Architettura degli Elaboratori

MIPS:

Prime istruzioni e organizzazione dei registri

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

Punto della situazione

- Abbiamo studiato
  - Il linguaggio dei calcolatori
  - L'algebra booleana, le reti logiche e la loro minimizzazione
  - Alcuni componenti di base utilizzati nei calcolatori, tra cui l'ALU

- Obiettivo di oggi
  - Studio dell'organizzazione generale di un calcolatore
  - Studio delle istruzioni che possiamo impartire al calcolatore nel suo linguaggio
  - Prenderemo come riferimento il processore MIPS

Un moderno elaboratore

Sistema elettronico digitale programmabile

Sistema:
Costituito da componenti che interagiscono in modo organico fra loro

Elettronico digitale:
Sfrutta componenti elettronici digitali

Programmabile:
Il comportamento del sistema è flessibile e specificato mediante un programma, ossia un insieme di ordini

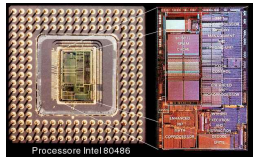
Componenti

Le componenti di un elaboratore si dividono in cinque categorie


- Dispositivi di input
- Dispositivi di output

- Memoria

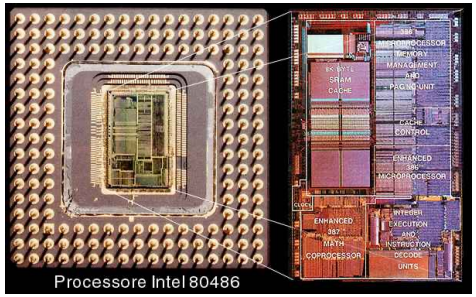
- Unità di elaborazione dati
- Unità di controllo



Le ultime due componenti sono spesso unificate in un'unica componente: il processore

Processore

- È il cuore di un elaboratore elettronico
- Si compone di
  - Unità di elaborazione dati
  - Unità di controllo



- Realizzato con milioni di piccoli componenti elementari (transistor)
- È impossibile da studiare e da capire partendo dal singolo transistor
  - Abbiamo necessità di astrazione (tralasciare dettagli non necessari)

Processore MIPS

- Studieremo una versione semplificata del processore MIPS
  - Microprocessor without Interlocked Pipeline Stages
  - Architettura RISC (Reduced Instruction Set Computer), proposta nel 1981 da MIPS Computer Systems Inc.

- Perché questa scelta?
  - Si tratta di un processore realmente utilizzato
    - Nintendo64, Sony PS, Sony PS2, Sony PSP
  - La sua architettura e il suo set di istruzioni sono molto semplici

Set di Istruzioni

- Per impartire istruzioni a un calcolatore è necessario parlare il suo linguaggio
- Il linguaggio si compone di un insieme di istruzioni (IS: Instruction Set), corrispondenti a sequenze binarie
- Ciascun tipo di processore ha il suo IS
  - Noi studieremo il linguaggio macchina del MIPS
  - Le differenze tra vari IS non sono eccessive: è come se fossero diversi "dialetti" di uno stesso linguaggio
- I programmatori però preferiscono utilizzare linguaggi ad alto livello per scrivere i loro programmi

Traduzione

- Passare da un'applicazione scritta in un linguaggio ad alto livello alla sua concreta esecuzione è un processo complesso
- Vediamo come funziona questo processo di traduzione

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
     int i;
     int sum = 0;

     for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
     printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Linguaggio ad alto livello (C)

Linguaggio macchina MIPS

Traduzione

- Il software che si occupa della traduzione di un programma scritto in un linguaggio ad alto livello in un programma equivalente in linguaggio macchina viene detto compilatore

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
     int i;
     int sum = 0;

     for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
     printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Compilatore

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111110010000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Linguaggio ad alto livello (C)

Linguaggio macchina MIPS

Il ruolo del Compilatore

- Spesso, il compilatore traduce il programma ad alto livello in uno equivalente scritto in linguaggio assembly
  - Si tratta di un linguaggio più leggibile del linguaggio macchina, perché usa simboli anziché sequenze binarie

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
     int i;
     int sum = 0;

     for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
     printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Compilatore

```
addiu       $29, $29, -32
sw          $31, 20($29)
sw          $4,  32($29)
sw          $5,  36($29)
sw          $0,  24($29)
sw          $0,  28($29)
lw          $14, 28($29)
lw          $24, 24($29)
multu       $14, $14
addiu       $8,  $14, 1
slti        $1,  $8, 101
sw          $8,  28($29)
mflo        $15
addu        $25, $24, $15
bne         $1,  $0, -9
sw          $25, 24($29)
lui         $4,  4096
lw          $5,  24($29)
jal         1048812
addiu       $4,  $4, 1072
lw          $31, 20($29)
addiu       $29, $29, 32
jr          $31
move        $2,  $0
```

Linguaggio ad alto livello (C)

Linguaggio assembly MIPS

Traduzione

- Successivamente, è necessaria una traduzione del programma da linguaggio assembly a linguaggio macchina
  - Il software che si occupa di questa cosa è l'assemblatore

```
addiu       $29, $29, -32
sw          $31, 20($29)
sw          $4,  32($29)
sw          $5,  36($29)
sw          $0,  24($29)
sw          $0,  28($29)
lw          $14, 28($29)
lw          $24, 24($29)
multu       $14, $14
addiu       $8,  $14, 1
slti        $1,  $8, 101
sw          $8,  28($29)
mflo        $15
addu        $25, $24, $15
bne         $1,  $0, -9
sw          $25, 24($29)
lui         $4,  4096
lw          $5,  24($29)
jal         1048812
addiu       $4,  $4, 1072
lw          $31, 20($29)
addiu       $29, $29, 32
jr          $31
move        $2,  $0
```

Linguaggio assembly MIPS

Assemblatore

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Linguaggio macchina MIPS

Il ruolo dell'Assemblatore

- Un assemblatore legge un singolo file sorgente in linguaggio assembly e produce un file oggetto contenente le istruzioni in linguaggio macchina

- Esistono diversi tipi di linguaggi assembly e di conseguenza diversi assembler:
  - Per programmare i microchip, per i Personal Computer, per telefoni cellulari, ecc.
  - Un assembler produce codice macchina per una specifica famiglia di processori (intel 8086, 80386, Motorola 68000, ecc.).

Il flusso completo

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
     int i;
     int sum = 0;

     for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
     printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Compilatore

```
addiu       $29, $29, -32
sw          $31, 20($29)
sw          $4,  32($29)
sw          $5,  36($29)
sw          $0,  24($29)
sw          $0,  28($29)
lw          $14, 28($29)
lw          $24, 24($29)
multu       $14, $14
addiu       $8,  $14, 1
slti        $1,  $8, 101
sw          $8,  28($29)
mflo        $15
addu        $25, $24, $15
bne         $1,  $0, -9
sw          $25, 24($29)
lui         $4,  4096
lw          $5,  24($29)
jal         1048812
addiu       $4,  $4, 1072
lw          $31, 20($29)
addiu       $29, $29, 32
jr          $31
move        $2,  $0
```

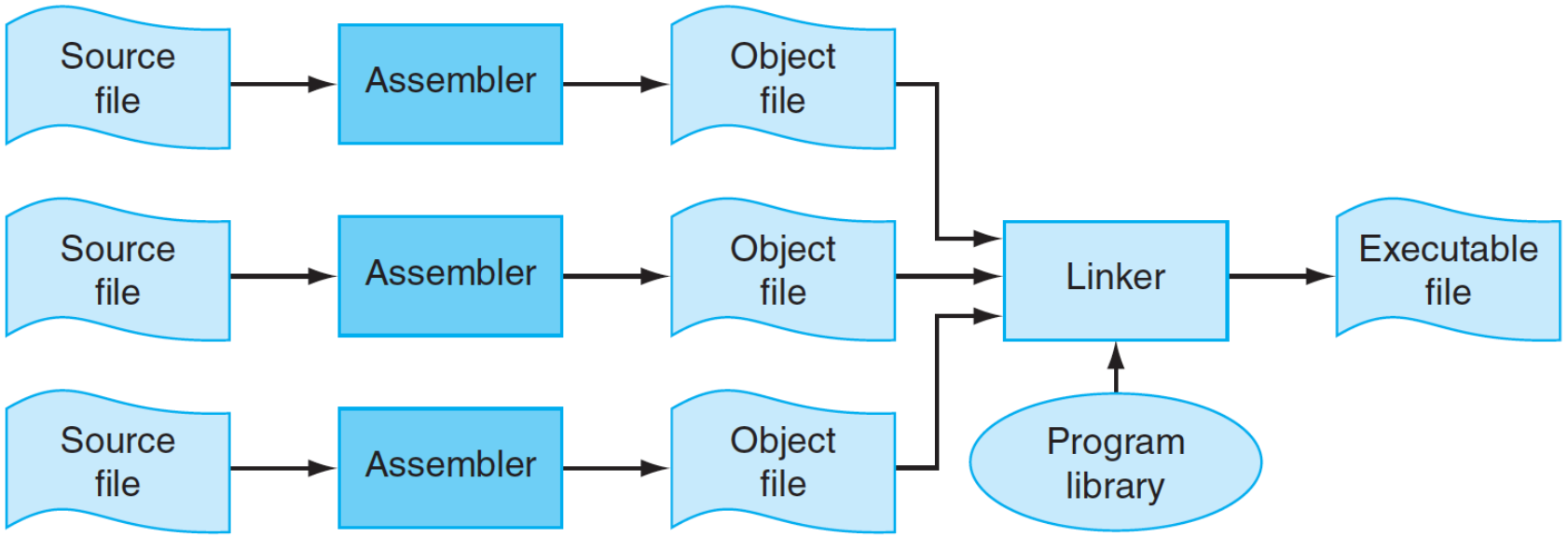
Assemblatore

```
00100111101111101111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00101011100100000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Il ruolo del Linker

- La maggior parte dei programmi
  - Consiste di diversi file, detti anche moduli, che vengono scritti, compilati e assemblati separatamente
  - Può utilizzare procedure scritte in precedenza, appartenenti ad una libreria
- I file oggetto corrispondenti ai diversi moduli e i file di libreria devono essere "collegati" per creare un singolo file eseguibile
- Di questo compito si occupa il Linker (o link editor)

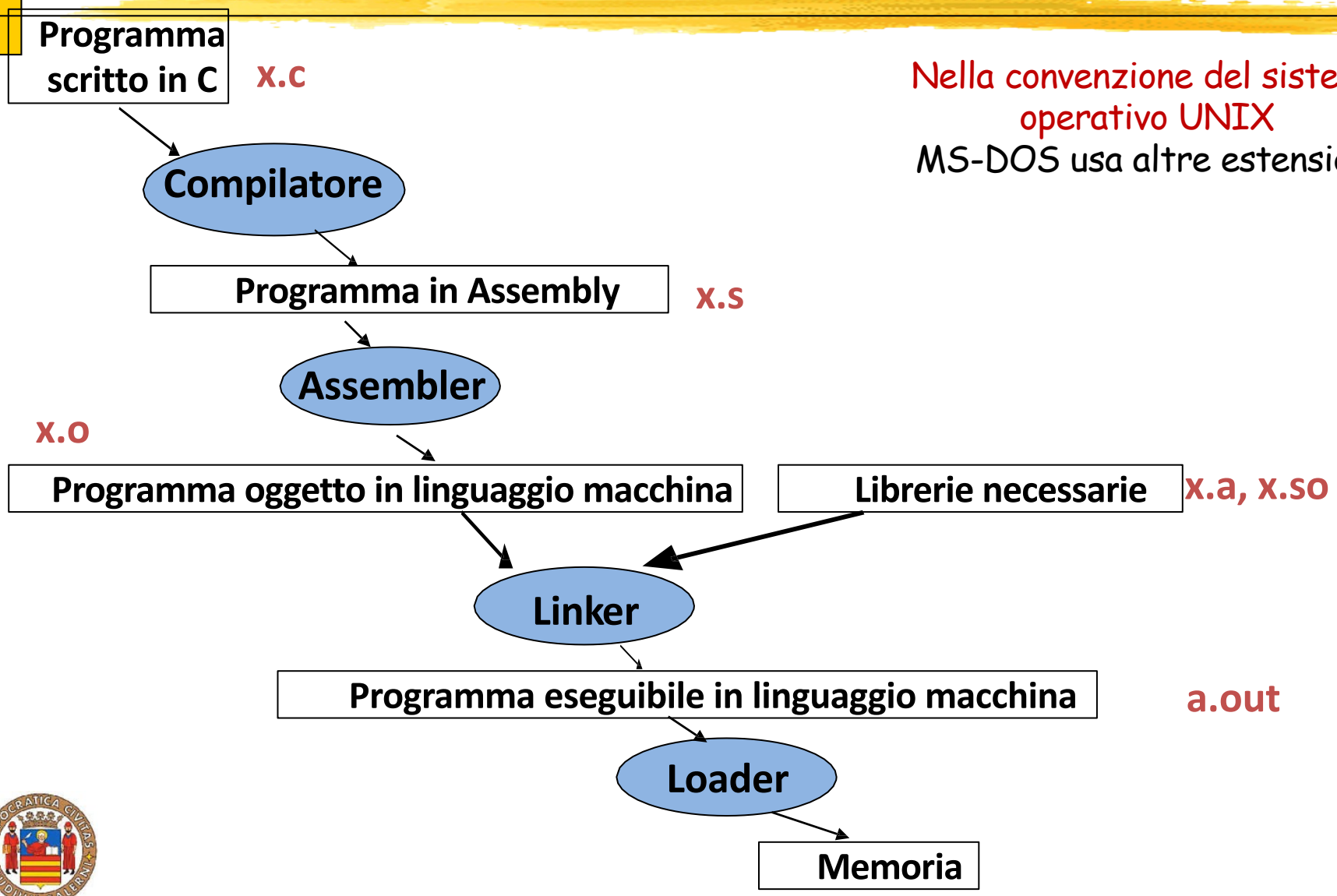
Il ruolo del Linker



Il ruolo del Loader

- Un programma il cui linking si conclude senza errori produce un file eseguibile, pronto per essere eseguito
- E' necessario individuare un'area nella memoria principale in cui caricare (to load) il programma per poi lanciarne l'esecuzione
- Di questo compito si occupa il Loader

Tradurre e avviare un programma



Nella convenzione del sistema operativo UNIX
MS-DOS usa altre estensioni

Assembly del MIPS

- Abbiamo quindi capito che ciascuna istruzione nel linguaggio macchina del MIPS è composta da una sequenza binaria
- Per semplificare l'apprendimento dell'insieme delle istruzioni del MIPS, studieremo il linguaggio assembly MIPS
  - Iniziamo dalle istruzioni aritmetiche
  - Poi passiamo alle istruzioni di trasferimento dati

Istruzioni aritmetiche

- Il formato delle istruzioni aritmetiche nell'IS del MIPS è rigido e prevede solo istruzioni a tre operandi
  - Il primo operando corrisponde alla destinazione del risultato dell'operazione tra il secondo e il terzo operando
- Addizione: add a, b, c
  - Corrisponde ad a = b + c
- Sottrazione: sub a, b, c
  - Corrisponde ad a = b – c
- Perché questa scelta?
  - Principio di Progettazione n. 1:
    La semplicità favorisce la regolarità

Istruzioni aritmetiche

- Per effettuare operazioni tra più di due operandi bisogna utilizzare più istruzioni
  - Ad esempio, l'operazione d = b + c – e si può realizzare mediante due istruzioni:

```
add a, b, c
sub d, a, e
```

  - L'operazione f = (g + h) – (i + j) si può realizzare mediante tre istruzioni:

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

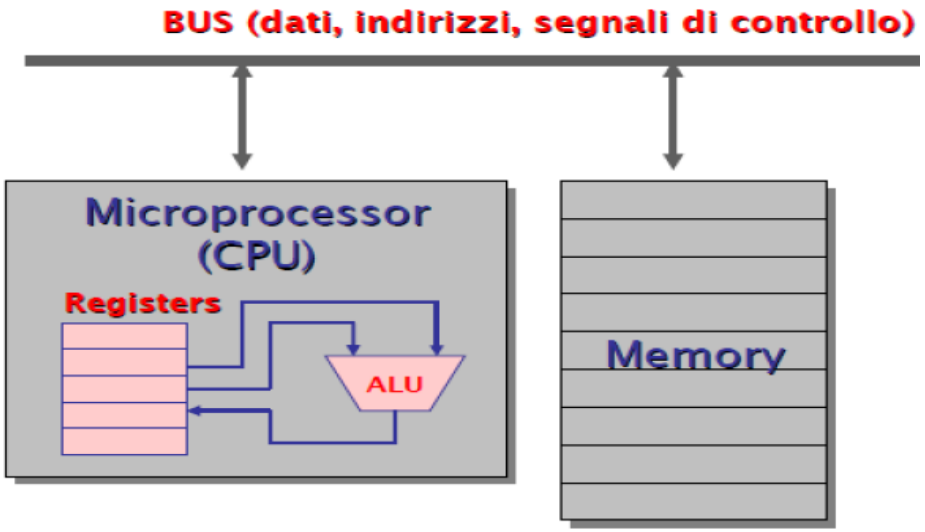
Commenti

- Possiamo inserire un commento a un'istruzione usando il carattere #
  - Il commento inizia da # e termina alla fine della linea
- Esempio

```
add t0, g, h #la variabile temporanea t0 contiene g+h
add t1, i, j #la variabile temporanea t1 contiene i+j
sub f, t0, t1 #la variabile temporanea f contiene
                #(g+h) – (i+j)
```

Registri del MIPS

- Nelle istruzioni del MIPS, gli operandi sono vincolati ad essere scelti tra un numero limitato di locazioni di memoria interne al processore: i registri
  - Si può accedere a un registro in un solo ciclo di clock!



Registri

- Il MIPS contiene 32 registri a 32 bit

- Perché solo 32 registri?
  - Principio di Progettazione n. 2:
    Minori sono le dimensioni, maggiore è la velocità
  - Un numero elevato di registri obbligherebbe i segnali elettrici ad impiegare tempo maggiore a spostarsi

Registri

- Ciascun registro ha un nome preceduto dal simbolo $
- Per convenzione, alcuni dei 32 registri hanno dei nomi particolari, associati al loro utilizzo:
  - $zero contiene sempre la costante 0
  - $at riservato all'assemblatore per la gestione di costanti molto lunghe
  - $v0, $v1 utilizzati per valutare le espressioni
  - $a0,…,$a3 utilizzati per gli argomenti delle operazioni
  - $t0,…,$t9 utilizzati per le variabili temporanee
  - $s0…$s7 utilizzati per memorizzare variabili
  - $k0, $k1 riservati al sistema operativo
  - $gp, $fp, $sp, $ra …altri utilizzi

Registri

- Tornando all'esempio precedente, come viene eseguita l'operazione f = (g + h) – (i + j)?
- I valori sono assegnati ai registri:
  - $s0 contiene f
  - $s1 contiene g
  - $s2 contiene h
  - $s3 contiene i
  - $s4 contiene j
- E le istruzioni MIPS sono le seguenti:

```
add $t0, $s1, $s2 #il registro temporaneo $t0 contiene g+h
add $t1, $s3, $s4 #il registro temporaneo $t1 contiene i+j
sub $s0, $t0, $t1 #f assume il valore (g+h) – (i+j)
```

Operandi immediati

- Spesso nelle operazioni aritmetiche uno degli operandi è una costante
- Per evitare di allocare memoria per ciascuna costante, il MIPS fornisce istruzioni che operano con costanti
  - Principio di Progettazione n. 3:
    Rendi le operazioni comuni il più veloce possibile
- Esempio: all'operazione f = f + 4 corrisponde l'istruzione

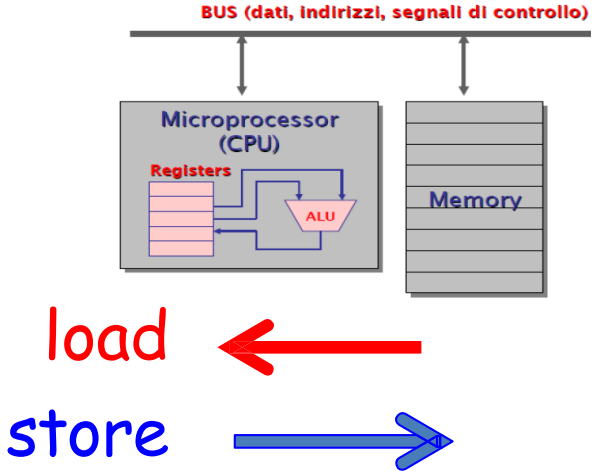
```
addi $s3, $s3, 4 #il contenuto del registro $s3,
                 #cioè f, viene incrementato di 4
```

immediate

Nota: la costante può anche essere negativa

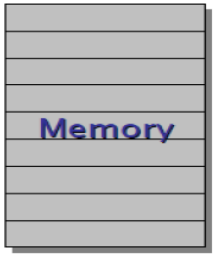
Istruzioni di trasferimento

- Le operazioni aritmetiche si effettuano solo tra registri, ma i registri sono pochi!
- Occorrono istruzioni di trasferimento tra la memoria e i registri:
  - Load: preleva il contenuto di una locazione di memoria e lo trasferisce in un registro
  - Store: salva il contenuto di un registro in una locazione di memoria



Memoria

- E' una sequenza (array) di 2^{32}=4.294.967.292 celle
  - Memoria[0], Memoria[1],….,Memoria[2^{32}-1]
  - Ciascuna cella può contenere 8 bit (1 byte)

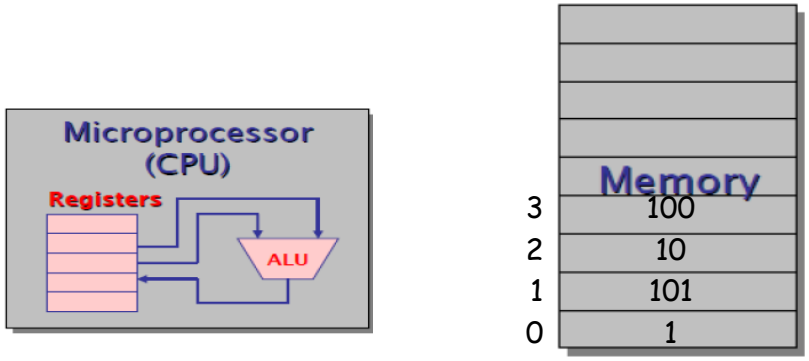


- Come memorizzare parole di 32 bit?
  - Ciascuna parola necessita di 4 locazioni consecutive
    - La prima parola andrà in Memoria[0]Memoria[1]Memoria[2]Memoria[3]
    - La seconda parola andrà in Memoria[4]Memoria[5]Memoria[6]Memoria[7]
    - La terza parola andrà in Memoria [8]….Memoria[11]

- Il numero di parole a 32 bit che si possono memorizzare quindi è $2^{32}:4 = 2^{32}:2^2 = 2^{30}$

Memoria

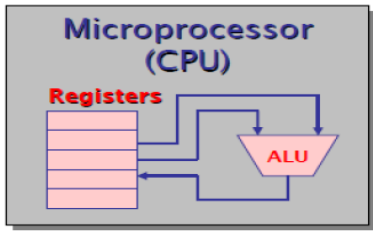
- Quindi la memoria è una sequenza di gruppi di byte
- Ciascun byte è assegnato a un indirizzo lineare e progressivo tramite cui può essere prelevato

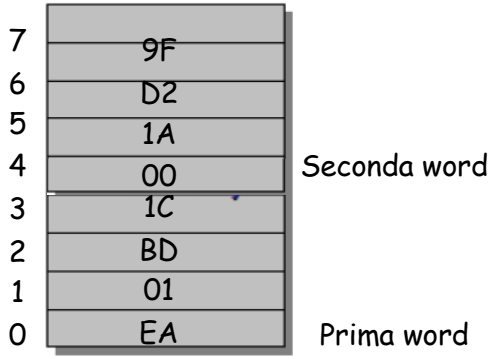


- Esempio:
  - Memoria[0]=1, Memoria[1]=101, Memoria[2]=10, Memoria[3]=100,…

Word

- Una word è costituita da 32 bit = 4 byte = 8 cifre esadecimali e quindi occupa 4 locazioni di memoria
- Per passare da una word in memoria alla successiva bisogna incrementare l'indirizzo della cella di 4





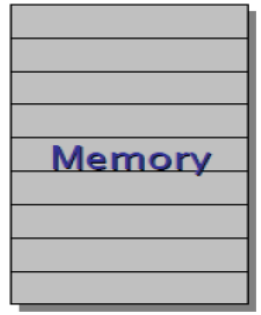
- E' possibile accedere solo a parole poste ad indirizzi multipli dell'ampiezza della parola
  - Prima word: EA 01 BD 1C
  - Seconda word: 00 1A D2 9F

Trasferimento: Load

- Per caricare in un registro una word contenuta in memoria bisogna specificare l'indirizzo della cella in cui la word è memorizzata
- Nel MIPS l'indirizzo si specifica mediante
  - Un registro di base
  - Uno spiazzamento costante (offset)
- L'istruzione

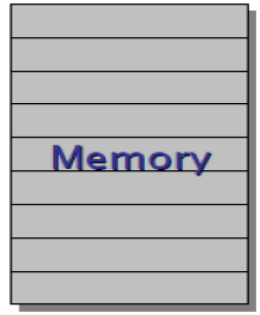
  lw $t0, 4($s3)

  carica nel registro $t0 la word che si trova in memoria all'indirizzo dato dal contenuto del registro $s3 a cui va sommato 4
  - Base: registro $s3
  - Offset: 4



Trasferimento: Store

- Per registrare in memoria una word contenuta in un registro bisogna specificare l'indirizzo della cella in cui memorizzare la word



- L'istruzione

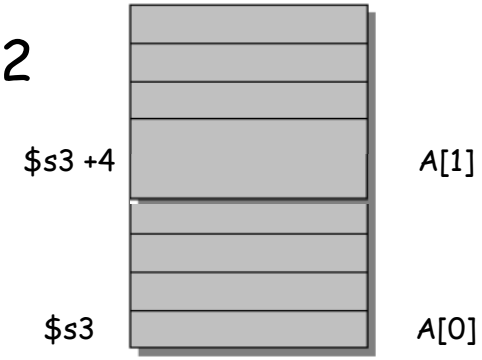
  sw $t0, 32($s3)

  registra la word contenuta nel registro $t0 nella locazione di memoria il cui indirizzo è dato dal contenuto del registro $s3 a cui va sommato 32
  - Base: registro $s3
  - Offset: 32

Esempio

- Supponiamo di avere
  - Un array A di 100 word memorizzate a partire dall'indirizzo base contenuto nel registro $s3
  - Il valore h memorizzato nel registro $s2
- Per effettuare l'istruzione

  A[12] = h + A[8]



il codice MIPS è

```
lw $t0, 32($s3)#carica nel registro $t0 il contenuto
                #della locazione di memoria $s3+32, cioè A[8]
add $t0, $s2, $t0 #somma al contenuto di $t0, cioè A[8], il
                  #contenuto
                  #di $s2, cioè h, e lo memorizza in $t0
sw $t0, 48($s3)#registra il contenuto di $t0, cioè A[8]+h, nella
               #locazione $s3+48, cioè A[12]
```

Riepilogo e riferimenti

- Organizzazione generale di un calcolatore
  - [PH] Par 1.3, 1.4, Appendice A (da A.1 ad A.4)
- Prime istruzioni del MIPS: add, sub
  - [PH] Parr. 2.1, 2.2
- Organizzazione dei registri e della memoria; Istruzioni di trasferimento
  - [PH] par. 2.3