

ΗΥ 232

Οργάνωση και Σχεδίαση Υπολογιστών

Διάλεξη 9

Επεξεργαστής

Υλοποίηση ενός κύκλου μηχανής

Νίκος Μπέλλας

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ

Τι είναι Αρχιτεκτονική και τι Μικροαρχιτεκτονική

- **Αρχιτεκτονική:** τα χαρακτηριστικά του επεξεργαστή που είναι ορατά στον προγραμματιστή (εντολές assembly, μνήμη, καταχωρητές, κοκ)
- **Μικροαρχιτεκτονική:** μία από τις πολλές πιθανές hardware υλοποιήσεις μιας αρχιτεκτονικής
- **Παραδείγματα:**
 - Intel x86-64 αρχιτεκτονική,
 - *Intel Core (2006), Sandy Bridge (2011), Skylake (2015)* είναι κάποιες από τις μικρο-αρχιτεκτονικές.
Υλοποιήσεις της αρχιτεκτονικής x86-64 (Intel 64 bit)
 - Αρχιτεκτονική MIPS-32
 - *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)

MIPS Arithmetic Logic Unit (ALU)

Αριθμητική και Λογική μονάδα

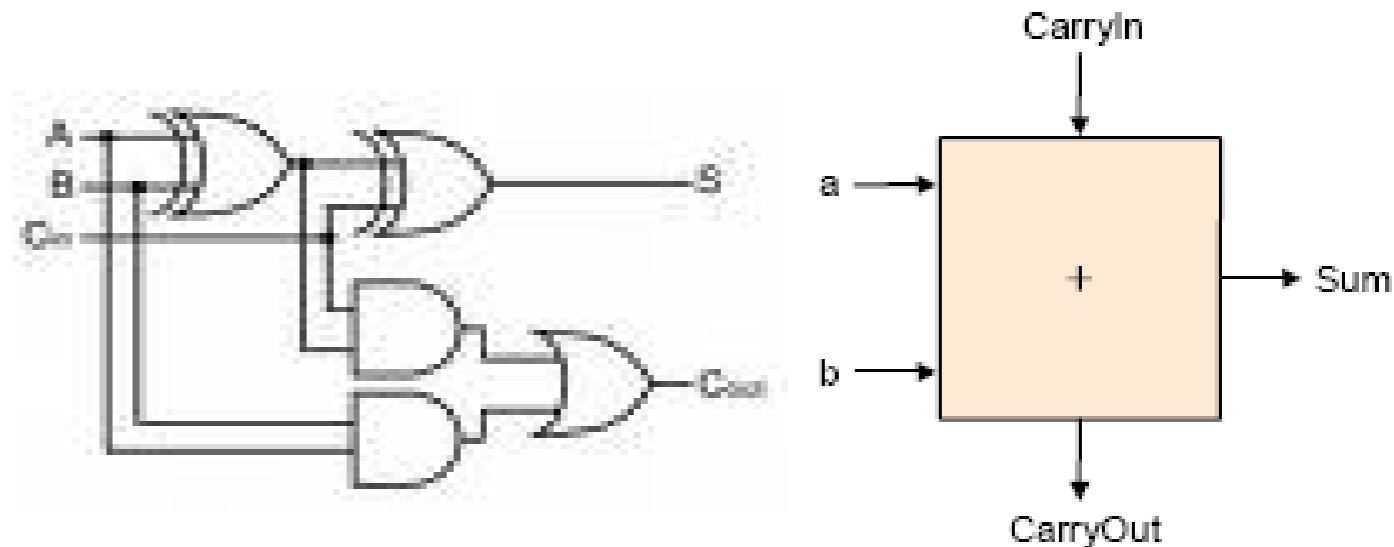
- Όλες οι αριθμητικές και λογικές πράξεις στον MIPS γίνονται σε μιά 32-bit Αριθμητική και Λογική μονάδα (ALU)
 - add/addi/addu/addiu
 - sub/subi/subu/subiu
 - and/andi
 - or/ori
 - nor
 - slt/slti/sltu/sltiu
- Ας προσπαθήσουμε να δούμε πώς υλοποιούνται όλες αυτές οι εντολές στην ALU του MIPS
 - Χρησιμοποιώντας τις γνώσεις σας στην Ψηφιακή Σχεδίαση I

Αριθμητική και Λογική μονάδα

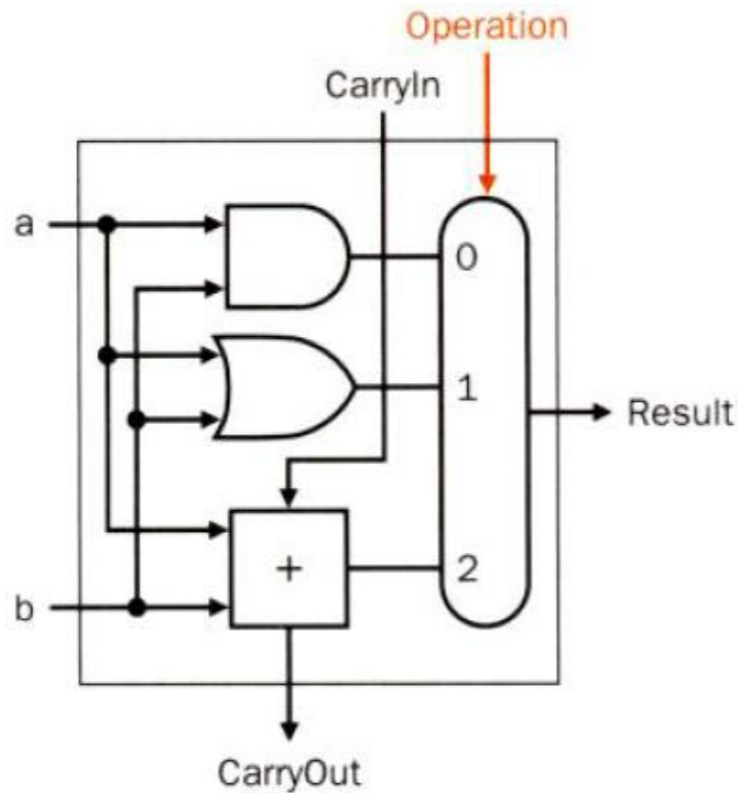
1-bit Full adder

Input			Output	
<i>a</i>	<i>b</i>	<i>c_{in}</i>	<i>c_{out}</i>	<i>Sum</i>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$$
$$sum = a \text{ xor } b \text{ xor } c_{in}$$

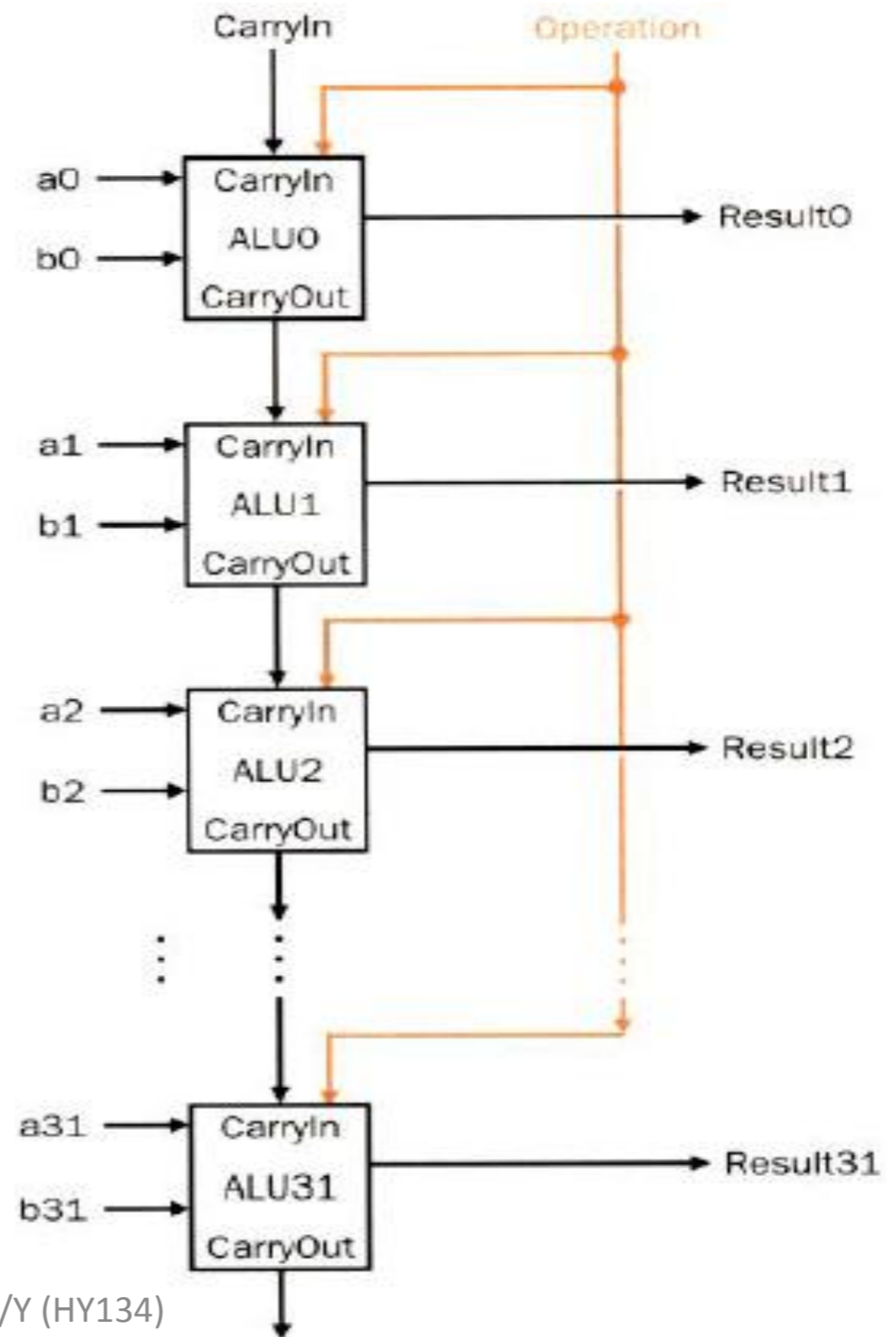


Αριθμητική και Λογική μονάδα

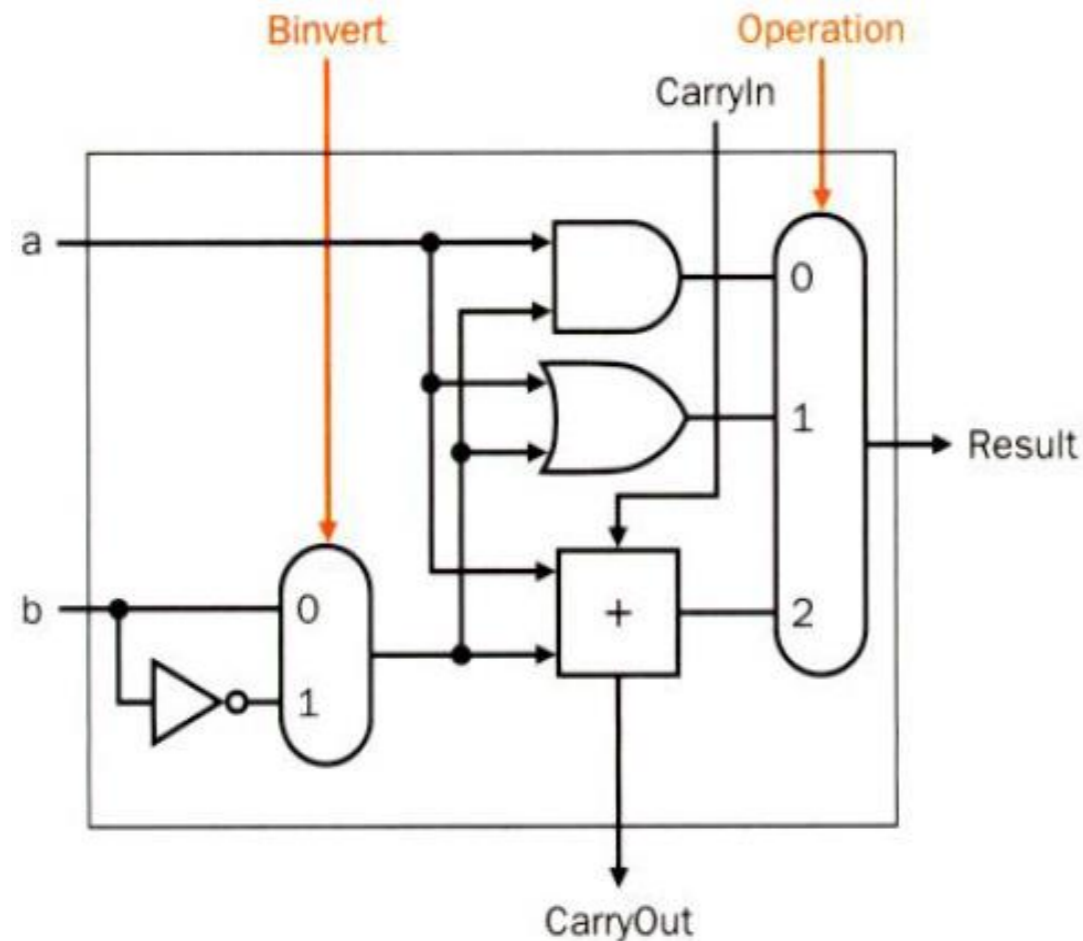


1-bit ALU
(and, or, add)

32-bit ALU
(*ripple-carry* configuration)



Αριθμητική και λογική μονάδα



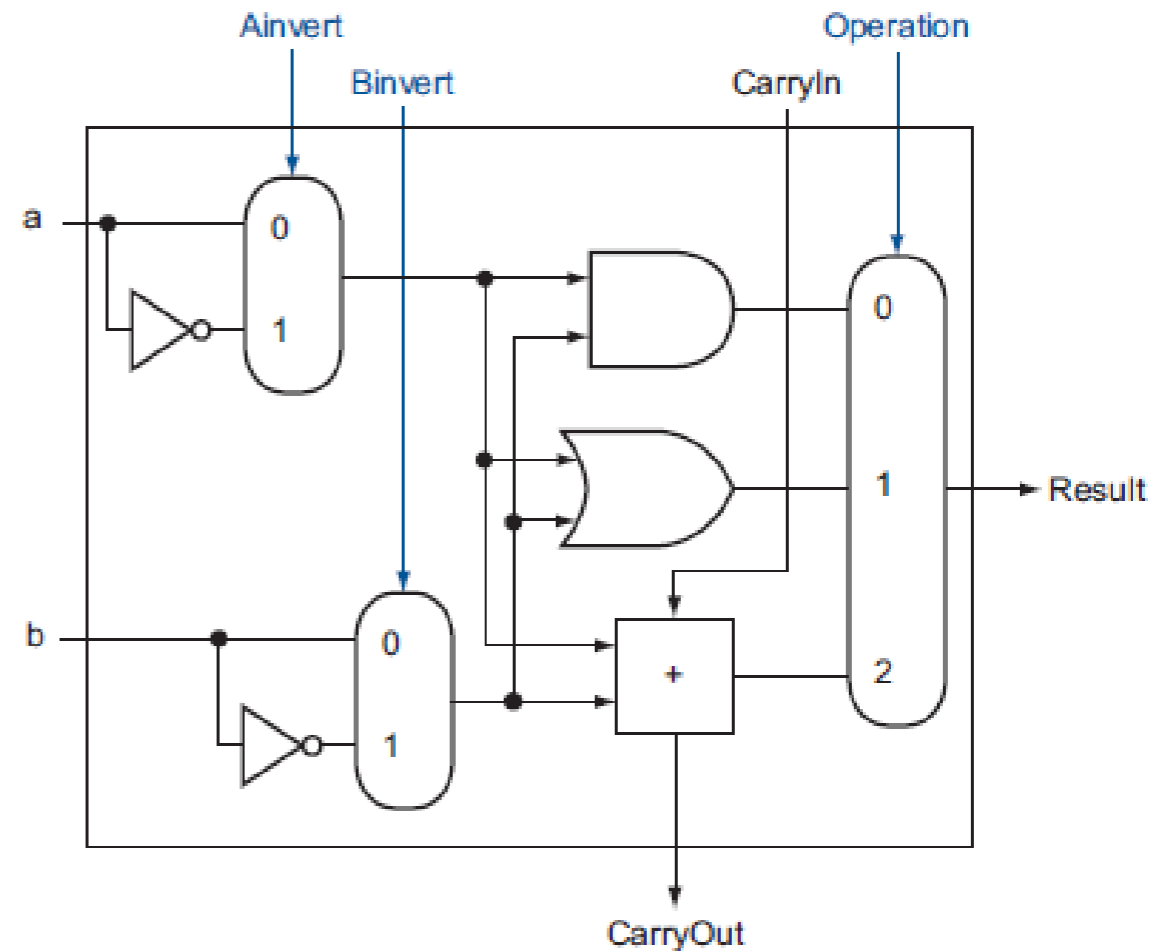
1-bit ALU

υλοποιεί $a+b$ και $a+b'$

$$a-b = a+(-b)=a+b'+1$$

(and, or, add, **sub**)

[$c_{in}=1$ in ALU0 for sub]



1-bit ALU

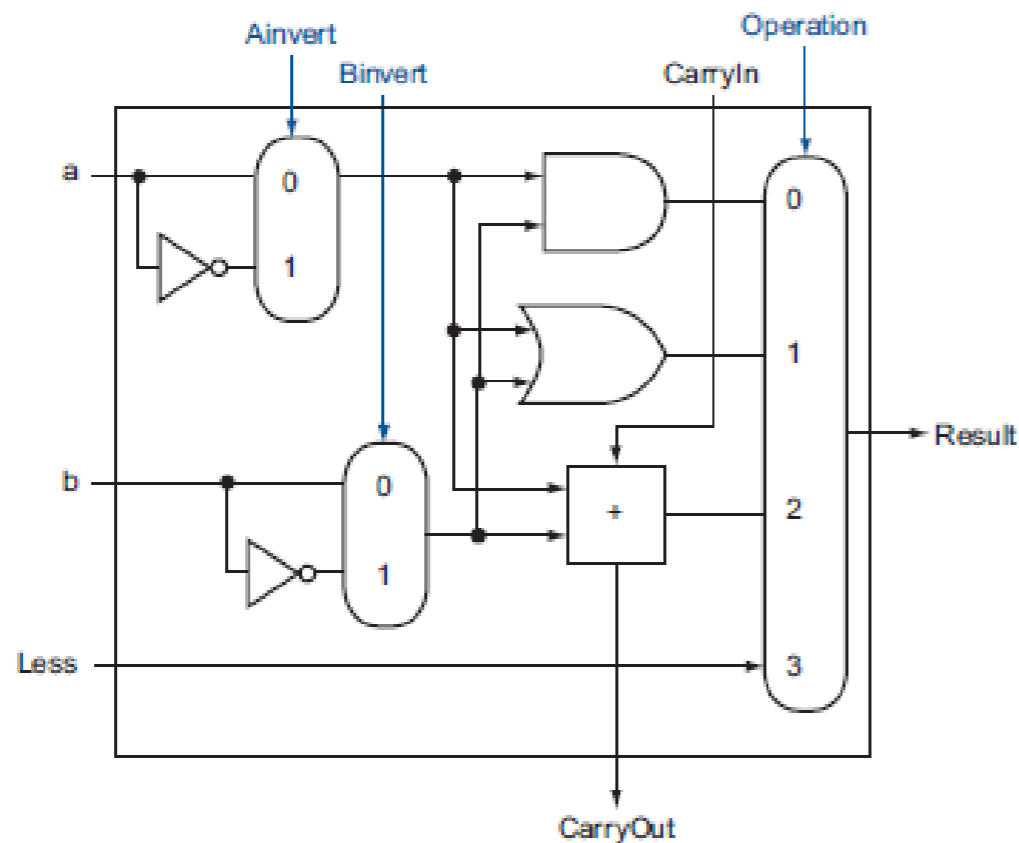
(and, or, **nor**, add, sub)

$$(a \text{ OR } b)' = a' \text{ AND } b'$$

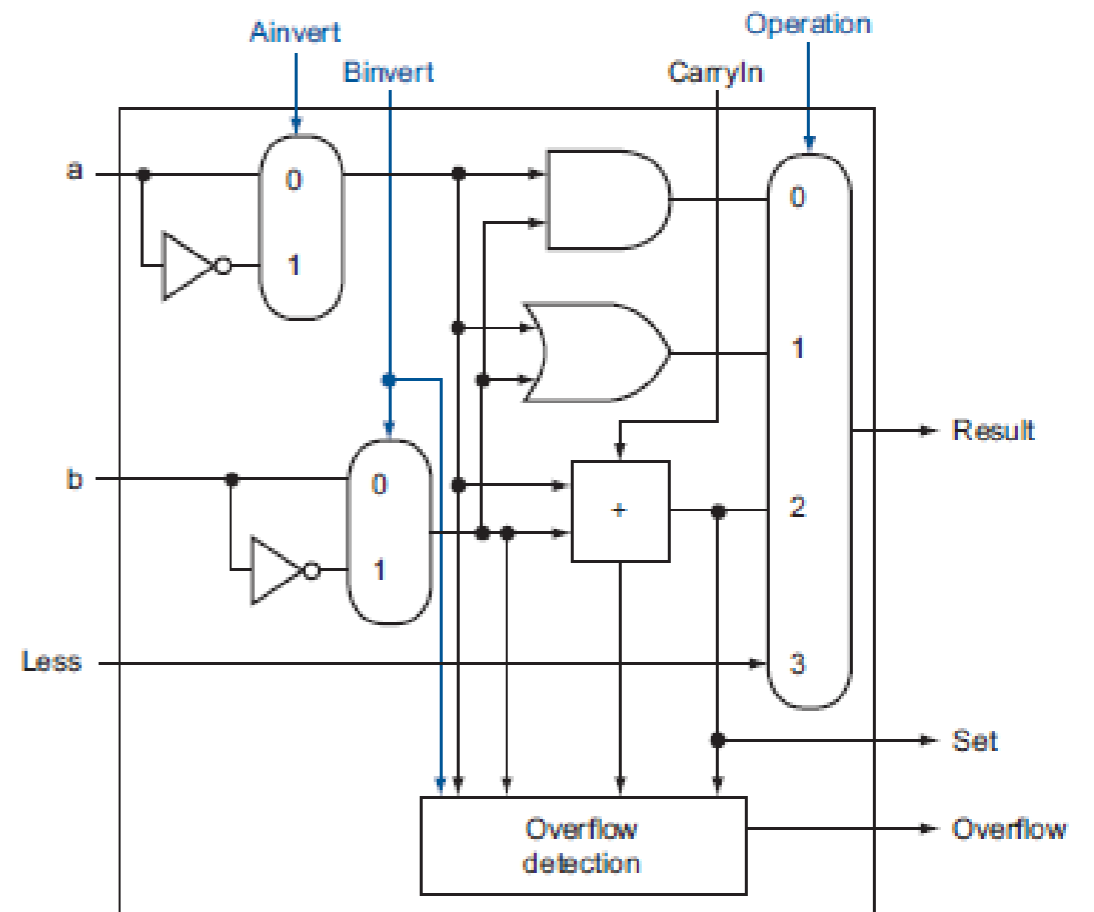
Implements *nor* and *not* (if $b==0$)

Αριθμητική και λογική μονάδα (slt)

Για να υλοποιηθεί η **slt c, a, b** πρέπει να υπολογίσουμε εάν $a < b \rightarrow a - b < 0$. Εάν $x = a - b < 0$, το κρατούμενο εξόδου είναι 1 επειδή το x είναι αρνητικός αριθμός



1-bit ALU (ALU0-ALU30)



1-bit ALU (ALU31)

(and, or, nor, add, sub, **slt**)

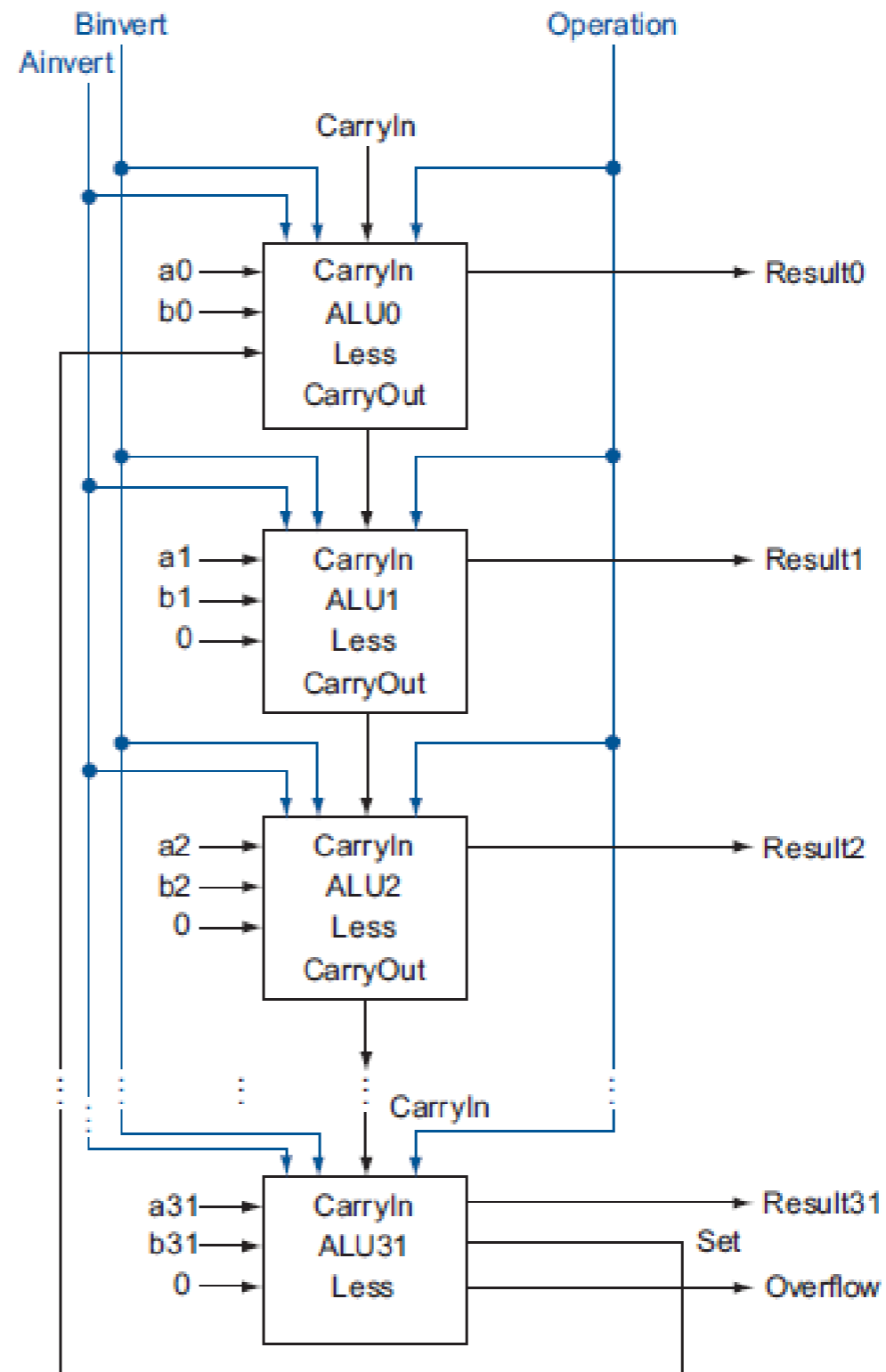
[overflow detection]

[Less=0 in ALU1-ALU31]

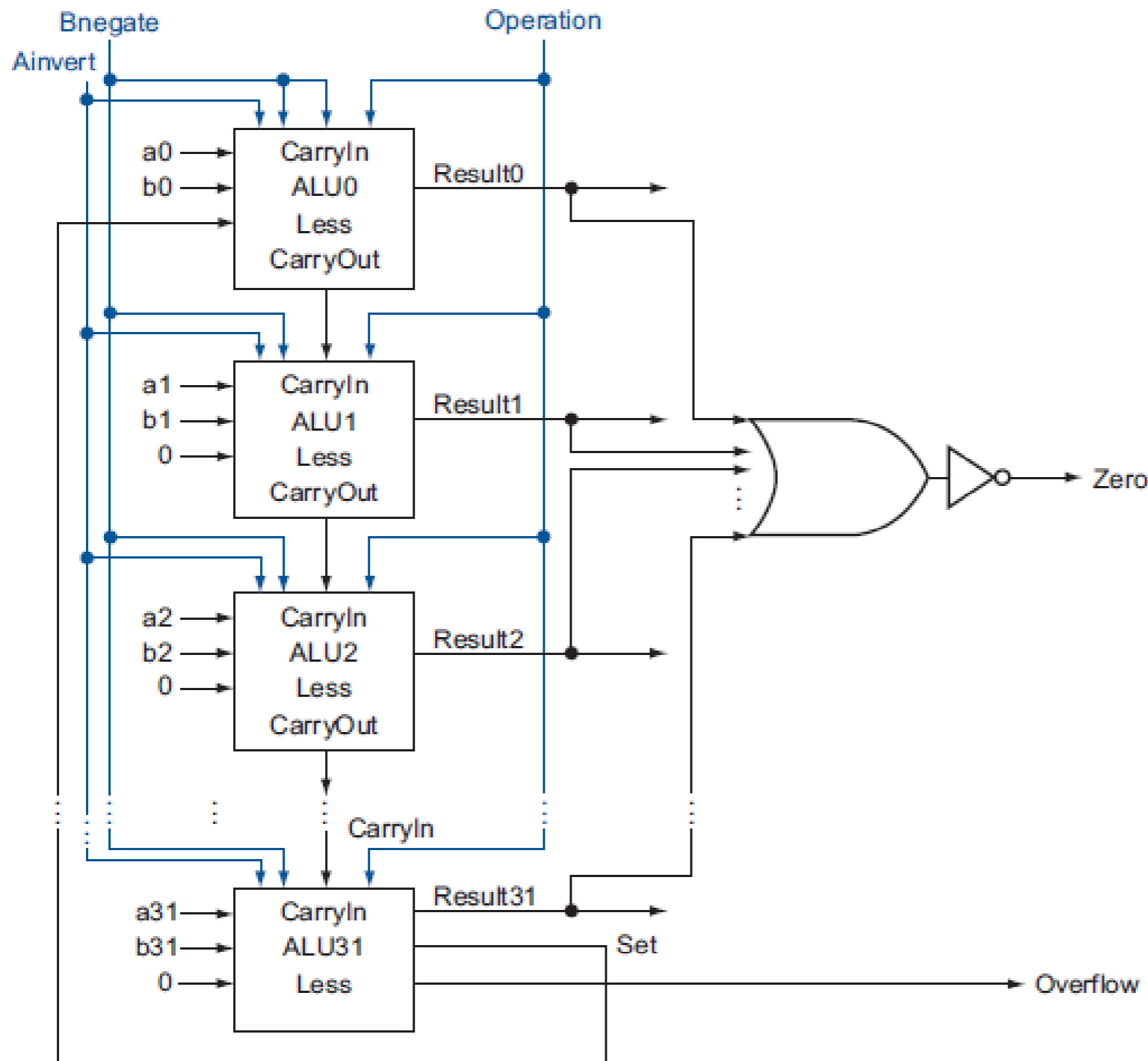
[Less=(a-b)₃₁ in ALU0]

Αριθμητική και λογική μονάδα

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



Αριθμητική και λογική μονάδα



Για εντολές διακλάδωσης

beq a, b, L

θα πρέπει να δούμε εάν $a=b \Rightarrow (a-b == 0)$

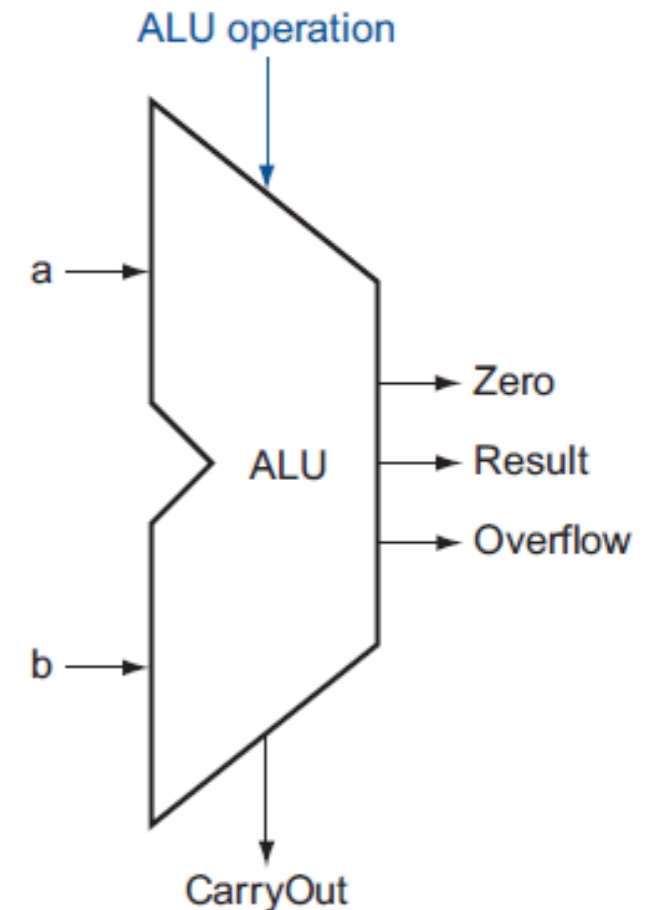
Αφαιρούμε $a-b$ στην ALU και διαπιστώνουμε εάν το αποτέλεσμα είναι 0 ή όχι.

32-bit ALU
with zero detection
(for *beq*, *bne*)

Αριθμητική και λογική μονάδα

Τα 3 σήματα ελέγχου *Ainvert*, *Bnegate*, *Operation* καθορίζουν και την λειτουργία της Αριθμητικής και Λογικής μονάδας (ALU)

Ainvert	Bnegate	Operation	Function
ALU Operation			
	0_0_00		AND
	0_0_01		OR
	0_0_10		ADD
	0_1_10		SUB
	0_1_11		SLT
	1_1_00		NOR

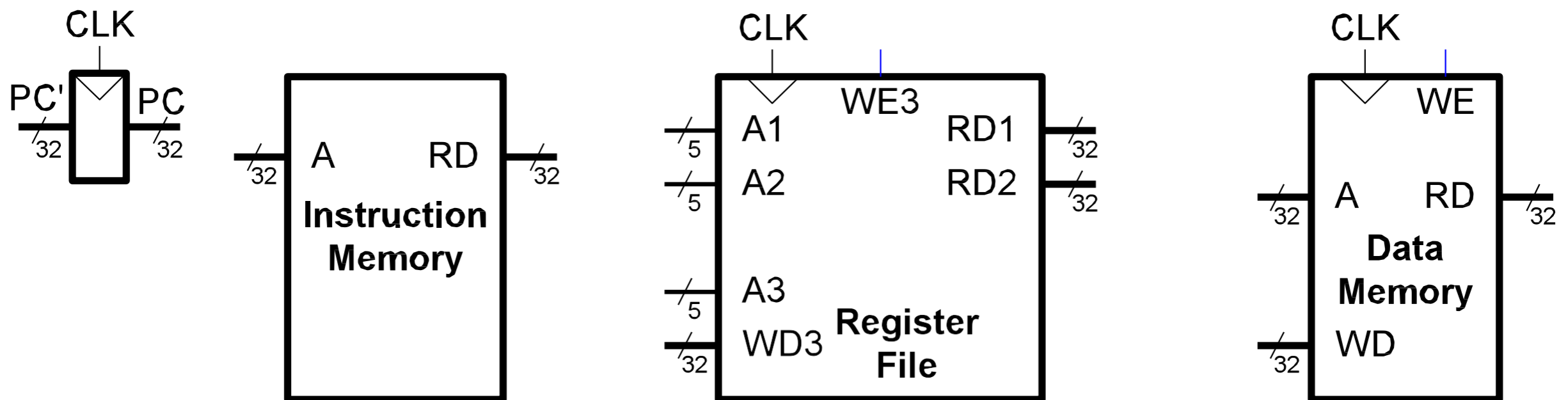


Υλοποίηση ενός υποσυνόλου εντολών

- 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

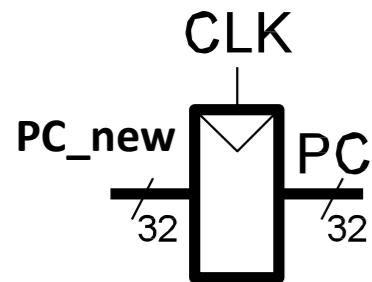
Αρχιτεκτονικά στοιχεία του MIPS

- 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

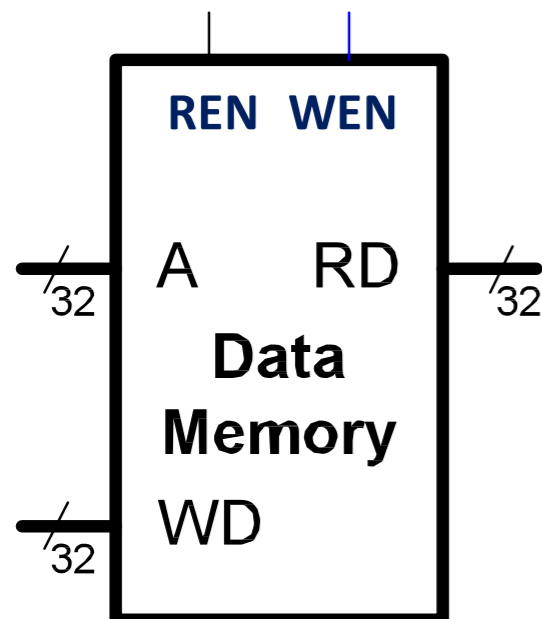
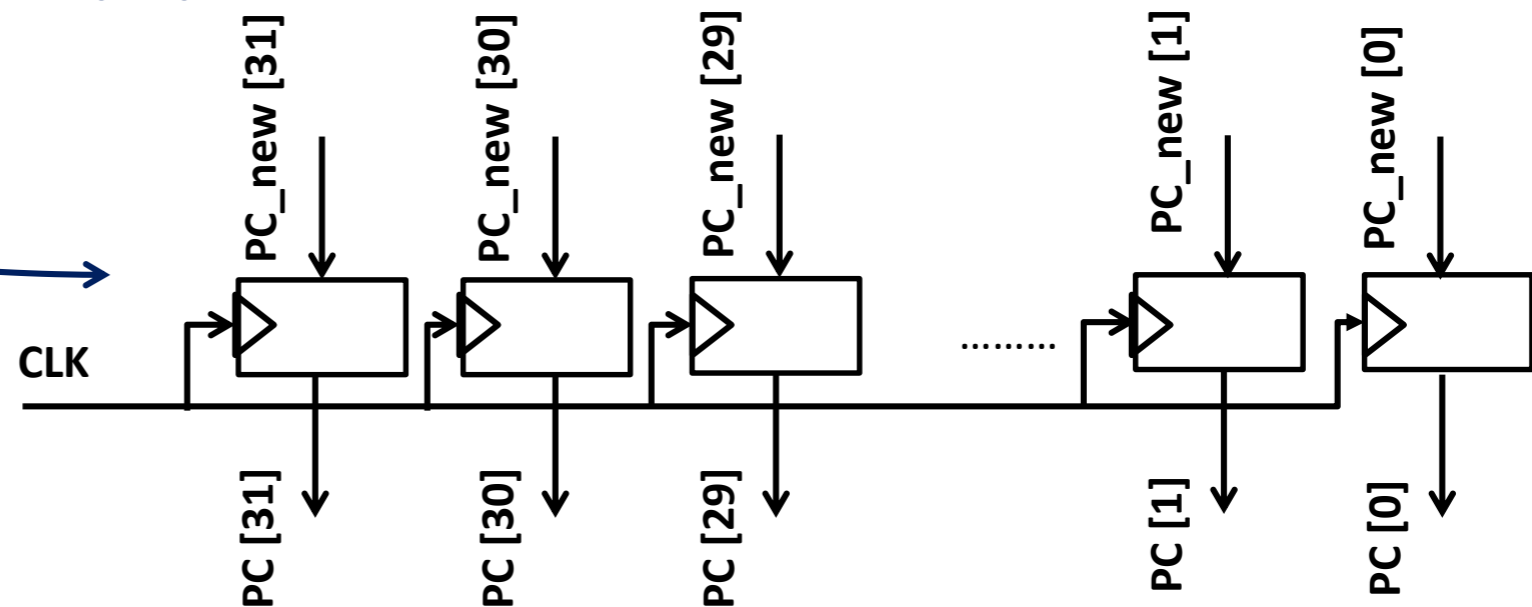
- Για να τα δούμε με λεπτομέρεια:



```

always @(posedge CLK or
        negedge reset)

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



```

module Memory (input ren, wen,
               input [31:0] addr, din,
               output [31:0] dout); // 4k words mem

    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 @(*)
        begin
            if ((wen == 1'b1) && (ren==1'b0))
                data[addr[11:0]] = din;
        end
    endmodule
    
```

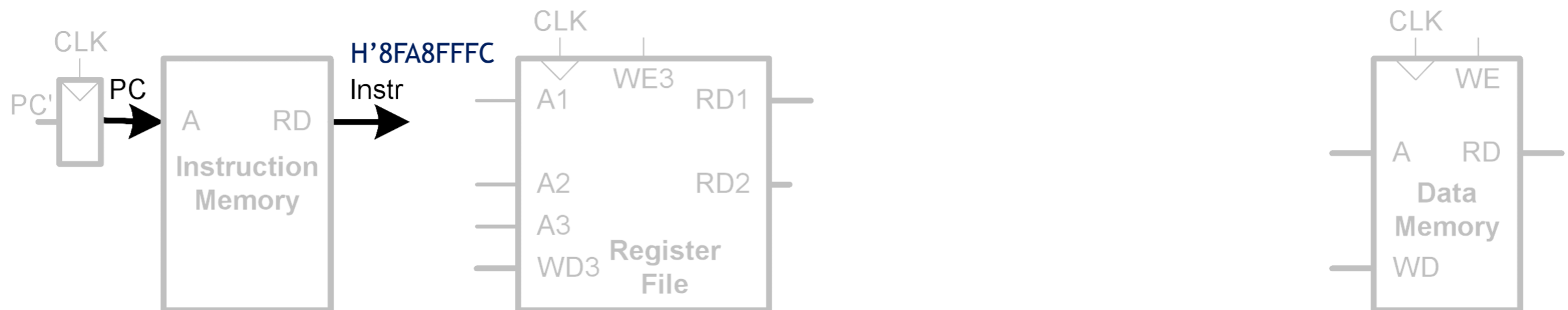
Υλοποίηση της πρώτης μας εντολής: *lw*

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)

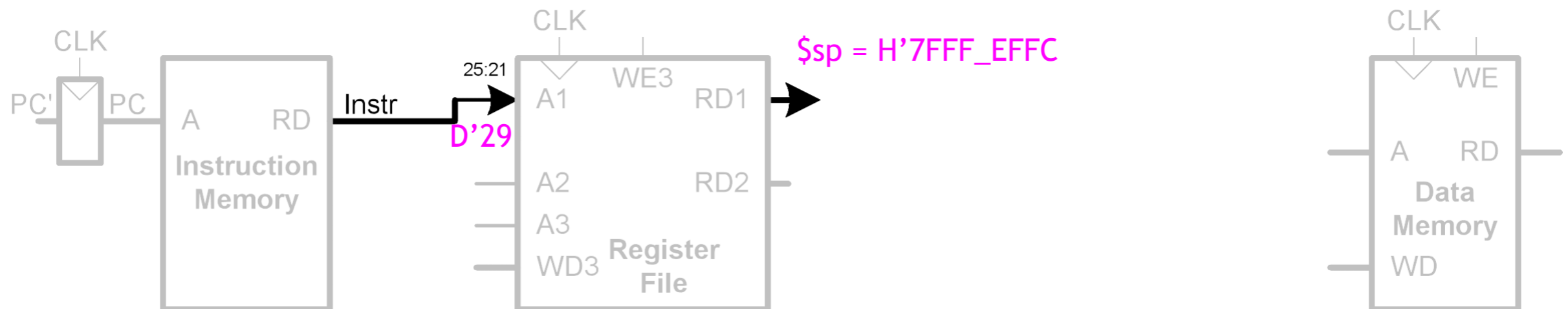


Υλοποίηση της πρώτης μας εντολής: lw

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

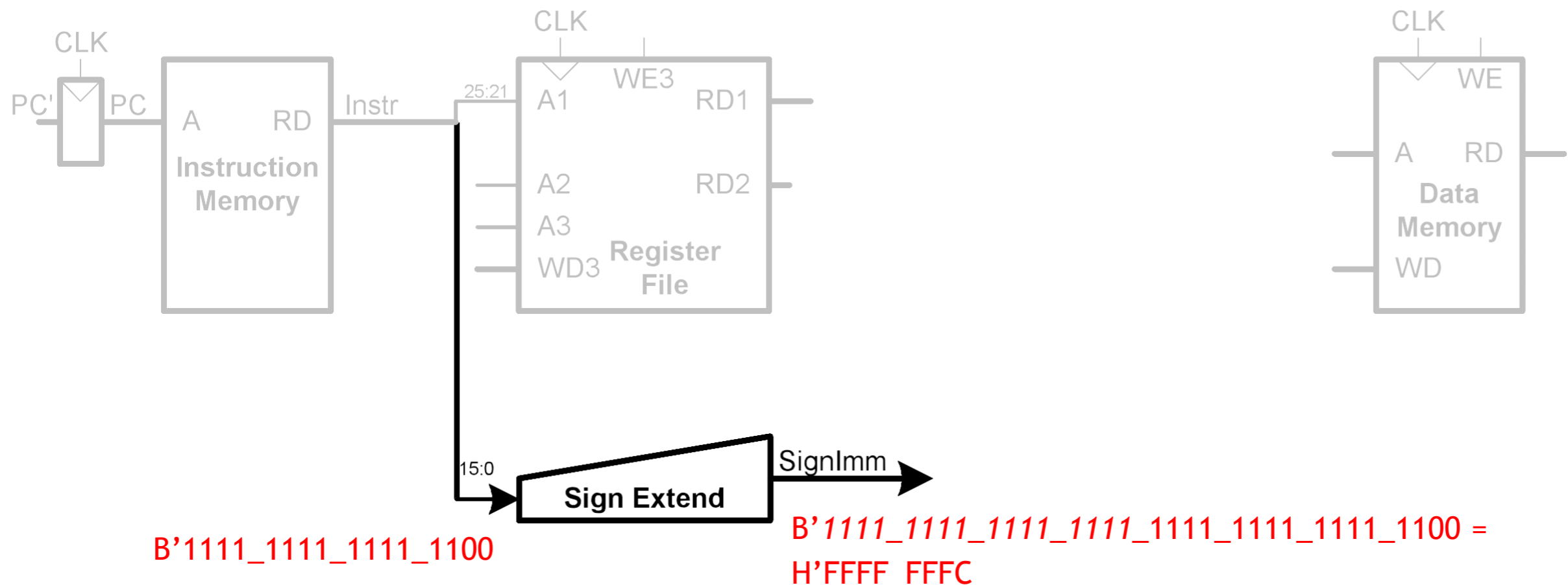
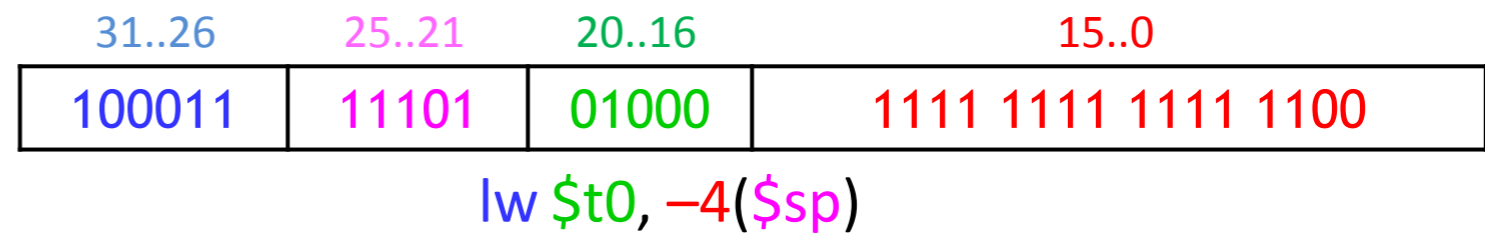


lw \$t0, -4(\$sp)



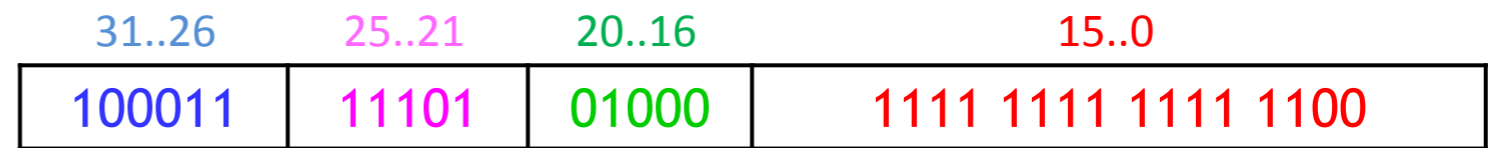
Υλοποίηση της πρώτης μας εντολής: lw

Βήμα 3: *Sign Extension*
Επέκταση πρόσημου της σταθεράς

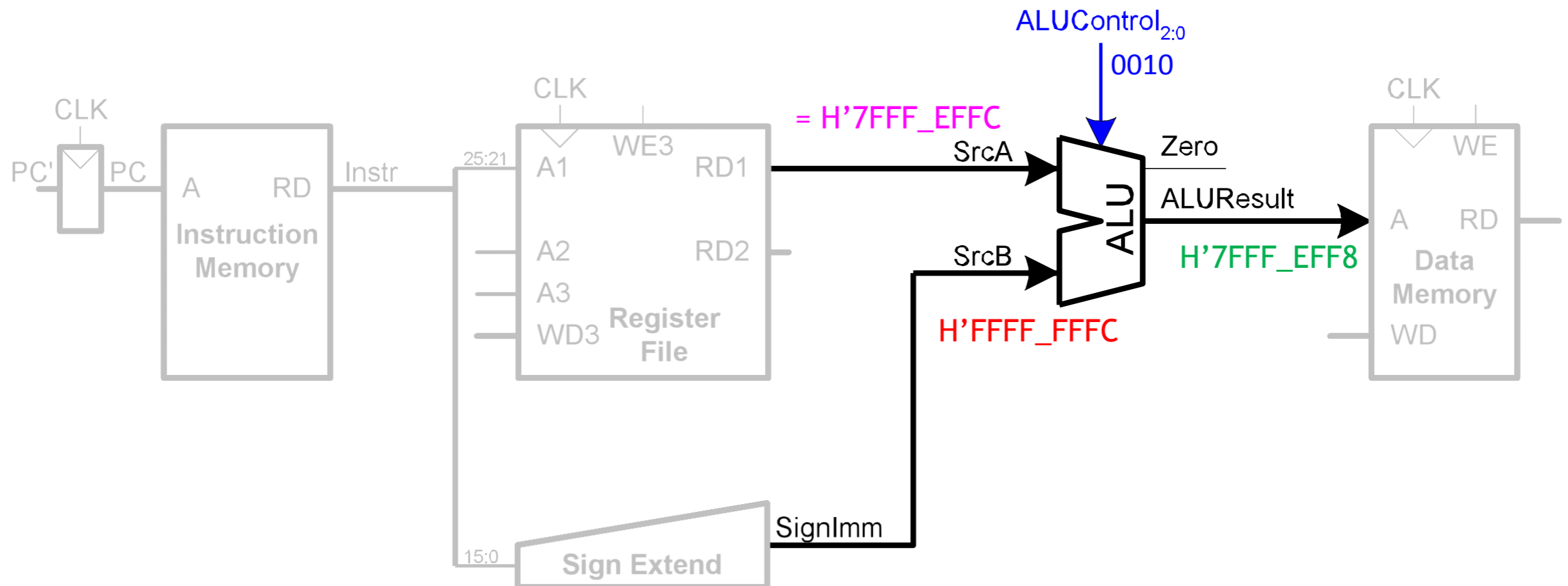


Υλοποίηση της πρώτης μας εντολής: lw

Βήμα 4: Υπολόγισε την διεύθυνση μνήμης κάνοντας πρόσθεση στην ALU



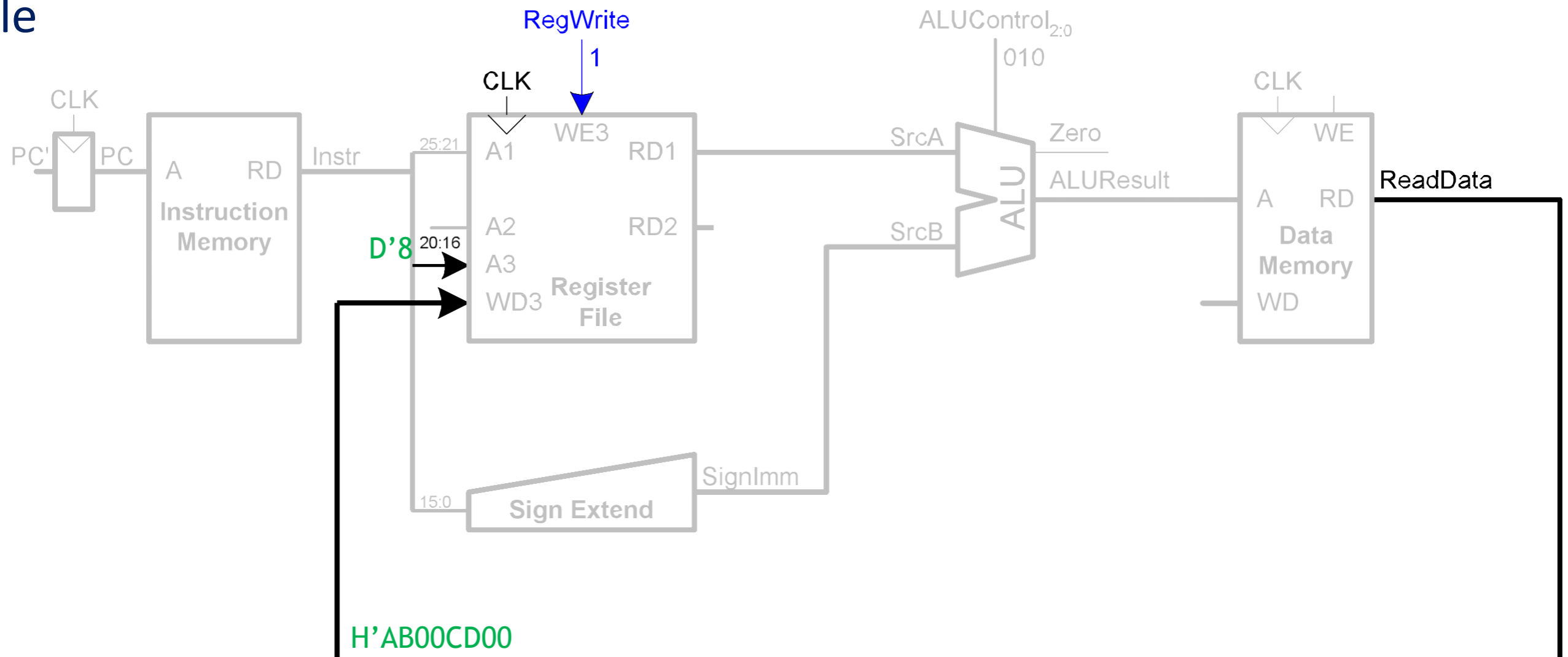
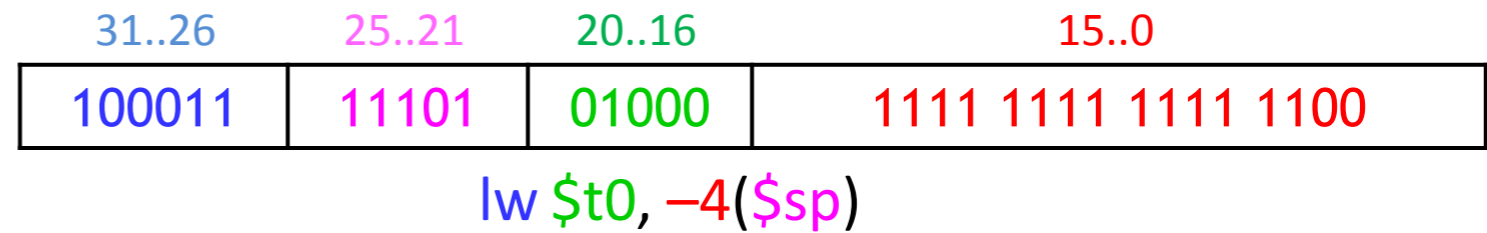
lw \$t0, -4(\$sp)



Υλοποίηση της πρώτης μας εντολής: lw

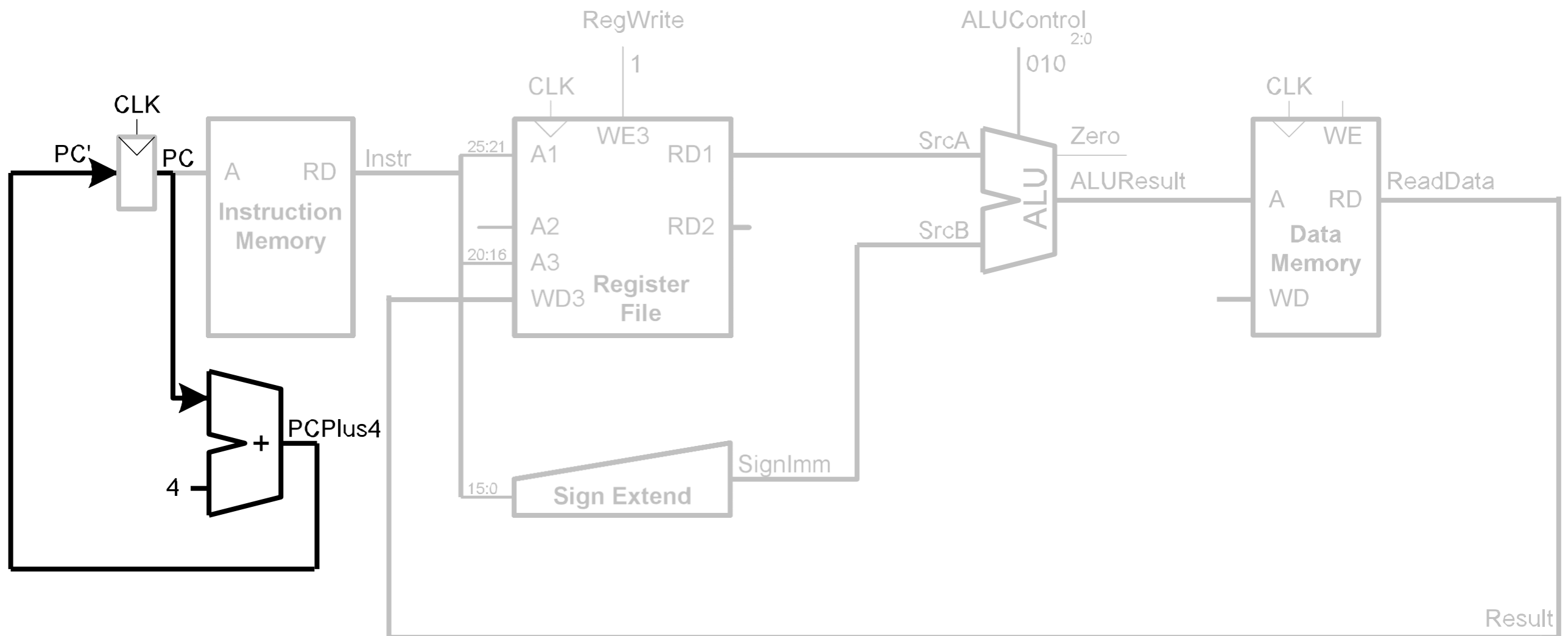
Βήμα 5: Διάβασε δεδομένα από την μνήμη χρησιμοποιώντας την διεύθυνση που μόλις υπολόγισες.

Γράψε τα δεδομένα στο register file



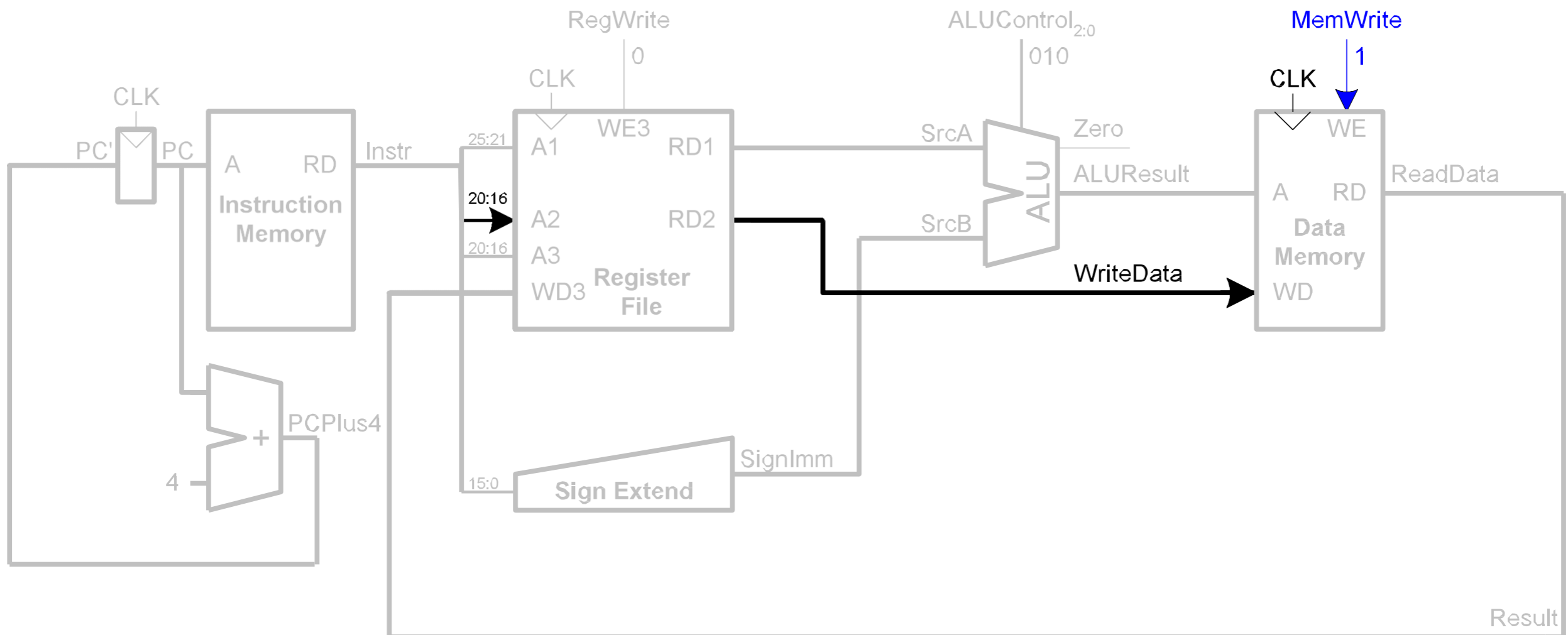
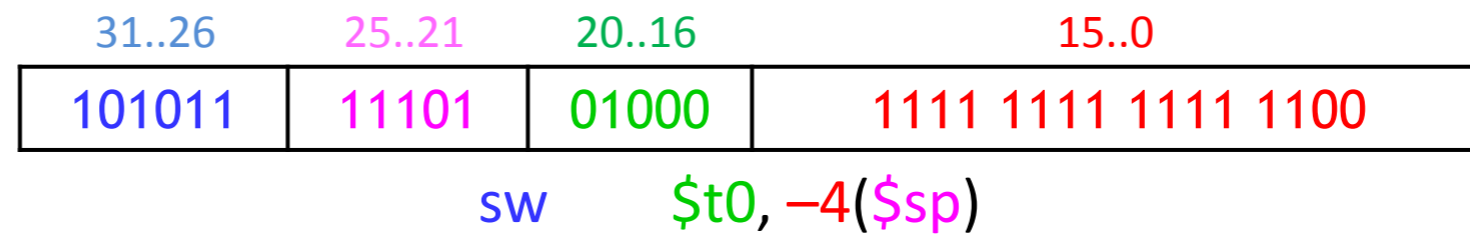
Υλοποίηση της πρώτης μας εντολής: lw

Βήμα 6: Βρες την διεύθυνση της επόμενης εντολής $PC \leq PC+4$



Υλοποίηση της εντολής sw

Με λίγο extra hardware, μπορούμε να υλοποιήσουμε και την sw

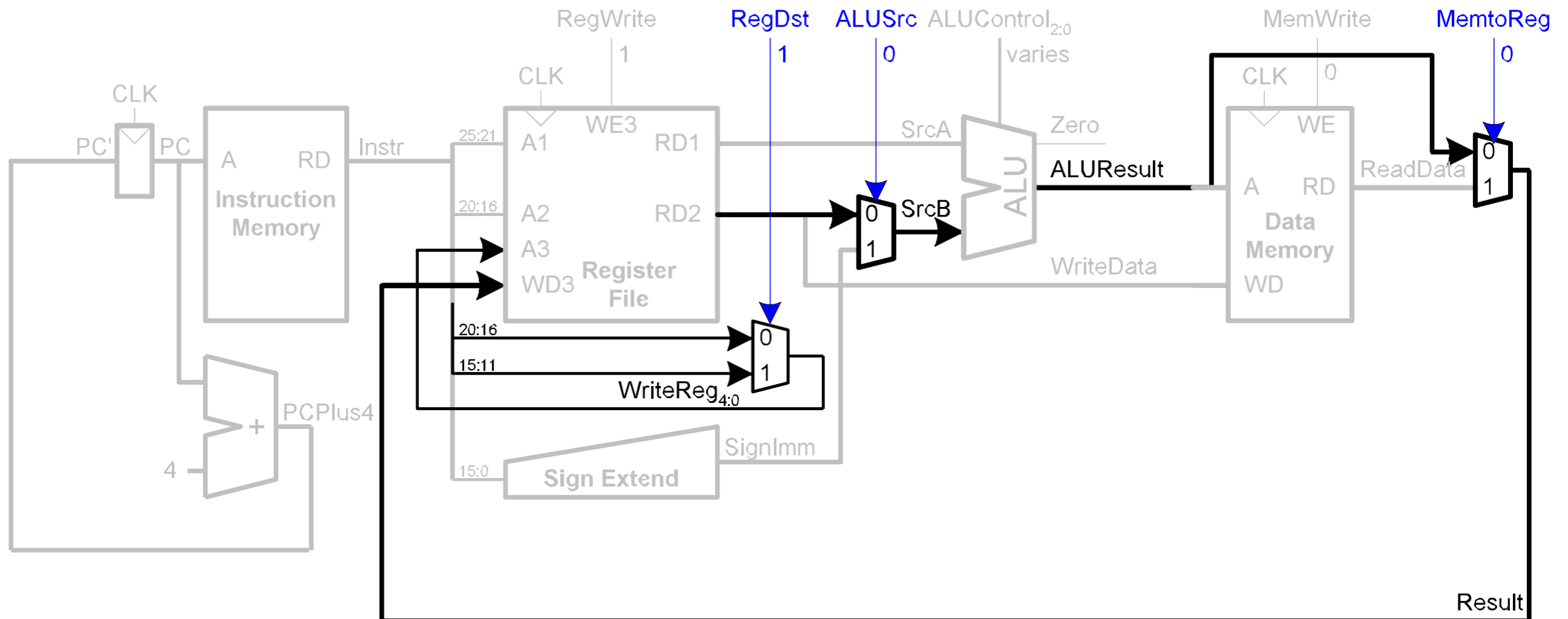


Υλοποίηση εντολών *R-format*

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

31..26	25..21	20..16	15..11	10..6	5..0
op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

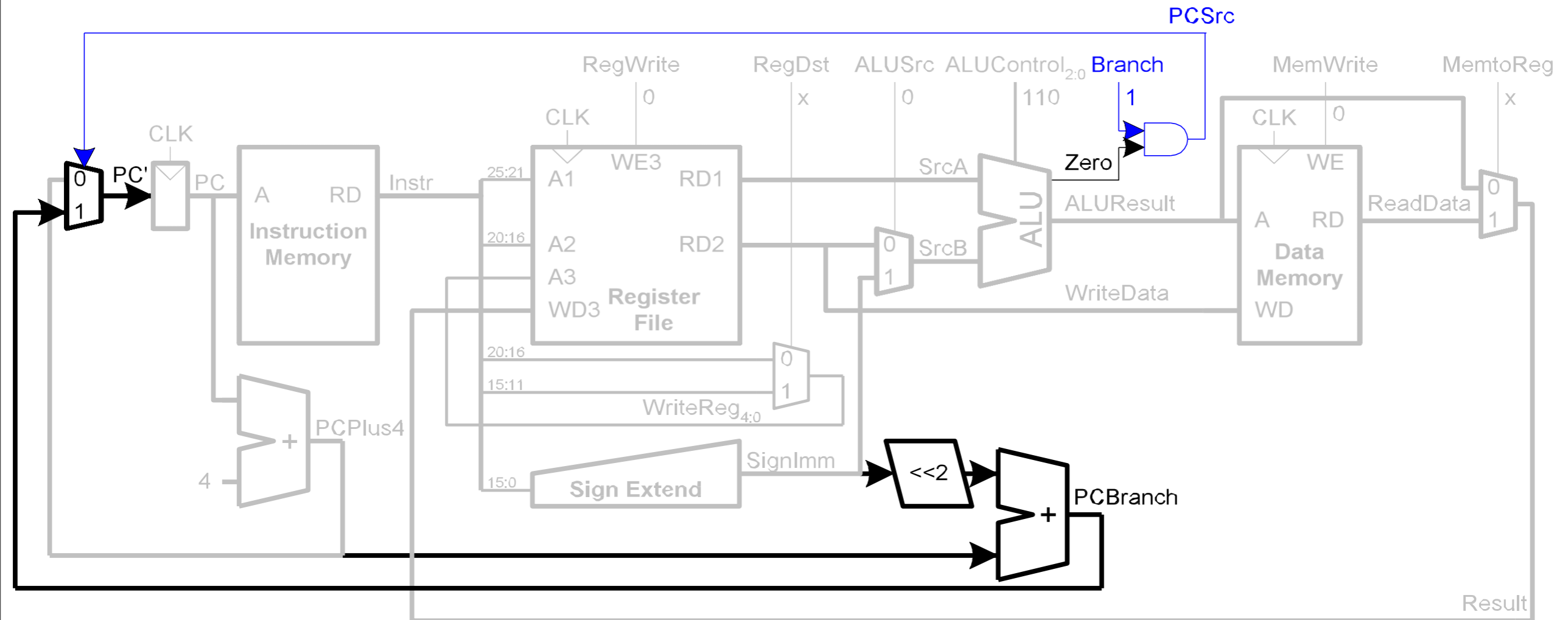
31..26	25..21	20..16	10..0
op	rs	rt	address
6 bits	5 bits	5 bits	16 bits



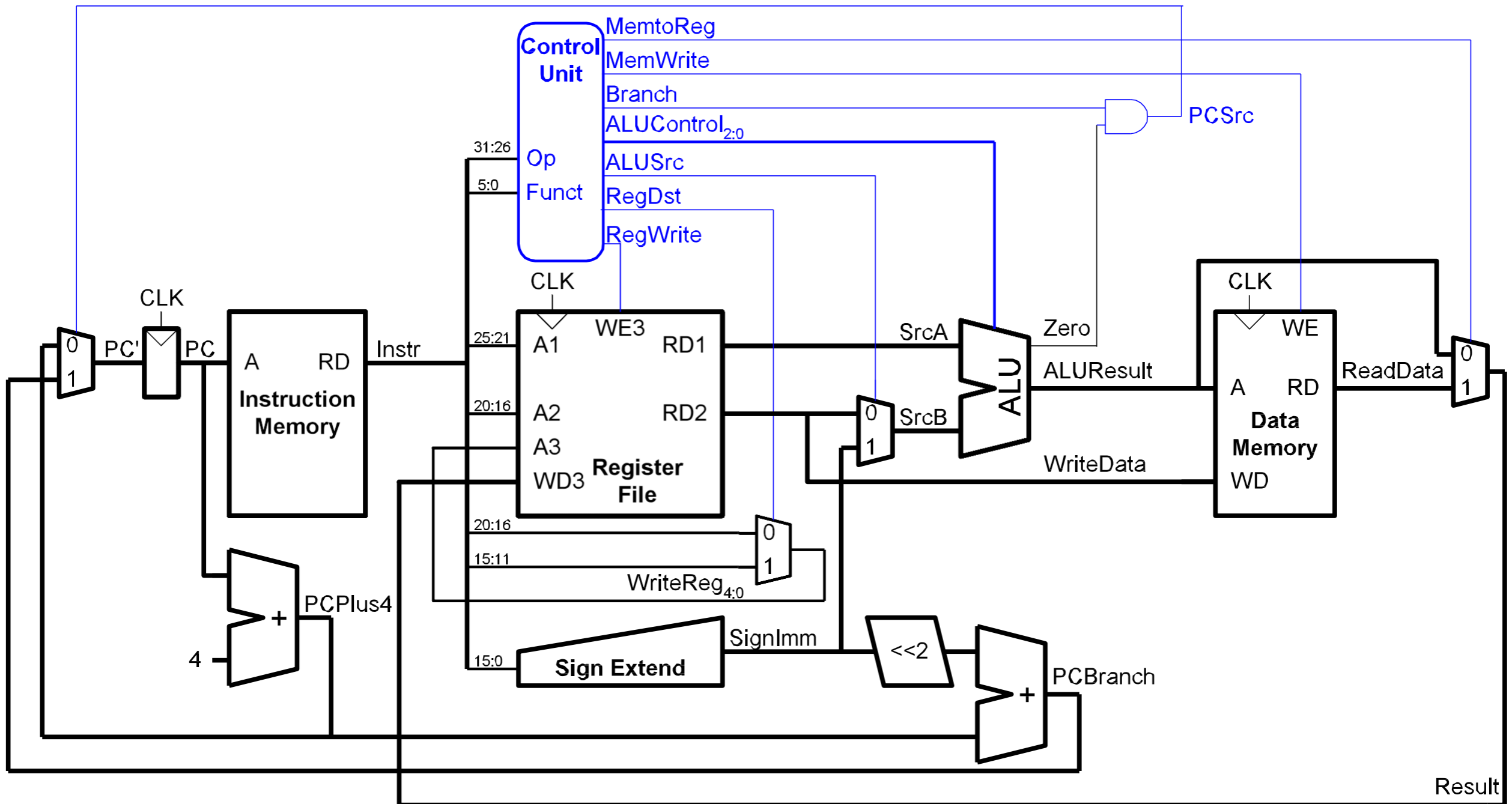
Υλοποίηση εντολής *beq*

31..26	25..21	20..16	15..0
op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

- Διάβασε τους καταχωρητές *rs* και *rt* και έλεγξε εάν οι τιμές τους είναι ίδιες.
- Υπολόγισε την διεύθυνση $\Delta = (PC+4) + 4 * (\text{sign extended immediate})$
- Εάν $rs == rt$, θέσε $PC \leq \Delta$, αλλιώς $PC \leq PC+4$

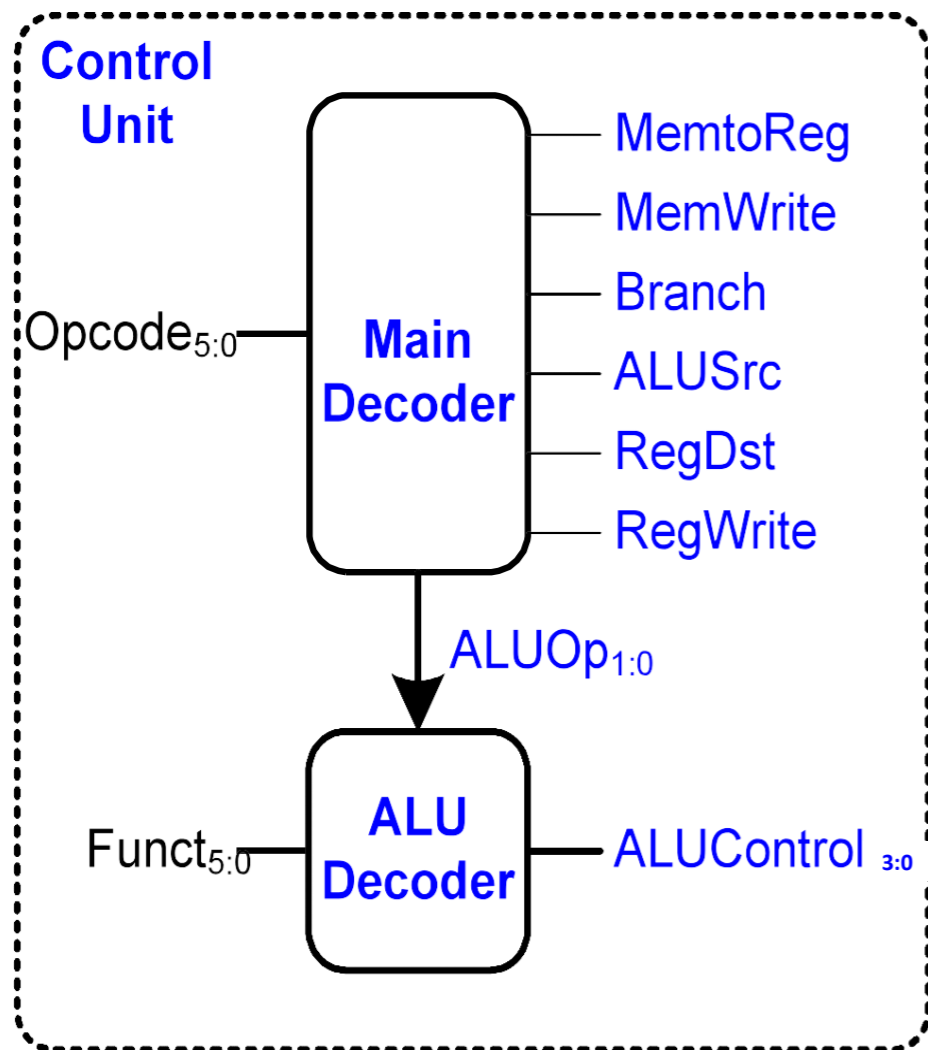


Υλοποίηση του επεξεργαστή ενός κύκλου



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 *func[5:0]*

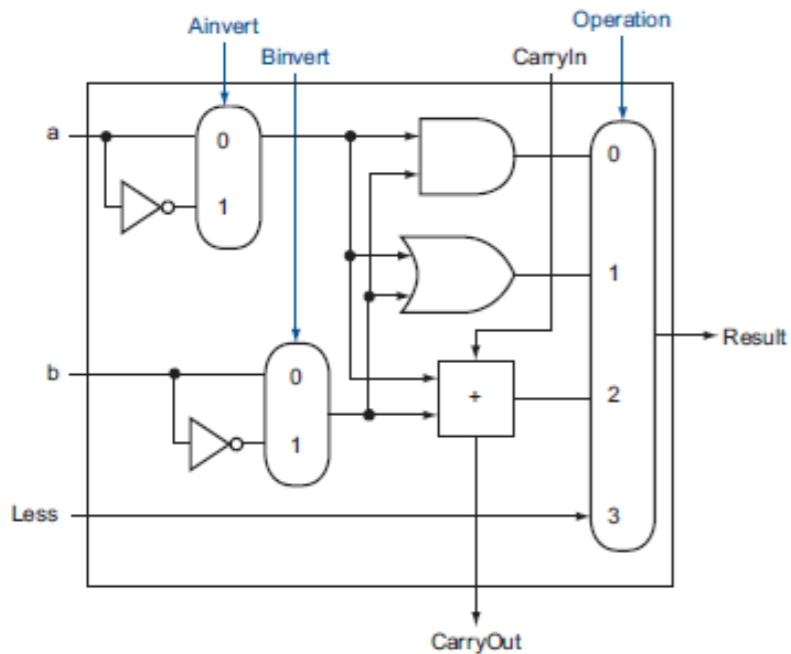


op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

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

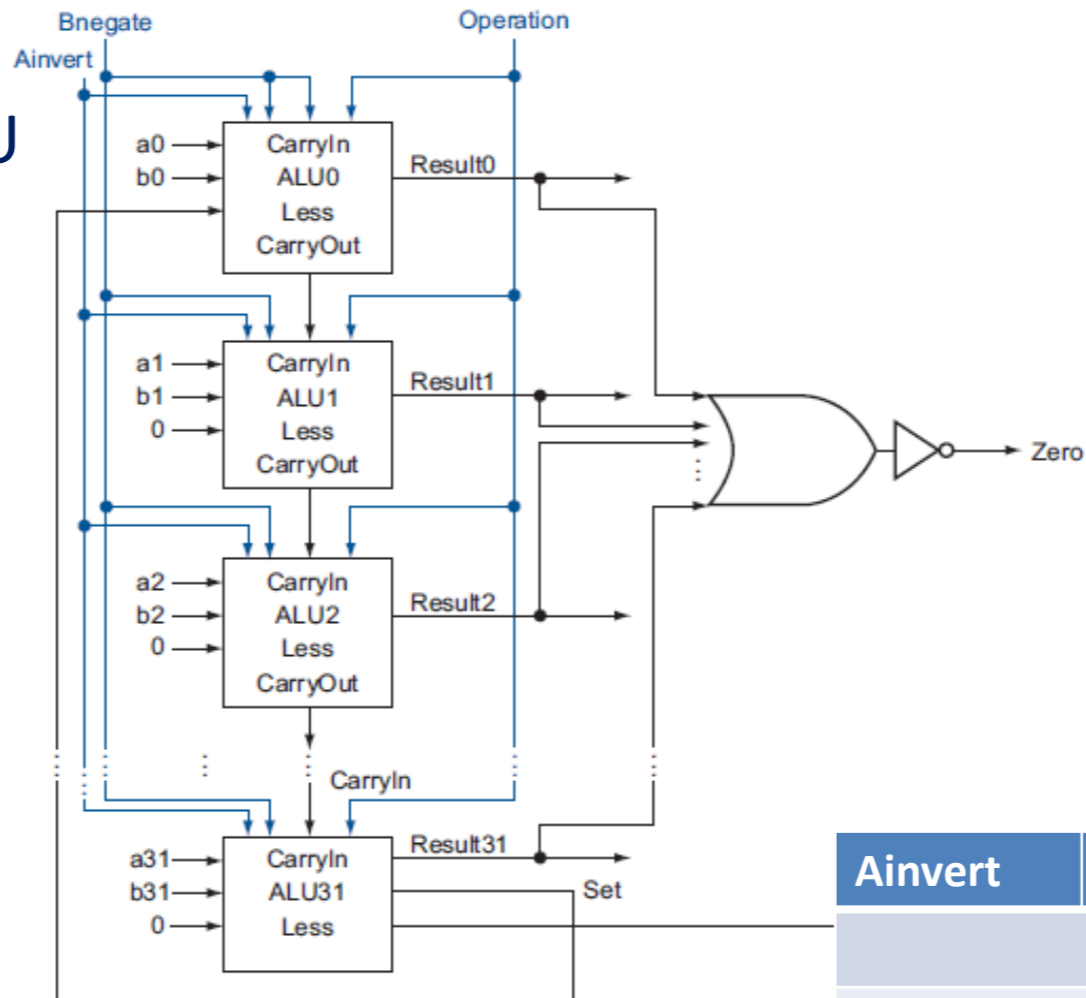
ALU και Μονάδα Ελέγχου

Για να θυμηθούμε την ALU



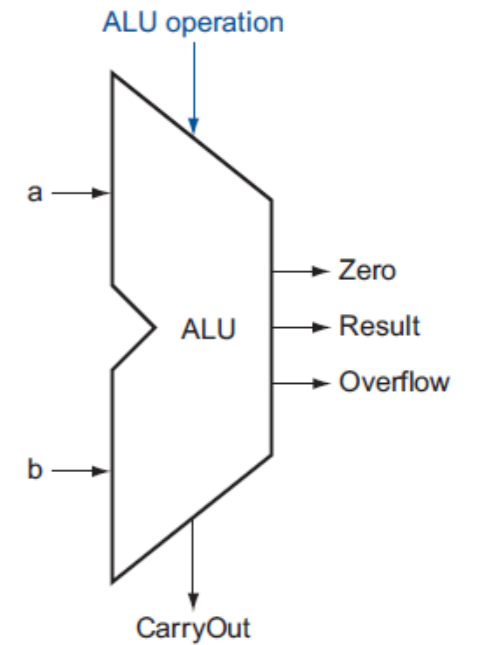
1-bit ALU

(and, or, add, sub, nor, slt, beq)
(4 control signals)



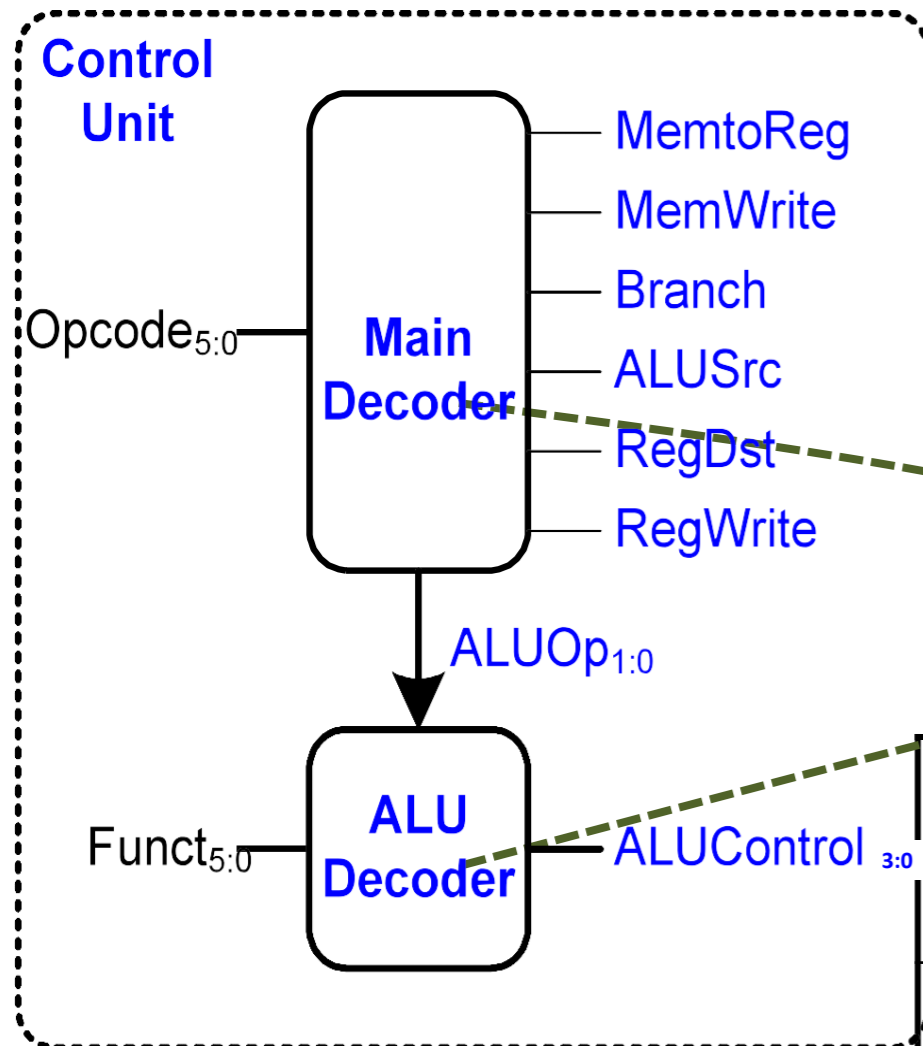
32-bit ALU

(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

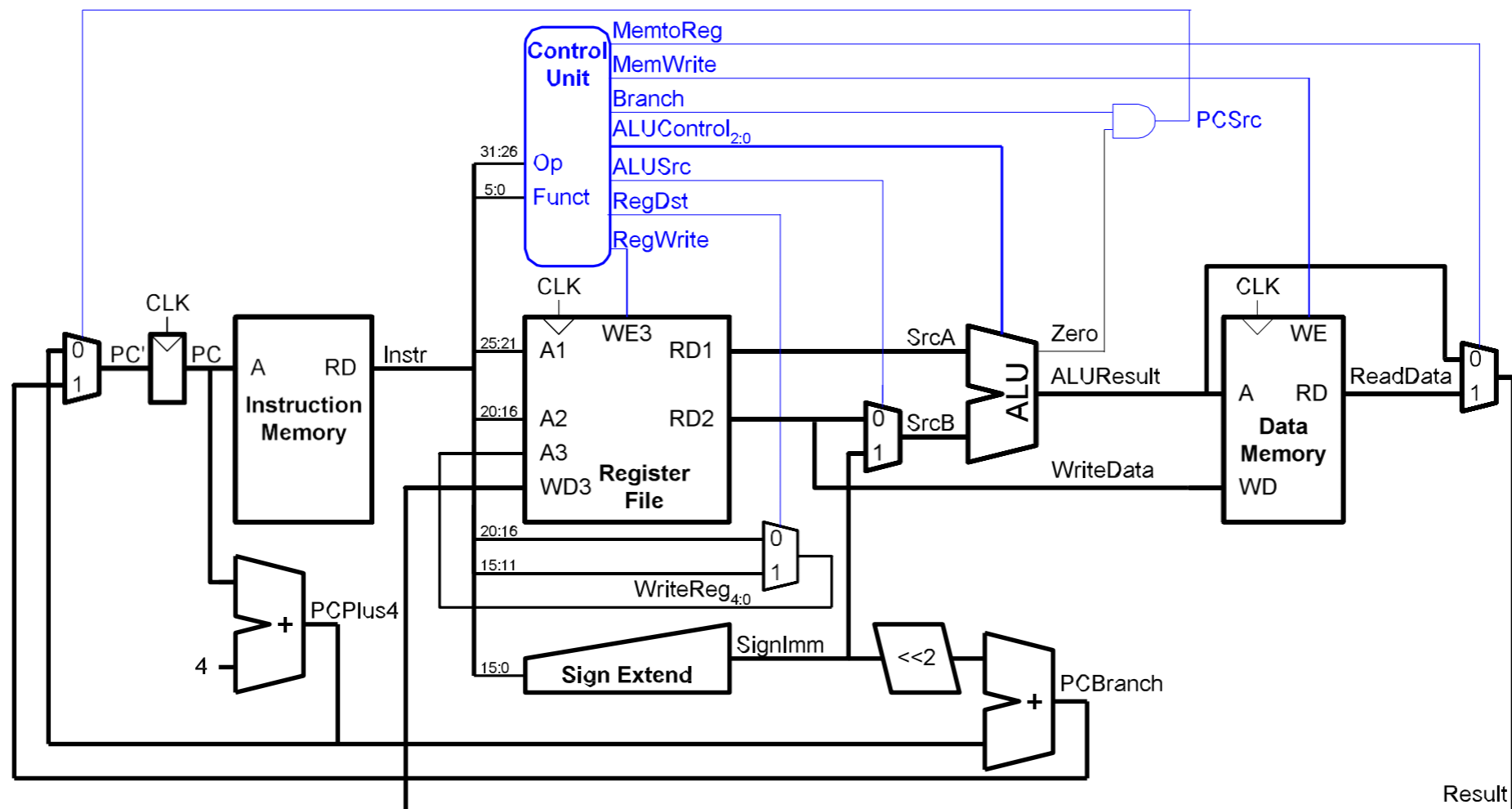


- Για μία συγκεκριμένη εντολή η μονάδα ελέγχου παράγει τα 4 σήματα ελέγχου ALUControl[3:0]

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

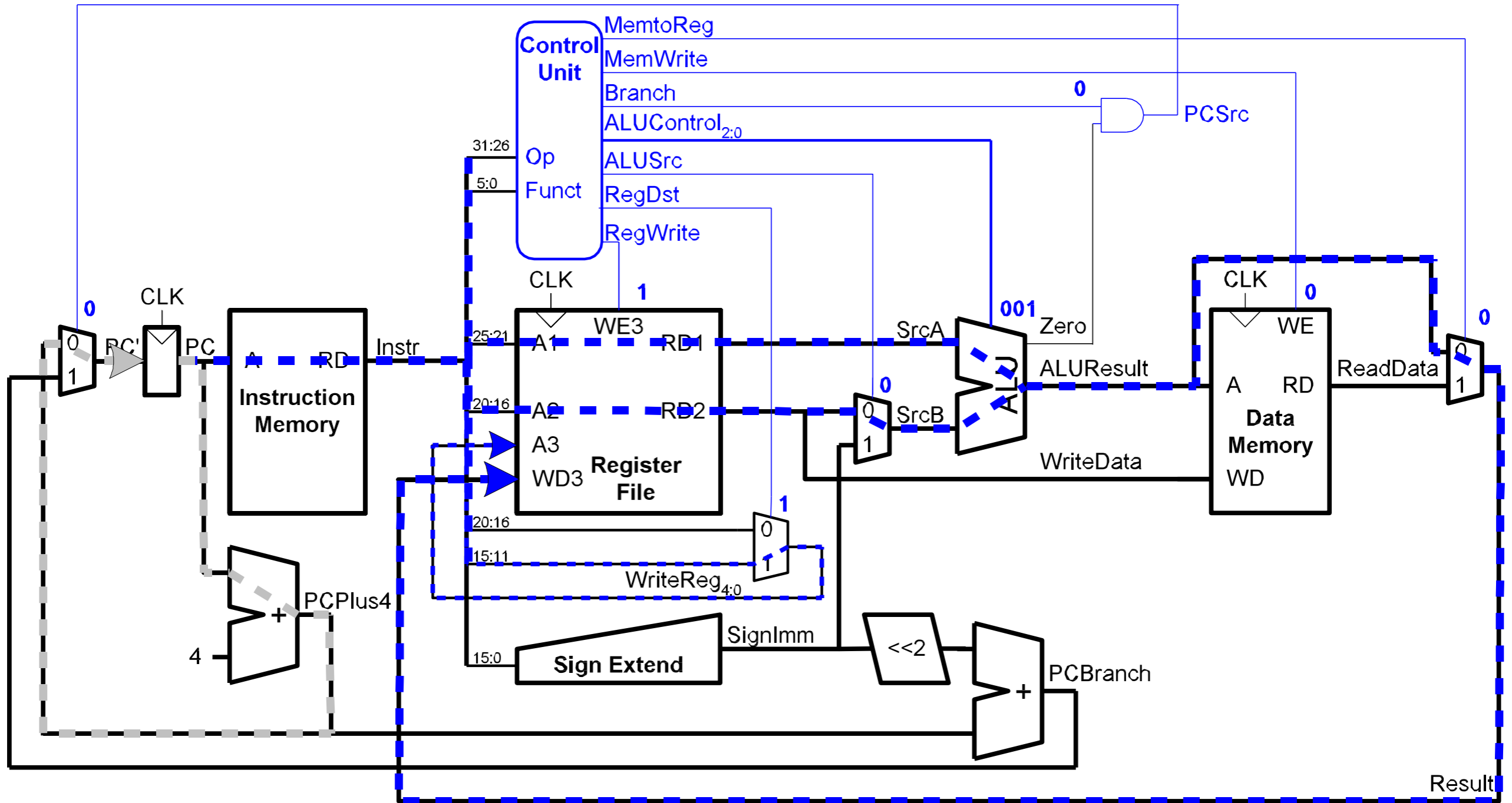
Κύρια Μονάδα Ελέγχου

Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp _{1:0}
R-type	0 0 0 0 0 0	1	1	0	0	0	0	10
lw	1 0 0 0 1 1	1	0	1	0	0	1	00
sw	1 0 1 0 1 1	0	X	1	0	1	X	00
beq	0 0 0 1 0 0	0	X	0	1	0	X	01



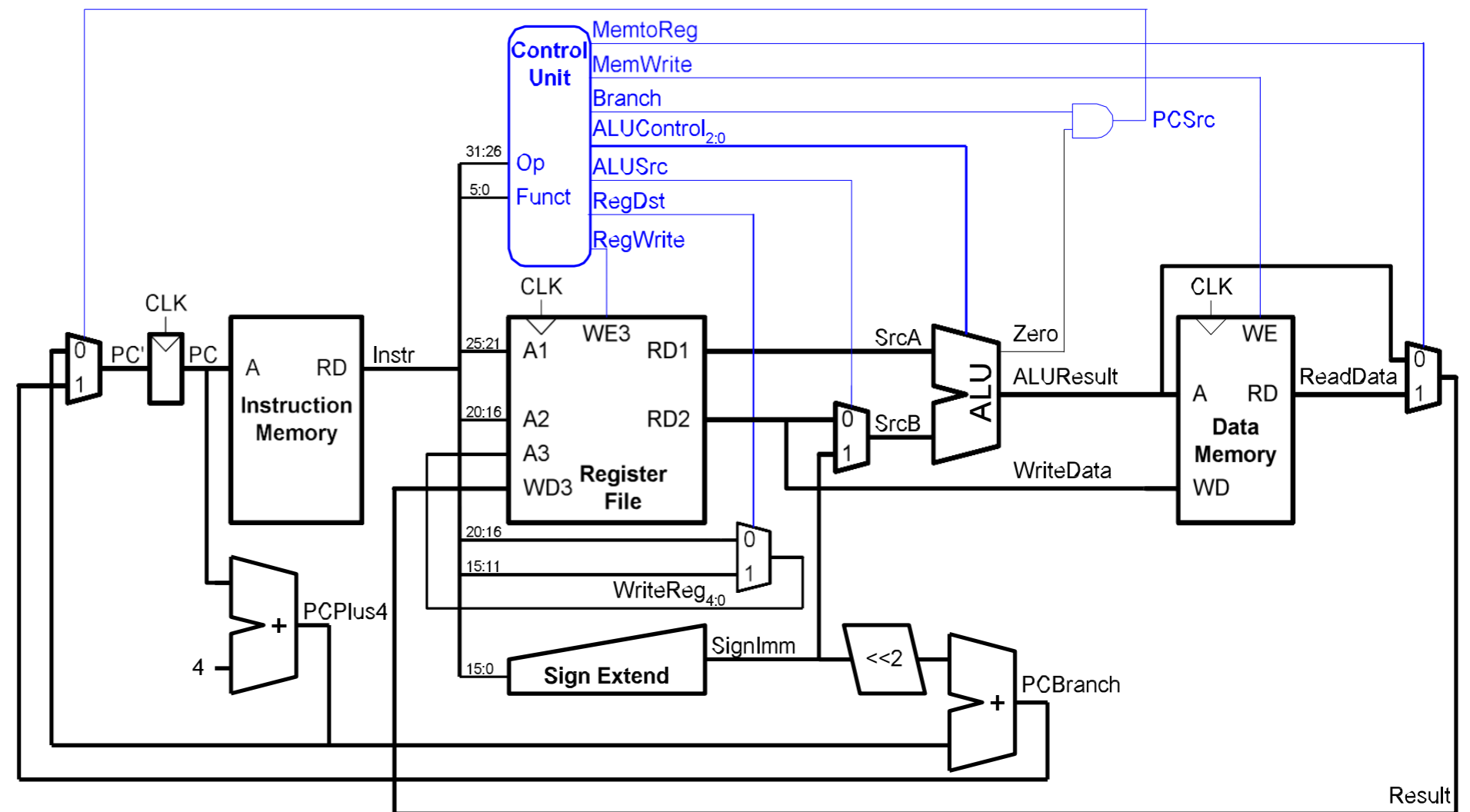
Υλοποίηση εντολής *or*

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



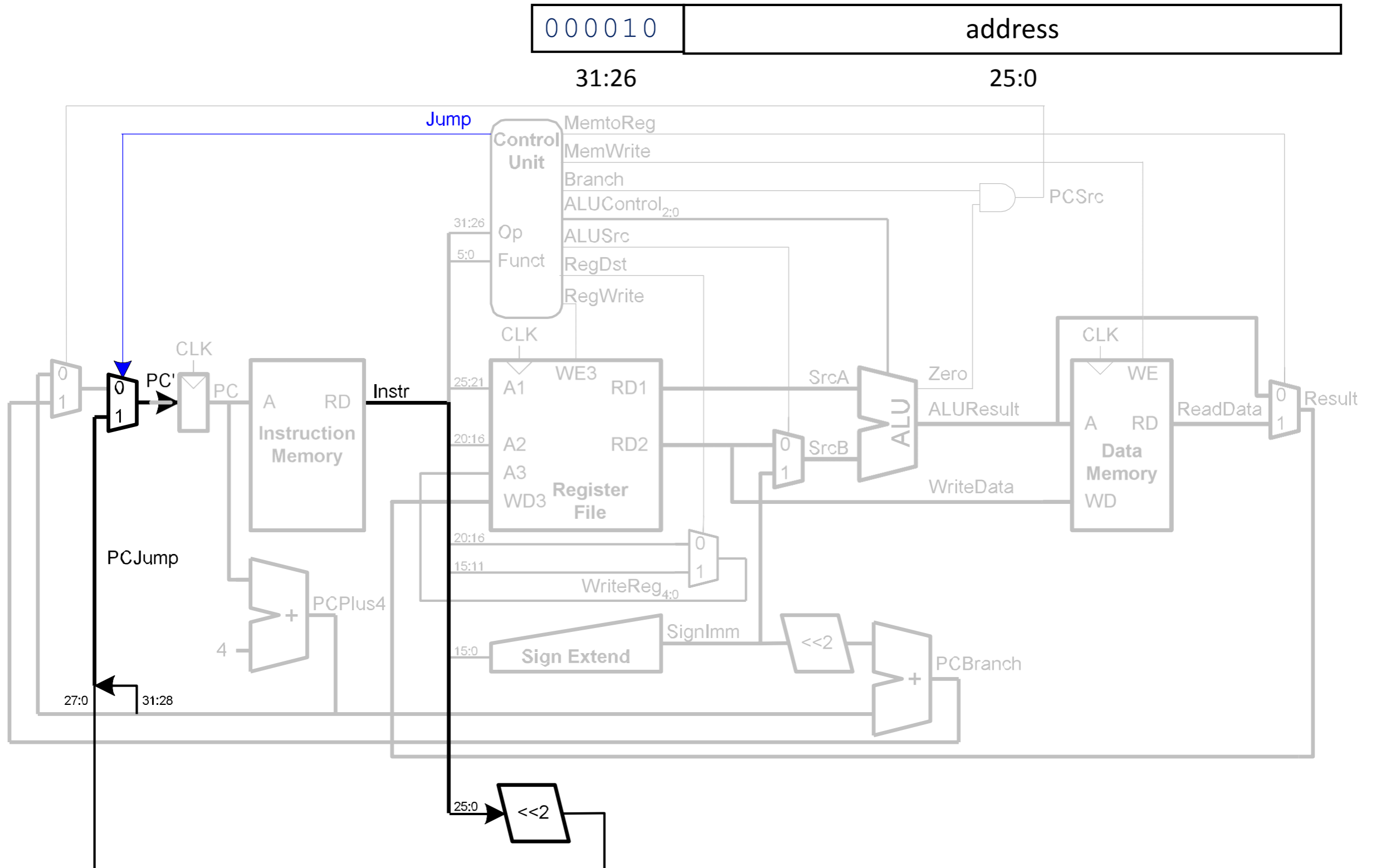
Υλοποίηση εντολής addi

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



Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp _{1:0}
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

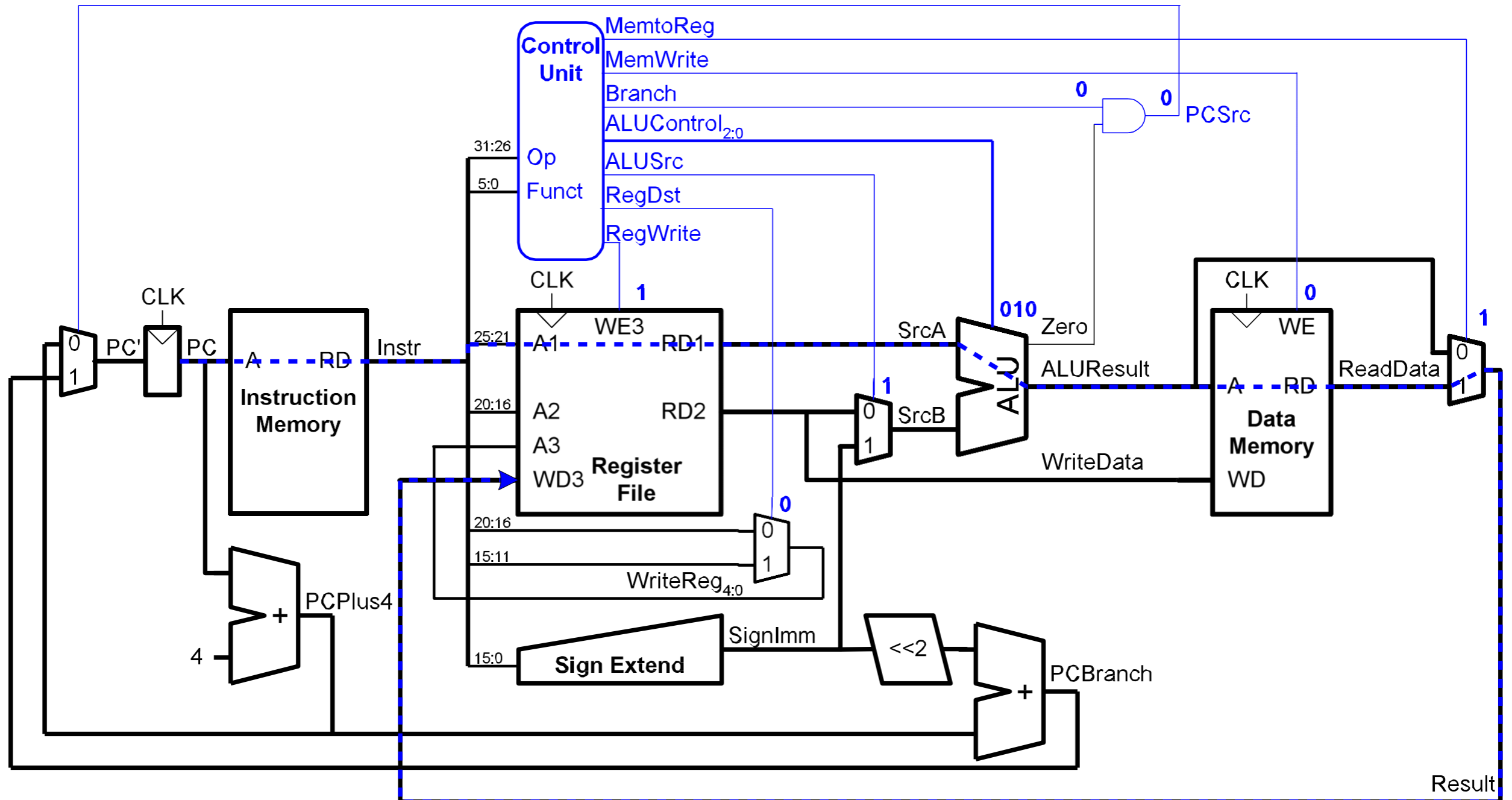
Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemToReg	ALUOp _{1:0}	Jump
R-type	0 0 0 0 0 0	1	1	0	0	0	0	1 0	0
lw	1 0 0 0 1 1	1	0	1	0	0	1	0 0	0
sw	1 0 1 0 1 1	0	x	1	0	1	x	0 0	0
beq	0 0 0 1 0 0	0	x	0	1	0	x	0 1	0
addi	0 0 1 0 0 0	1	0	1	0	0	0	0 0	0
j	0 0 0 0 1 0	0	x	x	x	0	x	x x	1

Η Απόδοση του υπολογιστή μας

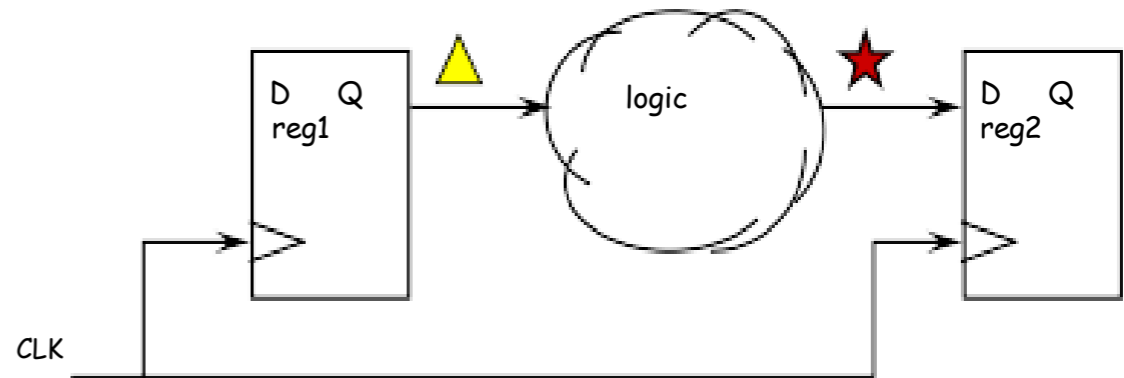
$$\begin{aligned}\text{Χρόνος CPU} &= \text{Αριθμός Εντολών στο Πρόγραμμα} \times \text{CPI} \times \text{Περίοδος Ρολογιού} \\ &= \frac{\text{Αριθμός Εντολών} \times \text{CPI}}{\text{Συχνότητα Ρολογιού}}\end{aligned}$$

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:

$$T_c = t_{pcq_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

- 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

$$\begin{aligned} T_c &= t_{pcq_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup} \\ &= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ ps} \\ &= 925 \text{ ps}] = 0.925 \text{ ns} \\ F_c &= 1/T_c = 1.08 \text{ GHz} \end{aligned}$$

Παράδειγμα υπολογισμού χρόνου εκτέλεσης

$$\begin{aligned}\text{Χρόνος CPU} &= AE \times \text{CPI} \times \text{Περίοδος ΡΡολογιο} \\ &= \frac{AE \times \text{CPI}}{\text{Συχνότητα ΡΡολογιο}}\end{aligned}$$

- Για ένα πρόγραμμα με 100 δις εκτελούμενων εντολών και $\text{CPI} = 1$

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