Aritmetica MIPS

- **Tutte le istruzioni aritmetiche hanno 3 operandi**
- **L'ordine degli operandi è fissato (destinazione per prima)**
- **Forma:**

```
operatore operando1, operando2, operando3
```

- **Significato:**

```
operando1 = operando2 operatore operando3
```

- **Esempio:**

codice MIPS : `add $s0, $s1, $s2`

codice C : `a = b + c`

- **Principio di progetto 1: semplicità e regolarità dell'hardware**

Aritmetica MIPS

- **Svantaggi della regolarità**

C code:

```
a = b + c + d;
e = f - a;
```

MIPS code:

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

- **Gli operandi devono essere registri**
  - solo 32 registri da 4Bytes (= 1 Word)
- **Principio di progetto 2: dimensioni minori => maggiore velocità**
  - Num. registri molto alto → probabile aumento della durata del ciclo di clock
  - Num. registri molto alto → aumenta il num. di bit necessari per indirizzarli

Qualche esempio

- Esempio 1

```
#a = b + c + d + e
add   $t0, $t1, $t2      #a = b + c
add   $t0, $t0, $t3      #a = a + d
add   $t0, $t0, $t4      #a = a + e
```

- Esempio 2

```
#f = (g + h) - (i + j)
add   $t0, $s1, $s2          #$t0 = g + h
add   $t1, $s3, $s4          #$t1 = i + j
Sub   $s0, $t0, $t1          #f = $t0 - $t1
```

Istruzioni di trasferimento dati - lw

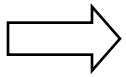
- `lw` **(Load Word)**

  word in memoria => reg.

- **Forma:**
  `operatore registro, offset(indirizzo_base)`
- **Esempi:**
  - A è un vettore di 100 parole
  - Il compilatore associa le variabili g e h ai registri $s1 e $s2
  - L'indirizzo di partenza (detto *di base*) del vettore è contenuto in $s3
- **C code:** `g = h + A[2];`

  MIPS code:

```
lw  $t0, 2($s3)
add $s1, $s2, $t0
```



```
lw  $t0, 8($s3)
add $s1, $s2, $t0
```

Attenzione: l'indirizzamento è al byte e gli indici degli array partono da 0

(quindi vogliamo leggere la terza parola del vettore)

Istruzioni di trasferimento dati - sw

- **`sw`** **(Store Word)**

  reg. => word in memoria

- **Forma:**

  `operatore registro, offset(indirizzo_base)`

- **Esempi:**
  - A è un vettore di 100 parole
  - L'indirizzo di partenza (detto *di base*) del vettore è contenuto in $s3
  - Il compilatore associa la variabile h al registro $s2
- **C code:** `A[12] = h + A[8];`

  MIPS code:

```
lw  $t0, 32($s3)
add $t0, $s2, $t0
sw  $t0, 48($s3)
```

Esempio con indice variabile

- `g = h + A[i]`

  `g` ⟷ `$s1`

  `h` ⟷ `$s2`

  `A` ⟷ `$s3 (indirizzo base)`

  `i` ⟷ `$s4 (offset)`

MIPS code:

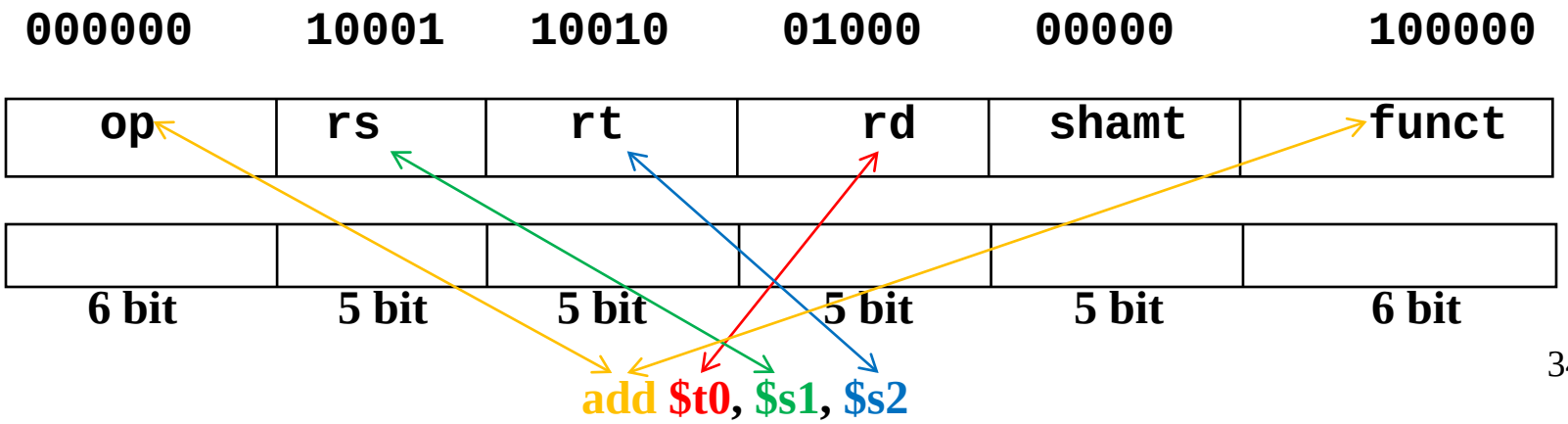
```
add $t1, $s4, $s4
add $t1, $t1, $t1
add $t1, $t1, $s3
lw $t0,  0($t1)
add $s1, $s2, $t0
```

Rappresentazione delle Istruzioni Aritmetiche

- **Istruzioni, registri e words di dati sono di 32 bits**
  - sono codificati all'interno del calcolatore come sequenze di valori binari (es., voltaggio "alto" e "basso")
  - ovvero, rappresentano numeri in base 2
- **Esempio:** `add $t0, $s1, $s2`
  - I registri sono numerati:
    - `$t0->8, …..,  $t7->15`
    - `$s0->16, …..,  $s7->23`

Formato istruzione:
modo di rappresentare un'istruzione, attraverso un insieme di campi di numeri binari

- **Formato istruzione *R-type*:**

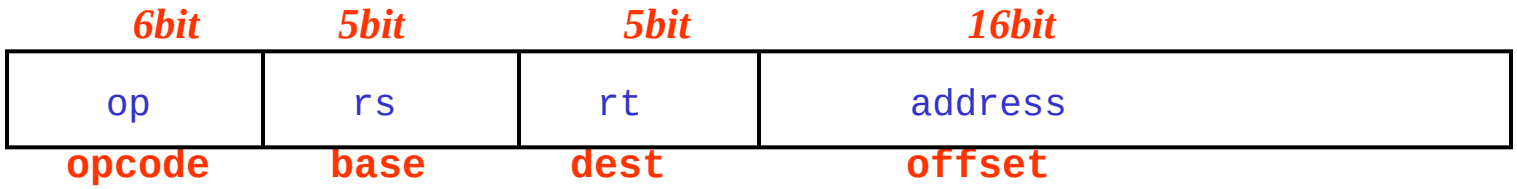


Formato *R-type*

- **op : codice operativo (opcode) dell'istruzione**
- **rs: primo operando (registro)**
- **rt: secondo operando (registro)**
- **rd: registro destinazione, che contiene il risultato dell'operazione**
- **shamt: valore dello scalamento (shift amount)**
- **funct: seleziona una variante dell'operazione base specificata dall'opcode (codice funzione, function code)**

Rappresentazione delle istruzioni lw/sw

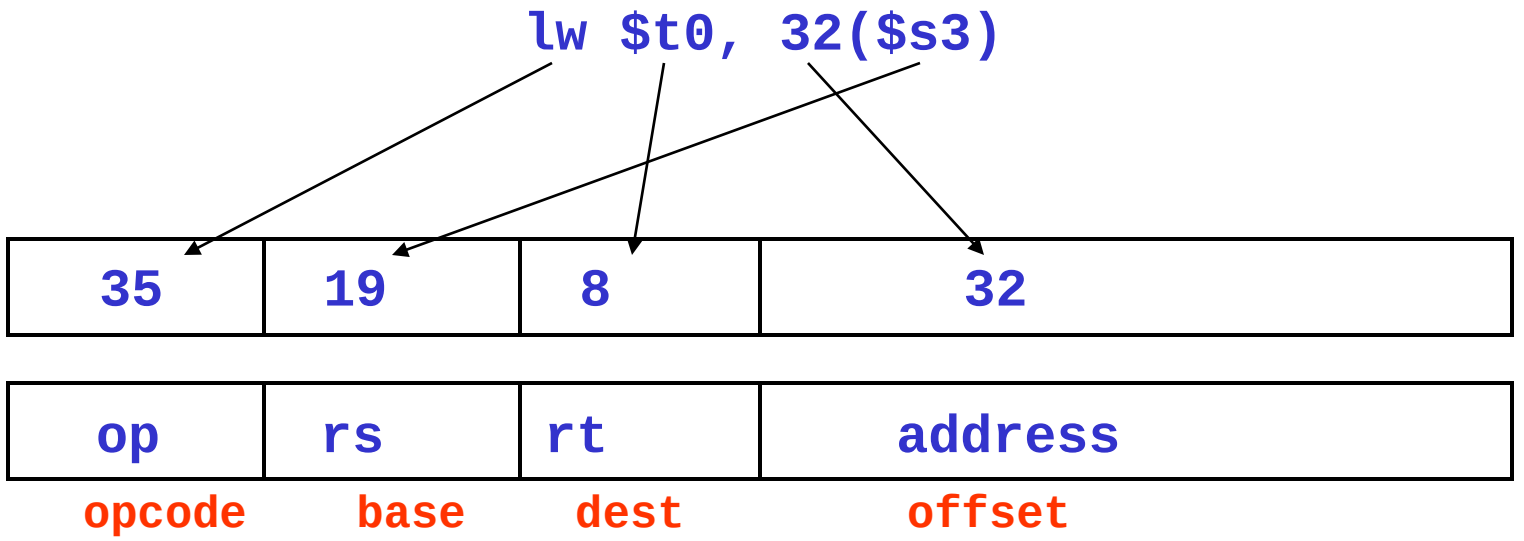
- **Formato istruzione (*R-type*): 3 registri**
- **E le istruzioni lw e sw?**
  - 2 registri e una costante
  - 5 bit non sono sufficienti
  - occorre nuovo formato
- **Formato istruzione (*I-type*)**



- **Spazio di indirizzamento: ± 2^{15} bytes = ± 2^{13} parole da indirizzo base (rs)**

Formato *I-type*

- **Esempio**



- **Principio di progetto 3:**
  - un buon progetto richiede buoni compromessi
    - si mantiene uguale la lunghezza delle istruzioni, predisponendo formati diversi per tipi di istruzioni diverse

Riassunto

- **MIPS**
  - aritmetica solo su registri
  - load/store word, con indirizzamento al byte

Istruzioni	Significato
`add $t1, $t2, $t3`	`$t1 = $t2 + $t3`
`sub $t1, $t2, $t3`	`$t1 = $t2 - $t3`
`lw $t0, 100($t1)`	`$t0 = Memory[$t1+100]`
`sw $t0, 100($t1)`	`Memory[$t1+100] = $t0`

Riassunto

- **Istruzioni aritmetiche `add, sub`**
  - Esempio: `add $t0,$s1,$s2`

000000	10001	10010	01000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bit	**5 bit**	**5 bit**	**5 bit**	**5 bit**	**6 bit**

- **Istruzioni di trasferimento dati `lw, sw`**
  - Esempio: `lw $t0,32($s3)`

100011	10011	01000	0000000000100000
op	**rs**	**rt**	**address**
6 bit	**5 bit**	**5 bit**	**16 bit**