Il linguaggio Assembly

Architettura
degli Elaboratori e delle Reti
Turno I

Alberto Borghese
Università degli Studi di Milano
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Linguaggio assembly

- 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 C: esempio

```
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: esempio

```
.text
.align 2
.globl main
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

……
```

Codifica binaria

- 00100111101111011111111111100000
- 10101111101111110000000000010100
- 10101111101001000000000000100000
- 10101111101001010000000000100100
- 10101111101001010000000000100100
- 10101111101001010000000000011000
- ……

Linguaggio assembly

- 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.

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

Assembly come linguaggio di programmazione

- Principali *vantaggi* della programmazione in linguaggio assembly:
  - Ottimizzazione delle prestazioni.
  - Massimo sfruttamento delle potenzialità dell'hardware sottostante.

Assembly come linguaggio di programmazione

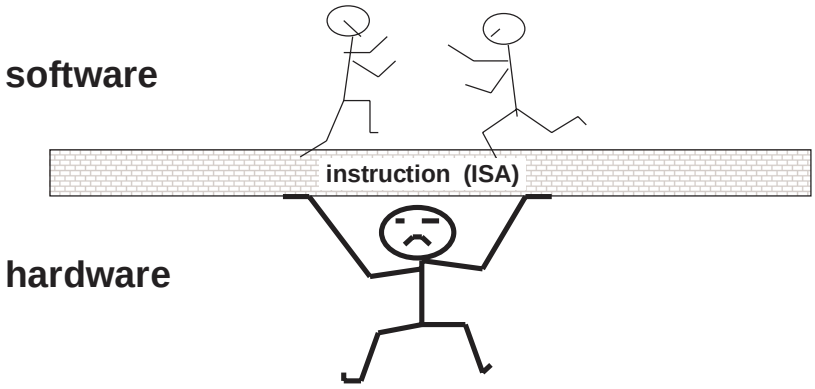
- Le strutture di controllo hanno forme limitate
- Non esistono tipi di dati all'infuori di interi, virgola mobile e caratteri.
- La gestione delle strutture dati e delle chiamate a procedura deve essere fatta in modo esplicito dal programmatore

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).

Insieme delle istruzioni



Quale è più facile modificare?

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)
```

Esempio di compilazione da C ad assembly (MIPS)

- Si consideri il seguente segmento di programma C che utilizza 5 variabili (f, g, h, i e j):

```
f = (g + h) - (i + j)
```

- Il compilatore associa ad un'istruzione C complessa delle istruzioni assembly a tre operandi e introduce due variabili temporanee (t0 e t1)

```
add t0, g, h        # var. temp t0 ⇐ g + h
add t1, i, j        # var. temp. t1 ⇐ i + j
sub f, t0, t1       # t2 ⇐ t0 - t1
```

- Le istruzioni assembly sono una rappresentazione simbolica del linguaggio macchina comprensibile dal processore MIPS.

Sommario

- Architettura di riferimento dei calcolatori
- Esecuzione delle istruzioni
- Il linguaggio assembly
- Il linguaggio macchina
- Insieme delle istruzioni
- Formato delle istruzioni
- Codifica delle istruzioni
- Modalità di indirizzamento

Linguaggio macchina

- Linguaggio di programmazione direttamente comprensibile dalla macchina
  - Alfabeto binario (alfabeto posizionale)
  - Parole sono le *istruzioni*
  - Vocabolario è *l'insieme delle istruzioni (instruction set)*

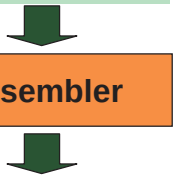
Linguaggio macchina

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

Fase di compilazione da assembly a linguaggio macchina

Programma in linguaggio assembly (MIPS)

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



Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```


Programma in linguaggio ad alto livello (C)

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

Programma in linguaggio assembly (MIPS)

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

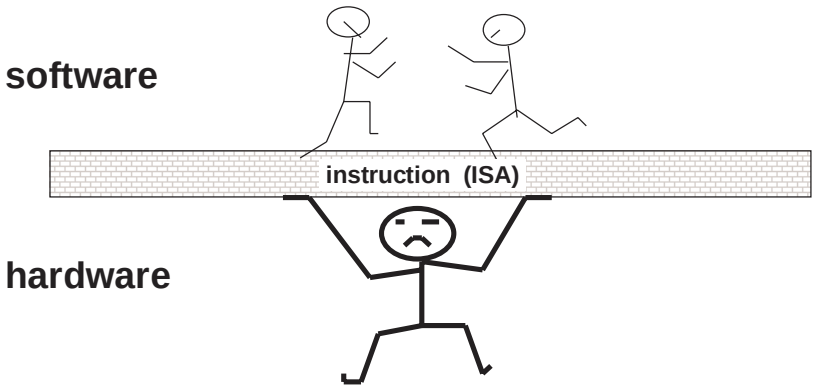
Programma in linguaggio macchina

```
011100010101010
000110101000111
000010000010000
001000100010000
```

Sommario

- Architettura di riferimento dei calcolatori
- Esecuzione delle istruzioni
- Il linguaggio assembly
- Il linguaggio macchina
- Insieme delle istruzioni (Innstruction Set Architecture, ISA)
- Formato delle istruzioni
- Codifica delle istruzioni
- Modalità di indirizzamento

Insieme delle istruzioni



Quale è più facile modificare?

Insieme delle istruzioni (Instruction Set)

- 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).

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)

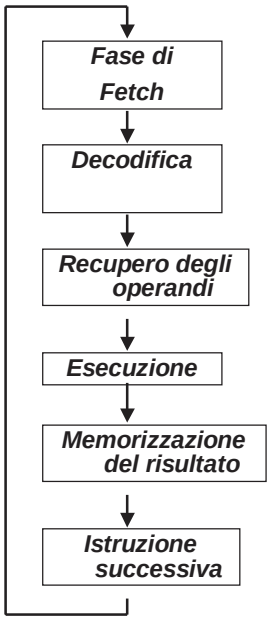
Istruzioni di salto condizionato e incondizionato

- 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

- Architettura di riferimento dei calcolatori
- Esecuzione delle istruzioni
- Il linguaggio assembly
- Il linguaggio macchina
- Insieme delle istruzioni (Instruction Set Architecture, ISA)
- Formato delle istruzioni
- Codifica delle istruzioni
- Modalità di indirizzamento

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

- Fetch-decodifica-esecuzione è implicita!

Architetture RISC

- Caratteristiche principali:
  - Istruzioni dell'ISA eseguite direttamente dall'hardware.
  - Massimizzare la frequenza di completamento dell'esecuzione delle istruzioni
  - Istruzioni facili da decodificare
    - "poche", poco "varie", tutte di uguale lunghezza
  - Accesso alla memoria solo attraverso istruzioni dedicate di caricamento/memorizzazione **(load/store)**
    - *Gli operandi di un'istruzione devono sempre risiedere nei registri del processore.*

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)**
- 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

- Esistono dei registri *special purpose (32 registri)*
  - per l'esecuzione di alcune operazioni
- 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, …`

Esempio di compilazione da C ad assembly usando registri

- Si consideri il seguente segmento di programma C che utilizza 5 variabili (f, g, h, i e j):

```
f = (g + h) – (i + j)
```

- Il compilatore associa alle variabili presenti nel programma i registri presenti nella CPU. Ad es. le variabili (f, g, h, i e j): sono associate ai registri **$s0, $s1, $s2, $s3** e **$s4**
- Il compilatore introduce due variabili temporanee (t0 e t1) che associa a due registri temporanei **$t0** e **$t1**

```
add $t0, $s1, $s2       # var. temp t0 ⇐ g + h
add $t1, $s3, $s4       # var. temp. t1 ⇐ i + j
sub $s0, $t0, $t1       # f ⇐ t0 – t1
```


Formato delle istruzioni MIPS

- 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
  - 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)**
  - Istruzioni aritmetico-logiche
- **Tipo I (immediate)**
  - Istruzioni di accesso alla memoria o contenenti delle costanti
- **Tipo J (jump)**
  - Istruzioni di salto

I tipi di istruzione

- Istruzioni aritmetico-logiche
- Istruzioni di trasferimento dati
- Istruzioni di salto

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
```

Esempio: istruzione add

add serve per sommare il contenuto di due registri sorgente rs e rt:

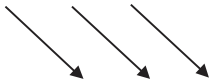
```
add rd, rs, rt
```

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

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


Esempio: istruzione add

Codice C: `R = A + B`



Codice assembler MIPS: `add $s0, $s1, $s2`

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

Esempio: istruzione sub

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.

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**

L'overflow verrà analizzato in seguito.

Divisione

- Due istruzioni:
  - **`div rs rt`** **`#divide rs per rt`**
  - **`divu rs rt`** **`#unsigned`**
- Come nella moltiplicazione, anche nella divisione il registro destinazione è ***implicito.***
- Il quoziente della divisione è posto nel registro ***lo,*** mentre il resto è posto nel registro ***Hi***

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

Esempio

- `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`