Architetture dei Calcolatori

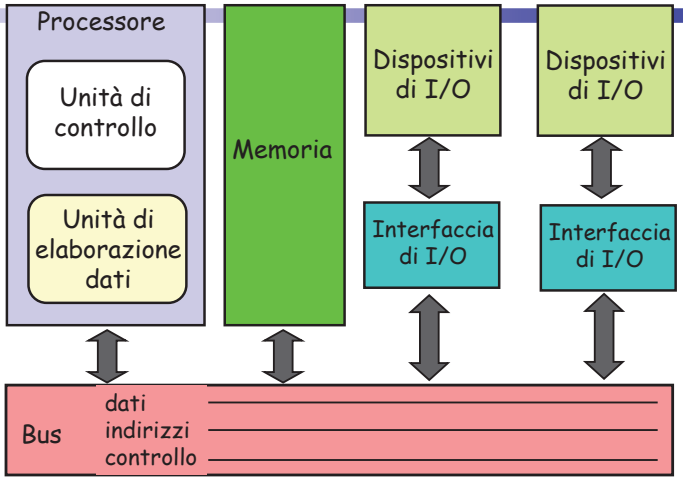
Il Processore

Prof. Francesco Lo Presti

Architettura a Livelli: Livelli 1 e 2

- Livello 2: Livello del Linguaggio Macchina (ISA)
  - Macchina nuda come appare al programmatore di sistema. Le istruzioni del suo linguaggio sono interpretate ed eseguite dai microprogrammi del processore
  - Moduli: Programmi
  - L2: Linguaggio macchina
  - R2: Registri, spazio di memoria
- Livello 1: Macchina Firmware – Microarchitettura
  - Interpreta ed esegue le istruzioni del linguaggio macchina
  - E' direttamente realizzato con i componenti della macchina hardware
  - Moduli - Unita' di Elaborazione: CPU, Memoria, Unita' di I/O
  - L1: Linguaggio di Microprogrammazione
  - R1: Reti combinatorie e sequenziali

Organizzazione di un Calcolatore



Processore - Central Processing Unit (CPU)

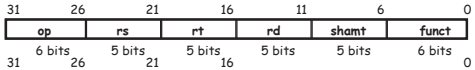
- Provvede all'esecuzione delle istruzioni macchina
- Ciclo di Esecuzione
  1. Prelievo Istruzione dalla Memoria
  2. Decodifica Istruzione
  3. Esecuzione Istruzione
- Processore e' composto da due sottosistemi:

1. Unità di Controllo (Control) – Parte di Controllo
   - Controlla il sequenziamento e l'esecuzione delle istruzioni generando i segnali di controllo
2. Unita' di Elaborazione Dati (Datapath) – Parte Operativa
   - Esegue le istruzioni
   - ALU
     - ✓ Esegue operazioni logico aritmetiche sui dati
   - Banco di Registri (Register File)
     - ✓ Memoria interna CPU
   - Program Counter (PC)
     - Indirizzo Prossima Istruzione
   - Instruction Register (IR)
     - Codice Istruzione da eseguire

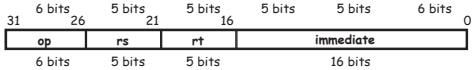
Processore - Central Processing Unit (CPU)

- Implementazione set ridotto del MIPS
  - Istruzioni di accesso alla memoria: lw e sw
  - Istruzioni logico-aritmetiche: add, sub, and, or e slt
  - Istruzioni di branch: beq
  - Istruzioni di jump: j

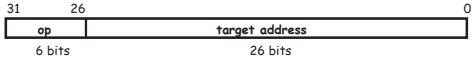
- Lunghezza formato: 32 bit; i tre formati delle istruzioni:
  - Tipo R



  - Tipo I



  - Tipo J



- I campi
  - *op*: operazione dell'istruzione
  - *rs*, *rt*, *rd*: registri sorgente (due) e destinazione
  - *shamt*: shift amount
  - *funct*: seleziona la variante dell'operazione nel campo op
  - *address/immediate*: offset dell'indirizzo o valore immediato
  - *target address*: indirizzo dell'istruzione di jump

Passi di Progetto

1. Analizzare il set di Istruzioni: Requisiti del Dapapath
   - Analizzare la semantica di ogni istruzione
     - ✓ Espressa intermini di trasferimenti e operazioni tra registri
   - Il datapath deve includere il banco dei registri (register file)
     - ✓ Sono necessari altri registri, non visibili a livello ISA, e.g., PC
   - Il datapath deve fornire i cammini per permettere tutti i trasferimenti tra registri necessari, e gli accessi in memoria
   - Includeremo la memoria nel progetto (per semplicita')
2. Selezionare i Componenti del Datapath
3. Assemblare il Datapath secondo i requisiti aggiungendo i segnali di controllo
4. Analizzare l' implementazione di ogni istruzione per determinare quali segnali di controllo devo essere affermato o meno per permetterne l' esecuzione
5. Realizzare la Parte di Controllo (Control) in accordo a quanto stabilito al punto 4

Semantica Istruzioni e RTL

- RTL (Register-Transfer Language): Linguaggio per esprimere i trasferimenti tra registri (e memoria),
  - Permette di definire la semantica di ogni istruzione
  - M[x] contenuto della memoria indirizzo x
  - R[y] contenuto registro y

- Es: `add rd, rs, rt`
  - `R[rd]=R[rs]+R[rt], PC=PC+4;`
- Es: `load rt, offset(rs)`
  - `R[rt]=M[R[rs]+sign_ext(offset)], PC=PC+4;`
- Es: `beq rs, rt, address`
  - `If(R[rs]==R[rt])`
    `then PC=PC+4+sign_ext(address)<<2;`
    `else PC=PC+4;`
- Tutte richiedono come passo preliminare il prelievo dell' istruzione dalle memoria (fetch)
  - `Istruzione=M[PC]`

Implementazione del set ridotto

- I primi due passi da implementare sono comuni a tutte le istruzioni,indipendentemente dalla classe di istruzione:
  - Inviare l'uscita del Program Counter (PC) alla memoria che contiene il programma, caricare l'istruzione (Fetch) ed aggiornare PC
  - Decodifica dell'Istruzione (Decode) (e lettura registri)
- La fase di esecuzione (Execute) dipende dall'istruzione
  - Sono comunque raggruppabili per classi di istruzioni
- Dopo aver letto i registri, tutte le istruzioni usano l'ALU (eccetto j)
  - Le istruzioni di accesso alla memoria per calcolare l'indirizzo
  - Le istruzioni logico-aritmetiche per effettuare l'operazione
  - L'istruzione `beq` per verificare l'uguaglianza fra registri

Implementazione del set ridotto (2)

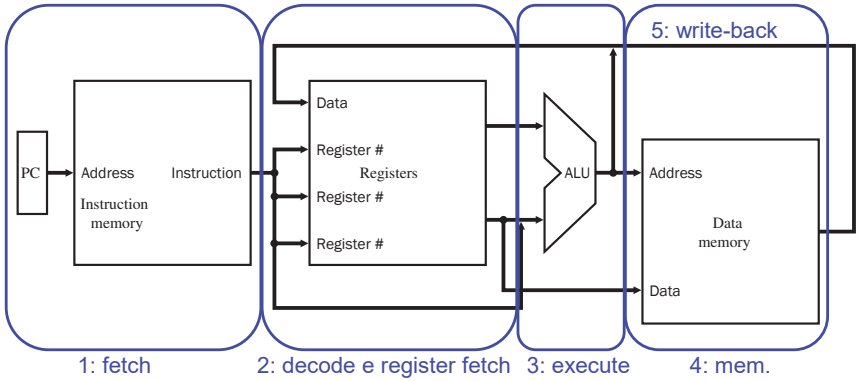
- Poi il comportamento delle istruzioni si differenzia
  - Istruzioni di accesso alla memoria
    - ✓ Devono accedere alla memoria per leggere/scrivere il dato
  - Istruzioni logico-aritmetiche
    - ✓ Devono accedere ad un registro per scrivere il risultato
  - Istruzioni di branch e jump
    - ✓ Devono modificare il Program Counter
- Vantaggio della semplicità nella progettazione
  - Pochi formati di istruzione facilitano l'implementazione dell'unità di elaborazione

I cinque passi delle istruzioni

- I cinque passi delle istruzioni da effettuare

1. Fetch (caricamento) dell'istruzione dalla memoria
2. Decode dell'istruzione e fetch dei registri
3. Execute
   - Uso della ALU (esecuzione dell'operazione o calcolo dell'indirizzo)
4. Memory Access
   - Accesso ad un operando in memoria
5. Write-back
   - Scrittura del risultato in un registro

Una visione astratta



Implementazione del Processore: Approcci

- Singolo Ciclo
  - Esecuzione di ogni istruzione richiede 1 ciclo di clock

  ⇒Il ciclo di clock deve essere abbastanza lungo da permettere il completamento dell' istruzione piu' lenta
  - Svantaggio: velocita' limitata dall' istruzione piu' lenta supportata, alcune risorse devono essere replicate
- Multi-Ciclo
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Vantaggio: ogni istruzione richiede il solo numero di cicli (tempo) richiesto
    - ✓ T_{clock}(Singolo Ciclo)> T_{clock} (Multiplo Ciclo)
- Pipelined
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Processare piu' istruzioni in parallelo
    - ✓ Elaborazione in contemporanea di step diversi di istruzioni consecutive (linea di assemblaggio)

Implementazione Singolo Ciclo

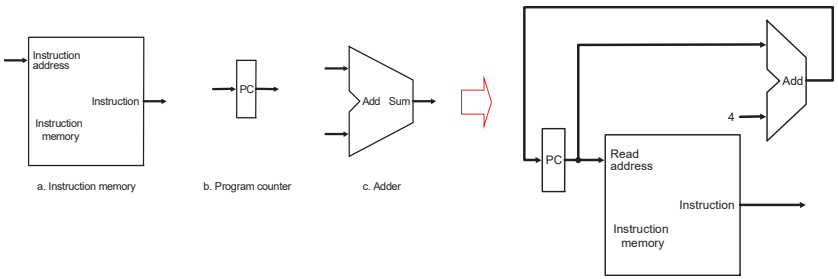
- Prima implementazione impiega in singolo ciclo di clock per ogni istruzione
  - Ogni istruzione inizia sul fronte attivo di un ciclo di clock e termina sul fronte attivo del ciclo di clock successivo
    - ✓ Approccio non pratico e inefficiente rispetto ad una implementazione multiciclo
  - Ogni istruzione richiede esattamente tanto tempo quanto il tempo di esecuzone dell' istruzione piu' lenta
    - ✓ Nell' implementazioni multiciclo questo problema si evita permettendo alle istruzioni piu' veloci di essere eseguite in un numero inferiore di cicli
- Tuttavia e' semplice e utile da un punto di vista didattico
- *Nota: Implementeremo il jump solo alla fine della trattazione*

I Blocchi della Progettazione

1. Fase di Fetch
   - Accedere all' istruzioni in memoria ed aggiornare PC
2. Decode, Accesso ai registri ed esecuzione istruzioni formato R
   - Istruzioni logico-aritmetiche
3. Decode, Accesso ai registri ed operazioni di accesso alla memoria
   - Istruzioni load/store
4. Decode, Accesso ai registri per i branch
   - Istruzione beq

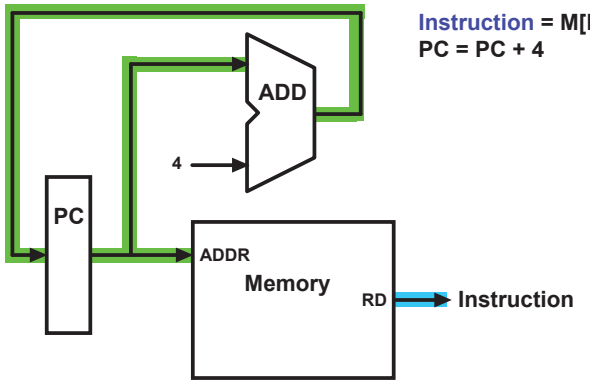
Datapath: Fetch Istruzione e aggiornamento PC

Instruction = M[PC]
PC = PC + 4



Datapath: Fetch Istruzione e aggiornamento PC

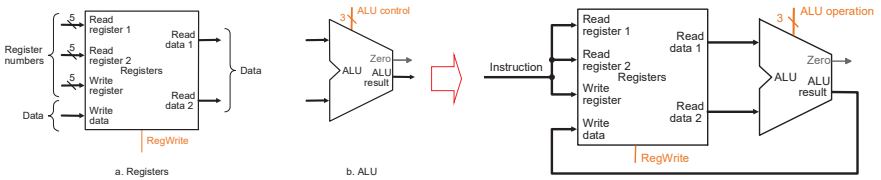
Instruction = M[PC]
PC = PC + 4



Datapath: Istruzioni formato R

```
add rd, rs, rt
```

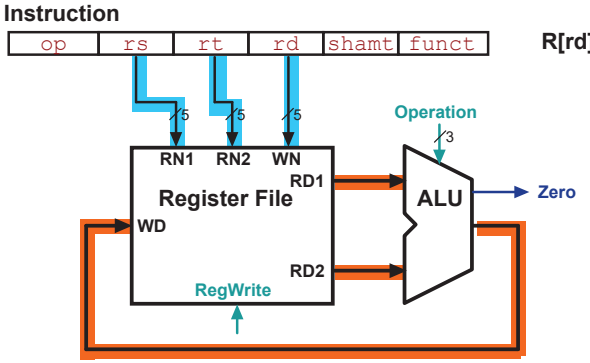
R[rd] =R[rs] + R[rt];



Datapath: Istruzioni formato R

```
add rd, rs, rt
```

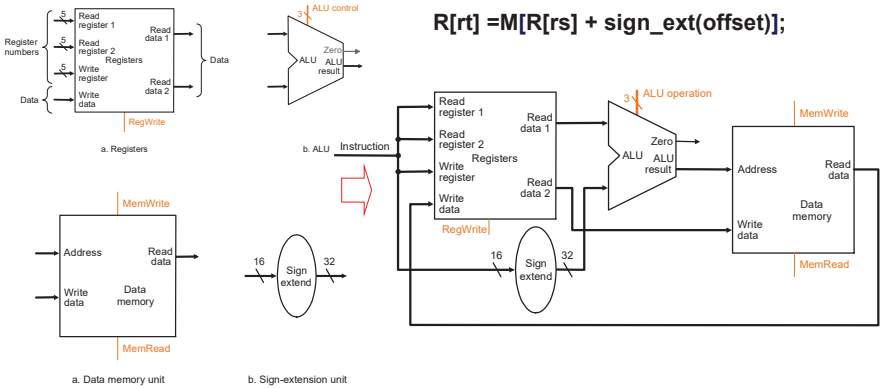
R[rd] =R[rs] + R[rt];



Datapath: Istruzioni Load/Store

```
lw rt, offset(rs)
```

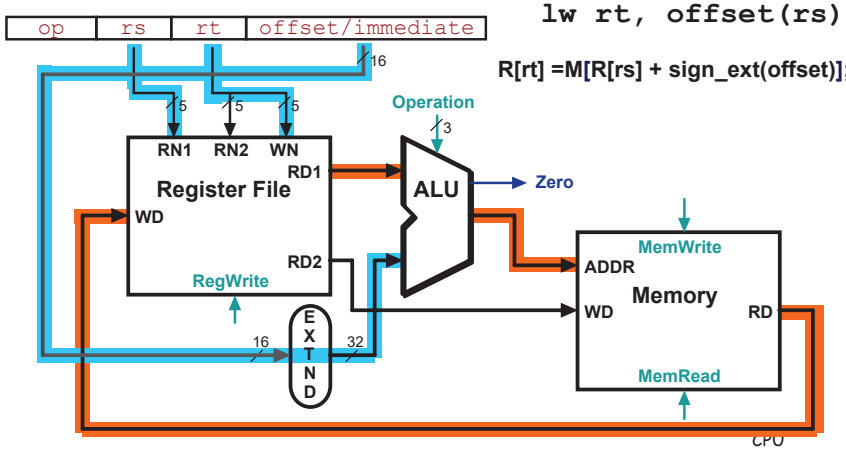
R[rt] =M[R[rs] + sign_ext(offset)];



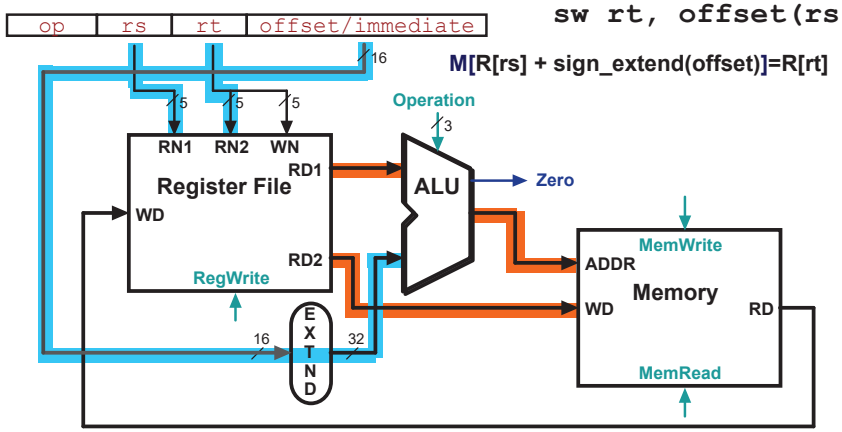
Datapath: Istruzione Load

```
lw rt, offset(rs)
```

R[rt] =M[R[rs] + sign_ext(offset)];



Datapath: Istruzione Store



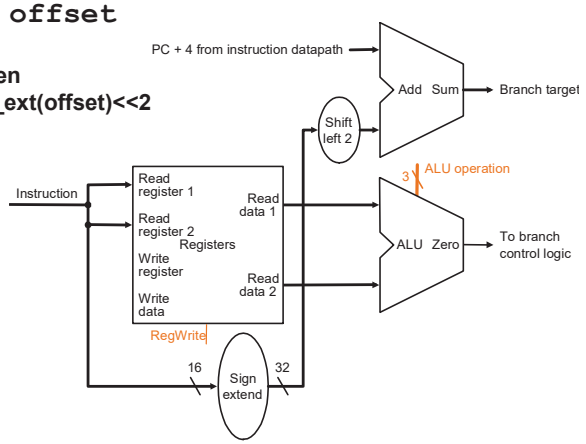
`sw rt, offset(rs)`

M[R[rs] + sign_extend(offset)]=R[rt]

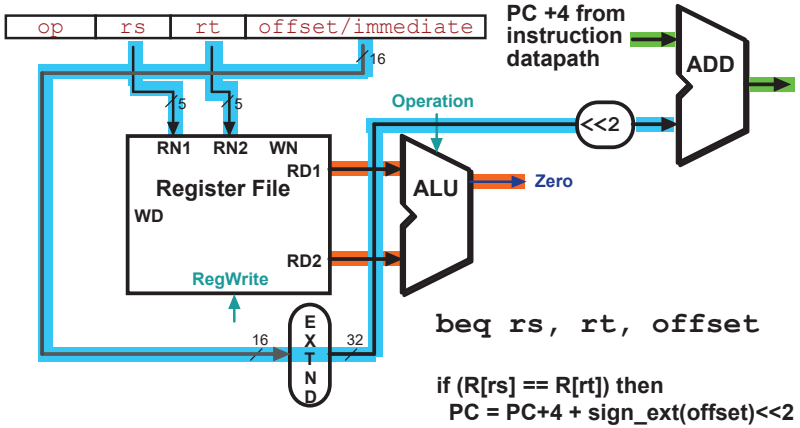
Datapath: Istruzione di Salto

`beq rs, rt, offset`

if (R[rs] == R[rt]) then
PC = PC+4 + sign_ext(offset)<<2



Datapath: Istruzione di Salto `beq`

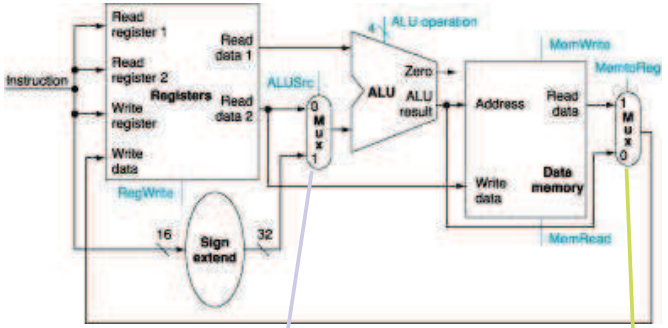


`beq rs, rt, offset`

if (R[rs] == R[rt]) then
PC = PC+4 + sign_ext(offset)<<2

Composizione dei blocchi

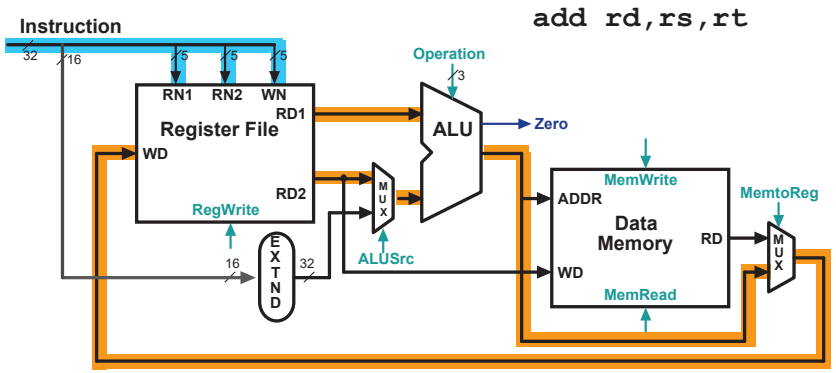
- Uniamo il blocco relativo alle istruzioni di accesso alla memoria con quello per le istruzioni di tipo R



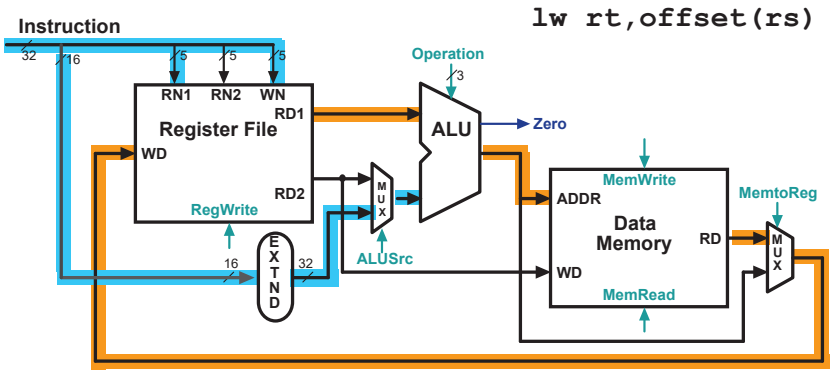
Multiplexer per scegliere se il secondo operando è un indirizzo (tipo I) oppure il dato in un registro (tipo R)

Multiplexer per scegliere se ai registri va il dato dalla memoria (tipo I) oppure il risultato dell'operazione (tipo R)

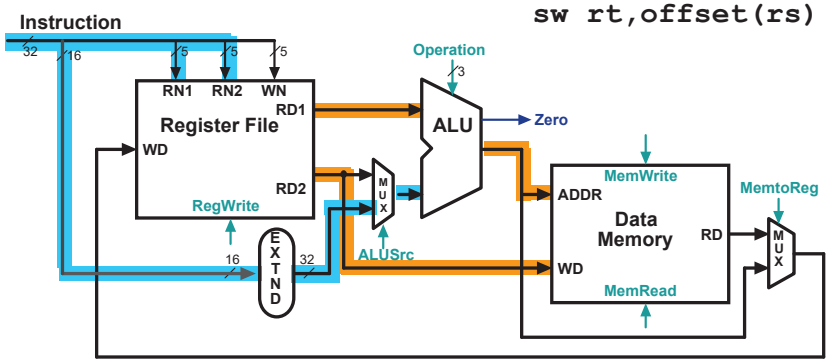
Datapath: Istruzioni formato R



Datapath: Istruzione Load

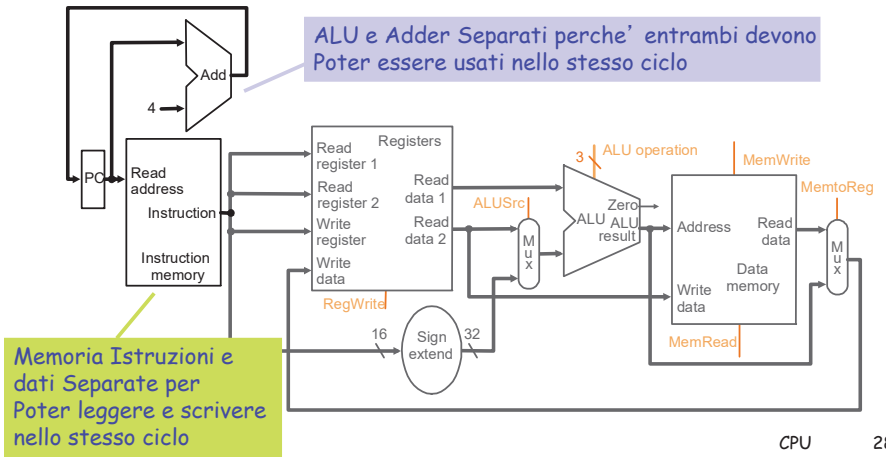


Datapath: Istruzione Store



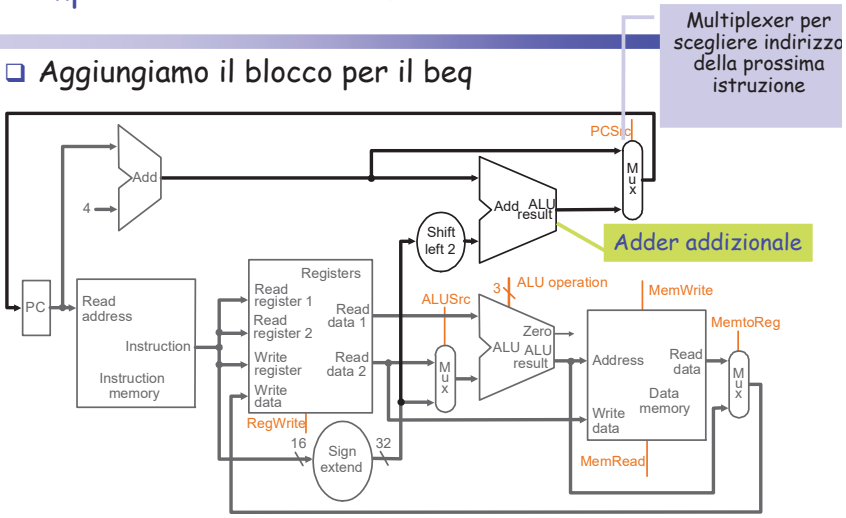
Composizione dei blocchi

- Aggiungere il blocco che esegue il fetch

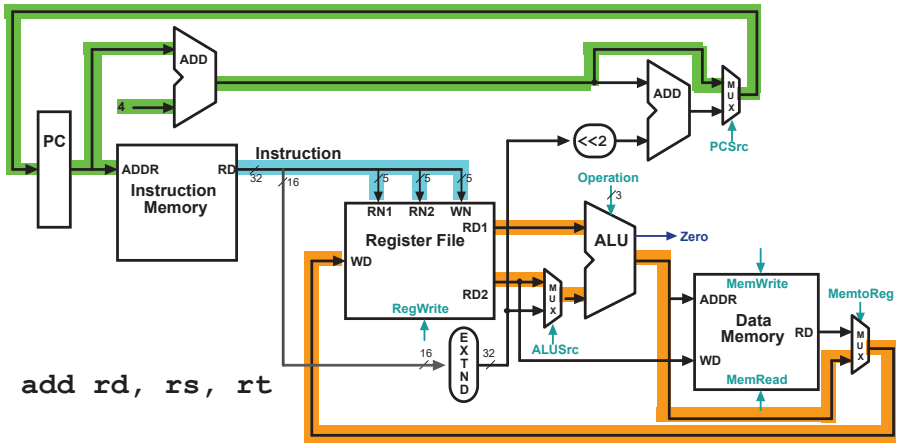


Composizione dei blocchi

- Aggiungiamo il blocco per il beq

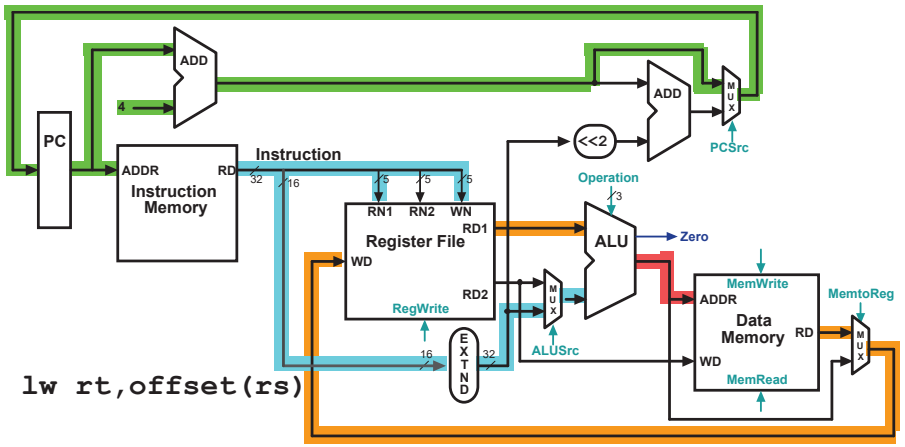


Datapath: `add`



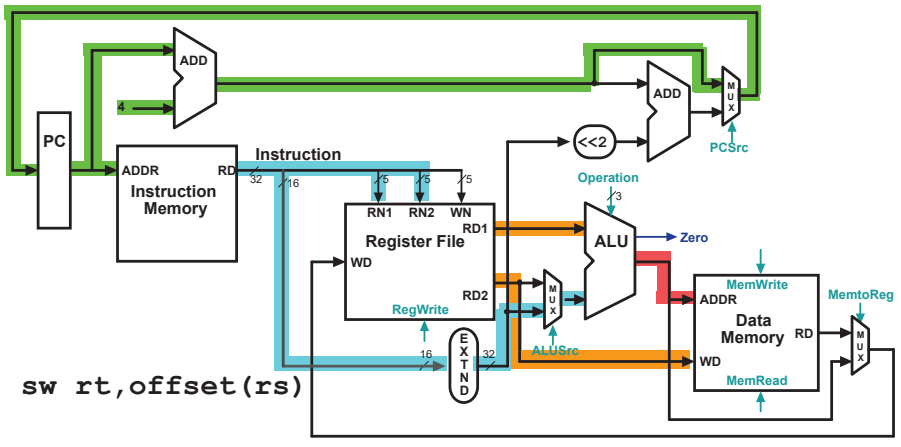

```
add rd, rs, rt
```


Datapath: `lw`



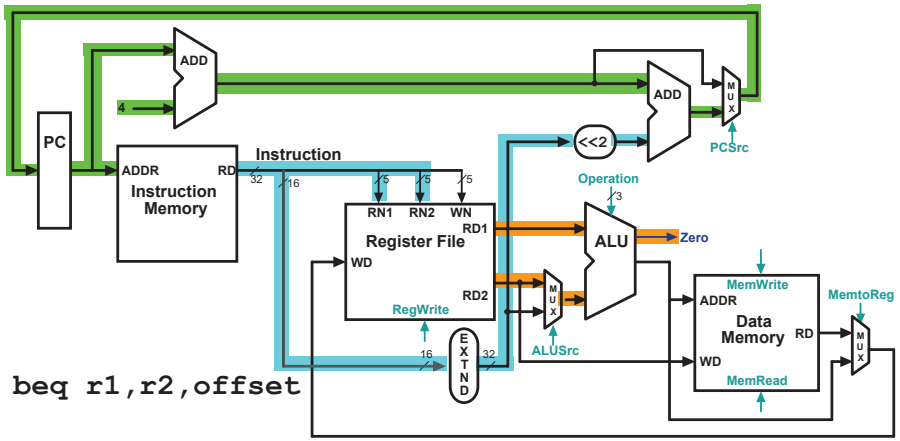

```
lw rt,offset(rs)
```


Datapath: `sw`




```
sw rt,offset(rs)
```

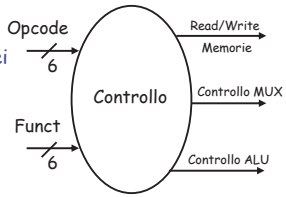
Datapath: beq

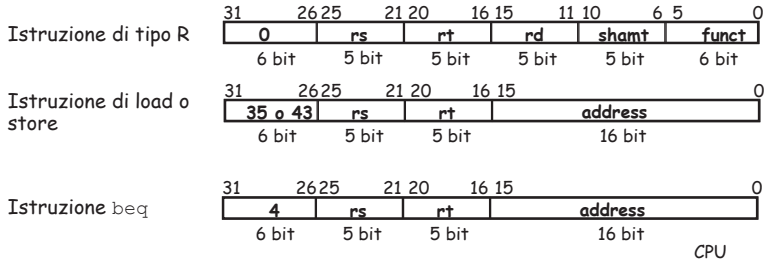


beq r1,r2,offset

Controllo

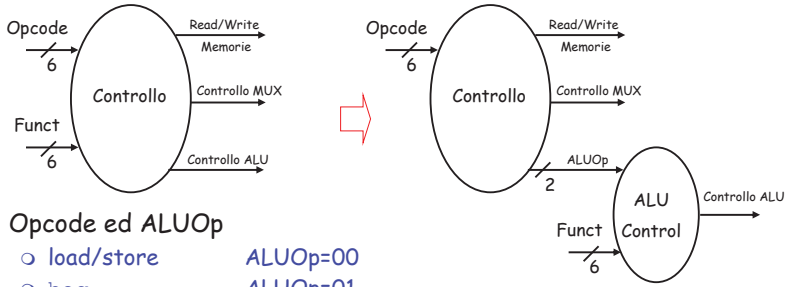
- L' Unita di Controllo deve generare:
  - I segnali di controllo dell' ALU
  - I segnali di controllo dei vari multiplexer
  - I segnali di abilitazione alla scrittura/lettura dei diversi elementi di memoria
- Basandosi su...
- I segnali in ingresso all' unita' di Controllo
  - I bit del campo opcode (e funct nel caso del formato R) dell' istruzione





Suddivisione in Main Control e ALU Control

- Per semplificare la progettazione/circuito
  - Bit di funct (formato R) servono solo per il controllo dell' ALU
    - ✓ Specificano l' operazione da eseguire: `add`, `sub`, `and`, `or`, o `slt`
- ALUOp serve per specificare il tipo di istruzione `lw/sw` `beq` o formato R)

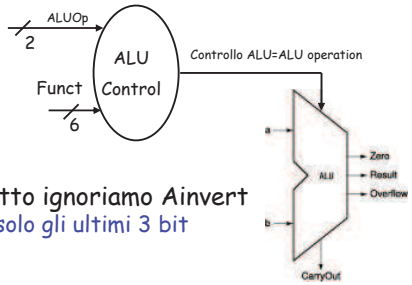


- Opcode ed ALUOp
  - load/store ALUOp=00
  - `beq` ALUOp=01
  - Formato R ALUOp=10

Controllo dell' ALU

- Istruzioni ed Operazioni Richieste
  - Load/store ALUOp=00 somma
  - `beq` ALUOp=01 sottrazione
  - Formato R ALUOp=10 spec. dal campo funct
- Linee di controllo Ainvert (1 bit), Bnegate (1 bit) e Operation (2 bit) dell'ALU

```
0000 = and
0001 = or
0010 = add
0110 = sub
0111 = slt
1100 = nor
```



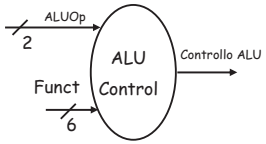
- Poiche' manca nor nel set ristetto ignoriamo Ainvert
  - Il controllo dell' ALU richiede solo gli ultimi 3 bit

Controllo dell' ALU (2)

Instruction opcode	AluOp	Instruction operation	Funct Field	Desired ALU action	ALU control
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch eq	01	branch eq	xxxxxx	sub	110
R-type	10	add	100000	add	010
R-type	10	sub	100010	sub	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less	101010	set on less	111

Tabella di Verita' dell' ALU Control

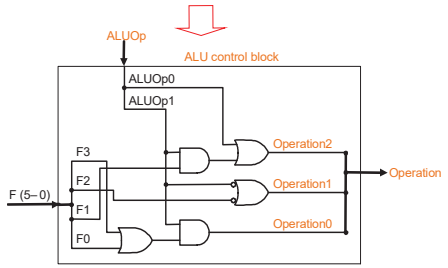
Input: ALUOp		Input: Funct field						Output: ALU Control
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0 *	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111



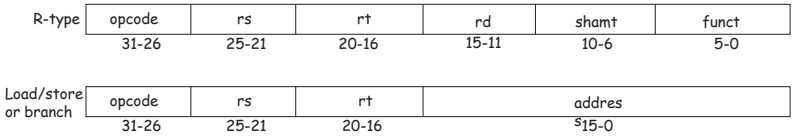
Controllo dell' ALU: Implementazione

Tabella di Verita' dell' ALU Control

Input: ALUOp		Input: Funct field						Output: ALU Control
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0 *	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

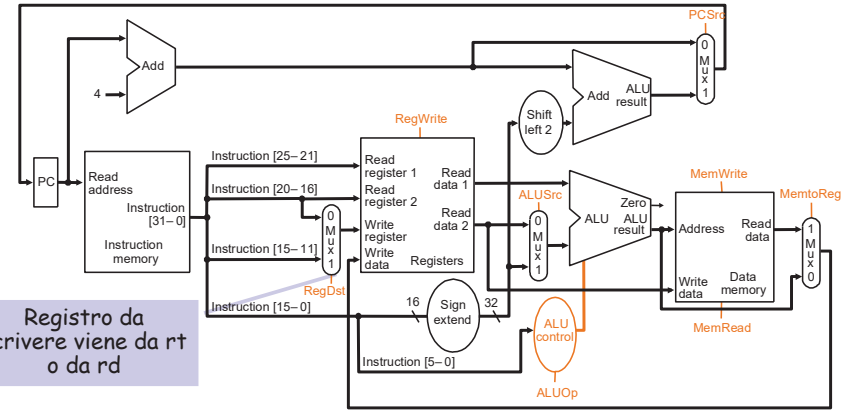


Progettazione dell' Unita' di Controllo



- Osservazioni sul formato Istruzioni MIPS
  - Il codice operativo (campo opcode) e' sempre nei bit 31-26
  - I due registri da leggere sono sempre rs (bit 25-21) e rt (bit 20-16)
  - Il registro base per load e store e' sempre il registro rs (bit 25-21)
  - L' offset a 16-bit per `beq`,load e store e' sempre nei bit 15-0
  - Il registro di destinazione e'
    - ✓ bit 20-16 (rt) per load
    - ✓ bits 15-11 (rd) per le istruzioni R
    - ⇒Occorre un ulteriore multiplexer per indicare quale campo dell' istruzione indica il registro destinazione

Datapath e Controllo I

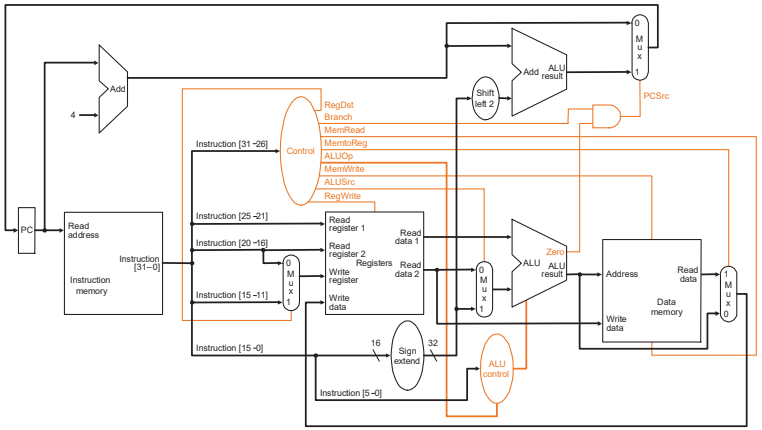


Registro da scrivere viene da rt o da rd

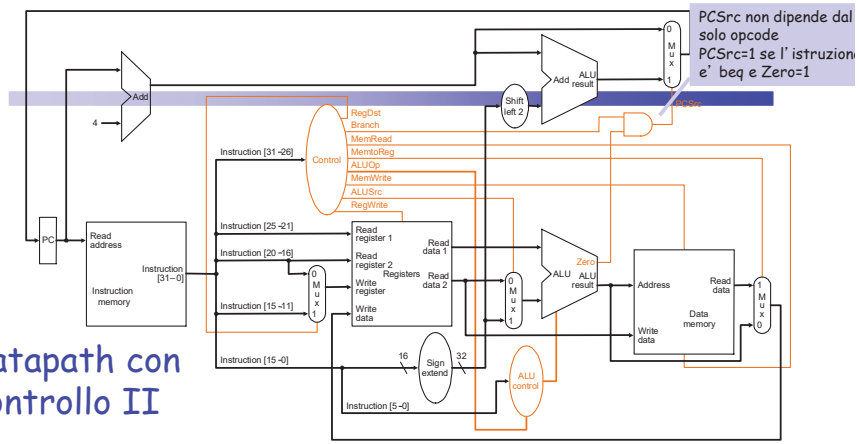
I segnali di controllo ad 1 bit

Segnale	Effetto quando vale 0	Effetto quando vale 1
RegDst	Registro destinazione = rt	Registro destinazione = rd
RegWrite	Nessuno	Nel registro indicato sull'ingresso *Write register* viene scritto il valore *Write data*
ALUSrc	Il secondo operando di ALU viene da *Read data 2*	Il secondo operando di ALU viene dall'estensione di segno
PCSrc	Scrittura di PC con PC+4	Scrittura di PC con l'output del sommatore per il branch
MemRead	Nessuno	Lettura della locazione di memoria indicata da *Address*
MemWrite	Nessuno	Scrittura della locazione di memoria indicata da *Address*
MemtoReg	Il valore in *Write data* (registri) viene dalla ALU	Il valore in *Write data* (registri) viene dalla memoria dati

Datapath e Controllo II



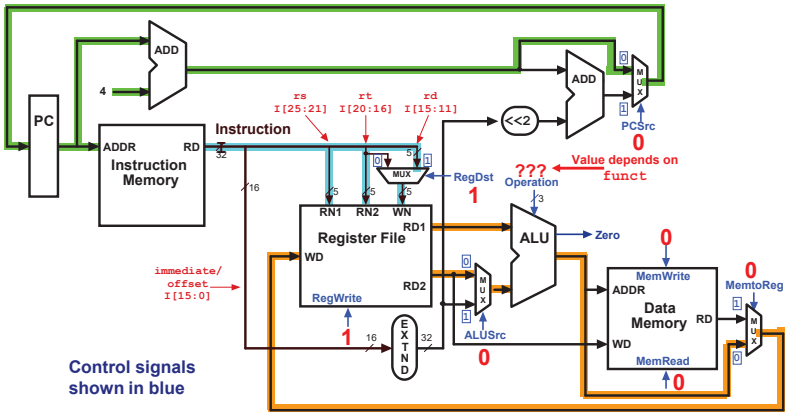
Datapath con Controllo II



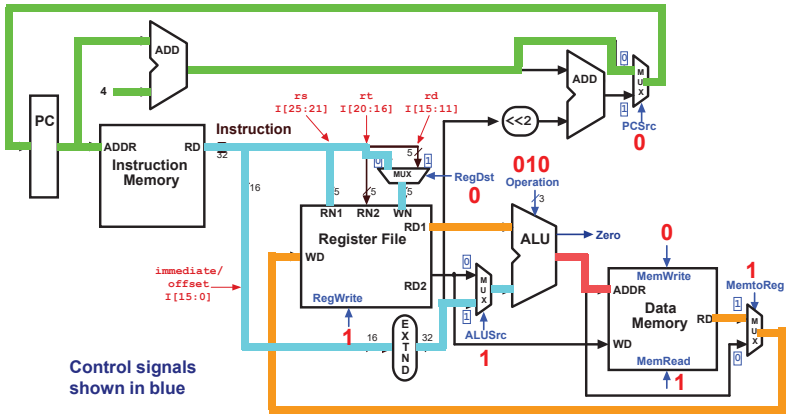
Opcode Istruzione e Segnali di Controllo

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
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

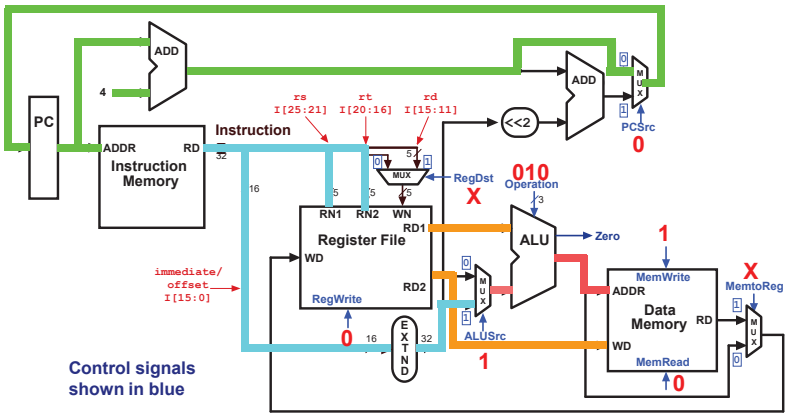
Segnali di Controllo: Istruzioni Formato R



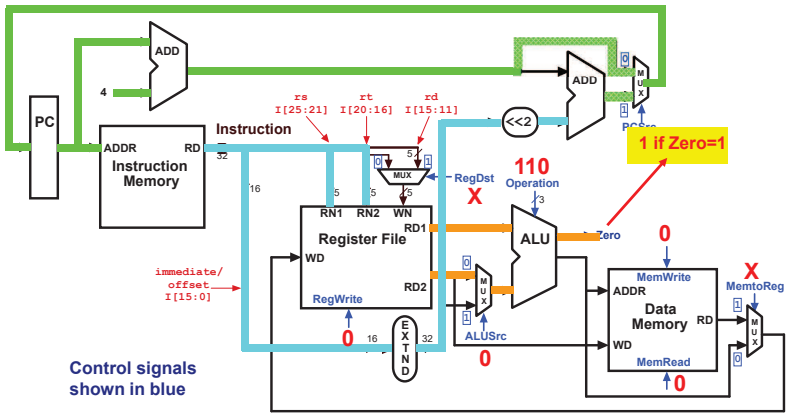
Segnali di Controllo: Istruzione lw



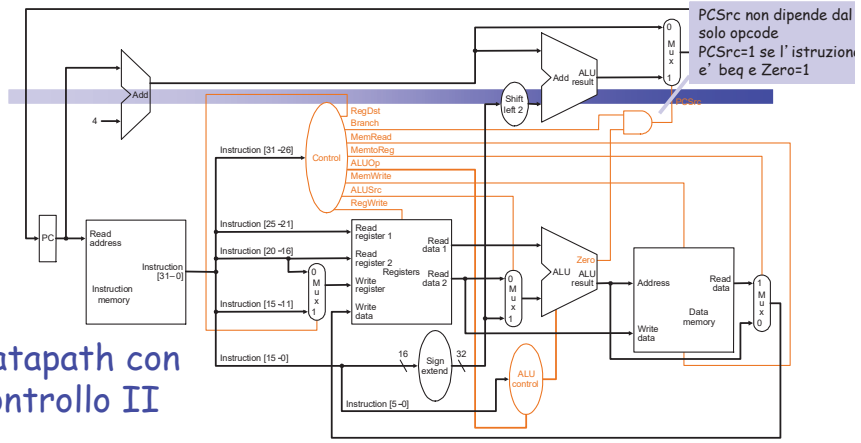
Segnali di Controllo: Istruzione sw



Segnali di Controllo: Istruzione beq



Datapath con Controllo II



Opcode Istruzione e Segnali di Controllo

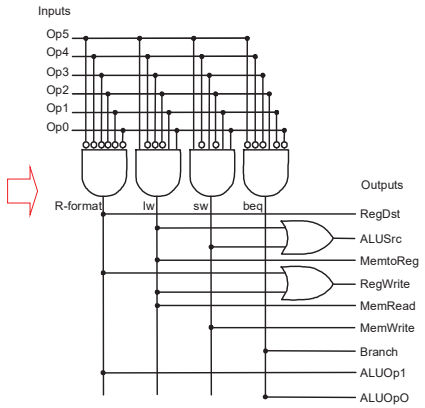
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Implementazione dell' unita' di Controllo Principale

Tabella di verita' dell' unita' di Controllo

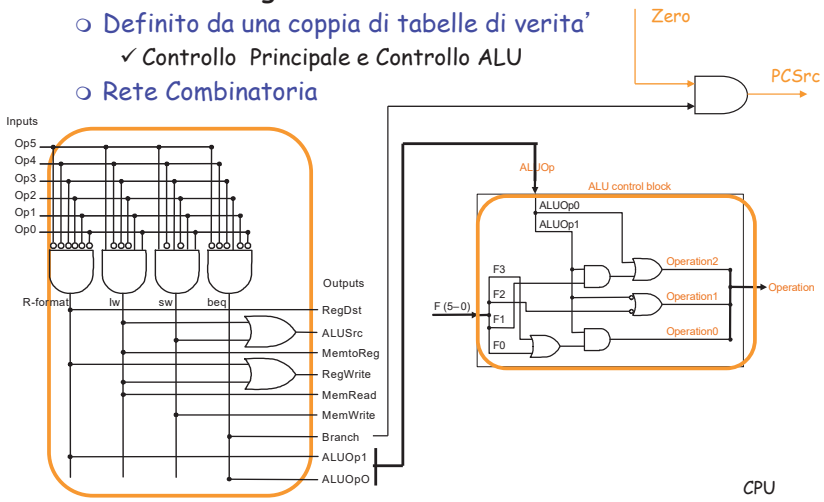
	Segnale	form. R	lw	sw	beq
Inputs =Opcode	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOP0	0	0	0	1

Rete Combinatoria realizzabile tramite PLA



Controllo a Singolo Ciclo

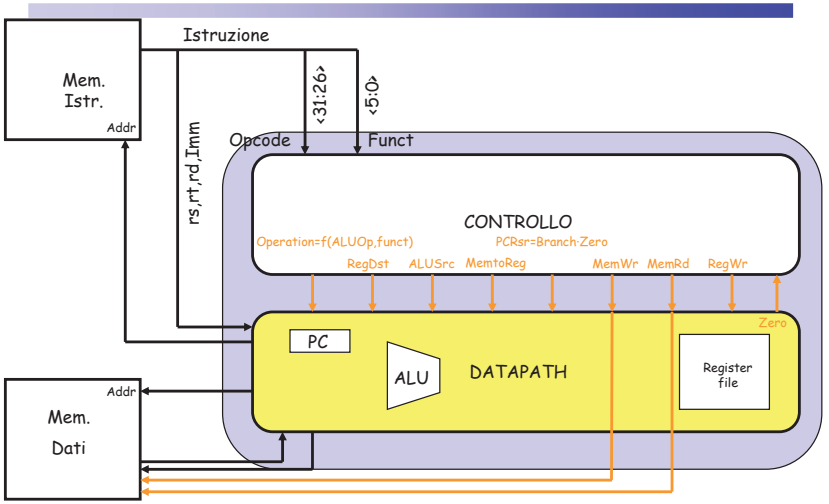
- Controllo a singolo ciclo
  - Definito da una coppia di tabelle di verita'
    - ✓ Controllo Principale e Controllo ALU
  - Rete Combinatoria



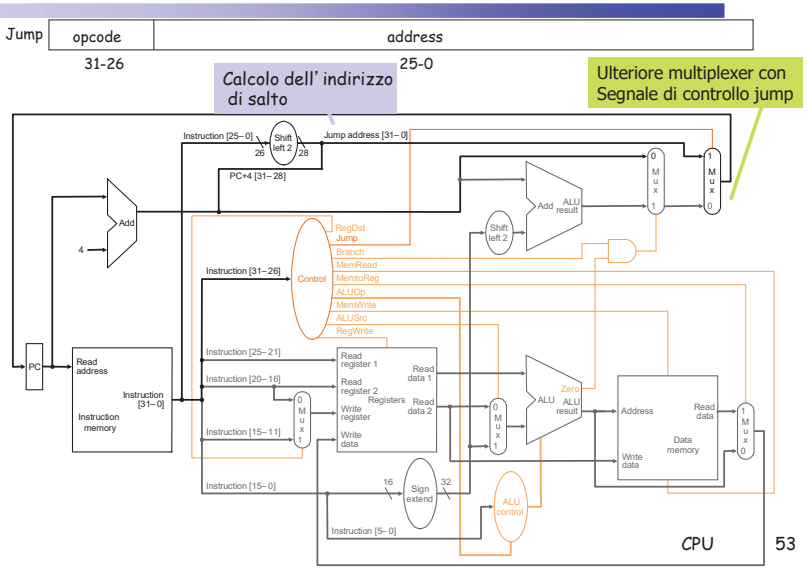
Controllo a Singolo Ciclo

- Il Controllo della CPU a singolo ciclo e' una rete combinatoria
- Il Datapath e' una rete sequenziale
  - L' output dipende dagli ingressi e dai valori dagli elementi di memoria (Registri e PC)
  - Il ciclo di clock deve durare abbastanza da stabilizzare le uscite di tutte le reti combinatorie prima del fronte di salita del clock
  - Clock in AND con i segnali di controllo di scrittura
    - ✓ I valori in ingresso vengono scritti solo se i segnali sono affermati
  - Ciclo di Clock determinato sulla base del "percorso" piu' lungo

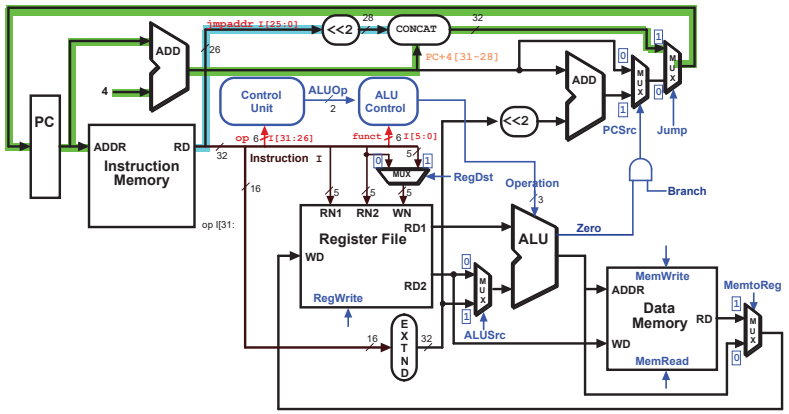
Diagramma a blocchi della CPU (Datapath e Control) e Memoria



Datapath e Controllo III: Istruzione j



Datapath e Controllo III: Istruzione j



Ciclo di Clock dell' Implementazione a singolo ciclo

- L' implementazione singolo ciclo e' inefficiente
  - Una istruzione per ciclo, ma…
- Il tempo di ciclo e' determinato dall' istruzione piu' lunga
- Quale?

Ciclo di Clock dell' Implementazione a singolo ciclo

- Calcolo del tempo di ciclo assumendo ritardi nulli per multiplexer, unita' di controllo, estensione del segno, accesso PC, shift left, linee, eccetto :
  - Memoria Istruzione e Dati (2ns)
  - ALU ed addizionatori (2ns)
  - Accesso al banco dei registri (1ns)

Instr.	Mem I	Reg Rd	ALU Op	Mem D	Reg Wr	Total
R-type	2	1	2		1	6
load	2	1	2	2	1	8
store	2	1	2	2		7
beq	2	1	2			5
jump	2					2

Problemi con l' Implementazione a Singolo Ciclo

- Il tempo di ciclo e' determinato dall' istruzione piu' lenta
  - Nel nostro caso e' load, ma cosa succederebbe se considerassimo anche istruzioni floating point?
  - Perdita di tempo
    - ✓ molte istruzioni possono essere eseguite in un tempo minore

Implementazione del Processore: Approcci

- Singolo Ciclo
  - Esecuzione di ogni istruzione richiede 1 ciclo di clock
  - ⇒Il ciclo di clock deve essere abbastanza lungo da permettere il completamento dell' istruzione piu' lenta
  - Svantaggio: velocita' limitata dall' istruzione piu' lenta supportata, alcune risorse devono essere replicate
- Multi-Ciclo
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Vantaggio: ogni istruzione richiede il solo numero di cicli (tempo) richiesto
    - ✓ T_{clock}(Singolo Ciclo)> T_{clock} (Multiplo Ciclo)
- Pipelined
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Processare piu' istruzioni in parallelo
    - ✓ Elaborazione in contemporanea di step diversi di istruzioni consecutive (linea di assemblaggio)

Implementazione del Processore: Approcci

- Singolo Ciclo
  - Esecuzione di ogni istruzione richiede 1 ciclo di clock
  - ⇒Il ciclo di clock deve essere abbastanza lungo da permettere il completamento dell' istruzione piu' lenta
  - Svantaggio: velocita' limitata dall' istruzione piu' lenta supportata, alcune risorse devono essere replicate
- Multi-Ciclo
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Vantaggio: ogni istruzione richiede il solo numero di cicli (tempo) richiesto
    - ✓ T_{clock}(Singolo Ciclo)> T_{clock} (Multiplo Ciclo)
- Pipelined
  - Suddividere l' esecuzione in piu' passi
  - Eseguire un passo per ciclo
  - Processare piu' istruzioni in parallelo
    - ✓ Elaborazione in contemporanea di step diversi di istruzioni consecutive (linea di assemblaggio)