Lezione 16
Tecniche di pilotaggio dei dispositivi di I/O

- Le periferiche si differenziano notevolmente fra loro perché
  - Trasferiscono differenti quantità di dati
  - Funzionano a velocità diverse
  - Utilizzano 'formati di dato' differenti

- Tipicamente funzionano a velocità più basse rispetto a CPU e Memoria

- Per interfacciarsi con il resto del sistema una periferica ha bisogno di un "controller" o "interfaccia"

Velocità di trasferimento tipiche per dispositivi di I/O

Dispositivo	Vel.Trasferimento (MB/s)
Tastiera	0.00001
Mouse	0.0001
Modem 56k	0.007
Canale telefonico	0.008
Doppia Linea ISDN	0.016
Stampante Laser	0.1
Scanner	0.4
Ethernet classica	1.25
USB (Universal Serial Bus)(2.5W)	1.5
Cinepresa Digitale	4
Disco IDE	5
CD-ROM 40x	6
Fast Ethernet	12.5
Bus ISA	16.7
Firewire (IEEE 1394) (15W)	50
Monitor XGA	60
Rete SONET OC-12	78
Disco SCSI Ultra2	80
Gigabit Ethernet	125
Conventional PCI 32-bit/33MHz	133
Nastro Ultrium	320
Bus PCI-X (1998)	533
10xGigabit Ethernet (2002)	1250
HyperTransport 1.0 (2001)	6400
100xGigabit Ethernet (2010)	12500
Bus PCI-E (2004)	16000
QuickPath (QPI)	25600
HyperTransport 3.1 (2008)	51200

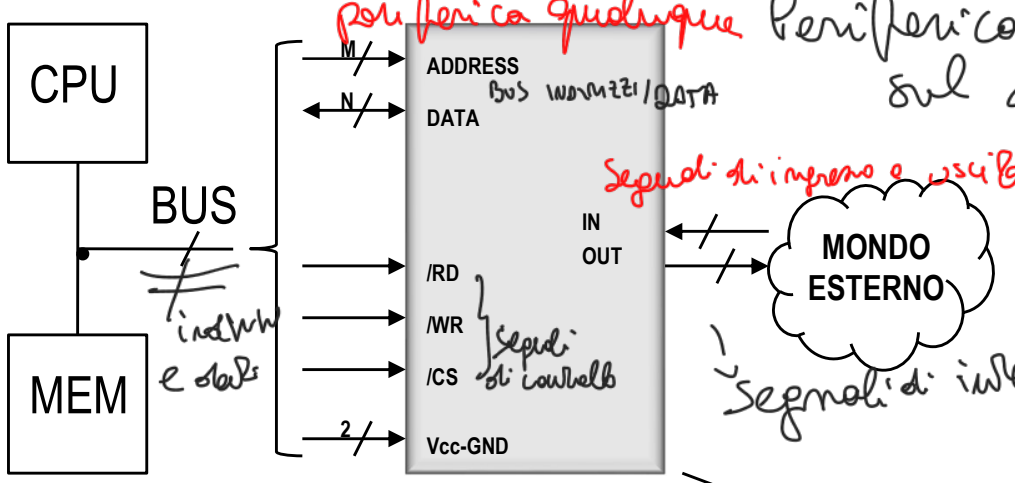
- **USB**
  - v1 (low speed) 1.5Mb/s=187 KB/s
  - v1.1 (full speed) 12Mb/s=1.5MB/s
  - v2.0 (hi-speed,2001) 480Mb/s=60MB/s
  - v3.0 (super speed,2009,4.5W) 5Gb/s=625MB/s
  - v3.2gen1 (superspeed,2013) 10Gb/s=1.25GB/s
  - V3.2gen3 (sperspeed,2019) 40Gb/s=5GB/s
- **SATA**
  - 1.5Gb/s (2001) → 130MB/s(HDD),200MB/s(SSD)
  - 3.0Gb/s (2001) → 200MB/s (reale)
  - 6.0Gb/s (2008) → 400MB/s (reale)
  - classica (1980) 10 MB/s = 1.25MB/s
- **ETHERNET**
  - fast (1995) 100MB/s = 12.5MB/s
  - Giga (1999) 1Gb/s = 125MB/s
  - 10xGiga (2002) 10Gb/s = 1.25GB/s
  - 100xGiga (2010) 100Gb/s = 12.5GB/s
  - 400xGiga (2017) 400Gb/s = 50GB/s
- **PCI-EXPRESS (PCI-E o PCIe)**
  - v1x (2004) 250MB/s(1lane),4GB/s(16-lane)
  - v2.0 (2007) 500MB/s(1lane),8GB/s(16-lane)
  - v3.0 (2010) 1GB/s(1lane),16GB/s(16-lane)
  - v4.0 (2017) 2GB/s(1lane),32GB/s(16-lane)
  - V5.0 (2019) 4GB/s(1lane),63GB/s(16-lane)
  - V6.0 (2022) 8GB/s(1lane),121GB/s(16-lane)
  - V7.0 (2025 – atteso) 16GB/s(1lane),242GB/s(16-lane)

Architettura del Sistema di I/O

- **L'architettura del sistema di I/O è… l'interfaccia fra il dispositivo e il processore**
- **I dispositivi di I/O sono costituiti da**
  - **Componenti sensori/attuatori (e.g. tastiera, display, disco, …)**
  - **Componenti elettronici (controller del dispositivo)**
  - **Componenti software (il "device driver")**
- **Compiti del controller**
  - **Eventuale controllo di più dispositivi**
  - **Convertire il flusso seriale di bit in BLOCCHI di byte**
  - **Effettuare eventuali correzioni di errore**
  - **Rendere disponibili i dati alla memoria principale o prelevare i dati dalla memoria principale**
- **Compiti del device driver**
  - **Inizializzare il controller/dispositivo**
  - **Gestire le operazioni fra controller/dispositivo e processore**

Nota: nel seguito si parlerà di "dispositivo" sottintendendo che ci si riferisce sempre al "controller del dispositivo"

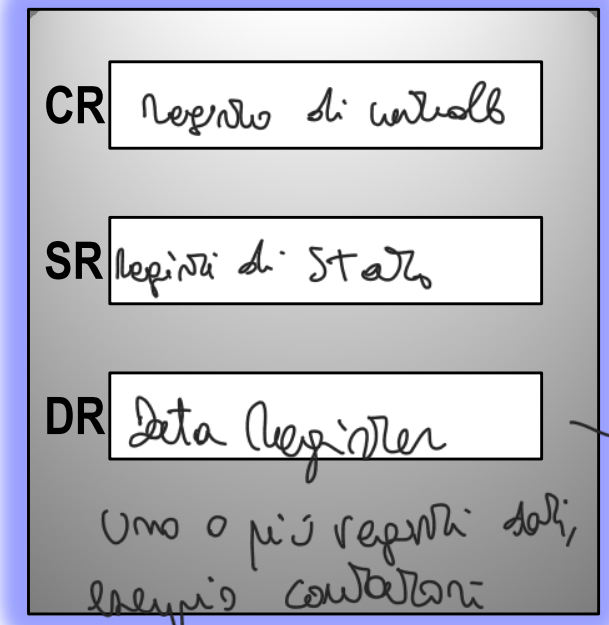
Visione elettrica e logica del dispositivo



Posso raggruppare i registri presenti in 3 categorie:

Il dispositivo viene visto dal processore come un insieme di locazioni dette "PORTE di I/O"

Vari registri



Porte di I/O tipicamente presenti in un dispositivo:

CR=CONTROL REGISTER
SR=STATUS REGISTER
DR=DATA REGISTER

nella porta seriale erano i dati che ricevo o trasmetto

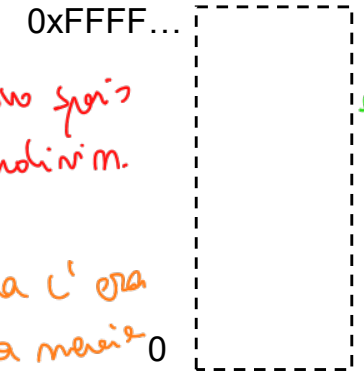
Metodi di comunicazione fra processore e dispositivo

(a) Spazi separati di I/O e memoria

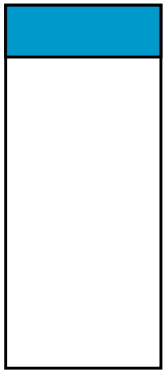
(b) Memory-mapped I/O

(c) Soluzione ibrida

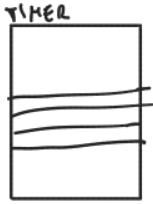
(metodo a)

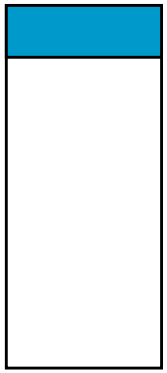


(metodo b)



(metodo c)





Esempio (architettura x86 antiche):
- IN AL,[0x60] → legge un byte all'indirizzo dello spazio di I/O 0x60

Esempio (architettura RISC-V):
- LB t0,0(0x60) → legge un byte all'indirizzo dello spazio di I/O 0x60 **mappato in memoria**

Esempio (architettura x86 moderne):
- IN AL,0x60 → legge un byte all'indirizzo dello spazio di I/O 0x60
- MOV AL,[0x60] → legge un altro byte all'indirizzo di memoria 0x60 (es. da un altro dispositvo)

Nei casi a e b, i dispositivi si basano solo sulle porte o sulla memoria rispettivamente

Nel caso c, il processore può parlare con i dispositivi usando uno dei due metodi o entrambi

Istruzioni per la comunicazione fra processore e dispositivo

- Se si usano i metodi (a) o (c) per scambiare dati fra processore e memoria si rendono necessarie
  - Istruzioni speciali di I/O (es. Architetture x86)
  - L'istruzione speciale specifica il numero del dispositivo
    - Questo può essere trasmesso "come un indirizzo" sul bus di I/O
  - L'istruzione speciale specifica il comando da dare al dispositivo
    - Questo può essere trasmesso "come un dato" sul bus di I/O

- Se si usa il metodo (b) non è necessario introdurre nuove istruzioni per comunicare con l'I/O

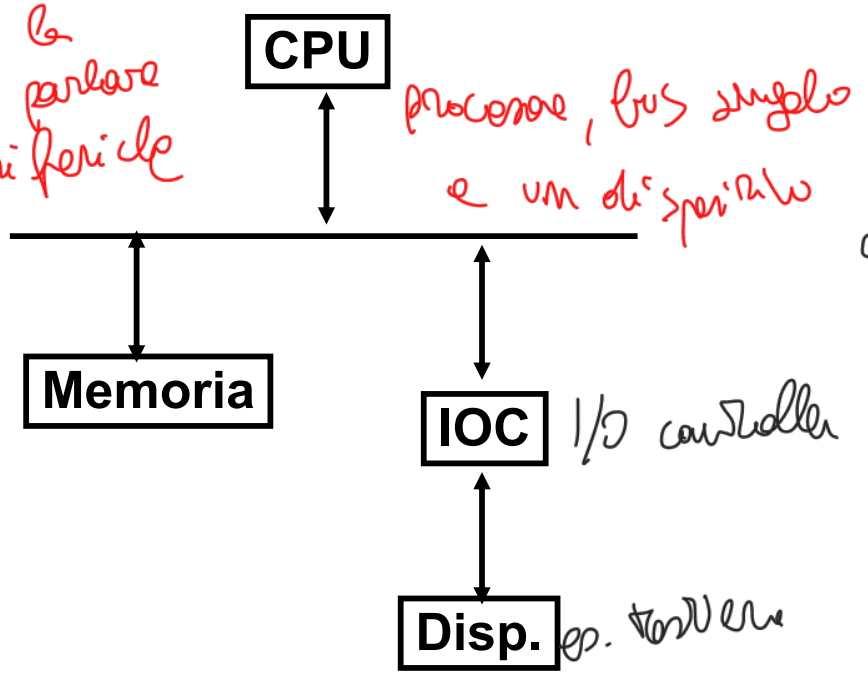
  riservo quegli indirizzi

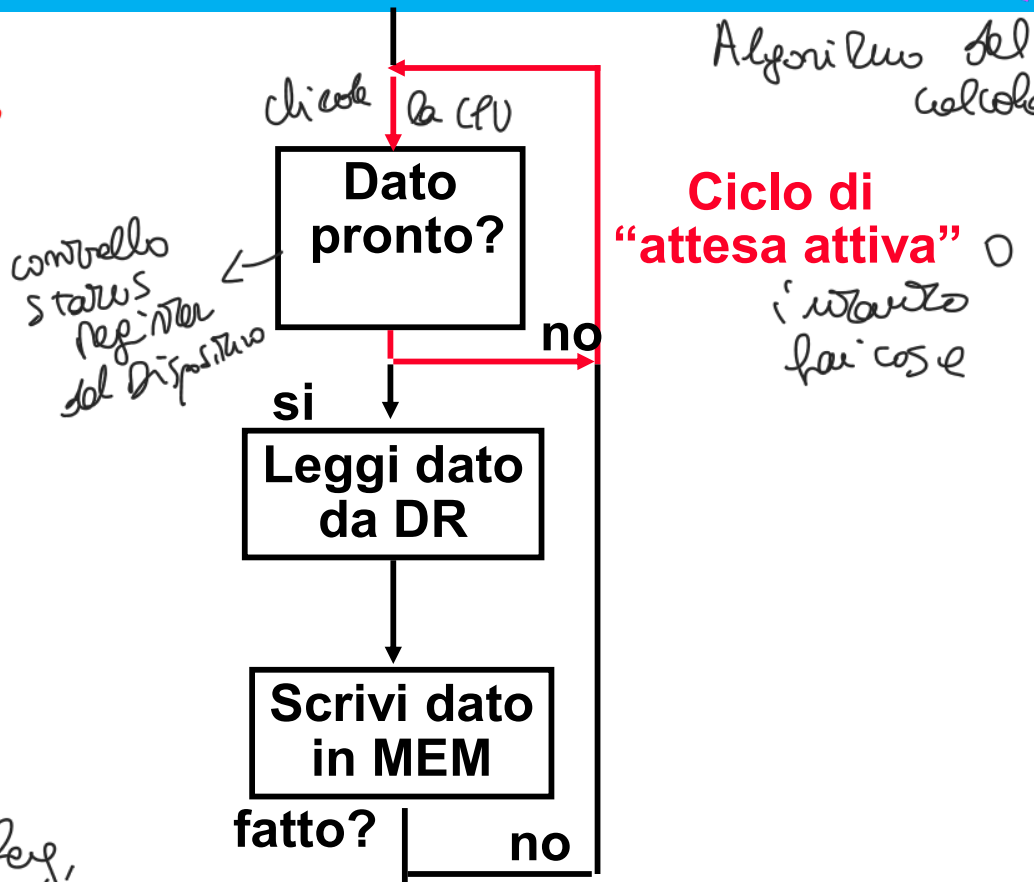
  - è sufficiente riservare certi indirizzi di memoria, non per mappare RAM ma per mappare indirizzi relativi al dispositivo
  - Letture e scritture a tali indirizzi sono interpretati dalla periferica come comandi
  - Si impedisce all'utente l'uso di questi indirizzi perché risiederanno in una zona riservata allo spazio di indirizzamento kernel
  - Questa tecnica è utilizzabile anche nel caso (c)

Protocolli di comunicazione fra processore e dispositivo

- Il processore ha bisogno di sapere quando
  - Il dispositivo ha completato un'operazione
  - Il dispositivo ha incontrato un errore
- Esistono tre protocolli fondamentali
  - **Polling**:
    - Il dispositivo mette il dato in un "data register" e segnala questo fatto tramite uno "status register"
    - Il processore controlla periodicamente lo "status register"
  - **Interrupt**:
    - Ogniqualvolta il dispositivo ha bisogno di attenzione dal processore, interrompe il processore tramite linea dedicata
  - **DMA/IOP**:
    - Il processore inizia l'operazione
    - Un processore ausiliario si occupa del trasferimento del dato
    - Al processore è notificato che l'operazione è finita con un interrupt dal DMA/IOP verso il processore

→ Protocollo di POLLING (o di I/O Programmato)



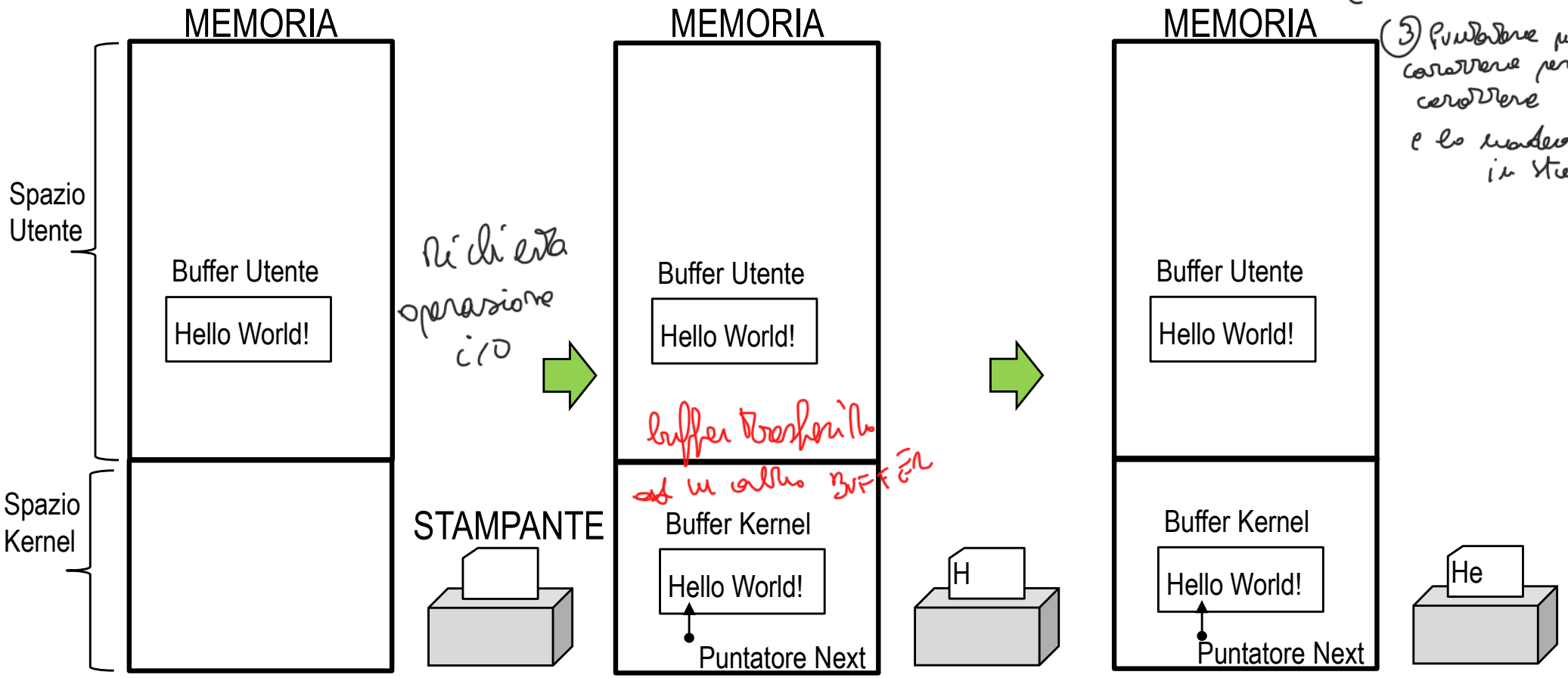


- **Vantaggio**
  - **Semplice: il processore ha il controllo totale e fa tutto il lavoro**
- **Svantaggio**
  - **La gestione del polling può consumare molto tempo del processore**
    - **Trucco: distribuire i controlli di I/O fra codice che effettua altre operazioni**

Polling - dettaglio

- La CPU richiede un'operazione di I/O
- Il controller di I/O effettua l'operazione di I/O
- Il controller di I/O attiva un bit nello "status register"
  - Il controller di I/O non informa direttamente la CPU
  - Il controller di I/O non interrompe la CPU
- La CPU guarda periodicamente il bit nello "status register"
  - La CPU può attendere oppure ritornare a vedere più tardi

Esempio di Polling: stampa di una stringa



Stampa di una stringa con Polling (Driver)

Faccio ecall
Es. ecall Copio stringa in spazio kernel

spazio kernel
spazio utente
caratteri stringa

```
copy_from_user(buffer, p, count);          /* p è un buffer nel kernel */
for (k=0; k<count; ++k) {                  /* ripeti per ogni carattere */
    while (*printer_status_reg != READY);  /* attendi il bit di READY */
    *printer_data_register = p[k];         /* uscita di un carattere */
}
return_to_user();
```

Driver a Polling

ciclo while infinito

nello status register

utente bloccato sulla ecall finché non si finisce

Notare il punto e virgola !

→ il driver che funziona a polling scandisce carattere per carattere verifico in STATUS REGISTER se è pronta la periferica (leggo il bit). Se sì, esco dal ciclo, STAMPO il carattere e quando finisco, ritorno il controllo all'utente, che era stato fermo bloccato sulla ecall

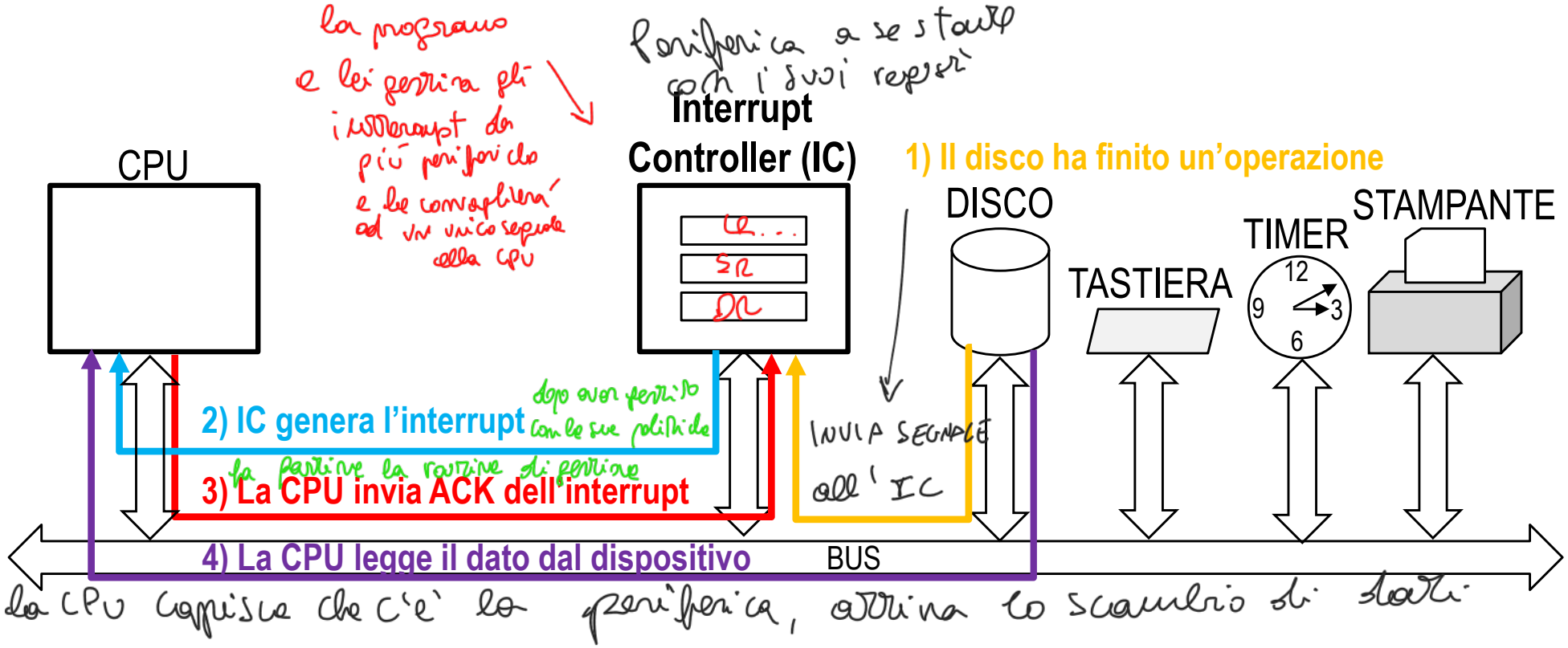
NON È EFFICIENTE

→ Protocollo ad INTERRUPT

- Evita l'attesa attiva del processore
  - Evita di fare numerosi controlli sullo stato del dispositivo
  - Il controller del dispositivo interrompe una sola volta quando ha bisogno la CPU

Protocollo ad Interrupt: funzionamento con più dispositivi

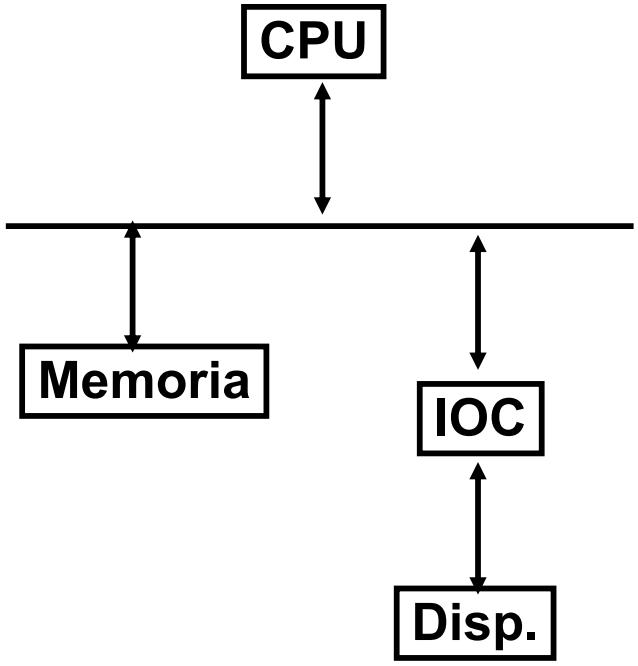
- La CPU dà un comando di lettura (es. dato da disco) e a questo punto la CPU si occupa di altro… (trascorre tempo)…
  1) Quando il dato è pronto, il dispositivo invia un interrupt al controller di interrupt (IC), che quindi raccoglie gli eventuali interrupt dai vari dispositivi
  2) IC interrompe la CPU
  3) La CPU disabilita gli interrupt e notifica IC (ACK)
  4) La CPU legge il dato dal disco e riabilita gli interrupt

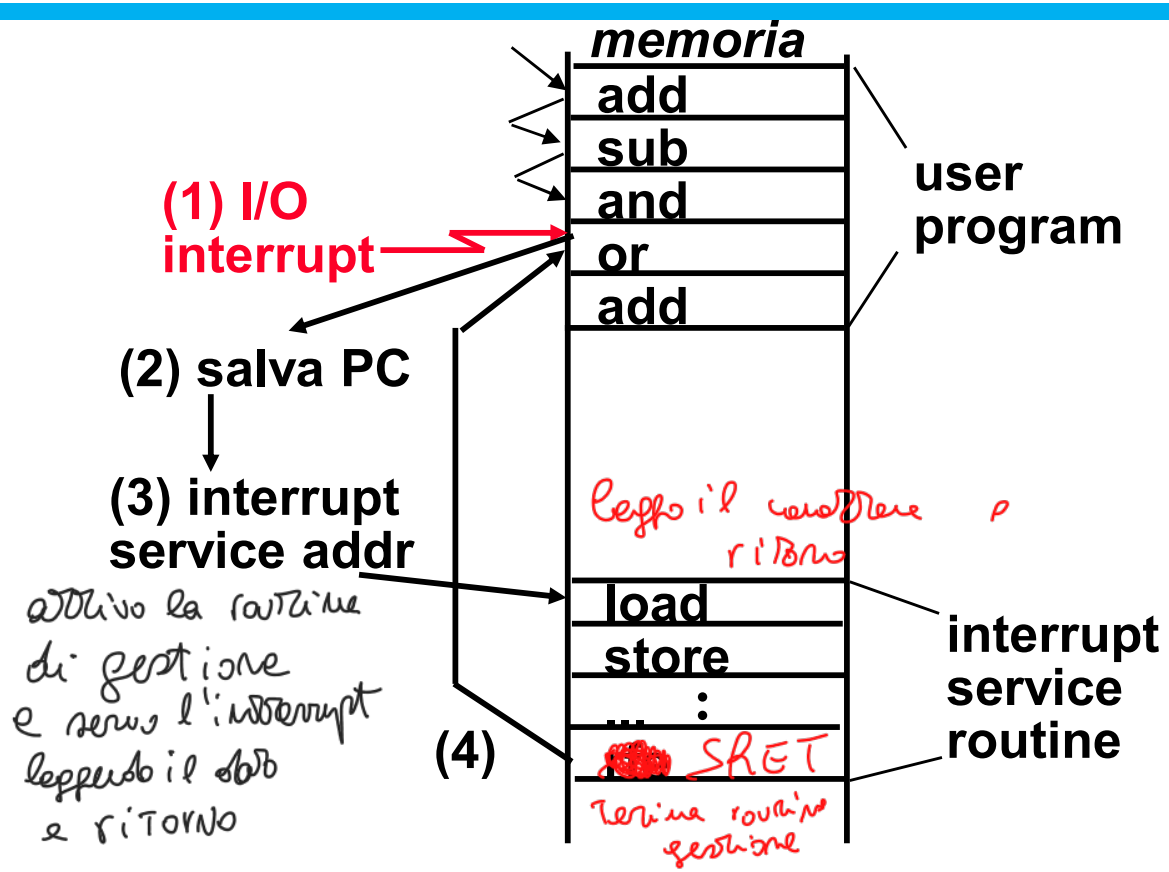


Dal punto di vista della CPU…

- La CPU dà un comando di lettura (vuole leggere dal disco)
- La CPU passa a fare altro lavoro
- Al termine di ogni istruzione guarda se c'è un interrupt pendente
- Se c'è un interrupt pendente => attiva la routine di gestione
  - Salva il contesto
  - Serve l'interrupt: preleva il dato e lo mette in memoria

Trasferimento dati a Interrupt





- **Vantaggio:**
  - **Il programma utente è bloccato solo durante il tempo effettivo per trasferire il dato dal dispositivo alla memoria** senza cicli di attesa attivi
- **Svantaggio: è necessario hardware dedicato per** gestire l'interrupt
  - **Generare l'interrupt (lato controller di I/O)**
  - **Accorgersi dell'interrupt (lato CPU)**
  - **Salvare lo stato della CPU e ripristinare lo stato dopo l'interrupt**

Stampa di una stringa ad Interrupt

Codice di preparazione alla stampa (es. Implementazione di un system call):

```
copy_from_user(buffer, p, count);
while (*printer_status_reg != READY);
*printer_data_register = p[0];
count = count - 1;
enable_interrupts();
scheduler();
```

La chiamata allo scheduler comporta il blocco (cioè la sua deschedulazione dalla CPU) del processo che ha invocato questa system call

Trasferisco il primo carattere

Routine di servizio dell'interrupt

```
if (count == 0) {
   unblock_user();
} else {
   *printer_data_register = p[k];
   count = count - 1;
   k = k + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

Inizialmente avrò k=1

mi sblocca l'utente solo quando ho stampato tutto

Interrupt

- **Per la CPU, l'interrupt è esattamente come un'eccezione salvo che**
  - **L'interrupt è un evento asincrono**, è HW, esterno del codice utente (?)
    - Non è provocato da istruzioni
    - Non impedisce il completamento di alcuna istruzione
  - **Ad esso fa seguito un trasferimento di informazioni**

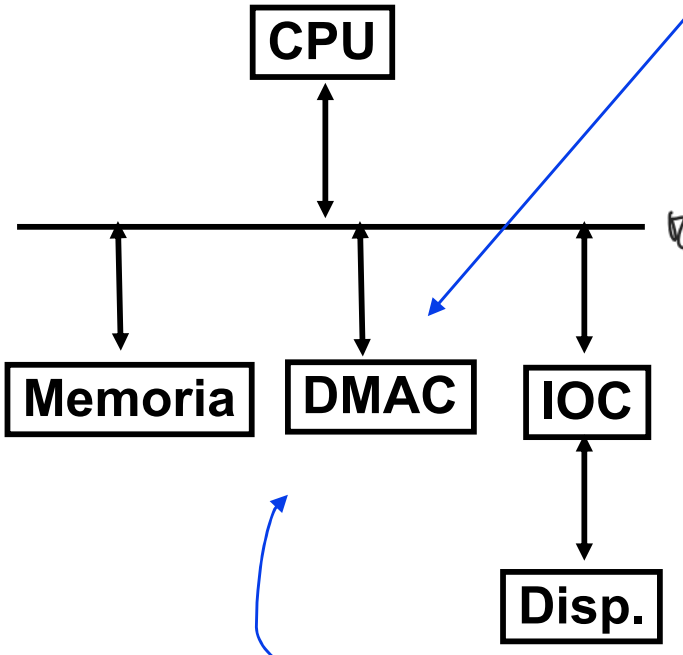
- **L'implementazione dell'interrupt è più complicata di quella di un'eccezione, perché**
  - **Deve consentire di identificare il dispositivo che ha generato l'interrupt** l'IC deve identificare il dispositivo che ha generato l'interrupt e comunicarlo alla CPU tramite CAUSE
  - **Le richieste di interruzione possono avere priorità diversa**
  - **Le richieste di interruzione provenienti da interrupt diversi possono sovrapporsi (interrupt multipli)**

priorità e fairness saranno gestite dall'IC

Identificare il dispositivo che ha generato l'interrupt

- **Differenti linee per ogni controller**
  - **Usato nei PC**
  - **Svantaggio: limita il numero dei dispositivi**
- **Software poll**
  - **La CPU chiede ad ognuno dei controller se ha generato un interrupt**
  - **Lento**
- **Daisy Chain (Hardware poll)**
  - **Il segnale di Interrupt Acknowledge viene inviato in daisy-chain**
  - **Il controller è responsabile di mettere sul bus un "vettore"**
  - **La CPU usa il vettore per identificare la routine di servizio**
- **Bus Mastering**
  - **Il controller deve richiedere il bus prima di fare interrupt**
  - **e.g. PCI, SCSI**

→ Direct Memory Access (DMA)



1) La CPU dice al DMA Controller:

- Indirizzo iniziale della zona di memoria coinvolta
- Indirizzo del controller del dispositivo
- Direzione (Read/Write)
- Quantità di dati da trasferire
- Infine la CPU da lo "start"

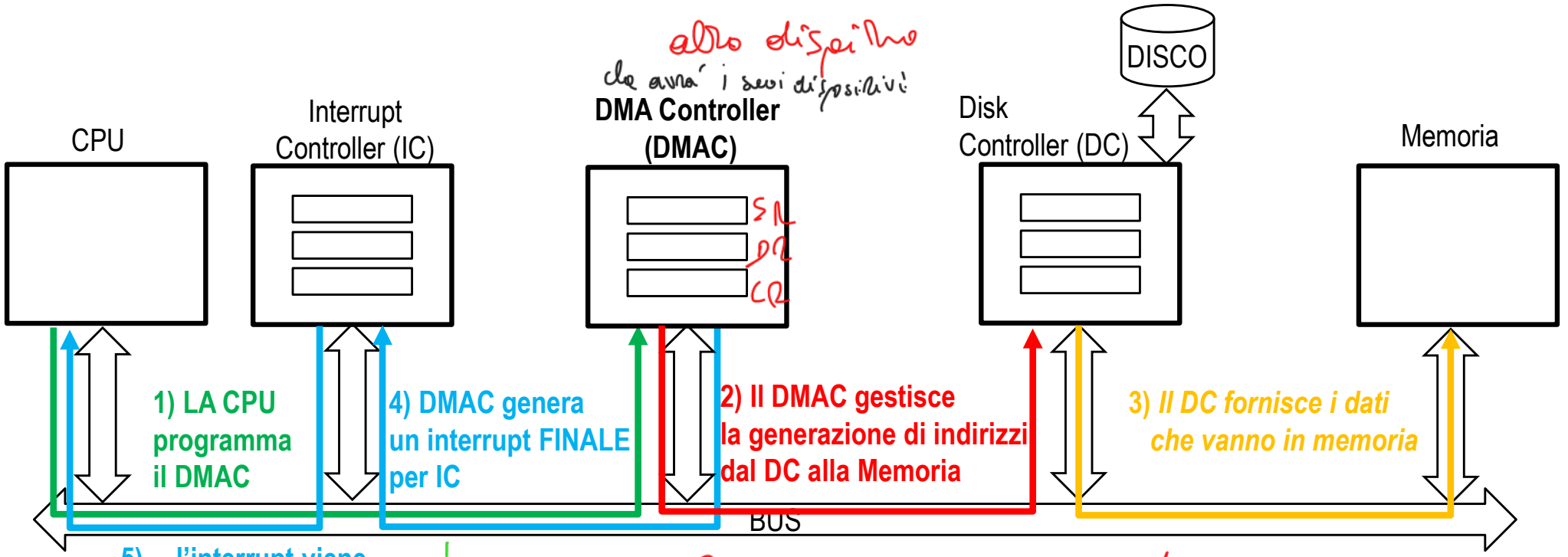
- La CPU può svolgere altro lavoro

2) Il DMA Controller si occupa del trasferimento

- genera tutti i necessari segnali per indirizzare la periferica e la memoria e per gestire l'handshake

- Il DMA Controller manda un interrupt quando ha finito

DMA -dettaglio



Stampa di una stringa con DMA

quando ha finito, il DMA genera un interrupt e IC ferma il processore

Codice di preparazione alla stampa (es. Implementazione di un system call):

1

```
copy_from_user(buffer, p, count);
setup_DMA_controller();
scheduler();
```

Serve

passo controllo

la CPU fa altro

La chiamata allo scheduler comporta il blocco (cioè la sua deschedulazione dalla CPU) del processo che ha invocato questa system call → il controllo viene quindi passato al sistema operativo che schedula un altro processo

2

Routine di servizio dell'interrupt

riceve solo in fondo

al ho ricevuto l'input.

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

> blocco utente, ha ricevo solo in fondo

IRET

ok ho ricevuto l'interrupt, sblocco

DMA Controller (DMAC)

- Il DMA può supportare diverse modalità di indirizzamento
  - 1D, ovvero un singolo registro per gli indirizzi
  - 2D, ovvero due registri per l'indirizzamento
  - 3D, ovvero tre o più registri per l'indirizzamento

- Esempio di indirizzamento 2D
  - Utile ad es. per prelevare i payload dai pacchetti Ethernet