Lezione 20

ISA (Instruction Set Architecture) della CPU MIPS32

Prof. Federico Pedersini
Dipartimento di Informatica
Università degli Studi di Milano

Linguaggio macchina e architettura: ISA

- Compito di un elaboratore: **eseguire ciclicamente istruzioni**
- Le istruzioni sono rappresentate in un linguaggio direttamente comprensibile dalla macchina: il ***linguaggio MACCHINA***
- Ogni architettura di processore è in grado di eseguire un insieme di istruzioni nel proprio linguaggio macchina: ***"Instruction Set"***
- **Un'architettura** è **completamente definita** dal proprio **Instruction Set**
  - Due processori hanno la stessa architettura se hanno lo stesso linguaggio macchina, anche se le implementazioni hardware fossero diverse

ISA (Instruction Set Architecture):
l'insieme delle istruzioni-macchina eseguibili
da una architettura di processore

- La **ISA** definisce **un'architettura di elaboratore** in modo **equivalente** al suo **schema circuitale.**

Linguaggio Assembly

ASSEMBLY: rappresentazione **simbolica** del **linguaggio macchina**

- Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit
- Utilizzabile come linguaggio di programmazione vero e proprio

- **VANTAGGI:** visibilità diretta dell'hardware
  - → Massimo sfruttamento delle potenzialità HW della macchina
  - → Ottimizzazione delle prestazioni

- **SVANTAGGI:** comunque meno comprensibile di linguaggi ad alto livello

  ...e mancanza di portabilità dei programmi (ogni CPU ha il proprio Assembly)

- **LIMITI** di Assembly vs. linguaggi di alto livello:
  - Le strutture di controllo hanno forme limitate
  - Pochi tipi di dati a disposizione: **interi, virgola mobile, caratteri**
  - Gestione delle strutture dati e delle chiamate a procedura deve essere fatta in modo esplicito dal programmatore

Esempio: linguaggio C, Assembly, macchina (MIPS)

C:

```
main()
{
   int i;
   int sum = 0;

   for (i = 0; i <= 100; i = i + 1)
               sum = sum + i*i;
}
```

Linguaggio macchina (MIPS)

```
00:     00100111101111011111111111100000
04:     10101111101111110000000000010100
08:     10101111101001000000000000100000
0C:     10101111101001010000000000100100
10:     10101111101001010000000000100100
14:     10101111101001010000000000011000
18:     ........................
```

Assembly (MIPS):

```
main:
    subu $sp, $sp, 32
    sw $ra, 20($sp)
    sw $a0, 32 ($sp)
    sw $0, 24($sp)
    sw $0, 28($sp)
loop:
    lw  $t6, 28($sp)
    lw $t8, 24($sp)
    mult $t4, $t6, $t6
    addu $t9, $t8, $t4
    addu $t9, $t8, $t7
    sw  $t9, 24($sp)
    addu $t7, $t6, 1
    sw $t7, 28($sp)
    bne $t5, 100, loop
```

Classificazione delle istruzioni

Classificazione duplice delle istruzioni macchina

1. **Categoria:**
   definizione della funzione operativa dell'istruzione
   - logico-aritmetica
   - trasferimento dati (memoria, I/O)
   - controllo di flusso (salto)

2. **Formato (tipo):** definizione di come un'istruzione macchina viene codificata e organizzata in una parola binaria
   - Tipo e dimensione istruzione
   - Posizione operandi e risultato
   - Tipo e dimensione dei dati
   - Operazioni consentite

Classificazione "ortogonale" del Set Istruzioni:

Formato (tipo) \ Categoria/funzione				
	$\texttt{add}_1$	$\texttt{load}_1$	...	$\texttt{jmp}_1$
	$\texttt{add}_1$	$\texttt{load}_1$		--
	...	...		...
	$\texttt{add}_N$	--		$\texttt{jmp}_N$

L'insieme delle istruzioni (ISA)

Le istruzioni del linguaggio macchina di ogni calcolatore possono essere classificate in base:

- alla loro **funzione (Categoria)**
- al loro **formato (Tipo)**

Categorie MIPS:

1. Istruzioni **aritmetico-logiche**;
2. Istruzioni di **trasferimento da/verso la memoria**;
3. Istruzioni di **salto** condizionato e non condizionato per il controllo del flusso di programma;
4. Istruzioni di **trasferimento in ingresso/uscita** (I/O).
5. Istruzioni di **controllo**.

Formato delle istruzioni MIPS

Formato del **Set Istruzioni** di **MIPS:**

1. Tutte le istruzioni MIPS hanno la **stessa dimensione: 32 bit**

- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione

2. La **categoria** di istruzione è riconosciuto in base al valore dei **6 bit più significativi:**

(6 bit: codice operativo - "*OPCODE*")

3. Le istruzioni MIPS sono di **3 formati ("tipi")**

- Tipo **R (register)** Istruzioni che operano su registri
- Tipo **I (immediate)** Istruzioni in cui un operando è "immediato", cioè è contenuto nell'istruzione stessa
- Tipo **J (jump)** Istruzioni di salto

I tipi di istruzione MIPS

- Categorie di istruzioni MIPS
  - Istruzioni aritmetico-logiche
  - Istruzioni di trasferimento dati
  - Istruzioni di salto

Istruzioni aritmetico-logiche

Istruzioni aritmetico-logiche

In MIPS un'istruzione aritmetico-logica possiede **tre** operandi:

- **due registri** contenenti i valori da elaborare (**2 registri sorgente**)
  oppure **1 registro** (1º operando) ed un **numero** (2º operando)
- **un registro** che conterrà il risultato (**registro destinazione**)

- L'ordine degli operandi è **fisso:**
  prima il registro destinazione, poi i due registri sorgente, in ordine

- SINTASSI di un'istruzione aritmetico/logica in Assembly:
  - codice operativo e **tre campi** relativi ai tre operandi:

    DESTINAZIONE → OPERANDO 1 → OPERANDO 2

```
OPERAZIONE  DEST, SORG1, SORG2
```


Istruzioni: add/sub, tipo R

Istruzioni aritmetico-logiche in MIPS:

- **`add`**: addizione

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

  - somma il contenuto di due registri sorgente **`rs`** e **`rt`**
  - e mette la somma nel registro destinazione: **`rd`**

```
add rd, rs, rt  # rd ← rs + rt
```

- **`sub`**: sottrazione

```
sub rd rs rt
```

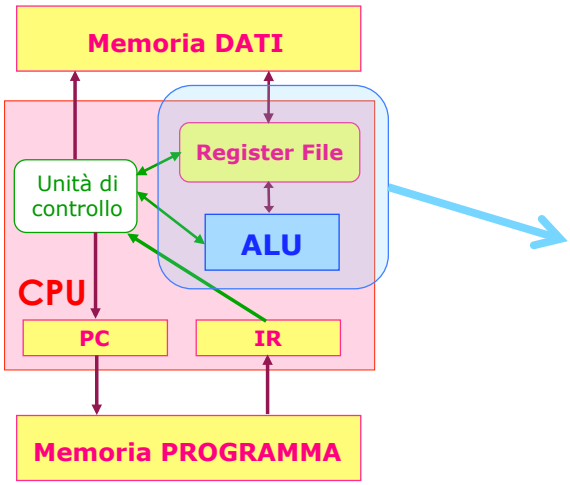
  - sottrae il contenuto di due registri sorgente **`rs`** e **`rt`**
  - e mette la differenza nel registro destinazione **`rd`**

```
sub rd, rs, rt  # rd ← rs - rt
```

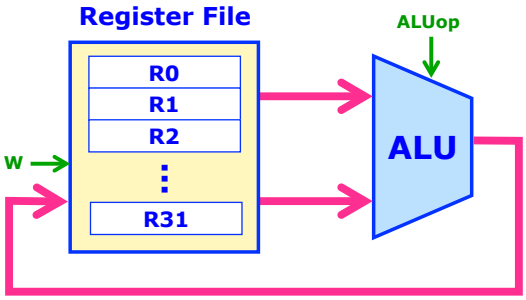
Istruzioni: add/sub, tipo R

add/sub, tipo R: Configurazione circuitale (***datapath***):

- **2 porte di lettura RF → 2 porte di ingresso ALU**
- **ALU Out → porta di scrittura RF**



CPU datapath: A/L tipo R



Varianti: unsigned, tipo I

add/sub – variante "unsigned":

```
addu rd, rs, rt    #add unsigned
subu rd, rs, rt    #sub unsigned
```

- L'operazione viene eseguita tra numeri senza segno

Istruzioni a/l MIPS con operando immediato (tipo I):

```
addi rd, rs, IMM        #add immediate
subi rd, rs, IMM        #sub immediate
```

- Somma/sottrazione di una costante: il valore del secondo operando è presente nell'istruzione come costante (ultimi 16 bit dell'istruzione)

addi/subi – variante "unsigned":

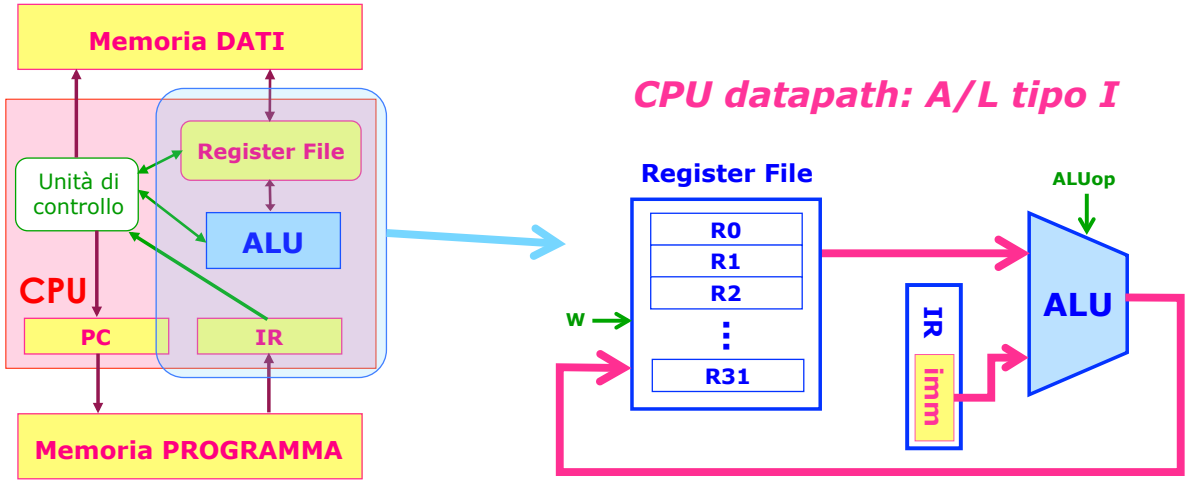
```
addiu rd, rs, IMM    #add immediate unsigned
subiu rd, rs, IMM    #sub immediate unsigned
```

- Il secondo operando è una costante, senza segno

Istruzioni: add/sub, tipo I

Configurazione circuitale (datapath): **add/sub, tipo I**

porta di lettura RF → I porta ingresso ALU
IMMEDIATO IR → II porta ingresso ALU
ALU OUT → porta scrittura RF



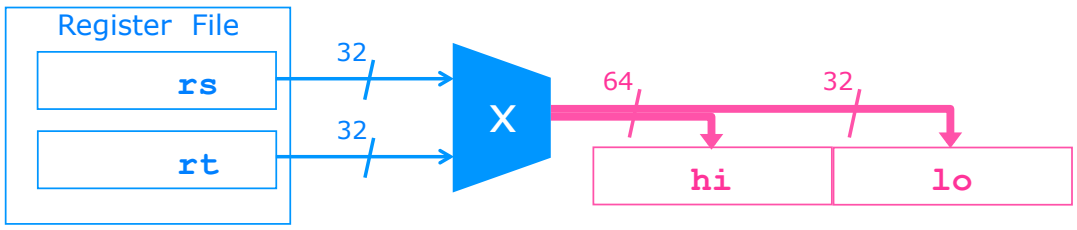
Moltiplicazione

MOLTIPLICAZIONE in Assembly MIPS:

```
mult rs rt          # multiply
multu rs rt         # unsigned multiply
```

- La moltiplicazione di due numeri di **32 bit** dà come risultato un numero rappresentabile in **64 bit**
- Il registro destinazione è ***implicito***: il risultato viene posto sempre in due registri dedicati: **[hi , lo]**

HIGH-order 32-bit word → hi
LOW-order 32-bit word → lo



Moltiplicazione / Divisione

- Il risultato di moltiplicazione / divisione si preleva dal registro **hi** e dal registro **lo** utilizzando le due istruzioni:

```
mfhi rd     # move from hi: rd ← hi
mflo rd     # move from lo: rd ← lo
```

- Il risultato viene trasferito nel registro destinazione specificato

Divisione

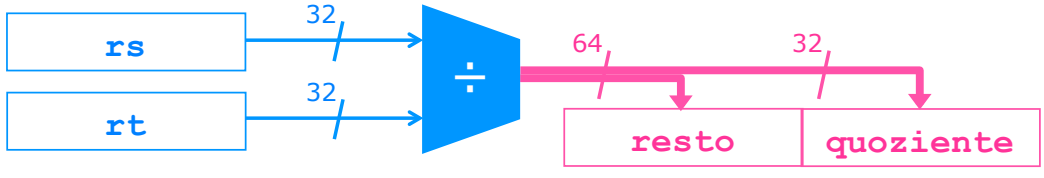
Divisione in Assembly MIPS32:

```
div rs rt   # division: rs/rt
divu rs rt  # unsigned division
```

- Come nella moltiplicazione, anche nella divisione, il registro destinazione è **implicito**, di **dimensione doppia**: [hi , lo]
  - Necessari 64 bit: quoziente (32 bit) + resto (32 bit)
  - Quoziente (intero) della divisione nel registro **lo**
  - Resto della divisione nel registro **hi**

QUOZIENTE (32-bit) → lo

RESTO (32-bit) → hi



I tipi di istruzione

- Categorie di istruzione:
  - Istruzioni aritmetico-logiche
  - Istruzioni di trasferimento dati
  - Istruzioni di salto

Istruzioni di trasferimento dati

MIPS32 fornisce due operazioni base per il trasferimento dei dati:

- **`lw` (load word)**
  per trasferire una parola di memoria in un registro
- **`sw` (store word)**
  per trasferire il contenuto di un registro in una cella di memoria

`lw` e **`sw`** necessitano, come argomenti:

- dell'**indirizzo** della cella **di memoria** su cui operare
- del **registro** in cui scrivere (**`lw`**) / da cui leggere (**`sw`**) il dato

Istruzione MIPS: **lw** (load word)

- In **MIPS32** le istruzioni **`lw/sw`** **hanno 3 argomenti:**
  - il ***registro destinazione*** in cui caricare la parola letta dalla memoria
  - una costante di ***spiazzamento (offset)***
  - un registro base (***base register***) che contiene il valore dell'indirizzo base ***(base address)*** da sommare alla costante.

```
lw   reg_dest, offset( reg_base )
sw   reg_src,  offset( reg_base )
```

- L'indirizzo della parola di memoria da caricare nel registro destinazione è ottenuto come:
  somma dell' **OFFSET** e del **contenuto del Registro Base**.

l'indirizzo è espresso in bytes !!!

lw/sw: *trasferimenti memoria – registro*

```
lw $10, 100($2)      # $10 ← M[$2+100]
```

- Al registro destinazione `$10` è assegnato il valore contenuto all'indirizzo di memoria **A**:

A = $2 + 100 (A: indirizzo di byte)

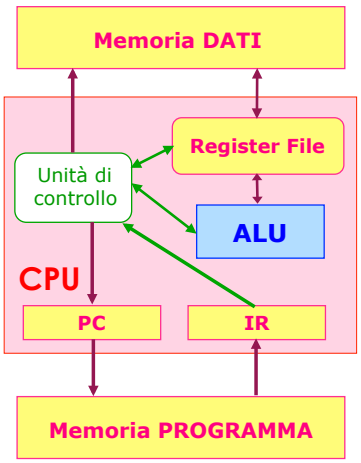
```
sw $10, 100($2)      # M[$2 + 100] ← $10
```

- Alla locazione di memoria di indirizzo `($2 + 100)` è assegnato il valore contenuto nel registro `$10`

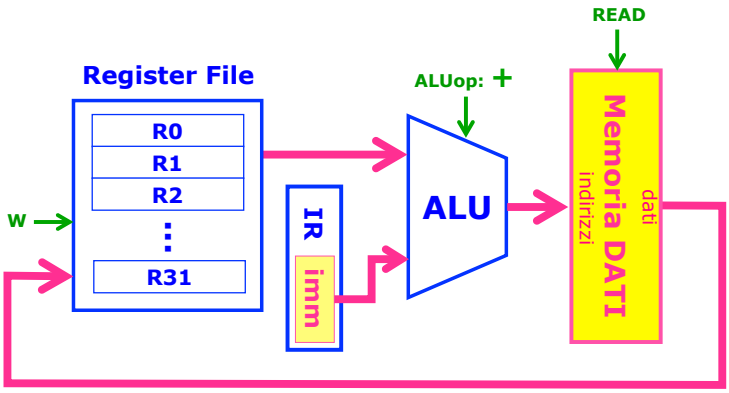
Istruzioni: lw/sw

Configurazione circuitale (datapath): **lw/sw (tipo I)**

- **porta di lettura RF** → **I porta ingresso ALU**
- **IMMEDIATO IR** → **II porta ingresso ALU**
- **ALU OUT** → **porta INDIRIZZI Memoria dati**
- **DATI Memoria dati** → **porta scrittura RF**



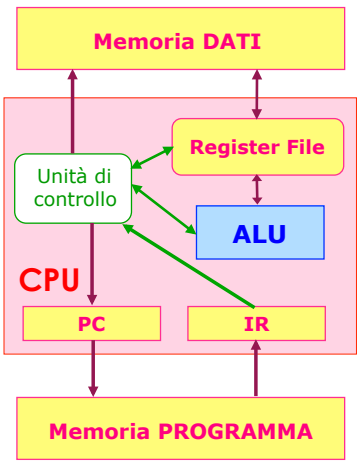
CPU datapath: **lw**



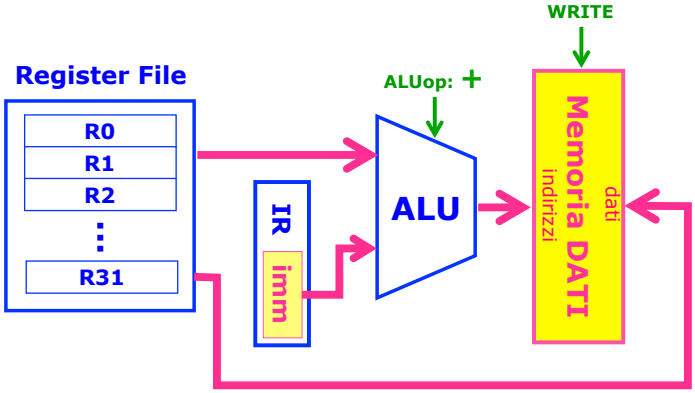
Istruzioni: lw/sw

Configurazione circuitale (datapath): **sw (tipo I)**

- **porta 1 lettura RF** → **I porta ingresso ALU**
- **porta 2 lettura RF** → **I porta DATI Memoria dati**
- **IMMEDIATO IR** → **II porta ingresso ALU**
- **ALU OUT** → **porta INDIRIZZI Memoria dati**



CPU datapath: **sw**



Memorizzazione di arrays

Esempio: accesso ad arrays di parole di 32 bit (4 byte):

- L'elemento **i-esimo A[i]** si trova all'indirizzo: **`br + 4*i`**
  - **`br`** è il registro base; **`i`** è l'indice ad alto livello
  - Il registro base punta all'inizio dell'array: primo byte del primo elemento
  - Spiazzamento ***4i*** per selezionare l'elemento ***i*** dell'array

Elem. array	parola (word)			
A[0] :	0	1	2	3
A[1] :	4	5	6	7
A[2] :	8	9	10	11
	....			
A[k–2] :				
A[k–1] :	4k–4	4k–3	4k–2	4k–1

Indirizzo	contenuto
$s3 ➔	A[0]
$s3 + 4 ➔	A[1]
$s3 + 8 ➔	A[2]
	.....
$s3 + 4i ➔	A[i]
	.....

I tipi di istruzione

- Categorie di istruzione:
  - Istruzioni aritmetico-logiche
  - Istruzioni di trasferimento dati
  - Istruzioni di salto

Istruzioni di salto condizionato e incondizionato

- Istruzioni di ***salto condizionato (branch)***: il salto viene eseguito solo se una certa condizione risulta soddisfatta.
  - **`beq`** (*branch on equal*)
  - **`bne`** (*branch on not equal*)

```
Esempi:     beq r1, r2, L1      # go to L1 if (r1 == r2)
            bne r1, r2, L1      # go to L1 if (r1 != r2)
```

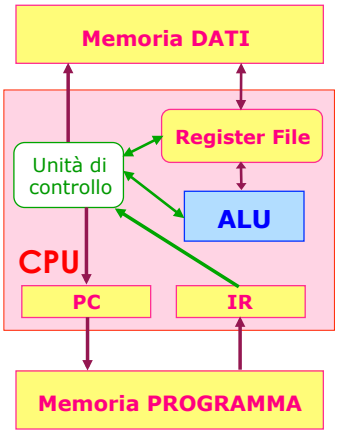
- Istruzioni di ***salto incondizionato (jump):*** il salto va sempre eseguito.
  - **`j`** (jump)
  - **`jr`** (jump register)
  - **`jal`** (jump and link)

```
Esempi:     j   L1          # go to L1
            jr  r31         # go to add. contained in r31
            jal L1          # go to L1, save address of next
                            # instruction in ra
```

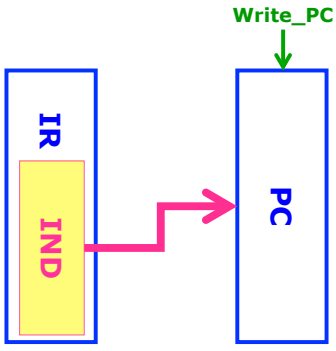

Istruzioni: *jump (tipo J)*

Configurazione circuitale (datapath): **`j INDIRIZZO (tipo J)`**

IMMEDIATO IR ➔ Program Counter (PC)



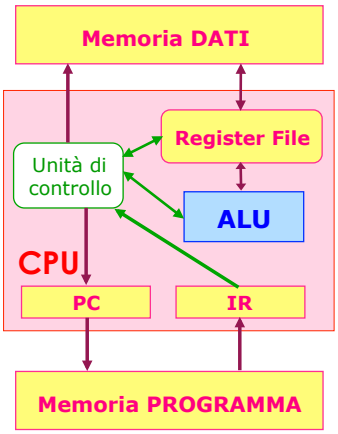
CPU datapath: **`j IND`**



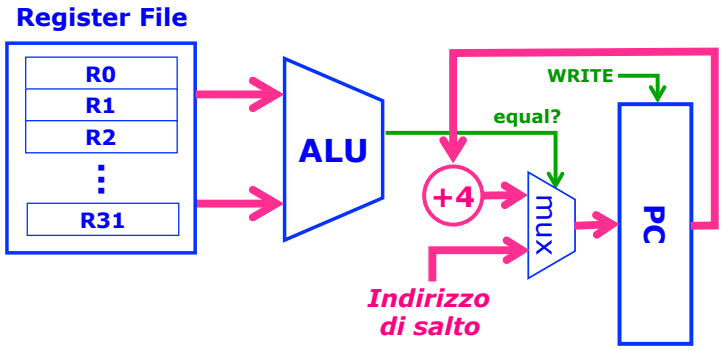
Istruzioni: beq (branch on equal)

Configurazione circuitale (datapath): **beq (branch on equal)**

- **porta 1 lettura RF → I porta ingresso ALU**
- **porta 2 lettura RF → I porta DATI Memoria dati**
- **IMMEDIATO IR → II porta ingresso ALU**
- **ALU OUT → porta INDIRIZZI Memoria dati**



CPU datapath: **beq rA rB LOC**



Salto condizionato relativo

Branch in MIPS: **salti condizionati relativi**:

- `beq r1, r2, LOC` (*branch on equal*)
- `bne r1, r2, LOC` (*branch on not equal*)

- **condizionati**: il flusso sequenziale cambia solo se la condizione è vera.
- **relativi**: il calcolo del valore dell'etichetta `LOC` (indirizzo di destinazione del salto) è relativo al Program Counter (PC):
  Mi sposto di LOC parole rispetto alla posizione attuale indicata dal PC