

Lezione 18 Il Set di Istruzioni (6)

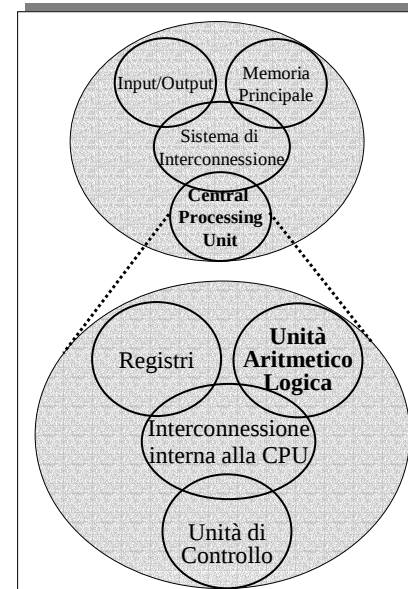
Vittorio Scarano

Architettura

Corso di Laurea in Informatica
Università degli Studi di Salerno

Un quadro della situazione

Architettura (2003-2004). Vittorio Scarano



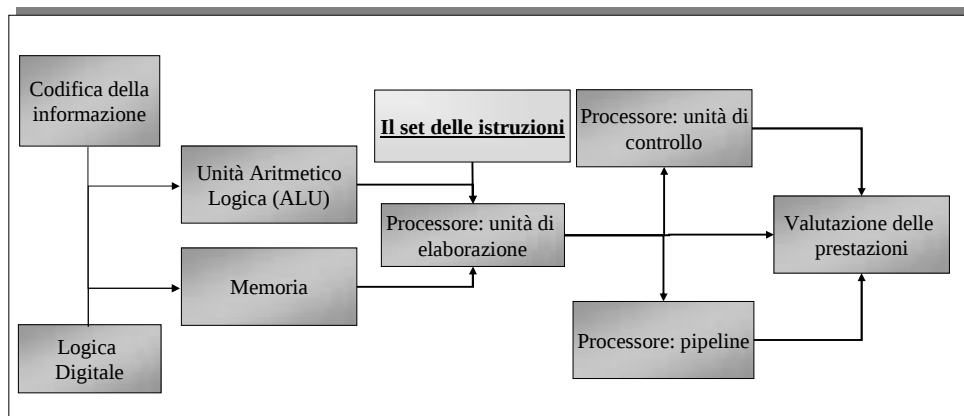
- Cosa abbiamo fatto...
 - Il Set di Istruzioni
- Dove stiamo andando..
 - progettazione del processore
- Perché:
 - per poter capire cosa deve offrire al programmatore il processore come istruzioni

2

Dove siamo nel corso...

- Il set di istruzioni

Architettura (2003-2004). Vittorio Scarano



3

Le operazioni e gli operandi del MIPS

Operazioni:

- ➡ – operazioni aritmetiche
- operazioni di trasferimento (memoria ↔ registri)
- operazioni di scelta e controllo del flusso
- operazioni di supporto alle procedure

• Operandi:

- accesso ai registri della macchina
- accesso alla memoria
- uso di costanti

Architettura (2003-2004). Vittorio Scarano

4

Il formato della istruzione add

- La istruzione

add \$8, \$17, \$18

- viene rappresentata in una parola di 32 bit:

0	17	18	8	0	32
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	op

- Questa codifica viene detta
– *formato della istruzione register (R)*

5

I campi del formato R

add \$8, \$17, \$18

0	17	18	8	0	32
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	op

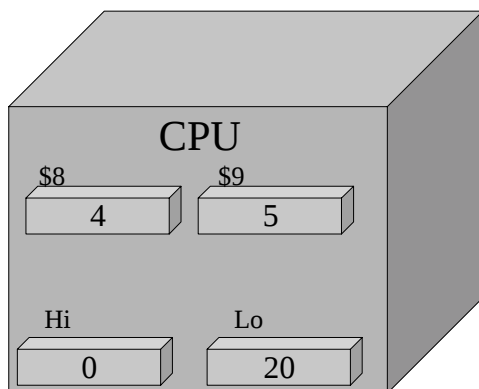
- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *rd*: registro destinazione
- Campo *shamt*: scorrimento

6

Istruzione di moltiplicazione (formato R)

mult \$8, \$9

moltiplica il valore di \$8 e \$9 e mette i 32 bit più significativi in *Hi* e i 32 bit meno significativi in *Lo*



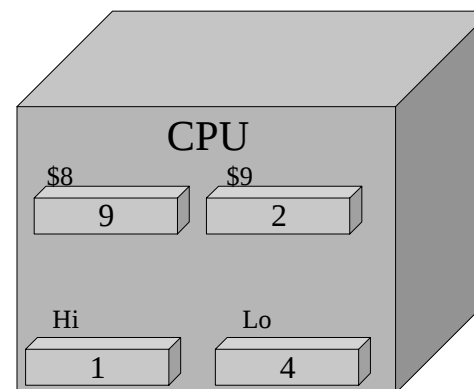
.....
mult \$8, \$9
.....

7

Istruzione di divisione (formato R)

div \$8, \$9

divide il valore di \$8 con quello in \$9 e mette il quoziente in *Lo* ed il resto in *Hi*



.....
div \$8, \$9
.....

8

Il formato R della istruzione mult

- La istruzione **mult \$8, \$9**
- viene rappresentata in una parola di 32 bit:

0	8	9	0	0	24
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	01000	01001	00000	00000	011000
op	rs	rt	rd	shamt	op

- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *rd*: registro destinazione (nullo)
- Campo *shamt*: scorrimento

9

Il formato R della istruzione div

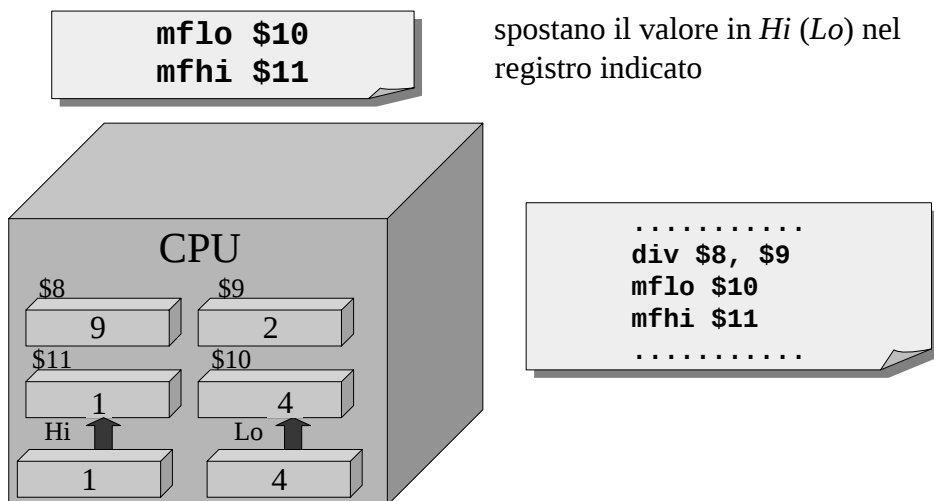
- La istruzione **div \$8, \$9**
- viene rappresentata in una parola di 32 bit:

0	8	9	0	0	26
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	01000	01001	00000	00000	011010
op	rs	rt	rd	shamt	op

- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *rd*: registro destinazione (nullo)
- Campo *shamt*: scorrimento

10

Utilizzo dei registri Hi e Lo (formato R)



11

Il formato R della istruzione mfhi

- La istruzione **mfhi \$8**
- viene rappresentata in una parola di 32 bit:

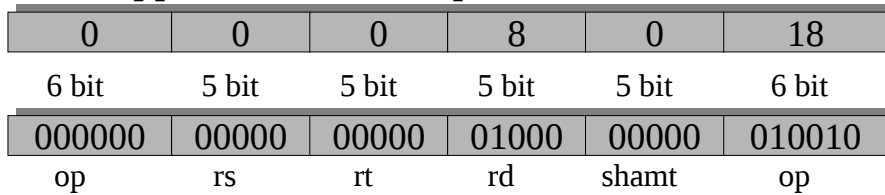
0	0	0	8	0	16
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	00000	00000	01000	00000	010000
op	rs	rt	rd	shamt	op

- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando (nullo)
- Campo *rt*: registro con il secondo operando (nullo)
- Campo *rd*: registro destinazione
- Campo *shamt*: scorrimento

12

Il formato R della istruzione mflo

- La istruzione **mflo \$8**
- viene rappresentata in una parola di 32 bit:



- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando (nullo)
- Campo *rt*: registro con il secondo operando (nullo)
- Campo *rd*: registro destinazione
- Campo *shamt*: scorrimento

13

Le operazioni e gli operandi

- Operazioni di tipo diverso:
 - operazioni aritmetiche
 - operazioni di trasferimento (memoria ↔ registri)
 - operazioni di scelta e controllo del flusso
 - operazioni di supporto alle procedure

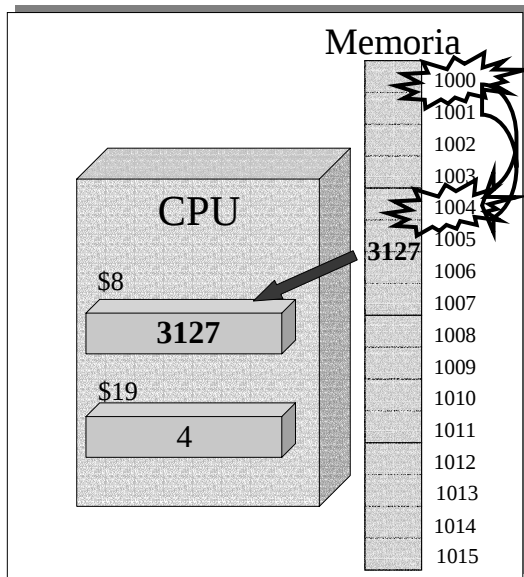
Operandi:

- accesso ai registri della macchina
- accesso alla memoria
- uso di costanti

Architettura (2003-2004). Vittorio Scarano

14

Istruzioni di trasferimento dati: la *load-word*

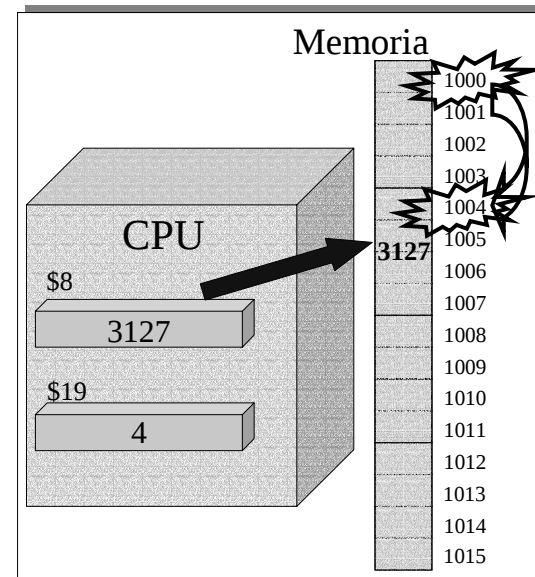


lw \$8, 1000(\$19)

- Indirizzamento indicizzato
 - nel registro \$8 va caricato
 - l'elemento con indirizzo uguale a 1000 + il valore in \$19
- Uso di un registro *indice*

15

Istruzione di trasferimento dati: la *store-word*



sw \$8, 1000(\$19)

- Indirizzamento indicizzato
 - l'elemento nel registro \$8 va memorizzato nel..
 - l'elemento con indirizzo uguale a 1000 + il valore in \$19
- Uso di un registro *indice*

16

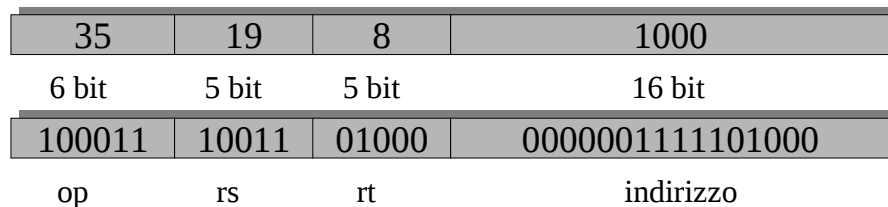
Architettura (2003-2004). Vittorio Scarano

Il formato della istruzione lw

- La istruzione

lw \$8, 1000(\$19)

- viene rappresentata in una parola:

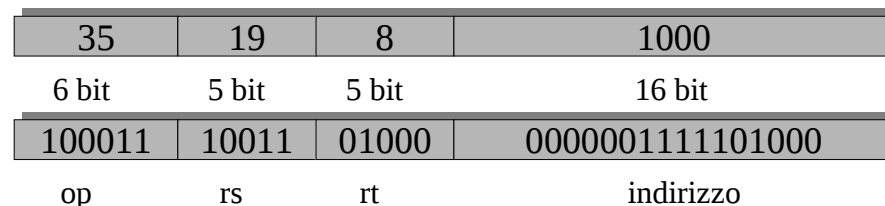


- Questa codifica viene detta
– *formato della istruzione Instruction (I)*

17

I campi del formato I

lw \$8, 1000(\$19)



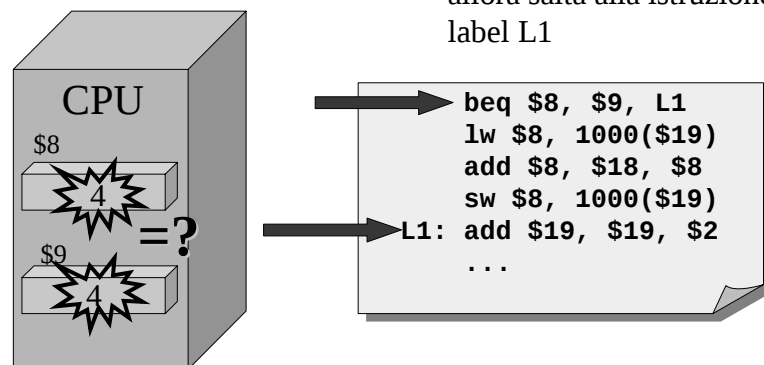
- Campo *op*: operazione effettuata
- Campo *rs*: registro con indice
- Campo *rt*: registro destinazione
- Campo *indirizzo*: indirizzo partenza

18

Istruzioni di branch: Branch-if-Equal (1)

beq \$8, \$9, L1

se il valore del registro \$8 è uguale al valore del registro \$9 allora salta alla istruzione con label L1

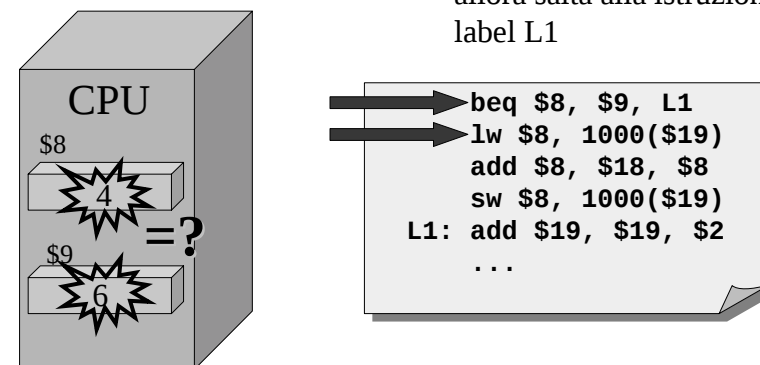


19

Istruzioni di branch: Branch-if-Equal (2)

beq \$8, \$9, L1

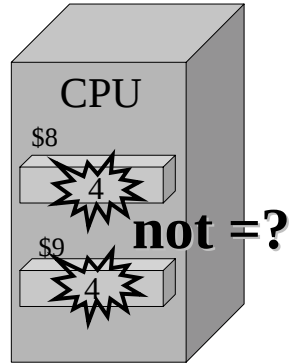
se il valore del registro \$8 è uguale al valore del registro \$9 allora salta alla istruzione con label L1



20

Istruzioni di branch: Bbranch-if-Not-Equal (1)

bne \$8, \$9, L1



se il valore del registro \$8 **NON** è uguale al valore del registro \$9 allora salta alla istruzione con label L1

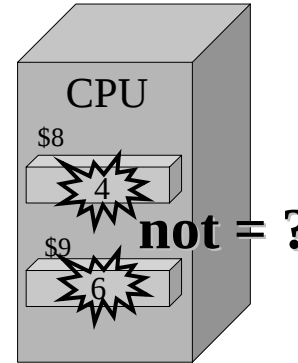
```

bne $8, $9, L1
lw $8, 1000($19)
add $8, $18, $8
sw $8, 1000($19)
L1: add $19, $19, $2
...
    
```

21

Istruzioni di branch: Bbranch-if-Not-Equal (2)

bne \$8, \$9, L1



se il valore del registro \$8 **NON** è uguale al valore del registro \$9 allora salta alla istruzione con label L1

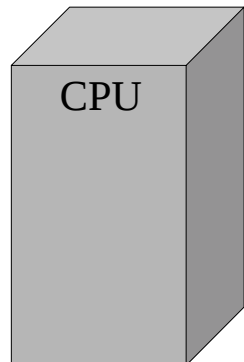
```

bne $8, $9, L1
lw $8, 1000($19)
add $8, $18, $8
sw $8, 1000($19)
L1: add $19, $19, $2
...
    
```

22

Istruzioni di jump: Jump

j L1



salta alla istruzione con label L1

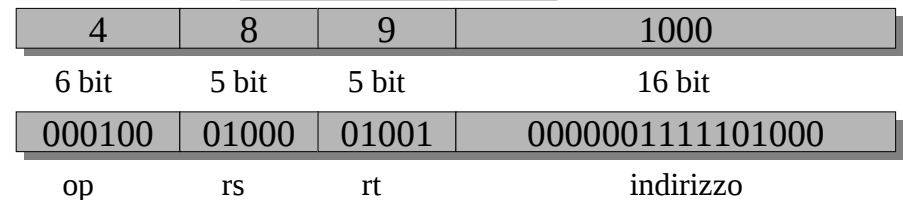
```

....
j    L1
add $8, $18, $8
sw $8, 1000($19)
L1: add $19, $19, $2
...
    
```

23

Il formato I: le istruzioni di *branch*

beq \$8, \$9, 1000

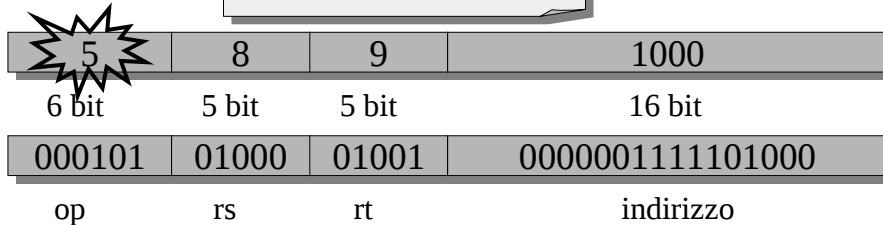


- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *indirizzo*: indirizzo della label

24

Il formato I: le istruzioni di *branch*

bne \$8, \$9, 1000

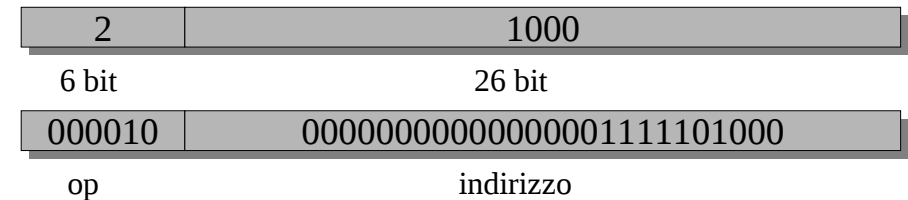


- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *indirizzo*: indirizzo della label

25

Il formato J: le istruzioni di *jump*

j 1000

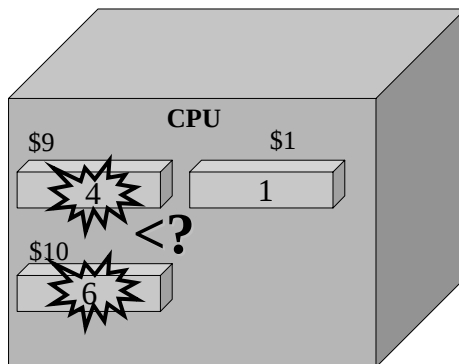


- Campo *op*: operazione effettuata
- Campo *indirizzo*: indirizzo della label

26

Istruzioni di scelta: *Set-on-Less-Than* (1)

slt \$1, \$9, \$10

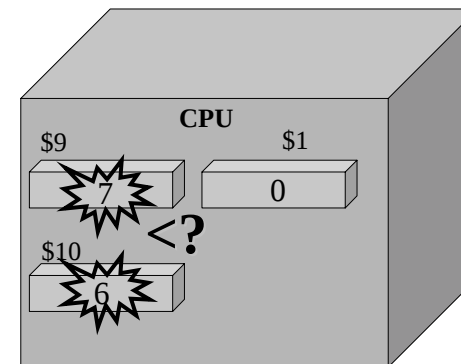


se il valore del registro \$9 è minore del valore del registro \$10 allora poni a 1 il registro \$1, altrimenti poni a 0 il registro \$1

27

Istruzioni di scelta: *Set-on-Less-Than* (2)

slt \$1, \$9, \$10

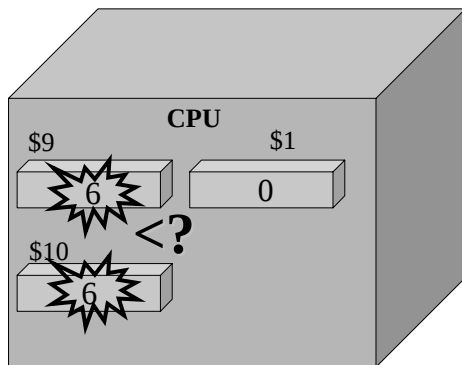


se il valore del registro \$9 è minore del valore del registro \$10 allora poni a 1 il registro \$1, altrimenti poni a 0 il registro \$1

28

Istruzioni di scelta: *Set-on-Less-Than* (3)

slt \$1, \$9, \$10



se il valore del registro \$9 è minore del valore del registro \$10 allora poni a 1 il registro \$1, altrimenti poni a 0 il registro \$1

29

Il formato di *slt* : il formato R

slt \$1, \$2, \$3

0	2	3	1	0	42
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	00010	00011	00001	00000	101010

op rs rt rd shamt op

- Campo *op*: operazione effettuata
- Campo *rs*: registro con il primo operando
- Campo *rt*: registro con il secondo operando
- Campo *rd*: registro destinazione
- Campo *shamt*: scorrimento

30

Il formato di *jr* : il formato R

jr \$8

0	8	0	0	0	8
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
000000	01000	00000	00000	00000	001000

op rs rt rd shamt op

- Campo *op*: operazione effettuata
- Campo *rs*: registro con l'operando
- Campo *rt*: inutilizzato
- Campo *rd*: inutilizzato
- Campo *shamt*: scorrimento

31

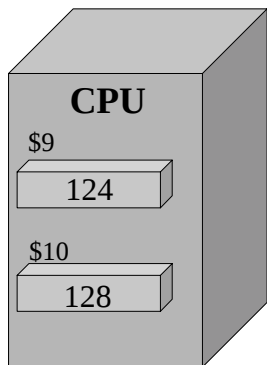
Le operazioni e gli operandi

- Operazioni di tipo diverso:
 - operazioni aritmetiche
 - operazioni di trasferimento (memoria ↔ registri)
 - operazioni di scelta e controllo del flusso
 - operazioni di supporto alle procedure
- Operandi:
 - accesso ai registri della macchina
 - accesso alla memoria
 - ➔ – uso di costanti

32

Istruzioni di somma con operando: *addi*

addi \$10, \$9, 4

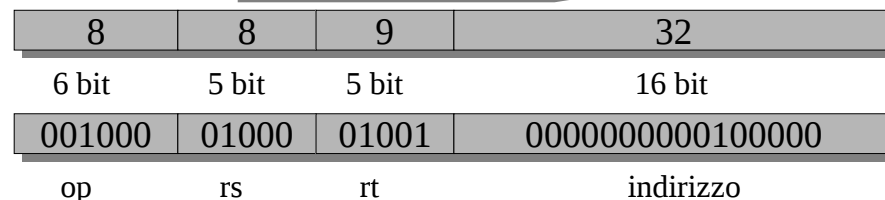


somma la costante 4 al valore contenuto in \$9 e poni il risultato in \$10

33

Il formato per ind. immediato: il formato I

addi \$8, \$9, 32

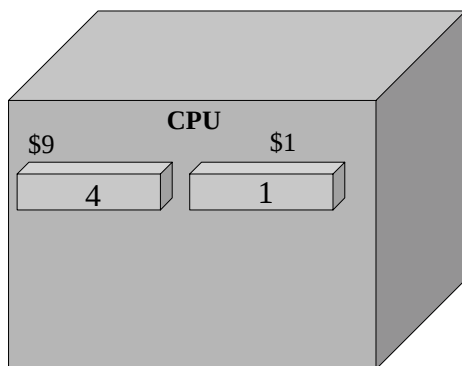


- Campo *op*: operazione effettuata
- Campo *rs*: registro destinazione
- Campo *rt*: registro con il primo operando
- Campo *indirizzo*: operando da sommare

34

Istruzioni di scelta con operando: *slti*

slti \$1, \$9, 120

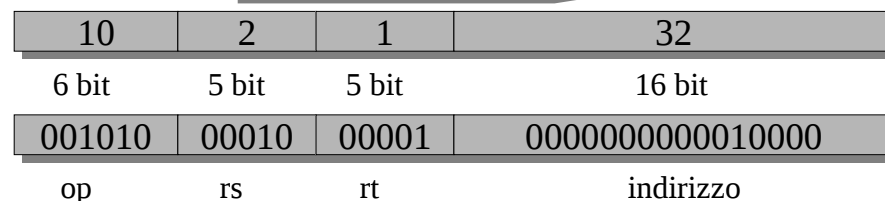


se il valore del registro \$9 è minore di 120 allora poni a 1 il registro \$1, altrimenti poni a 0 il registro \$1

35

Il formato per ind. immediato: il formato I

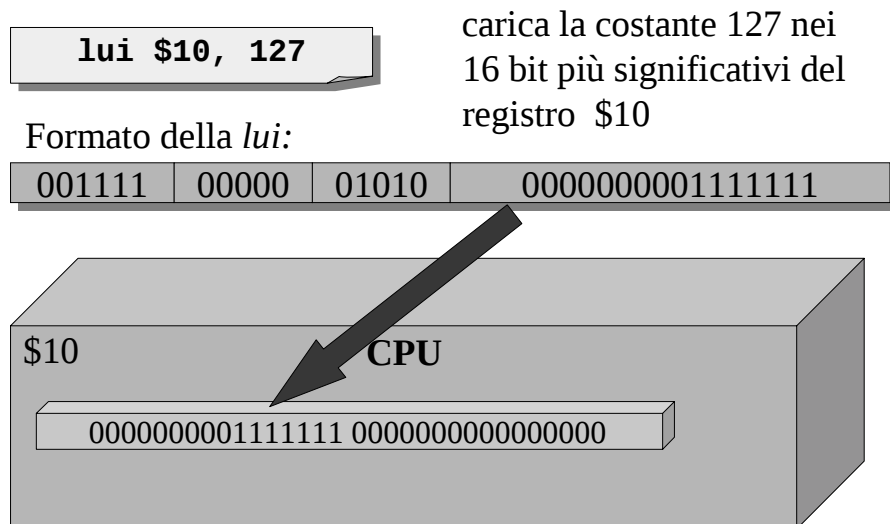
slti \$1, \$2, 32



- Campo *op*: operazione effettuata
- Campo *rs*: registro destinazione (set)
- Campo *rt*: registro con il primo operando
- Campo *indirizzo*: operando da confrontare

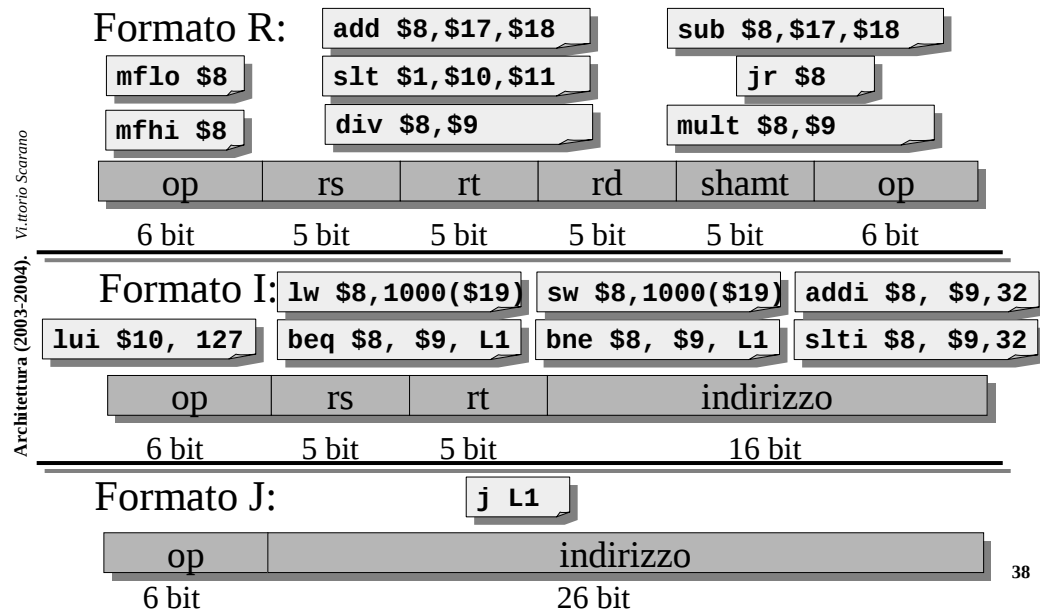
36

Istruzioni di caricamento costante: *lui*



37

Riepilogo: i formati delle istruzioni



38

Organizzazione della lezione

- Istruzioni per la gestione delle procedure
 - la istruzione jal
- Come si avvia un programma
 - compilatore
 - assembler
 - linker
 - loader
- Ruolo dei registri e convenzioni utilizzate
- Alcune note conclusive

39

Le operazioni e gli operandi del MIPS

Operazioni:

- operazioni aritmetiche
- operazioni di trasferimento (memoria ↔ registri)
- operazioni di scelta e controllo del flusso
- operazioni di supporto alle procedure

Operandi:

- accesso ai registri della macchina
- accesso alla memoria
- uso di costanti

40

Le procedure

- In ogni linguaggio di programmazione si usa strutturare il proprio codice in *procedure* (funzioni, metodi, etc.)
- Permette
 - la riusabilità del codice
 - un testing più approfondito
 - la creazione di librerie di funzioni utilizzate di frequente
 - fornite al programmatore dall'ambiente
 - non necessario scrivere funzioni comuni (matematiche, di input/output, etc.), basta usarle!

41

Prima della chiamata di una procedura

- E' necessario:
 1. mettere i parametri alla procedura in una posizione accessibile alla procedura
 2. trasferire il controllo alla procedura (salto)
 3. ottenere i valori restituiti
 4. ritornare al punto di chiamata della procedura
- 1: Accesso ai parametri
 - si usano 4 registri specifici \$a0, \$a1, \$a2, \$a3
- 2: trasferire il controllo alla procedura
 - istruzione jal

42

La istruzione *jump-and-link*

jal 1000

- La istruzione jal permette di fare due cose:
 - saltare all'indirizzo specificato (come la istruzione j)
 - memorizza l'indirizzo di ritorno nel registro \$ra
 - indirizzo della istruzione successiva da eseguire
- 3. Valori restituiti
 - la procedura mette i risultati nei registri \$v0 e \$v1
- 4. Ritorno al punto di chiamata della procedura
 - si può usare la istruzione jr con il registro \$ra

jr \$ra

43

Procedure annidate

- In caso di una chiamata a procedura che chiama una altra procedura che chiama un'altra procedura ...
 - il programmatore deve provvedere a salvare i vari indirizzi di ritorno su uno stack
- Lo stack viene gestito tramite un puntatore alla cima dello stack memorizzato in un registro apposito \$sp
- Nello stack vengono anche salvati i valori di 8 registri indicati con \$s0, ..., \$s7
 - che quindi vengono ripristinati quando si ritorna dalla chiamata effettuata
- I registri \$t0, ..., \$t7 non vengono salvati su stack

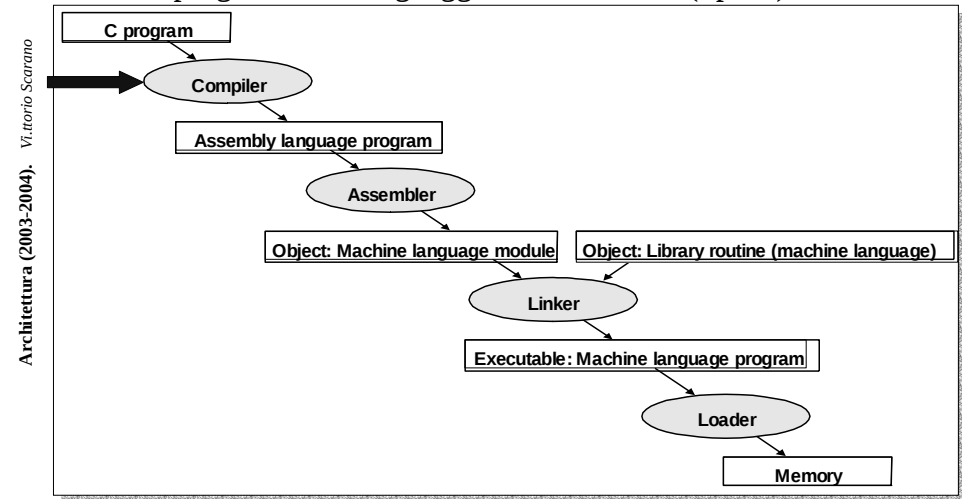
44

Organizzazione della lezione

- Istruzioni per la gestione delle procedure
 - la istruzione jal
- Come si avvia un programma
 - compilatore
 - assemblatore
 - linker
 - loader
- Ruolo dei registri e convenzioni utilizzate
- Alcune note conclusive

Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco

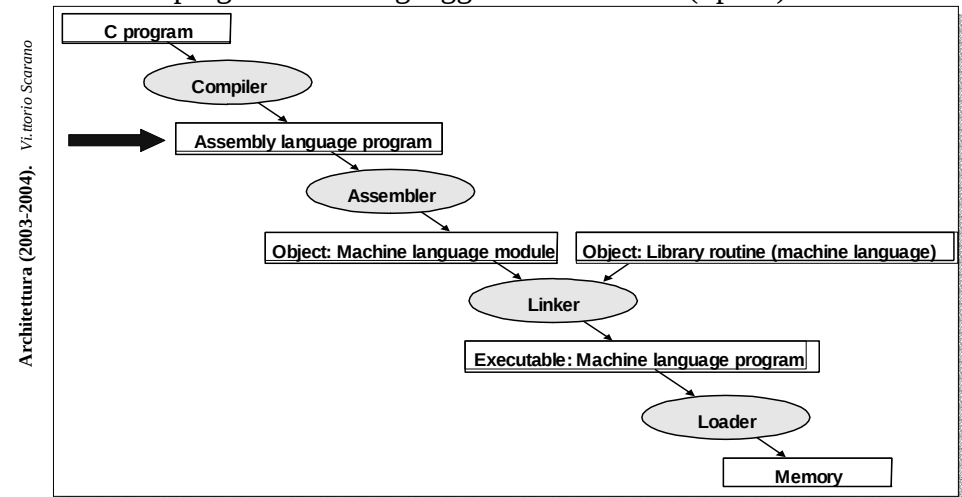


1. Il compilatore

- Trasforma il programma C in un programma in linguaggio assembler
- Il linguaggio assembler
 - forma simbolica di un programma in linguaggio macchina
- Efficienza nella traduzione
 - i compilatori generano codice assembler che risulta ottimizzato quasi quanto quello scritto “a mano”
 - precedentemente, i programmi “critici” (Sistemi operativi, etc.) erano scritti direttamente in assembler

Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco



Il linguaggio Assembler

- Il linguaggio assembler
 - rappresenta una “interfaccia” verso il software di livello più alto
- Possibile che siano “aggiunte” al linguaggio macchina delle *pseudoistruzioni*
 - istruzioni che non sono presenti nel linguaggio macchina
 - non sono implementate in hardware
 - ma che sono abbastanza semplici da tradurre in linguaggio macchina
 - e rappresentano un aiuto per il programmatore

49

Le pseudoistruzioni MIPS - 1

- Ricordiamo che il registro \$0 è mantenuto sempre a zero dal processore
- Quindi può essere usato per una pseudoistruzione che sposti il valore di un registro in un altro registro:
- La istruzione `move $8, $9`
 - viene tradotta dall’assemblatore in `add $9, $0, $8`
- Si possono caricare costanti a 32 bit in un registro
 - l’assemblatore la traduce usando le istruzioni lui e add
- Possibile specificare anche costanti in base esadecimale

50

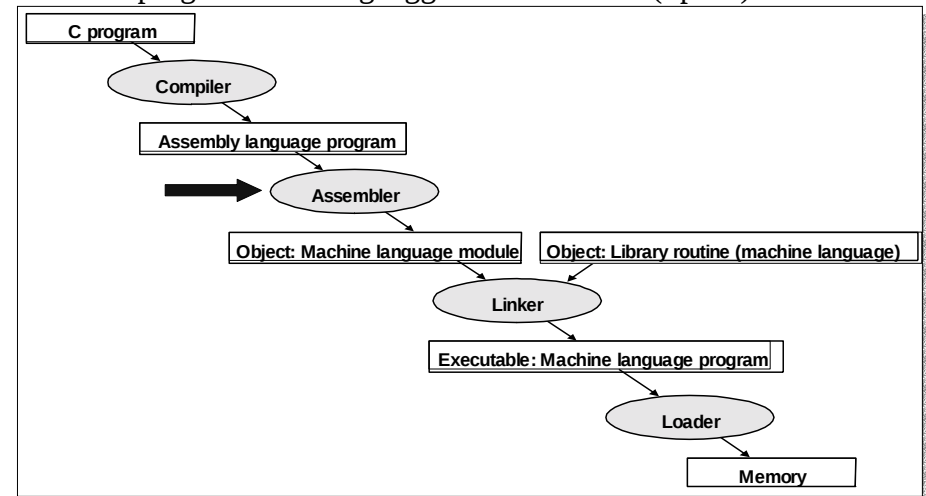
Le pseudoistruzioni MIPS - 2

- Le istruzioni di branch con la slt sono “complesse”
 - L’assemblatore permette l’uso di alcune pseudoistruzioni di branch che vengono aggiunte a beq e bne
- Serve un registro riservato per l’assemblatore \$at
- Un esempio: la *branch-on-less-than* `blt $8,$9,1000`
- Viene implementata con:
`slt $at, $8, $9`
`bne $at, $0, 1000`
- Esistono anche
 - bgt (*branch-on-greater-than*),
 - bge (*branch-on-greater-than-or-equal*),
 - ble (*branch-on-less-than-or-equal*)

51

Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco



52

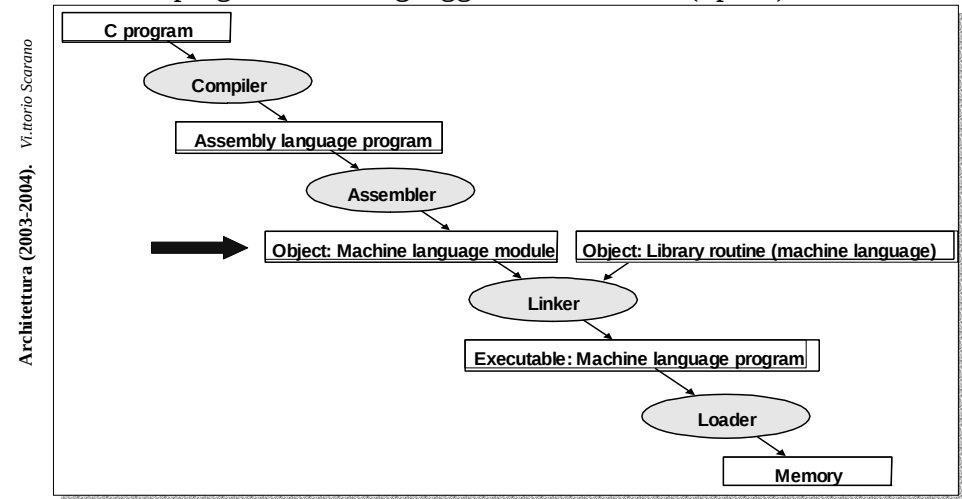
2. L'assemblatore

- Traduce un programma in assembler in un *file oggetto*
 - una combinazione di
 - istruzioni in linguaggio macchina
 - dati e informazioni per la collocazione del programma in memoria
- Ad esempio:
 - per poter tradurre in linguaggio macchina i salti
 - deve mantenere tabelle per i riferimenti dei salti
- Il file oggetto prodotto dall'assemblatore assume che il programma parta dalla locazione di memoria 0
 - sarà il linker a produrre informazioni di rilocazione
 - locazione dipendente da un indirizzo assoluto

53

Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco



54

File oggetto - 1

- Un esempio (in Unix)
- Composto da 6 parti diverse
 - intestazione *object file header*
 - dimensione e posizione delle altre parti
 - segmento di testo (*text segment*)
 - codice in linguaggio macchina
 - segmento di dati (*data segment*)
 - dati che fanno parte del programma (sia *statici* che *dinamici*)
 - informazioni di rilocazione
 - quali sono le istruzioni ed i dati che hanno un indirizzo assoluto

55

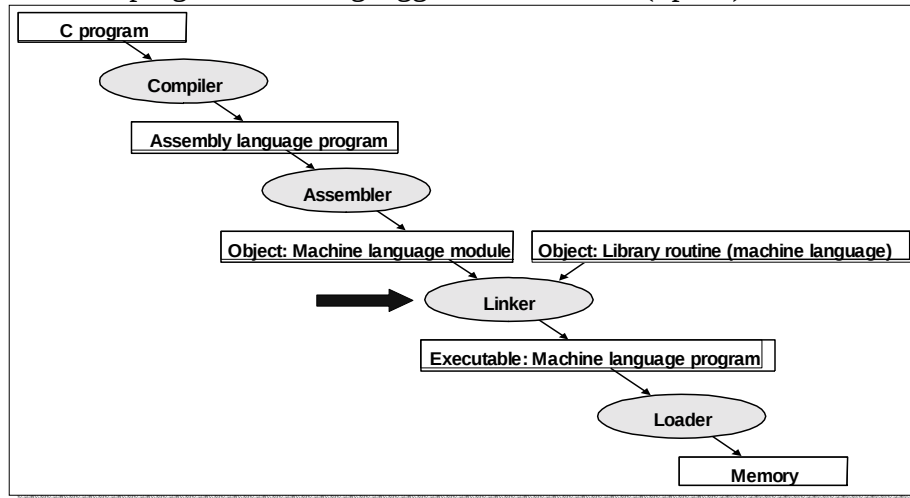
File oggetto - 2

- Composto da 6 parti diverse (continua)
 - *symbol table*
 - label non ancora definite (ad esempio per uso di funzioni di libreria)
 - informazioni per il debugger
 - necessarie per poter associare istruzioni macchina a istruzioni del linguaggio ad alto livello (da parte del debugger)

56

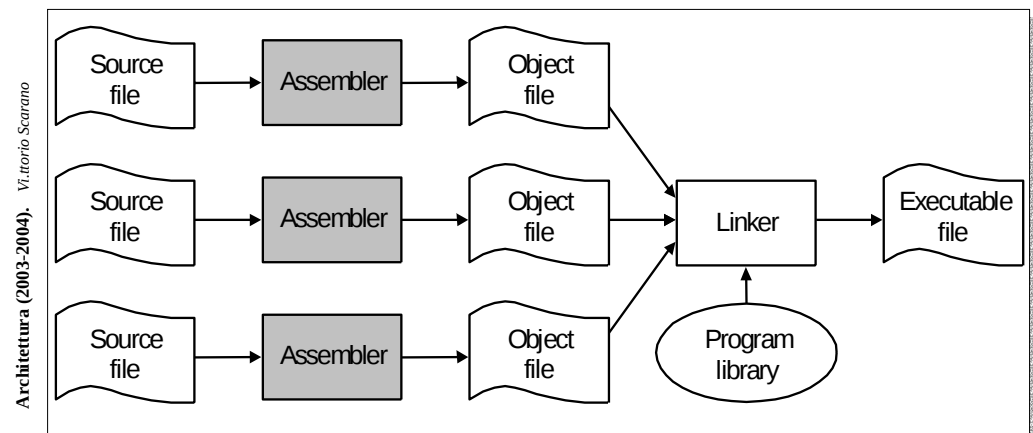
Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco



57

Il ruolo del linker



58

3. Linker - 1

- Per efficienza nella produzione di software
 - il software viene suddiviso in moduli
 - ognuno compilato separatamente
 - e magari raccolto in *librerie*
- Necessario però effettuare il collegamento tra procedure precedentemente compilate
- Compiti del *linker* (a partire da *symbol table* e informazioni di rilocalizzazioni)
 - determinare gli indirizzi dei dati e delle etichette di salti
 - unire i riferimenti interni ed esterni alle procedure

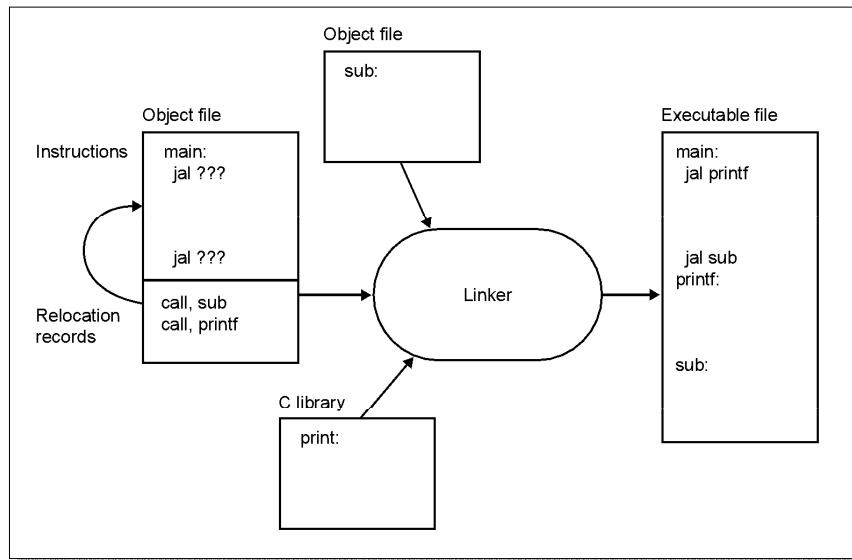
59

3. Linker - 2

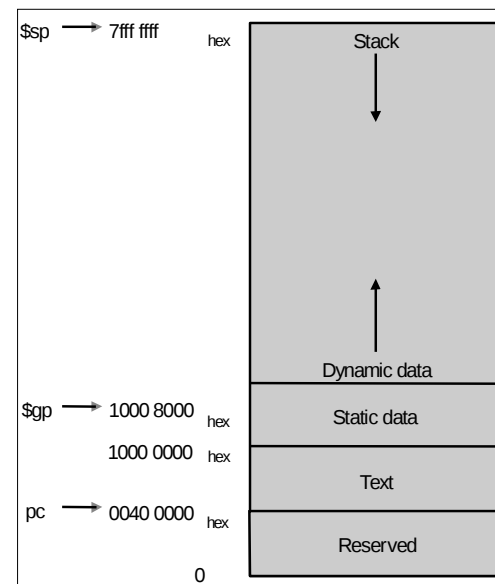
- Il linker effettivamente ha il compito di sostituire e risolvere i riferimenti esterni
- Questo gli permette di sapere esattamente le posizioni di memoria che ogni modulo occupa
 - fornendo gli indirizzi *assoluti* (cioè come se fosse l'unico programma in esecuzione sulla macchina)
- Il linker produce un file eseguibile che può essere eseguito sulla macchina
 - stesso formato del file oggetto ma con tutti i riferimenti risolti

60

Il lavoro del linker in dettaglio

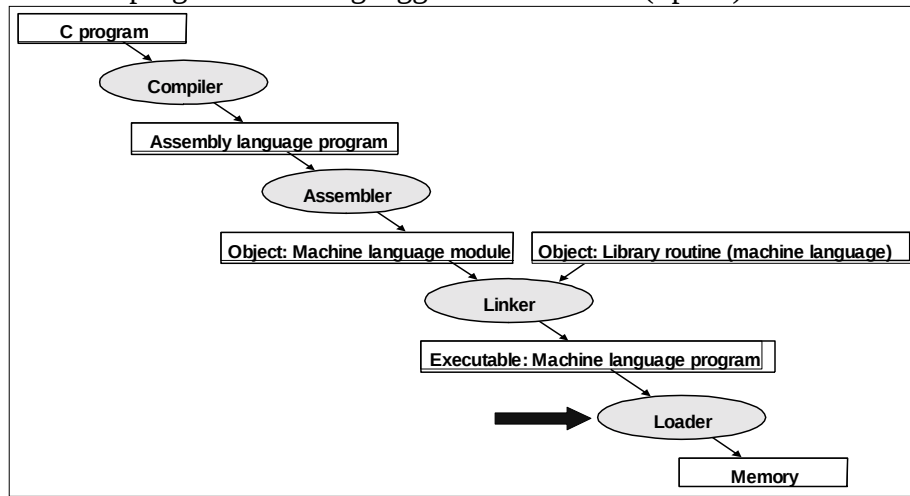


Convenzione MIPS per la allocazione



Dalla compilazione alla esecuzione

- Passi per trasformare ed eseguire
 - un programma in linguaggio ad alto livello (tipo C) che sia su disco



4. Loader - 1

- Una volta che il linker ha creato il file eseguibile, esso di solito viene memorizzato su disco
- All'atto dell'esecuzione, il S.O. lo carica da disco e ne avvia la esecuzione
- Il *loader* si occupa di far partire il programma:
 - legge la intestazione per determinare la dimensione
 - crea uno spazio di memoria sufficiente per testo e dati
 - copia istruzioni e dati in memoria
 - copia nello stack parametri passati al programma principale
 - (continua)

4. Loader - 2

- Il *loader* si occupa di far partire il programma:
 - (continua)
 - inizializza i registri e posizione dello stack pointer
 - salta ad una procedura di inizializzazione che copia i parametri nei registri appositi e poi chiama la procedura di inizio del programma (*main()* in C)

Organizzazione della lezione

- Istruzioni per la gestione delle procedure
 - la istruzione *jal*
- Come si avvia un programma
 - compilatore
 - assemblatore
 - linker
 - loader
- Ruolo dei registri e convenzioni utilizzate
- Alcune note conclusive

Il ruolo dei registri e le convenzioni - 1

- Alcuni registri sono riservati ad un uso particolare

\$zero	0	costante 0
\$at	1	riservato all'assemblatore
\$v0	2	valutazione espressioni e risultato funzioni
\$v1	3	valutazione espressioni e risultato funzioni
\$a0	4	argomento 1
\$a1	5	argomento 2
\$a2	6	argomento 3
\$a3	7	argomento 4
\$t0	8	temporaneo (non mantenuto durante le chiamate)
\$t1	9	temporaneo (non mantenuto durante le chiamate)
\$t2	10	temporaneo (non mantenuto durante le chiamate)
\$t3	11	temporaneo (non mantenuto durante le chiamate)
\$t4	12	temporaneo (non mantenuto durante le chiamate)
\$t5	13	temporaneo (non mantenuto durante le chiamate)
\$t6	14	temporaneo (non mantenuto durante le chiamate)
\$t7	15	temporaneo (non mantenuto durante le chiamate)

Il ruolo dei registri e le convenzioni - 2

- Alcuni registri sono riservati ad un uso particolare

\$s0	16	temporaneo (salvato durante le chiamate)
\$s1	17	temporaneo (salvato durante le chiamate)
\$s2	18	temporaneo (salvato durante le chiamate)
\$s3	19	temporaneo (salvato durante le chiamate)
\$s4	20	temporaneo (salvato durante le chiamate)
\$s5	21	temporaneo (salvato durante le chiamate)
\$s6	22	temporaneo (salvato durante le chiamate)
\$s7	23	temporaneo (salvato durante le chiamate)
\$t8	24	temporaneo (non mantenuto durante le chiamate)
\$t9	25	temporaneo (non mantenuto durante le chiamate)
\$k0	26	riservato al kernel (nucleo) del S.O.
\$k1	27	riservato al kernel (nucleo) del S.O.
\$gp	28	puntatore alla area globale di memoria
\$sp	29	stack pointer
\$fp	30	puntatore al frame (gestione di memoria S.O.)
\$ra	31	indirizzo di ritorno

Organizzazione della lezione

- Istruzioni per la gestione delle procedure
 - la istruzione jal
- Come si avvia un programma
 - compilatore
 - assemblatore
 - linker
 - loader
- Ruolo dei registri e convenzioni utilizzate
- **Alcune note conclusive**

69

Alcune note conclusive

- I principi della architettura a programma memorizzato:
 - istruzioni non distinguibili dai dati
 - memoria del programma modificabile
- Conseguenza: lo stesso calcolatore
 - può essere utilizzato:
 - in contesti diversi
 - da persone diverse
- Queste scelte progettuali hanno influenzato in maniera sensibile il successo dei calcolatori
 - insieme allo sviluppo prettamente tecnologico

70

La scelta del set di istruzioni

- Un delicato compromesso tra
 - numero di istruzioni necessarie per scrivere un programma
 - istruzioni più potenti potrebbero rendere più breve i programmi
 - .. e quindi la occupazione di memoria
 - numero di cicli di clock per istruzione
 - complessità delle operazioni da compiere per la istruzione
 - periodo di clock
 - la dimensione influenza negativamente il periodo di clock
 - maggiore è la dimensione, maggiore il periodo di clock

71

I quattro principi guida per gli autori del set di istruzioni

- *Semplicità e regolarità sono strettamente correlate*
 - stessa dimensione per le istruzioni
 - operazioni aritmetiche solo con registri
- *Minori le dimensioni, maggiore è la velocità*
- *Un buon progetto richiede compromessi*
 - possibilità di specificare costanti di grandi dimensioni mantenendo la dimensione fissa delle istruzioni
- *Rendere più veloce l'evento più frequente*
 - l'indirizzamento relativo al PC per i branch (frequentissimi)
 - rendere veloci le istruzioni più frequenti

(maggiori dettagli seguono...)

72

Architetture RISC/CISC (1)

- Una statistica sulle frequenza di istruzioni
 - in due classici programmi
 - gcc, compilatore C GNU
 - spice, programma di simulazione di circuiti

Classe di istruzioni	Esempi MIPS	Istruzioni ad alto livello	Frequenza gcc	Frequenza in spice
Aritmetiche	add, sub, addi	assegnazioni	48%	50%
Trasferimento dati	lw, sw, lui	strutture dati (vettori)	33%	41%
Salti condizionati	beq, bne, slt, slti	istruzioni <i>if</i> e cicli	17%	8%
Salti incondizionati	j, jr, jal	chiamate a procedura	2%	1%

73

Architetture RISC/CISC (2)

- Il tipo di indirizzamento
 - nella architettura VAX che offre una varietà di tipi
- Uso dei salti

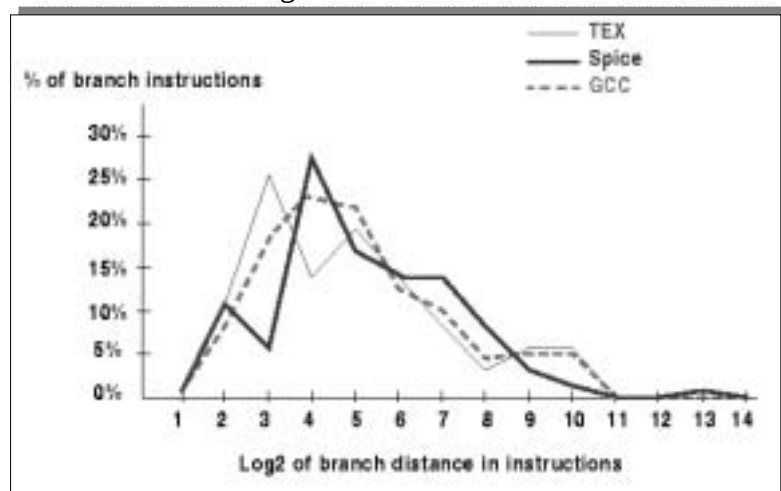
Tipo di indirizzamento	gcc	TeX	spice
Indicizzato	51%	56%	58%
Immediato	39%	43%	17%
Altri...	10%	1%	25%

Salti usati per..	gcc	TeX	spice
Condizioni	78%	66%	75%
Chiamata e ritorno da proc.	10%	16%	13%
Jump	12%	18%	12%

74

Architetture RISC/CISC (3)

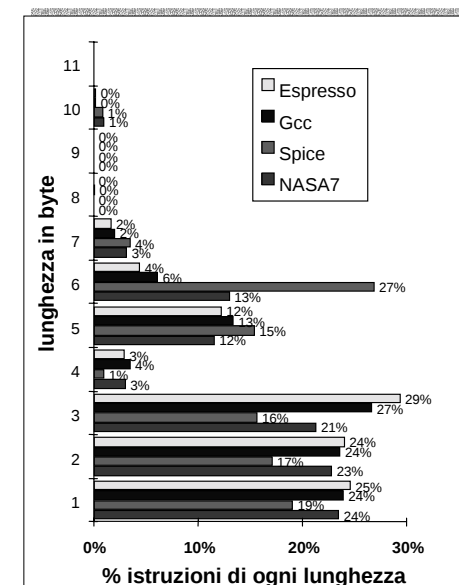
- Quanto lontano saltano le istruzioni di branch?
 - N.B. i dati sono in log base 2 !



75

Architetture RISC/CISC (4)

- Lunghezza delle istruzioni variabili
 - quanto sono usati in realtà?
- Per la architettura INTEL 80x86...
 - le istruzioni possono avere lunghezza da 1 a 17 byte
 - ma, in genere, la maggior parte delle istruzioni hanno lunghezza da 1 a 4 byte



76

Architetture RISC/CISC (5)

- Importante “rendere veloce il caso più frequente”!
- Reduced Instruction Set Computer (RISC)
 - architetture dove sono implementate in maniera semplice le istruzioni più frequenti, in modo da avere una maggiore velocità di esecuzione
- Complex Instruction Set Computer (CISC)
 - architetture dove sono a disposizione molte istruzioni, con formati diversi, etc.
 - un obiettivo: minimizzare la lunghezza del codice e quindi la occupazione di memoria
 - Esempi: le istruzioni di Intel 80x86 possono essere lunghe da 1 a 17 byte, quelle VAX da 1 a 54 byte

77

Esercizi

- Riscrivere il programma che calcola un array con i numeri di Fibonacci, utilizzando una routine che riceve (in \$a0 e \$a1) due interi e ne restituisce (in \$v0) la somma
- Scrivere come vengono tradotti dall’assemblatore le istruzioni blt, bge, bgt, ble

78