# Gestione Input/Output

**Architettura degli elaboratori**

**Esercitazione 9**

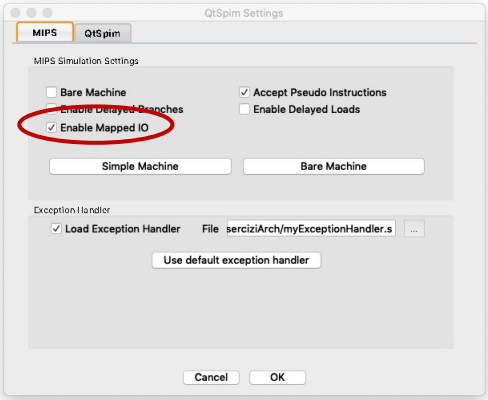
1

# I/O in SPIM

2

# I/O in SPIM

- SPIM simula una periferica di I/O (terminale/console) che permette a un programma MIPS in esecuzione su SPIM di:
  - leggere i caratteri digitati a tastiera
  - scrivere caratteri sulla console
- Mentre un programma è in esecuzione, SPIM collega il proprio terminale di I/O al processore
- Nelle impostazioni di SPIM deve essere abilitata l'opzione Enable Mapped I/O (v. immagine)




3

# SPIM terminal device

- Il terminale è composto da due dispositivi *distinti* e *indipendenti*:
  - Receiver: legge i caratteri ASCII dalla tastiera
  - Transmitter: visualizza caratteri ASCII sulla Console
- Le unità Receiver e Transmitter sono completamente **indipendenti**:
  - un tasto premuto sulla tastiera non viene automaticamente mostrato sulla console
  - un programma deve esplicitamente fare "eco" di un carattere letto dal Receiver inviandolo al Transmitter per mostrarlo sulla console

4

# Gestione del device SPIM terminal

- Per il controllo del device di I/O da parte di un programma sono utilizzati **4 registri memory-mapped**
- Ogni registro è in realtà un'area di memoria ad un indirizzo noto

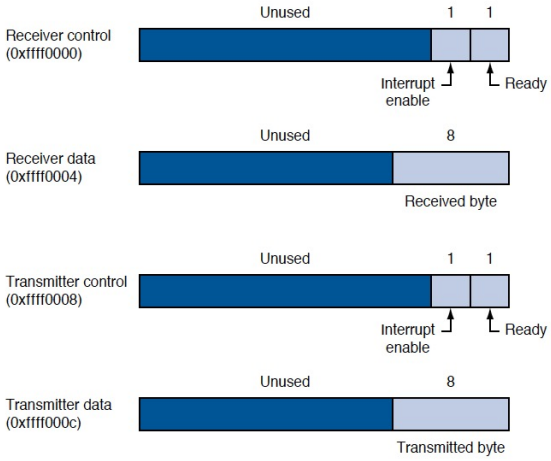
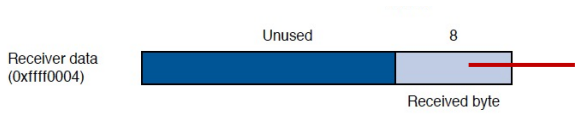



**FIGURE A.8.1 The terminal is controlled by four device registers, each of which appears as a memory location at the given address.** Only a few bits of these registers are actually used. The others always read as 0s and are ignored on writes.

5

# Receiver Data register (at address 0xffff0004)

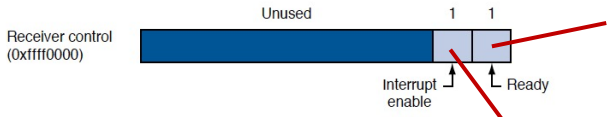



8 bit che contengono il codice ASCII dell'ultimo carattere digitato da tastiera

- registro read-only
- cambia quando un nuovo carattere è digitato a tastiera
- Il codice ASCII del tasto premuto è valido soltanto se Ready = 1
- Il valore in questo registro non è definito se il ready bit del registro Receiver Control è a 0
- La sua lettura riporta a 0 il ready bit del registro Receiver Control
- .

6

# Receiver Control register (address 0xffff0000)




Bit 0 (**keyboard ready bit**):

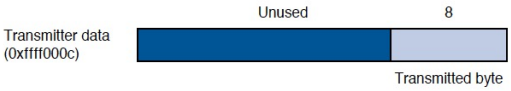
- è in sola lettura (le scritture vengono ignorate)
- passa da 0 a 1 quando è digitato un carattere da tastiera
- resta a 1, fintanto che il carattere non è prelevato

Bit 1 (**keyboard "interrupt enable" bit)**

- può essere sia letto che scritto da un programma
- inizialmente è a 0
- se il programma lo mette a 1, il terminale genera un interrupt ogni volta che è digitato un carattere
- Perchè gli interrupt abbiano effetto sul comportamento del processore, **gli interrupt devono anche essere abilitati nello Status register**

7

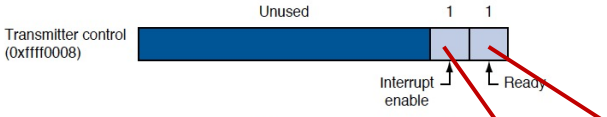
# Transmitter Data register (at address 0xffff000c)




- Quando un valore è scritto in questo registro,
  - gli 8 bit 0-7 di questo registro (i.e., codifica ASCII di un carattere) sono trasmessi alla console
  - il Ready bit del registro Transmitter Control è messo a 0. Rimarrà a 0, finchè sarà trascorso un tempo sufficiente per trasmettere il carattere al terminale di output e poi il Ready bit tornerà ad essere 1
- Può essere scritto quando il Ready bit del registro Transmitter Control è a 1
- Se Ready bit=0 (il trasmitter non è pronto), i dati scritti nel Transmitter Data register sono ignorati

8

## Transmitter Control register (at address 0xffff0008)




- usati solo 2 bit
- comportamento simile a Receiver Control register
- Bit 0 (**ready bit**)
  - read-only
  - se è a 1, il Transmitter è pronto per accettare un nuovo carattere
  - se è 0, il Trasmitter sta ancora scrivendo il carattere precedente
- Bit 1 (**interrupt enable bit**)
  - può essere letto o scritto
  - se è a 1, il terminale richiede un interruzione all'hardware quando il transmitter è pronto per un nuovo carattere (Ready bit a 1)
  - se il bit di Ready è 0, il carattere non sarà visualizzato e andrà perso

9

## Interrupt

- Sono tipicamente causati da dispositivi hardware esterni (come ad esempio le periferiche I/0), collegati al processore attraverso delle linee di controllo di bus
- Quando le periferiche sono pronte ad effettuare operazioni impostano una linea di interruzione collegata alla CPU
- Quando viene attivata una linea di interruzione, *se il sistema ha abilitato le interruzioni della CPU*, il normale flusso di esecuzione viene interrotto, e la CPU passa a gestire l'interrupt

10

# Registro Status

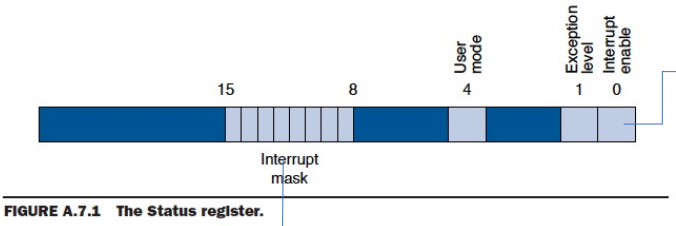



FIGURE A.7.1 The Status register.

Interrupt enable:
0 –interruzioni non abilitate
1 – interruzioni abilitate

Interrupt mask – **interrupt abilitati**

Un bit per ognuna delle 6 linee delle periferiche e
2 bit per interrupt a livello sw

11

# Interrupt abilitati

- All'interno dell'interrupt mask (dello **Status register**) ciascun bit corrisponde all'abilitazione di una differente linea di interruzione collegata ad una o più periferiche
  - Se il bit è impostato a 1, il corrispondente interrupt è abilitato e la periferica può generare una eccezione
  - Se il bit è impostato a 0, il corrispondente interrupt NON è abilitato e anche se la linea di interruzione è attivata dalla periferica, la CPU non genera l'eccezione
- Quando viene sollevata una eccezione a seguito di un interruzione abilitata, nel **Cause register** viene impostato a 1 il bit corrispondente all'interruzione (e i 5 bit dell'**exception code** sono impostati **a 0**)

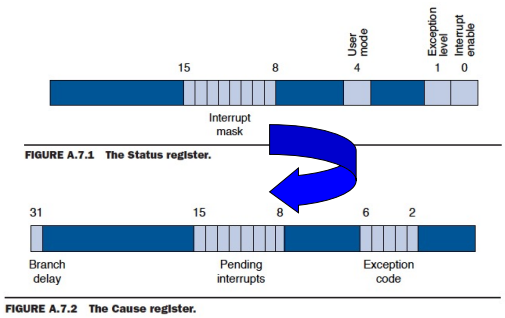



FIGURE A.7.1 The Status register.

FIGURE A.7.2 The Cause register.

12

# Esempio

Realizzare un programma assembly che stampa tramite syscall, un carattere inserito da tastiera (utilizzando i registri Receiver Control e Receiver Data) attraverso le due tecniche:

- **Busy Wait** (gestione da programma)
- **interrupt** (e quindi invocando l'exception handler)

13

# Esempio (busy wait)

```
.data
Messaggio:
        .ascii "Il tasto premuto e':"  #NB questa stringa non e'
                                       #terminata con byte 0!
Carattere:
        .asciiz "\n\n"      # due a capo (il primo sara' sovrascritto)
Fine:   .ascii "q"          # carattere da premere per terminare

.text
main:
        li $t0, 0xFFFF0000          # Receiver Control
        li $t2, 0xFFFF0004          # Receiver Data
        lb $t4, Fine                # carattere per finire
BusyWaitRead:       # "busy wait"
    lw   $t1, 0($t0)  # Receiver Control Register
    andi  $t1, $t1, 0x1          # tiene solo il bit 0
    beqz  $t1, BusyWaitRead  # se=0, non e' arrivato nulla
                             # se= 1, e' arrivato un carattere
     lb   $t3, 0($t2)            # legge byte dal Receiver Data
     beq   $t3, $t4, fine    # se e' il tasto di fine, esce
     sb    $t3, Carattere    # salva il carattere letto in memoria
                             #sovrascrivendo il primo 'a capo'
     li    $v0, 4        # stampa il messaggio con la syscall
     la    $a0, Messaggio    # il carattere e' incorporato !
     syscall                 # stampa con syscall
     j BusyWaitRead
fine:
    li   $v0, 10        #terminazione
    syscall
```

14

# Esempio (interrupt) – 1/2

```
# Porzione di codice dell'Exceptions.s per gestione interrupt
IntKbd:
 andi  $a0, $k0, 0x0100
          # Receiver mask: 0x0100 = 0000 0001 0000 0000
 beq   $a0, $0,  EndIntKbd
          # branch se e' interrupt ma non di Receiver

 # E' interrupt del Receiver, quindi ...
 # Stampa le info sull'interrupt
 li        $v0, 4
 la        $a0, __m5_
 syscall                    # print "  by Receiver "
```

```
# DISABILITA Interruzioni dal Receiver (tastiera)
 lw    $a0, 0xFFFF0000      # Leggi attuale Receiver Control
 andi  $a0, $a0, 0xFFFD     # Azzera bit 1 = INT enable
 sw    $a0, 0xFFFF0000      # Scrivi nuovo Receiver Control
 # DISABILITA Interrupt Receiver nella CPU
 mfc0  $a0, $12             # Leggi il CPU Status Reg.
 andi  $a0, $a0, 0xFEFF     # Azzera il bit INT del Receiver
 mtc0  $a0, $12             # Scrivi il nuovo CPU Status Reg.

 # Gestione del char digitato
 lw    $a0, 0xFFFF0004      # prelevo il carattere dal Data Reg.
 sw    $a0, BuffKbd         # e lo metto in memoria (buffer)

#(continua)
```

15

# Esempio (interrupt) – 2/2

```
EndIntKbd:
 # ABILITA interrupt della CPU
  mfc0  $t0, $12            # Leggi CPU Status register
  ori   $t0, $t0, 1         # abilita in generale gli INT
  ori   $t0, $t0, 0x100     # abilita il bit INT del Receiver
  mtc0  $t0, $12            # Scrivi nuovo CPU Status register

  # ABILITA Interrupt dal Receiver (tastiera)
  lw    $t1, 0xFFFF0000     # Leggi attuale Receiver Control
  ori   $t1, $t1, 2         # abilita il bit 1 (= INT enable)
  sw    $t1, 0xFFFF0000     # scrivi nuovo Receiver Control
```

16

# Esercizio 1

Determinare la percentuale di tempo in cui viene effettivamente utilizzata la CPU per trasferire **1 parola da una tastiera** utilizzando la tecnica di **gestione a controllo di programma** se:

- la tastiera trasferisce **10 byte/s**
- frequenza di clock: **50 MHz**
- sono richiesti **20 cicli di clock per ogni byte**

18

# Esercizio 1 – soluzione

- Poichè sono richiesti 20 cicli per trasferire ogni byte → sono richiesti 20·4 = 80 cicli della CPU per trasferire 4 byte
- Ma se il trasferimento della tastiera avviene 10 byte al sec → il tempo richiesto per il trasferimento di **4** byte è **4/10 = 0,4** sec
- In 0,4 sec, con una frequenza di 50 MHz, sono effettuati = $50 \cdot 10^6$ * 0,4 = **$2 \cdot 10^7$ cicli**
- **% uso effettivo CPU = ($80 / 2 \cdot 10^7$) = $4 \cdot 10^{-6}$ = 0,0004 %**

19

# Esercizio 2

Qual è la frazione del tempo del processore usata per la **gestione degli interrupt** necessari al trasferimento da/verso una periferica di I/O, se abbiamo:

- Frequenza di clock di **250 Mhz**
- Trasferimento di **4 Mbyte/s**, a blocchi di **4 word**
- Il costo di ogni interruzione è **500 cicli di clock**

20

# Esercizio 2 – soluzione

- Poichè il trasferimento è a blocchi di 4 word, avremo un interrupt ogni volta che sono trasferiti 16 byte (4 word)
- Poichè il trasferimento è di 4 Mbyte al secondo, avremo

  $4 \cdot 10^6$ byte/s / 16 byte = **250.000 interrupt/s**
- Poichè sono richiesti 500 cicli per gestire ogni interrupt, il costo totale della gestione delle interruzioni è:
  *250 k interrupt/s * 500 cicli/interrupt = **125.000 k cicli/sec***
- Frazione di utilizzo del processore per il trasferimento (interrupt):
  *125M/250M = **50%***

21

# Esercizio 3

Qual è la frazione del tempo del processore usata per la **gestione con DMA** di un trasferimento dati da periferica, se abbiamo:

- Frequenza di clock è **250 Mhz**
- Trasferimento di blocchi di **8 kbyte per ogni DMA**
- Trasferimento a **4 Mbyte/s**
- Il costo dell'inizializzazione del DMA è di **1000 cicli di clock**
- Il costo dell'interruzione al termine del DMA è di **500 cicli di clock**

22

# Esercizio 3 – soluzione

Per ciascun trasferimento DMA servono: (**1000 + 500) cicli di clock**

Numero di DMA/sec: 4 Mbyte/s / 8 kbyte = **500 DMA/sec**

Numero cicli_clock CPU richiesti: 1500*500 = 750.000 cicli/sec

Frazione del processore utilizzata per DMA: 750k / 250M = **0,3 %**

23

# Esercizio 4

Si supponga di voler trasferire un testo di 2 Mbyte e che

1. la CPU esegua 1 istruzione ogni ciclo di clock
2. il clock della CPU sia di 500 MHz (500*10^6 Hz)
3. il device emetta al massimo 1 KB/s (1000 byte/s)
4. impostare i registri per DMA richiede 10 cicli di clock
5. eseguire alcune istruzioni di prologo e epilogo per quanto riguarda il controllo di programma sia trascurabile

Qual è il rapporto tra il tempo di impiego della CPU nel caso di DMA e di I/O a controllo di programma?

24

# Esercizio 4 (soluzione 1/3)

Si supponga di voler trasferire un testo di 2 Mbyte e che

1. la CPU esegua 1 istruzione ogni ciclo di clock
2. il clock della CPU sia di 500 MHz (500*10^6 Hz)
3. il device emetta al massimo 1 KB/s (1000 byte/s)
4. impostare i registri per DMA richiede 10 cicli di clock
5. eseguire alcune istruzioni di prologo e epilogo per quanto riguarda il controllo di programma sia trascurabile

Qual è il rapporto tra il tempo di impiego della CPU nel caso di DMA e di I/O a controllo di programma?

Tempo di impiego della CPU nel caso di DMA?

Cicli di clock della CPU = 10 (4)

$$\text{Tempo CPU} = \frac{\text{num cicli clock CPU}}{\text{freq. clock}} = \frac{10}{500*10^6} = 2*10^{-8} \text{ sec}$$

25

# Esercizio 4 (soluzione 2/3)

Si supponga di voler trasferire un testo di 2 Mbyte (0) e che

1. la CPU esegua 1 istruzione ogni ciclo di clock
2. il clock della CPU sia di 500 MHz (500*10^6 Hz)
3. il device emetta al massimo 1 KB/s (1000 byte/s)
4. impostare i registri per DMA richiede 10 cicli di clock
5. eseguire alcune istruzioni di prologo e epilogo per quanto riguarda il controllo di programma sia trascurabile

Qual è il rapporto tra il tempo di impiego della CPU nel caso di DMA e di I/O a controllo di programma?

Tempo di impiego della CPU nel caso di I/O a controllo di programma?

Tempo CPU per esecuz. progr. dato da

per (1, 5) = $\frac{\text{Num. byte da trasmettere}}{\text{velocità emissione}}$

per (0, 3) = $\frac{2 * 10^6 \text{ byte}}{1*10^3 \text{ byte/sec}}$ =

= $2*10^3$ sec

Cicli di clock della CPU =

= Tempo CPU * frequenza_clock

per (2) = $2*10^3 * 500*10^6$

= $1000*10^9$

= $10^{12}$

26

# Esercizio 4 (soluzione 3/3)

Si supponga di voler trasferire un testo di 2 Mbyte e che

1. la CPU esegua 1 istruzione ogni ciclo di clock
2. il clock della CPU sia di 500 MHz (500*10^6 Hz)
3. il device emetta al massimo 1 KB/s (1000 byte/s)
4. impostare i registri per DMA richiede 10 cicli di clock
5. eseguire alcune istruzioni di prologo e epilogo per quanto riguarda il controllo di programma sia trascurabile

Qual è il rapporto tra il tempo di impiego della CPU nel caso di DMA e di I/O a controllo di programma?

Il rapporto tra num.cicli →

$\frac{\text{cicli CPU per DMA}}{\text{cicli CPU programmed I/O}} = \frac{10}{10^{12}} = 10^{-11}$

oppure tempo CPU

$\frac{\text{Tempo CPU DMA}}{\text{Tempo CPU prog. I/O}} = \frac{2*10^{-8}}{2*10^3}$

27

# Esercizio 5

Determinare la percentuale di tempo in cui verrebbe effettivamente utilizzata la CPU per trasferire **1 parola** (con gestione di programma) da un floppy disk e da un hard disk, tenendo conto che sono richiesti **20 cicli di clock per ogni byte**, che la **frequenza di clock è 50 Mhz** e

1. Floppy disk opera a **50 Kbyte/s**
2. Hard-disk lavora a **2 MByte/s**

28

# Esercizio 5 – soluzione

**Tempo di CPU richiesto per trasferire la word**: 20 * 4 = **80 cicli di clock**

1. **Floppy-disk:**

tempo di trasferimento: 4 byte / 50 kbyte/s = $0{,}08*10^{-3}$ s

$50*10^6$ hz x $0{,}08*10^{-3}$ = 4 x $10^3$ cicli_clock ➔ 80/4.000 = **2%**

2. **Hard-disk:**

tempo di trasferimento: 4 byte / 2Mbyte /s = $2*10^{-6}$ s

$50*10^6$ hz x $2*10^{-6}$ = 100 cicli_clock => 80/100 = **80%**

29

- Uteriori esercizi di trovano:
  - Quiz I/O
  - Laboratorio 8 – Gestione I/O

30