Introduzione a MIPS64
(II parte)

Chiamate di Sistema

`syscall n`

- Utilizzate come interfaccia col sistema operativo, funzioni diverse (n = 0..5)

- Sono simili alle chiamate `exit()`, `open()`, `close()`, `read()`, `write()`, `printf()`

Chiamate di sistema

- **I parametri di una syscall devono essere posti, consecutivamente, in un indirizzo che va specificato in R14**
- **Il valore di ritorno sarà posto in R1**

```
.data
primo_parametro: .......
secondo_parametro: ......
...

.code
...
daddi r14, r0, primo_parametro
syscall n
```

Syscall 0 – exit()

```
syscall 0
```

- Termina il programma, non ha argomenti di input e di output

syscall 1 - open()

`syscall 1`

- Significato: *apre un file*
- Due parametri:
  - L'indirizzo di una stringa terminata con byte 0 che indica il path del file da aprire
  - Un intero che specifica la modalità di apertura

    O_RDONLY (0x01) Opens the file in read only mode;

    O_WRONLY (0x02) Opens the file in write only mode;

    O_RDWR (0x03) Opens the file in read/write mode;

    O_CREAT (0x04) Creates the file if it does not exist;

    O_APPEND (0x08) In write mode, appends written text at the end of the file;

    O_TRUNC (0x08) In write mode, deletes the content of the file as soon as it is opened.
- Valore resituito:
  - Un intero corrispondente al file descriptor della risorsa aperta

Syscall 1: esempio

```
.data
primo_parametro: .asciiz "dati.txt   ;nomefile
secondo_parametro:  .space 8         ;modalità di apertura

.code

daddi r9,r0, 1       ; read mode
sd r9, secondo_parametro(r0)
daddi r14, r0, primo_parametro
syscall 1
daddi r15, r1, 0   ; copia in r15 il descrittore del file
```

syscall 2 – close()

`syscall 2`

- Significato: *chiude un file precedentemente aperto*
- Un parametro:
  - Un intero corrispondente al file descriptor
- Valore restituito:
  - Codice che indica il successo o meno (vedere manuale EduMIPS64)

Es.
```
daddi r14, r15, 0  ; copia in r1 il descrittore
syscall 2
```

syscall 3: read()

`syscall 3`

- Significato: *legge un certo numero di byte da un file ponendoli in una parte di memoria*
- Tre parametri:
  - Un intero corrispondente al file descriptor (NB: 0=standard input)
  - Un indirizzo di memoria dove porre i byte letti
  - La quantità di byte da leggere
- Valore restituito:
  - Il numero di bytes letti, -1 in caso di errore

Syscall 3: esempio

```
.data
Buffer: .space 4

par:   .space 8 ; descrittore
ind:   .space 8 ; indirizzo di partenza
num_byte: .word 4


.code
   sd r0,par(r0);  stdin
   daddi r11, r0, buffer
   sd r11, ind(r0) ; buffer
   daddi r14, r0, par
   syscall 3              ;read
```

syscall 4: write()

`syscall 4`

- Significato: *scrive su file un insieme di byte letti da una certa regione della memoria*
- Tre parametri:
  - Un intero corrispondente al file descriptor del file su cui scrivere (NB: 1=standard output)
  - L'indirizzo di memoria a partire dal quale leggere
  - Il numero di byte da scrivere
- Valore restituito:
  - Numero di byte scritti o -1 in caso di errore

syscall 5: printf(...)

`syscall 5`

- Significato: *stampa un messaggio sulla base di una stringa di formattazione (stile `printf` C)*
- Variabile numero di parametri:
  - Il primo è l'indirizzo della stringa di formattazione, terminata con byte 0
  - I successivi, per ogni segnaposto (es %d), seguono subito dopo
- Valore restituito:
  - Numero di byte stampati, -1 in caso di errore

Esempio utilizzo syscall 5

```
; NB: esempio con due valori, uno pronto l'altro no
.data
mess:    .asciiz "Stampa i numeri %d e %d"
arg_printf:      .space 8
num1:            .word 255
num2:            .space 8
str:             .asciiz "stringa non usata"

.code
addi r8, r0, mess
sd r8, arg_printf(r0)
addi r9, r0, 50  ; scelta di 50 arbitraria
sd r9, num2(r0)
addi r14, r0, arg_printf

syscall 5
syscall 0
```

Eseguire step-by-step con F7 osservando i registri e la memoria

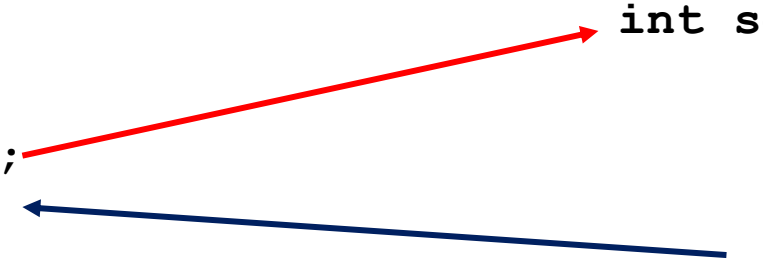
Chiamata a Procedura

- Procedura chiamante

```
.....
if(a<= 10)
   s=somma(a,b);
if(s>10)
.....
```

Procedura chiamata

```
int somma( int x, int y)
    {
        int t;
        t=x+y;
        retun t;
    }
```



- Cosa succede?
  - Cambio di flusso
  - Passaggio di informazioni
- Come vengono implementate nell'Assembly ?

Chiamata a Procedure

- Si utilizzano `JAL` e `JR`

```
        ...
i:      JAL proc_chiamata
i+4:    ...
```

- jal, prima di saltare all'indirizzo indicato, setta R31 a PC +4 (prossima istuzione)

```
proc_chiamata:
….
JR R31
```

Alla fine, prima terminare, la procedura salta all'indirizzo indicato in R31

Ok, molto bello ma...

Se la procedura chiama un'altra procedura ?!?

Chiamate annidate

```
ProcA:
…..
JAL procB
….
JR R31
```

```
ProcB:
….
JR R31
```

Ipotizziamo che inizialmente la procA venga invocata da una `jal procA` che si trova all'indirizzo 1028 della sezione `.code`, riusciremo mai a continuare sull'istruzione successiva che si trova a PC+4 ossia 1032 ?

Stack pointer

- In un stack (pila) vengono accumulati i valori salvati da recuperare, R29 indica la cima.
- NB: lo stack cresce verso l'alto (indirizzi più piccoli)

Chiamate con stack pointer

```
ProcA:
DADDI R29,R29,-8
SD R31 0(R29)
JAL procB
LD R31,0(R29)
DADDI R29,R29,8
JR R31
```

```
ProcB:
….
JR R31
```

Se la procA è invocata dall'indirizzo 1028, il valore 1032 è salvato sullo stack e poi recuperato

Aggiornamento stack pointer

- Ogni procedura che chiama un'altra deve occuparsi di aggiornare lo stack pointer e salvare quello che andrà recuperato

Chiamata a Procedura

- La procedura chiamante deve
  - predisporre i parametri d'ingresso
  - Trasferire il controllo alla procedura chiamata
- La procedura chiamata deve
  - Allocare lo spazio di memoria necessario all'esecuzione e salvare i registri utilizzati dalla procedura
  - Eseguire l'operazione richiesta
  - Conservare il risultato in un area accessibile al chiamante
  - Ripristinare i registri precedentemente salvati nello stack e liberare la memoria precedentemente allocata
  - Restituire il controllo di flusso

Passaggio dei parametri della procedura

Il passaggio dei parametri può avvenire copiando nei registri r4-r7 i parametri della procedura chiamata

Es. La chiamata a una funzione con 3 parametri

func_chiamata(p1,p2,p3)

può essere realizzata

1) copiando nei registri r4, r5 e r6 i valori di p1, p2 e p3
2) Chiamando la funzione mediante la jal

```
daddi r4, r0, p1
daddi r5, r0, p2
daddi r6, r0, p2
jal func_chiamata
```

Allocazione della memoria e salvataggio dei registri

- Durante l'esecuzione della procedura chiamata vengono utilizzati registri che potrebbero essere utilizzati anche dalla procedura chiamante.
- Per evitare che il valori dei registri del chiamante vengano persi definitivamente è necessario salvare i registri modificati dal chiamante nel preambolo della procedura.
- E' necessario preventivamente allocare una certa quantità di memoria per questo salvataggio.
- L'allocazione della memoria viene realizzata decrementando lo Stack Pointer (r29) della quantità di memoria necessaria al salvataggio dei registri

Allocazione della memoria e salvataggio dei registri

- Esempio. Nel caso in cui la procedura chiamata debba modificare i registri r8, r9, r10, essi vengono salvati nello stack nel seguente modo

```
daddi  r29, r29, -16  ;allocazione della memoria
sd r8, 0(r29)
sd r9, 8(r29)
sd r10, 16(r29)
```

Risultato della procedura

- Per convenzione il risultato della funzione viene restituito nel registro R1

- Se ad esempio il risultato della procedure è nel registro R8, prima di terminare l'esecuzione il contenuto in R8 è copiato in R1

```
dadd R1, R0, R8
```

Ripristino dei valori dei registri precedentemente salvati e deallocazione della memoria allocata

- La deallocazione della memoria viene realizzata incrementando lo Stack Pointer (r29) della quantità di memoria precedentemente allocata al momento dell'attivazione della procedura
- Se precedentemente sono stati salvati i registri R8, R9, R10, in R29, R29+8 e R29+16, il ripristino avviene con

```
ld R8, 0(R29)
ld R9, 8(R29)
ld R10, 16(R29)
daddi  R29, R29, 16 ;deallocazione della memoria
```

Restituzione del controllo alla procedura chiamante

- Il controllo viene restituito alla procedura chiamante saltando all'indirizzo contenuto nel registro R31

```
JR R31
```

Esempio di chiamata a funzione (1/2)

```
#include <stdio.h>

int operazione( int a, int b, int c)
{
    int t;

    t=a+b;
    t=t*c;
    return t;
}

main()
{ int x,y,z;
  int ris;

  scanf("%d",x);
  scanf("%d",y);
  scanf("%d",z);

  ris=operazione(x,y,z);


}
```

Esempio di chiamata a funzione (21/2)

```
.data
x: .space 8
y: .space 8
z: .space 8
ris: .space 8
stack:  .space 32
.code
; allocazione dello stack
daddi r29, r0, 32

jal input_unsigned ;scanf("%d",x);
sd r1, x(r0)
jal input_unsigned ;scanf("%d",y);
sd r1, y(r0)
jal input_unsigned ; ;scanf("%d",z);
sd r1, z(r0)

; ris=operazione(x,y,z);
; predisposizione dei parametri d'ingresso
ld r4, x(r0)
ld r5, y(r0)
ld r6, z(r0)
;chiamata della procedura operazione
jal operazione
sd r1, ris(r0)
syscall 0
#include input_unsigned.s
```

```
operazione:

;allocazione e salvataggio registri
daddi r29, r29, -8
sd r8, 0(r29)

;esecuzione del corpo della procedura
dadd r8, r4, r5
mult r6, r8

; copia del risultato in r1
mflo r1

; ripristino dei registri e
deallocazione della memoria
ld r8, 0(r29)
daddi r29, r29, 8

; restituzione del controllo al
chiamante
jr r31
```