APPUNTI DI INFORMATICA

di Andrea de Prisco

Il Sistema Operativo Batch

Seconda puntata di Appunti di Informatica. Dopo aver visto il mese scorso abbastanza in generale la struttura di un sistema di calcolo, l'argomento di questo mese riguarda le tecniche di comunicazione tra dispositivi di ingresso/uscita (stampanti, nastri, dischi, plotter, ecc.) e il calcolatore vero e proprio. Analizzeremo il problema non tanto dal punto di vista tecnico, cioè i vari metodi di trasmissione dell'informazione dal punto A al punto B di un sistema di calcolo, ma elevandoci a considerare solo gli aspetti logici di una comunicazione. In altre parole, posto che due oggetti possano comunicare, come sia possibile organizzare un traffico di comunicazioni. Come esempio studieremo un embrione di sistema operativo Batch, molto in voga alcuni anni fa, quando l'utente del centro di calcolo per battere i propri programmi doveva dapprima sedere alla perforatrice di schede per poi passare all'operatore di sala il pacchetto di schede perforate per l'esecuzione. Dopo qualche ora (non sempre era possibile far inserire le proprie schede nel computer) l'utente tornava per prelevare il tabulato dei risultati del proprio programma. Tale sistema era detto Batch (a lotti) proprio perché veniva raccolto un certo lotto di schede (più programmi) e mandati in esecuzione a determinate ore del giorno.

Comunicazione Computer-Device

Come descritto il mese scorso, un sistema di calcolo è sempre formato da più parti tra loro interagenti. Ogni computer avrà certamente una memoria per contenere programmi e dati, un'unità controllo processo (CPU) per eseguire programmi e un numero più o meno esteso di dispositivi (device) di ingresso/uscita. Ad esempio la stampante è un'unità di uscita (output device), la tastiera di un terminale un'unità di ingresso (input device), infine un driver per dischi sarà un'unità allo stesso tempo di ingresso e di uscita, essendo possibili su tale dispositivo sia operazioni di lettura che di scrittura.
In generale ogni dispositivo sarà collegato sia con la CPU, per ricevere comandi circa l'operazione da eseguire, sia con la memoria, dato che il "qualcosa" da far entrare o far uscire deve finire o essere parcheggiato in tale unità. In alcuni personal la connessione tra dispositivi e memoria non è diretta ma passa sempre per la CPU che riceve ad esempio il dato richiesto e "personalmente" lo scrive in memoria. Nei sistemi un po' più seri ciò non accade mai in quanto la CPU, per minimizzare il costo dell'esecuzione di un programma (centinaia di migliaia di lire al secondo), deve sempre e solo elaborare programmi, non curandosi di particolari tecnici come il prelevare dati dal disco. Al punto che se un dato richiesto tarda ad arrivare (e ciò da disco o da nastro o da qualsiasi dispositivo meccanico è normale) la CPU molla il programma che stava eseguendo e comincia (o continua) un altro programma. L'esecuzione del primo riprenderà quando la CPU si dovrà nuovamente fermare perché è stato richiesto un altro dato dal disco o è scaduto il "quanto" di tempo che la CPU poteva dedicare di filato ad una sola elaborazione. Comunque di questo genere di sincronizzazioni tra processi ne riparleremo a tempo debito, sempre in un altro articolo di Appunti.
In figura 1 è mostrato un primo approccio per il collegamento tra computer e dispositivi. Concettualmente è il più semplice di tutti, ma anche il più costoso ed efficiente. In tale schema ogni dispositivo ha una connessione fisica con la CPU ed una con la memoria. Si noti che tali connessioni sono per così dire private: ogni connessione è adoperata da un solo dispositivo. In questo modo, nulla vieta che mentre (ad esempio) il lettore di nastri riversa un programma in memoria, la CPU tramite un plotter fa un bel disegno, come da programma in corso di esecuzione. La costosità sta nel fatto che occorre disporre di tanti collegamenti quanti sono i dispositivi i quali, dal canto loro, possono benissimo essere tanti, dislocati a notevole distanza tra loro e dal computer.
Un secondo schema di collegamento è detto Unibus (adottato dai sistemi PDP/11) ed è mostrato in figura 2a. È

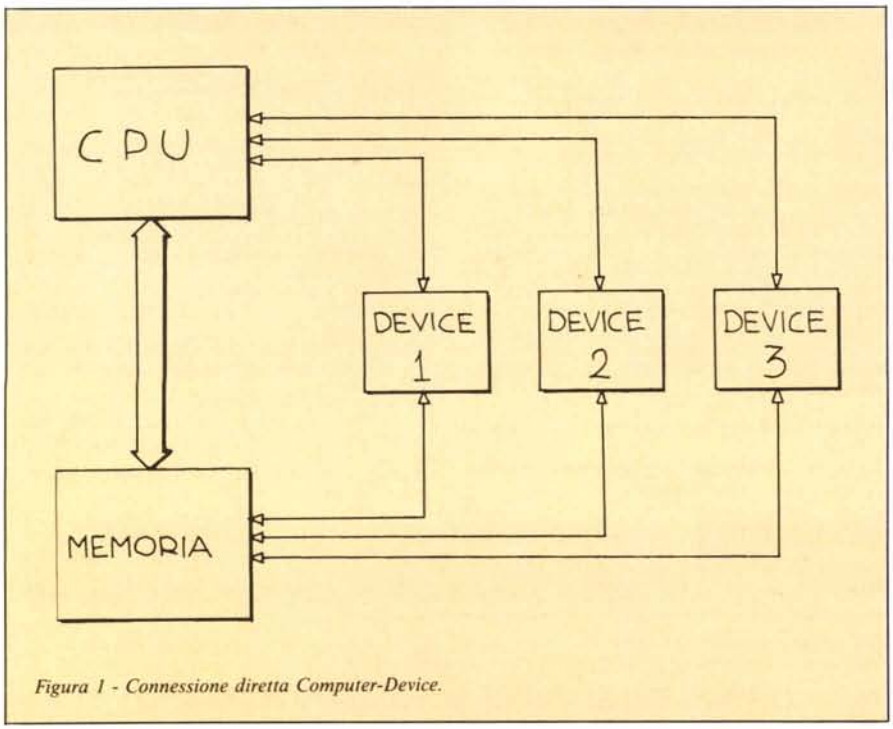

Figura 1 - Connessione diretta Computer-Device.

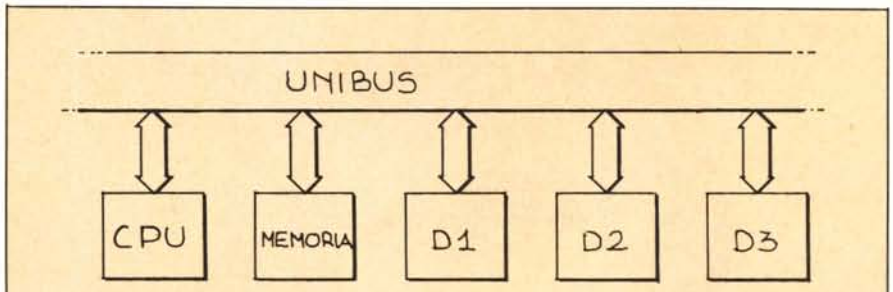

Figura 2a - Schema di collegamento tramite Unibus.

presente un unico collegamento, il bus, al quale fanno capo sia i dispositivi che la CPU e la memoria. In tale genere di connessione ogni unità per dialogare con un'altra unità deve chiedere l'uso del bus a un particolare supervisore, detto arbitro, che gestisce le concessioni. Lo schema è più semplice in quanto la connessione è unica, ma ha lo svantaggio che un solo dispositivo alla volta può essere all'opera. Schemi ancora più semplici di tale meccanismo non prevedono neanche l'arbitro, e ogni dispositivo accede brutalmente al bus eventualmente ripetendo il tutto se l'operazione è fallita per "bus già occupato". In figura 2b è mostrato un altro sistema di collegamento detto "bus ad anello". La CPU e la memoria sono collegate, come sempre, tra loro e a un processore di Input Output che gestisce le operazioni di ingresso/uscita. I dispositivi sono collegati a catena e il flusso delle informazioni circola in un solo senso (vedi frecce). Ogni messaggio per un dispositivo corre sul bus ad anello e contiene in testa il nome del destinatario. Per fare un esempio, immaginiamo di dover dialogare col dispositivo 2 che è un lettore di dischi: il nostro messaggio è "preleva il dato xxx e mandamelo". A questo dovremo aggiungere in testa il destinatario, cioè Dispositivo 2. In tale modo quando D1 riceverà il messaggio saprà che non è per lui e lo "girerà" a D2. D2 lo riconoscerà come proprio ed eseguirà l'ordine. Per risposta scriverà un nuovo messaggio, questa volta etichettato "CPU" che come prima oltrepasserà D3 e giungerà a destinazione. Semplice, no?

Sincronizzazione Computer-Device

Per prima cosa vedremo come funziona un semplice modello di sincronizzazione tra computer (CPU + Memoria Principale) e dispositivo. Occorre un sincronismo, dato che generalmente i dispositivi sono assai più lenti delle CPU: pensate per un attimo ad una stampante, potrà stampare 120 caratteri al secondo... il computer gliene può passare anche 2000 nello stesso tempo. Se non si procede con ordine state pur certi che molto presto la stampante impazzirebbe urlando: Bastaaaa!!!
Scherzi a parte è ovvio che se la stampa avviene a 120 cps, il computer mediamente dovrà al massimo inviarglieli alla stessa frequenza, chiedendo il permesso prima di spedire un nuovo lotto di caratteri da stampare. Su questo principio, funziona il primo meccanismo di sincronizzazione illustrato in figura 3.
Computer e device sono tra loro collegati oltre che dal bus principale dove ci corrono messaggi, risposte e dati, anche da due bus di controllo (1 e 2) che servono il primo per dare il via al dispositivo e l'altro per controllare se questo è libero o sta ultimando un'operazione. Così, se il device è una stampante che stampa linee di max. 80 caratteri, la sequenza delle operazioni compiute dalla CPU per stampare un po' di linee è, tenendo sott'occhio figura 3, la seguente:
— si pongono 80 caratteri da stampare in una zona della memoria
— si attende che il bus di controllo 2 sia in stato di Ready (pronto)
— si scrive sul bus principale il comando (nel nostro caso è un ordine di stampa con la specifica di dove i dati da stampare sono stati posti in memoria)
— si invia il segnale Busy alla stampante
— se ci sono altre linee si ritorna al primo passo
analogamente la stampante avrà una sequenza di operazioni da compiere del tipo:
— attendo che il bus di controllo 1 sia in stato di Busy
— prelevo il comando dal bus principale
— eseguo il comando (equivale a prelevare gli 80 caratteri dalla memoria e a stamparli)
— invio il segnale Ready sul bus di controllo 2 e ricomincio dal primo passo.
I due processi all'interno del computer e all'interno della stampante evolvono parallelamente e in perfetto sincronismo grazie ai segnali di Busy e di Ready.
Questo metodo, assai semplice, ha il considerevole svantaggio di far restare la CPU in attesa che la stampante sia

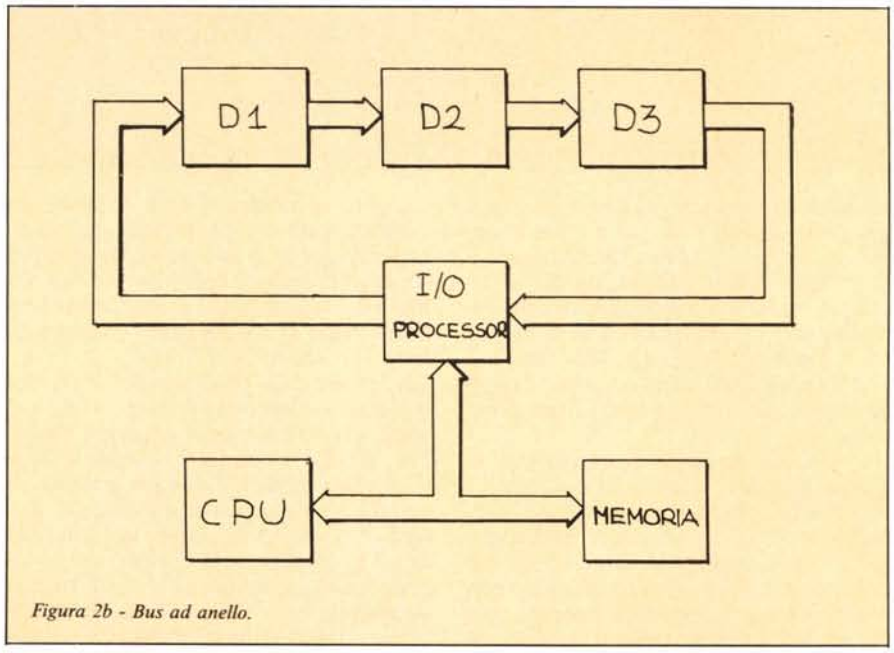

Figura 2b - Bus ad anello.

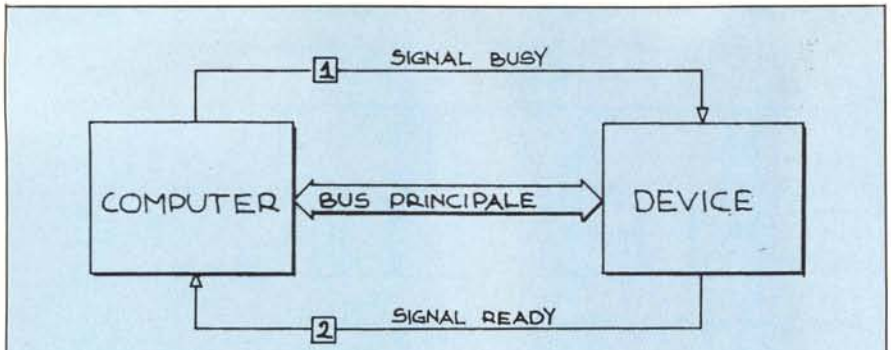

Figura 3 - Sincronizzazione tramite Segnali.

pronta per ricevere altri caratteri da stampare. E questo, come detto prima, non è cosa buona per un calcolatore che si rispetti. Un miglioramento delle prestazioni si ha impegnando la CPU in qualcos'altro invece di aspettare che un dispositivo si liberi. Per esempio far andare avanti un altro programma contenuto in memoria che in quel momento non ha da far aspettare la CPU.
Per migliorare la sincronizzazione tra computer e device occorre introdurre il meccanismo delle interruzioni o interrupt. Un interrupt è un segnale proveniente dall'esterno della CPU che fa interrompere l'esecuzione del programma in corso per far partire immediatamente una subroutine detta appunto di gestione delle interruzioni. È all'interno di tale subroutine che si decide il da farsi a seconda della provenienza dell'interruzione o di altri fattori. Terminato questo "da farsi" il programma interrotto continua la sua elaborazione come se nulla fosse accaduto.
Il meccanismo delle interruzioni si amplia poi col concetto di maschera e di priorità che qui illustreremo solo brevemente. Può succedere infatti che la CPU preferisca non essere disturbata da interruzioni, dovendo per esempio compiere operazioni delicate (come inviare un messaggio a un dispositivo), e allora impone una maschera all'interrupt. In questo modo l'interruzione è completamente ignorata o al più resta pendente ossia in attesa che la maschera venga tolta.
Le priorità infine permettono di dare per così dire una forza variabile all'interrupt. Per esempio la CPU invece di mascherare completamente le interruzioni potrebbe porsi in stato di priorità 5. In questa maniera se l'interrupt ha una priorità maggiore di 5 viene accolto altrimenti è ignorato. Se la CPU vuole accogliere tutte le interruzioni si porrà in priorità 0; se vorrà mascherare qualsiasi interrupt si porrà in stato di priorità massima. Con questo meccanismo si possono dare priorità diverse ai vari dispositivi che generano interruzioni in modo da accogliere più sovente le richieste dai device più veloci (dischi, tamburi) e dare meno retta a device di natura lenta (stampanti, lettori di schede e di nastri).
La sincronizzazione Computer-Device col meccanismo delle interruzioni avviene pressappoco nel seguente modo. Per meglio intenderci, immaginiamo di aver in memoria due programmi. Il primo deve stampare un determinato numero di linee lunghe 80 colonne l'una; il secondo... non ci interessa: è un programma qualsiasi. Diamo ordine di stampare la prima linea come abbiamo visto prima: mettiamo il comando sul bus principale e inviamo il segnale Busy per far partire la stampante. A questo punto, invece di attendere per inviare un'altra linea da stampare cediamo il controllo all'altro programma (che fa quello che deve fare). La stampante, finito di scrivere la linea manda (vedi figura 4) un interrupt al computer. La subroutine di manipolazione delle interruzioni darà semplicemente ordine di stampare una nuova linea (se ce ne sono altre) e... basta. La manipolazione delle interruzioni in questo caso non fa altro e riparte così il programma (quello che fa quello che deve fare) che era stato interrotto. Tutto qui.
Se qualcosa non vi è chiaro non disperate questo tipo di sincronizzazione lo vedremo meglio ora che commenteremo il funzionamento di un piccolo...

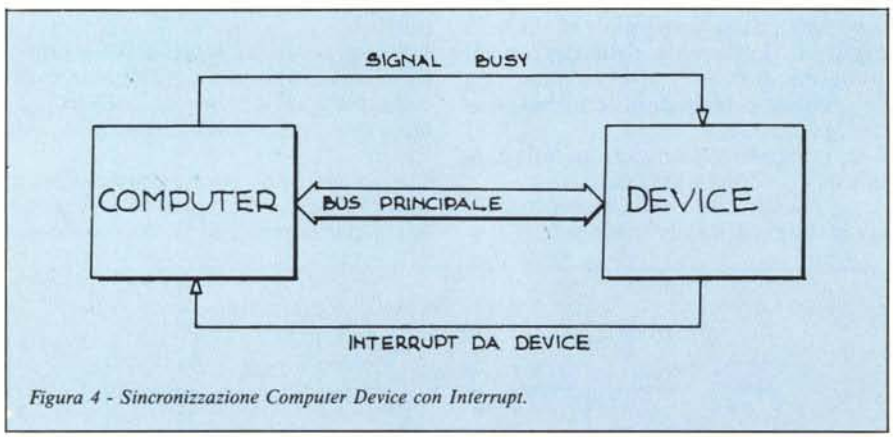

Figura 4 - Sincronizzazione Computer Device con Interrupt.

Sistema Operativo Batch

Dicevamo nell'introduzione che col sistema operativo Batch l'utente di un centro di calcolo per far girare i propri programmi doveva innanzitutto trasferirli su schede perforate. L'operazione veniva compiuta presso un apposito (rumorosissimo) marchingegno detto appunto perforatrice di schede. Generalmente ogni scheda conteneva una linea di programma o una linea di dati per un massimo di 80 caratteri. Finito di perforare le schede contenenti programma e dati, bisognava aggiungere alcune schede di controllo contenenti caratteri speciali (asterischi, barre o altro) per specificare alcune opzioni per la compilazione e, giustamente, per delimitare il proprio pacchetto di schede. Infatti i vari programmi (leggi: pacchetti di schede perforate) raccolti nella giornata, venivano introdotti nel lettore di schede, uno di seguito all'altro. Era il sistema che doveva riconoscere (grazie alle schede di controllo) dove terminava ogni programma per poterlo compilare.
Chi legge avrà certamente notato l'uso di tempi passati dei verbi: il sistema Batch è ormai in via di estinzione, ne sarà rimasto vivo qualcuno in centri di calcolo universitari dove gli studenti vanno per farsi "le ossa". Oggi si usano prevalentemente i terminali, non importa se collegati direttamente col calcolatore (e quindi con la CPU in linea, come nei personal), o sempre stile perforatrici e schede, con la differenza che i programmi vengono salvati su disco magnetico, è possibile editarli su video e l'operatore a determinate ore

del giorno prende il disco pieno di programmi da elaborare e lo introduce nel computer (come faceva prima coi pesanti cassetti di schede).
In questa sede vedremo due embrioni di sistema operativo Batch, il primo sfrutta semplicemente i segnali di Ready e di Busy, il secondo, più evoluto, la gestione degli interrupt da dispositivo.
Il primo algoritmo è mostrato in figura 5. Cominciamo dall'interfaccia CLE.
CLE sta per Compiler Loader Execute e riassume le tre fasi comprese dall'ingresso di un programma nel computer fino all'uscita dei suoi risultati (escluso). Infatti un programma scritto in qualsiasi linguaggio ad alto livello viene prima compilato poi caricato in memoria (load) e poi mandato in esecuzione. Interfaccia CLE sta per l'algoritmo che preleva i vari programmi dal lettore schede, esegue la fase CLE e stampa i risultati.
L'interfaccia CLE (fig. 5a) è composta da 6 passi. Il primo passo (gosub Controllo lettore) corrisponde a leggere una scheda tramite il lettore. Col passo 2 (posto che ogni programma termini con un asterisco come carattere di controllo) si cicla continuamente finché non è stato letto un intero programma.
Il passo 3 è la fase CLE e quindi dopo tale passo il programma "ha girato". Col passo 4 se non ci sono linee da stampare si continua col programma successivo (vai a 1) altrimenti (passo 5) si salta a Controllo stampante per stampare una linea e (passo 6) si torna al passo 4 per vedere se ci sono altre linee da stampare.
Per completezza vediamo come funzionano anche le altre routine di figura 5. La 5b come detto legge una scheda tramite il lettore, e per fare ciò utilizza i meccanismi di sincronizzazione CR-busy e CR-ready (CR sta per Card-Reader, lettore di schede). L'algoritmo è assai semplice: definisce un'area buffer (trova un posto libero in memoria dove parcheggiare ciò che leggerà); attende che il lettore sia libero (wait CR-ready); dà il comando al lettore e segnala CR-busy (dà ordine al lettore schede di partire e leggere una scheda). Basta.
Parallelamente (fig. 5c) il lettore schede attende il segnale di partenza (Wait CR-busy); preleva il comando dal Bus principale e l'esegue (legge la scheda e pone in memoria il suo contenuto); segnala che è pronto per una nuova lettura (signal CR-ready) e torna al primo passo in attesa di un nuovo comando.
Per la stampa, 5d e 5e, funzionano praticamente nello stesso modo: unica differenza il prefisso LP che sta per Line Printer e [linee da stampare] decrementato ogni volta che viene chiamata Controllo stampante, in modo da fermarsi (passo 4 di 5a) quando non vi sono più linee da stampare.
Il sistema Batch ottimizzato è mostrato schematicamente in figura 6 mentre l'algoritmo è quello di figura 7. Per ottimizzare un sistema Batch occorre impegnare in qualche modo la CPU mentre bisogna attendere l'arrivo di dati dal lettore di schede o si aspetta che la stampante termini prima di inviargli una nuova linea. L'idea è di sovrapporre le fasi di ingresso, di CLE e di uscita di più programmi. Per essere più precisi, mentre si stampano i risultati del programma x, si compila il programma y e si leggono le schede del programma z. Il tutto col meccanismo di sincronizzazione con interrupt da dispositivo.
Spieghiamoci meglio. Il primo programma è caricato in memoria azionando il lettore di schede. Dato che in tale istante non vi sono altre cose da fare, è ovvio che il primo programma sarà caricato come abbiamo sempre fatto ossia aspettando che sia terminata l'operazione. Avendo ora un programma in memoria, possiamo contemporaneamente leggere le schede del secondo programma e nelle pause tra l'arrivo di una scheda e l'altra, compilare (per essere più precisi eseguire la fase CLE de) il primo. Terminata la fase CLE, se anche il secondo programma è stato letto, possiamo iniziare a leggere il terzo, stampare i risultati del primo e compilare il secondo. Se non vi è sfuggito nulla, capirete bene che eccetto il primo "giro" il nostro sistema operativo Batch in ogni momento stamperà i risultati del programma i, compilerà, caricherà ed eseguirà il programma i+1 leggerà le schede del programma i+2.
Commenteremo ora l'algoritmo di figura 7 cominciando dalla subroutine di gestione delle interruzioni (7b). Come detto prima questa routine è chiamata ogni volta che è stato mandato un interrupt da uno dei due dispositivi, stampante (ha finito di stampare una linea) o lettore (ha finito di leggere una scheda). Priorità e maschere in questo contesto non ci sono, per non complicare ulteriormente le cose. Allora, Gestione interrupt: il primo passo

```
a Interfaccia CLE 1:
   1. gosub Controllo lettore
   2. se [ultimo carattere letto] ≠ "*" vai a 1
   3. compilazione caricamento ed esecuzione
   4. se [linee da stampare] = 0 vai a 1
   5. gosub Controllo stampante
   6. vai a 4

b Controllo lettore:
   1. definire area buffer
   2. wait CR-ready
   3. comando = (leggischeda, area buffer)
   4. signal CR-busy
   5. return

c Funzionamento lettore:
   1. wait CR-busy
   2. preleva ed esegui comando
   3. signal CR-ready
   4. vai a 1

d Controllo stampante:
   1. [linee da stampare] = [linee da stampare] - 1
   1. area dati = locazione linea da stampare
   2. wait LP-ready
   3. comando = (stampalinea, area dati)
   4. signal LP-busy
   5. return

e Funzionamento stampante:
   1. wait LP-busy
   2. preleva ed esegui comando
   3. signal LP-ready
   4. vai a 1
```

Figura 5 - Sistema Operativo Batch non ottimizzato.

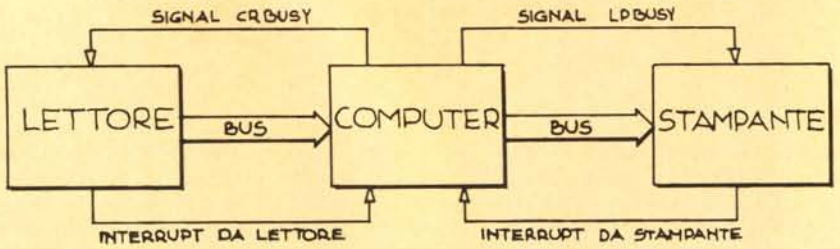

Figura 6 - Sistema Batch.

assegna ad I la provenienza dell'interruzione, lettore o stampante. Il resto si autocommenta: se l'interruzione proviene dal lettore (ripetiamo: ciò accade se è stata letta una scheda) si passa il controllo al Controllo lettore, altrimenti (passo 3) al Controllo stampante (non vorremmo essere noiosi: ciò accade se è stata stampata una linea). Il return del passo 4 (di 7b) fa ritornare al punto interrotto dall'interrupt di uno dei due dispositivi.

Passiamo ora all'interfaccia CLE di figura 7a. Avevamo detto che il primo programma doveva essere letto per intero. Ciò è realizzato dai primi 4 passi:

```
a Interfaccia CLE 2:

   1. definire area buffer
   2. comando = (leggischeda, area buffer)
   3. signal CR-busy
   4. wait fine lettura
   5. [linee da stampare] = 0
   6. definire area buffer
   7. comando = (leggischeda, area buffer)
   8. signal CR-busy
   9. se [linee da stampare] = 0 vai a 12
  10. comando = (stampalinea, area dati)
  11. signal LP-Busy
  12. compilazione caricamento er esecuzione
  13. area dati = locazione prima linea da stampare
  14. wait fine lettura
  15. wait fine stampa
  16. vai a 6


b Gestione interrupt:

  1. I = provenienza interrupt
  2. se I = lettore gosub Controllo lettore
  3. se I = stampante gosub Controllo stampante
  4. return


c Controllo lettore:

  1. se [ultimo carattere letto] = "*" signal fine lettura & return
  2. definire area buffer
  3. comando = (leggischeda, area buffer)
  4. signal CR-busy
  5. return


d Controllo stampante:

  1. [linee da stampare] = [linee da stampare] - 1
  2. se [linee da stampare] = 0 signal fine scrittura & return
  3. area dati = locazione linea da stampare
  4. comando = (stampalinea, area dati)
  5. signal LP-busy
  6. return
```

Figura 7 - Sistema Operativo Batch ottimizzato.

si definisce un'area buffer (come prima) e si pone sul bus principale il comando per il lettore. Con signal CR-busy (passo 3) il lettore legge una scheda. Mentre fa questo il controllo è fermo sul passo 4: il processo aspetta il segnale di fine lettura (del programma, non di una scheda). Aspetta e basta. Intanto il nostro lettore ha completato la lettura della prima scheda e come promesso manda un interrupt alla CPU (che aspettava, ferma al passo 4). Parte la routine di 7b che gestisce le interruzioni. L'interruzione proviene dal lettore e viene invocata la subroutine Controllo Lettore. Lì (7c) se l'ultimo carattere letto è "*" vuol dire che il primo programma è finito e occorre mandare il segnale di fine lettura. Altrimenti (passi 2 e seguenti di 7c) si dà il comando di leggere un'altra scheda. Return di 7c ci fa ritornare a 7b dove return ci fa ritornare al passo 4 di 7a dove era avvenuta l'interruzione. È ovvio che tutto ciò continua (il processo aspetta sempre al passo 4 salvo quando è interrotto dal lettore) fino a quando non viene letto questo benedetto asterisco. Dopo tutto ciò abbiamo soltanto letto il primo programma; continuiamo. Passo 5 (sempre di 7a): [linee da stampare] è posto uguale a 0: è ovvio, in quest'istante abbiamo solo il primo programma in memoria e dobbiamo ancora eseguirlo: cosa vorremmo stampare?

Passi 6, 7 e 8: si dà ordine al lettore di leggere una nuova scheda. Passo 9: se [linee da stampare] è 0 vai a 12: è il caso nostro. Compilazione caricamento ed esecuzione. Mentre ciò avviene, il lettore terminerà la lettura della prima scheda del secondo programma e manderà un interrupt: ciò in parole povere equivale a dare un nuovo ordine di lettura e a continuare dal punto (del passo 12 di 7a) dove era avvenuta l'interruzione. Finita la fase CLE (passo 12), se era stato letto l'asterisco del secondo programma allora era stato dato anche il signal di fine lettura e al passo 14 non ci si ferma, altrimenti aspettiamo. A questo "giro" anche 15 ci dà via libera dato che nessuna linea doveva essere stampata. Da questo momento siamo nel pieno del funzionamento dato che con "vai a 6" inizia un ciclo dove stamperemo i risultati del programma 1, eseguiremo la fase CLE del programma 2 (che è finito in memoria mentre eseguivamo la fase CLE del programma 1) e leggeremo il programma 3. Della serie: ... e così via! MC