

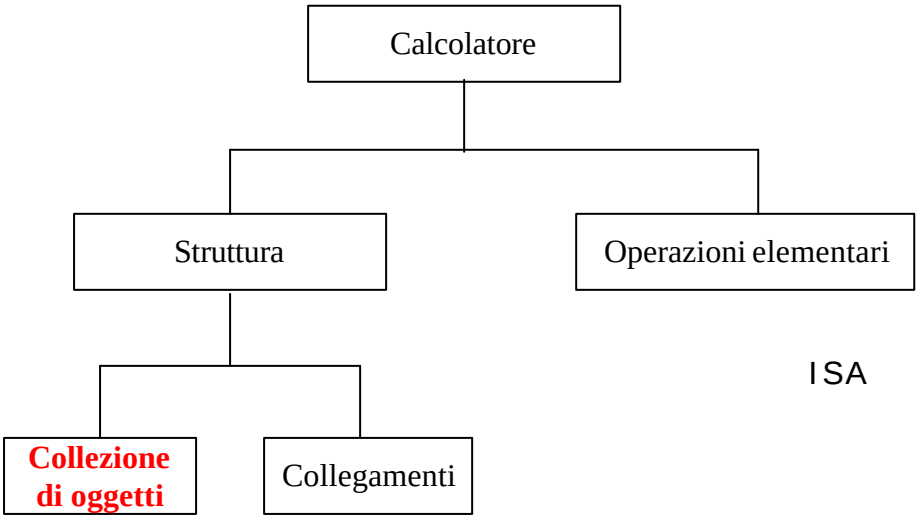

L'Instruction Set Architecture ed il Linguaggio Assembly

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano


Sommario

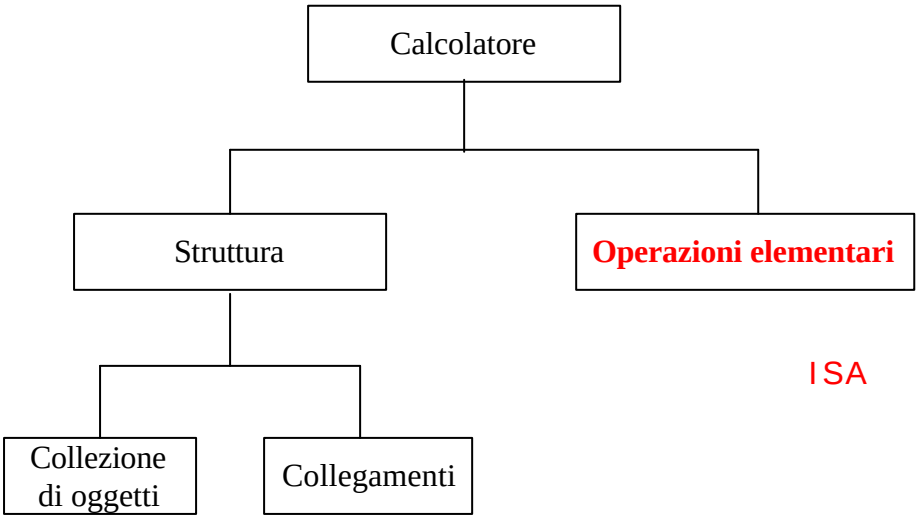
Descrizione di un elaboratore



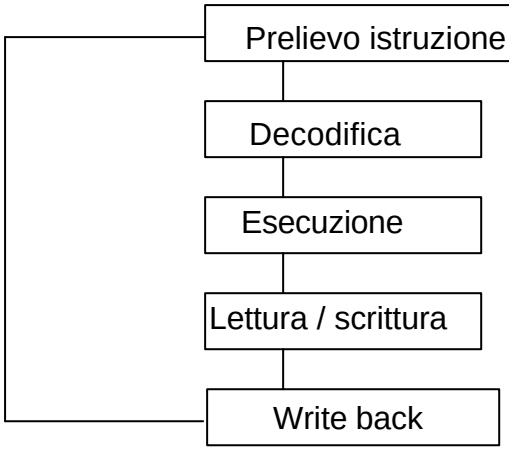
Elementi principali della CPU

- Banco di registri *(Register File)* ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale. Tra questi registri si distinguono il *MAR* e *MDR.*
- Registro *Program counter (PC).* Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzione;
- Registro *Instruction Register (IR)*. Contiene l'istruzione in corso di esecuzione.
- Unità per l'esecuzione delle operazioni aritmetico-logiche (*Arithmetic Logic Unit - ALU*). I dati forniti all'*ALU* possono provenire da registri oppure direttamente dalla memoria, a seconda delle modalità di indirizzamento previste;
- Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile *(Floating Point Unit – FPU)*, sommatori ausiliari, ecc.;
- *Unità di controllo*. Controlla il flusso e determina le operazioni di ciascun blocco.

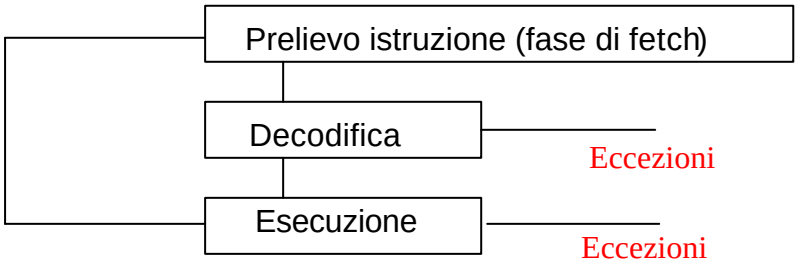
Descrizione di un elaboratore



Ciclo di esecuzione di un'istruzione MIPS



Ciclo di esecuzione di un'istruzione



Il normale flusso di esecuzione di un programma può essere interrotto da un segnale di interruzione esterno *(INTERRUPT)* per una richiesta di intervento che un dispositivo di I/O ad esempio invia al processore.

Lettura dell'istruzione (fetch)

- Istruzioni e dati risiedono nella memoria principale, dove sono stati caricati attraverso un'unità di ingresso.
- L'esecuzione di un programma inizia quando il registro PC punta alla prima istruzione del programma.
- Il segnale di controllo per la lettura (READ) viene inviato alla memoria.
- Trascorso il tempo necessario all'accesso in memoria, la parola indirizzata (in questo caso la prima istruzione del programma) viene letta dalla memoria e trasferita nel registro IR.

Decodifica dell'istruzione

- L'istruzione contenuta nel registro IR viene decodificata per essere eseguita. Alla fase di decodifica corrisponde la predisposizione della CPU (apertura delle vie dicomunicazione appropriate) all'esecuzione dell'istruzione.
- In questa fase vengono anche recuperati gli operandi. Nelle architetture MIPS gli operandi possono essere solamente nel Register File (oppure contenuti nell'istruzione stessa).

Esecuzione dell'istruzione

Viene selezionato il circuito / i circuiti combinatori appropriati per l'esecuzione delle operazioni previste dall'istruzione e determinate in fase di decodifica.

Tra le operazioni previste, c'è anche la formazione dell'indirizzo di memoria da cui leggere o su cui scrivere un dato.

Lettura / Scrittura in memoria

In questa fase il dato presente in un registro, viene scritto in memoria oppure viene letto dalla memoria un dato e trasferito ad un registro.

Questa fase non è richiesta da tutte le istruzioni!

Nel caso particolare di Architetture LOAD/STORE, quali MIPS, le istruzioni di caricamente dalla memoria sono separate da quelle aritmetico/logiche. Se effettuo una Letura / Scrittura, **non** eseguo operazioni aritmetico logiche sui dati.

Sistema di memoria “sganciato” dalla coppia register-file + CPU.

Scrittura in register file (write-back)

- Il risultato dell’operazione può essere memorizzato nei registri ad uso generale oppure in memoria.
- Mentre viene eseguita un’istruzione, il contenuto del PC viene incrementato in modo da puntare all’istruzione successiva.
- Non appena è terminato il ciclo di esecuzione dell’istruzione corrente (termina la fase di Write Back), si preleva l’istruzione successiva dalla memoria.

Esecuzione dell'istruzione

Viene selezionato il circuito / i circuiti combinatori appropriati per l'esecuzione delle operazioni previste dall'istruzione e determinate in fase di decodifica.

Eesempio: add $s3, $s2, $s1

Fase di fetch:	Caricamento dell'istruzione.
Decodifica:	Preparazione della CPU a svolgere una somma.
Lettura dei dati:	Indirizzamento adeguato del Register File.
Esecuzione:	Esecuzione della somma.

Recupero degli operandi

- Se l'istruzione deve essere svolta dall'unità aritmetico-logica è necessario recuperare gli operandi richiesti, che possono risiedere nei registri di uso generale oppure in memoria.
- Architetture a registri:
  - Se un operando risiede in memoria, deve essere prelevato caricando l'indirizzo dell'operando nel registro MAR e attivando un ciclo di READ della memoria.
  - L'operando letto dalla memoria viene posto nel registro MDR per essere trasferito alla ALU, che esegue l'operazione. Nelle architetture MIPS, l'operando viene trasferito nel Register file nella fase di Scrittura.
- Architetture LOAD/STORE:
  - Le istruzioni di caricamente dalla memoria sono separate da quelle aritmetico/logiche.

Esempio ciclo di esecuzione

Somma: add $s3, $s2, $s1

Fase di fetch:	Caricamento dell'istruzione.
Decodifica:	Preparazione della CPU a svolgere una somma. Determinazione dei segnali di controllo. Lettura degli operandi (che sono contenuti nei registri $s2, $s1).
Esecuzione:	Esecuzione della somma.
R / W:	Nulla
Write-back	Scrittura del registro $s3.

Sommario

Il ciclo di esecuzione di un'istruzione

L'ISA ed il linguaggio macchina

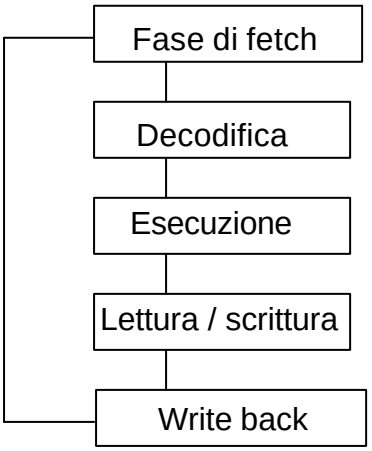
L'Assembly

I registri

Analisi dell'Instruction Set Architecture

I tipi di istruzioni: istruzioni aritmetiche

Caratteristiche di un'ISA



Formato e codifica di un'istruzione
– tipo e dimensione.

Posizione degli operandi e risultato.
– quanti?
– memoria e/o registri

Tipo e dimensione dei dati

Operazioni consentite

Definizione di un'ISA

Definizione del funzionamento: insieme delle istruzioni (interfaccia verso i linguaggi ad alto livello).

- Tipologia di istruzioni.
- Meccanismo di funzionamento.

Definizione del formato: codifica delle istruzioni (interfaccia verso l'HW).

- Formato delle istruzioni.
- Suddivisione dei bit che costituiscono l'istruzione in gruppi omogenei.

Tipi di istruzioni

- Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:
  - Istruzioni aritmetico-logiche;
  - Istruzioni di trasferimento da/verso la memoria *(load/store);*
  - Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;
  - Istruzioni di trasferimento in ingresso/uscita (I/O).

Codifica delle istruzioni

- Tutte le istruzioni MIPS hanno la **stessa** dimensione **(32 bit) – Architettura RISC.**
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit **(6 bit)** più significativi **(codice operativo - *OPCODE*)**
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register) – Lavorano su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate) – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - **Tipo J (jump) – Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

Le istruzioni in linguaggio macchina

- Linguaggio di programmazione direttamente comprensibile dalla macchina
  - Le parole di memoria sono interpretate come *istruzioni*
  - Vocabolario è *l'insieme delle istruzioni (instruction set)*

Programma in linguaggio ad alto livello (C)

```
a = a + c
b = b + a
var = m [a]
```

Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```

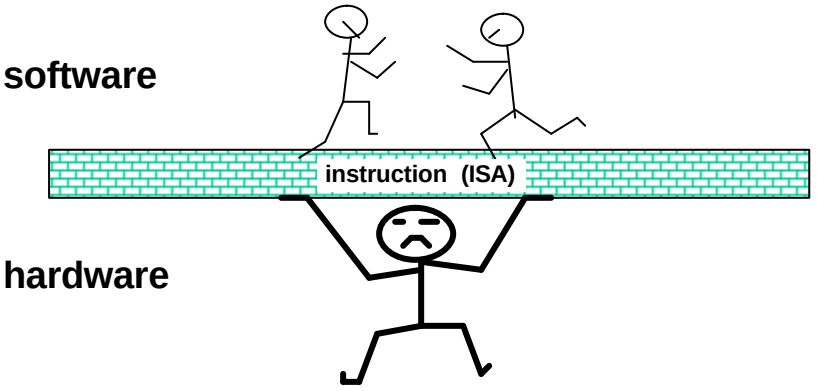

Le istruzioni di un'ISA

Devono contenere tutte le informazioni necessarie ad eseguire il ciclo. Registri, comandi, ...

Ogni architettura di processore ha il suo linguaggio macchina

- Architettura definita dall'insieme delle istruzioni elementari.
  - **ISA (Instruction Set Architecture)**
- Due processori con lo stesso linguaggio macchina hanno la stessa architettura delle istruzioni anche se le implementazioni hardware possono essere diverse.
- Consente al SW di accedere direttamente all'hardware di un calcolatore

Insieme delle istruzioni



Quale è più facile modificare?

Sommario

Il ciclo di esecuzione di un'istruzione

L'ISA ed il linguaggio macchina

L'Assembly

I registri

Analisi dell'Instruction Set Architecture

I tipi di istruzioni: istruzioni aritmetiche

Linguaggio assembly

- Le istruzioni assembly sono una rappresentazione simbolica del linguaggio macchina comprensibile dall'HW.

- Rappresentazione simbolica del linguaggio macchina
  - Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit

- Rispetto ai linguaggi ad alto livello, l'assembly fornisce limitate forme di controllo del flusso e non prevede articolate strutture dati

- Linguaggio usato come linguaggio target nella fase di compilazione di un programma scritto in un linguaggio ad alto livello (es: C, Pascal,ecc.)

- Vero e proprio linguaggio di programmazione che fornisce la visibilità diretta sull'hardware.

Linguaggio C: somma dei primi 100 numeri

```
main()
{
      int i;
      int sum = 0;
      for (i = 0; i <= 100; i = i + 1)
            sum = sum + i*i;
      printf("La somma da 0 a 100 è %d\n",sum);
}
```

Linguaggio assembly: somma dei primi 100 numeri

```
.text                          addu $t9, $t8, $t4
.align 2                       addu $t9, $t8, $t7
.globl main                    sw  $t9, 24($sp)
main:                          addu $t7, $t6, 1
        subu $sp, $sp, 32      sw $t7, 28($sp)
        sw $ra, 20($sp)        bne $t5, 100, loop
        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
```


Fase di compilazione da C ad assembly

Programma in linguaggio ad alto livello (C)

```
n_maschi = n_maschi + nuovoMaschio
n_femmine = n_femmine + nuovaFemmina
n_personePerEta = n_persone[eta]
```

Compilatore

Programma in linguaggio assembly (MIPS)

```
add $2, $2, $4
add $3, $3, $2
lw $15, 4($2)
```

Programma in linguaggio ad alto livello (C)

```
a = a + c
b = b + a
var = m [a]
```

Compilatore

Programma in linguaggio assembly (MIPS)

```
add $2, $4, $2
add $3, $3,$2
lw $15, 4($2)
```

Assemblatore

Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```


Assembly come linguaggio di programmazione

- Principali *svantaggi* della programmazione in linguaggio assembly:
  - Mancanza di portabilità dei programmi su macchine diverse
  - Maggiore lunghezza, difficoltà di comprensione, facilità d'errore rispetto ai programmi scritti in un linguaggio ad alto livello
- Principali *vantaggi* della programmazione in linguaggio assembly:
  - Ottimizzazione delle prestazioni.
  - Massimo sfruttamento delle potenzialità dell'hardware sottostante.

- Le strutture di controllo hanno forme limitate
- Non esistono tipi di dati all'infuori di interi, virgola mobile e caratteri.
- La gestione delle strutture dati e delle chiamate a procedura deve essere fatta in modo esplicito dal programmatore.

Assembly come linguaggio di programmazione

- Alcune applicazioni richiedono un approccio *ibrido* nel quale le parti più critiche del programma sono scritte in assembly (per massimizzare le prestazioni) e le altre parti sono scritte in un linguaggio ad alto livello (le prestazioni dipendono dalle capacità di ottimizzazione del compilatore).

  Esempio: Sistemi embedded o dedicati

Sistemi "automatici" di traduzione da linguaggio ad alto livello (linguaggio C) ad assembly e codice binario ed implementazione circuitale (e.g. sistemi di sviluppo per FPGA).

Sommario

Il ciclo di esecuzione di un'istruzione

L'ISA ed il linguaggio macchina

L'Assembly

I registri

Analisi dell'Instruction Set Architecture

I tipi di istruzioni: istruzioni aritmetiche

I registri

- I registri sono associati alle variabili di un programma dal compilatore.
- Un processore possiede un numero limitato di registri: ad esempio il processore MIPS possiede **32 registri composti da 32-bit (word), register file.**
- Per convenzione si usano nomi simbolici preceduti da $ per denotare i registri, ad esempio:
  `$s0,$s1,...,$s7($s8)` Per indicare variabili in C
  `$t0, $t1, ... $t9` Per indicare variabili temporanee
- I registri possono essere anche direttamente indicati mediante il loro numero (0, …, 31) preceduto da $: ad es.
  `$0, $1, …, $31`

Uso dei registri: convenzioni

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (fp)
$ra	31	indirizzo di ritorno

I registri per le operazioni floating point

- Esistono 32 registri utilizzati per l'esecuzione delle istruzioni.
- Esistono **32** registri per le operazioni floating point (virgola mobile) indicati come

  `$f0, …, $f31`

  – Per le operazioni in doppia precisione si usano i registri contigui

  `$f0, $f2, $f4, …`

Sommario

Il ciclo di esecuzione di un'istruzione

L'ISA ed il linguaggio macchina

L'Assembly

I registri

Analisi dell'Instruction Set Architecture

I tipi di istruzioni: istruzioni aritmetiche

Istruzioni aritmetiche

- Ogni istruzione aritmetica ha **un numero prefissato di** operandi (generalmente tre)
- L'ordine degli operandi **è fisso:**
  - Prima il registro contenente il risultato dell' operazione (registro destinazione) e poi i due operandi (registri sorgente)
  - In alcuni casi (es. moltiplicazione e divisione non floating point) il registro destinazione è implicito.

Istruzioni di trasferimento da/verso la memoria *(load/store)*

- Per eseguire un'istruzione, essa deve essere trasferita dalla memoria alla *CPU*.
- Operandi e risultati delle istruzioni devono essere trasferiti tra memoria e *CPU*.
- Necessarie due modalità di trasferimento di dati/istruzioni tra memoria e registri della *CPU*:
  - *load* (caricamento) o *fetch* (prelievo) o *read* (lettura)
  - *store* (memorizzazione) o *write* (scrittura)

Istruzioni di salto condizionato e incondizionato

- Istruzioni di salto: viene caricato un nuovo indirizzo nel registro contatore di programma (PC) invece dell'indirizzo seguente l'indirizzo di salto secondo l'ordine sequenziale delle istruzioni.
- Istruzioni di salto *condizionato (branch):* il salto viene eseguito solo se una certa condizione risulta soddisfatta.
- Istruzioni di salto *incondizionato (jump):* il salto viene sempre eseguito.

Sommario

Il ciclo di esecuzione di un'istruzione

L'ISA ed il linguaggio macchina

L'Assembly

Analisi dell'Instruction Set Architecture

I registri

I tipi di istruzioni: istruzioni aritmetiche

Istruzioni aritmetico-logiche

- In MIPS, un'istruzione aritmetico-logica possiede *tre* operandi: i due registri contenenti i valori da elaborare *(registri sorgente)* e il registro contenente il risultato *(registro destinazione).*
- L'ordine degli operandi è **fisso:** prima il registro contenente il risultato dell'operazione e poi i due operandi.
- L'istruzione assembly contiene il codice operativo e tre campi relativi ai tre operandi:

```
OPCODE  DEST,  SORG1, SORG2
```


Esempi: istruzioni `add` e `sub`

Codice C: `R = A + B`

Codice assembler MIPS:

```
add $s0,  $s1,  $s2
add rd,   rs,   rt
```

mette la somma del contenuto di rs e rt in rd:

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

Nella traduzione da linguaggio ad alto livello a linguaggio assembly, le variabili sono associate ai registri dal compilatore

sub serve per sottrarre il contenuto di due registri sorgente rs e rt:

```
sub rd rs rt
```

e mettere la differenza del contenuto di rs e rt in rd

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

Istruzioni aritmetico-logiche

Il fatto che ogni istruzione aritmetica ha tre operandi sempre nella stessa posizione consente di semplificare l'hw, ma complica alcune cose…

Codice C:

```
A = B + C + D
E = F – A
```

Codice MIPS:

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


Istruzioni aritmetico-logiche

- Operazioni con un numero di operandi maggiore di tre possono essere effettuate scomponendole in operazioni più semplici.
- Ad esempio, per eseguire la somma delle variabili b, c , d ed e nella variabile a servono tre istruzioni :

Codice C:

```
A = B + C + D + E
```

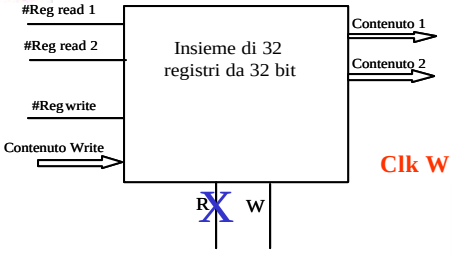
Codice MIPS:

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

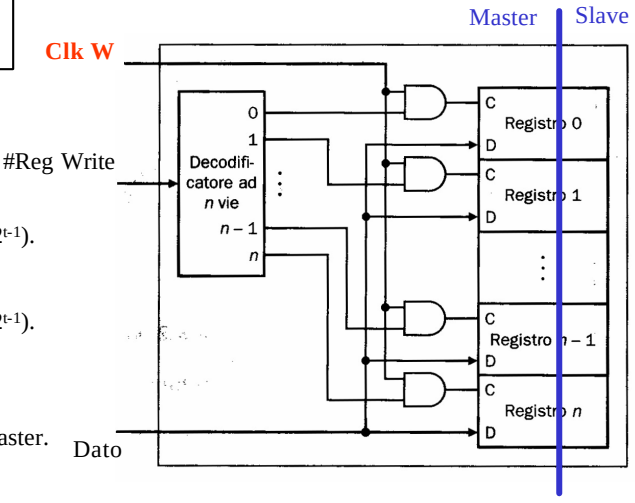
add: varianti

- **`addi $s1, $s2, 100`** **`#add immediate`**
  - Somma una costante: il valore del secondo operando è presente nell'istruzione come costante e sommata estesa in segno
- **`addu $s0, $s1, $s2`** **`#add unsigned`**
  - Evita overflow: la somma viene eseguita tra numeri senza segno
- **`addiu $s0, $s1, 100`** **`#add immediate unsigned`**
  - Somma una costante ed evita overflow.

Lettura / Scrittura nel Register file



Posso eseguire: add $s2, $s2, $t0



Clock Basso.
Leggo l'uscita dello Slave ($s2^{t-1}).

Clock Alto.
Leggo l'uscita dello Slave ($s2^{t-1}).
Scrivo sul Master ($s2^{t}) .

Clock Basso.
Trasferisco il contenuto del Master.
nello Slave.

Moltiplicazione

- Due istruzioni:
  - `mult rs rt`
  - `multu rs rt` `# unsigned`
- Il registro destinazione è ***implicito.***
- Il risultato della moltiplicazione viene posto sempre in due registri dedicati (special purpose) denominati ***hi (High order word)*** e ***lo (Low order word)***
- La moltiplicazione di due numeri rappresentabili con 32 bit può dare come risultato un numero non rappresentabile in 32 bit

Moltiplicazione

- Il risultato della moltiplicazione si preleva dal registro **hi** e dal registro **lo** utilizzando le due istruzioni:
  - `mfhi rd` `# move from hi`
    - Sposta il contenuto del registro **hi** nel registro **rd**
  - `mflo rd` `# move from lo`
    - Sposta il contenuto del registro **lo** nel registro **rd**

Pseudoistruzioni

- Per semplificare la programmazione, MIPS fornisce un insieme di *pseudoistruzioni*
- Le pseudoistruzioni sono un modo compatto ed intuitivo di specificare un insieme di istruzioni
  - Non hanno un corrispondente 1 a 1 con le istruzioni dell'ISA.
- La traduzione della pseudoistruzione nelle istruzioni equivalenti è attuata automaticamente dall'assemblatore

Esempi:

- **`move $t0, $t1`**
  - **`add $t0, $zero, $t1`**
- **`mul $s0, $t1, $t2`**
  - **`mult $t1, $t2`**
  - **`mflo $s0`**
- **`div  $s0, $t1, $t2`**
  - **`div $t1, $t2`**
  - **`mflo $s0`**

Sommario

Il ciclo di esecuzione di un'istruzione.

L'ISA ed il linguaggio macchina

L'Assembly

Analisi dell'Instruction Set Architecture

I registri

I tipi di istruzioni: istruzioni aritmetiche