Architettura del calcolatore

Collegamento al bus

Stadi di uscita per un collegamento al bus

Esistono 3 stadi di uscita per collegare porte logiche alle linee del bus. Esistono tre tipologie di linee di bus:

Monosorgente: un solo dispositivo da il valore alle linee mentre molti dispositivi lo acquisiscono (ad esempio Address Bus). Per poter pilotare le linee monosorgente, i dispositivi che si interfacciano a queste linee (ossia porte logiche e bistabili) devono avere uno stadio di uscita detto **TOTEM POLE** che indica due possibili stati:

- 0 a bassa impedenza;
- 1 a bassa impedenza (piu' vicino possibile alla tensione di alimentazione).

Multisorgente sincrone: a turno diversi dispositivi danno il valore alle linee (ad esempio Data Bus). E' possibile decidere chi ha diritto alla scrittura del valore sulle linee in modo da sincronizzare i dispositivi Serve interfacciare le sorgenti alla linea di bus mediante lo stadio di uscita **TRI STATE** con 3 valori di tensione:

- 0 a bassa impedenza;
- 1 a bassa impedenza;
- Z ad alta impedenza (che indica che il dispositivo non fa transitare corrente alla sua uscita).

Dunque sul bus sara' un solo dispositivo a scrivere mentre tutti gli altri esporranno lo stato Z ad alta impedenza. Serve un **arbitro** che e' connesso a tutti i dispositivi che scrivono sulla linea del bus che decide quando uno di questi puo' scrivere, imponendo agli altri lo stato Z.

Multisorgente asincrone: diversi dispositivi danno valore alle linee ma non e' possibile sincronizzare i dispositivi, in quanto al momento in cui questi dispositivi devono diventare "sorgente", non c'e' una componente centralizzata che regola il comportamento di tutti gli altri Lo stadio di uscita si chiama **OPEN COLLECTOR** e prevede due stati:

- 0 a bassa impedenza;
- Z ad alta impedenza.

I dispositivi collegati alla linea possono indicarne il valore solo se vogliono imporre il valore 0. Dunque se nessun dispositivo vuole imporre 0, e quindi si "scollegano" tramite lo stato Z ad alta impedenza, una **Resistenza di Pull-Up** collegata alla linea e alimentata con tensione di alimentazione porta la linea al valore 1.

Memoria di lavoro

Chip di Memoria

Alla CPU serve una memoria di lavoro elettronica per avere dei tempi di risposta accettabili per poter eseguire il proprio ciclo fetch-decode-execute milioni di volte al secondo.

Nella memoria di lavoro devono essere inseriti sia dati sia codici macchina del programma da eseguire.

Si tratta quindi di una memoria a lettura e scrittura detta **memoria RAM** (Random Access Memory), che non tiene conto di alcun rapporto causa-effetto tra un accesso alle proprie celle ed il successivo. Per la RAM ogni accesso e' indipendente.

Esistono due tipi di memoria RAM: **SRAM** (Static RAM): un componente con 4 celle di memoria di 3 bit. Ogni bit e' memorizzato in un bistabile di tipo D. Per selezionare una cella mi servono due linee di indirizzo (le linee A0 e A1) che entrano in una sezione composta da 4 porte AND che operano come un "decoder": ciascuna di esse esegue un **mintermine** dei due ingressi A0 e A1 (funzione booleana come AND e NOR che assume il valore 1 in un'unica configurazione di variabili d'ingresso booleane indipendenti). Dunque solo una delle 4 linee di selezione parola sara' attiva in base all'indirizzo.

Per scrivere la parola selezionata (che quindi ha 1 sulla riga) diamo il comando **WE** (Write Enabled) che consente ai bistabili di tipo D della parola della cella stessa di campionare alle linee di ingresso dati Di0, Di1 e Di2. Alle linee di uscita D0, D1 e D2 e' presente l'output dei bistabili. La porta logica OR che si trova all'uscita mi consente di portare in uscita il valore del bistabile della colonna della cella selezionata.

Se facciamo uso di stadi uscita TRI-STATE alle porte OR delle colonne possiamo lavorare bidirezionalmente sul Data Bus. Aggiungiamo un segnale aggiuntivo **CS** (Chip Select) che fa da arbitro tra le uscite multisorgente sincrone e la CPU per quanto riguarda la lettura.

DRAM (Dynamic RAM): si differenzia dalla SRAM per il fatto che i bistabili, per risparmiare spazio sul chip, sono sostituiti da condensatori. Il valore del bit e' associato alla carica del condensatore. Poiche' i condensatori si scaricano e perdono l'informazione e' necessaria un'attivita' periodica di **refresh** che ripristini la carica sui condensatori che si stavano scaricando (questa attivita' e' delegata a componenti esterne). In questo modo si ottengono elevatissime densita' di memoria (chip da 1Gbit).

Un generico chip di RAM e' costituito da:

- na piedini di indirizzo monodirezionali (in input nel chip) (indicati come **memory address**), il numero di piedini dipende dal numero di celle presenti nel chip (numero di celle = 2^numeri di fili di indirizzo);

- nb piedini di dato bidirezionali (**memory data**), il numero di piedini dipende dalla grandezza della parola di memoria (nummero bit per cella di memoria);
- 1 linea in input nel chip: chip select (**CS**);
- 1 linea in input che indica read/write (**R/W**);
- eventuale linea in input: output enable (**OE**), che puo' servire ad avviare in momenti ritardati i TRI-STATE di uscita;
- eventuale linea in output: output ready (**RD**), che serve a comunicare se il componente e' pronto a trasferire o se ha bisogno di altro tempo per completare le operazioni.

Oltre alla memoria di tipo RAM, al calcolatore serve una memoria a sola lettura **ROM** (Read Only Memory) che mantenga il proprio contenuto anche in assenza di alimentazione. Ci sono diversi momenti in cui puo' essere necessaria un'informazione in assenza di alimentazione:

- Quando c'e' un programma da eseguire in fase di accensione del calcolatore (fase di **bootstrap**);
- Quando il programma da eseguire e' sempre lo stesso (applicazioni **embedded**).

Il chip di ROM si presenta essere molto simile al chip di RAM riguardo la **piedinatura**, differisce solo per i piedini di dato nd (**memory data**) che sono monodirezionali in output e per la linea **R/W** in quanto l'operazione Write sulla ROM non e' un'operazione possibile.

Banchi di Memoria

Un banco di memoria e' un insieme di chip di memoria che "riempie" una porzione dello spazio di indirizzamento della CPU considerata. Serve a dare all'unita' centrale una quantita' di memoria fisica accessibile attraverso il bus.

Ovviamente ogni cella di memoria presente nel banco deve avere un numero di bit pari al numero di linee del Data Bus della CPU. Il numero di celle di memoria del banco e' tipicamente una potenza di 2. Il banco appare alla CPU come una sequenza di celle adiacenti in una determinata posizione dello spazio di indirizzamento della CPU (non necessariamente tutto lo spazio di indirizzamento).

Facciamo un esempio di banco di memoria per la CPU LC-2 che ha:

- Address Bus a 16 linee;
- Data Bus a 16 linee

Supponiamo di avere a disposizione dei chip integrati di RAM da 1K x 8 ovvero:

- integrati aventi ciascuno 1024 celle;
- parole da 8 bit per cella.

Vogliamo realizzare un banco di memoria RAM da 4k celle da 16 bit, quindi:

- 4096 celle;
- parole da 16 bit per cella;
- posizionato a partire dall'indirizzo 0000 0000 0000 0000: dunque i primi 4096 indirizzi dello spazio di indirizzamento della LC-2 dovranno accedere a nostro banco.

In primis dobbiamo realizzare delle celle da 16 bit, in quanto la CPU LC-2 pretende di vedere delle celle da 16 bit grandi come il proprio Data Bus, ma noi abbiamo a disposizione degli integrati con celle da 8 bit: dunque servono 2 componenti accoppiati per ogni "riga".

Il componente di sinistra conterra' gli 8 bit piu' significativi della cella di memoria LC-2, e il componente di destra conterra' gli 8 bit meno siginificativi. In questo modo il Data Bus verra' diviso in 2 parti: gli 8 bit piu' significativi saranno collegati al componente di sinitra e gli 8 bit meno significativi al componente di destra. Indirizzando, con lo stesso indirizzo, due celle di memoria "allineate", riusciamo a far vedere alla LC-2 una cella di memoria da 16 bit.

Il secondo problema da risolvere e' realizzare un banco da 4096 celle dal momento in cui ogni coppia di componenti contiene 1024 celle individate da una configurazione da 10 bit dell'Address Bus. Per ottenere un banco da 4096 celle ci servono 4 coppie di componenti. A queste 4096 celle la LC-2 potra' accedere tramite una configurazione da 12 bit dell'Address Bus (2^12 = 4096). Colleghiamo i 12 bit dell'Address Bus al banco di memoria:

- I 10 bit meno significativi vengono collegati in parallelo a tutte le coppie di componenti, in modo date che una qualsiasi configurazione di questi 10 bit selezioni una parola in ciascuna coppia delle 4 componenti. La parola avra' la stessa posizione in ciascuna delle coppie;
- Per scegliere quale delle 4 parole indirizzate usiamo i 2 bit rimanenti;
- Serve un decoder avente un determinato numero di ingressi e un numero di uscite pari a 2^numero di ingressi, collegato ai Chip Select dei componenti. Per ogni configurazione che appare sugli ingressi, il decoder attiva una sola delle sue uscite. Nel nostro caso il decoder deve avere 2 ingressi ai quali deve collegare i 2 bit di selezione. Le sue 4 uscite saranno collegati ai Chip Select delle singole coppie dei componenti di memoria.

Rimane solo capire come posizionare il banco da 4K. Ad ora abbiamo usato solo 12 dei 16 bit dell'Address Bus. I 4 bit piu' significativi ci dicono a quale delle 2^4=16 "pagine" da 4K celle ci stiamo riferendo nello spazio di indirizzamento da 64K della CPU LC-2. Serve inserire una selezione di banco: fare in modo che il decoder venga abilitato solo se i 4 bit dell'Address Bus assumono la configurazione associata al banco e quindi, solo in questo, una riga di componenti RAM del banco sia abilitata ad interagire con la CPU.

Ad ogni componente di memoria viene distrbuito un segnale di controllo lettura/scrittura per indicare il senso di trasferimento. Le memorie sono munite di

un segnale "Ready" per informare la CPU che e' pronta a scambiare dati. Nel caso del banco ROM manca il segnale di lettura/scrittura poiche' si puo' solo leggere.

Input/Output a controllo di programma

Problematica delle operazioni di Input/Output

Le interfacce di I/O sono visibili alla CPU come "celle di memoria dedicate a particolari funzioni":

- alcune "celle" trasferiscono dati;
- alcune "celle" danno comandi alla periferica o ne riportano lo stato.

Ci sono due modalita' di collegamento tra interfacce di I/O e CPU:

- **Interfacce Memory Mapped**:
  - le celle dedicate, ossia i registri contenuti nell'interfaccia I/O, rispondono a normali indirizzi di memoria generati dall'unita' centrale;
  - dunque le normali istruzioni macchina che servono per leggere e scrivere le celle di memoria possono essere utilizzate anche per accedere alle interfacce I/O, saranno solamente differenti gli indirizzi da specificare;
  - Il problema di questa soluzione e' che poiche' i registri delle interfacce sono molti di meno delle celle di memoria, lo spazio di indirizzamento della CPU viene "bucato" in quanto non puo' essere sfruttato come memoria classica.
- **Interfacce I/O Mapped**:
  - la CPU ha un doppio spazio di indirizzamento, uno per la memoria e uno specifico per le interfacce I/O;
  - l'accesso ai due spazi di indirizzamento avviene tramite istruzioni macchina dedicate.

Il problema piu' importante dell'interazione tra CPU e interfacce di I/O e' quello della **sincronizzazione**. Ogni operazione di I/O implica la sincronizzazione tra due riferimenti temporali:

- clock della CPU;
- l'orologio dei fenomeni esterni che agiscono sulla periferica collegata all'interfaccia.

Ci sono 3 modalita' di sincronizzazione:

- **Controllo di Programma**: il clock e' il riferimento temporale che impone i ritmi delle interazioni esterne (visione Tolemaica del calcolatore);

  - La CPU esegue operazioni di I/O (dunque interagisce con l'interfaccia) quando il programma in esecuzione decide di interrogare l'interfaccia stessa.
  - Per alcuni aspetti questo approccio e' concettualmente sbagliato perche' la CPU non sa quando sono state richieste operazioni di I/O e quindi e' costretta a controllare periodicamente. Tuttavia questa modalita' funziona poiche' il clock della CPU e' all'ordine dei GHz mentre l'orologio esterno si muove all'ordine degli Hz/kHz.
- **Interrupt**: sono gli eventi esterni e il loro tempo ad imporre il lavoro alla CPU (visione Copernicana del calcolatore);
  - La CPU esegue operazioni di I/O quando l'interfaccia lo richiede, a seguito di quanto segnalato dalla periferica.
  - Questo approccio e' concettualmente piu' corretto, na richiede che l'interfaccia possa "interrompere" la CPU durante lo svolgimento delle sue attivita' per dedicarsi esclusivamente a quanto richiesto.
  - L'interrupt diventa indispensabile per fenomeni "urgenti" dove la modalita' Controllo di Programma non e' piu' sufficiente a garantire un corretto funzionamento.
- **Direct Memory Access (DMA)**: i due riferimenti temporali rimangono indipendenti e solo al termine di un'attivita' di interazione completa I/O si cerca una sincronizzazione.
  - Il DMA e' particolarmente utile quanto una singola operazione di I/O e' costituita da molti singoli passi (ad esempio durante grossi trasferimenti da RAM a disco rigido). In questi casi l'interfaccia esegue autonomamente le singole operazioni di I/O e avvisa la CPU solo a lavoro finito, evitandole l'onere di occuparsi esclusivamente di quelle attivita'.

Controllo di Programma

Facciamo un esempio di attivita' di I/O di una tastiera tramite Controllo di Programma:

- Nel registro Comando dell'interfaccia viene acceso il bit di "attesa tasto" (si attiva l'interfaccia ad attendere la pressione di un tasto sulla periferica);
- Si legge il bit "tasto premuto" nel registro Stato ed effettuamo un controllo: se la condizione e' verificata si va avanti, ma finche' la risposta e' no continuamo a leggere il bit ed effettuare il controllo;
- Possiamo prelevare il contenuto del registro DATO-IN (esempio codice ASCII del tasto premuto) e copiarlo in un registro GPR della CPU;
- Controlliamo se il bit "video libero" del registro Stato e' abilitato, altrimenti attendiamo;
- Quando il video e'pronto a ricevere il nostro dato copiamo il contenuto del registro GPR precedentemente inserito nel registro DATO-OUT dell'unita' controllo interfaccia;

- Infine accendiamo il bit "visualizza carattere" nel registro Comando.

In caso di **piu' periferiche** si puo' decidere a programma con quale frequenza interrogare ciascuna interfaccia e che priorita' assegnarle nel caso di piu' periferiche pronte ad effettuare operazioni di I/O.

In particolare notiamo come nel ciclo precedentemente descritto ci sono due condizioni d'attesa:

- STATO.tasto_premuto = TRUE ?
- STATO.video_libero = TRUE ? Se queste due condizioni non sono immediatamente verificate, comportano un'attesa e dunque una continua e periodica interrograzione da parte della CPU verso l'unita' di controllo interfaccia fino al verificarsi della condizione che consente di proseguire. Invece di sprecare tutto questo tempo si possono svolgere altre attivita' che non hanno bisogno della verifica di quei requisiti. Per fare cio' serve un Sistema Operativo **Multitasking** che gestisca la ripartizione del tempo di CPU tra attivita' (processi o task) differenti.

Il Controllo di Programma **non e' piu' adatta** in due circostanze:

- **Fenomeni urgenti**: quando la CPU e' impegnata in attivita' onerose puo' trascurare troppo a lungo la periferica i cui fenomeni richiedono di essere gestiti entro un certo tempo limite (scaduto il quale il calcolatore non rispetta piu' le specifiche di funzionamento: come in casi di allarme dove il computer deve prendere opportuni provvedimenti entro un certo periodo di tempo);
- **Fenomeni che si ripetono ad elevata frequenza**: poiche' le operazioni di I/O richiedono un certo numero di istruzioni macchina che rispettino il ciclo fetch-decode-execute, qualora il contesto richieda esclusivamente di leggere/scrivere da/su periferiche esterne l'ammontare di istruzioni macchina diventa troppo elevato tanto che le alcune operazioni possono definirsi superflue.

Input/Output a Interrupt

Principio di funzionamento dell'Interrupt

Consente alla periferica di segnalare la **necessita' di servizio** alla CPU. Questa segnalazione avviene tramite l'invio di un segnale mediante una linea del bus di controllo dedicata **INTREQ** (Interrupt Request).

L'interrupt si rivela particolarmente adatto a gestire fenomeni urgenti che non possono attendere un tempo "forzatamente casuale" di interrogazione di una soluzione a controllo di programma. Non risolve pero' il problema dei fenomeni che si ripetono ad alta frequenza.

Struttura dell'interrupt:

- Richiesta di interruzione da parte della periferica (tramite la linea INTREQ del bus di controllo);
- Riconoscimento di una richiesta di interrupt da parte della CPU al termine dell'istruzione in corso (tramite la linea INTACK: Interrupt Acknowleged del bus di controllo);
- Salvataggio automatico da parte della CPU del Program Counter attuale;
- Salto a **sub-routine** di risposta all'interrupt (chiamata a sottoprogramma generata dall'harware);
- La sub-routine di risposta deve preoccuparsi di **salvare il contesto** del programma interrotto;
- Al termine della risposta la sub-routine ripristina il contesto e si ritorna alla posizione salvata del programma interrotto.

In un calcolatore possono esistere piu' periferiche che richiedono gestione a interrupt e non c'e' alcuna possibilita' di sincronizzazione tra le richieste di interrupt (poiche' le richieste sono indipendenti e provengono dal mondo esterno al calcolatore): ogni interfaccia a periferica interrompe quando la propria periferica lo richiede;

La linea di richiesta INTREQ (**attiva bassa**) deve essere gestita mediante porte **OPEN COLLECTOR** dove:

- chi vuole interrompere forza a 0 a bassa impedenza la linea;
- normalmente la linea e' tenuta a 1 dalla resistenza di pull-up.

Interrupt cablato

Quando la CPU e' dotata di linee INTREQ e INTACK, nel caso di interrupt cablato, alla ricezione di un interrupt:

- Salva il valore del Program Counter;
- Disabilita il riconoscimento di ulteriori interrupt;
- Attiva la linea INTACK (segnala alla periferica di aver accettato l'interruzione);
- Salta ad un indirizzo di risposta predefinito.

La routine di risposta deve effettuare il **polling** delle interfacce che possono essere la sorgente dell'interruzione: ogni interfaccia deve essere interrogata per sapere se e' la responsabile dell'interruzione. L'ordine di interrogazione delle periferiche e' associato all'urgenza della periferica: le periferiche che hanno piu' urgenza ad essere servite verranno interrogate prima.

Lo schema circuitale tra le interfacce di IO che possono interrompere la CPU e' il meccanismo a **Daisy Chain**. Quando un'interfaccia riceve **INTIN**:

- se non ha richiesto interrupt, propaga **INTOUT**;

- se ha richiesto interrupt, non propaga INTOUT e attende di essere interrogata dalla CPU.

La priorita' delle periferiche e' associata alla posizione fisica delle interfacce rispetto alla CPU: le interfacce piu' vicine hanno priorita' maggiore e la priorita' non e' modificabile a run-time.

Se una periferica ha gia' propagato INTOUT non puo' piu' interrompere perche' ha gia' passato la priorita' all'interfaccia successiva. Deve dunque attendere la fine del ciclo di interrupt per poter effettuare una nuova richiesta di interrupt.

Se la CPU viene riabilitata a sentire gli interrupt, una periferica ad alta priorita' puo' essere interrotta da una a bassa.

L'interrupt cablato e' semplice ma presenta dei limiti:

- il polling rallenta l'inizio delle attivita' di risposta vere e proprie (in una fase in cui l'urgenza e' fondamentale, perdendo dunque tempo prezioso);
- la priorita' delle periferiche e' fissa;
- se la CPU non viene riabilitata agli interrupt, lo rimane fino al servizio della periferica interrompente: se questo servizio e' lungo la CPU rimane "cieca" rispetto nuove richieste di interruzione magari piu' urgenti;
- se invece la CPU viene riabilitata agli interrupt, puo' essere interrotta da una periferica meno urgente di quella in servizio.

Interrupt vettorizzato

Le linee sono sempre INTREQ (attiva bassa) e INTACK.

Alla ricezione dell'interrupt, la CPU:

- salva il valore del PC al termine dell'istruzione che stava svolgendo;
- disabilita il riconoscimento di ulteriori interrupt;
- attiva l'accettazione INTACK;
- attende che sul Data Bus compaia un **identificativo a 8bit** inserito da parte dell'interfaccia a periferica che serve all'unita' centrale come indice in un **vettore di interrupt**:
  - si tratta di una tabella di celle di memoria, dove ogni cella e' associata ad ogni possibile sorgente di interrupt (poiche' l'identificativo e' a 8bit abbiamo 2^8=256 possibili celle e sorgenti di interrupt);
  - ogni cella contiene l'indirizzo di inizio della sub-routine di risposta all'interrupt associato: a tutte le periferiche che hanno un indentificativo definito e che possono interrompere la CPU viene inserito, in fase di inizializzazione del sistema, nella cella corrispondente a quell'identificativo, l'indirizzo di inizio della risposta di interrupt associato.

Il principale vantaggio dell'interrupt vettorizzato e' la bassa latenza per il riconoscimento della sorgente di interrupt in quanto non serve il polling.

Ci sono pero' anche degli svantaggi:

- le interfacce a periferica si complicano:
  - ogni interfaccia deve essere capace di generare il proprio identificativo sul Data Bus non appena riceve il segnale INTACK;
  - ogni interfaccia deve sapere il proprio identificativo.
- non abbiamo risolto i problemi di priorita':
  - se la CPU viene riabilitata a sentire gli interrupt, puo' essere interrotta da chiunque.

Per risolvere i problemi di interfacce complicate e priorita' non gestita introduciamo il componente denominato **PIC** (Programmable Interrupt Controller). Si tratta di un circuito integrato di supporto alla gestione degli interrupt:

- L'interfaccia comunica al PIC la richiesta di interrupt;
- Se l'interfaccia e' abilitata, il PIC attiva INTREQl;
- Quando riceve INTACK, il PIC comunica l'identificativo della periferica sul Data Bus:
  - gli identificativi delle periferiche sono gestiti dal PIC riducendo la complessita' delle interfacce.
- Se il PIC riceve piu' richieste di interrupt da diverse periferiche e' in grado di scegliere quella piu' prioritaria, in quanto la priorita' e' programmabile a livello software.

I registri del PIC sono accessibili alla CPU come normali registri di interfaccia:

- **IVR** - Interrupt Vector Register: contiene l'identificativo associato a ciascuna periferica collegata (si tratta di una tabella di byte, dove ci sara' spazio per 1byte per ogni periferica collegabile al PIC, normalmente 8);
- **IPR** - Interrupt Priority Register: contiene le informazioni necessarie per stabilire l'ordine di priorita' delle periferiche (riprogrammabile dalla CPU in qualsiasi momento, anche a run-time);
- **IMR** - Interrupt Mask Register: contiene le informazioni per sapere quali periferiche possono generare interruzioni e quali no (riprogrammabile come IPR).

L'interrupt vettorizzato unito all'inserimento del PIC presenta i seguenti vantaggi:

- Il riconoscimento della sorgente di interrupt e' estremamente rapido e l'interfaccia di controllo periferica non deve essere piu' complessa;
- La priorita' delle periferiche e' dinamicamente variabile;
- Grazie al registro IMR, prima che la CPU venga riabilitata agli interrupt, si puo' decidere quali periferiche potranno interrompere quella in servizio, riprogrammando l'IMR.

Input/Output a DMA

Modalita' di I/O a DMA

Alcuni fenomeni di I/O si ripetono ad alta frequenza ed effettuano un trasferimento di sequenze di dati da periferica a memoria o viceversa:

- Trasferimento di settori da/verso memoria di massa;
- Trasmissione/Ricezione di frames da rete.

Solo quando il trasferimento dell'intera sequenza e' terminato, si puo' procedere con l'elaborazione.

Quando si effettuano queste operazioni, la CPU ne risulta fortemente penalizzata in quanto deve:

- Effettuare numerossissimi fetch di istruzioni (uno per ogni cella di memoria da trasferire);
- Incrementare il puntatore all'area di memoria da/in cui trasferire i dati;
- Aggiornare il contatore dei dati trasferiti.

Per evitare che sia la CPU ad occuparsi del trasferimento di ogni singola cella di memoria e' stato messa appunto la DMA - Direct Memory Access: questa tecnica prevede la possibilita' che altri dispositivi, oltre la CPU, possano accedere a memoria occupandosi loro dei trasferimenti. Questi dispositivi chiedono alla CPU la possibilita' di usare il Bus e la promozione a Master. Si fa uso di una linea dedicata del Control Bus: **HOLDREQ** (Hold Request) dove avviene una richiesta di sospensione di interazione alla CPU sul bus di sistema.

Dunque anche gli stadi di uscita della CPU che pilotano le linee dell'Address Bus e del Control Bus devono essere TRI-STATE per lasciare agli altri Master la possibilita' di pilotarle.

Anche la linea HOLDREQ, come la INTREQ, e' attiva-bassa poiche' in un calcolatore possono esistere piu' periferiche che richiedono DMA e come nel caso dell'interrupt non c'e' possibilita' di sincronizzazione delle richieste. La linea HOLDREQ e' gestita mediante porte OPEN COLLECTOR:

- Chi vuole i bus forza la linea a 0 a bassa impedenza;
- Normalmente la linea e' tenuta a 1 dalla resistenza di pull-up.

Come nell'interrupt vettorizzato esiste un circuito integrato di supporto alla gestione: DMAC (DMA Controller) La CPU programma il DMAC comunicando:

- Indirizzo della zona di memoria da/in cui deve trasferire i dati;
- Numero di dati da trasferire;
- Identificativo della periferca tra tutte quelle che possono trasferire i dati e senso di trasferimento.

Quando la periferica segnala di essere pronta a trasferire un singolo byte:

- Il DMAC richiede i bus con il segnale HOLDREQ;
- Quando la CPU ha terminato l'uso, eventuale, del bus ne segnala il rilascio con HOLDACK sul Controller Bus (HOLD AKNOWLEDGED).
- Il DMAC effettua il trasferimento e aggiorna puntatori e contatori;
- Quando l'intero trasferimento e' finito, segnala al software in esecuzione sulla CPU che tutta l'operazione e' finita mediante la generazione di un interrupt.

I registri del DMAC sono accessibili dalla CPU come normali registri di interfaccia programmabili:

- **PA** - Peripheral Address: contiene l'identificativo dell'interfaccia a periferica con cui interagire per scambiare i dati;
- **MDA** - Memory Data Address: contiene l'indirizzo della prossima cella di memoria in cui inserire o da cui prelevare il dato;
- **DC** - Data Counter: contiene il numero di dati ancora da trasferire, inizializzato alla lunghezza della tabella di dati;
- **TD** - Transfer Direction: indica se l'operazione e' il lettura o in scrittura.

Il DMAC ha il vantaggio di essere realizzato per questo scopo e non perde tempo per scoprirlo da programma con numerose fetch. Il trasferimento dati avviene in modo trasparente al programma in esecuzione sulla CPU (che viene semplicemente rallentata perche' deve occasionalmente rilasciare i bus ma non c'e' percezione a livello software di un'attivita' che si svolge completamente a livello hardware).

Comportamento HW/SW durante I/O in DMA

Esempio: lettura di un settore da disco. Il programma in esecuzione da parte della CPU richiede il trasferimento di un settore da 1KByte da memoria di massa (supporti esterni come cd) a disco fisso. Il calcolatore e' dotato di DMAC per le operazioni I/O su disco fisso e di un'interfaccia di I/O per comunicare con la memoria di massa.

Operazioni SW da parte della CPU:

- Il programma in esecuzione chiama la routine READISK (pezzo di programma in linguaggio macchina appartenente al Sistema Operativo);
- La routine READISK inizializza il DMAC:
  - Inserisce nel registro PA l'identificativo dell'interfaccia;
  - Inserice nel registro MDA l'indirizzo della zona di memoria che fa da buffer (zona da 1KByte nella quale trasferire il settore in ingresso) del disco fisso;
  - Inserisce nel registro DC il valore 1024 (numero di byte da leggere);
  - Inserisce in TD l'indicazione di lettura (IN dall'esterno, dunque un trasferimento da memoria di massa esterna a disco fisso interno).

- La routine READISK inizializza l'interfaccia I/O per la memoria di massa esterna:
  - Comunica il numero della traccia e del settore da leggere da memoria esterna indicando che si tratta di una lettura.
- Il Sistema Operativo sospende il programma in esecuzione e lancia altre attivita'.

Quando l'interfaccia a memoria esterna presenta un byte pronto ad essere trasferito, segnala il dato pronto al DMAC. Il DMAC chiede l'uso dei bus alla CPU attivando la linea HOLDREQ. La CPU rilascia i bus attivando la linea HOLDACK (quest'azione e' puramente elettrica mettendo in alta impedenza i propri piedini, non accendendo in nessun modo al bus). Il DMAC:

- Pone sull'Address Bus il contenuto di MDA;
- Da' il segnale IN all'interfaccia e WRITE al disco fisso:
  - Il dato viene scritto direttamente da interfaccia a disco fisso mediante il Data Bus;
- Al completamento del trasferimento di 1byte il DMAC incrementa MDA per puntare alla prossima cella da trasferire e decrementa DC per indicare che 1byte e' stato trasferito quindi il totale dei byte da trasferire e' diminuito di 1;
- Il DMAC toglie la richiesta HOLDREQ di modo che la CPU possa procedere con le sue operazioni. La CPU disattiva HOLDACK e torna ad essere Master del bus (operazione puramente HW). Il trasferimento del singolo byte comporta una semplice attesa nell'uso dei bus da parte della CPU (nessuna attivita' SW e' coinvolta).

Dopo il trasferimento dell'ultimo byte (il registro DC del DMAC e' arrivato a 0), il DMAC deve segnalare al software in esecuzione sulla CPU che l'intera operazione di trasferimento e' conclusa e genera pertanto una richiesta di interrupt. La CPU a livello HW riconosce questa interruzione e secondo il metodo previsto (interrupt cablato o interrupt vettorizzato) attiva la routine di risposta. Parte dunque una routine SW che puo' segnalare alla routine di Sistema Operativo READISK che l'operazione e' finita (per esempio forzando a TRUE un opportuno flag). A questo punto la routine READISK del Sistema Operativo puo' riattivare il programma che aveva richiesto la lettura del settore da memoria di massa. La percezione a livello SW dell'attivita' di lettura/trasferimento si ha solo ad attivita' completata mediante interrupt.