MIPS

Struttura base del programma. In questo caso, abbiamo un esempio del semplice **return** che abbiamo su C. Cioè una funzione **main** che non ha altro che **return 0** come funzione interna.

Variabile globale

Direttiva **.globl** per la variabile globale "standard", mentre **.local** per la variabile globale statica, che però *non* è supportata sullo **SPIM**.

```
    .text
    .globl main
main:
    jr $31
    nop     # branch delay slot
```

Delay Slot - Cosa sono e come comportarsi

Esistono due tipo di **Delay**:

- Dopo un'operazione di salto
- Dopo un'operazione di **lw**/**sw**

Nel primo caso, abbiamo un **Delay slot** causato dall'istruzione **jr**. Il branch delay slot è un effetto indesiderato della pipeline, che non può sapere se un'istruzione di salto sia eseguita fino a quando l'istruzione di salto non sia stata realmente valutata. La pipeline deve decidere come gestire le istruzioni successive al salto che attraversano la pipeline. Questo tipo di situazione può essere trattata in due modi modi differenti:

1. Dopo l'istruzione **jr** si inserisce l'istruzione **nop**
2. Dopo l'istruzione **jr** si inserisce un istruzione che va eseguita qualsiasi sia l'esito dell'operazione di salto

Nel secondo caso abbiamo invece un **Delayed load slot**. In pratica, una volta eseguita una **lw/sw**, il risultato dell'istruzione non può essere utulizzato dall'istruzione successiva. Anche in questo caso, possiamo comportarci in due differenti modi:

1. Dopo l'istruzione si inseriesce l'istruzione **nop**
2. Dopo l'istruzione, si inserisce un istruzione che non fa utilizzo del risultato dell'istruzione precedente

```
#Codice non ottimizzato
lw $t1,0($t2)
nop
addi $t3, $t3, 1

# Codice ottimizzato
lw $t1,0($t2)
addi $t3, $t3, 1

Tramite il lw carico un dato nel registro t1 ma, visto che non
devo utilizzare quello specifico registro nell'istruzione successiva,
sfrutto la nope per eseguire un'altra istruzione che non dipende dal
registro di destinazione della lw
```

Dichiarazione e definizione di variabili - Variabili non automatiche

Prima di dichiarare una riabile, si deve indicare la sezione di cui farà parte. Abbiamo tre possibili sezioni:

- .**data**: variabili leggibili/scrivibili inizializzate
- .**bss**: variabili leggibili/scrivibili non inizializzate o inizializzate a zero
- .**rdata**: variabili solamente leggibili

Lo SPIM riconosce solamente la sezione **.data** e la sezione **.rdata**

Dichiarazione: [etichetta:] [.tipo]

L'etichetta indentificat un indirizzo per mezzo di un nome. Le direttive dell'assemblatore GNU per definire il tipo sono:

- .**byte**: dimensione pari ad un byte;
- .**word**: dimensione pari a 4 byte;
- .**asciiz**: stringa di caratteri

Variabili non inizializzate od inizializzate a zero

Non serve specificare un tipo, ma bisogna dichiarare lo spazio occupato dalla variabile per mezzo della direttiva **.space**. Si utilizza tipicamente per variabili della sezione .bss, che non occupa spazio all'interno del file oggetto.

```
.rdata #Dati in sola lettura

.data #Dati in lettura e scrittura

  *etichetta*:   #Nome che voglio da associare al dato

    .word x y z 1 2 0x3 0x4    #Singole parole oppure vettori (lista di parole)

    .asciiz "*Stringa*"   #Stringhe di caratteri

#.bss
  *etichetta*:

    .space n  #Riservo uno spazio di n byte, non inizializzati oppure inizializzati a 0
```

Calling convenction

La **funzione chiamante** salva tutti i registri potenzialmente modificabili dalla *funzione chiamata* che utilizza:

- I registri **$t(0-9)** se utilizzati;
- I registri argomento **$a(0-3)** e **$v(0-1)** se ne riutilizza i valori al ritorno dell'invocazione;
- Il registro **$ra;**

La **funzione chiamata** salva:

- Il registro **frame pointer $fp** se utilizzato;
- I registri **$s(0-7)** prima del loro utilizzo;

I registri salvati vengono infine ristabiliti.

Stack

Lo **Stack** cresce verso il "basso", cioè da inidirzzi di memoria grandi ad inidirizzi di memoria più piccoli. Per allocare spazio si sottrae la quantità di byte voluta al registro $sp. Per rilasciare spazio, si somma la quantità di byte voluta al registro $sp. Il suo valore deve essere allineato ad 8 byte.

Passaggio dei parametri

Ogni parametro più piccolo di 32 bit è esteso a 32 bit. I primi quattro parametri sono passati nei registri **$a0-$a3**, mentre i successivi devono essere passati per mezzo dello **stack** (lasciando 16 byte inutilizzati in cima quando si alloca spazio).

Valore di ritorno

- Valore a 32 bit, nel registro **$v0**
- Valore a 64 bit, nei registri **$v1** e **$v0**

Invocazione di procedura

- **Stack frame**: regione dello stack che contiene tutti i dati relativi ad una procedura
- **Actiation Record (AR)**: regione dello stack allocata prima di invocare una funzione (ovvero prima della sua "attivazione")
- **Operazioni sullo stack**: *Push* -> salvataggio di elementi sullo stack | *Pop* -> rimozione di elementi dallo stack

Chiamante - Chi invoca la funzione

Call Setup: fase precedente all'invocazione

- Allocare spazio sullo stack (Activation Record)
- Salvare i registri **$ra** e **$fp**;
- Porre **$fp** uguale a **$sp**;
- Salvare i registri non preservati dalla funzie chiamata (solo quelli necessari);
- Supponendo K argomenti, posizionare gli argomenti da K-1 a 4 sullo Stack (dal basso verso l'alto) ed i restatnti da 0 a 3 nei registri **$a3, $a2, $a1, $a0** (per K<4 bastano i registri);
- Eseguire l'istruzione **jal** per invocare la funzione voluta

Return cleanup: fase successiva al ritorno

- Copiare il contenuto dei registri di ritorno **$v(0-1)** (se precedentemente salvati);
- Ristabilire il contenuto del registro **$sp** ed i registri salvati sullo stack

Chiamato - Funzione invocata

Prologo

- Salvare sullo stack il regitro **$fp**;
- Salvare il valore dello stack pointer nel registro **$fp**;
- Salvare anche **$ra**

Epilogo

- Porre il valore di ritorno in **$v(0-1)**;
- Ripristinare i registri salvati;
- Eseguire l'istruzione di return **jr $ra**;

Scheletro funzione

```
.rdata #Dati in sola lettura

.data #Dati in lettura e scrittura

 *etichetta*:   #Nome che voglio da associare al dato

    .word x y z    #Singole parole oppure vettori (lista di parole)

    .asciiz "*Stringa*"   #Stringhe di caratteri

    .space n  #Riservo uno spazio di n byte, non inizializzati oppure inizializzati a 0

.text
.globl main

main:

    #Calling convenction - Call setup
    #Alloco spazio sullo Stack

    addi sp, sp, -8
    sw fp, 4(sp)
    sw ra, 0(sp)
    move fp, sp

    la $a0, *etichetta* # Carico l'indirizzo del tipo di dato associato all'etichetta

    jal *funzione_da_invocare* #Chiamo la funzione

    #Calling convenction - Return cleanup
    #Ripristino i valori dello Stack

    move sp, fp
    lw ra, 0(sp)
    lw fp, 4(sp)
    addi sp, sp, 8

    jr $ra
```