Architettura degli Elaboratori

MIPS:

Decisioni e cicli

Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Stiamo studiando l'Instruction Set del MIPS
- Abbiamo visto
  - Istruzioni aritmetiche, di trasferimento dati, logiche
  - L'organizzazione dei registri e della memoria
  - La codifica delle istruzioni in codice macchina

- Obiettivo di oggi
  - Istruzioni per prendere decisioni (salti condizionati e non) e per realizzare cicli
  - Modalità di indirizzamento nel MIPS
  - Decodifica di istruzioni in codice macchina

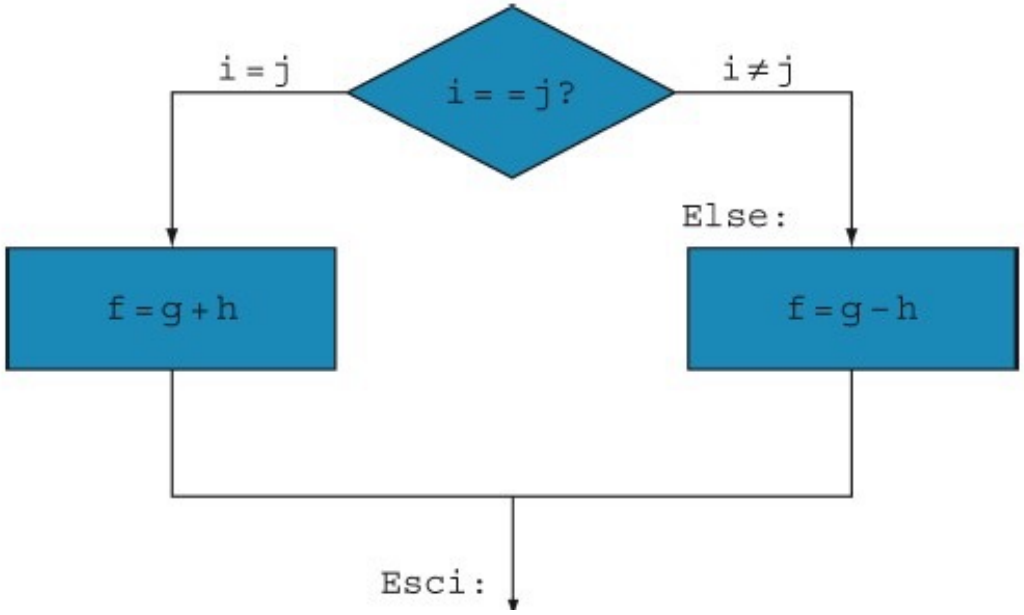
Prendere decisioni

- Talvolta si ha la necessità di alterare il flusso di un programma al verificarsi di certe condizioni
- Il MIPS supporta questa necessità fornendo istruzioni di salto condizionato
  - Branch if EQual: beq reg1, reg2, L1
    - Salta all'istruzione con etichetta L1 se il contenuto del registro reg1 è uguale al contenuto del registro reg2
  - Branch if Not EQual: bne reg1, reg2, L1
    - Salta all'istruzione con etichetta L1 se il contenuto del registro reg1 è diverso dal contenuto del registro reg2
- Inoltre, il MIPS offre una istruzione di salto incondizionato
  - Jump: j L1
    - Salta all'istruzione con etichetta L1

Costrutto if...else

- Supponiamo di avere il frammento di codice C

  if (i==j) f = g+h; else f = g-h;



Costrutto if...else

- Vediamo la traduzione in istruzioni assembly MIPS di

  if (i==j) f = g+h; else f = g-h;

  - Supponiamo che
    - Il registro $s0 contenga f,
    - Il registro $s1 contenga g, il registro $s2 contenga h,
    - Il registro $s3 contenga i, il registro $s4 contenga j

- La traduzione in assembly MIPS è

```
        bne $s3, $s4, ELSE #salta a ELSE se i ≠ j
        add $s0, $s1, $s2 #f = g+h (saltata se i ≠ j)
        j ESCI #salto incondizionato a ESCI
ELSE: sub $s0, $s1, $s2 #f = g-h (saltata se i == j)
ESCI: #segna la fine del costrutto if...else
```

Costrutto if...else

- Vediamo la traduzione in istruzioni assembly MIPS di

  if (i==j) f = g+h; else f = g-h;

  - Supponiamo che
    - Il registro $s0 contenga f,
    - Il registro $s1 contenga g, il registro $s2 contenga h,
    - Il registro $s3 contenga i, il registro $s4 contenga j

- Altra traduzione in assembly MIPS è

```
        beq $s3, $s4, THEN #salta a THEN se i == j
        sub $s0, $s1, $s2 #f = g-h (saltata se i ==j)
        j ESCI #salto incondizionato a ESCI
THEN: add $s0, $s1, $s2 #f = g+h (saltata se i ≠ j)
ESCI: #segna la fine del costrutto if...else
```

Costrutto if...else

- Tra le due soluzioni viste nelle slide precedenti si preferisce la prima
- Motivo: subito dopo l'istruzione di salto condizionato c'è l'istruzione corrispondente alla prima diramazione della scelta

```
if (i==j) f = g+h; else f = g-h;
```

```
    bne $s3, $s4, ELSE #salta a ELSE se i ≠ j
    add $s0, $s1, $s2 #f = g+h (saltata se i ≠ j)
    j ESCI #salto incondizionato a ESCI
ELSE: sub $s0, $s1, $s2 #f = g-h (saltata se i == j)
ESCI: #segna la fine del costrutto if...else
```

Formato istruzioni di salto condizionato

- Le istruzioni di salto condizionato

  bne $s0, $s1, L1

  beq $s0, $s1, L1

hanno formato I

Etichetta o indirizzo di salto

op	rs	rt	16 bit address
6 bit	5 bit	5 bit	

Formato istruzioni di salto condizionato

- Poiché il campo per l'etichetta è di 16 bit, possiamo indirizzare 2^{16} locazioni della memoria istruzioni

Etichetta o indirizzo di salto

op	rs	rt	16 bit address

- Il range di rappresentabilità in complemento a 2 su 16 bit è $[-2^{15}, 2^{15}-1]$
- In realtà, sui 16 bit dell'indirizzo di salto si eseguono queste operazioni:
  - Estensione del segno a 32 bit;
  - Shift logico a sx di 2 posti dei 32 bit ottenuti (corrisponde a moltiplicare per 4);
  - Somma dei 32 bit ottenuto dopo lo shift con il contenuto (a 32 bit) di uno specifico registro

Formato istruzioni di salto condizionato

- Quale registro utilizzare nel calcolo dell'indirizzo di salto?

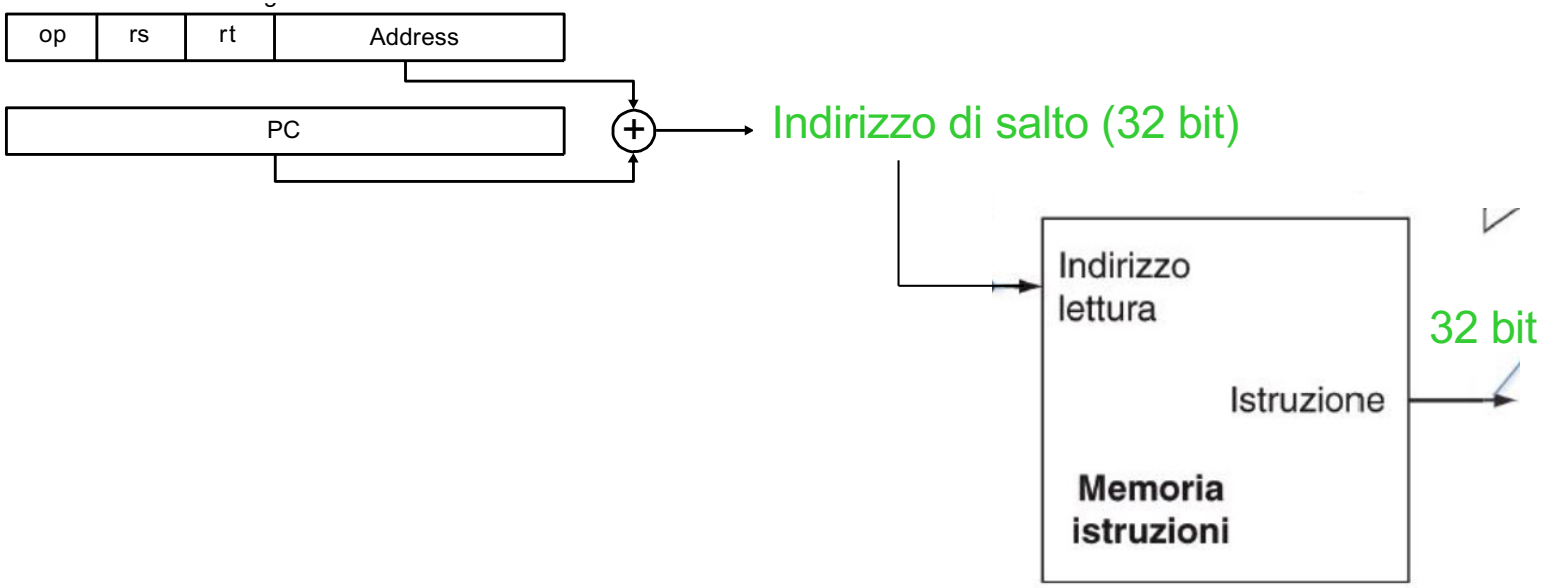
Etichetta o indirizzo di salto

op	rs	rt	16 bit address

- Viene utilizzato il Program Counter (PC)
  - Tale registro contiene l'indirizzo dell'istruzione corrente in memoria istruzioni
  - Non appena una istruzione viene prelevata, il PC viene incrementato (in modo da puntare all'istruzione successiva)
  - Quindi l'indirizzo considerato per il calcolo dell'indirizzo di salto sarà quello dell'istruzione successiva (PC+4) e non quello dell'istruzione corrente (PC)

Indirizzamento relativo al Program Counter

- Quindi l'indirizzo di salto è la somma del contenuto del PC (incrementato di 4) e di una costante a 16 bit contenuta nell'istruzione (di tipo I)
  - Dopo l'estensione del segno e lo shift logico a sx di 2 posti
- Questa modalità di calcolo dell'indirizzo viene detta Indirizzamento relativo al Program Counter



Esempio

- Supponiamo di avere nella memoria istruzioni, all'indirizzo 80000, la seguente istruzione di salto condizionato

beq $s1, $s2, label

Es: - istruzione beq a indirizzo 80000
- label a indirizzo 80104

#words da saltare a partire da PC+4

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, label`	000100	10001	10010	0000 0000 0001 1001

25

- L'indirizzo dell'istruzione a cui saltare in caso di eguaglianza dei registri $s1 ed $s2 si ottiene
  - Moltiplicando l'indirizzo per 4: 25*4 = 100;
  - Sommando al valore ottenuto (100) il valore di PC+4 (80004)

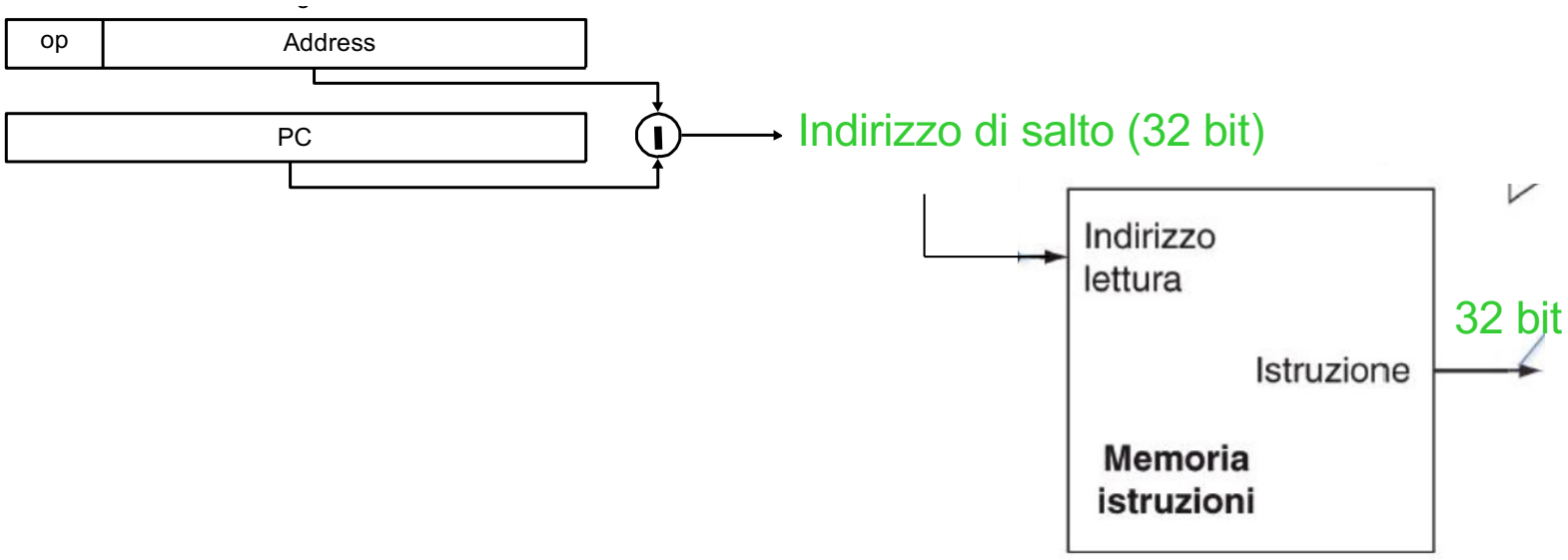
Formato istruzioni di salto incondizionato

- L'istruzione di salto incondizionato j L1 ha formato J e codice operativo 2

  - Consente quindi di indirizzare 2^{26} locazioni in memoria istruzioni
  - In realtà, l'indirizzo di salto è ottenuto nel modo seguente:
    - Si effettua uno shift logico a sx di 2 posizioni a partire dai 26 bit dell'etichetta (senza cestinare i 2 bit in posizione più significativa)
    - Si aggiungono in testa alla stringa di 28 bit, ottenuta dopo lo shift, i 4 bit più significativi del PC

Indirizzamento Pseudodiretto

- Quindi l'indirizzo di salto è la concatenazione dei 4 bit più significativi del contenuto del PC e di una costante a 26 bit contenuta nell'istruzione (di tipo J), dopo uno shift a sx di 2 posti
- Questa modalità di calcolo dell'indirizzo viene detta Indirizzamento pseudodiretto

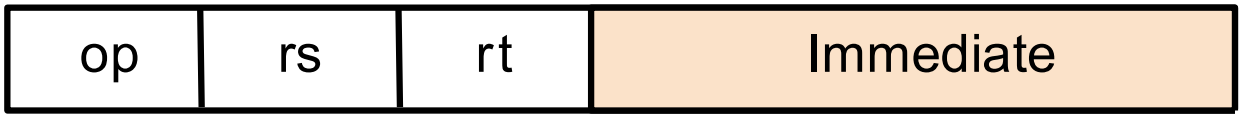


Modalità di indirizzamento

- Il MIPS, oltre alla modalità di indirizzamento relativa al Program Counter e alla modalità di indirizzamento pseudodiretto, ne offre altre tre
  - Indirizzamento immediato
  - Indirizzamento tramite registro
  - Indirizzamento tramite registro base e spiazzamento

Indirizzamento Immediato

- In questa modalità, l'operando è una costante di 16 bit contenuta nell'istruzione (di tipo I)

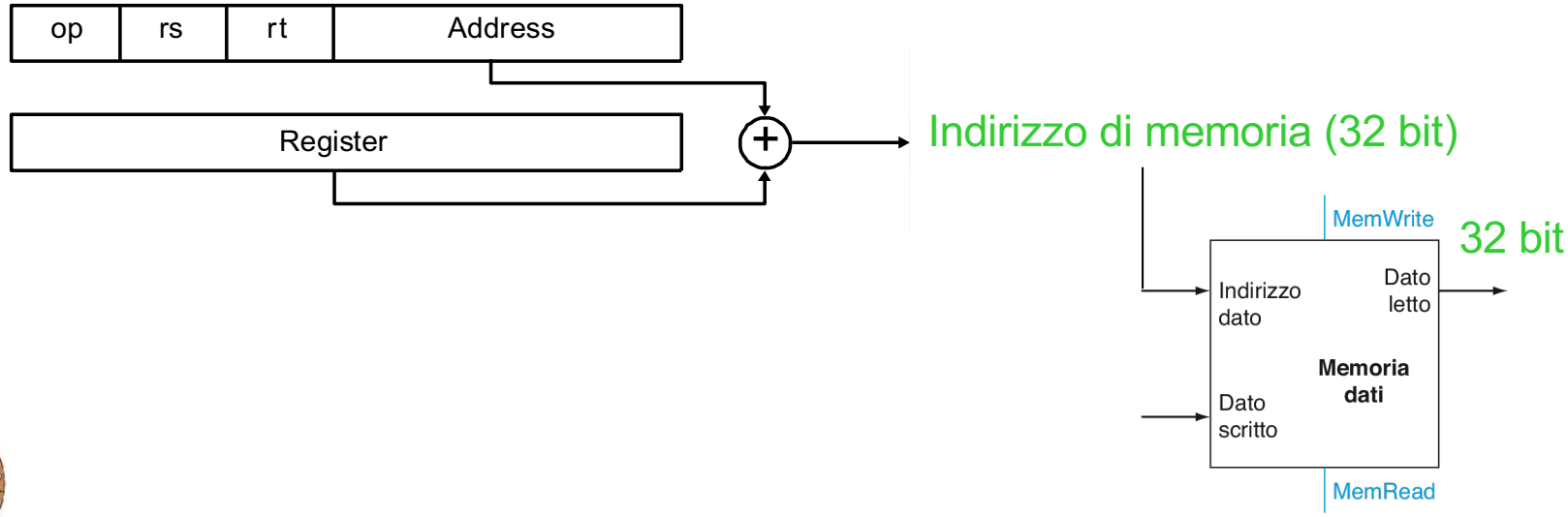


Indirizzamento tramite Registro

- In questa modalità, l'operando è il contenuto di un registro specificato nell'istruzione (di tipo R)

Indirizzamento tramite Registro Base e Spiazzamento

- In questa modalità, l'operando è una locazione di memoria
  - L'indirizzo della locazione è dato dalla somma del contenuto di un registro (base) e di una costante (spiazzamento o offset) contenuta nell'istruzione (di tipo I)



Altri confronti

- Il MIPS offre anche il confronto
  "un operando è minore di un altro o di una costante?"
  - Il confronto avviene mediante istruzioni che settano a 1 il valore di registri di flag
- Set if Less Than
  - slt $t0, $s3, $s4 #setta $t0=1 se il contenuto di $s3 #è minore del contenuto di $s4
  - slti $t0, $s3, 10 #setta $t0=1 se il contenuto di $s3 #è minore di 10
- I salti su condizioni di tipo minore uguale o maggiore uguale si ottengono
  - Combinando slt con beq o bne e
  - Usando la costante 0 (disponibile nel registro $zero) per fare i test

Altri confronti

- Vediamo la traduzione in istruzioni MIPS di

  if (i<j) f = g+h; else f = g-h;

  - Supponiamo che
    - Il registro $s0 contenga f
    - Il registro $s1 contenga g, il registro $s2 contenga h
    - Il registro $s3 contenga i, il registro $s4 contenga j
- La traduzione MIPS è

```
        slt $t0, $s3, $s4 #setta $t0=1 se i < j
        beq $t0, $zero, ELSE #salta a ELSE se  i ≥ j
        add $s0, $s1, $s2 #f = g+h (saltata se i ≥ j)
        j ESCI #salto incondizionato a ESCI
ELSE: sub $s0, $s1, $s2 #f = g-h (saltata se i < j)
ESCI: #segna la fine del costrutto if…else
```

Altri confronti

- L'esito di un confronto con slt o slti dipende chiaramente dai segni degli operandi

- Se si vuole operare su numeri senza segno (unsigned) bisogna usare sltu o sltui
  - sltu: Set Less Than Unsigned
  - sltui: Set Less Than Unsigned Immediate

Esempio

- Supponiamo che il contenuto dei due registri $s0 e$s1 sia il seguente

```
s0 = 1111 1111 1111 1111 1111 1111 1111 1111
```

```
s1 = 0000 0000 0000 0000 0000 0000 0000 0001
```

- L'istruzione slt $t0, $s0, $s1 produce $t0=1
- Invece, l'istruzione sltu $t0, $s0, $s1 produce $t0=0

Formato istruzioni di confronto

- L'istruzione

  slt $t0, $s1, $s2

  ha formato R

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

I cicli

- I cicli vengono utilizzati per iterare un certo calcolo più volte
- Per realizzare un ciclo si utilizzano le stesse istruzioni usate per prendere decisioni
- Vediamo come tradurre in istruzioni MIPS il seguente frammento di codice C

```
while (salva[i]==k)
    i +=1;
```

- Supponiamo che
  - $s3 e $s5 contengano i e k
  - $s6 contenga l'indirizzo base del vettore salva

I cicli

- Il primo passo consiste nel caricare l'indirizzo di salva[i] in un registro temporaneo ($t0)
  - Otteniamo innanzitutto 4*i usando lo shift logico a sx di 2 posti a partire dal registro $s3, che contiene i

    ```
    sll $t1, $s3, 2 #registro temporaneo $t1=4*i
    ```

  - Poi sommiamo a $s6, che contiene l'indirizzo base del vettore salva, il valore 4*i, contenuto nel registro temporaneo $t1

    ```
    add $t1, $t1, $s6 #$t1 contiene indirizzo di salva[i]
    ```

  - Carichiamo nel registro temporaneo $t0 la word contenuta alla locazione con indirizzo contenuto in $t1, cioè la word contenuta in salva[i]

    ```
    lw $t0, 0($t1) #registro temporaneo $t0=salva[i]
    ```

  - Infine aggiungiamo l'etichetta Ciclo alla prima istruzione, per poter tornare indietro al termine del ciclo

    ```
    Ciclo: sll $t1, $s3, 2
    ```

I cicli

- Il secondo passo consiste nell'eseguire il controllo di uscita dal ciclo:
  - Si esce dal ciclo se salva[i]≠k

```
bne $t0, $s5, Esci #salta a Esci se salva[i]≠k
```

- Poi effettuiamo la somma

```
addi $s3, $s3, 1 #$s3 contiene i+1
```

- Alla fine del ciclo c'è un salto indietro al test

```
j Ciclo #vai a Ciclo
```

- Infine aggiungiamo l'etichetta Esci alla fine

```
Esci:
```

I cicli

In definitiva, la traduzione in istruzioni MIPS di

```
while (salva[i]==k)
        i +=1;
```

```
Ciclo:  sll $t1, $s3, 2 #registro temporaneo $t1=4*i
        add $t1, $t1, $s6 #$t1 contiene indirizzo di salva[i]
        lw $t0, 0($t1) #registro temporaneo $t0=salva[i]
        bne $t0, $s5, Esci #salta a Esci se salva[i]≠k
        addi $s3, $s3, 1 #$s3 contiene i+1
        j       Ciclo #vai a Ciclo
Esci: #segna la fine del ciclo
```

Decodificare il linguaggio macchina

- In alcuni casi è utile ricostruire il codice assembly a partire dal codice macchina
- Ad esempio, qual è l'istruzione MIPS corrispondente a codice macchina $00AF8020_{16}$?
- Il primo passo consiste nel convertire il valore esadecimale in binario, in modo da poter esaminare i campi dell'istruzione:

0000 0000 1010 1111 1000 0000 0010 0000

  - I primi 6 bit indicano il codice operativo: essendo 000000, si tratta di un'istruzione in formato R

Decodificare il linguaggio macchina

- Formattiamo l'istruzione secondo i campi del formato R

op (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
000000	00101	01111	10000	00000	100000

- Il campo funct (6 bit) contiene 100000, che corrisponde all'istruzione add
- Il campo rs contiene 5_{10} (corrisponde al registro $a1)
- Il campo rt contiene 15_{10} (corrisponde al registro $t7)
- Il campo rd contiene 16_{10} (corrisponde al registro $s0)
- Il campo shamt non viene utilizzato (contiene 0)

L'istruzione MIPS corrispondente è quindi

add $s0, $a1, $t7

Riepilogo e riferimenti

- Istruzioni MIPS per prendere decisioni e realizzare cicli
  - [PH] par. 2.7
- Modalità di indirizzamento
  - [PH] par. 2.10
- Decodifica di istruzioni in codice macchina
  - [PH] par. 2.10