ARCHITETTURA DEI CALCOLATORI

[Esempi in Verilog]

A CURA DI ALESSANDRO PAGHI

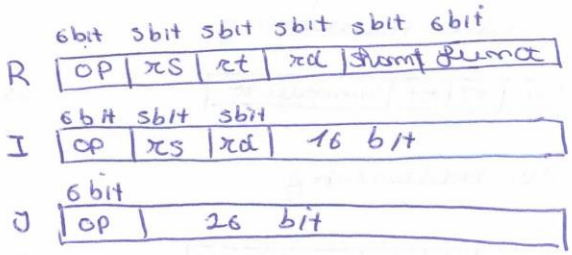
PROFESSORE: Roberto Giorgi (http://www3.diism.unisi.it/people/person.php?id=73&aa=2015)

LINK AL CORSO ANNO 2015/2016:
http://www3.diism.unisi.it/FAC/index.php?bodyinc=didattica/inc.insegnamento.php&id=55204&aa=2015

FREQUENTAZIONE: Sconsigliata.

FORMATI ISTRUZIONI

rs : source register
rt : second source register
rd : destination register
op : opcode
shamt : shift amount
funct : codice funzione



REGISTRI INT : 32 bit
REGISTRI FLOAT : 32 bit
REGISTRI SPECIALI : 32 bit
- PC
- HI
- LO
- EPC
- CAUSE
- STATUS
- BADVADDR

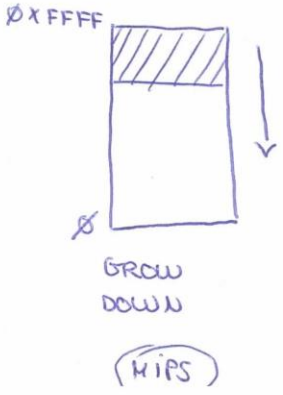
MEMORIA : 8 bit

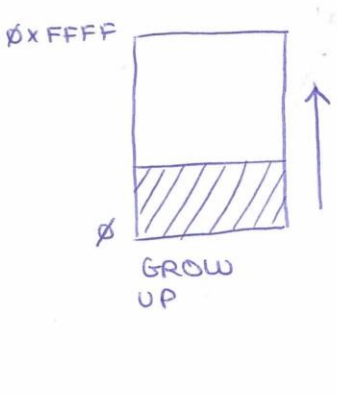
PSEUDOISTRUZIONI :

lui $1, 16bit_alti
ori $1, $1, 16bit_bassi
=> la $1, costante_32_bit

ORDINAMENTO BYTE : ENDIANESS
- Big Endian : ai indirizzi più bassi i byte più significativi;
- Little Endian : ai indirizzi più bassi i byte meno significativi;

CONVENZIONE STACK : VERSO DI CRESCITA E SP

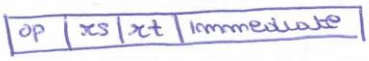




- SP next empty
- SP last Free
MIPS

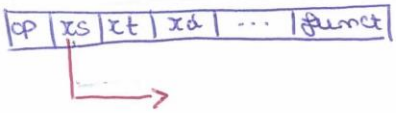
INDIRIZZAMENTO MIPS

1. Immediate Addressing



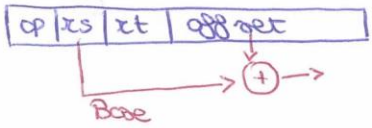
addi, andi, ...

2. Register Addressing



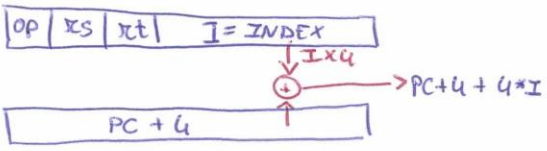
add, sub, ...

3. Base Addressing (Base + offset)



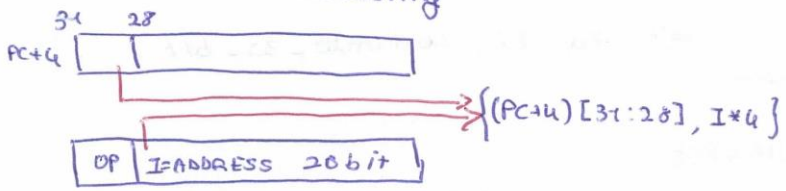
lw, sw

4. PC relative Addressing (Base + Index)



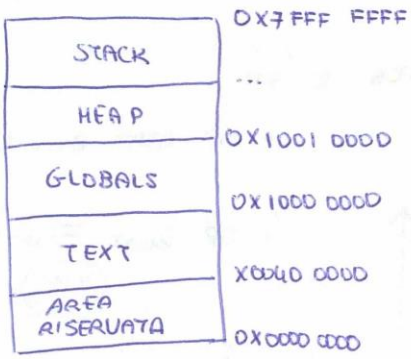
bne, bqe

5. Pseudodirect Addressing



j, jal

MEMORY LAYOUT



CHIAMATA DI FUNZIONE

OPERAZIONI CONNESSE

1) Eventuale salvataggio di registri da preservare ;
2) Preparazione parametri di ingresso ;

→ PRE-CHIAMATA (lato chiamante)

3) Chiamata alla funzione ;

→ Jal myfun

4) Allocazione del call-frame nello stack ;
5) Eventuali salvataggi vari ;
6) Eventuale inizializzazione di $fp ;

→ PROLOGO (lato chiamato)

7) Esecuzione codice della funzione ;

→ CORPO (lato chiamato)

8) Preparazione parametri di uscita ;
9) Ripristino parametri salvati ;

→ EPILOGO (lato chiamato)

10) Ritorno al codice originario ;

→ Jr $ra

11) Eventuale ripristino dei vecchi valori ;
12) Uso dei valori di uscita della funzione ;

→ POST-CHIAMATA (lato chiamante)

COMPILAZIONE DI UN PROGRAMMA STRUTTURATO A MODULI

1) compilazione separata:

```
cc -c main1.c
cc -c funct.c
```

2) collegamento (LINKING):

```
cc -o main.o funct.o
```

3) caricamento ed esecuzione

```
./main
```

ESERCIZIO ASSEMBLATORE A 2 PASSATE

```
addi  $18, $18, $0        LC = 0
addi  $19, $19, 0         LC = 4
loop: add  $4, $18, 0     LC = 8
add   $19, $19, $4        LC = 12
add   $18, $18, $19       LC = 16
bne   $18, $5, loop       LC = 20
la    $18, mynar          LC = 24
      ...
etie1:                    LC = 1000
      ..
j etie1                   LC = 3000
      ...
mynar:                    LC = 7000
```

1) SYMBOL TABLE ①

SYMBOL	LC
loop	8
etie1	1000
mynar	7000

2) RELOCATION TABLE

LC	TYPE	SYMBOL
24	la	mynar
3000	j	etie1

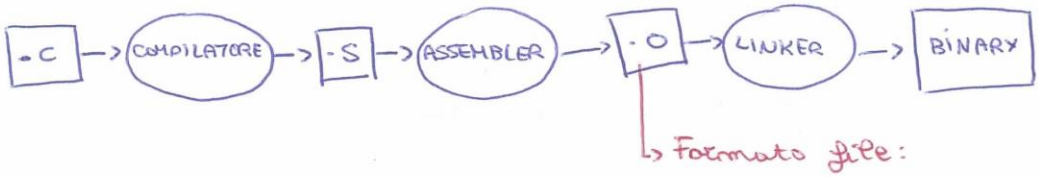
SYMBOL TABLE ②

SYMBOL	LC
etie1	1000
mynar	7000

Siccome loop è riferito ad una bne, beq, viene sostituito con:

$$\text{offset} = \{ LC(\text{SYMBOL}) - [LC(BNE) + 4] \} / 4$$
$$= \{ 8 - [20 + 4] \} / 4 = -4$$

COSTRUZIONE DI UN BINARIO



↳ Formato file:

HEADER
TEXT
DATA
RELOCATION
SYMBOL
DEBUGGING

ASSEMBLATORE A DUE PASSATE

1) Determinazione locazione di memoria etichette;
2) traduzione istruzione in OPCODE;
   determinazione # registri;
   Sostituzione etichette con indirizzo al PC e produzione symbol table;
   Creazione tabella di rilocazione;

SYMBOL TABLE

SYMBOL	LC
etic1	1000
etic2	7000

RELOCATION TABLE

LC	TIPO	SIMBOLO
24	la	etic1
3000	J	etic2

FLOATING POINT IEEE 754

Single Precision : 1 bit segno
8 bit esponente
23 bit mantissa

Double Precision : 1 bit segno
11 bit esponente
52 bit mantissa

MACHEPS ε_{MACH}

1+ = più piccolo numero rappresentabile maggiore di 1.

VALUTAZIONE DELLE PRESTAZIONI

PRESTAZIONI

$$P_X = \frac{1}{E_X}$$ E_X: tempo di esecuzione

SPEED UP

$$S_{XY} = \frac{P_X}{P_Y}$$: "X è S_{XY} volte più veloce di Y"

TEMPO TOTALE E DI CPU

T_J: tempo totale

T_{CPU}: tempo di CPU

$T_{CPU} = T_U + T_K$ con T_U: tempo di CPU relativo all'utente
T_K: tempo di CPU relativo al Kernel

Prestazioni sistema $<= T_J$
Prestazioni CPU $<= T_U$

$T_{CPU} = C_{CPU} \cdot T_C$ con $C_{CPU} = N_{CPU} \cdot \overline{CPI}$

- T_{CPU} ↑ Equazione delle Prestazioni
- C_{CPU} ↑ cicli CPU
- N_{CPU} ↑ #istruzioni
- $\overline{CPI}$ ↑ # cicli per istruzione

$T_C = \frac{1}{f_C}$ ↑ periodo di clock

$\Rightarrow \overline{CPI} = \frac{C_{CPU}}{N_{CPU}}$

LEGGE DI AMDAHL

$$S = \frac{P_{DOPO}}{P_{PRIMA}} = \frac{T_{PRIMA}}{T_{DOPO}} = \frac{T_{PRIMA}}{\frac{T_{MIGLIOR.}}{a} + T_{NON-MIGLIOR.}} = \frac{1}{\frac{p}{a} + (1-p)}$$

$p = \frac{T_{MIGLIOR}}{T_{PRIMA}}$

a: SPEED UP della parte migliorabile

COME VALUTARE N_{CPU}

1) Strumentazione Hardware (sonda) che legge le istruzioni che il processore prende dalla memoria;

2) Contatori Hardware interni alla CPU che contano il numero di istruzioni;

3) Uso di un codice strumentato: riserva un registro per contare le istruzioni ed un registro per contare i cicli di ogni basic block;

4) Uso di simulatori (SPIM)

BENCHMARK

Programma campione che gli utenti sperano abbia un comportamento simile ad un insieme di programmi effettivamente usati da un dato calcolatore (WORKLOAD).

Vantaggi di SPEC2000:
- Portabilità;
- Ampiamente usati;
- Catturano i miglioramenti;

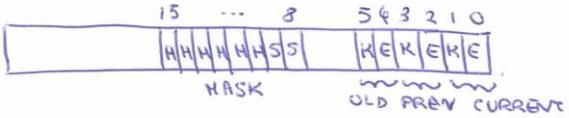
Svantaggi:
- Poco rappresentativi

GESTIONE DI UN'ECCEZIONE

1) Modifica dei registri speciali:
   - EPC: ind. istruzione col pendente (EPC = old PC)
   - STATUS: contiene bit di mascheramento e di esecuzione
   - CAUSE: codifica la sorgente di eccezione
   - BADVADDR: ind. di memoria al quale si è verificato un indirizzamento di memoria illegale

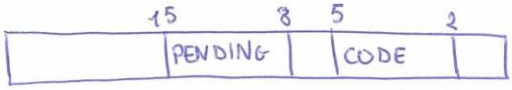
2) PC ← 0x8000'0080

STATUS REGISTER



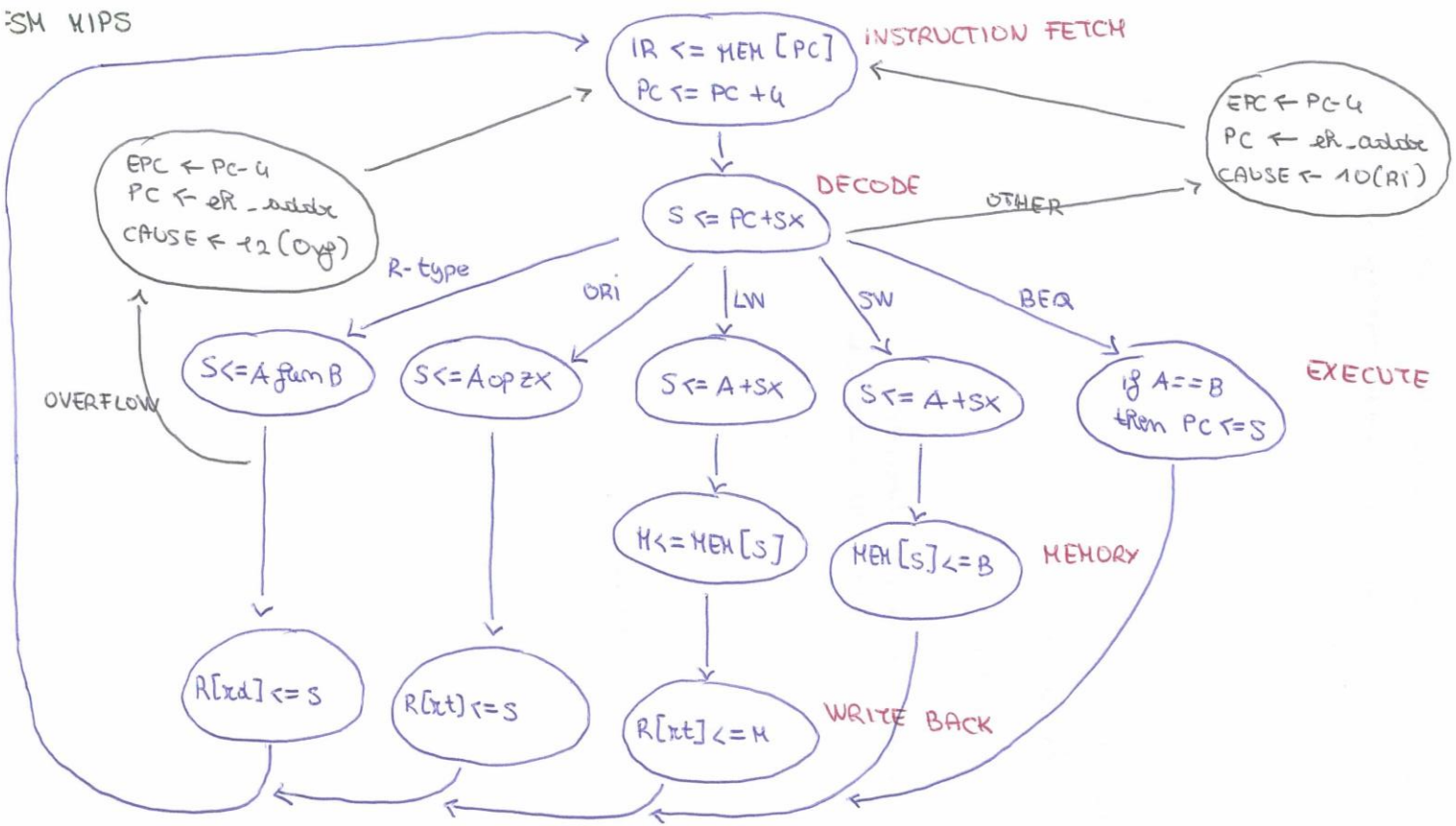
se verificarsi di un'eccezione:
- see di due bit;
- i nuovi 2 bit sono nulli -> mod. Kernel
  intx. disabilitati

CAUSE REGISTER



CODE: codifica il tipo di eccezione

SYSCALL: 08
OVERFLOW: 12



2A) L'istruzione syscall fornisce al codice che si trova nello spazio di indirizzamento Utente un meccanismo controllato per ottenere accesso alle funzionalita' del Nucleo (o Kernel) del Sistema Operativo. Il codice e i dati che si trovano nello spazio di indirizzamento del Nucleo NON risultano normalmente accessibili al codice che si trova nello spazio di indirizzamento Utente (Figura 1).

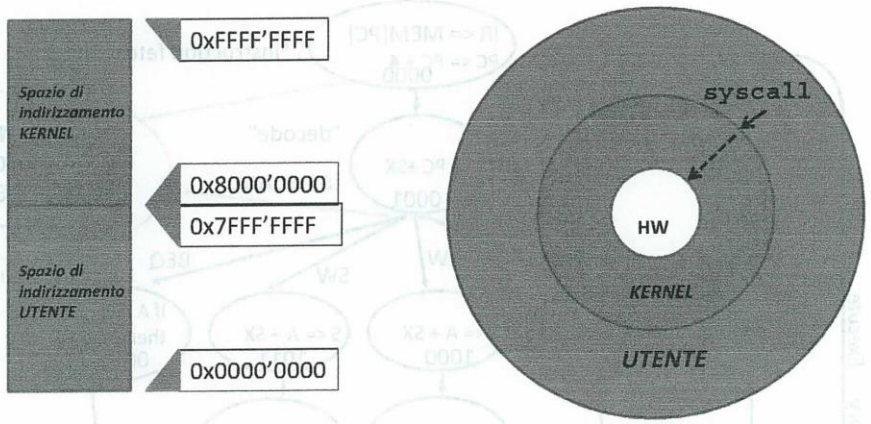


Figura 1. A sinistra, lo spazio di indirizzamento KERNEL e USER nel processore MIPS. A destra, e' rappresentata l'azione della syscall.

2B) Nel momento in cui il processore esegue l'istruzione syscall, vengono modificati alcuni registri speciali dell'architettura MIPS. In particolare: i registri speciali STATUS (bit K e E), CAUSE (bit 5-2), EPC; inoltre, possono essere modificati i registri generali $k0 e $k1.

2C) Passo-passo, cio' che avviene DURANTE l'esecuzione e':
i) i bit 5-0 del registro STATUS vengono traslati verso sinistra di due posizioni ed andranno a costituire le due coppie di bit K e E precedente (bit 3-2) e precedente-precedente (bit 5-4);
ii) i bit 1-0 del registro STATUS vengono inizializzati coi valori 00 indicando rispettivamente che il processore si trova da quel momento in stato Kernel e che gli interrupt sono disabilitati;
iii) i bit 5-2 del registro CAUSE vengono posti al valore 0x8 (o 1000) per indicare che si sta servendo una eccezione del tipo "trap" ovvero la syscall appunto;
iv) il registro EPC viene inizializzato con il PC dell'istruzione syscall (nota: il registro BADVADDR NON VIENE modificato); (nota: il registro $ra NON deve essere modificato perche' potrebbe essere gia' in uso);
v) il PC viene inizializzato con il valore (fisso) 0x8000'0080 e quindi l'esecuzione procede da tale punto del codice Kernel
vi) i registri generali $k0 e $k1 sono a questo punto di pieno dominio del codice Kernel che ne disporra' a suo piacimento, eventualmente modificandoli;
vii) si esegue la routine di servizio che gestisce la syscall, utilizzando eventualmente altri registri generali e in particolare $v0 tipicamente contiene il "numero" o "nome" del servizio richiesto al Sistema Operativo, predisposti PRIMA della chiamata alla syscall, come pure altri registri quali ad es. $a0 possono contenere ulteriori parametri di ingresso; DOPO la chiamata alla syscall, altri registri come $v0 stesso possono contenere dei risultati;
viii) deve infine essere eseguita la istruzione rfe (Return From Exception) che non fa altro che traslare verso destra i bit 5-0 del registro STATUS, in particolare disponendo nei bit K ed E i valori precenti al servizio della system call; dato che la system call e' chiamata dal codice Utente, ora il bit 1 (bit K attuale) varra' di nuovo 0.
ix) il controllo ritorna all'istruzione puntata da EPC+4 predisponendo in un

registro di appoggio (es. $k1) il valore del EPC+4 ed effettuando (es.) "jr $k1" al termine della routine generale di gestione dell'eccezione.

2D) Nella figura 2 e' rappresentato come potrebbe essere modificato il ciclo di Fetch-Decode-Execute-Write_back (in maniera simile a cio' che avviene per una eccezione) al fine di gestire la syscall.

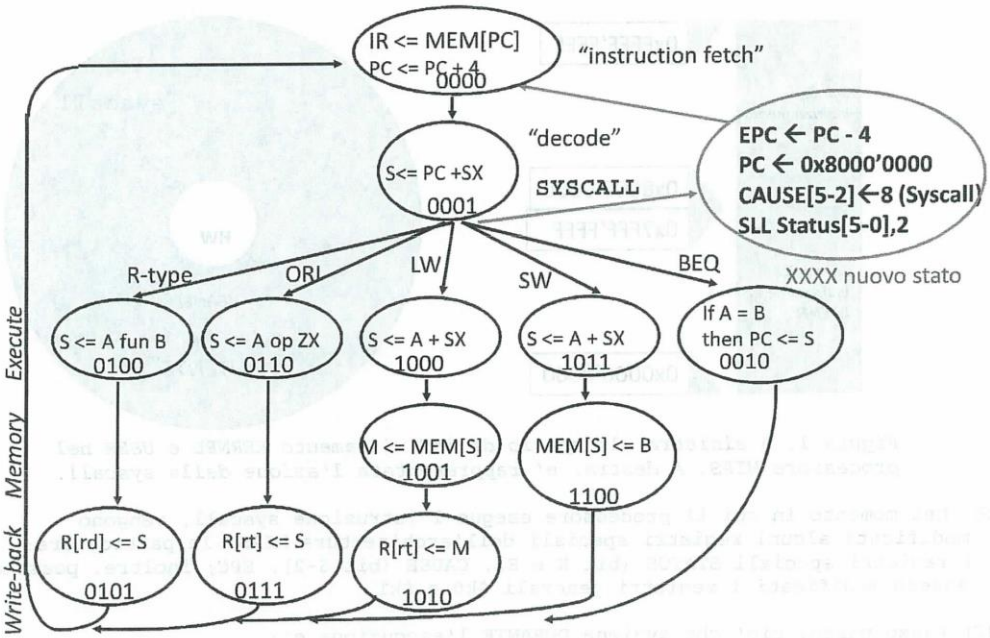


Figura 2. Modifica al ciclo Fetch-Decode-Execute-Write_back per implementare la syscall.

ECCEZIONI E INTERRUPT

ECCEZIONE

Trasferimento non previsto del flusso di controllo

TIPI DI ECCEZIONI

INTERRUPT : Causata da eventi esterni, avvengono rispetto all'esecuzione del programma;

TRAP : Causata da eventi interni, avvengono rispetto all'esecuzione del programma;

ROUTINE DI GESTIONE DELL'INTERRUZIONE

VETTORE DI INTERRUZIONE : memorizza in una tabella gli indirizzi delle Routine di gestione.

TABELLA DELLE ROUTINE DI GESTIONE : memorizza in una tabella direttamente le Routine.

SALTO AD INDIRIZZO FISSO : Salto ad un indirizzo fisso ed indirizzo quale causa la generata la routine.

INTERRUZIONI PRECISE ED IMPRECISE

Precise: Lo stato della macchina viene conservato, come se il programma avesse eseguito fino all'istruzione coperose;

Imprecise: È il software di sistema che deve occuparsi di scoprire in che stato il sistema si trova e quindi prendere le opportune decisioni.

BUS

Collegamento condiviso per comunicare.
Insieme di fili usato per connettere sottosistemi.

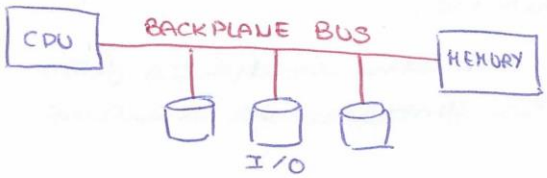
Vantaggi
- versatilità;
- basso costo;
- facile gestione della comunicazione;

Svantaggi:
- genera un collo di bottiglia;
- velocità influenzata da
  - lunghezza bus;
  - # dispositivi;

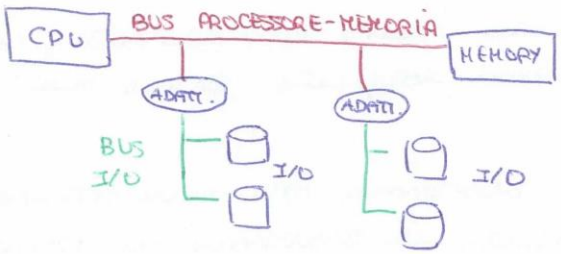
Tipi di BUS

Bus Processore-memoria: corto e veloce;
Bus di I/O: lungo e lento;
Backplane BUS: connette fra loro CPU, I/O e memoria;

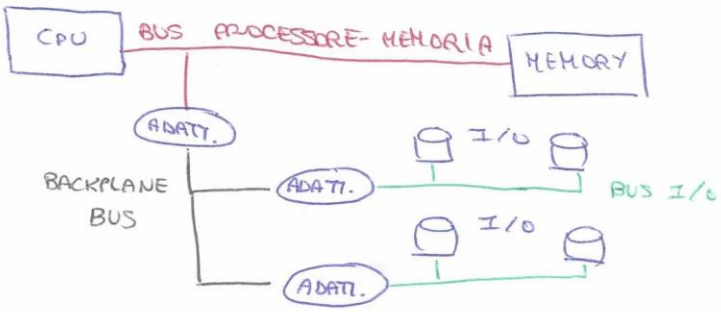
Sistema a singolo BUS:



Sistema a doppio BUS:



Sistema a triplo BUS:



LINEE :

- CONTROLLO : Segnalano Request e Acknowledgement;
- DATI : Trasportano informazioni;

MASTER / SLAVE

Fasi :

1) Fase di arbitraggio (scelta del master);
2) Fase di richiesta;
3) Fase di risposta;

BUS SINCRONO (con clock fra le linee di controllo)

BUS ASINCRONO (senza clock fra le linee di controllo)

BUS-MASTER MULTIPLI

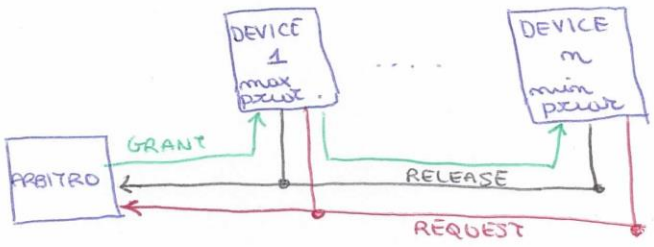
Necessità di un arbitro

- Bus-master richiede all'arbitro l'uso del bus;
- Aspetta il grant;
- Segnala che ha finito di usare il bus;

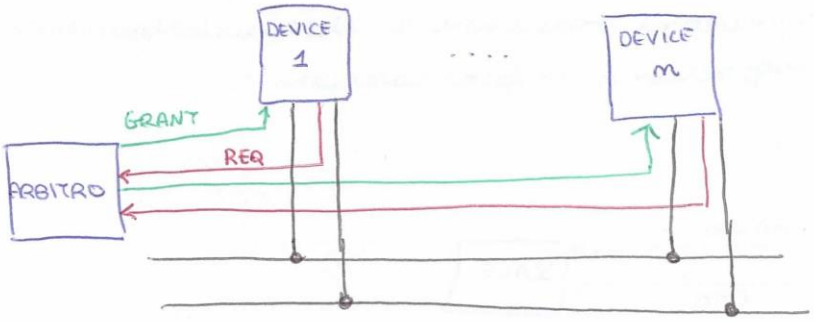
Arbitro bilancia due fattori:

- Priorità della richiesta;
- Fairness;

ARBITRAGGIO CENTRALIZZATO DAISY CHAIN



ARBITRAGGIO CENTRALIZZATO PARALLELO



ARBITRAGGIO DISTRIBUITO CON AUTO SELEZIONE

Bus controllato da un'unità con priorità intermedia, se un'unità con priorità più alta richiede il bus gli viene passato.

ARBITRAGGIO DISTRIBUITO CON RILEVAMENTO COLLISIONI

Ethernet

Bus può essere tenuto per tempi limitati e prima di trasmettere dobbiamo controllare che sia libero.

TECNICHE DI PILOTAGGIO DI DISPOSITIVI I/O

PROTOCOLLI

POLLING

Il dispositivo mette in un "data register" il dato e segnala questo fatto tramite uno "status register".
Il processore controlla periodicamente lo "status register".

INTERRUPT

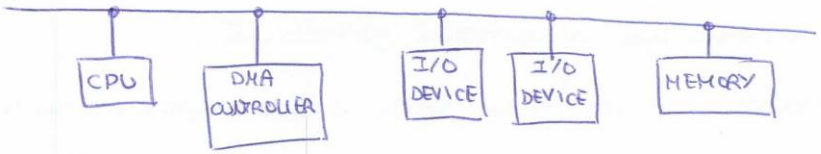
Ogni volta che il dispositivo ha bisogno di attenzione dal processore, interrompe il processore tramite una linea dedicata.

DMA (DIRECT MEMORY ACCESS)

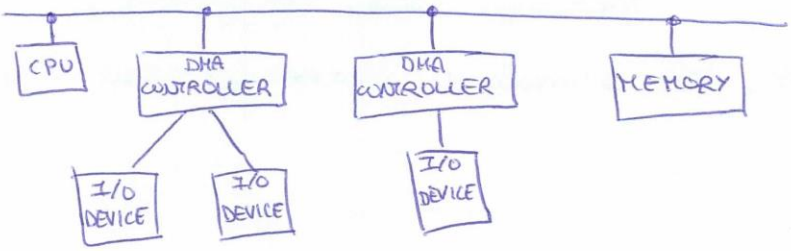
Il processore inizia l'operazione e poi un processore ausiliario si occupa del trasferimento del dato. Il processore è notificato che l'operazione è finita con un interrupt dal DMA/IOP

CONFIGURAZIONI

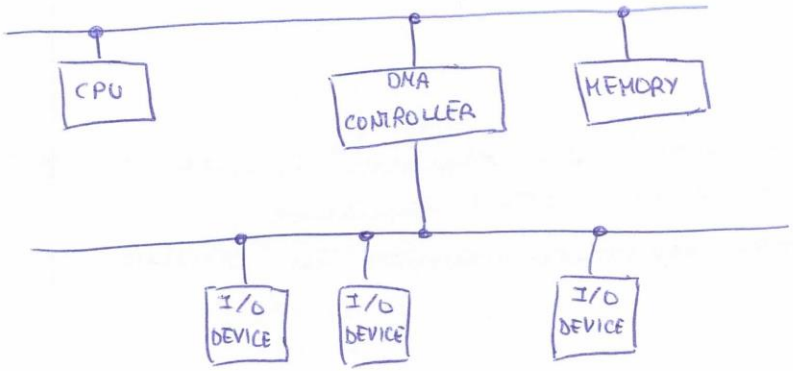
Singolo BUS, DMA controller separato



Singolo BUS, DMA controller Integrato



BUS Separato



messaggio che manda la CPU:

OP	DEVICE	ADDRESS

DMA CONTROLLER (DMAC)

Periferica dedicata che:

- intervetta e controlla accessi al bus tra un dispositivo I/O e la memoria e tra due aree di memoria

STRUTTURA:

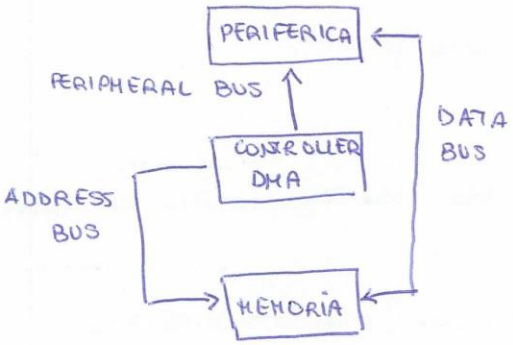
- Address Generator: genera indirizzi di memoria o di periferica connessi;
- Address Bus: Contiene gli indirizzi generati;
- Data Bus: Trasferisce i dati dal DMAC alla destinazione;
- Bus Requester: Usciti per richiedere il BUS alla CPU;
- Local Peripheral Control: Permette la selezione e la gestione della periferica;
- Interrupt Signals: Per l'invio di interrupt alla CPU;

FUNZIONAMENTO:

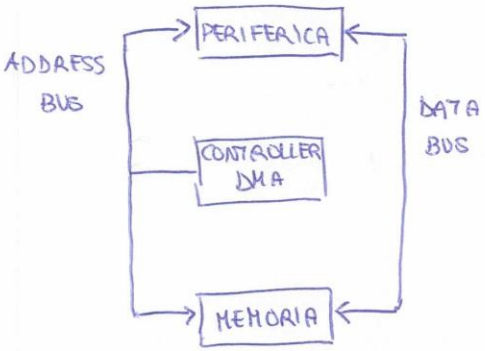
1) Programmazione controller;
2) Inizio trasferimento;
3) Richiesta del BUS
4) Generazione indirizzo;
5) Trasferimento dati;
6) Aggiornamento indirizzo;
7) Aggiornamento del processore;

MODELLI:

Trasferimento dei dati singolo



Trasferimento dei dati doppio

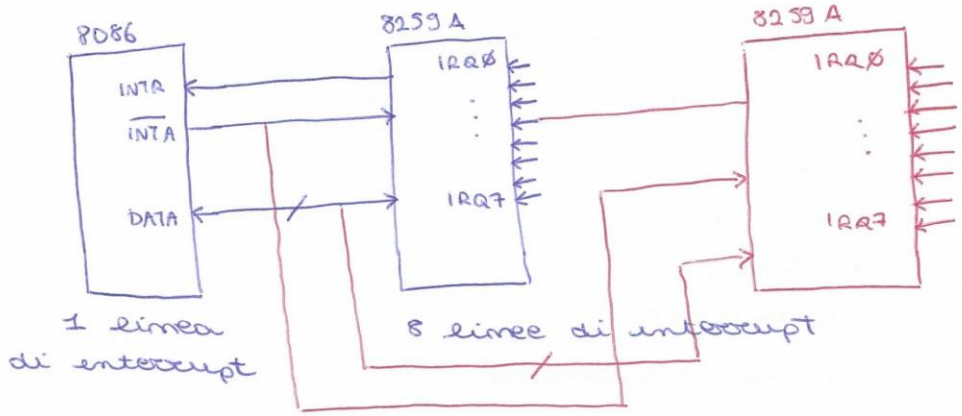


N.B. Si usano due indirizzi e due accessi.

TIPI:

- 1D: Singolo registro per l'indirizzamento;
- 2D: Doppio Registro per l'indirizzamento;
- 3D: Triplo Registro per l'indirizzamento;

CONTROLLORE DI INTERRUPT 8259A



Sequenza eventi :

N.B. Cascading

1) 8259A accetta interrupt ;
2) 8259A determina priorità ;
3) 8259A avvisa 8086 attivando INTR ;
4) 8086 invia conferma su INTA ;
5) 8259A invia su DATA il numero che identifica la sorgente di interrupt
6) CPU serve l'interrupt mandando in esecuzione la relativa routine di servizio ;

TIMER INTEL-8254

Può servire a:

- Generare ritardi preprogrammati;
- real time clock;
- Conteggio eventi;
- Gen. frequenza programmabile;
- Gen. onda quadra programmabile;
- Controllo motori;

...

Formato da:

- 3 contatori a 16 bit;
- 6 modalità di conteggio;

Modalità di conteggio:

0) Interrupt al termine del conteggio;

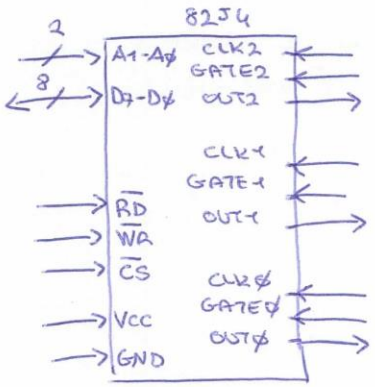
1) Programmable one shot;

2) Rate generator;

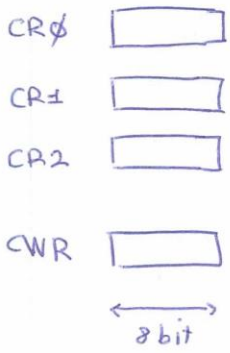
3) Square wave rate generator;

4) Software triggered strobe;

5) Hardware triggered strobe;



Visione Logica del CHIP



4 Registri a 8 bit

Programmazione del Timer

1) Scrivere in CWR per selezionare contatore e metodo;
2) Scrivere in CR_j la costante di tempo;
3) Abilitare conteggio con segnale alto sul GATE;
4) Prelevare l'uscita su OUT;

COMUNICAZIONE SERIALE ASINCRONA

- Ad un certo istante è possibile individuare un TX e un RX;
- 0 bit che compongono il dato sono trasmessi in sequenza;
- La trasmissione avviene senza segnale di clock;

La velocità di trasmissione è data dal numero di bit trasmessi per secondo (bit-rate)

Formato FRAME:

- 1 bit start;
- 8 bit dati;
- 1 bit di parità;
- 1 bit di stop;

PORTA SERIALE (UART)

Costituita da due registri a scorrimento, uno TX e uno RX.

Si usa un BUFFER FIFO per evitare la perdita dei dati.

Connettore DB9 (9 pin):

- TXD (Transmit Data)
- RXD (Receive Data)
- RTS (Request To Send): TX è pronto a inviare dati.
- CTSC (Clear To Send): RX è pronto a ricevere dati.
- DSR (Data Set Ready): modem pronto per la connessione
- DTR (Data Terminal Ready): UART pronto a stabilire la connessione.
- DCD (Data Carrier Detected): modem ha rilevato la portante sulla linea telefonica.
- RI (Ring Indicator): Squilli in uscita sulla linea telefonica.

INTRODUZIONE AL SOTTOSISTEMA DI MEMORIA

Tecnologie usate per realizzare memorie:

- ACCESSO SEQUENZIALE (Nastro)
- ACCESSO SEMICASUALE (Disco fisso)
- ACCESSO CASUALE (RAM, ROM, ...)

Memorie ad ACCESSO CASUALE di tipo NON VOLATILE:

- ROM (Read Only Memory)
- EPROM (Erasable Programmable Read-Only Memory)
- EEPROM (Electrically EPROM)

Memorie ad ACCESSO CASUALE di tipo VOLATILE:

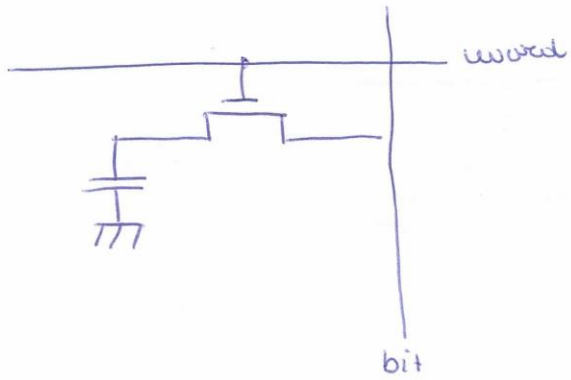
- DRAM (Dynamic Random Access Memory)
  Deve essere rinfrescata periodicamente
- SRAM (Static Random Access Memory)
  Basta alimentare il chip per non perdere dati

Parametri delle prestazioni di memoria:

- Access Time (tempo inizio lettura - arrivo 1° dato)
- Cycle Time (tempo inizio 1° lettura - 2° lettura)
- Latency Time (tempo accesso al dato completo)
- Bandwidth (byte/sec)

DRAM

Cella di DRAM



Scrittura:

1) Preparazione dato sulla bit line;
2) Selezione della word line;

Lettura:

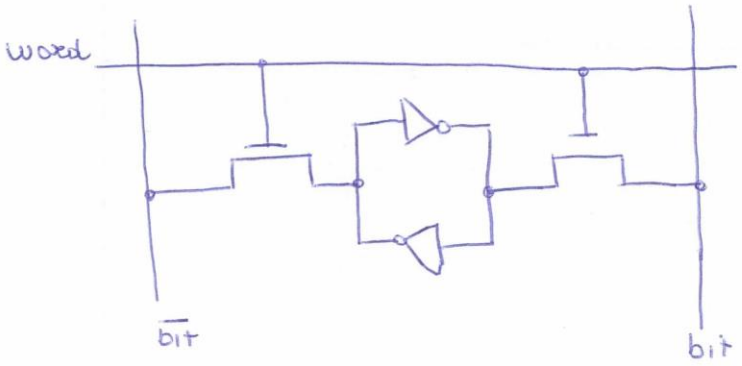
1) Caricamento di bit a $V_{DD}/2$;
2) Selezione della word-line;
3) Scambio di carica fra cella e bit line;
4) Rilevazione della variazione;
5) Caricamento del bit line col dato letto;

Refresh:

1) Effettuo una finta lettura di tutte le celle;

SRAM

Cella di SRAM



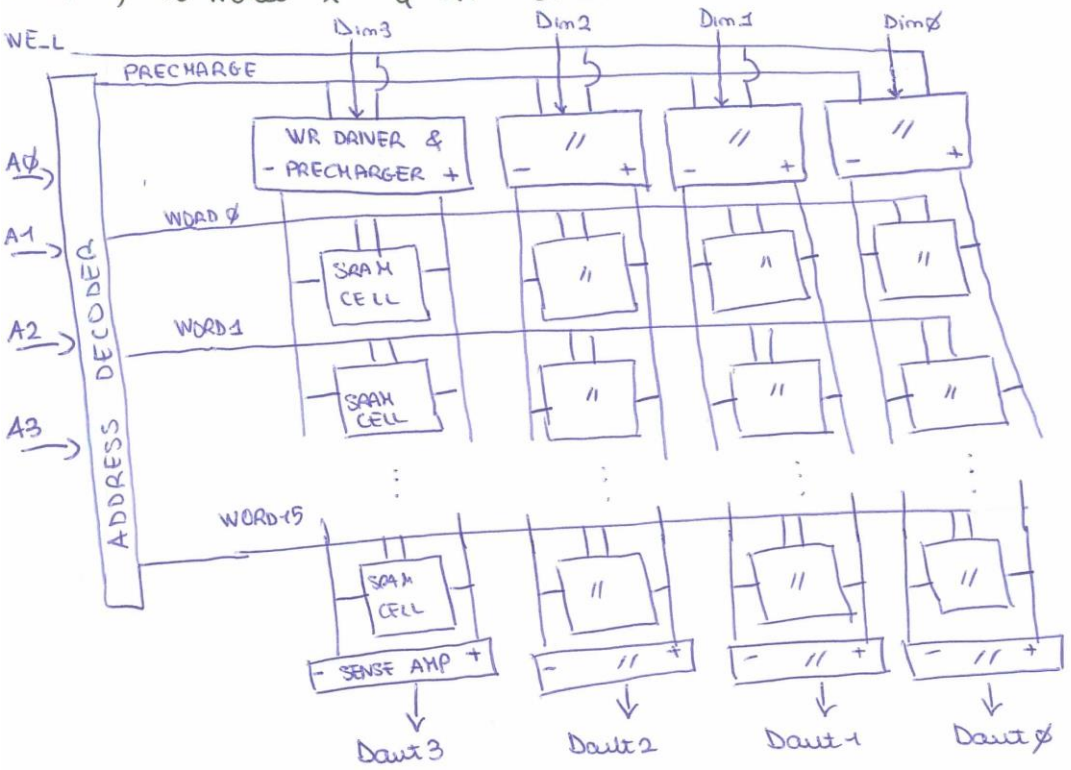
Scrittura:

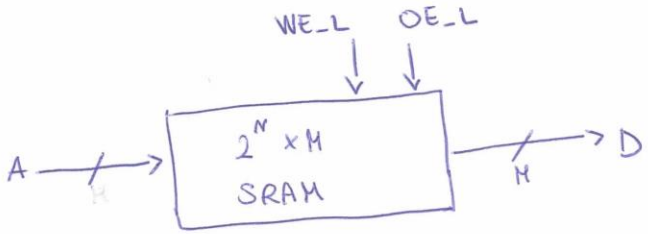
1) Preparazione dato sulla bit line;
2) Selezione della word line;

Lettura:

1) Precaricamento di $\overline{bit}$ e bit a V_{DD};
2) Selezione della word line;
3) La cella preca-carica da bit o $\overline{bit}$;
4) Il sense-amplifier in fondo alla colonna amplifica la differenza fra bit e $\overline{bit}$;

SRAM , 16 word x 4 bit - ORGANIZZAZIONE CELLE





MECCANISMI DI MIGLIORAMENTO DELLE PRESTAZIONI

PRINCIPIO DI LOCALITA'

Località Temporale: un riferimento in memoria tende ad essere ripetuto dopo poco tempo.

Località Spaziale: un riferimento in memoria tende ad essere a locazioni vicine a quelle usate recentemente.

IMPLICAZIONI

1) Posso utilizzare piccole memorie veloci per mantenere i riferimenti in memoria più utilizzati dal processore.
2) Posso creare N livelli di memoria sempre più grandi e meno veloci per contenere riferimenti utilizzati che non possono essere contenuti nella memoria veloce.

GERARCHIA DI MEMORIA

-> Presenta al processore una quantità di memoria pari a tutta quella disponibile nell'ultimo livello.

-> Fornisce al processore un tempo di accesso che si avvicina a quello del primo livello.

MEMORIA CACHE

Scopo: memorizzare copie di blocchi di memoria principale.

Funzionamento: ogni volta che faccio accesso ad una locazione di memoria, lascio in cache una copia del corrispondente blocco.

B: dimensione del blocco di Cache in Byte

C: dimensione della cache in Byte

M: dimensione della memoria in Byte

$M \gg C$

$N_M = \frac{M}{C}$: numero di blocchi della memoria

$N_C = \frac{C}{B}$: numero di blocchi della Cache.

X: indirizzo di byte

$X_H = \frac{X}{B}$: indirizzo del blocco di memoria

Ogni dato in cache è associato in formato:

DATA | TAG | BIT VALIDITA'

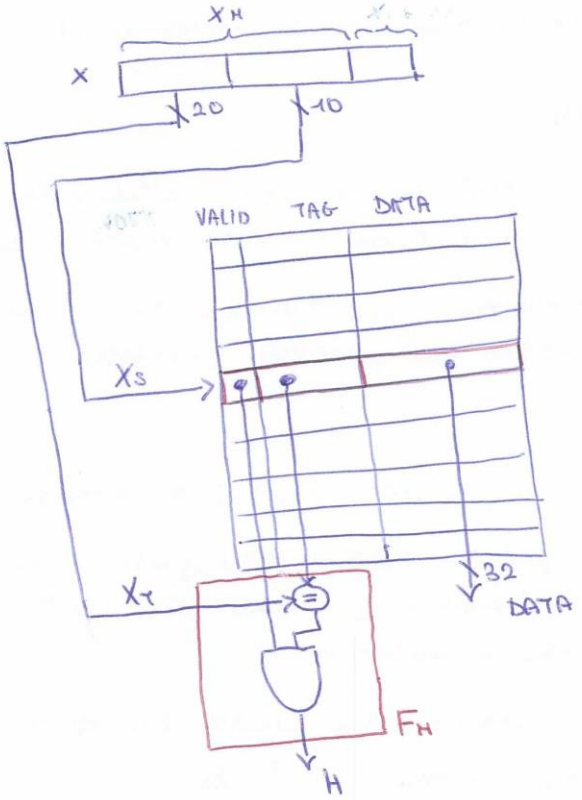
$X_S = X_H \,\%\, N_c$: indirizzo del dato

$X_T = X_H / N_c$: tag

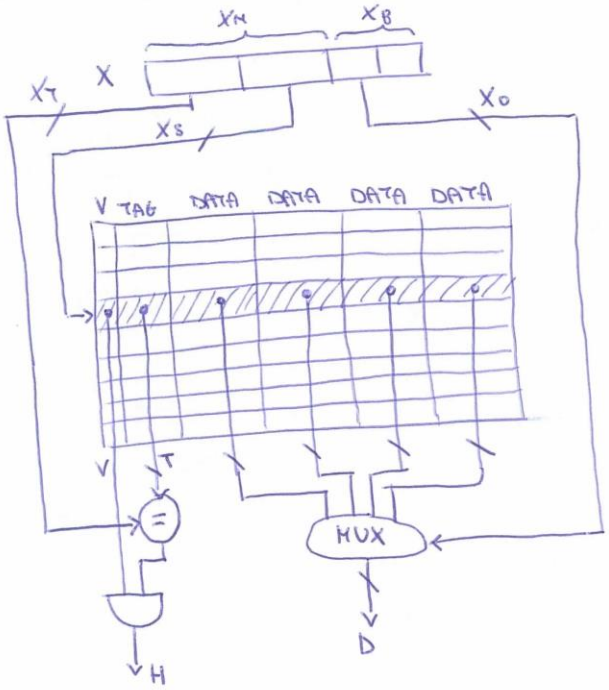
Funzione di Hit:

F_H: ($X_T == T$ && $V == 1$? 1 : 0)

Schema logico della Cache ad Accesso Diretto:



CACHE AD ACCESSO DIRETTO PIÙ GRANDI



X: indirizzo al byte

$X_H = \frac{X}{B}$: indirizzo al blocco di memoria

$N_C = \frac{C}{B}$: numero di blocchi della cache

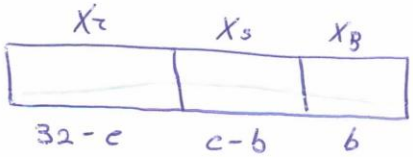
$X_T = X_H / N_C$: tag

$X_S = X_H \,\%\, N_C$: indirizzo in cache

$X_B = X \,\%\, B$: byte del blocco di offset

$X_O = X_B / W$: offset

con W : numero di byte di un dato

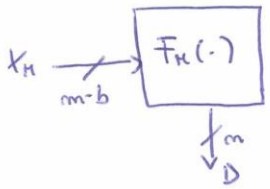


con $B = 2^b$

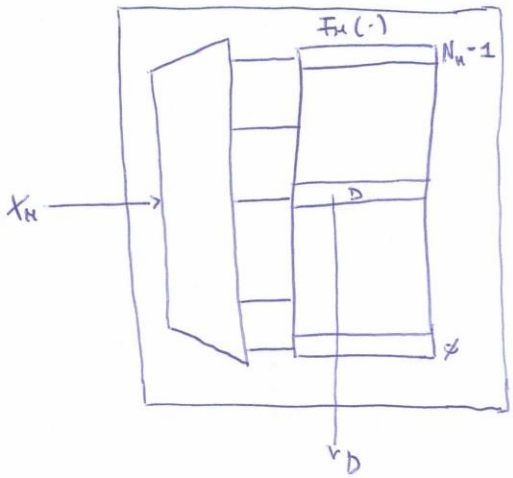
$C = 2^c$

MODELLO BLACK-BOX DI MEMORIA E CACHE

MEMORIA



$\Rightarrow$

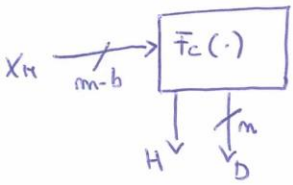


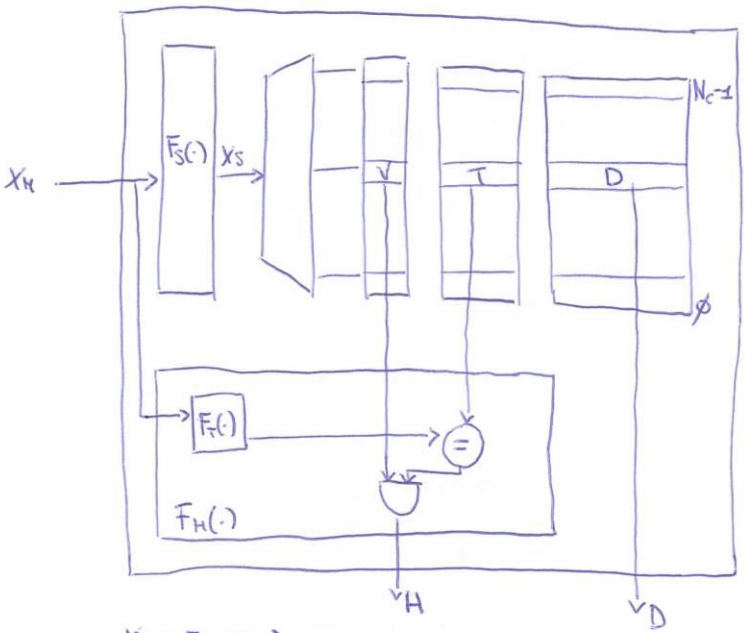
$X_M = \frac{X}{B}$: indirizzo del blocco di memoria

F_M : funzione di prelievo

$D = F_M(X_M)$

CACHE





$X_M = \frac{X}{B}$: indirizzo del blocco di memoria

F_C : funzione di prelievo del blocco di cache

$$D = \begin{cases} F_C(X_M) & \text{se } H = 1 \\ \text{niente} & \text{se } H = \emptyset \end{cases}$$

H : segnale di HIT

$$H = \begin{cases} \emptyset & : \text{miss} \\ 1 & : \text{Hit} \end{cases}$$

$H = F_H(X_M)$

F_H : Funzione di Hit

$X_S = F_S(X_M)$: indirizzo in cache

$X_T = F_T(X_M)$: tag del blocco

h: probabilità di hit

$m = 1 - h$: probabilità di miss

t_{hit} : tempo di accesso in caso di hit

t_{miss} : tempo di accesso in caso di miss

$t_{penalty} = t_{miss} - t_{hit}$

AMAT (Average Memory Access Time)

$AMAT = h \times t_{hit} + m \cdot t_{miss} = t_{hit} + m \cdot t_{penalty}$

CACHE ASSOCIATIVE

Stesso stesso indirizzo possono risiedere più blocchi di dato

A: grado di associatività della cache (# di vie).

N_S: numero di set $= \frac{C}{B \cdot A}$

$X_S = X_H \,\%\, N_S$: indirizzo del set

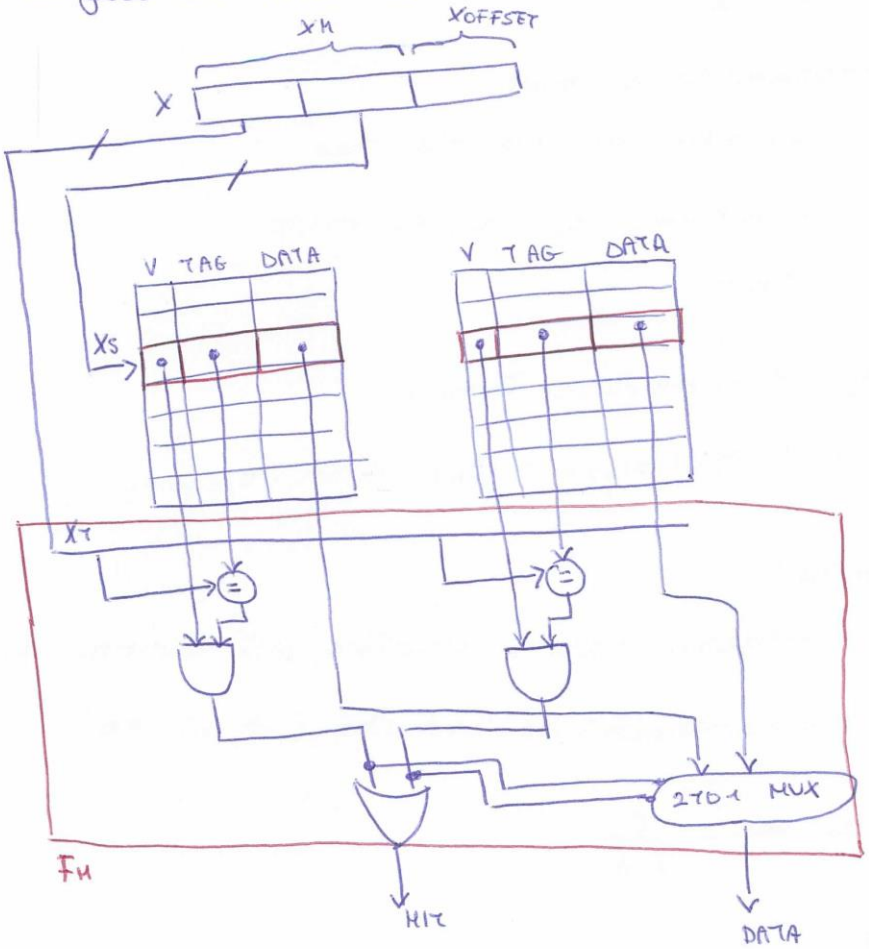
$X_T = X_H / N_S$: tag

Cache Full Associative : $N_S = 1$.

Funzione di Hit

F_H : $(\forall K=1,\dots,A \;,\; X_T == Y_{T,K} \;\&\&\; Y_{V,K} == 1 \;?\; 1 : \emptyset)$

Schema Logico di una Cache associativa a 2 set



Politica di Rimpiazzamento:

RANDOM : scelgo un blocco a caso;

LRU : scelgo il blocco usato meno recentemente;

FIFO : scelgo il blocco caricato meno recentemente;

Politica di Gestione scrittura su HIT:

WAITE BACK:

1) hit: scrivo in cache e metto il bit di modifica M=1;

2) in caso di rimpiazzamento copio il blocco in memoria;

WAITE THROUGH

1) hit: scrittura sia in cache che in memoria;

Politica di Gestione Scrittura su MISS:

WRITE ALLOCATE

1) miss;
2) leggo blocco dalla memoria e scrivo in cache;

WRITE NOT ALLOCATE

1) miss e scrittura in memoria;

AMAT per cache a 2 livelli:

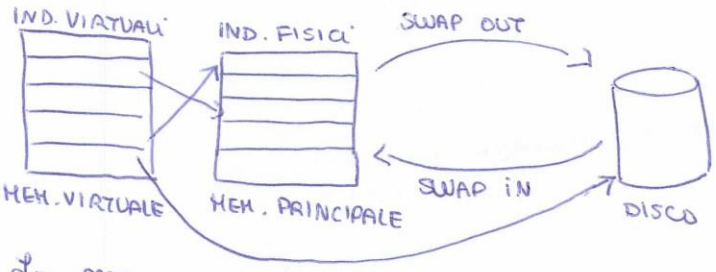
$$AMAT = t_{hit_1} + m_1 \left(t_{hit_2} + m_2 \cdot t_{penalty_2} \right)$$

MEMORIA VIRTUALE

DISK CACHING : utilizzo della memoria principale come se fosse la cache del disco

MEM. VIRTUALE : Uso uno spazio di indirizzamento più ampio

↳ Vantaggi :
- Gestione di un enorme quantità di memoria ;
- Rilocazione programmi non necessaria ;
- Protezione dei programmi ;



L'unità di trasferimento è la pagina.

La memoria virtuale si comporta come una cache fully associativa con politica di Rimpiazzamento LRU, politica di Scrittura WB e di Scrittura ad MISS WA.

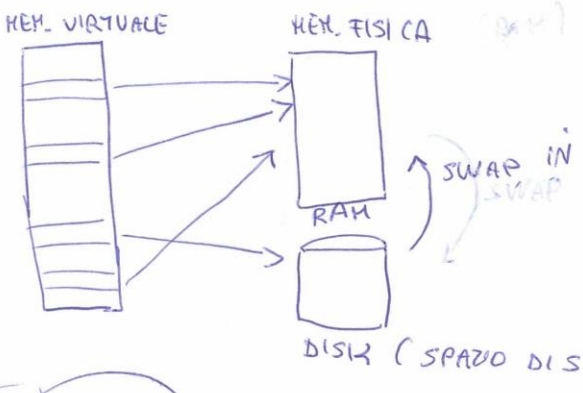
Working Set : insieme delle pagine che è necessario mantenere in memoria per avere un HIT rate accettabile ;

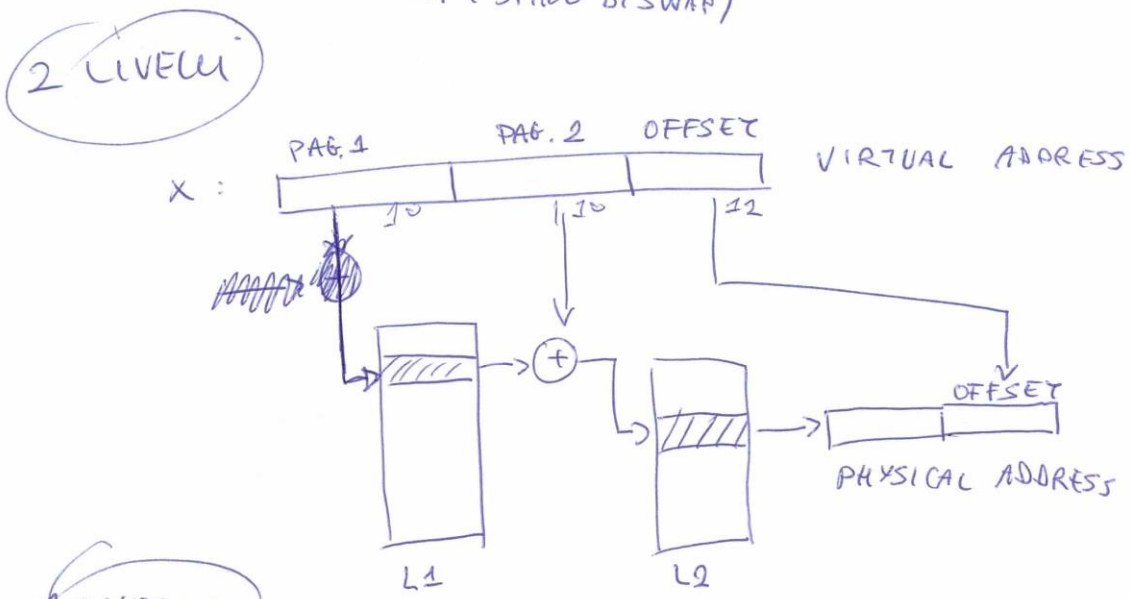
Thrashing : Perdita di prestazione causata da un working set troppo grande rispetto alla mem. principale ;

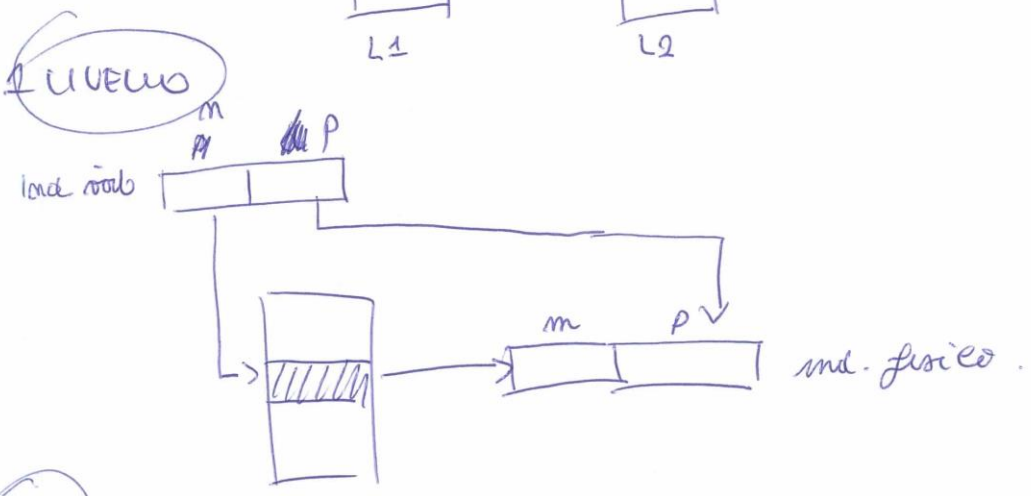
Page Swap : Trasferimento di una nuova pagina in mem. principale ;

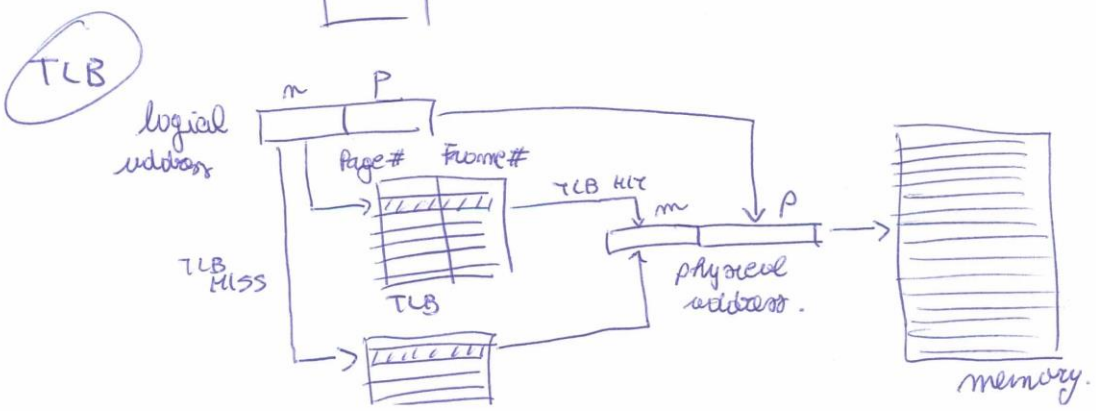
La CPU si riferisce alla memoria con un indirizzo virtuale che viene poi tradotto in indirizzo fisico.
La funzione di MAPPING è una tabella

V	D	U	IDENTIFICATORE DI PAG. FISICA


MEM. VIRTUALE
MEM. FISICA
SWAP IN
RAM
DISK (SPAZIO DI SWAP)


2 LIVELLI
X :
PAG. 1
PAG. 2
OFFSET
VIRTUAL ADDRESS
10
10
12
OFFSET
PHYSICAL ADDRESS
L1
L2


1 LIVELLO
m
p
ind. virt.
m
p
ind. fisico.


TLB
logical address
m
p
Page#
Frame#
TLB HIT
m
p
TLB MISS
TLB
physical address.
memory.

V: Validity : mi dice se la pagina è in memoria o no

D: Dirty : mi informa se la pagina è stata modificata

U: Usage : mi informa se è stata utilizzata.

Se la pagina cercata non è in memoria, la routine di servizio PAGE-FAULT HANDLER si occupa di prelevare tale pagina dal disco nella memoria.

TECNICHE PER RIDURRE LA TABELLA DELLE PAGINE

1) Paginazione della tabella delle pagine

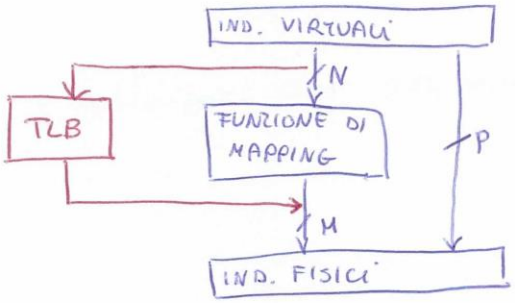
-> Parte della tabella ata in mem. virtuale anziché in mem. fisica. Sta in mem. fisica solo la parte più usata

2) Tabella delle Pagine Inversa

-> Si applica una funzione di Hashing all'indirizzo virtuale così che il numero di entry nella tabella delle pagine è al più pari al numero totale di pagine fisiche.

TLB (TRANSLATION LOOKASIDE BUFFER)

Piccola cache per ridurre la disposizione un pò di elementi della tabella delle pagine



PIPELINE

I PASSI DI UN'ISTRUZIONE

F: INSTRUCTION FETCH

- Letta dalla memoria l'istruzione puntata da PC;
- Posta tale istruzione nell'Instruction Register;
- PC aumentato di 4 e inserito in PC;

D: INSTRUCTION DECODE e REGISTER FETCH

- Il contenuto dei registri rs e rt vengono trasferiti nei registri di appoggio A e B;
- Nel caso sia un istruzione di salto, calcolo l'indirizzo target;

X: EXECUTE

- In base al tipo di istruzione la ALU esegue:
  - accesso alla memoria;
  - un'operazione;
  - un salto;

M: MEMORY ACCESS

- Istruzioni load/store accedono alla memoria;
- Le istruzioni R scrivono il risultato;

W: WRITE BACK

- Il risultato viene aggiornato nel registro;

LOAD: F D X M W

STORE: F D X M

R : F D X W

BEQ : F D X

CRITICITÀ

STRUTTURALI : tentativo di utilizzare le stesse risorse due volte allo stesso tempo.

modi per risolvere:

1) Inserire un BOLLA;

2) Rendere tutte le istruzioni a 5 stadi;

SUI DATI : tentativo di utilizzare un dato prima che sia prodotto;

TIPI:

- RAW: Read After Write
- WAR: Write After Read
- WAW: Write After Write

Le dipendenze di tipo RAW sono dipendenze "inviolate nel tempo". Uso il FORWARDING.

SUL CONTROLLO: tentativo di evitare pure evendo eseguito istruzioni del ramo non preso;

Supponendo che la logica di decisione dei salti sia nello stadio EXECUTE allora due istruzioni in più dopo il salto saranno prelevate.

Soluzione:

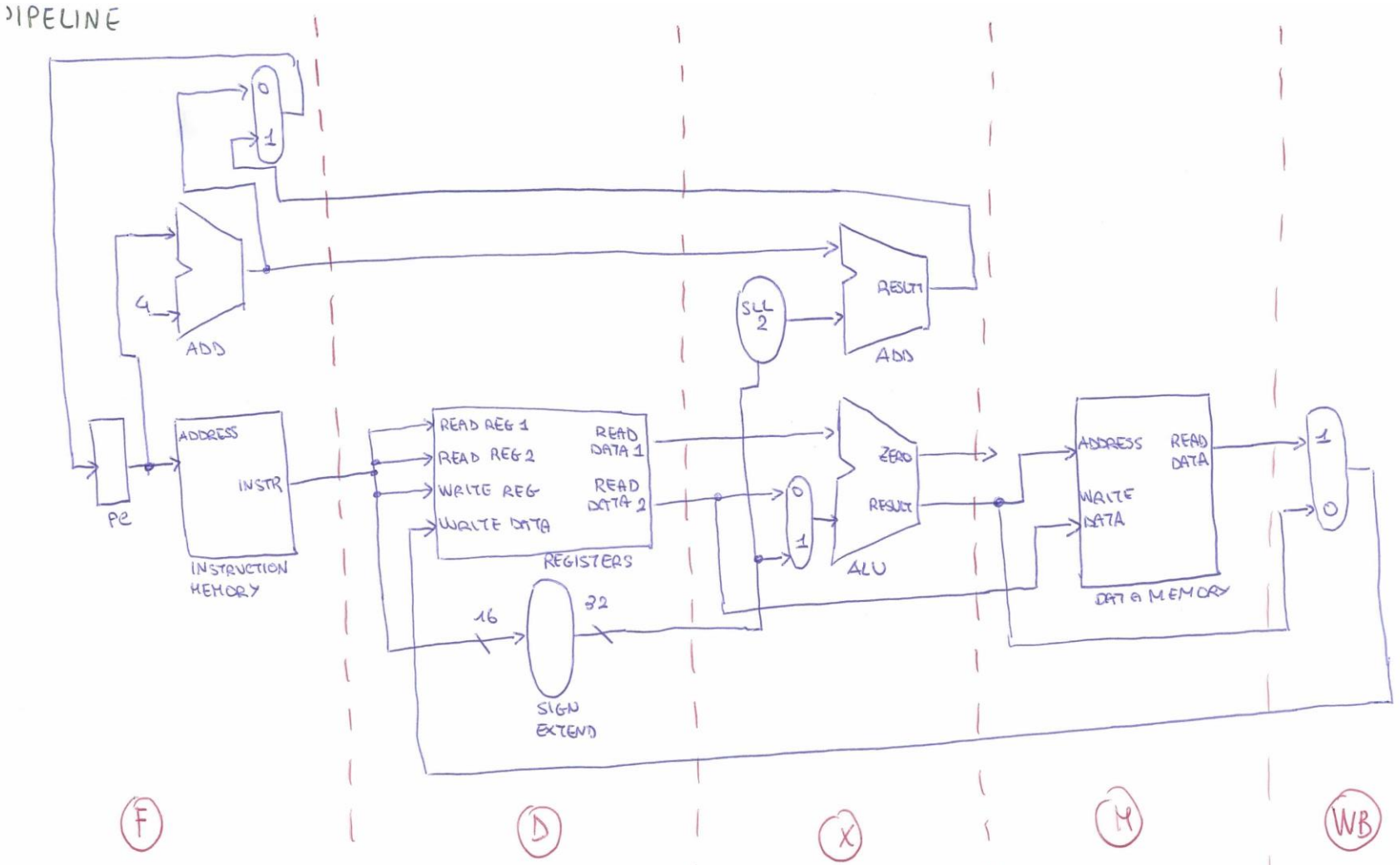
1) inserire 2 nop dopo il salto;

2) Spostare la decisione nello stadio D ed eseguire una sola nop.

3) ridefinire il salto:

BRANCH-DELAY SLOT: Sia che si faccia, che non si faccia, il salto, eseguirà l'istruzione successiva.

-> Riordinamento del codice!



PORTE LOGICHE

AND

$Y = X_1 \cdot X_2$

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

$Y = X_1 + X_2$

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

$Y = \overline{X}$

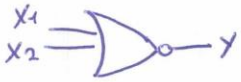
X	Y
0	1
1	0

NAND

$Y = \overline{X_1 \cdot X_2}$

X_1	X_2	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR



$Y = \overline{X_1 + X_2}$

X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR

$Y = X_1 \oplus X_2$

$= \overline{X_1} X_2 + X_1 \overline{X_2}$

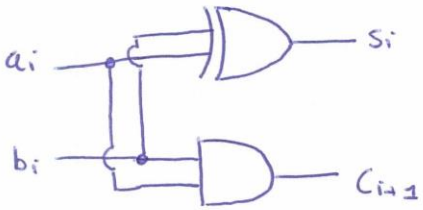
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

TEOREMA DI DE MORGAN

$$\overline{(x+y)} = \bar{x} \cdot \bar{y}$$

$$\overline{(x \cdot y)} = \bar{x} + \bar{y}$$

HALF ADDER (Somma di due bit senza riporto in ingresso)

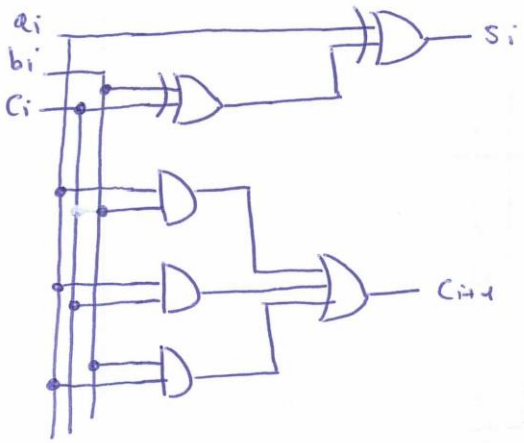


a_i	b_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S_i = a_i \oplus b_i$$

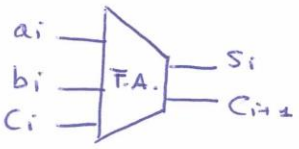
$$C_{i+1} = a_i \cdot b_i$$

FULL ADDER



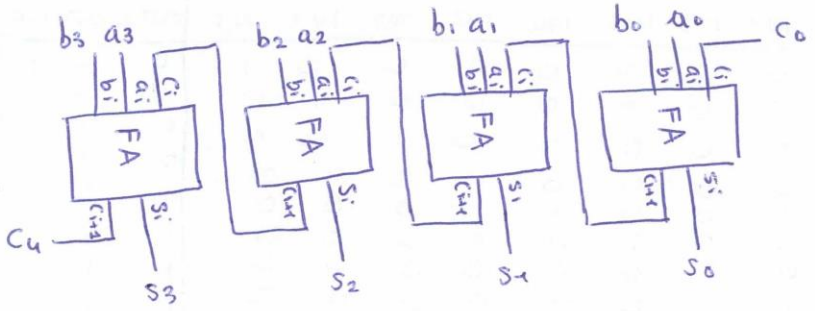
$$S_i = a_i \oplus b_i \oplus c_i$$

$$C_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$



C_i	a_i	b_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

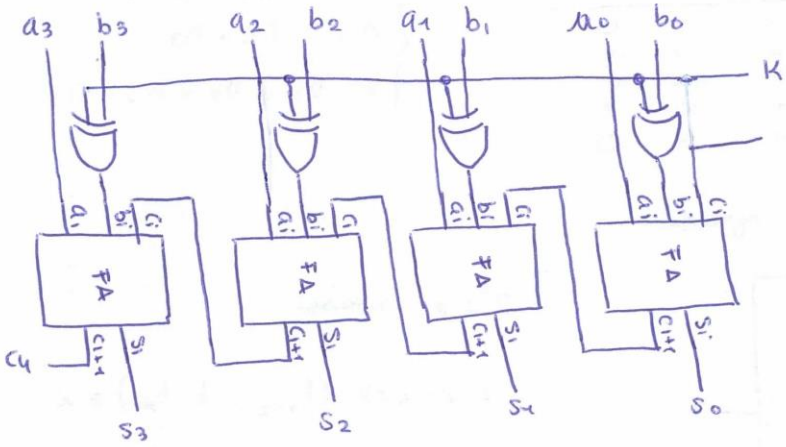
RIPPLE CARRY ADDER



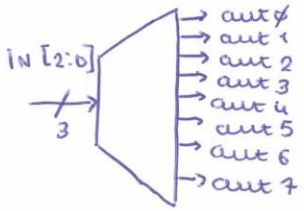
SOMMATORE - SOTTRATTORE

A - B => K = 1

A + B => K = 0

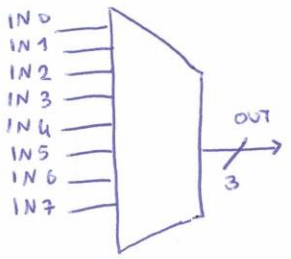


DECODER



INPUTS			OUTPUTS							
IN2	IN1	IN0	out 7	out 6	out 5	out 4	out 3	out 2	out 1	out 0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

ENCODER



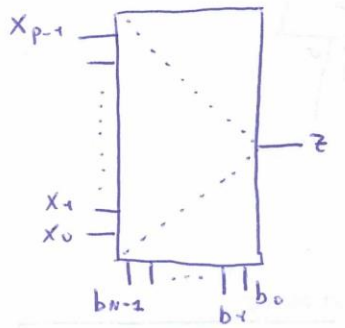
IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	out2	out1	out0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

PRIORITY ENCODER

Input				Output		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

$$\begin{cases} A_0 = D_3 + D_1\bar{D}_2 \\ A_1 = D_2 + D_3 \\ V = D_0 + D_1 + D_2 + D_3 \end{cases}$$

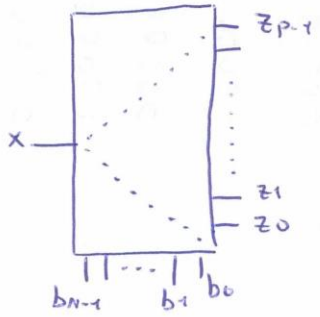
MULTIPLEXER 2^N ingressi



$2^N = p$ ingressi

$z = X_i \iff (b_{N-1} \dots b_1 b_0) \equiv i$

DEMULTIPLEXER 2^N uscite



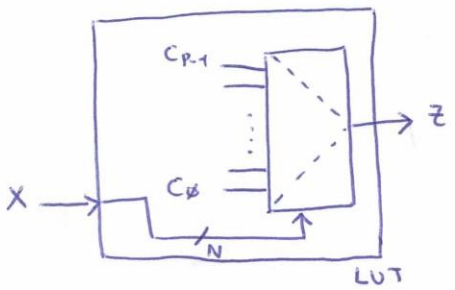
$2^N = p$ uscite

$z_0 = X \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0}$

$z_1 = X \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot b_0$

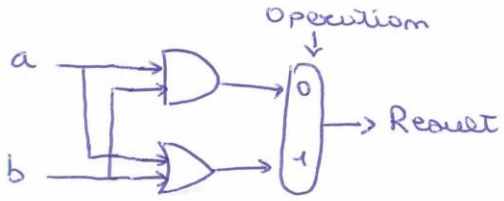
$\vdots$

LOOK UP TABLE (LUT)

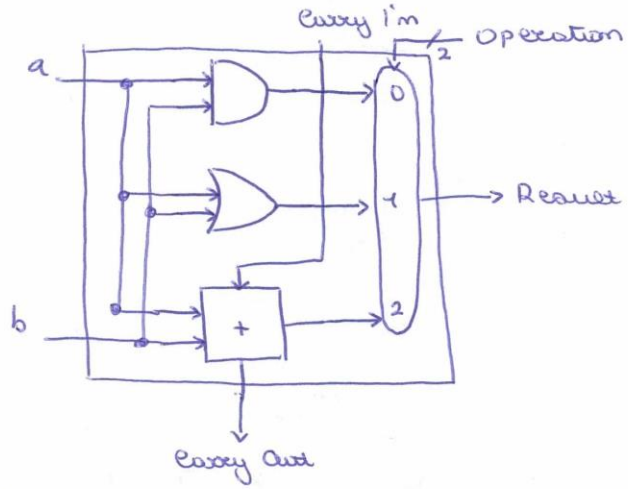


MUX con ingressi costanti

ALU 1 bit (AND - OR)

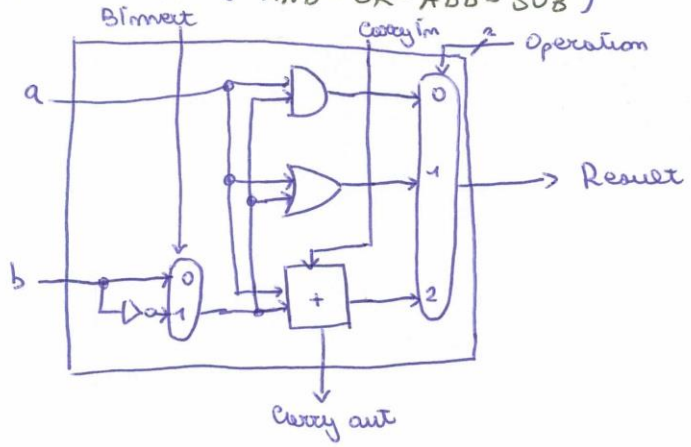


ALU 1 bit (AND - OR - ADD)

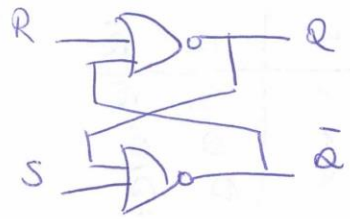


Operation	Function
0	AND
1	OR
2	ADD

ALU 1 bit (AND - OR - ADD - SUB)

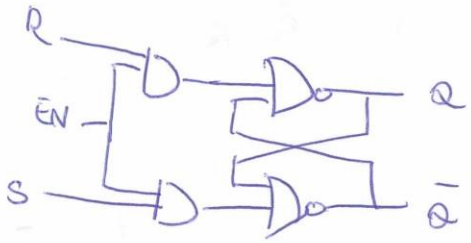


LATCH SR



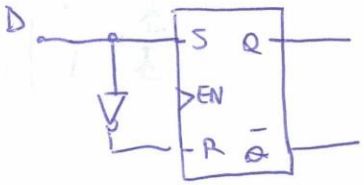
S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	-	-

LATCH SR com ENABLE



EN	S	R	Q	$\bar{Q}$
0	x	x	Q	$\bar{Q}$
1	0	0	Q	$\bar{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	-	-

LATCH D



EN	D	Q	$\bar{Q}$
0	x	Q	$\bar{Q}$
1	0	0	1
1	1	1	0

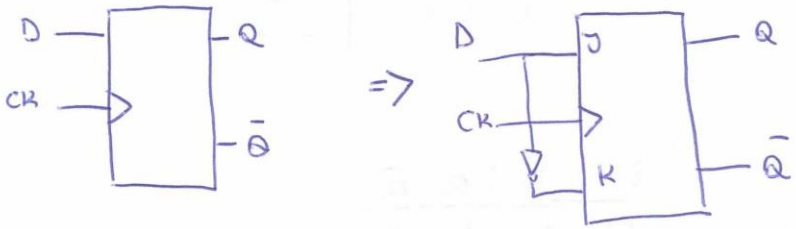
FLIP FLOP SR ↓

CK	S	R	Q	$\bar{Q}$
0	x	x	Q	$\bar{Q}$
1	x	x	Q	$\bar{Q}$
↓	x	x	Q	$\bar{Q}$
↓	0	0	Q	$\bar{Q}$
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	-	-

FLIP FLOP JK ↓

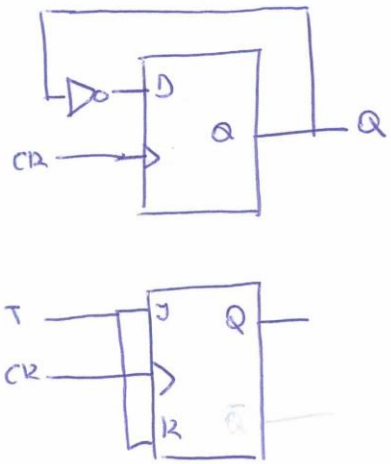
CK	J	K	Q	$\bar{Q}$
0	x	x	Q	$\bar{Q}$
1	x	x	Q	$\bar{Q}$
↓	x	x	Q	$\bar{Q}$
↓	0	0	Q	$\bar{Q}$
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	$\bar{Q}$	Q

FLIP FLOP D ↓



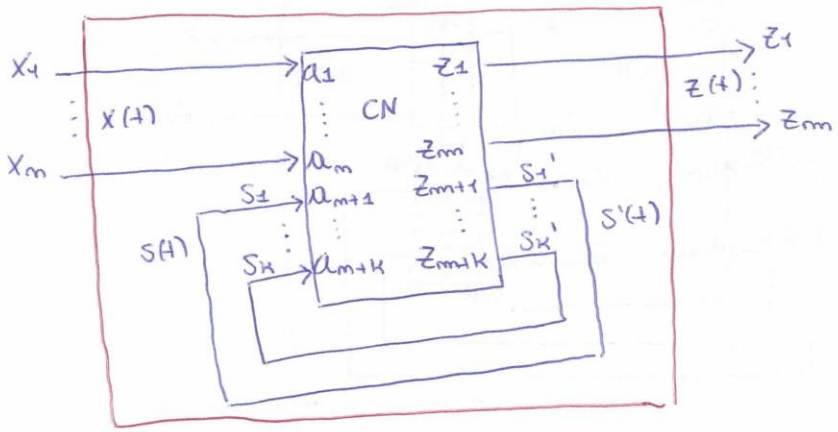
CK	D	Q	$\bar{Q}$
0	x	Q	$\bar{Q}$
1	x	Q	$\bar{Q}$
↓	x	Q	$\bar{Q}$
↓	0	0	1
↓	1	1	0

FLIP FLOP T ↓



CK	T	Q
0	x	Q
1	x	Q
↓	x	Q
↓	0	Q
↓	1	$\bar{Q}$

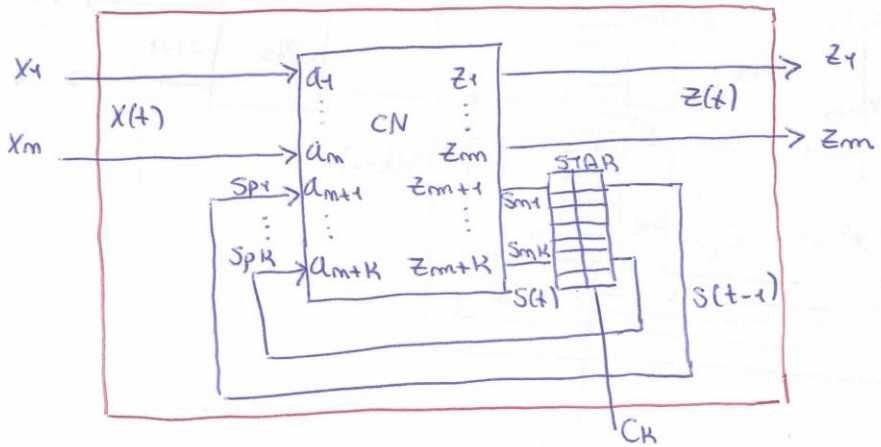
MACCHINA DI MEALY ASINCRONA



Le variabili di uscita, in un determinato istante, sono funzione del valore degli ingressi e della variabile di stato.

$$Z(t) = F(X(t), S(t))$$
$$S'(t) = F(X(t), S(t))$$

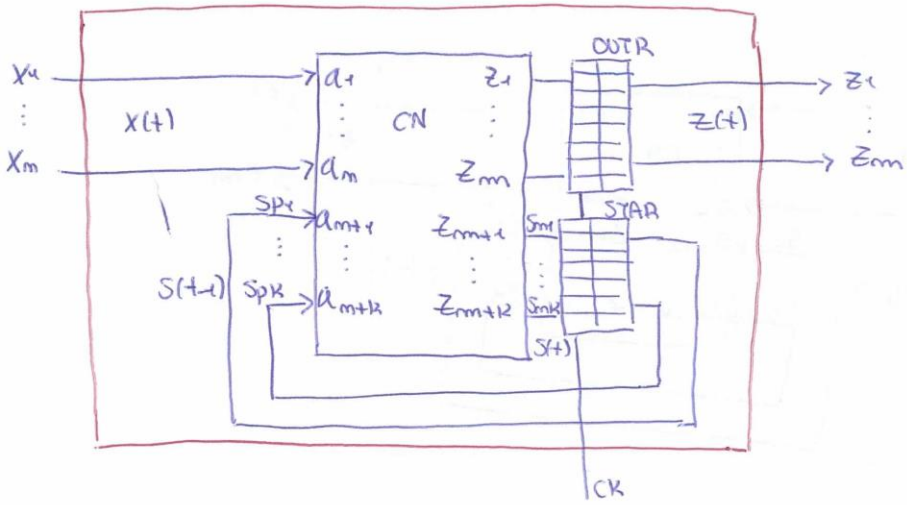
MACCHINA DI MEALY SINCRONA



$$Z(t) = f(X(t), S(t-1))$$
$$S(t) = g(X(t), S(t-1))$$

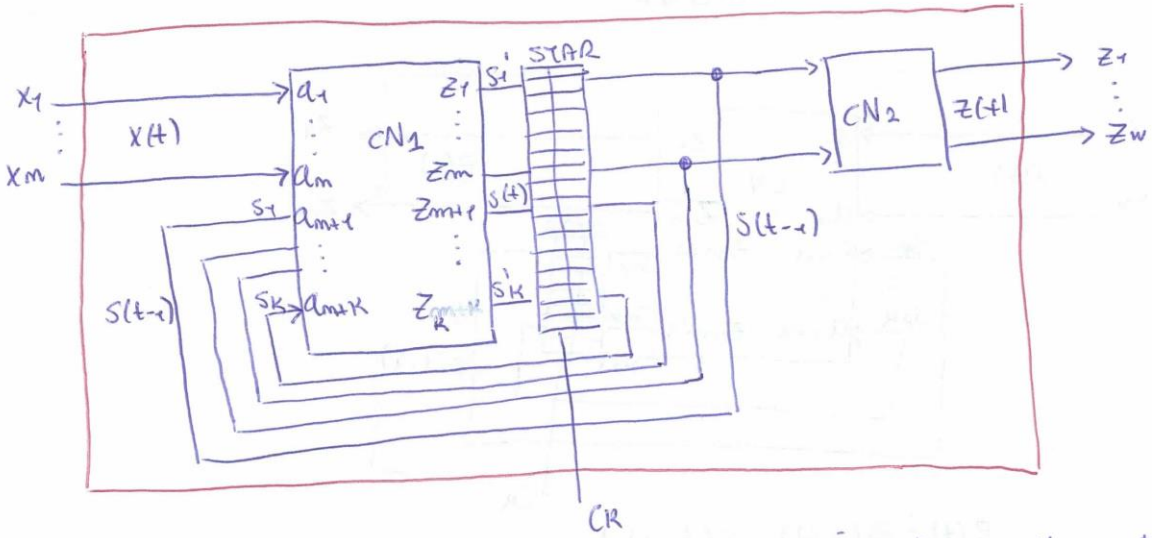
STAR è costituito da un FLIP FLOP D.

MACCHINA DI MEALY SINCRONA RITARDATA



$$Z(t) = f(X(t-1), S(t-1))$$
$$S(t) = g(X(t), S(t-1))$$

MACCHINA DI MOORE SINCRONA

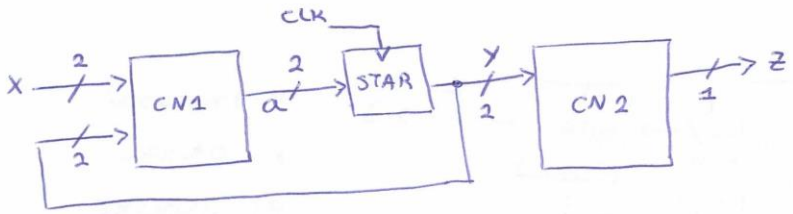


Le variabili di uscita, in un determinato istante, sono funzione delle variabili di stato.

$$Z(t) = f(S(t-1))$$
$$S(t) = g(X(t), S(t-1))$$

S(t-1): stato presente ; S(t): stato successivo

RICONOSCITORE DI SEQUENZA 11,01,10 CON MOORE



X: ingressi
y: stati
z: uscite

STATO	y_1 y_0
S0	0 0
S1	0 1
S2	1 1
S3	1 0

CN1

Y_1Y_0 \ X_1X_0	00	01	11	10
S0	S0	S0	S1	S0
S1	S0	S2	S1	S0
S2	S0	S0	S1	S3
S3	S0	S0	S1	S0

⇓

Y_1Y_0 \ X_1X_0	00	01	11	10
00	00	00	01	00
01	00	11	01	00
11	00	00	01	10
10	00	00	01	00

a_1 a_0

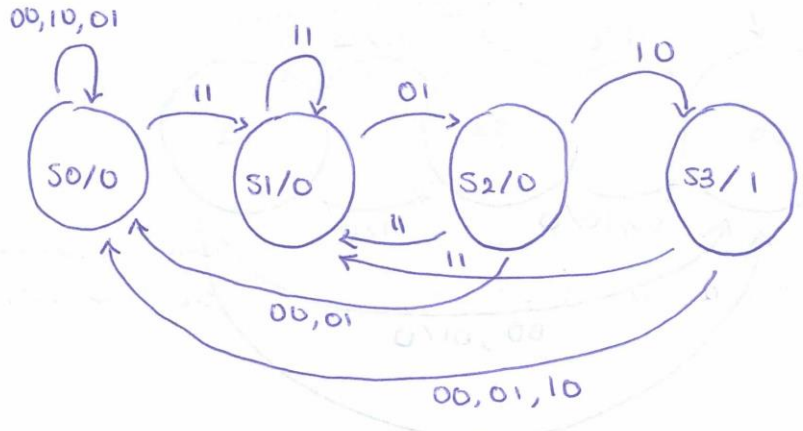
$a_0 = Y_1X_0 + X_0\bar{Y_1}Y_0$

$a_1 = \bar{X_1}X_0Y_1\bar{Y_0} + X_1\bar{X_0}Y_1Y_0$

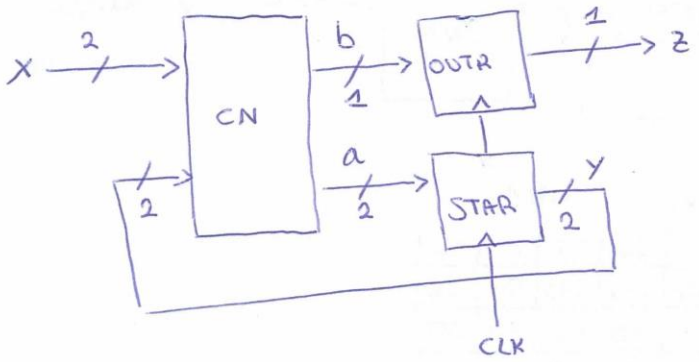
CN2

Y_1Y_0	z
0 0	0
0 1	0
1 1	0
1 0	1

$z = Y_1\bar{Y_0}$



RICONOSCITORE DI SEQUENZA CON MEALY (11, 01, 10)



x: ingressi
y: stati
z: uscite

STATO	Y_1 Y_0
S0	0 0
S1	1 1
S2	0 1

Y_1Y_0 \ X_1X_0	00	01	11	10
00	00	00	11	00
01	00	00	11	00
11	00	01	11	00
10	-	-	-	-

a_1 a_0

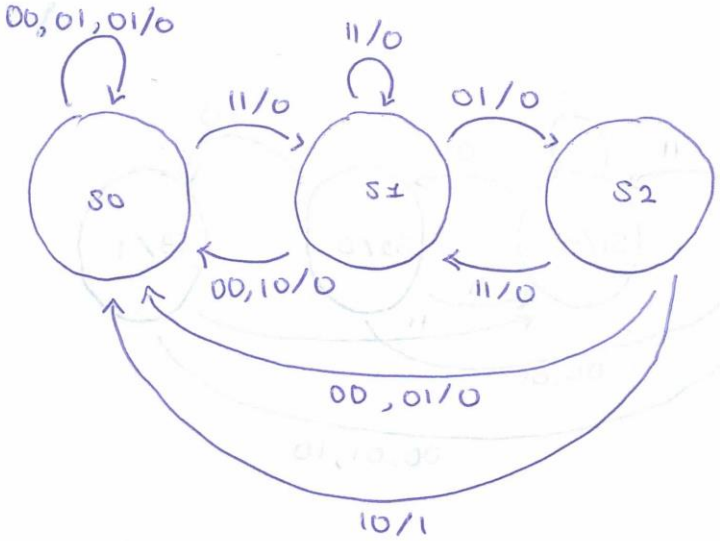
$a_0 = X_1X_0 + X_0\bar{y}_1$

$a_1 = X_1X_0$

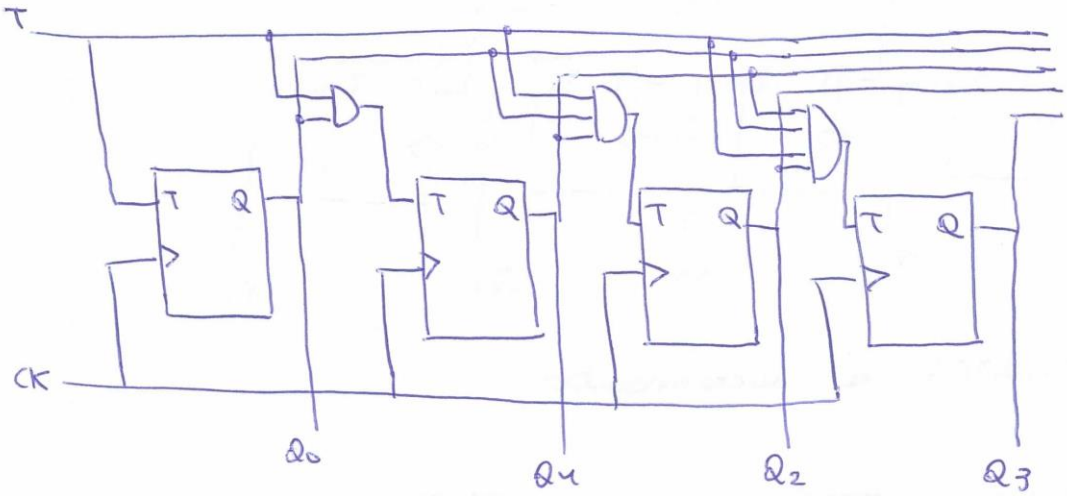
Y_1Y_0 \ X_1X_0	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	0	0	0	0
10	-	-	-	-

$b_0 = X_1\bar{X}_0\bar{y}_1y_0$

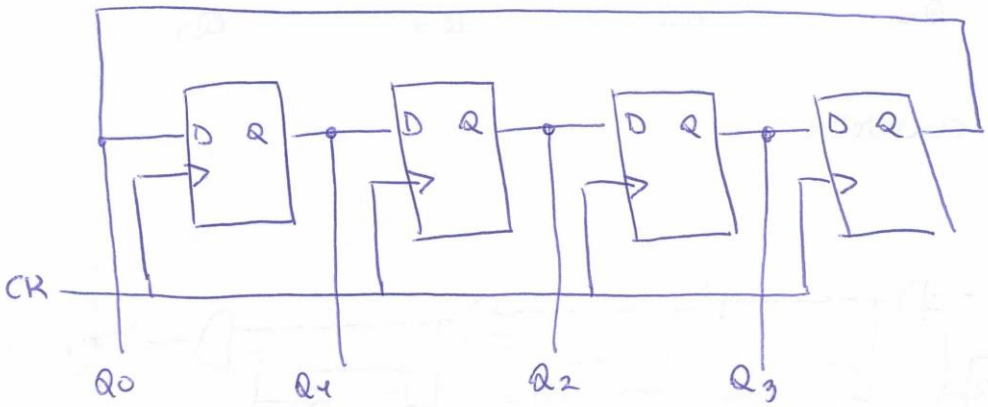
$b_0 = z$



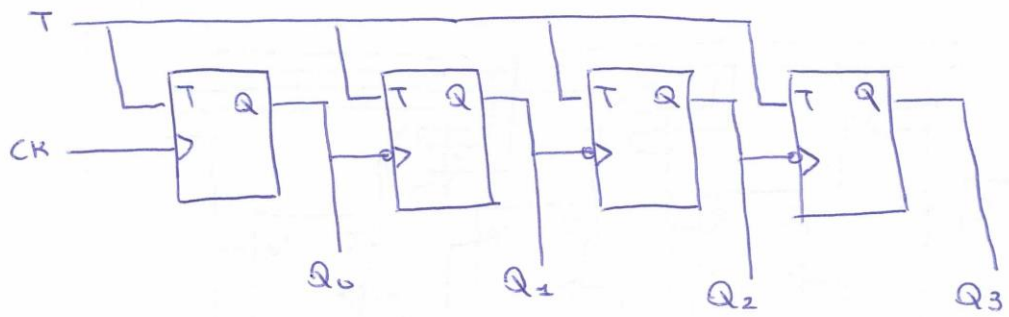
PARALLEL CARRY COUNTER



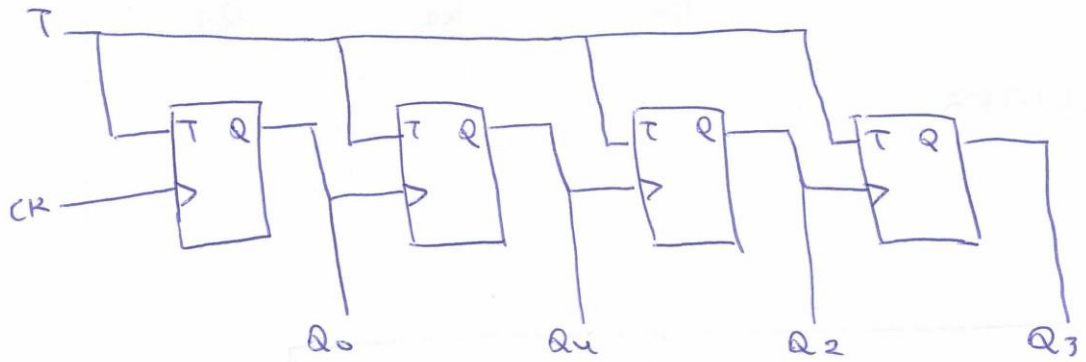
RING COUNTER



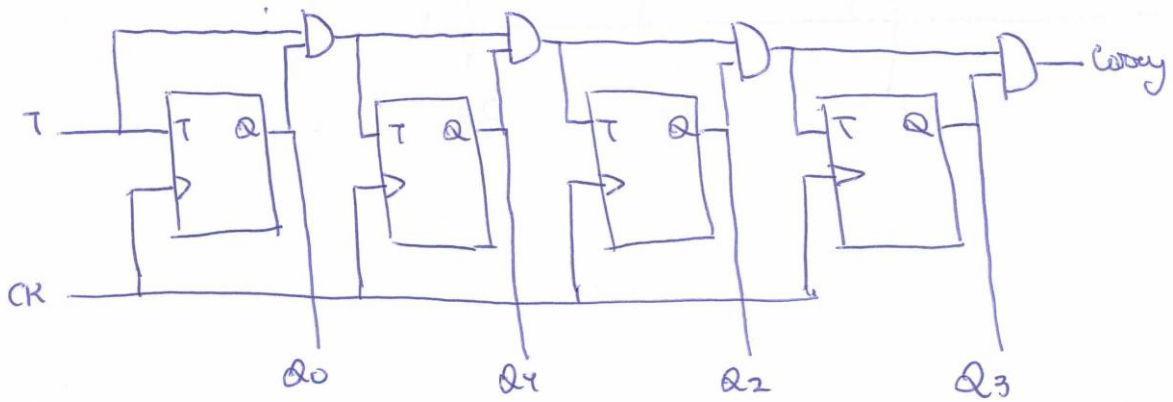
RIPPLE COUNTER ad incremento



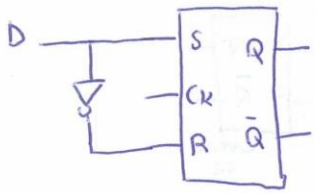
RIPPLE COUNTER a decremento



SERIAL CARRY COUNTER

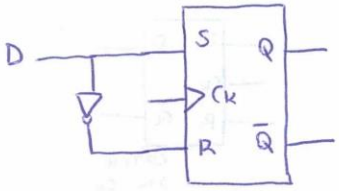


LATCH D



D	Ck	Q
x	0	Q
0	1	0
1	1	1

LATCH D EDGE TRIGGERED

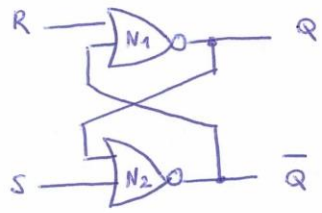


D	Ck	Q
x	1	Q
x	0	Q
x	⌊	Q
0	⌋	0
1	⌋	1

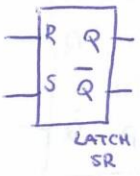
LATCH : Uscita varia non appena cambia l'ingresso e il clock ha valore alto ;

F. FLOP : Uscita cambia solo all'fronte del clock

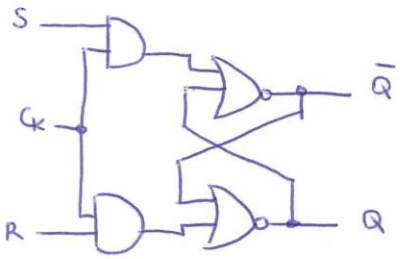
LATCH SR



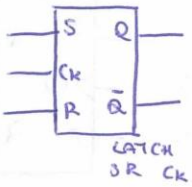
S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	-	-



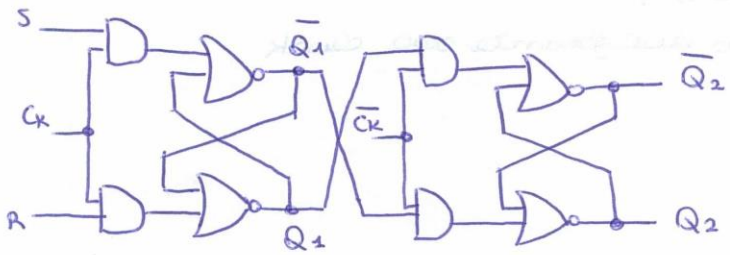
LATCH SR CLOCCATO



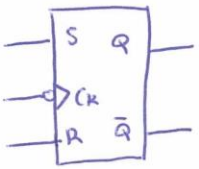
Ck	S	R	Q	$\bar{Q}$
1	0	0	Q	$\bar{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	-	-
0	X	X	Q	$\bar{Q}$



LATCH SR EDGE TRIGGERED (Sensibile ad fronte di discesa)



Ck	S	R	Q	$\bar{Q}$
⌊	0	0	Q	$\bar{Q}$
⌊	0	1	0	1
⌊	1	0	1	0
⌊	1	1	-	-
0	X	X	Q	$\bar{Q}$
1	X	X	Q	$\bar{Q}$
⌋	X	X	Q	$\bar{Q}$



19/04/2015

ES. 1

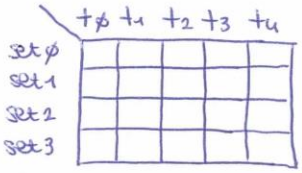
C = 160
A = 5
B = 8

$T_{hit} = 4\,ms$

$T_{miss} = 40\,ms$

$N_c = \frac{C}{B} = 20$

$N_S = \frac{C}{B \cdot A} = 4$



$X_M = \frac{X}{B}$; $X_T = X_M / N_S$; $X_S = X_M \,\%\, N_S$

T	X	X_M	X_T	X_S
R	123	15	3	3
W	639	79	19	3
R	327	40	10	0
W	679	84	21	0
R	878	109	27	1
W	639	79	19	3
R	133	16	4	0
W	654	81	20	1
R	125	15	3	3
W	454	56	14	0
R	122	15	3	3
W	654	81	20	1
R	939	117	29	1
W	626	78	19	2
R	954	119	29	3
W	724	90	22	2
R	254	31	7	3
W	829	103	25	3
R	154	19	4	3
W	828	103	25	3
R	194	24	6	0

X_S : MODIF

3 : 4 0 0 0 0
3 : 3 4 0 0 0
0 : 4 0 0 0 0
0 : 3 4 0 0 0
1 : 4 0 0 0 0
3 : 3 4 0 0 0
0 : 2 3 4 0 0
1 : 3 4 0 0 0
3 : 4 3 0 0 0
0 : 2 3 4 0 0
3 : 4 3 0 0 0
1 : 3 4 0 0 0
1 : 2 3 4 0 0
2 : 4 0 0 0 0
3 : 3 2 4 0 0
2 : 3 4 0 0 0
3 : 2 4 3 4 0
3 : 4 0 2 3 4
3 : 0 4 1 2 3
3 : 0 3 1 2 4
0 : 1 2 3 4 0

X_S : TAB

3 : 3 - - - - MISS
3 : 3 19 - - - MISS
0 : 10 - - - - MISS
0 : 10 21 - - - MISS
1 : 27 - - - - MISS
3 : 3 19 - - - HIT
0 : 10 21 4 - - MISS
1 : 27 20 - - - MISS
3 : 3 19 - - - HIT
0 : 10 21 14 - - MISS
3 : 3 19 - - - HIT
1 : 27 20 - - - HIT
1 : 27 20 29 - - MISS
2 : 19 - - - - MISS
3 : 3 19 29 - - MISS
2 : 19 22 - - - MISS
3 : 3 19 29 7 - MISS
3 : 3 19 29 7 25 MISS
3 : 3 4 29 7 25 MISS
3 : 3 4 29 7 25 HIT
0 : 10 21 14 6 - MISS

out: $X_M = 79$
$X_S = 3$
$X_T = 19$

$A_{TAH} = T_{HIT} + M_{RATE} \cdot T_{PEN}$

$M_{RATE} = \frac{n.\ miss}{n.\ ref}$

17/04/2015

Es. 2

$68,7_{10} = 68,\overline{571428}$ → Double Precision

- Segno: 1 bit
- exp: 11 bit
- mantissa: 52 bit
- BIAS: 1023

Segno = 0

Parte intera ↓

68	2
34	0
17	0
8	1
4	0
2	0
1	0
0	1

→ $1000100 = 1.000100 \cdot 2^6$

$exp = 1023 + 6 = 1029$

$exp_2 = 10000000101$

1029	2
514	1
257	0
128	1
64	0
32	0
16	0
8	0
4	0
2	0
1	0
0	1

Parte decimale

0,571428	* 2
1,142856	1
0,285712	0
0,571424	0
1,142848	1
0,285696	0
0,571392	0
1,142784	1
0,285568	0
⋮	

presuppongo periodicità.

0|10000000101|0001001001001001001001001001001001001001001001|0

53-esima cifra è 0

OK! ✓

17/04/2015

ES. 4

```
lw   $5, 40($2)
add  $6, $3, $2
or   $7, $2, $1
and  $8, $4, $3
sub  $9, $2, $1
```

PC = 100

$x = 10 + x

Memory[y] = 1000 + y

		1	2	3	4	5
I/F	PC	100	104	108	112	116
F/D	ID		lw $5, 40($2)	add $6, $3, $2	or $7, $2, $1	and $8, $4, $3
D/X	IX			lw	add	or
	A			12	26	22
	B/imm			40	23	24
	RD			5	6	7
	RS			2	3	2
	RT			-	2	4
X/M	R/W				R	-
	MAR/Res				52	3
	DIN → per la sw				-	-
	RT/RD				5	6
M/W	Dout					1052
	RES → Risultato di una add					-
	RD					5

20/02/2025

ES. 2

$460/7_{10} = 65,\overline{714285}$ in Single Precision

- Segno: 1 bit
- exp: 8 bit
- mantissa: 23 bit
- BIAS = 127

Segno = 0

65	2
32	1
16	0
8	0
4	0
2	0
1	0
0	1

-> Parte intera

$1000001 = 1.000001 \cdot 2^6$

$127 + 6 = 133$

exp = 10000101

133	2
66	1
33	0
16	1
8	0
4	0
2	0
1	0
0	1

0,714285	*2
1,428570	1
0,857140	0
1,714280	1
1,428560	1
0,857120	0
⋮	

-> 10110110110110110110110

0|10000101|00000110110110110110110

↳ $\frac{460}{7}_{10} = 0|10000101|00000110110110110110111_2$

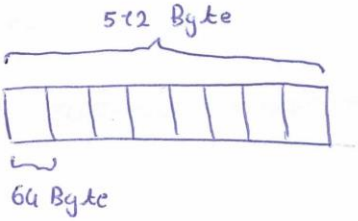
15/12/2009

ES. 1

```
int i, j, c, stride, array[1024];
for (i=0; i<5000; i++)
    for (j=0; j<1024; j=j+stride)
        c = array[j] + 17;
```

CACHE:

- Blocchi da 64 Byte;
- Grande 512 Byte



Consideriamo solo l'attività sulla cache causata da array.

Int = 4 Byte

$stride_1 = 256$

$stride_2 = 255$

Miss Rate ?

Con $Stride_1$ ho i seguenti accessi:

array[0] = a

array[256] = a + 256 · 4 = a + 1024

array[512] = a + 512 · 4 = a + 2048

array[768] = a + 768 · 4 = a + 3072

Supponiamo come indirizzo di array[0] -> a

Ma i risultati della Cache non dipendono da a

-> a = 0

Con stride 2 ho i seguenti accessi:

array [0] = a + 0

array [255] = a + 255 · 4 = a + 1020

array [510] = a + 510 · 4 = a + 2040

array [765] = a + 765 · 4 = a + 3060

array [1020] = a + 1020 · 4 = a + 4080

-> a = 0

Tali accessi vengono ripetuti 5.000 volte.

CASO 1)

Stride = 256

A = numero di vie = 1

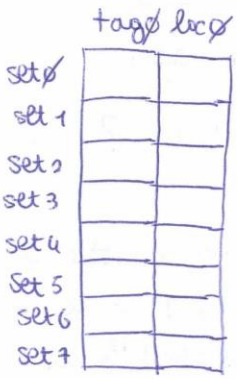
B = dim. blocco cache = 64

C = capacità cache = 512

$N_c = \frac{C}{B}$ = # blocchi = 8 -> $N_s = \frac{C}{B \cdot A} = N_c$

↓

set della cache



X : indirizzo in Byte

$X_M = \frac{X}{B}$ memoria

$X_S = X_M \% N_S$ set

$X_T = X_M / N_S$ tag

X	X_M	X_T	X_S	
0	0	0	0	MISS
1024	16	2	0	MISS
2048	32	4	0	MISS
3072	48	6	0	MISS
⋮	⋮	⋮	⋮	⋮

Ho sempre MISS!

$m_1 = 1$

CASO 2)

Stride 255

X	X_H	X_T	X_S	
0	0	0	0	MISS
1020	15	1	7	MISS
2040	31	3	7	MISS
3060	47	5	7	MISS
4080	63	7	7	MISS
0	0	0	0	HIT
1020	15	1	7	MISS
2040	31	3	7	MISS
⋮	⋮	⋮	⋮	

$N_{REF} = 5000 \cdot 5 = 25.000$

$N_{MISS} = 5000 \cdot 4 + 1 = 20.001$

$m_1 = \frac{N_{MISS}}{N_{REF}} = 0,80004$

CASO 3)

A=2

B=64

C=512

$N_C = 8$

$N_S = 4$

$X_H = \frac{X}{B}$; $X_S = X_H \% N_S$; $X_T = X_H / N_S$

X	X_H	X_T	X_S	LRU
0	0	0	0	1 0
1024	16	4	0	0 1
2048	32	8	0	1 0
3072	64	12	0	0 1
0	0	0	0	1 0
1024	16	4	0	0 1
2048	32	8	0	1 0
3072	64	12	0	0 1
⋮				

TAG		
0	-	MISS
0	4	MISS
8	4	MISS
8	12	MISS
0	12	MISS
0	4	MISS
8	4	MISS
8	12	MISS

	tag ϕ	loc ϕ	tag ϕ	loc 1
set ϕ				
set 1				
set 2				
set 3				

$m_3 = 1$

CASO 4)

$A = 2$ $\quad N_C = 8$

$B = 64$ $\quad N_S = 4$

$C = 512$

X	X_H	X_T	X_S	LRU	TAG	
0	0	0	0	1 0	0 -	MISS
1020	15	3	3	1 0	3 -	MISS
2040	31	7	3	0 1	3 7	MISS
3060	47	11	3	1 0	11 7	MISS
4080	63	15	3	0 1	11 15	MISS
0	0	0	0	1 0	0 -	HIT
1020	15	3	3	1 0	3 15	MISS
2040	31	7	3	0 1	3 7	MISS
3060	47	11	3	1 0	11 7	MISS
4080	63	15	3	0 1	11 15	MISS

$N_{REF} = 5000 \cdot 5 = 25.000$

$N_{MISS} = 4 \cdot 4999 + 5 = 20.001$

$$m_4 = \frac{N_{MISS}}{N_{REF}} = 0,80004$$

15/12/2009

ES. 2

```
lw  $5, 100($2)
sub $6, $3, $2
and $7, $2, $1
add $8, $4, $3
or  $9, $2, $1
```

Pc = 100

$X = 20 + X

[Memory [x] = 2000 + x

-> 5 Stadi

		1	2	3	4	5
I/F	PC	100	104	108	112	116
F/D	ID		lw $5, 100($2)	sub $6, $3, $2	and $7, $2, $1	add $8, $4, $3
D/X	IX		lw	sub	and	
	A			22	23	22
	B/imm			100	22	21
	RD			5	6	7
	RS			2	3	2
	RT			-	2	1
X/M	R/W				R	-
	MAR/Res				122	1
	DIN				-	-
	RT/RD				5	6
M/W	Dout					2122
	Res					-
	RD					5

04/12/2007

ES. 2

a) L1

C = 256 THit = 2 ms

A = 2

B = 64

b)

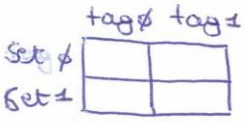
C = 2048 THit = 8 ms + 2 ms

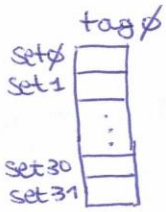
A = 1 Tmiss = 90 ms

B = 64

$N_{C_1} = \frac{C}{B} = 4$; $N_{S_1} = \frac{C}{B \cdot A} = 2$

$N_{C_2} = \frac{C}{B} = 32 = N_{S_2}$





L1)

T	X	X_H	X_T	X_S	X_S : MODIF	X_S : TAG	
R	22	0	0	0	0 : 1 0	0 : 0 - MISS	$X_H = \frac{X}{B}$
W	71	1	0	1	1 : 1 0	1 : 0 - MISS	
R	65	1	0	1	1 : 1 0	1 : 0 - HIT	$X_T = X_H / N_S$
W	90	1	0	1	1 : 1 0	1 : 0 - HIT	
R	143	2	1	0	0 : 0 1	0 : 0 1 MISS	$X_S = X_H \% N_S$
W	81	1	0	1	1 : 1 0	1 : 0 - HIT	
R	57	0	0	0	0 : 1 0	0 : 0 1 HIT	
W	133	2	1	0	0 : 0 1	0 : 0 1 HIT	
R	61	0	0	0	0 : 1 0	0 : 0 1 HIT	
W	190	2	1	0	0 : 0 1	0 : 0 1 HIT	
R	10	0	0	0	0 : 1 0	0 : 0 1 HIT	
W	12	0	0	0	0 : 1 0	0 : 0 1 HIT	
R	10	0	0	0	0 : 1 0	0 : 0 1 HIT	
W	15	0	0	0	0 : 1 0	0 : 0 1 HIT	
R	98	1	0	1	1 : 1 0	1 : 0 - HIT	
W	75	1	0	1	1 : 1 0	1 : 0 - HIT	
R	64	1	0	1	1 : 1 0	1 : 0 - HIT	
W	259	4	2	0	0 : 0 1	0 : 0 2 MISS	out : $X_H = 2$ $X_T = 1$ $X_S = 0$
R	130	2	1	0	0 : 1 0	0 : 1 2 MISS	out : $X_H = 0$ $X_T = 0$ $X_S = 0$
W	67	1	0	1	1 : 1 0	1 : 0 - HIT	
R	70	1	0	1	1 : 1 0	1 : 0 - HIT	
W	25	0	0	0	0 : 0 1	0 : 1 0 MISS	out : $X_H = 4$ $X_T = 2$ $X_S = 0$

$N_{miss} = 6$ $m_{rate} = \frac{N_{miss}}{N_{ref}} = 0,2727$

$N_{hit} = 16$

$N_{ref} = 22$ $AMAT = T_{hit} + m_{rate} \cdot T_{miss} = 4,1818$

T	X	X_H	X_T	X_S	X_S : MODIF	X_S : TAG	
R	22	0	0	0	0 : 1	0 : 0	MISS
W	71	1	0	1	1 : 1	1 : 0	MISS
R	143	2	0	2	2 : 1	2 : 0	MISS
W	259	4	0	4	4 : 1	4 : 0	MISS
R	130	2	0	2	2 : 1	2 : 0	HIT
W	25	0	0	0	0 : 1	0 : 0	HIT

$N_{miss} = 4$ $\quad m_{rate} = \frac{N_{miss}}{N_{req}} = 0,6667$

$N_{hit} = 2$

$N_{req} = 6$ $\quad AMAT = T_{hit} + m_{rate} \cdot T_{miss} = 70$

$$AMAT_{ger} = \underset{2}{T_{hit_1}} + (\underset{8}{T_{hit_2}} + \underset{90}{T_{pen_2}} \cdot \underset{0,6669}{m_{rate_2}}) \cdot \underset{0,2727}{m_{rate_1}} = 20,5456$$

c) $L_1 = L_2$: un solo livello di cache , $T_{hit} = 10\,ms$

T	X	X_H	X_T	X_S	X_S : MODIF	X_S : TAG	
R	22	0	0	0	0 : 1	0 : 0	MISS
W	71	1	0	1	1 : 1	1 : 0	MISS
R	65	1	0	1	1 : 1	1 : 0	HIT
W	80	1	0	1	1 : 1	1 : 0	HIT
R	143	2	0	2	2 : 1	2 : 0	MISS
W	81	1	0	1	1 : 1	1 : 0	HIT
R	57	0	0	0	0 : 1	0 : 0	HIT
W	133	2	0	2	2 : 1	2 : 0	HIT
R	61	0	0	0	0 : 1	0 : 0	HIT
W	180	2	0	2	2 : 1	2 : 0	HIT
R	10	0	0	0	0 : 1	0 : 0	HIT
W	12	0	0	0	0 : 1	0 : 0	HIT
R	10	0	0	0	0 : 1	0 : 0	HIT
W	15	0	0	0	0 : 1	0 : 0	HIT
R	88	1	0	1	1 : 1	1 : 0	HIT
W	75	1	0	1	1 : 1	1 : 0	HIT
R	64	1	0	1	1 : 1	1 : 0	HIT
W	259	4	0	4	4 : 1	4 : 0	MISS
R	130	2	0	2	2 : 1	2 : 0	HIT
W	67	1	0	1	1 : 1	1 : 0	HIT
R	70	1	0	1	1 : 1	1 : 0	HIT
W	25	0	0	0	0 : 1	0 : 0	HIT

02/11/2007

ES. 1

$0,045_{10}$ -> Singola Precisione

Singola Precisione 32 bit:
- segno: 1 bit
- exp: 8 bit
- mantissa: 23 bit

BIAS = 127

0,045	*2
0,09	
0,18	
0,36	
0,72	
1,44	

$1,44 \cdot 2^{-5} = 0,045$

exp = +127 - 5 = 122

122	2
61	0
30	1
15	0
7	1
3	1
1	1
0	1

Segno = 0

exp = 01111010

Parte intera: $1_{10} = 1_2$

Parte Decimale: 0,44

0,44	*2
0,88	0
1,76	1
1,52	1
1,04	1
0,08	0
0,16	0
0,32	0
0,64	0
1,28	1
0,56	0
1,12	1
0,24	0
0,48	0
0,96	0
1,92	1
1,84	1
1,68	1
1,36	1
0,72	0
1,44	1
0,88	0
1,76	1
1,52	1

$0,045_{10} = 0|01111010|01110000101000111101011$ 1

↓

24 esima cifra = 1

↓

$0,045_{10} = 0|01111010|01110000101000111101100$

$f_{clk} = 4\ GHz$ Sysenne : ALJ 27/10/2006

ES. 1

$$\begin{cases} AJ = 1 \\ B = 3 \\ L\text{-}S = 100 \end{cases}$$

B_i	N_i	ALJ	B	LS	cBi_{CPU}	$CBi_{CPU} = cBi_{CPU} \cdot N_i$	$cBi_{CPU,MEM}$	$CBi_{CPU,MEM} = cBi_{CPU,MEM} \cdot N_i$
BBM1	1	10	∅	3	310	310	300	300
BBM2	51	1	1	∅	4	204	∅	∅
BBM3	50	23	∅	80	823	41150	800	40000
BBM4	1	2	∅	∅	2	2	∅	∅
BBH1	50	8	∅	2	208	10400	200	10000
BBR1	50	3	∅	∅	3	150	∅	∅
BBR2	1325	∅	1	∅	3	3975	∅	∅
BBR3	1275	4	∅	1	104	132600	100	127500
BBR4	50	∅	1	∅	3	150	∅	∅
BBR5	50	1	∅	2	201	10050	200	10000
BBR6	∅	1	∅	1	101	∅	100	∅
BBR7	50	2	∅	∅	2	100	∅	∅

$$\sum_{i=2}^{51} = 1325$$

$$\sum_{i=1}^{50} i = 1275$$

$CBi_{CPU_{TOT}} = 199091$

$CBi_{CPU_{MEM_{TOT}}} = 187800$

$$T_{CPU} = \frac{C_{CPU}}{f_{CPU}} = 0{,}00004977\ sec = 49{,}77\ msec$$

$$T_{CPU,MEM} = \frac{C_{CPU,MEM}}{f_{CPU}} = 0{,}00004695\ sec = 46{,}95\ msec$$

30/11/2005

ES. 2

C = 192

A = 3

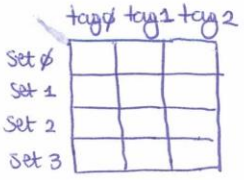
B = 16

$t_{Rut} = 5ms$

$t_{miss} = 60ms$

$N_c = \frac{C}{B} = 12$

$N_S = \frac{C}{B \cdot A} = 4$



T	X	X_H	X_T	X_S	X_S : MODIF	X_S : TAG	
R	22	1	0	1	1 : 2 0 0	1 : 0 - -	MISS
W	71	4	1	0	0 : 2 0 0	0 : 1 - -	MISS
R	65	4	1	0	0 : 2 0 0	0 : 1 - -	HIT
W	143	8	2	0	0 : 1 2 0	0 : 1 2 -	MISS
R	81	5	1	1	1 : 1 2 0	1 : 0 1 -	MISS
W	17	1	0	1	1 : 2 1 0	1 : 0 1 -	HIT
R	133	8	2	0	0 : 1 2 0	0 : 1 2 -	HIT
W	61	3	0	3	3 : 2 0 0	3 : 0 - -	MISS
R	190	11	2	3	3 : 1 2 0	3 : 0 2 -	MISS
W	211	13	3	1	1 : 1 0 2	1 : 0 1 3	MISS
R	212	13	3	1	1 : 1 0 2	1 : 0 1 3	HIT
W	210	13	3	1	1 : 1 0 2	1 : 0 1 3	HIT
R	115	7	1	3	3 : 0 1 2	3 : 0 2 1	MISS
W	98	6	1	2	2 : 2 0 0	2 : 1 - -	MISS
R	275	17	4	1	1 : 0 2 1	1 : 0 4 3	MISS
W	64	4	1	0	0 : 2 1 0	0 : 1 2 -	HIT
R	259	16	4	0	0 : 1 0 2	0 : 1 2 4	MISS
W	130	8	2	0	0 : 0 2 1	0 : 1 2 4	HIT
R	61	3	0	3	3 : 2 0 1	3 : 0 2 1	HIT
W	67	4	1	0	0 : 2 1 0	0 : 1 2 4	HIT
R	70	4	1	0	0 : 2 1 0	0 : 1 2 4	HIT
W	25	1	0	1	1 : 2 1 0	1 : 0 4 3	HIT

$X_H = 5$ aut $X_T = 1$ $X_S = 1$

$X_H = \frac{X}{B}$

$X_T = X_H / N_S$

$X_S = X_H \% N_S$

03/12/2003

ES. 1

```
addi  $12, $0, 2
addi  $10, $0, 288
loop: beq $10, $0, esci
      lw  $2, 100($10)
      add $2, $2, $12
      sw  $2, 200($10)
      j loop
addi  $10, $10, -4
esci: halt
```

R3000 con $f_{clk} = 2\,GHz$

- scrittura e lettura di un Reg. nello stesso ciclo;
- gestibile il delay slot;
- possibilità di decidere il salto in Decodifica;

? Speed-Up in circuiti:

- con forwarding;
- senza forwarding;

03/11/2009

ES. 1

$f_{CLK} = 1\ GHz$? Tempo Esecuzione BenchMark

$CPI_{unit\_log} = 1$

$CPI_{branch} = 3$

$CPI_{load-store} = 10$

$$T_{CPU} = \frac{C_{CPU}}{f_{CPU}}$$

$$Cbi_{CPU} = cBi_{CPU} \cdot N_i$$

BenchMark 1

BASIC BLOCK	N_i	ALU	B	LS	cBi_{CPU}	Cbi_{CPU}
BB1	1	2	0	0	2	2
BB2	11	1	1	0	4	44
BB3	10	5	0	0	5	50
BB4	1	2	0	0	2	2
						98

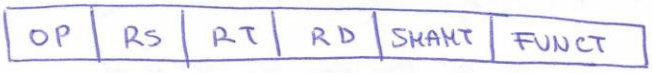
$C^1_{CPU} = 98$

$$T^1_{CPU} = \frac{98}{10^9} = 98\ m\,sec.$$

BenchMark 2

BASIC BLOCK	N_i	ALU	B	LS	cBi_{CPU}	Cbi_{CPU}
BB1		2	1	1	15	
BB2		1	1	0	4	
BB3		5	0	4	45	
BB4		2	0	1	12	

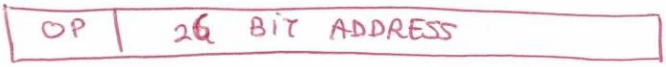
Differenza R, I, J

R 

→ 32 bits

I 

→ 32 bits

J 

→ 32 bits

OP : OPCODE (6 bits)
RS : SOURCE REGISTER (5 bits)
RT : SECOND SOURCE REGISTER (5 bits)
RD : DESTINATION REGISTER (5 bits)
SHAMT : SHIFT AMOUNT (5 bits)
FUNCT : CODICE FUNZIONE (6 bits)

R : istruzioni aritmetico-logiche
I: formato immediate -> istruzioni di trasferimento, immediate e di branch
J: formato Jump.

Sono necessari più formati di istruzioni in quanto istruzioni diverse hanno sintassi diverse:

```
add $s0, $s1, $s2
lw $s0, 0($sp)
J Salto
```

m = 3
m = 5
*a = 0x1000
*b = 0x2000
*e = 0x3000

Scrivere la Traccia

$sp = 0x7000 all'inizio di ogni funzione

Indicare se si tratta di R o W.

→ R 0x7000

{ R 0x1000
  R 0x2000

{ R 0x1004
  R 0x2004

{ R 0x1008
  R 0x2008

x 15 ...

{ R 0x1012
  R 0x2012

{ R 0x1016
  R 0x2016

W 0x3000

Nome	Numero	Utilizzo	Preservato durante le chiamate
$zero	0	costante zero	*Riservato MIPS*
$at	1	riservato per l'assemblatore	*Riservato Compiler*
$v0-$v1	2-3	valori di ritorno di una procedura	No
$a0-$a3	4-7	argomenti di una procedura	No
$t0-$t7	8-15	registri temporanei (non salvati)	No
$s0-$s7	16-23	registri salvati	Si
$t8-$t9	24-25	registri temporanei (non salvati)	No
$k0-$k1	26-27	gestione delle eccezioni	*Riservato OS*
$gp	28	puntatore alla global area (dati)	Si
$sp	29	stack pointer	Si
$s8	30	registro salvato (fp)	Si
$ra	31	indirizzo di ritorno	No

MIPS Floating-Point ABI (Application Binary Interface)

- Definisce la convenzione di uso dei registri nel software
- Valido nel caso single e in quello double precision
  - $f0, $f2 → valori di ritorno delle funzioni
  - $f12,$f14 → argomenti delle funzioni
  - $f20,$f22,$f24,$f26,$f28,$f30 → registri "saved"
  - $f4,$f6,$f8,$f10,f16,$f18 → registri temporanei
- I registri dispari sono tipicamente usati per le load/store e move

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	$a0 = integer	(none)
read character	12	(none)	char in $v0

Nome	Numero	Utilizzo	Preservato durante le chiamate
$zero	0	costante zero	*Riservato MIPS*
$at	1	riservato per l'assemblatore	*Riservato Compiler*
$v0-$v1	2-3	valori di ritorno di una procedura	No
$a0-$a3	4-7	argomenti di una procedura	No
$t0-$t7	8-15	registri temporanei (non salvati)	No
$s0-$s7	16-23	registri salvati	Si
$t8-$t9	24-25	registri temporanei (non salvati)	No
$k0-$k1	26-27	gestione delle eccezioni	*Riservato OS*
$gp	28	puntatore alla global area (dati)	Si
$sp	29	stack pointer	Si
$s8	30	registro salvato (fp)	Si
$ra	31	indirizzo di ritorno	No

MIPS Floating-Point ABI (Application Binary Interface)

- Definisce la convenzione di uso dei registri nel software
- Valido nel caso single e in quello double precision
  - $f0, $f2 → valori di ritorno delle funzioni
  - $f12,$f14 → argomenti delle funzioni
  - $f20,$f22,$f24,$f26,$f28,$f30 → registri "saved"
  - $f4,$f6,$f8,$f10,f16,$f18 → registri temporanei
- I registri dispari sono tipicamente usati per le load/store e move

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	$a0 = integer	(none)
read character	12	(none)	char in $v0