ΤΕΧΝΙΚΕΣ ΑΥΞΗΣΗΣ ΤΗΣ ΑΠΟΔΟΣΗΣ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ I

MIPS

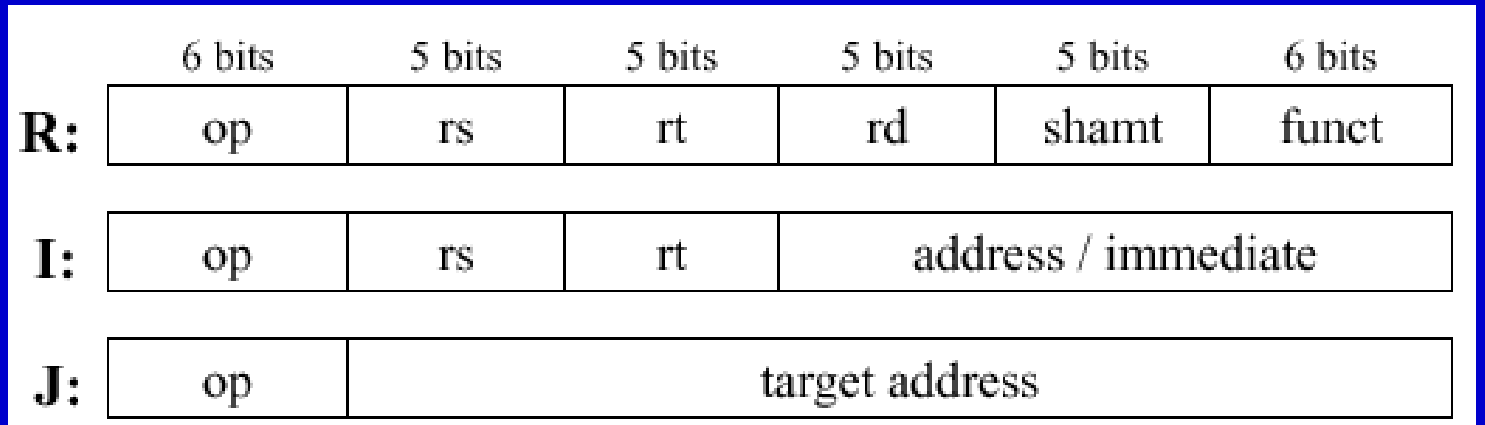
Η **MIPS** (*Microprocessor without Interlocked Pipeline Stages*) είναι μία αρχιτεκτονική συνόλου εντολών (ISA) γλώσσας μηχανής που αναπτύχθηκε από την εταιρεία MIPS Technologies (προηγουμένως MIPS Computer Systems Inc.) και ανήκει στην κατηγορία RISC (Reduced Instruction Set Computer). Οι πρώτες αρχιτεκτονικές MIPS ήταν των 32 bit, ενώ οι επόμενες ήταν των 64 bit.

Υπάρχουν πολλές εκδόσεις του συνόλου εντολών MIPS στις οποίες περιλαμβάνονται οι MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32 (32 bit), and MIPS64 (64 bit).

Subset of the MIPS instructions

Class	Instruction	Usage		Meaning	op	fn
Copy	Load upper immediate	`lui`	`rt,imm`	rt ← (imm, 0X0000)	15	
Arithmetic	Add	`add`	`rd,rs,rt`	rd ← (rs) + (rt)	0	32
	Subtract	`sub`	`rd,rs,rt`	rd ← (rs) − (rt)	0	34
	Set less than	`slt`	`rd,rs,rt`	rd ← if (rs) < (rd) then 1 else 0	0	42
	Add immediate	`addi`	`rt,rs,imm`	rt ← (rs) + imm	8	
	Set less than immediate	`slti`	`rd,rs,imm`	rd ← if (rs) < imm then 1 else 0	10	
Logic	AND	`and`	`rd,rs,rt`	rd ← (rs) ∧ (rt)	0	36
	OR	`or`	`rd,rs,rt`	rd ← (rs) ∨ (rt)	0	37
	XOR	`xor`	`rd,rs,rt`	rd ← (rs) ⊕ (rt)	0	38
	NOR	`nor`	`rd,rs,rt`	rd ← ((rs) ∨ (rt))′	0	39
	AND immediate	`andi`	`rt,rs,imm`	rt ← (rs) ∧ imm	12	
	OR immediate	`ori`	`rt,rs,imm`	rt ← (rs) ∨ imm	13	
	XOR immediate	`xori`	`rt,rs,imm`	rt ← (rs) ⊕ imm	14	
Memory access	Load word	`lw`	`rt,imm(rs)`	rt ← mem[(rs) + imm]	35	
	Store word	`sw`	`rt,imm(rs)`	mem[(rs) + imm] ← (rt)	43	
Control transfer	Jump	`j`	`L`	goto L	2	
	Jump register	`jr`	`rs`	goto (rs)	0	8
	Branch less than 0	`bltz`	`rs,L`	if (rs) < 0 then goto L	1	
	Branch equal	`beq`	`rs,rt,L`	if (rs) = (rt) then goto L	4	
	Branch not equal	`bne`	`rs,rt,L`	if (rs) ≠ (rt) then goto L	5	

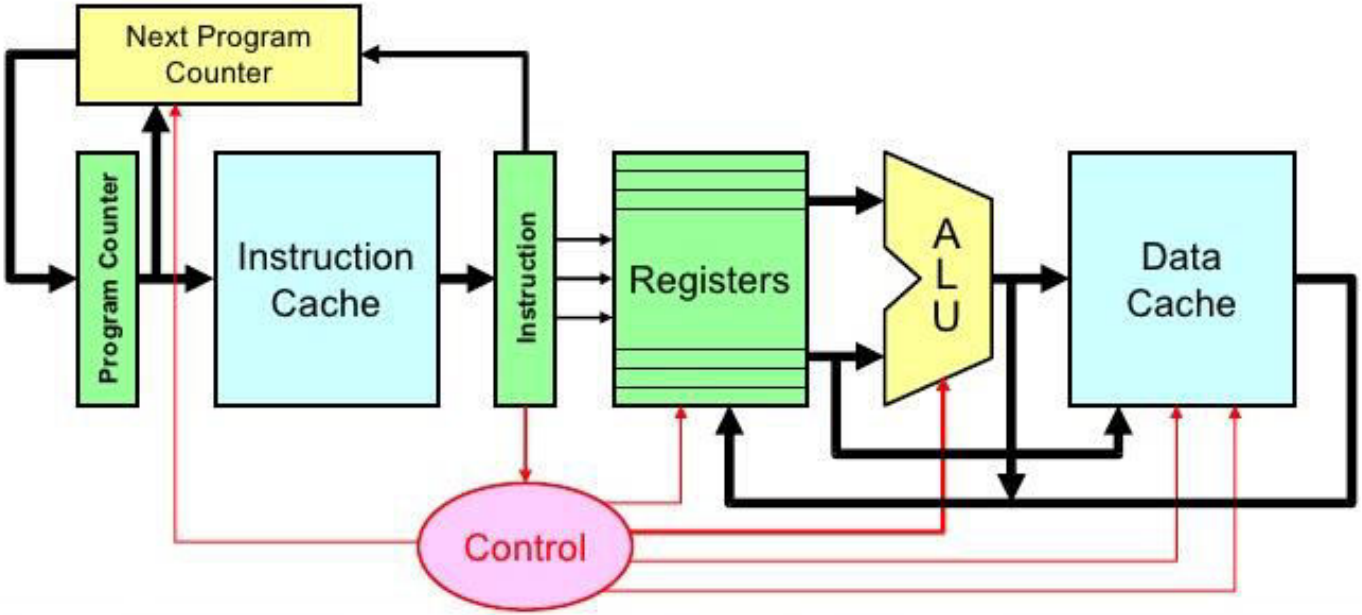
MIPS instructions formats



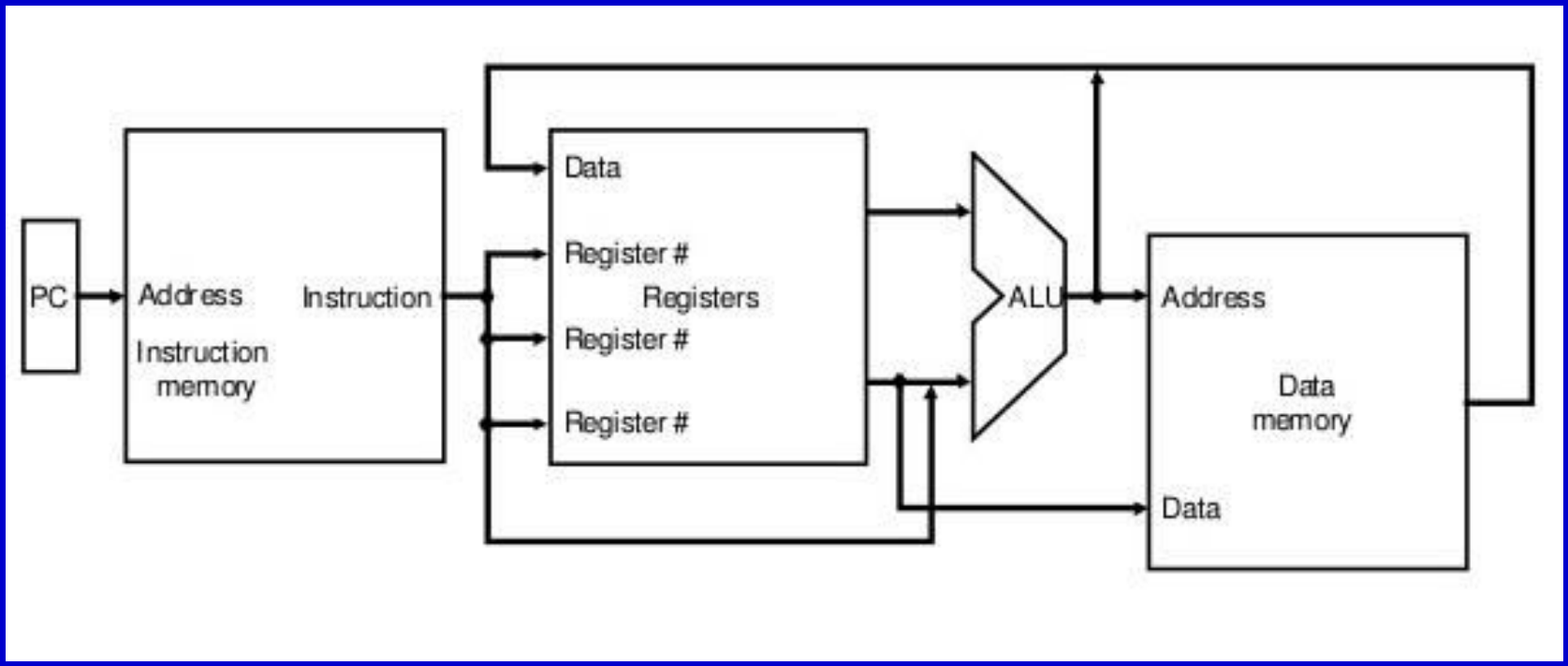
op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions (+/-2^{15})
immediate: constants for immediate instructions

Αρχιτεκτονική του MIPS

- Datapath: part of a processor that executes instructions
- Control: generates control signals for each instruction

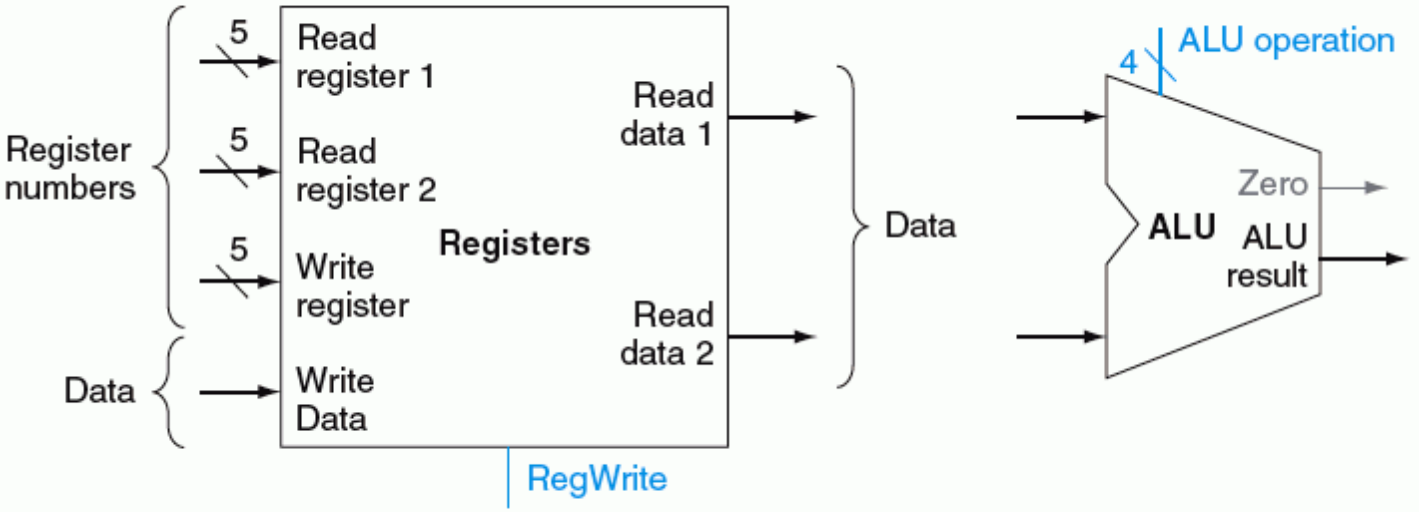


Διασύνδεση βασικών μονάδων στον MIPS



PC: Program Counter **ALU**: Arithmetic Logic Unit

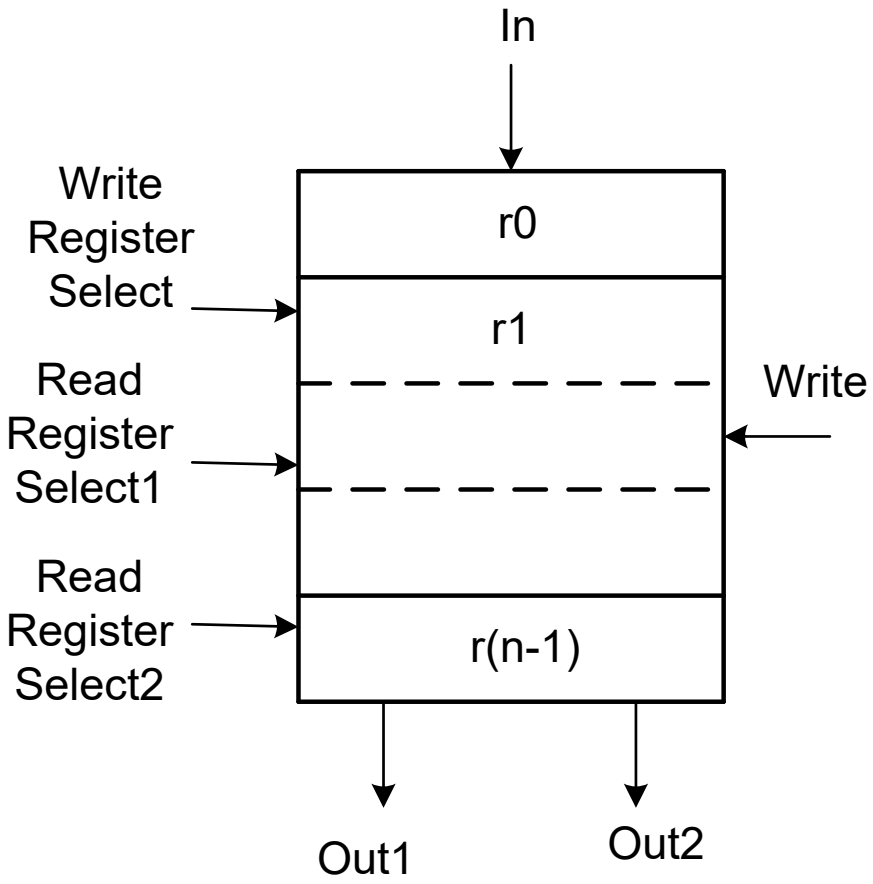
Register File και ALU



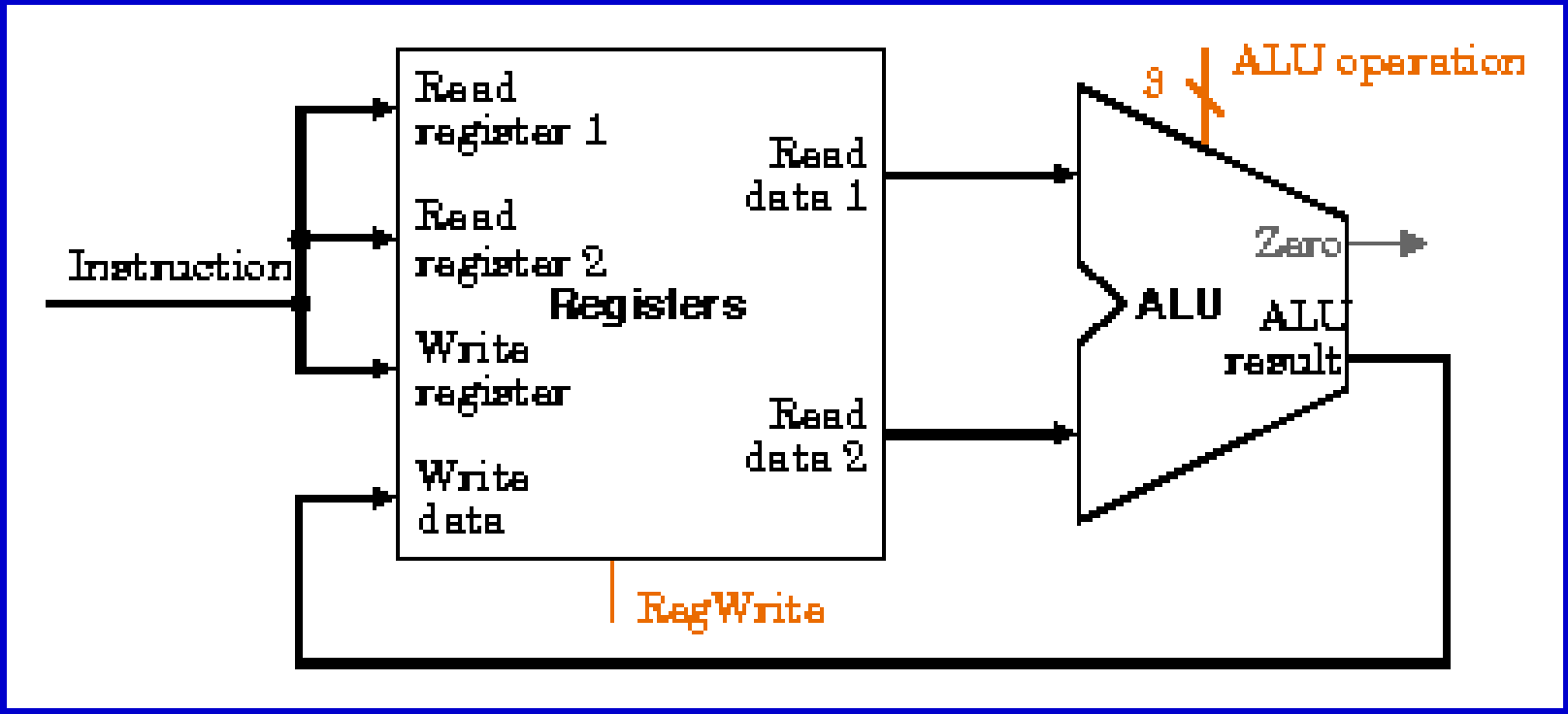
Register file

Το *register file* είναι διατάξεις από καταχωρητές που χρησιμοποιούνται εσωτερικά στις μονάδες επεξεργασίας δεδομένων των επεξεργαστών. Στους σύγχρονους επεξεργαστές υπάρχουν register file με πολλές πόρτες ανάγνωσης δεδομένων. Στην συνέχεια δίδεται το λογικό σύμβολο ενός register file με *n* καταχωρητές r0, r1,…, r(n-1) και δύο πόρτες ανάγνωσης δεδομένων.

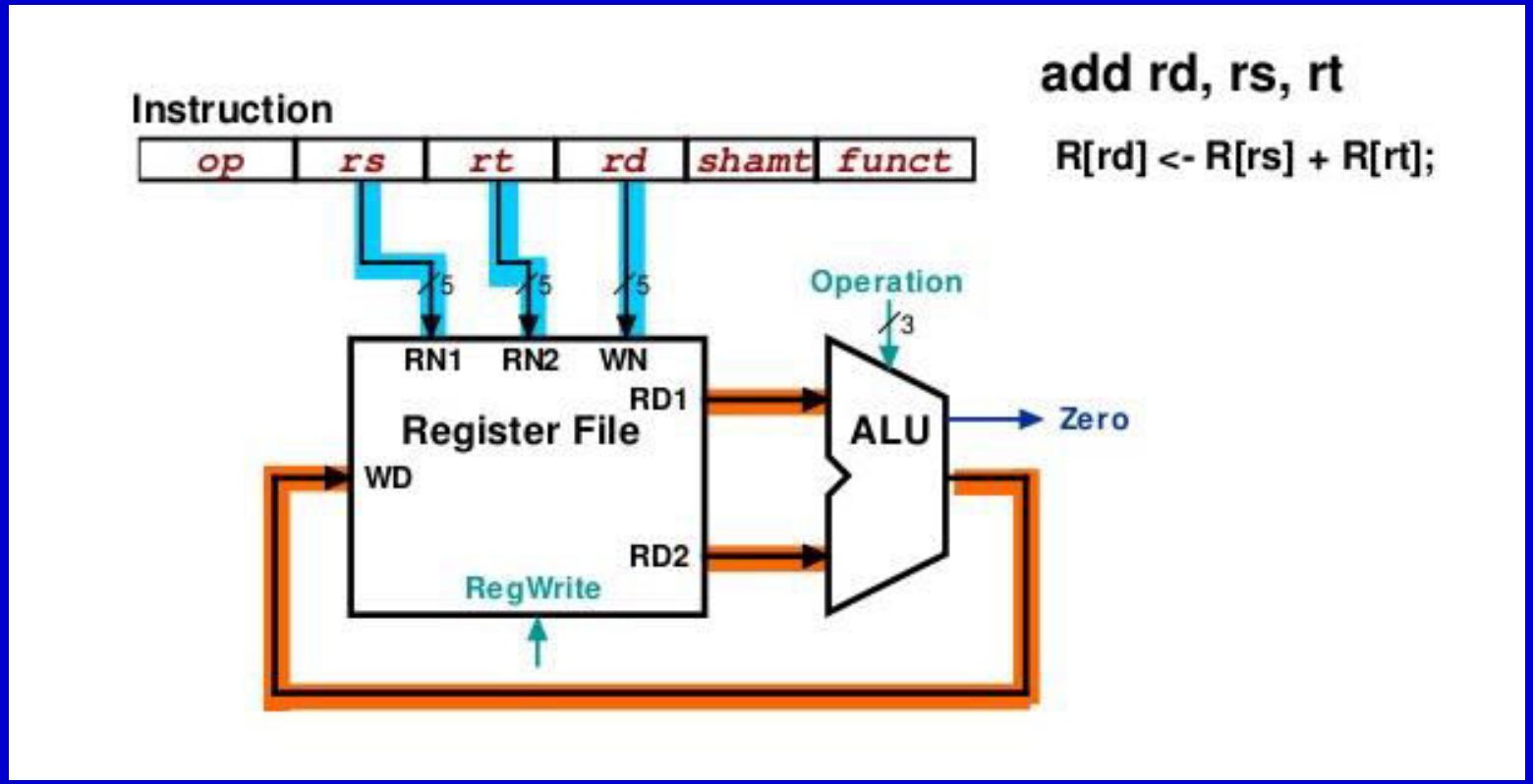
Λογικό σύμβολο ενός register file με *n* καταχωρητές r0, r1,…, r(n-1) και δύο πόρτες ανάγνωσης δεδομένων.



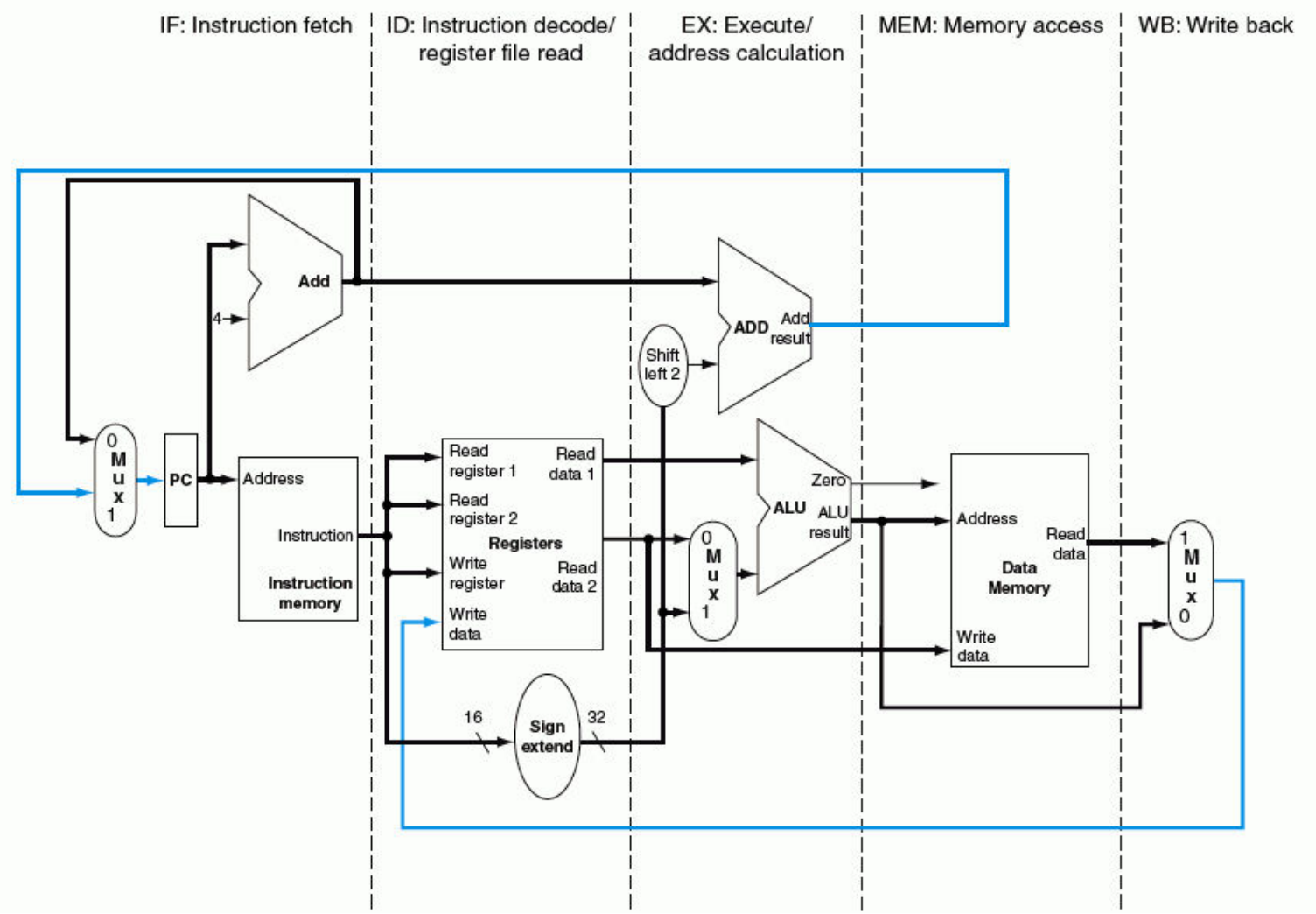
Διασύνδεση Register File και ALU στον MIPS



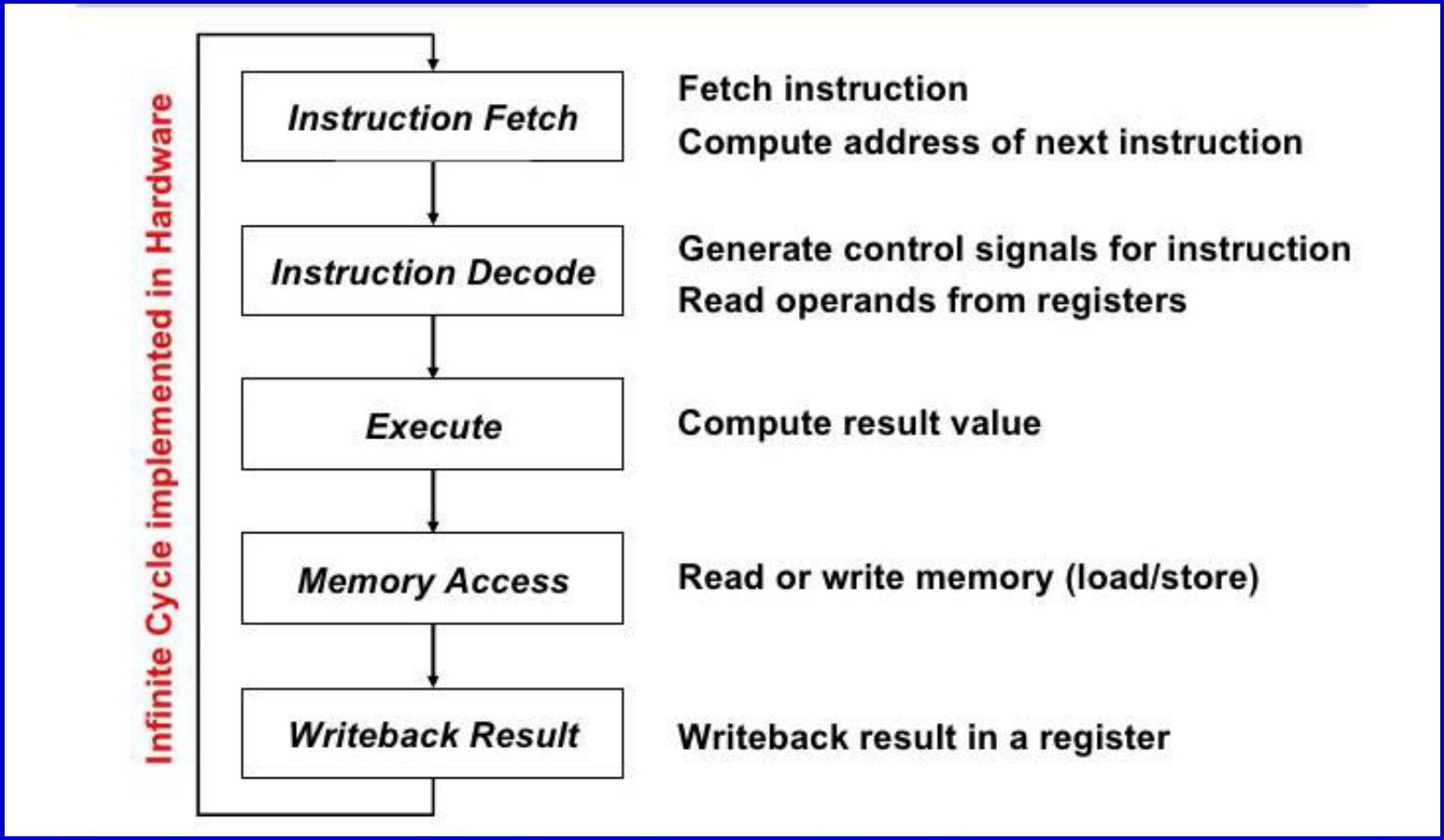
Εκτέλεση της εντολής add rd, rs, rt στον MIPS



Basic MIPS architecture



Instruction cycle του MIPS



Εκτέλεση διαδοχικών εντολών στον MIPS χωρίς pipeline

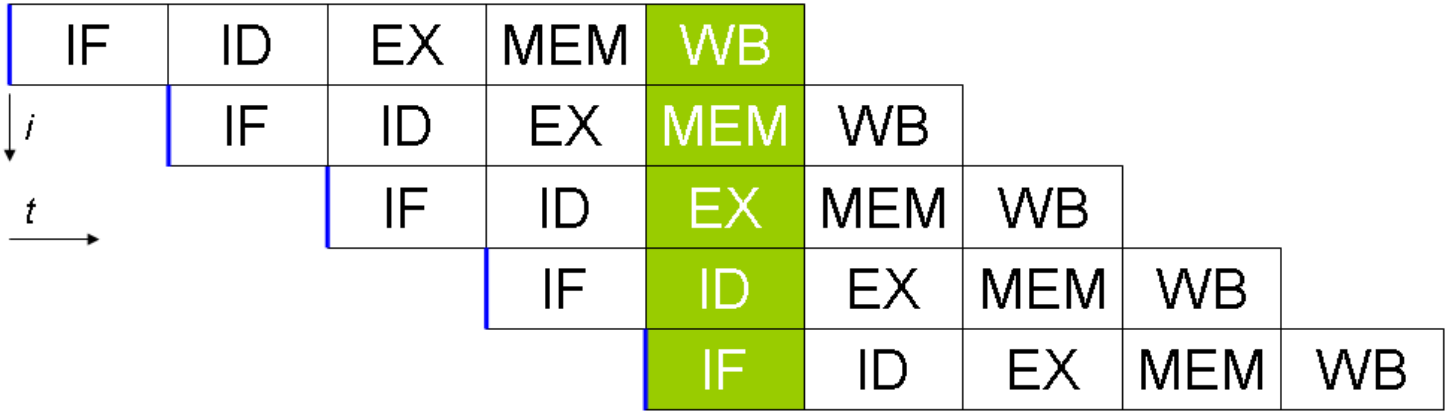
IF: Instruction Fetch, **ID**: Instruction Decode, **EX**: Execute,
MEM: Memory access, **WB**: Write Back

CPU pipelining

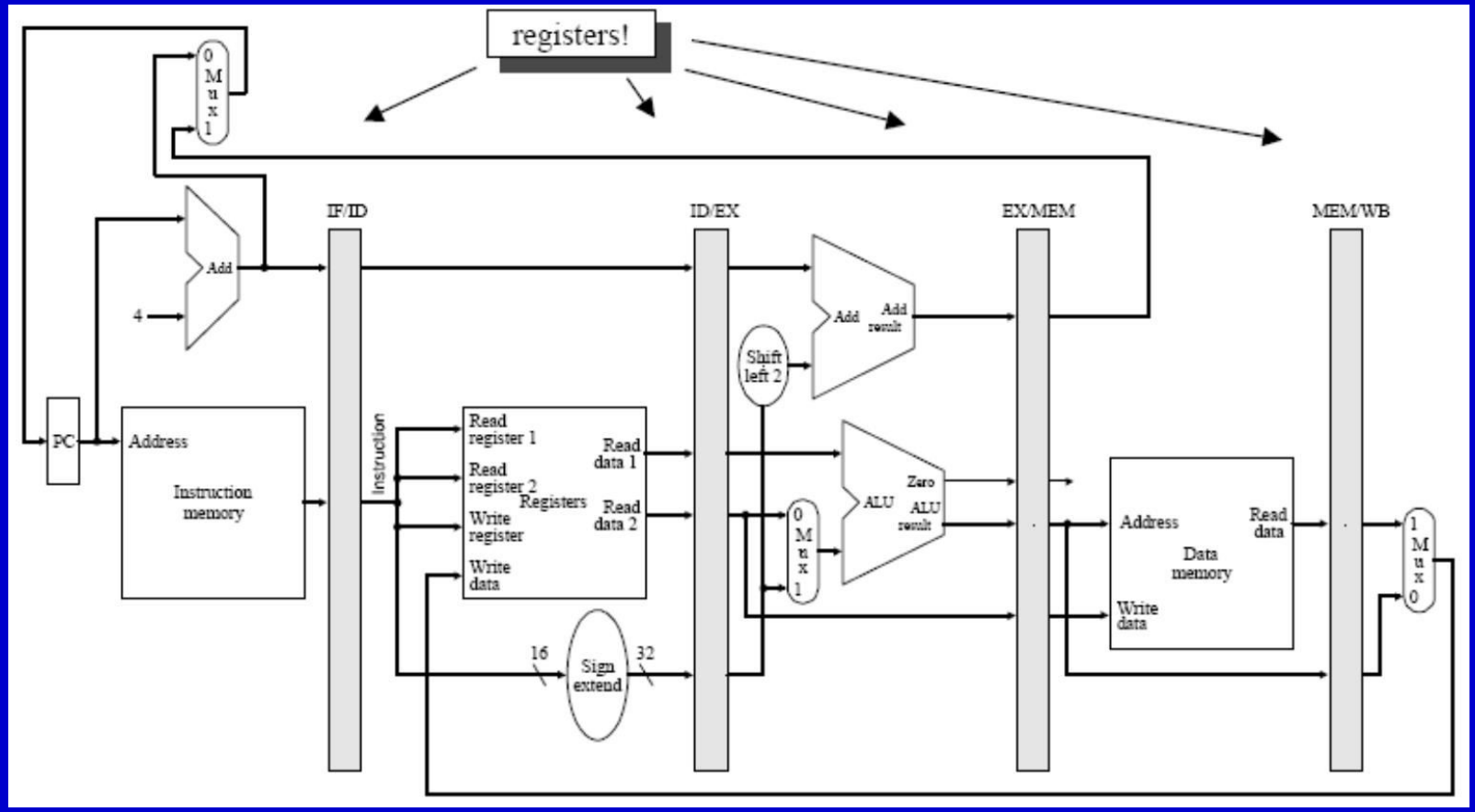
Το *pipelining* είναι τεχνική αλληλοεπικάλυψης της εκτέλεσης των εντολών γλώσσας μηχανής σε έναν επεξεργαστή, ώστε να περιορισθεί ο χρόνος εκτέλεσης ενός συνόλου εντολών. Για την *υλοποίηση του pipelining* η μονάδα επεξεργασίας δεδομένων διαιρείται σε βαθμίδες και τοποθετούνται pipeline latches (flip-flops) μεταξύ των βαθμίδων.

Στην αρχή κάθε κύκλου ωρολογίου γίνεται εγγραφή των εξόδων των βαθμίδων στα pipeline latches των οποίων οι έξοδοί τους παραμένουν σταθερές κατά το υπόλοιπο του κύκλου για να χρησιμοποιηθούν σαν είσοδοι από την επόμενη βαθμίδα.

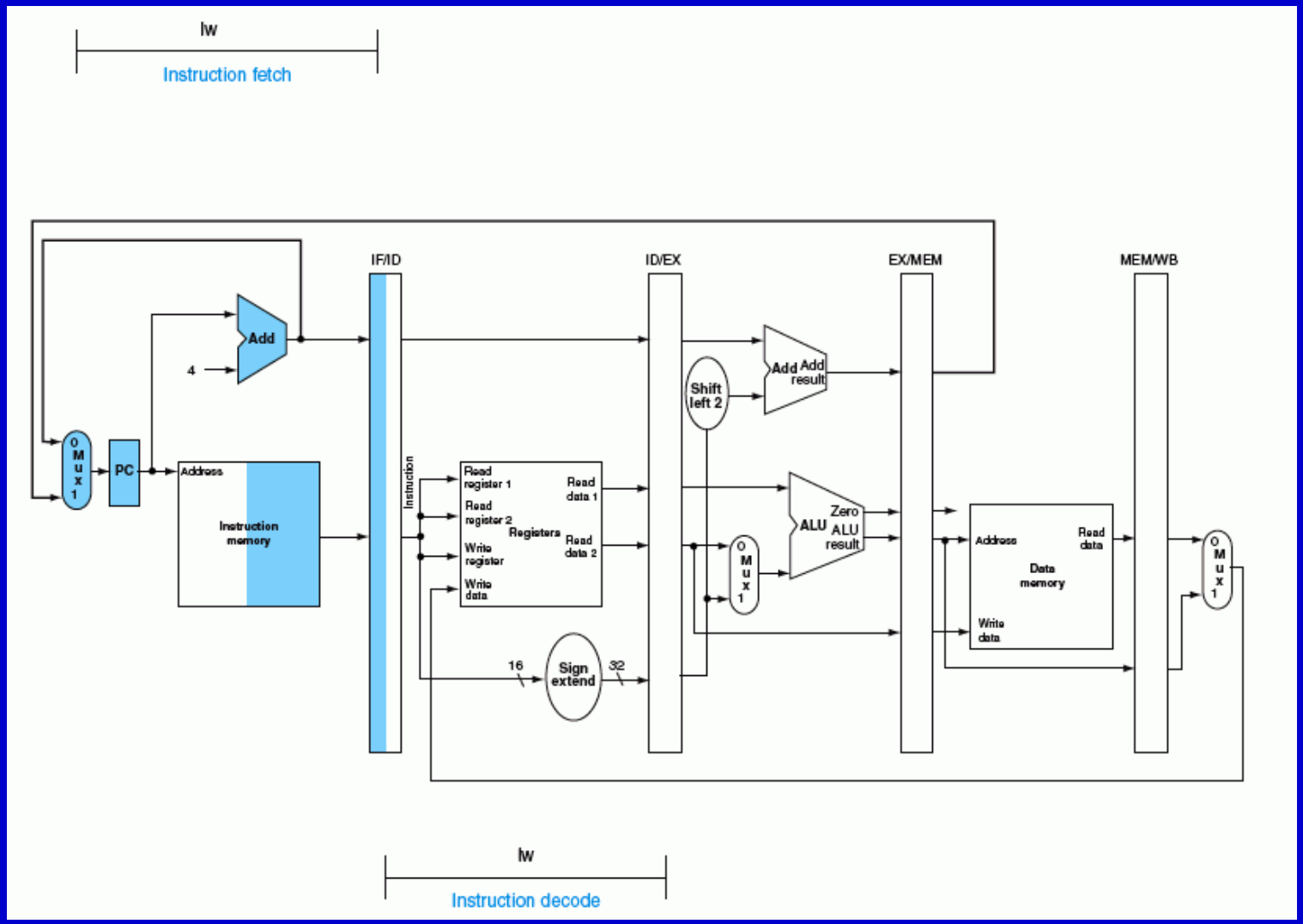
Εκτέλεση εντολών στον MIPS με Pipeline



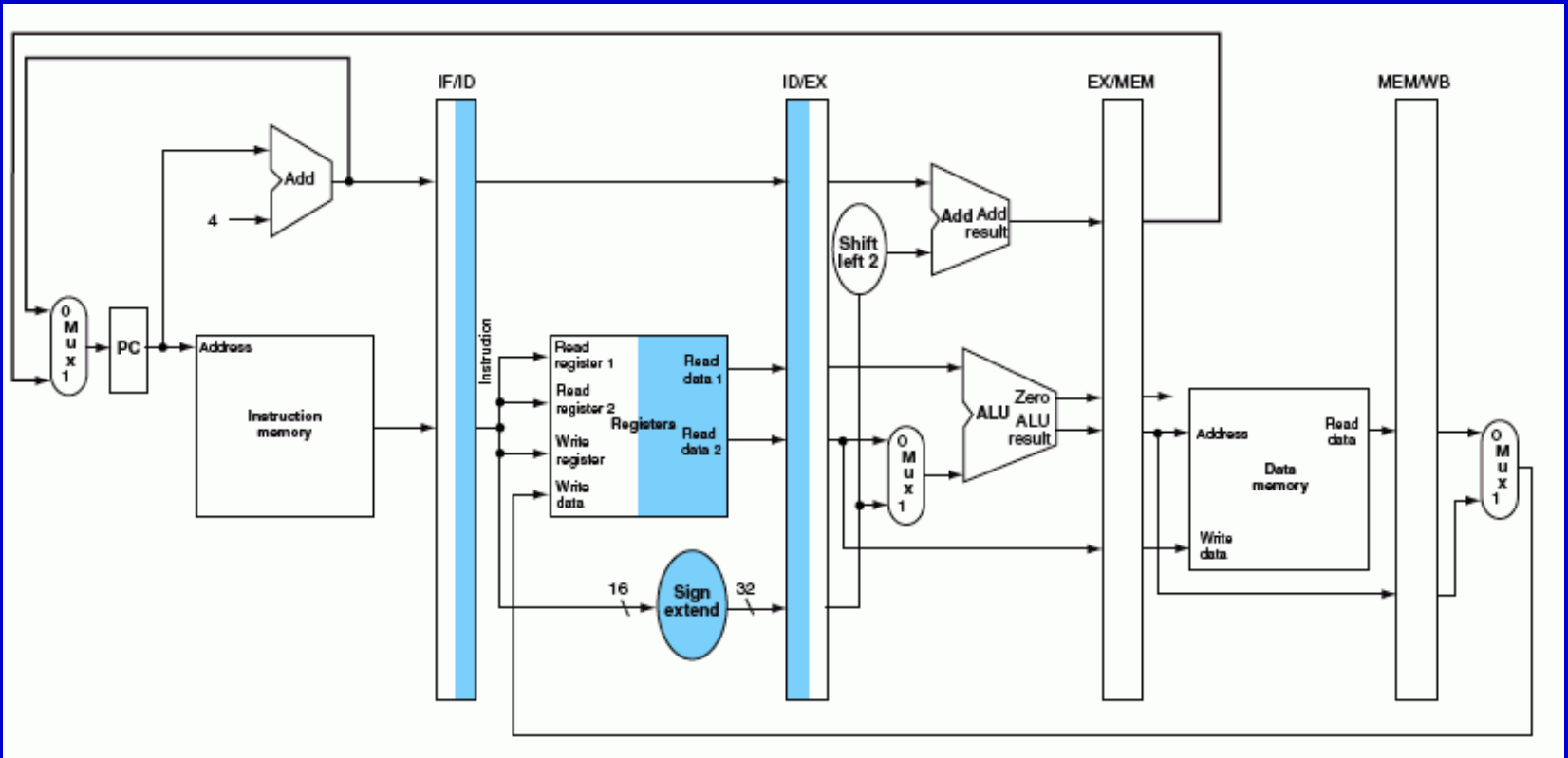
Pipelined MIPS architecture



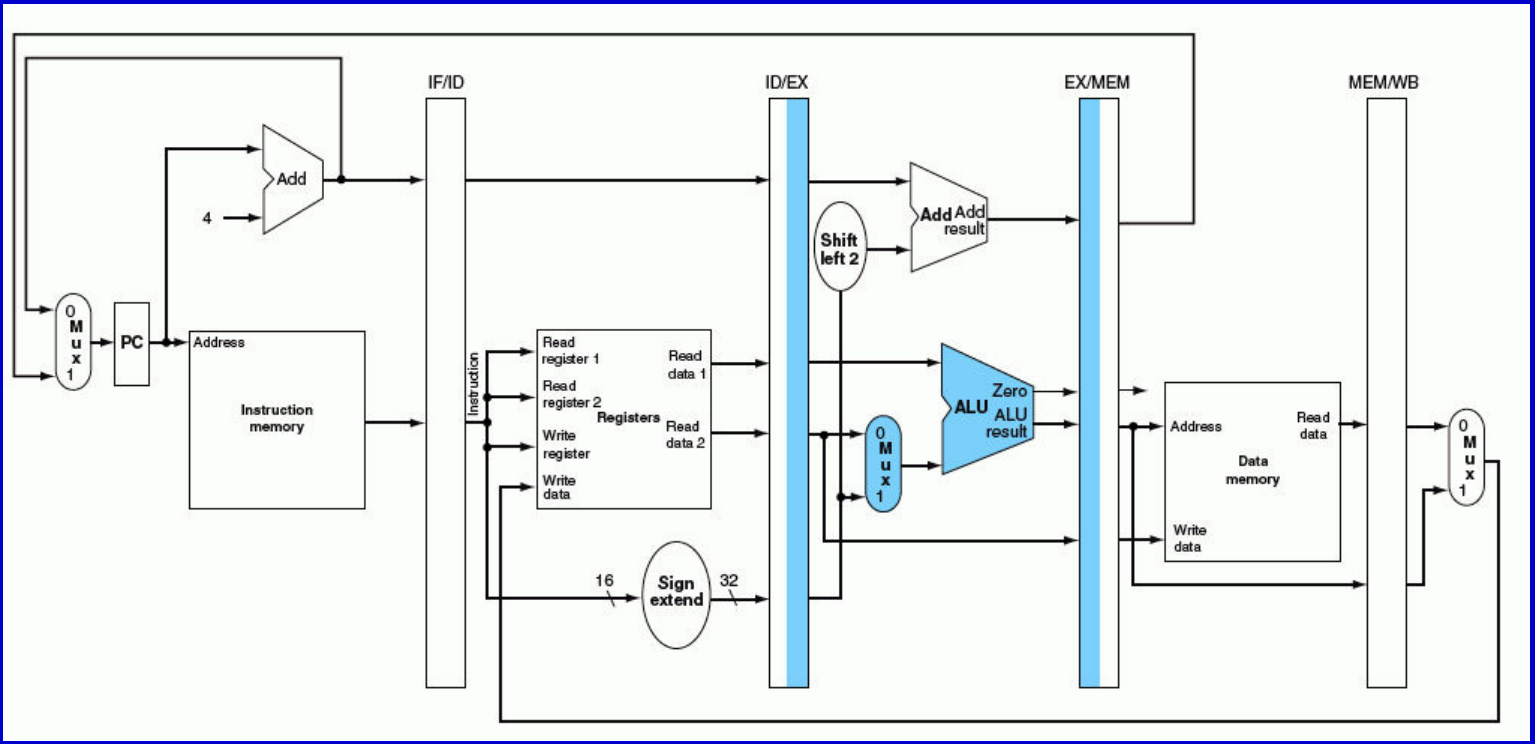
IF


lw
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
lw
Instruction decode

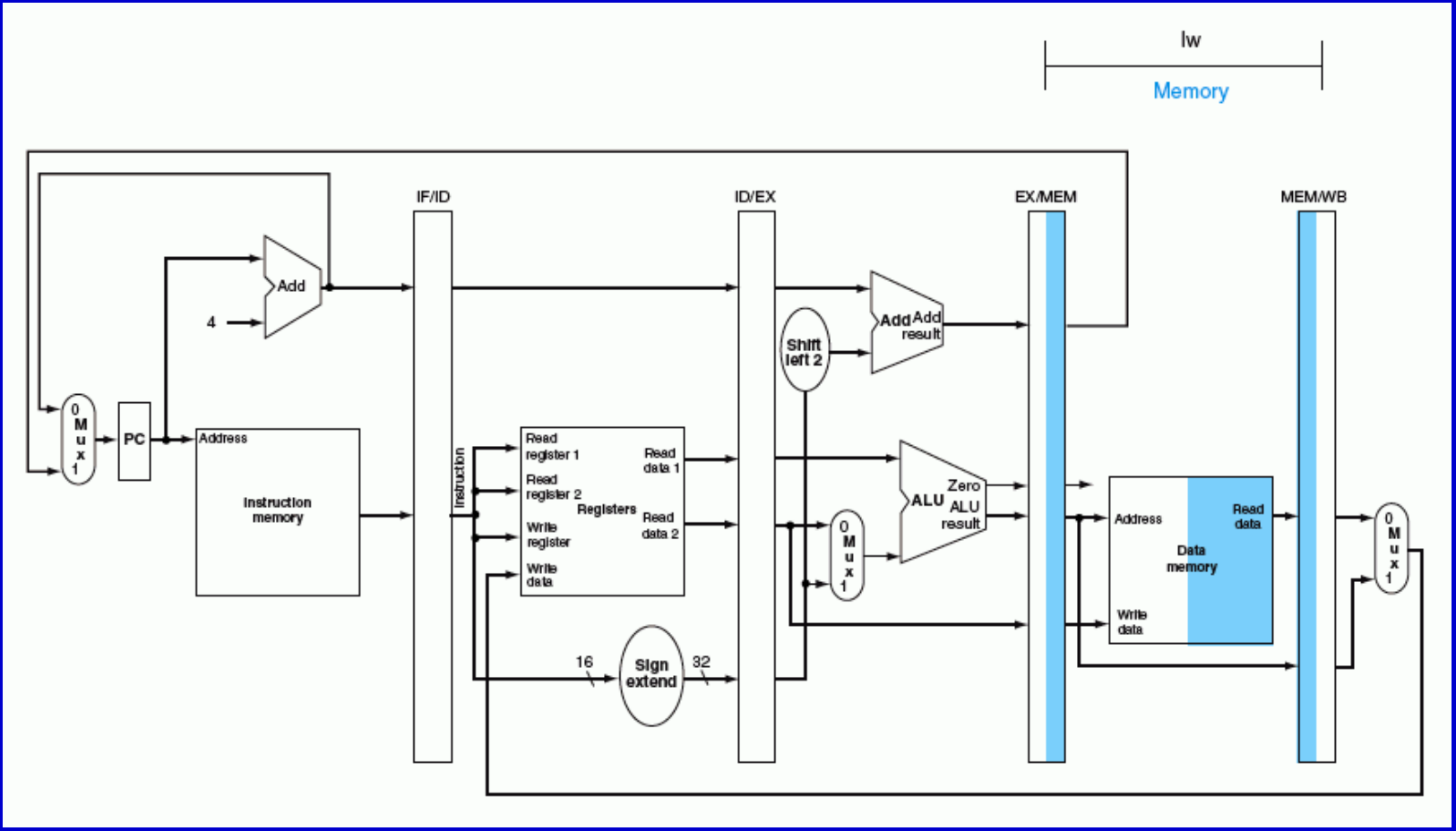
ID (RR)


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign extend
32
Address
Data memory
Read data
Write data

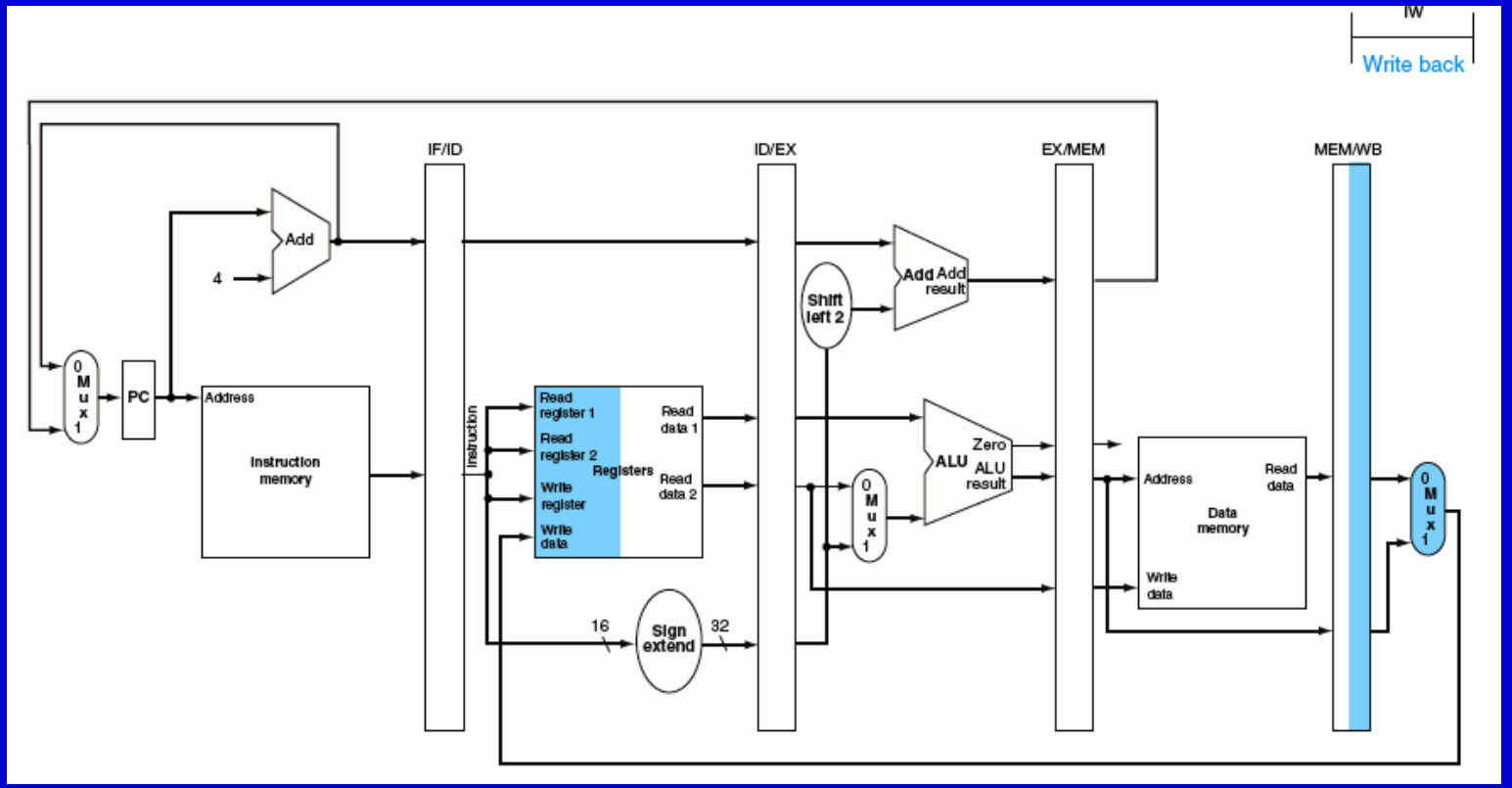
EX


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign extend
32

MEM


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

WB

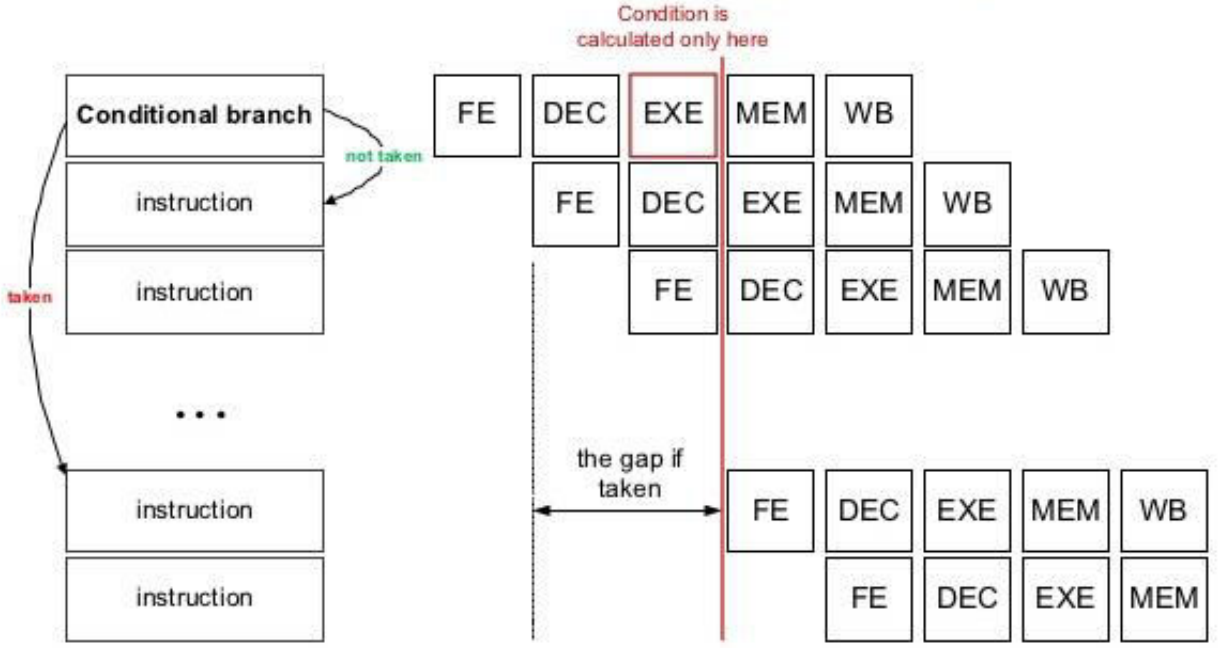

IW
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

Branch Prediction

Με τον όρο *branch prediction* (*πρόβλεψη διακλάδωσης*) εννοούμε ένα σύνολο από τεχνικές που υλοποιούνται στους σύγχρονους επεξεργαστές, με τις οποίες επιχειρείται να προβλεφθεί εάν θα εκτελεστεί ένα conditional branch και ανάλογα να γίνει fetch εντολών σε γλώσσα μηχανής από την κατάλληλη θέση μνήμης, ώστε να μην διακοπεί το pipelining των εντολών.

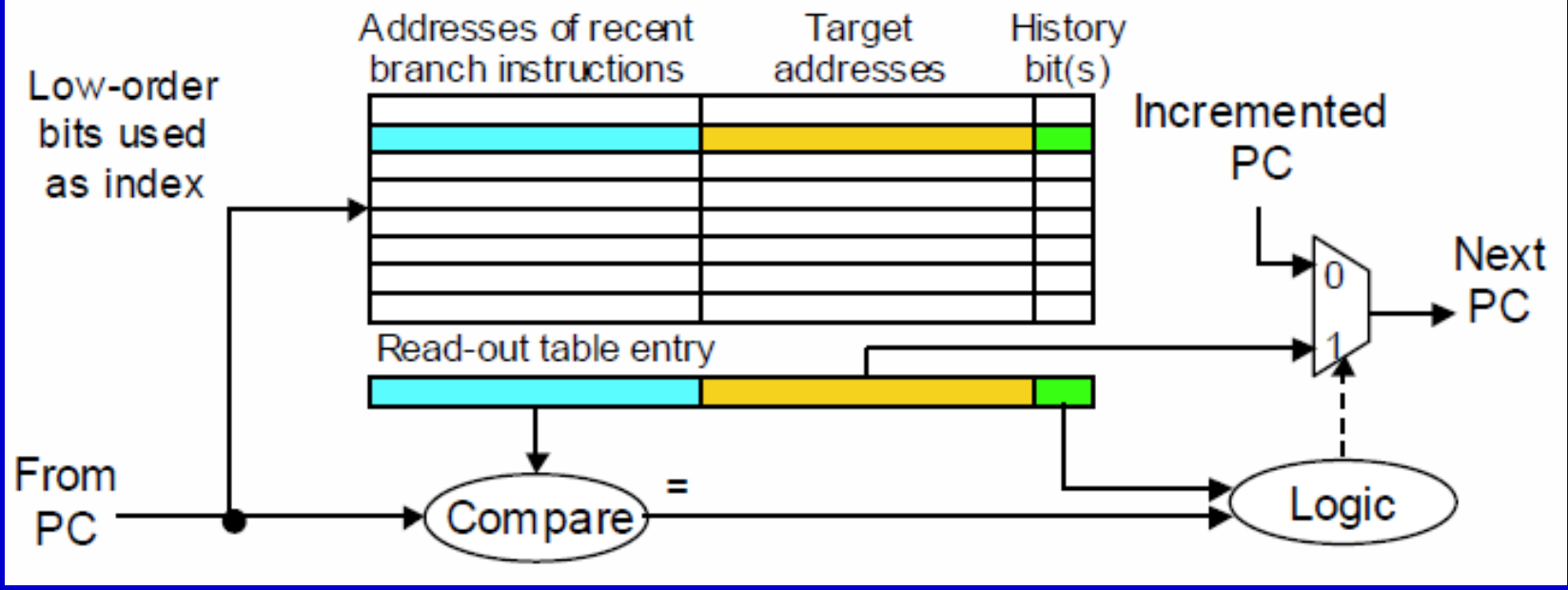
Υπάρχουν ***στατικές*** και ***δυναμικές*** τεχνικές πρόβλεψης διακλάδωσης.

Εκτέλεση εντολής διακλάδωσης σε pipeline αρχιτεκτονική χωρίς branch prediction

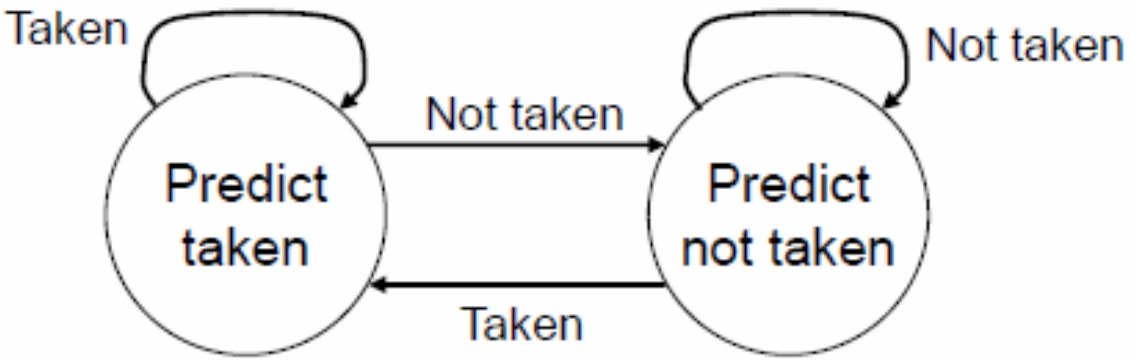


- The time that is wasted in case of a branch misprediction is equal to the number of stages in the pipeline from the fetch stage to the execute stage. Modern microprocessors tend to have quite long pipelines so that the misprediction delay is between 10 and 20 cycles (if not consider misses in caches).

Hardware Implementation of Branch Prediction

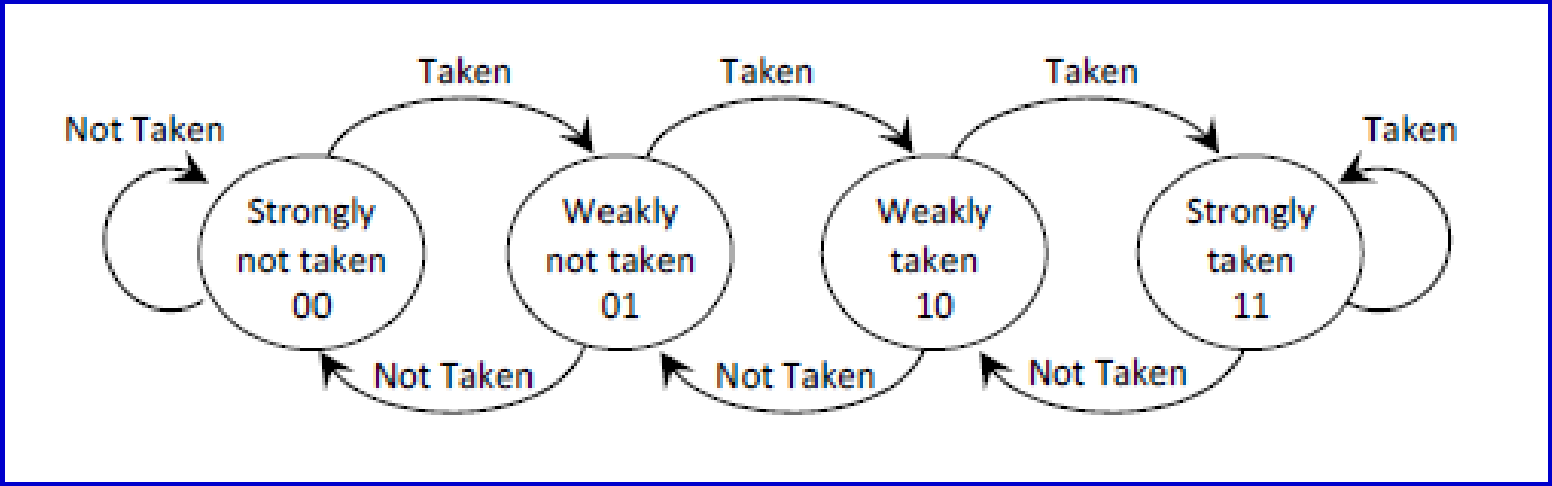


Simple Branch Prediction: 1-Bit History



Two-state branch prediction scheme.

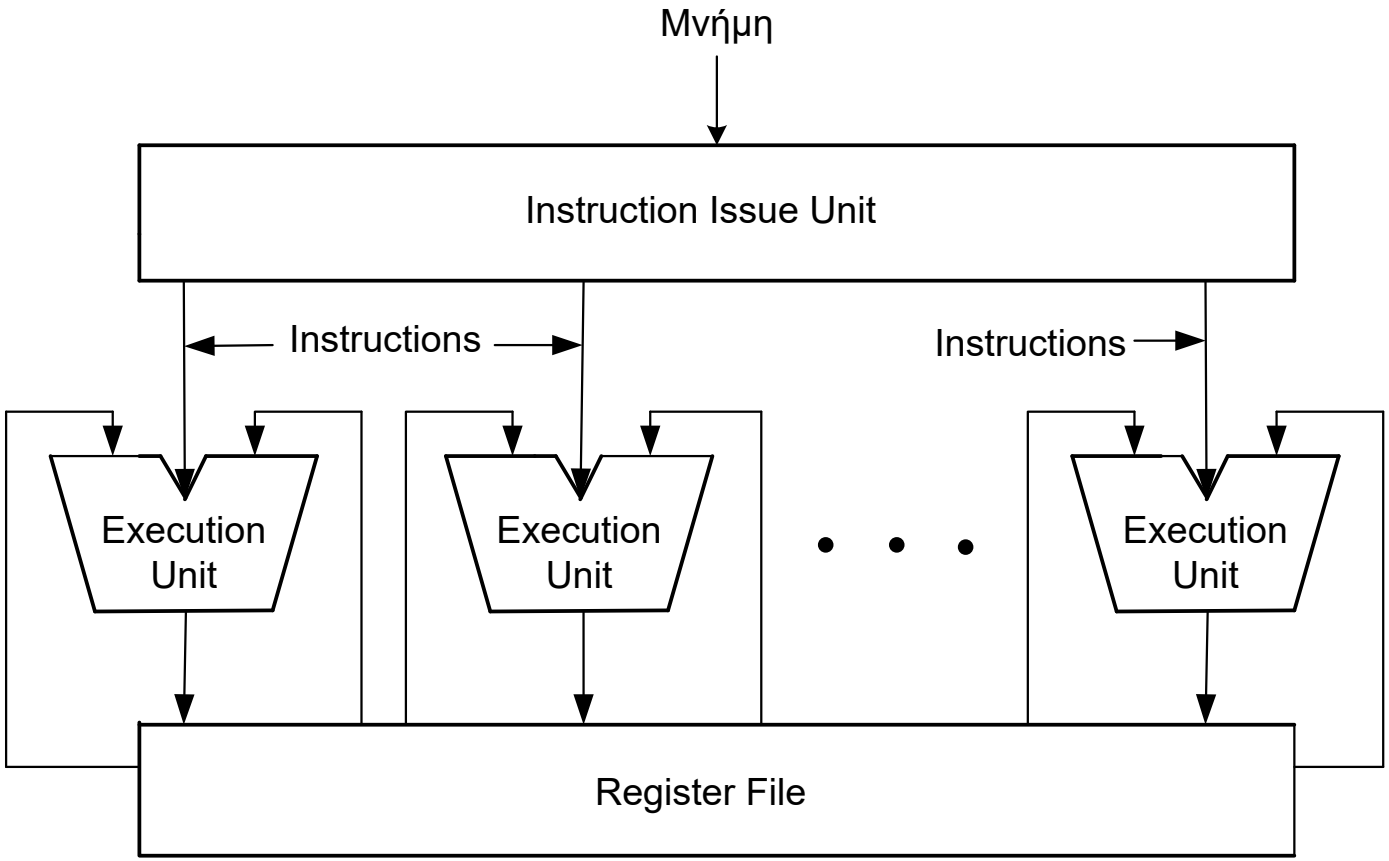
Simple Branch Prediction: 2-Bit History



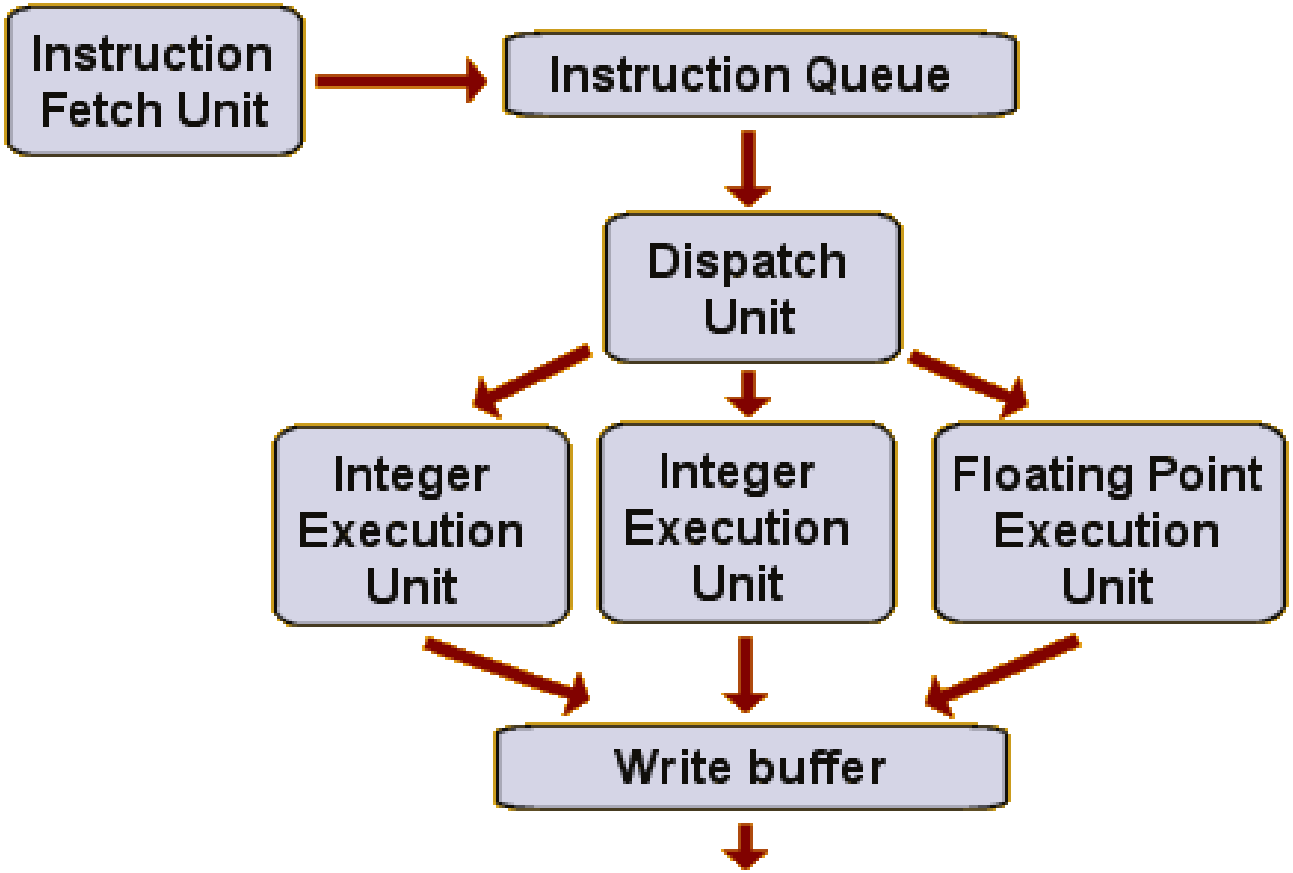
Superscalar architectures

Στους επεξεργαστές με *υπερβαθμωτή* (*superscalar*) αρχιτεκτονική υπάρχουν δύο ή περισσότερες λειτουργικές μονάδες όπως Integer Execution Units (IEU), Floating Point Units (FPU), Branch Units.
Μία ειδική μονάδα (Instruction Issue Unit) ανακαλεί εντολές από την μνήμη cache, ανιχνεύει εάν υπάρχει δυνατότητα παραλληλισμού μεταξύ των εντολών (ύπαρξη ανεξάρτητων εντολών) και αναθέτει κατάλληλα την εκτέλεση των εντολών στις διάφορες λειτουργικές μονάδες, ώστε αυτές να εκτελούνται ταυτόχρονα.

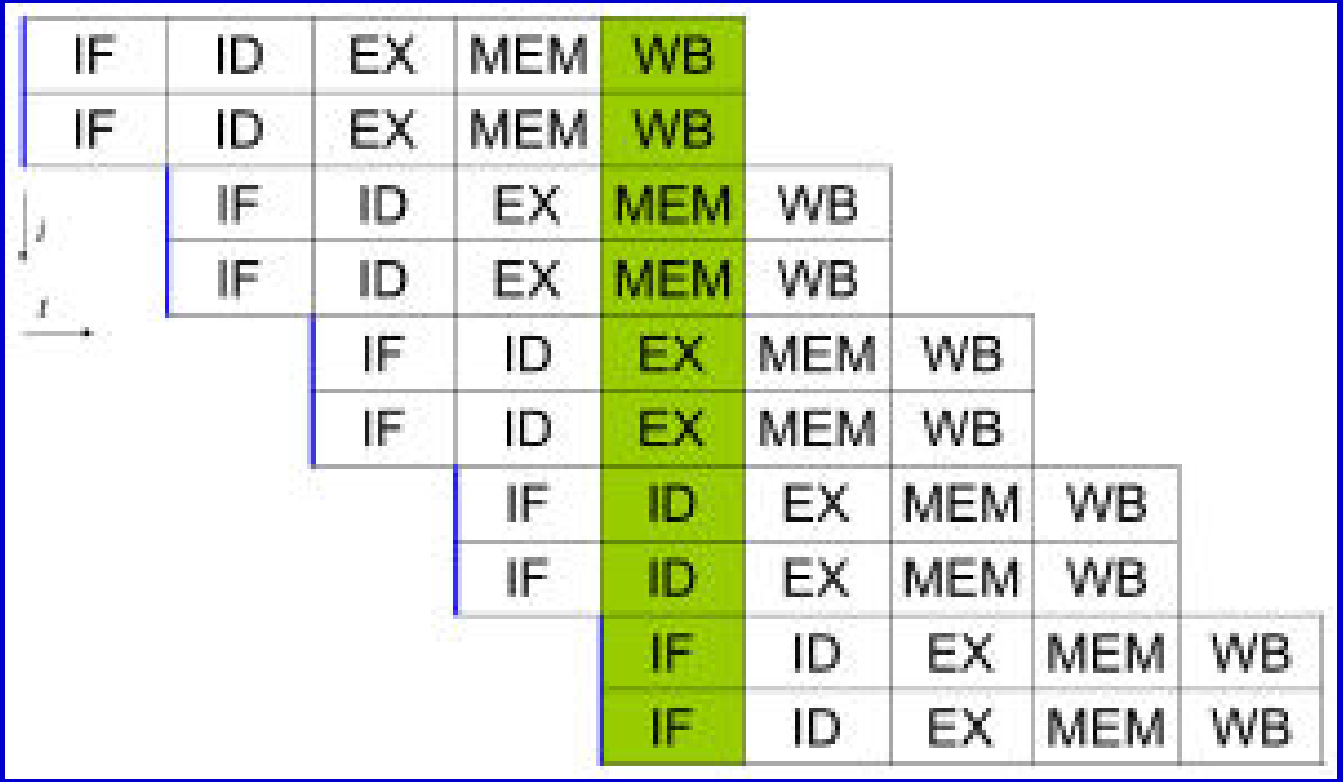
Superscalar architecture



Superscalar architecture



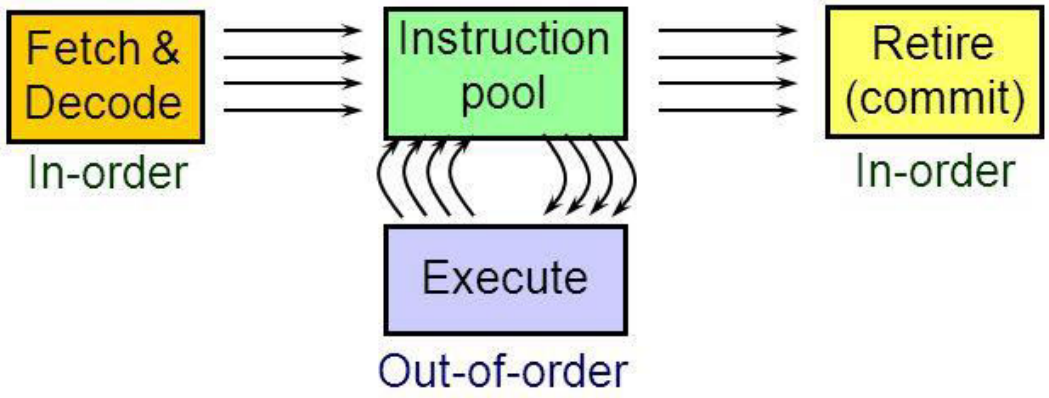
Εκτέλεση εντολών από επεξεργαστή με superscalar αρχιτεκτονική και δύο μονάδες εκτέλεσης εντολών



Out-of-Order Eexecution (OoOE)

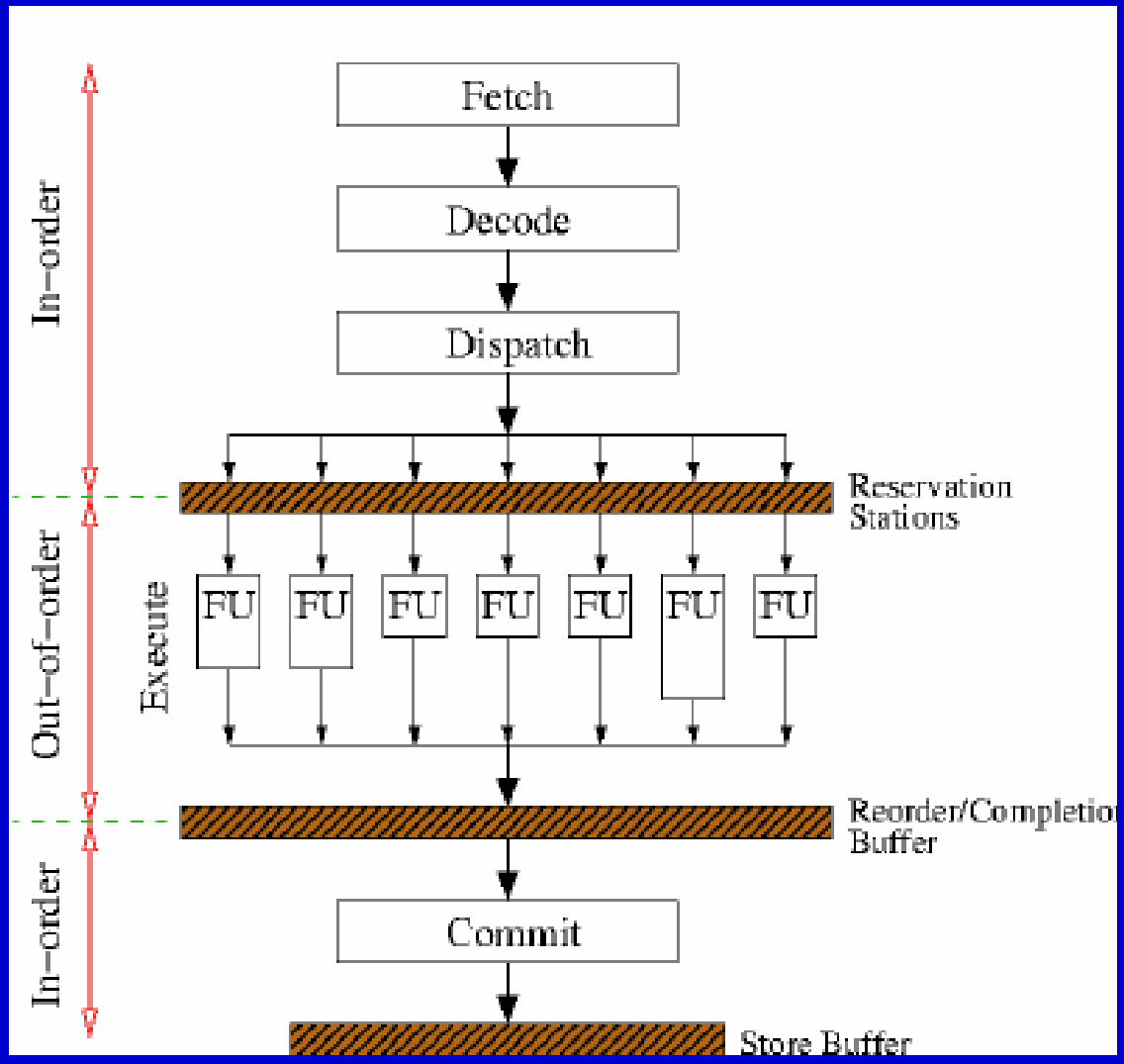
Η τεχνική *Out-of-Order Eexecution* (OoOE) ενσωματώνεται στους περισσότερους σύγχρονους επεξεργαστές για να εκτελεστούν εντολές σε κύκλοι ωρολογίου των οποίων η μη χρήση θα είχε σαν αποτέλεσμα την καθυστέρηση στην εκτέλεση του προγράμματος.
Με την τεχνική OoOE ο επεξεργαστής ανακαλεί ομάδες και τις εκτελεί με σειρά η οποία προσδιορίζεται από την διαθεσιμότητα δεδομένων εισόδου σε αυτές, παρά με την σειρά τους στο πρόγραμμα.
Με αυτό τον τρόπο αποφεύγεται ο επεξεργαστής να παραμένει ανενεργός, ενώ ανακτώνται τα δεδομένα για την επόμενη εντολή στο πρόγραμμα, και εκτελεί αντί της αμέσως επόμενης εντολής κάποια άλλη που μπορεί να εκτελεστεί άμεσα.

OOOE – general scheme



- **Fetch & decode instructions in parallel but in order, to fill inst. pool**
- **Execute ready instructions from the instructions pool**
  - All the data required for the instruction is ready
  - Execution resources are available
- **Once an instruction is executed**
  - Signal all dependent instructions that data is ready
- **Commit instructions in parallel but in-order**
  - Can commit an instruction only after all preceding instructions (in program order) have committed

Αρχιτεκτονική OoOE



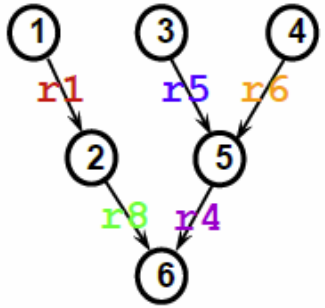
FU: Functional Unit

OoOE

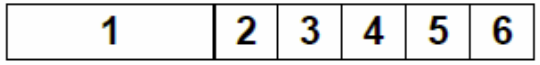
- **Example:**

```
(1) r1 ← r4 / r7      /* assume division takes 20 cycles */
(2) r8 ← r1 + r2
(3) r5 ← r5 + 1
(4) r6 ← r6 - r3
(5) r4 ← r5 + r6
(6) r7 ← r8 * r4
```

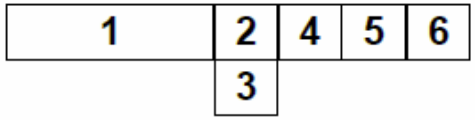
Data Flow Graph



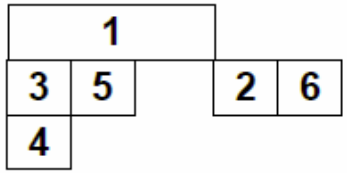
In-order execution



In-order (2-way superscalar)



Out-of-order execution



Register renaming

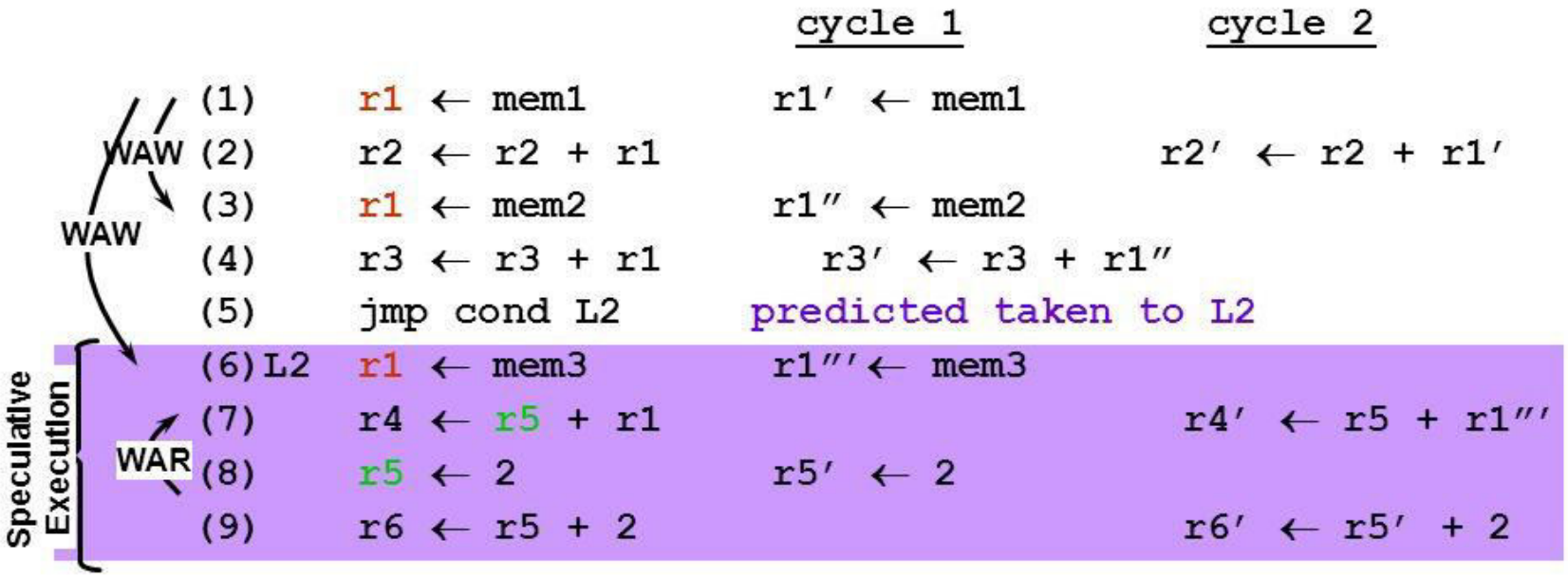
In computer architecture, **register renaming** is a technique that eliminates the false data dependencies arising from the reuse of architectural registers by successive instructions that do not have any real data dependencies between them. The elimination of these false data dependencies reveals more instruction-level parallelism in an instruction stream, which can be exploited by various and complementary techniques such as superscalar and out-of-order execution for better performance.

Τεχνική register renaming

#	Instruction
1	R1 = M[1024]
2	R1 = R1 + 2
3	M[1032] = R1
4	R1 = M[2048]
5	R1 = R1 + 4
6	M[2056] = R1

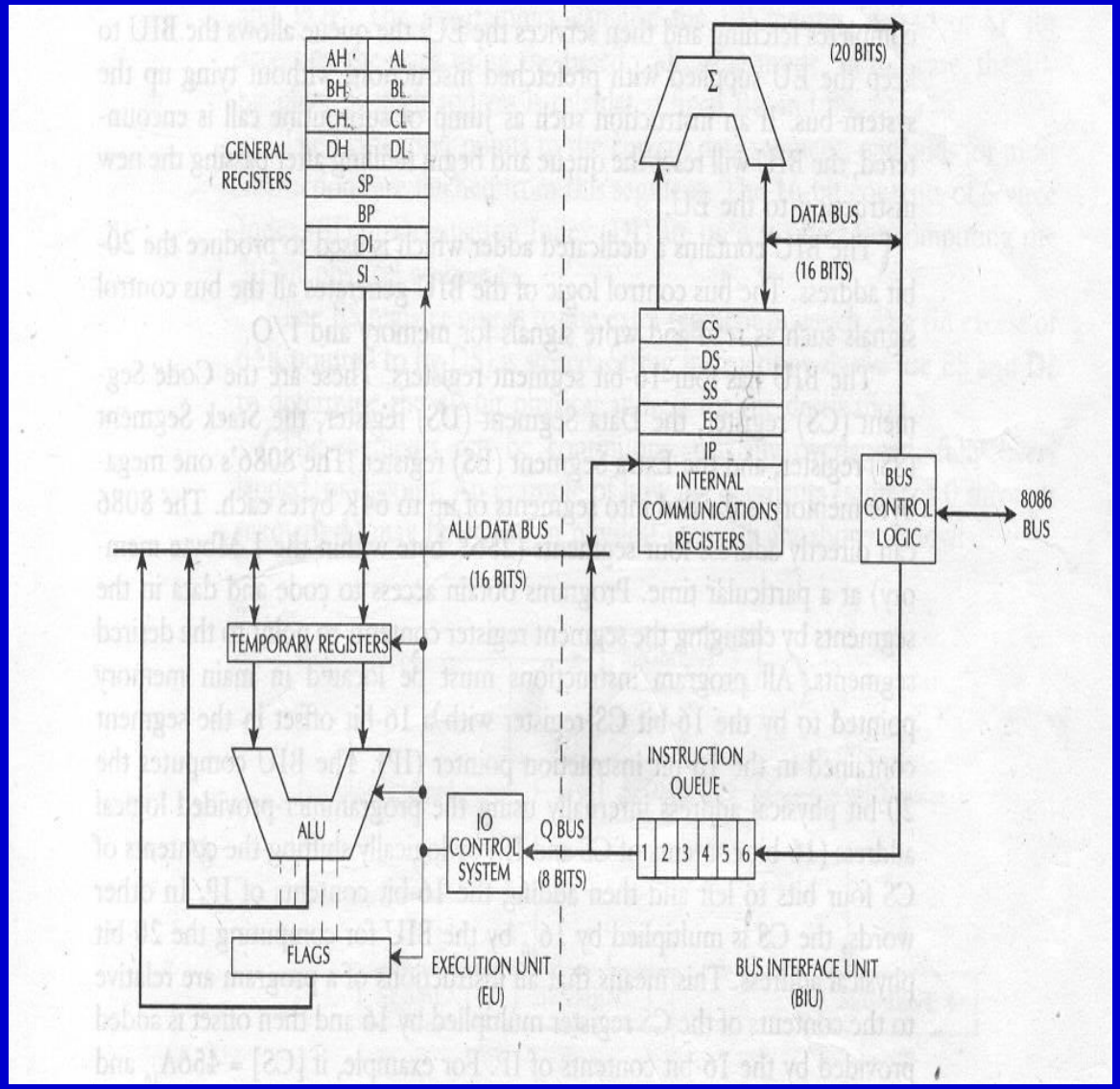
#	Instruction		#	Instruction
1	R1 = M[1024]		4	R2 = M[2048]
2	R1 = R1 + 2		5	R2 = R2 + 4
3	M[1032] = R1		6	M[2056] = R2

Speculative execution – example

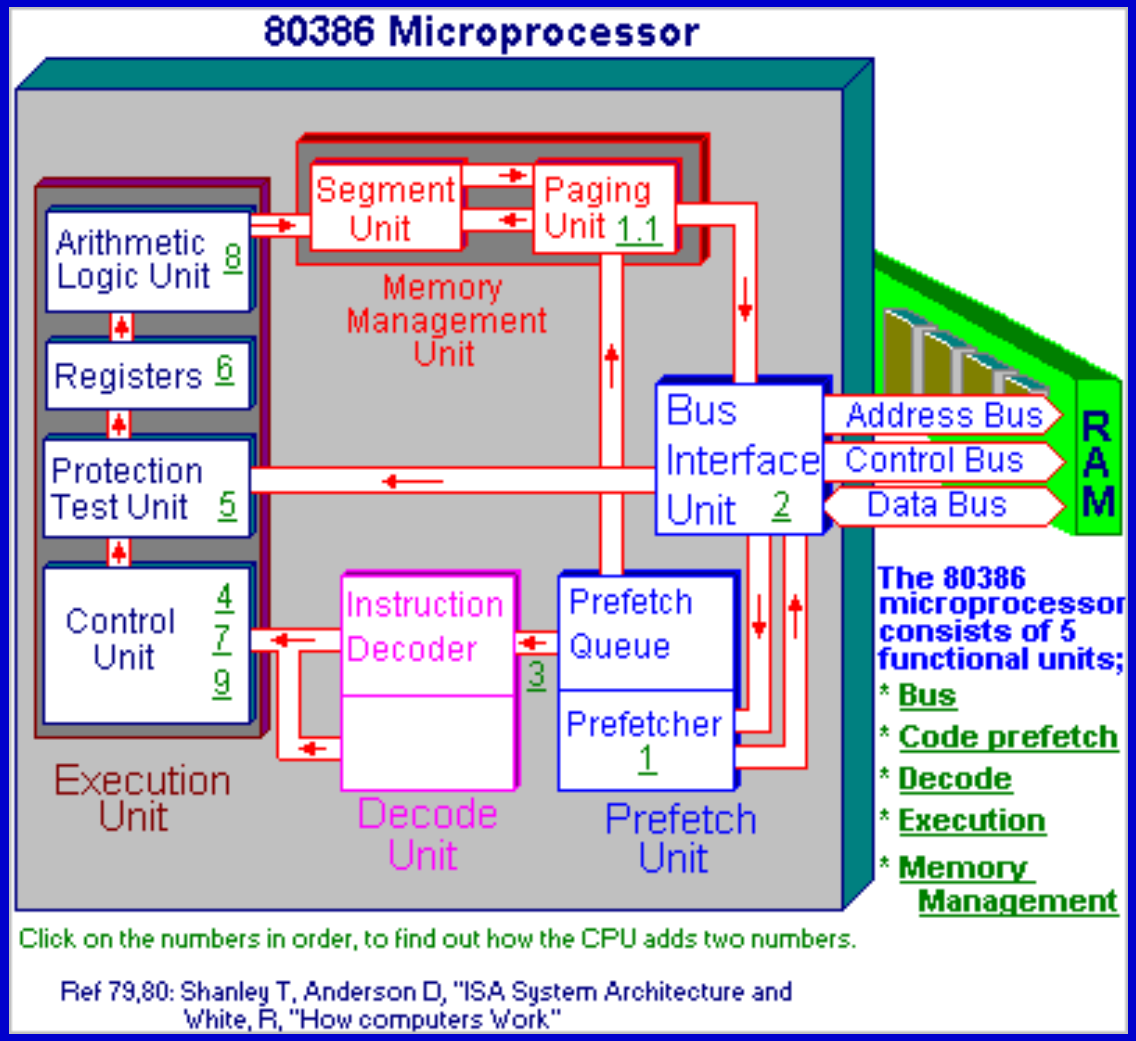


- **Instructions 6-9 are speculatively executed**
  - If the prediction turns wrong, they will be flushed
- **If the branch was predicted not-taken**
  - The instructions from the other path would have been speculatively executed

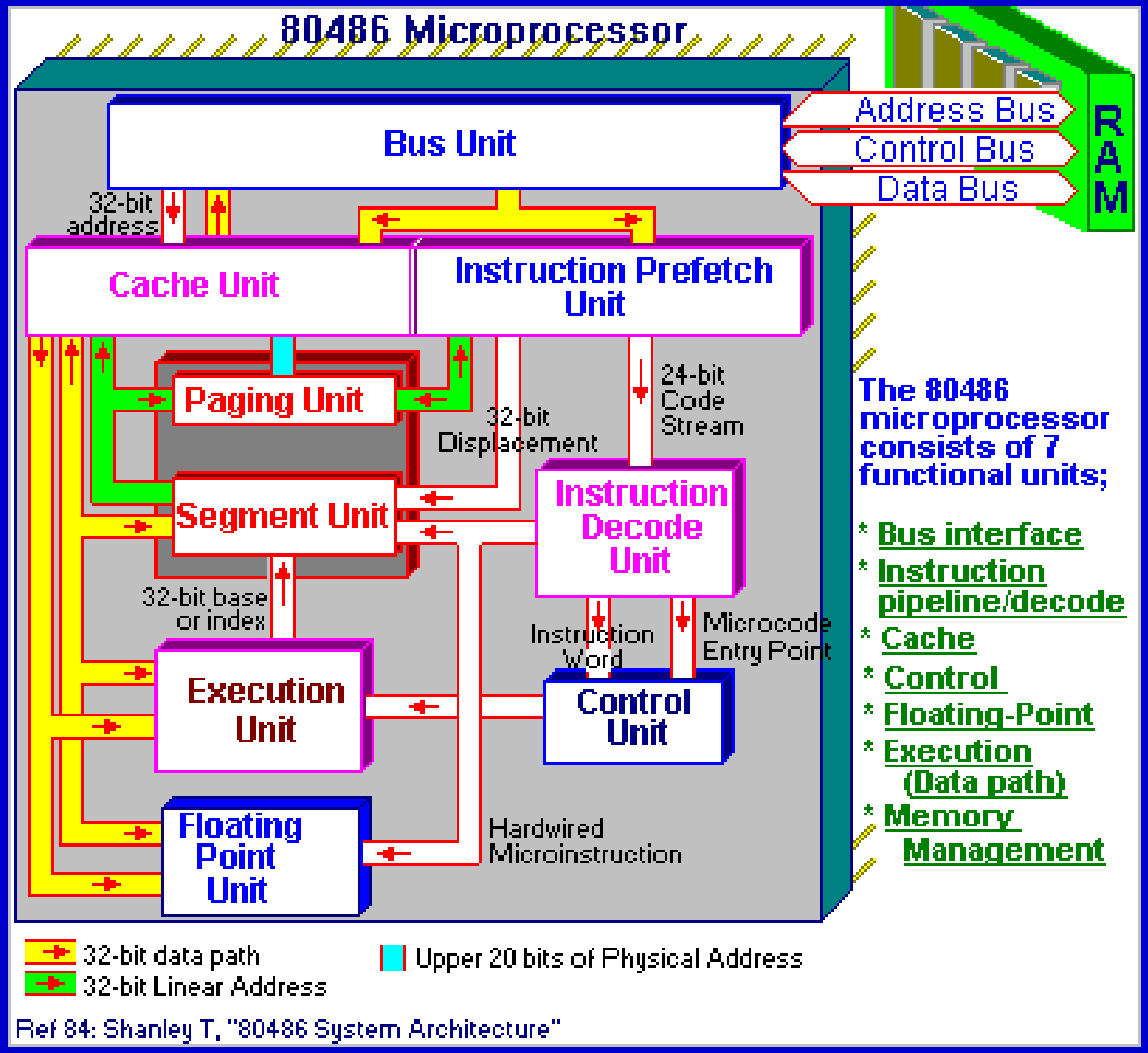
Αρχιτεκτονική 8086



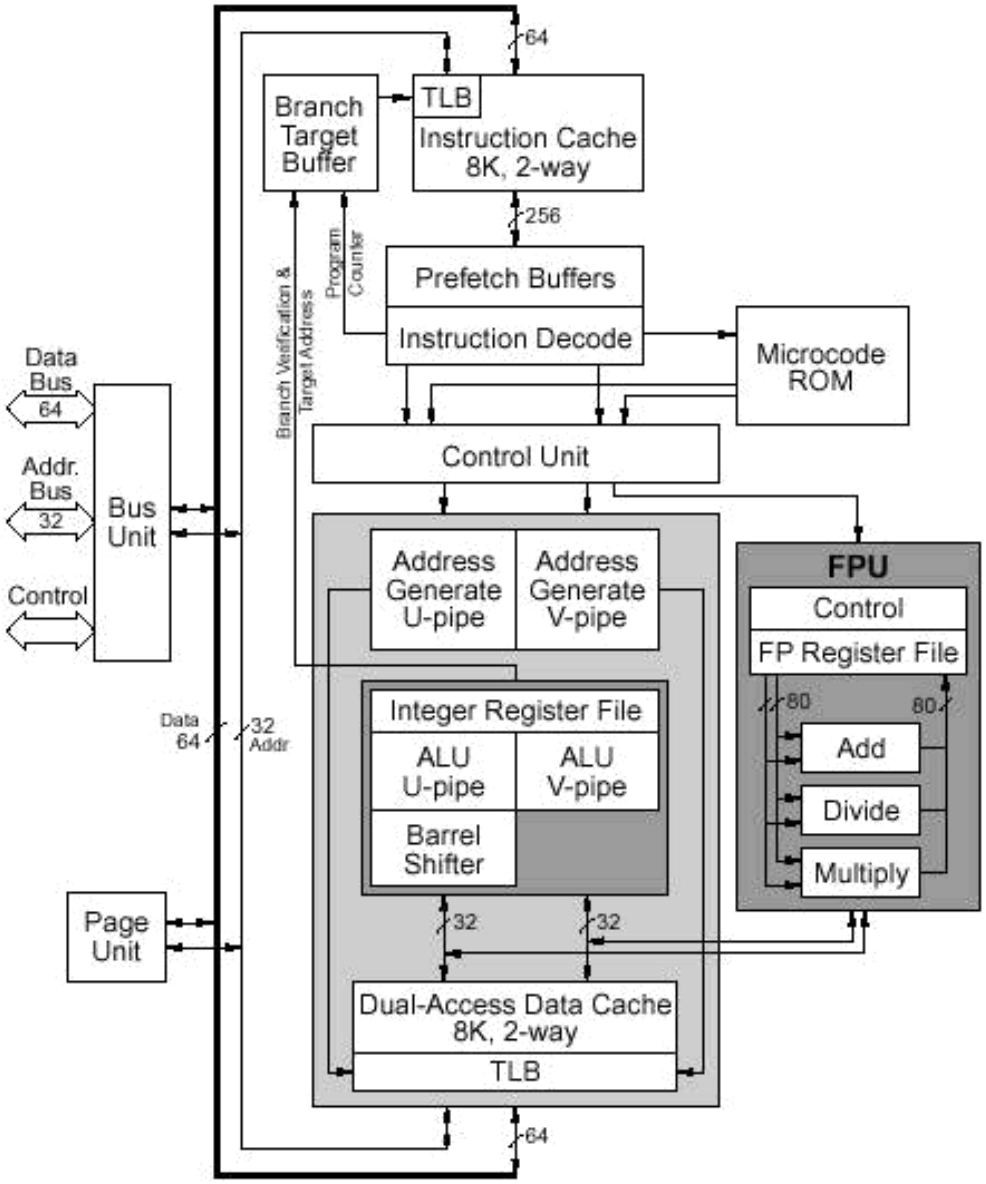
Αρχιτεκτονική 80386



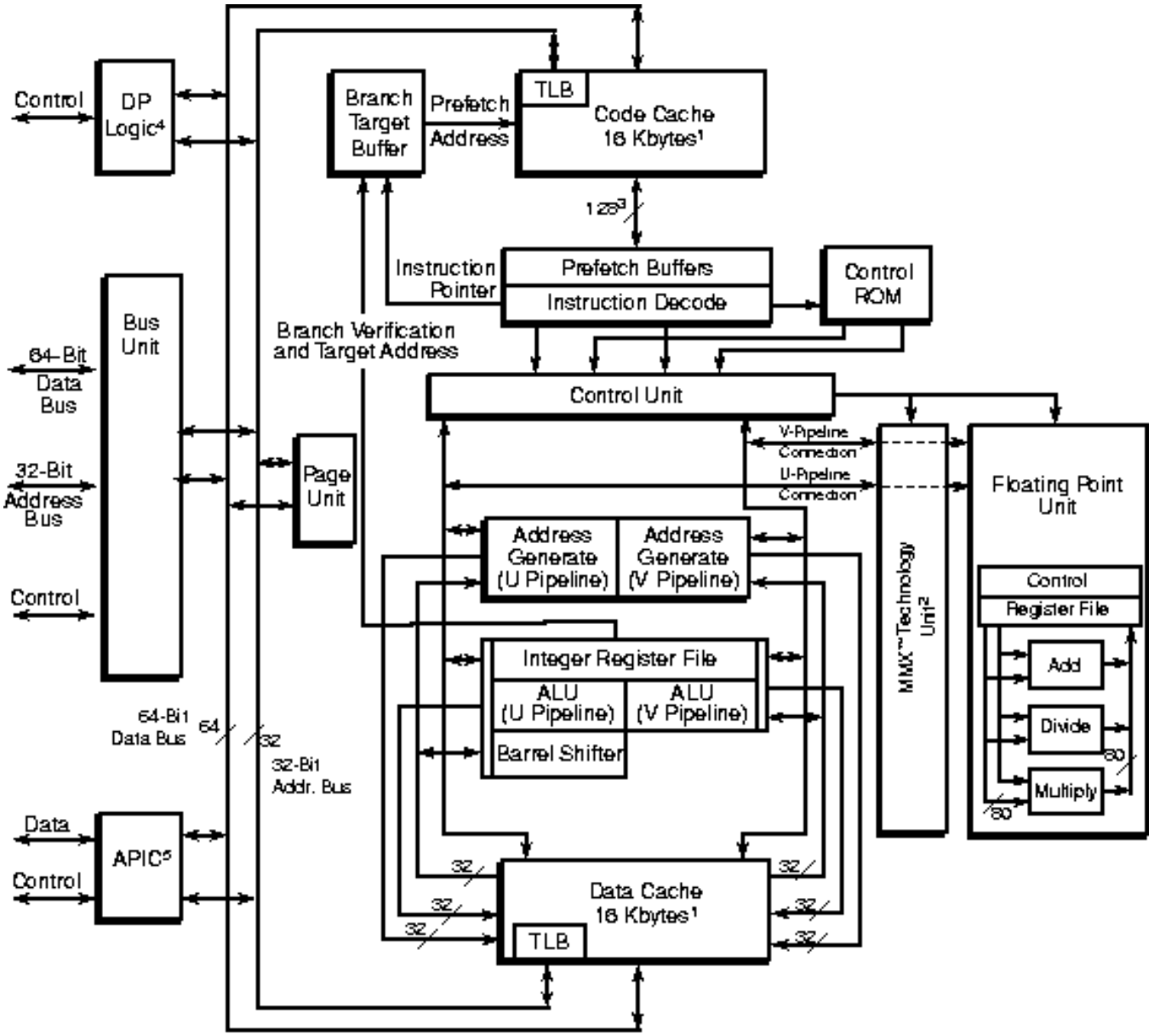
Αρχιτεκτονική 80486



Αρχιτεκτονική Pentium



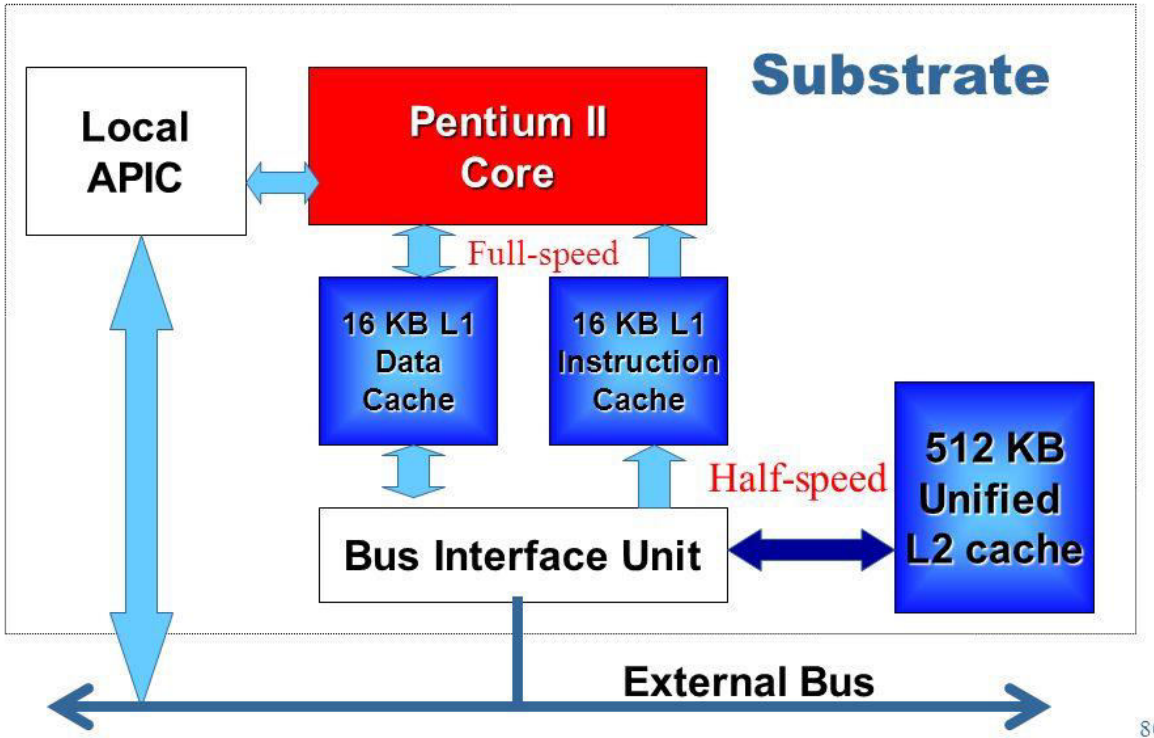
Αρχιτεκτονική Pentium MMX



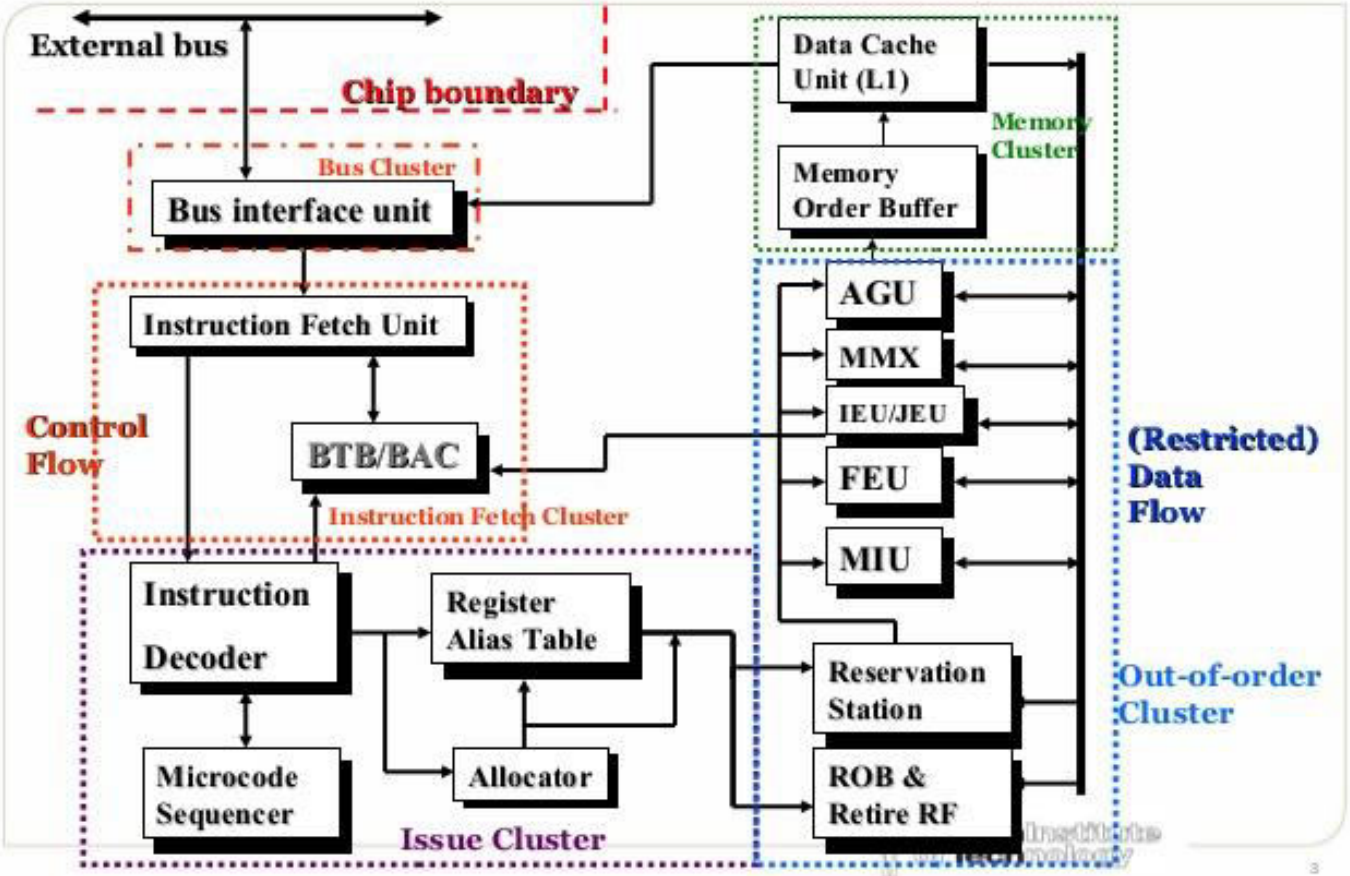
Pentium Pro

Pentium II

Pentium II (P6)



P6 Microarchitecture

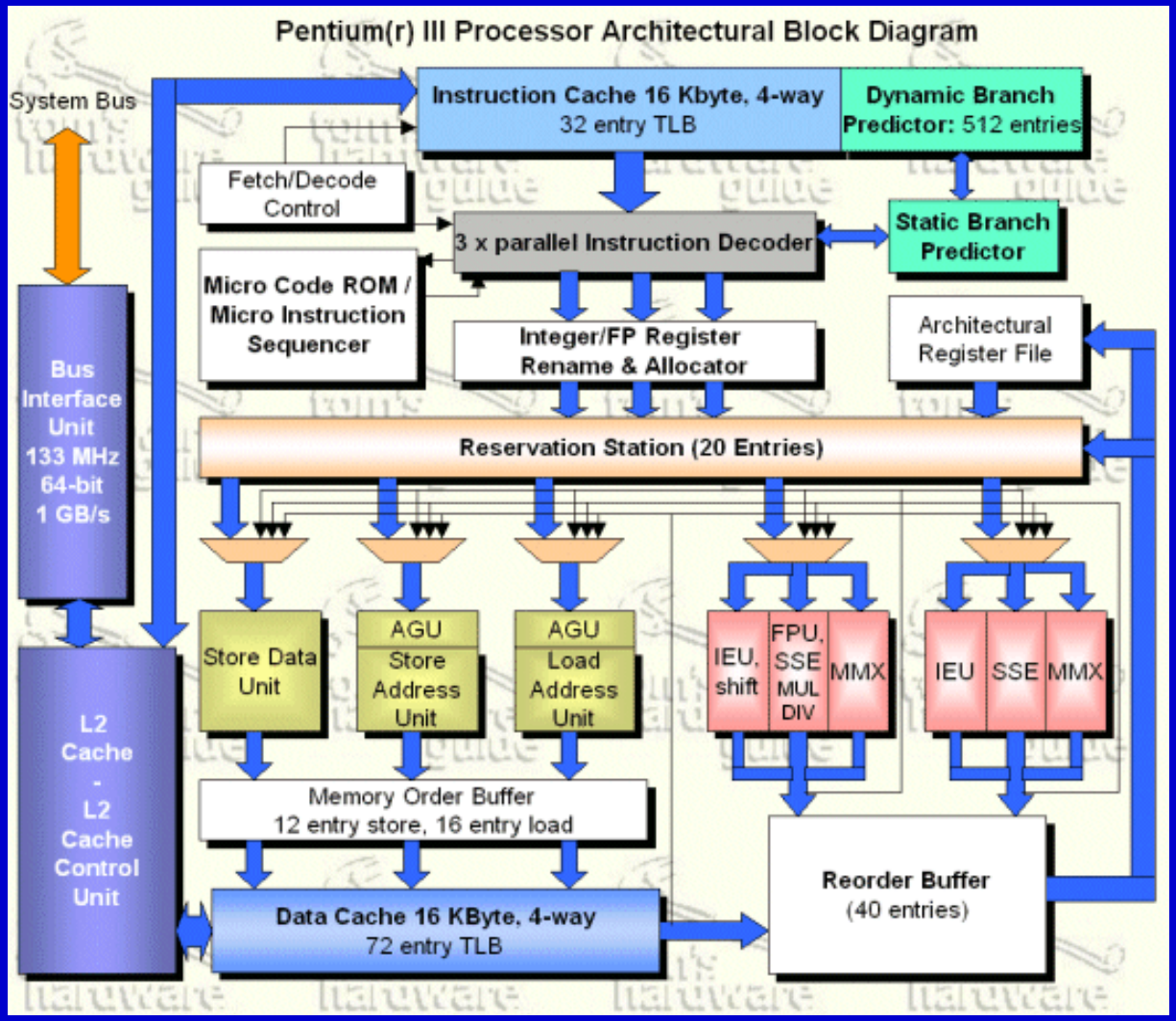


Pentium III

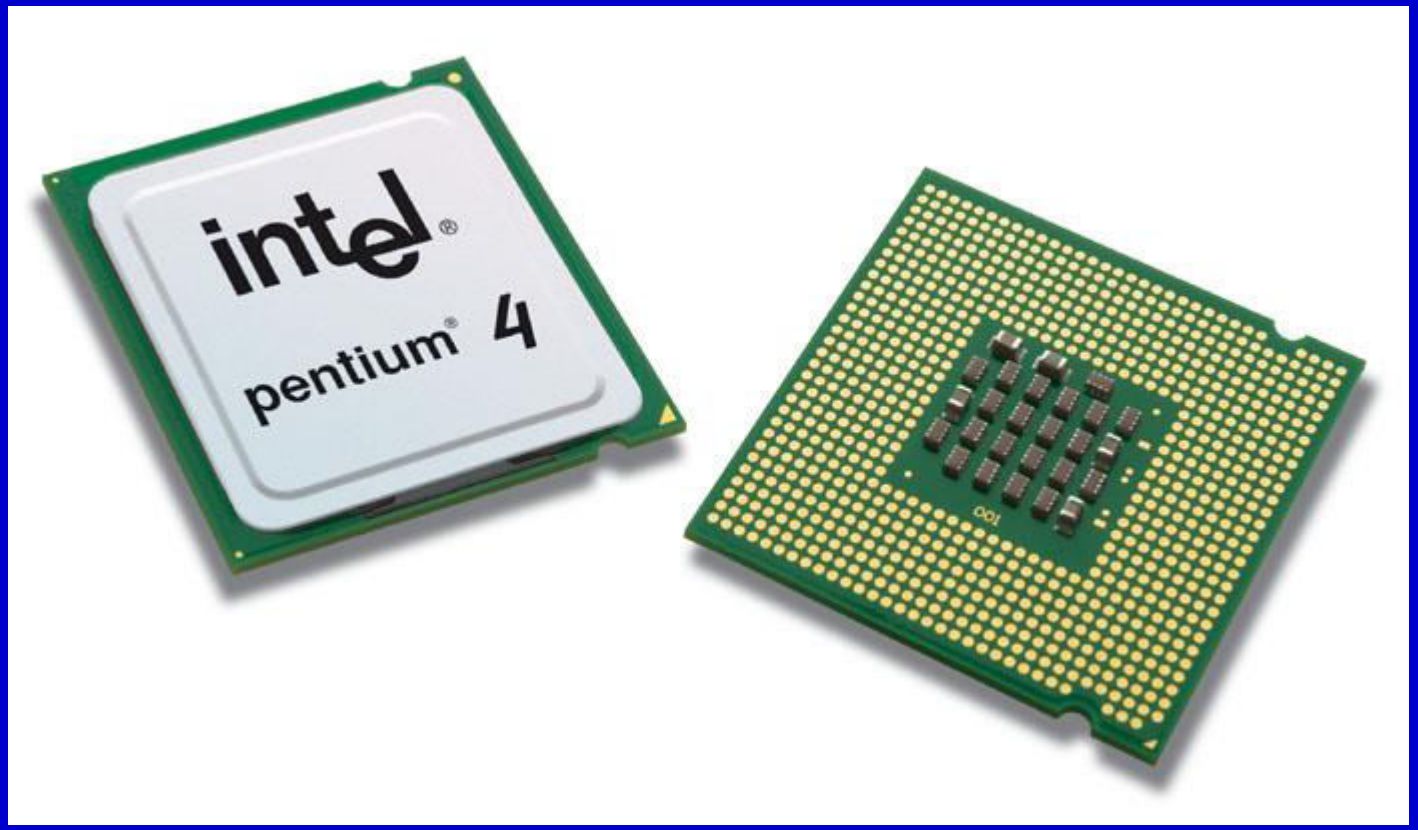

3936A845
U1
U2
U3
J1


INTEL©'01 PHILIPPINES
1133/256/133/1.475
7130B208-0273
PENTIUM III SL5GQ

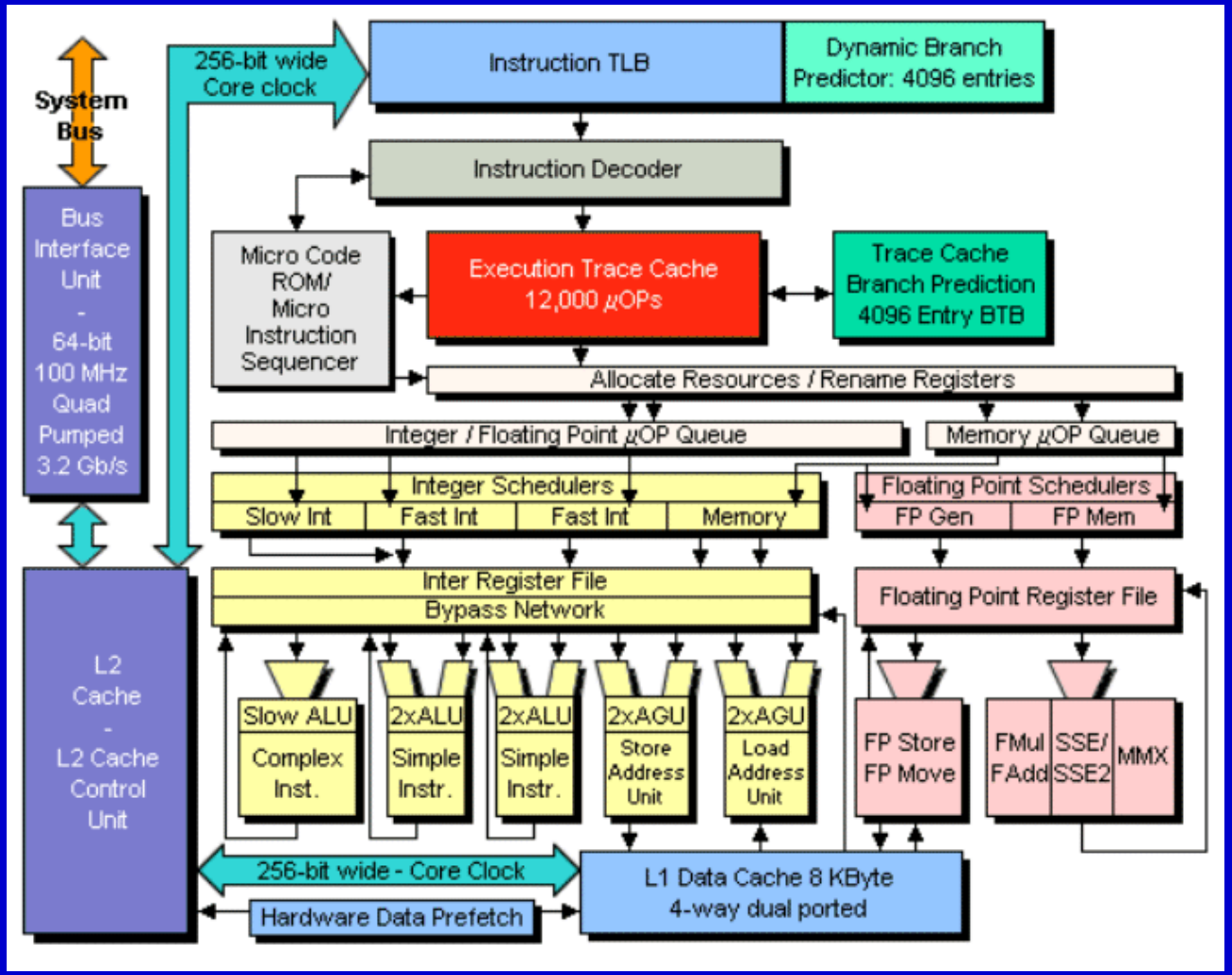
Αρχιτεκτονική Pentium III



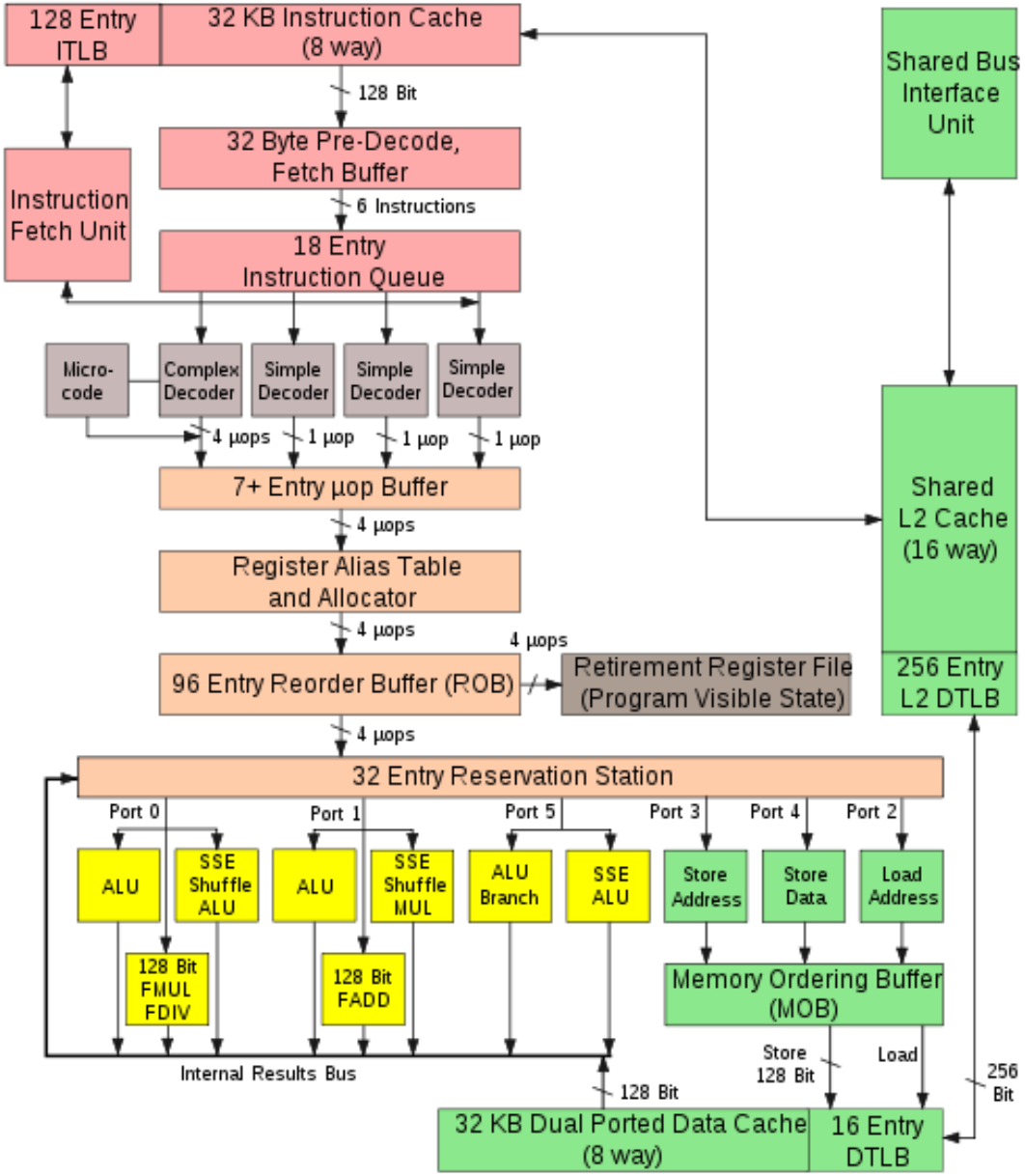
Intel Pentium 4



Αρχιτεκτονική Pentium 4

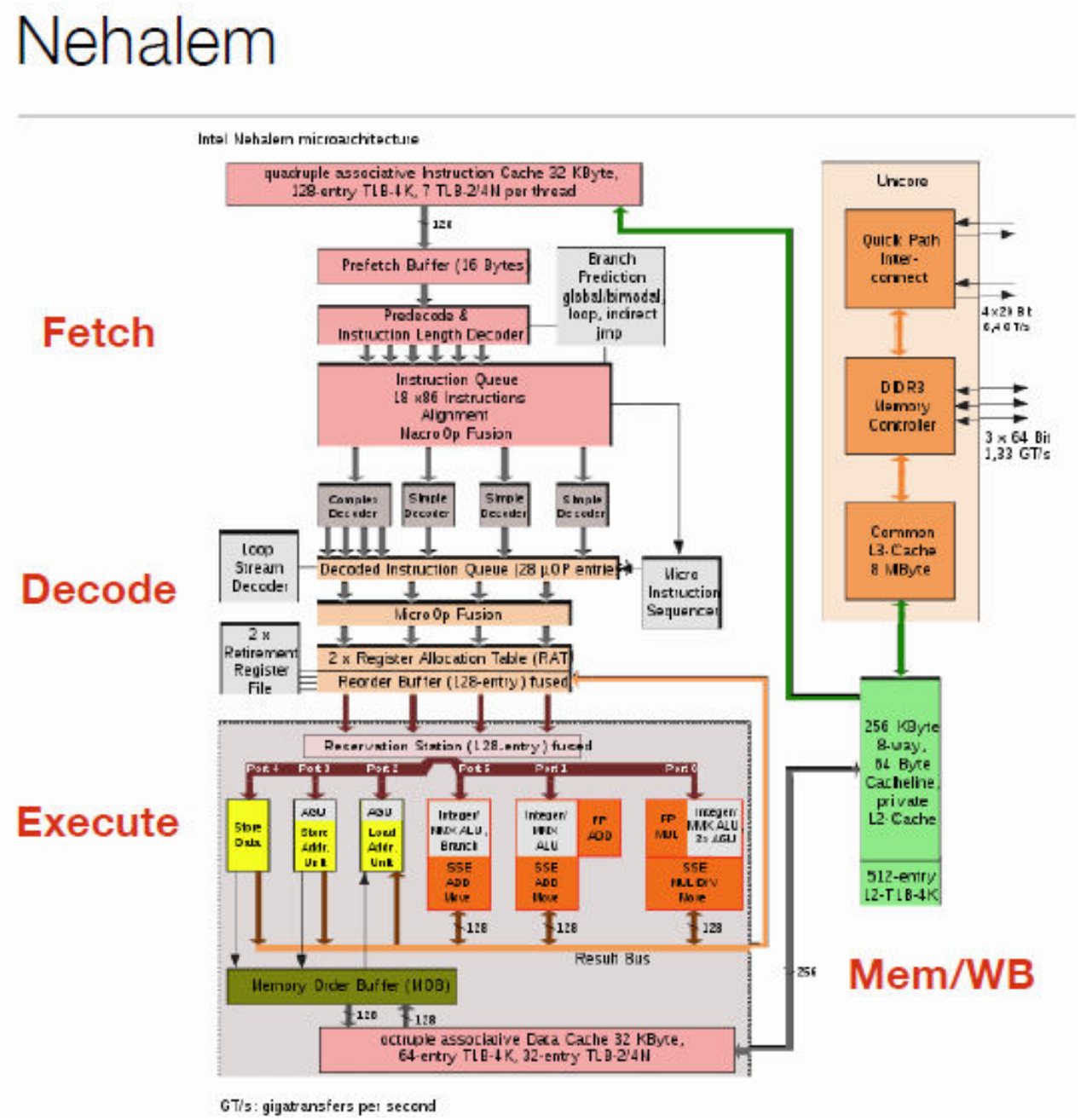


Core 2 Architecture



Intel Core 2 Architecture

Αρχιτεκτονική core i7



Sequential Processing (386)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$					Fetch	Decode	Execute	Write	
$Instr_3$									Fetch

- Sequential processing works on one instruction at a time

Pipelined Processing (486)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$		Fetch	Decode	Execute	Write				
$Instr_3$			Fetch	Decode	Execute	Write			
$Instr_4$				Fetch	Decode	Execute	Write		
$Instr_5$					Fetch	Decode	Execute	Write	
$Instr_6$						Fetch	Decode	Execute	Write

- Latency — elapsed time from start to completion of a particular task
- Throughput — how many tasks can be completed per unit of time
- Pipelining only improves throughput
  - Each job still takes 4 cycles to complete
- Real life analogy: Henry Ford's automobile assembly line

In-Order Pipeline (486)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$		Fetch	Decode	Wait		Execute	Write		
$Instr_3$			Fetch	Decode	Wait		Execute	Write	
$Instr_4$				Fetch	Decode	Wait		Execute	Write
$Instr_5$					Fetch	Decode	Wait		Execute
$Instr_6$						Fetch	Decode	Wait	

- In-Order execution requires instructions to be executed in the original program order

Superscalar Issue (Pentium)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$	Fetch	Decode	Wait			Execute	Write		
$Instr_3$		Fetch	Decode	Execute	Write				
$Instr_4$		Fetch	Decode	Wait			Execute	Write	
$Instr_5$			Fetch	Decode	Execute	Write			
$Instr_6$			Fetch	Decode	Execute	Write			
$Instr_7$				Fetch	Decode	Execute	Write		
$Instr_8$				Fetch	Decode	Execute	Write		

- Superscalar issue allows multiple instructions to be issued at the same time

Out-of-Order Execution (Pentium II)

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$		Fetch	Decode	Wait		Execute	Write		
$Instr_3$			Fetch	Decode	Execute	Write			
$Instr_4$				Fetch	Decode	Wait	Execute	Write	
$Instr_5$					Fetch	Decode	Execute	Write	
$Instr_6$						Fetch	Decode	Execute	Write

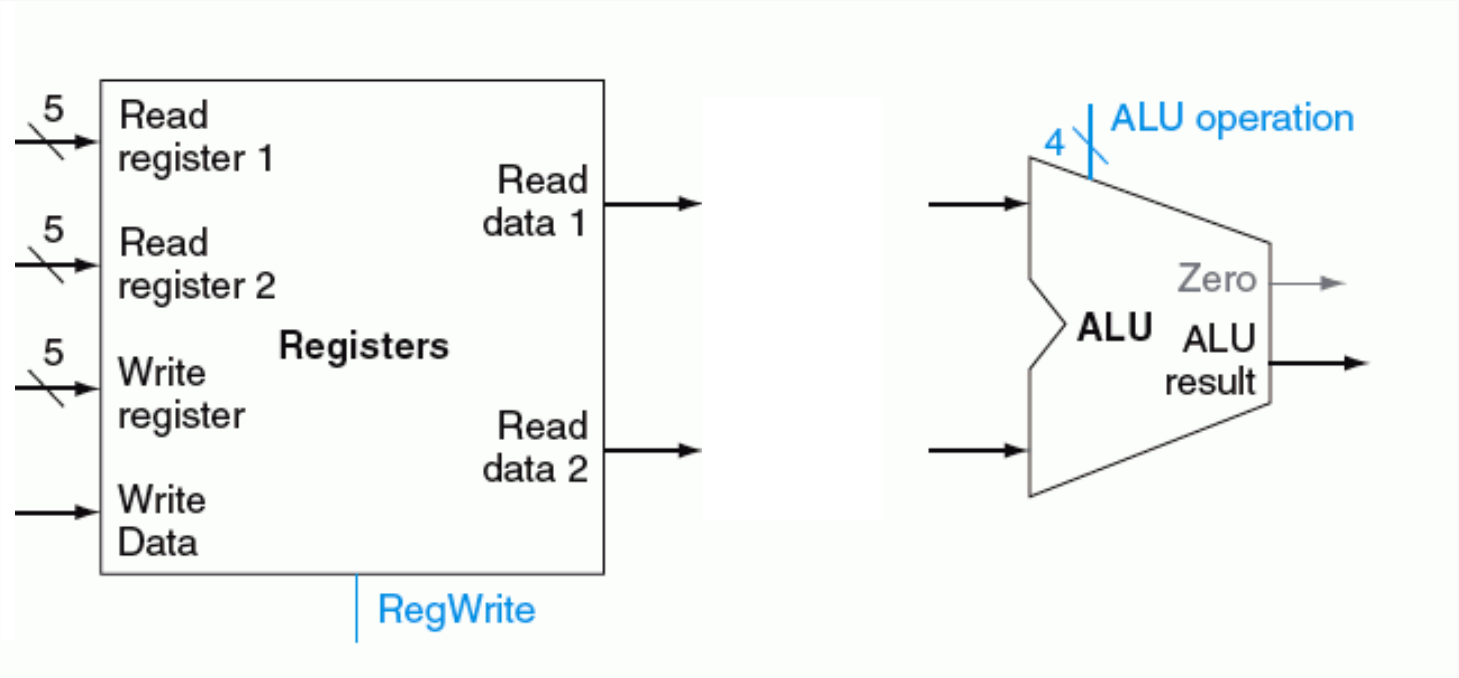
- Program Order vs Dataflow Order
- Dataflow: data-driven scheduling of events
  - The start of an event should be enabled by the availability of its required input (data dependency)
- Real life analogy: taking tests — work on the questions you know first

10.1. Έστω register file με δομή όπως αυτή της προηγούμενης άσκησης. Υποθέστε ότι *n*=8, δηλαδή ότι το register file έχει 8 καταχωρητές και έστω ότι οι καταχωρητές έχουν μήκος 16 bit. Σε δεδομένη χρονική στιγμή οι τιμές των περιεχομένων των καταχωρητών είναι αυτές που δίδονται στην συνέχεια στο δεκαεξαδικό σύστημα.

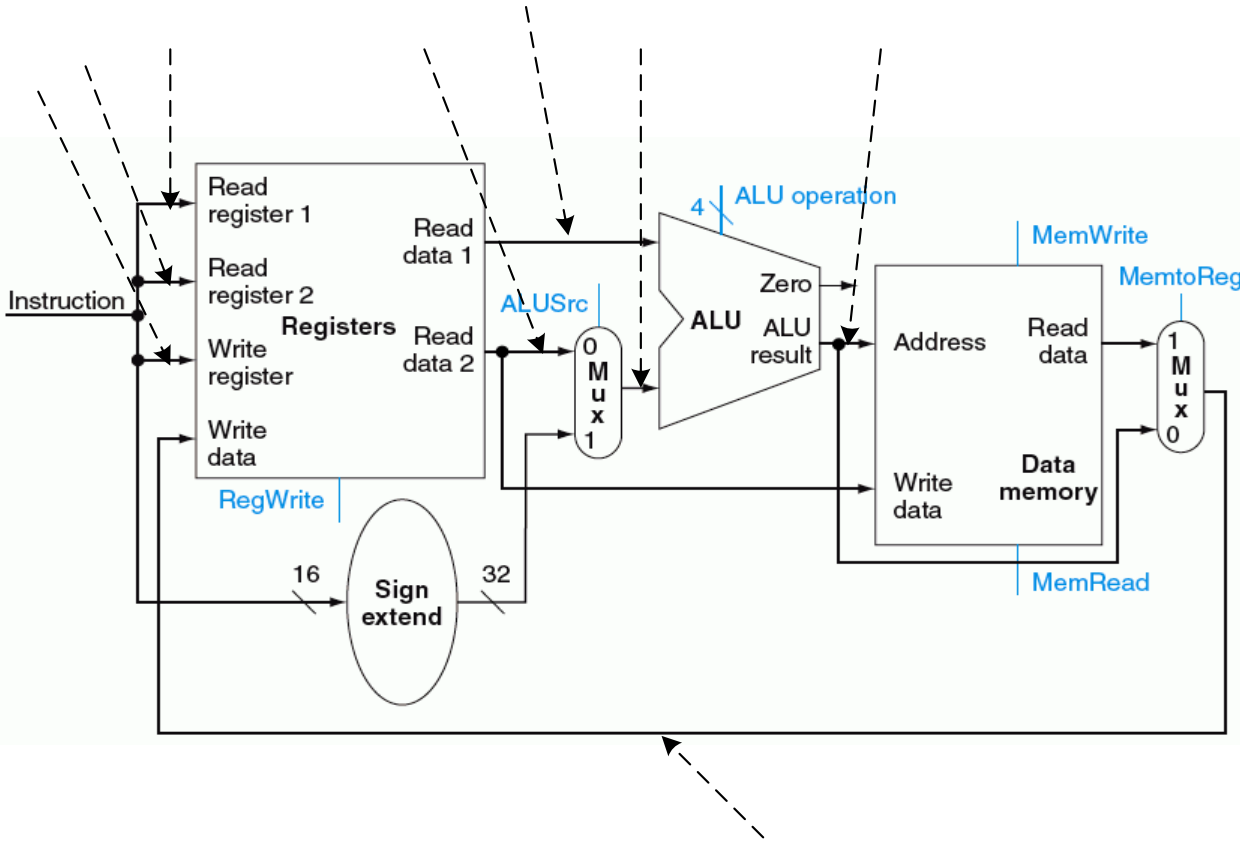
R0=0x0000, R1=0x0001, R2=0x0002, R3=0x0003,
R4=0x00F4, R5=0x00F5, R6=0x00F6, R7=0x00F7.

10.2. Στο σχήμα που δίδεται στην συνέχεια κάντε τις κατάλληλες συνδέσεις για να είναι δυνατή η εκτέλεση εντολών τύπου

add rd, rs, rt, or rd, rs, rt, ……..



10.3. Έστω ότι στον επεξεργαστή μMIIPS εκτελείται η εντολή add $1, $2, $3 και ότι οι τιμές των καταχωρητών $2, $3 είναι 0x00000000, 0x00000001. Στο διάγραμμα βαθμίδων που ακολουθεί δώστε τις τιμές των διασυνδέσεων που σημειώνονται.

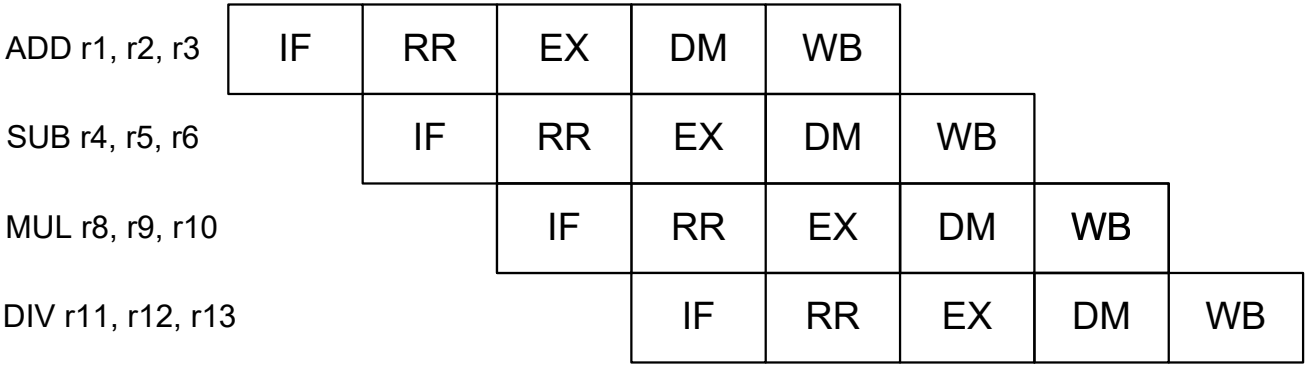


10.4. Ένας επεξεργαστής χωρίς pipeline έχει χρόνο κύκλου εντολής 25 ns. Ποιος είναι ο χρόνος κύκλου ενός αντίστοιχου επεξεργαστή με 5 βαθμίδες pipeline που έχουν αντίστοιχα καθυστέρηση 5, 7, 3, 6, 4 ns, εάν η καθυστέρηση των pipeline latch είναι 1 ns.

10.5. Ας υποθέσουμε ότι διαθέτουμε μία μονάδα επεξεργασίας δεδομένων με 5 βαθμίδες pipeline (IF, RR, EX, DM, WB). Να δοθεί το διάγραμμα (χρόνου) της εκτέλεσης των εντολών του επόμενου τμήματος προγράμματος. (IF: Instruction Fetch, RR: Read Registers, EX: Execute, DM: Access Data Memory, WB: Write Back).

```
ADD r1, r2, r3
SUB r4, r5, r6
MUL r8, r9, r10
DIV r12, r13, r14
```

Λύση

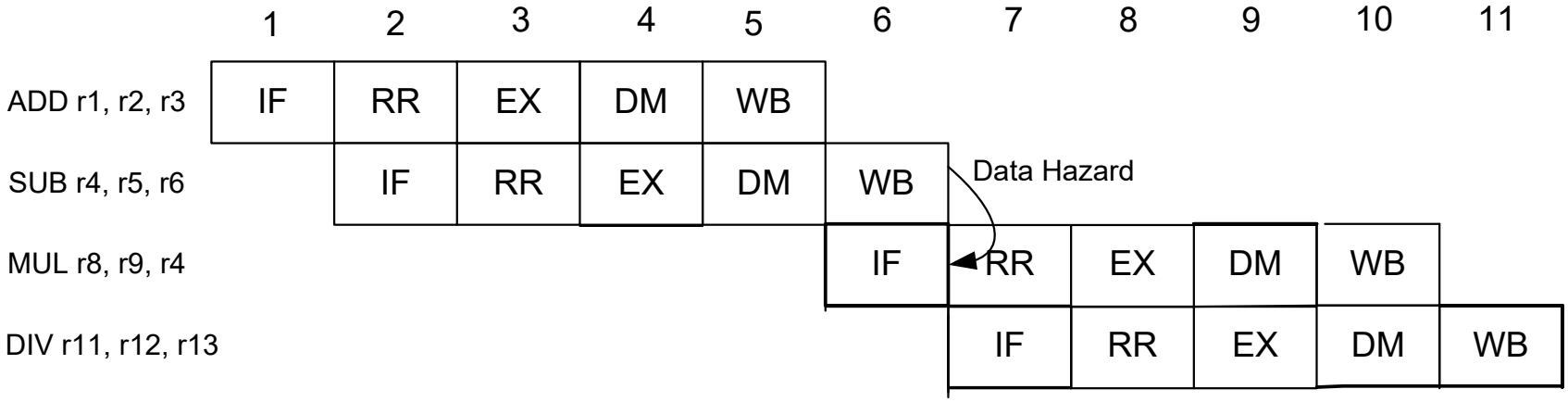


10.6. Ας υποθέσουμε ότι διαθέτουμε μία μονάδα επεξεργασίας δεδομένων με 5 βαθμίδες pipeline (IF, RR, EX, DM, WB). Να δοθεί το διάγραμμα (χρόνου) της εκτέλεσης των εντολών του επόμενου τμήματος προγράμματος.
(IF: Instruction Fetch, RR: Read Registers, EX: Execute, DM: Access Data Memory, WB: Write Back).

```
ADD r1, r2, r3
SUB r4, r5, r6
MUL r8, r9, r4
DIV r11, r12, r13
```

Λύση

Για την ορθή εκτέλεση των εντολών πρέπει η εντολή MUL να διαβάσει την τιμή του καταχωρητή r4 όπως αυτή υπολογίστηκε από την SUB. Επομένως η εκτέλεση των εντολών πρέπει να γίνει όπως την συνέχεια



10.7. Σε έναν επεξεργαστή με υπερβαθμωτή (superscalar) αρχιτεκτονική με δύο μονάδες επεξεργασίας δεδομένων στην οποία η εκτέλεση όλων των εντολών διαρκεί έναν κύκλο εκτελείται το επόμενο τμήμα προγράμματος. Ποιες εντολές εκτελούνται σε κάθε κύκλο.

```
ADD  r1, r2, r3
LD   r4, (r5)
SUB  r7, r1, r9
MUL  r5, r4, r4
SUB  r1, r12, r10
ST   (r13), r14
OR   r15, r14, r12
```

Λύση

Κύκλος 1	ADD r1, r2, r3	LD r4, (r5)
Κύκλος 2	SUB r7, r1, r9	MUL r5, r4, r4
Κύκλος 3	SUB r1, r12, r10	ST (r13), r14
Κύκλος 4	OR r15, r14, r12	

Η διάρκεια εκτέλεσης είναι 4 κύκλοι.

10.8. Σε έναν επεξεργαστή με αρχιτεκτονική in-order superscalar με **δύο** μονάδες επεξεργασίας δεδομένων (ΜΕ) στην οποία η εκτέλεση όλων των εντολών διαρκεί έναν κύκλο. Ποιες εντολές εκτελούνται σε κάθε κύκλο.

```
LD     r1, (r2)
SUB  r4, r5, r6
ADD r3, r1, r7
MUL r8, r3, r3
ST (r11), r4
ST (r12), r8
ADD r15, r14, r13
SUB r10, r15, r10
ST (r9), r10
```

Λύση

	ME1	ME2
Κύκλος 1	LD r1, (r2)	SUB r4, r5, r6
Κύκλος 2	ADD **r3**, r1, r7	
Κύκλος 3	MUL r8, r3, **r3**	ST (r11), r4
Κύκλος 4	ST (r12), r8	ADD r15, r14, r13
Κύκλος 5	SUB **r10**, r15, r10	
Κύκλος 6	ST (r9), **r10**	