



Architetture e Reti Logiche

Seconda parte

Corso SSRI – Online
Insegnamento di Architetture e Reti Logiche del
prof. Nello Scarabottolo –

Architetture e reti logiche (2 parte)

Modulo 5 - Componenti elettronici integrati:	4
Transistore bipolare	4
PIN OUT	4
– Decoder:	5
– Multiplexer:	6
– Shift register:	7
– VARIANTI dello Shift register:	7
– Shift register a caricamento parallelo:	7
– Shift register a scorrimento bilaterale (Left/right):	8
– Counter:	8
– VARIANTI del Counter:	9
– Counter a caricamento parallelo:	9
– Counter bidirezionale (UP/DOWN-negato):	10
ROM (Read Only Memory):	11
PROM (Programmable Read Only Memory):	12
EPROM (Erasable Programmable Read Only Memory):	12
EEPROM (E ² PROM- Electrically erasable), EAROM	13
PLA (programmable logic array):	13
PAL (programmable array logic):	15
PAL a stadio di uscita programmabile	16
EPLD, FPGA:	17
Modulo 6 – Linguaggio macchina:	18
Schema della macchina di Von Neumann:	18
Funzionamento della CPU:	18
Struttura della memoria di lavoro:	19
Struttura dell'interfaccia I/O:	20
Struttura del BUS:	20
Struttura interna della CPU:	22
Struttura interna della CPU LC2:	23
Struttura del linguaggio Assembly:	25
Modulo 7 – Architettura del calcolatore:	27
Tipi di linee di bus	27
Memoria RAM	28
Memoria ROM	29
Realizzazione di un banco di memoria	30

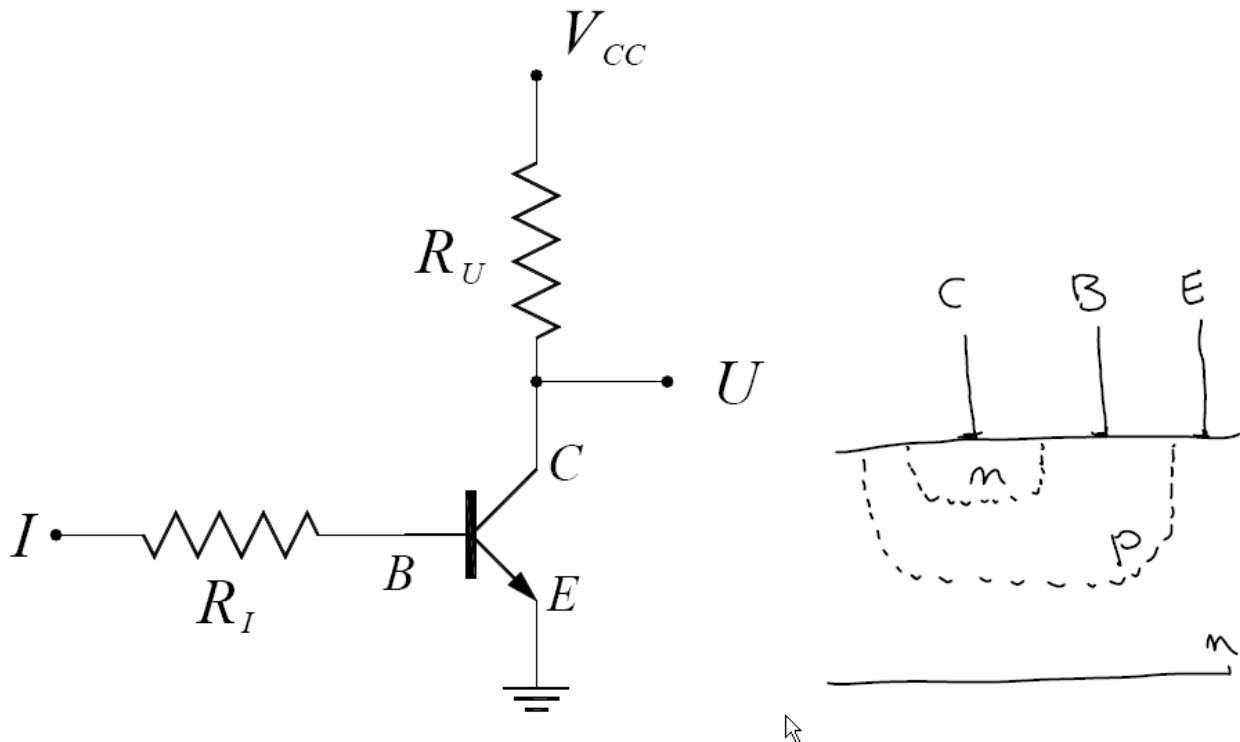
Esempio di banco di memoria per CPU LC-2 (address bus 16 linee e Data bus a 16 linee): ...	30
Collegamento CPU-interfacce di I/O	34
Sincronizzazione CPU-interfacce di I/O	34
1) controllo di programma (vince il clock):	34
2) interrupt (vince l'orologio):	35
Interrupt Cablato:	36
Interrupt Vettorizzato:	36
PIC (Programmable Interrupt Controller):	37
3) Direct Memory Access o DMA (i 2 riferimenti temporali restano indipendenti):	38
DMAC (DMA Controller)	38
Modulo 8 - Struttura della CPU:	40
La CPU NS-0 (didattica).....	40
Struttura interna CPU NS0:	40
Scopo degli Internal Bus: il Data Path	41
il Control Path	42
La Control Unit cablata (approccio hardware)	45
La Control Unit microprogrammata (approccio software)	46
Compiti dell'ALU.....	47
Struttura interna dell'ALU.....	48
Modulo 9 - Principali linee di evoluzione architetturale:	50
Funzionamento della memoria CACHE	50
Principio di località:	50
Memoria cache e politica Tag Associative.....	51
Politica Fully Associative	52
Accesso in scrittura alla memoria cache	53
Gerarchia di memoria.....	54
Disk cache	54
Memoria virtuale	55
Motivazioni del ricorso alle strutture pipeline	55
Funzionamento di una CPU normale.....	55
Funzionamento di una CPU pipeline	56
Struttura di una CPU pipeline	56
Control Dependency:	57
Data Dependency:	58
Resource Dependency:	59

Modulo 5 - Componenti elettronici integrati:

Transistore bipolare

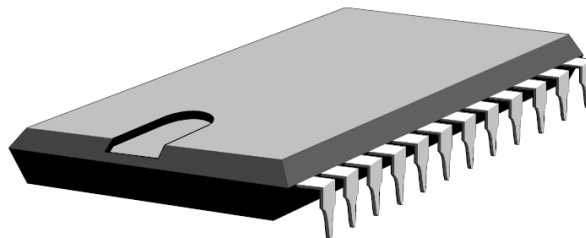
Tre cristalli di silicio a contatto, con caratteristiche elettriche diverse:

- zona n con eccesso di elettroni liberi (silicio "drogato" con antimonio, fosforo o arsenico);
- zona p con penuria di elettroni liberi (silicio "drogato" con boro, gallio o indio);
- zona n con eccesso di elettroni liberi (silicio "drogato" con antimonio, fosforo o arsenico).



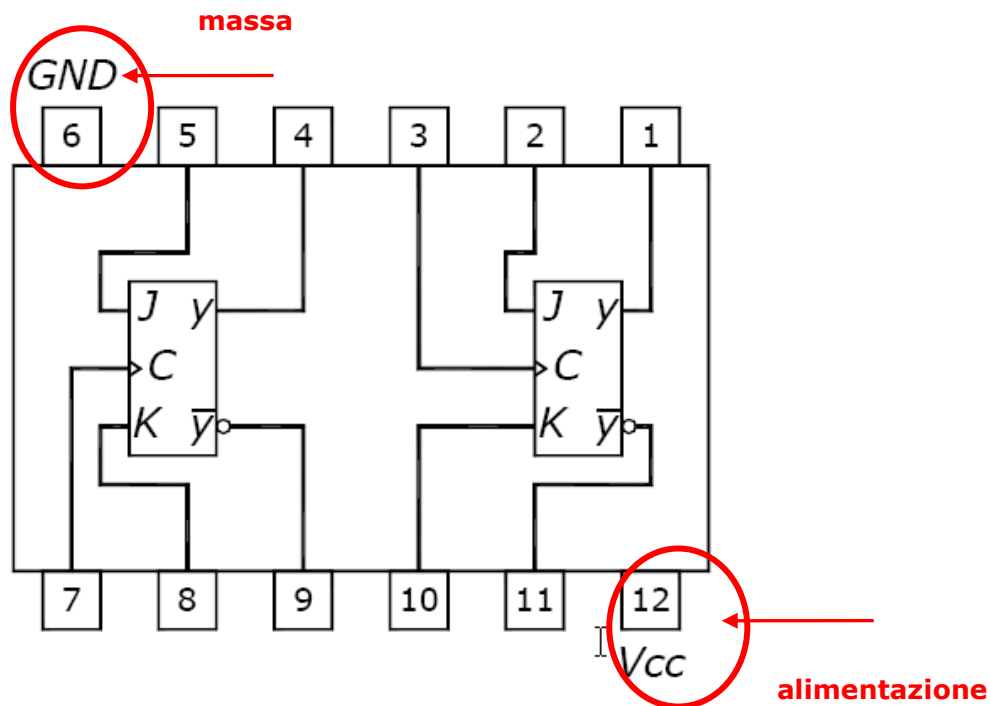
PIN OUT

Contatti elettrici esterni migliaia di volte più grandi dei dispositivi integrati (transistori). Per ottimizzare, serve collegare direttamente sul chip molti dispositivi integrati e ridurre i contatti esterni.



SSI : Small Scale Integration

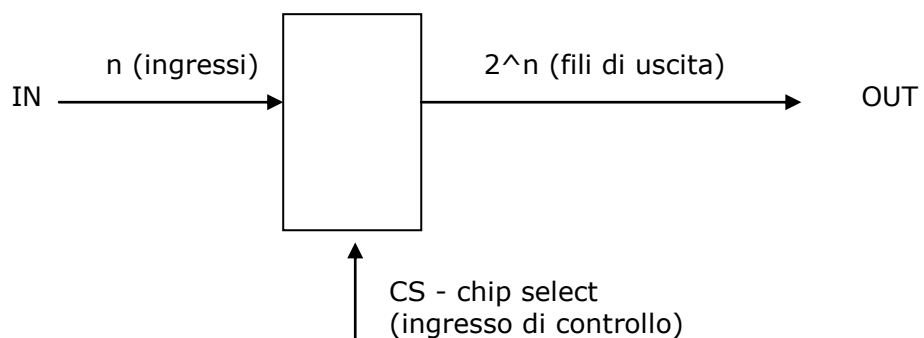
- alcune porte logiche in ogni chip;
- 10-20 pin per chip;
- ogni porta logica singolarmente accessibile;
- esempi di componenti combinatori:
 - 6 NOT;
 - 4 2-input NAND;
 - 1 8-input NOR.
- esempi di componenti sequenziali:
 - 2 JK flip-flop (figura seguente):



MSI: Medium Scale Integration

- alcune decine di porte logiche in ogni chip;
- 10-20 pin per chip;
- i componenti collegano fra loro le porte logiche per realizzare funzioni utili in molte situazioni;
- esempi di componenti combinatori:

– Decoder:

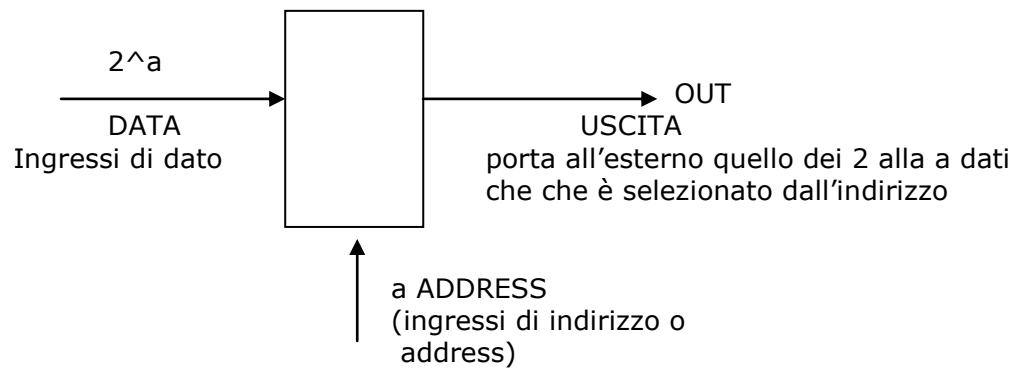


La struttura del decoder prevede **n** ingressi (solitamente 3 o 4) e **2ⁿ** uscite. La stringa di bit in ingresso seleziona qual'è l'uscita da attivare. Solitamente è presente il componente Chip Select che, se attivo il componente risponde al codice binario in ingresso se non è attivo il componente non abilita nessuna delle proprie uscite.

Il DECODER è praticamente un generatore di mintermini.

Una porta OR collegata alle uscite opportune può sintetizzare una qualsiasi rete combinatoria di n ingressi mediante la prima forma canonica (forma SP).

– Multiplexer:



La struttura del Multiplexer è quindi quella di un componente combinatorio con **a** ingressi di indirizzo o address, solitamente 2,3 o 4, **2 alla a** ingressi di dato (quindi 4, 8 o 16), **1** uscita; Il codice binario (stringa di bit) agli ingressi di indirizzo seleziona quale dato passare in uscita. Solitamente è presente il componente **Chip Select** che, se non attivo indipendentemente da cosa fanno gli ingressi tiene a zero l'uscita. L'uso tipico del multiplexer è quello di "multiplexare" 2 alla a ingressi in un unico filo di uscita in base a quello che selezionano gli indirizzi.

- **a** ingressi vengono collegati agli ingressi di indirizzo del multiplexer;
- la tabella delle verità della funzione viene divisa in **2a** sottotabelle;
- ad ogni ingresso del multiplexer, viene collegata una semplice rete combinatoria ($n-a$) che realizza una delle **2a** sottotabelle.

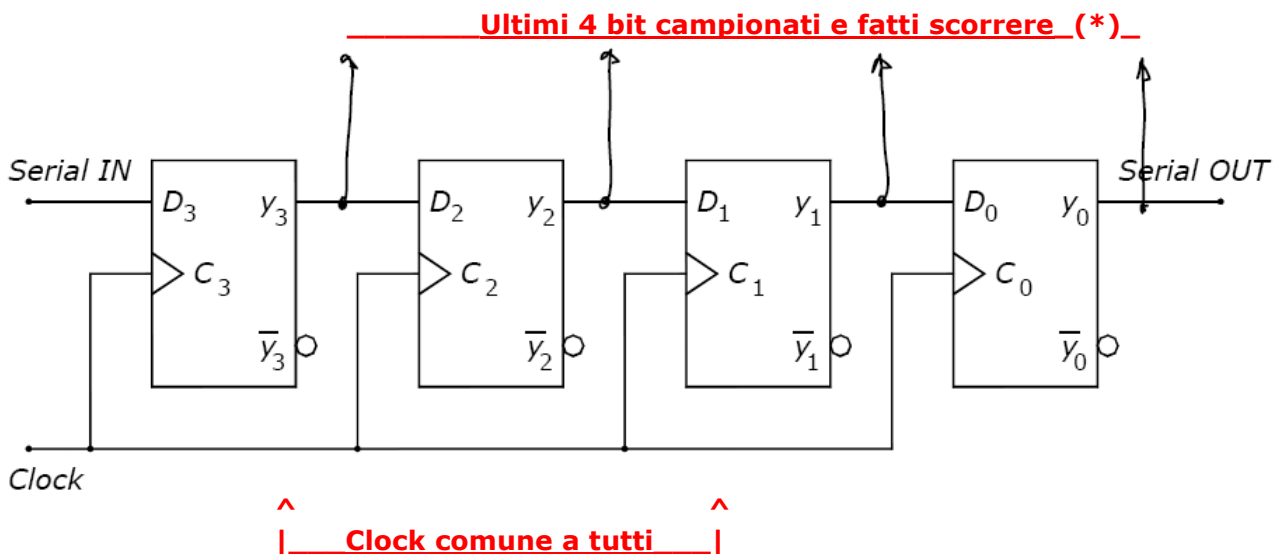
I multiplexer possono essere collegati a stadi in cascata, qualora non siano sufficienti singolarmente a soddisfare i requisiti in termini di numero di ingressi gestiti. Ad esempio, si può realizzare un multiplexer con 16 ingressi di dato e 4 di indirizzo partendo da 5 multiplexer a 4 di dato e 2 ingressi di indirizzo.

Gli ingressi di dato vengono collegati a 4 multiplexer, ai quali vengono collegati in parallelo due segnali di indirizzo. Le uscite di questi 4 multiplexer vengono poi collegate agli ingressi di dato del 5° multiplexer, i cui ingressi di indirizzo vengono collegati ai rimanenti 2 segnali di indirizzo.

Può essere usato nella sintesi combinatoria in quanto consente di partizionare una tabella delle verità in un certo numero di sottotabelle, di più semplice realizzazione.

- esempi di **componenti sequenziali**:

– Shift register:



E' un componente sequenziale formato appunto da una sequenza di bistabili di tipo D. L'uscita **diritta** di ogni bistabile è collegata all'ingresso D del bistabile successivo.

Ad ogni colpo di clock (il segnale di clock che abilita il campionamento è comune a tutti) i bit contenuti nei bistabili D avanzano di una posizione (scorrono) permettendo la conversione di dati che arrivano in SERIE in dati leggibili in PARALLELO fino a perdersi una volta raggiunto l'ultimo bistabile.

Osservando (campionando) in qualsiasi momento ciò che è presente alle uscite dei bistabili che compongono il registro a scorrimento la configurazione di 4 bit visualizzata (*) ci dà in parallelo (simultaneamente) gli ultimi 4 bit campionati e fatti scorrere nel registro a scorrimento.

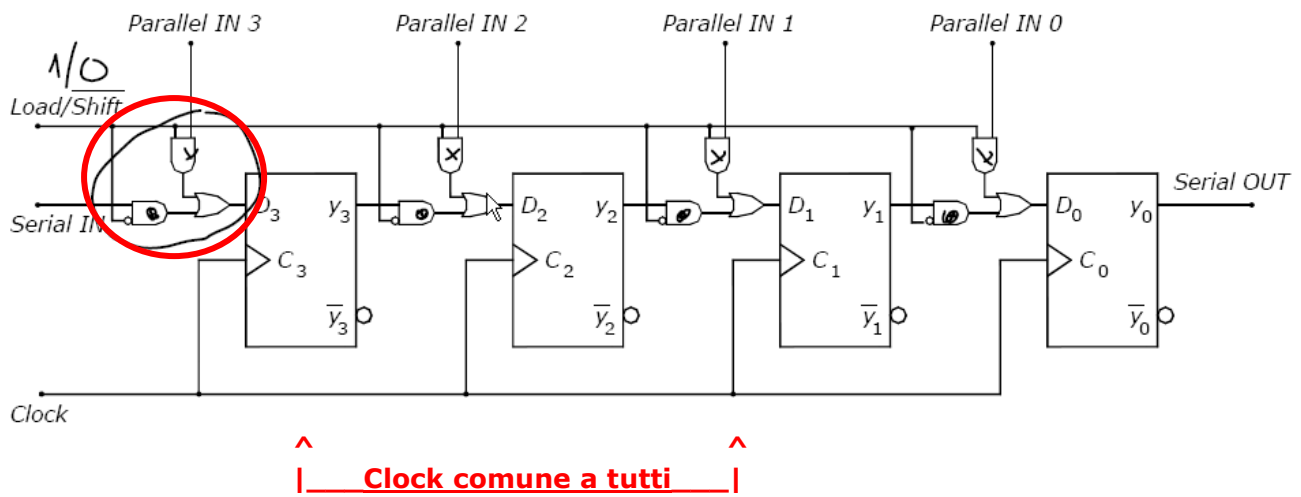
Necessita di bistabili Flip-Flop (non trasparenti) perchè se si usassero i bist. trasparenti il primo bit in ingresso del primo bistabile sarebbe visto immediatamente dall'ultimo bistabile. E' necessario un DISACCOPIAMENTO in modo che i bit possano avanzare in modo ordinato senza essere raggiunti e cancellati dai bit a monte.

– VARIANTI dello Shift register:

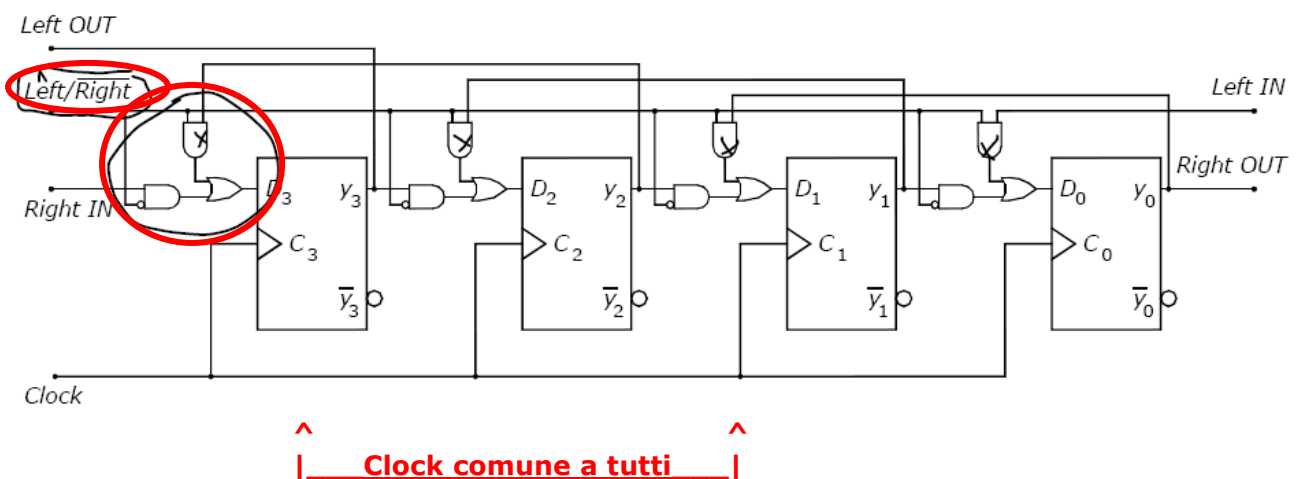
– **Shift register a caricamento parallelo:** (per caricare una stringa in parallelo per trasmetterla in modo seriale, un dato dopo verso l'altro lungo un cavo)

L'ingresso di dato che decide quale bit verrà campionato è una rete combinatoria che permette di decidere tra 2 percorsi, quello orizzontale NORMALE o un secondo VERTICALE viene dagli ingressi PARALLEL IN che possono essere caricati in maniera simultanea dall'esterno.

Chi decide quale percorso abilitare è il segnale LOAD/SHIFT negato che entra diritto nelle porte AND parallele e negato nelle porte AND del percorso seriale normale. Se LOAD/SHIFT vale 1 sono aperte le porte and superiori e quindi sono i dati PARALLELI ad entrare nei bistabili, se viceversa vale ZERO sono le AND orizzontali a funzionare facendo entrare i dati SERIALI



– Shift register a scorrimento bilaterale (Left/right): che mediante un segnale LEFT/RIGHT-negato decide in che direzione (verso destra o vs., Sinistra) far scorrere il contenuto in bit dei bistabili. E' un componente di un moltiplicatore o di un divisore. Come prima il contenuto abilitato può provenire da 2 strade, una "normale" (segnale right abilitato, scorrimento verso destra) mentre l'altra strada prevede che il bistabile prenda come ingresso l'uscita del bistabile a valle (bist. che lo precede). Ogni colpo di clock ci muove a destra o a sinistra a seconda dello stato 0/1 LEFT/RIGHT-negato dell'uscita di controllo.

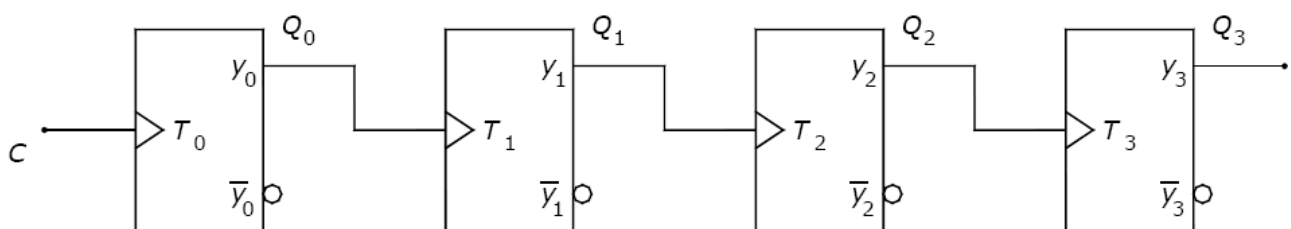


Lo shift register (registro a scorrimento) è un dispositivo sequenziale in grado di effettuare conversioni serie-parallelo-serie.

Può essere usato nella sintesi sequenziale per catturare gli ultimi n bit di una sequenza di ingresso e valutarne il valore.

In una sintesi sequenziale il registro a scorrimento è una finestra sugli ultimi n -bit di una sequenza di ingresso.

– Counter:

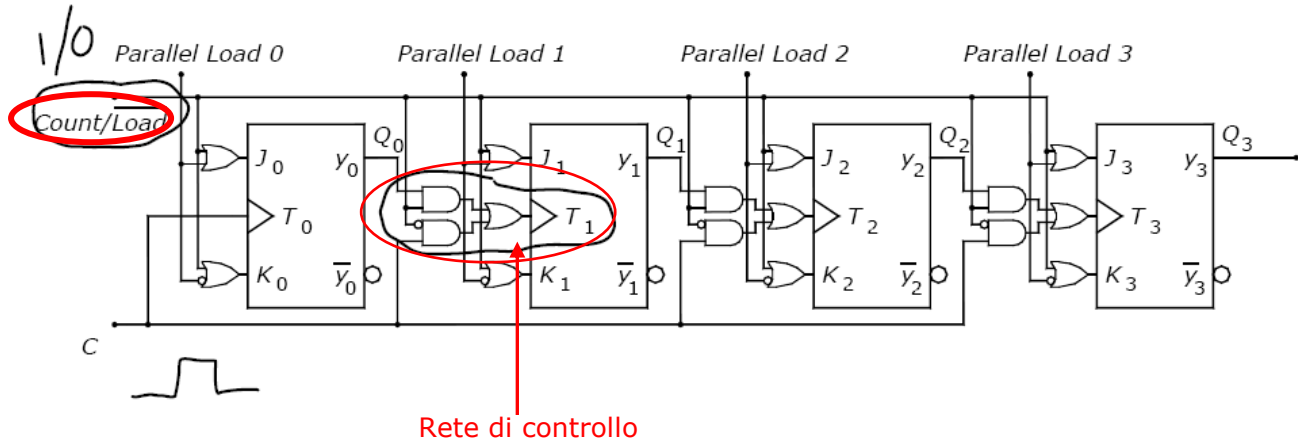


E' costituito da n bistabili di tipo T (commutano la propria uscita da 1 a 0 (zero) o da 0 ad 1 ad ogni colpo di clock) collegati in cascata: l'uscita di ogni bistabile diventa il segnale di abilitazione del successivo.

E' un componente sequenziale, a ogni colpo di clock, la configurazione binaria contenuta nei bistabili si incrementa di uno, può essere usato come divisore di frequenza, necessita di bistabili NON trasparenti (flip-flop) di tipo T.

- VARIANTI del Counter:

- Counter a caricamento parallelo: per conteggi a partire da valori qualsiasi (invece che con partenza da tutti zeri):



Dovendo poter inizializzare i bistabili ad 1 non possiamo usare i bistabili T ma dobbiamo usare i JK. Si utilizza un ingresso di comando COUNT/LOAD-negato che arriva con porte OR a tutti gli ingressi JeK dei bistabili.

Se COUNT vale 1, si attiva la funzione conteggio, mediante le porte OR tutti gli ingressi JeK di tutti e 4 i bistabili sono forzati ad 1 pronti a svolgere la funzione di commutazione. A questo punto è necessario che arrivi a tutti e 4 il colpo di clock.

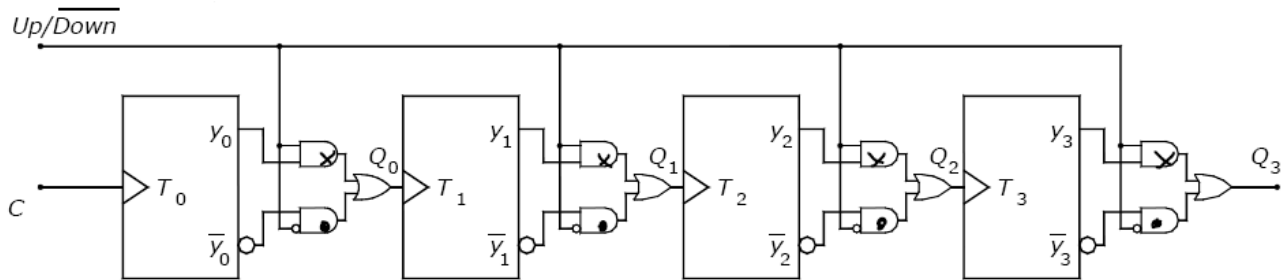
Il clock entra nell'ingresso di controllo del primo bistabile, l'uscita entra nell'ingresso di controllo del secondo bistabile e così fino al quarto.

Ecco perchè serve la "rete di controllo" su tutti i bistabili tranne il primo: se il COUNT/LOAD vale 1 viene portato sulla parte inferiore della porta AND di questa rete lasciando così passare il segnale che arriva dall'uscita Q0 del bistabile precedente e propagandolo tramite la porta OR all'ingresso di controllo del bistabile successivo.

Se invece COUNT/LOAD-negato vale ZERO, si attiva la funzione di caricamento parallelo, se il valore in arrivo è ZERO arriva 0 a J ed 1 a K attivando la funzione RESET, se il valore in ingresso è 1 arriva 1 a J e 0 a K attivando la funzione SET. E' necessario che tutti e 4 i bistabili svolgano la funzione SET o RESET simultaneamente in modo da caricare il valore richiesto in un'unico clock, quindi il segnale dev'essere propagato simultaneamente e non a cascata come nella funzione conteggio. La "rete di controllo" ora riceve ZERO da COUNT/LOAD chiudendo la parte superiore e aprendo la parte inferiore lasciando passare all'ingresso di controllo ciò che arriva dal segnale di controllo (clock).

Con un colpo di clock invece di fare il conteggio carichiamo simultaneamente ciò che appare agli ingressi parallel-load.

– **Counter bidirezionale (UP/DOWN-negato):** per contare in CRESCITA o in DECRESCITA (cioè sommare o sottrarre uno a seconda di cosa fanno gli ingressi di pilotaggio)



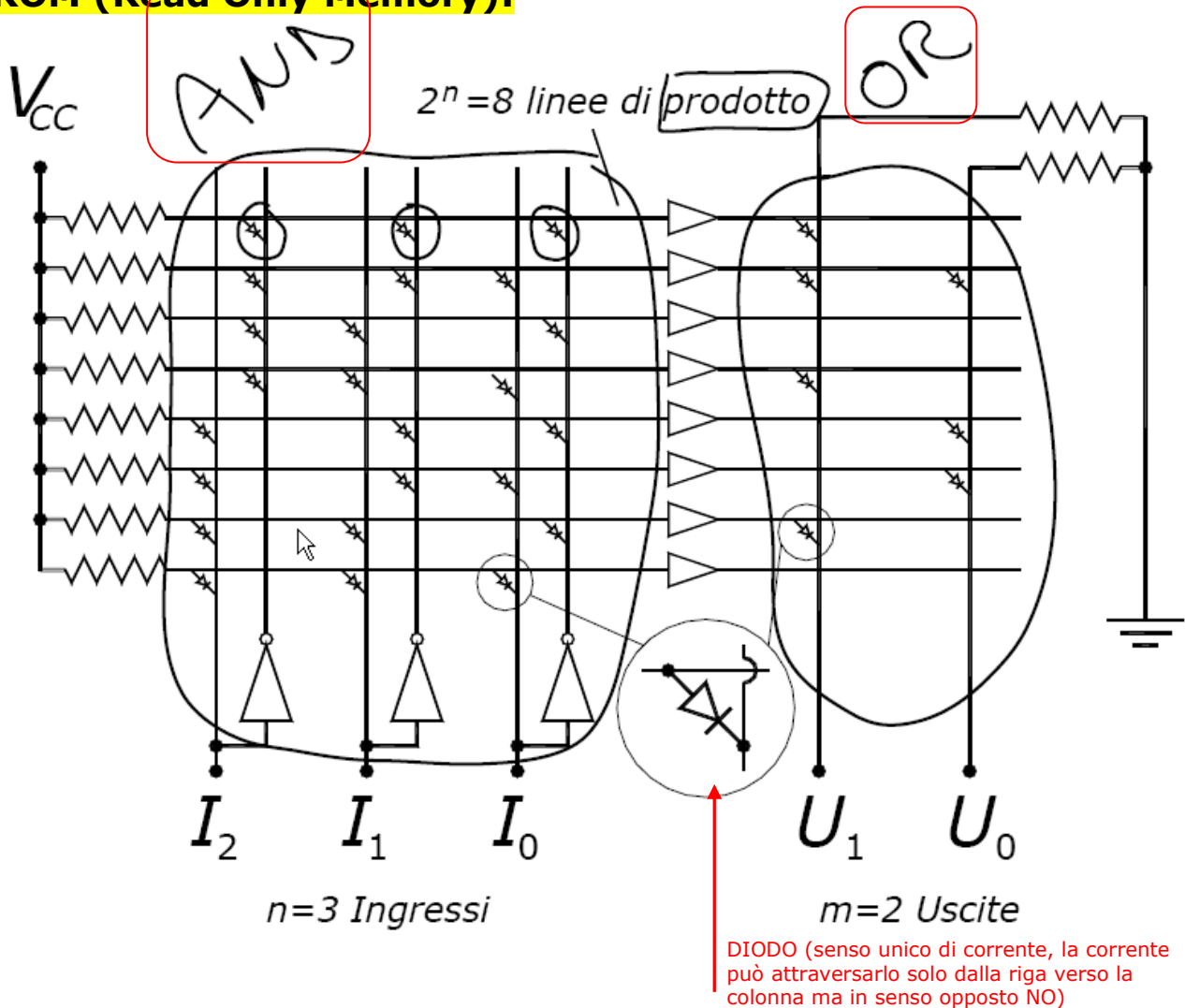
Se UP/DOWN-negato sono aperte le porte AND superiori, se vale ZERO si aprono le porte AND inferiori.

Ad ogni colpo di C, se UP/DOWN-negato vale 1 otteniamo un contatore che incrementa il valore di 1, se vale 0 un valore DECREMENTATO di 1 ad ogni clock.

LSI: Large Scale Integration

- alcune migliaia di porte logiche in ogni chip;
- 20-100 pin per chip;
- realizzano funzioni complesse (come le varie parti di un calcolatore):
 - unità centrale (di qualche anno fa ...);
 - gestori di periferiche (disco, video, ecc.).
- possono essere componenti programmabili. Si parla di programmabilità perché la funzione svolta dal componente può essere definita (programmata) dall'utilizzatore, con una interazione ridotta o nulla con il produttore del componente.:

ROM (Read Only Memory):



Nella prima riga i DIODI sono stati collegati ai 3 ingressi negati, nella seconda riga ai primi 2 negati ed al terzo diritto, nella terza riga negato, diritto, negato e avanti così "esplorando" tutte le 8 possibili configurazioni degli ingressi I_2, I_1, I_0 .

Questo significa che in presenza di 1 configurazione degli ingressi solo una delle righe si trova con tutti e 3 gli incroci ad alta tensione quindi nessuno dei 3 diodi riesce a far scendere la tensione della riga. Tutte le altre righe hanno almeno uno dei 3 diodi a tensione nulla e conseguentemente la tensione della riga si trova a valore ZERO.

In questa zona del componente si realizzano gli 8 mintermini degli ingressi I_0, I_1, I_2 .

In pratica, data una configurazione degli ingressi, rimane ad 1 un'unica linea di prodotto, se la linea "accesa" è collegata tramite diodo ad una delle 2 uscite anche l'uscita vale 1 altrimenti l'uscita resta a ZERO.

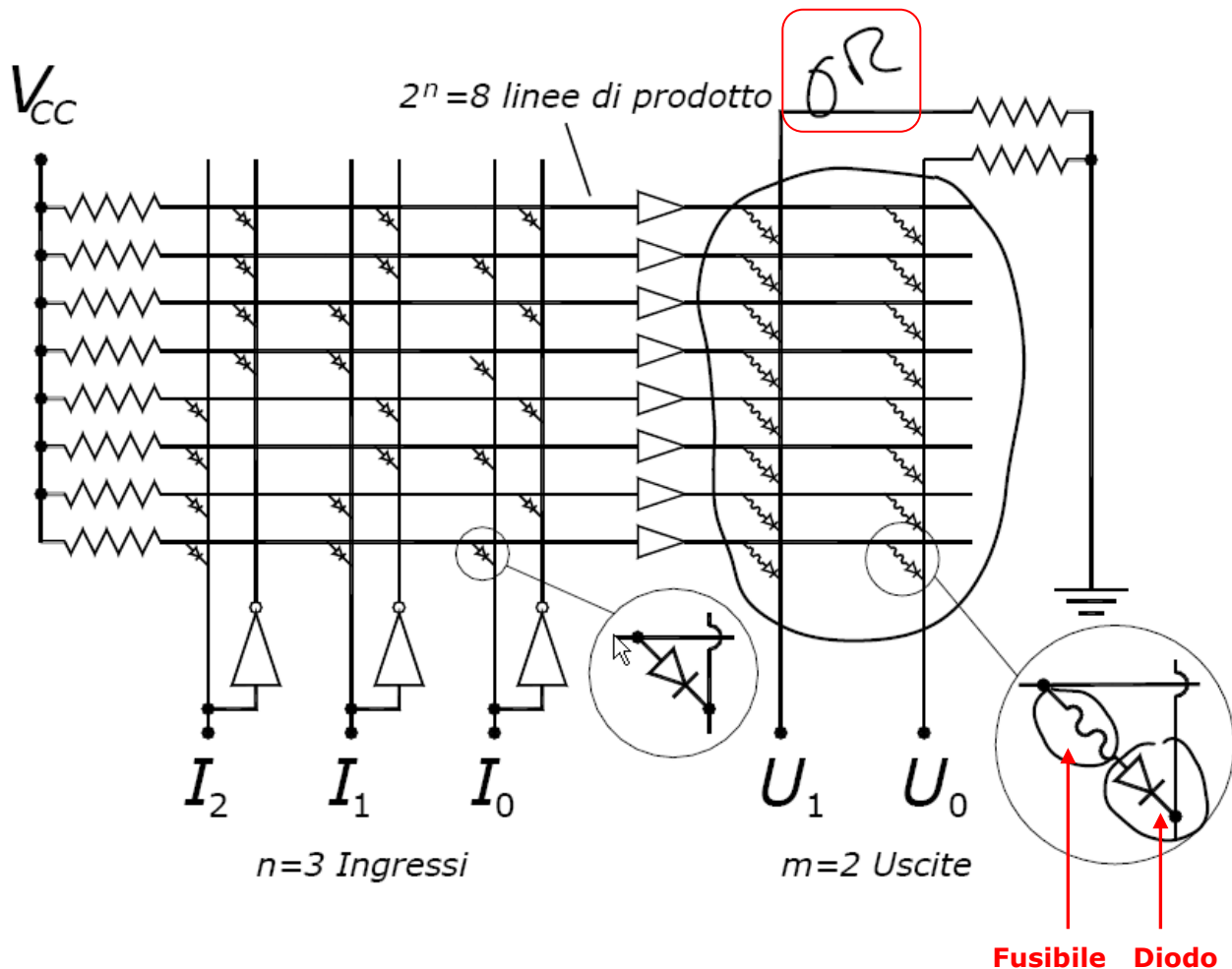
La prima sezione (ingresso) della ROM effettua un'operazione AND e genera tutti i mintermini di una funzione a n ingressi ed, a livello elettrico, riproduce su ogni linea di prodotto la funzione AND dei valori di ingresso, diritti o negati.

Invece la zona di uscita effettua un'operazione OR per ogni linea di uscita, sommando i mintermini ai quali rimane collegata ed, a livello elettrico, riproduce su ogni linea di uscita la funzione OR dei mintermini.

Le CARATTERISTICHE DELLA ROM:

- ⊙ Ha una struttura regolare, indipendente dalla configurazione della sezione OR.
- ⊙ Le due sezioni AND e OR sono costituite da diodi realizzati in fase di produzione del circuito integrato, quindi estremamente affidabili.
- ⊙ La personalizzazione della ROM richiede l'invio alla silicon foundry della struttura della sezione OR e la produzione di circuiti integrati specifici per il singolo utilizzatore.
- ⊙ Eventuali errori di definizione della sezione OR richiedono la costruzione di un nuovo integrato.

PROM (Programmable Read Only Memory):



Per eliminare i difetti della ROM sono state create le PROM: nella sezione OR invece di DIODI precostituiti in produzione realizzo una matrice completa di connessioni fatte da un diodo ed un FUSIBILE in serie. Inizialmente la PROM è completamente connessa, ogni uscita è collegata a tutte le linee di prodotto.

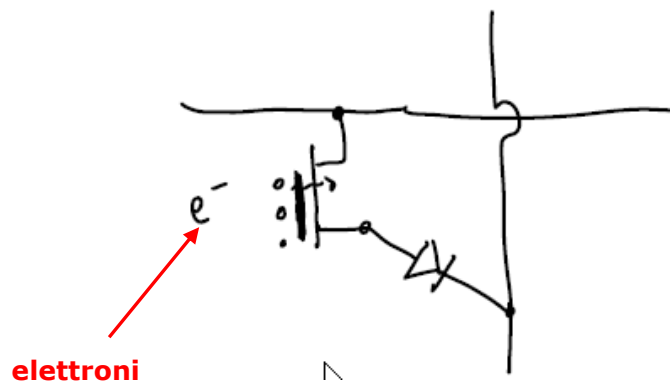
L'utilizzatore può bruciare (fondere) i fusibili dei contatti non desiderati mediante il prom-programmer

Le CARATTERISTICHE DELLA PROM:

- ☺ Ha una struttura regolare.
- ☺ Sia la sezione AND sia la sezione OR sono realizzate in modo standard dalla silicon foundry.
- ☺ La personalizzazione della sezione OR avviene da parte dell'utilizzatore finale.
- ☺ La personalizzazione è irreversibile (bruciatura dei fusibili per i contatti da eliminare).
- ☺ La presenza dei fusibili nei contatti da mantenere rallenta la propagazione dei segnali elettrici, quindi la sezione OR risulta più lenta di una sezione OR in una ROM

EPROM (Erasable Programmable Read Only Memory):

Ha, a differenza della PROM, una sezione OR con i diodi collegati in serie a un transistor MOS (metallo ossido semiconduttore) a gate sommerso.



La **programmazione** avviene forzando cariche elettriche nel gate sommerso mediante tensioni imposte dal PROM Programmer, la **cancellazione** avviene mediante esposizione a luce ultravioletta.

Le CARATTERISTICHE DELLA EPROM:

- ☺ Ha una struttura regolare.
- ☺ Sia la sezione AND sia la sezione OR sono realizzate in modo standard dalla silicon foundry.
- ☺ La personalizzazione della sezione OR avviene da parte dell'utilizzatore finale.
- ☺ La personalizzazione è reversibile.
- ☺ La cancellazione richiede estrazione del componente dal circuito per esposizione a luce UV.
- ☺ Il componente è vulnerabile a raggi solari forti o non protetti dall'atmosfera (missioni spaziali).

EEPROM (E²PROM- Electrically erasable), EAROM

Sfruttano dispositivi il cui stato può essere alterato elettricamente.

Caratteristiche:

- ☺ Programmazione e cancellazione sono effettuate con tensioni e correnti: i componenti possono essere programmati e cancellati "in campo" (nel circuito in cui dovranno lavorare).
- ☺ Cancellazione e programmazione sono operazioni lente.
- ☺ Il numero di cancellazioni sopportabile dal componente è limitato.

Riassumendo, i **componenti della famiglia ROM** consentono una facile realizzazione di reti combinatorie utilizzando la prima forma canonica (SP): somma di mintermini.

La sezione AND, fissa, realizza tutti i 2^n mintermini degli n ingressi.

La sezione OR, programmabile, consente di scegliere quali mintermini sommare in ciascuna uscita.

I diversi componenti si differenziano per il modo di programmare (ed eventualmente ripristinare) la sezione OR.

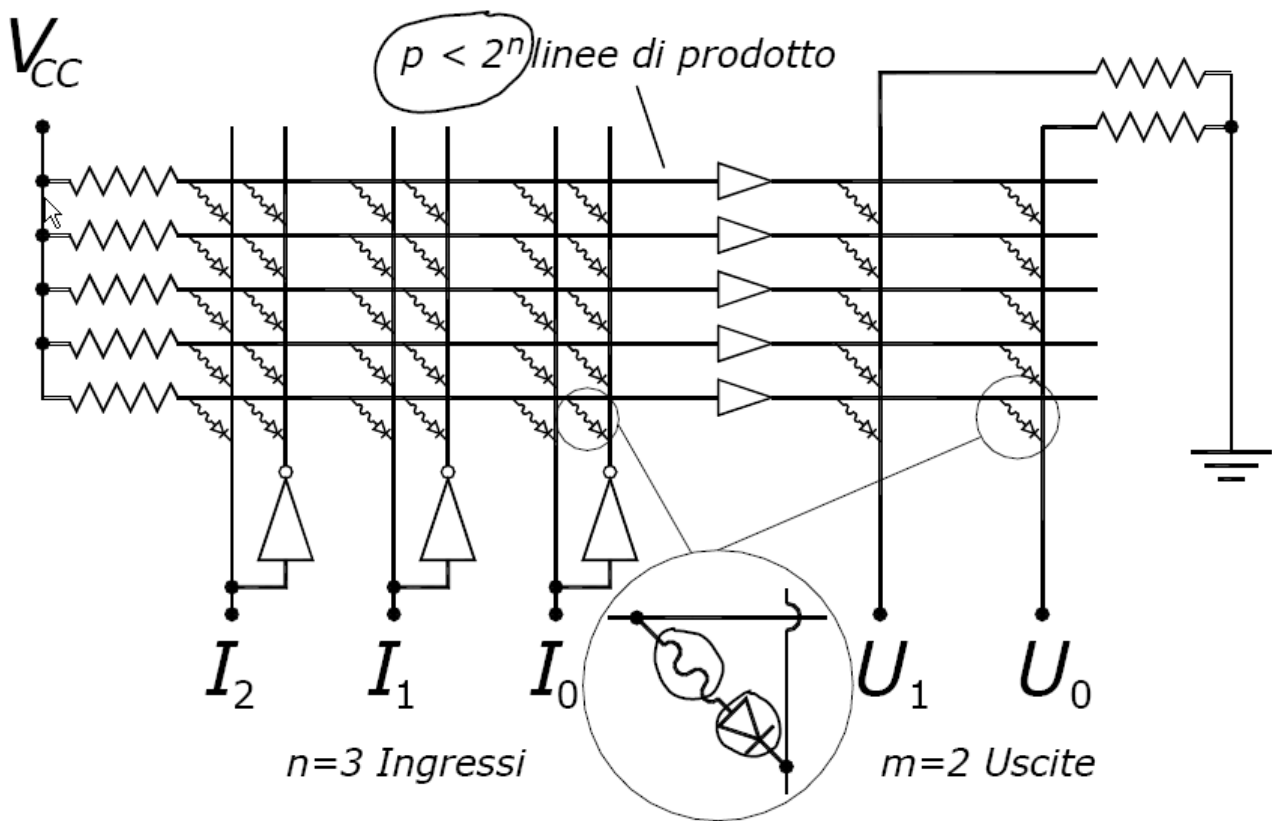
MA:

la sezione AND deve generare tutti i 2^n mintermini degli n ingressi;
 al crescere di n le dimensioni della sezione AND crescono in modo esponenziale;
 la sintesi come forma canonica SP non sfrutta nessuna ottimizzazione.

Per ovviare a questi problemi sono stati inventati i componenti della famiglia:

PLA (programmable logic array):

Il numero di linee prodotto è diventato **$p < 2^n$** , quindi la sezione AND ha un numero di linee prodotto minore del numero di mintermini associati agli ingressi (a fronte di 3 ingressi, quindi 8 mintermini, sono disponibili solo 5 linee prodotto):



Tutte e 2 le sezioni, sia la AND che la OR sono COMPLETAMENTE programmabili mediante bruciatura dei fusibili che non servono mediante programmatore di PLA.

La Sezione AND contiene quindi un numero di linee di prodotto decisamente inferiore a 2^n ; i collegamenti fra linee di ingresso e linee di prodotto sono programmabili, con tecnologia a fusibile ed ogni linea di prodotto può realizzare un **implicante** (non più un mintermine!!)

La sezione OR quindi, per ogni linea di uscita somma gli implicanti ai quali rimane collegata

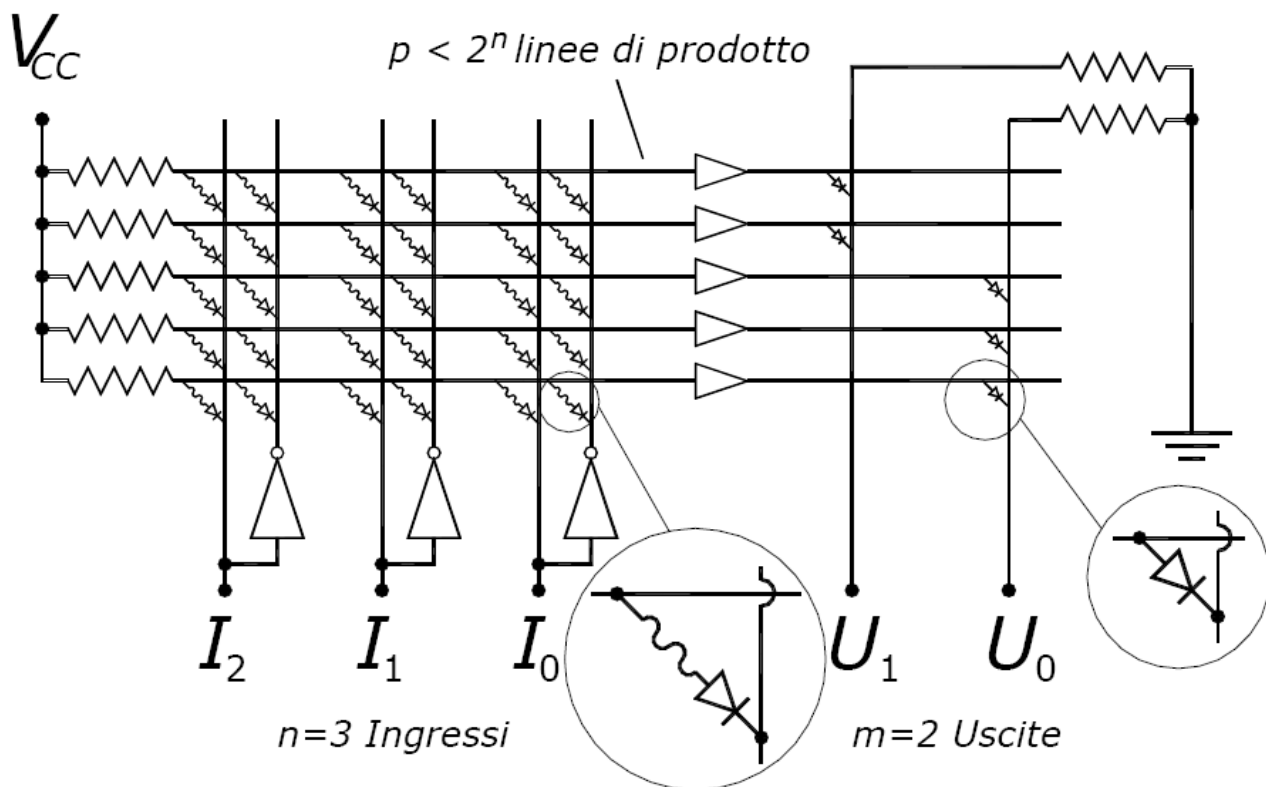
Le CARATTERISTICHE DELLA PLA:

- ☉ Ha una struttura regolare, indipendente dalla configurazione delle sezioni AND e OR.
- ☉ Le due sezioni AND e OR sono entrambe programmabili mediante bruciatura dei fusibili in serie ai diodi.
- ☉ La sintesi è: ottimizzata SP (implicanti) e soprattutto ottimizzata per reti a più uscite, perché uno stesso implicante può essere usato in più uscite.

tuttavia:

- ⊗ Il segnale elettrico deve attraversare DUE livelli di fusibili (AND e OR), quindi il componente risulta più lento.
- ⊗ il componente risulta più complesso da programmare;
- ⊗ la sintesi ottimizzata per reti a più uscite non è banale (le mappe di Karnaugh non sono di fatto utilizzabili).

PAL (programmable array logic):

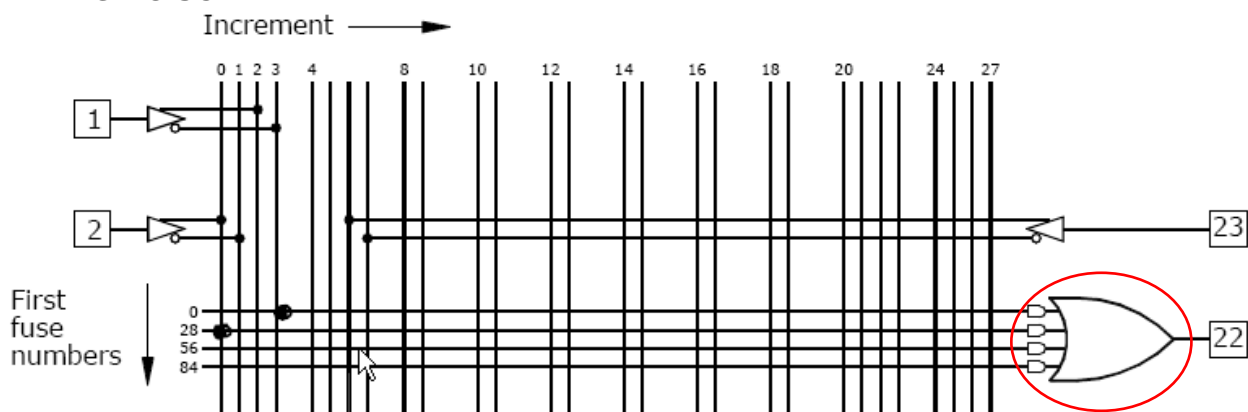


La sezione AND programmabile consente la sintesi ottimizzata, ma per non avere un secondo livello di fusibili da attraversare esemplificare la fase di programmazione, le uscite diventano OR in modo esclusivo di alcune linee prodotto. (ogni linea di prodotto è collegata in fase di produzione ad un'unica linea di uscita)

La sezione AND quindi contiene un numero di linee di prodotto decisamente inferiore a $2n$; i collegamenti fra linee di ingresso e linee di prodotto sono programmabili, con tecnologia a fusibile ed ogni linea di prodotto può realizzare un implicante.

La sezione OR invece ha ogni linea di uscita collegata a priori (diodi senza fusibili) ad alcune linee di prodotto e, ogni linea di prodotto è collegata a un'unica uscita.

DETTAGLIO SCHEMA PAL:



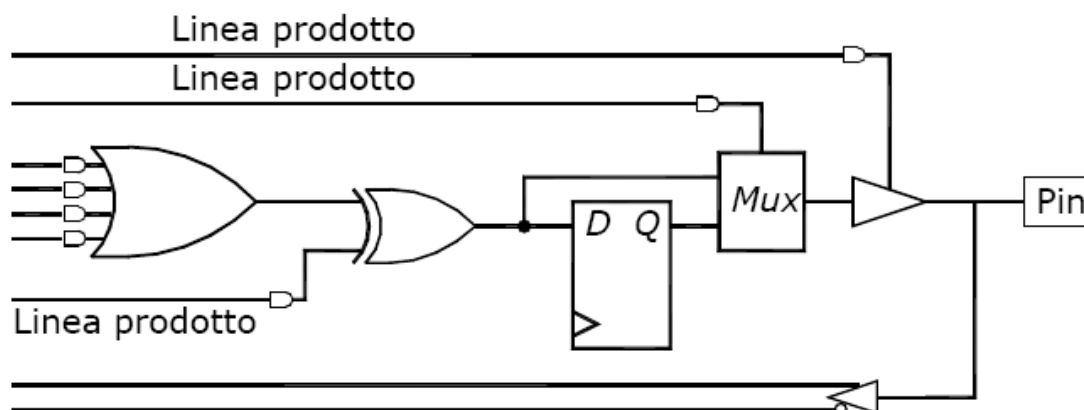
Nello schema elettrico in realtà non esistono né la OR né le AND ma collegamenti che fanno la funzione AND ed OR

Le CARATTERISTICHE DELLA PAL:

© Ha una struttura regolare, indipendente dalla configurazione delle sezioni AND e OR.

- ☺ La sola sezione AND è programmabile mediante bruciatura dei fusibili in serie ai diodi.
- ☺ La sintesi è: ottimizzata SP (implicanti) MA per reti a singola uscita, perché ogni implicante è collegato a un'unica uscita.
- ☺ Il segnale elettrico deve attraversare un solo livello di fusibili (AND), quindi la PAL risulta più veloce della PLA

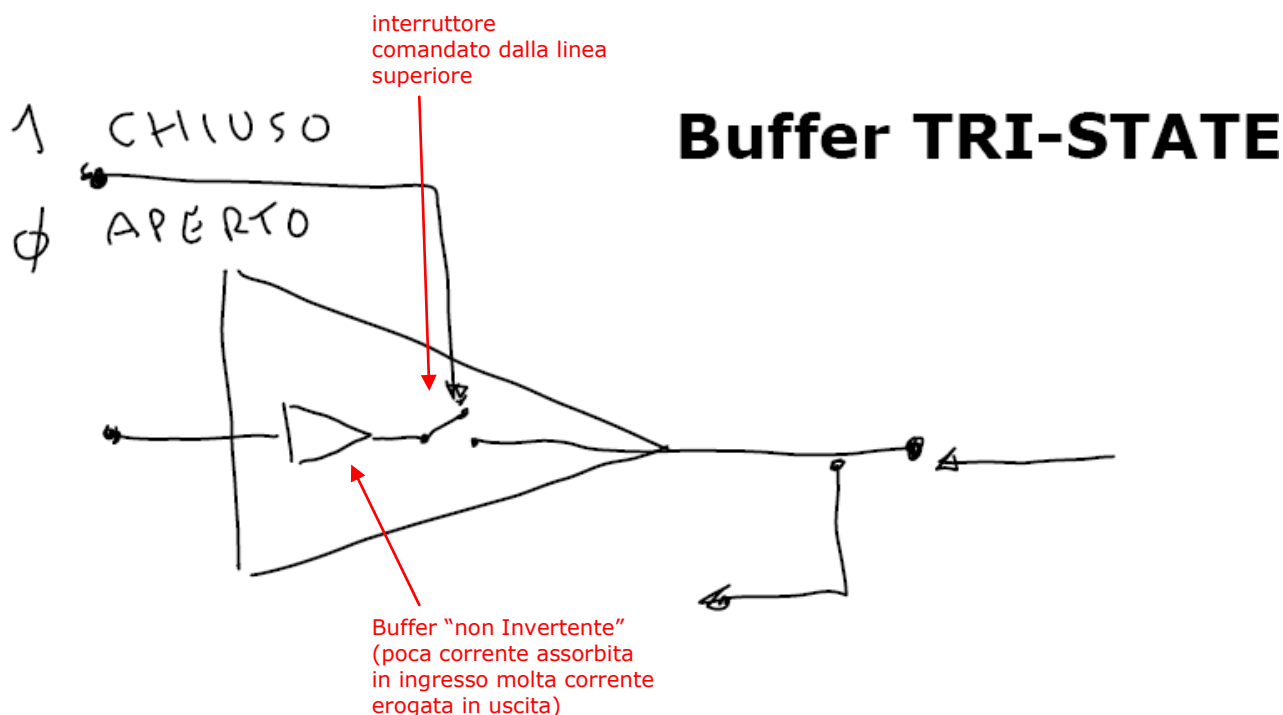
PAL a stadio di uscita programmabile (evoluzione più interessante della PAL tradizionale):



Questa variante ci dà la possibilità di programmare un piedino ad essere ingresso oppure uscita, a polarità diretta o negata, o ancora di variare dinamicamente la direzione di propagazione dei segnali di uno stesso piedino (linea bidirezionale).

Infine, avendo bistabili a bordo, dà la possibilità di realizzare reti sequenziali sia alla Mealy sia alla Moore, in base al fatto che le uscite finali vengano tutte da piedini staticizzati (stato presente) oppure che in parte vengano da reti combinatorie passate direttamente in uscita con la trasparenza agli ingressi tipica delle macchine alla Mealy.

L'elemento circuitale che permette la realizzazione di linee bidirezionali è il Buffer Tri-state:



Dispositivo la cui uscita può trovarsi in 3 stati elettrici. Se abbiamo 1 sulla linea superiore, l'interruttore si chiude e possiamo avere 2 stati:

- 0 a bassa impedenza, il buffer cerca di tenere la tensione a zero assorbendo la corrente necessaria
- 1 a bassa impedenza, il buffer si porta a tensione di alimentazione ed eroga corrente cercando di mantenere quella tensione

Se invece abbiamo ZERO sulla linea superiore, l'interruttore si apre ed abbiamo alta impedenza (Z), non c'è passaggio di corrente, il dispositivo non sente la tensione e consente di "scollegare" il dispositivo dalla linea di uscita permettendole di diventare un ingresso e propagare il suo segnale in un'altra direzione.

In conclusione, i componenti PAL consentono la realizzazione di reti combinatorie utilizzando la forma ottimizzata SP: somma di implicanti:

La sezione AND, programmabile, consente di scegliere quali implicanti realizzare.

La sezione OR, fissa, usa un certo numero di implicanti per ogni uscita.

EPLD, FPGA:

I componenti PAL consentono una realizzazione ottimizzata di reti combinatorie e sequenziali.

La complessità delle reti realizzabili, tuttavia, è modesta: ingressi e uscite della rete sono collegati ai piedini del dispositivo (problema del pin-out).

Per riuscire a migliorare lo sfruttamento del silicio, serve realizzare componenti nei quali più reti combinatorie e/o sequenziali sono connesse all'interno del dispositivo che non richiedono di uscire all'esterno del chip andando su piedini e collegamenti elettrici esterni.

Per superare i limiti delle PAL sono successivamente nate le seguenti famiglie di devices (recenti):

EPLD (erasable programmable logic devices):

Componenti programmabili in tecnologia EPROM. La programmazione si effettua con correnti elettriche ad elevata intensità fatte circolare all'interno del dispositivo e la cancellazione con esposizione ai raggi ultravioletti.

La caratteristica principale è che nel componente sono presenti numeri anche elevati (decine o centinaia) di macrocelle (semplici PAL), ovvero l'OR di 8 termini prodotto (implicanti), con un bistabile D per poter essere staticizzata, e con tutti i multiplexer necessari per programmare l'uscita (possibilità di scegliere polarità, direzione).

Le connessioni fra le macrocelle e la distribuzione del segnale di sincronismo sono realizzate con collegamenti direttamente sul chip che non richiedono pin-out, programmabili in modo analogo alla programmazione del comportamento delle macrocelle.

Il vantaggio è che possiamo avere un numero elevato di semplici funzioni interconnesse fra loro a costruire funzioni anch'emo molto complesse sul chip mantenendo il pin-out relativamente contenuto ed usandolo solo per i segnali destinati ad entrare o uscire dall'EPLD

FPGA (field programmable gate array):

Componenti simili ai componenti EPLD, ovvero matrici anche dense di macrocelle interconnettibili sul componente. La vera differenza con le EPLD, è che la programmazione non viene effettuata da transistor MOS a gate sommerso, ma da un bistabile D che apre/chiude il singolo contatto. La programmazione richiede il caricamento iniziale dello stato aperto/chiuso di tutti i punti programmabili

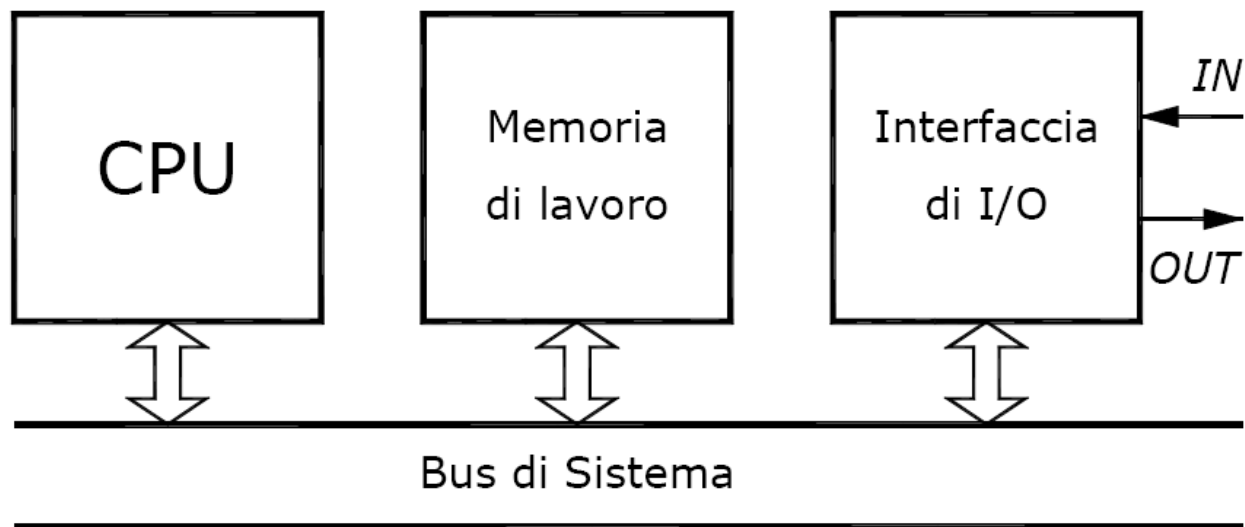
La prototipazione risulta perciò estremamente facilitata (basta riconfigurare sul campo il componente cambiando lo stato dei contatti errati e non è necessario portare il componente fuori dal circuito). È possibile inoltre progettare sistemi che si riconfigurano dinamicamente in base alle esigenze operative (evolvable hardware, hw capace di automodificarsi in base alle esigenze o sotto il controllo di un sw in esecuzione su un'unità centrale che verifica il comportamento dell'hw).

La tecnologia FPGA, trasforma la programmazione dei dispositivi in scrittura di bit in Bistabili e facilita la prototipazione e la riconfigurazione ma consente soprattutto di progettare sistemi capaci di variare dinamicamente la propria struttura sulla base di quello che il mondo esterno richiede.

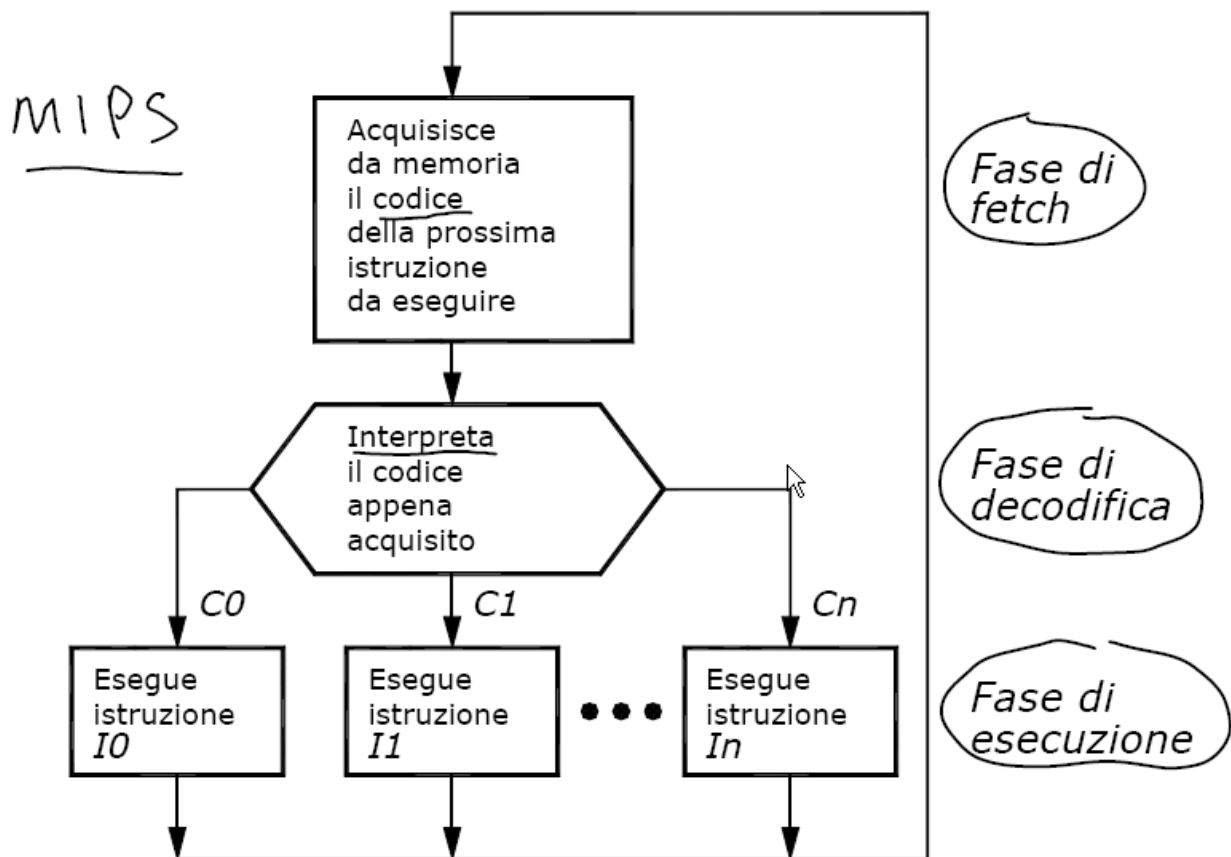
RIVEDERE CON ATTENZIONE Modulo 5 – U.D. 3 – Lez. 5

Modulo 6 – Linguaggio macchina:

Schema della macchina di Von Neumann:



Funzionamento della CPU:



MIPS: unità di misura che ci dice quante volte al secondo l'unità centrale gira in questo schema (mega instructions per seconds – utilizzata per quantificare le prestazioni di un calcolatore).

Fase di fetch: preleva dall'esterno una stringa di bit che indica il prossimo passo da fare (macchina programmabile), esegue quindi, uno dopo l'altro, una sequenza di passi (programma).

Fase di decodifica: interpreta, decodifica, la stringa di bit come istruzione macchina.

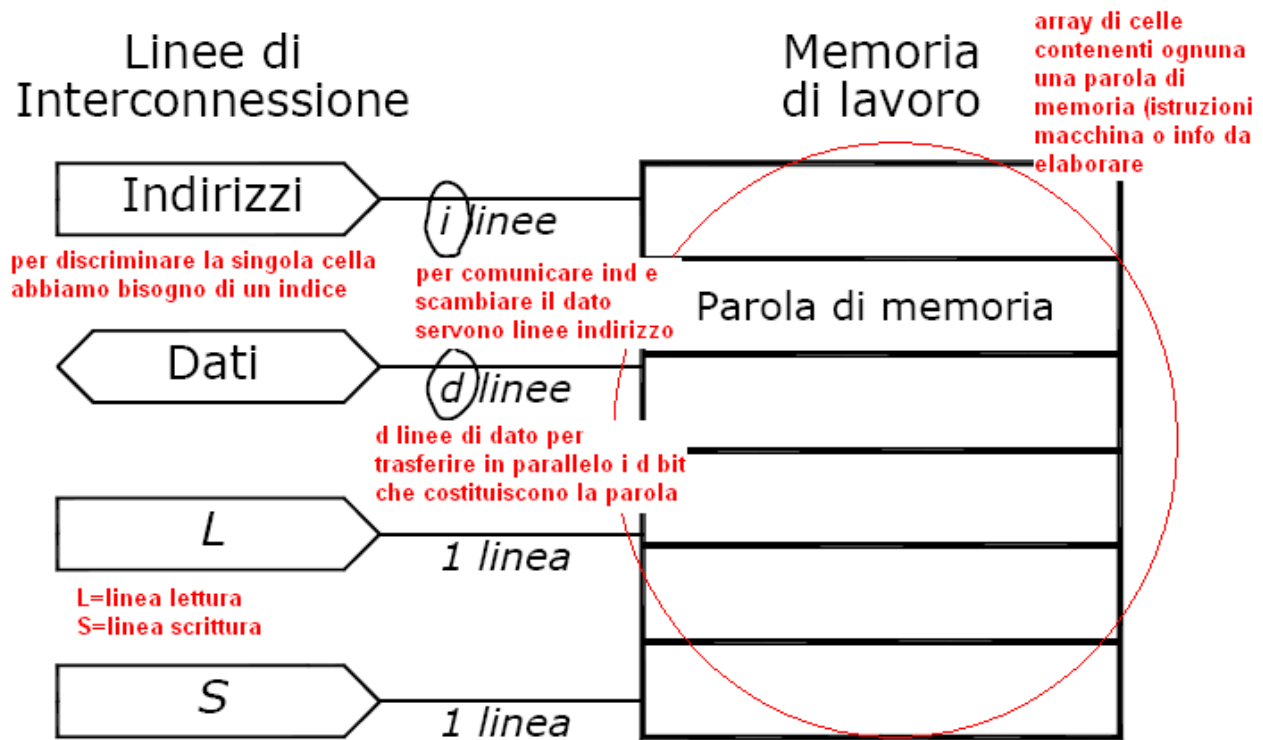
Fase di esecuzione: svolge quanto richiesto accede all'esterno per scambiare dati

La CPU deve scoprire i propri "compiti" volta per volta in modo tale da variare il proprio risultato sulla base delle esigenze esterne, ovvero le sequenze di stringhe di bit.

Ogni stringa di bit rappresenta istruzione, o informazioni (dati).

Serve un contenitore di stringhe di bit (tante) che possa scambiarle con la CPU rispettandone i tempi di lavoro (velocemente!): la **MEMORIA di LAVORO**.

Struttura della memoria di lavoro:



Nell'interazione tra CPU e Memoria di lavoro, la CPU segnala alla memoria la cella a cui è interessata (mediante l'indirizzo, cioè la configurazione di bit che segnalano la posizione della cella all'interno dell'array), il tipo di operazione che intende svolgere:

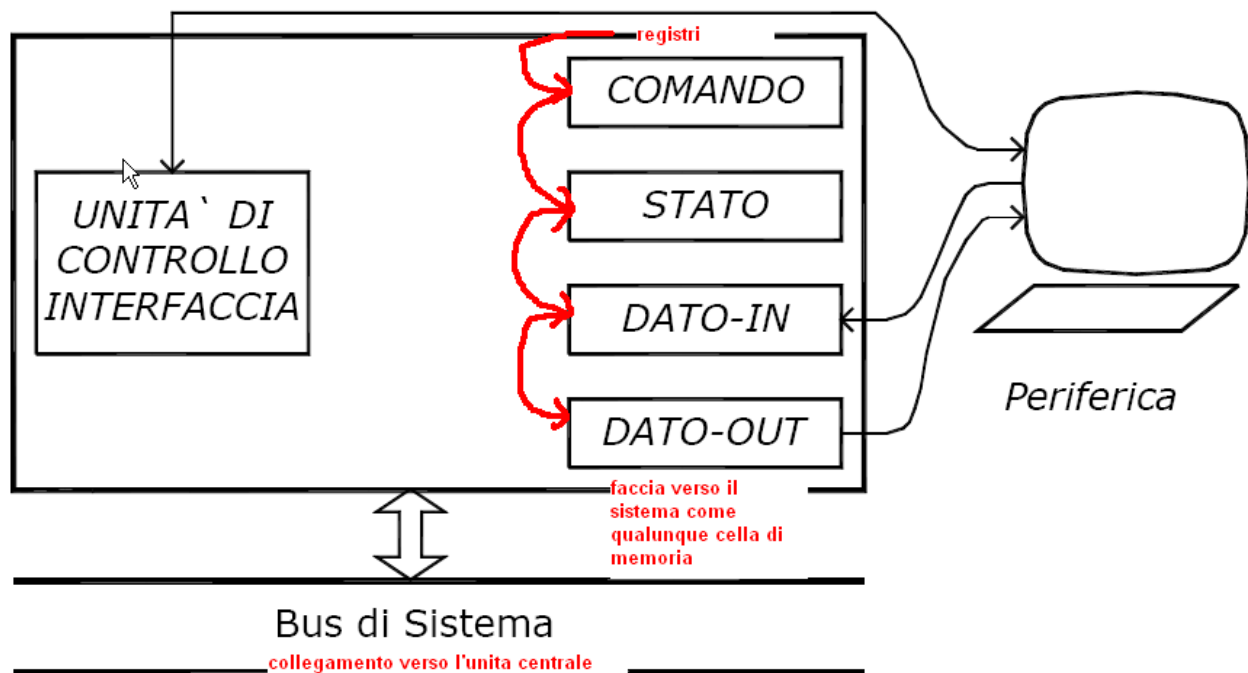
- prelievo o Lettura del contenuto della cella
- modifica o Scrittura del contenuto nella cella

CPU e memoria si scambiano il contenuto della cella, secondo la direzione richiesta dalla CPU: da mem a cpu in caso di lettura, da cpu a memori ain caso di scrittura.

La CPU svolge ruolo Master, decide infatti quando e cosa fare mentre la memoria svolge il ruolo Slave e risponde alle richieste della CPU.

Poichè è necessario poter inserire in memoria di lavoro i programmi da eseguire, inserire i dati da elaborare e prelevare i risultati, servono interfacce fra mondo elettronico del calcolatore e mondo esterno (periferia: tastiera, video, mouse...)

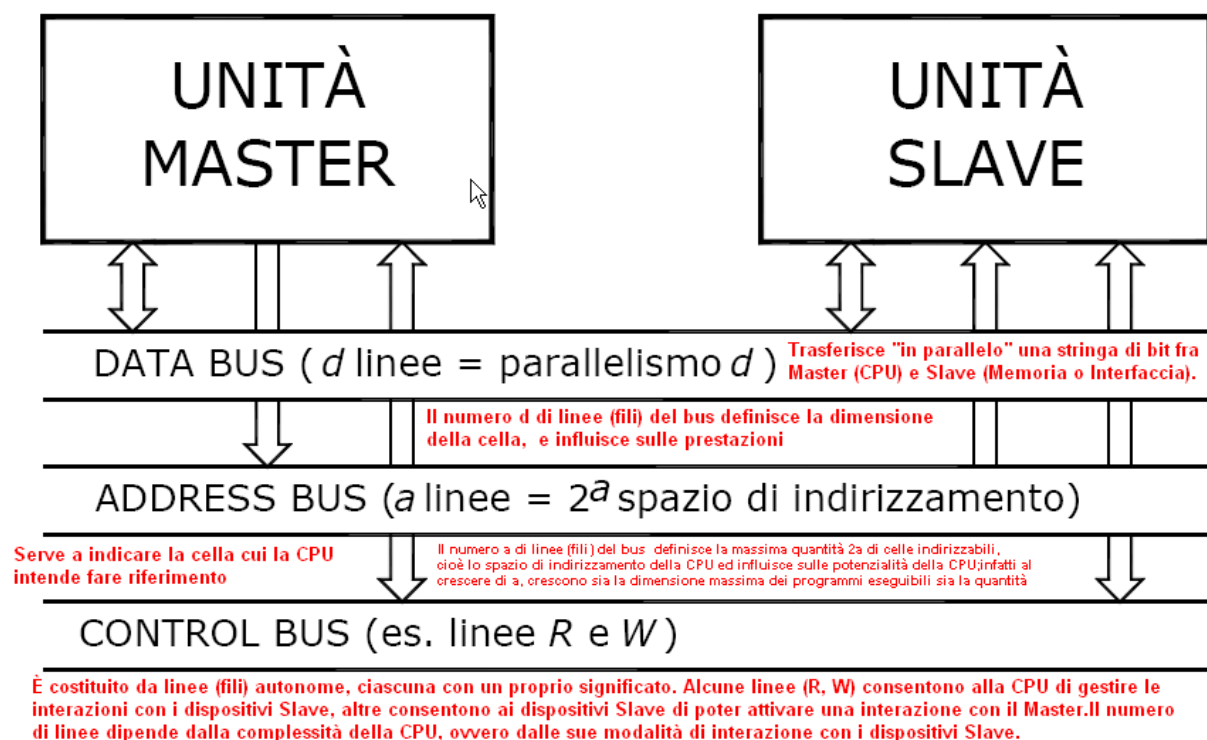
Struttura dell'interfaccia I/O:



La CPU vede l'interfaccia simile alla memoria ma con alcune "celle" o registri (COMANDO, STATO, DATO-IN, DATO-OUT) che hanno significati particolari e che consentono all'interfaccia di gestire la periferica connessa sulla base delle indicazioni dell'unità centrale. L'unità di controllo dell'interfaccia è progettata appositamente per gestire la specifica periferica connessa e le operazioni di lettura e scrittura nei registri diventano interazioni con il mondo esterno permettendo l'invio di comandi alla periferica, la conoscenza dello stato della periferica e lo scambio di dati.

Struttura del BUS:

consente ad un'unità Master di "parlare" a tutte le unità Slave che possono essere presenti



Il numero **d** di linee (fili) del bus dati definisce la dimensione della cella, quindi della parola di memoria e influisce sulle prestazioni, in quanto la banda passante del bus (bit trasferibili per unità di tempo) cresce al crescere di d.

Ormai da tempo, d è multiplo del byte: 8 bit, 16 bit, 32 bit, 64 bit, 128 bit...

Il numero **a** di linee (fili) del bus address definisce la massima quantità 2^a di celle indirizzabili, cioè lo spazio di indirizzamento della CPU (non necessariamente pieno di memoria) ed influisce sulle potenzialità della CPU; infatti al crescere di a, crescono sia la dimensione massima dei programmi eseguibili, sia la quantità massima di dati elaborabili.

Ha avuto una evoluzione un po' diversa da quella del bus dati: 16 bit, 20 bit, 24 bit, 32 bit...

Il Control Bus è costituito da linee (fili) autonome, ciascuna con un proprio significato.

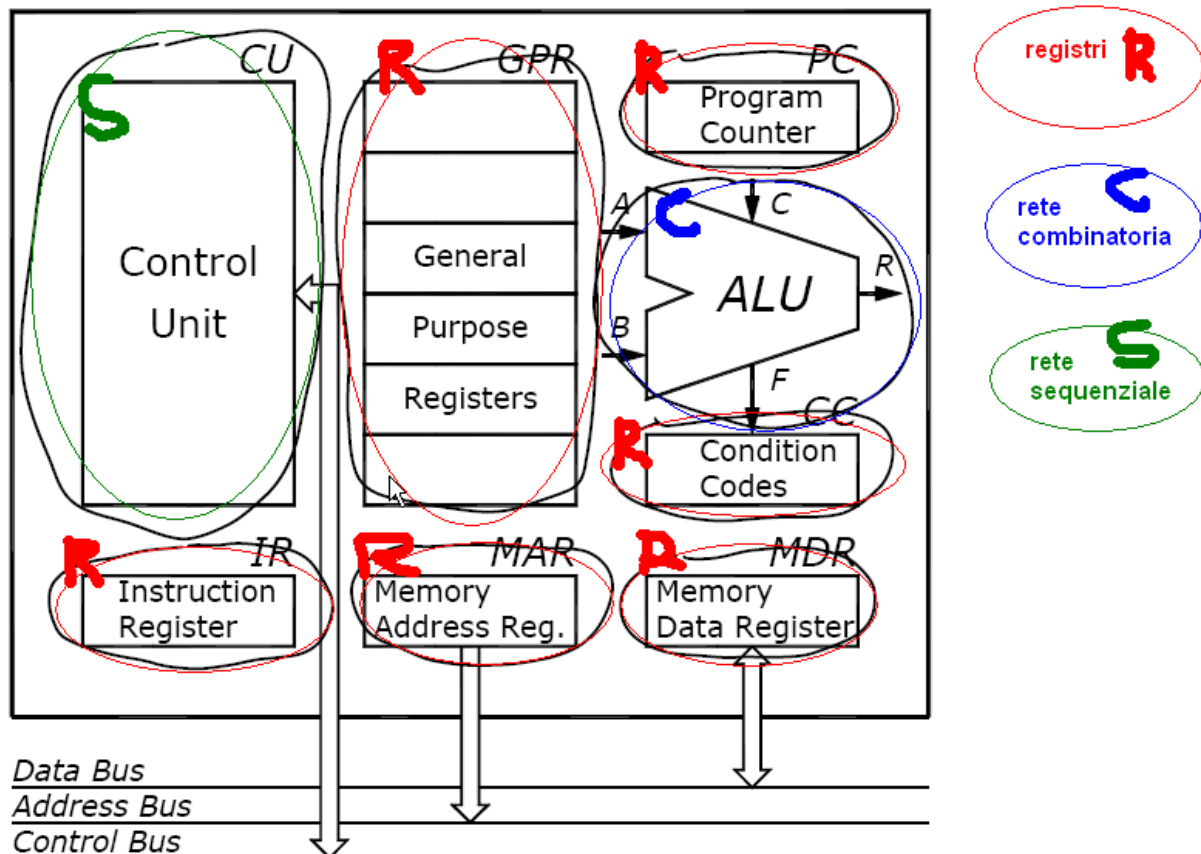
Alcune linee (R, W) consentono alla CPU di gestire le interazioni con i dispositivi Slave, altre consentono ai dispositivi Slave di poter attivare una interazione con il Master.

Il numero di linee dipende dalla complessità della CPU, ovvero dalle sue modalità di interazione con i dispositivi Slave.

La CPU quindi interagisce con il resto del mondo elettronico (memoria e interfacce) mediante una serie di linee parallele (BUS).

Alcune linee sono dedicate al trasferimento di bit (bus dati), altre sono dedicate all'indicazione della cella cui la CPU fa riferimento (bus address) oppure servono ad orchestrare le interazioni fra CPU e resto del mondo elettronico (control bus). I numeri di linee (dimensioni del bus) sono correlati alle prestazioni della CPU.

Struttura interna della CPU:



Ci sono un certo numero di "Registri", contenitori di bit, costituiti tipicamente da un certo insieme di bistabili. Non hanno iniziativa propria ma sono dei contenitori di informazioni destinati a vari scopi che servono per portare avanti il lavoro della CPU:

PC (Program Counter): registro che contiene l'indirizzo della cella di memoria nella quale si andrà a prelevare il codice della prossima istruzione macchina da eseguire (puntatore)

GPR (General Purpose Registers) registri di uso generale, che contengono i dati (cioè le informazioni, codificate anch'esse mediante stringhe di bit) in corso di elaborazione.

CC (Condition Codes) registro che contiene informazioni sull'esito dell'ultima elaborazione (per es.: il risultato di un'operazione aritmetica è stato negativo, nullo, positivo oppure un overflow etc.)

Questi 3 registri sono NOTI al programmatore, le istruzioni in linguaggio macchina fanno esplicito riferimento a questi registri.

I seguenti invece, NON sono direttamente visibili al programmatore ma sono comunque fondamentali per il funzionamento della CPU:

IR (Instruction Register): registro che contiene il codice (stringa di bit) dell'istruzione in corso di esecuzione. Alla fine della fase di fetch, ciò che era presente nella cella indirizzata del Program Counter, deve venire ricopiato nell'IR per consentire alla unità centrale di proseguire con la fase di decode.

MAR (Memory Address Register): registro che consente alla CPU di emettere sull'Address Bus l'indirizzo della cella del dispositivo Slave che intende leggere o scrivere, quindi l'indirizzo del contenitore di bit con il quale interagire.

MDR (Memory Data Register): registro che consente il trasferimento di un dato dalla CPU al Data Bus durante la scrittura nei dispositivi Slave, oppure dal Data Bus alla CPU durante la lettura dai dispositivi Slave.

Gli ultimi 2 registri (MAR ed MDR) sono le "finestre" della CPU rispettivamente sull'address BUS e sul Data BUS (i contenitori di informazione verso questi 2 bus)

Esistono poi 2 elementi specifici del lavoro dell'unità centrale:

La **ALU (arithmetic logic unity)** che è una rete combinatoria destinata a svolgere operazioni aritmetiche e logiche sulle operazioni contenute nei registri. (somma in complemento a due, a volte sottrazione, moltiplicazione e divisione) e logiche (AND, OR, NOT, confronti); l'esito delle sue operazioni viene memorizzato nel registro CC (Condition Codes).

La **Control Unit** che ha lo scopo di orchestrare il comportamento della CPU e le sue interazioni con il mondo esterno (è una rete sequenziale): acquisisce e decodifica le istruzioni macchina presenti in memoria di lavoro e controlla il funzionamento di tutti gli elementi della CPU e, mediante il bus di controllo, del resto del calcolatore (dispositivi Slave).

Il progetto della CPU è il progetto degli elementi interni all'UC e del suo set di istruzioni:

L' **ISA (Instruction Set Architecture)** insieme di attività elementari (istruzioni) che la specifica CPU è in grado di comprendere (decodificare) e svolgere (vocabolario di parole comprensibili alla CPU);

la codifica binaria delle istruzioni costituisce il linguaggio macchina della specifica CPU; ogni istruzione è caratterizzata da:

- codice operativo (**opcode**) che indica di quale istruzione si tratta;
- operandi (**operands**) che costituiscono i dati o le informazioni aggiuntive necessarie per eseguire l'istruzione.

Esistono vari **tipi di istruzioni**, i 3 fondamentali sono:

Operative: richiedono alla CPU di svolgere elaborazioni sui dati, utilizzando l'ALU (somme e sottrazioni, operazioni logiche, confronti, ecc.).

Trasferimento: servono a prelevare da memoria di lavoro o da interfaccia di I/O i dati su cui operare e a trasferire in memoria di lavoro o interfaccia di I/O i risultati.

Controllo: servono a variare l'esecuzione in sequenza delle istruzioni macchina (salti condizionati e incondizionati); sono l'essenza stessa della programmazione e permettono ad esempio i "cicli".

Infine i **modi di indirizzamento** ovvero le diverse modalità, disponibili nel linguaggio macchina, per recuperare i dati necessari per l'esecuzione delle istruzioni:

Immediato: il dato è fornito all'interno dell'istruzione macchina (quindi durante la fase di fetch si prelevano anche le informazioni necessarie per eseguire l'istruzione)

Diretto : l'istruzione macchina fornisce l'indirizzo della locazione di memoria contenente il dato da utilizzare e richiede quindi all'UC un nuovo accesso a memoria per prelevare il dato.

Indiretto: l'istruzione macchina fornisce l'indirizzo di una cella che contiene a sua volta l'indirizzo della cella contenente il dato (sono necessari quindi 2 ulteriori accessi a memoria, il primo per prelevare l'indirizzo dell'operando ed il secondo per prelevare l'operando vero e proprio)

Base+offset: l'istruzione macchina indica un registro GPR interno cui sommare un offset (costante numerica) per ottenere l'indirizzo della cella contenente il dato.

Struttura interna della CPU LC2:

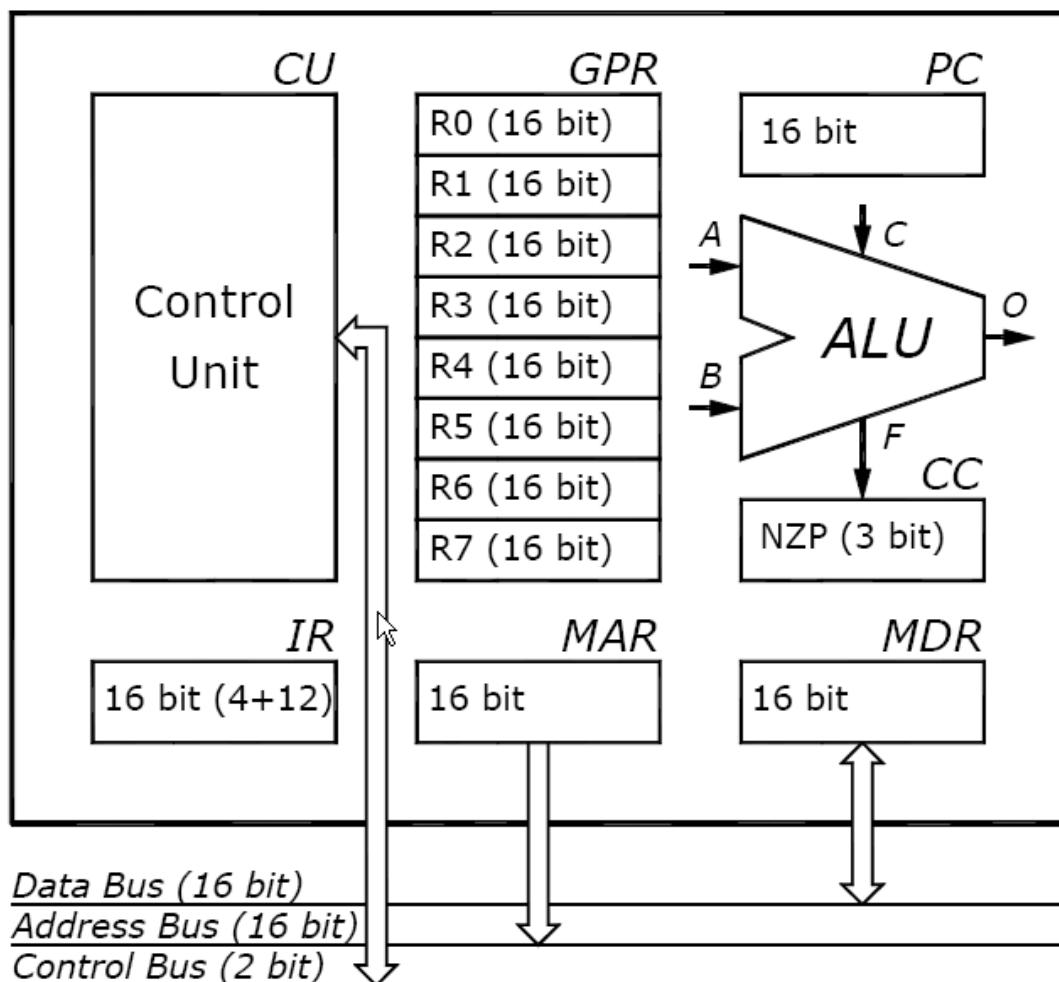
È una macchina RISC (Reduced Instruction Set Computer).

Poche istruzioni macchina: opcode 4 bit -> $2^4=16$ istruzioni.

Istruzioni macchina tutte di uguale lunghezza;

Pochi modi di indirizzamento: immediato, diretto, indiretto, base+offset.

Macchina a 16 bit: data bus a 16 bit -> celle di memoria da 16 bit; address bus a 16 bit -> $2^{16}=64K$ (65536) celle di spazio di indirizzamento



GPR: 8 registri a 16 bit, numerati da R0 a R7.

CC: costituito da 3 bistabili, è il segno (N, Z, P) dell'ultimo valore numerico scritto in un qualsiasi registro GPR.

ALU: solo le operazioni strettamente indispensabili:

- ADD
- AND
- NOT

Ogni istruzione ha 4 bit di opcode e usa gli altri bit per specificare - ove necessario - gli operandi.

Poiché lo spazio destinato a ogni istruzione macchina non è sufficiente per un indirizzamento immediato o diretto a 16 bit, usa indirizzamento a pagina corrente.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LEA	1	1	1	0	DR			pgoffset9								

ind16 → DR (indirizzamento immediato)

$$\text{ind16} = \text{PC}_{\text{MS7}} \& \text{pgoffset9}$$

occorre costruire un indirizzo a 16 bit ma poichè nell'istruzione sono disponibili solo 9 bit (pgoffset9), concateniamo (operatore &) i 7 bit più significativi del PC (MS: Most Significant) con i 9 bit di pgoffset9;

l'indirizzo creato appartiene alla pagina corrente (la zona di memoria di $2^9=512$ parole nella quale si trova il programma in esecuzione, quindi alla quale "punta" il PC).

L'ISA della CPU LC2 è il seguente:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD ⁺	0001				DR			SR1			0	00		SR2		
ADD ⁺	0001				DR			SR1			1	imm5				
AND ⁺	0101				DR			SR1			0	00		SR2		
AND ⁺	0101				DR			SR1			1	imm5				
BR	0000				n	z	p	pgoffset9								
JSR	0100				L	00		pgoffset9								
JSRR	1100				L	00		BaseR			index6					
LD ⁺	0010				DR			pgoffset9								
LDI ⁺	1010				DR			pgoffset9								
LDR ⁺	0110				DR			BaseR			index6					
LEA ⁺	1110				DR			pgoffset9								
NOT ⁺	1001				DR			SR			111111					
RET	1101				000000000000											
RTI ⁺	1000				000000000000											
ST	0011				SR			pgoffset9								
STI	1011				SR			pgoffset9								
STR	0111				SR			BaseR			index6					
TRAP	1111				0000				trapvect8							

Struttura del linguaggio Assembly

Label Opcode Operands ;comments

Label riferimento simbolico scelto dal programmatore per indicare l'indirizzo di memoria dell'istruzione.

Opcode codice mnemonico dell'istruzione (ADD, JSR, ...).

Operands riferimenti simbolici a registri o indirizzi di memoria.

comments testo libero di spiegazione del significato dell'istruzione.

Esempio:

loop: LDR R1, R0, #0 ;legge prossimo numero

Routine di I/O

Può essere utile scrivere programmi in Assembly LC-2 che interagiscano con l'operatore. Sono previste tre semplici routine di interazione, associate ad altrettante posizioni nel vettore di trap:

TRAP x21 emette su video il carattere il cui codice ASCII è contenuto in R0.

TRAP x23 legge un carattere da tastiera e ne riporta il codice ASCII in R0.

TRAP x25 (TRAP HALT) consente di arrestare l'esecuzione del programma dopo l'ultima istruzione utile.

Pseudo-istruzioni

La traduzione da linguaggio Assembly a codice macchina viene effettuata automaticamente da un opportuno programma (Assembler).Può essere utile fornire all'Assembler opportune direttive che guidino tale traduzione automatica. Le direttive si presentano come istruzioni macchina (da cui il nome pseudo-istruzioni) ma con l'Opcode preceduto dal carattere 'punto'.

.orig consente di segnalare all'Assembler da quale indirizzo di memoria caricare il programma.
.fill consente di inizializzare con un valore costante il contenuto di una cella di memoria.
.blkw consente di riservare un certo numero di celle di memoria (per es. per contenere variabili).
.stringz consente di inizializzare una sequenza di celle di memoria con la codifica ASCII di una frase (racchiusa fra "doppi apici"). Dopo l'ultimo carattere, viene inserito il terminatore 0.
.end segnala all'Assembler il termine del programma da tradurre.

Costanti

Per inserire valori costanti, si possono utilizzare le seguenti notazioni:

Binaria bBBBBBBBBBBBBBBBB 16 bit codificati come 0 o 1, preceduti dal carattere 'b'.

Esadecimale xEEEE 4 cifre esadecimali (0÷9, a, b, c, d, e, f minuscole o maiuscole) precedute dal simbolo 'x'.

Decimale #DDDDD fino a 5 cifre decimali, precedute dal carattere '#', eventualmente seguito dal segno '-'.

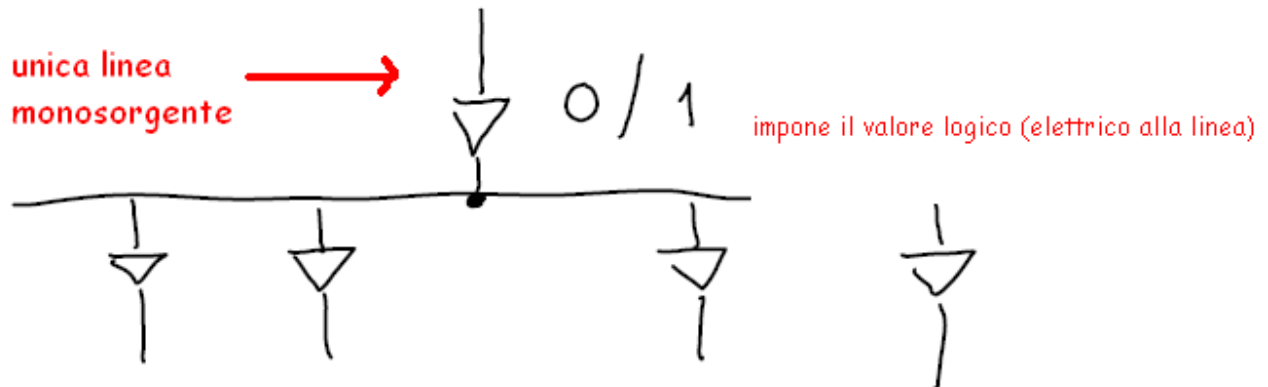
ESEMPIO:

	.orig	x3000	
	LEA	R0, table	;carica inizio tabella in punt.
	AND	R2, R2, #0	;azzerà totalizzatore
loop :	LDR	R1, R0, #0	;legge prossimo numero
	BRZ	finish	;se nullo ha finito
	ADD	R2, R2, R1	;somma a totalizzatore
	ADD	R0, R0, #1	;incrementa puntatore
	BRNZP	loop	;prossimo numero
finish	ST	R2, result	;scrive risultato in memoria
table	.blkw	#4	;vettore di numeri
	.fill	#0	
result	.blkw	#1	;risultato
	.end		

Modulo 7 – Architettura del calcolatore:

Tipi di linee di bus

Monosorgente: un solo dispositivo impone il valore logico alle linee, molti dispositivi lo acquisiscono (es. linee dell'Address Bus, un'uscita e tanti ingressi). Per poter pilotare linee monosorgente, i dispositivi elettronici che si interfacciano a queste linee (bistabili o porte logiche) devono avere lo stadio di uscita **TOTEM POLE:**



L'uscita può trovarsi in 2 stati:

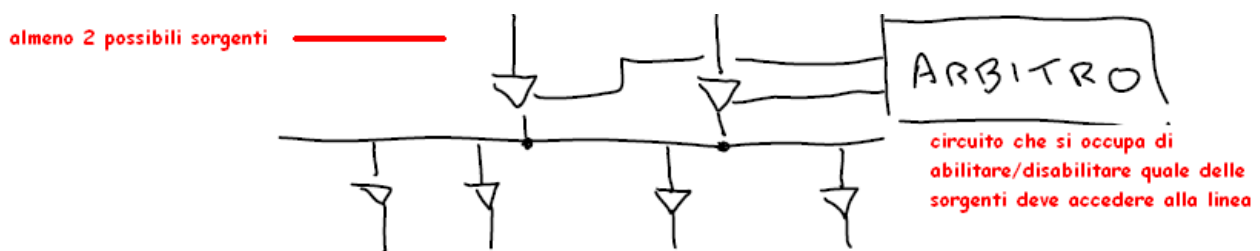
0 a bassa impedenza – lo stadio di uscita impone tensione zero cercando di assorbire la massima corrente possibile per tenere a zero la tensione

1 a bassa impedenza - tensione di alimentazione ed erogazione di corrente.

Il dispositivo sorgente pilota SEMPRE la linea cui è collegato, forza sempre un valore di tensione.

Multisorgente sincrono: a turno, diversi dispositivi impongono il valore alle linee (es. linee del Data Bus); è possibile decidere a chi tocca (sincronizzare l'accesso alle linee).

Serve interfacciare le sorgenti alla linea di bus mediante lo stadio di uscita **TRI-STATE:**



L'uscita può trovarsi in 3 stati:

-0 a bassa impedenza;

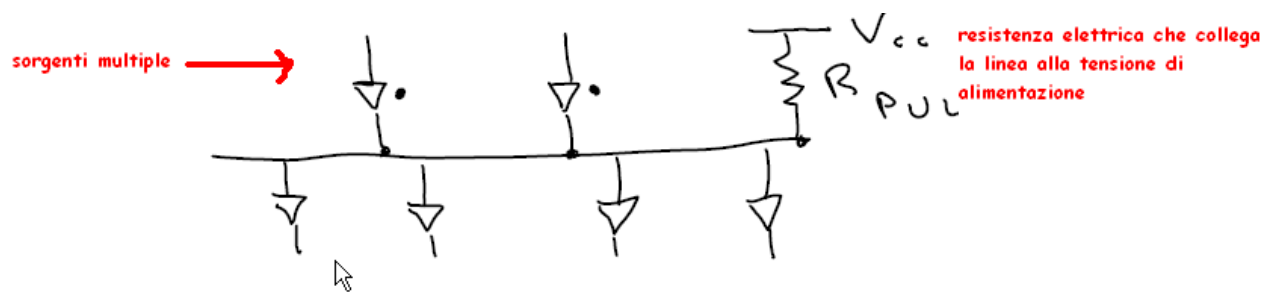
-1 a bassa impedenza;

-alta impedenza (Z).

Serve un "arbitro" per decidere a chi tocca pilotare la linea di uscita e che porta una delle due linee ad alta impedenza, di fatto disabilitandola (e abilitando l'altra a bassa impedenza).

Multisorgente asincrono: diversi dispositivi impongono il valore alle linee ma non è possibile sincronizzare i dispositivi.

Questo tipo di linee richiede un terzo tipo di stadio di uscita per le sorgenti collegate a tali linee, l'**OPEN COLLECTOR:**



L'uscita può trovarsi in 2 stati:

- 0 a bassa impedenza;
- alta impedenza (Z).

Il dispositivo pilota la linea a cui è collegato solo se vuole imporre il valore 0, se nessun dispositivo pilota la linea, serve la resistenza di pull-up per portare la linea a 1.

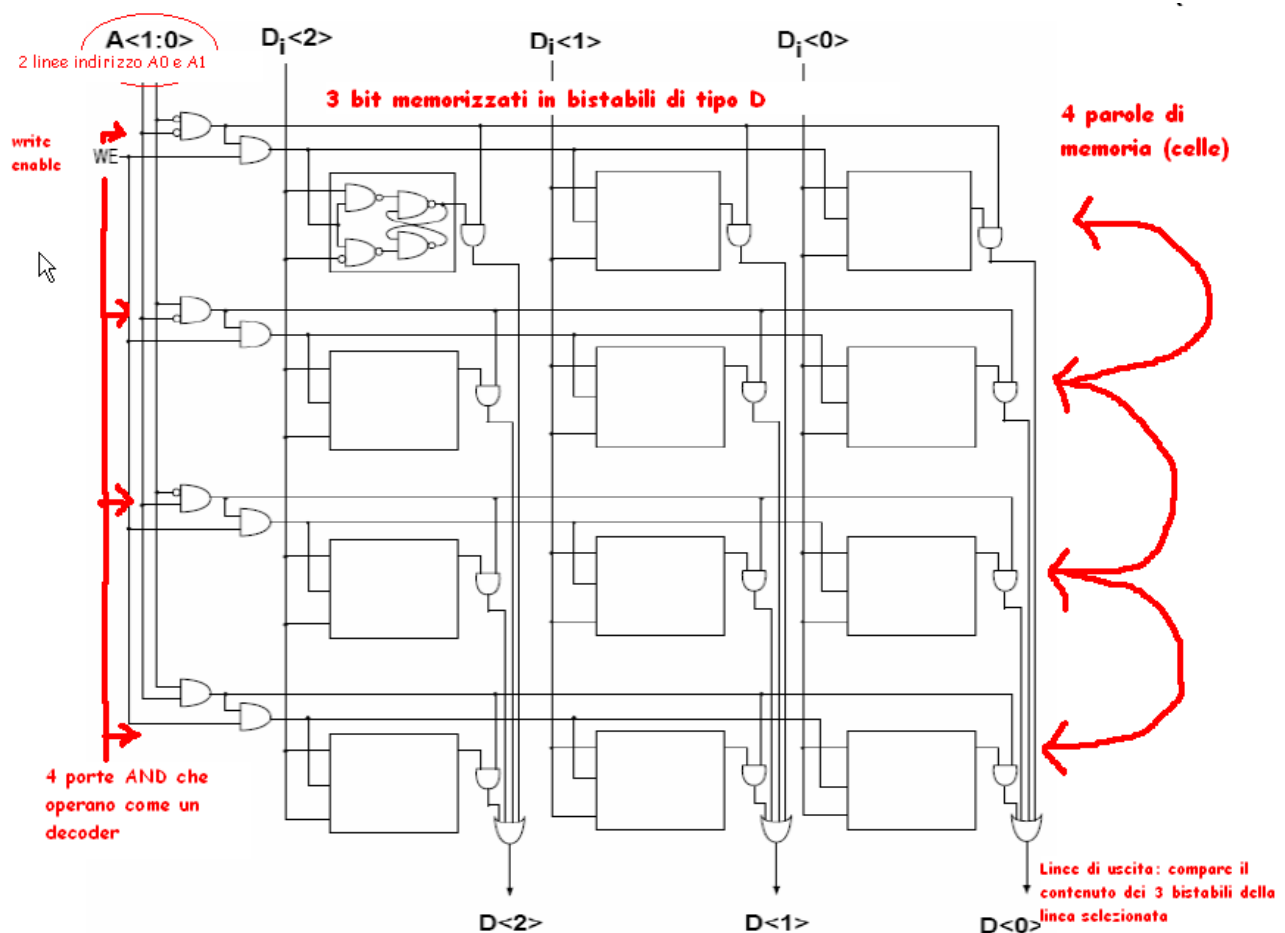
Memoria RAM

Alla CPU serve una memoria di lavoro elettronica (per avere tempi di risposta accettabili). Nella memoria di lavoro devono essere inseriti dati (variabili) e codice macchina del programma volta per volta necessario (variabile).

Serve una memoria a lettura e scrittura: RAM (Random Access Memory):

- non c'è rapporto causa-effetto tra un accesso a una cella e il successivo (ogni cella è ugualmente accessibile);
- due tipi di dispositivi: SRAM (Static RAM) e DRAM (Dynamic RAM)

Struttura della SRAM:



Struttura della DRAM

Si differenzia dalla SRAM perchè, per risparmiare spazio sul chip, i bistabili sono sostituiti da condensatori.

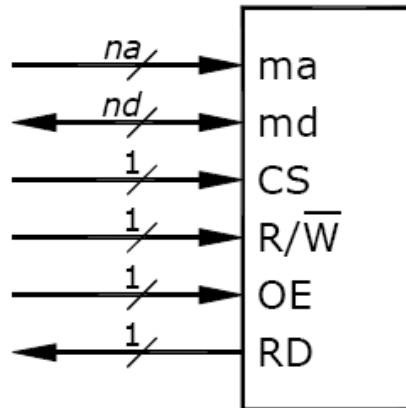
Il valore del bit è associato alla carica presente sul condensatore (carico bit a 1, scarico bit a zero).

Se il bit vale 1, dopo un certo tempo, il condensatore si scarica e "perde" l'informazione.

È necessaria una attività periodica di refresh (fatta da circuiti esterni) che ripristini la carica sui condensatori.

Con questa soluzione si riescono a ottenere elevatissime densità di memoria: chip da 1 Gbit.

Un generico CHIP di RAM si presenta così:



- na piedini di indirizzo monodirezionali (ma) entranti nel chip (il numero di piedini dipende dal numero di celle presenti nel chip)
- nd piedini di dato bidirezionali (md) il numero dipende dalla larghezza di parola di memoria
- linea di input *chip select* (CS) dice al chip di essere coinvolto nell'operazione
- linea di input *read/write* (R/W) dice al chip se l'operazione è una lettura o una scrittura
- eventuale linea di input *output enable* (OE) abilita in momenti ritardati i TRI State di uscita
- eventuale linea di output *ready* (RD) uscente dal componente che dice se il componente è pronto o ha bisogno di tempo per completare l'operazione.

Memoria ROM

Serve comunque, oltre la RAM, una memoria a sola lettura ROM (Read Only Memory) che mantenga il proprio contenuto anche in assenza di alimentazione:

- programma da eseguire all'accensione del calcolatore (fase di bootstrap);
- situazioni nelle quali il programma da eseguire è sempre lo stesso (applicazioni embedded).

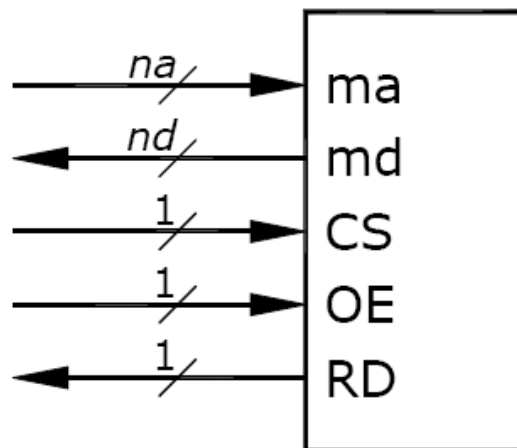
D: Perché una memoria ROM ha la stessa sigla di un componente combinatorio?

- Gli ingressi della ROM possono essere considerati gli indirizzi della cella di ROM che vogliamo leggere
- Le linee di prodotto possono essere considerate linee di parola (una per ogni configurazione degli indirizzi). A ogni configurazione degli ingressi si accende un mintermine che è una delle parole della memoria ROM
- Le linee di somma possono essere considerate linee di bit (uno per ogni uscita della ROM). Ci danno ciascuna un bit della parola di ROM indirizzata dalla configurazione degli ingressi.
- A ogni configurazione degli ingressi, appare in uscita ciò che viene programmato nella sezione OR come "parola" di m bit.

D: La stessa cosa può essere fatta con componenti di tipo PLA o PAL?

- Solo una sezione AND completa consente questo tipo di funzionamento cioè una sezione AND per la quale a qualsiasi configurazione degli ingressi si accenda una diversa linea prodotto. Quindi né una PLA né una PAL possono andare bene come memorie, poichè per loro la sezione AND non è completa e non c'è una linea prodotto per ogni configurazione degli ingressi.

Un generico CHIP di ROM si presenta così:



- na piedini di indirizzamento monodirezionali (ma);
- nd piedini di dato monodirezionali (md);
- linea di input *chip select* (CS);
- eventuale linea di input *output enable* (OE);
- eventuale linea di output *ready* (RD).

Le differenze non sono negli indirizzi ma nei piedini di dato che sono linee monodirezionali (il calcolatore può solo leggere e non scrivere).

Realizzazione di un banco di memoria

E' un insieme di chip di memoria, che "riempie" una porzione dello spazio di indirizzamento della CPU considerata, quindi un insieme di componenti in grado di fornire all'unità centrale una certa quantità di memoria fisica accessibile mediante il BUS.

Ogni "cella di memoria" del banco deve avere un numero di bit pari al numero di linee del Data Bus della CPU in modo che ad ogni accesso a memoria l'UC possa scambiare in parallelo tutti i bit che il proprio data bus è in grado di scambiare.

Il numero di celle di memoria del banco è tipicamente una potenza di 2 in modo tale che un certo numero di fili del bus indirizzi della CPU possa essere utilizzato per identificare una cella all'interno dell'insieme di celle indirizzabili con tali fili del bus indirizzi

Naturalmente, il banco deve "apparire" come una sequenza di celle adiacenti in una determinata posizione dello spazio di indirizzamento della CPU. Il banco di memoria fisica deve riempire una fetta di celle consecutive nell'intero spazio di indirizzamento della CPU (non necessariamente l'intero spazio di memoria fisica).

Esempio di banco di memoria per CPU LC-2 (address bus 16 linee e Data bus a 16 linee):

Partiamo da chip di RAM da **1K** (1024 celle) **x8** (8 bit ciascuna, 1byte):

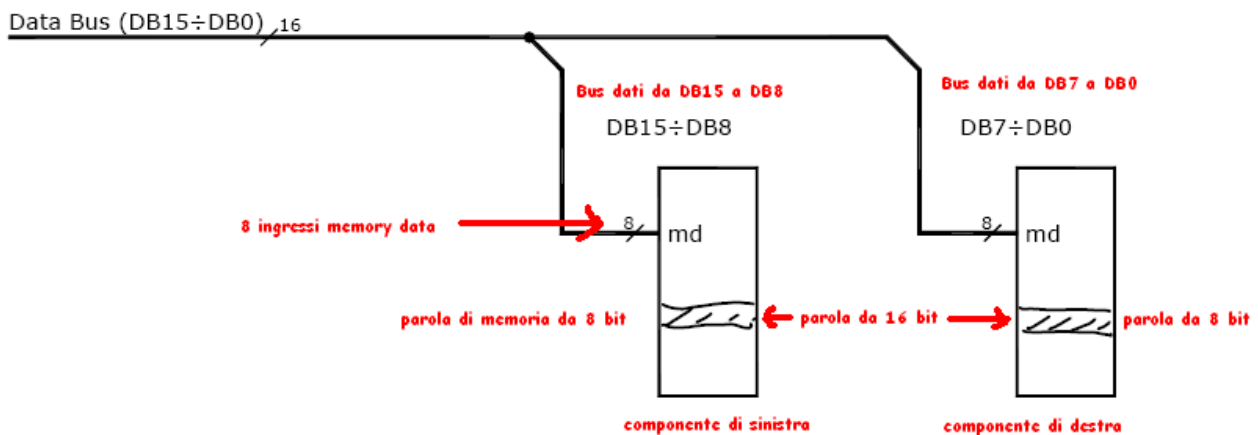
Vogliamo realizzare un banco di memoria RAM da 4K celle (4096 celle) da 16 bit a partire dall'indirizzo 0000 0000 0000 0000

Come realizzare celle da 16 bit (perchè partiamo da chip a 8 bit):

Ogni componente di RAM ha celle da 8 bit ma la CPU LC-2 pretende di "vedere" celle da 16 bit grandi come il suo data bus.

Possiamo ottenere il risultato accoppiando idealmente per riga 2 componenti: il componente di sinistra contiene gli 8 bit più significativi della cella di memoria LC-2 il componente di destra contiene gli 8 bit meno significativi della cella di memoria LC-2.

Il Data Bus LC-2 viene quindi diviso in due parti: gli 8 bit più significativi vengono collegati al componente di sinistra, gli 8 bit meno significativi al componente di destra.



Come realizzare un banco da 4K (perchè partiamo da un chip a 1K):

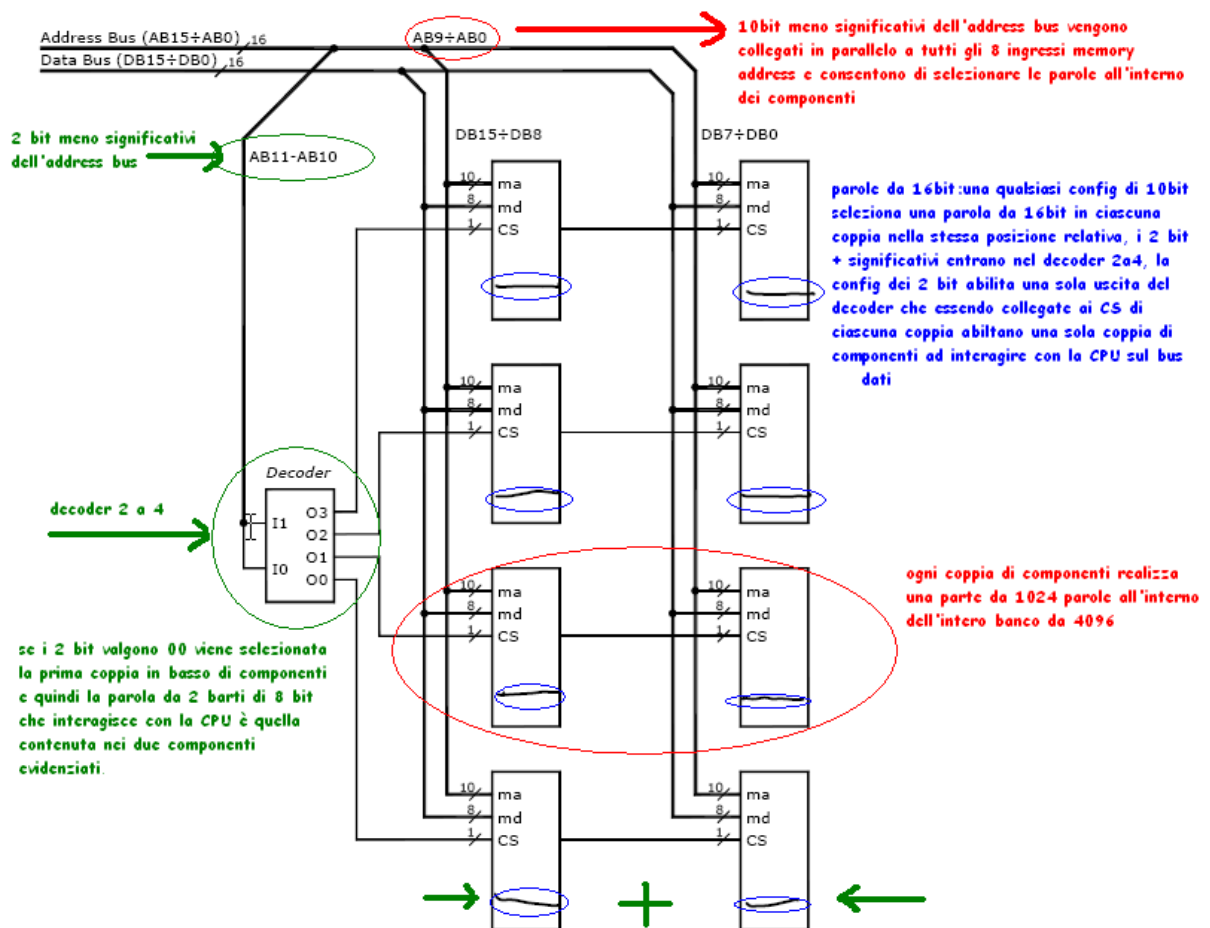
Ogni coppia di componenti contiene 1K (1024) celle, individuate da una configurazione di 10 bit dell'Address Bus (dal momento che 2 elevato alla 10 fa appunto 1024).

Per realizzare 4K (4096) celle servono quindi 4 coppie di componenti a cui l'unità centrale potrà accedere generando una qualunque configurazione di 12 bit dell'Address Bus (2 alla 12 = 4096).

Come colleghiamo questi 12 bit dell'address BUS al banco di memoria?

I 10 bit meno significativi dell'Address Bus (AB0÷AB9) vengono collegati in parallelo a tutte le coppie di componenti in modo tale che una qualunque configurazione di questi 10 bit selezioni una parola in ciascuna coppia di questi componenti (parola che ha la stessa posizione in ciascuna coppia) e per scegliere quale delle 4, (una per ogni coppia di componenti) possiamo usare i 2 bit successivi (AB10,AB11) che devono appunto selezionare una coppia di componenti.

Per far questo serve un decoder collegato ai Chip Select dei componenti, che riceva in ingresso i 2 bit di address bus AB10 e AB11 e che generi le selezioni della singola coppia di componenti, cioè abbia le sue 4 uscite, collegate agli ingressi CS delle singole coppie di componenti:



Come posizionare il banco da 4K:

Abbiamo usato solo 12 dei 16 bit dell'Address Bus (10 per selezionare le parole e 2 per selezionare i componenti)

I 4 bit rimasti dell'address BUS, i più significativi, ci dicono a quale delle 2 alla 4=16 "pagine" o banchi da 4K (4096) celle ci stiamo riferendo nello spazio di indirizzamento da 64K celle della CPU LC-2.

Serve inserire una selezione di banco (tra i 16 possibili) per far sì che il decoder venga abilitato solo se la configurazione di 4 bit AB15÷AB12 assume la configurazione associata al banco, abilitando così il Chip Select del decoder e, solo in questo caso, una riga del banco di memoria viene abilitata a interagire con la CPU.

Il segnale r/W è 1, il banco è impostato in lettura
 $MEMR = 1$ $MEMW = 0$
La porta OR, con gli ingressi negati, produce 1, il blocco RAM è abilitato.
Il segnale r/W è 0, il banco è impostato in scrittura

- Il segnale ready va propagato alla CPU, per consentirle di sapere quando la memoria è pronta per trasferire il dato.
- In caso si realizzi un banco di memoria ROM, l'unica differenza è l'assenza del segnale di lettura/scrittura (dalla ROM si può solo leggere...).

Collegamento CPU-interfacce di I/O

Le interfacce di I/O sono dispositivi visibili alla CPU come "celle dedicate": alcune "celle" trasferiscono dati altre "celle" danno comandi alla periferica o ne riportano lo stato.

Ci sono 2 modalità di collegamento tra le interfacce di I/O e la CPU:

Interfacce memory mapped: le "celle dedicate" (cioè questi registri contenuti nell'interfaccia IO) rispondono a indirizzi di memoria generati dall'unità centrale, si possono usare le normali istruzioni macchina per accedere alle interfacce I/O, saranno diversi gli indirizzi da specificare. Nell'UNICO spazio di indirizzamento della CPU alcuni indirizzi saranno occupati da vere celle di memoria altri dai registri delle interfacce. Lo spazio di indirizzamento rimane "bucato", si creano zone nelle quali non possiamo mettere memoria perchè sono occupate dalle interfacce.

Interfacce I/O mapped: alcune CPU usano uno spazio di indirizzamento diverso, doppio, uno per la memoria ed uno per le interfacce di I/O al quale l'unità centrale accede con istruzioni macchina dedicate (IN/OUT).

Sincronizzazione CPU-interfacce di I/O

Ogni operazione di I/O implica la sincronizzazione fra due riferimenti temporali:

- Il clock della CPU (tempo interno);
- L' "orologio" dei fenomeni esterni, che agiscono sulla periferica collegata all'interfaccia.

A seconda delle diverse frequenze dei due riferimenti temporali, dell'urgenza dei fenomeni e della loro natura, esistono **tre modalità fondamentali di gestione(sincronizzazione)**:

1) controllo di programma (vince il clock):

La CPU esegue le operazioni di I/O quando il programma in esecuzione interagisce con l'interfaccia (è sotto il controllo del programma in esecuzione decidere quando andare a valutare, leggendo i registri dell'interfaccia, se fenomeni esterni si sono verificati o no). Per certi versi questa modalità è concettualmente "sbagliata" (dovrebbero essere i fenomeni esterni a dettare i tempi di lavoro non il calcolatore – Tolomeo- a decidere quando farlo). Questa modalità funziona grazie alla differenza fra clock della CPU e orologio delle periferiche (clock estremamente veloce, GHz e orologio normalmente molto più lento Hz÷kHz).

Varianti del controllo di programma:

In caso di più periferiche collegate al calcolatore, si può decidere a livello di programma con quale frequenza interrogare l'interfaccia e, per ciascuna interfaccia, quale priorità assegnarle. La seconda variante invece riguarda i cicli di attesa: anzichè sprecare tempo non trovando la periferica pronta e attendendola, si può sospendere l'interrogazione dell'interfaccia e fare altro. Serve però un Sistema Operativo multitasking che gestisca la ripartizione del tempo di CPU fra attività (processi o task) differenti.

Limiti del controllo di programma:

Impone al mondo esterno i ritmi della CPU (quindi dei programmi in esecuzione) in una visione tolemaica del calcolatore. Non è adatto per fenomeni urgenti: se la CPU è impegnata in attività onerose, può trascurare troppo a lungo la periferica. Non è adatto nemmeno per fenomeni che si ripetono a elevata frequenza: ogni operazione di I/O è comunque svolta dalla CPU, che per scoprire cosa fare, deve comunque acquisire da memoria mediante fasi di fetch le istruzioni macchina da eseguire.

In pratica, la **modalità di controllo di programma è molto semplice MA:**

- impone al mondo esterno i ritmi interni;

- non è adeguata per fenomeni urgenti;
- non è adeguata per fenomeni ripetitivi ad alta frequenza.

2) interrupt (vince l'orologio):

La CPU esegue le operazioni di I/O quando l'interfaccia lo richiede. Consente quindi alla periferica di segnalare la necessità di servizio alla CPU ma per farlo serve una linea dedicata del bus di controllo: **INTREQ** (Interrupt Request), ed è una linea "**attiva bassa**" che quindi segnala la necessità di interruzione PASSANDO a ZERO.

E' concettualmente più "corretta" (sono i fenomeni esterni a dettare i tempi di lavoro, visione Copernicana) e richiede che l'interfaccia possa "interrompere" la CPU nello svolgimento delle sue attività per dedicarsi all'evento che l'ha interrotta.

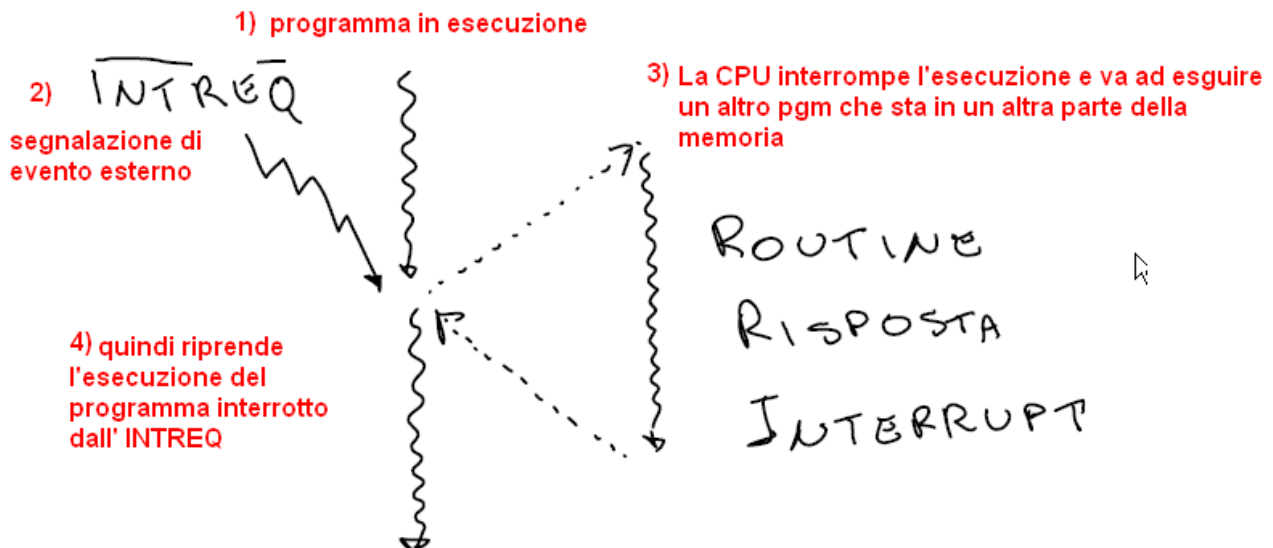
E' particolarmente adatta a gestire fenomeni "urgenti", che non possono attendere il tempo (forzatamente casuale) di interrogazione di una soluzione a controllo di programma (la differenza fra clock e orologio non è più tale da garantire il corretto funzionamento, quindi il controllo di programma non va più bene ma l'Interrupt si!).

L'interrupt però:

non risolve il problema dei fenomeni che si ripetono ad alta frequenza

Struttura dell'interrupt

- Richiesta di interruzione da parte della periferica (linea INTREQ del bus di controllo).
- Riconoscimento da parte della CPU, al termine dell'istruzione in corso prima di procedere al fetch della successiva, si chiede se c'è stata una richiesta di Interrupt (risposta affermativa mediante linea del bus di controllo INTACK) e segnala all'interfaccia interrompente di aver riconosciuto la richiesta di interruzione.
- Salvataggio automatico da parte della CPU del PC attuale e salto alla routine di risposta all'interrupt (cioè chiamata a sottoprogramma generata dall'hardware).
- Routine di risposta, che si deve preoccupare di salvare il contesto del programma interrotto.
- Al termine della risposta all'interrupt, ripristino del contesto (a cura della routine di risposta) e ritorno alla posizione salvata del programma interrotto.



Linea INTREQ

In un calcolatore possono esistere più periferiche che richiedono gestione a interrupt e non c'è possibilità di sincronizzazione fra le richieste di interrupt: ogni interfaccia a periferica interrompe quando la propria periferica lo richiede.

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

- chi vuole interrompere forza uno 0 a bassa impedenza la linea;
- normalmente la linea è tenuta a 1 dalla resistenza di pull-up (in assenza di richieste di interruzione la linea a INTREQ è tenuta ad uno dalla resistenza di pull-up. Se almeno una, ma anche n delle periferiche che possono interrompere decidono di farlo, da uno ad n delle

rispettive porte passano a zero a bassa impedenza trascinando a zero attivo basso la linea Interrupt request).

Rimangono aperti questi problemi:

- in caso di interrupt simultaneo da più periferiche, come decidere a quale periferica dare retta
- come riconoscere, nel caso di più periferiche interrompenti, quale periferica ha effettivamente richiesto l'interruzione
- come decidere se e quando è opportuno (periferica critica o non critica) che la CPU possa essere interrotta.

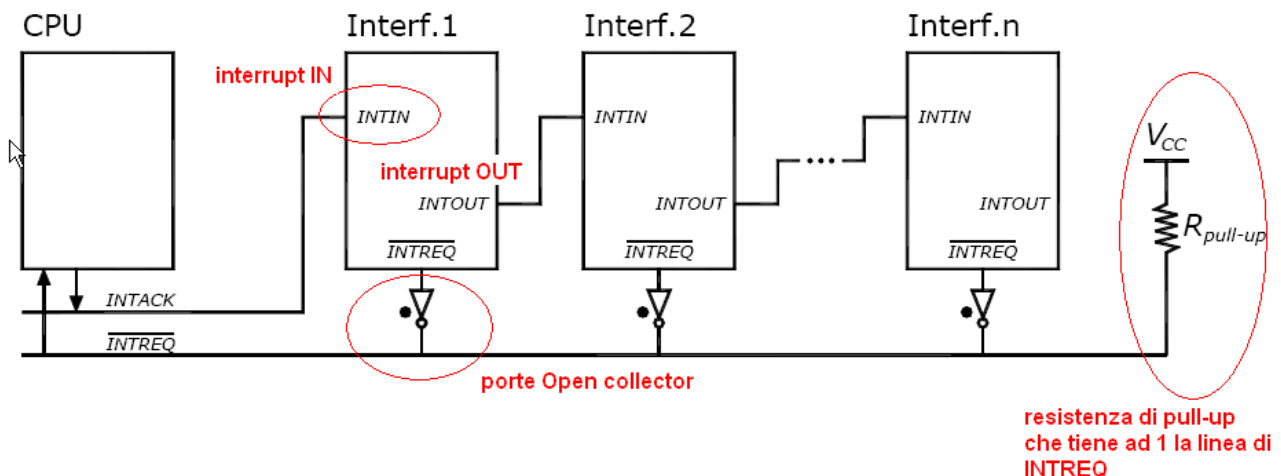
- Soluzioni -

Interrupt Cablato: la CPU è dotata delle 2 linee INTREQ e INTACK.

Alla ricezione di un interrupt, dopo aver terminato l'istruzione corrente e prima di passare alla successiva, la CPU si accorge che c'è una richiesta di interrupt dall'esterno:

- salva il valore del PC;
- disabilita il riconoscimento di ulteriori interrupt;
- attiva la linea INTACK;
- salta a un indirizzo di risposta predefinito.

La routine di risposta deve effettuare **il polling** delle interfacce che possono essere sorgente di interruzione: ogni interfaccia deve essere interrogata per sapere se è la responsabile dell'interruzione (l'ordine di interrogazione è associato all'urgenza della periferica).



Quando una 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 priorità è associata alla posizione fisica dell'interfaccia rispetto alla CPU:

le interfacce più vicine hanno priorità maggiore e la priorità non è modificabile run time.

Se una periferica ha già propagato INTOUT non può interrompere (deve attendere fine ciclo).

Se la CPU viene riabilitata a sentire gli interrupt, una periferica ad alta priorità può essere interrotta da una a bassa.

L'interrupt cablato è relativamente semplice, **ma presenta i seguenti limiti:**

- il polling rallenta l'inizio delle attività di risposta vere e proprie (in una fase in cui l'urgenza è fondamentale);
- la priorità delle periferiche è fissa perché dipende dalla loro posizione rispetto alla CPU;
- se la CPU non viene riabilitata agli interrupt, rimane "cieca" fino alla fine del servizio della periferica interrompente;
- se la CPU viene riabilitata agli interrupt, può essere interrotta da una periferica meno urgente di quella in servizio.

Interrupt Vettorizzato: La CPU è dotata delle 2 linee INTREQ e INTACK.

Alla ricezione di un interrupt la CPU finita l'esecuzione in corso:

- salva il valore del PC;
- disabilita il riconoscimento di ulteriori interrupt;
- attiva INTACK;
- (differenza con Int. Cablato) attende sul Data Bus la comparsa di un identificativo a 8 bit - inserito dall'interfaccia a periferica - che serve alla CPU (simile all'istruzione TRAP) come indice in un vettore di interrupt:
 - il vettore di Interrupt è tabella di celle di memoria, una cella associata a ogni possibile sorgente di interrupt (es. 8bit di id fanno $2^{8}=256$ possibili sorgenti id interrupt);
 - ogni cella contiene l'indirizzo di inizio della routine di risposta all'interrupt associato.

☺ Il principale vantaggio è che il tempo di riconoscimento della sorgente di interrupt è minimizzato, non serve il polling, ed è sufficiente che la CPU usi l'id che riceve sul data bus per andare a prendere sulla tabella del vettore di interrupt l'indirizzo di inizio della routine di risposta dell'interfaccia che si è identificata.

MA:

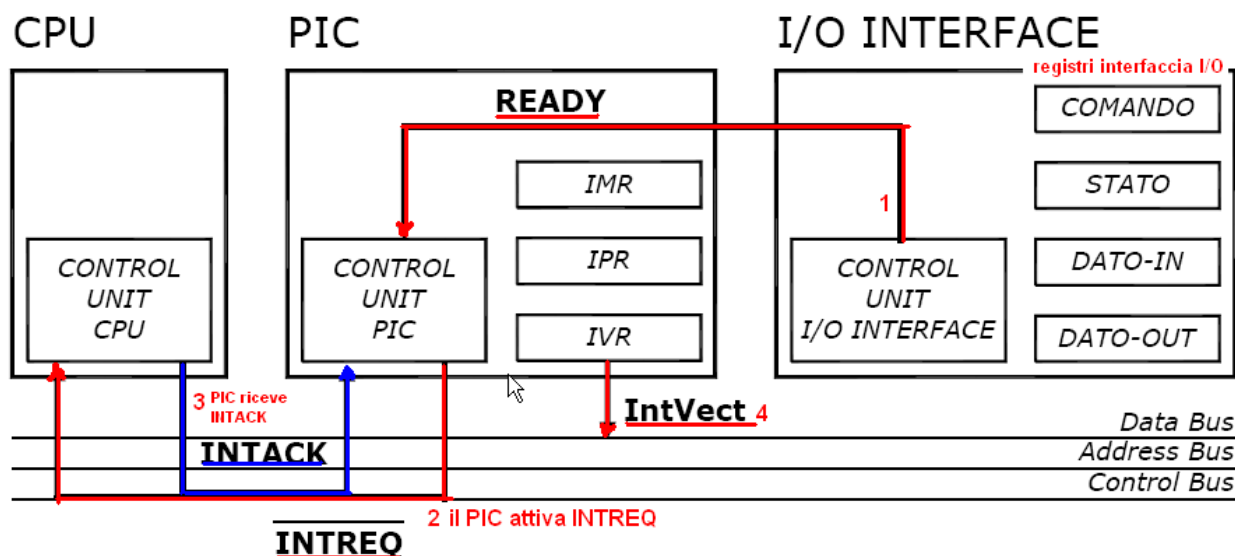
☹ Le interfacce a periferica si complicano: ogni interfaccia deve essere capace di generare il proprio identificativo sul Data Bus quindi deve "sapere" il proprio identificativo, mettere nell'interfaccia un ID fisso non è possibile.

☹ Non abbiamo risolto i problemi di priorità: se la CPU viene riabilitata a sentire gli interrupt, può essere interrotta da chiunque anche da periferiche a priorità o urgenza inferiore.

Per risolvere questi problemi inseriamo nel sistema un dispositivo denominato PIC che è un circuito integrato di supporto alla gestione degli interrupt.

PIC (Programmable Interrupt Controller):

- L'interfaccia comunica al PIC la richiesta di interrupt.
- Se l'interfaccia è abilitata, il PIC attiva INTREQ.
- Quando riceve INTACK, il PIC comunica l'identificativo della periferica sul Data Bus:
 - gli identificativi delle periferiche sono gestiti dal PIC, senza aggiunta di complessità per le interfacce.
- Se riceve più richieste di interrupt da diverse periferiche, dà precedenza a quella più prioritaria.



I registri del PIC, accessibili alla CPU come normali registri di interfaccia (componente programmabile):

IVR Interrupt Vector Register contiene l'identificativo associato a ciascuna periferica collegata (tabella di byte) ci sarà quindi spazio per un byte per ogni periferica collegabile al PIC (fino a 8 periferiche di solito).

IPR Interrupt Priority Register contiene le informazioni necessarie per stabilire l'ordine di priorità delle periferiche (siccome è scrivibile dalla CPU in ogni momento, le periferiche hanno

priorità che possono variare durante lo svolgimento di un programma senza nessuna modifica circuitale)

IMR Interrupt Mask Register contiene le informazioni per sapere quali periferiche possono generare interruzioni e quali no (in modo dinamico).

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

- **il riconoscimento della sorgente di interrupt è rapido** perchè, senza che l'interfaccia a periferica debba essere più complessa del normale, grazie al PIC arriva alla CPU l'indicazione immediata di quale routine di risposta all'interrupt iniziare.
- **la priorità delle periferiche è dinamicamente variabile** e posso decidere in base alla situazione esterna quale è la posizione relativa nell'ordine delle priorità delle periferiche interrompenti
- prima che la CPU venga riabilitata agli interrupt, **si può decidere quali periferiche potranno interrompere quella in servizio** riprogrammando il registro IMR e quali no.

3) Direct Memory Access o DMA (i 2 riferimenti temporali restano indipendenti):

In alcuni casi, una operazione di I/O è costituita da molti passi singoli che si ripetono ad alta frequenza, per esempio il trasferimento di interi settori (sequenze di byte nell'ordine ad es. di 1024) da/verso memoria di massa, o la trasmissione/ricezione di un frame da rete.

L'interfaccia esegue autonomamente le operazioni di I/O, e avvisa la CPU solo a lavoro finito evitandole l'onere di occuparsi del trasferimento in "prima persona".

Anche qualora sia completamente dedicata a questo tipo di operazioni di I/O, la CPU è comunque penalizzata perché:

- essendo un componente general purpose cioè in grado di fare tante cose, deve scoprire (fetch di istruzioni macchina) passo passo cosa le si chiede di fare;
- per trasferire un singolo byte dell'intera tabella da periferica a memoria o viceversa, il singolo accesso utile (trasferimento del dato) è penalizzato da un elevato numero di accessi "inutili":
 - fetch delle istruzioni che dicono alla CPU di trasferire un singolo byte
 - incremento del puntatore all'area di memoria da/in cui trasferire i dati dopo ogni byte trasferito
 - aggiornamento del contatore di dati trasferiti più altre variabili di servizio

Questa tecnica quindi, (Direct Memory Access) prevede la possibilità che altri dispositivi - oltre alla CPU - possano accedere a Memoria occupandosi loro di effettuare il trasferimento dei singoli byte: questo significa che altri dispositivi possano diventare quindi temporaneamente Master del bus.

Per far questo, serve poter richiedere alla CPU la possibilità di utilizzare il bus e chiedere alla CPU di smettere di essere lei il Master del sistema. A questo scopo serve la linea dedicata del bus di controllo: HOLDREQ.

Anche gli stadi di uscita della CPU che pilotano le linee dell'Address Bus e del Control Bus devono essere stadi TRI STATE (per lasciare agli altri Master la possibilità di pilotarle).

Linea HOLDREQ

E' una linea "attiva bassa" come la INTREQ vista in precedenza, poichè in un calcolatore possono esistere più periferiche che richiedono DMA (quindi uso del BUS), non c'è possibilità di sincronizzazione fra le richieste di DMA quindi, per non avere conflitti elettrici, ogni gestore di DMA chiede il bus quando la propria periferica deve trasferire un dato.

La linea di richiesta HOLDREQ deve essere gestita mediante porte OPEN COLLECTOR:

- linea attiva bassa;
- chi vuole l'uso del bus forza a 0 a bassa impedenza la linea;
- se nessuno vuole l'uso del bus, normalmente la linea è tenuta a 1 dalla resistenza di pull-up

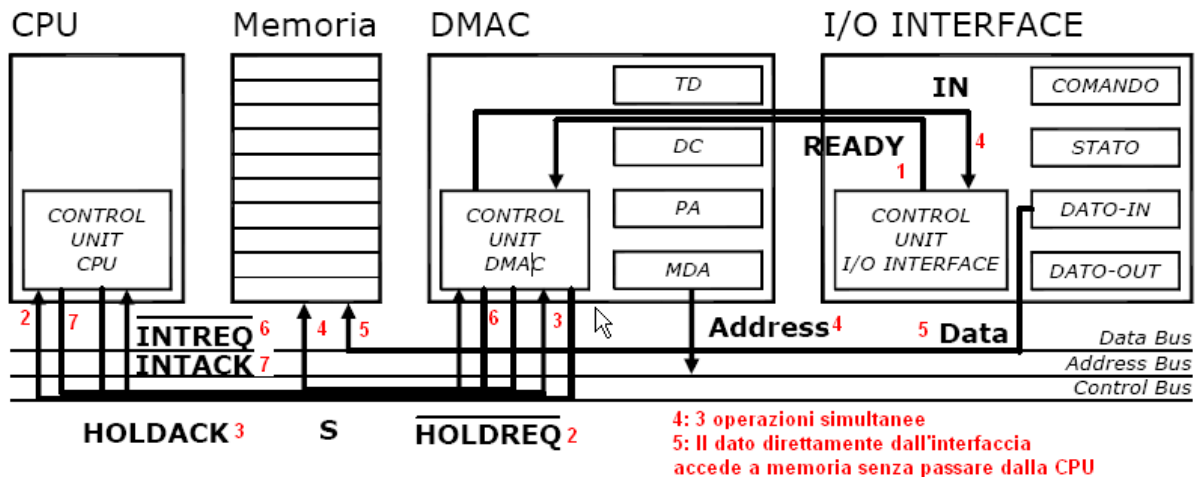
DMAC (DMA Controller) Integrato di supporto alla gestione del DMA (come per l'interrupt vettorizzato il PIC)

La CPU programma il DMAC comunicando:

- indirizzo della zona di memoria da/in cui trasferire i dati (prima delle celle che devono contenere la tabella di dati);
- Il numero di dati da trasferire;
- L'identificativo della periferica, fra tutte quelle che possono trasferire, e senso di trasferimento (in/out)

Quando la periferica segnala di essere pronta:

- il DMAC richiede i bus con il segnale HOLDREQ;
- quando la CPU ne ha terminato l'eventuale uso in corso, ne segnala il rilascio con HOLDACK;
- il DMAC effettua il trasferimento e aggiorna puntatori e contatori;
- finito l'intero trasferimento, segnala al software che tutta l'operazione è finita generando l'interrupt.



Come per il PIC anche **per il DMAC esistono dei registri** accessibili alla CPU come normali registri di interfaccia (componente programmabile):

PA Peripheral Address: contiene l'identificativo dell'interfaccia a periferica con cui interagire per scambiare i dati nel caso in cui possano essere servite dallo stesso DMAC più periferiche.

MDA Memory Data Address: contiene l'indirizzo della prossima cella di memoria in cui inserire il byte in ingresso o da cui prelevare il dato in uscita.

DC Data Counter: contiene il numero di dati ancora da trasferire e che viene inizializzato alla lunghezza della tabella dei dati

TD Transfer Direction: indica se l'operazione è una lettura (IN) o una scrittura (OUT).

La tecnica di I/O mediante DMA ha le seguenti caratteristiche:

- consente di trasferire dati da/verso la memoria sotto il controllo di un componente diverso dalla CPU;
- tale componente - il DMAC - ha il vantaggio di essere realizzato a questo scopo, e non perde tempo per scoprirlo da programma;
- il trasferimento avviene in modo "trasparente" al programma in esecuzione sulla CPU che viene semplicemente rallentata perché deve occasionalmente rilasciare i bus (non c'è percezione a livello sw di un'operazione che si svolge a livello HW)
- al termine dell'intera attività, il programma che aveva richiesto I/O viene avvisato con interrupt.

Modulo 8 - Struttura della CPU:

La CPU NS-0 (didattica)

Macchina RISC (Reduced Instruction Set Computer):

- pochissime istruzioni macchina: opcode 2 bit quindi $2^{2}=4$ istruzioni.
- istruzioni macchina NON tutte di uguale lunghezza;
- pochissimi modi di indirizzamento: immediato e diretto.

Macchina a 16 bit:

data bus a 16 bit quindi celle di memoria da 16 bit;

- address bus a 14 bit quindi $2^{14}=16K$ celle di spazio di indirizzamento.

GPR (general purpose register): 1 registro a 16 bit, denominato R0

CC con solo eventuale valore nullo (Z) dell'ultimo risultato di un'operazione di somma da parte dell'ALU.

ALU solo le operazioni strettamente indispensabili: ADD

Set di istruzioni:

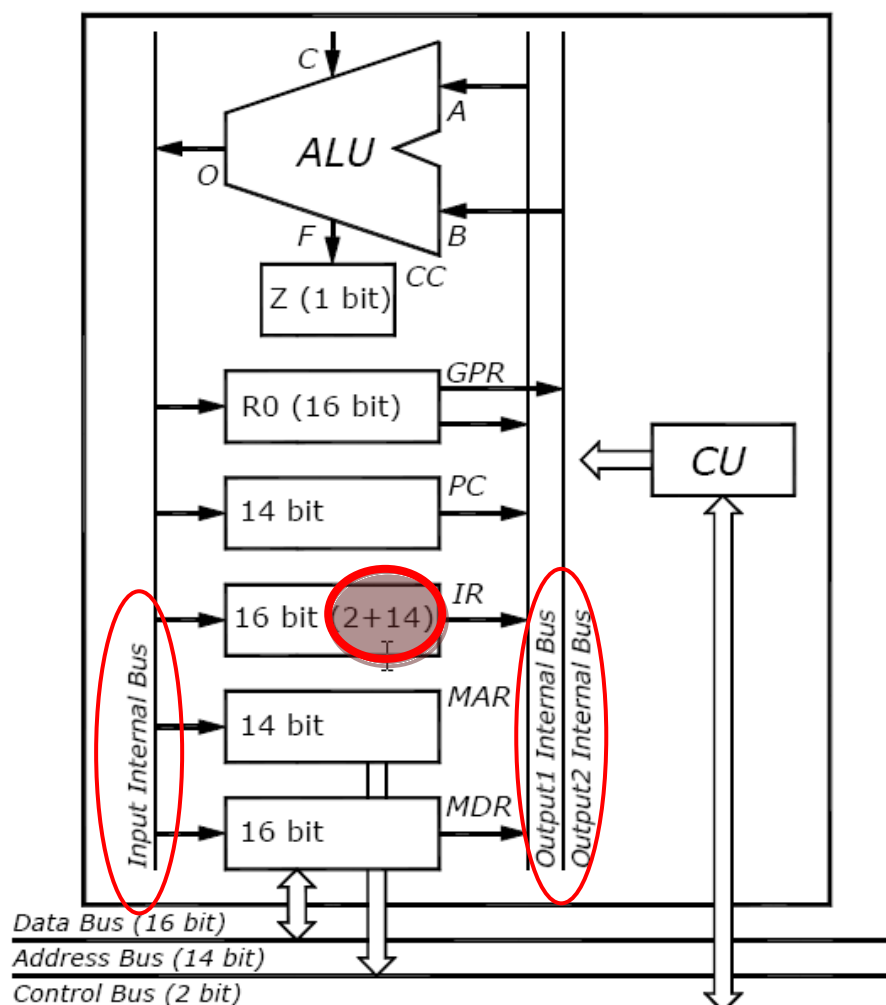
LOAD

STORE

ADD

BRZ

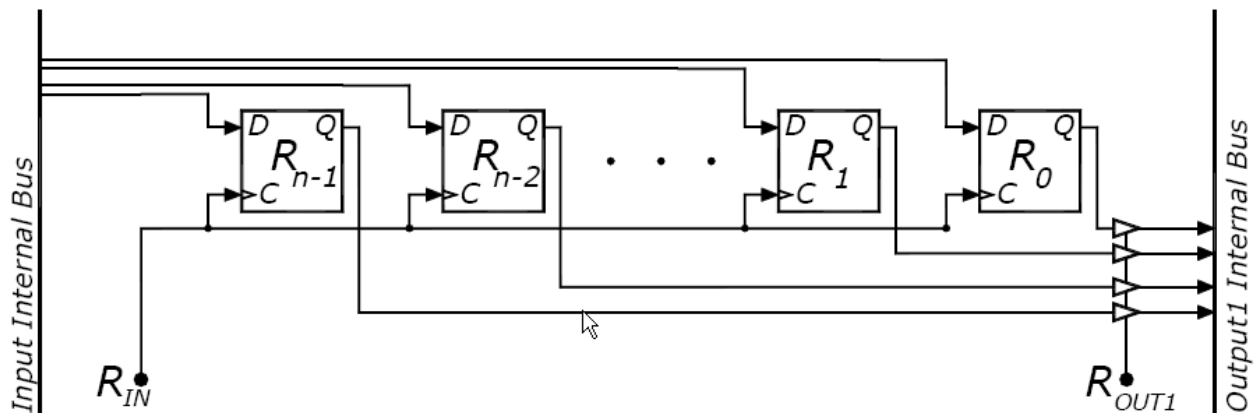
Struttura interna CPU NS0:



Bus Interni all'unità centrale: 2 bus di output e il bus di input che collegano in ingresso ai registri (input) e in uscita (output) gli elementi della CPU e che vedono un collegamento in direzione opposta tramite l'ALU. Sono questi bus che ci consentono di far girare i dati tramite i registri.

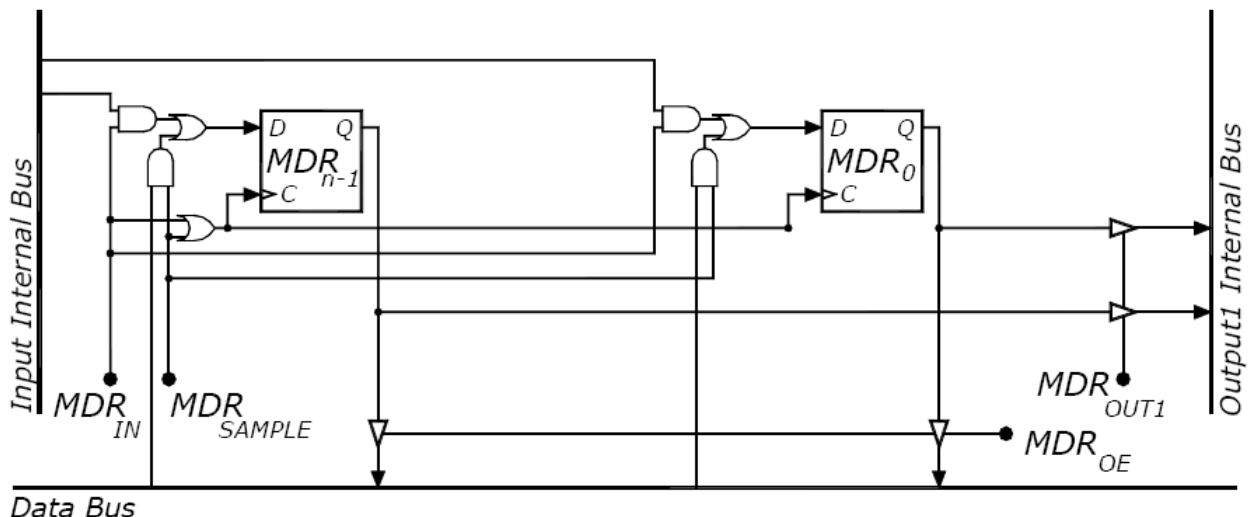
Registri della CPU NS0:

Registro generico (n bistabili di tipo D tanti quanti sono i bit del registro), ha un comando (registro IN) che da un colpo di abilitazione a tutti i bistabili e fa sì che quello che c'è nel bus interno di input venga campionato nel registro (scrive il contenuto del bus interno) ed un comando (registro OUT1 o OUT2) che disabilita o abilita i buffer TRI STATE che consentono al contenuto del registro di essere messo sul buffer di uscita o di restare nel registro perché le uscite sono ad alta impedenza.



Memory data register

L'unico registro un po' diverso è il registro di interfaccia al bus dati esterno che può ricevere informazioni da campionare sia dal bus interno di input sia dal data bus e può emettere sia sul bus interno di input sia sul data bus.



Scopo degli Internal Bus: il Data Path

Un registro può emettere il suo contenuto sugli Output Internal Bus cui è collegato e campionare il valore presente sull'Input Internal Bus.

Se l'ALU (collegata in senso contrario) è capace di propagare ciò che si presenta ad uno dei suoi ingressi A o B in uscita O le informazioni possono circolare fra i registri della CPU:

Comando ALU (C) Significato

MSB	LSB	
0	0	NOP
0	1	PASS (O=A)
1	0	INC (O=A+1)
1	1	ADD (O=A+B) (if O=0: 1→F)

Abbiamo definito il DATA PATH:

tre Internal Bus per far circolare i valori presenti nei registri della CPU, l'ALU per collegare i due Internal Bus di Output all'Internal Bus di Input ed i comandi per estrarre il contenuto di

ciascun registro o per modificare il contenuto di ciascun registro: comandi per emettere il contenuto di MAR sull'Address Bus e per scambiare il contenuto di MDR con il Data Bus; comandi MEMR e MEMW del Control Bus per accedere a memoria.

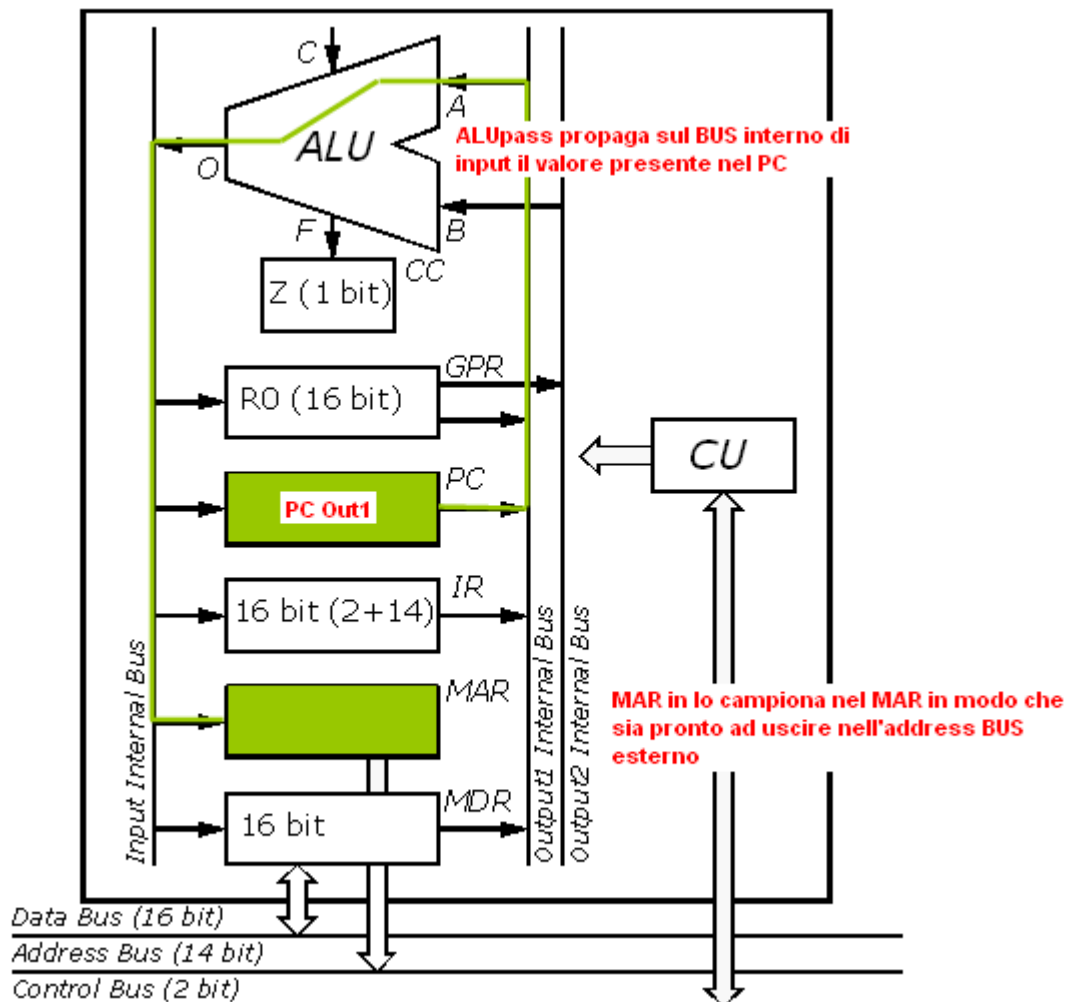
Per muovere i dati lungo il Data Path, oltre ai comandi all'ALU ci servono i seguenti comandi per i registri:

Z_{SAMPLE}
R0_{IN} , R0_{OUT1} , R0_{OUT2}
PC_{IN} , PC_{OUT1}
IR_{IN} , IR_{OUT1}
MAR_{IN} , MAR_{OE}
MDR_{IN} , MDR_{OUT1} , MDR_{SAMPLE} , MDR_{OE}

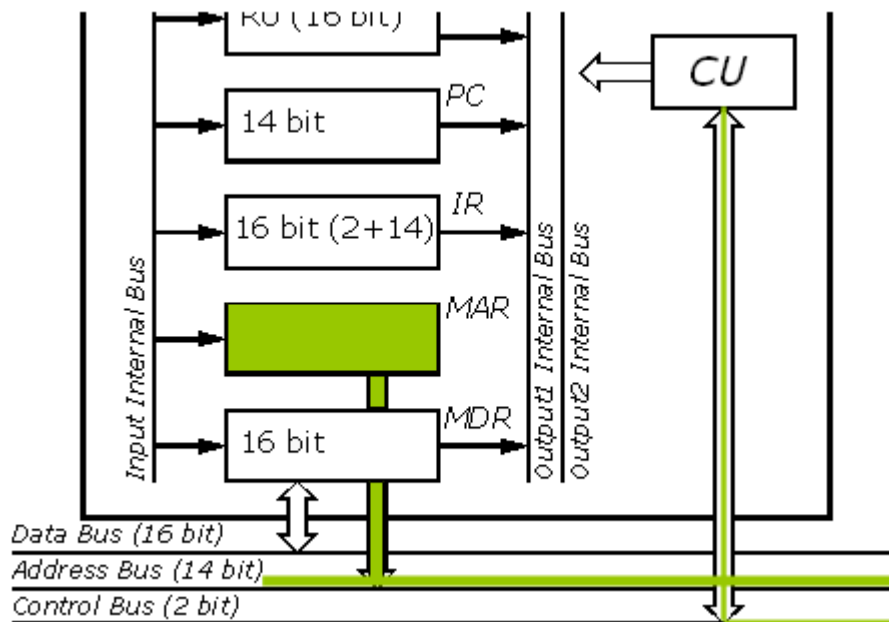
il Control Path

I comandi precedenti "orchestrano" i trasferimenti di dati (Data Path) nella CPU NS-0, i seguenti controllano tali trasferimenti (Control Path):

step	comandi
s0	PCOUT1 , ALUPASS , MARIN

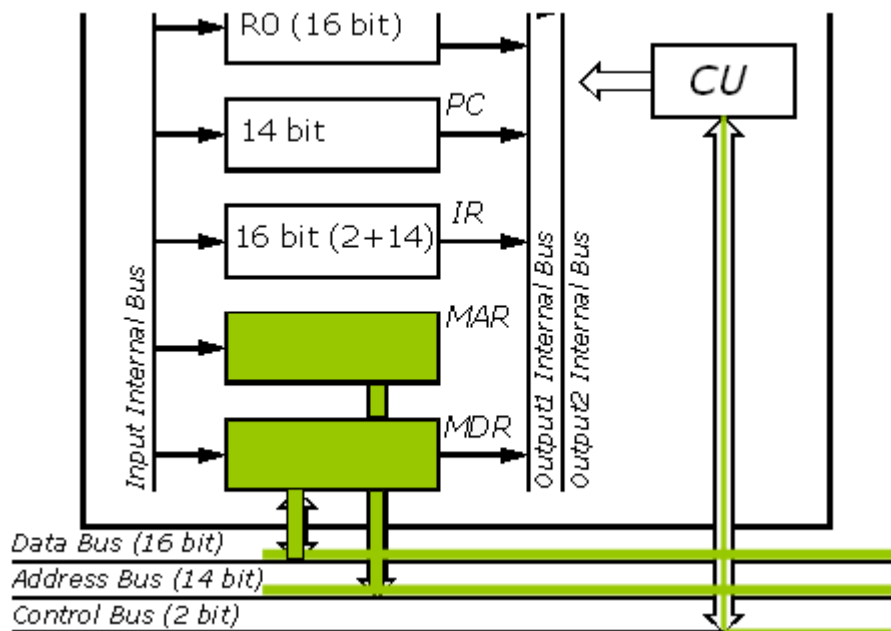


s1	MAROE , MEMR , ALUNOP
----	-----------------------



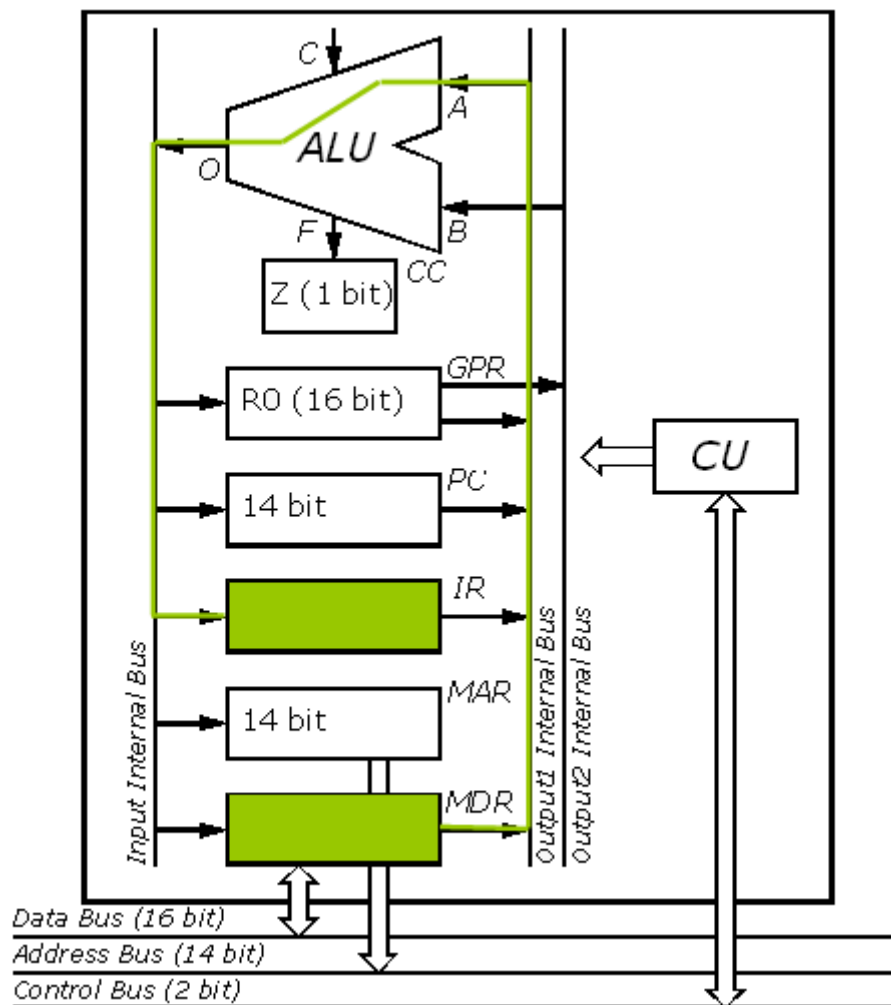
s2

MAROE , MEMR , MDRSAMPLE , ALUNOP



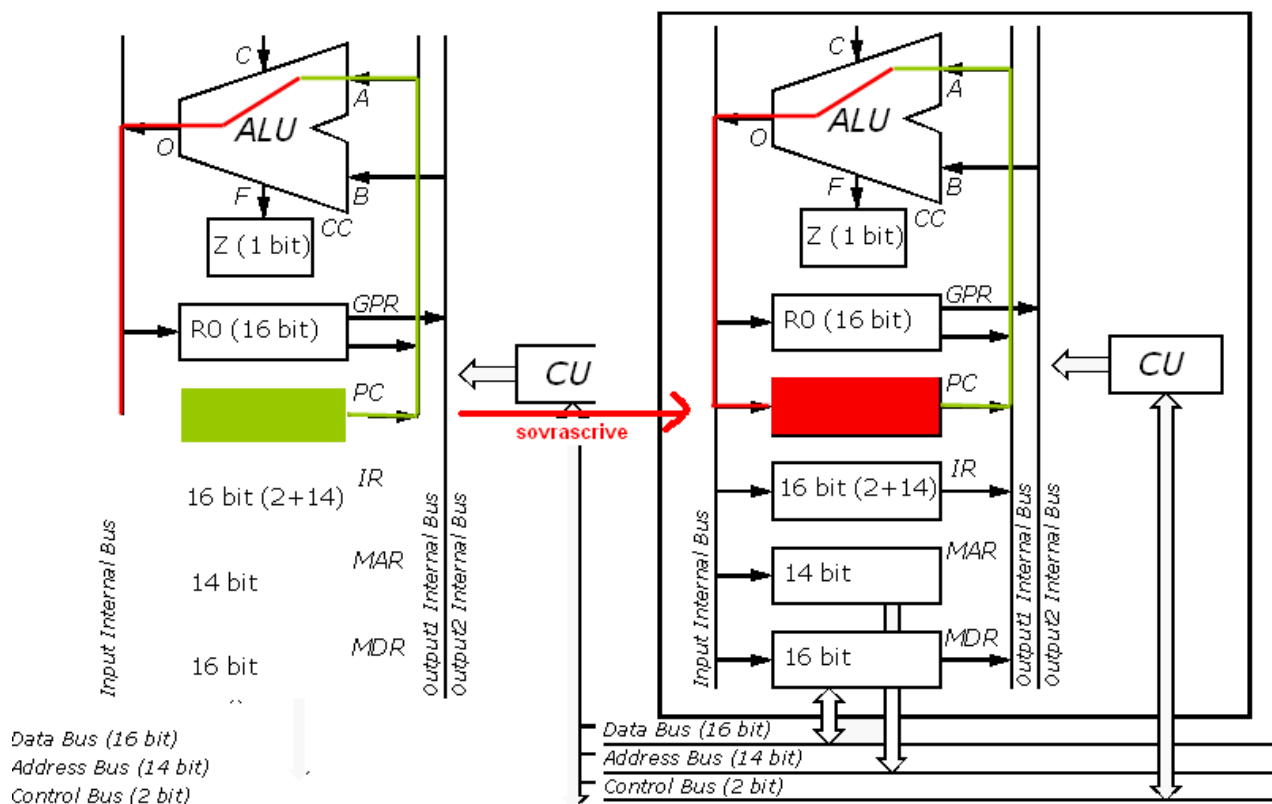
s3

MDROUT1 , ALUPASS , IRIN



s4

PCOUT1 , ALUINC , PCIN



Fase di decodifica: In base all'opcode acquisito nella fase di fetch (i due bit più significativi del registro IR) si decide quale tra le 4 istruzioni macchina dell'ISA NS-0 deve essere eseguita.

In pratica, il **control path**, è una sequenza di comandi all'ALU, ai registri, al Control Bus per svolgere le varie fasi di esecuzione di una istruzione macchina;
 la fase di fetch è comune a tutte le istruzioni, e usa i suddetti comandi;
 la fase di decodifica richiede di scegliere quali passi svolgere in funzione dell'istruzione acquisita durante il fetch; la fase di esecuzione usa di nuovo gli stessi comandi visti sopra.

La Control Unit cablata (approccio hardware)

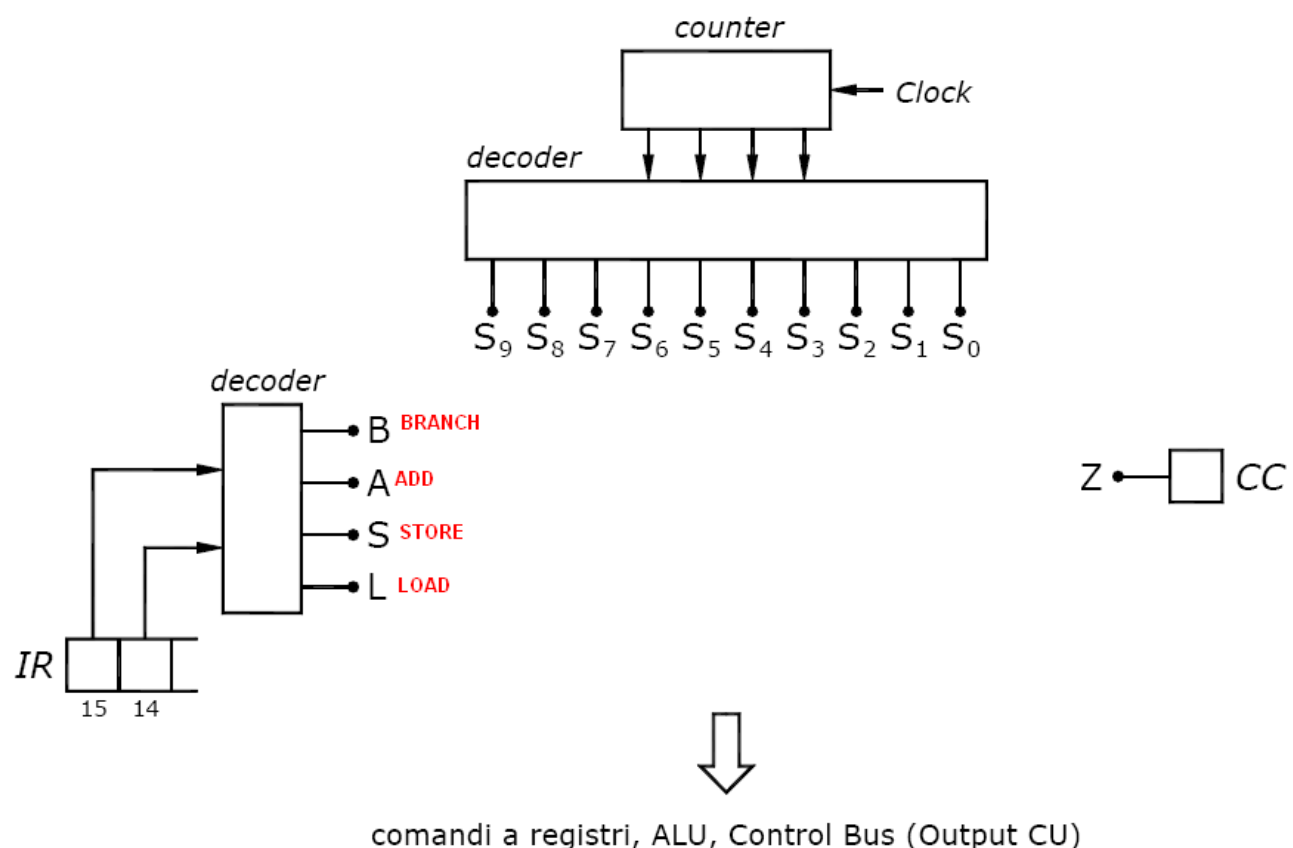
I/O della CU:

Input

- opcode dell'istruzione acquisita durante il fetch (L, S, A, B);
- situazione del registro CC (Z on/off);
- stato di avanzamento (n° dello step da eseguire).

Output

- comandi ai registri;
- comandi all'ALU;
- comandi al Control Bus.



Nella realizzazione della CU cablata si segue un **approccio hardware**: nel control path ogni Output è attivo in determinati step di esecuzione.

Costruiamo una espressione logica (combinatoria) degli Output come funzione degli Input elencati in precedenza e realizziamo il circuito come Logic Array (una PLA che però non richiede di essere programmata, ma realizzata ad hoc, quindi senza i problemi di scarsa velocità legati ai fusibili).

Caratteristiche della CU cablata

- ⊗ Una struttura particolarmente efficiente (la rete combinatoria logic array è il modo più rapido per generare le uscite a partire dagli ingressi)
- ⊗ Assicura la massima velocità di esecuzione
- ⊗ Particolarmente adatta a CPU RISC: motivo del successo iniziale dell'approccio RISC (poche istruzioni perché la complessità del logic array è dominabile)

- ⊗ Di difficile modifica (evoluzioni dell'architettura richiedono la riprogettazione completa)
- ⊗ Di difficile utilizzabilità per macchine CISC, per l'esplosione della complessità del Logic Array (il Logic Array tende ad esplodere esponenzialmente con il crescere delle uscite).

⊗ Solo recentemente, grazie all'evoluzione tecnologica, è applicabile a CU complesse.

L'approccio HW quindi si rivela ottimo per CPU RISC, dove il ridotto set di istruzioni implica una CU semplice, meno per CPU CISC, per le quali la CU cablata diventa molto più complessa.

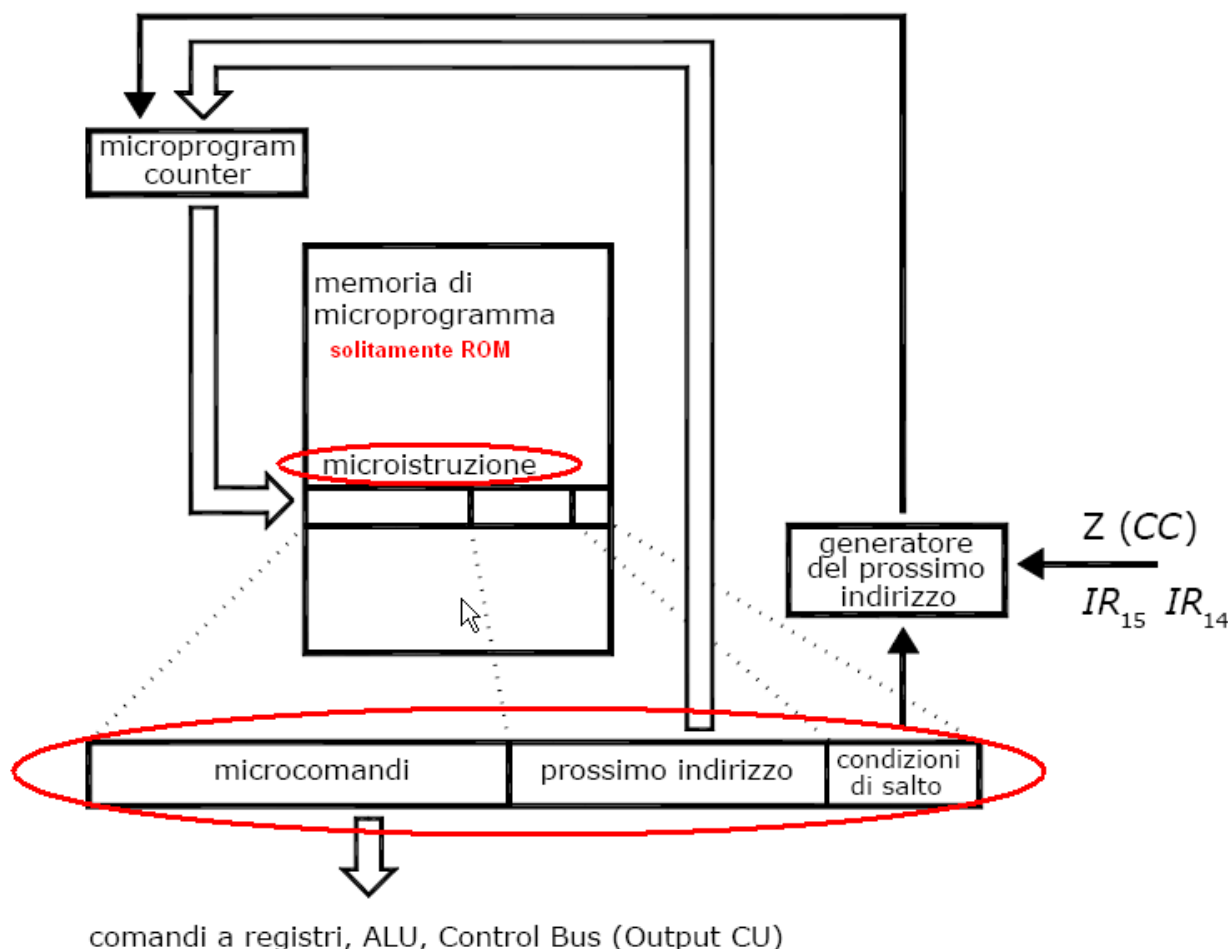
La Control Unit microprogrammata (approccio software)

Input

- opcode dell'istruzione acquisita durante il fetch (L, S, A, B)
- situazione del registro CC (Z on/off).

Output

- comandi ai registri;
- comandi all'ALU;
- comandi al Control Bus.



In questo tipo di control unit quindi abbiamo un **approccio software**: ogni Output è attivo in determinate microistruzioni del microprogramma che guida il comportamento della CU.

Costruiamo una memoria di microprogramma (tipicamente una ROM) che contiene le sequenze di passi.

Per ramificare il flusso di esecuzione del microprogramma (fase di decodifica, salti condizionati, ecc.) inseriamo una logica di salto che decide la prossima microistruzione da eseguire.

		micro address		comandi uscite della control unit																				segnali per decidere la destinazione delle microistruzioni			
	mA	MAR		MDR				IR		PC		R0				Z	ALU		MEM		J	JNZ	J15	J14	mA		
		IN	OE	IN	OUT1	SAM	OE	IN	OUT1	IN	OUT1	IN	OUT1	OUT2	SAM	MSB	LSB	R	W								
FETCH	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0			
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	3	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0			
	4	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	10		
LOAD	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	15			
	6	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0			
	7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	8	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	9	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0		
STORE	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	20			
	11	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0			
	12	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0			
	13	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	14	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0		
ADD	15	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0			
	16	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	17	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0			
	18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0			
	19	0	0	0	1	0	0	0	0	0	0	1	0	1	1	1	1	1	0	0	1	0	0	0	0		
BRZ	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		
	21	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0		

Caratteristiche della CU microprogrammata

- ☺ Una struttura particolarmente regolare: la ROM cambia le dimensioni al variare della complessità del microprogramma ma rimane molto regolare dal punto di vista circuitale.
- ☺ Particolarmente adatta a CU complesse (CISC) con tecnologia hardware non evoluta da consentire CU cablate estremamente sofisticate.
- ☺ Sfrutta le tecniche di progettazione software (micro assembly language)
- ☺ Molto utilizzata negli anni 1980.

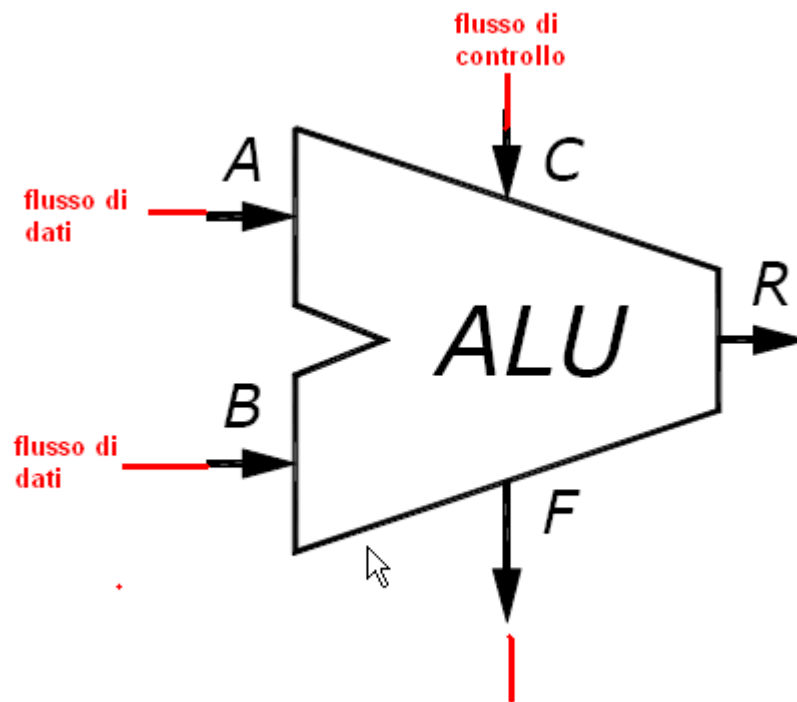
MA:

- ☺ Problemi di velocità di esecuzione dovuti alla presenza della memoria di microprogramma (sempre necessario ciclo di lettura del prossimo passo).
- ☺ Soppiantata da un approccio cablati quando la tecnologia di integrazione lo consente

Compiti dell'ALU

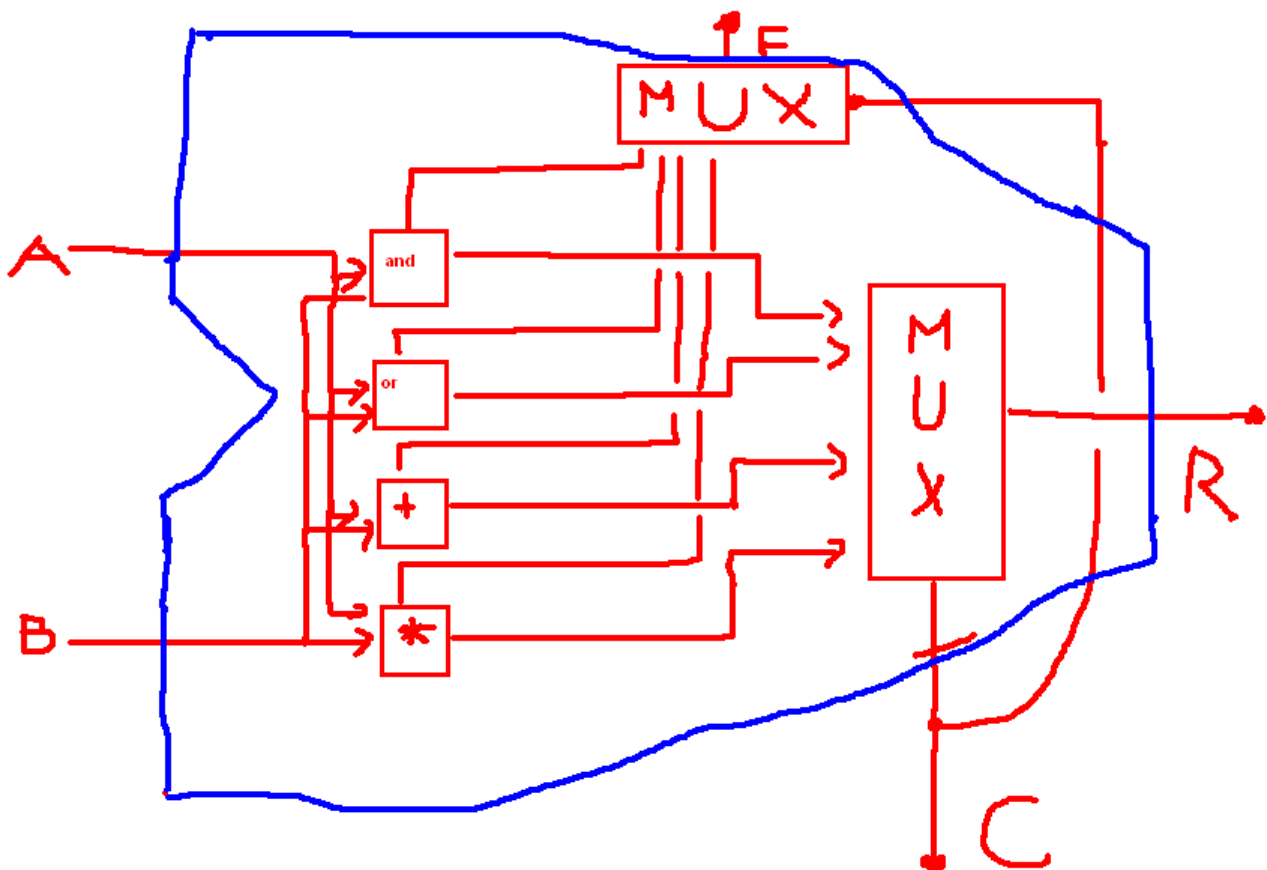
L'ALU (unità aritmetico logica) deve:

- 1) provvedere a chiudere il Data Path fra i vari bus interni della CPU per consentire i trasferimenti di informazioni tra i vari registri
- 2) svolgere operazioni logiche sugli operandi A e B: AND bit a bit; OR bit a bit; NOT bit a bit.
- 3) svolgere operazioni aritmetiche sui numeri A e B: confronti, somme (e sottrazioni) in complemento a due, a volte moltiplicazioni e divisioni (alu sofisticate)



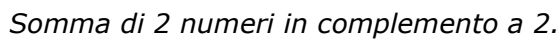
- A e B: ingressi operandi e rappresentano il tipico parallelismo della CPU, ovvero un numero di fili pari al numero di registri di lavoro dell'unità centrale
- R: risultato sempre parallelismo della CPU quindi stesso numero di bit dei registri dell'unità centrale
- C: comando (2C = numero di operazioni disponibili)
- F: flag di esito (una per ogni condizione verificata).

Struttura interna dell'ALU



Al suo interno può avere, in alcune sue componenti (es. moltiplicatore), un comportamento sequenziale ma all'esterno presenta sempre comunque un comportamento combinatorio. L'ALU è una rete combinatoria, o con comportamento esterno combinatorio, che effettua le reali elaborazioni (trasformazioni) di informazioni nel calcolatore. Opera in base al comando ricevuto, e produce un risultato e alcune flag di indicazione dell'esito dell'operazione (segno del risultato, errori aritmetici, ecc.).

Somma di 2 numeri a n bit:



Modulo 9 - Principali linee di evoluzione architeturale:

Funzionamento della memoria CACHE

Principio di località:









se all'istante t la CPU genera l'indirizzo di memoria $xNNNN$, è molto probabile che nell'immediato futuro generi di nuovo lo stesso indirizzo $xNNNN$ o indirizzi vicini (cioè "locali") all'indirizzo citato $xNNNN$. Ci sono 2 motivazioni per questo fenomeno:

- una località detta **spaziale**: il fetch delle istruzioni procede in celle consecutive, ma non solo, i programmi sono organizzati in moduli, con le variabili del singolo modulo memorizzate vicine.
- una località detta **temporale**: l'essenza della programmazione sono i **cicli**, gruppi di istruzioni e variabili usate nei cicli vengono "ripassate", riutilizzate ripetutamente

Sfruttiamo quindi il **principio di località** e lavoriamo su base statistica.

Quando la CPU genera un indirizzo di memoria, portiamo il contenuto della cella richiesta dalla CPU e un certo numero di celle vicine (che definiamo **blocco**) in una memoria, ovviamente a lettura e scrittura, più veloce della DRAM ma ovviamente più piccola, perché più costosa da realizzare.

Chiamiamo questa memoria **cache** (contenitore di una copia dei blocchi di celle di memoria di lavoro che circondano la cella richiesta dalla CPU in un dato accesso a memoria) derivata dal francese *caché* (nascosto) perché la sua esistenza non è nota né al programmatore (che ha in mente lo spazio di indirizzamento della CPU) né alla CPU, serve quindi solo a velocizzare gli accessi a memoria.

Processore	Anno	Costo	MIPS iniziali	MIPS massimi	n° transistor
8086	1978	-	0.33	 0.75	29 K
286	1982	\$ 8	1.20	 2.66	 134 K
386	1985	\$ 91	5.00	 11.40	 275 K
486	1989	\$ 317	20.00	 54.00	 1.2 M
Pentium	1993	\$ 900		112.00	 3.1 M

Ogni 4 anni una nuova generazione, lo stesso processore raddoppia le proprie prestazioni: e tra una generazione e l'altra, triplica la complessità.

Come usiamo i transistori in più? Per esempio per realizzare una memoria cache a bordo del processore, che lavora alla sua stessa frequenza di clock:

-cache L1 (di primo livello) - qualche KB.

La frequenza del processore, però, è cresciuta ancora, e la sua differenza rispetto alla DRAM si è enfatizzata:

-cache L2 (di secondo livello) esterna al processore - qualche centinaio di KB.

E se le cose degenerano...:

-cache L3 (di terzo livello) esterna al processore - qualche decina di MB!

Riassumendo, grazie alla località degli accessi a memoria da parte della CPU possiamo giocare di statistica e copiare in una memoria ad alte prestazioni (cache) le celle richieste, che hanno maggiore probabilità di essere usate di nuovo a breve.

Possiamo anche creare una gerarchia di cache via via più grandi e più lente man mano che ci allontaniamo dalla CPU e ci avviciniamo alla memoria di lavoro;

Le celle con più alta probabilità di riutilizzo sono ricopiate nella cache a bordo della CPU (L1); tutte le celle disponibili sono presenti in memoria di lavoro.

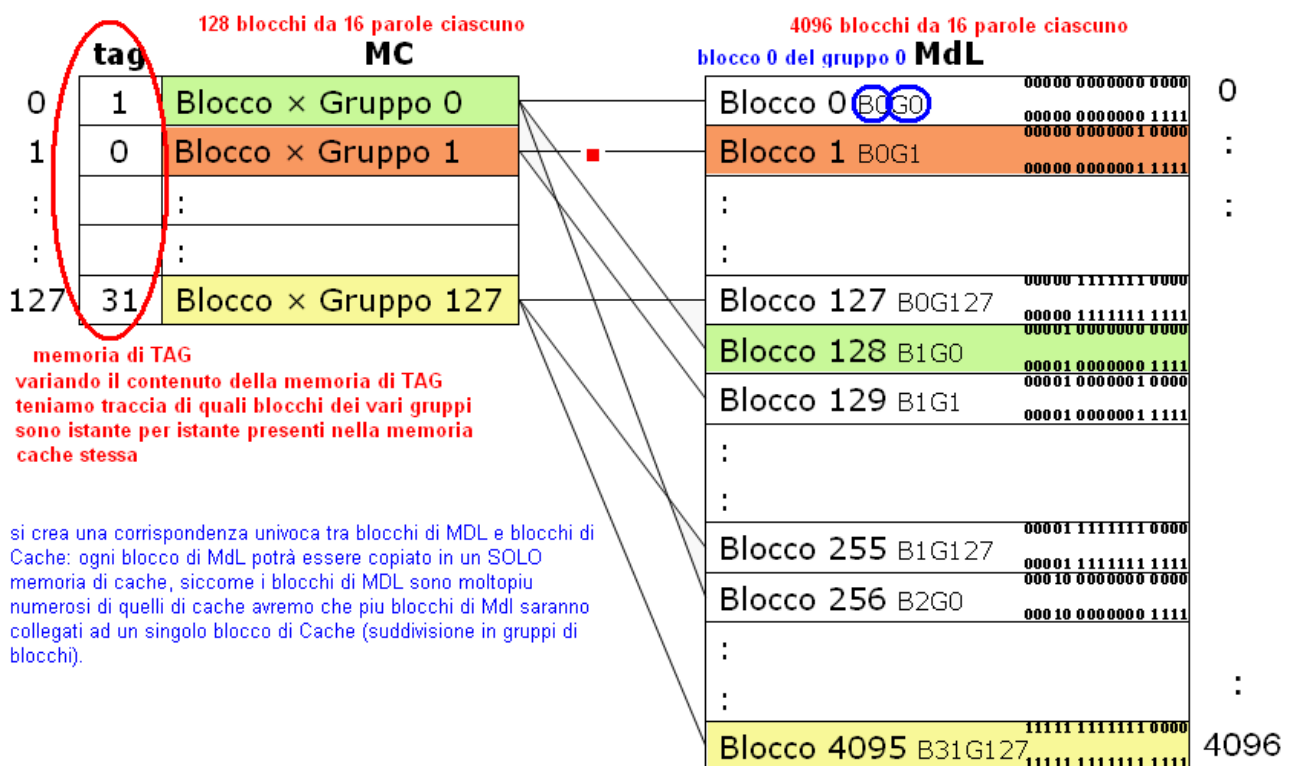
Memoria cache e politica Tag Associative

Il progetto di una struttura con cache richiede la seguente organizzazione generale:

- memoria di lavoro (Mdl) divisa in blocchi di dimensione predefinita (es. 16 celle o parole);
- memoria cache (MC) divisa a sua volta in blocchi di dimensione uguale a quelli di MdL;
- ogni volta che non è presente una cella in cache, si trasporta in MC l'intero blocco di MdL che contiene la cella desiderata.

Abbiamo 3 problemi da risolvere:

- Mapping dei blocchi da MdL a MC: ogni blocco di MdL, quando dev'essere portato in cache in quale (o quali) blocco di MC può essere copiato.
- Ricerca della parola richiesta dalla CPU: quando l'UC genera un indirizzo di memoria dobbiamo verificare se la parola richiesta si trova in cache (HIT- la parola richiesta si trova in cache) o non si trova in cache (MISS - la parola richiesta NON si trova in cache e deve essere prelevata da MdL).
- Sostituzione del blocco di MC in caso di MISS ovvero quali azioni si devono intraprendere per ottimizzare l'uso della cache, per fare in modo che l'HIT RATIO (volte in cui troviamo in cache il dato e volte in cui non lo troviamo) sia il più alto possibile.



Mapping nella politica Tag Associative

Si definisce a priori una corrispondenza univoca fra gruppo di blocchi in memoria di lavoro (MdL) e blocco di possibile destinazione in cache (MC).

Nell'esempio l'indirizzo generato dalla CPU (16 bit) ha la seguente struttura:

NB = N° blocco
nel gruppo

NG = N° gruppo
di blocchi

NP = N° parola
nel blocco



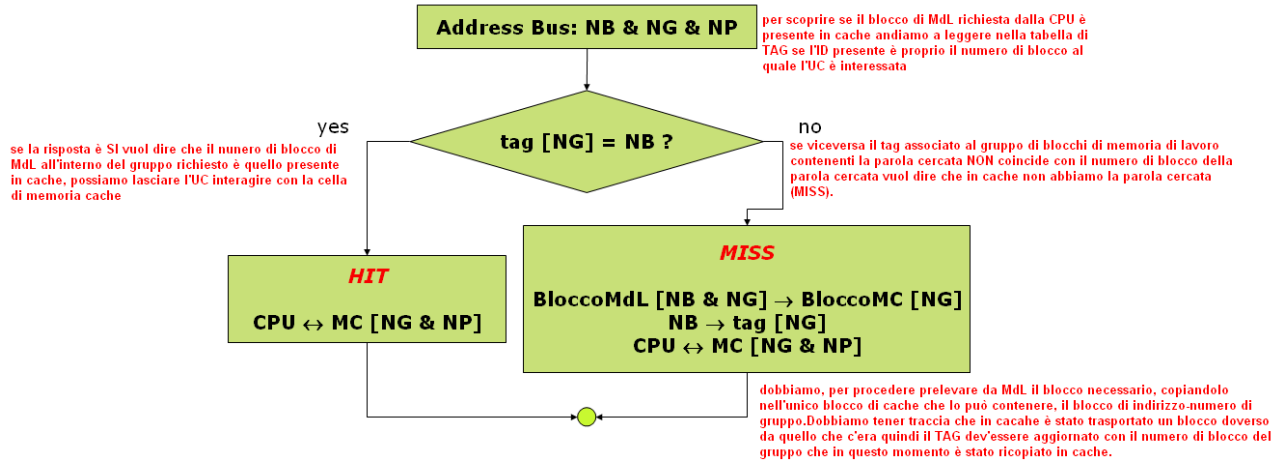
5 bit $\rightarrow 2^5=32$
blocchi per gruppo

7 bit $\rightarrow 2^7=128$
gruppi (blocchi di cache)

4 bit $\rightarrow 2^4=16$
parole \times blocco

Ricerca parola nella cache Tag Associative

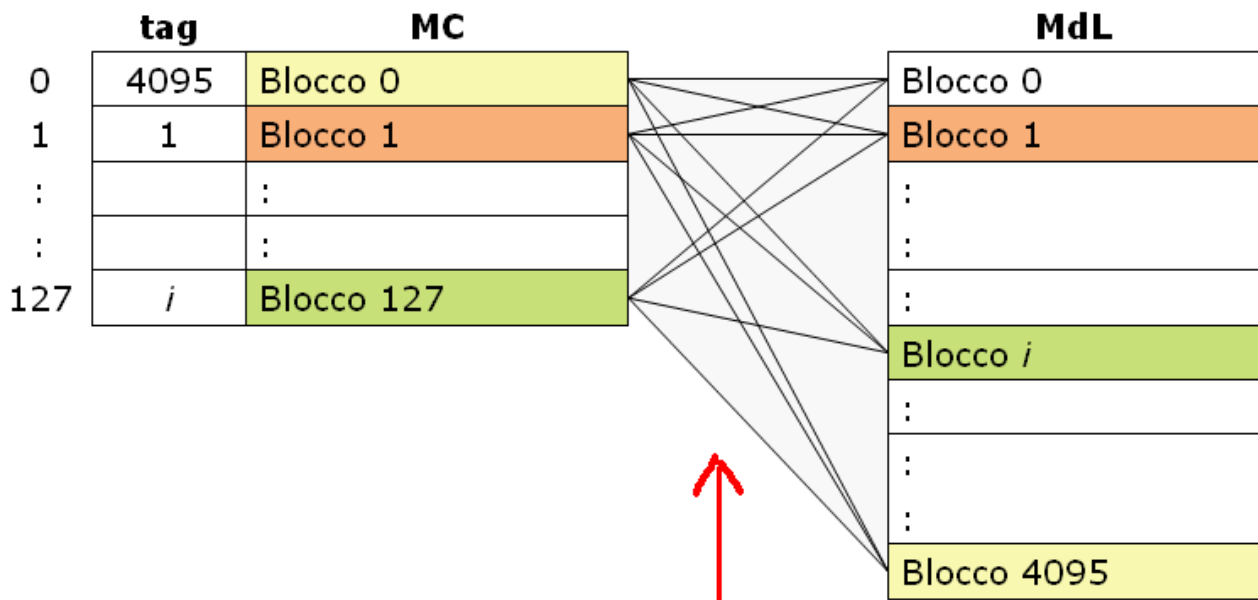
sul bus indirizzi dell'unità centrale compaiono concatenati i bit che indicano a quale numero di blocco all'interno del gruppo la CPU è interessata a quale di blocchi e quale parola all'interno del blocco la CPU è interessata



Caratteristiche Tag Associative

- ☺ Politica semplice: il blocco richiesto dalla CPU può trovarsi solo in un blocco di cache quindi la scoperta di HIT/MISS è rapida e priva di problemi basta andare a leggere il TAG associato all'unico blocco di cache dove può trovarsi il blocco di MdL richiesto dalla CPU con un unico accesso alla memoria; in caso di MISS, il blocco richiesto può essere ricopiato in un'unica posizione, dobbiamo sostituire il blocco che non ci interessa con il blocco interessato.
- ⊗ Politica non ottimizzata: ogni blocco di MC ottimizza solo localmente l'accessibilità ai blocchi di MdL cui è collegato ovvero dei blocchi assegnati e lo sfruttamento dei blocchi di MC non è uniforme, troveremo alcuni blocchi che devono essere sostituiti spesso ed altri non utilizzati. La politica Tag Associative di allocazione dei blocchi di MdL in cache privilegia la semplicità realizzativa a scapito dello sfruttamento ottimale della MC, il problema sta nell'allocazione fissa fra blocchi di MdL e blocco di MC.

Politica Fully Associative



Rimuoviamo l'allocazione fissa che è il principale problema della TAG associative: ogni blocco di MdL può andare a finire in qualsiasi blocco di MC, sparisce il concetto di gruppo, il TAG aumenta però di dimensioni.

nell'esempio l'indirizzo generato dalla CPU (16 bit) ha la seguente struttura:

NBMdL = N° blocco
in MdL

NP = N° parola
nel blocco



12 bit $\rightarrow 2^{12}=4096$
blocchi in MdL

4 bit $\rightarrow 2^4=16$
parole \times blocco

Per poter verificare rapidamente se il blocco richiesto è in cache si utilizza la memoria dei tag ad accesso associativo: presentare il valore cercato ed ottenere in un unico tempo di accesso l'indirizzo della cella che lo contiene oppure segnalazione di assenza.

Per poter decidere dove scrivere il blocco cercato se non è presente: politica LRU (Least Recently Used) elimina il blocco usato meno recentemente. Si associa ad ogni blocco di cache un contatore a saturazione per ogni blocco di MC che viene azzerato quando si accede al blocco associato ed incrementato di 1 se si accede a un altro blocco; se questi contatori, come valori di fine scala, hanno il numero di blocchi totali di cache e si è sicuri di avere almeno un contatore sempre saturo (tutti 11...11). Quel blocco non è stato utilizzato e viene eliminato.

Caratteristiche Fully Associative

☺ Politica ottimizzata: i blocchi presenti in MC sono sempre quelli che nel recente passato sono stati più richiesti dalla CPU e quindi abbiamo un'ottimizzazione globale con sfruttamento omogeneo dei blocchi di MC.

⊗ Politica complessa e costosa: la ricerca del blocco richiesto e la verifica se si trova o meno in cache implica il ricorso a memoria associativa per i tag, la ricerca del blocco di MC da sostituire implica l'uso dei contatori a saturazione, che devono anch'essi essere accessibili in modo associativo.

La politica Fully Associative di allocazione dei blocchi di MdL in cache privilegia lo sfruttamento ottimale della MC ma introduce pesanti complessità realizzative

Politica Set Associative

La politica Set Associative cerca il compromesso fra complessità e sfruttamento della cache: un blocco di MdL può essere copiato in un insieme limitato (set) di blocchi di MC, non in uno solo, non in tutti, in alcuni; si parla infatti di n-way cache Set Associative:

con $n = 2$ ogni blocco di MdL può essere copiato solo in 2 blocchi di MC;

con $n = 4$ ogni blocco di MdL può essere copiato solo in 4 blocchi di MC;

con $n = 8$ ogni blocco di MdL può essere copiato solo in 8 blocchi di MC;

Cerca di ottimizzare pregi e difetti dei due approcci precedenti.

Accesso in scrittura alla memoria cache

Il problema della scrittura in cache: se la CPU fa una scrittura in memoria, il dato in cache viene modificato rispetto al valore precedente sia in caso di hit sia in caso di miss. Cosa facciamo del blocco originale in memoria di lavoro? Abbiamo 2 possibilità.

- **Politica store thru**: si modifica il dato sia in Cache sia in MdL, privilegia la facilità di realizzazione **MA** non ottimizza l'uso della cache.

☺ **Politica semplice**:

- le informazioni in MdL e le loro copie in cache rimangono sempre congruenti;
- non si hanno ulteriori complessità a livello hardware.

⊗ Politica non ottimizzata:

- gli accessi a memoria in scrittura non vengono velocizzati dalla presenza della cache;
- è accettabile solo perché gli accessi in scrittura a memoria da parte della CPU sono decisamente meno di quelli in lettura: per produrre un risultato servono in genere più operandi ma soprattutto, ci sono tutte le fasi di fetch

- **Politica store in**: si modifica il dato soltanto in Memoria Cache, privilegia l'uso ottimizzato della *cache*, introduce complessità realizzative e gestionali per garantire la congruenza fra *cache* e Memoria di Lavoro

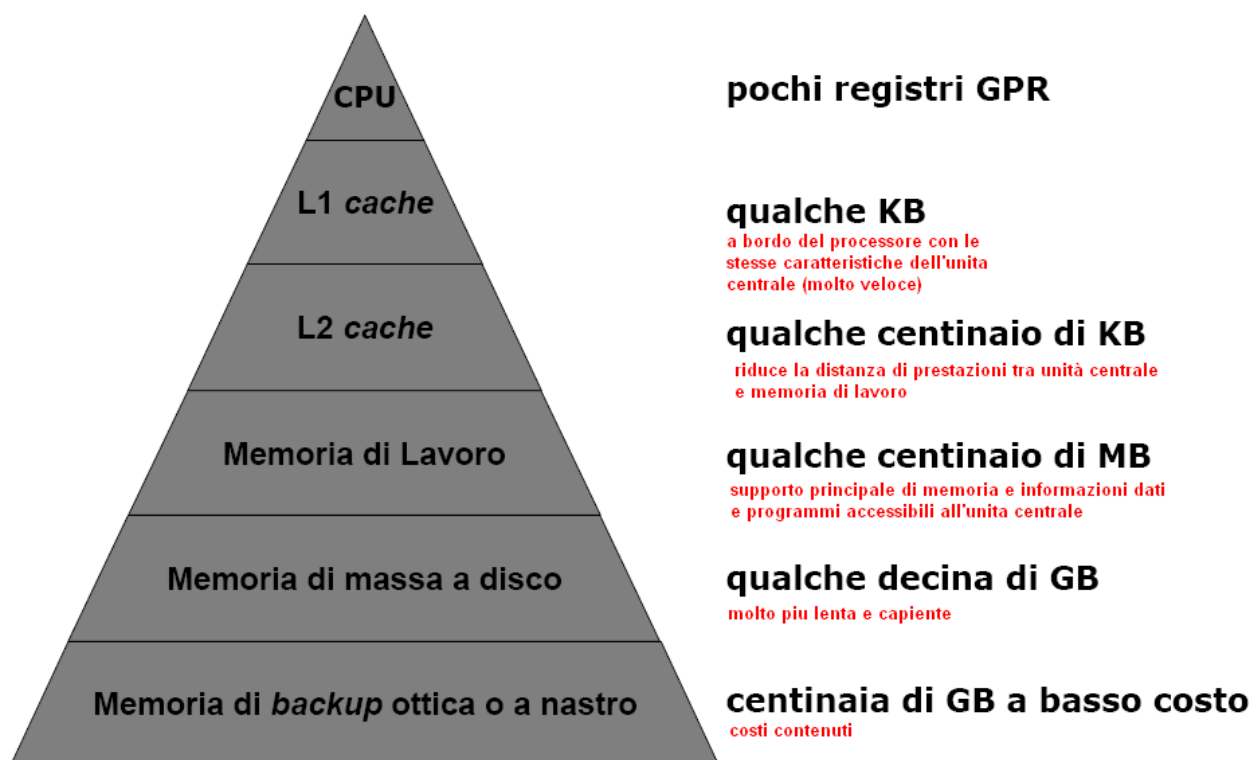
☺ **Politica ottimizzata**:

- anche gli accessi in scrittura vengono velocizzati se trovano la cella desiderata in cache.

☹ **Politica complessa**:

- tra MC e MdL si crea incongruenza: la copia aggiornata è in cache;
- si potrebbe riscrivere il blocco da Cache a MdL prima di eliminarlo, ma se i due blocchi sono uguali si perde tempo: si introduce quindi un bit di modifica M (per ogni blocco di Cache) che viene settato a ogni scrittura, se il blocco da sostituire ha M=1, lo si riscrive in MdL.

Gerarchia di memoria



Rapporto fra gli ultimi due livelli

Memoria di massa a disco magnetico: tempo di accesso: qualche msec. Dimensioni, qualche decina di GB, tipica memoria di massa online (programmi di uso quotidiano dati di uso quotidiano).

Memoria di massa a disco ottico o a nastro: tempo di accesso: dalle centinaia di msec. alle decine di sec., dimensioni centinaia di GB a basso costo tipica memoria di massa offline (originali dei programmi, backup dei propri dati)

Rapporto Memoria di Lavoro ÷ Disco

La memoria di lavoro (con i livelli di cache soprastanti) è una memoria di natura elettronica quindi con tempi di lavoro in linea con quelli della CPU e ad accesso casuale quindi ogni cella richiesta è accessibile nello stesso tempo (cache permettendo, per il discorso satistico...).

La memoria a disco magnetico è una memoria di natura meccanica quindi con tempi di lavoro inaccettabili per la CPU e ad accesso sequenziale o misto nel senso che il tempo di accesso varia a seconda della posizione sul disco del dato.

Disk cache

Un primo modo di usare i primi 2 livelli della gerarchia. Facciamo in modo che una parte della memoria di lavoro, della RAM, sia destinata a contenere i settori di disco più recentemente richiesti (come per la cache). Si velocizza l'esecuzione di programmi che accedono spesso a file su disco. E' relativamente poco usata, le prestazioni sono molto dipendenti dal tipo di programma ed in generale molto modeste.

Memoria virtuale

Si fa vedere a ogni programma un intero spazio di indirizzamento a disposizione dell'unità centrale stessa e tutto a sua disposizione: la memoria virtuale (che sembra esserci, ma non c'è) viene vista da ogni programma perchè parzialmente caricata in memoria di lavoro, il resto rimane su disco;

quando la CPU esegue il programma, genera indirizzi virtuali; la quantità di memoria fisica presente non limita le dimensioni del singolo programma, limita solo le prestazioni.

La quantità di memoria fisica presente non limita il numero di programmi in esecuzione, solo le prestazioni.

Motivazioni del ricorso alle strutture pipeline

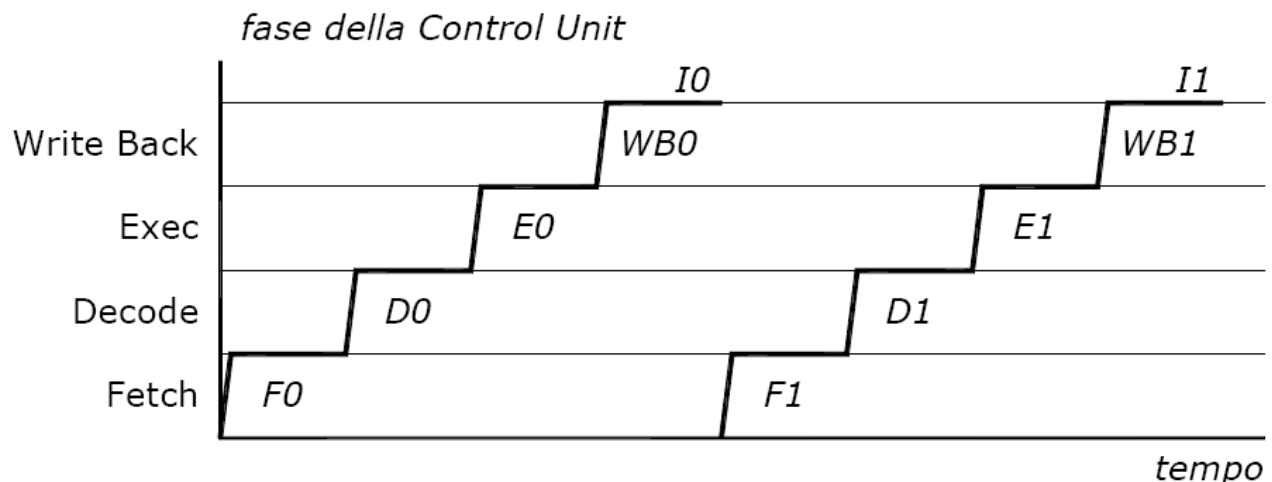
La CPU passa la sua esistenza da accesa a eseguire istruzioni macchina e ogni istruzione macchina viene portata a termine in diverse fasi:

- Fetch (prelievo di istruzione da memoria)
- Decode (scopriamo quale istruzione abbiamo acquisito e cosa ci si aspetta di fare)
- Exec (esecuzione vera e propria con eventuale elaborazione di dati)
- Write Back (scrittura finale dei risultati nei registri o nelle celle di memoria)

C'è poca sovrapposizione fra gli elementi coinvolti nelle varie fasi.

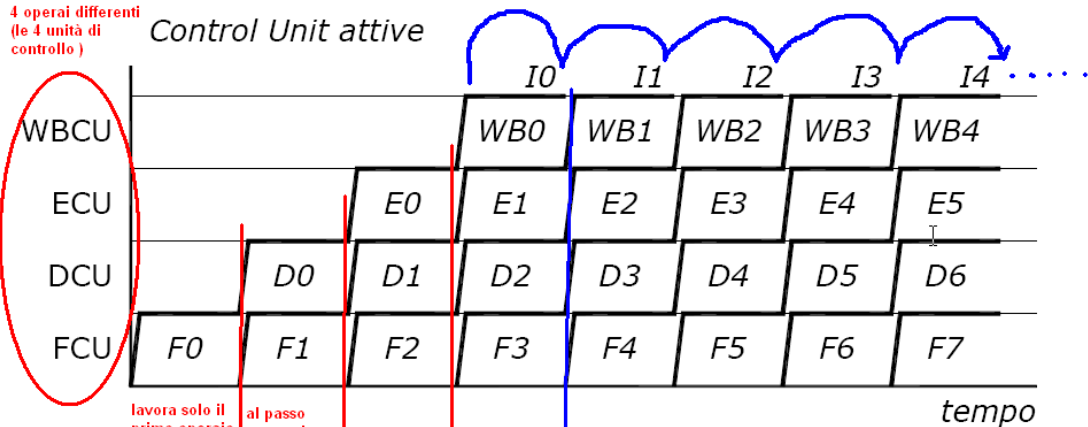
Proprio per la divisione in fasi e perchè gli elementi che lavorano non sono sempre gli stessi possiamo adottare una catena di montaggio (pipeline):

Funzionamento di una CPU normale



Funzionamento di una CPU pipeline

4 operai differenti
(le 4 unità di controllo)

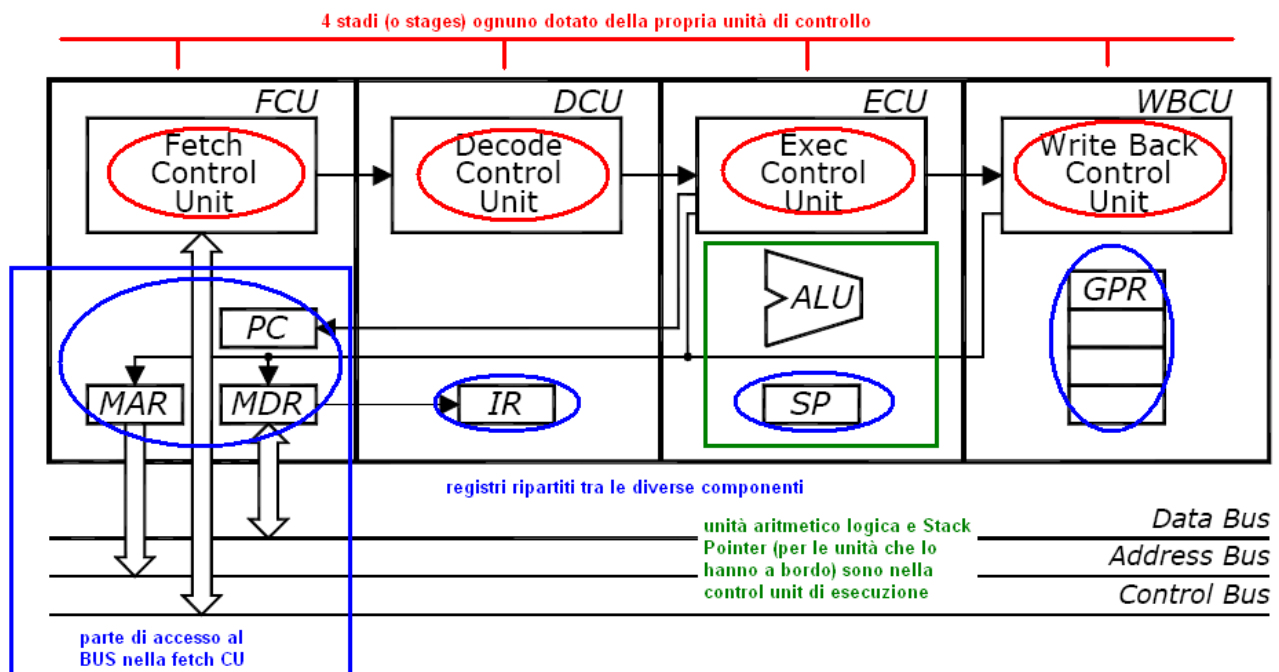


lavora solo il primo operaio che fa il fetch dell'istruzione zero

al passo successivo mentre il secondo operaio decodifica istr. zero il primo può fare fetch

dopo il transitorio del riempimento della catena di montaggio tutti lavorano. Da qui in avanti, ad ogni intervallo di tempo di avanzamento della catena di montaggio un'intera istruzione macchina viene terminata dalla struttura PIPELINE.

Struttura di una CPU pipeline



Una struttura a catena di montaggio (pipeline) consente di velocizzare l'esecuzione non della singola istruzione quanto del numero di istruzioni macchina completate per unità di tempo [MIPS]. L'incremento di prestazioni è dovuto a un intervento architetturale: la tecnologia di realizzazione dell'integrato, quindi la frequenza di lavoro, non cambia. A parità di tecnologia l'adozione architetturale pipeline consente un aumento di prestazioni di un fattore da 2 a 8

Problemi di esecuzione di programmi con CPU pipeline

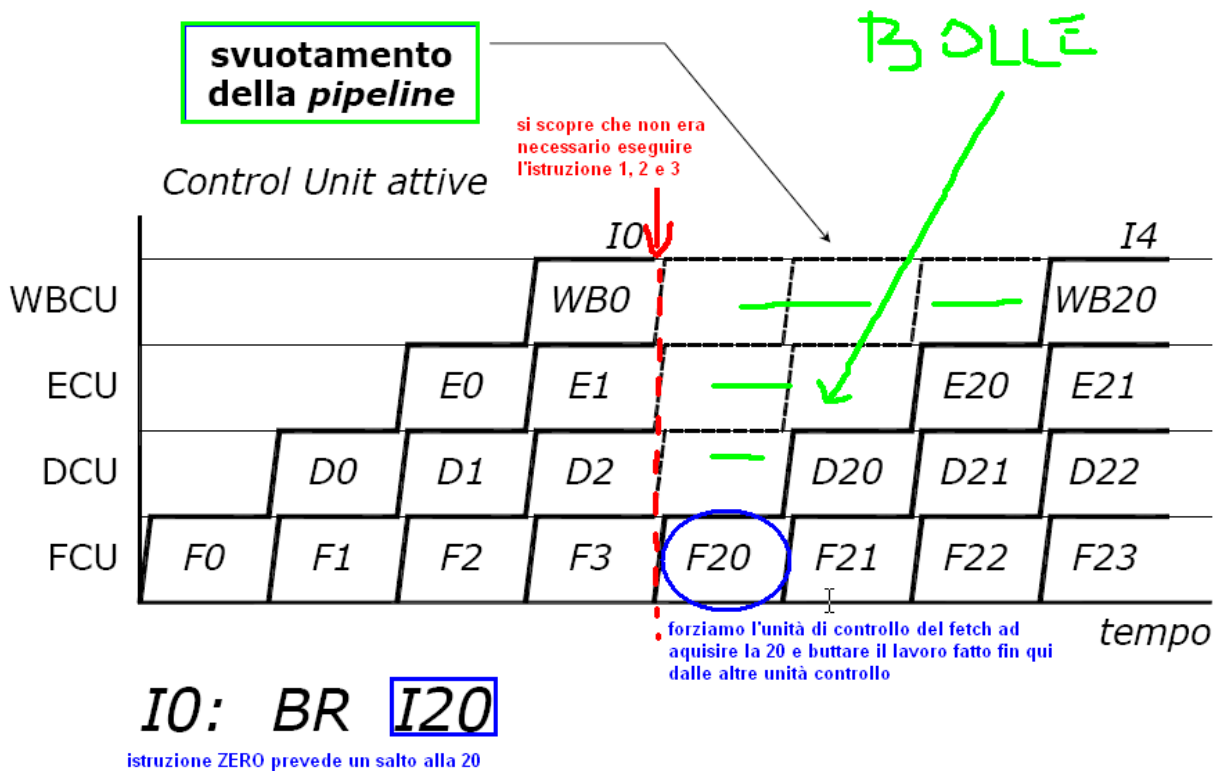
Durante l'esecuzione di programmi da parte di una CPU pipeline, si incontrano situazioni di DIPENDENZA tra le diverse istruzioni del programma o tra le diverse risorse che il programma richiede di usare che comportano il rallentamento delle attività, quindi la perdita di efficacia.

Control dependency: il flusso di esecuzione delle istruzioni interrompe il normale comportamento sequenziale (ad es. istruzioni di salto)

Data dependency: due istruzioni vicine utilizzano uno stesso dato quindi hanno una dipendenza relativa in base all'avanzamento della loro esecuzione.

Resource dependency: due istruzioni entrano in conflitto per una risorsa del sistema (es. l'accesso alla RAM)

Control Dependency:



In questo caso il vantaggio in termini di prestazioni si riduce a nulla.

Per risolvere il problema della control dependency, usiamo la **Branch prediction**

I salti sono comunque una minoranza delle istruzioni (dal 10 al 20% in funzione del programma e possono essere risolti a livello di FCU senza arrivare alla WBCU.

Per migliorare l'efficienza con i salti condizionati utilizza un approccio statistico: **se ho già incontrato l'istruzione di salto mi comporto come la volta precedente, se NON ho mai incontrato l'istruzione di salto (non ho mai saltato in maniera condizionata, è la prima volta che trovo questa istruzione di salto condizionato) se è un salto all'indietro provo a saltare (di solito i salti indietro sono cicli e vanno eseguiti) se è un salto in avanti, provo a NON saltare (tipicamente sono costrutti if...then...else... e il salto è associato ad else)**

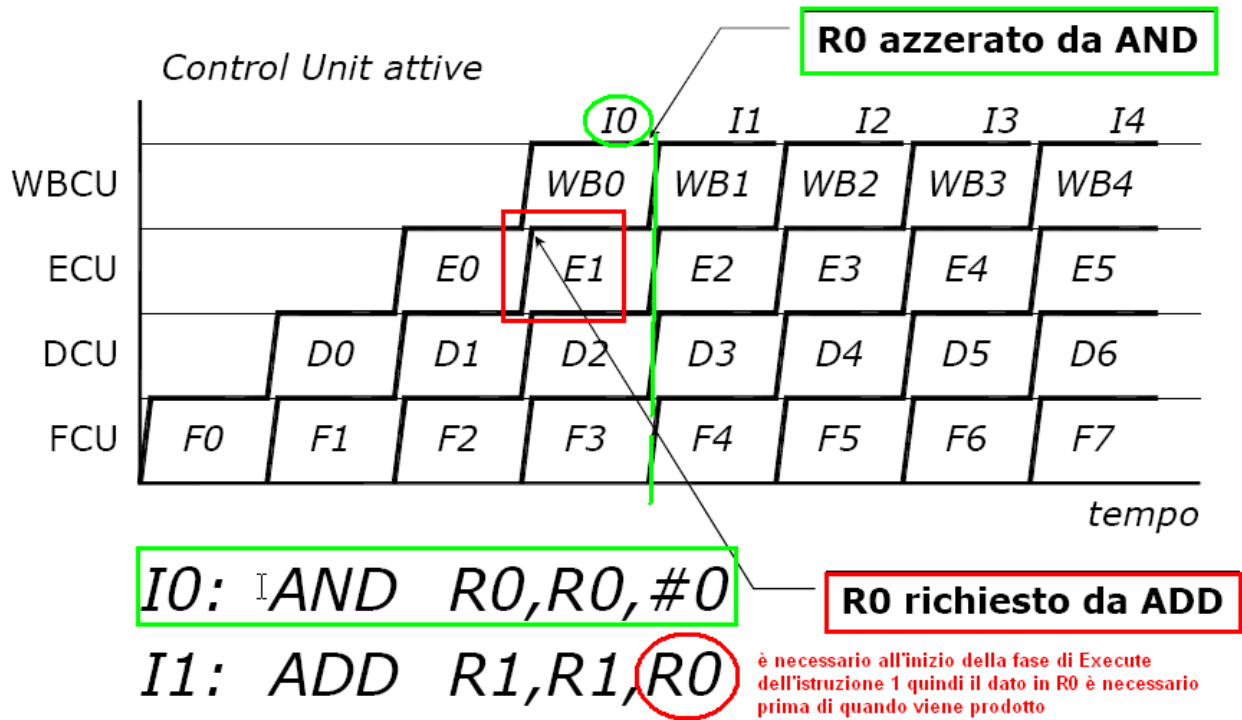
Branch Prediction Table (BPT)

Questo approccio statistico si basa sulla BPT situata a bordo dell'UC e costituita in parte da memoria associativa:

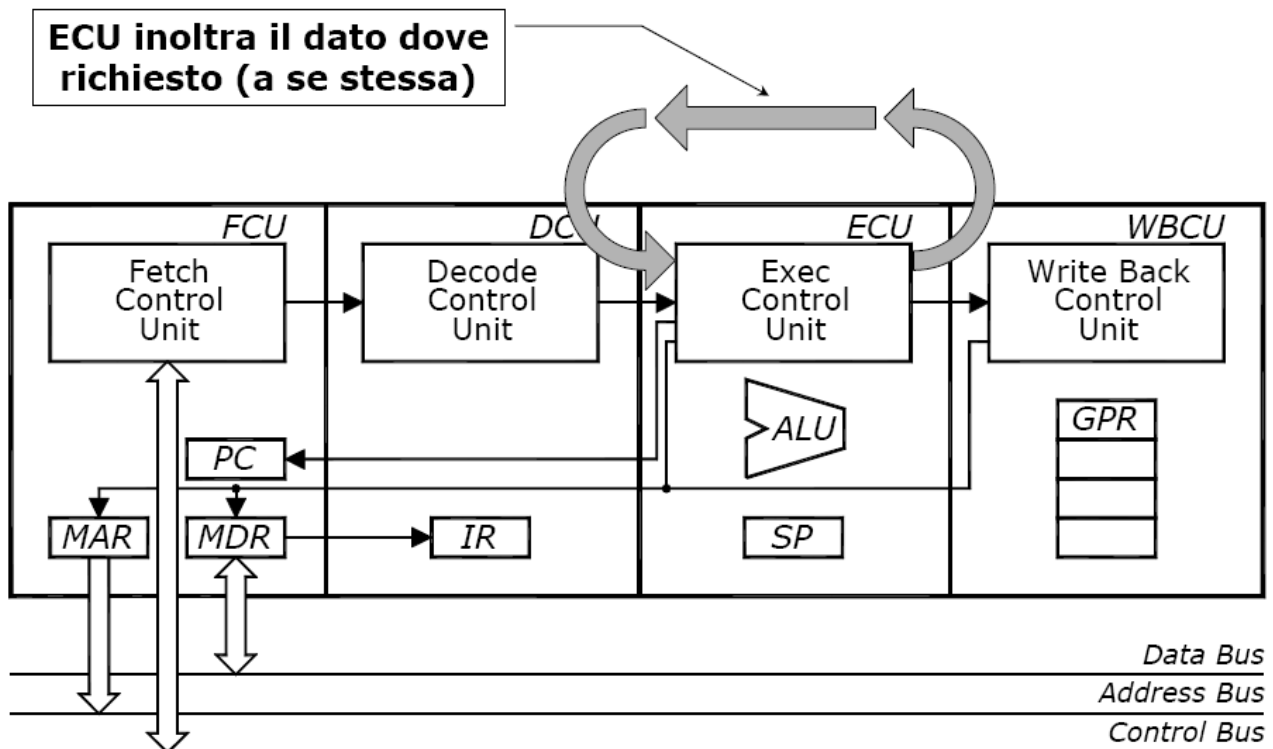
Branch address	Branch destination	Statistics
0	20	T

Quando FCU incontra istruzione di salto, consulta Branch address in modo associativo: se aveva già incontrato l'istruzione, si adegua a Statistics (Taken / Not taken) ed eventualmente, se il salto fosse stato intrapreso la volta precedente salta a Branch destination. Se non aveva incontrato l'istruzione, l'UC del fetch inserisce in BPT e salta indietro/NON salta in avanti. Quando Execution CU risolve il salto, aggiorna BPT e in caso di scelta "sbagliata" blocca l'andamento del programma di FCU provocando lo svuotamento della pipeline.

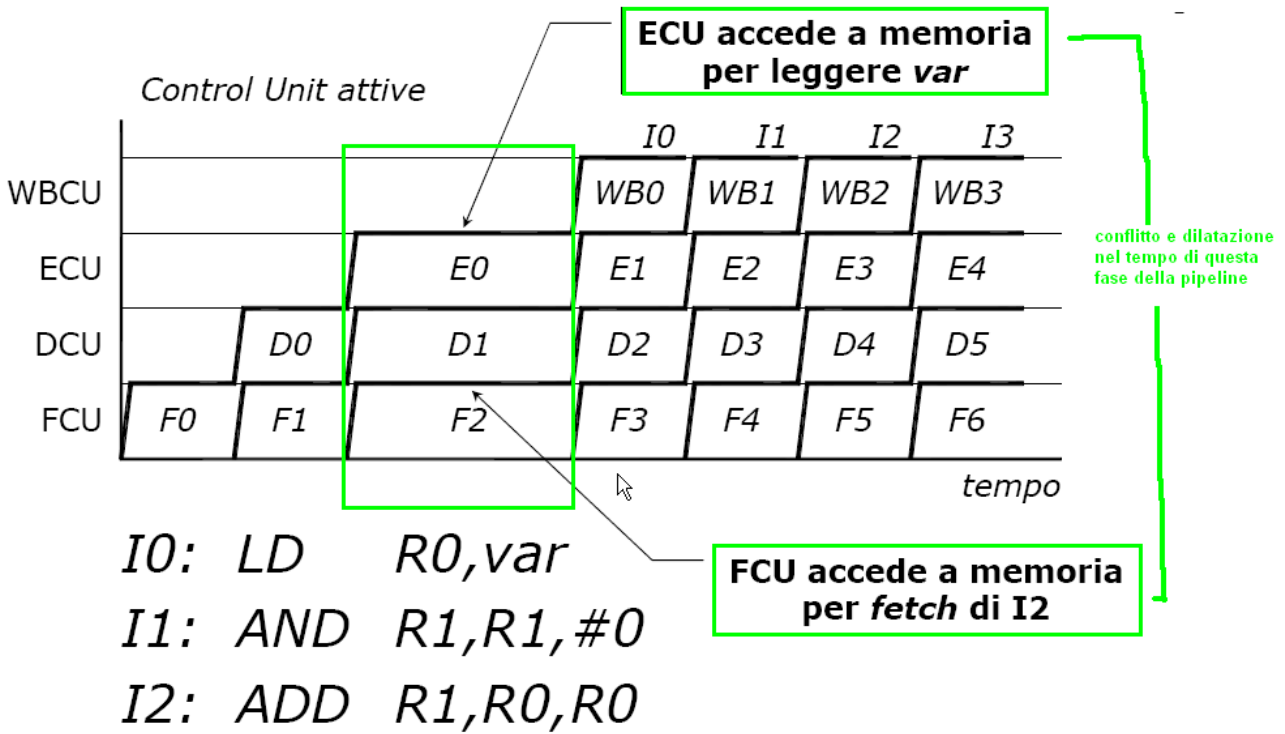
Data Dependency:



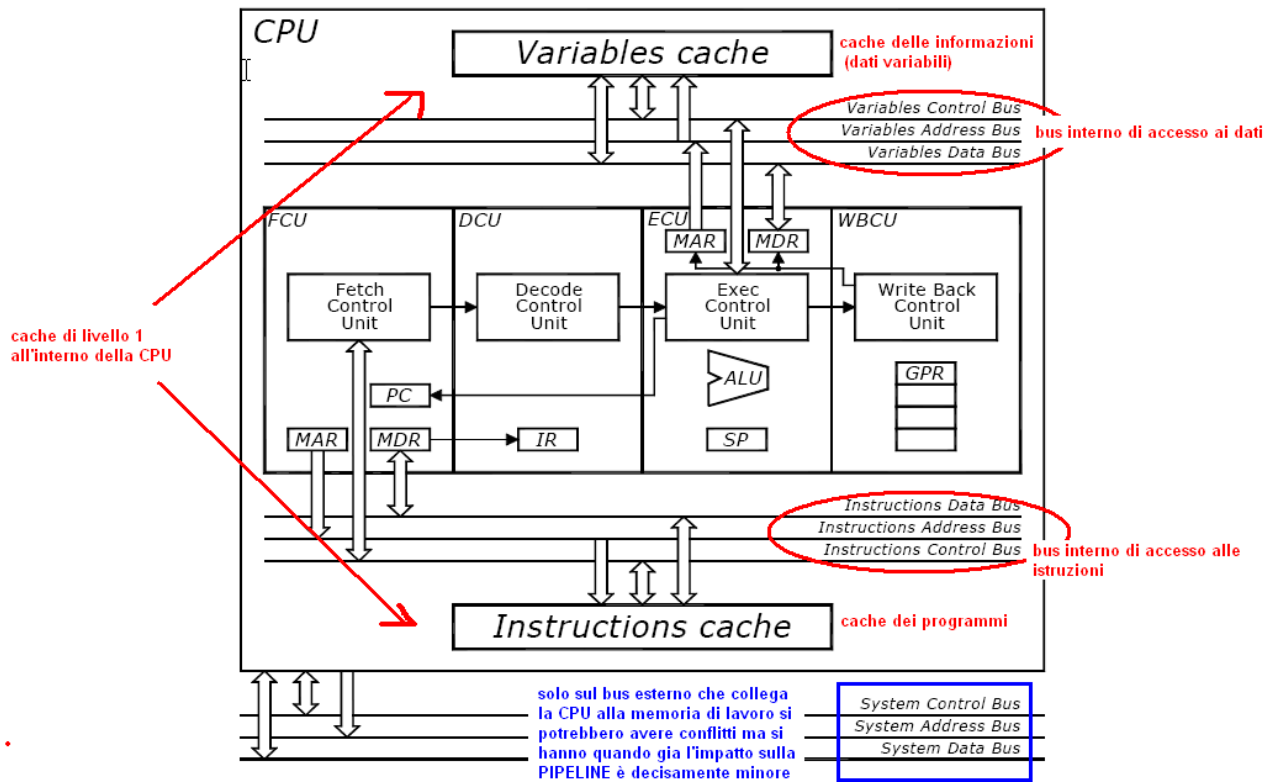
Possibile soluzione il **DATA FORWARDING**



Resource Dependency:



Per risolvere il problema esiste la **MACCHINA DI HARWARD:**



La macchina di Harvard ha spazi di indirizzamento separati per istruzioni e dati variabili.

Con due cache a bordo della CPU elimino i conflitti fra fasi di fetch e fasi di accesso ai dati variabili, senza duplicare il System Bus con un'unica MdL

Rimane solo il conflitto fra ECU e WBCU in caso di letture e scritture simultanee di dati variabili: si ricorre a tecniche di compilazione ottimizzata, che riordinano le istruzioni macchina in modo che una istruzione di STORE e una di LOAD non entrino in conflitto sull'accesso al Variables Bus.