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

syscall 5
syscall 0
```

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

Chiamata a Procedure

- Si utilizzano `JAL` e `JR`

```
…..                    procA:
JAL procA              ….
….                     JR R31
```

- NB: jal, prima di saltare all'indirizzo indicato, setta R31 a PC +8 (prossima istuzione)

NB: 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 1024 della sezione `.code`, riusciremo mai a continuare sull'istruzione successiva che si trova a PC+8 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 1024, 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