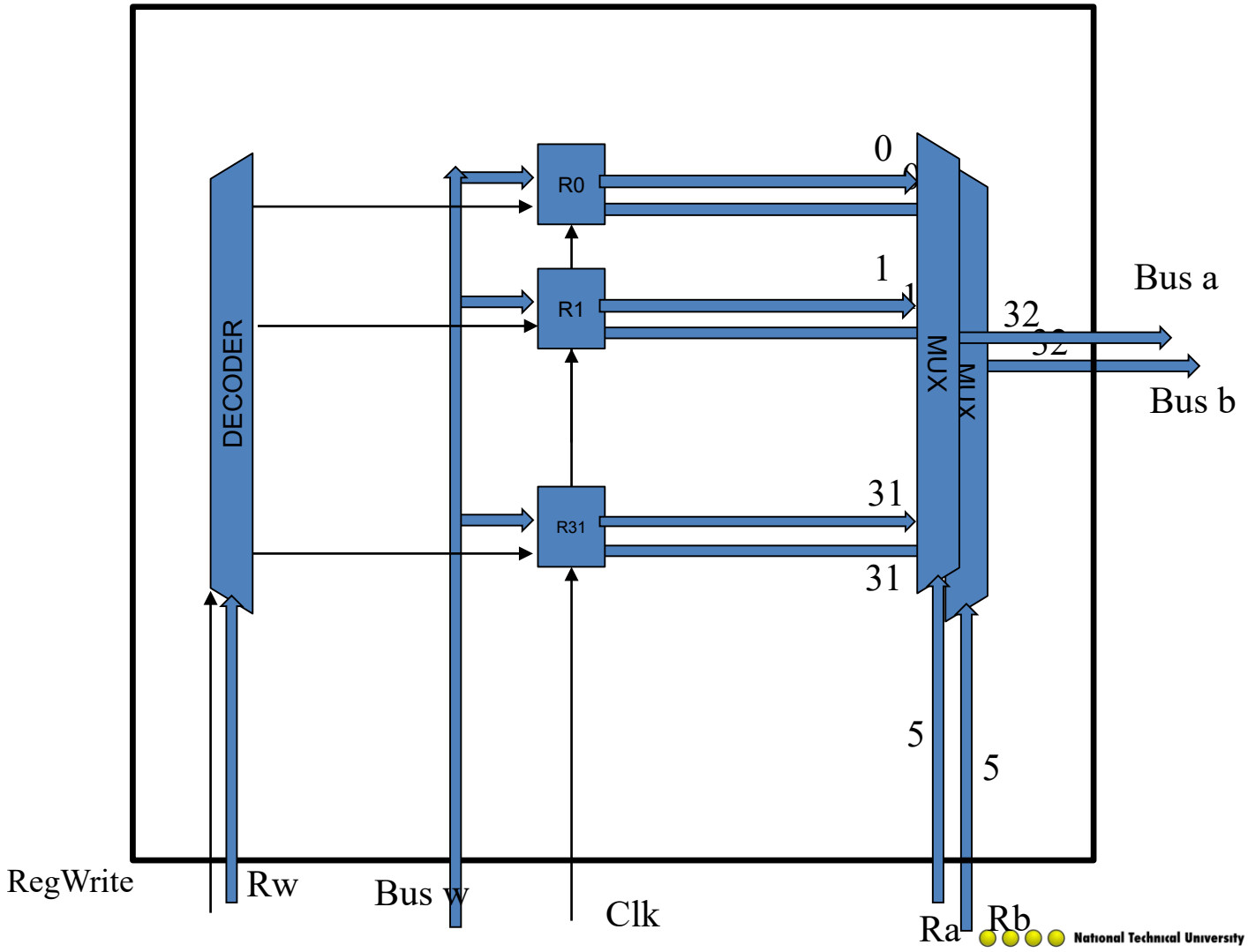
- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - Negated (or deasserted) (0): Data Out will not change
  - Asserted (1): Data Out will become Data In on positive edge of clock

Storage Element: Register File



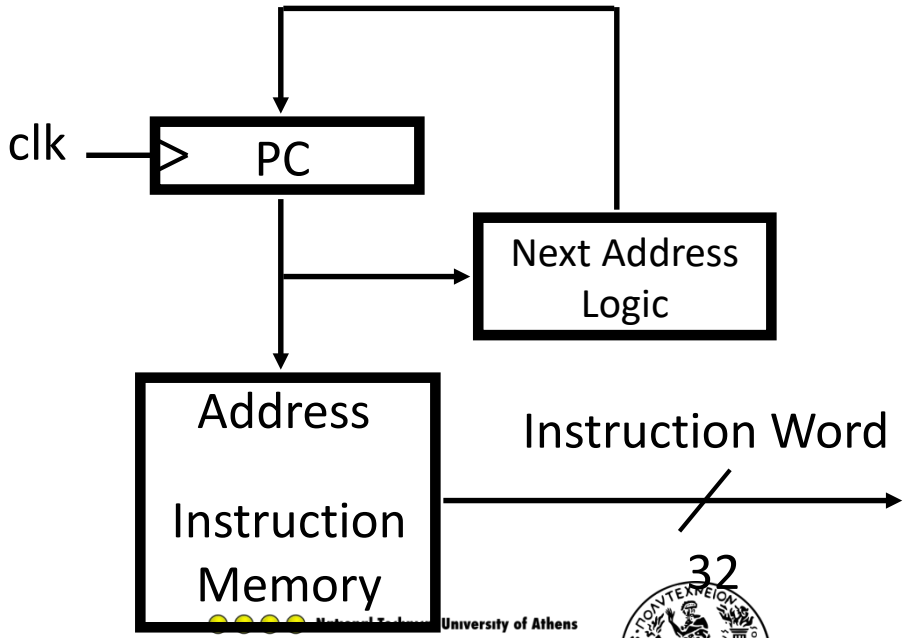
- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (clk)
  - Clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after "access time."

Δομή Register File



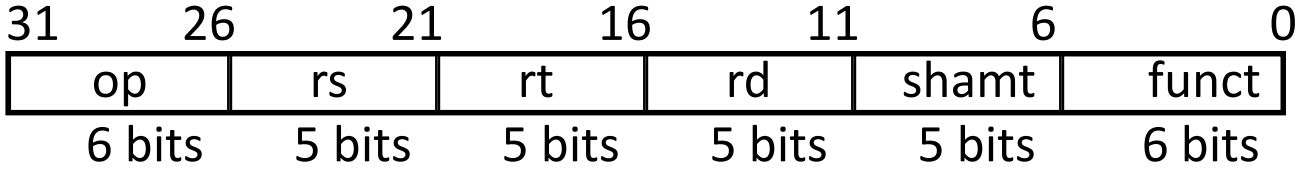
Step 3a: Instruction Fetch Unit

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

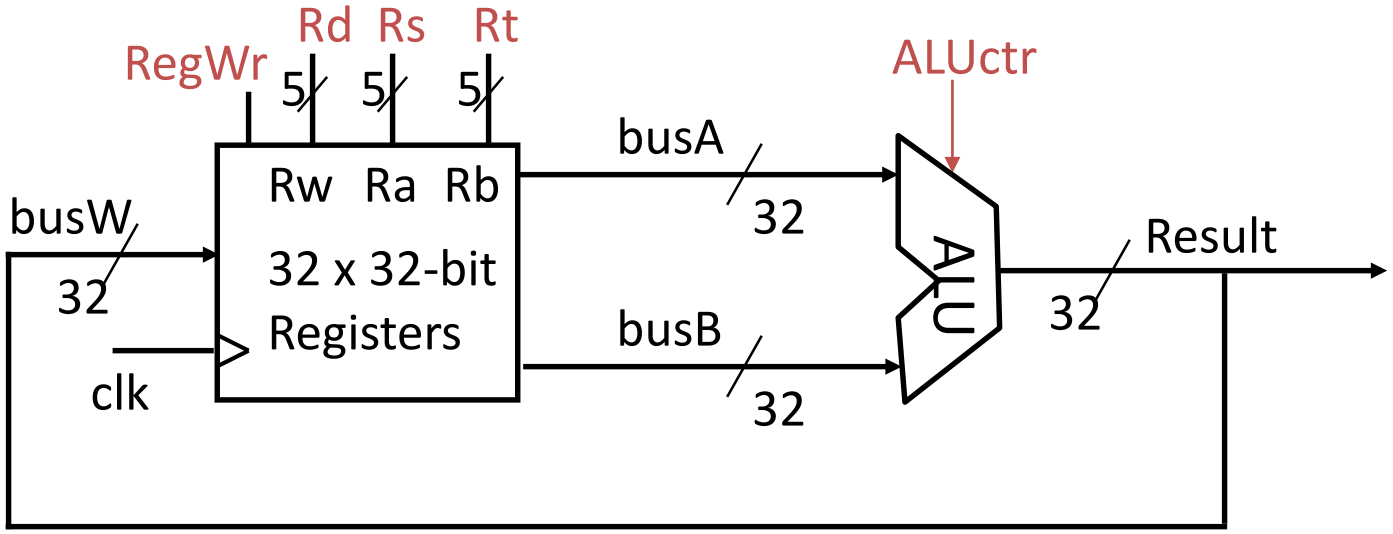


Step 3b: Add & Subtract

- `R[rd] = R[rs] op R[rt] (addu rd,rs,rt)`
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields

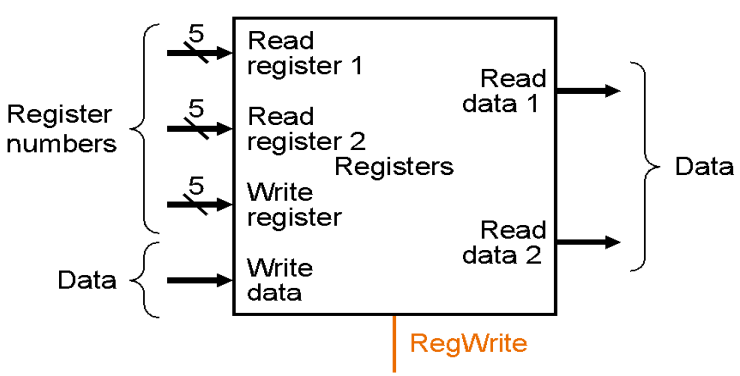


  - ALUctr and RegWr: control logic after decoding the instruction

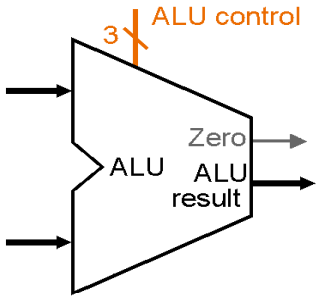


- … Already defined the register file & ALU

Α. Υλοποίηση R-Type εντολών:



a. Registers



b. ALU

Register file:

Two read ports, one write port (32 bit)

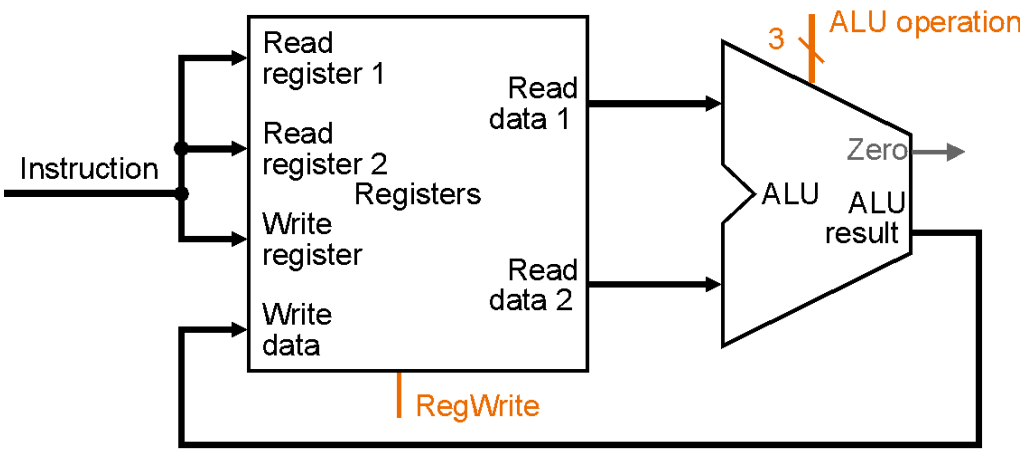
Δεν υπάρχει RegRead!

Υπάρχει RegWrite (edge triggered) όταν πρόκειται να γράψουμε Read and write Reg File στον ίδιο κύκλο (το read παίρνει την τιμή που γράφτηκε στον προηγούμενο κύκλο ενώ η τιμή που γράφεται είναι διαθέσιμη στον επόμενο κύκλο

ALU:

Zero output για branches

Τμήμα του Datapath για R-Type:



R-Type
(register type)

op	rs	rt	rd	shamt	funct
6 bits	*5bits*	*5bits*	*5bits*	*5bits*	*6bits*

```
add $rd, $rs, $rt
```

π.χ. add $s1, $s1, $s2

Σχεδίαση datapath για I-Type εντολές:

Πρώτα θα σχεδιάσουμε για load-stores

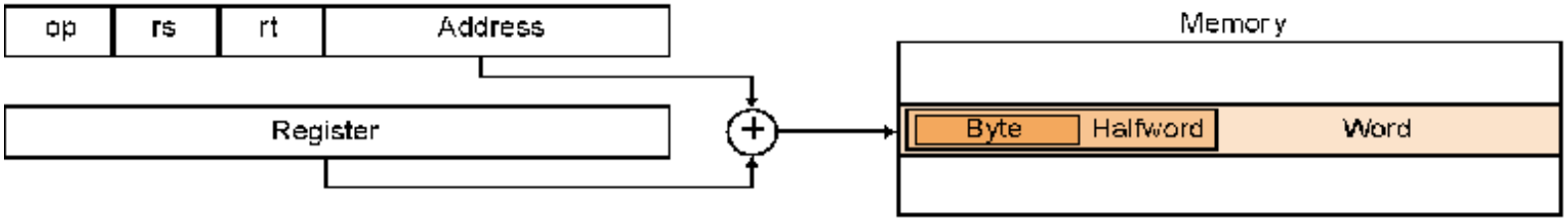
I-Type:

op	rs	rt	address_offset
6 bits	*5 bits*	*5 bits*	*16 bits*

```
lw $rt, offset_value($rs)
          ή
sw $rt, offset_value($rs)
```

Offset_value: 16bit signed field

(Χρειάζεται sign extension γιατί προστίθεται σε 32 bit register)

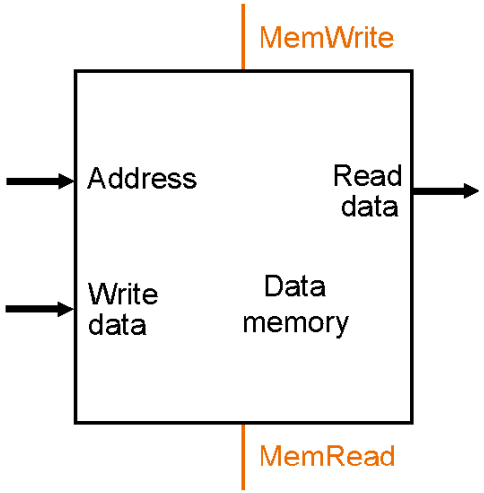


lw $rt, address($rs)

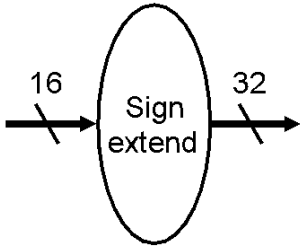
π.χ. lw $t1,100($s2)

Τα 3 πρώτα πεδία (op,rs, rt) έχουν το ίδιο όνομα και μέγεθος όπως και πριν

Επιπλέον δομικές μονάδες για load-store datapath:



a. Data memory unit



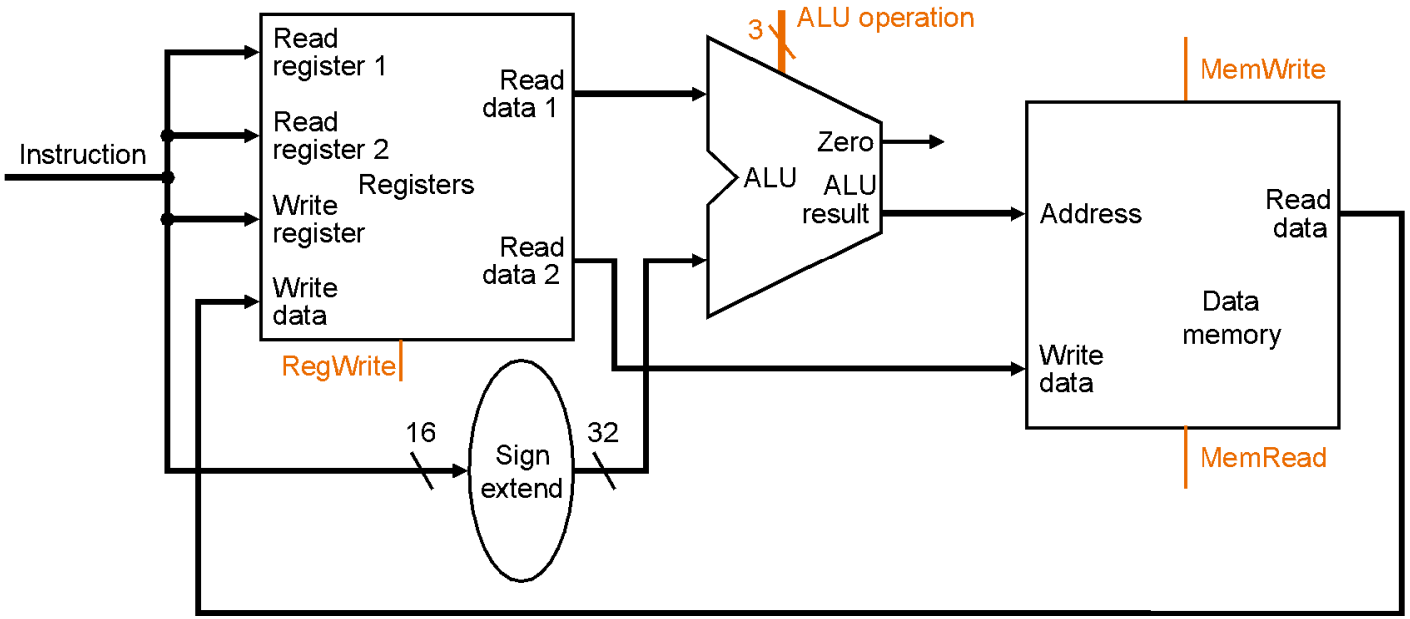
b. Sign-extension unit

Memory:

address port ,read, write data port(32bit)

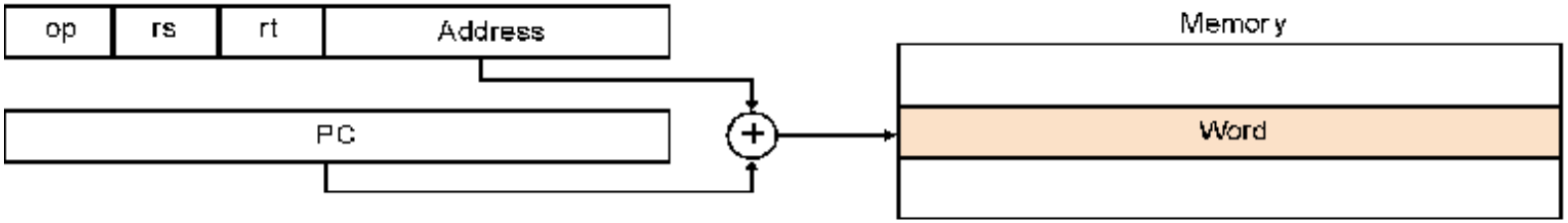
Sign extension!

Datapath για load-store:



register file access → memory address calculation → read or write from memory → write back to rf if we have lw instruction

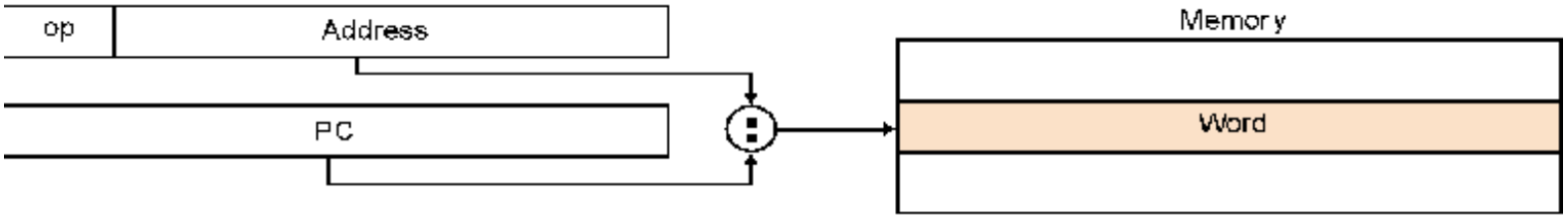
Σχεδίαση datapath για branch instructions:



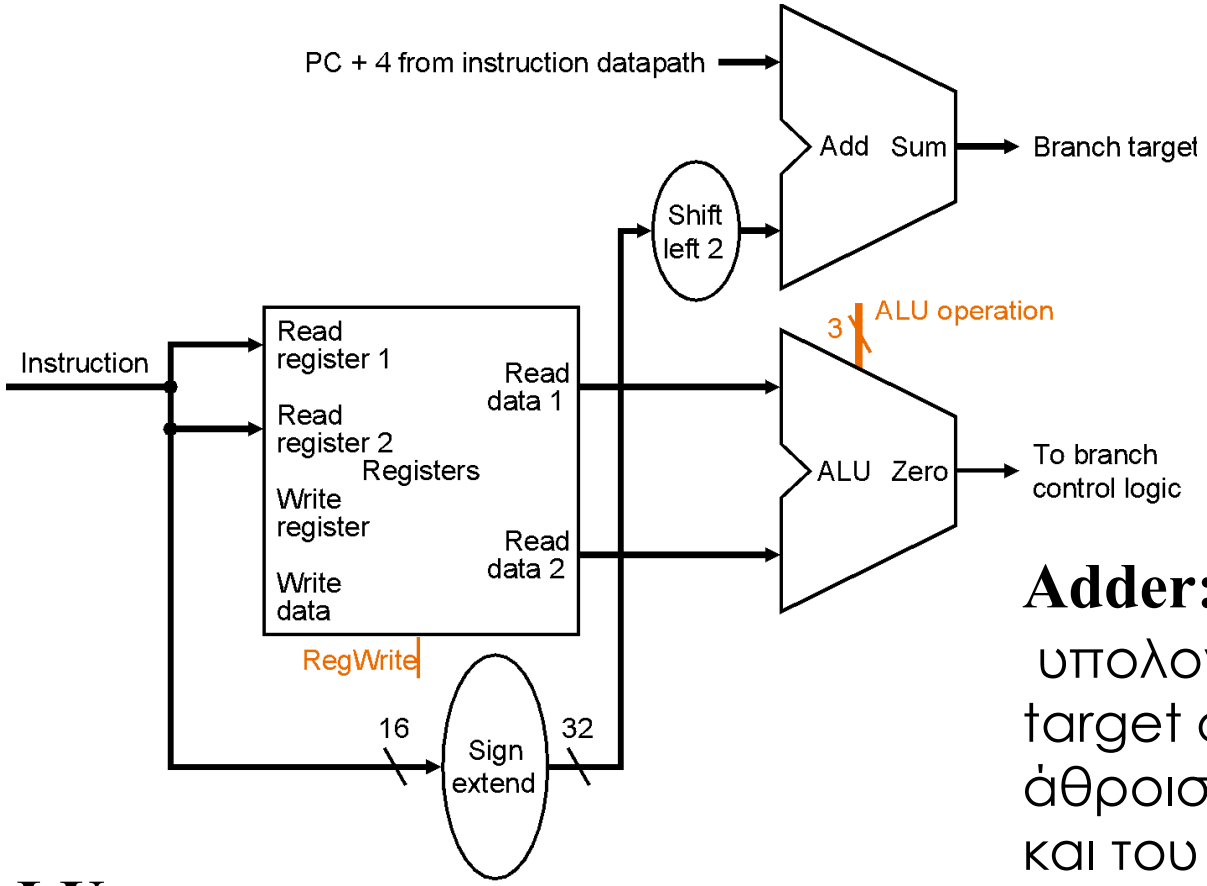
bne $rs, $rt, address

π.χ. bne $s0,$s1,L2

PC relative addressing άρα *address* χρειάζεται sign extension και x4 (αναφορά σε διευθύνσεις λέξεων)



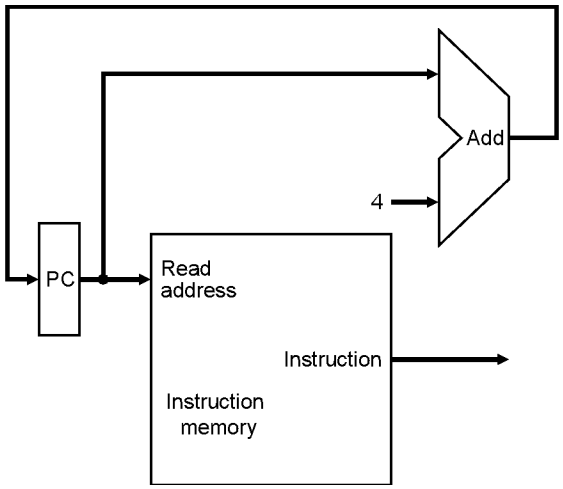
Datapath για branch:



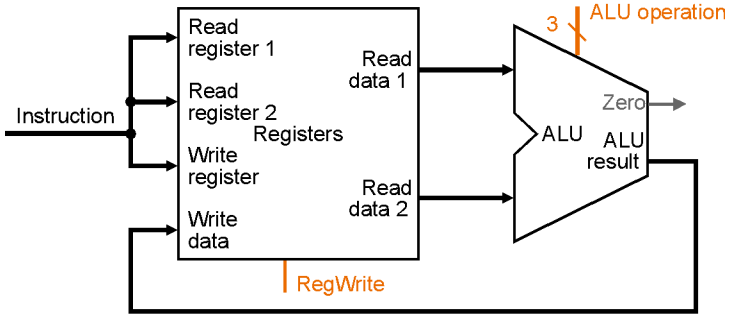
ALU:
evaluates the branch condition

Adder:
υπολογίζει το branch target σαν το άθροισμα του PC+4 και του sign extended, x4 όρισμα της εντολής

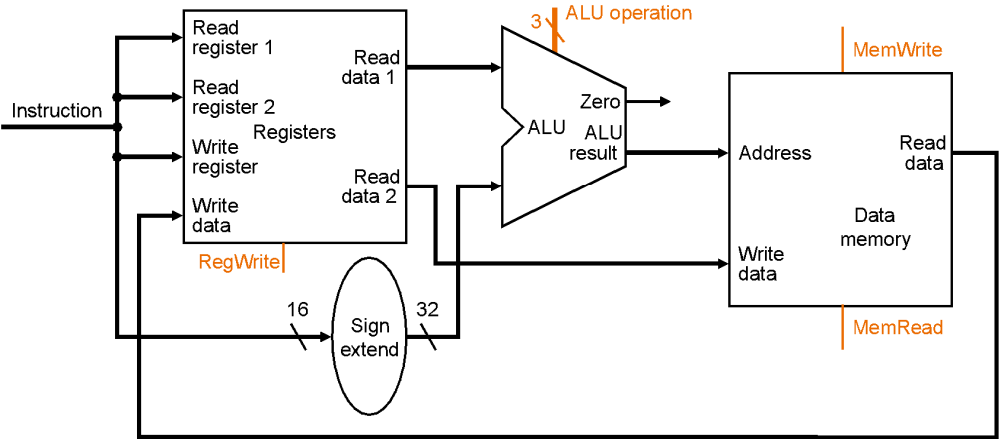
Ενώνοντας τα… όλα μαζί (δημιουργία single datapath):



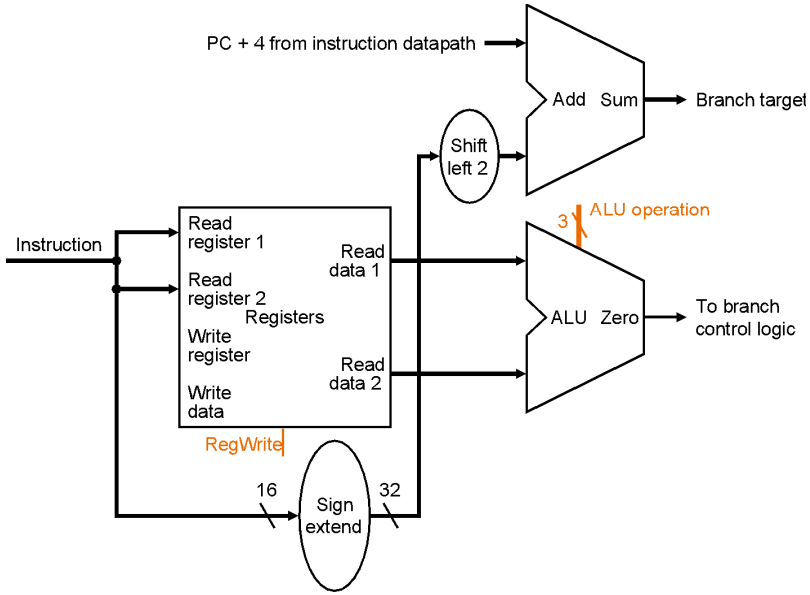
Fetch-Decode+Register File Read



R-Type



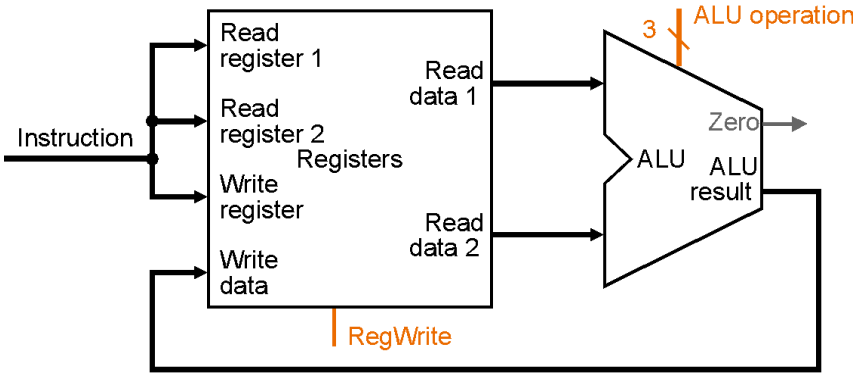
I-Type Load store



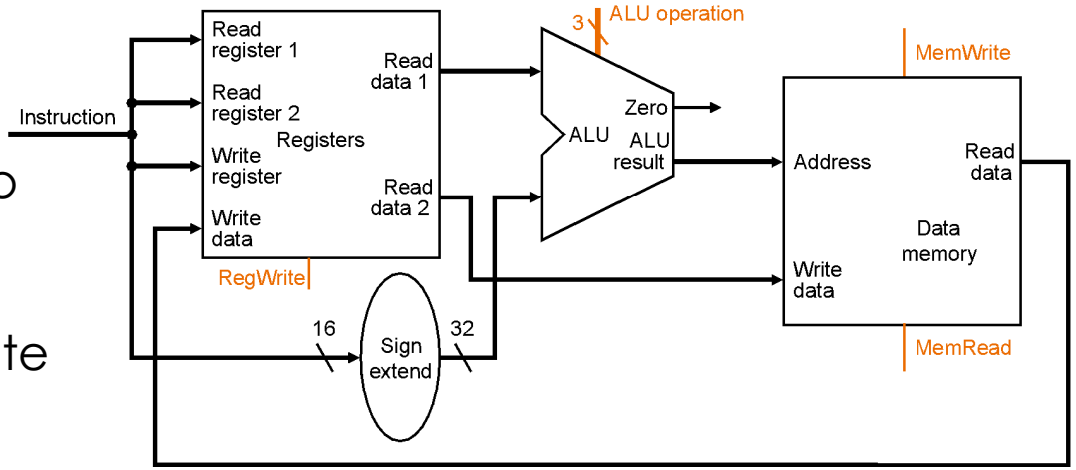
I-Type branch

Μήπως ...μοιάζουν;

2η είσοδος στην ALU είναι είτε καταχωρητής (R-Type) είτε το sign-extended lower half (16bit) της εντολής (address_offset-αν είναι load-store



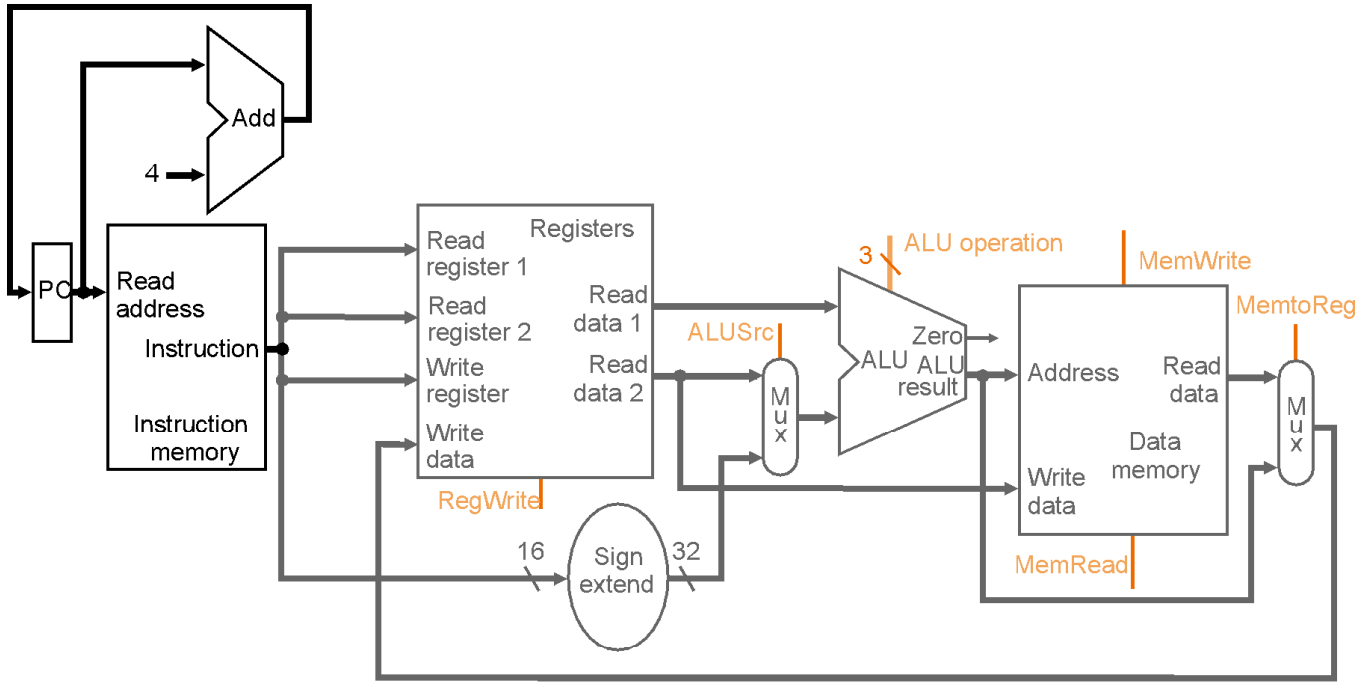
Η τιμή που αποθηκεύεται στο write data του register file έρχεται είτε από την ALU (R-Type) είτε από τη μνήμη (write back σε lw εντολή)



R-Type + load-store instructions combined datapath:

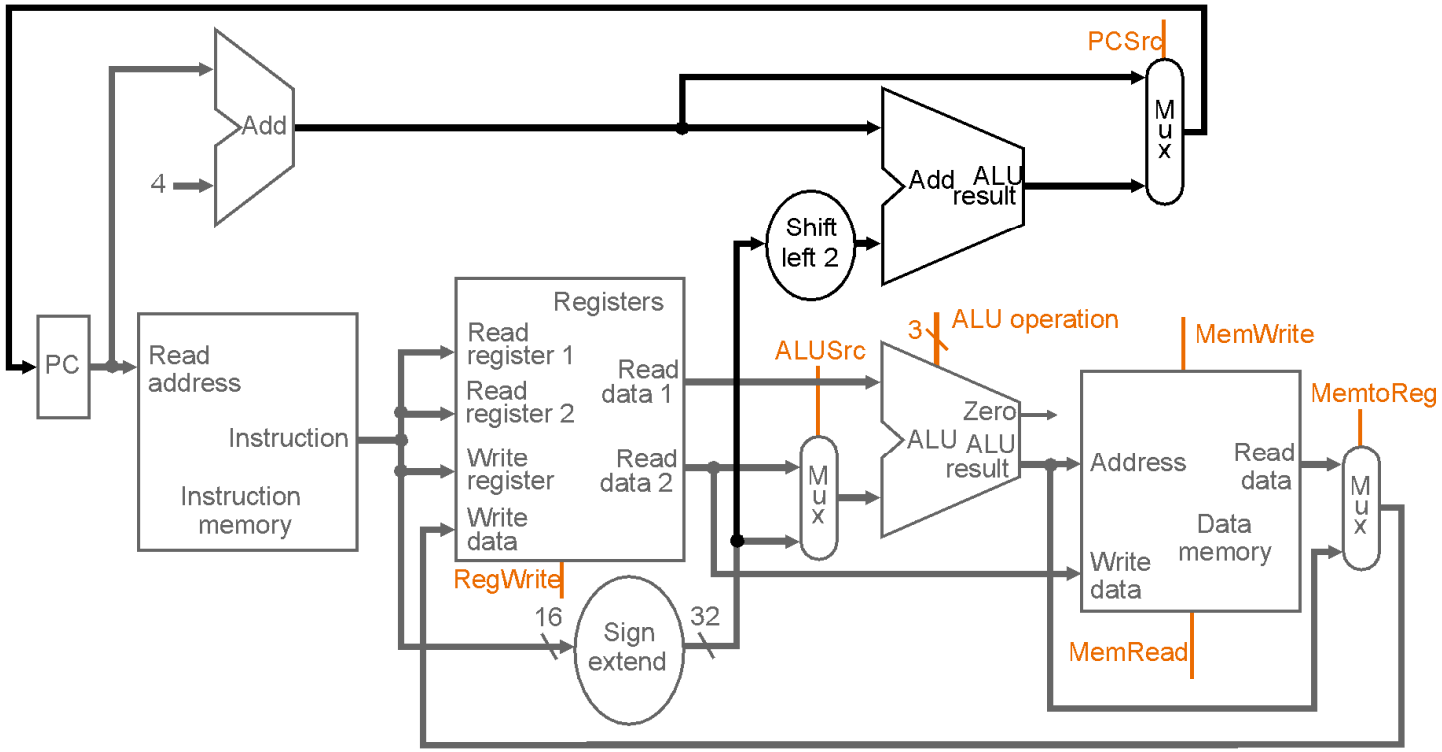
Χρησιμοποιούμε πολυπλέκτες (mux)

Single register file and single ALU:

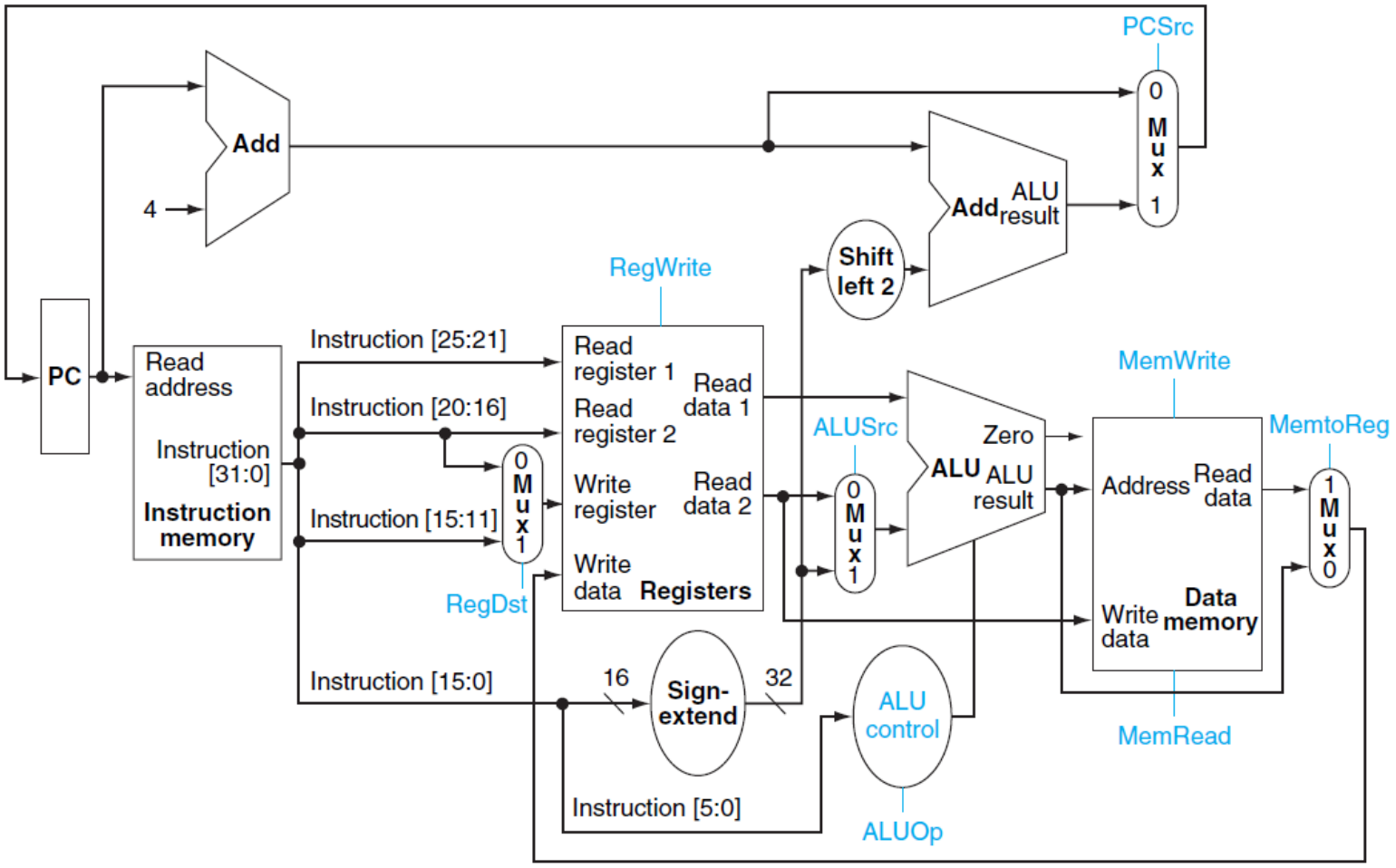


Τι γίνεται με τα branch instructions;

Mux για επιλογή μεταξύ PC+4 και Label από branch instruction



Final Datapath : Πολυπλέκτες και control signals



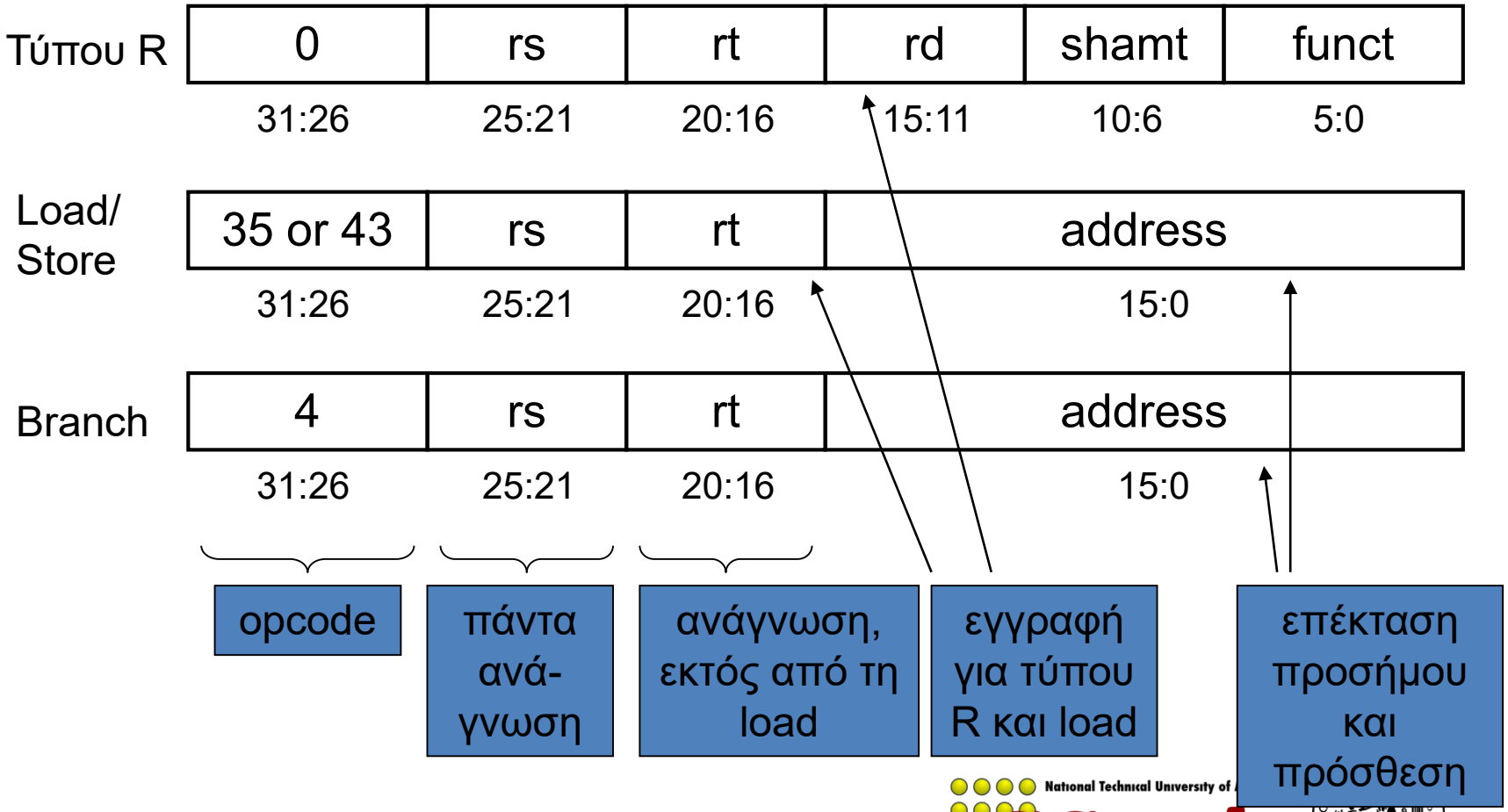
Control Signals

- Προκύπτουν με βάση το opcode της εντολής (bits 31-26)

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	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

Η κύρια μονάδα ελέγχου

Σήματα ελέγχου που εξάγονται από εντολή



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

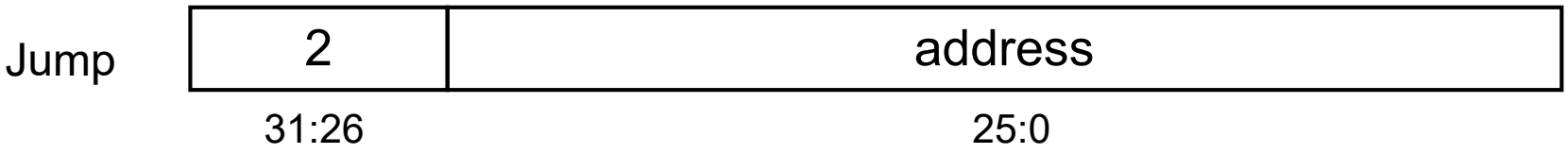
Έλεγχος ALU

- 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

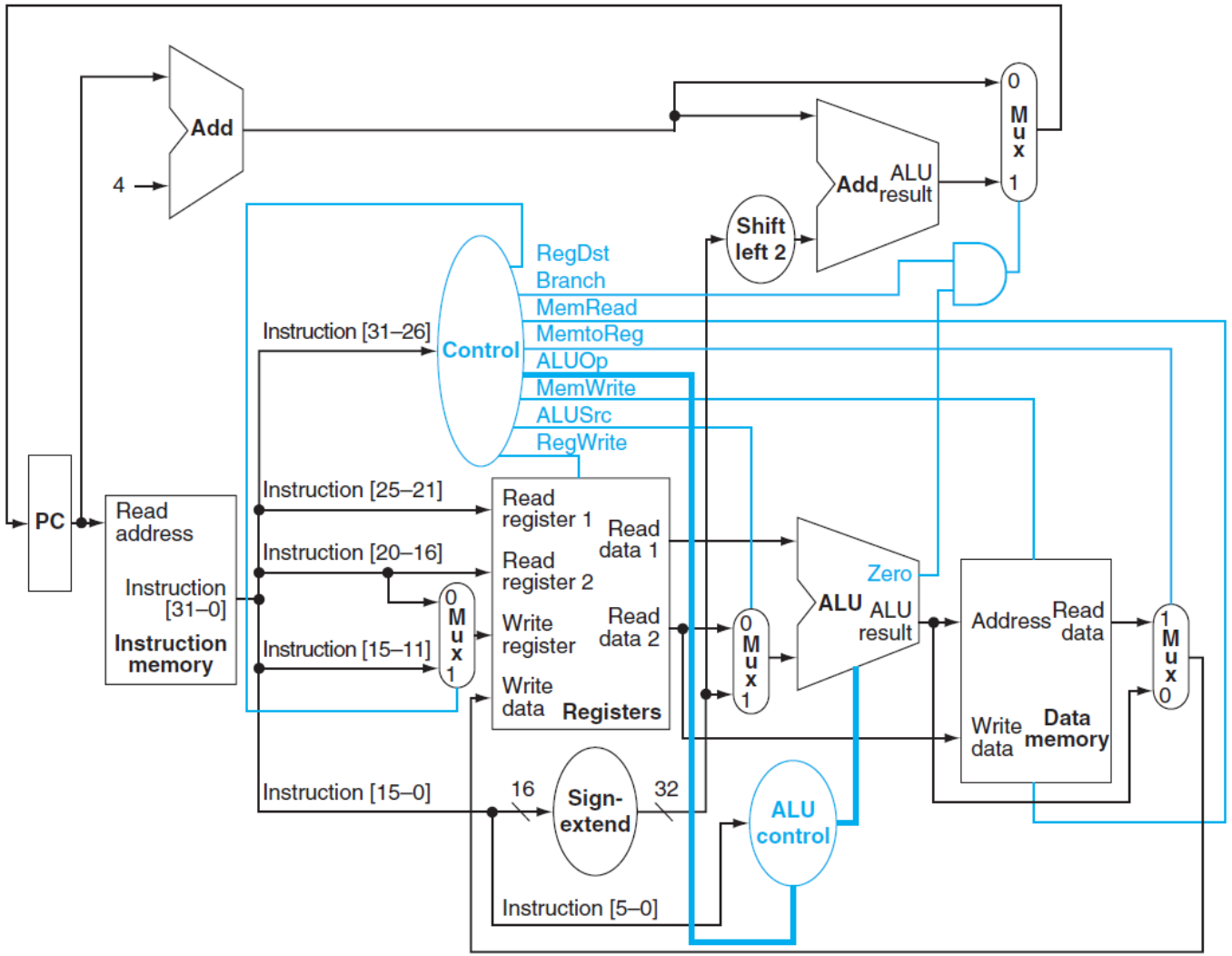
Δεν υπάρχει στο παράδειγμα του βιβλίου

Υλοποίηση αλμάτων

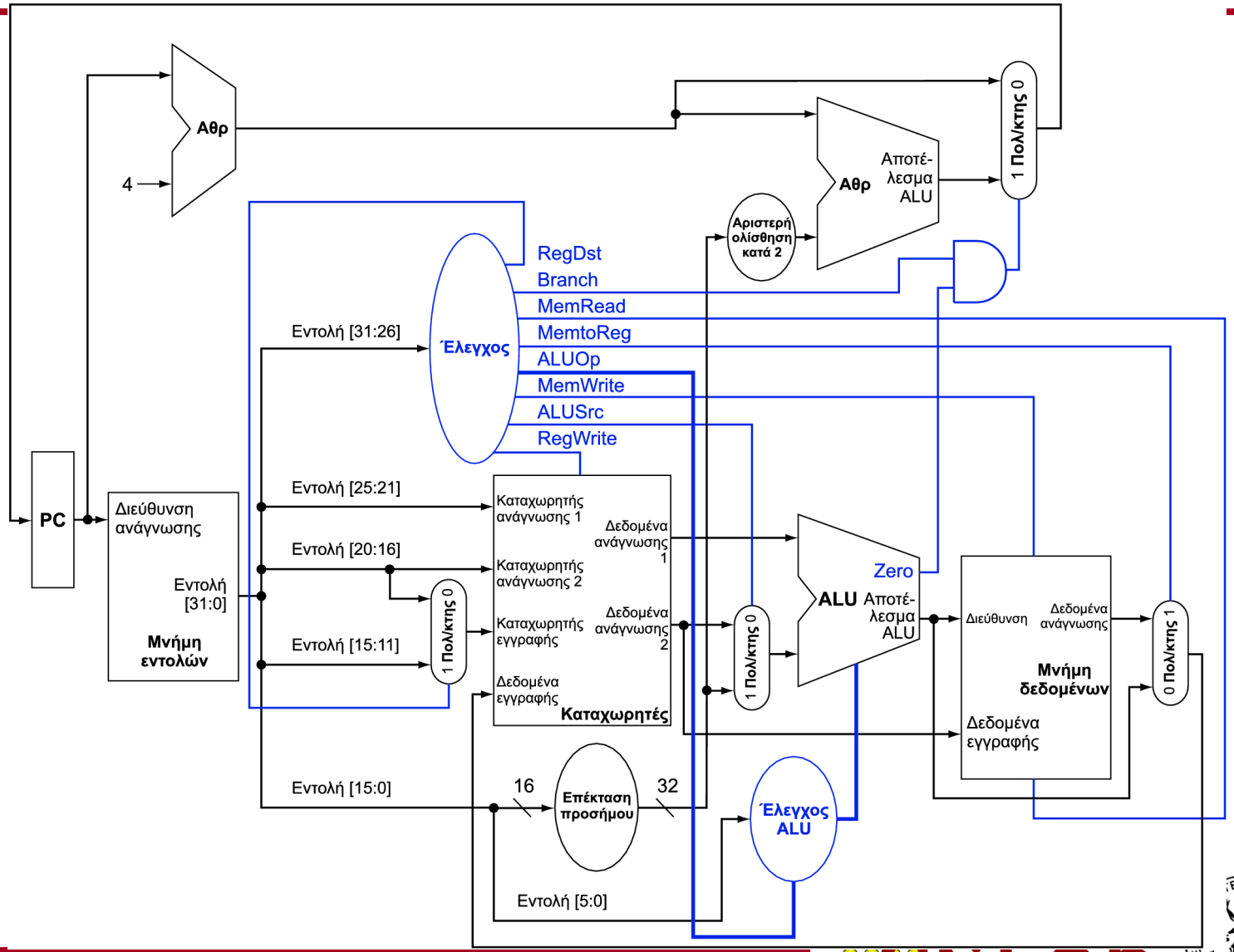


- Η Jump χρησιμοποιεί διεύθυνση λέξης
- Ενημέρωση του PC με συνένωση των
  - Υψηλότερων 4 bit του παλιού PC
  - Διεύθυνσης άλματος των 26 bit
  - 00
- Χρειάζεται ένα επιπλέον σήμα ελέγχου από την αποκωδικοποίηση του opcode

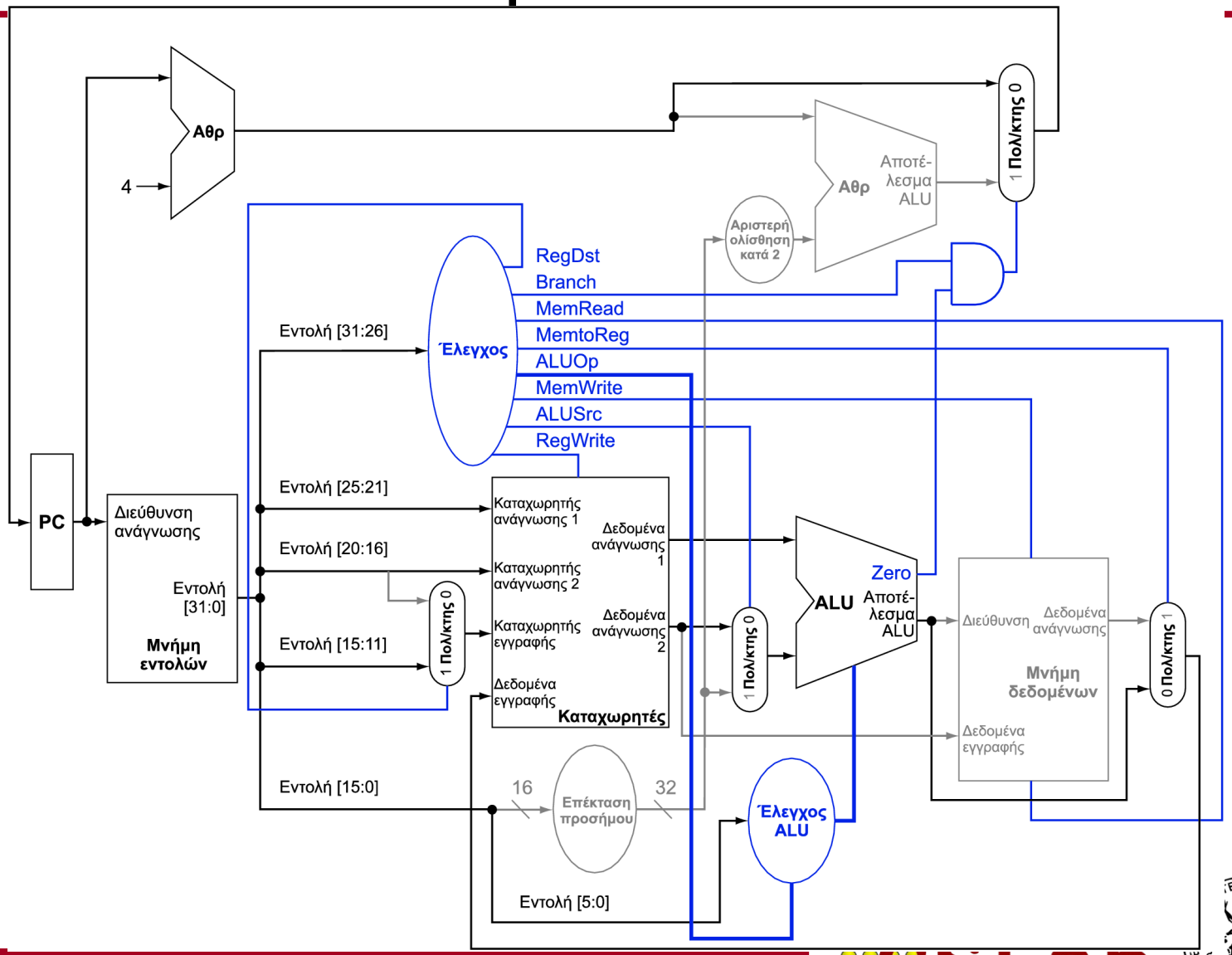
Final Datapath + Control Unit



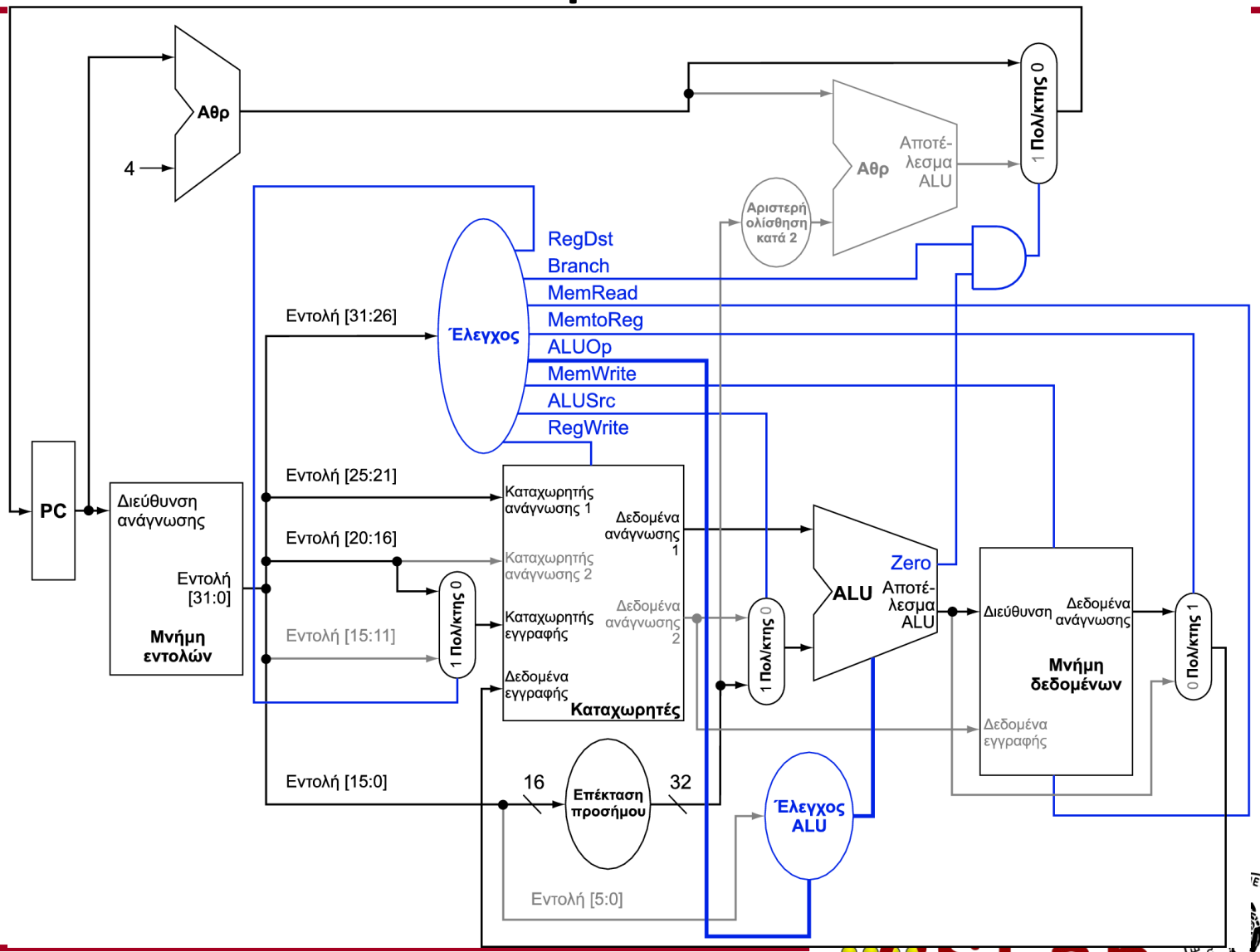
Διαδρομή δεδομένων και έλεγχος



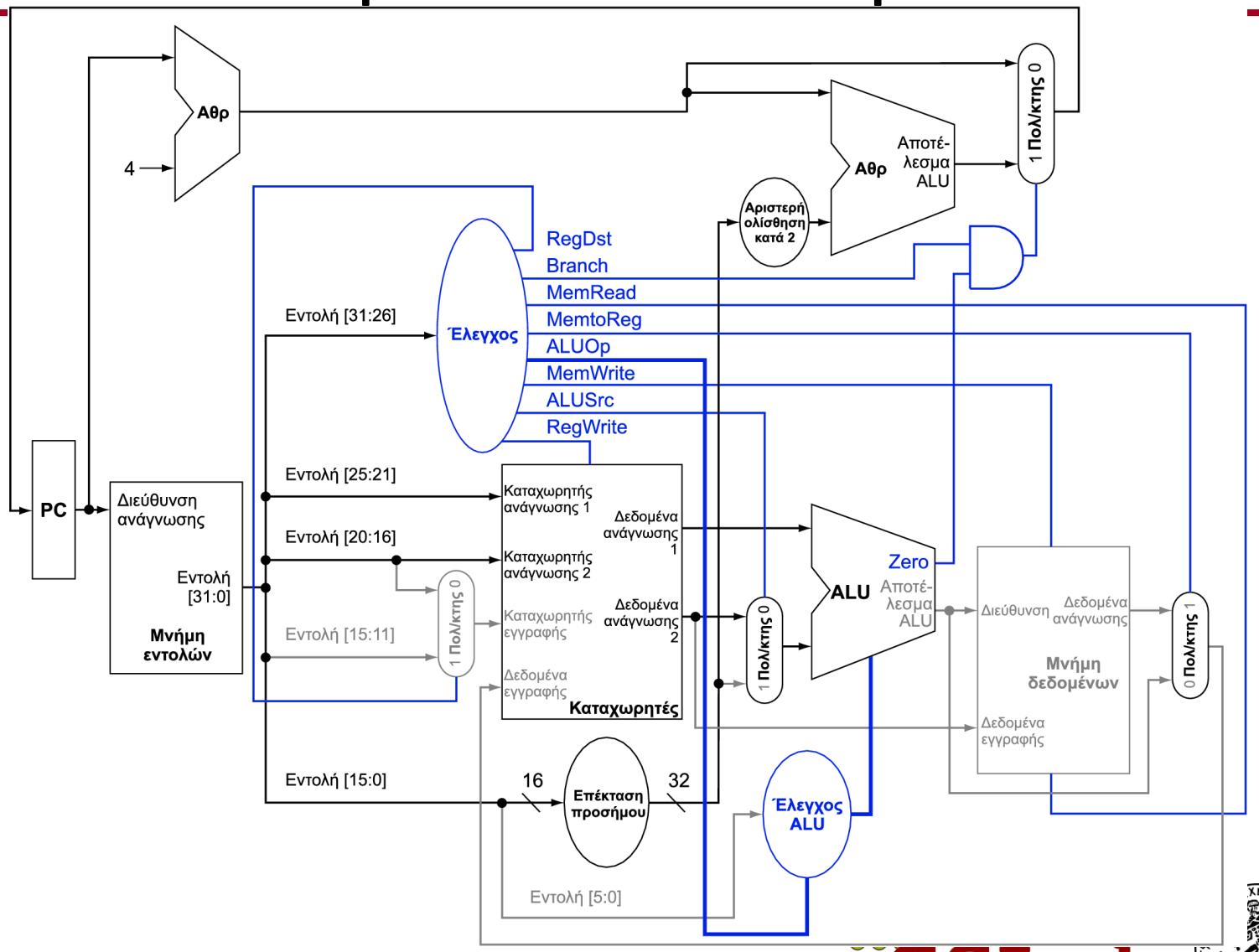
Εντολή τύπου R



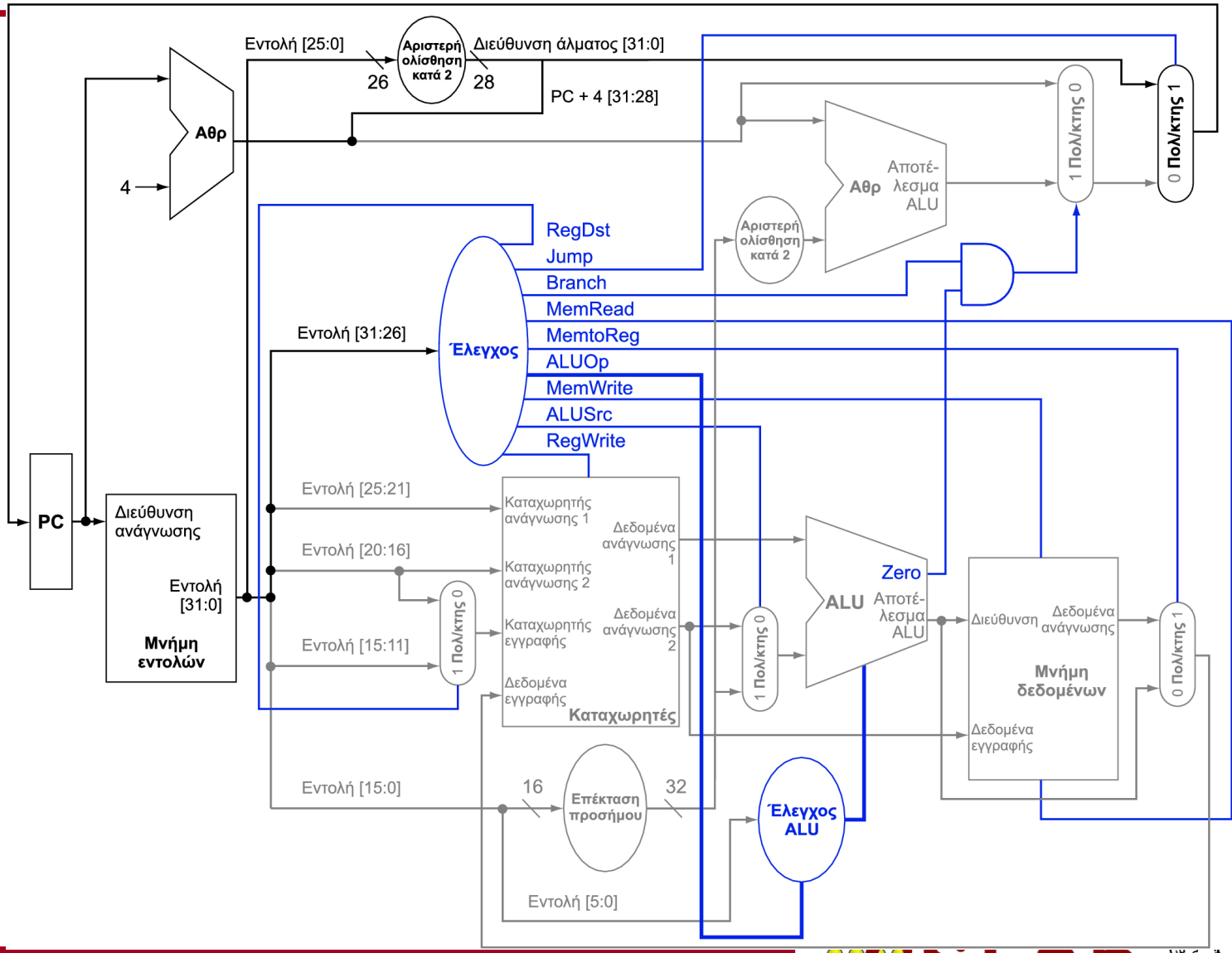
Εντολή Load


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]

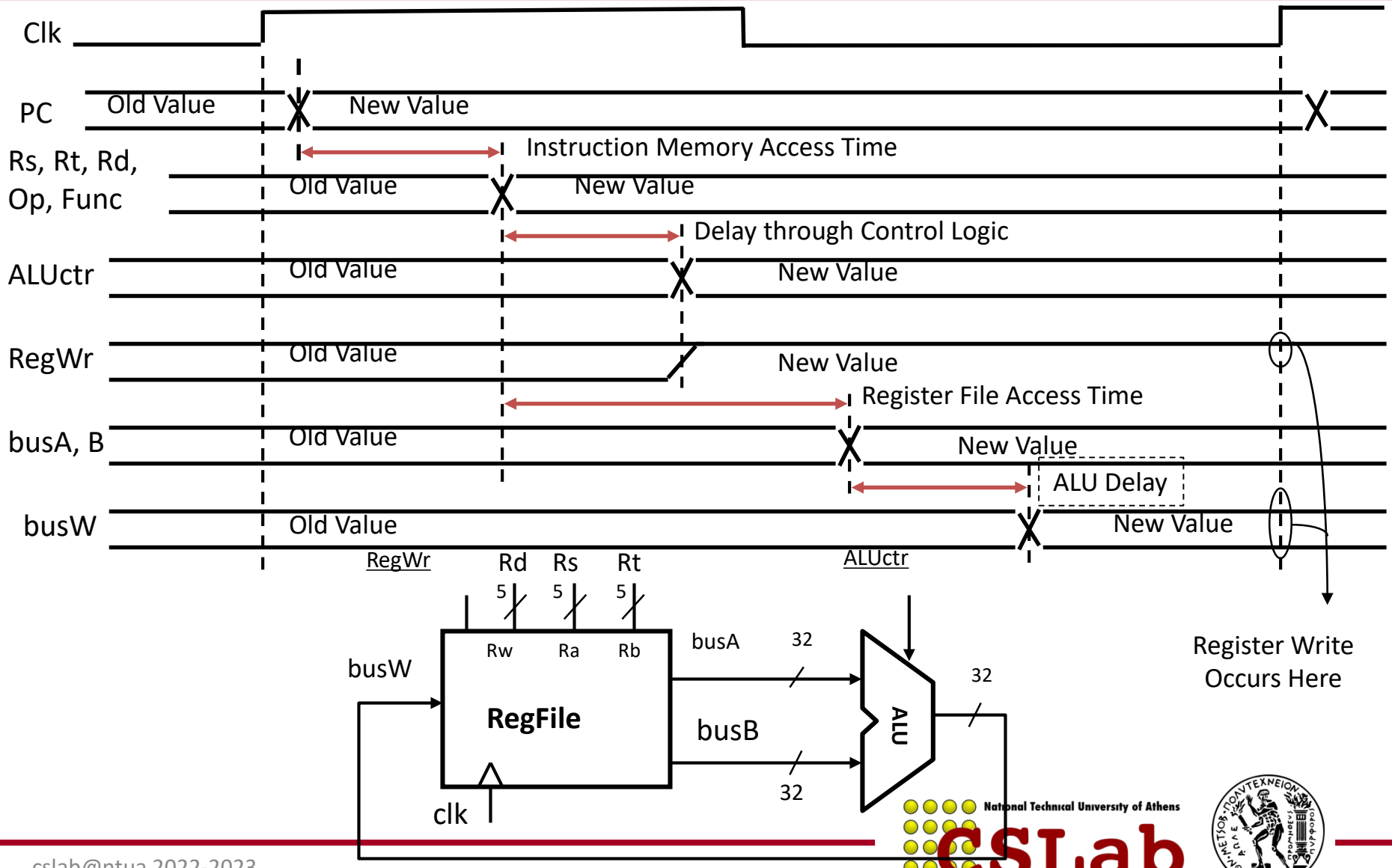
Εντολή Branch-on-Equal



Διαδρομή δεδ. με προσθήκη αλμάτων



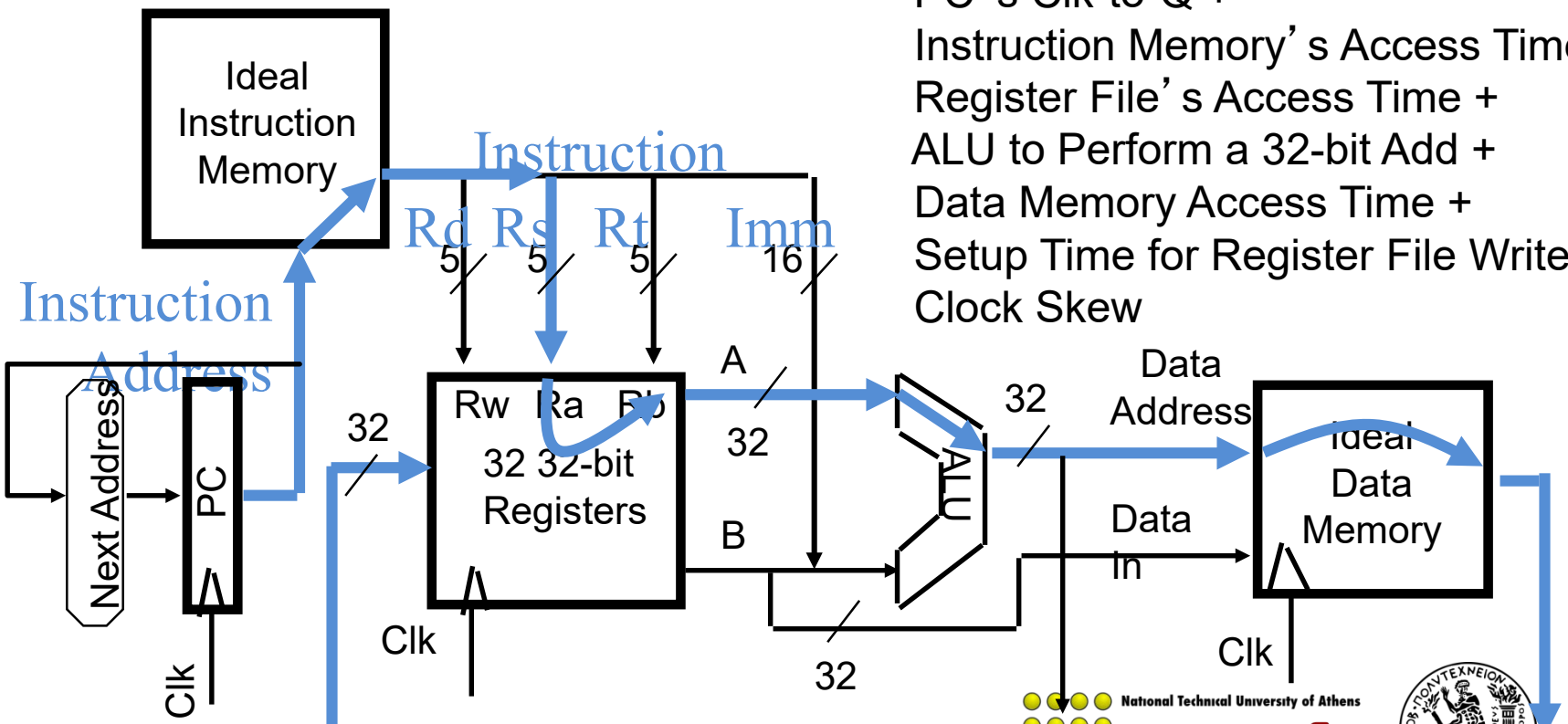
Register-Register Timing: One Complete Cycle



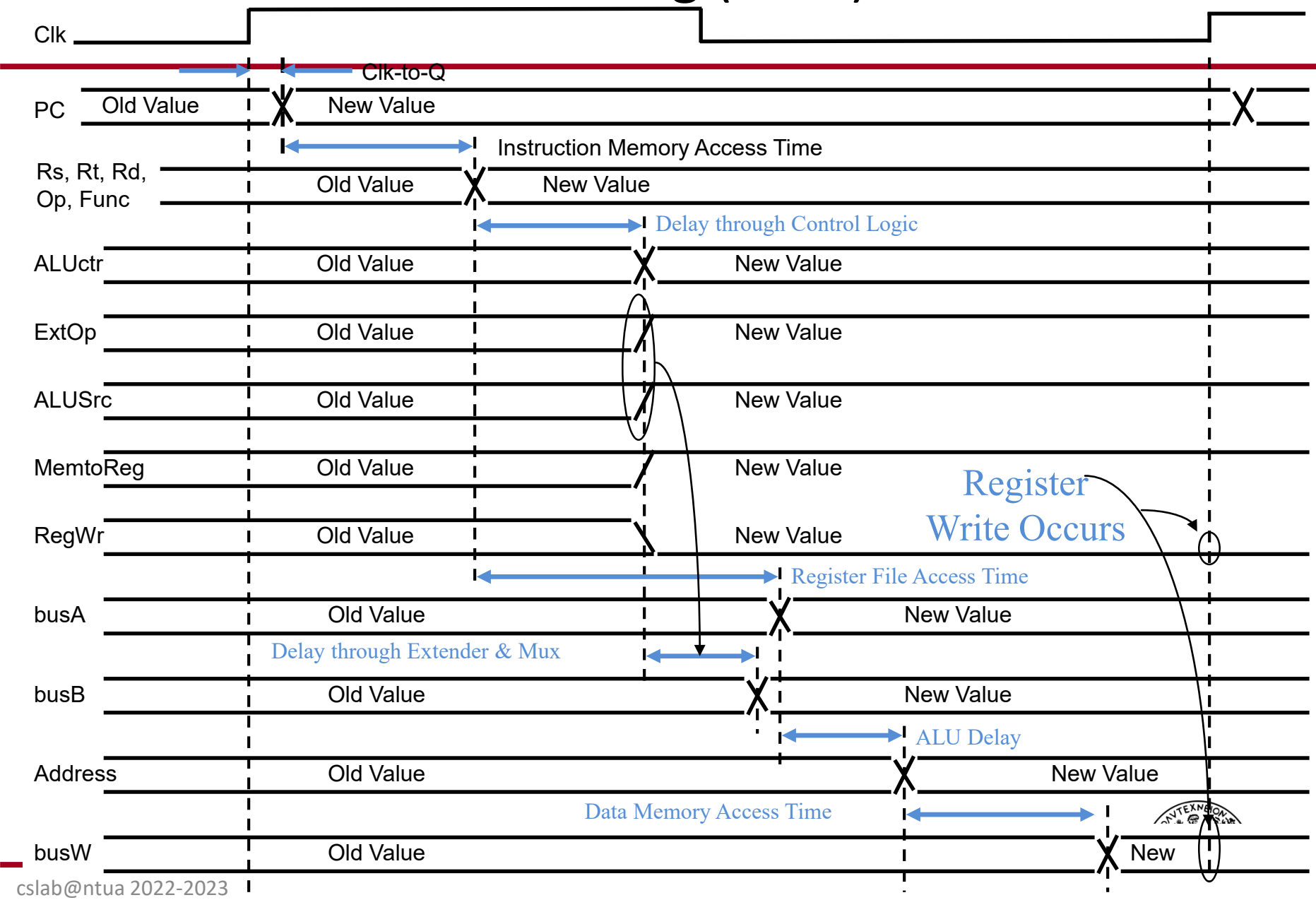
An Abstract View of the Critical Path (Load)

- 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

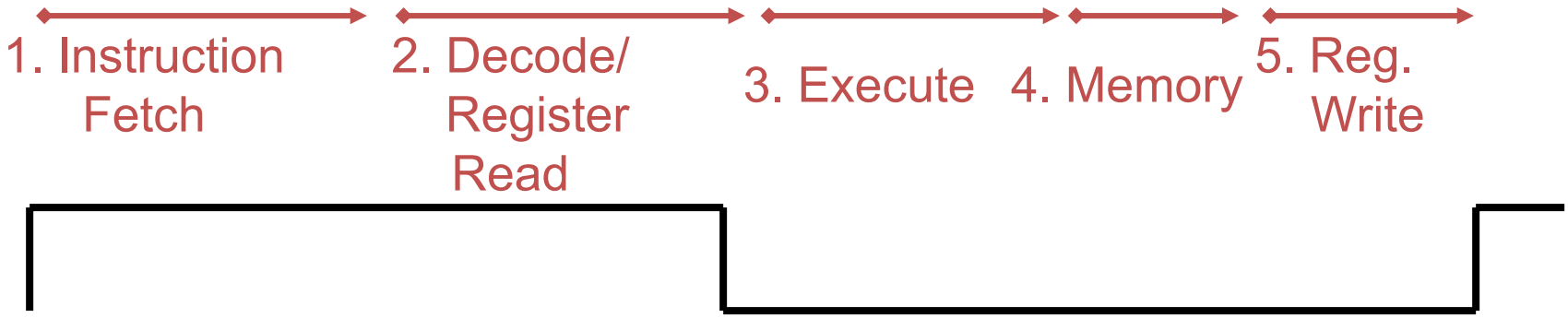


Worst Case Timing (Load)



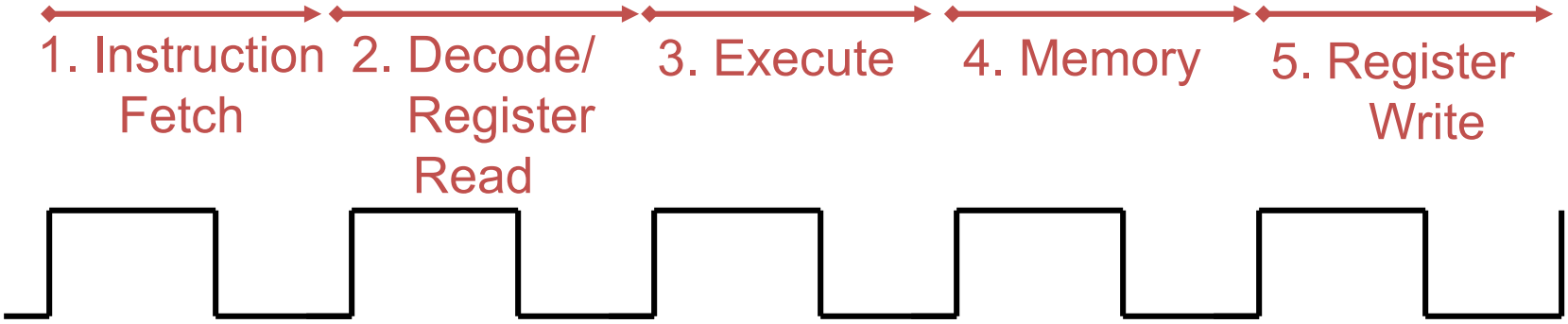
CPU Clocking (1/2)

- For each instruction, how do we control the flow of information though the datapath?
- Single Cycle CPU: All stages of an instruction completed within one long clock cycle
  - Clock cycle sufficiently long to allow each instruction to complete all stages without interruption within one cycle



CPU Clocking (2/2)

- Alternative multiple-cycle CPU: only one stage of instruction per clock cycle
  - Clock is made as long as the slowest stage



  - Several significant advantages over single cycle execution: Unused stages in a particular instruction can be skipped OR instructions can be pipelined (overlapped)

Single-cyle υλοποίηση:

Διάρκεια κύκλου ίση με τη μεγαλύτερη εντολή-worst case delay (εδώ lw)

Αντιβαίνει με αρχή: Κάνε την πιο απλή περίπτωση γρήγορη.

Κάθε functional unit χρησιμοποιείται μια φορά σε κάθε κύκλο ανάγκη για πολλαπλό hardware.

Λύση: Multicycle υλοποίηση

Μικρότεροι κύκλοι ρολογιού, από τις καθυστερήσεις των επιμέρους functional units

control signals in HDL

Σύνοψη Σημάτων Ελέγχου

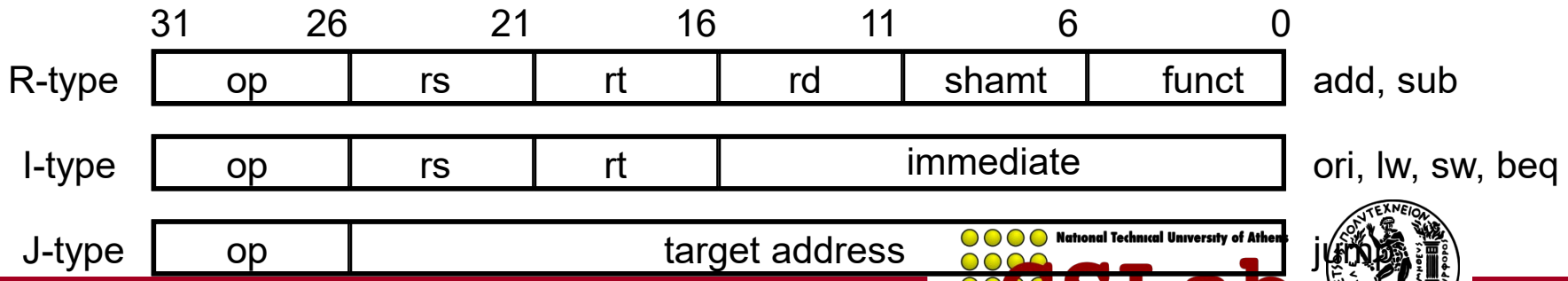
Σήματα που δεν εμφανίζονται είναι 0 («απενεργοποιημένα»)

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+4+sign_ext(Imm16)] || 00} else PC <– PC + 4	
	ALUsrc = RegB, nPC_sel = “Br”, ALUctr = “sub”	

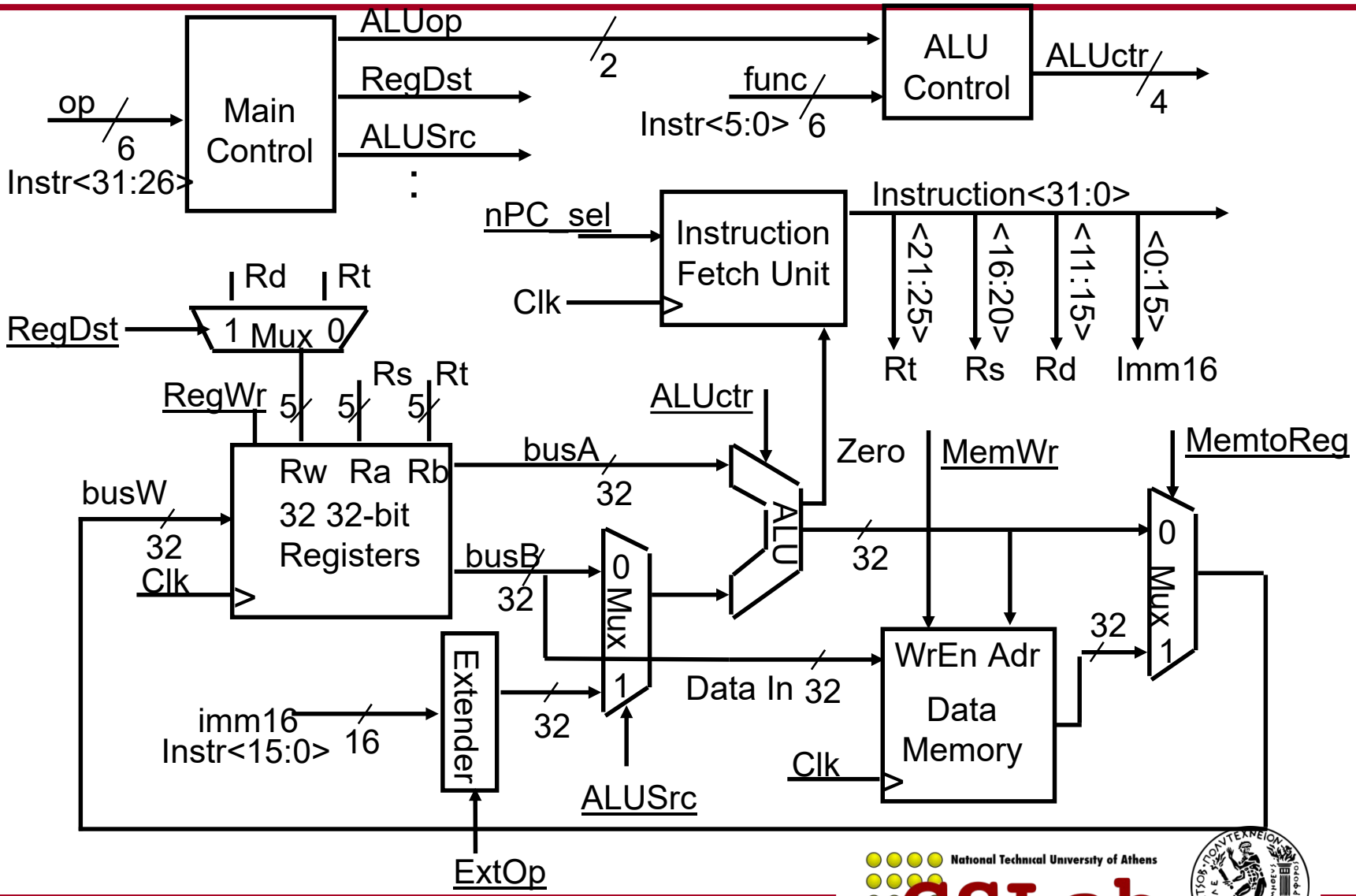
Σύνοψη Σημάτων Ελέγχου

See Appendix A

	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
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



Putting it All Together: A Single Cycle Processor



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: <= _____

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 <= _____
- MemtoReg <= _____
- RegWr: <= _____
- RegDst: <= _____

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 <= _____
- RegWr: <= _____
- RegDst: <= _____

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: <= _____
- RegDst: <= _____

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: <= _____

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