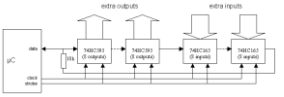
Esempio di architettura RISC: famiglia MIPS

- Riferimenti bibliografici: **Stallings + Hennessy & Patterson**
- Architettura molto regolare con insieme di istruzioni semplice e compatto
- Architettura progettata per una implementazione efficiente della pipeline (lo vedremo più avanti)
- Codifica delle istruzioni omogenea: 32 bit
- Co-processore per istruzioni a virgola mobile e gestione delle eccezioni

MIPS (a 32 bit)

Registri



- 32 registri di 32 bit (registro 0 contiene sempre il valore 0)
- Architettura Load / Store
  - Istruzioni di trasferimento per muovere i dati tra memoria e registri
  - Istruzioni per la manipolazione di dati operano sui valori dei registri
  - Nessuna operazione memoria ↔ memoria
- Quindi: le istruzioni operano su registri (registro i riferito con $i)
- Esempio: add $1, $2, $3

MIPS

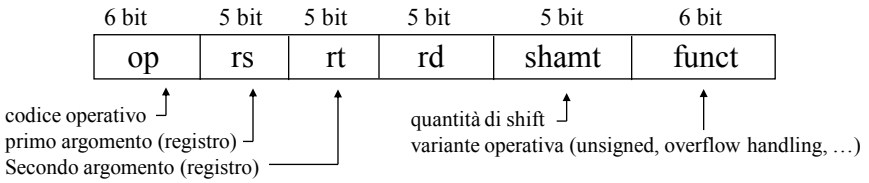
Dati e modi di indirizzamento

- Registri possono essere caricati con byte, mezze parole, e parole (riempiendo con 0 quando necessario o estendendo, cioè replicando, il segno sui bit non coinvolti del registro)
- Modalità di indirizzamento ammesse (con campi di 16 bit):
  - Immediata es. add $2, $2, 0004
  - Displacement es. sw $1, 000c($1)
- Altre modalità derivabili:
  - Indiretta registro (displacement a 0) es. sw $2, 0000($3)
  - Assoluta (registro 0 come registro base) es. lw $1, 00c4($0)

MIPS

Formato Istruzioni

- 32 bit per tutte, 3 formati diversi (formato R, formato I, formato J)
- **Formato R (registro)**



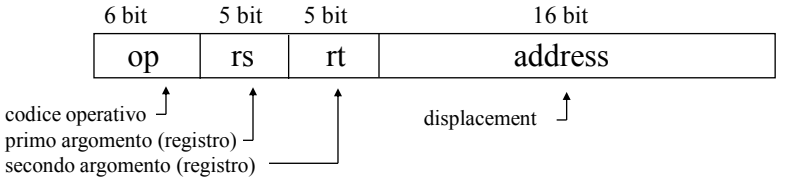
Es: `0x02484020` (`add $8,$18,$8` `[R8] ← [R18] + [R8]`)

op	rs	rt	rd	shamt	funct
000000	10010	01000	01000	00000	100000

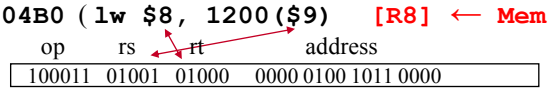
MIPS

Formato Istruzioni

- **Formato I (istruzioni load / store)**



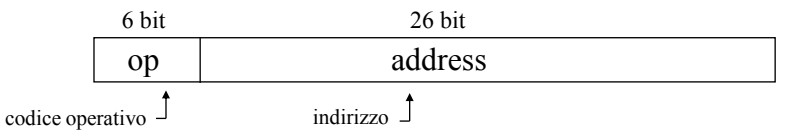
Es: **0x8D2804B0** (**lw $8, 1200($9)** **[R8] ← Mem[1200+[R9]]**)



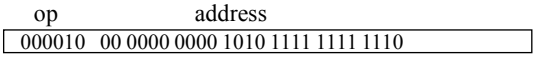
MIPS

Formato Istruzioni

- **Formato J (istruzioni jump)**



Es: **0x0800AFFE** (**j 45054** **[PC] ← 45054**)



Fasi (MIPS)

Fasi senza pipeline:

IF (istruction fetch):

- IR ← Mem[PC] ;
- NPC ← PC + 4 ;

Dove NPC è un registro temporaneo

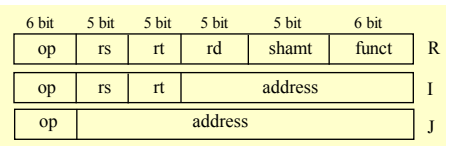
PC è il program counter

Fasi (MIPS)

ID (instruction decode/register fetch cycle):

- A ← Regs[rs] ;
- B ← Regs[rt] ;
- Imm ← campo immediato di IR con segno esteso ;

Dove A, B, Imm sono registri temporanei



Fasi (MIPS)

EX (execution/effective address cycle):

1. ***Riferimento a memoria*** `lw $8, 1200($9)`
   - ALUOutput ← A + Imm ;
2. ***Istruzione ALU registro-registro*** `add $8,$18,$8`
   - ALUOutput ← A ***func*** B ;
3. ***Istruzione ALU registro-immediato*** `addi $8,$18,4`
   - ALUOutput ← A ***op*** Imm ;
4. ***Salto*** `j 45054`
   - ALUOutput ← NPC + (Imm << 2);
   - Cond ← (A = = 0) ;

shift a sinistra

6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	
op	rs	rt	rd	shamt	funct	R
op	rs	rt	address			I
op	address					J

Fasi (MIPS)

MEM (memory access/branch completion cycle):

- PC ← NPC ; in tutti i casi

1. ***Riferimento a memoria*** `lw $8, 1200($9)`
   - LMD ← Mem[ALUOutput] or
     Mem[ALUOutput] ← B; `sw $5, 1700($4)`
2. ***Salto***
   - if (Cond) PC ← ALUOutput ; `j 45054`

6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	
op	rs	rt	rd	shamt	funct	R
op	rs	rt	address			I
op	address					J

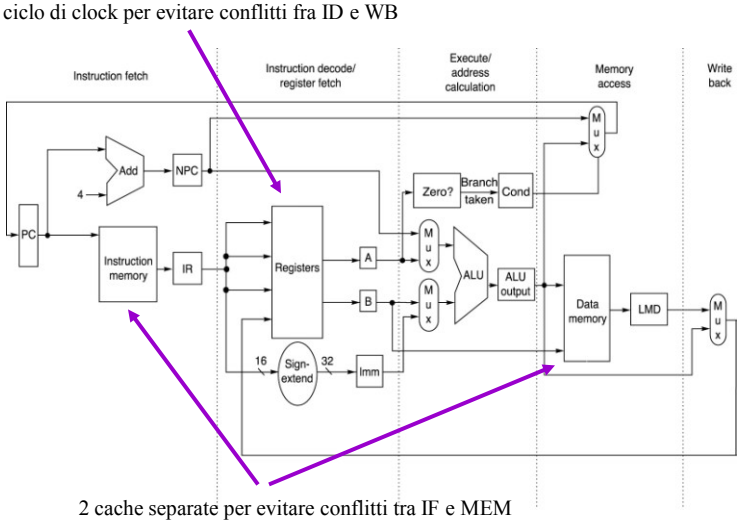
Fasi (MIPS)

WB (write/back cycle):

1. ***Istruzione ALU registro-registro*** `add $8,$18,$8`
   - Regs[rd] ← ALUOutput ;
2. ***Istruzione ALU registro-immediato*** `addi $8,$18,4`
   - Regs[rt] ← ALUOutput ;
3. ***Istruzione Load*** `lw $8, 1200($9)`
   - Regs[rt] ← LMD ;

6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	
op	rs	rt	rd	shamt	funct	R
op	rs	rt	address			I
op	address					J

registri letti (anche 2 volte) e scritti nello stesso ciclo di clock per evitare conflitti fra ID e WB

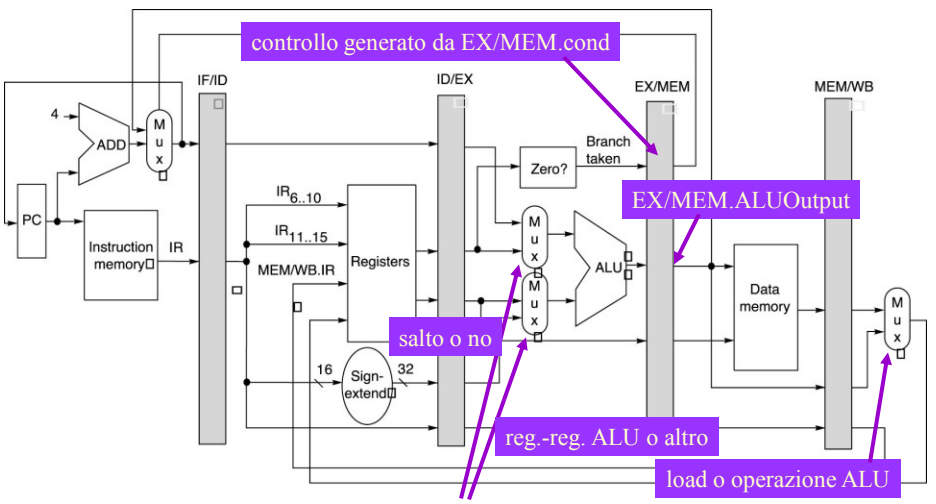


2 cache separate per evitare conflitti tra IF e MEM

Pipeline (MIPS)

- Architettura che si presta ad una facile introduzione della pipeline: uno stadio per fase, 1 ciclo di clock per stadio
- Occorre memorizzare i dati fra una fase e la successiva: si introducono opportuni registri (denominati pipeline registers o pipeline latches) fra i vari stadi della pipeline
- Tali registri memorizzano sia dati che segnali di controllo che devono transitare da uno stadio al successivo
- Dati che servono a stadi non immediatamente successivi vengono comunque copiati nei registri dello stato successivo per garantire la correttezza dei dati

Pipeline (MIPS)



Settati a seconda del tipo di istruzione (codificato nel campo ID/EX.IR)

Pipeline (MIPS)

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC] IF/ID.NPC,PC ← (if ((EX/MEM.opcode == branch) && EX/MEM.cond){EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR[rs]]; ID/EX.B ← Regs[IF/ID.IR[rt]]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← sign-extend (IF/ID.IR[immediate field]);		
	ALU instruction	Load or store instruction	Branch instruction
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm;	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.B ← ID/EX.B	EX/MEM.ALUOutput ← ID/EX.NPC + (ID/EX.Imm << 2); EX/MEM.cond ← (ID/EX.A == 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOputput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR[rd]] ← MEM/WB.ALUOutput; or Regs[MEM/WB.IR[rt]] ← MEM/WB.ALUOutput;	For load only: Regs[MEM/WB.IR[rt]] ← MEM/WB.LMD;	

Pipeline (MIPS)

- Quando una istruzione passa dalla fase ID a quella EX si dice che la istruzione è stata “rilasciata” (issued)
- Nella Pipeline MIPS **è possibile individuare tutte le dipendenze dai dati nella fase ID**
- Se si rileva una dipendenza dai dati per una istruzione, questa **va in stallo prima di essere rilasciata**
- Inoltre, sempre nella fase ID, è possibile determinare che tipo di **data forwarding** adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo
- Vediamo di seguito come realizzare un forwarding nella fase EX per una dipendenza di tipo RAW (Read After Write) con sorgente che proviene da una istruzione load (load interlock)

Pipeline (MIPS)

Possibili casi

Situazione	Esempio di codice	Azione
Nessuna dipendenza	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, $6, $7 OR $9, $6, $7	Non occorre fare nulla perché non c'è dipendenza rispetto alle 3 istruzioni successive
Dipendenza che richiede uno stallo	LD **$1**, 45($2) DADD $5, **$1**, $7 DSUB $8, $6, $7 OR $9, $6, $7	Opportuni comparatori rilevano l'uso di $1 in DADD ed evitano il rilascio di DADD
Dipendenza risolvibile con un forwarding	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, **$1**, $7 OR $9, $6, $7	Opportuni comparatori rilevano l'uso di $1 in DSUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di DSUB
Dipendenza con accessi in ordine	LD **$1**, 45($2) DADD $5, $6, $7 DSUB $8, $6, $7 OR $9, **$1**, $7	Non occorre fare nulla perché la lettura di $1 in OR avviene dopo la scrittura del dato caricato

Pipeline (MIPS)

Condizioni per riconoscere le dipendenze

Opcode (ID/EX)	Opcode (IF/ID)	Matching operand fields
Load	R-R ALU	ID/EX.IR[rt] == IF/ID.IR[rs]
Load	R-R ALU	ID/EX.IR[rt] == IF/ID.IR[rt]
Load	Load, Store, Imm ALU, branch	ID/EX.IR[rt] == IF/ID.IR[rs]

Pipeline (MIPS)

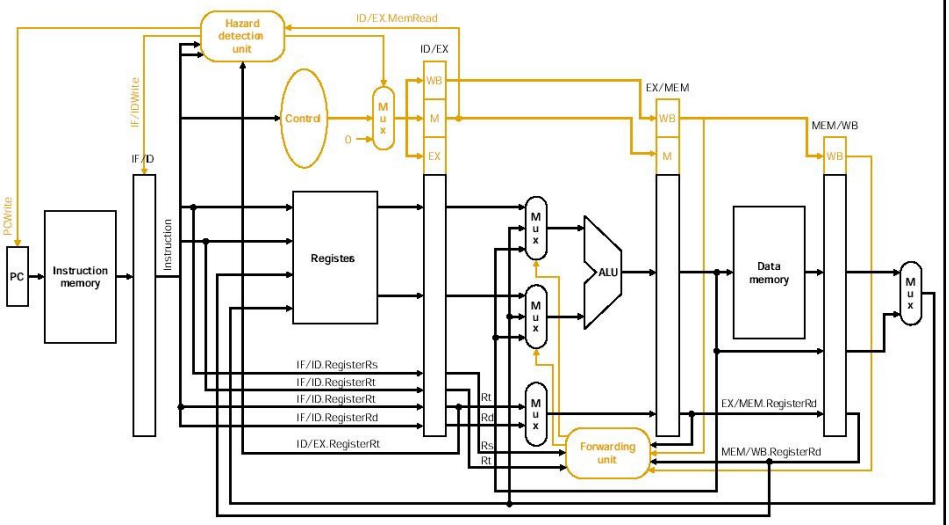
- La logica per decidere come effettuare il forwarding è simile a quella appena vista per individuare le dipendenze, ma considera molti più casi
- Una osservazione chiave è che i registri di pipeline contengono:
  - dati su cui effettuare il forwarding
  - i campi registro sorgente e destinazione
- Tutti i dati su cui effettuare il forwarding provengono:
  - dall'output della ALU
  - dalla memoria dati
- … e sono diretti verso:
  - l'input della ALU
  - l'input della memoria dati
  - il comparatore con 0
- Quindi occorre confrontare i registri destinazione di IR in EX/MEM e MEM/WB con i registri sorgente di IR in ID/EX e EX/MEM

Pipeline (MIPS)

Pipeline register containing source instruction	Opcode of source instruction	Pipeline register containing destination instruction	Opcode of destination instruction	Destination of the forwarded result	Comparison (if equal then forward)
EX/MEM	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	EX/MEM.IR[rd] == ID/EX.IR[rs]
EX/MEM	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	EX/MEM.IR[rd] == ID/EX.IR[rt]
MEM/WB	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rd] == ID/EX.IR[rs]
MEM/WB	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rd] == ID/EX.IR[rt]
EX/MEM	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	EX/MEM.IR[rt] == ID/EX.IR[rs]
EX/MEM	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	EX/MEM.IR[rt] == ID/EX.IR[rt]
MEM/WB	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rt] == ID/EX.IR[rs]
MEM/WB	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rt] == ID/EX.IR[rt]
MEM/WB	Load	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	MEM/WB.IR[rt] == ID/EX.IR[rs]
MEM/WB	Load	ID/EX	Register-register ALU	Bottom ALU input	MEM/WB.IR[rt] == ID/EX.IR[rt]

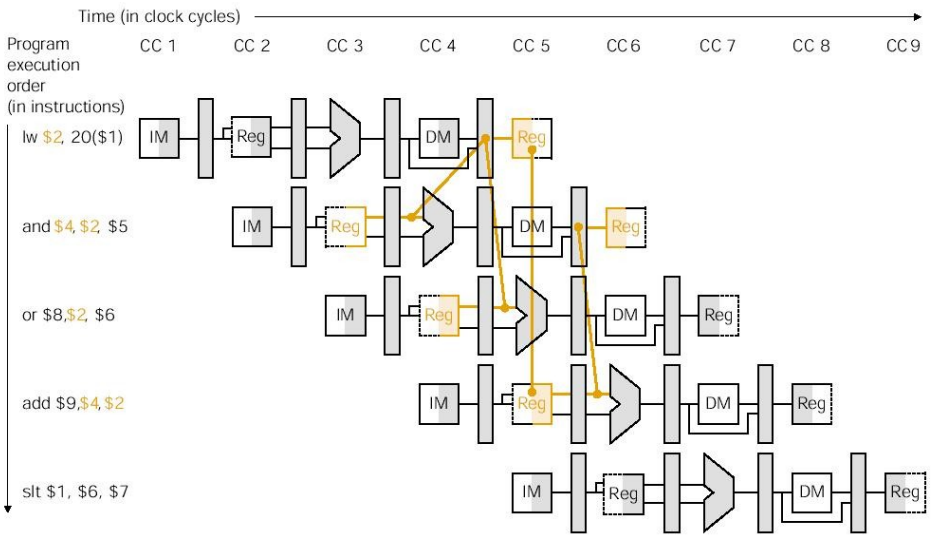
Pipeline (MIPS)

Introduzione hardware aggiuntivo per gestire il data forwarding



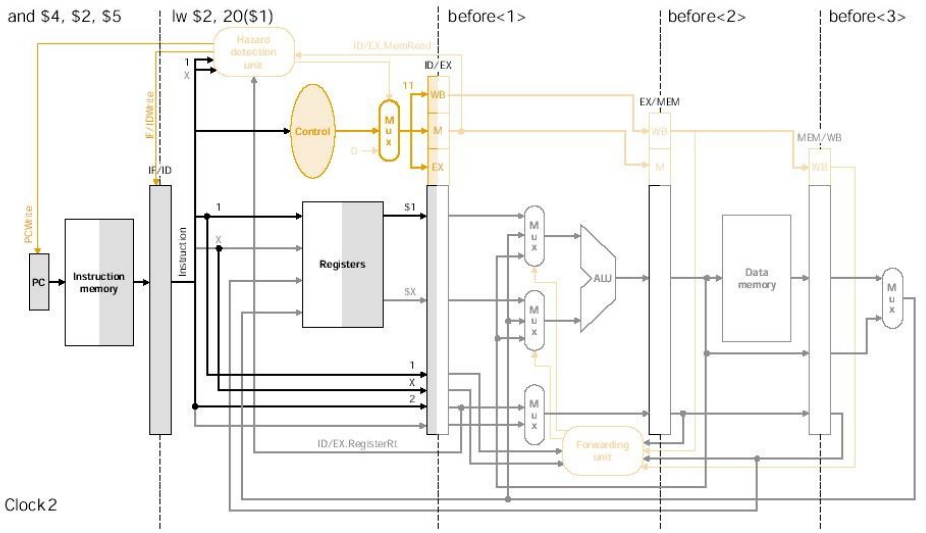
Pipeline (MIPS)

Esempio

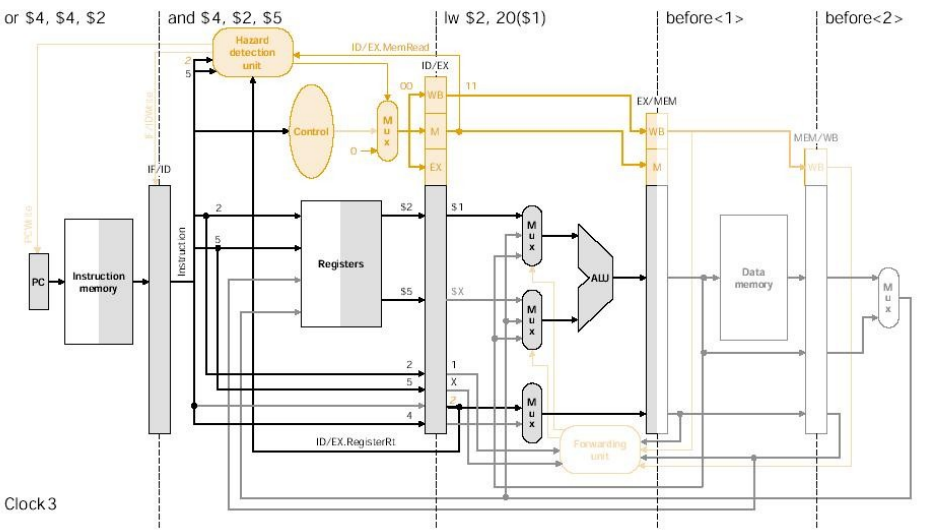


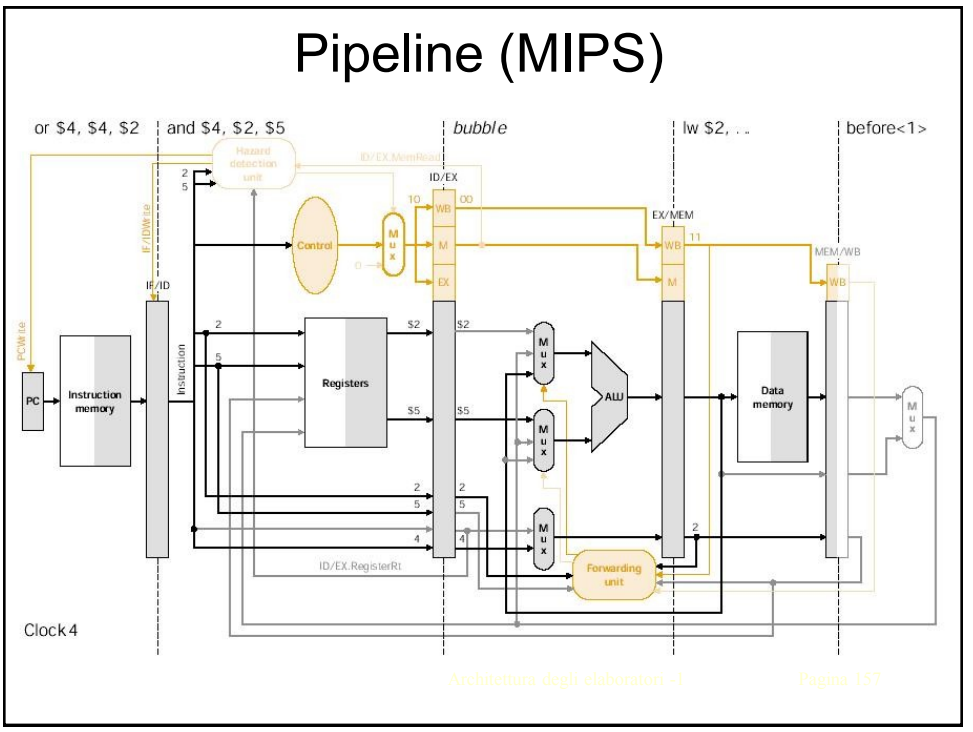
i

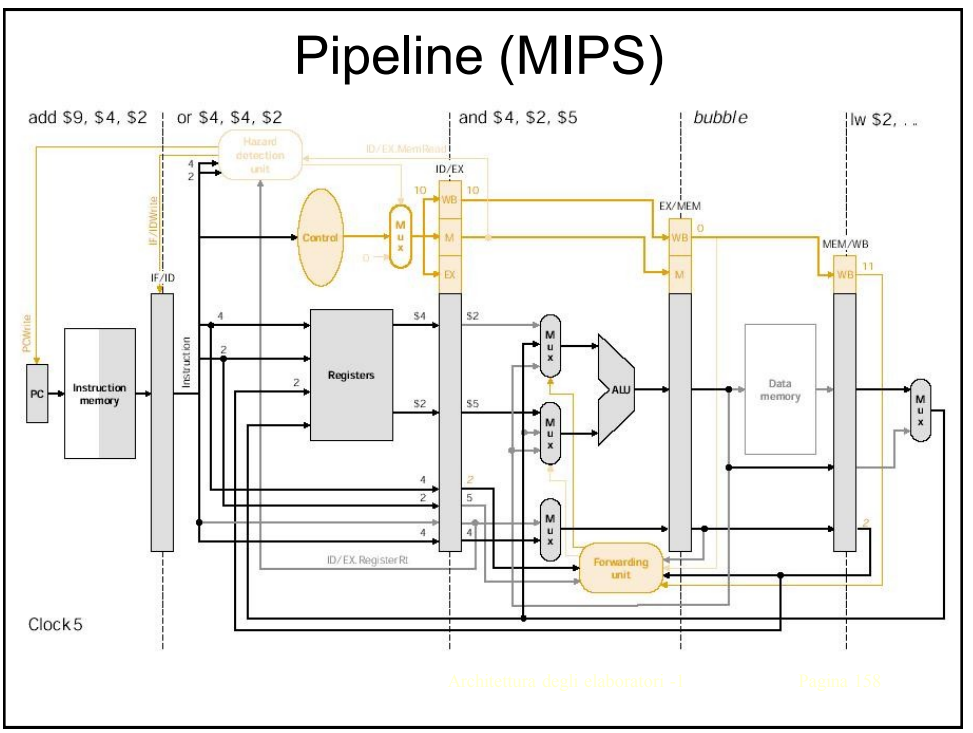
Pipeline (MIPS)



Pipeline (MIPS)

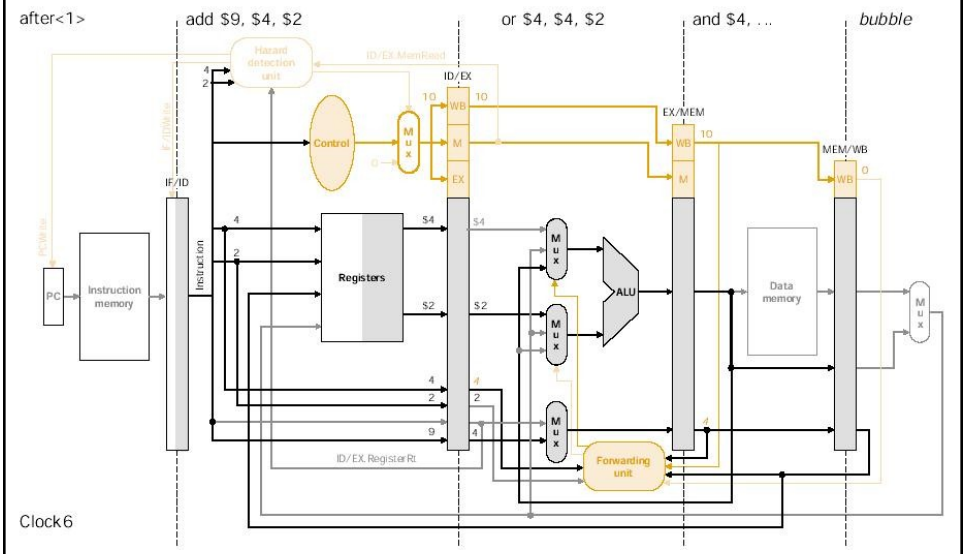



Pipeline (MIPS)
or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . .
before<1>
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
Control
Mux
WB
M
EX
IF/IDWrite
IF/ID
PCWrite
PC
Instruction memory
Instruction
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
Clock 4
Architettura degli elaboratori - I
Pagina 157


Pipeline (MIPS)
add $9, $4, $2
or $4, $4, $2
and $4, $2, $5
bubble
lw $2, . .
Hazard detection unit
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
Control
Mux
WB
M
EX
IF/IDWrite
IF/ID
PCWrite
PC
Instruction memory
Instruction
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
Clock 5
Architettura degli elaboratori - I
Pagina 158

i

Pipeline (MIPS)



Pipeline (MIPS)