Progetto di Laboratorio di Architettura degli Elaboratori

Anno accademico 2007/2008

Realizzatori:

Marco Peca matricola 182746
Claudio Soprano matricola 181099

Analisi del problema

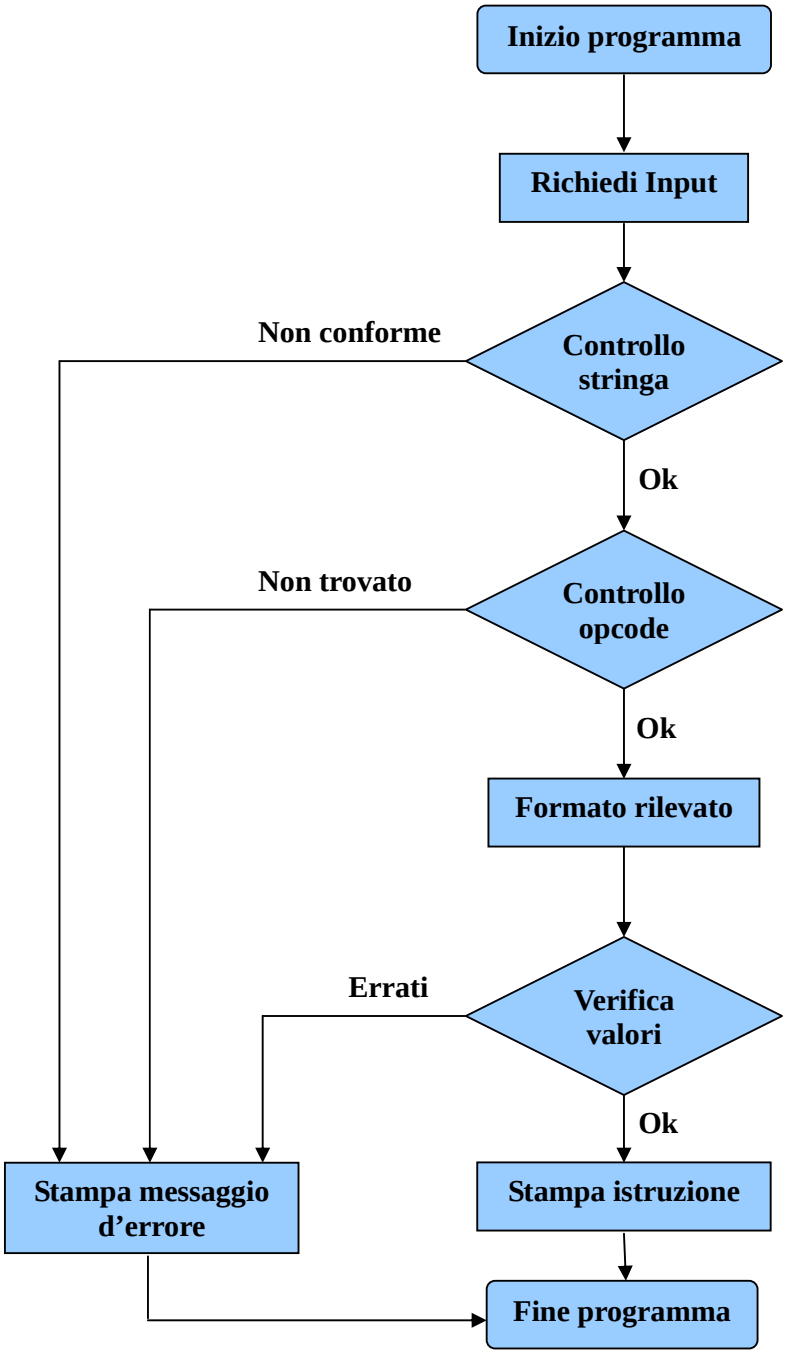
Il nostro progetto consiste nel convertire un codice binario, dato in input, nella relativa istruzione in MIPS.

Abbiamo scelto di immagazzinare il codice binario in ingresso in una stringa; questa decisione è sembrata ovvia perché un intero non andrebbe a considerare nella parte più significativa gli zeri.
Abbiamo da subito escluso il confronto tra stringhe perché troppo dispendioso e poco pratico; convertire i vari campi dell'istruzione MIPS in valore decimale è stato di gran lunga più facile e flessibile.
La parte .data è stata molto utilizzata, poiché ci ha permesso la creazione di array di interi e array di puntatori molto importanti per il nostro pensiero di risoluzione del problema, perché facili da utilizzare per il confronto ed estremamente rapidi nell'aggiungere nuove istruzioni da decodificare.

Abbiamo previsto dei registri “globali” $si che saranno visibili ed eventualmente modificabili in tutto il programma; questo perché alla chiamata di una specifica procedura, i valori precedentemente accumulati nei registri serviranno per calcolare il valore decimale o per puntare alla funzione che ci permette di andare avanti nella decodifica del codice binario: per questo motivo non verranno mai salvati nello stack.
Salvare nello stack comporterebbe, dopo una procedura, il ripristino del vecchio valore o indirizzo, mentre a noi serve invece continuare il programma con le modifiche apportate a tali registri lungo il loro percorso.
Ovviamente siamo stati attenti a utilizzare gli stessi registri per lo stesso tipo di funzioni:
$s0 - contiene l'indirizzo del carattere “0”.
$s2 - contiene l'indirizzo di partenza della stringa str inserita dall'utente.
$s3 - contiene l'indirizzo della stringa relativa al comando MIPS inserito.
$s4 - contiene l'indirizzo della stringa corrispondente al registro rs.
$s5 - contiene l'indirizzo della stringa corrispondente al registro rt.
$s6 - contiene l'indirizzo della stringa corrispondente al registro rd o l'indirizzo o il valore immediato.
$s7 - è il contatore che permette di spostarci nei vettori relativi ai codici operativi e alle funzioni.

Descrizione dei passi

La stringa inserita viene controllata per lunghezza e per validità binaria, con le eventuali modalità di errore.
Dopo la considerazione di un'immissione corretta, viene analizzato il campo relativo al codice operativo dell'istruzione. Tale campo è fondamentale per la successiva decodifica dell'istruzione, infatti il suo valore verrà usato per differenziare i tre formati: R, I e J.
Una volta determinato il formato relativo all'istruzione inserita, la stringa viene scomposta in diversi campi (composti da più bit) nello stesso modo in cui lo farebbe il MIPS. Se l'istruzione corrisponde ad una di quelle in elenco e se i valori dei vari campi sono corretti, si procede alla stampa dell'istruzione corrispondente, altrimenti verrà restituito un messaggio d'errore.



Spiegazione delle istruzioni

Il **main** è la parte del programma che svolge i seguenti compiti:

- Immissione stringa da parte dell'utente.

```
li $v0, 8                    # Mette in $v0 il codice di servizio per la lettura di una stringa
move $a0, $s2                # Copia in $a0 l'indirizzo di destinazione (“str”)
li $a1, 33                   # Carica in $a1 la lunghezza della stringa (32 + terminazione)
syscall                      # Carica la stringa
```

- Verifica codice operativo tramite procedura **checkstropcode**.

```
jal checkstropcode
```

- Stampa un messaggio di errore o una newline.

```
move $a0, $v0                # Copia in $a0 la stringa di errore o una newline
li $v0, 4                    # Copia in $v0 il codice di servizio per la stampa di una stringa
syscall                      # Stampa la stringa

li $v0, 10                   # Termina esecuzione programma
syscall
```

La procedura **checkstropcode** ha come parametro in ingresso la stringa data in input e svolge i seguenti compiti:

- Verifica se la stringa inserita è composta da caratteri binari e di avere lunghezza pari a 32, altrimenti restituisce un messaggio d'errore.

```
lb $t2, 0($a0)               # Estrae byte da verificare
beq $t2, $zero, finewhilec # Se il carattere è di terminazione esce dal ciclo
beq $t2, $s0, avantic        # Se il carattere è 0=30 allora va avantic
beq $t2, $t1, avantic        # Se il carattere è 1=31 allora va avantic
...
li $t0, 32                   # Carica in $t0 il valore decimale 32
beq $v1, $t0, continueopcode # Verifica se la stringa è lunga 32 caratteri
```

- Calcola il codice operativo tramite la procedura **stringtodec**.

```
move $a0, $s2                # Copia in $a0 l'indirizzo di partenza della stringa
li $a1, 5                    # Carica in $a1 indica l'indice di partenza del codice operativo
move $a2, $zero              # Carica in $a2 l'indice dove si fermerà il calcolo
jal stringtodec              # Richiama la procedura
```

- Verifica se il codice operativo è tra quelli riconosciuti, altrimenti restituisce un messaggio d'errore.

```
la $t0, opcode               # $t0 punta al primo elemento dell'array contente gli opcode
...
beq $t1, $v0, exitcycle      # Confronta il valore del codice operativo con ogni  elemento di $t1
```

La procedura **stringtodec** ha come parametri in ingresso la stringa data in input ($a0), l'indice superiore ($a1) e l'indice inferiore ($a2); ha il compito di calcolare il valore decimale del campo di bit identificato dai parametri in ingresso:

```
add $t0, $a0, $a1          # $t0 punta alla posizione $a0 + $a1
lb $t1, 0($t0)             # $t1 contiene il byte meno significativo a partire da $t0
...
add $v0, $v0, $t2          # Se $t1 = 1 allora $v0 = $v0 + $t2
...
add $t2, $t2, $t2          # Incrementa l'esponente di una base 2
```

La procedura **checkcomp2** è stata creta per i valori negativi nel MIPS. Ha come parametri in ingresso la stringa data in input ($a0), l'indice superiore ($a1) e l'indice inferiore ($a2). Svolge i seguenti compiti:

- Controlla se il valore immesso è positivo o negativo.

```
add $t0, $a0, $a2          # Si posiziona sul bit piu' significativo
...
lb $t0, 0($t0)             # Estrae il bit piu' significativo
bne $t0, $t1, numpositivo  # Se il bit piu' significativo e' <> '1' allora il numero e' positivo
```

- Se negativo complementa i bit e aggiunge 1.

```
lb $t1, cuno               # Carica in $t1 il carattere '1'
lb $t2, 0($t0)             # Estrae il bit più significativo
bne $t2, $t1, numpositivo  # Se il bit più significativo è <> '1' allora il numero e' positivo
...
sb $t1, 0($t0)             # Il carattere '0' diventa '1'
...
sb $s0, 0($t0)             # Il carattere '1' diventa '0'
...
jal stringtodec            # Richiama la procedura per calcolare il valore decimale
addi $v0, $v0, 1           # Aggiunge uno al valore numerico per finire il complemento a 2
sub $v0, $zero, $v0        # Trasforma il valore in negativo
```

- Se positivo richiama la procedura per calcolare il valore decimale.

```
jal stringtodec
```

La procedura **extractreg** ha come parametri in ingresso la stringa data in input ($a0), l'indice superiore ($a1) e l'indice inferiore ($a2); ha il compito di ritornare al chiamante una stringa corrispondente al nome del registro relativo al contenuto del campo di bit dato in ingresso.

```
jal stringtodec            # Richiama la procedura per calcolare il valore decimale
la $t0, registri           # $t0 punta all'inizio dell'array registri
sll $v0, $v0, 2            # $v0 x 4
add $t0, $t0, $v0          # $t0 punta all' $t0[i] posizione
lw $v0, 0($t0)             # Estrae la stringa dalla memoria
```

La procedura **formator** serve a gestire le istruzioni del formato R. Viene richiamata dalla procedura **checkstropcode** dopo esser stata indirizzata dall'array **formattype**. Non ha nessun parametro in ingresso. Svolge i seguenti compiti:

- Estrae il campo funct e verifica se è una funzione riconosciuta, altrimenti restituisce un messaggio d'errore.

```
move $a0, $s2                # $a0 punta all'inizio di str
li $a1, 31                   # $a1 punta alla fine di str
li $a2,26                    # $a2 punta all'indice inferiore
jal stringtodec              # Converte la stringa funct in numero
move $s3, $v0                # Mette il valore decimale della funzione in $s3
...
lw $t2, 0($t0)               # Estrae codice funzione da array functr[$t0] e lo mette in $t2
...
beq $t2, $s3 endloopfunctr   # Se $t2 = $s3 significa che abbiamo trovato la funzione nell'array
```

- Estrae le stringhe dei vari registri

```
...
jal extractreg               # Richiama la procedura per l'estrazione della stringa relativa ai registri
```

- Estrae lo shamt

```
move $a0, $s2                # $a0 punta all'inizio di str
li $a1, 25                   # $a1 punta all'indice superiore
li $a2, 21                   # $a2 punta all'indice inferiore
jal stringtodec              # Richiama la procedura per calcolare il valore decimale
```

- Salta al case dello switch relativo alla funzione trovata; i vari case fanno un controllo per verificare se i parametri hanno valori congrui.

```
la $t1, switchr              # Si posiziona al primo elemento dell'array switchr
add $t1, $t1, $s7            # $t1 punta all'elemento relativo alla funzione selezionata
lw $t1, 0($t1)               # Estrae l'indirizzo dello switch da eseguire
jr $t1                       # Salta al case relativo alla funzione selezionata
```

- Richiama la procedura di stampa.

```
lw $a0, 0($t1)               # Estrae l'indirizzo del case dello switch di stampa e lo mette in $a0
jal printmips                # Chiama la procedura di stampa
```

La procedura **formatoi** serve a gestire le istruzioni del formato I. Viene richiamata dalla procedura **checkstropcode** dopo esser stata indirizzata dall'array **formattype**. Non ha nessun parametro in ingresso. Svolge i seguenti compiti:

- Estrae la stringa relativa all'opcode specificato

```
la $t0, instructions         # $t0 punta all'elenco delle istruzioni
add $t0, $t0, $s7            # $t0 si sposta avanti di opcode posizioni
lw $s3, 0($t0)               # $s3 contiene la stringa relativa all'opcode richiesto
```

- Estrae le stringhe dei vari registri oppure calcola il valore/indirizzo

```
...
jal extractreg               # Richiama la procedura per l'estrazione della stringa relativa ai registri
```

- Salta al case dello switch

```
la $t1, switchother          # si posiziona al primo elemento dell'array switchother
add $t2, $t1, $s7            # $t2 punta all'elemento relativo alla funzione selezionata
lw $t2, 0($t2)               # estrae l'indirizzo dello switch da eseguire
jr $t2                       # va al case relativo alla funzione selezionata
```

- Richiama la procedura di stampa

```
lw $a0, 0($t1)               # Estrae l'indirizzo del case dello switch di stampa e lo mette in $a0
jal printmips                # Chiama la procedura di stampa
```

La procedura **formatoj** è relativa per il formato J. Viene richiamata dalla procedura **checkstropcode** dopo esser stata indirizzata dall'array **formattype**. Non ha nessun parametro in ingresso. Svolge i seguenti compiti:

- Estrae la stringa relativa all'opcode specificato

```
la $t0, instructions         # $t0 punta all'elenco delle istruzioni
add $t0, $t0, $s7            # $t0 si sposta avanti di opcode posizioni
lw $s3, 0($t0)               # $s3 contiene la stringa relativa all'opcode richiesto
```

- Estrae le stringhe dei vari registri oppure calcola il valore/indirizzo

```
move $a0, $s2                # $a0 punta all'inizio di str
li $a1, 31                   # $a1 punta all'indice superiore
li $a2, 6                    # $a2 punta all'indice inferiore
jal stringtodec              # Richiama la procedura per calcolare il valore decimale
```

La procedura **printmips** ha come unico parametro d'ingresso l'indirizzo del salto da effettuare per andare ad eseguire il case corretto.

```
move $t2, $a0                # Sposta in $t2 il salto da fare
jr $t2                       # Salta in $t2
...
...                           # Vari casi di stampa
...
move $v0, $t0                # Ritorna in $v0 la newline
```

Le pseudo-istruzioni

Nel nostro programma ci siamo serviti di non molte pseudo-istruzioni.
In diversi punti ci siamo serviti dell'istruzione “move”:

move $1, $2

che ha il compito di copiare il contenuto di $2 nel registro $1. Tale istruzione può essere così convertita nelle istruzioni di base MIPS:

```
addu $1, $zero, $2
```

Un'altra pseudo-istruzione è stata utilizzata per copiare in un registro il byte di una stringa; in modo formale l'istruzione è scritta così:

lb $2, label

L'istruzione è tradotta in questo modo:

```
lui $1, 4097      # 4097 è l'inizio della memoria
lb $2, 0($1)      # Il valore prima di $1 cambia a seconda dello scostamento  dall'inizio della memoria
```

I problemi riscontrati e le relativi risoluzioni

Durante lo svolgimento del progetto abbiamo riscontrato diversi problemi.

Primo fra tutti il debug, piuttosto difficile da gestire col MIPS perché bisogna controllare passo dopo passo lo scostamento dei registri nella memoria; trovare un errore è piuttosto difficile anche se hai ben chiaro l'intero codice del programma.
Diciamo che per questo problema è servita solo un po' di praticità.

Un altro problema è stato l'input da tastiera, infatti considerando che la stringa da prendere in esame deve essere lunga 32 caratteri (standard MIPS), bisognava aggiungere alla lunghezza complessiva un carattere in più per l'identificatore di fine stringa (carattere 00). Nella versione per Windows limitando la lunghezza della stringa di inserimento a 33 caratteri, lo Spim non permetteva di inserire via console caratteri oltre il 32-esimo, inserendo automaticamente il carattere di fine stringa, era quindi impossibile inserire stringhe piu’ lunghe, mentre Mars (altro emulatore assembler MIPS) lo permetteva ma poi troncava a 32 caratteri inserendo il fine stringa. È stato un problema perché volendo controllare per eccesso la lunghezza della stringa, questo non era possibile.
Il problema è stato risolto andando a escludere l'eccesso come fa qualsiasi altro sistema di input; per esempio in una form internet, i caratteri oltre la lunghezza predefinita dal programmatore vengono ignorati. Risulta quindi inutile fare un controllo in questo senso perché il programma comunque non li considererebbe.