ISTRUZIONI MARS

Dott. Franco Liberati

Argomenti

01 Struttura di una istruzione MIPS

02 Il set delle istruzioni
- Spostamento
- Logiche aritmentiche
- Confronto e settaggio,
- Salto
- Macchina

03 Pseudoistruzione

Struttura di una istruzione

Struttura di una istruzione

Una istruzione in linguaggio assemblativo MIPS ha la seguente struttura:

INDIRIZZO etichetta: direttiva/istruzione[opcode,addressing mode] # commento

Struttura di una istruzione

Indirizzo

L'INDIRIZZO è la locazione di memoria dove risiede l'istruzione. Poiché nel MIPS le istruzioni sono di 32bit e le locazioni fisiche della memoria sono di 8bit ogni istruzione risiede in un indirizzo multiplo di quattro

L'INDIRIZZO dove risiede la prima istruzione del programma in MARS è alla posizione 4194304 (locazione stabilita in fase di caricamento)

I dati, nel MARS, risiedono a partire dalla locazione 268500992

Text Segment

Bkpt	Address	Code	Basic	Source		
	4194304	0x3c011001	lui $1,4097	11:	lw $t0, valore1	#Prelievo del primo operando dalla locazione di memoria con etichett...
	4194308	0x8c280000	lw $8,0($1)			
	4194312	0x3c011001	lui $1,4097	12:	lw $t1, valore2	#Prelievo del secondo operando dalla locazione di memoria con etiche...
	4194316	0x8c290004	lw $9,4($1)			
	4194320	0x01095020	add $10,$8,$9	13:	add $t2,$t0,$t1	#Somma degli operandi e scrittura del risultato nel registro $t2
	4194324	0x3c011001	lui $1,4097	14:	sw $t2,risultato	#Salvataggio dell'addizione nella locazione di memoria risultato
	4194328	0xac2a0008	sw $10,8($1)			
	4194332	0x2402000a	addiu $2,$0,10	15:	li $v0, 10	#Richiesta del servizio di interruzione del programma
	4194336	0x0000000c	syscall	16:	syscall	#Attivazione del servizio

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	23	12	0	0	0	0	0	
268501024	0	0	0	0	0	0	0	

Organizzazione Memoria MARS

Il MARS ha una Memoria suddivisa logicamente in segmenti.
A partire dalla locazione 4194304 è presente il **Text Segment** in cui è presente il programma
Con inizio 268500992 c'è il **Data Segment**, in cui sono allocati i dati.
Dalla locazione 268697600 inizia l'area **heap**, ovvero la parte riservata per le strutture dati dinamiche (liste, grafi,…).
Lo **stack** comincia da 2147479529; l'area in mezzo è occupata in base alle necessità.

Oltre a questo ci sono anche delle locazioni per la memorizzazione temporanea dei dati del **kernel** (da 2415919104) e per lo scambio di informazioni con le periferiche **MMIO** (da 4294902016).

MEMORIA MARS
RESERVED
TEXT
DATA
HEAP
FREE
STACK
KDATA
MMIO

Struttura di una istruzione

Il MIPS segue una **sintassi di tipo *line-oriented***, ovvero ogni istruzione è scritta su una singola riga. L'unica eccezione è riservata all'etichetta che può risiedere sulla stessa linea dell'istruzione o prima di essa senza altre informazioni laterali (etichetta isolata) ad eccezione dei commenti

Etichetta	Direttiva/istruzione	Commento
	.text	#Direttiva del Segmento Istruzioni
	.globl main	
main:		#Etichetta main: inizio del programma **NB: ETICHETTA ISOLATA**
	lw $t0,valore1	#Etichetta valore1 fa riferimento all'area dati
	lw $t1,valore2	
	beq $t0,$t1,uguali	#Etichetta uguali (e fine) fa riferimento all'area istruzioni
	li $t2,1	
	j fine	
uguali:		**#NB: ETICHETTA ISOLATA**
	li $t2,0	
fine:	**li** $v0,10	#Richiesta del servizio di terminazione del programma (con etichetta in linea)
	syscall	#Attivazione del servizio
	.data	#Direttiva del Segmento Dati
	valore1: .word 23	#Definizione di una variabile (operando intero a 32bit e inizializzazione a 23)
	valore2: .word 12	#Definizione di una variabile (operando intero a 32bit e inizializzazione a 12)

Direttive

Una **direttiva** (*directive*) è un identificatore con un punto iniziale (**.**directive) che indica all'assemblatore di svolgere alcune operazioni preliminari, come ad esempio allocare spazio per un operando (cioè definire una variabile); stabilire la funzione di inizio del programma; marcare la parte riservata ai dati e quella relativa alle istruzioni; archiviare sequenze di istruzioni o dati nel *kernel* (la parte essenziale) del Sistema Operativo; comporre macro; ed altro

Direttiva	Significato
.text	Sequenza delle istruzioni da archiviare nel Text Segment
.data	Sequenza di operandi da archiviare nel Data Segment
.align	Allineamento del dato successivo a 8bit (0), 16bit (1), 32bit (2), 64bit (3)
.ascii	Definizione di una stringa nel Data Segment priva del terminatore
.asciiz	Definizione di una stringa nel Data Segment provvista del terminatore '0'
.byte	Definizione di un operando intero di 8bit nel Data Segment
.half	Definizione di un operando intero di 16bit nel Data Segment
.word	Definizione di un operando intero di 32bit nel Data Segment
.double	Definizione di un operando reale di 64bit con rappresentazione IEEE754 nel Data Segment
.float	Definizione di un operando reale di 32bit con rappresentazione IEEE754 nel Data Segment
.space *n*	Riserva uno spazio di memoria nel Data Segment del numero di byte (*n*) specificato
.set	Inizializzazione di una variabile
.macro *etichetta*	Inizio della definizione di una macro
.end_macro	Fine della definizione di una macro
.eqv	Sostituzione del primo elemento con il secondo: il primo elemento è un simbolo, il secondo è una espressione (ha lo stesso funzionamento di #define nel linguaggio ad alto livello C)
.external	Dichiarazione di variabili globali
.globl	Dichiarazione di etichette globali
.include "*path***"**	Inserimento del contenuto di un file. Il percorso e nome del file (*path*) è riportato all'interno delle virgolette (es.: .include "C:\\MARS\\numero_massimo.asm")
.ktext	Sequenza delle istruzioni da archiviare nel Kernel Data Segment
.kdata	Sequenza dei dati da archiviare nel Kernel Istruction Segment

Direttive

In fase di assemblaggio il codice assemblativo è scisso in due sezioni: il **Segmento Testo** (*Text Segment*), in cui sono presenti le istruzioni, e il **Segmento Dati** (*Data Segment*) dove sono definite, in modo sequenziale, le variabili; ovvero le locazioni di memoria in cui risiedono gli operandi

Questa suddivisione è consentita, ed indicata, rispettivamente dalle direttive **.text** e **.data**

Nel MIPS, questa suddivisione avvantaggia la dislocazione dei due sottocodici nella memoria contenente le istruzioni (*Instruction Memory*) e in quella in cui risiedono gli operandi (*Data Memory*); suddivisione che consente un miglioramento delle prestazioni grazie all'uso della canalizzazione

Direttive

Definizione delle istruzioni: .text

Il linguaggio assemblativo MIPS consente la scrittura del programma, e quindi delle istruzioni, dopo la direttiva **.text**

Di solito il programma utilizza anche una direttiva iniziale (**.global** *etichetta*) per rendere il programma accessibile da altri sottoprogrammi (l'etichetta è posta all'inizio del programma).

Esempio:

```
        .text
        .global main

main:
        ...
        ...
```

Direttive

Etichette in .text ed Etichette in .data

Nel Segmento Dati, una etichetta ha il significato di una locazione di memoria (corrisponde a una variabile in un linguaggio ad alto livello e per questo motivo si può parlare di **nome della variabile**); mentre nel Segmento Testo rappresenta l'indirizzo di una istruzione; ed è usata per reindirizzare il flusso di esecuzione quando, ad esempio, si effettua un salto

	Etichetta	Direttiva/istruzione	Commento
Segmento testo		**.text**	**#DIRETTIVA DEL SEGMENTO TESTO**
		.globl main	#Direttiva che rende etichetta main globale
	main:		#Etichetta in vece di un indirizzo
		lw $t0,valore1	#Lettura dell'operando dalla memoria
		mul $t1,$t0,$t0	#Elevamento al quadrato dell'operando
		sw $t1,risultato	#Salvataggio risultato in memoria
		li $v0,10	#Richiesta del servizio di terminazione del programma
		syscall	#Attivazione del servizio
Segmento dati		**.data**	**#DIRETTIVA DEL SEGMENTO DATI**
		valore1: .word 35	#Definizione della variabile valore1 (operando intero di 32bit)
		risultato: .word 0	#Definizione della variabile risultato (operando intero di 32bit)

Direttive

Definizione dei dati

Il linguaggio assemblativo è senza tipizzazione dei dati (*typeless*). Questo significa che il **tipo degli operandi è determinato dalle istruzioni** e non dalle variabili

Le direttive usate indicano solo lo spazio di memoria che deve essere allocato e l'intervallo dei valori rappresentabili:

.byte, intero (in complemento a due) di 8bit
.half, intero (in complemento a due) di 16bit
.word, intero (in complemento a due) di 32bit

.float (32bit) e .**double** (64bit) per numeri reali rappresentati in virgola mobile IEEE754

.asciiz e **.ascii** per stringhe, cioè una sequenza di byte contigui rappresentanti caratteri alfanumerici, con terminatore e senza

Direttive

Definizione dei dati

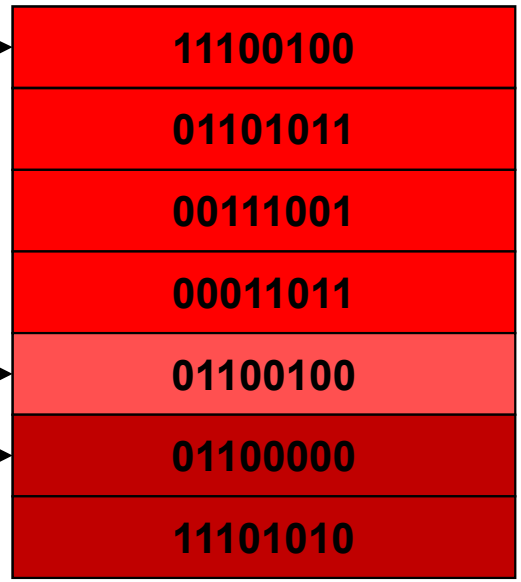
La sintassi è *etichetta* **: .word** *operando*

Esempio:

Batman : .word *456748004* (0001 1011 0011 1001 0110 1011 1110 0100)

Superman: .byte 100 (0110 0100)

WonderWoman : .hafword *60000* (1110101001100000)



Direttive

Allineamento dati in memoria

La direttiva **.align** consente di allineare la prossima variabile ad un multiplo di quattro byte (*word boundary*).

In MIPS, come detto, si lavora con parole a 32bit e la dichiarazione delle variabili avviene in locazioni contigue. Quindi avere variabili site in locazioni con indirizzi che non sono multipli di quattro può generare confusione, errori o rallentamenti.

Pertanto se per gli operandi di tipo *word* non c'è alcun problema, in quelli di tipo *half* è opportuno che inizino in una locazione con indirizzo multiplo di quattro e quindi c'è bisogno di un *boundary* di due byte, mentre per i dati ad 8bit è richiesto un boundary di tre byte

Direttive

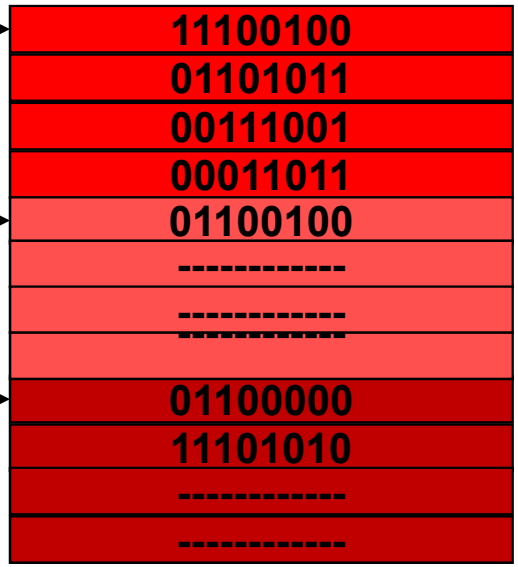
Definizione dei dati: allineamento

La sintassi è

etichetta: **.align** spazio_in_byte

Esempio:

Batman : .word *456748004* (0001 1011 0011 1001 0110 1011 1110 0100)
Superman: .byte 100 (0110 0100)
space1 : .align 3
WonderWoman : .half *60000* (1110101001100000)
space2 : .align 2



Direttive

Definizione delle costanti numeriche

Le **costanti numeriche** usate per inizializzare le variabili hanno uguale sintassi ai linguaggi C, C++ e Java

0xcifre	Interi espressi in esadecimale
0cifre	Interi espressi in ottale
Cifre	Interi espressi in decimale
Parte Intera**.**parteDecimale	Numeri reali in virgola mobile

Esempio:

```
Batman: .word 0x10      #valore numerico in decimale 16
Superman: .word 020     #valore numerico in decimale 16
WonderWoman: .word 16   #valore numerico in decimale 16
Flash: .float 16.0      #valore numerico decimale reale 16.0
```

Direttive

.EQV

La direttiva **.eqv** è utile per una maggiore comprensione del codice: simula la definizione di costanti ed è usata per rinominare i registri.
In fase di assemblaggio c'è la sostituzione tra identificativo e registro o costante numerica.

Sintassi

.eqv *identificatore* valore/registro

Direttive

EQV

Calcolo del perimetro di un quadrato

	Etichetta	Direttiva/istruzione	Commento
1		**.eqv** NUM_LATI 4	#Definizione di una costante indicante il numero di lati
2		**.eqv** PERIMETRO $t0	#Rinomina di un registro in cui riportare il perimetro
3		**.eqv** SYS_CLOSE_PROGRAM 10	#Definizione di una costante per il servizio di terminazione #del programma
4		**.text**	#Direttiva del Segmento Testo
5		**.globl** main	#Direttiva per definire il main globale
6	**main:**		#Etichetta *main*: inizio del programma
7		**lw** $t1,dim_lato	#Prelievo della dimensione del lato
8		**mul** PERIMETRO, $t1, NUM_LATI	#Calcolo del perimetro di un quadrato (in $t0 c'è il risultato)
9		**li** $v0, SYS_CLOSE_PROGRAM	#Richiesta del servizio di terminazione del programma
10		**syscall**	#Attivazione del servizio
11			
12		**.data**	#Direttiva del Segmento Dati
13		dim_lato: .word 0xbe	#Definizione di una variabile che specifica la dimensione di #un lato (intero a 32bit inizializzato a 190)

Direttive

EQV: risoluzione in fase di pre-assemblaggio

Calcolo del perimetro di un quadrato

	Etichetta	Direttiva/istruzione	Commento
1		**.eqv** NUM_LATI 4	#Definizione di una costante indicante il numero di lati
2		**.eqv** PERIMETRO $t0	#Rinomina di un registro in cui riportare il perimetro
3		**.eqv** SYS_CLOSE_PROGRAM 10	#Definizione di una costante per il servizio di terminazione #del programma
4		**.text**	#Direttiva del Segmento Testo
5		**.globl** main	#Direttiva per definire il main globale
6	**main:**		#Etichetta *main*: inizio del programma
7		**lw** $t1,dim_lato	#Prelievo della dimensione del lato
8		**mul** $t0, $t1, 4	#Calcolo del perimetro di un quadrato (in $t0 c'è il risultato)
9		**li** $v0, 10	#Richiesta del servizio di terminazione del programma
10		**syscall**	#Attivazione del servizio
11			
12		**.data**	#Direttiva del Segmento Dati
13		dim_lato: .word 0xbe	#Definizione di una variabile che specifica la dimensione di #un lato (intero a 32bit inizializzato a 190)

Direttive

MACRO

La direttiva **.macro** consente di riscrivere più volte le stesse istruzioni sfruttando un modello, da definire prima del programma e da riportare al suo interno
Il modello è formato da un identificatore di identificazione e il blocco di istruzioni da riutilizzare
La definizione di una macro, come conseguenza, offre un codice più leggibile e comprensibile

Sintassi:

.macro ***identificatore***
istruzione$_1$
…
istruzione$_n$
.end_macro

Direttive

Esempio: MACRO

Calcolo del perimetro di un rettangolo (uso di una macro per la terminazione del programma)

	Etichetta	Direttiva/istruzione	Commento
1		**.eqv** NUM_LATI 2	#Definizione di una costante indicante il numero di lati uguali
2		**.eqv** PERIMETRO $t0	#Rinomima di un registro in cui riportare il perimetro
3			
4		**.macro** *END_PROGRAM*	#Macro per la terminazione del programma
5		li $v0,10	#Richiesta del servizio
6		syscall	#Attivazione del servizio
7		**.end_macro**	#Fine della macro
8			
9		**.text**	#Direttiva del Segmento Testo
10		**.globl** main	#Direttiva per definire il main globale
11	**main:**		#Etichetta *main*: inizio del programma
12		**lw** $t1,dim_lato_piccolo	#Prelievo della dimensione del lato piccolo
13		**lw** $t2,dim_lato_grande	#Prelievo della dimensione del lato grande
14		**mul** $t3, $t1, NUM_LATI	#Somma dei lati piccoli
15		**mul** $t4, $t2, NUM_LATI	#Somma dei lati grandi
16		**add** PERIMETRO, $t3,$t4	#Nel registro $t0 si trova il perimetro del rettangolo
17		***END_PROGRAM***	#Macro per la terminazione del programma (chiamata)
18			
19		**.data**	#Direttiva del Segmento Dati
20		dim_lato_piccolo: .word 0xa	#Definizione di una variabile che indica la dimensione del lato #piccolo inizializzata a 10 (rappresentazione esadecimale)
21		dim_lato_grande: .word 075	#Definizione di una variabile che indica la dimensione del lato #grande inizializzata a 61 (rappresentazione ottale)

Direttive

INCLUDE

La direttiva **.include *"path"*** permette di inserire il contenuto di un file all'interno del programma (path identifica la posizione del file nel sistema di calcolo)

È una strategia utile per includere librerie, funzioni generiche o costanti note (es.: pigreco) in un programma senza doverli riscrivere o ridefinire

Sintassi:

.include *"path"*

Esempio:

```
.include "C:/MARS/FunzioniPersonali/MyLibrerie.asm"
```

Istruzioni

Generalità

Le **istruzioni** (*instruction*) manipolano indirizzi o operandi

Sono formate da un menmonico (add, addition-addizione; sub, subtraction - sottrazione; mul, multiplication – moltiplicazione; lw, load word, caricamento di un dato a 32bit…) e da un modo di indirizzamento (dove solo locati i dati da elaborare)

Esempio:

add $t2,$t0,$t1

I numeri reali, che hanno rappresentazione in virgola mobile secondo lo standard IEEE754, sono elaborati dal coprocessore matematico e meritano una discussione a parte

Commenti

Generalità

Un **commento** (*comment*) è una stringa a cui si antepone il simbolo #; mentre **un blocco di commenti** è formato da più linee di commento

I commenti svolgono un ruolo essenziale nella programmazione in linguaggio assemblativo (nella fase di collegamento ed esecuzione non sono riportati) perché è l'unico mezzo per spiegare agli altri programmatori, o ai generici utenti, come opera l'algoritmo. Oltre a questo, la leggibilità del codice e la comprensione dei programmi è offerta da etichette ed identificatori chiari ed attinenti al loro ruolo, da un ridotto numero di istruzioni e da una ordinata scrittura del listato

Sintassi:

Commento

```
# Questo è un commento
```

Blocco di commento

```
########################################
# Questo è un blocco di commenti
########################################
```

FINE