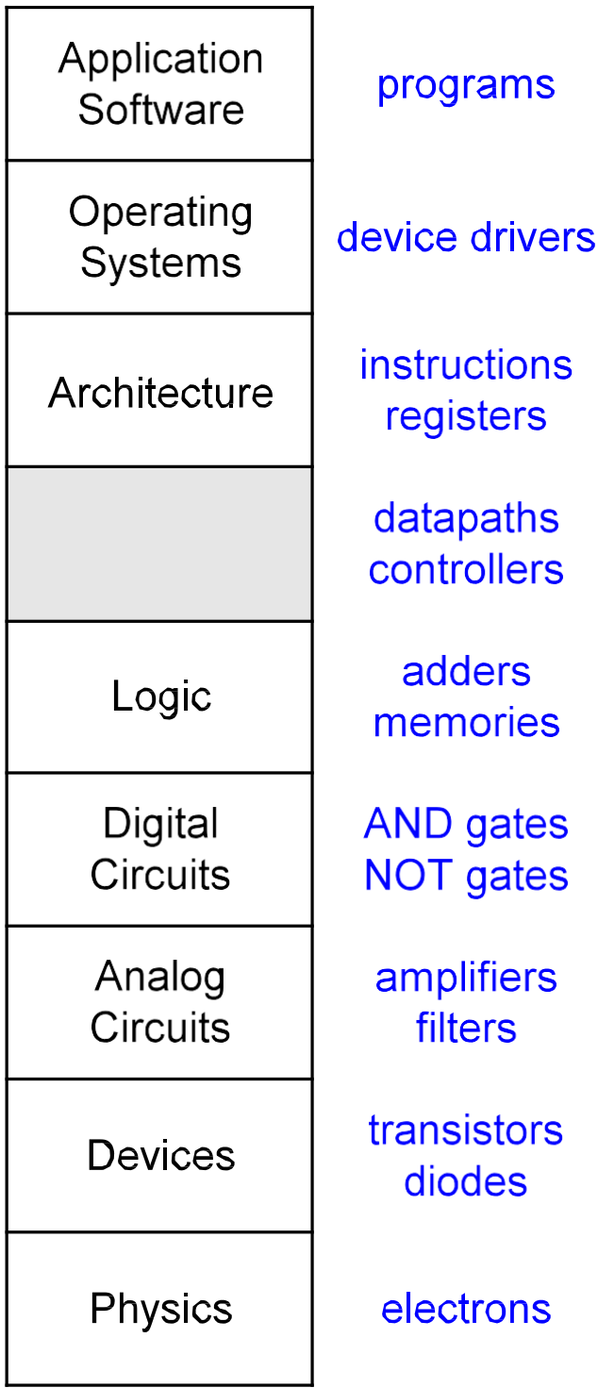
ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

Διάλεξη 9

Επεξεργαστής
Υλοποίηση ενός κύκλου μηχανής

Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Τι είναι Αρχιτεκτονική και τι Μικροαρχιτεκτονική



- **Μικροαρχιτεκτονική:** μία από τις πολλές πιθανές υλοποιήσεις μιας αρχιτεκτονικής

- **Παραδείγματα**:
  - x86 αρχιτεκτονική (Intel),
    - *Intel Core i7, Core 2 Duo* υλοποιήσεις του x86
  - Αρχιτεκτονική MIPS
    - *Μικροαρχιτεκτονική ενός κύκλου μηχανής* του MIPS (κάθε εντολή assembly ολοκληρώνεται σε έναν κύκλο μηχανής)
    - *Μικροαρχιτεκτονική πολλαπλών κύκλων μηχανής* του MIPS (κάθε εντολή assembly ολοκληρώνεται σε πολλαπλούς κύκλους μηχανής)
    - *Μικροαρχιτεκτονική διοχέτευσης (pipeline)* του MIPS (κάθε εντολή assembly ολοκληρώνεται πολλαπλούς κύκλους μηχανής, και πολλαπλές εντολές εκτελούνται ταυτόχρονα)

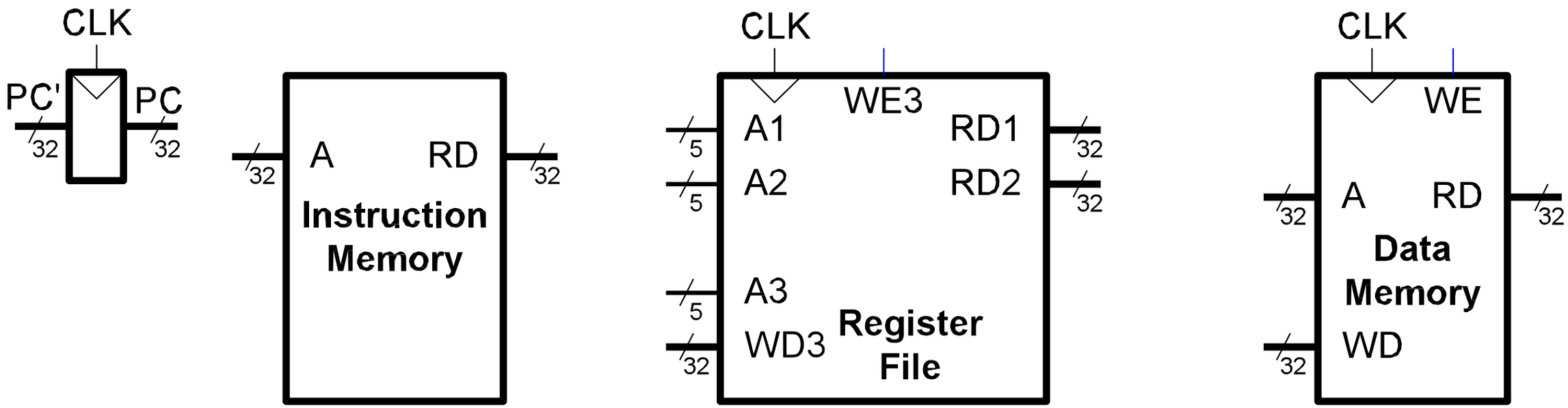
Υλοποίηση ενός υποσυνόλου εντολών

- Στις επόμενες διαφάνειες θα δείξουμε την σταδιακή υλοποίηση μερικών βασικών εντολών του MIPS
  - Εντολές format-R : **`and, or, add, sub, slt`**
  - Εντολές προσπέλασης μνήμης: **`lw, sw`**
  - Εντολές διακλάδωσης: **`beq`**
  - Extra εντολές: **`addi, j`**

- Υλοποίηση σε έναν κύκλο μηχανής

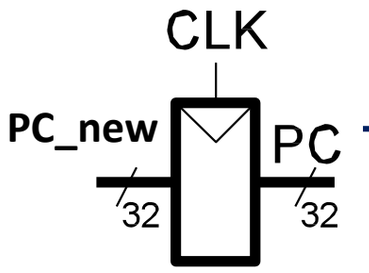
Αρχιτεκτονικά στοιχεία του MIPS

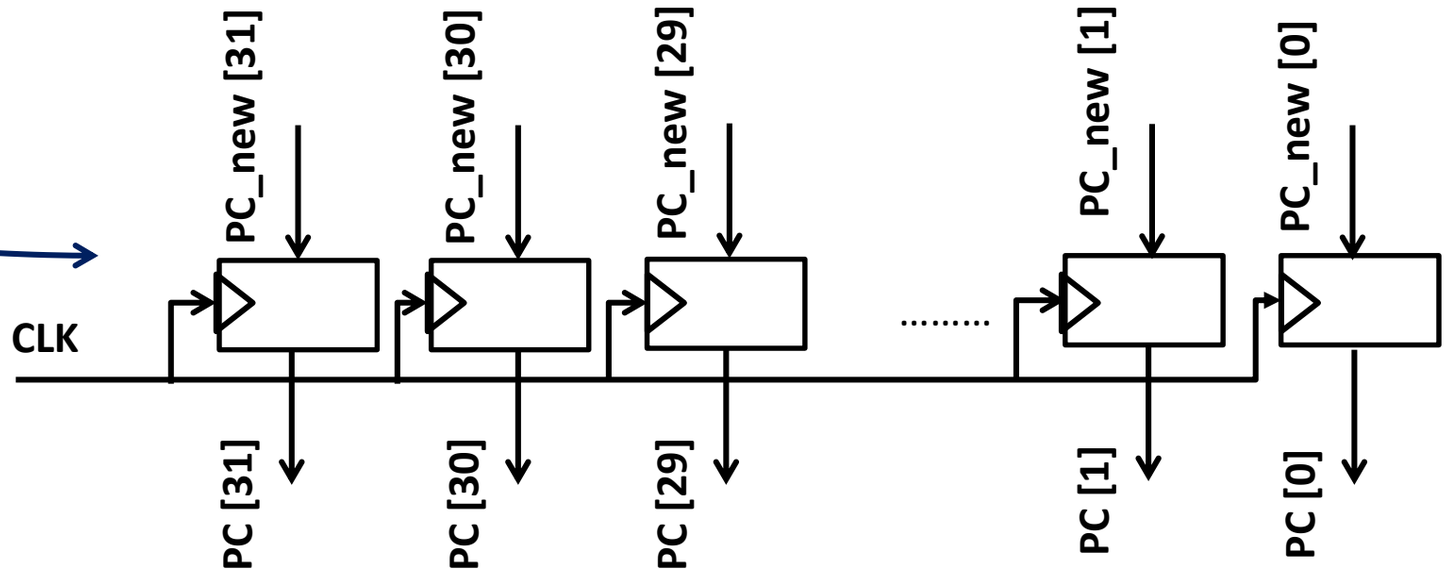
- Τα παρακάτω στοιχεία μνήμης, αποτελούν την Αρχιτεκτονική κατάσταση του επεξεργαστή MIPS
- Με άλλα λόγια, τα περιεχόμενά τους είναι ορατά στον προγραμματιστή



Αρχιτεκτονικά στοιχεία του MIPS

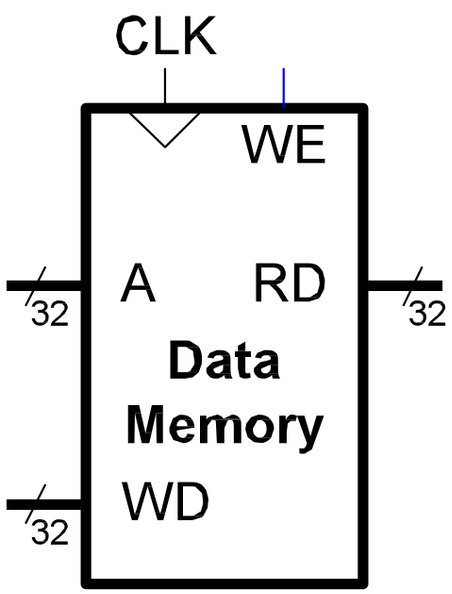
- Για να τα δούμε με λεπτομέρεια:






```
always @(posedge CLK or
         negedge reset)

    if (reset == 1'b0)
       PC <= 0;
    else
        PC <= PC_new
```




```
Memory (input ren, wen,
        input [31:0] addr, din,
        output [31:0] dout);  // 4k words mem
input ren, wen;
input  [31:0] addr, din;
output [31:0] dout;
reg [31:0] data[4095:0];
wire [31:0] dout;
    assign dout = ((wen==1'b0) && (ren==1'b1)) ?
                    data[addr[11:0]] : 32'bx;
    always @(din or wen or ren or addr)
       begin
           if ((wen == 1'b1) && (ren==1'b0))
           data[addr[11:0]] = din;
       end
```

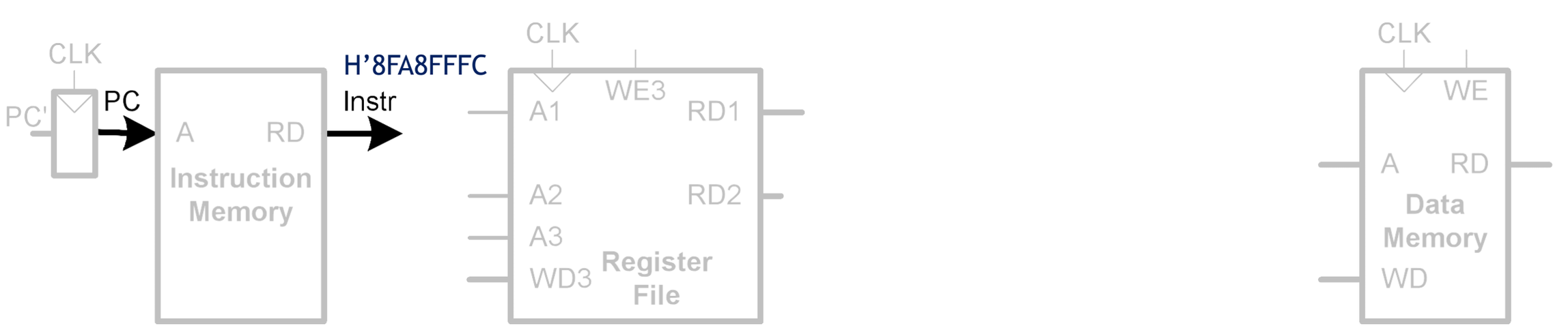
Υλοποίηση της πρώτης μας εντολής: *lw*

Βήμα 1: *Instruction Fetch*

Το *Fetch* γίνεται για κάθε εντολή

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits
31..26	25..21	20..16	15..0
100011	11101	01000	1111 1111 1111 1100

lw $t0, –4($sp)

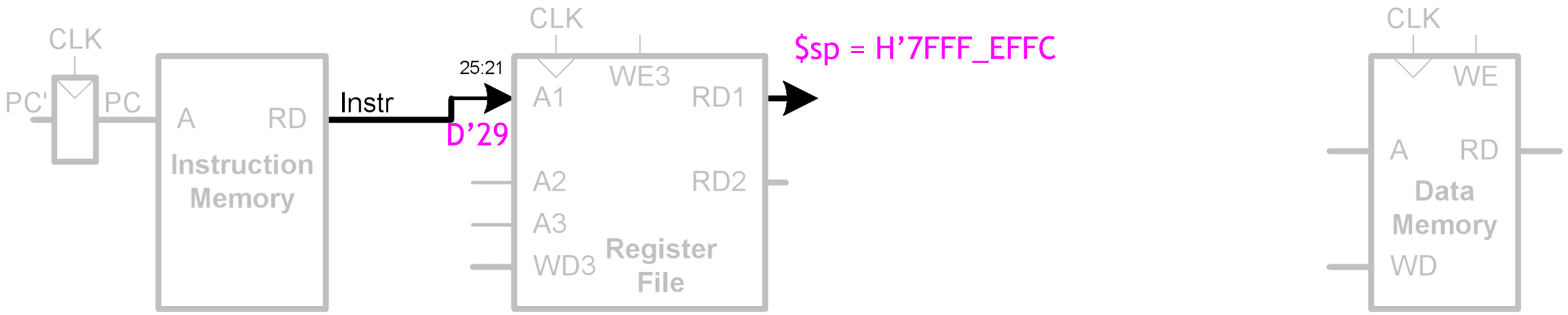


Υλοποίηση της πρώτης μας εντολής: *lw*

Βήμα 2: *Instruction Decode*
Διάβασε τους καταχωρητές εισόδου από το register file.

31..26	25..21	20..16	15..0
100011	11101	01000	1111 1111 1111 1100

lw $t0, –4($sp)

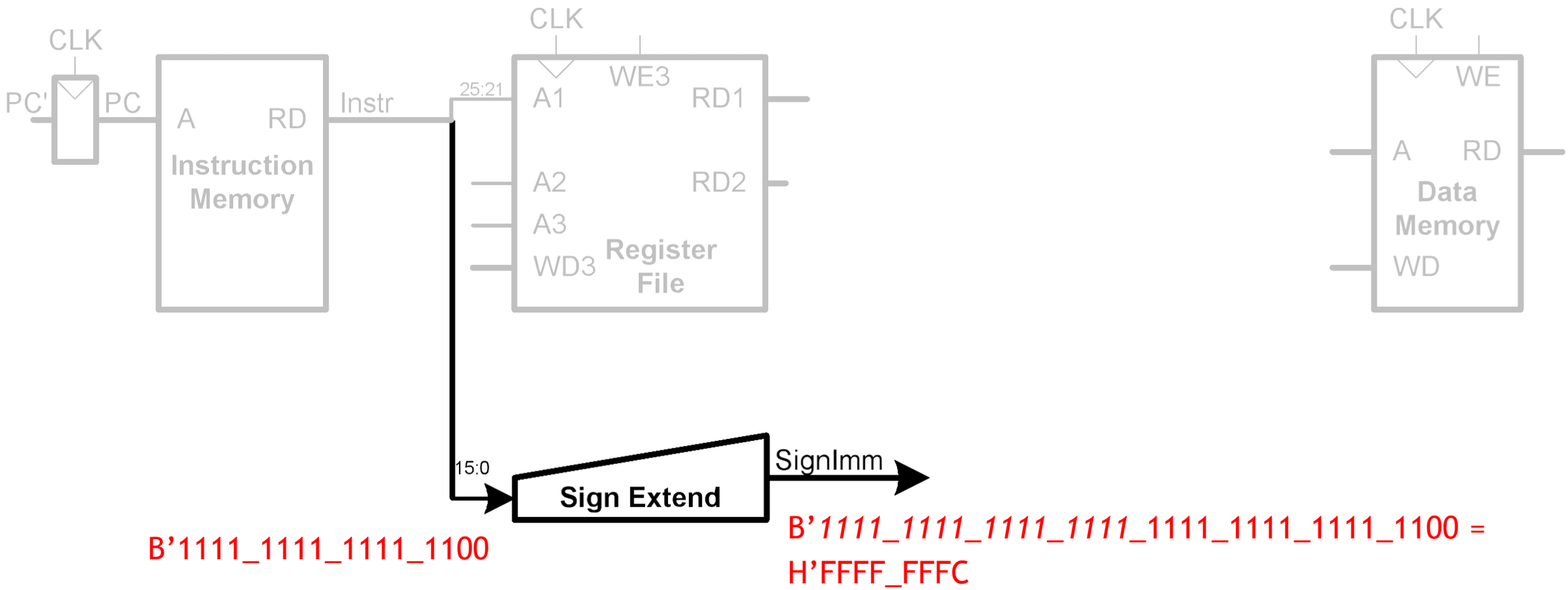


Υλοποίηση της πρώτης μας εντολής: *lw*

Βήμα 3: *Sign Extension*
Επέκταση πρόσημου της σταθεράς

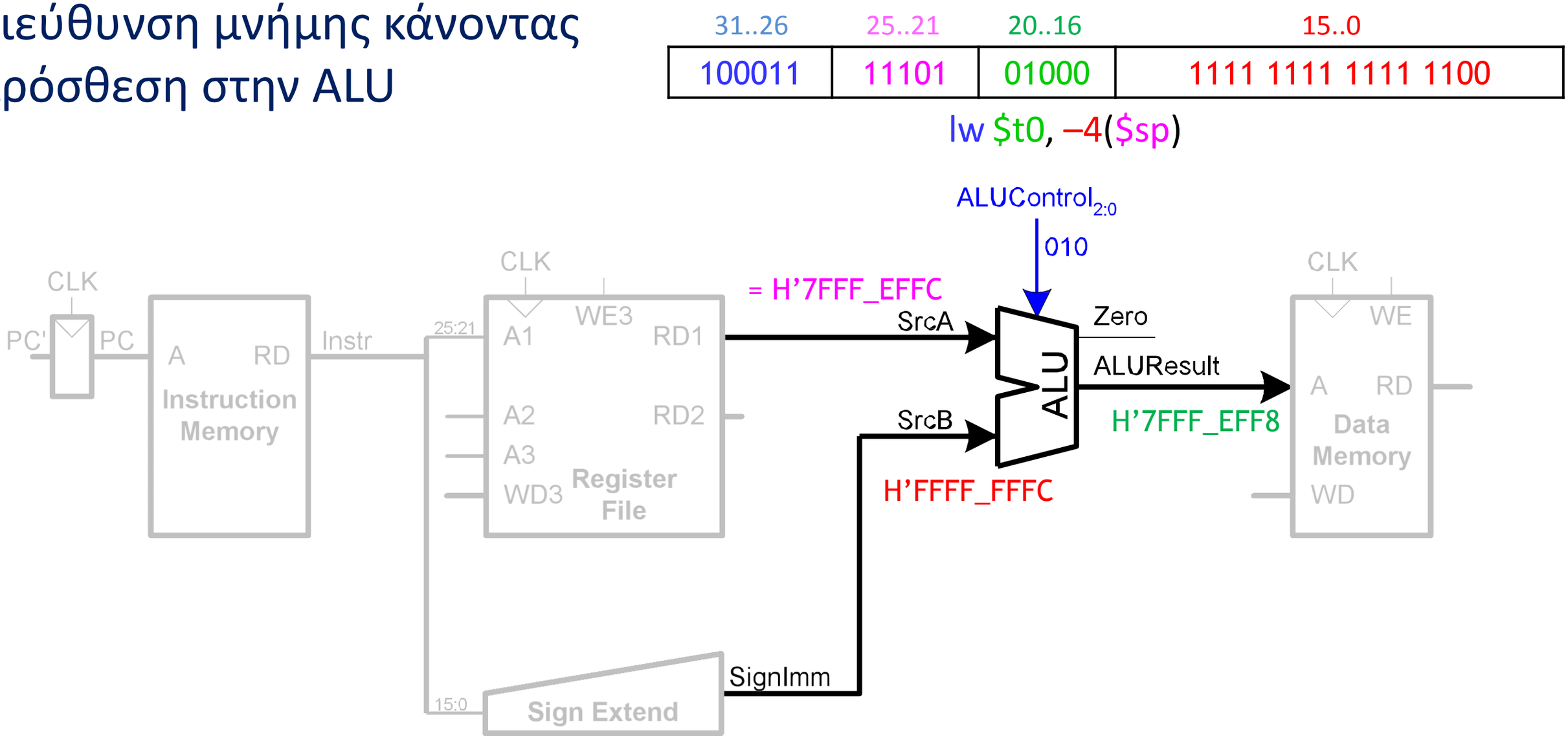
31..26	25..21	20..16	15..0
100011	11101	01000	1111 1111 1111 1100

lw $t0, –4($sp)



Υλοποίηση της πρώτης μας εντολής: *lw*

Βήμα 4: Υπολόγισε την διεύθυνση μνήμης κάνοντας πρόσθεση στην ALU

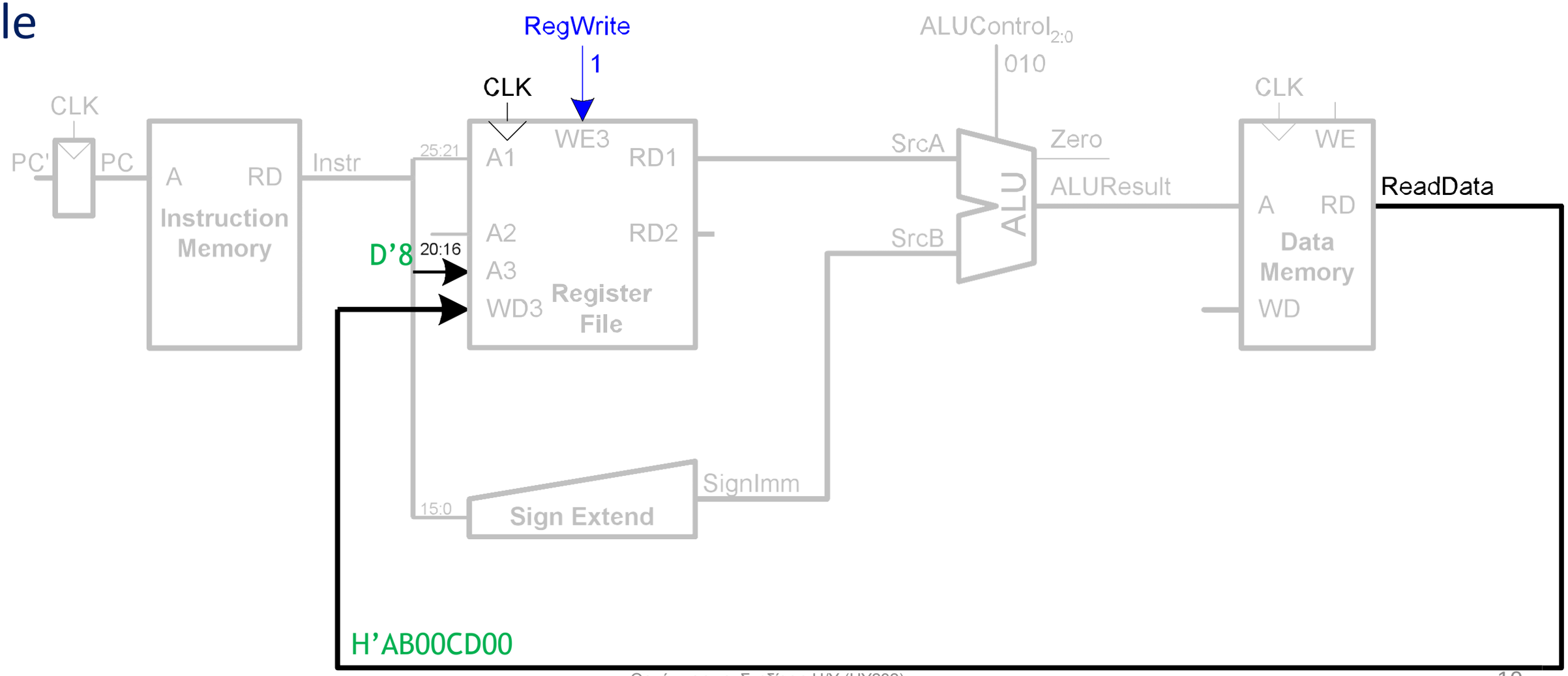


Υλοποίηση της πρώτης μας εντολής: *lw*

Βήμα 5: Διάβασε δεδομένα από την μνήμη χρησιμοποιώντας την διεύθυνση που μόλις υπολόγισες.
Γράψε τα δεδομένα στο register file

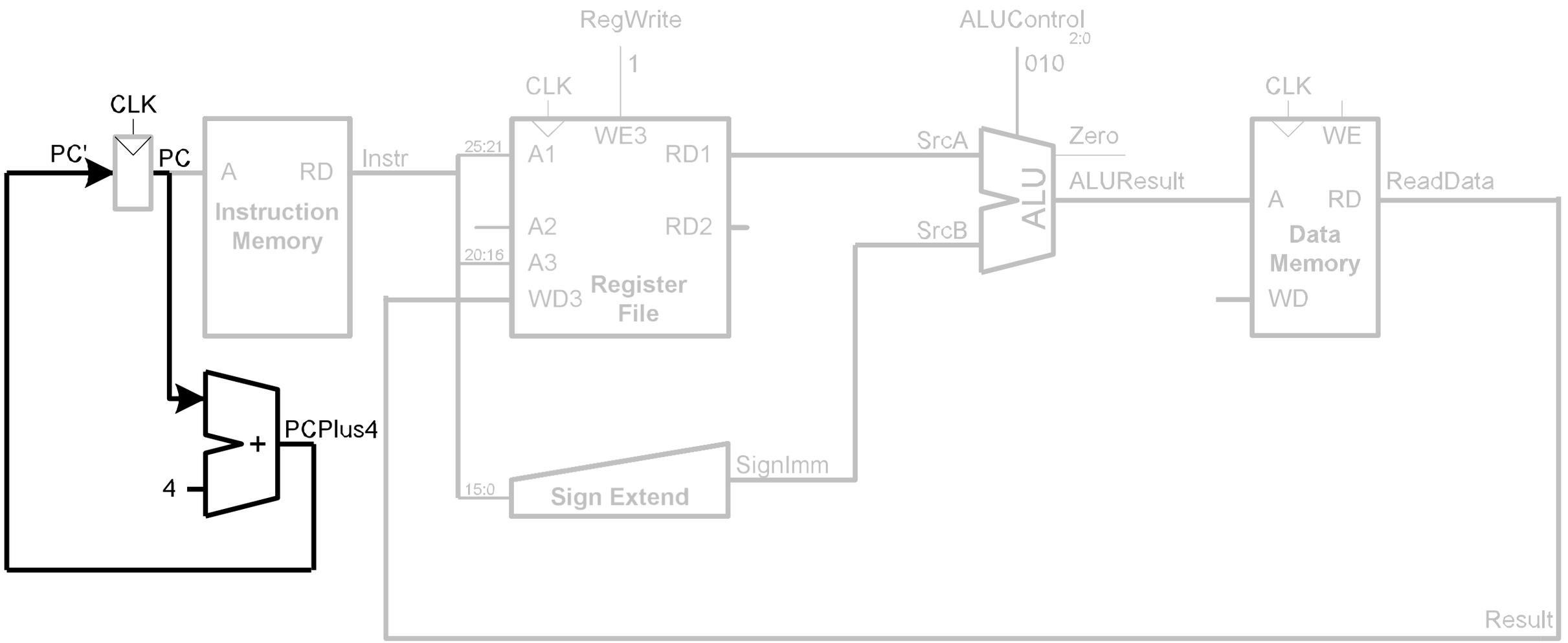
31..26	25..21	20..16	15..0
100011	11101	01000	1111 1111 1111 1100

lw $t0, –4($sp)



Υλοποίηση της πρώτης μας εντολής: *lw*

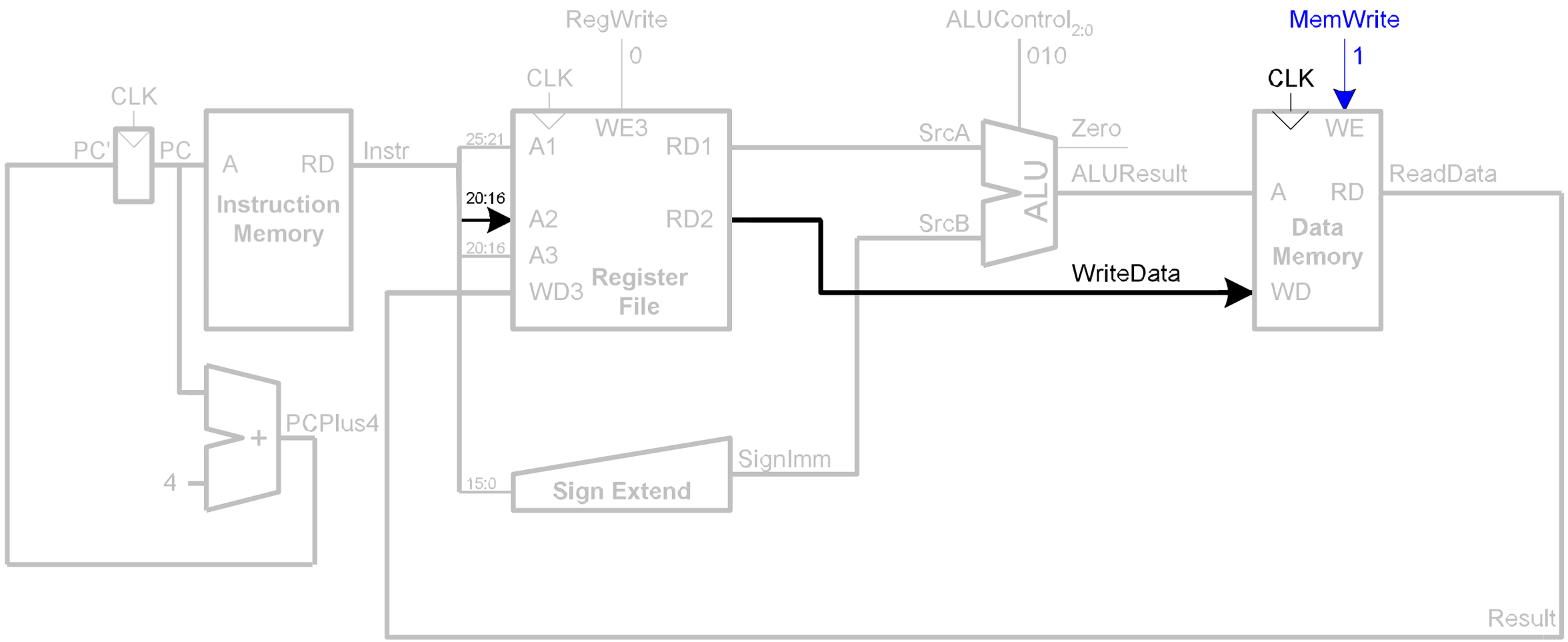
Βήμα 6: Βρες την διεύθυνση της επόμενης εντολής PC <= PC+4



Υλοποίηση της εντολής *sw*

Με λίγο extra hardware, μπορούμε να υλοποιήσουμε και την sw

```
sw $t0, -4($sp)
```

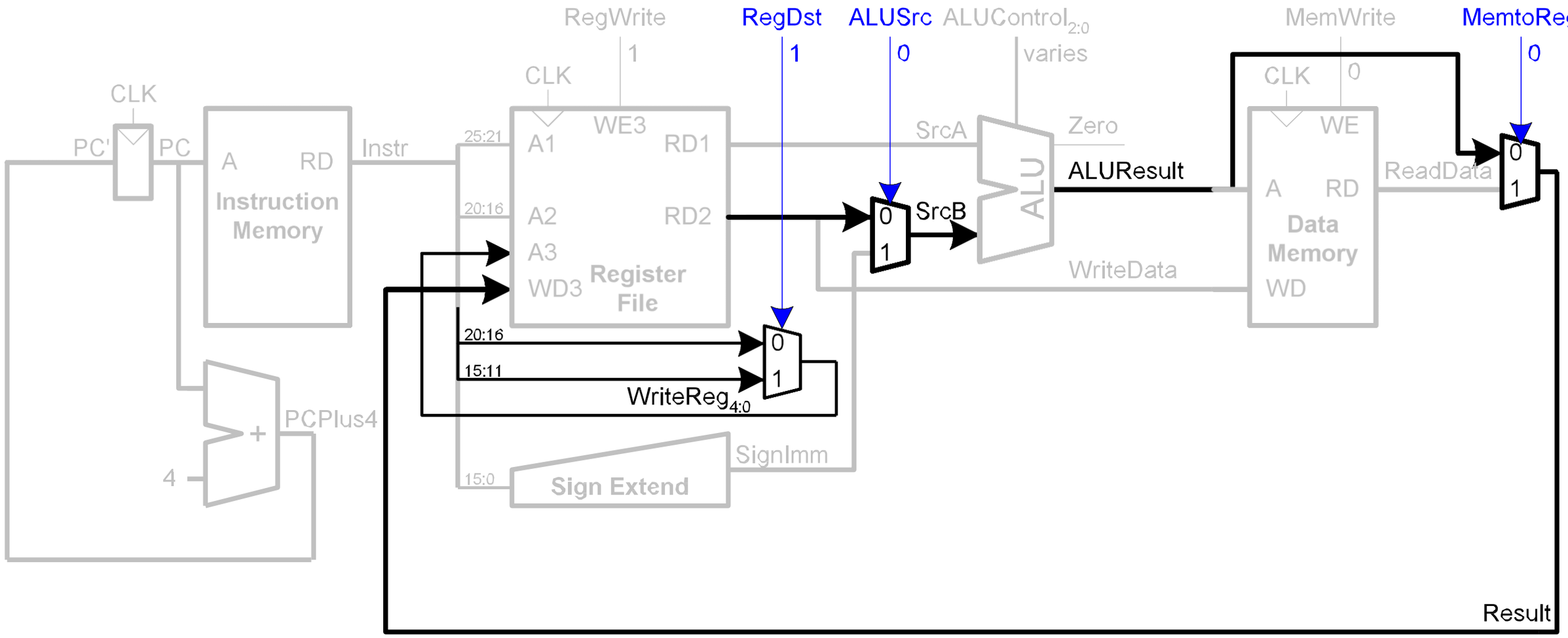


Υλοποίηση εντολών *R-format*

Διάβασε τους καταχωρητές `rs` και `rt` από το register file.
Κάνε τον υπολογισμό στην ALU.
Γράψε το αποτέλεσμα στον `rd`

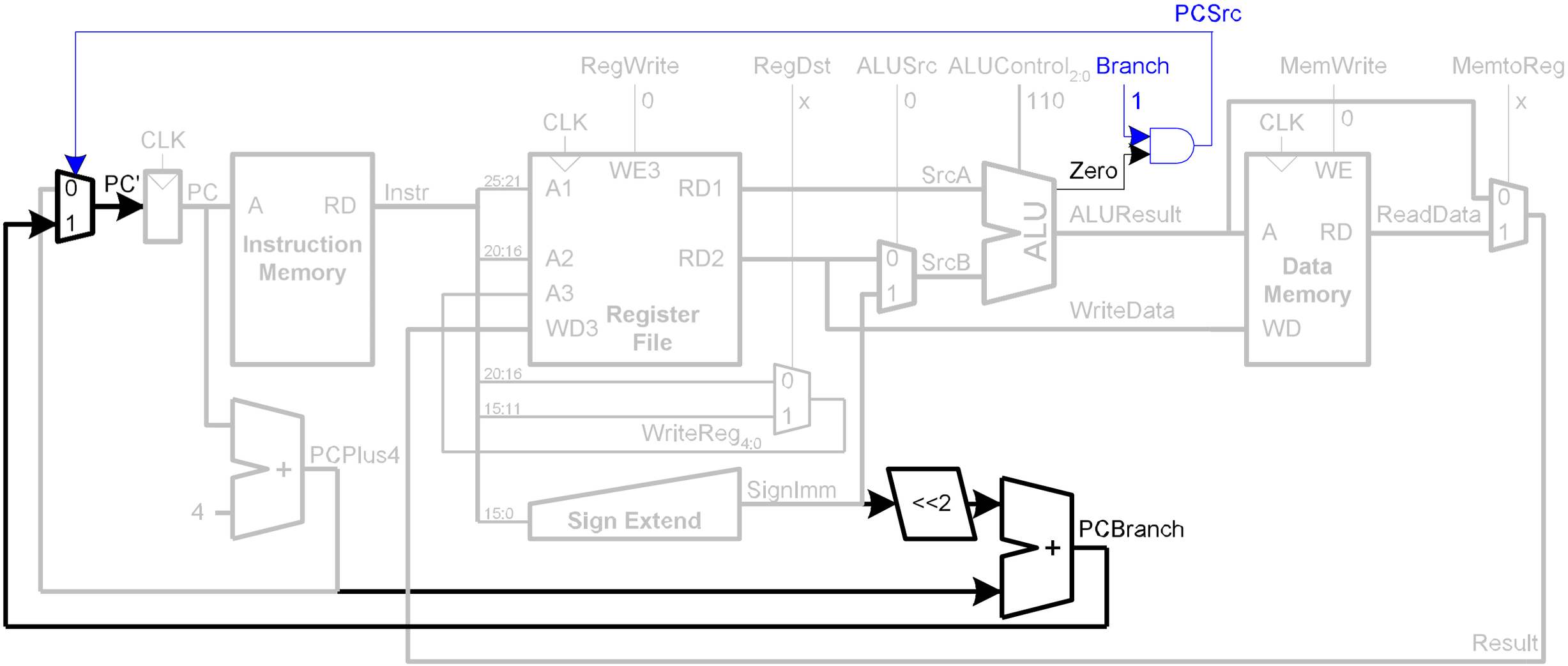
op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

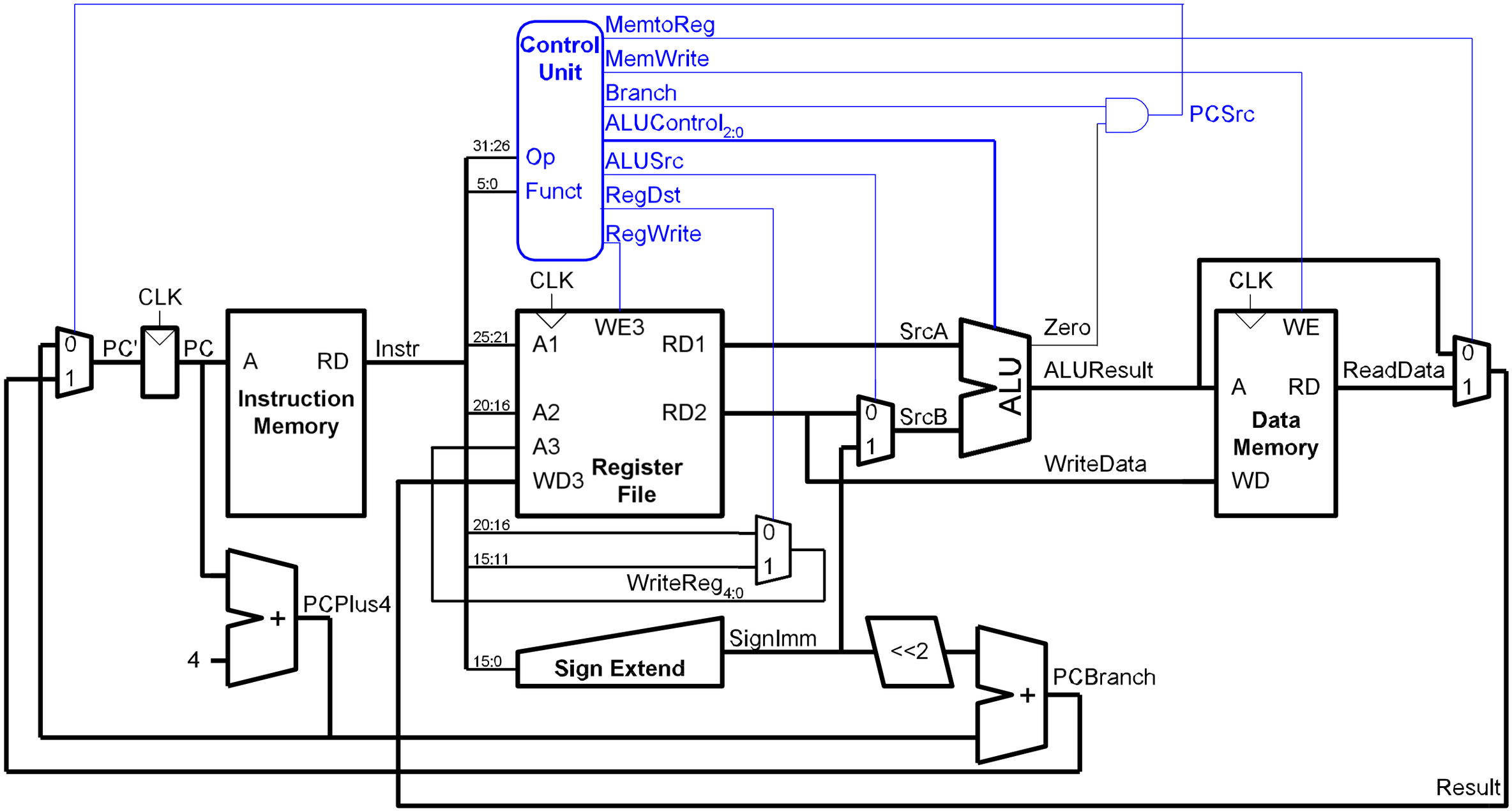


Υλοποίηση εντολής *beq*

- Διάβασε τους καταχωρητές `rs` και `rt` και έλεγξε εάν οι τιμές τους είναι ίδιες.
- Υπολόγισε την διεύθυνση Δ=(PC+4) + 4*(sign extended immediate)
- Εάν `rs` ==`rt`, θέσε PC <= Δ, αλλιώς PC <= PC+4

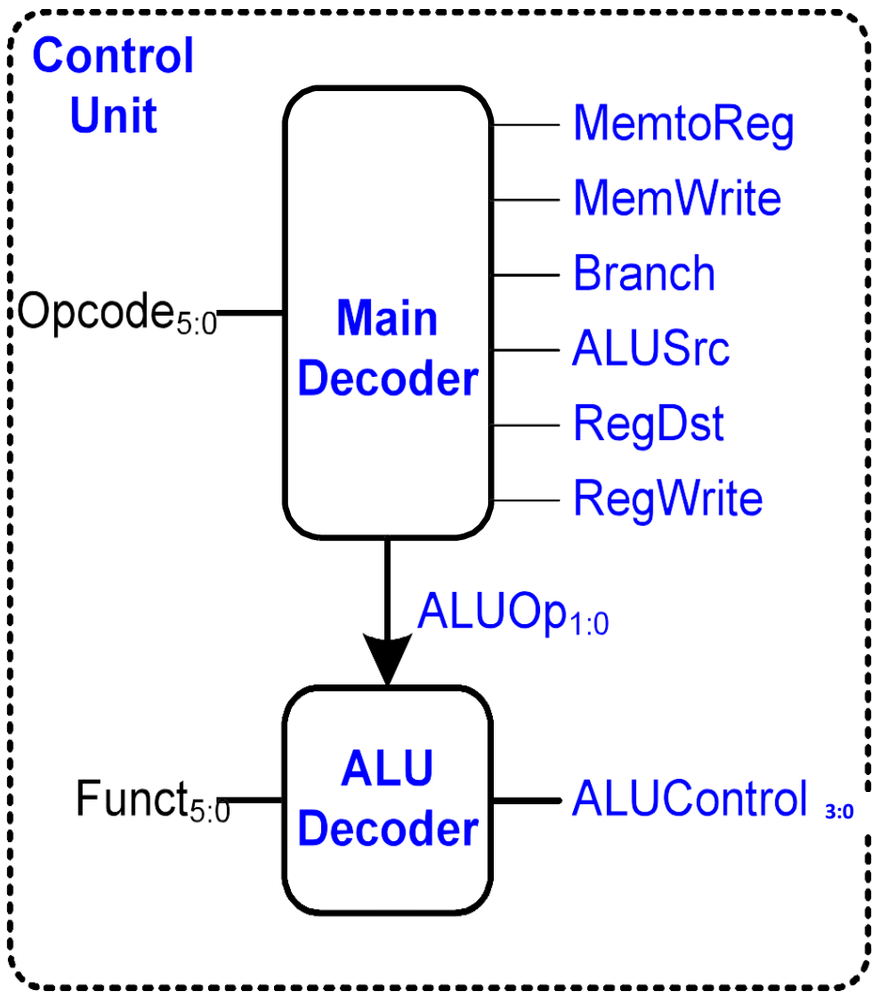


Υλοποίηση του επεξεργαστή ενός κύκλου



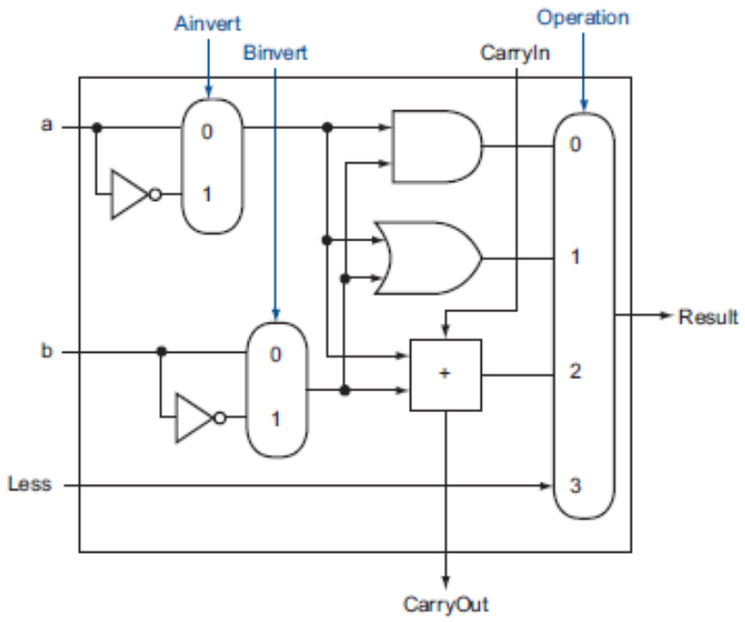
Μονάδα Ελέγχου (Control Unit)

- ALU Decoder ως ξεχωριστό κομμάτι της μονάδας ελέγχου. Με αυτόν τον τρόπο διαχωρίζουμε τα δύο πεδία της εντολής *opcode[5:0]* και *funct[5:0]*

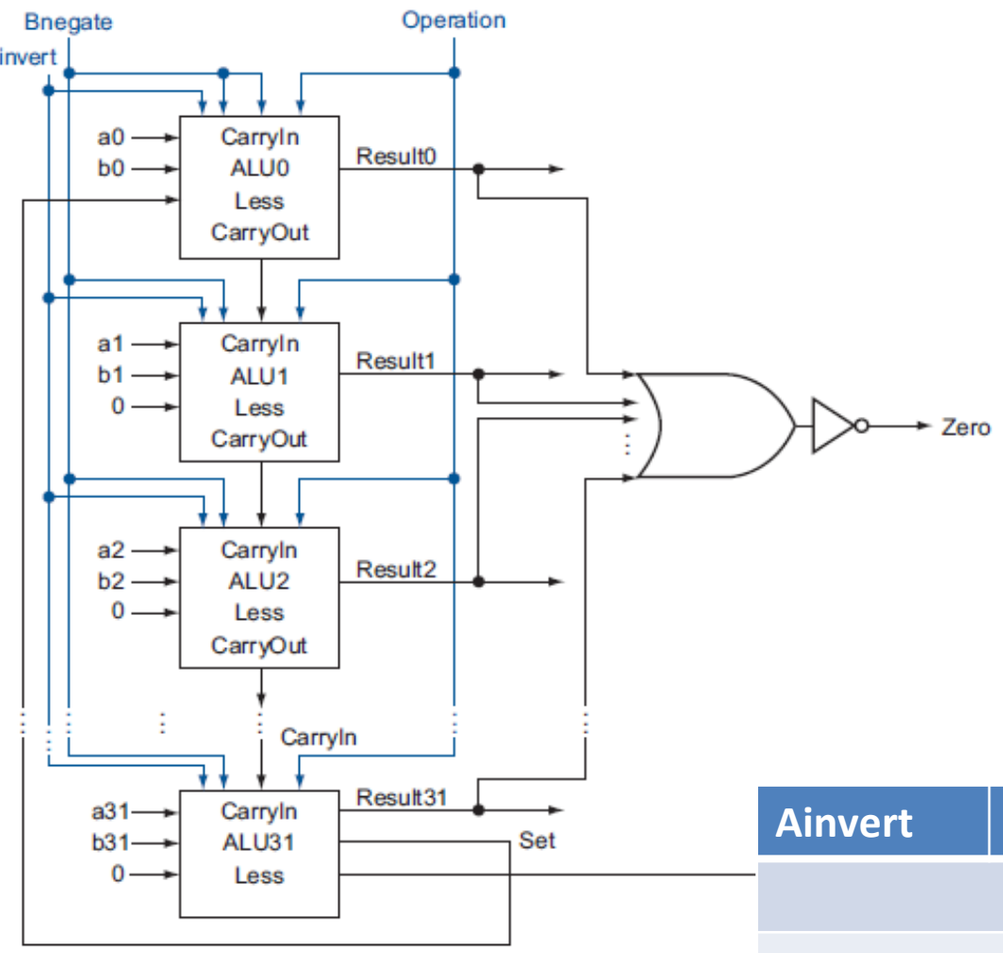


ALU και Μονάδα Ελέγχου

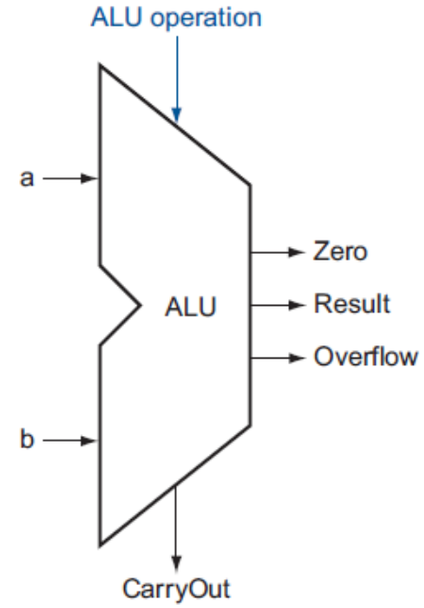
Για να θυμηθούμε την ALU



1-bit ALU
(and, or, add, sub, nor, slt, beq)
(4 control signals)

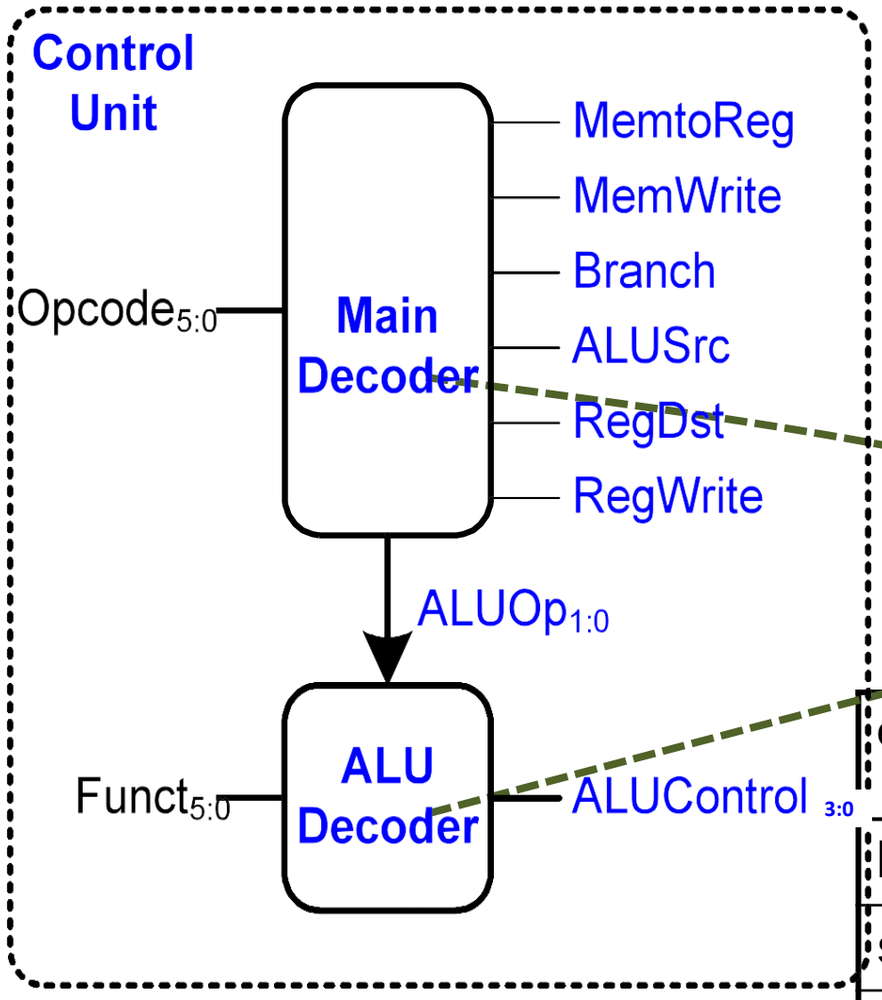


32-bit ALU
(and, or, add, sub, nor, slt, beq)



Ainvert	Bnegate	Operation	Function
ALU Control			
0	0	00	AND (A&B)
0	0	01	OR (A|B)
0	0	10	ADD (A+B)
0	1	10	SUB (A-B)
0	1	11	SLT
1	1	00	NOR (A|B)’

ALU και Μονάδα Ελέγχου

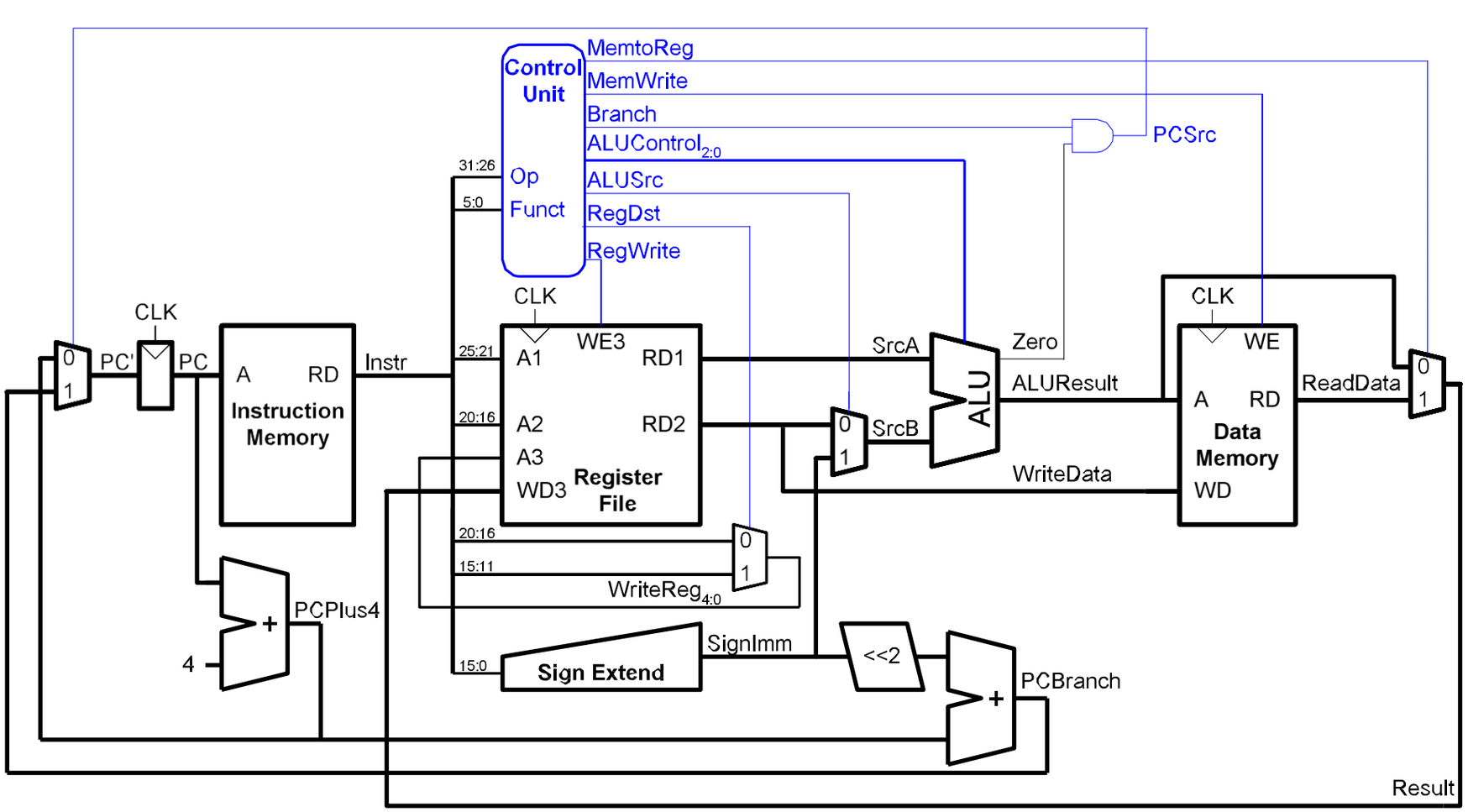


- Για μία συγκεκριμένη εντολή η μονάδα ελέγχου παράγει τα 4 σήματα ελέγχου ALUControl[3:0]

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
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

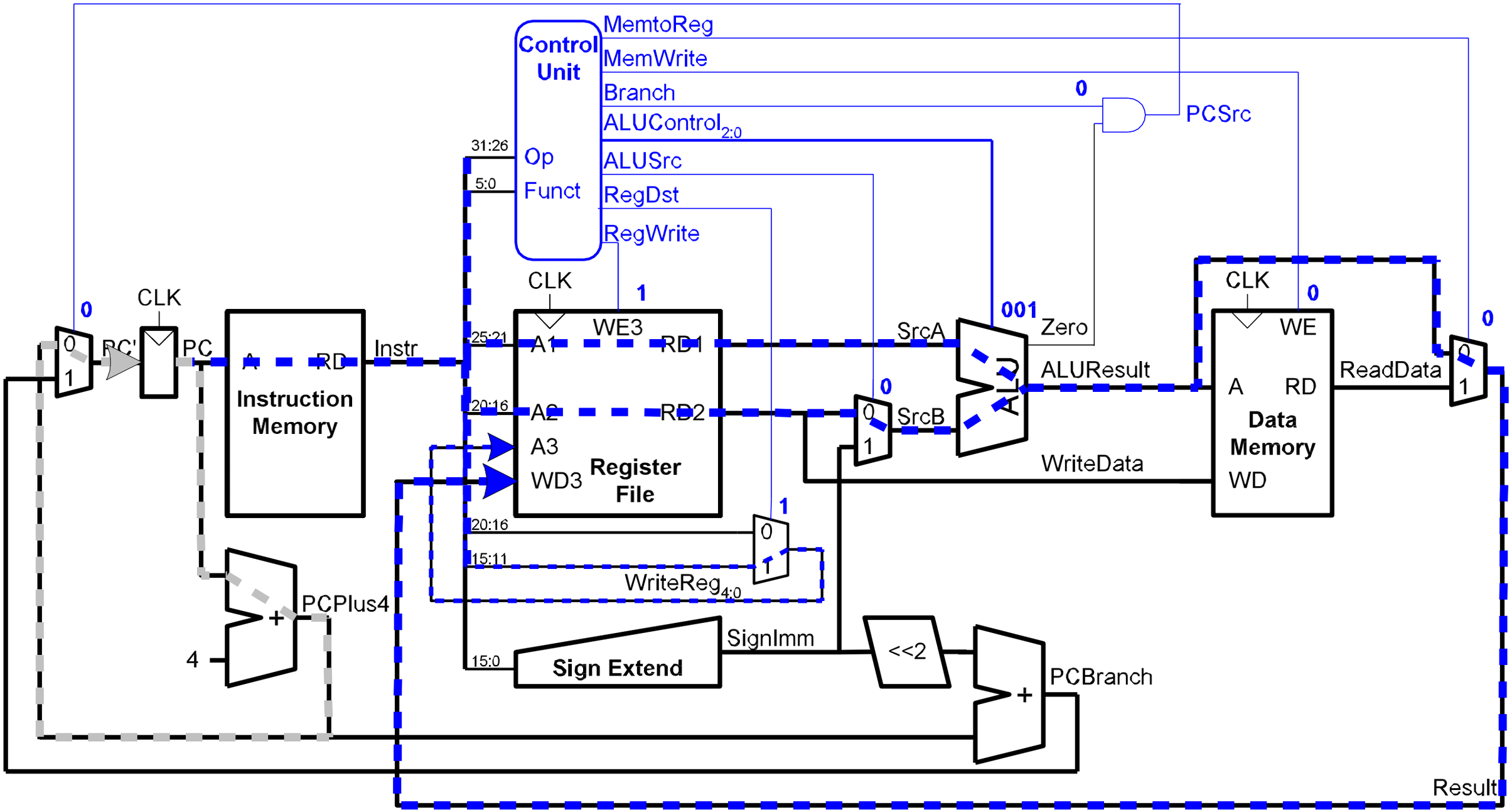
Κύρια Μονάδα Ελέγχου

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01



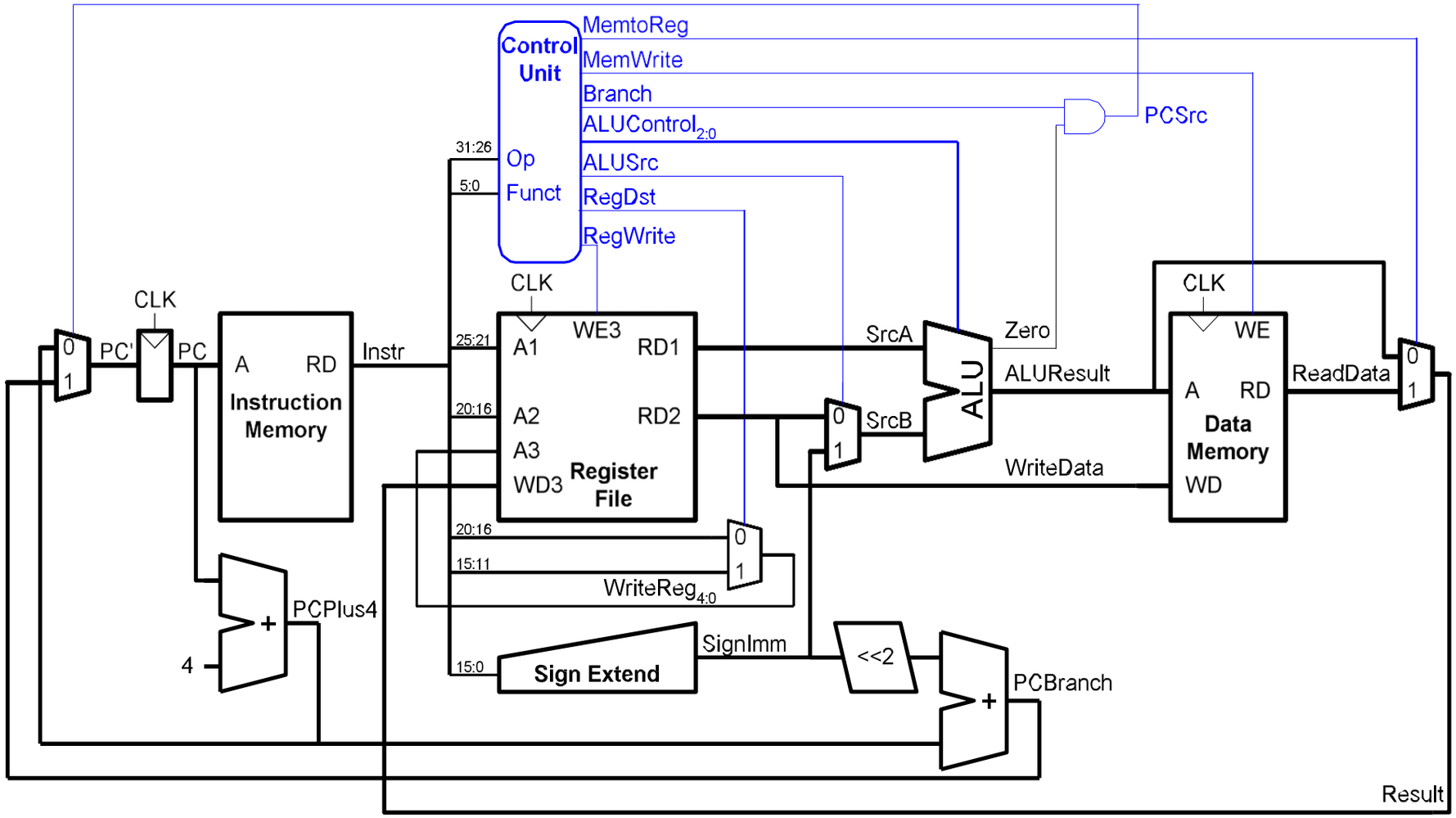
Υλοποίηση εντολής *or*

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



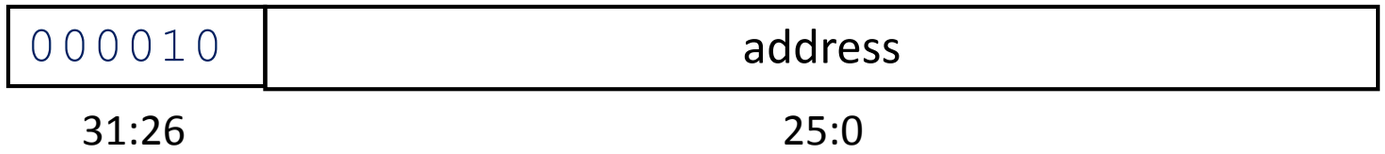
Υλοποίηση εντολής `addi`

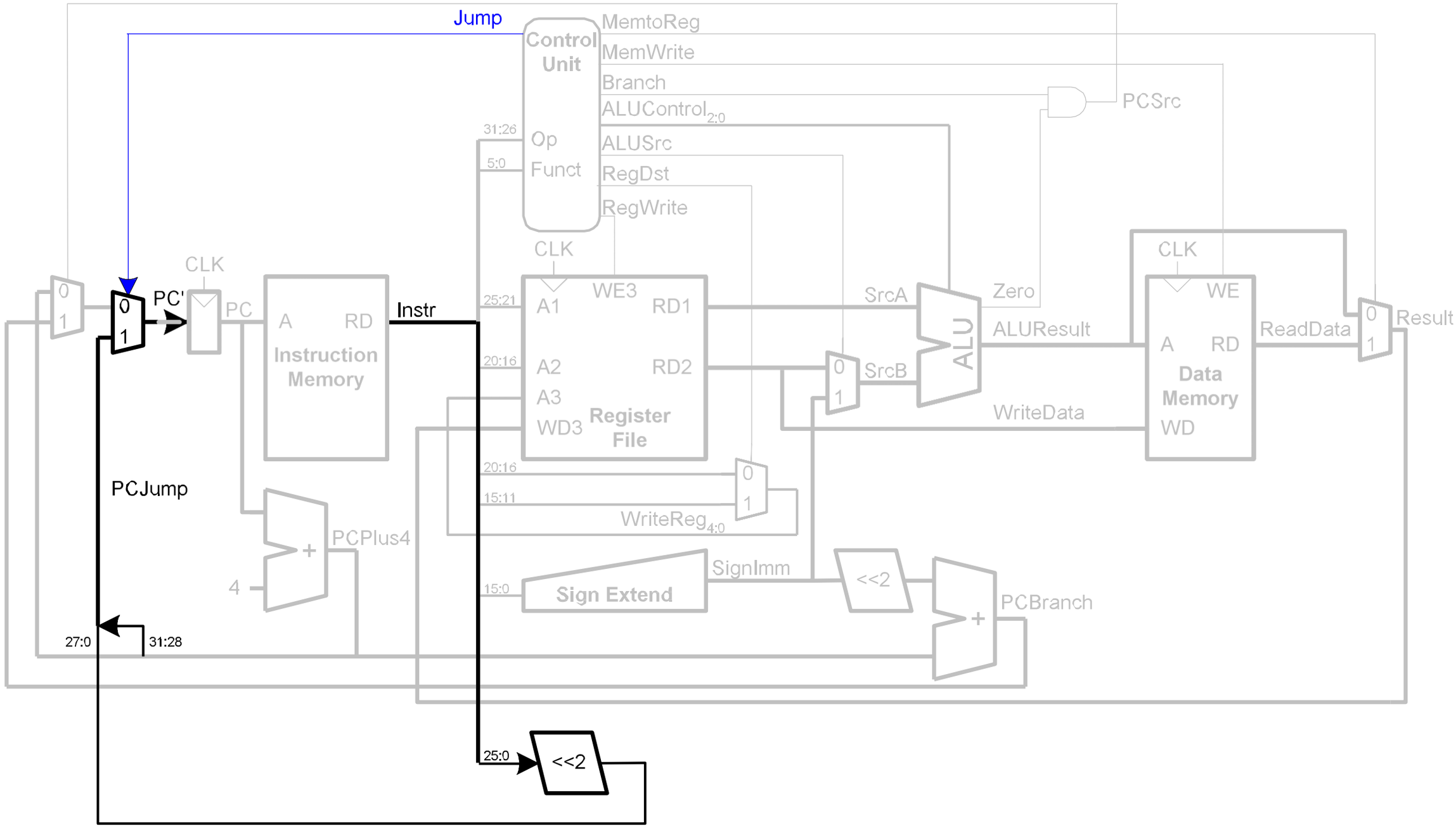
- Δεν απαιτείται αλλαγή στο Data Path
- Πρέπει όμως να επεκταθεί η μονάδα ελέγχου



Instruction	Op5:0	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp1:0
R-type	0 0 0 0 0 0	1	1	0	0	0	0	1 0
lw	1 0 0 0 1 1	1	0	1	0	0	1	0 0
sw	1 0 1 0 1 1	0	x	1	0	1	x	0 0
beq	0 0 0 1 0 0	0	x	0	1	0	x	0 1
addi	0 0 1 0 0 0	1	0	1	0	0	0	0 0

Υλοποίηση εντολής `j`





Υλοποίηση εντολής j

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$	Jump
R-type	0 0 0 0 0 0	1	1	0	0	0	0	1 0	0
lw	1 0 0 0 1 1	1	0	1	0	0	1	0 0	0
sw	1 0 1 0 1 1	0	x	1	0	1	x	0 0	0
beq	0 0 0 1 0 0	0	x	0	1	0	x	0 1	0
addi	0 0 1 0 0 0	1	0	1	0	0	0	0 0	0
j	0 0 0 0 1 0	0	x	x	x	0	x	x x	1

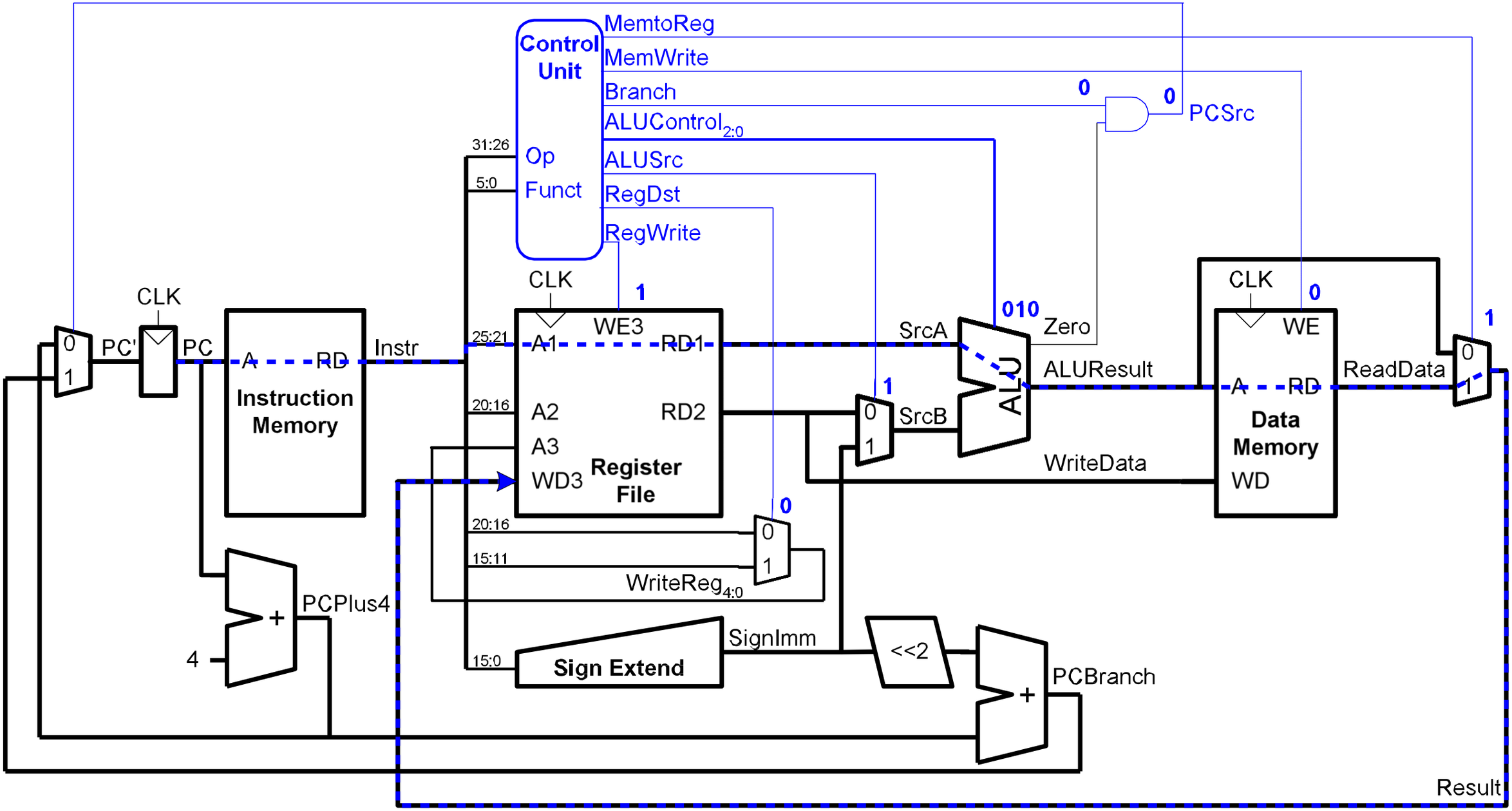
Η Απόδοση του υπολογιστή μας

$$\text{Χρόνος CPU} = \text{Αριθμός Εντολών στο Πρόγραμμα} \times \text{CPI} \times \text{Περίοδος Ρολογιού}$$

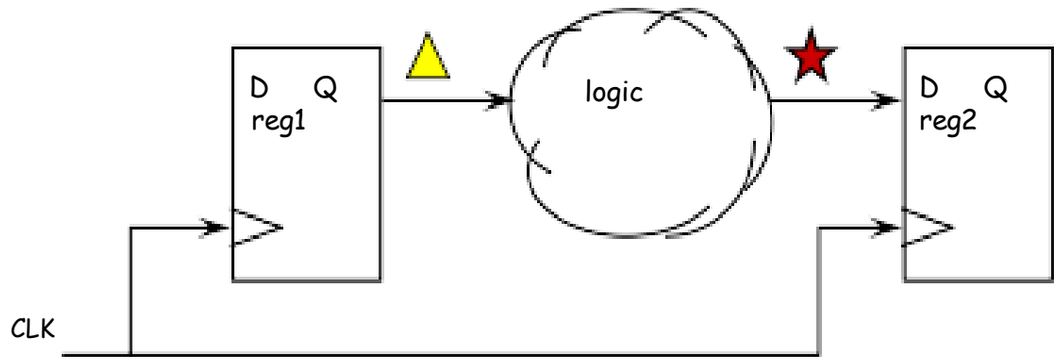
$$= \frac{\text{Αριθμός Εντολών} \times \text{CPI}}{\text{Συχνότητα Ρολογιού}}$$

Η Απόδοση του υπολογιστή μας

- Για αυτόν τον επεξεργαστή CPI=1
- Ας επικεντρωθούμε στην περίοδο του ρολογιού (πχ η εντολή *lw*).



Περίοδος Ρολογιού



- Η περίοδος ρολογιού καθορίζεται από το χειρότερο μονοπάτι μεταξύ δύο flip-flops σε ένα κύκλωμα. Στον επεξεργαστή μας αυτό το μονοπάτι ΜΑΛΛΟΝ είναι το παρακάτω:

$$T_c = t_{pcq\_PC} + t_{\text{mem}} + \max(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- Στους περισσότερους επεξεργαστές το λεγόμενο critical path είναι η μνήμη, η ALU, και οι καταχωρητές.
- Ο καθορισμός του critical path σε πραγματικούς επεξεργαστές είναι μια πολύ επίπονη εργασία που γίνεται από εξειδικευμένα εργαλεία CAD (Computer-Aided Design).

Παράδειγμα υπολογισμού περιόδου ρολογιού

Στοιχείο Υλικού	Συμβολικός Χρόνος	Χρόνος (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$

$= [30 + 2(250) + 150 + 25 + 200 + 20]$ ps

$= 925$ ps $] = 0.925$ns

$Fc = 1/Tc = 1.08$ GHz

Παράδειγμα υπολογισμού χρόνου εκτέλεσης

$$\text{Χρόνος CPU} = \text{Αριθμός ΕΕντολώστο ΠΠρόγραμμ} \times \text{CPI} \times \text{Περίοδος ΡΡολογιο}$$

$$= \frac{\text{Αριθμός ΕΕντολώ} \times \text{CPI}}{\text{Συχνότητα ΡΡολογιο}}$$

- Για ένα πρόγραμμα με 100 δις εκτελούμενων εντολών και CPI = 1

Χρόνος CPU = $(100 * 10^9)$ * (1) * (925 *10^{-12}) = 92.5 secs