Appunti di Architettura e Reti logiche

Realized by Francesco Saonara

Avvertenze

Il seguente materiale didattico è stato da me scritto in tarde ore e mai riletto. Mi hanno già avvisato della presenza di errori grammaticali, logici, di lessico, in italiano come anche in inglese.
Nonostante le innumerevoli incomprensioni, spero che tali appunti potranno esservi utili, così come lo sono stati per me.

Indice

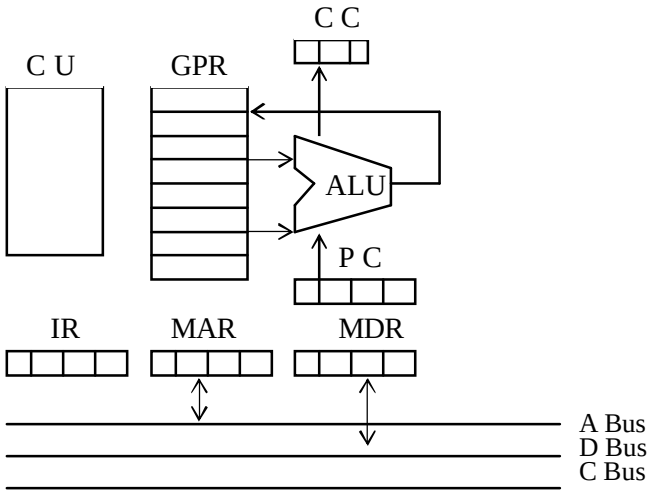
- **Teoria**

  - Struttura interna dell'LC2 _____4
  - ALU (Arithmetic and Logic Unit) _____4
  - Full Adder (interno all' ALU) _____5
  - Il moltiplicatore _____6
  - Memoria Cache _____7
  - Come lavora il Gestore di Cache _____8
  - Politica Fully Associative _____9
  - Politica Set Associative _____10
  - La memoria virtuale _____11
  - Pipeline _____12
  - Problemi dovuti ai legami fra istruzioni _____13
  - I Problema: Controllo _____13
  - II Problema: Data _____14
  - III Problema: Resurce_____15
  - Interazione fra CPU e mondo esterno _____16
  - Modalità per sincronizzare il mondo della CPU e delle periferiche _____17
  - Interrupt Cablato _____17
  - Interrupt Programmabile o Vettorizzato _____20
  - DMA-DMAC_____21
  - Unità di controllo cablata _____23
  - Unità di controllo microprogrammata _____24

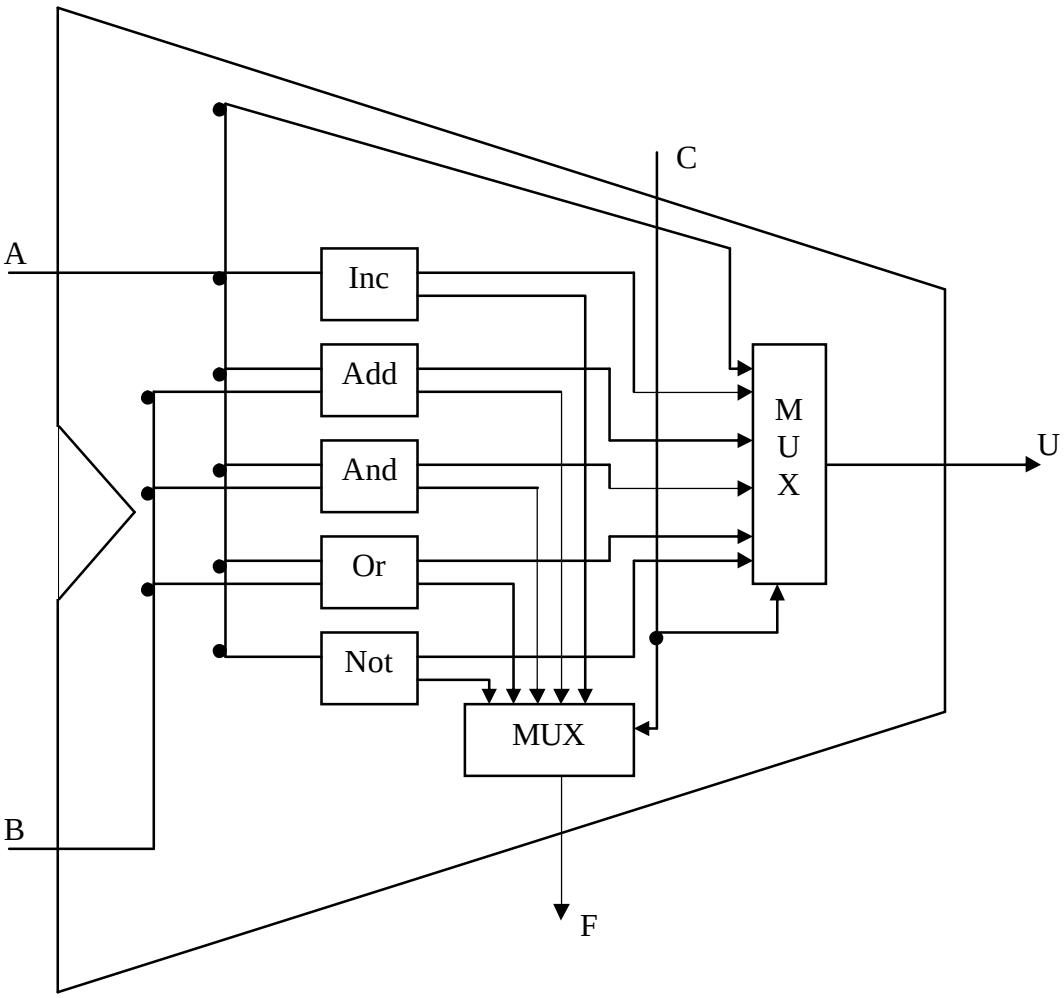
- **Esercizi**

  - Appello di esame – 7 Novembre 2005 _____26
  - Appello di esame – 14 Settembre 2005 _____28
  - Appello di esame – 18 Aprile 2005_____29
  - Appello di esame – 11 Luglio 2005_____30
  - Appello di esame – 13 Giugno 2005 _____31
  - Appello di esame – 21 Febbraio 2005 _____33

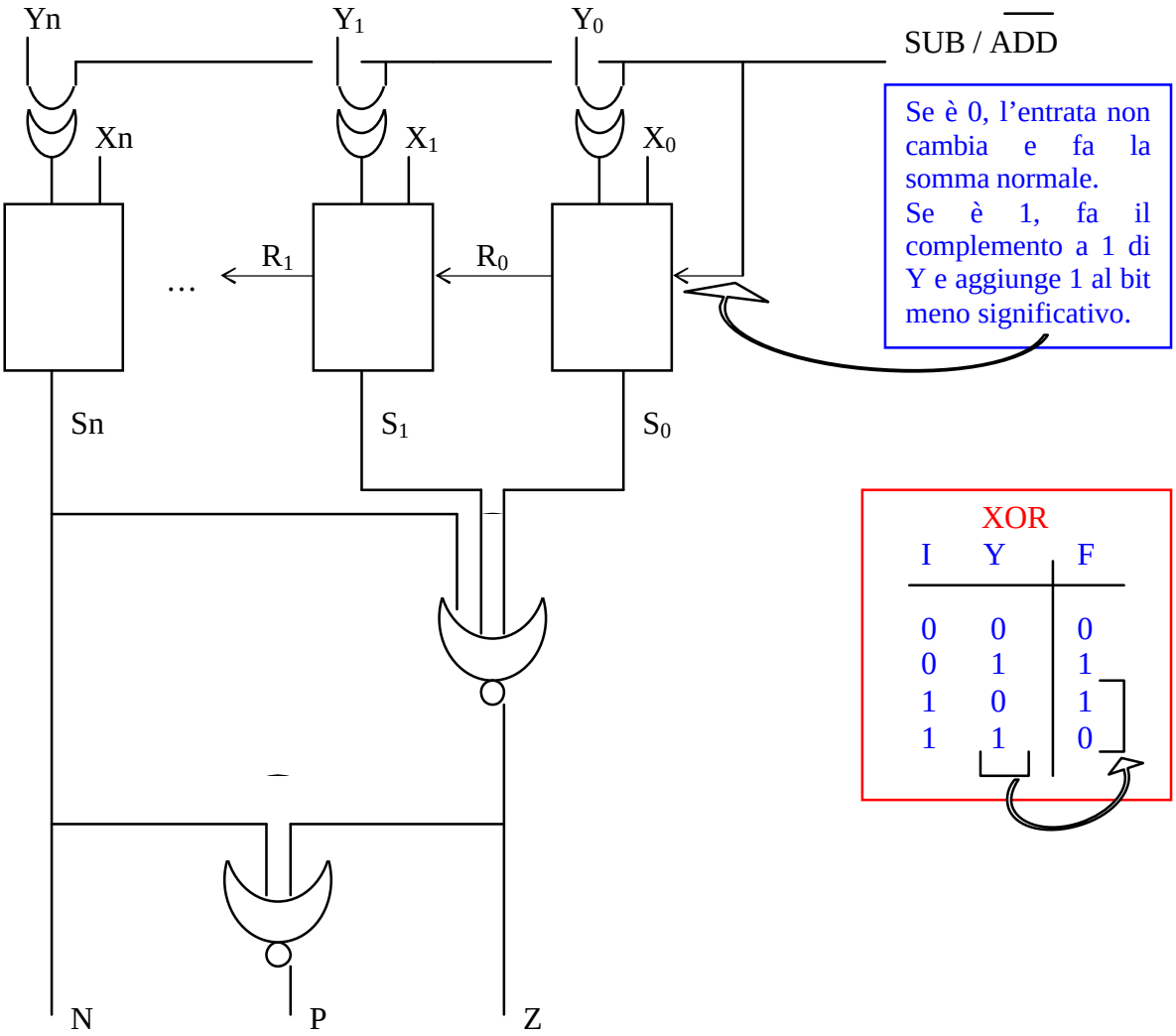
Struttura interna dell' LC2



ALU (Arithmetic and Logic Unit)

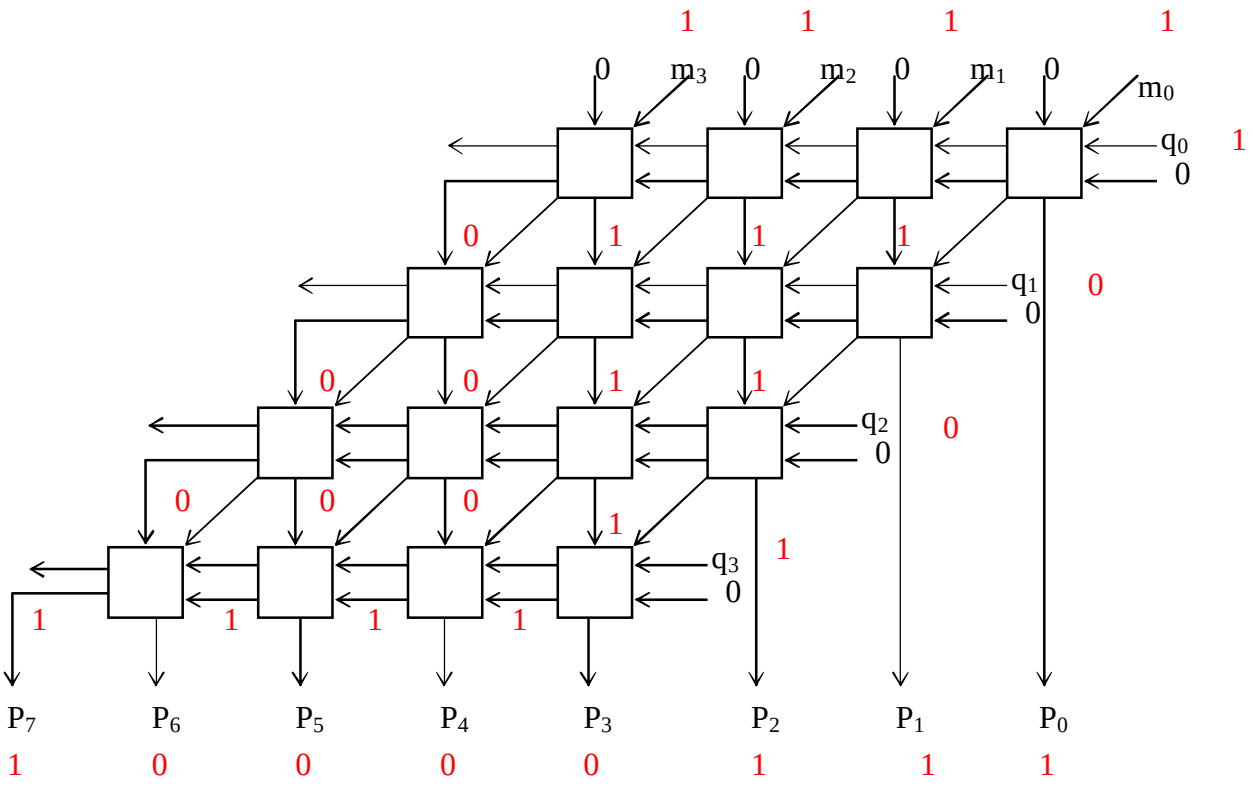


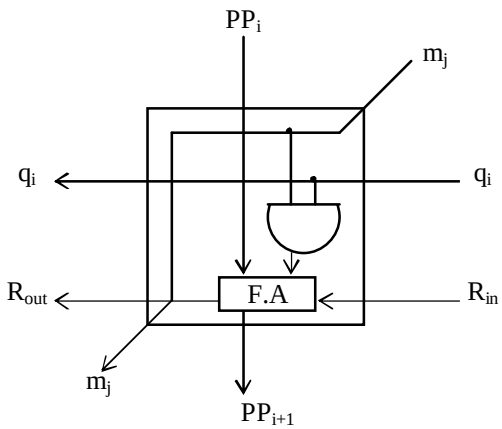
Full Adder (interno all' ALU)



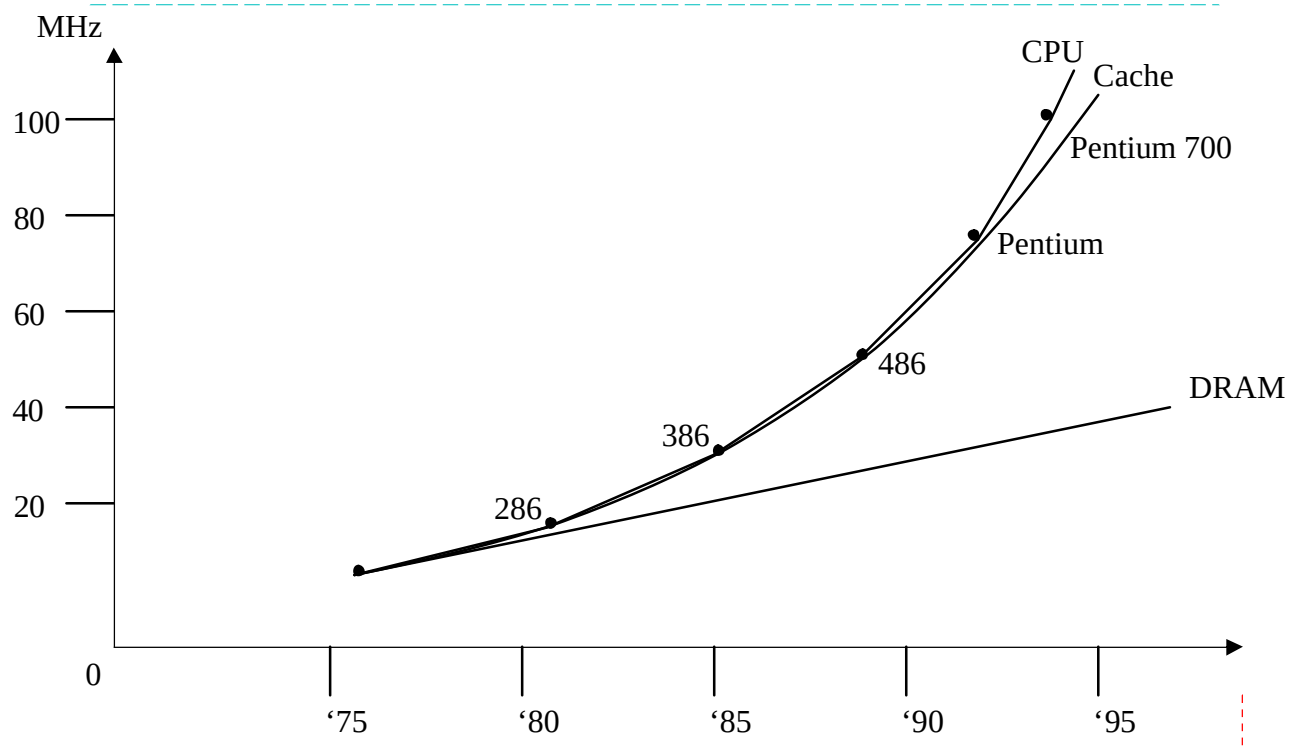
Domanda a cui non si è data risposta: Problema del calcolo del riporto e possibili soluzioni.

Il moltiplicatore

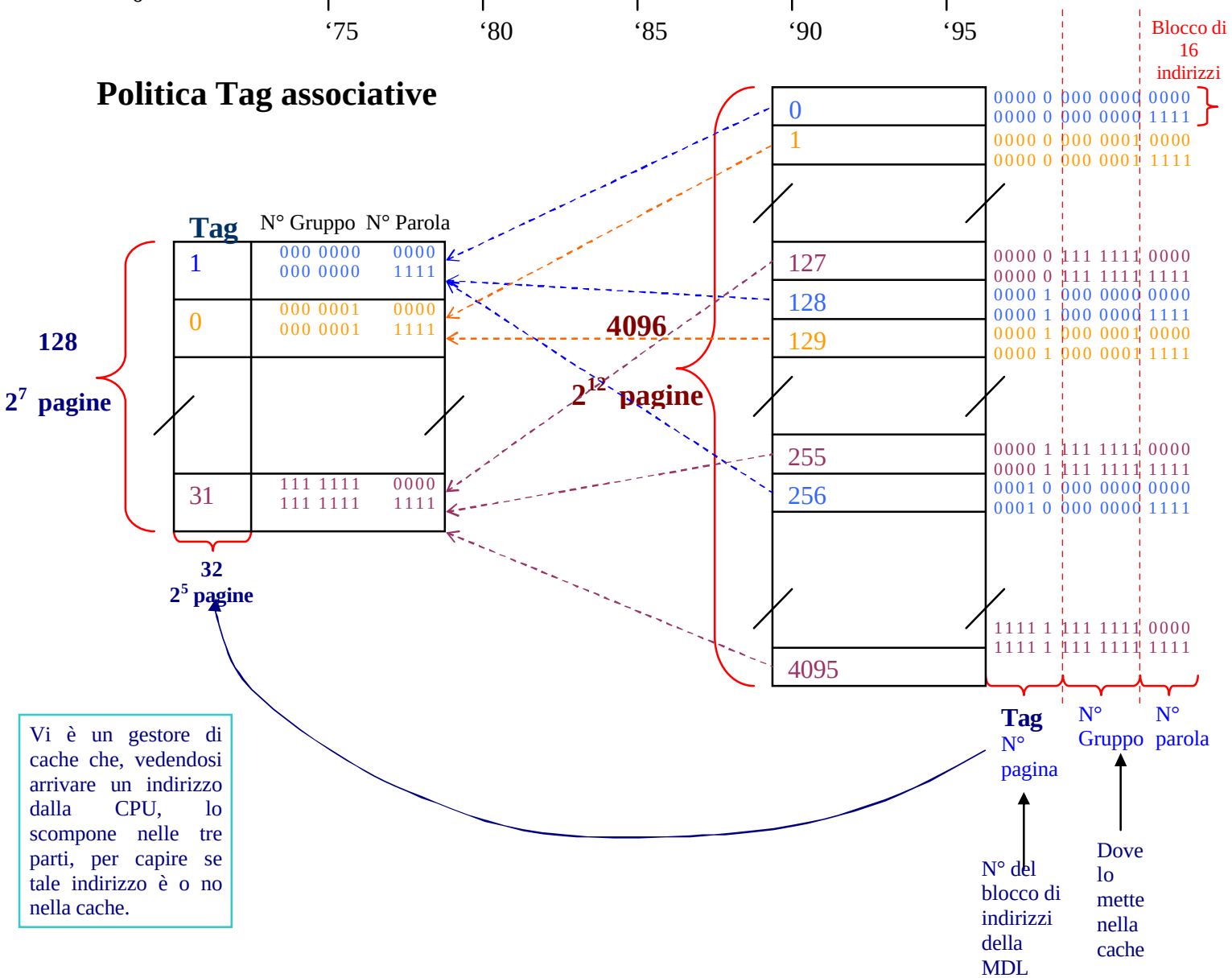




Memoria Cache



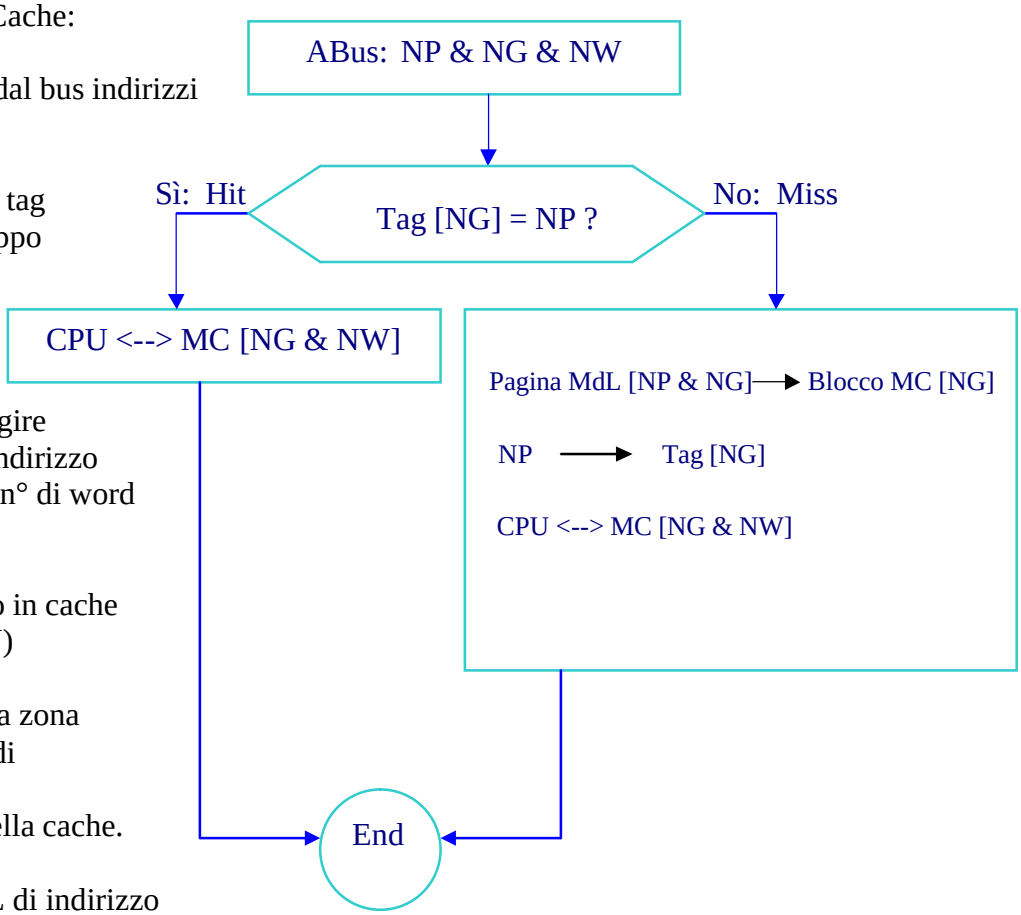
Politica Tag associative



Vi è un gestore di cache che, vedendosi arrivare un indirizzo dalla CPU, lo scompone nelle tre parti, per capire se tale indirizzo è o no nella cache.

Come lavora il Gestore di Cache:

Dalla CPU arrivano 16 bit dal bus indirizzi



Se nella cella della colonna tag in posizione numero di gruppo c'è il n° di pagina richiesto.

Se sì: Hit (colpito) consente alla CPU di interagire con la cella della cache di indirizzo n° di gruppo concatenato a n° di word

Se no: Miss (non ho trovato in cache la pagina cercata dalla CPU)

la CPU vuole leggere in una zona potenzialmente calda, quindi devo prendere quella parte della memoria e copiarla nella cache.

Prendo la pagina della MdL di indirizzo NP&NG e la porto nel blocco MC in posizione NG

Devo ricordarmi che nella cache c'è qualcosa di diverso e quindi il NP del gruppo lo devo andare a scrivere nella memoria cache nel tag di NG, per dire che non è più quella di prima, ma è un'altra pagina di quel gruppo quella presente nella cache.

In fine faccio interagire la CPU con la MC di indirizzo NG&NW

Vi è però 1 problema: devo sapere se la CPU vuole accedere a memoria per leggere o per scrivere. Per risolvere questo problema, vi sono 2 soluzioni:

1_ Store thru (store thru cache carico attraversando la cache direttamente nella MdL)
2_Store in (store in cache carico in cache e aggiorno la MdL solo nel momento in cui cambio la cache)

1_Store thru: Ogni volta che apporto modifiche ad una cella della cache, modifico anche la corrispondente cella nella MdL. Vantaggio: l'originale e la copia rimangono allineati, quindi nel momento in cui vado a inserire nella cache un diverso blocco di indirizzi, posso sovrascrivere la copia, in quanto l'originale è già stato aggiornato. Svantaggio: le scritture sono lente. Considerando, però, che le fasi di fetch sono tutte letture e che le elaborazioni di dati sono quasi sempre sintesi (da tanti valori di ingresso produco pochi valori in uscita), posso anche accettare tali rallentamenti.

2_Store in: Associo ad ogni blocco caricato nella cache un bit che, se è settato su 0 significa che non ho apportato modifiche e quindi nel caso devo cancellarlo dalla cache non mi pongo problemi, viceversa, se è settato su 1 significa che è stato modificato e, quindi, aggiorno la MdL nel momento in cui lo voglio cancellare per mettercene un altro.

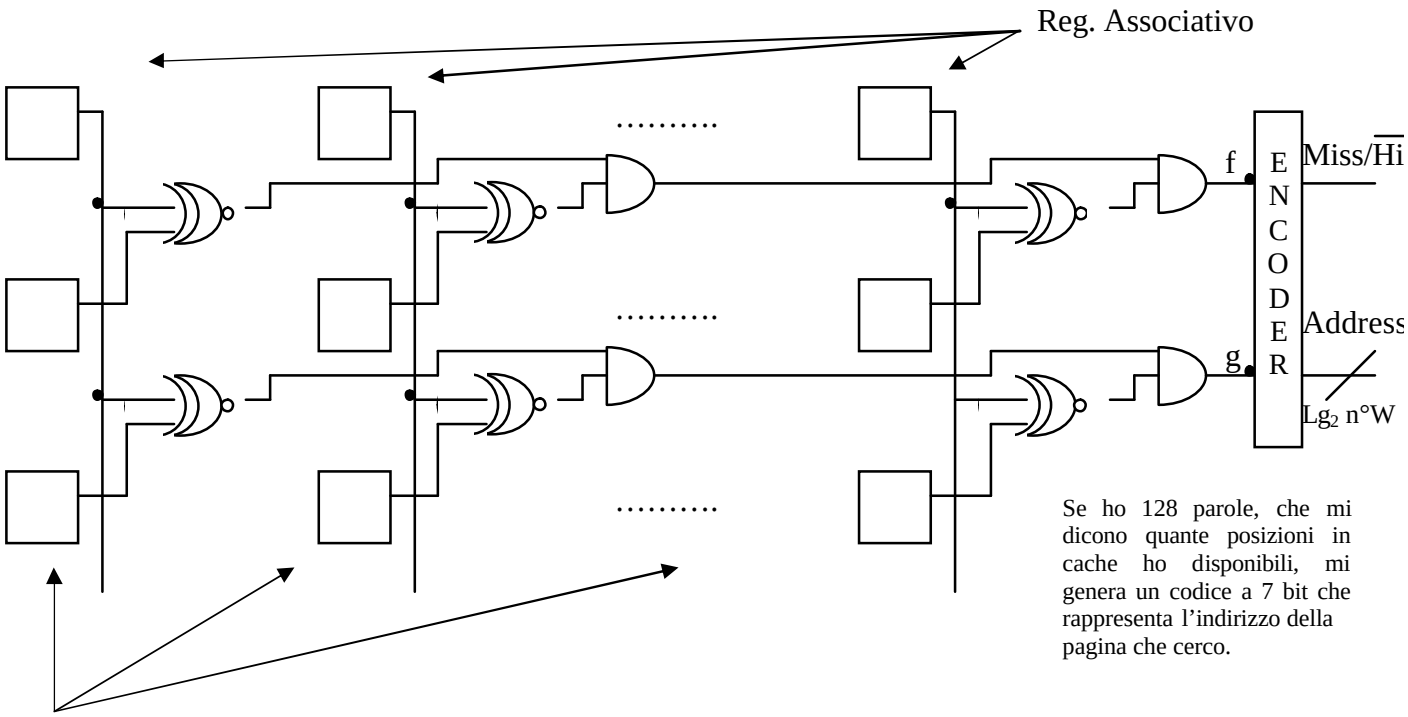
A questo punto però mi si pone un problema: se ho delle istruzioni che fanno riferimento a dei dati che fanno parte dello stesso gruppo, ma di un'altra pagina, nella cache non ho la possibilità di caricare tutti e due i blocchi di indirizzi (pagine).

La soluzione ideale sarebbe quindi avere nella cache le pagine di MdL più richieste, indipendentemente dal gruppo cui fanno parte.

Quindi elimino i gruppi, portando i tag da 5 a 12 bit. Di conseguenza non vi è più accoppiamento prestabilito fra alcune zone di memoria cache e MdL, ma ogni pagina di MdL può andare in qualsiasi pagina di cache e quindi l'indirizzo generato dalla CPU viene diviso dal Gestore di Cache in NP & NW. Non vi saranno più, quindi, 32 indirizzi possibili per identificare la pagina, bensì 4095 (nel nostro esempio dove la cache contiene 128 pagine).

Politica Fully Associative

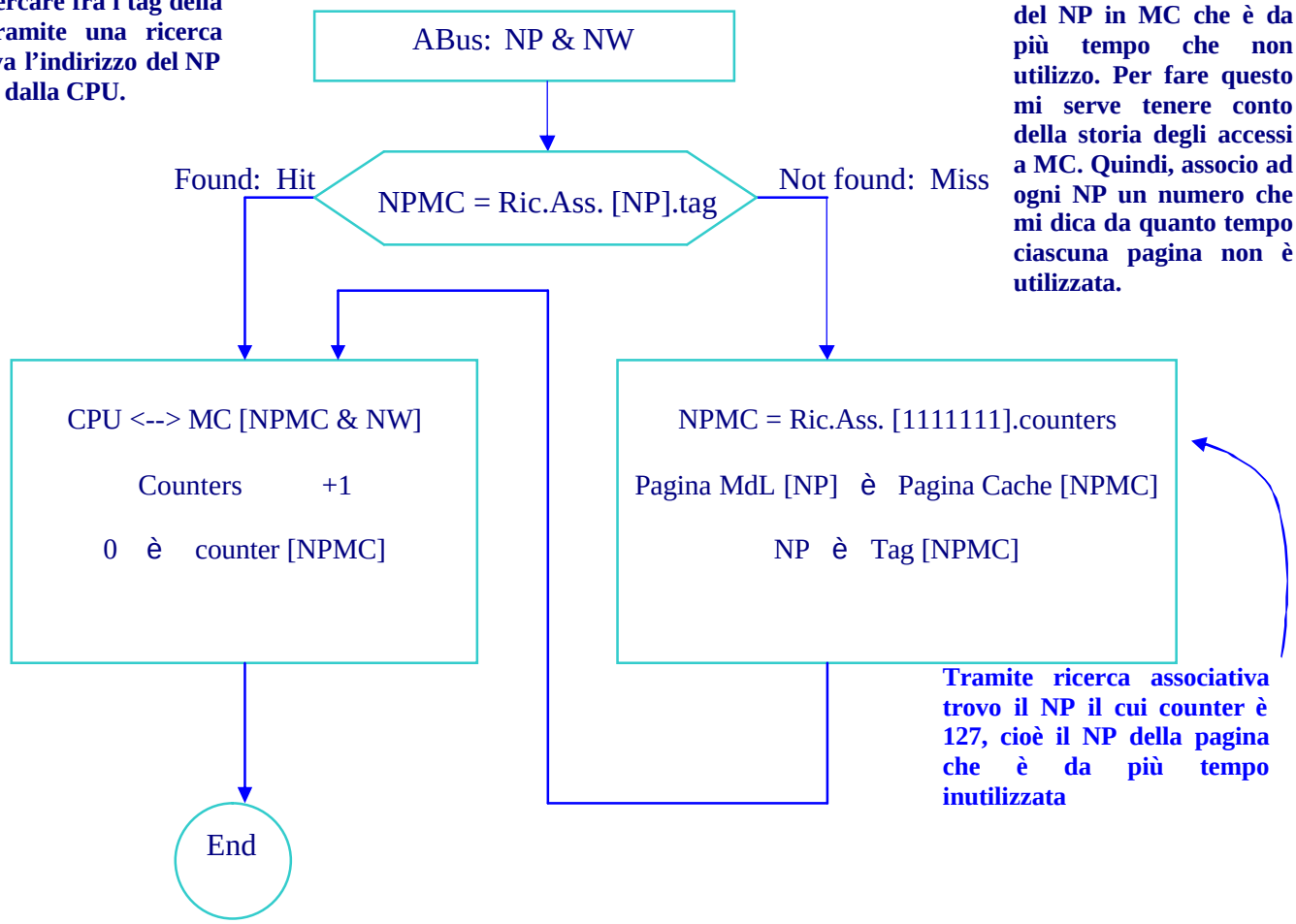
Nel momento in cui la CPU mi richiede l'accesso ad un indirizzo di memoria, non posso controllare in maniera sequenziale se tale indirizzo è presente nei tag della cache, ma devo trovare un sistema per avere una risposta immediata alla richiesta dell'indirizzo: se non è presente nella cache me lo deve dire subito, è se è presente, mi deve anche indicare la posizione in cui si trova.
Per fare questo è stata inventata la memoria associativa:



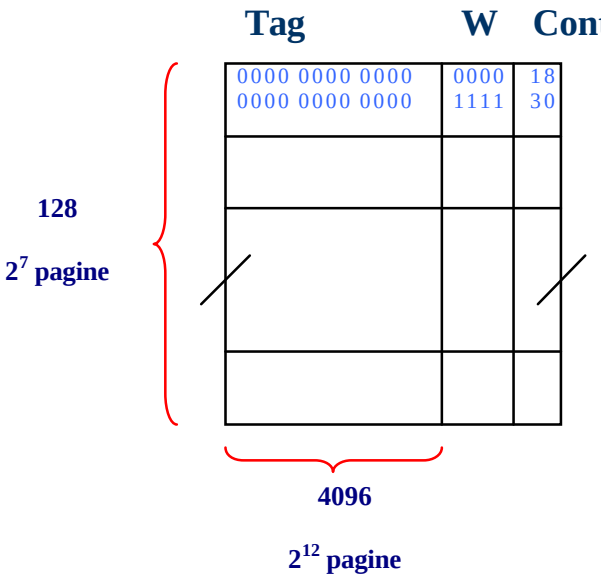
Celle della mia memoria (Latch D), con un proprio indirizzo, che contengono i dati.
La prima cella confronta il proprio contenuto con quello di tutte le altre celle tramite l'operatore or esclusivo negato, che dà come uscita 1 se sono uguali, 0 se sono diversi. Quindi, all'uscita f,g,… avrò 0 se vi è almeno un registro di quella riga il cui contenuto è diverso dal registro associativo, oppure 1 se tutti i registri di tale riga sono uguali al registro associativo, perché solo in quel caso passo tutti gli and.
Quindi se nel registro associativo scrivo l'indirizzo della pagina che mi interessa otterrò 1 solo se nel set della cache è presente tale indirizzo.

Vado a cercare fra i tag della Cache tramite una ricerca associativa l'indirizzo del NP chiamato dalla CPU.

Il NP che mi serve lo andrò a mettere al posto del NP in MC che è da più tempo che non utilizzo. Per fare questo mi serve tenere conto della storia degli accessi a MC. Quindi, associo ad ogni NP un numero che mi dica da quanto tempo ciascuna pagina non è utilizzata.



Tramite ricerca associativa trovo il NP il cui counter è 127, cioè il NP della pagina che è da più tempo inutilizzata



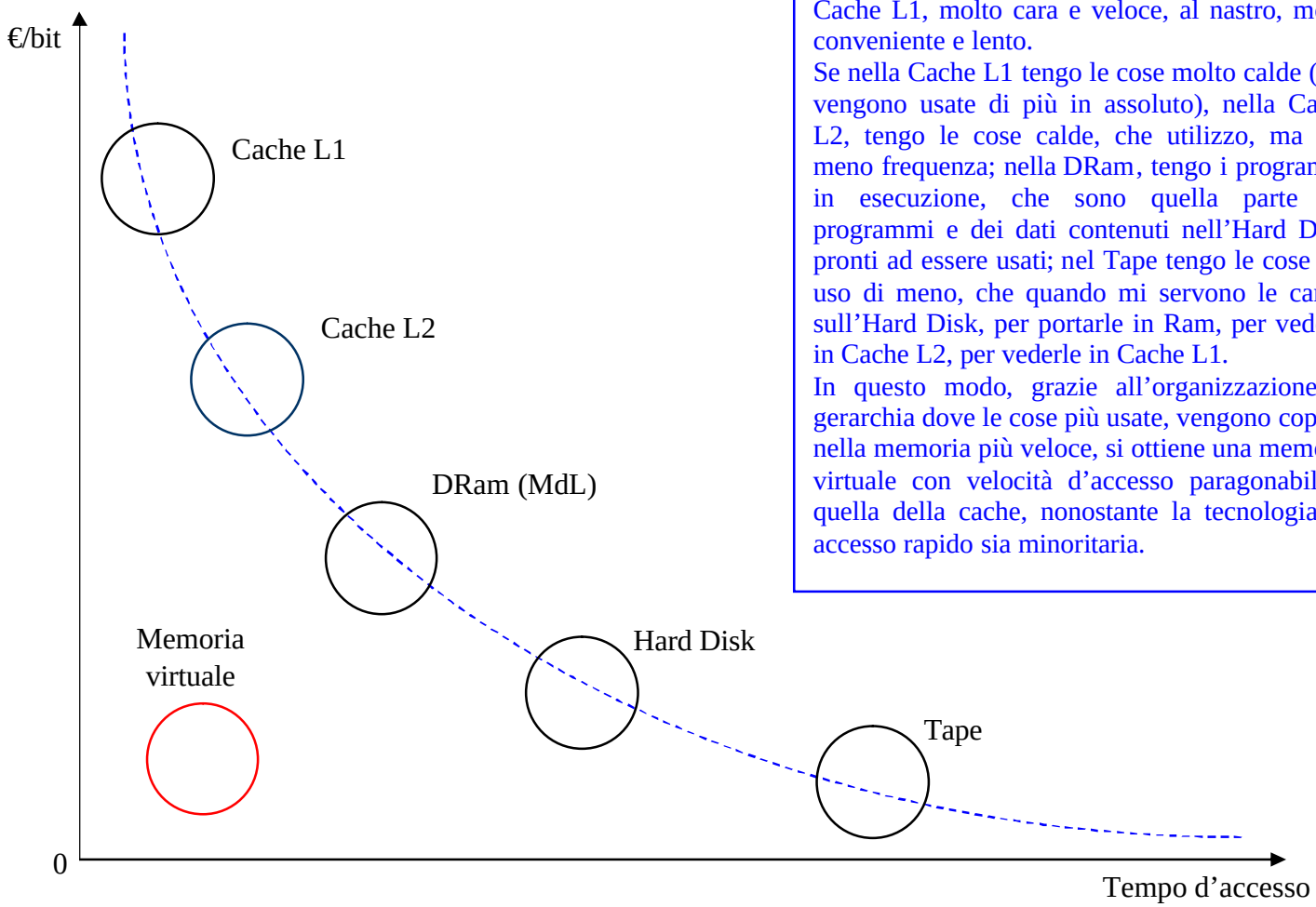
Ogni volta che viene inizializzato parte a contare da 0, ogni volta che conta si incrementa di 1 e quando arriva a fondo scala non riparte da 0, ma sta fermo a 127 (nel nostro caso).
Ogni volta che accedo in cache e quindi ho usato una parola di una certa pagina, azzero il suo contatore e incremento di 1 tutti gli altri contatori.
In ogni istante c'è almeno un contatore a 127.
In base alla politica Less recently used (meno recentemente usate) elimino le pagine con contatore a 127 e le sostituisco con quelle richieste dalla CPU.

Politica Set Associative

Invece di dire che ogni blocco della MdL può andare in un solo blocco della MC, oppure che ogni pagina può andare ovunque, dice che ogni pagina di MdL può andare in un certo sottoinsieme di NP della MC.

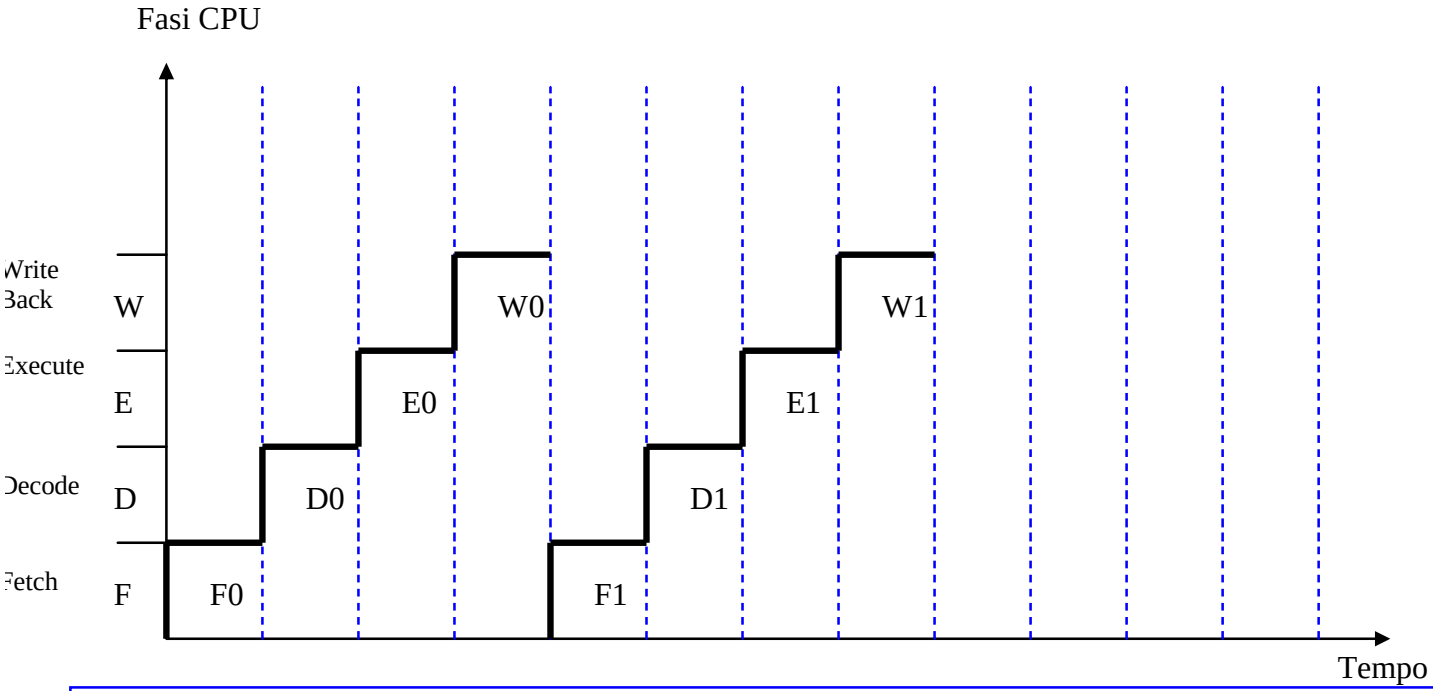
La memoria virtuale

La cache è utile inserirla direttamente nella CPU, nello stesso chip unità centrale, in modo da sfruttare la stessa tecnologia. Però, siccome tale chip è già occupato da molti altri circuiti, la cache potrà essere grande solo pochi kilobyte a fronte dei gigabyte della MdL. Per questo all'esterno della CPU costruisco una cache di secondo livello, un po' più grossa, ma un po' più lenta rispetto a quella di primo livello, presente nella CPU.

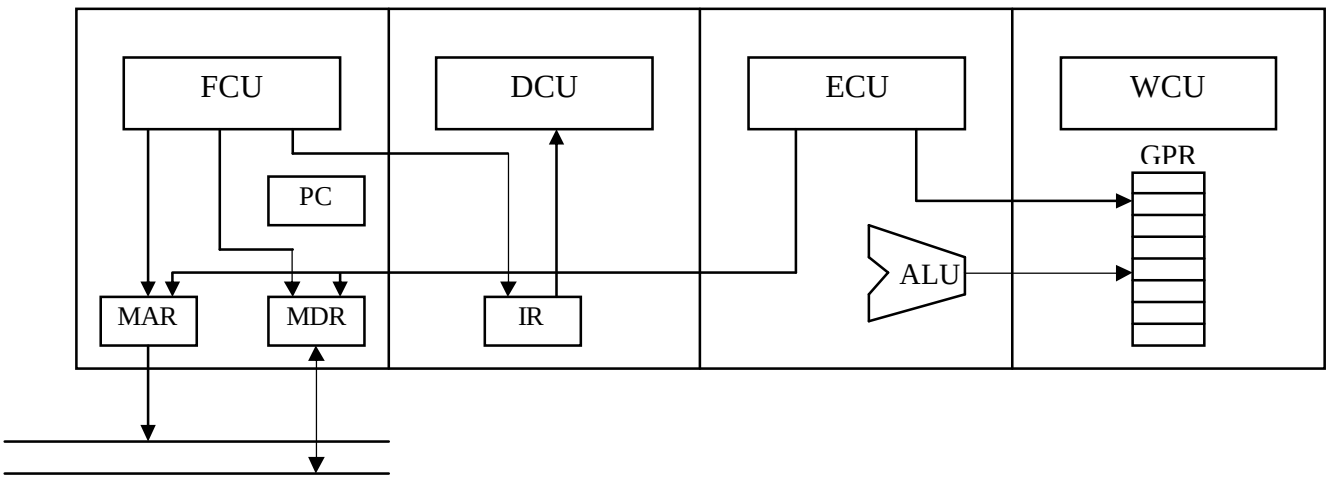


Le diverse tecnologie disegnano approssimativamente una iperbole, che va dalla Cache L1, molto cara e veloce, al nastro, molto conveniente e lento.
Se nella Cache L1 tengo le cose molto calde (che vengono usate di più in assoluto), nella Cache L2, tengo le cose calde, che utilizzo, ma con meno frequenza; nella DRam, tengo i programmi in esecuzione, che sono quella parte dei programmi e dei dati contenuti nell'Hard Disk, pronti ad essere usati; nel Tape tengo le cose che uso di meno, che quando mi servono le carico sull'Hard Disk, per portarle in Ram, per vederle in Cache L2, per vederle in Cache L1.
In questo modo, grazie all'organizzazione in gerarchia dove le cose più usate, vengono copiate nella memoria più veloce, si ottiene una memoria virtuale con velocità d'accesso paragonabile a quella della cache, nonostante la tecnologia ad accesso rapido sia minoritaria.

Pipeline



Andamento ciclico delle 4 fasi della macchina di Von Neumann.
Nella fase di Fetch ci serve il PC, che contiene l'indirizzo della prossima istruzione da eseguire, ci serve l'accesso al bus (MAR, MDR) per acquisire un'istruzione che dev'essere poi depositata nell'Instruction Register.
Nella fase di decode, l'Instruction Register viene letto, analizzato ed utilizzato; inoltre esso preleva e costruisce gli operandi (quando ad esempio si ha lo spiazzamento da un registro e bisogna costruire l'indirizzo unendolo col program counter).
Nella fase di esecuzione si usa l'Unità Aritmetico Logica per operazioni strettamente logiche e matematiche, non come nelle altre fasi dove viene usata l'ALU, ma solo per il passaggio di dati o per l'incremento del contatore.
Infine, nell'ultima fase, la Write Back, usa i registri General Pack Register per salvare il risultato delle istruzioni eseguite.

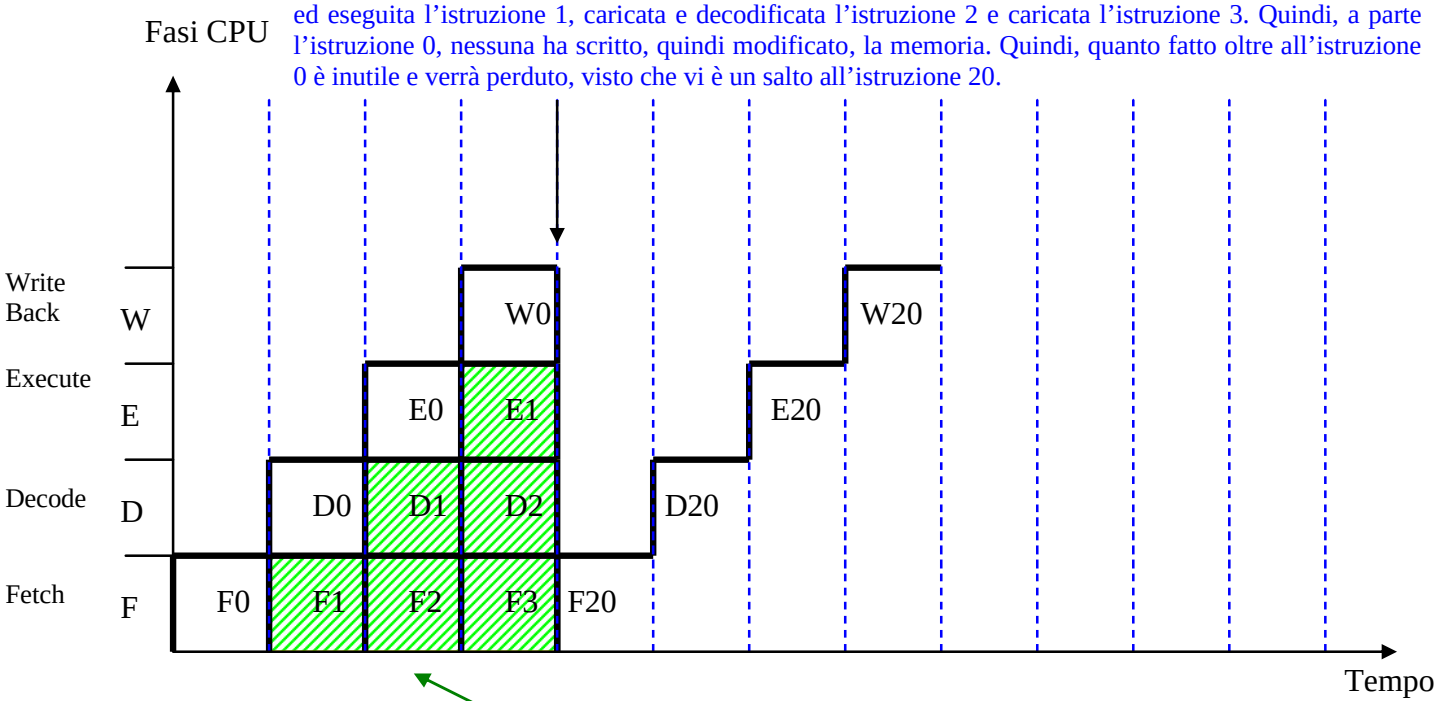


Problemi dovuti ai legami fra istruzioni:

Dependences:	**Risolto con:**
1- Control (Branch)	la Branch Prediction Table
2- Data (Out_j è In_{j+1})	Data Forwarding
3- Resource (Acc. a MDR e MDA)	Macc. Harvard

I Problema: Controllo

L'istruzione 0 è stata letta da memoria, ho scoperto che è un salto, eseguo, nel senso che vado a vedere le condizioni N,Z,P come stanno, scrivo nel PC la destinazione 20 a cui devo saltare. Quindi in questo istante scopro che non devo eseguire l'istruzione 1, ma la 20. Nel frattempo è stata caricata, decodificata ed eseguita l'istruzione 1, caricata e decodificata l'istruzione 2 e caricata l'istruzione 3. Quindi, a parte l'istruzione 0, nessuna ha scritto, quindi modificato, la memoria. Quindi, quanto fatto oltre all'istruzione 0 è inutile e verrà perduto, visto che vi è un salto all'istruzione 20.



Es:
I0 : BrNZP 20

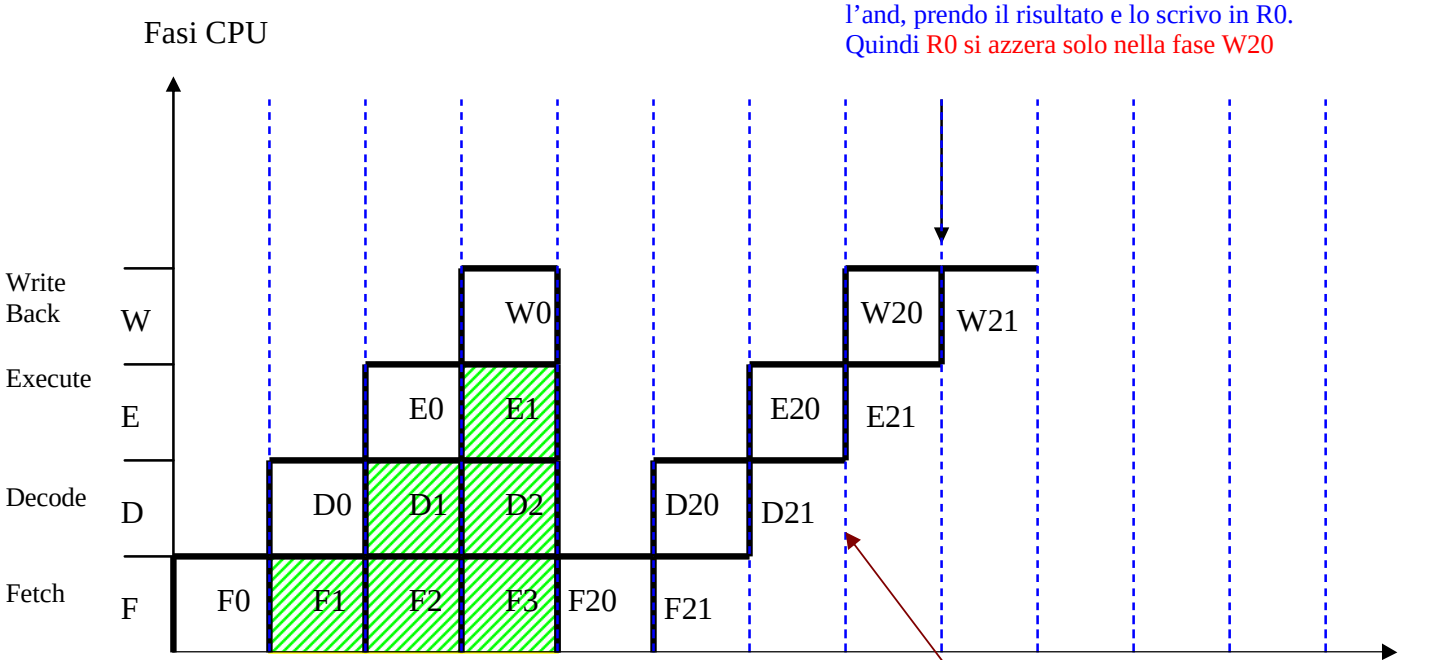
Istruzione 0

Svuotata la Pipeline, devo aspettare che l'istruzione 20 arrivi alla fase di Write Back, per avere un risultato. Così perdo il vantaggio della catena di montaggio, in quanto il lavoro eseguito in contemporanea alla prima istruzione viene perso. Ciò diventa un problema se tale situazione si presenta frequentemente. Siccome di solito la percentuale di operazioni di salto in un programma in linguaggio assembly è del 5-10%, tale problema non ci preoccupa.
Comunque, visto che ci vogliamo male, per eliminare tale problema, possiamo mettere un controllo al fetch che legga i primi 4 bit e, se l'istruzione caricata è BrNZP, copia nel PC l'indirizzo della cella cui fa riferimento.

Il problema si pone se il salto è condizionato e, quindi, non possiamo sapere già dalla Fetch se l'istruzione da eseguire dopo è la successiva o quella indicata come destinazione nell'istruzione di Branch. Per risolvere tale problema si crea una tabella in cui inserisco l'indirizzo delle celle in cui viene effettuato un salto, la cella di destinazione e un bit a cui assegno valore 1 se la cella è stata oggetto di Branch viceversa assegno valore 0. A questo punto, nel momento in cui il controllo della Fetch identifica un'istruzione Branch, controlla se è condizionata o no; nel caso lo sia, tramite ricerca associativa, cerca l'istruzione nella Branch Prediction Table e, se il bit associato è 0, prosegue normalmente con l'esecuzione dell'istruzione, altrimenti, se il bit è 1, inserisce nel PC l'indirizzo di destinazione contenuto nel Branch. Tale scelta è però basata sulla probabilità, in quanto non è sempre detto che se prima ho saltato allora lo farò anche adesso, così se trovo come bit 1 e mi metto a leggere,decodificare ed eseguire l'istruzione a cui faceva riferimento la Branch, ignoro quanto fatto se nella fase di Write Back della Branch scopro che il salto non era da fare, ripartendo, così dalla fase di fetch dell'istruzione successiva.
Bisogna poi fare un'ulteriore considerazione: siccome si tratta di probabilità, inizializzerò i bit associati agli indirizzi soggetti a salto in base alla posizione dell'indirizzo di destinazione: se quest'ultimo è maggiore dell'indirizzo dell'istruzione di branch, significa che il salto è in avanti, quindi, si ripresenta con minore frequenza e, quindi, gli assegnerò al bit associato il valore 0.
Viceversa, se l'indirizzo di destinazione è minore dell'indirizzo sorgente, significa che, molto probabilmente, siamo in presenza di un ciclo e, quindi, sarà facile che si ripresenti un salto; al bit di tale indirizzo assegnerò valore 1.

II Problema: Data

L'istruzione 20 viene letta, la decodifico e scopro che devo fare l'and del registro Ro con 0, eseguo, cioè tramite l'ALU faccio l'and, prendo il risultato e lo scrivo in R0. Quindi R0 si azzera solo nella fase W20



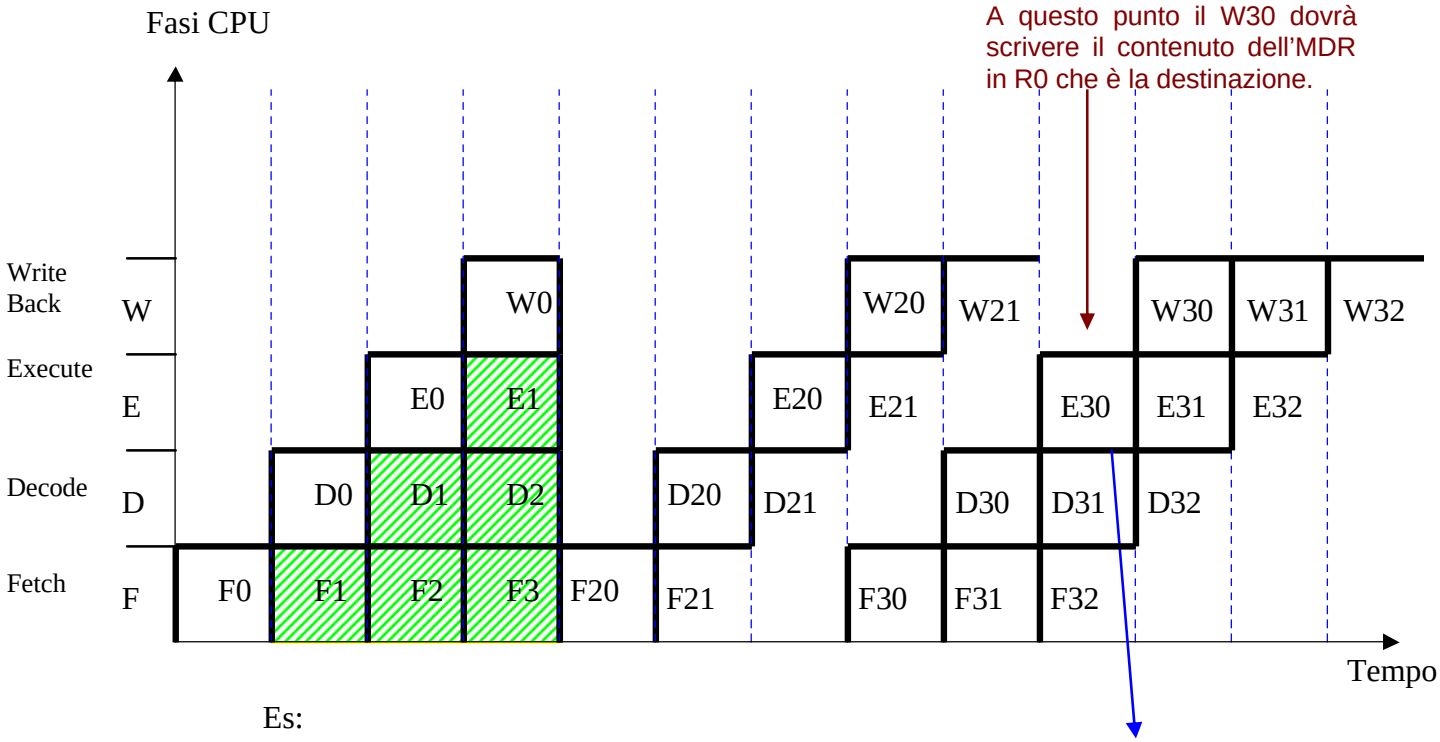
Es:
I20 : And R0,R0,0
I21 : Add R1,R1,R0

Istruzione 20

Leggo l'istruzione 21, la decodifico, e scopro che devo fare l'add fra R1 e R0. Il problema è che R0 non è ancora stato azzerato, quindi, o attendo che R0 sia azzerato, oppure faccio in modo che il risultato dell'E20 entri direttamente nell'E21, senza causare, così, ritardi.
A fare ciò è l'unità di controllo dell'esecuzione che rimanda indietro un dato per farlo usare direttamente all'istruzione successiva. Tale processo prende il nome di **Data Forwarding**.

III Problema: Resource

L'esecuzione dell'istruzione 30 deve prendere l'indirizzo che gli arriva dall'IR, metterlo nel MAR, dare il comando MemR, vedere dal bus dati arrivare i 16 bit da campionare nell'MDR.
A questo punto il W30 dovrà scrivere il contenuto dell'MDR in R0 che è la destinazione.



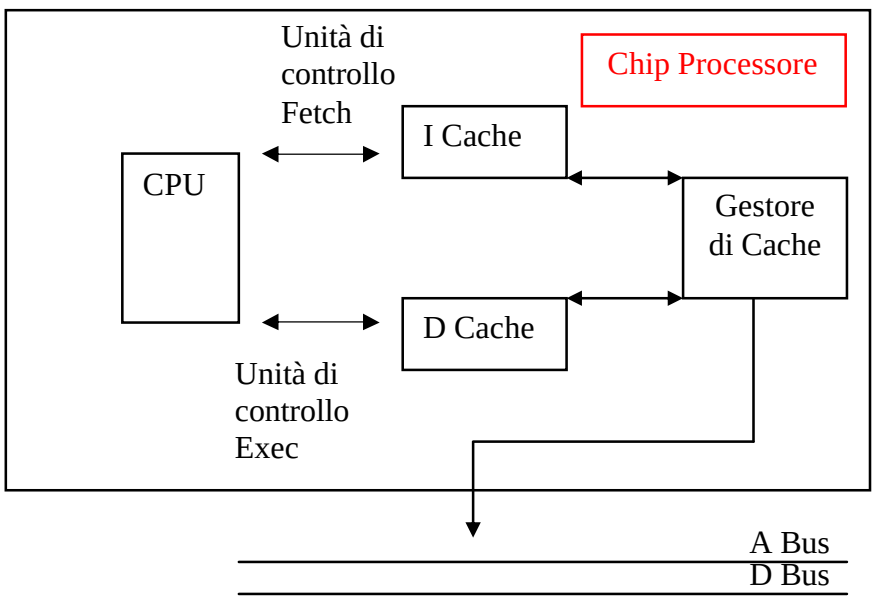
Es:
I30 : Ld R0,var

Istruzione 30

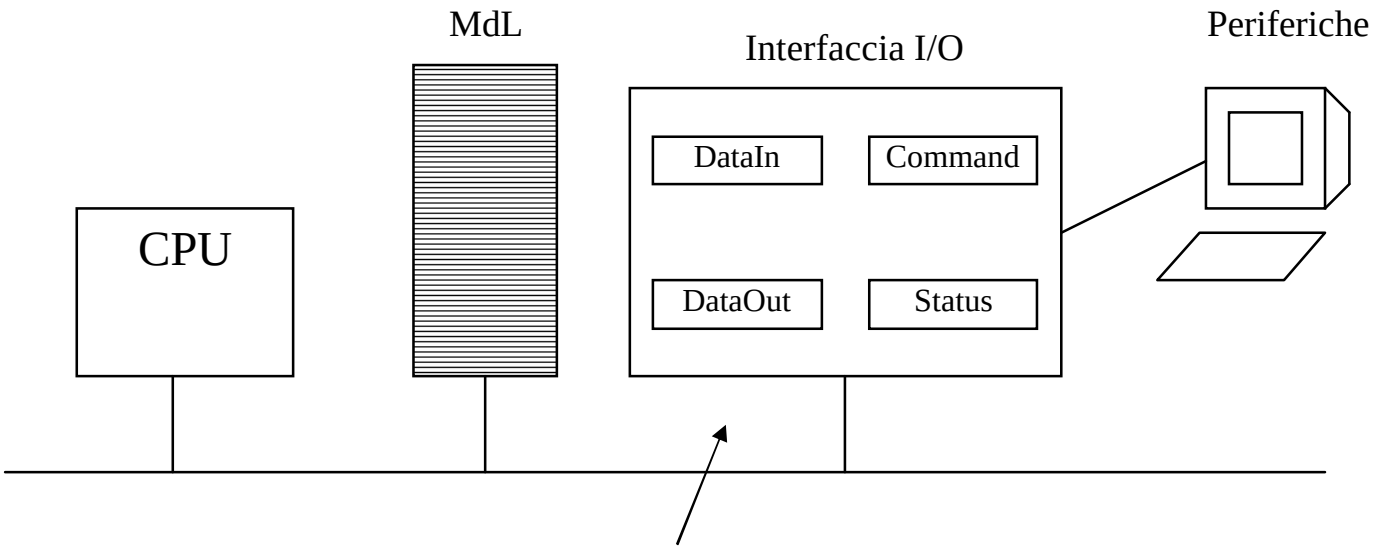
L'istruzione dice di prendere var, che è una stringa di 9 bit, attaccarla ai 7 più significativi del PC, considerare questo come un indirizzo di memoria, andare a leggere il contenuto della cella il cui indirizzo è quello appena costruito, portare tale contenuto in R0.

Vi è però un problema: siccome il W30 deve prendere il contenuto di MDR e metterlo in R0 e per farlo ha bisogno dell'unità di controllo dell'esecuzione che usa il MAR e l'MDR, entra in conflitto con l'F32 che ha bisogno del MAR e dell'MDR per andare a prendere il contenuto della cella 32. Siccome il bus è unico, passa prima uno e poi l'altro. Quindi dovrò rallentare tutto il processo per permettere a tutti e due di effettuare l'accesso. Quindi questo problema lo incontro ogni qualvolta vado incontro a Load o Store.
C'è da tenere presente però una cosa: mentre la Fetch va a cercare una cella nella zona della memoria contenente istruzioni, la Load o la Store in una zona contenente dati. Quindi non avranno mai la necessità di accedere alla stessa cella. Separando, quindi le memorie, facendo una memoria istruzioni ed una memoria variabili, risolvo il problema: l'unità di controllo del Fetch va a prendere le istruzioni nella memoria istruzioni e l'unità di controllo dell'esecuzione va a cercare i dati nella memoria variabili.
Per fare ciò, però, dovrei avere un bus per andare alla memoria istruzioni e uno per andare alla memoria variabili, inoltre dovrei gestire la dimensione delle partizioni della Ram in base al programma che sto usando e ciò è impossibile.

La soluzione è, quindi quella di inserire nello stesso chip del processore due memorie Cache, una per i dati, e una per le istruzioni. Così, vi è un unico modo di andare in MdL e l'unico che può andarci è il gestore della memoria Cache nel momento in cui deve cambiare qualche pagina.
In questo modo ho all'interno una macchina di Harvard, che lavora, quindi con due memorie separate, ma che all'esterno sembra una macchina di Von Neumann. Il gestore di cache quando ha bisogno di cambiare qualcosa va in MdL copia quello che serve e lo incolla nella cella della Cache interessata.



Interazione fra CPU e mondo esterno

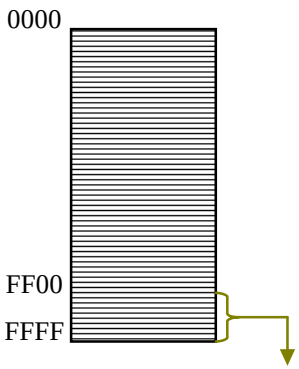


Celle di memoria particolari (Registri), con indirizzo dedicato.
Data Out: emette dei dati generati dalla CPU destinati alle periferiche.
Data In: Fa in modo che la periferica possa comunicare stringhe di bit che, poi, l'unità centrale possa utilizzare per eseguire delle istruzioni.
Status: Permette alla CPU di capire in che stato si trova la periferica (se è accesa, se è pronta,…).
Command: Comandi tramite i quali la CPU dice alla periferica cosa fare.

Due modalità di vedere le periferiche all'interno del mondo della CPU:

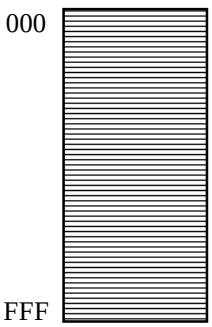
1 - I/O Memory Mapped
2 - I/O I/O Memory Mapped

1 - Spazio indirizzamento a memoria



Posso decidere nelle ultime 256 celle di memoria di dedicarle alle periferiche di ingresso e uscita. Quindi quando la CPU crede di accedere a memoria in queste celle, in realtà sta leggendo i registri dell'interfaccia I/O.

2 - Spazio indirizzamento I/O



E' una memoria totalmente dedicata ai registri I/O.
Questo implica la necessità di avere altri 2 fili I/O Read, I/O Write, oltre a MemR e MemW. Così, in base a quale filo viene attivato, andrò a leggere o scrivere nella memoria di lavoro o nella memoria I/O.

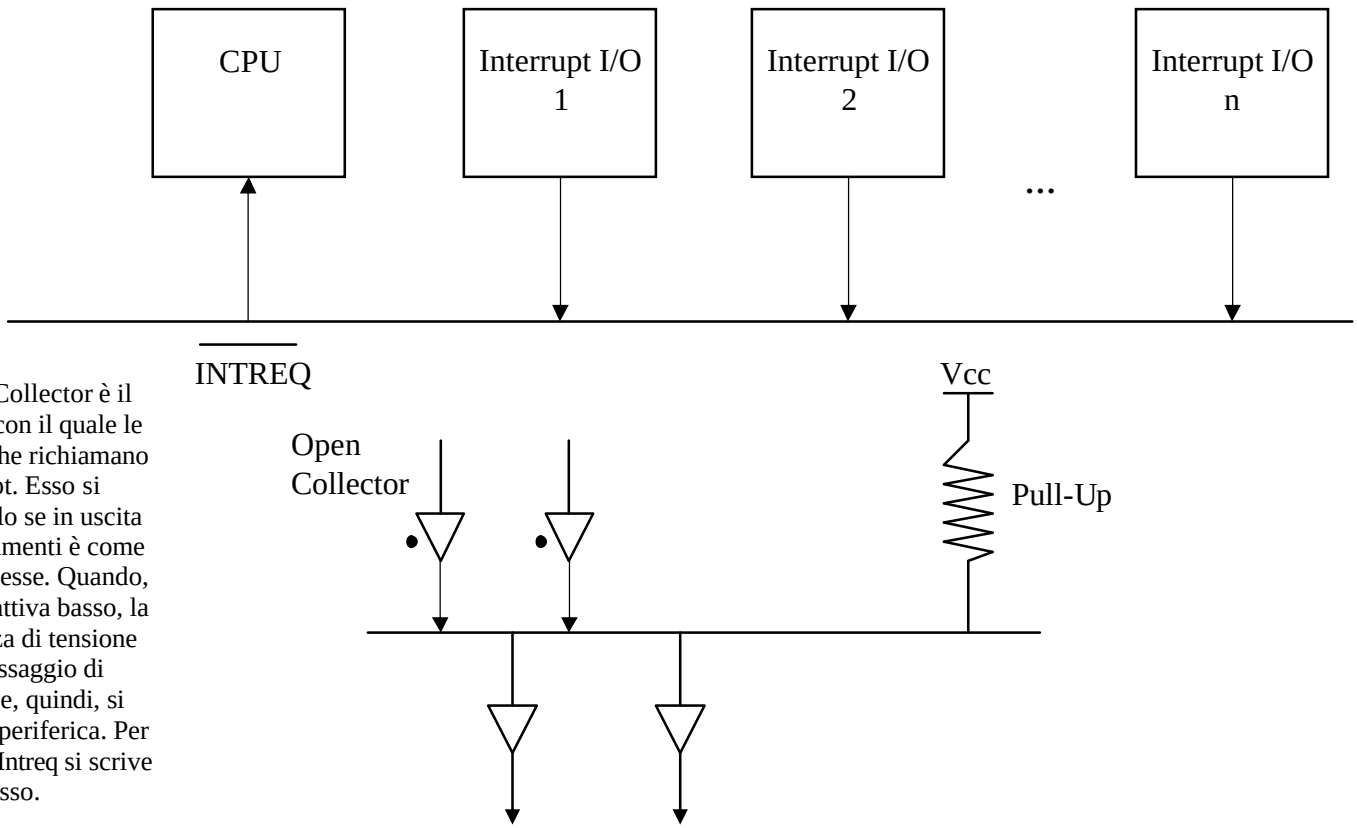
Due modalità per sincronizzare il mondo della CPU e delle periferiche:

1_Modalità di ingresso uscita a controllo di programma.
2_Modalità di ingresso uscita a interrupt.

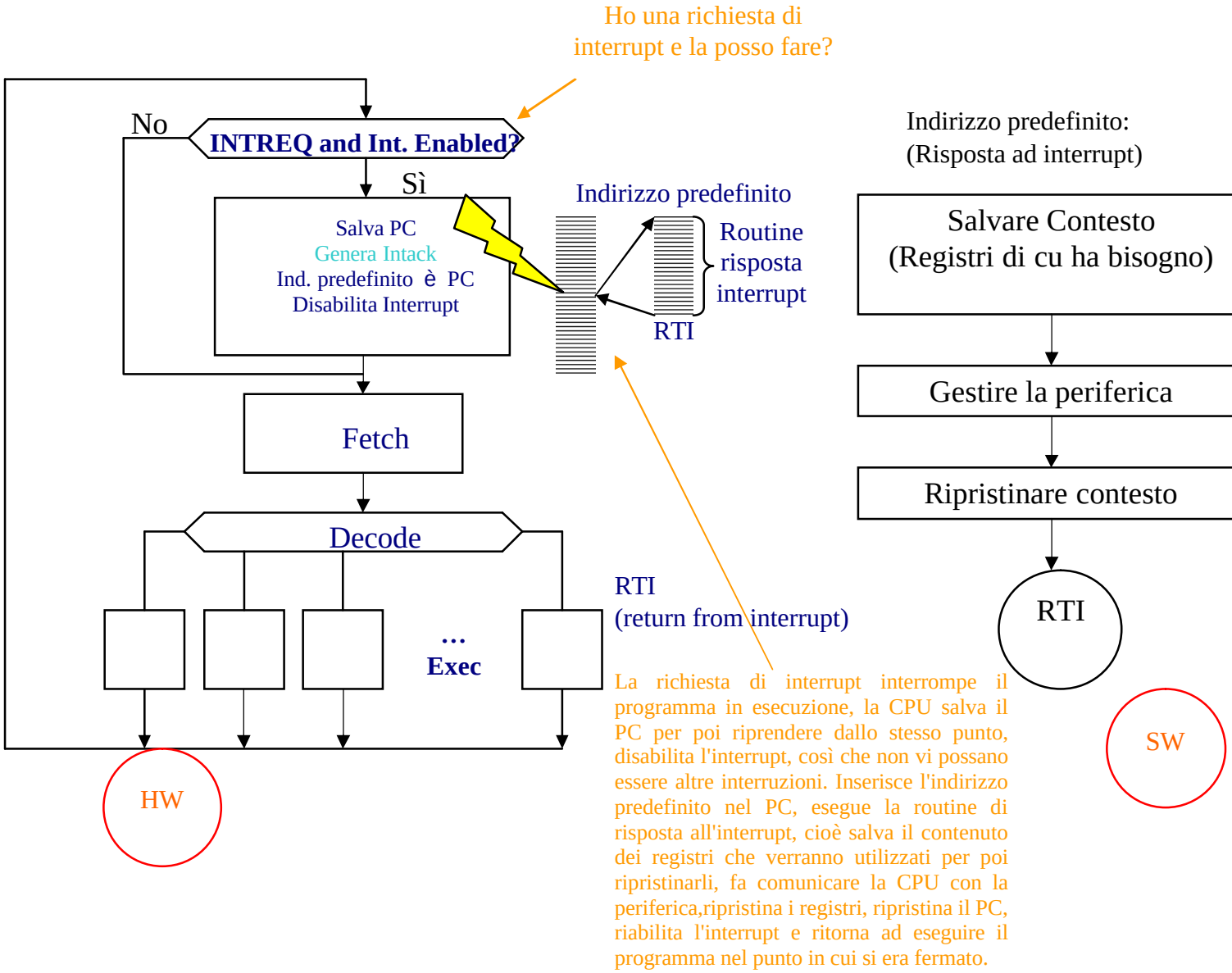
1_ E' il programma in esecuzione, che decide quando interrompersi per permettere il flusso di informazioni fra CPU e periferiche e poi continua da dove era arrivato (visione tolemaica del calcolatore: la CPU decide tutto e il resto le gira attorno, anche l'utente può agire solo quando la CPU lo permette). L'utente non si accorge molto di questo fatto, perché l'unità centrale può eseguire molte operazioni al secondo mentre l'uomo decisamente di meno. Il problema sorge quando la CPU esegue un programma particolarmente critico, nel quale deve fare tante cose e, quindi, si dimentica di chiedere all'utente cosa vuole fare. Oppure, può essere necessaria l'interruzione istantanea del lavoro della CPU per affrontare un problema urgente, che può essere, ad esempio, il superamento del limite critico di velocità su di un treno Trenitalia (morte sicura).

2_ Il mondo esterno, quando ne ha bisogno, interrompe il programma in esecuzione ed esegue il programma di cui lui ha bisogno. Per fare questo ho bisogno di una linea del bus controllo chiamata INTREQ (Interrupt Request) che inoltri la mia richiesta di Interrupt alla CPU. Inoltre ho bisogno di salvare per poi ripristinare il programma interrotto; tale funzione è svolta dalla routine di risposta.

Interrupt Cablato

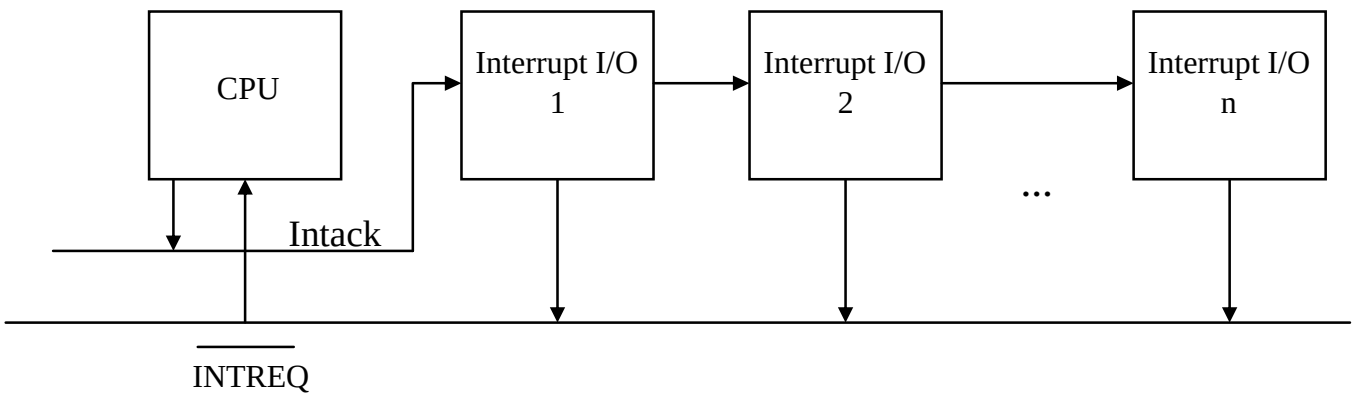


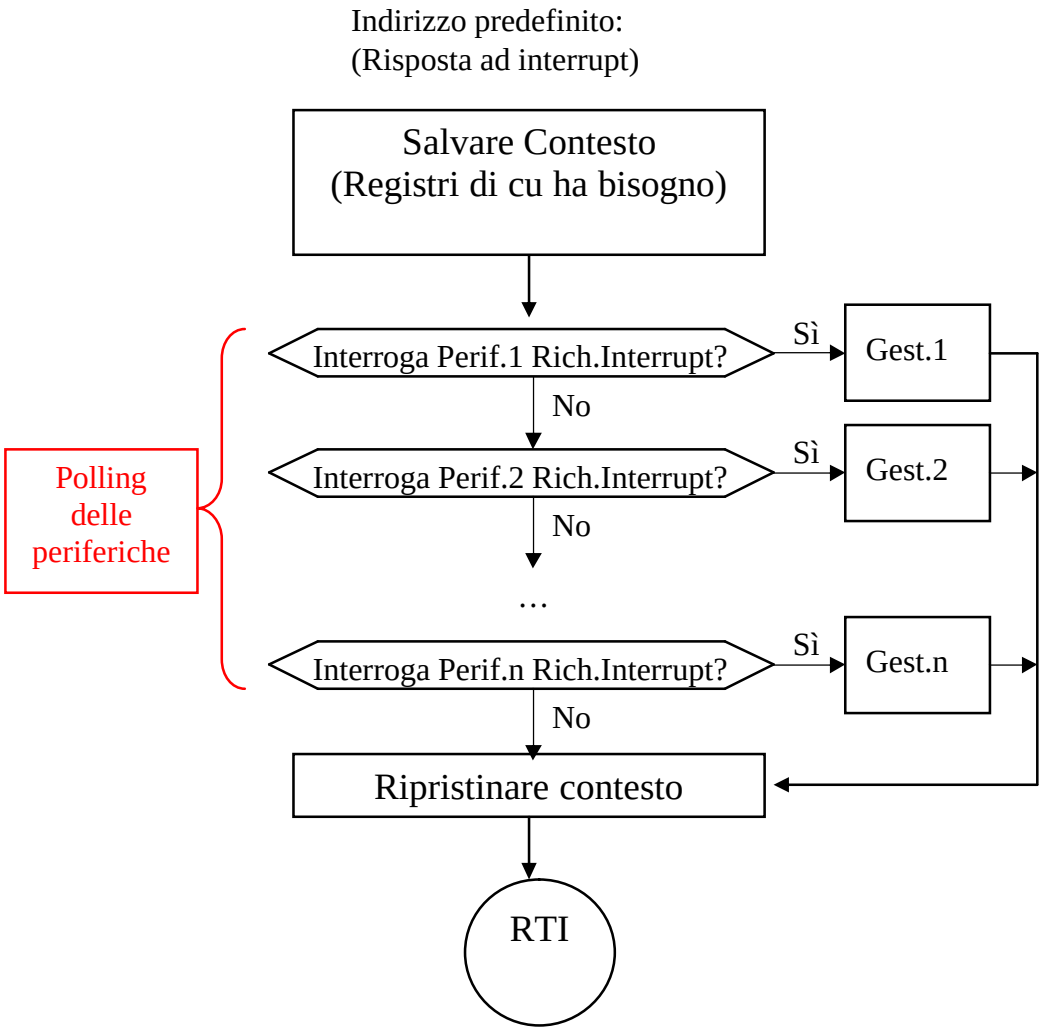
L'Open Collector è il sistema con il quale le periferiche richiamano l'interrupt. Esso si attiva solo se in uscita ha 0 altrimenti è come non esistesse. Quando, però, si attiva basso, la differenza di tensione causa passaggio di corrente e, quindi, si attiva la periferica. Per questo l'Intreq si scrive attivo basso.



Questa struttura ha però un difetto: permette di richiedere interrupt da parte di una sola periferica alla volta

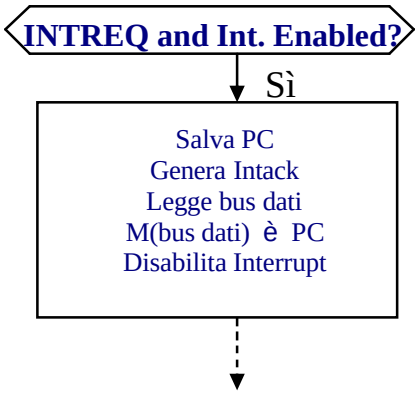
Occorre un messaggio di ritorno che faccia capire alla periferica che ha effettuato la richiesta che è stata riconosciuta e che è ascoltata. Per questo genera un segnale di risposta (Intack – Interrupt Acknowledged) che viene propagato fra le diverse periferiche. Arriva alla prima, essa si chiede se è stata lei a generare l'Interrupt. Se non è stata lei, propaga l'Intack alla periferica successiva. Se è stata questa, essa inizia a interagire con la CPU, interrompendo la propagazione dell'Intack. In questo modo, se l'Interrupt viene generato da più periferiche contemporaneamente, viene ascoltata solo la periferica più vicina alla CPU. La periferica che non è stata ascoltata perpetua la richiesta di interruzione fino a quando non viene ascoltata dalla CPU.
Tale organizzazione delle periferiche prende il nome di **Configurazione a collana di margherite.**





Però sorge un problema: noi ci siamo posti il problema dell'Interrupt, proprio perchè avevamo bisogno di avere risposte celeri. Con questo metodo, invece, ogni volta devo andare a cercare la periferica che ha inoltrato la richiesta, con una notevole perdita di tempo. Inoltre, le periferiche hanno importanza diversa a seconda della loro posizione, che non posso modificare. In più, se l'Intreq è abilitato, qualsiasi periferica può interrompere la CPU.

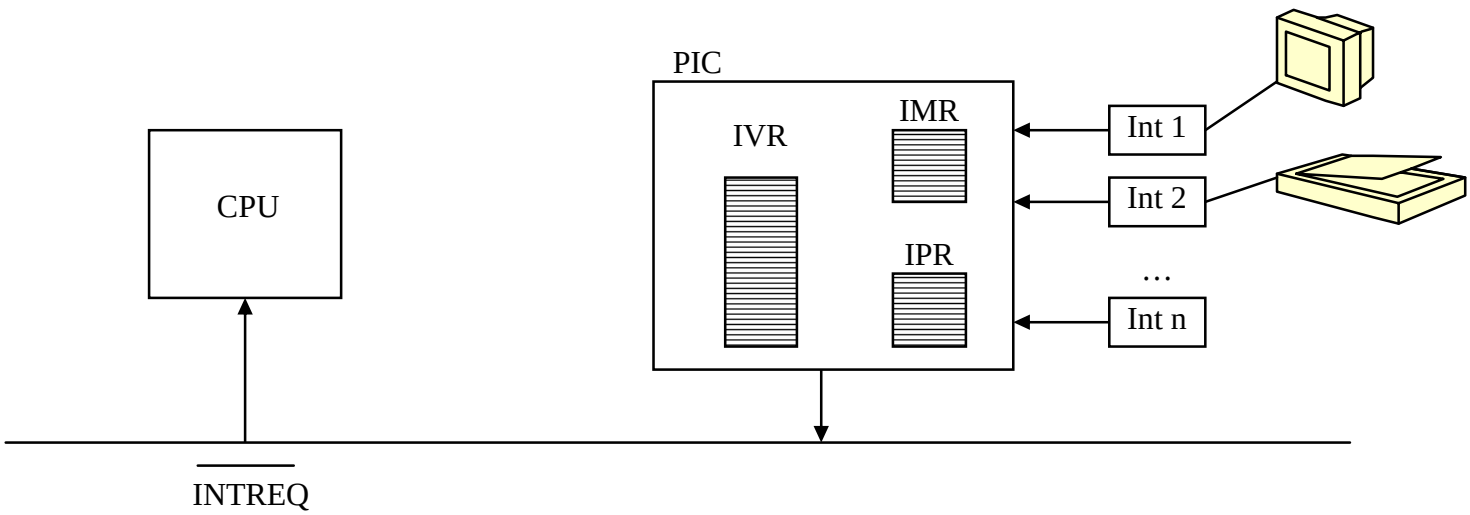
Interrupt Programmabile o Vettorizzato



L'unità centrale, quando riconosce un interrupt e si appresta a servirlo, genera l'intack, pretende che qualcuno metta sul bus dati un valore, legge la sequenza di zeri ed uni messi sul bus dati e li usa come indirizzo di una cella di memoria il cui contenuto viene messo nel PC.
Il meccanismo di chiamata si aspetta che qualcuno nella cella di memoria M(bus dati) metta l'indirizzo d'inizio di una routine e io leggendo il bus dati scopro qual è quella cella che contiene l'indirizzo d'inizio.
Questa complicazione è dovuta al fatto che ora cosa compare nel bus dati non è più deciso da chi ha programmato la CPU, ma da chi ci mette le periferiche.
Ogni periferica può dire un numero diverso sul bus dati e quindi dire all'unità centrale che ad esempio è la periferica 7 e quindi andare alla routine di risposta alla cella 7.
Quindi, nelle prime 256 celle della memoria posso tenere le celle di indirizzamento di ogni periferica. Tali celle prendono il nome di **Vettore di Interrupt,** vettore perchè è una struttura matriciale, monodimensionale, costituita da 256 elementi, che sono altrettanti puntatori a routine di risposta.
Quindi se voglio usare ad Interrupt una tastiera, devo fare in modo che l'interfaccia tastiera generi un Interrupt e che quando la CPU accetta l'interrupt, la tastiera metta sul bus dati il numero della cella in cui è contenuto l'indirizzo della cella da cui inizia la sua routine di risposta.

Problema: In questo modo ogni interfaccia periferica ha un prefissato indirizzo di routine di risposta

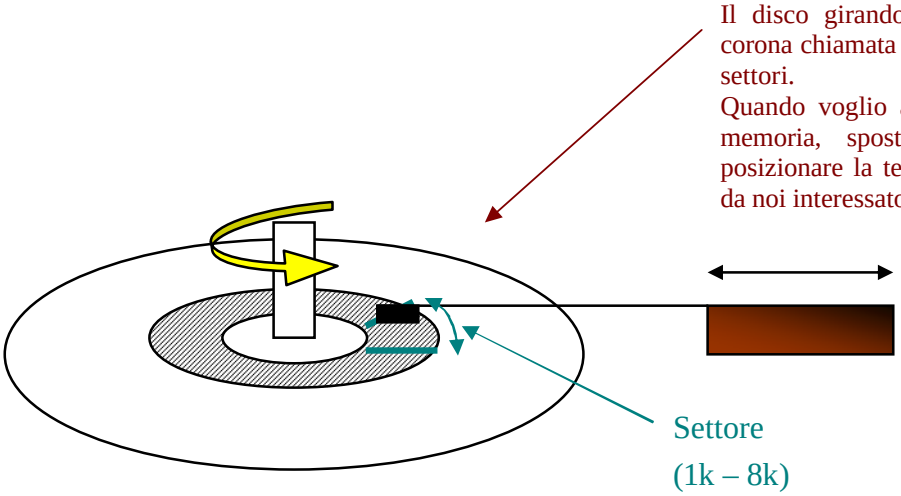
Per risolvere tale problema si ricorre al PIC (Programmable Interrupt Controller), che è collegato alle diverse interfacce e che contiene un numero di registri pari al numero delle periferiche (Interrupt Vector Register – IVR), registri nei quali andare a scrivere i numeri associati alle diverse periferiche.
Nella fase di accessione del computer (bootstrap) la CPU va a vedere che periferiche sono collegate e associa nel registro corrispondente il puntatore a routine di risposta di ogni periferica.
Se, ad esempio, la prima periferica interrompe la CPU, essa va a vedere nel PIC chi l'ha interrotta, scopre che è la prima, allora cerca nel primo registro l'indirizzo (da mettere nel bus dati) della cella in cui è contenuto l'indirizzo della cella da cui comincia la routine di risposta a quella periferica.
Per farmi maggiormente del male, posso anche mettere nel PIC l'IMR (Interrupt Mask Register), in cui dico quale di quelle periferiche mi può interrompere e quale no. Inoltre, posso anche mettere nel PIC l'IPR (Interrupt Priority Register), che è un modo per stabilire dinamicamente le priorità fra le varie periferiche.
Quindi se, ad esempio, la tastiera genera un Interrupt, la richiesta passa nel PIC che controlla se la tastiera è abilitata (IMR) e se non ce n'è un altro più prioritario (IPR). Se supera tali controlli, il PIC invia l'Intreq alla CPU. Quando la CPU risponde con l'Intack, il PIC va a vedere qual era la periferica interrompente, il codice associato e mette sul bus dati questo codice. La CPU prende il codice, va alla cella con tale indirizzo e interpreta il contenuto di questa cella come l'indirizzo della cella da cui partire a eseguire la routine di risposta all'Interrupt. Quando queste istruzioni finiscono, Return To Interrupt, la CPU recupera l'indirizzo che aveva salvato e lo mette nel PC, ritorna nella posizione in cui si era interrotta e procede con l'esecuzione del programma



DMA-DMAC

Hard Disk:

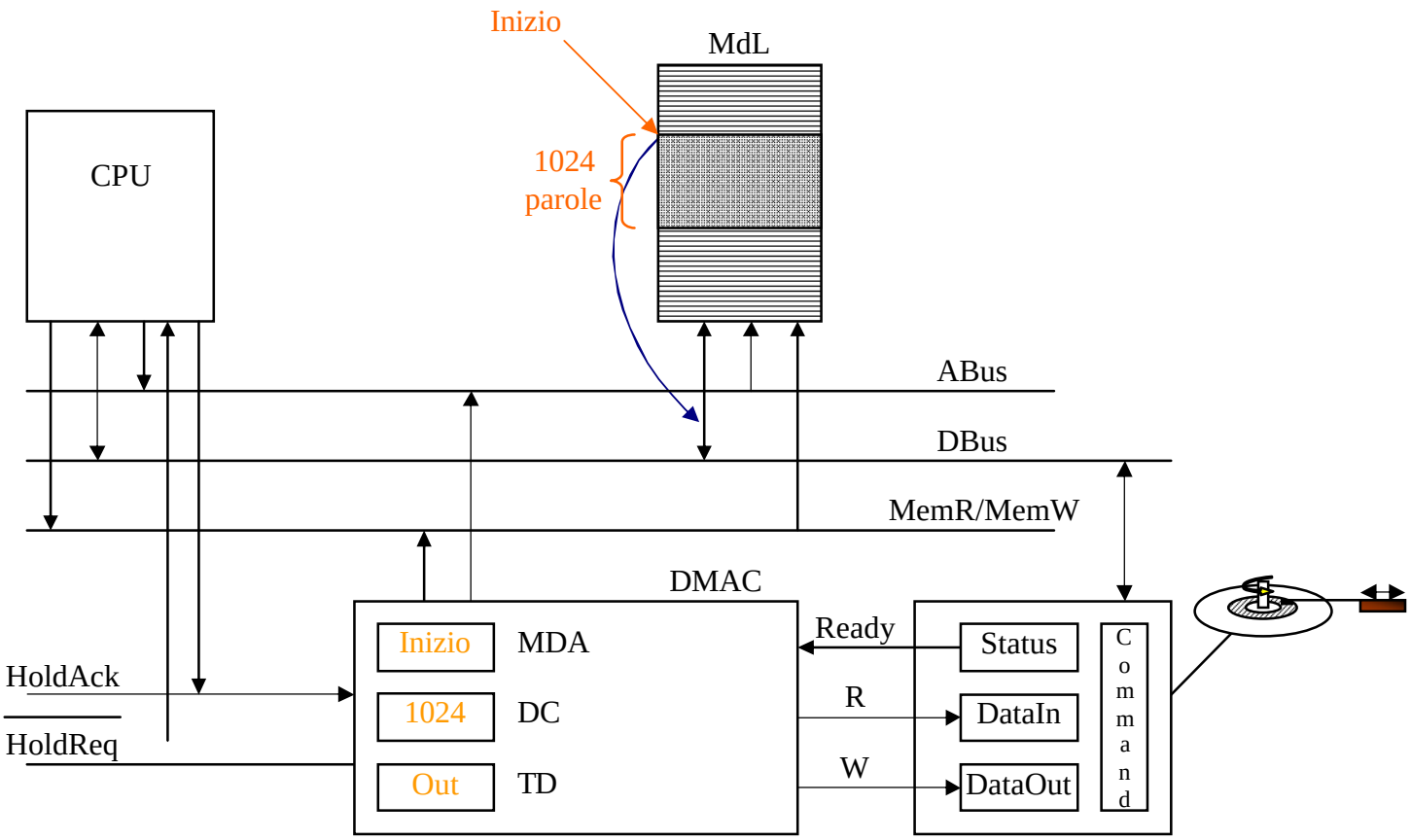
Testina montata sul braccio che la può far avvicinare o allontanare dal centro del disco.
Il disco girando passa sotto la testina delineando una corona chiamata Track del disco. Ogni track è suddivisa in settori.
Quando voglio andare a leggere una in una zona della memoria, sposto avanti o indietro il braccio per posizionare la testina sulla Track in cui si trova il settore da noi interessato.



Nell'interfaccia disco ci sarà un registro Command che permette di dire al braccio di spostarsi avanti e indietro, un registro status che mi dice se la testina sta arrivando sul settore richiesto, il Data In e Data Out, che permettono di trasferire 1 byte alla volta.

Se voglio scrivere un settore su disco la CPU deve dire all'interfaccia disco a quale traccia vuole arrivare, a quale settore è interessata, mettersi lì ed aspettare che il disco ci arrivi, quando mi trovo sul settore, devo cominciare a prendere 1 byte dalla memoria, scriverlo su disco, il byte dopo scriverlo su disco, per 1024 volte se il settore è da 1 Kb. Tutto questo fatto da programma (può essere sia a Controllo di Programma che a Interrupt) che prende dei Byte e li scrive dalla memoria di lavoro alla memoria di massa per salvarli su disco.
Per fare questo, la CPU deve fare il fetch di tutte le istruzioni del ciclo che scrive da MdL a Interfaccia Disco, fare tutti gli incrementi e decrementi dei contatori per sapere quanti Byte ancora le mancano, incrementare l'indirizzo di memoria per potersi muovere lungo la tabella da 1024 byte che dalla memoria devono andare su disco.
PROBLEMA: Il trasferimento di 1 byte dalla memoria a disco diventa uno di 15, 20 accessi a memoria per fare il fetch delle istruzioni, per tenere aggiornati i contatori, per incrementare gli indirizzi.

SOLUZIONE: accedo direttamente a memoria tramite il DMA



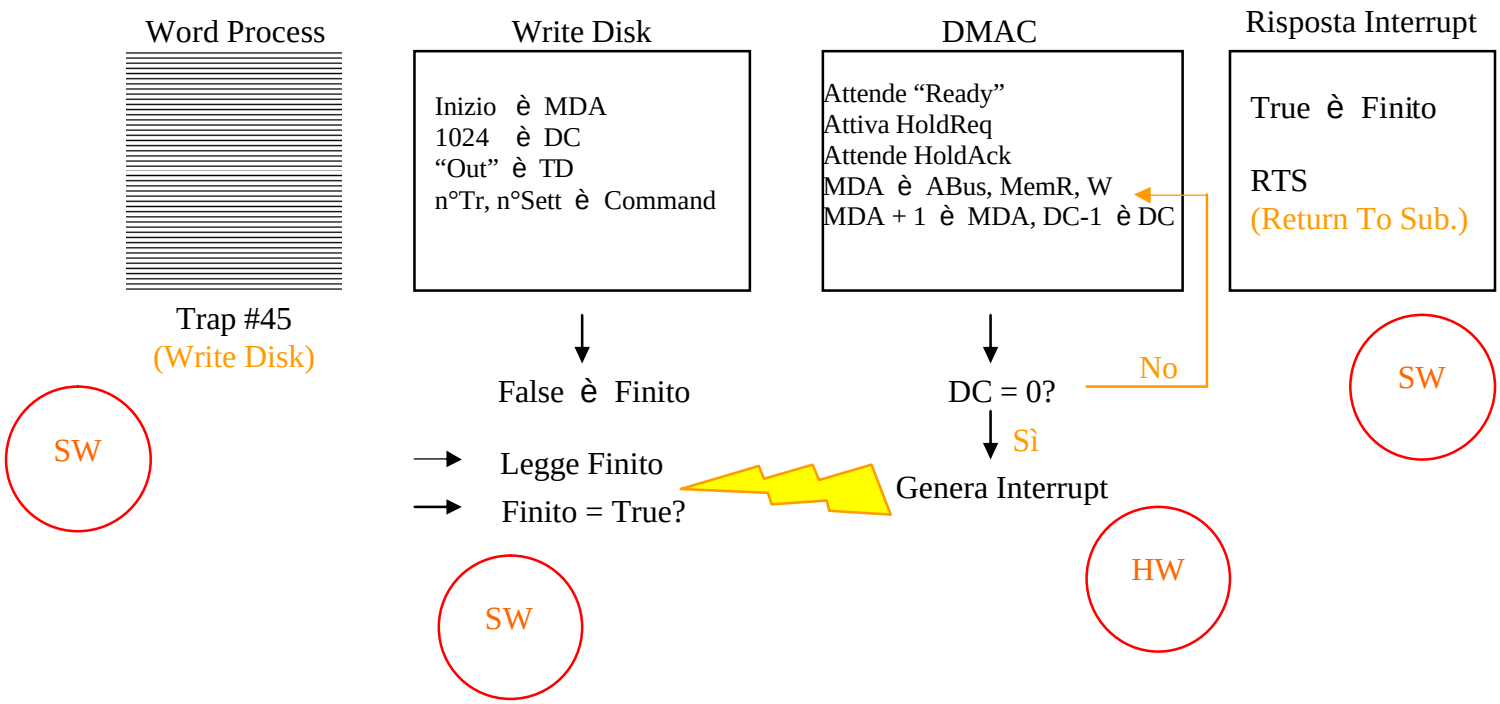
Quello che voglio fare è trasferire le 1024 parole che stanno in MdL che sono il risultato del lavoro svolto dal mio programma, per esempio Word, in un settore dell'hard disk che ho scoperto essere libero.
Per fare questo posso utilizzare il DMAC (controllore di accesso diretto a memoria, simile al PIC, cioè un componente programmabile che ha alcuni registri che la CPU può utilizzare per dire al DMAC cosa fare).

Word ha scritto nella memoria una tabella con tutti i byte che devono essere trasferiti. Per copiarli su disco richiama la routine di sistema operativo che serve a scrivere su disco, tramite il codice Trap #45 (Write Disk). Il Write Disk è un programma che la CPU deve eseguire per scrivere su disco. La CPU prepara la DMAC a effettuare cambiamenti sul disco: prende la cella d'inizio e la scrive in MDA (Memory Data Access), dice al DMAC che è da lì che deve trasferire, prende 1024 e lo scrive nel DC (Data Count) e poi prende il codice di operazione di uscita e lo mette in TD (direzione di trasferimento: in questo caso Out è trasferimento in uscita), viene impostata una variabile Finito su False ed avviare un ciclo che termina quando Finito diventa True. Il DMAC è pronto a partire. La CPU dice il n° di traccia e il n° di settore in cui scrivere nel registro command dell'interfaccia, finendo, così, l'inizializzazione. Scrivendo traccia e settore la testina si muove su disco, attende che il disco ruoti e, ad un certo punto si trova sulla posizione del settore. Tramite registro di stato dell'interfaccia, dice al DMAC che è pronto nella posizione richiesta. La CPU dice alla MdL di prepararsi alla lettura, attivando il MemR, legge da memoria l'indirizzo d'inizio e lo mette sul bus dati, il quale bus dati va nell'interfaccia e scrive l'indirizzo d'inizio nel registro Data Out, tramite il bus W del DMAC dico all'interfaccia di scrivere la cella su disco.

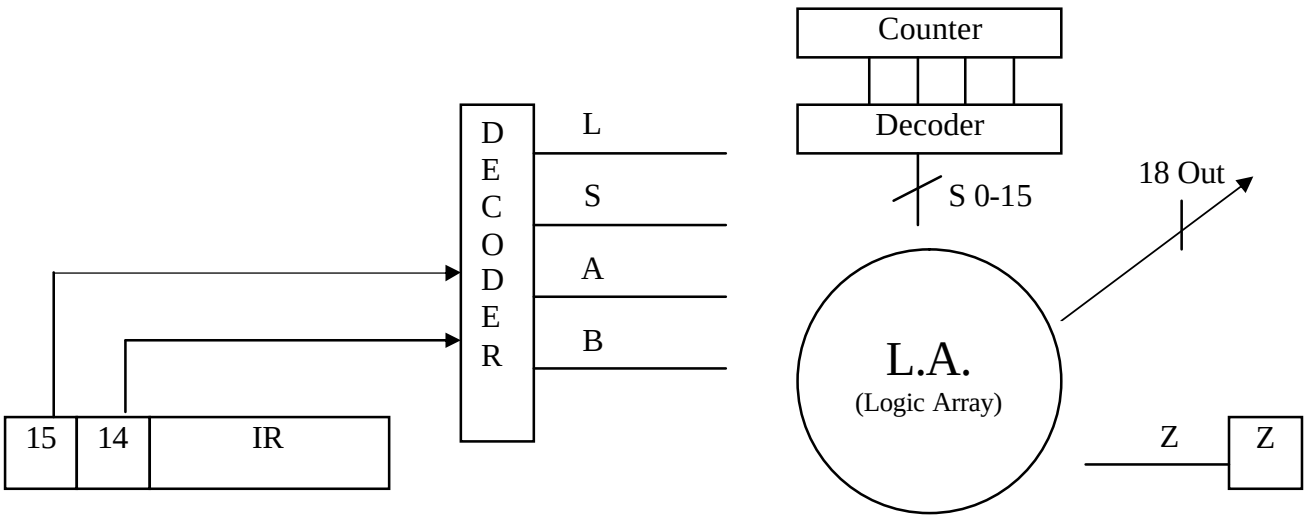
MA C'E' UN PROBLEMA!!! I bus sono della CPU

E' infatti la CPU che mette sul bus indirizzi gli indirizzi delle celle cui vuole accedere, sul bus di controllo i segnali di che direzione, ecc… In questo caso, invece, serve che la CPU ceda al DMAC i propri bus per permetterle di trasferire i byte da MdL a Interfaccia disco.

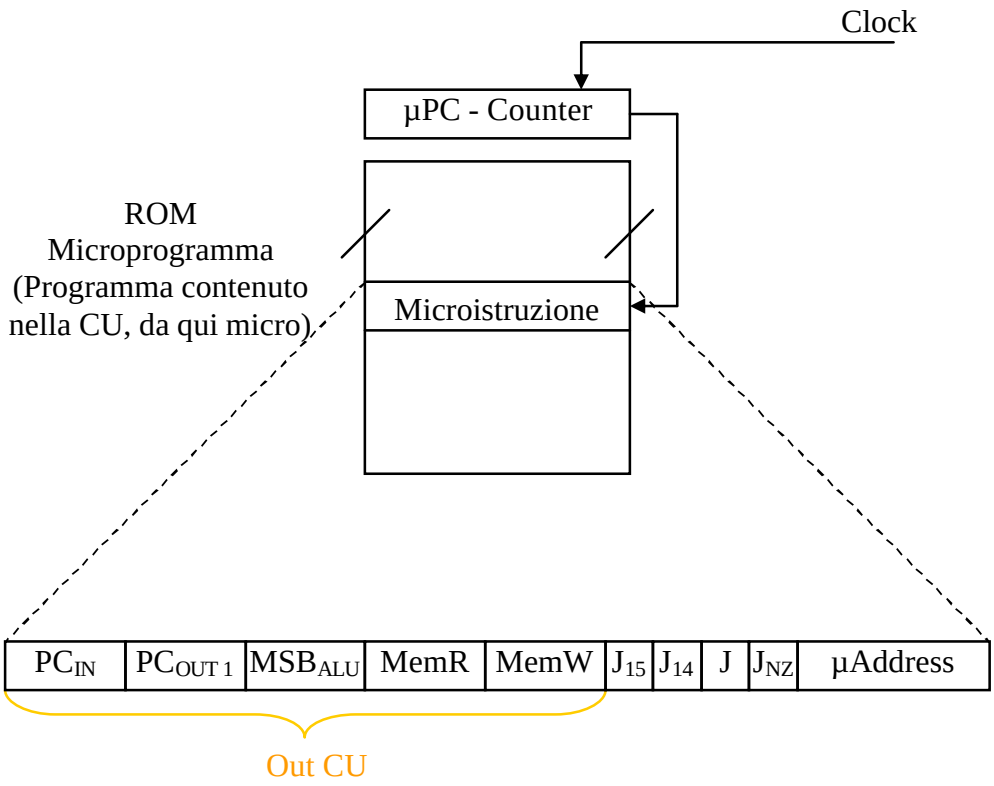
Per fare questo vengono utilizzati due fili: HoldReq (Hold Request, richiesta di sospensione) e HoldAck (Hold Acknowledged, richiesta capita, puoi usarli). Quindi, quando l'interfaccia comunica alla DMAC che l'Hard Disk è pronto (Ready), la DMAC invia alla CPU il segnale di richiesta di sospensione (HoldReq) e attende l' HoldAck. Ricevuto il via libera mette MDA sull' ABus, attiva MemR, attiva Write. L'indirizzo Inizio va sul ABus (quindi la memoria vede l'indirizzo d'inizio della tabella), attiva il MemR per dire alla MdL che vuole leggere (la MdL non sa che non è la CPU: vede la richiesta di leggere e l'indirizzo d'inizio), la MdL prende l'indirizzo della prima cella e lo mette sul bus dati, il bus dati è collegato all'interfaccia, da DMAC arriva il comando Write, i dati vengono scritti nel Data Out e il Data Out scrive il primo di questi 1024 dati su disco. A questo punto il DMAC incrementa di 1 l'MDA, decrementa di 1 il DC e, se Dc è diverso da 0, ricomincia il ciclo. Nel caso in cui DC fosse uguale a 0, invio una richiesta di Interrupt con conseguente esecuzione della routine di risposta ad Interrupt. Essa impone la variabile Finito a True e termina. La CPU ritorna a leggere il ciclo della variabile finito e, siccome ora è impostata su True, la CPU non è più costretta a lavorare nella DMAC e col comando Ret, le si dice di tornare a lavorare nel punto in cui si era fermata.



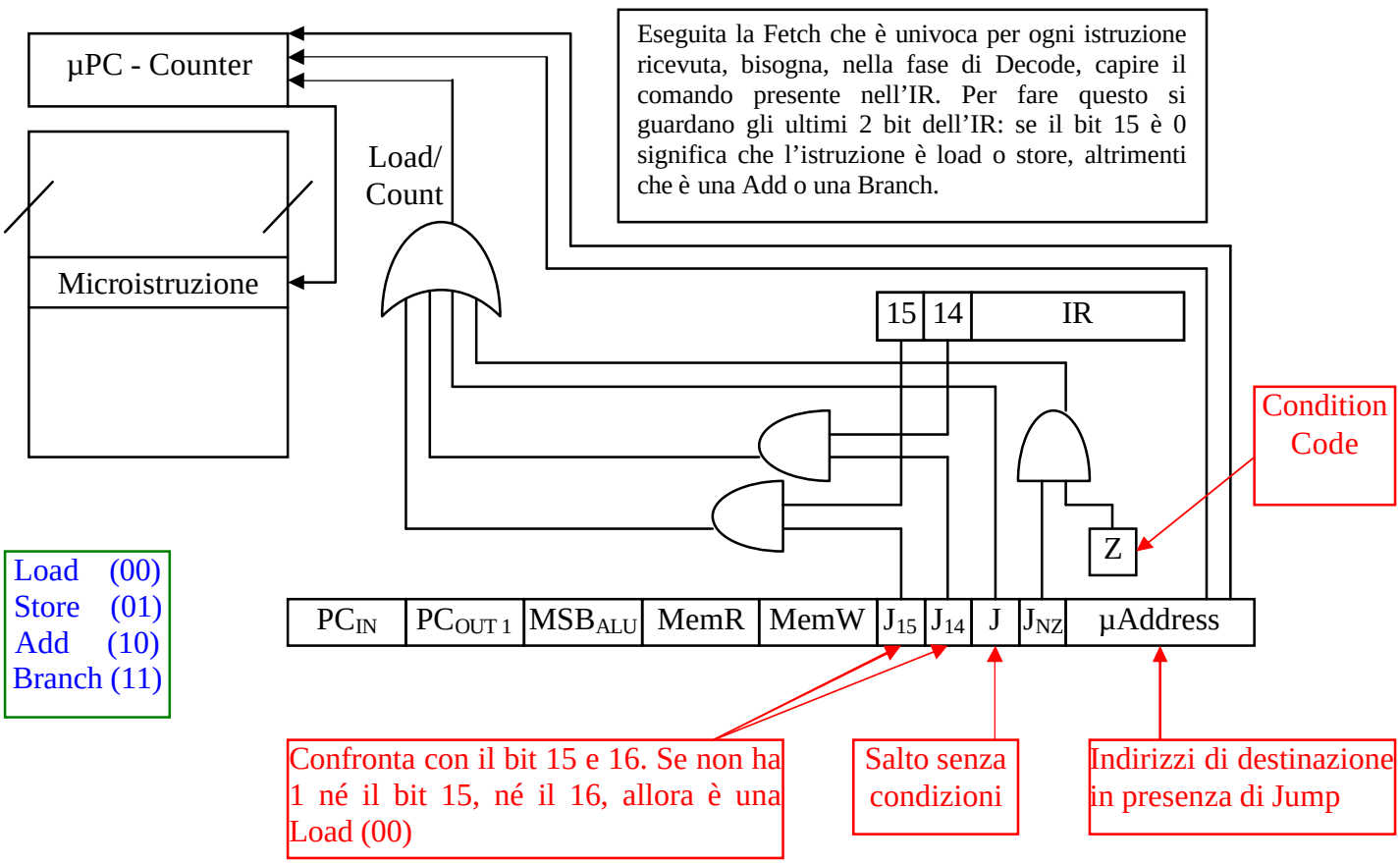
Unità di controllo cablata



Unità di controllo microprogrammata



Bit che vanno sul resto dell'unità di controllo o ai componenti esterni che l'unità di controllo deve gestire.



Schema del contenuto della µROM, dove ogni parola contiene i comandi presenti in ogni microistruzione. Esso rappresenta lo schema sequenziale delle fasi Fetch, Decode, Exec.

Confronta con il bit 15 e 16

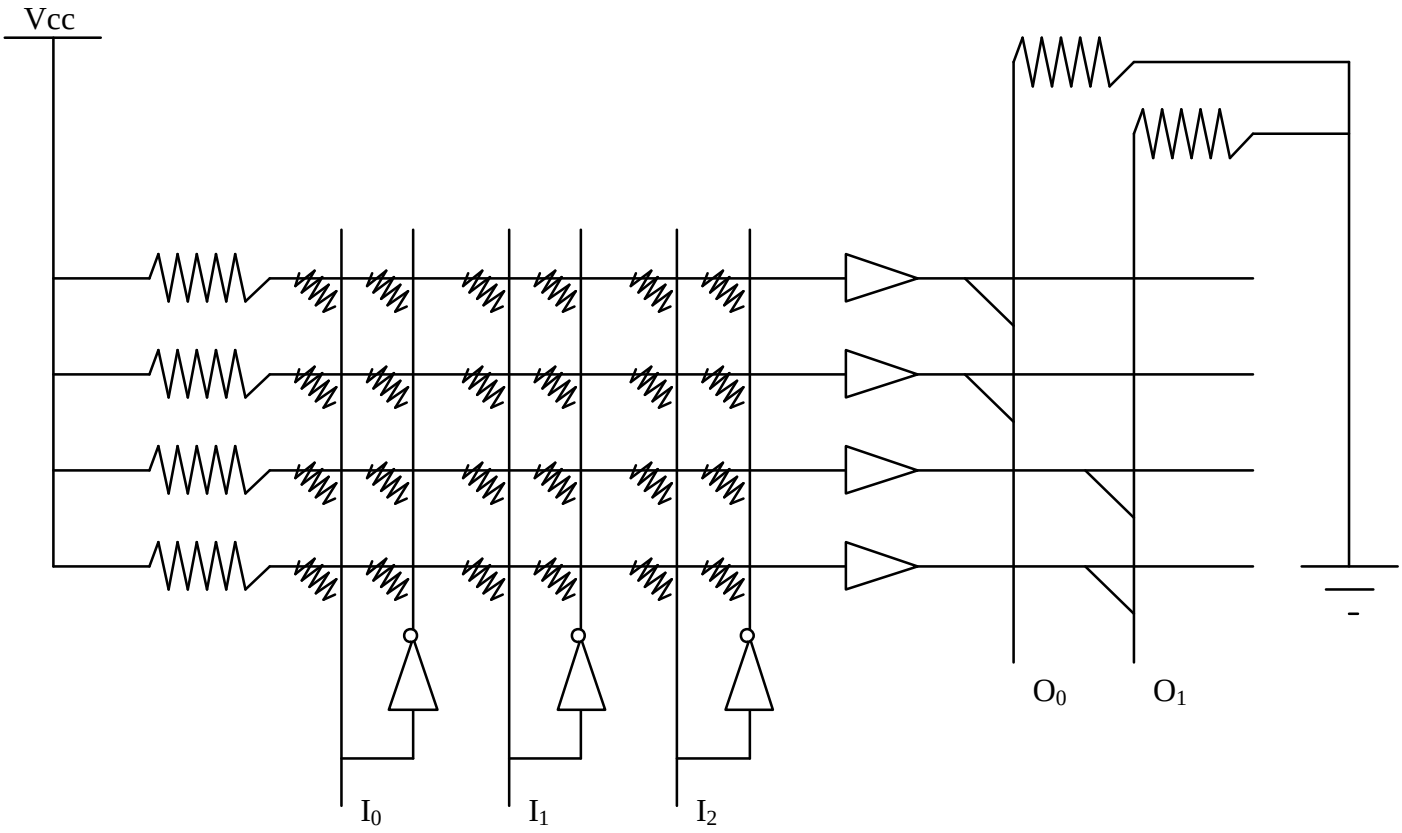
	mA	MAR		MDR				IR		PC		R0			Z	ALU		MEM		mA	J15	J14	JNZ	J	Address
		IN	OE	IN	OUT1	SAM	OE	IN	OUT1	IN	OUT1	IN	OUT1	OUT2	IN	MSB	LSB	R	W						
fetch	0	1									1					0	1								
	1		1															1							
	2		1			1												1							
	3									1	1					1	0								
	4				1			1								0	1								
	5																			17	1				
	6																			12		1			
load	7	1							1							0	1								
	8		1															1							
	9		1			1												1							
	10				1							1				0	1								
	11																			0				1	
store	12	1							1							0	1								
	13			1									1			0	1								
	14		1																1						
	15		1				1												1						
	16																			0				1	
	17																			24		1			
add	18	1									1					0	1								
	19		1															1							
	20		1			1												1							
	21									1	1					1	0								

Se non ha 1 né il bit 15, né il 16, allora è una Load (00)

Pregi: Uso un approccio software alla costruzione del cuore hardware della CPU che è l'unità di controllo. E' un pregio perché per quanto sia complicata l'attività della CPU, io posso esplicitare le fasi Fetch, Decode, Exec, facendomi aiutare da un compilatore che trasformi i miei comandi in codice binario.
Difetti: E' un po' più lento della CU Cablata.
Riassumendo, la CU Microprogrammata, rispetto alla CU Cablata, è più regolare, è più facile da costruire, anche se il set di istruzioni è più complesso, però è inevitabilmente più lento.

Appello di esame – 7 Novembre 2005

Mostrare la struttura interna di una PAL e discuterne le caratteristiche essenziali.



Il numero di linee prodotto è decisamente inferiore a 2^n, dove n è il numero di ingressi. Ciò è dovuto al fatto che, mentre nella ROM erano prese in considerazione tutte le possibili configurazioni degli ingressi, e ad esse veniva fatto corrispondere un particolare valore d'uscita, cioè venivano rappresentati dei mintermini, con le PAL e le PLA vengono rappresentati degli implicanti.
E' programmabile, nel senso che posso bruciare i fusibili, solo la sezione and, quella or, infatti è prestabilita dal produttore. La PAL rispetto alla PLA ha il vantaggio di essere più veloce, in quanto deve attraversare un solo livello di fusibili, però costringe a degli arrangiamenti nel caso abbiamo bisogno di un numero diverso di entrate o di uscite.
Nella sezione and viene effettuato il prodotto logico fra gli input collegati, mentre nella sezione or la somma logica.

Domanda a cui non si è data risposta: Registered Pal

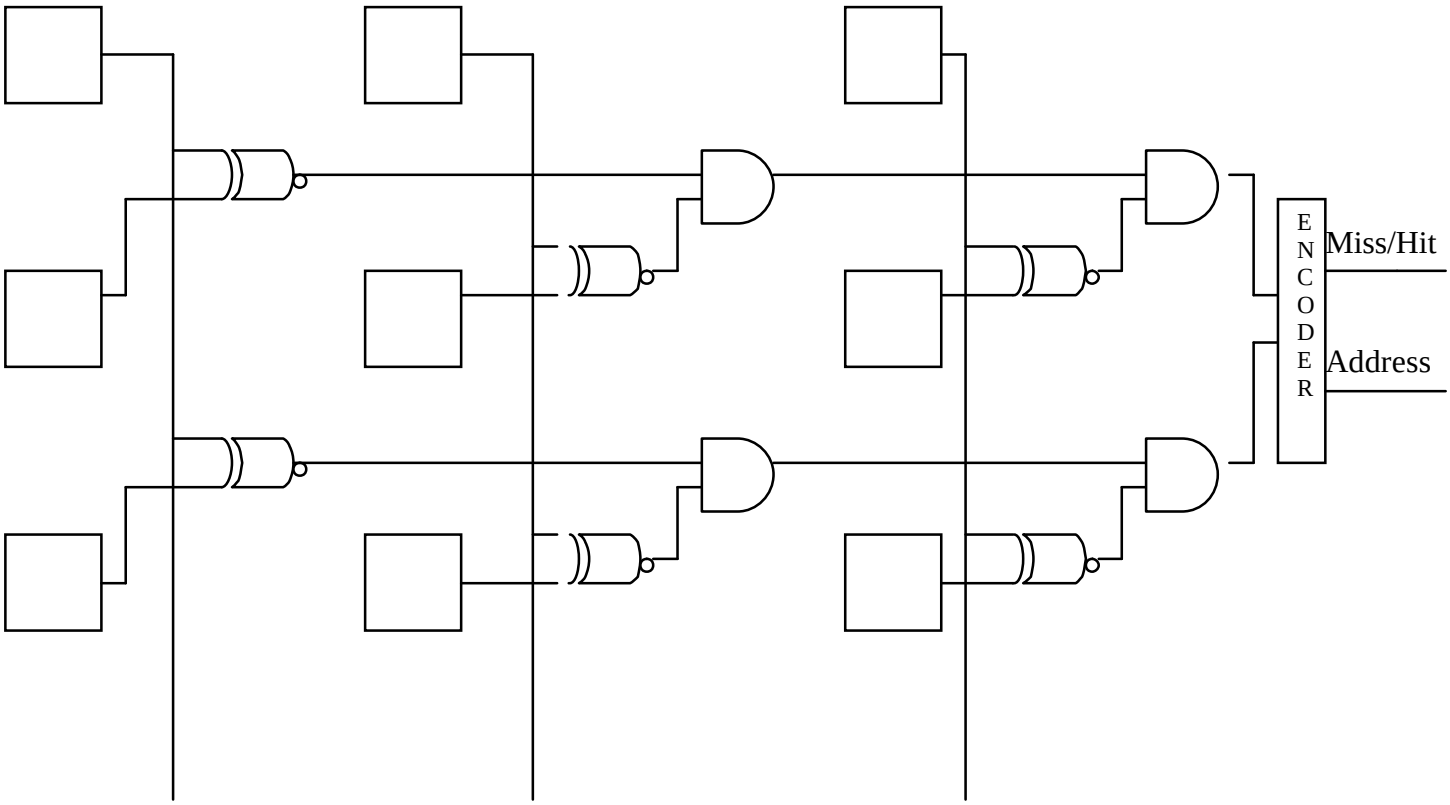
Sottoprogramma che riceve in R0 una stringa di 16 bit e restituisce, sempre in R0 la posizione dell'uno più significativo della stringa. Le posizioni sono numerate da 0, per il bit meno significativo, a 15, per il bit più significativo. Gli altri registri non devono risultare alterati.

???????????

Modi di indirizzamento dell'ISA della CPU LC2

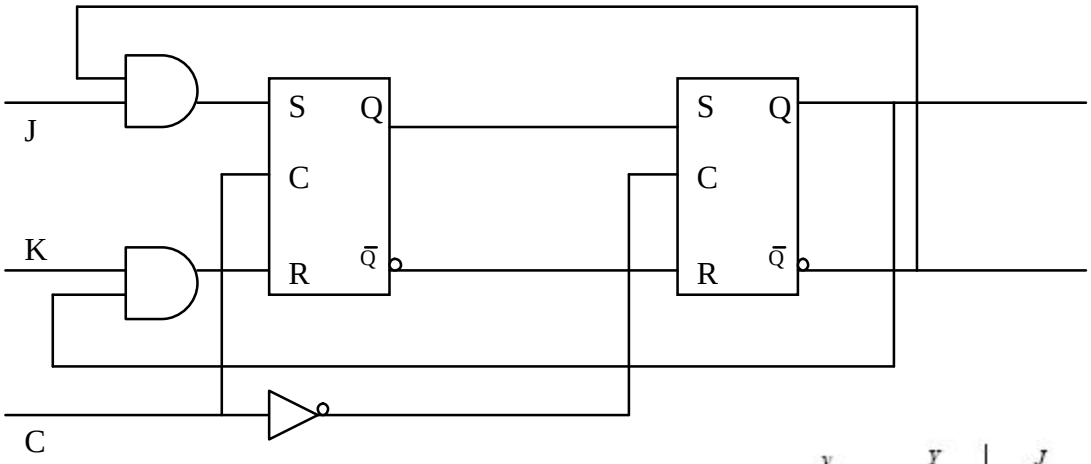
Immediato: L'indirizzo della cella viene già fornito all'interno dell'istruzione
Diretto: Nell'istruzione viene fornito l'indirizzo della cella in cui si trova il dato da noi cercato
Indiretto: Nell'istruzione viene fornita la cella in cui si trova l'indirizzo della cella in cui si trova il dato da noi cercato
A registro: Nell'istruzione compare il registro nel quale si trova l'indirizzo di una cella e il piazzamento da aggiungere a quest'ultimo per ottenere l'indirizzo della cella che contiene il dato da noi cercato.

Modalità di funzionamento e struttura della memoria associativa



Appello di esame – 14 Settembre 2005

Struttura interna di un bistabile J/K



y	Y	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

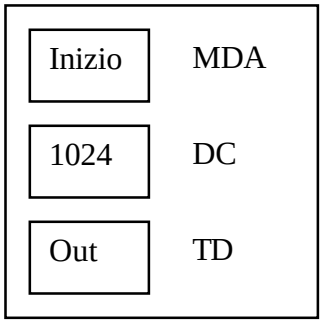
Tabella 5.11 — Bistabile J-K

Operazioni di ingresso/uscita mediante accesso diretto a memoria (DMA)

Nel caso in cui il trasferimento dei Byte è da CPU a Periferica, la CPU effettua la richiesta della routine di sistema operativo che serve a scrivere su disco. A questo punto la CPU inizializza la DMAC, inserendo nel MDA l'indirizzo della cella d'inizio, nel DC il numero di byte da trasferire, imposta Out nel TD e imposta una variabile finito a false; quindi inizia un ciclo in cui la DMAC continua a chiedersi se la variabile finito è uguale a True; finché ciò non accade il ciclo continua. Inoltre, la CPU scrive nel registro Command dell'interfaccia periferica il numero di traccia ed il settore in cui vuole andare a scrivere. L'interfaccia esegue il comando ricevuto dalla CPU e posiziona la testina nel settore indicato in command. Quando è arrivata al punto giusto, tramite registro di stato, invia un segnale Ready alla DMAC. La DMAC è pronta a partire, quindi invia una richiesta di HoldReq (sospensione) alla CPU, che risponde con un segnale HoldAck quando è pronta a cedere i propri bus. La DMAC trasmette l'indirizzo d'inizio salvato nel registro MDA tramite l'Address Bus alla MdL e invia il segnale MemR. Mentre la MdL risponde alla richiesta di lettura, trasferendo la cella indicata sul bus dati, la DMAC invia all'interfaccia il segnale Write, così che essa scrive nel proprio registro Data Out il dato che riceve sul Data Bus. A questo punto scrive su disco nel settore individuato il dato presente in Data Out. Il DMAC incrementa di 1 l'MDA e decrementa di 1 il DC; se DC è diverso da zero ripete il ciclo, altrimenti significa che ha finito di copiare e, quindi, genera un interrupt la cui routine di risposta impone la variabile finito su true, col conseguente ritorno al lavoro della CPU dal punto in cui si era fermata.
Se, invece, il trasferimento dati è nella direzione disco-CPU, il procedimento sarà uguale, con la differenza che TD sarà impostato su IN, la DMAC invierà alla MdL il segnale MemW e all'interfaccia il segnale Read, mentre il dato da copiare comparirà nel registro Data In dell'interfaccia.

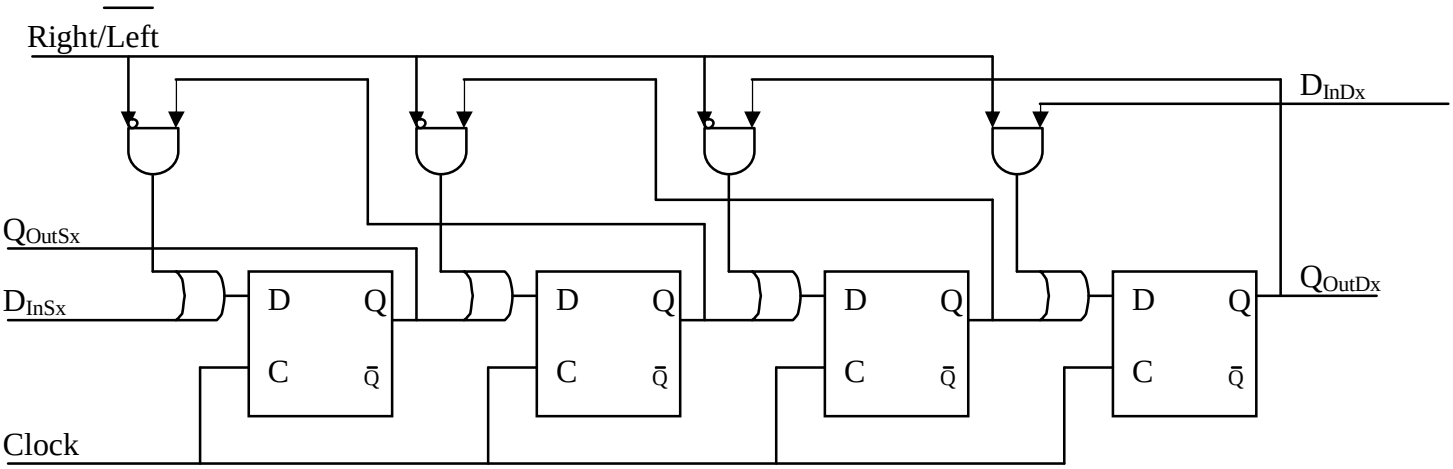
Appello di esame – 18 Aprile 2005

Struttura interna e funzionalità del DMAC



X le funzionalità guarda l'esercizio precedente

Struttura interna di un registro a scorrimento bidirezionale



Criteri di scelta fra memoria cache di tipo Tag Associative e Fully Associative

??????????

Appello di esame – 11 Luglio 2005

Principali cause di limitazione di funzionamento di una CPU pipeline.

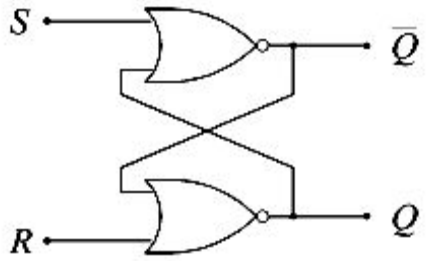
Una CPU pipeline può avere 3 tipi di problemi: problemi di controllo, di dato e di risorse.
Infatti, ho problemi di controllo, nel momento in cui l'istruzione è una Branch, in quanto, scopro solo nella fase Write Back se devo effettuare il salto e la cella di destinazione, quindi, le istruzioni successive, svolte nel frattempo, risultano inutili e vengono cancellate. Quindi, devo fare in modo di sapere subito dalla fetch se l'istruzione è una branch e, se è incondizionata, tramite il controllo de fetch, scrivere nel PC l'indirizzo della cella di destinazione, se è condizionata, tramite ricerca associativa, cercare l'istruzione branch interessata nella branch prediction table e, se il bit associato è uno, caricare l'indirizzo di destinazione nel PC, altrimenti proseguire con l'istruzione successiva.
Il secondo tipo di problema si pone quando un registro, modificato da un'istruzione, viene anche utilizzato nell'istruzione successiva. Infatti, se un registro viene modificato, tali modifiche vengono salvate solo nella fase di Write Back, mentre l'istruzione successiva necessita del registro già nella fase di Exec. La soluzione consiste nel trasmettere, tramite il controllo dell'Exec, il risultato dell'ALU direttamente nell'Exec dell'istruzione successiva. Tale processo prende il nome di Data Forwarding.
Il terzo problema si pone quando ho una Load o una Store; infatti, nella fase di Exec hanno bisogno di utilizzare i registri MDR e MAR, che nel frattempo, sono usati anche dalla Fetch dell'istruzione in quel momento letta. Il problema si risolve notando che, mentre la fase di Fetch legge una zona della memoria in cui sono presenti istruzioni, la fase di Exec legge una zona della memoria in cui sono presenti dati. Quindi, inserendo nella CPU una cache dedicata ai dati ed una cache dedicata alle istruzioni, il problema si risolve.

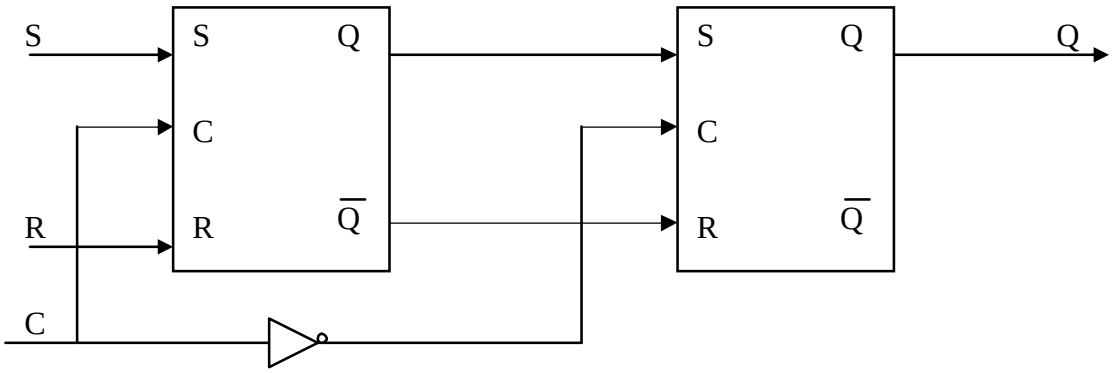
Appello di esame – 13 Giugno 2005

Mostrare la struttura interna dei Latch SR e flip-flop SR

y	Y	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Tabella 5.9 — Bistabile S-R





Comportamento della CPU in caso di Interrupt Vettorizzato e di Interrupt Cablato

In caso di interrupt cablato, la periferica invia il segnale di richiesta di interrupt alla CPU, la quale interrompe il programma in esecuzione, salvando il PC ed i registri in modo da ripristinarli, una volta concluso l'interrupt e disabilita l'Intreq, cosi che le altre periferiche non possano richiedere l'interrupt. Alla CPU arriva il segnale Intreq e risponde con un segnale Intrack per dire che è pronta a eseguire le istruzioni della periferica. Il segnale attraversa le periferiche, che sono collegate in serie; ogni periferica si domanda se è stata lei a generare l'Interrupt e, in caso, si avvia la routine di risposta all'interrupt di quella periferica. Se altre periferiche avevano effettuato la richiesta, manterranno la loro richiesta finché non riceveranno risposta. Quindi, a seconda della posizione fisica delle periferiche si determina una scala di priorità di risposta al cui vertice è posta la periferica più vicina alla CPU. Al termine riabilita l'Intreq, ripristina i registri e il PC e continua il suo lavoro dal punto in cui si era fermata.

In caso di Interrupt Vettorizzato, invece, quando la periferica invia il segnale Intreq, esso entra nel PIC, nel quale sono contenuti tre registri: IVR, IMR, IPR. Se la periferica non è mascherata, cioè ha la possibilità di richiedere l'Interrupt (controllo che avviene tramite l'IMR), e se nessun altra periferica ha priorità maggiore (controllato tramite l'IPR), allora il PIC invia l'intreq alla CPU la quale, dopo aver salvato il PC e i registri, per ripristinarli al termine, risponde con un segnale Intrack che entra nel PIC, il quale riconosce la periferica che ha richiesto l'Interrupt e pone sul DBus l'indirizzo della cella che fa riferimento a tale periferica. La CPU legge tale indirizzo e lo usa per andare a leggere in memoria la routine di risposta alla periferica interessata. Al termine dell'Interrupt, la CPU ripristina registri e PC e continua col suo lavoro nel punto in cui si era fermata.

Appello di esame – 21 Febbraio 2005

Organizzazione interna dell'unità di controllo microprogrammata

Guarda il disegno a pag. 20

Pal 12 linee prodotto, 4 ingressi, 3 uscite.
Rappresenta la funzione logica: $f(w,x,y,z) = \overline{w}\,x\,\overline{y} + x\,\overline{z} + y\,\overline{z} + w\,\overline{x}\,z + \overline{w}\;\overline{z}$