ΗΜΥ 314 – Εργαστήριο Αρχιτεκτονικής Υπολογιστών

Εργαστήριο 1

Single-Cycle MIPS

Μικροεπεξεργαστής MIPS: Υλοποίηση Ενός Κύκλου

Μαρία Κ. Μιχαήλ
Αναπληρώτρια Καθηγήτρια, ΗΜΜΥ

Εργαστηριακή Άσκηση 1

- Η περιγραφή της εργαστηριακής άσκησης γίνεται στο ακόλουθο PDF αρχείο, που είναι αναρτημένο στο Blackboard:

ECE314_LabExercise_1

- Αφού μελετήσετε **ενδελεχώς** το πιο πάνω αρχείο, προχωρήστε στα επόμενα slides…

Εργαστηριακή Άσκηση 1

Supported MIPS32 Instructions

25 supported instructions

Singe-cycle implementation does **NOT** support:
jal jr lui

Pipelined implementation does **NOT** support:
j jal jr lui

Add	add
Add Immediate	addi
Add Imm. Unsigned	addiu
Add Unsigned	addu
And	and
And Immediate	andi
Branch On Equal	beq
Branch On Not Equal	bne
Jump	j
Jump And Link	jal
Jump Register	jr
Load Upper Imm.	lui
Load Word	lw
Nor	nor
Or	or
Or Immediate	ori
Set Less Than	slt
Set Less Than Imm.	slti
Set Less Than Imm. Unsigned	sltiu
Set Less Than Unsig.	sltu
Shift Left Logical	sll
Shift Right Logical	srl
Store Word	sw
Subtract	sub
Subtract Unsigned	subu

Εργαστηριακή Άσκηση 1 – Μέρος Α

Supported MIPS32 Instructions

25 supported instructions

Singe-cycle implementation does **NOT** support:
jal jr lui

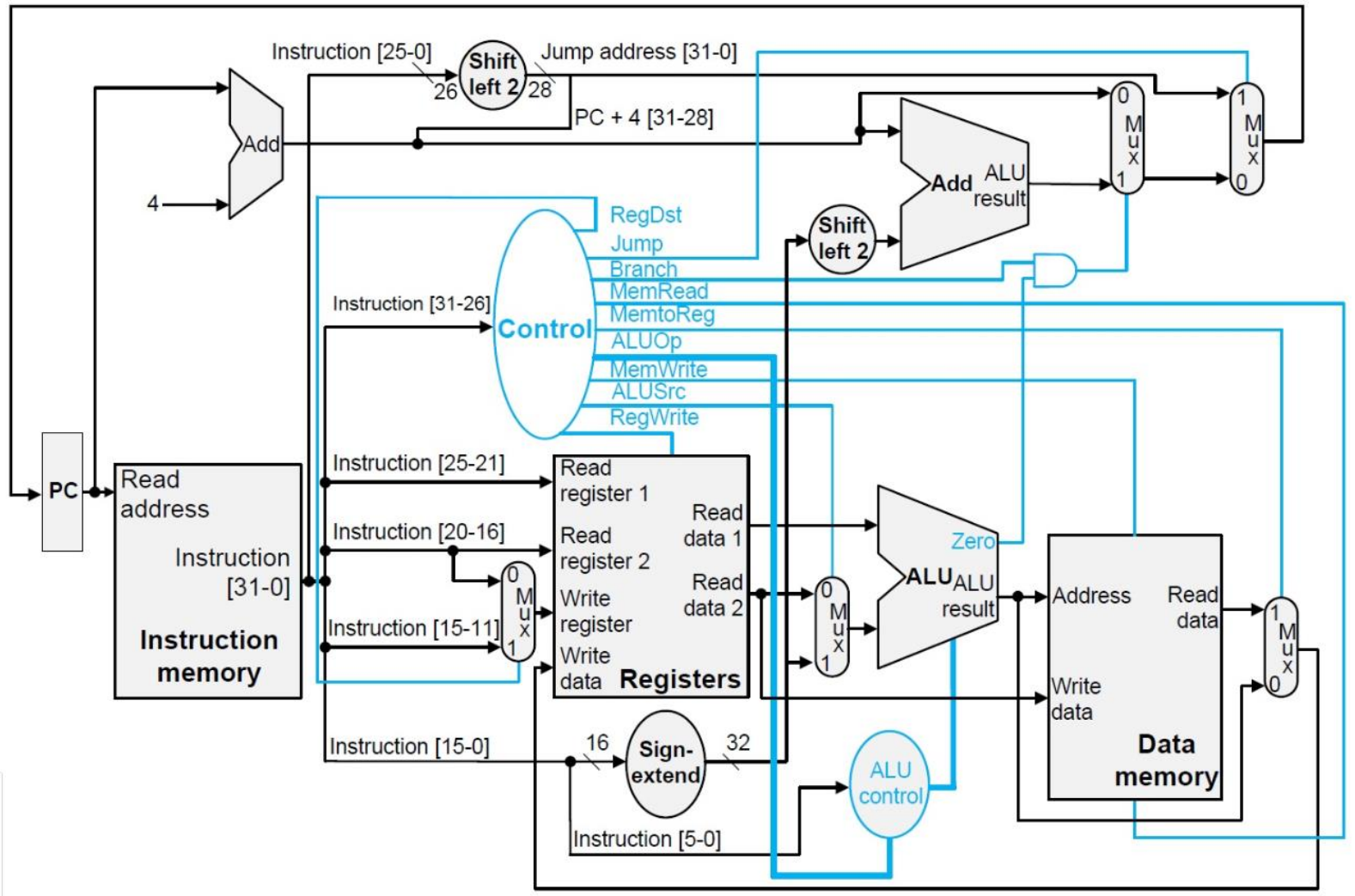
Pipelined implementation does **NOT** support:
j jal jr lui

Add	add
Add Immediate	addi
Add Imm. Unsigned	addiu
Add Unsigned	addu
And	and
And Immediate	andi
Branch On Equal	beq
Branch On Not Equal	bne
Jump	j
Jump And Link	jal
Jump Register	jr
Load Upper Imm.	lui
Load Word	lw
Nor	nor
Or	or
Or Immediate	ori
Set Less Than	slt
Set Less Than Imm.	slti
Set Less Than Imm. Unsigned	sltiu
Set Less Than Unsig.	sltu
Shift Left Logical	sll
Shift Right Logical	srl
Store Word	sw
Subtract	sub
Subtract Unsigned	subu

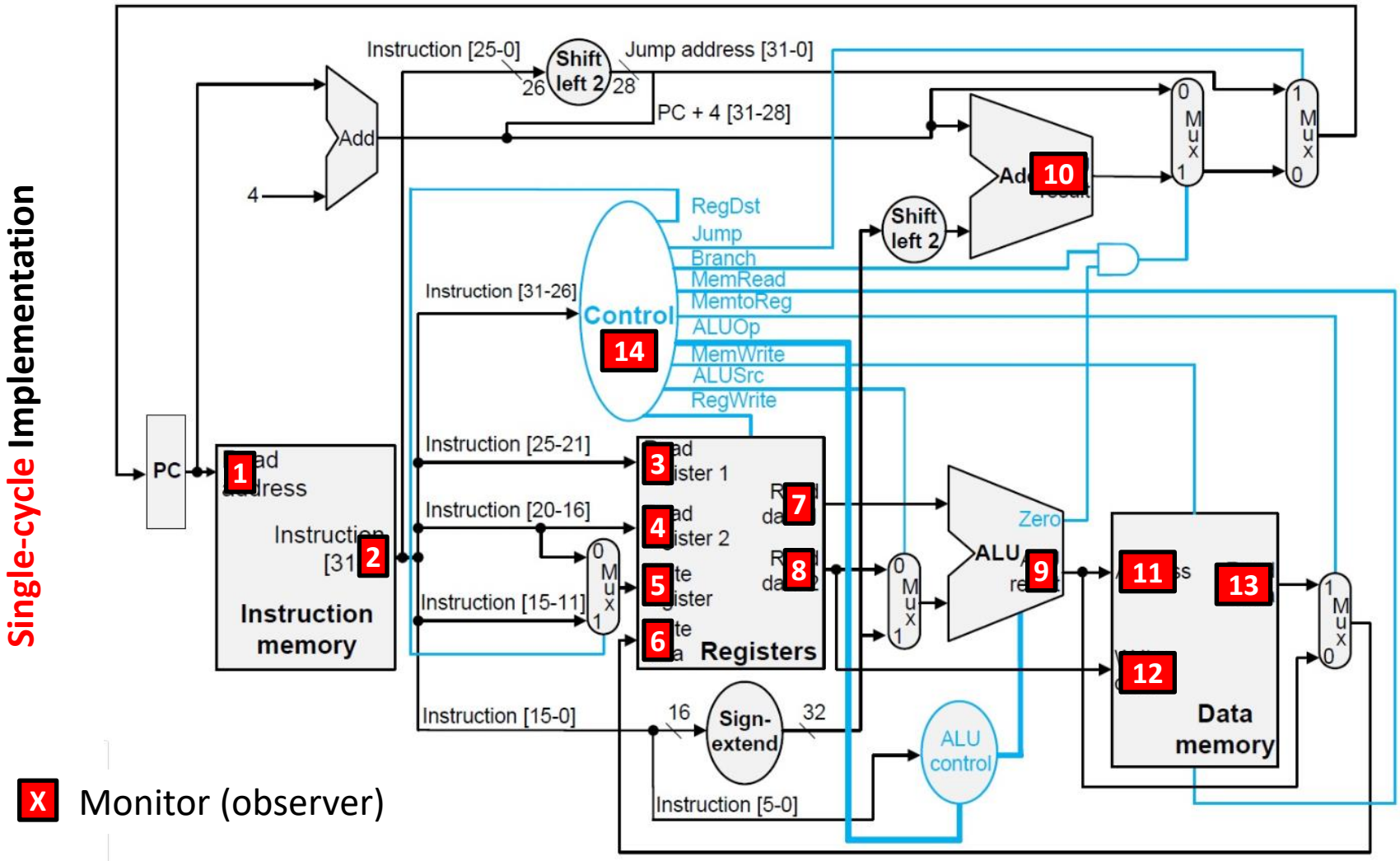
Part A

Εργαστηριακή Άσκηση 1

Single-cycle Implementation



Εργαστηριακή Άσκηση 1



Διαδικαστικά της ημέρας της εργαστηριακής εξέτασης

Η πιο κάτω διαδικασία ισχύει για **όλα τα εργαστήρια κατά τα οποία είναι προγραμματισμένη εργαστηριακή εξέταση**

- Π.χ., η εξέταση για το Μέρος Α του Εργαστηρίου 1 είναι προγραμματισμένη για την **Τετάρτη, 05 Φεβρουαρίου 2025**, κατά τη διάρκεια του εργαστήριου, δηλ., 13:00-16:00
- Την ημέρα της κάθε εξέτασης, **όλοι/όλες οι φοιτητές/τριες οφείλουν να βρίσκονται στην αίθουσα του εργαστήριου μέχρι τις 13:00 το αργότερο**
- Η εξέταση θα πραγματοποιείται **στα πρώτα 20 λεπτά** (δηλ. 13:00 – 13:20) και μετά θα ξεκινά η εκφώνηση του επόμενου Εργαστηρίου και η σχετική διάλεξη
- Δεν υπάρχει δυνατότητα εκπρόθεσμων εξετάσεων

Επανάληψη σχετικής ύλης από ΗΜΥ 212

Σχεδιασμός επεξεργαστή MIPS
με υλοποίηση **ενός κύκλου**

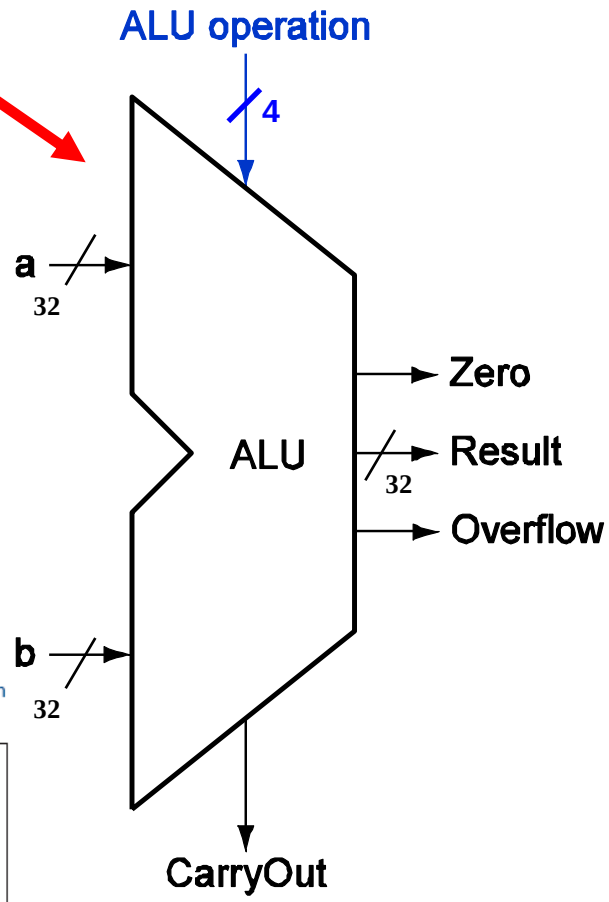
Ο Μικροεπεξεργαστής: Διάδρομος Δεδομένων και Μονάδα Ελέγχου

- Ας δούμε την υλοποίηση ενός κύκλου του μικροεπεξεργαστή MIPS
- Απλοποιημένη (για να είναι πιο εύπεπτη η ανάλυση εδώ), περιέχει μόνο **9 εντολές**:
  - Εντολές προσπέλασης μνήμης: **`lw, sw`**
  - Εντολές αριθμητικής/λογικής: **`add, sub, and, or, slt`**
  - Εντολές ελέγχου ροής: **`beq, j`**
- **Η δική σας υλοποίηση θα περιέχει περισσότερες εντολές** (βλέπε προηγούμενα)
- Γενική Υλοποίηση:
  - Χρησιμοποιείται ο μετρητής προγράμματος (PC) για να παρέχει την διεύθυνση της εντολής
  - Ανάκτηση εντολής από την μνήμη
  - Ανάγνωση καταχωρητών
  - Αποκωδικοποίηση εντολής για να αποφασιστεί η πράξη που θα εκτελεστεί
- ΟΛΕΣ οι εντολές χρησιμοποιούν το ALU, αφού διαβαστούν οι καταχωρητές (εκτός από μία εντολή, ποια;) [The Jump instruction]

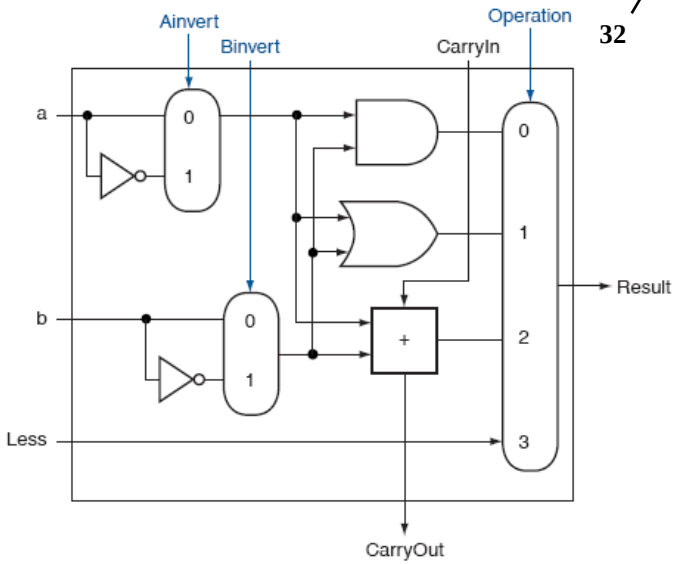
  Γιατί; προσπέλασης μνήμης; αριθμητικής/λογικής; ελέγχου ροής; [Memory access uses the ALU to calculate source/destination address; branches for comparison and new-address calculation]

Οικουμενικό σύμβολο ALU (32-bit)

Σήματα ελέγχου ALU (*Ainvert, Binvert, Operation(2)*)		Λειτουργία
0_{10}	0000	and
1_{10}	0001	or
2_{10}	0010	add
6_{10}	0110	subtract
7_{10}	0111	slt
12_{10}	1100	NOR



The 2-bit "*Operation*" control signal to control the 4-input MUX



ALU -- Περιληπτικά

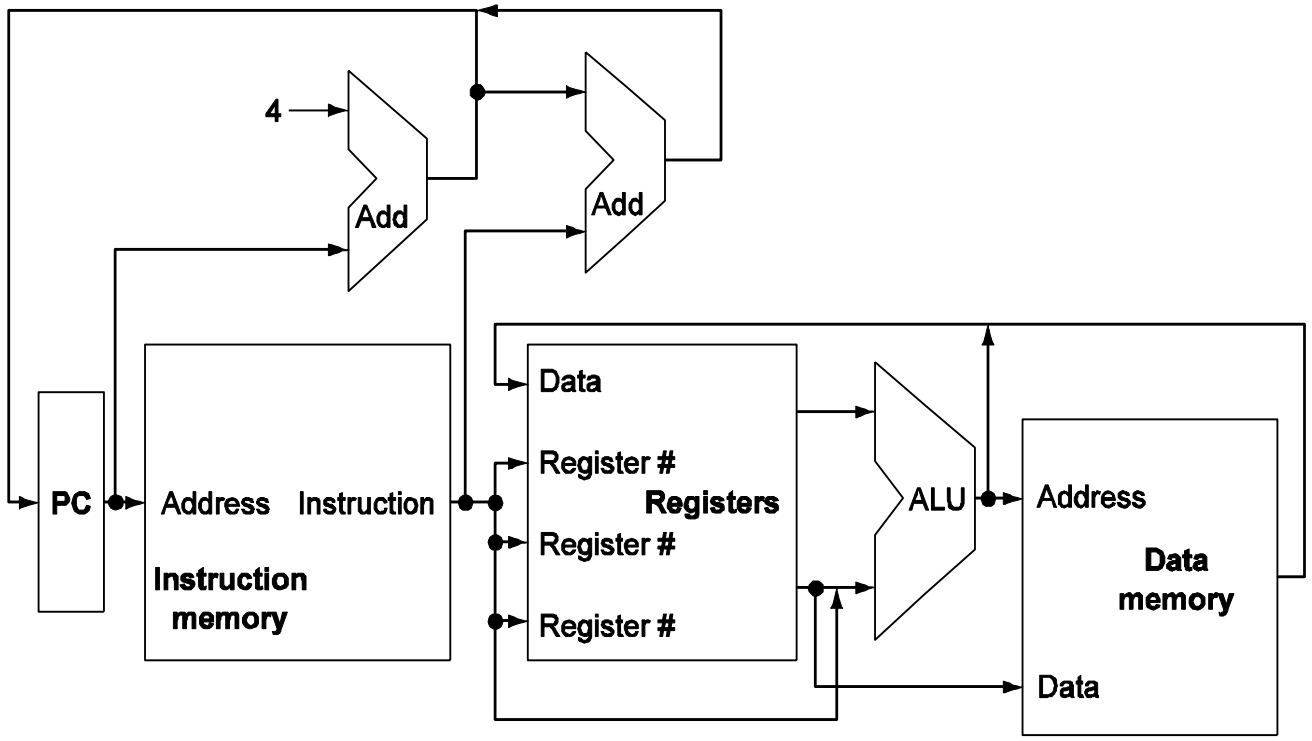
- Οι γλώσσες περιγραφής υλικού (*hardware description languages*) επιτρέπουν στους σχεδιαστές να αυτοματοποιήσουν την διαδικασία σχεδιασμού του υλικού (δείτε παράδειγμα πιο κάτω)

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0; goes anywhere
   always @(ALUctl, A, B) //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1:0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0; //default to 0, should not happen;
      endcase
endmodule
```

FIGURE B.4.3 A Verilog behavioral definition of a MIPS ALU. This could be synthesized using a module library containing basic arithmetic and logical operations.

Υλοποίηση MIPS σε υψηλό επίπεδο

Περιληπτική/Απλοποιημένη Μορφή: Βασικές λειτουργικές μονάδες και οι διασυνδέσεις τους

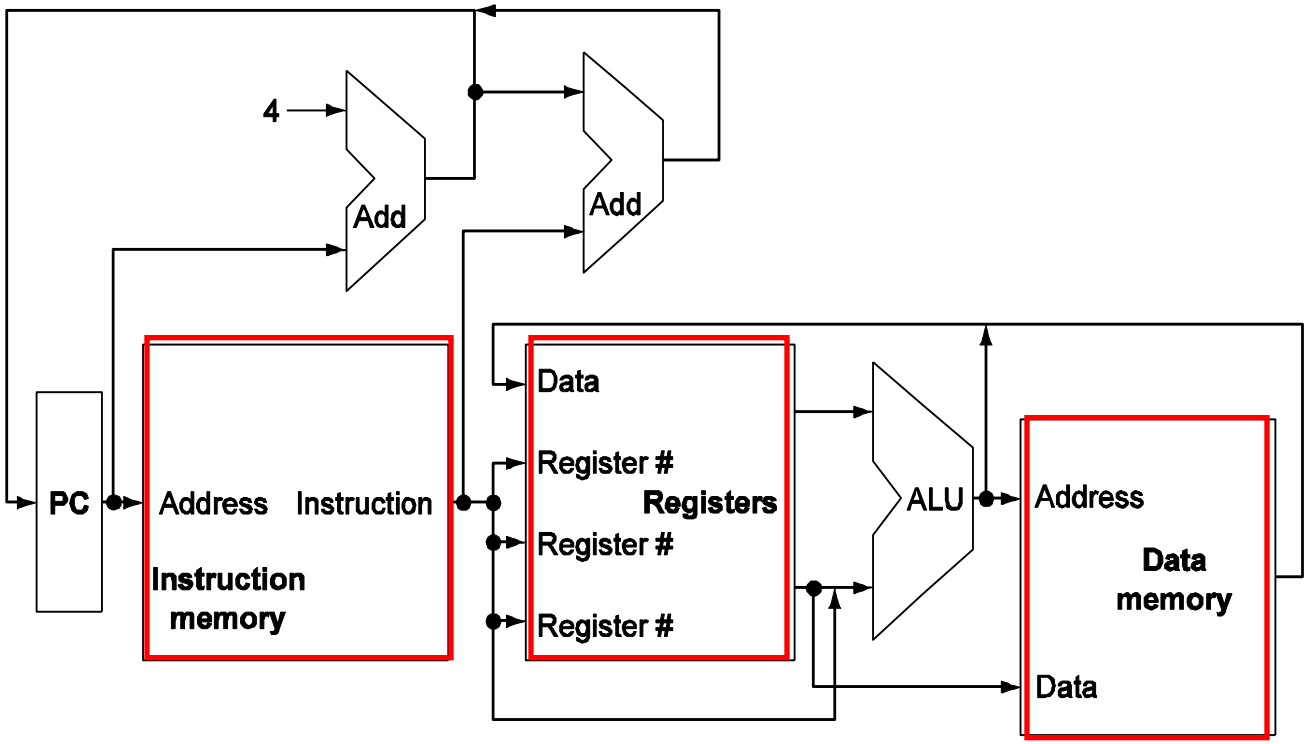


2 κατηγορίες λειτουργικών μονάδων:

- Στοιχεία που χειρίζονται τιμές δεδομένων

Υλοποίηση MIPS σε υψηλό επίπεδο

Περιληπτική/Απλοποιημένη Μορφή: Βασικές λειτουργικές μονάδες και οι διασυνδέσεις τους



2 κατηγορίες λειτουργικών μονάδων:

- Στοιχεία που χειρίζονται τιμές δεδομένων
- Στοιχεία που περιέχουν την κατάσταση/μνήμη

Υλοποίηση με έναν κύκλο ρολογιού

- Σε αυτήν την υλοποίηση, ***κάθε*** εντολή ξεκινά να εκτελείται στην ακμή του ρολογιού (*clock edge*) και ολοκληρώνει την εκτέλεση της στην επόμενη ακμή του ρολογιού
  → **κάθε εντολή απαιτεί πάντα 1 κύκλο του ρολογιού (CPI=1)**

- Αυτό απλοποιεί την κατανόηση του σχεδιασμού, αλλά δεν είναι πρακτικό, από άποψης απόδοσης. Γιατί;

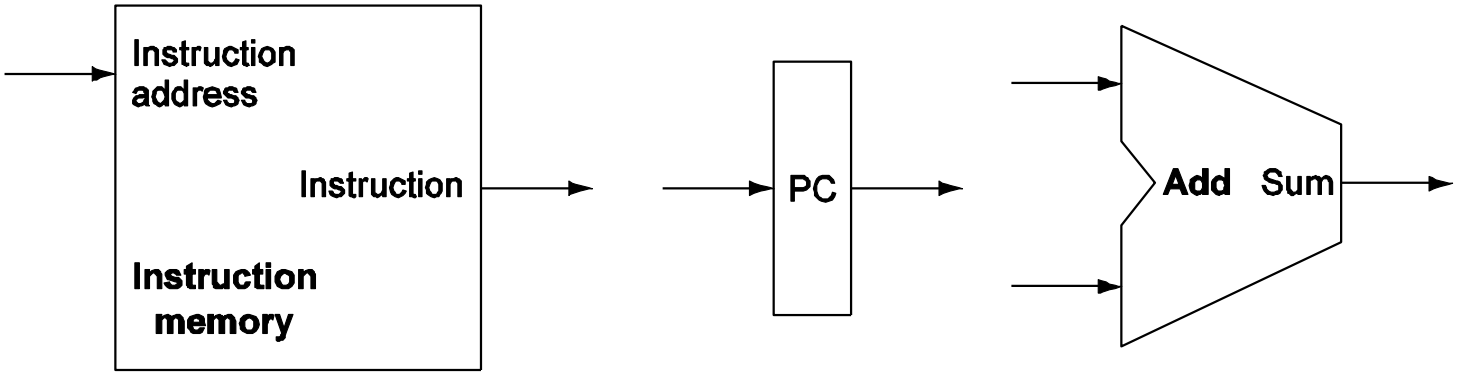
Σχεδιασμός Διαδρόμου Δεδομένων

- **Στοιχείο Διαδρόμου Δεδομένων** (*Datapath element*) λειτουργική μονάδα που χρησιμοποιείται για την εκτέλεση ή κράτηση των δεδομένων σε ένα επεξεργαστή. Στον MIPS, περιλαμβάνει την μνήμη εντολών, την μνήμη δεδομένων, τους καταχωρητές, το ALU και κάποιους αθροιστές.

- Ας εξετάσουμε τις λειτουργικές μονάδες που απαιτούνται ανά κατηγορία εντολών:
  - Διατήρηση του PC
  - Εντολές τύπου R
  - Εντολές Μνήμης τύπου I
  - Εντολές Διακλάδωσης τύπου I και J

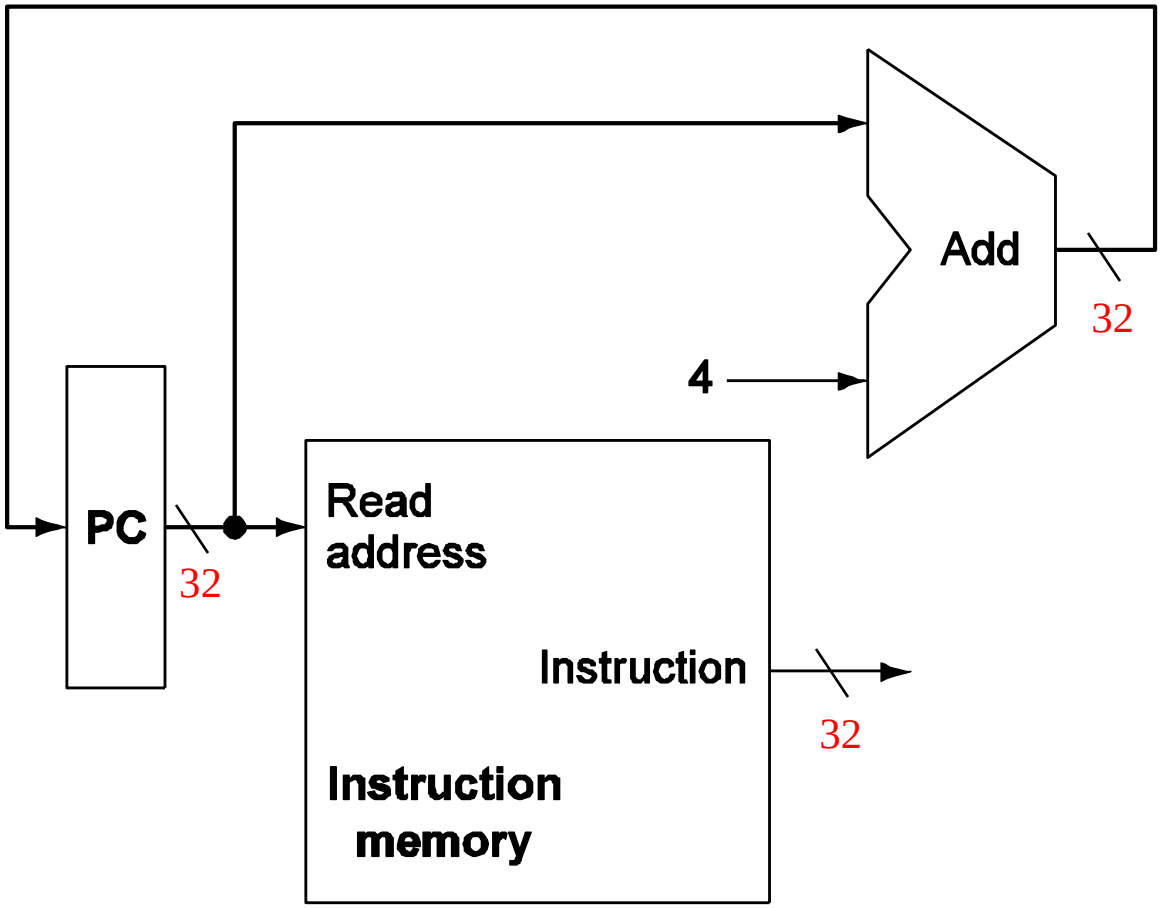
Μετρητής Προγράμματος (PC)

- Απαραίτητες Λειτουργικές Μονάδες:



- Βήματα:
  1. Προσκόμιση εντολής από τη μνήμη
  2. Αύξηση του PC κατά 4 για να ετοιμαστεί για την επόμενη εντολή

Μετρητής Προγράμματος (PC) (συν.)

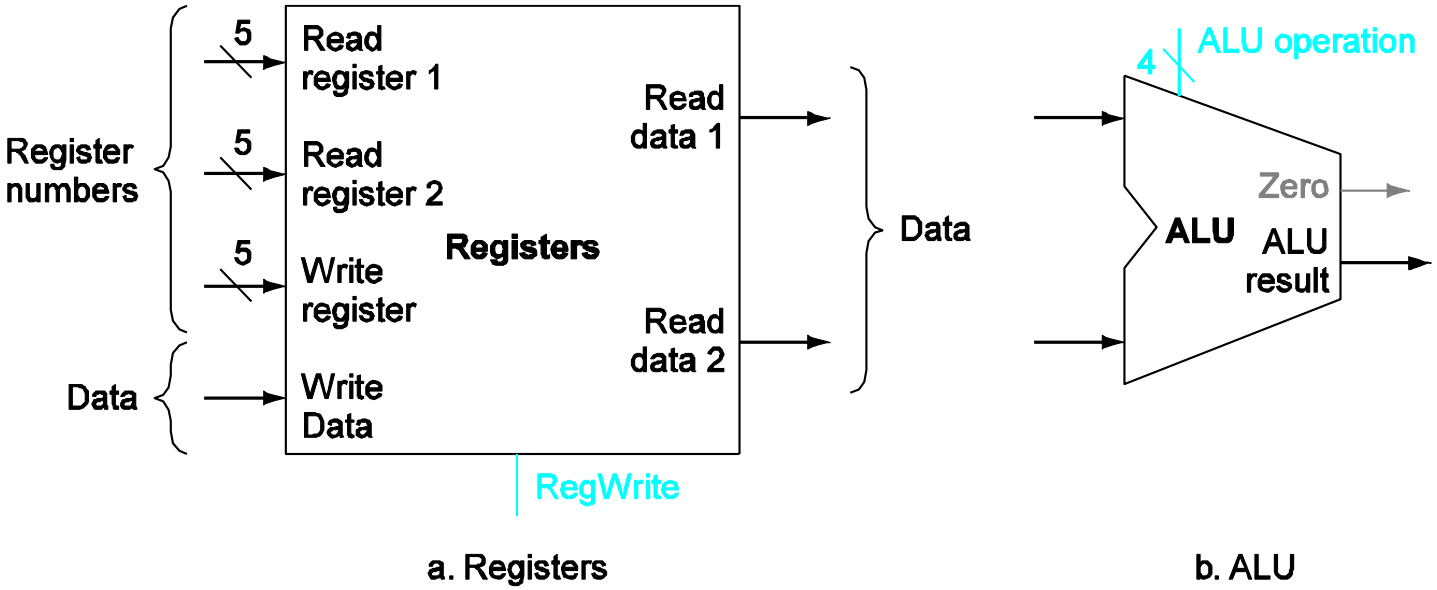


Εντολές τύπου R (R-type)

- Περιλαμβάνει **`add, sub, and, or, slt`**
- Χειρίζεται 2 καταχωρητές για ανάγνωση και 1 για γραφή
  - π.χ., **`add  $s0, $t1, $t2`**
  - **`$s0`** καταχωρητής γραφής και **`$t1, $t2`** καταχωρητές ανάγνωσης

- Οι 32 καταχωρητές γενικής χρήσεως του επεξεργαστή αποθηκεύονται στο αρχείο καταχωρητών (*register file*)
  - Το αρχείο καταχωρητών είναι ένα σύνολο από 32 32-μπιτους καταχωρητές όπου ο ***κάθε*** ένας μπορεί να διαβαστεί ή/και να ενημερωθεί, με τον ορισμό του αντίστοιχου αριθμού του καταχωρητή στο αρχείο
  - Το αρχείο καταχωρητών περιέχει την κατάσταση των καταχωρητών (*register state*) της μηχανής

Εντολές τύπου R (R-type)

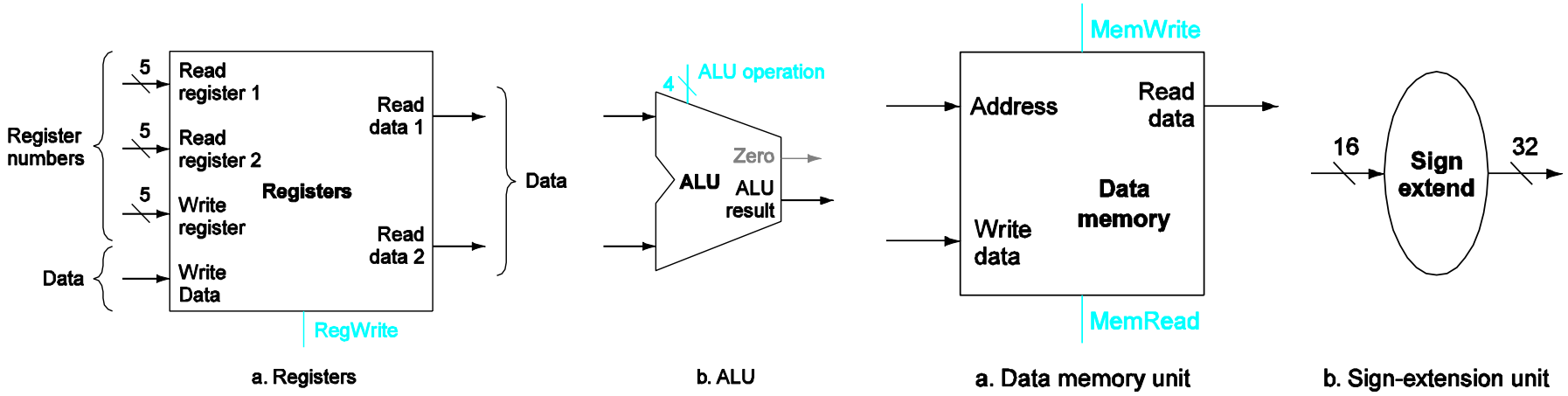


- Γιατί το σήμα επιλογής του ALU είναι 4-μπιτο;
- Μπορούμε να διαβάσουμε και να γράψουμε στον ΙΔΙΟ καταχωρητή κατά τη διάρκεια ενός μόνο κύκλου;

Εντολές Μνήμης (`lw`, `sw`)

- `lw $t1, offset_value($t2)`
- `sw $t1, offset_value($t2)`

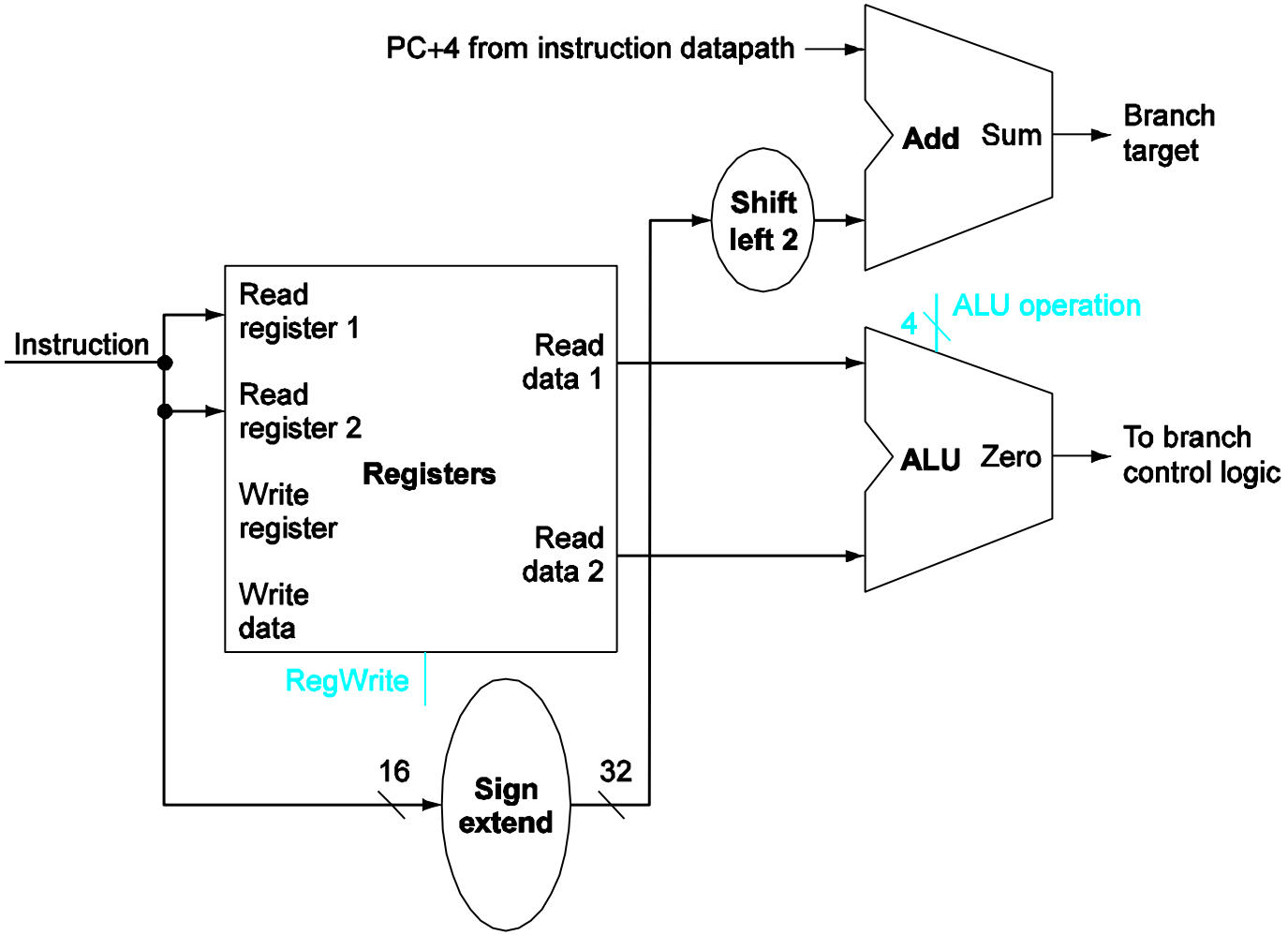
- Απαραίτητες Λειτουργικές Μονάδες:
  - Αρχείο Καταχωρητών **και** ALU.
  - Μονάδα για επέκταση πρόσημου (***sign-extension***) του 16-μπιτου offset σε προσημασμένη 32-μπιτη τιμή
  - Επίσης, μνήμη δεδομένων.



Διακλαδώσεις

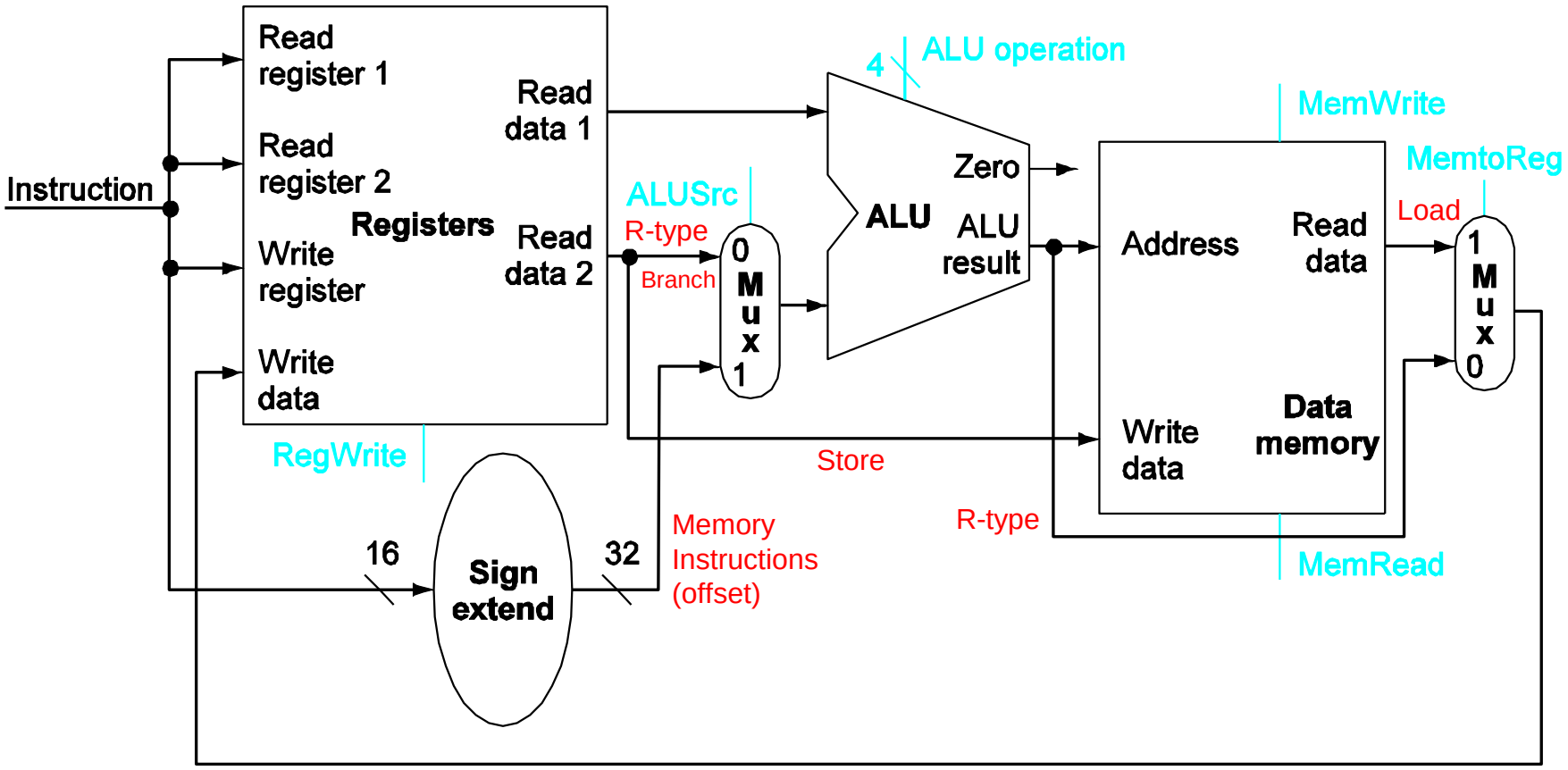
- **`beq $t1, $t2, branch_offset`**
- Αν **`$t1 == $t2`**, χρησιμοποιεί το branch_offset για να υπολογίσει την διεύθυνση της διακλάδωσης (*branch target address*) = τα καινούρια περιεχόμενα του PC = επόμενη εντολή προς εκτέλεση
- MIPS branch target address = PC+4 + branch_offset*4
- Ποια λειτουργία του ALU χρειαζόμαστε για να υπολογίσουμε τη MIPS branch target address; Έχουμε το PC+4 κάπου αλλού;
- Πρέπει να αποφασιστεί η επόμενη εντολή αναλόγως του αποτελέσματος της ισότητας (***branch taken or not taken***)
- Επομένως, ο διάδρομος δεδομένων μιας διακλάδωσης πρέπει να εκτελέσει 2 λειτουργίες:
  - Σύγκριση περιεχομένων των καταχωρητών
  - Υπολογισμός της διεύθυνσης της διακλάδωσης

Διακλαδώσεις (συν.)



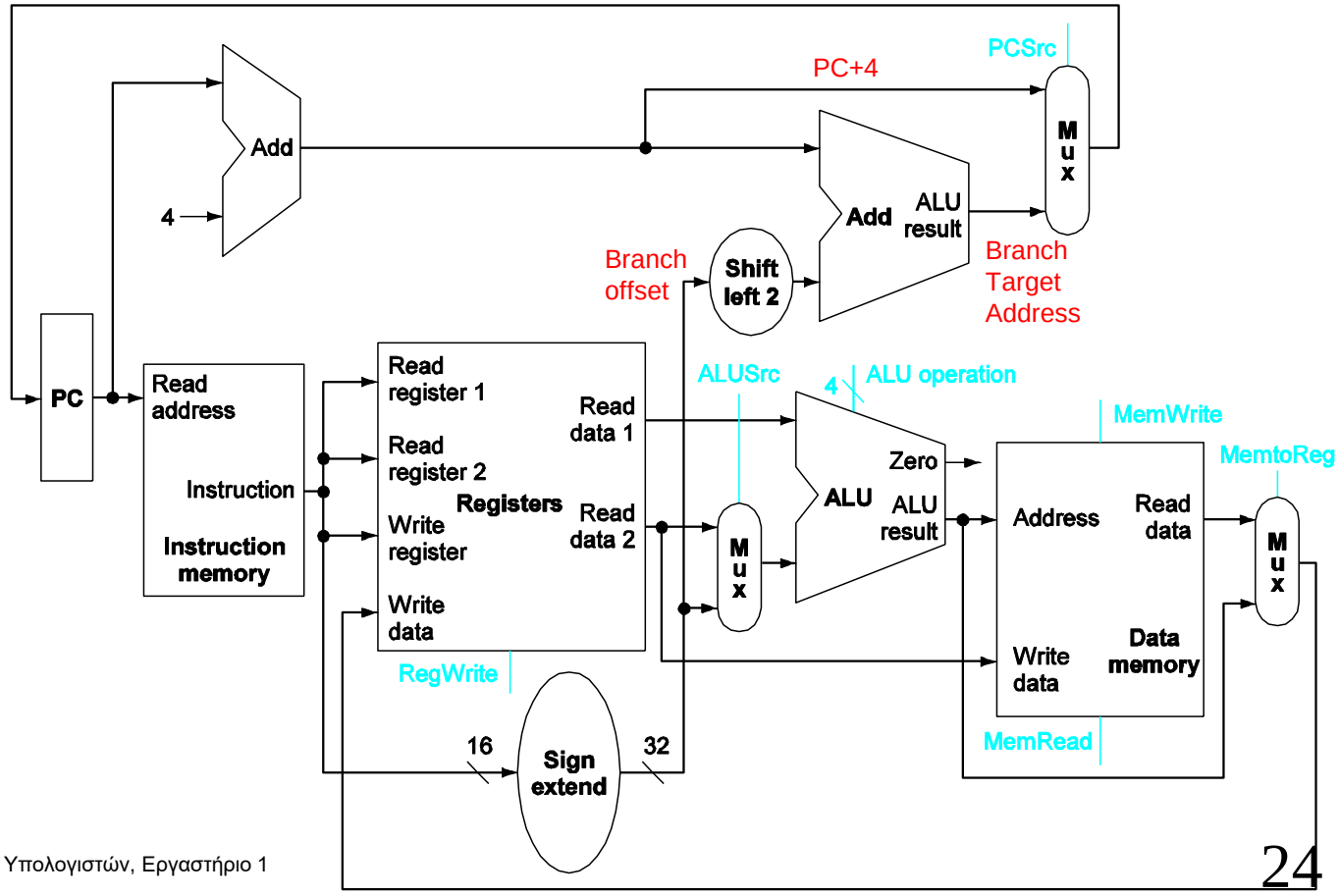
Πως νομίζετε ότι λειτουργεί το jump (`j jump_offset`);

Εντολές Μνήμης και τύπου R



Συνδυάζοντας τα όλα…

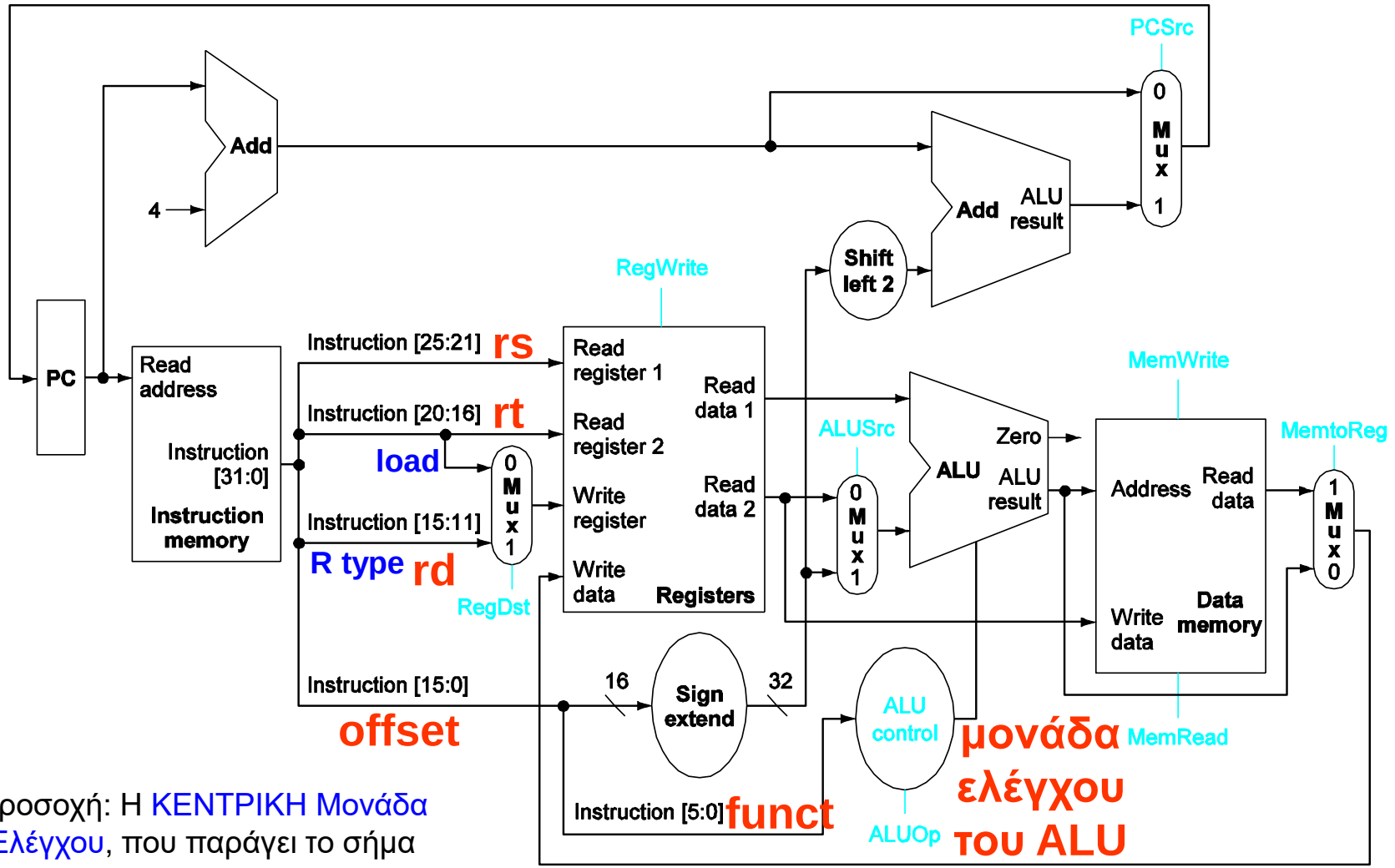
- Χρήση πολυπλεκτών για κατάλληλες επιλογές
- Αφού ολόκληρη η εντολή εκτελείται σε 1 κύκλο, καμία μονάδα του datapath δεν μπορεί να χρησιμοποιηθεί περισσότερο από 1 φορά
  → ανάγκη για ξεχωριστές μνήμες για δεδομένα και εντολές
- Υποστήριξη εντολής `j` θα προστεθεί αργότερα



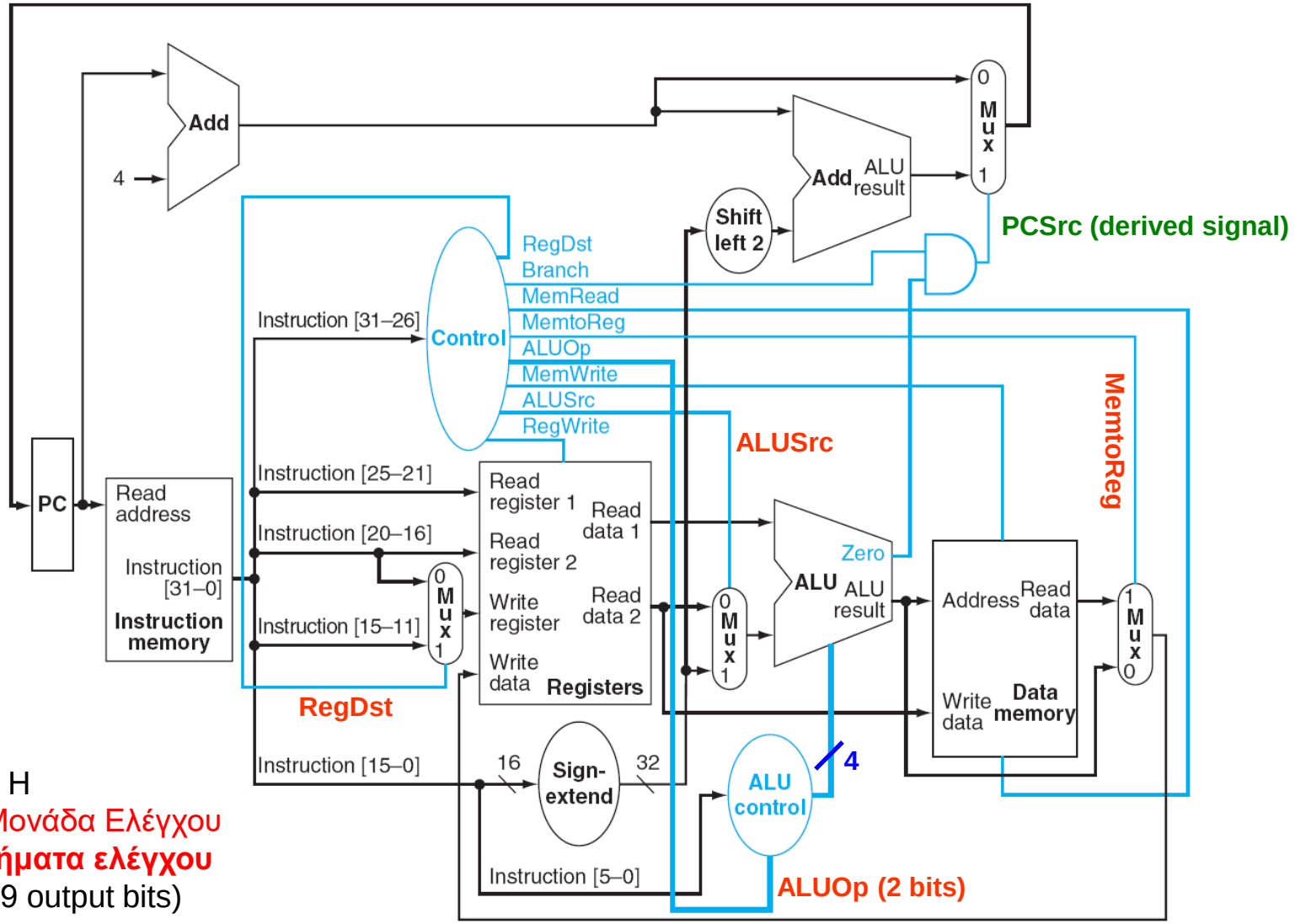
Υλοποίηση Μονάδας Ελέγχου

- Χρειαζόμαστε τώρα μια απλή μονάδα ελέγχου για να συντονίζει και να συγχρονίζει την εκτέλεση των διαφόρων εντολών
- Θεωρήστε μόνο τις πιο κάτω εντολές:
  `lw, sw, beq, add, sub, or, and, slt`
- Η εντολή `j` θα προστεθεί αργότερα
- Η **μονάδα ελέγχου** (***control unit***) χρησιμοποιείται για:
  - Επιλογή της λειτουργίας που θα εκτελεστεί (ALU, μνήμη, κτλ.)
  - Έλεγχο της ροής των δεδομένων (ανάθεση τιμών στα σήματα επιλογής των πολυπλεκτών)

Ενσωμάτωση Μονάδας Ελέγχου ALU



Προσοχή: Η ΚΕΝΤΡΙΚΗ Μονάδα Ελέγχου, που παράγει το σήμα **ALUOp**, δεν φαίνεται σε αυτό το διάγραμμα.



Η ΚΕΝΤΡΙΚΗ Μονάδα Ελέγχου παράγει **8 σήματα ελέγχου** (συνολικά 9 output bits)

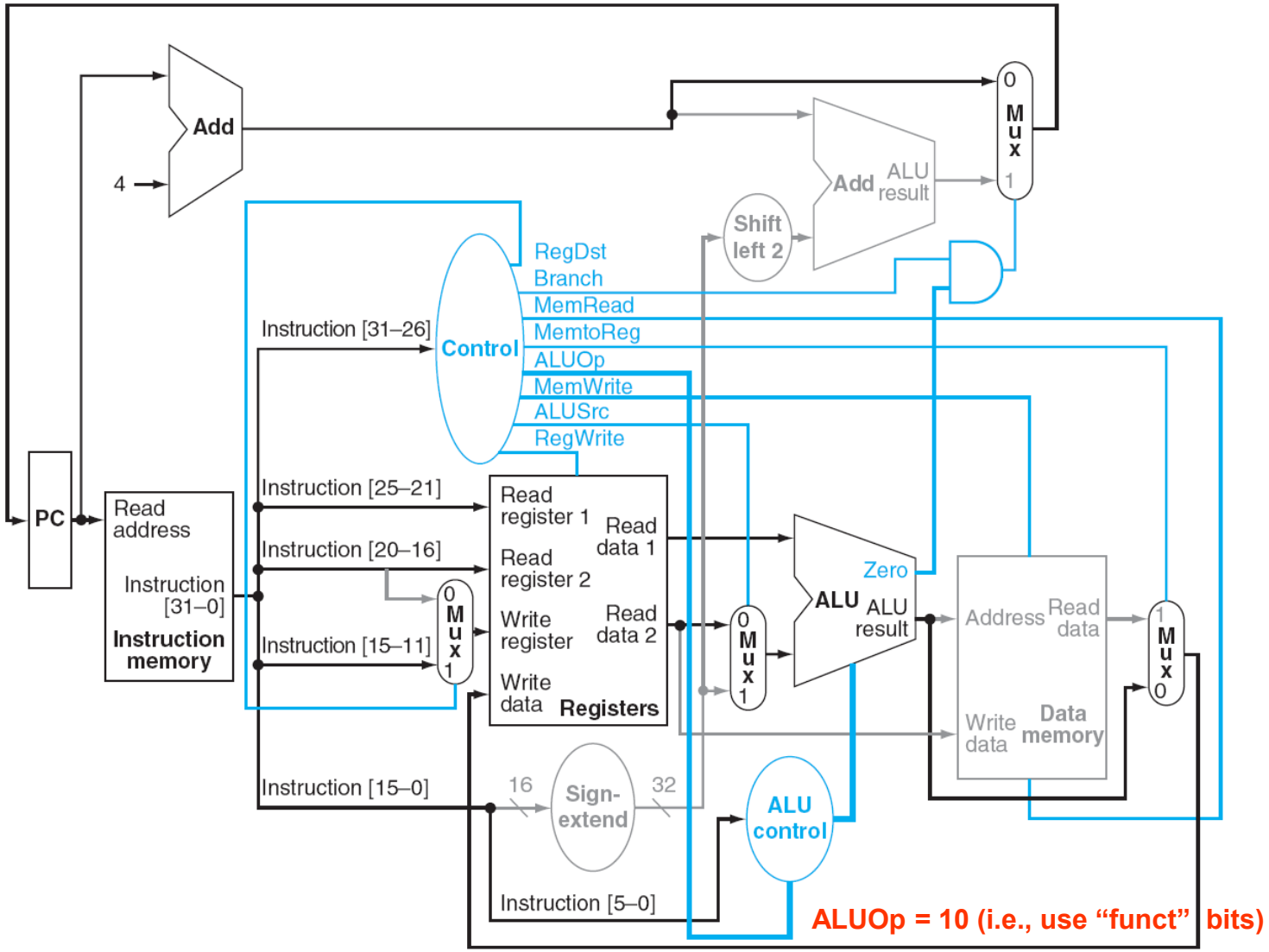
Εντολή	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 1: `add $t1, $t2, $t3`

Βασικά Βήματα:

1. Προσκόμιση εντολής και αύξηση του PC
2. Ανάγνωση `$t2` και `$t3` από το αρχείο καταχωρητών, η κύρια μονάδα ελέγχου θέτει τις τιμές των σημάτων ελέγχου
3. Το ALU λειτουργεί βάση του *funct* (ALUOp=10) → προσθέτει Read Data1 και ReadData2
4. Το αποτέλεσμα του ALU γράφεται πίσω στο αρχείο καταχωρητών στον `$t1`

Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 1: `add $t1, $t2, $t3`

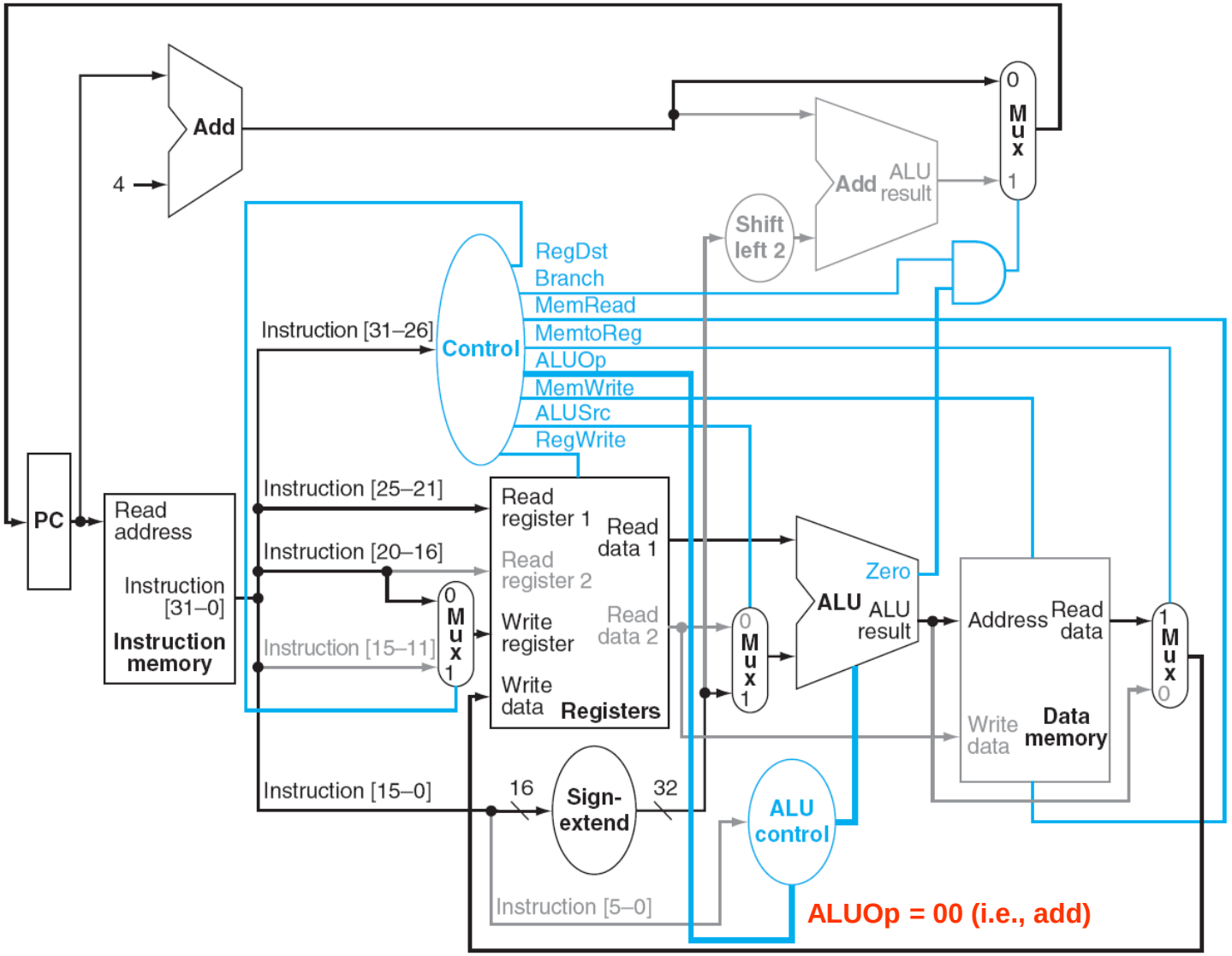


Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 2: `lw $t1, offset($t2)`

Βασικά Βήματα:

1. Προσκόμιση εντολής και αύξηση του PC
2. Ανάγνωση **`$t2`** από το αρχείο καταχωρητών, η κύρια μονάδα ελέγχου θέτει τις τιμές των σημάτων ελέγχου
3. Το ALU λειτουργεί βάση του *ALUOp* (ALUOp=00) → προσθέτει Read Data1 και **`offset`**
4. Το αποτέλεσμα του ALU είναι η διεύθυνση των δεδομένων στη μνήμη
5. Η έξοδος της μνήμης δεδομένων γράφεται στο αρχείο καταχωρητών στον **`$t1`**

Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 2: `lw $t1, offset($t2)`

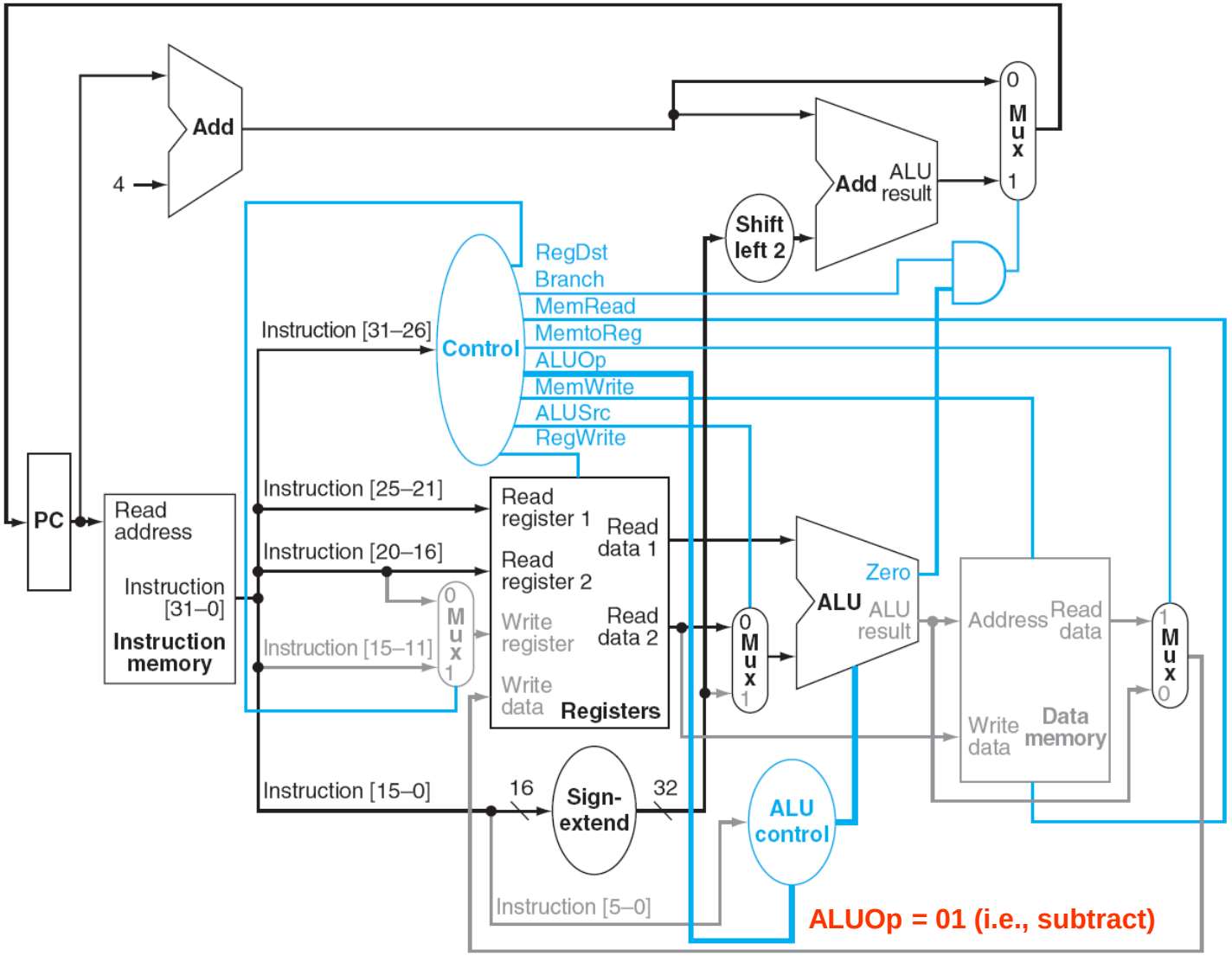


Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 3: `beq $t1, $t2, offset`

Βασικά Βήματα:

1. Προσκόμιση εντολής και αύξηση του PC
2. Ανάγνωση **`$t1`** και **`$t2`** από το αρχείο καταχωρητών, η κύρια μονάδα ελέγχου θέτει τις τιμές των σημάτων ελέγχου
3. Το ALU λειτουργεί βάση του *ALUOp* (ALUOp=01) → αφαιρεί ReadData2 από ReadData1 **`($t1-$t0<0`**). Προσθέτει το *sign-extended offset*, μετατοπισμένο αριστερά (*shifted left*) κατά 2 θέσεις, στο PC+4 για να υπολογίσει την διεύθυνση της διακλάδωσης
4. Ενεργοποιείται το ALU *zero* (=1) για να αποθηκευτεί η διεύθυνση διακλάδωσης στον PC

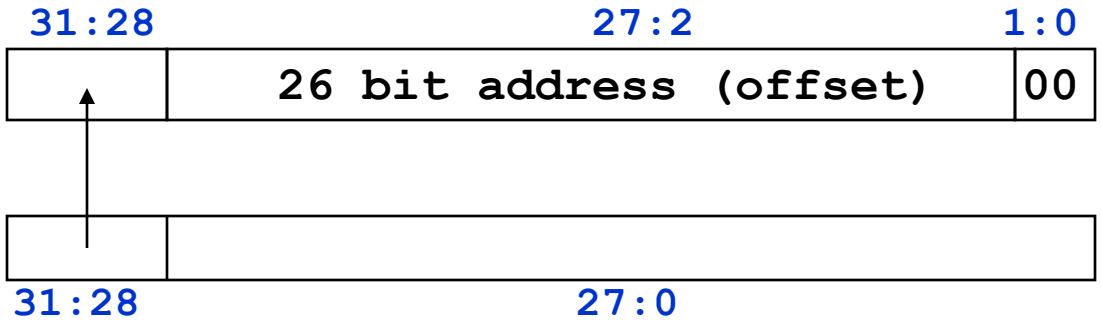
Ροή εντολής στο διάδρομο δεδομένων
Παράδειγμα 3: `beq $t1, $t2, offset`



Υλοποίηση ανεξάρτητου άλματος (*unconditional jump* – `j`)

- Παρόμοιο με το branch, στην πραγματικότητα είναι πιο απλό!
- Jump address = 4 MSB του (PC+4) : offset*4, όπου ":" υποδηλώνει αλληλουχία

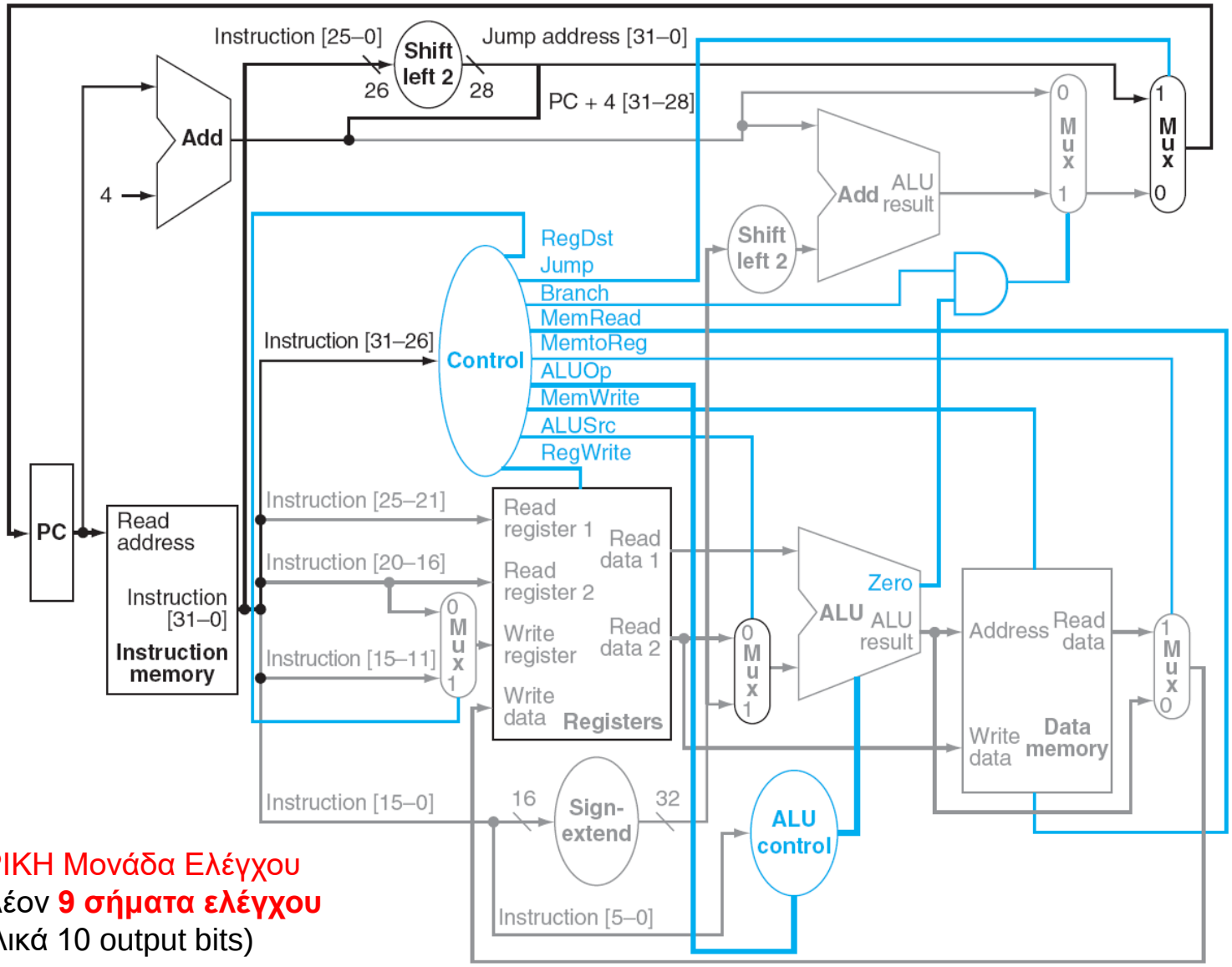
Εντολή J-type



Jump address

PC

Υλοποίηση ανεξάρτητου άλματος (*unconditional jump* – `j`)



Η ΚΕΝΤΡΙΚΗ Μονάδα Ελέγχου παράγει πλέον **9 σήματα ελέγχου** (συνολικά 10 output bits)

Σύνοψη: Εργαστηριακή Άσκηση 1

Single-cycle Implementation