LE ISTRUZIONI: IL LINGUAGGIO DEI CALCOLATORI

Sarro Federica, Phd Student
Facoltà di Scienze MM FF NN
Università degli Studi di Salerno
fsarro@unisa.it
http://www.dmi.unisa.it/people/sarro/www/

Attività tutoria per il Corso di Architettura degli Elaboratori – Classi 1, 2, 3
Anno Accademico 2011-2012

Presentazione

- Attività di tutorato per i corsi di
  - Architettura degli Elaboratori - Classi 1,2,3
    - http://www.dmi.unisa.it/people/sarro/www/tarch1112.html
  - Programmazione 1 - Classi 1,2,3
    - http://www.dmi.unisa.it/people/sarro/www/tlp1112.html
- Esercitazioni in aula/laboratorio
  - fisseremo un giorno in base alle vostre esigenze
    - …e alle disponibilità delle aule!
  - principalmente esercitazioni su linguaggio assembler
    - …altre esigenze?
- Ricevimento
  - inviate una e-mail a fsarro@unisa.it con oggetto [Tutorato Arch] o [Tutorato Prog1]

Sommario

- La programmazione dei calcolatori
  - Programmare a diversi livelli di astrazione
  - Tradurre e avviare un programma
- Il set di istruzioni MIPS
  - Architettura MIPS
  - Operandi e Registri
  - Il set di istruzioni
    - Operazioni aritmetiche e logiche
    - Operazioni di trasferimento dati
    - Istruzioni di controllo
      - Realizzazione di cicli
- Dal linguaggio assembler al linguaggio macchina
  - Rappresentazione delle istruzioni
    - Formato R, Formato I, Formato J

Programmare a diversi livelli di astrazione

```
A[300]= h+A[300];
```

COMPILATORE

```
lw  $t0, 1200($t1)
add $t0, $s2, $t0
sw $t0, 1200($t1)
```

ASSEMBLATORE

```
10001101001010000000
10010110000000000001001
00100001000000000100000
01010110100101000000
010010110000
```

Linguaggio C

- Linguaggio di alto livello
  - notazione naturale vicina al linguaggio corrente e alla notazione algebrica
- Incremento di produttività
- Indipendenza dalla architettura (processore)
- Riuso del codice (e.g., uso di librerie di funzionalità già scritte)

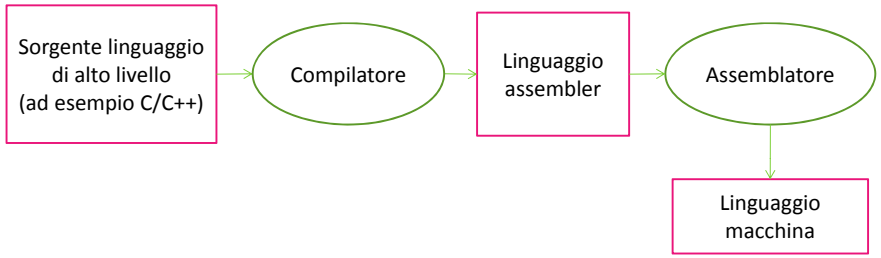
Linguaggio Assembler

- Linguaggio di basso livello
- Forma simbolica dell'istruzione macchina
  - Evita di programmare con sequenze di bit
  - Parole chiavi (codice mnemonico) al posto di linguaggio macchina (codice operativo)
  - Indirizzi di memoria indicati da identificatori testuali al posto di indirizzi binari
- Dipendenza dall'hw
  - programmi poco portabili ma più efficienti

Linguaggio Macchina

- Scrivere un programma in linguaggio macchina è un'attività tediosa e lunga
- È molto facile fare un errore
- Il codice risulta poco leggibile
- E' difficile correggere gli errori

Tradurre e avviare un programma



- Compilatore:programma che traduce istruzioni scritte in linguaggio di alto livello in linguaggio assembler
- Assemblatore: programma che traduce la versione simbolica di un istruzione in versione binaria

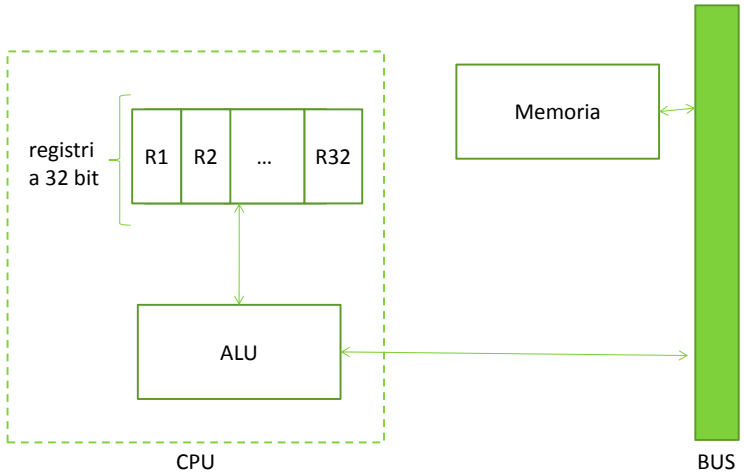
Sommario

- La programmazione dei calcolatori
  - Programmare a diversi livelli di astrazione
  - Tradurre e avviare un programma
- Il set di istruzioni MIPS
  - Architettura MIPS
  - Operandi e Registri
  - Il set di istruzioni
    - Operazioni aritmetiche e logiche
    - Operazioni di trasferimento dati
    - Istruzioni di controllo
      - Realizzazione di cicli
- Dal linguaggio assembler al linguaggio macchina
  - Rappresentazione delle istruzioni
    - Formato R, Formato I, Formato J

Architettura MIPS (1)

- Il processore MIPS
  - progettato negli anni ‘80 (Stanford)
  - prodotto e distribuito da MIPS Technologies negli anni ’90
  - usato da NEC, Nintendo, Silicon Graphics, Sony, …
- Reduced Instruction Set Computer (RISC)
  - insieme di istruzioni semplice
  - istruzioni a lunghezza fissa
  - utilizzo di più registri
- Ci permette di apprezzare i principi usati nella progettazione di tutte le ISA moderne
- MIPS32: architettura di esempio per questo corso

Architettura MIPS (2)



Operandi e registri (1)

- Nei linguaggi ad alto livello non ci si preoccupa
  - del numero e del tipo di variabili
  - della quantità di memoria utilizzata
  - di dover portare i dati dalla memoria ai registri e viceversa
    - è compito del compilatore inserire le istruzioni necessarie
- Nel linguaggio assembler a differenza dei linguaggi ad alto livello
  - gli operandi devono provenire da locazioni particolari di memoria dette registri
    - è compito del programmatore spostare i dati tra memoria e registri

Operandi e registri (2)

- Caratteristiche dei registri
  - alta velocità di accesso
  - limitati in numero (32 registri)
    - per creare un indice ho bisogno di soli 5 bit
  - ogni registro è grande 32 bit (parola=word)
- Convenzione di rappresentazione MIPS
  - registri che contengono variabili: $s0, $s1, …
  - registri di uso temporaneo: $t0, $t1, …
  - il registro $zero contiene sempre il valore 0
  - nelle esercitazioni indicheremo i registri i maniera generica
    - $1, $2, …

Il set di istruzioni

- Istruzioni aritmetiche e logiche
  - effettuano operazioni tra dati presenti nei registri
- Istruzioni di trasferimento dati
  - spostano i dati tra memoria e registri
- Istruzioni di controllo
  - salti condizionati e incondizionati

N.B. ogni linea contiene un'istruzione

i commenti sono preceduti da #

Il set di istruzioni

- Istruzioni aritmetiche e logiche
  - effettuano operazioni tra dati presenti nei registri
- Istruzioni di trasferimento dati
  - spostano i dati tra memoria e registri
- Istruzioni di controllo
  - salti condizionati e incondizionati

Istruzioni aritmetiche: somma e sottrazione

- Istruzioni a tre operandi
  - gli operandi devono essere memorizzati nei registri e non in variabili presenti in memoria
- Istruzione add
  - Semantica: op1 ← op2 + op3
  - Sintassi
    - add $1, $2, $3 # memorizzo nel registro $1la somma dei valori contenuti nei registri $2 e $3
- Istruzione sub
  - Semantica: op1 ← op2 - op3
  - Sintassi
    - sub $1, $2, $3 # memorizzo nel registro $1 la differenza dei valori contenuti nei registri $2 e $3

Istruzioni aritmetiche: esempio (1)

- Scrivere le istruzioni MIPS necessarie per
  - copiare un valore da un registro ad un altro
    - equivale all'assegnamento a=b
  - azzerare un registro
    - equivale all'assegnamento a=0
  - sia a in $2 e b in $3

```
add $2, $3, $zero       # a=b+0
```

```
add $2, $zero, $zero    # a=0+0
```


Istruzioni aritmetiche: esempio (2)

- Scrivere il codice MIPS relativo all'istruzione C

$$a = (b+c) – (d-f) + g$$

  - supponendo che alle variabili a, b, c, d, e, f, g siano assegnati, rispettivamente i registri $1, $2, $3, $4, $5, $6, $7

```
add $8, $2, $3      # memorizzo b+c in $8
sub $9, $4, $6      # memorizzo d-f in $9
sub $8, $8, $9      # memorizzo $8+$9 in $8
add $1, $8, $7      # memorizzo $8+g in a
```


Istruzioni aritmetiche: somma con operando immediato

- Add immediate
  - istruzione a tre operandi
    - 2 operandi memorizzati nei registri e uno "immediato"
- Semantica s1 ← s2 + 3
- Sintassi
  - addi $1, $2, 3 # $1 = $2+3

Istruzioni logiche: and e or

- Istruzioni a tre operandi
  - gli operandi devono essere memorizzati nei registri e non in variabili presenti in memoria
- Istruzione and
  - Semantica: op1 ← op2 & op3
  - Sintassi
    - and $1, $2, $3 # memorizzo nel registro $1 l'AND logico bit a bit dei valori contenuti nei registri $2 e $3
  - Utile per applicare una maschera
- Istruzione or
  - Semantica: op1 ← op2 | op3
  - Sintassi
    - or $1, $2, $3 # memorizzo nel registro $1 l'OR logico bit a bit dei valori contenuti nei registri $2 e $3
  - Utile per porre determinati bit a 1

Il set di istruzioni

- Istruzioni aritmetiche e logiche
  - effettuano operazioni tra dati presenti nei registri
- Istruzioni di trasferimento dati
  - spostano i dati tra memoria e registri
- Istruzioni di controllo
  - salti condizionati e incondizionati

Istruzioni di trasferimento dati: necessità

- I linguaggi di programmazione utilizzano variabili semplici e strutture complesse (e.g., vettori)
  - il numero di elementi può essere molto maggiore del numero dei registri presenti nel calcolatore
- Abbiamo a disposizione un numero finito di registri a 32 bit (i.e., 32 in MIPS) ma le variabili contenute in memoria possono essere un numero infinito (molto grande)
- Dato che le istruzioni macchina possono operare solo sugli operandi contenuti nei registri, i dati contenuti nelle variabili in memoria devono essere spostate nei registri
  - con i linguaggi ad alto livello è compito del compilatore assegnare i registri alle variabili
  - con il linguaggio assembler è compito del programmatore spostare i dati dalla memoria ai registri e viceversa

Istruzioni di trasferimento dati: Load e Store (1)

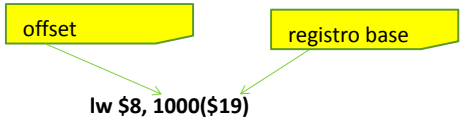
- Trasferiscono i dati dalla memoria ai registri e viceversa
  - Load word
    - permette di trasferire una parola di memoria (32 bit) in un registro
  - Store word
    - permette di trasferire il valore contenuto in un registro in una parola di memoria (32 bit)

Istruzioni di trasferimento dati: Load e Store (2)

- Istruzione lw
  - Semantica: rt ← M[rs + offset]
    - rt e rs sono registri, offset è una costante specificata nell'istruzione, M è la memoria vista come un vettore
  - Sintassi
    - lw $1, offset ($2) # trasferisce nel registro $1 la parola di memoria memorizzata all'indirizzo offset +(contenuto di $2)
- Istruzione sw
  - Semantica: M [rs+ offset] ← rt
    - rt e rs sono registri, offset è una costante specificata nell'istruzione, M è la memoria vista come un vettore
  - Sintassi
    - sw $1, offset ($2) # trasferisce il valore contenuto nel registro $1 all'indirizzo di memoria offset +(contenuto di $2)

Indirizzamento indicizzato

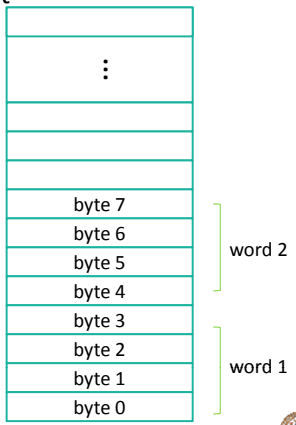
- Prevede l'uso di un indirizzo base
  - contenuto nel "registro base"
- … e di uno "spiazzamento" (o offset)
  - specificato nell'istruzione tramite una costante
- Es. se $19 contiene 12 nella seguente istruzione



l'indirizzo della locazione di memoria caricata è 1000 + (contenuto di $19) = 1012

Indirizzamento indicizzato: memoria

- La memoria è indirizzata al singolo byte (8bit)
  - ogni locazione ha un indirizzo di 32 bit
- Una parola è composta da 4 byte
  - pertanto occupa 4 locazioni
    - l'indirizzo della parola è l'indirizzo della prima locazione
    - gli indirizzi delle parole sono multipli di 4 (in una parola ci sono 4 byte)



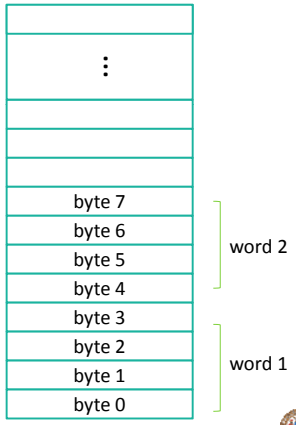
Indirizzamento indicizzato: array

- Le strutture dati (e.g.,array) dei linguaggi ad alto livello sono allocate in memoria
- Consideriamo un vettore A
  - ogni elemento del vettore è memorizzato in una parola
- Per calcolare l'indirizzo di memoria *I* dell'elemento A[i] è sufficiente conoscere l'indirizzo del primo elemento del vettore (A[0]) detto "indirizzo di partenza"

***I = indirizzo di partenza* + (4 x i)**

Indirizzamento indicizzato: esempio

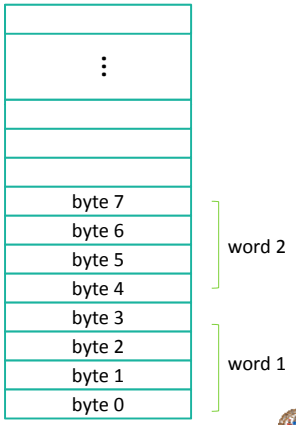
- Caricare A[6] in un registro
  - offset + contenuto del registro base



Indirizzamento indicizzato: esempio

- Caricare A[6] in un registro
  - offset + contenuto del registro base

```
lw $1, 6($2)
```

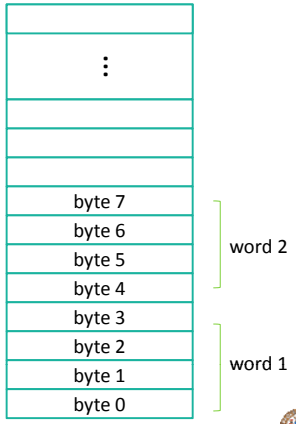


Indirizzamento indicizzato: esempio

- Caricare A[6] in un registro
  - offset + contenuto del registro base

~~lw $1, 6($2)~~

```
lw $1, 24($2)
```



Esempio 1

- Scrivere il codice MIPS relativo alla istruzione C

  A[12]= h + A[8]

- Supponendo che
  - la variabile h sia associata al registro $2
  - l'indirizzo di base del vettore A sia nel registro $3
- Soluzione

```
lw    $1, 32($3)        # carica A[8] in $1
add   $1, $2, $1        # somma h (in $2) con A[8] (in $1)
sw    $1, 48($3)        # memorizza $1 in A[12]
```


Esempio 2

- Scrivere il codice MIPS relativo alla istruzione C

  A[i] = j ;

  - l'offset non è costante, ma indicato da i
    - se i è memorizzato in un registro variando il suo contenuto possiamo scorrere l'array
      - basta incrementare il registro di 4 in 4

Esempio 2

- Supponiamo che l'indirizzo base di A sia in $1, che i sia in $2 e j sia in $3
  - calcolo l'offset relativo ad i

```
add  $4, $2, $2    # $4 = i+i =2i
add  $4, $4, $4    # $4 = 2i+2i = 4i
```

  - calcolo indirizzo di A[i]

```
add  $4, $4, $1    # $4= offset + indirizzo base
```

  - memorizzo j in A[i]

```
sw $2, 0($4)
```


Esercizi

1. Scrivere un programma che assumendo di avere dei valori nei registri $1, $2 e $3 scriva nella parola con indirizzo 1000 la somma dei tre registri
2. Scrivere un programma che assumendo di avere nel registro $1 il valore 10, nel registro $2 il valore 4, nel registro $3 il valore 3, scriva nelle parole con indirizzo 1000, 1004, 1008 rispettivamente il valore di 10, 13, 16
3. Come si realizza l'istruzione C g= h +A[i] in assembler?

Svolgimento dell'esercizio 3

- Scrivere il codice MIPS relativo all'istruzione C

  g = h + A[i]

- Supponendo che
  - le variabili g, h, i siano rispettivamente nei registri $1, $2, $4
  - l'indirizzo di base del vettore A sia nel registro $3

```
add   $5, $4, $4          #calcolo 2i = i+i
add   $5, $5, $5          #calcolo 4i = 2i+2i
add   $5, $3, $5          # calcolo l'indirizzo di A[i]
lw     $6, 0($5)          # carico A[i] in $6
add   $1, $2, $6          # g = h+ A[i]
```


Il set di istruzioni

- Istruzioni aritmetiche e logiche
  - effettuano operazioni tra dati presenti nei registri
- Istruzioni di trasferimento dati
  - spostano i dati tra memoria e registri
- Istruzioni di controllo
  - salti condizionati e incondizionati

Istruzioni di controllo

- Il calcolatore è in grado di prendere delle decisioni
  - in base ai dati in ingresso e ai valori calcolati durante l'elaborazione possono essere eseguite istruzioni diverse
- Nei linguaggi di programmazione un processo di decisione è rappresentato con il costrutto
  - *if (condizione) goto L*
    - dove L rappresenta una porzione di codice non successiva ad if
- Il linguaggio assembler MIPS contiene diverse istruzioni di controllo che permettono di implementare un processo di decisione
  - beq registro 1, registro2, L1
  - bne registro 1, registro2, L1
  - j L1 (ci servirà insieme alla beq per tradurre cicli)

Istruzioni di controllo: salto condizionato

- L'istruzione branch if equal (beq)
  - salta all'indirizzo indicato dell'istruzione se due valori sono uguali
- Semantica
  - if (rt == rs) go to L
    - in realtà: if (rt== rs) go to PC + # istruzioni da saltare
- Sintassi:
  - beq $1, $2, L # vai all'istruzione L se il valore contenuto nel registro $1 è uguale al valore contenuto nel registro $2

Istruzioni di controllo: salto condizionato

- L'istruzione branch if not equal (bnq)
  - salta all'indirizzo indicato dell'istruzione se due valori sono diversi
- Semantica
  - if (rt != rs) go to L
    - in realtà: if (rt != rs) go to PC + # istruzioni da saltare
- Sintassi:
  - bnq $1, $2, L # vai all'istruzione L se il valore contenuto nel registro $1 non è uguale al valore contenuto nel registro $2

Istruzioni di controllo: salto condizionato

- Un esempio di salto condizionato

```
if (i==j)
    go to L1
i = i+j;
L1:  j = j+4;
```

Supponiamo che le variabili i e j corrispondano ai registri $1 e $2

```
4    beq  $1, $2, 1        # se i==j vai all'istruzione 12
8    add  $1, $1, $2       # i = i+j
12   addi $2, $2, 4        # j = j + 4
```


Istruzioni di controllo: salto non condizionato

- L'istruzione Jump (j)
  - salta all'indirizzo indicato nell'istruzione
- Semantica
  - goto 32
- Sintassi
  - j 32 # salta all'istruzione con indirizzo 32 (*non è proprio così…*)

Istruzioni di controllo: salto condizionato e non condizionato

- Esempio

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

Supponiamo che
$1 ← f , $2 ← g , $3 ← h,
$4 ← i , $5 ← j

```
4    beq  $4, $5, 2     # se i==j vai all'istruzione 16
8    sub  $1, $2, $3    # f = g-h (eseguita solo se i != j)
12   j 20               # vai all'istr. 20 (eseguita solo se i != j)
16   add $1, $2, $3     #  f = g+h (eseguita solo se i==j)
20    …..
```


Istruzioni di controllo: salto non condizionato

- L'istruzione Jump register (jr)
  - salta all'indirizzo contenuto in un registro
  - è usata nei costrutti break/case e nel ritorno da funzione
- Sintassi: jr $ra
- Formato R

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
0	31	0	0	0	8

Istruzioni di controllo: Set on Less Than

- Oltre a testare l'uguaglianza tra due variabili, talvolta è necessario confrontarle e la beq in tal caso non basta
- L'istruzione Set on Less Than (slt) permette di confrontare i valori contenuti in due registri
  - assegna 1 se un valore è minore di un altro, 0 viceversa
- Semantica
  - if ($2 < $3) $1=1; else $1=0
- Sintassi
  - slt $1, $2, $3 # se il contenuto del registro $2 è minore del contenuto del registro $3 allora memorizza 1 nel registro $1, altrimenti memorizza 0

Istruzioni di controllo: Set on Less Than

- Esempio

```
if (i<=j)
    go to L1
i = i+j;
L1:  j = j+4;
```

Supponiamo che le variabili i e j corrispondano ai registri $1 e $2

```
4    slt   $3, $2, $1     # se j<i allora $3← 1  altrimenti $3 ← 0
8    beq   $3, $0,  1     # se i<=j ($3 ← 0) vai all'istruzione 16
12   add   $1, $1, $2     # i = i+j
16   addi $2, $2, 4       # j = j + 4
```


Ciclo while con beq e j

```
while (i<j) {
    …
    …
}
```

Supponiamo che le variabili i e j corrispondano ai registri $2 e $3

```
Ciclo: if  (!(i < j)) goto Exit
       …
       …
       goto Ciclo
Exit:
```

```
Ciclo: slt $1, $2, $3
       beq $zero, $1, Exit
       ……
       ……
       j Loop
Exit:
```


Ciclo while con beq e j

```
while (i>j) {
    …
    …
}
```

Supponiamo che le variabili i e j corrispondano ai registri $2 e $3

```
Ciclo: if  (!(j < i)) goto Exit
       …
       …
       goto Ciclo
Exit:
```

```
Ciclo: slt $1, $3, $2
       beq $zero, $1, Exit
       ……
       ……
       j Loop
Exit:
```


Esercizio

- Realizzare in assembler MIPS il seguente frammento di codice C

```
while (A[i] == k)
        i = i + j;
```

Si supponga che le variabili i , j e k corrispondano ai registri $1, $2 e $3 e che l'indirizzo di base di A sia in $4

```
4    add  $5, $1, $1      # calcolo 2i = i+i
8    add  $5, $5, $5      # calcolo 4i = 2i+2i
12   add  $5, $5, $4      # calcolo l'indirizzo di A[i]
16   lw   $6, 0($5)       # caricoA[i] nel registro temporaneo $4
20   beq $6, $3, 1        # se A[i]==k vai all'istruzione 28
24   J 36                 # se A[i]!=k vai all'istruzione 36 (esce dal ciclo)
28   add $1, $1, $2       # i = i+j
32   J 4                  # vai all'istruzione 4 (inizio del ciclo)
36   …
```


Sommario

- La programmazione dei calcolatori
  - Programmare a diversi livelli di astrazione
  - Tradurre e avviare un programma
- Il set di istruzioni MIPS
  - Architettura MIPS
  - Operandi e Registri
  - Il set di istruzioni
    - Operazioni aritmetiche e logiche
    - Operazioni di trasferimento dati
    - Istruzioni di controllo
      - Realizzazione di cicli
- Dal linguaggio assembler al linguaggio macchina
  - Rappresentazione delle istruzioni
    - Formato R, Formato I, Formato J

Rappresentazione delle istruzioni

- I dati all'interno della memoria sono numeri che rappresentano dati ed istruzioni
  - i numeri vengono rappresentati tramite la notazione binaria
- Il MIPS adotta istruzioni a lunghezza costante: 32 bit
- Vediamo come questi 32 bit sono stati suddivisi in diversi campi per definire
  - il formato delle istruzioni, rispettando il principio della regolarità
    - nel far questo, emergeranno i modi di indirizzamento del MIPS
- Da quanto visto, le istruzioni esaminate si possono suddividere in 3 categorie
  - istruzioni che devono indicare 3 registri
    - add, sub, and, slt
  - istruzioni che devono indicare due registri e una costante
    - lw e sw
    - istruzioni che riferiscono un operando immediato (addi)
    - salti condizionati (due registri per il confronto + costante per il salto)
  - istruzioni di salto incondizionato che non riferiscono alcun registro ma (presumibilmente) indicano una costante "con molti bit"…
- Le tre categorie danno luogo a tre (semplici) formati, con cui si riescono a rappresentare tutte le istruzioni

Rappresentazione delle istruzioni: Formato R

- Formato R (register)
  - Istruzione a 32 bit suddivisa in 6 campi

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

  - op: codice operativo
  - rs/rt: primo e secondo operando sorgente (registri)
  - rd: registro destinazione
  - shamt: shift amount (usato da operazioni di shift)
  - funct: in base a op indica una operazione specifica
    - ADD: op=0 e funct= 100000; SUB: op=0 e funct = 100010; AND op=0 e funct = 100100; OR: op=0 e funct=100101;SLT: op=0 e funct= 101010

Codifica dell'istruzione add

- Formato R
- L'istruzione add $8, $17, $18
  - viene rappresentata in una parola di 32 bit

op	rs	rt	rd	shamt	funct
0	**17**	**18**	**8**	**0**	**32**
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	**10001**	**10001**	**01000**	**00000**	**100000**

  - op e funct: operazione di add
  - rs: registro contenete il primo operando
  - rt: registro contenete il secondo operando
  - rd: registro destinazione
  - shamt: non utilizzato → messo a zero

Rappresentazione delle istruzioni: Formato I

- Formato I (immediate)
  - Istruzione a 32 bit suddivisa in 4 campi

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

  - op: codice operativo (opcode)
  - rs/rt: primo e secondo operando sorgente
  - address/immediate: indirizzo di partenza/costante

Codifica dell'istruzione lw

- Formato I
- L'istruzione lw $8, 1000$19
  - viene rappresentata in una parola di 32 bit

op	rs	rt	address
35	**19**	**8**	**1000**
6 bit	5 bit	5 bit	16 bit
100011	**10011**	**01000**	**00000011111010000**

  - op: operazione effettuata
  - rs: registro base a cui va sommato address
  - rt: registro destinazione per lw (/sorgente per sw)
  - address: indirizzo di partenza

Codifica dell'istruzione addi

- Formato I
- L'istruzione addi $s8, $s9, 32
  - viene rappresentata in una parola di 32 bit

op	rs	rt	address/immediate
8	**8**	**9**	**32**
6 bit	5 bit	5 bit	16 bit
001000	**01000**	**01001**	**0000000000100000**

  - op: operazione effettuata
  - rs: registro sorgente
  - rt: registro destinazione
  - immediate: operando da sommare

Codifica dell'istruzione beq

- Formato I
- L'istruzione beq $8, $9, 1000
  - viene rappresentata in una parola di 32 bit

op	rs	rt	address/immediate
4	8	9	1000
6 bit	5 bit	5 bit	16 bit
000100	01000	01001	0000001111101000

    - op: operazione effettuata
    - rs: registro con il primo operando
    - rt: registro con il secondo operando
    - Indirizzo: indirizzo della label
- Come viene calcolato il salto?
  - Indirizzamento relativo al program counter (pc-relative)

Indirizzamento relativo al Program Counter (1)

- Abbiamo a disposizione solamente 16 bit per memorizzare un indirizzo
  - se usiamo le stesse tecniche usate per il formato J non potremmo saltare a indirizzi più grandi di 218
    - usando l'indirizzamento alla parola (16 bit →18 bit) e concatenazione con i 14 bit più significativi del PC
  - i nostri programmi sarebbero limitati in dimensione ad avere 218 byte = 256 Kbyte
    - il che è ovviamente non accettabile

Indirizzamento relativo al Program Counter (2)

- Si sceglie di usare il valore nei 16 bit come un valore relativo alla posizione corrente
  - cioè indica la distanza (in numero di parole) tra l'istruzione corrente e la istruzione da eseguire se il test (di uguaglianza) ha successo
- Viene sommato il valore nel campo indirizzo al valore del registro Program Counter
  - che contiene l'indirizzo della prossima istruzione da eseguire
    - es. se stiamo eseguendo un branch alla istruzione 1000 allora il PC vale già 1004

[1000] beq $8, $9, 1400

Indirizzamento relativo al Program Counter (3)

- In pratica per l'istruzione

[1000] beq $8, $9, 1400

- poiché l'indirizzo 1400 si riferisce al byte dobbiamo calcolare quante parole ci sono tra l'indirizzo nel PC (1004) e l'indirizzo destinazione
  - 1400 – 1004 rappresenta il numero di byte che ci sono tra l'indirizzo nel PC e l'indirizzo destinazione
  - per calcolare il numero di parole dobbiamo dividere per 4
    - (1400 – 1004) / 4 = 396/4 = 99
  - si memorizza il numero 99 nei 16 bit

Indirizzamento relativo al Program Counter (4)

- In pratica per l'istruzione

[1036] beq $8, $9, 1000

- poiché l'indirizzo 1000 si riferisce al byte dobbiamo calcolare quante parole ci sono tra l'indirizzo nel PC (1040) e l'indirizzo destinazione
  - 1040 – 1000 rappresenta il numero di byte che ci sono tra l'indirizzo nel PC e l'indirizzo destinazione
  - per calcolare il numero di parole dobbiamo dividere per 4
    - (1000 – 1040) / 4 = -40 / 4 = -10
  - si memorizza il numero -10 nei 16 bit

Rappresentazione delle istruzioni: Formato J

- Formato J (jump)
  - Istruzione a 32 bit suddivisa in 2 campi

op	indirizzo
6 bit	26 bit

  - op: codice operativo
  - indirizzo: indirizzo a cui saltare

Il formato dell'istruzione jump

- Formato J
- L'istruzione j 1000
  - viene rappresentata in una parola di 32 bit

op	indirizzo
2	**1000**
6 bit	26 bit
000010	**00000000000000001111101000**

  - op: operazione effettuata
  - indirizzo: indirizzo a cui saltare
    - in realtà si utilizza un indirizzamento pseudo-diretto…
    - …ma per semplicità useremo l'indirizzo dell'istruzione

Riepilogo: i formati delle istruzioni

- Istruzioni formato R
  - add $8, $17, $18
  - sub $8, $17, $18,

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

- Istruzioni formato I
  - lw $8, 1000($19)
  - sw $8, 1000($19)
  - beq $8, $9, L1

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

- Istruzioni formato J
  - j 1000

op	indirizzo destinazione
6 bit	26 bit