

## Single-cycle υλοποίηση:

Διάρκεια κύκλου ίση με τη μεγαλύτερη εντολή-worst case delay (εδώ η  $lw$ ) = χαμηλή απόδοση!

Αντιβαίνει με αρχή: Κάνε την πιο απλή περίπτωση γρήγορη (ίσως και εις βάρος των πιο «σύνθετων» περιπτώσεων π.χ. (load 32bit constant to reg)).

Κάθε functional unit χρησιμοποιείται μια φορά σε κάθε κύκλο: ανάγκη για πολλαπλό hardware = κόστος υλοποίησης!

## Λύση: Multicycle υλοποίηση

Μικρότεροι κύκλοι ρολογιού, από τις καθυστερήσεις των επιμέρους functional units

*Τα σχήματα είναι από το βιβλίο  
COD2e/Patterson-Hennessy*

# Λίγο πριν το Pipeline...

## Multicycle υλοποίηση

Διαιρούμε την εκτέλεση της κάθε εντολής σε βήματα ανάλογα με τον αριθμό των functional units που χρειάζεται

*Κάθε βήμα και ένας ξεχωριστός παλμός ρολογιού*

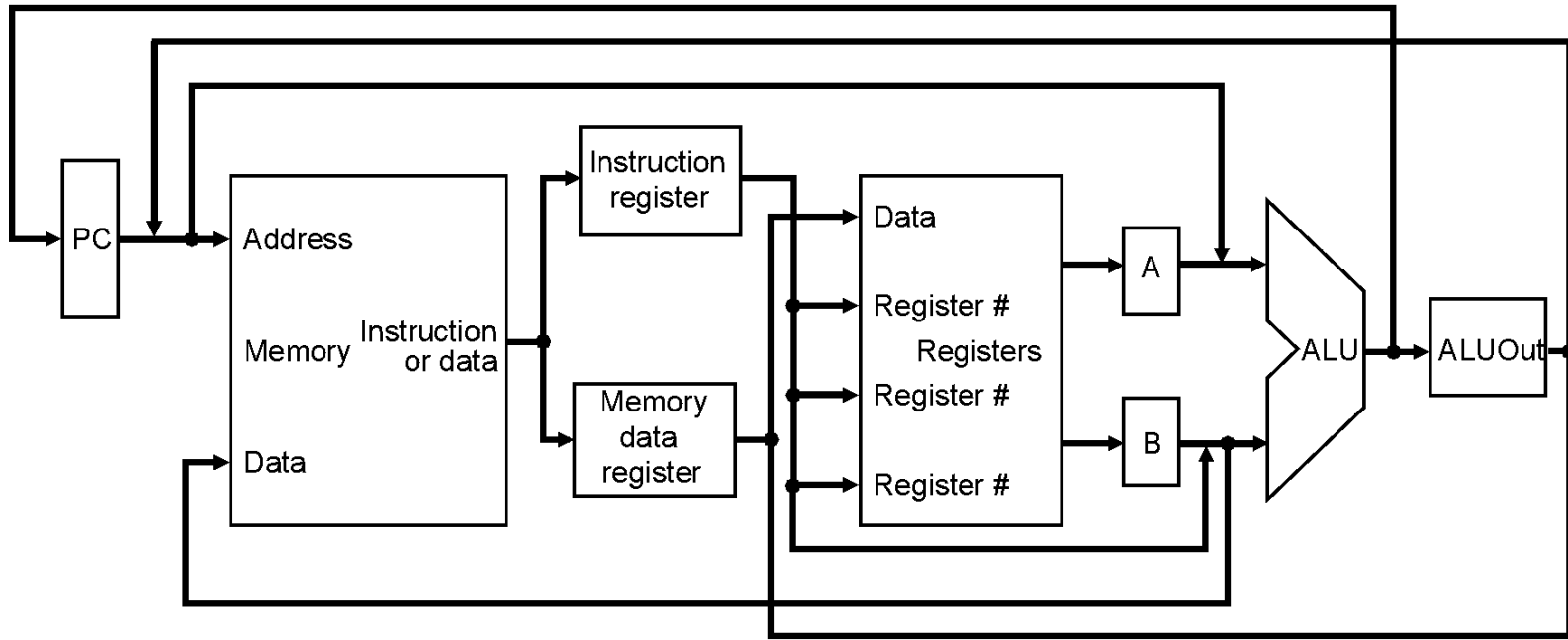
Όταν έχουμε multicycle υλοποίηση, μπορούμε το ίδιο functional unit να το χρησιμοποιήσουμε πολλές φορές στην ίδια εντολή, σε διαφορετικούς όμως κύκλους (οικονομία hardware)

*Οι εντολές διαρκούν μεταβλητό αριθμό κύκλων, άρα μπορούμε να κάνουμε την συνηθισμένη περίπτωση πιο γρήγορη.*

# MIPS Datapath-Multicycle Implementation

1. Χρησιμοποιούμε την ίδια memory unit τόσο για instructions όσο και για data
2. Χρησιμοποιούμε την ίδια ALU (αντί για μια ALU και δύο αθροιστές  $PC+4$  και  $PC+4+address\_offset$ )
3. Μετά από κάθε functional unit υπάρχουν καταχωρητές που κρατάνε το αποτέλεσμα μέχρις ότου το πάρει το επόμενο functional unit (στον επόμενο κύκλο)

# Multicycle Datapath:



Επιπλέον καταχωρητές: IR, MDR, A, B και ALUOut

Υπόθεση:

*«Σε κάθε κύκλο ρολογιού μπορεί να γίνει ένα από τα παρακάτω»*

- ✓ **Memory access**
- ✓ **Register file access (read or write)**
- ✓ **ALU op**

Οτιδήποτε παράγεται από αυτές τις μονάδες, σώζεται σε temporary register

Temporary registers μεταξύ των λειτουργικών μονάδων:

- IR
- MDR
- ALUOut
- A, B (έξοδος register file)

## ΔΕΔΟΜΕΝΑ που:



Θα χρησιμοποιηθούν  
από την ίδια εντολή σε  
επόμενους κύκλους:

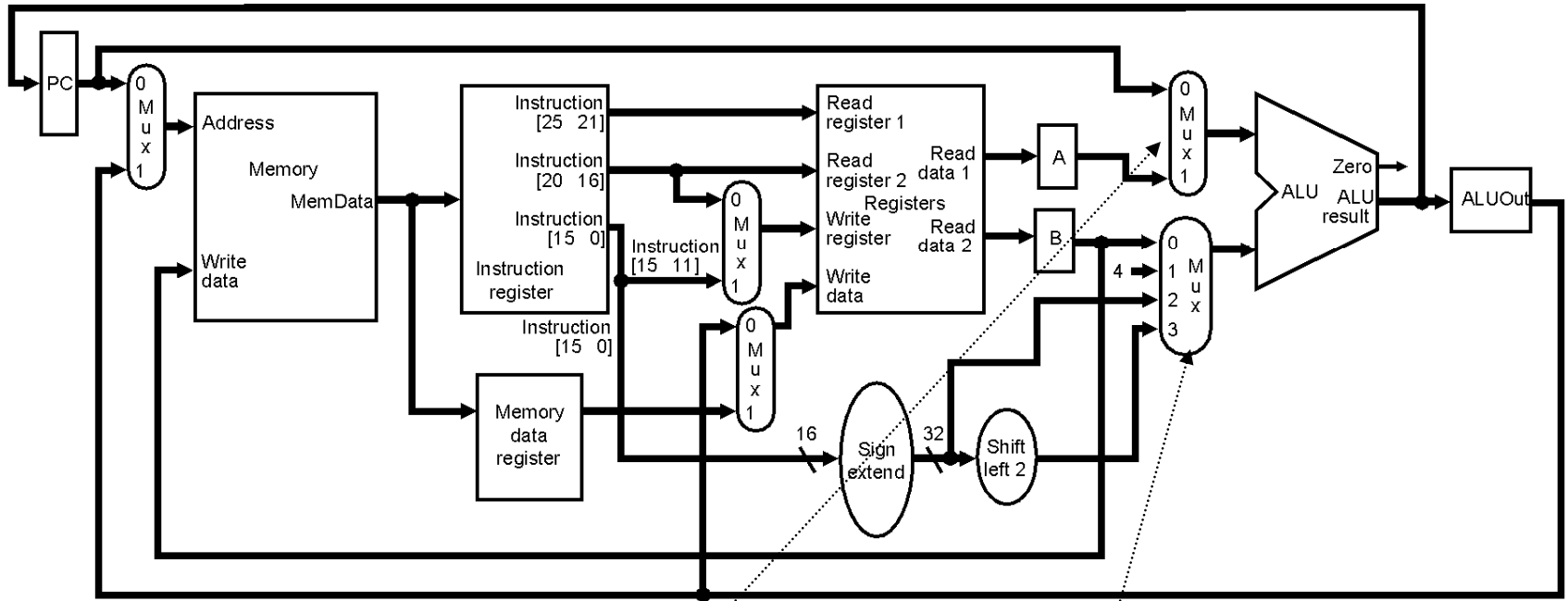
*Σώζονται σε temporary  
registers*

Θα χρησιμοποιηθούν  
από επόμενες εντολές:

*Σώζονται σε:*

- *register file,*
- *memory,*
- *PC*

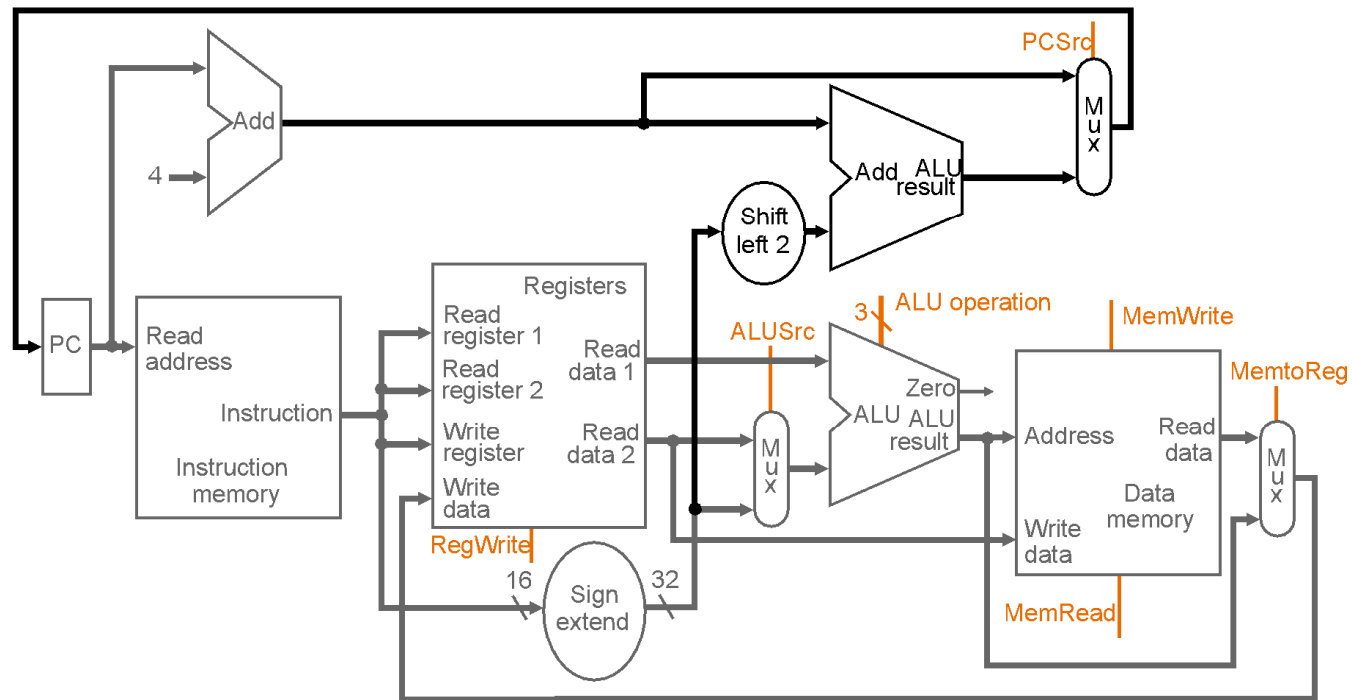
*Programmer visible  
state elements*



Επιλογή μεταξύ PC (για εντολή branch, PC+4) και A (για R-Type)

Επιλογή μεταξύ 4 (PC+4), B (R-Type), sign\_extend offset για I-Type (lw, sw) και branch offset

# Single-cycle datapath:





# Διαίρεση της εκτέλεσης κάθε εντολής σε πολλαπλούς κύκλους:

## 1. Instruction Fetch

«Φέρε την εντολή από τη μνήμη και υπολόγισε τα διεύθυνση ανάκλησης για την επόμενη εντολή»

$$IR = \text{Memory}[PC];$$

$$PC = PC + 4;$$

## 2. Instruction decode and register fetch (reg. File read)

«Διάβασε τους καταχωρητές rs και rt και αποθήκευσέ τους στους A και B αντίστοιχα»

$$A = \text{Reg}[IR[25-21]];$$

$$B = \text{Reg}[IR[20-16]];$$

$$\text{ALUOut} = PC + (\text{sign-extend}(IR[15-0] \ll 2));$$

## **2. Instruction decode and register fetch (reg. File read)**

Οι A και B «γεμίζουν» σε κάθε κύκλο! Πάντα ο IR περιέχει την εντολή από την αρχή μέχρι το τέλος!

Στο βήμα αυτό υπολογίζεται και η διεύθυνση «πιθανού» άλματος και αποθηκεύεται στο καταχωρητή ALUOut (αν πρόκειται για εντολή branch)

**Οι δύο παραπάνω λειτουργίες γίνονται ταυτόχρονα**

**..συνέχεια:**

### 3. Execution, memory address computation or branch completion

*Εδώ για πρώτη φορά, παίζει ρόλο τι είδους εντολή έχουμε*

a) Memory Reference:

$$\text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0]);$$

b) Arithmetic-Logical:

$$\text{ALUOut} = A \text{ op } B;$$

c) Branch:

$$\text{If } (A == B) \text{ PC} = \text{ALUOut};$$

d) Jump:

$$\text{PC} = \text{PC}[31-28] \parallel (\text{IR}[25-0] \ll 2);$$

..συνέχεια:

## 4. Memory Access or R-Type instruction completion

### a) Memory Reference:

MDR = Memory [ALUOut];

ή

Memory [ALUOut] = B;

«διάβασε από τη διεύθυνση που έχει σχηματιστεί στον ALUOut και αποθήκευσε στον MDR (load)»

ή

«διάβασε το B (που πάντα έχει τον destination reg rt) και αποθήκευσέ το στη μνήμη με δνση ALUOut

### β) Arithmetic-Logical:

Reg[IR[15-11]] = ALUOut;

Είναι πάντα το ίδιο, σε όλους τους κύκλους!

**..συνέχεια:**

## 5. Memory read completion (write back step)

$\text{Reg}[\text{IR}[20-16]] = \text{MDR};$

«Γράψε πίσω τα data που είχαν την προηγούμενη φάση αποθηκευτεί στον MDR, στο register file»

