



## Introduzione a MIPS64 (II parte)

### Chiamate di Sistema



#### `syscall n`

- Utilizzate come interfaccia col sistema operativo, funzioni diverse ( $n = 0..5$ )
- **Assunzioni importanti:**
  - **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**
- Sono simili alle chiamate `exit()`, `open()`, `close()`, `read()`, `write()`, `printf()`

# 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 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)

## 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 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 istruzione)

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