Riassunto Architettura degli Elaboratori

Alessandro Gerotto

Universitá degli studi di Udine

Insegnante *Federico Fontana*

Anno accademico 2021-22

1. Introduzione

Architettura di calcolo: macchina che esegue istruzioni macchina semplici.

Registri di memoria: Un registro è una piccola parte di memoria utilizzata per velocizzare l'esecuzione dei programmi fornendo un accesso rapido ai valori usati più frequentemente e/o i valori correntemente in uso in una determinata parte di un calcolo. Costituiscono il punto più alto della gerarchia della memoria, e sono il meccanismo più rapido per il sistema di manipolare i dati. I registri sono normalmente misurati in base al numero di bit che possono contenere (a.e. registri a 8 bit o registri a 32 bit).

Architettura ARM (Advanced RISC Machine): famiglia di cpu RISC dotata di 37 registri organizzati come segue:

- **16 registri** da 32 bit denominati $R0 \rightarrow R15$ dove:
    - $R0 \rightarrow R12$ sono registri di uso generale;
    - $R13$ usato come *stack pointer* (contiene l'indirizzo della locazione di memoria occupata dal top dello stack);
    - $R14$ usato come *link register* (contiene l'indirizzo a cui tornare al termine di una chiamata di funzione);
    - $R15$ usato come *program counter* (conservare l'indirizzo di memoria della prossima istruzione da eseguire);
- Un **registro di stato** $CPSR$ (Current Program Status Register)
    - 4 bit esprimono le condizioni (Negative, Carry, Zero e oVerflow);
    - Bit T;
    - I bit I and F;
    - I bit M4-M10;
- **20 registri duplicati**

Etichettatura di memoria: vengono usati n bit per indirizzare 2^n valori.

Legge di Moore: Il numero di transistor in un chip di memoria quadruplica ogni tre anni (dopo 18 mesi raddoppia per $2 \cdot 2 = 4$).

Memoria principale: tipo di memoria volatile a cui la CPU accede direttamente tramite un indirizzo e un bus per archiviare e recuperare informazioni.

Memoria di massa (memoria secondaria): tipo di memoria che raccoglie grandi quantità di dati rispetto alla memoria primaria e in maniera non volatile.

Capacitá di memoria [Byte]: $2^{linee\ indirizzo} \cdot linee\ dato$ dove $indirizzo$=locazione e $dato$ =info.

Banda passante: quantitá di dati trasmessi nell'unità di tempo [Bit/s].

Pipeline dati: tecnologia utilizzata nell'architettura hardware dei microprocessori dei computer per incrementare la quantità di istruzioni eseguite in una data quantità di tempo, *parallelizzando* i flussi di elaborazione di più istruzioni.

2. Porte algebriche

Segnali

Che possono essere di due tipi, positivi o negativi (associati a + e -):

- **Analogico**: segnale rappresentato da infiniti valori possibili, tipologia di segnale limitato dal rumore, che puó compromettere il valore lett;
- **Digitale**: segnale discreto che può assumere soltanto valori appartenenti ad un insieme discreto. Rappresentano quindi una approssimazione dei valori analogici, rimanendo quasi del tutto esenti da errori di rumore.

Circuiti logici

Un circuito logico è composto da porte e collegamenti interni che trasformano gli N segnali in input in M segnali in output. Possono essere di due tipi:

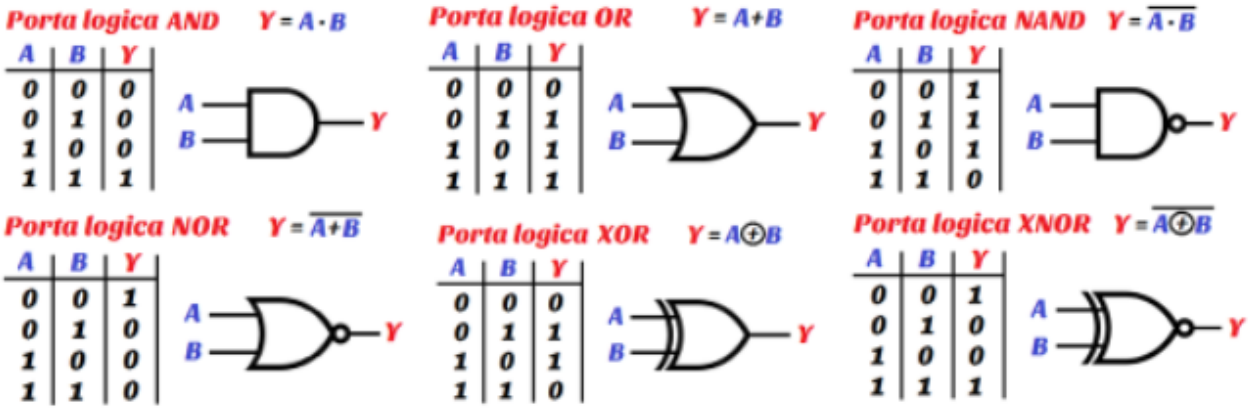
- **Combinatori:** circuiti logici in cui le uscite dipendono solo dagli ingressi;
- **Sequenziali**: circuiti logici in cui le uscite dipendono dagli ingressi e almeno una volta dallo stato precedente;

Porte logiche

Circuito digitale in grado di implementare un' operazione logica di una o più variabili booleane. Le porte elementari sono:

- AND x: y = 1 se tutti gli ingressi sono 1, altrimenti 0. # $ingressi + 1$ transistor
- OR +: y = 1 se almeno uno degli ingressi è 1, altrimenti 0. # $ingressi + 1$ transistor
- NOT: y = 1 se l'ingresso è 0, altrimenti 1. 1 transistor

Le sole porte OR, AND e NOT o le sole porte NAND ($B = \overline{A * B}$, due transistor) e NOR ($B = \overline{A + B}$, 2 transistor) costituiscono un **insieme funzionalmente completo**: attraverso gli operatori logici che implementano è possibile generare qualsiasi funzione logica.



Algebra booleana

Espressioni algebriche che descrivono reti di porte logiche e permettono di capire un circuito e migliorarlo cercando di limitare il numero delle porte in uso per limitare i costi. Per semplificare un circuito uso le proprietà dell'algebra booleana e il loro duale:

Elemento nullo	$0 * A = 0$	$0 + A = A$
Identità	$1 * A = A$	$1 + A = 1$
Idempotenza	$A + A = A$	$A * A = A$
Inverso	$A + \overline{A} = 1$	$A * \overline{A} = 0$
Commutatività	$A + B = B + A$	$A * B = B * A$
Associatività	$A + (B + C) = (A + B) + C$	$A * (B * C) = (A * B) * C$
Distibutività	$A(B + C) = AB + AC$	$A + (B * C) = (A + B) * (A + C)$
Assorbimento	$A + (A * B) = A$	$A * (A + B) = A$
De Morgan	$\overline{A + B} = \overline{A} * \overline{B}$	$\overline{A * B} = \overline{A} + \overline{B}$

Si può ottenere la negata di un'espressione negando le variabili, scambiando il + con * e scambiando gli 0 con 1. Nota: i circuiti con retroazione (dove l'uscita determina uno o più ingressi) non sono descrivibili con boole.

Mappe di karnaugh

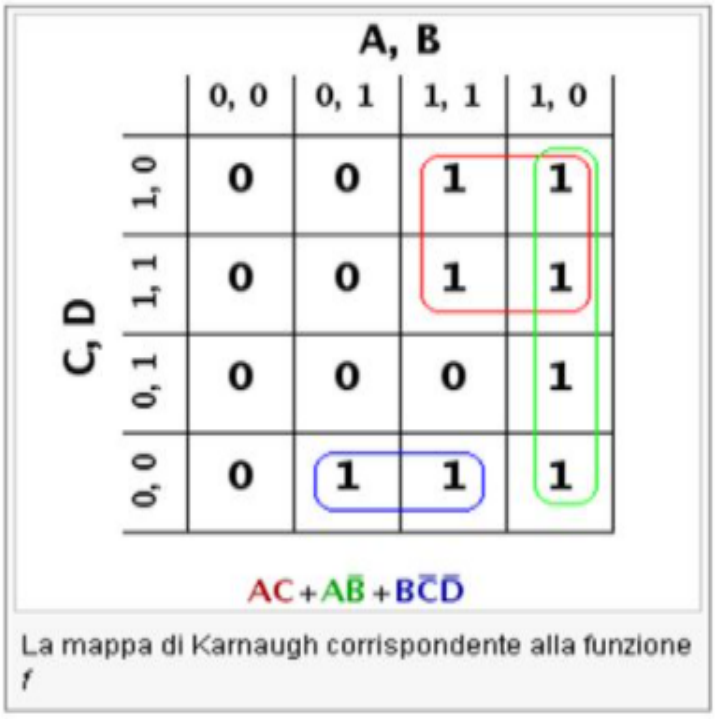
La mappa di Karnaugh è un metodo di **rappresentazione di sintesi di reti combinatorie**. Una tale mappa costituisce una rappresentazione visiva di una funzione booleana in grado di mettere in evidenza le coppie di mintermini o di maxtermini a distanza di Hamming unitaria (ovvero di termini che differiscono per una sola variabile binaria (o booleana)).

Una volta stesa la tabella, è necessario **raggruppare gli 1** in rettangoli multipli di 2^n elementi e poi fare un or di and in cui gli and sono i rettangoli. In maniera **duale** si può raggruppare gli 0, creando un and di or.

È possibile raggruppare anche i valori ai lati opposti dato che la matrice ha un comportamento toroidale (lati opposti sono adiacenti)

Si possono trovare anche dei valori x all'interno della matrice. Essi rappresentano un valore indeterminato che può rappresentare sia un 1 che uno 0.

	A	B	C	D	f
0.	0	0	0	0	0
1.	0	0	0	1	0
2.	0	0	1	0	0
3.	0	0	1	1	0
4.	0	1	0	0	1
5.	0	1	0	1	0
6.	0	1	1	0	0
7.	0	1	1	1	0
8.	1	0	0	0	1
9.	1	0	0	1	1
10.	1	0	1	0	1
11.	1	0	1	1	1
12.	1	1	0	0	1
13.	1	1	0	1	0
14.	1	1	1	0	1
15.	1	1	1	1	1



La mappa di Karnaugh corrispondente alla funzione *f*

Esercizio

Senza minimizzare, si dia l'espressione booleana in forma normale e dualizzata:

ABC| E
0 0 0 | 1
0 0 1 | 1
0 1 0 | 0
0 1 1 | 1
1 0 0 | 0
1 0 1 | 1
1 1 0 | 0
1 1 1 | 1

Forma normale: $E = \overline{ABC} + \overline{A\overline{B}C} + \overline{A}BC + A\overline{B}C + ABC$

Forma dualizzata: $E = (A + \overline{B} + C)(\overline{A} + B + C)(\overline{A} + \overline{B} + C)$

3. Reti combinatorie

Circuiti combinatori

Un circuito combinatorio è un circuito il cui funzionamento riguarda solo la relazione **ingresso-uscita**. Tale relazione è descritta da una funzione logica.

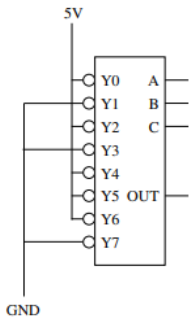
Si dividono tra **circuiti logici di base**:

Multiplexer

É formato da n ingressi di controllo, 2^n ingressi di segnale e da un'uscita di segnale. Gli ingressi selezionano un ingresso di segnale e lo instradano all'uscita;

Esercizio

Realizzare la rete booleana di $E = \overline{C} + A\overline{B}$ usando un multiplexer di cui non occorre specificare il circuito



ABC | E

0 0 0 | 1

0 0 1 | 0

0 1 0 | 1

0 1 1 | 0

1 0 0 | 1

1 0 1 | 1

1 1 0 | 1

1 1 1 | 0

È sufficiente scegliere un multiplexer a 8 ingressi e 3 controlli collegati rispettivamente ad ABC, e poi connettere a una sorgente in tensione gli ingressi associati ai valori di controllo 000, 010, 100, 101, 110. Viceversa, gli altri 3 ingressi dovranno essere collegati a una tensione nulla .

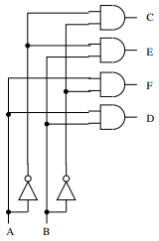
Decoder

è formato da n ingressi di controllo, 0 ingressi di segnale e da 2^n uscite di segnale. Gli ingressi di controllo selezionano una sola uscita che assume un valore alto;

Esercizio

Realizzare la tabella adoperando un decodificatore tenendo conto che AB sono gli ingressi mentre CDEF le uscite.

C	D	E	F	A	B
1	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	0	0	1	1	0



Demultiplexer

instrada un ingresso di segnale per selezionare il dispositivo desiderato;

Comparatore ad n bit

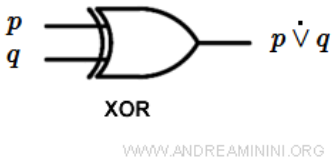
2 ingressi a n bit. L'uscita vale 1 se i due segnali in ingresso sono uguali, infatti viene usato per confrontare i bit.

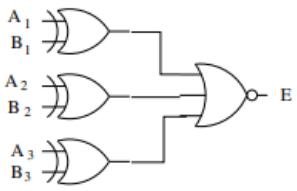
Esercizio

Progettare un comparatore di ingressi a 3 bit, cioè una rete combinatoria in grado di decidere se due ingressi $A_1A_2A_3$ e $B_1B_2B_3$ sono identici.

É necessario usare tre porte xor a due ingressi per rispettivamente A_iB_i e convogliare le uscite in una porta OR a tre ingressi e negando l'uscita.

p	q	$p \dot{\vee} q$
0	0	0
0	1	1
1	0	1
1	1	0





circuiti aritmetici

- **Mezzo sommatore**: esegue la somma su una cifra binaria, restituendo risultato e resto;
- **Sommatore completo**: mezzo sommatore + riporto precedente = 2 half adder;
- **Shifter**: traslazione dei bit a destra (se $c = 1$) o a sinistra (se $c = 0$).

Circuiti combinatori (o retroazionati)

Circuito i cui valori di uscita non dipendono soltanto dai valori di ingresso correnti, ma anche da quelli **precedenti**.

Essi sono i **latch** (SR, SR sincronizzato, D sincronizzato), il **clock** (e clock ad impulsi che sfrutta i ritardi delle porte logiche per generare un impulso brevissimo), **flip flop** ed i **registri** (circuiti combinatori realizzati con n flip flop sincronizzati mediante uno stesso clock in grado di memorizzare sequenze di n bit (tipicamente 8, 16, 32, 64)).

Macchine a stati finiti

Un automa a stati finiti è un modello matematico di calcolo: è un tipo di automa che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi istante per istante. Sono rappresentati da nodi collegati tramite archi orientati e sono di due tipi:

Macchina di Mealy:

Sugli archi ci sono input e output. I valori di uscita sono determinati dallo stato attuale e dall'ingresso corrente; Esempio: Macchina di Mealy che riconosce le sequenze *101* e *110*.

Macchina di Moore:

Le uscite sono determinate in funzione dei **soli stati correnti** (e non anche dagli stati d'ingresso, come accade invece nella macchina di Mealy). Il diagramma di stato di una macchina di Moore prevede un segnale d'uscita per ciascuno stato. Esempio: Macchina di Moore che riconosce le sequenze *101* e *110*.

4. Codifica e basi di dati

I dati sono codificati in sequenze di bit tramite mappatura in un **alfabeto** (cioè un insieme di simboli). L'alfabeto binario a n bit può rappresentare fino a 2^n dati differenti. Si possono rappresentare i numeri con **diverse basi** (2, 4, 8, 10, 16, 20, …). Per rappresentare n_{10} in base b servono $log_b n$ bit arrotondati per eccesso.

Conversioni di base

Da base B a base 10

- **Parte intera:** partendo da $pos = 0$, ad ogni passo moltiplico per $val_{pos} \cdot B^{pos}$ e spostandosi a sinistra sommo. A.e.: $10011_2 \rightarrow ?_{10}$:

  $1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 0 * 2^3 + 1 * 2^4 = 1 + 2 + 0 + 0 + 16 = 19_{10}$

- **Parte decimale**: segue la stessa logica della parte intera, ma le posizioni sono negative: Ae.: $0.1011_2 \rightarrow ?_{10}$:

  $1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4} = 1/2 + 0 + 1/8 + 1/16 = 0,6875_{10}$

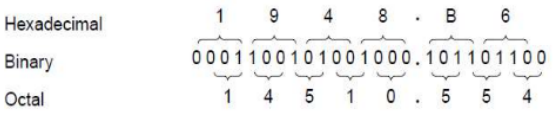
Da base 10 a base B

- **Parte intera:** divido il numero intero per *B*, il quoziente rappresenterà il nuovo numero da dividere mentre il resto invece sarà una cifra del numero convertito. Si prosegue dividendo e annotando i resti fino a quando il quoziente non è 0. Il numero in base *B* sarà quindi rappresentato dai resti delle operazioni, leggendo dall'ultimo calcolato al primo.
- **Parte decimale**: moltiplico il numero decimale per *B* e tengo solo la parte intera del risultato. Ripeto finchè $2^n = 1$.

Tra basi non decimali

- Facilmente calcolabile passando dalla base 10 usando le tecniche viste sopra (a.e. Passare da $B_3 \rightarrow B_8$ è facilmente attuabile facendo $B_3 \rightarrow B_{10} \rightarrow B_8$). Tuttavia per

le conversioni di un numero di base B_{2^k} mappa ogni cifra in base B in k cifre binarie in una cifra nella nuova base.



L'aritmetica del calcolatore

I numeri naturali (unsigned) e i numeri interi (signed) sono rappresentabili con 16, 32 o 64 bit (fixed point) mentre i numeri reali (float) con 32, 64 o 128 bit (floating point). Dunque esisteranno dei massimi e dei minimi poiché l'insieme dei valori rappresentabili è limitato a $2^{16,\,32,\,64\ o\ 128}$ codifiche $[-2^{n-1}+1,\ 2^{n-1}-1]]$ ($[-2^{n-1},\ 2^{n-1}-1]$ per il complemento a due (a.e. Con 4 bit: [-8, +7] perchè il primo bit è il segno).

Codifiche di interi (a.e. ad 8 bit)

1. **Segno e valore assoluto**: per il n negativo si nega il bit più significativo

   $9_{10} = 0\ 0001001 \rightarrow -9_{10} = 1\ 0001001$

2. **Complemento a uno**: per il n negativo si nega ogni bit. Codifica molto usata per l'immediato riconoscimento del segno in quanto il numero è < 0 se e solo se il bit più a sinistra è 1, mentre per la somma algebrica basta negare il secondo termine dopo averlo cambiato di segno

   $9_{10} = 0\ 0001001 \rightarrow -9_{10} = 1\ 1110110$

3. **Complemento a due**: per il numero negativo si nega ogni bit e si fa +1. Codice per ottenere un numero in complemento a due a n bit:

   - valori tra 0 e $2^{n-1}-1$ sono normalmente codificati in notazione binaria;
   - valori tra -1 e -2^{n-1} sono codificati negando ogni bit e poi sommando 1.

   $9_{10} = 0\ 0001001 \rightarrow -9_{10} = 1\ 1110111$

4. **Eccesso N**: cpl a due per $n < 0$, poi 1 ($n > 0$) e 0 ($n < 0$) vanno su $N = 2^{N-1}$

$$9_{10} = 0\,0001001 \rightarrow -9_{10} = 0\,1110111$$

Overflow

Il risultato di un'operazione non è rappresentabile nella codifica scelta. Nella codifica in complemento a due ci potrà essere **overflow solo se sommo (**mai overflow per divisioni e sottrazioni?) **numeri aventi lo stesso segno**.

Notazione mantissa ed esponente

Per rappresentare valori a virgola mobile in modo binario, si usa un sistema descritto dalla notazione $X = segno \cdot mantissa \cdot 10^{esponente}$ dove il **segno** può essere 1 o 0, la **mantissa** rappresenta la parte intera del numero e l'**esponente** è il numero di volte in cui la base deve essere moltiplicata per se stessa. (a.e.$1.\,0110000011 \cdot 2^{7}$);

Codifica IEEE 754 a 32 bit

- Il bit più a sinistra è riservato al segno (0 se positivo, 1 se negativo);
- I successivi 8 bit sono riservati all'esponente
- Gli ultimi 23 bit per la mantissa

Ae.: per ottenere la rappresentazione floating point del numero decimale -109,78125 bisogna **convertire in binario** la parte intera e la parte decimale ottenendo: 1101101,11001 che in notazione normalizzata corrisponde a: 1,10110111001x26. All'esponente (6) va aggiunto il valore del bias 127 e quindi l'esponente è costituito dal valore 33: 10000101. Poiché il numero è negativo il **bit del segno** va impostato a 1. Si ottengono quindi i valori:

- 1 per il segno
- 10000101 per l'esponente
- 10110111001 completato con zeri per la mantissa (l'1 della parte intera è sottinteso)

e quindi la rappresentazione finale è 1 | 10000101 | 10110111001000000000000 che in esadecimale è 0xc2db9000.

Operazioni floating point

- **Somme e sottrazioni**: sommare o sottrarre le mantisse dopo aver eguagliato gli esponenti
- **Moltiplicazioni**: moltiplico le mantisse e sommo gli esponenti
- **Divisioni**: dividere (moltiplicare il reciproco) le mantisse e sottrarre gli esponenti

Capacità di memoria come potenze di due

Simbolo	in Bit	in Byte	in potenze di 2
1 b (bit)	1	1/8	
1 B (byte)	8	1	
1 KB (kilobyte)	8.192	1.024	2^{10} byte
1 MB (megabyte)	8.388.608	1.048.576	2^{20} byte
1 GB (gigabyte)	8.589.934.592	1.073.741.824	2^{30} byte
1 TB (terabyte)	8.796.093.302.400	1.099.511.628.000	2^{40} byte

Esercizi

Esegui la sottrazione in complemento a 2 a 8 bit di 7 - 83:

$7 - 83 =$

$7 + (-83) =$

$00000111_2 - 01010011_2 =$

$00000111_2 + 10101101_{cpl\,2} = 10110100_{cpl\,2} =$

$-\ 01001100_2 = -\ 76_{10}$

Esegui la sottrazione binaria di 00000100-1:

$00000100 - 1 = 00000011$

5. Caratteri

Codice di parità

Il bit di parità è un codice di controllo utilizzato nei calcolatori per **prevenire errori** nella trasmissione o nella memorizzazione dei dati. Tale sistema prevede l'aggiunta di un bit ridondante ai dati, calcolato a seconda che il numero di bit che valgono 1 sia pari o dispari. Ci sono due varianti del bit di parità:

- Bit di **parità pari**, si pone tale bit uguale a 1 se il numero di "1" in un certo insieme di bit è dispari (facendo diventare il numero totale di "1", incluso il bit di parità, pari);
- Bit di **parità dispari**, si pone tale bit uguale a 1 se il numero di "1" in un certo insieme di bit è pari (facendo diventare il numero totale di "1", incluso il bit di parità, dispari).

7 bit di dati	Byte con bit di parità	
	Bit di parità pari	Bit di parità dispari
1101001	11010010	11010011
1111111	11111111	11111110

Distanza di Hamming

La distanza di Hamming è il **numero di bit diversi** tra due parole (a.e.: 1011001001 e 1001000011 Hamming distance = 3). Se due parole hanno distanza d significa che servono d errori per trasformare una nell'altra, quindi la distanza di Hamming indica **quanti errori si possono rilevare** e quanti **se ne possono correggere**. È possibile calcolare il numero di bit diversi facendo l'*or esclusivo* delle due stringhe e contando il numero di bit 1 del risultato;

Codice di Hamming

Il codice di Hamming è un codice che permette di aggiungere un certo numero di bit ai bit di dati in modo da comporre parole con distanza 3, in grado di rilevare e correggere errori su un singolo bit. Il numero di bit da aggiungere aumenta all'aumentare del numero dei bit di dati. Il codice di Hamming consiste nell'aggiungere dei bit di parità in posizioni che sono potenze di due in modo da poter distinguere i bit di controllo da quelli dati.

Esempio codice di Hamming

Partendo dalla sequenza: $0_1 1_2 1_3 0_4$ si calcolano e inseriscono i bit di parità nelle posizioni 1, 2 e 4: $?_1 ?_2 0_3 ?_4 1_5 1_6 0_7$

- Il bit 3 influenza i bit di parità 1 e 2 (3=1+2).
- Il bit 5 influenza i bit di parità 1 e 4 (5=1+4).
- Il bit 6 influenza i bit di parità 2 e 4 (6=2+4).
- Il bit 7 influenza i bit di parità 1, 2 e 4 (7=1+2+4).

Il bit di parità 1 è calcolato sui bit di dati 3, 5 e 7 (0 1 0) e quindi vale 1.

Il bit di parità 2 è calcolato sui bit di dati 3, 6 e 7 (0 1 0) e quindi vale 1.

Il bit di parità 4 è calcolato sui bit di dati 5, 6 e 7 (1 1 0) e quindi vale 0.

Perciò si ottiene la sequenza: $1_1 1_2 0_3 0_4 1_5 1_6 0_7$.

Se un errore modifica la sequenza in: $1_1 1_2 0_3 0_4 1_5 0_6 0_7$ ricalcolando i bit di parità si ottiene:

bit di parità 1 calcolato sui bit di dati 3, 5 e 7 (0 1 0): 1

bit di parità 2 calcolato sui bit di dati 3, 6 e 7 (0 0 0): 0

bit di parità 4 calcolato sui bit di dati 5, 6 e 7 (1 0 0): 1

Confrontando i bit della sequenza originale con quelli calcolati viene incrementato il contatore k:

il bit di parità 1 è uguale: k=0;

il bit di parità 2 è diverso: k=2;

il bit di parità 4 è diverso: k=6.

Pertanto si può stabilire che c'è un bit errato nella posizione 6 e lo si può correggere ottenendo la sequenza $1_1 1_2 0_3 0_4 1_5 1_6 0_7$.

6. Processore (livello 1)

Composizione cpu

Qualunque CPU contiene:

- Una **ALU** (Unità Aritmetico-Logica) che si occupa di eseguire le operazioni logiche e aritmetiche;
- Una **Unità di Controllo** (Control Unit, CU) che esegue operazioni (sincronizzazione con la RAM) finalizzate al trasferimento dati o al controllo dell'esecuzione dei programmi;
- Dei **registri**: dispositivi di memorizzazione usati per le elaborazioni interne alla CPU, cioè sequenze di N celle di memoria nelle quali si può leggere e scrivere un dato di N bit;
- **Bus interni** che collegano registri, ALU, unità di controllo e memoria primaria;
- Alcuni **segnali elettrici** come il RESET, le linee di IRQ e il CLOCK che si occupano di tenere la CPU al corrente dello stato del resto del sistema e di agire su di esso.

Fetch-decode-execute

In termini generali, un processore esegue **iterativamente** tre operazioni:

- preleva (**fetch**) un'istruzione dalla memoria primaria;
- decodifica (**decode**) il tipo di istruzione ed i relativi argomenti;
- esecuzione (**execute**) dell'istruzione e salvataggio dei risultati.

In questo modo il processore esegue sequenzialmente istruzioni che danno vita a **thread e processi**, sotto la supervisione del sistema operativo attraverso lo **scheduler**.

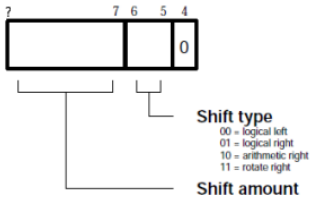
Logica

Per realizzare un set di micro-operazioni si può usare la **logica cablata**: si realizza l'hardware di un circuito sequenziale (veloce ma costosa) oppure la **logica programmata**: l'unità di controllo è a sua volta una micro-architettura capace di eseguire un microprogramma (semplice e flessibile ma lenta).

7. ISA (livello 2)

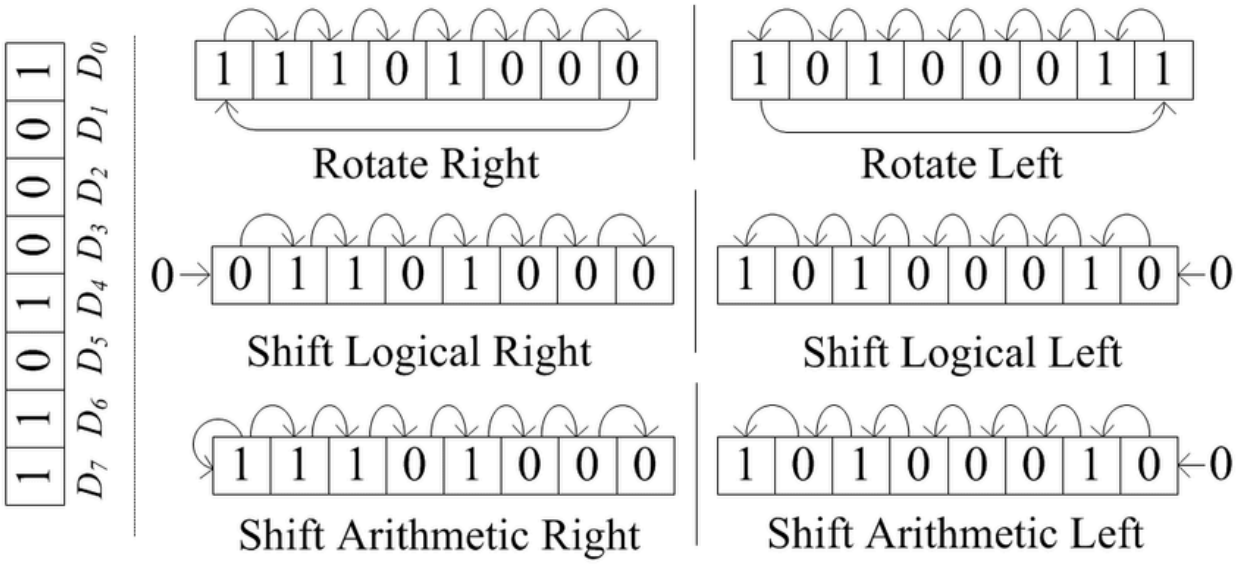
L'instruction set è l'insieme di **istruzioni macchina** che descrive quegli aspetti, visibili a basso livello al programmatore, dell'architettura di un calcolatore. Si tratta dell'**insieme di istruzioni base** che il processore può compiere e che costituiscono il suo **linguaggio macchina**, a partire dal quale vengono scritti i relativi programmi nei vari linguaggi di programmazione a più alto livello di astrazione.

Esercizio



I campi in figura appartengono a un'istruzione macchina per ARM che prevede la possibilità di traslare o ruotare il contenuto di un registro. Quale valore posizionale va sostituito al '?' in alto a sinistra nella figura? Quali sono le traslazioni o rotazioni minime e massime ammesse in tal caso sul registro?

R: Se parliamo di un'architettura a 32 bit, allora le traslazioni o rotazioni ammesse sul registro variano da un minimo di 0 bit a un massimo di 31 bit. Essedo 32 bit rappresentabili con 2^5 bit, occorreranno 5 bit per specificare le entità di queste operazioni. Da ciò si deduce che il campo più a sinistra in figura occuperà dal 7 all'11.



Interfaccia tra livello 3 e livello 1

L'insieme delle istruzioni macchina ha il compito di definire l'**interfaccia** tra software (livello 3) e hardware (livello 1). Per interfacciare il livello 1, l'isa deve descrivere:

- Le **funzioni della cpu**;
- Le **categorie di istruzioni eseguibili** (mov. dati, operazioni, salti, call utente o di OS);
- I **tipi comprensibili** di dati numerici (float, int) e non (bool, char, string, puntatori);
- Il **modello di RAM**: logicamente unica o suddivisa in area istruzioni e area dati;
- Il **modello dei registri**: generici (logicamente indistinti) o specializzati (PC, IR, PSW);
- Il **formato delle istruzioni**: *comando (tipo operazione) arg1, arg2 (dati);*
- Le **modalità di indirizzamento** per specificare dove si trova il dato
    - Immediato (#5): valore numerico;
    - Diretto (0x1000): indirizzo di una locaz. di mem contenente un valore numerico;
    - Di registro (dato in r5);
    - Indiretto (dato nella memoria indirizzata da r5): indirizzo di una locazione di memoria, tipicamente il nome di un registro, in cui leggere l'indirizzo di una locazione di memoria contenente un valore numerico.;
    - Base indicizzata [$r1, $r2];

Come interfaccia con il livello 3 specifica:

- La **modalità di funzionamento**: *utente* o *supervisore* (è possibile passare da una mod all'altra o con chiamate di sistema o mediante eventi della cpu);
- La **gestione dell'I/O**;
- La comunicazione tra cpu e controllori prevede la **lettura e la scrittura di dati** in un registro mentre controllo e info di stato in un altro e per accedervi bisogna segnalarlo. Tutto ciò porta quindi ad una **sincronizzazione** delle parti. Altri meccanismi di sincronizzazione sono:

- **I/O programmato** (busy waiting o polling): la cpu legge periodicamente le info di stato (semplice ma inefficente);
- **I/O controllato** (interrupt): la cpu è eccezionalmente interrotta dalle periferiche a cui ha richiesto un servizio, quando accetta passa in modalità protetta ed esegue una routine che abilita la periferica. Finita la routine torna come prima (migliori prestazioni ma la cpu deve sempre gestire ogni operazione);
- **I/O ad accesso diretto** (direct memory access): comunicazione con i controllori delegata al DMA (meno interrupt ma necessita di un chip DMA, può generare conflitti nell'accesso con la ram)

Esempio:

Polling di una stampante	**Interrupt da una stampante**
Stampante con segnalazione dello stato (bit READY) La cpu riceve un comando di stampa dall'utente e inizia un ciclo di stampa: 1- Cpu trova la stampante pronta (READY=1) 2- Cpu scrive il carattere nel registro e imposta READY=0 3- Il controllore legge il dato nel registro 4- La stampante stampa il carattere 5- Il controllore segnala che la stampante è pronta (READY=1)	Stampante con segnale di interrupt (bit PRINT_REQ) e registro dati (carattere da stampare). La cpu riceve un comando di stampa dall'utente e inizia il ciclo di stampa: 1- la stampante richiede di stampare (PRINT_REQ=1) 2- la cpu interrompe la sua attività e esegue una routine di servizio (scrittura del carattere sul registro) 3- la cpu rientra dalla routine (PRINT_REQ=0) 4- la stampante stampa il carattere (PRINT_REQ=0)

Interrupt

È un **segnale asincrono** che indica il "**bisogno di attenzione**" da parte di una periferica I/O finalizzata ad una particolare richiesta di servizio.

La cpu, nel momento in cui avviene un interrupt:

- Salva il suo stato di esecuzione in un **context switch** (conserva lo stato del processo o thread, in modo da poter essere ripreso in un altro momento);
- Identifica il dispositivo ed accede al **vettore degli interrupt** che contiene gli indirizzi di tutte le routine di servizio e il livello di protezione.
- Esegue la **chiamata alla routine** al livello di protezione previsto (interrompibile solo tramite interrupt di priorità maggiore).
- Appena la cpu termina la chiamata della routine, **ritorna** allo stato salvato prima e prosegue.

Tipologie di interrupt

- **Interrupt hardware**: generati da **dispositivi esterni alla CPU** (periferiche), che hanno il compito di comunicare il verificarsi di eventi esterni, di solito dispositivi di Input/Output. Un interrupt hardware costringe il processore a memorizzare il suo stato di esecuzione fino all'arrivo dell'interrupt e ad iniziare l'esecuzione della subroutine che esegue il compito richiesto dall'interrupt, terminato il quale il processore riprende l'esecuzione delle operazioni che stava precedentemente elaborando.
- **Interrupt software**: sono delle **istruzioni assembly**, tipo *INT xx* o *SYSCALL*, che possono essere assimilate alle chiamate di sottoprogrammi, ma che sfruttano il meccanismo delle interruzioni per passare il controllo dal programma chiamante a quello chiamato, e viceversa;

Mascheramento interrupt

Quando alla cpu arriva un interrupt di priorità minore rispetto a quello che sta eseguendo e lo lascia in "Stand by", ossia lo lascia da parte mascherandolo fino alla fine del processo che sta eseguendo. Alla fine di esso, ripesca l'interrupt che aveva mascherato e lo esegue normalmente.

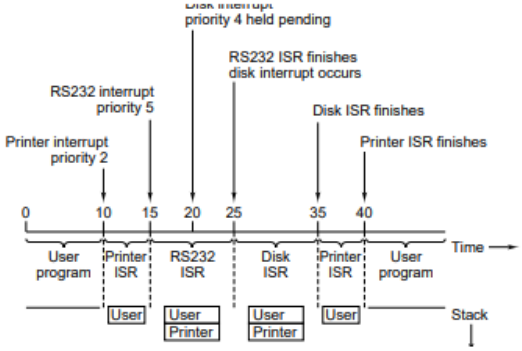
Il registro di mascheramento delle interruzioni è un **registro di stato**, che puó essere:

- **1**: quando deve bloccare i successivi interrupt in quanto hanno una priorità inferiore rispetto all'interrupt che al momento è in esecuzione;

- **0**: quando riceve richiesta di interrupt con priorità più alta dell'interrupt che è in esecuzione, così salva il contesto computazionale effettuando un context switching.

Esercizio

Si dia schematicamente un esempio di mascheramento dell'interrupt durante il servizio a dispositivi esterni da parte della CPU.



- 0-10: la CPU svolge il programma dell'utente;
- 10: arriva l'interrupt della stampante di priorità 2 e la CPU mette nello stack il programma utente perché ha una priorità più <2;
- 10-15: la CPU svolge il programma dedicato all'interrupt della stampante;
- 15: arriva l'interrupt della RS232 con priorità 5 e quindi svolge quella visto che ha priorità maggiore rispetto a quella precedente e mette nello stack il programma della stampante che dovrà finire in seguito;
- 15-20: esegue il programma per l'interrupt della RS232
- 20: arriva l'interrupt del disco che però ha priorità minore al programma che sta svolgendo e quindi lo maschera lasciandolo in sospeso;
- 20-25: finisce il programma per la RS232;
- 25-35: svolge il programma per l'interrupt mascherato;
- 35-40: finire il programma che è stato aggiunto per ultimo allo stack, ossia quello della stampante;
- 40: torna ad eseguire il programma utente.

8. Bus

Il bus è un **canale fisico di comunicazione** che permette a periferiche e componenti di un sistema elettronico di interfacciarsi tra loro scambiandosi informazioni o dati di vario tipo attraverso la trasmissione e la ricezione di segnali.

Caratteristiche fisiche

La prossimità alla cpu determina la banda passante, il costo e la flessibilità del bus:

- **Bus locale:** proprietario, banda elevata;
- **Bus della cache:** proprietario:
- **Bus della RAM;**
- **Bus di sistema per i principali dispositivi** (PCIe);
- **Bus per dispositivi esterni** (USB, Thunderbolt);

Per collegamenti lunghi i cavi spesso si trovano intrecciati: questo serve per eliminare o limitare la presenza di campi magnetici che possono compromettere l'integrità del segnale.

Condivisione di un bus

In un bus condiviso solo un dispositivo può trasmettere, tutti gli altri possono solo ricevere. I dispositivi che si trovano lontani dal trasmettitore devono amplificare il segnale.

Frequenza e banda passante

La **frequenza** equivale al numero di slot nel segnale in 1 secondo e si misura in Hz. La **banda passante** è invece il valore che coincide con il prodotto tra $freq \cdot n\ di\ bit\ in\ uno\ slot$ e si misura in bit/s.

Bus sincroni e asincroni

La realizzazione dei protocolli logici può avvenire mediante due schemi:

- I **bus sincroni** (clock condiviso): il clock occupa una **linea** del bus. Vantaggio di questo tipo di bus è la **semplicità** del protocollo di comunicazione e la **massima velocità** ottenibile a discapito però dell'**aggiunta di una linea fisica**. Questo sistema è inadatto a

comunicazioni con tempi di risposta non predeterminati e se il sincronismo viene meno tutto il sistema non funziona;

- I **bus asincroni** (handshaking): **negoziazione iniziale** per la sincronizzazione. La condivisione risulta più **semplice per i dispositivi con tempi di risposta diversi** e la trasmissione è più **economica per dispositivi fisicamente più distanti**. Tuttavia lo svantaggio di questa comunicazione è che il circuito è più **complesso.**

Bus seriali e paralleli

Esistono due tipi di bus:

- **Bus seriali**: i bit sono trasferiti lungo un canale di comunicazione **uno di seguito all'altro** e giungono sequenzialmente al ricevente nello stesso ordine in cui li ha trasmessi il mittente
- **Bus paralleli**: adotta una trasmissione parallela ovvero una trasmissione di dati in cui tutti i **bit sono trasferiti contemporaneamente** lungo canali separati di un mezzo di comunicazione. Sono di questo tipo i bus ISA, PCI e AGP.

  Prestazioni più elevate in termini di velocità di trasmissione a parità di frequenza, ma più ingombrante e costosa.

Principio di funzionamento

In ogni transazione sul bus:

- un dispositivo **prende il controllo** del bus;
- **invia una richiesta (I/O)** a un secondo dispositivo;
- **svolta la richiesta**, il bus viene liberato per un'altra comunicazione.

Il ruolo di un dispositivo può cambiare nel tempo; un dispositivo può comportarsi da master o da slave in contesti differenti.

Modalità di connessione

- **A stella**, in cui ogni nodo della rete è collegato agli altri passando per uno o più concentratori, detti **hub,** ed esiste un solo percorso che colleghi un nodo a un altro. Ogni nodo ha un solo ramo, collegato a un hub, mentre gli hub hanno almeno due rami di connessione verso altri nodi e altri hub.
- **Daisy-chain**, in cui i nodi sono collegati uno di seguito all'altro, e quindi ogni nodo ha due rami (connessioni) con l'eccezione dei 2 nodi posti alle estremità.
- **Ring**, simile alla rete daisy-chain in cui i punti estremi sono anch'essi connessi fra loro, creando quindi un anello.

Bus di sistema

È l'insieme di collegamenti il cui scopo è **coordinare le attività del sistema**; tramite esso, la CPU può decidere quale componente deve scrivere sul bus dati in un determinato momento, quale indirizzo leggere sul bus indirizzi, quali celle di memoria devono essere scritte e quali invece lette, etc. Può essere di tre tipi a seconda di come si presenta fisicamente:

- **Bus interno**: collega due componenti appartenenti alla stessa scheda integrata;
- **Bus esterno**: collega due componenti generici è detto bus esterno;
- **Bus di sistema**: unico bus esterno

Il bus di sistema si divide in tre bus minori:

- **Bus dati**: bus sul quale **transitano le informazioni**. È usufruibile da tutti i componenti del sistema, sia in scrittura sia in lettura. È bidirezionale. Alcuni bus dati importanti sono: ISA, Zorro, PCI, USB, SCSI, PCI Express, etc;
- **Bus indirizzi:** È il bus attraverso il quale la CPU **decide in quale indirizzo andare** a scrivere o a leggere informazioni. È unidirezionale;
- **Bus controllo**: insieme di collegamenti il cui scopo è **coordinare le attività del sistema**; tramite esso, la CPU può decidere quale componente deve scrivere sul bus dati in un

determinato momento, quale indirizzo leggere sul bus indirizzi, quali celle di memoria devono essere scritte e quali invece lette, etc.

Concessione del bus

È la funzione che gestisce il possesso del bus per evitare ambiguità quando più master richiedono contemporaneamente il suo utilizzo; Esistono due tipi di meccanismi di arbitraggio:

- **Centralizzata**: un **arbitro** decide l'accesso con criteri di priorità (gerarchia) o di fairness (prima o poi ogni dispositivo accede al bus). A.e. la richiesta indipendente o **daisy chain** dove la priorità viene decisa dalla posizione fisica lungo la linea di richiesta.
- **Decentralizzata**: Non esiste l'arbitro. Ogni unità funzionale ha **priorità fissata e diversa**. Se più dispositivi hanno bisogno del bus, questo viene preso da chi ha **priorità maggiore**. Questo tipo di arbitraggio utilizza solo 3 linee:
    - **Linea di Richiesta**, che effettua la richiesta del bus.
    - **Busy**, è attivata dal master che ne fa richiesta.
    - **Linea di arbitraggio**, che collega tutti i dispositivi, è usata per arbitrare il bus ed è alimentata a 5 volt.

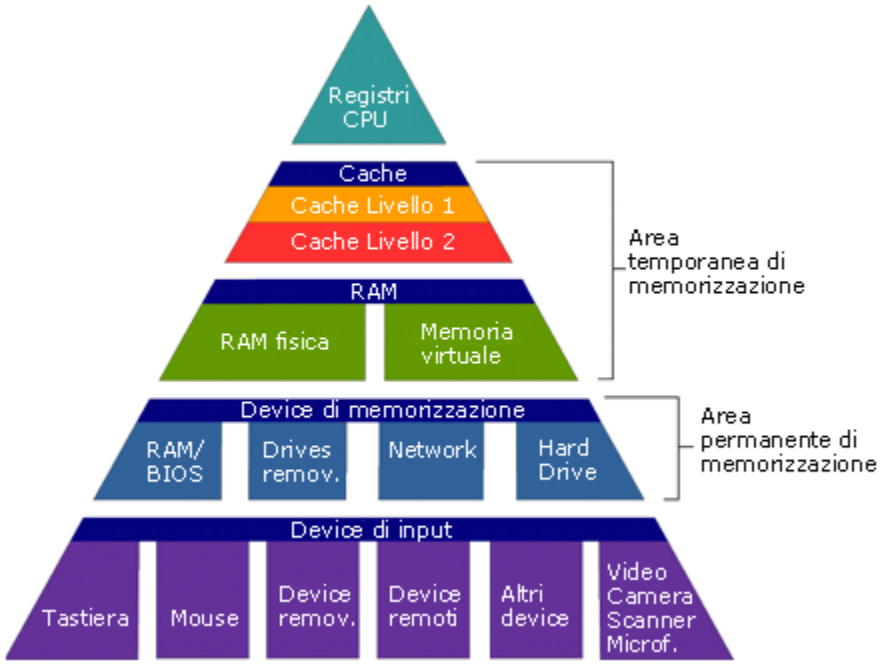
  Quando nessuno richiede il bus, la linea di arbitraggio si propaga per tutti i dispositivi, mentre se un dispositivo lo richiede, dovrà controllare se il bus è libero e se il segnale della linea di arbitraggio (IN) è attivato; in questo caso nega OUT in modo che tutti i dispositivi successivi vedranno IN negato e a loro volta negheranno OUT. In questo modo il dispositivo con priorità maggiore che ha fatto richiesta, avrà il possesso del bus, attiverà la linea di Busy e OUT, e inizierà il proprio trasferimento. Questo schema è simile a quello Daisy Chaining a differenza che è privo di arbitro, questa differenza lo rende più economico, veloce e non soggetto a eventuali guasti da parte dell'arbitro.

9. Organizzazione della memoria fisica

I dati in un elaboratore vengono salvati su supporti organizzati come una **struttura gerarchica** in cui al livello più basso troviamo i dispositivi di **memoria più lenti,** con **capacità maggiore** e **meno costosi**; man mano che si sale, i supporti hanno capacità sempre più piccola, tempo di accesso sempre più veloce e costo maggiore.

In un elaboratore comune una tipica gerarchia di memoria è composta da:

- Il **disco fisso**, dove le informazioni vengono conservate in maniera permanente;
- La **memoria principale** (DRAM/SDRAM), di elevata capacità nell'ordine del GB, basso costo e basso tempo di accesso. Qui vengono conservate le i**nformazioni relative ai processi in esecuzione**: quando un nuovo processo viene caricato, vi verranno trasferite almeno le informazioni basilari che ne permetteranno la messa in esecuzione.
- La **memoria cache**; possono esistere più livelli di questo tipo di memoria:
    - il livello cache L1 è “on-chip”, ossia è direttamente presente nella CPU in maniera tale da ridurre fisicamente la latenza dei collegamenti.
- I **registri del processore** il cui tempo di accesso è valutabile in un ciclo di clock.



10. Memoria cache

La memoria cache è una **memoria veloce** (rispetto alla memoria principale), relativamente **piccola**, non visibile al software e completamente gestita dall'hardware, che memorizza i dati più **recentemente usati** della memoria principale o memoria di lavoro del sistema. La funzione della memoria cache è di **velocizzare gli accessi alla memoria principale** aumentando le prestazioni del sistema.

Funzionamento

La logica funzionale della memoria cache è la seguente:

1. Verifica la **presenza del dato** memorizzato nella cache;
2. In caso di esito positivo (**Hit**), il dato è direttamente e velocemente trasferito dalla cache al processore senza coinvolgere la MM;
3. Altrimenti (**Miss**), il dato è letto dalla MM.

Il funzionamento della memoria cache si basa principalmente su due principi di località:

- **Località temporale**: dati recentemente usati hanno un'alta probabilità di essere nuovamente usati a breve. I dati letti dalla MM sono temporaneamente memorizzati in cache.
- **Località spaziale:** se un dato viene fornito, è molto probabile che dati adiacenti siano a breve a loro volta acceduti. Le istruzioni e i dati sono trasferiti dalla MM alla cache in **blocchi fissi**, noti come **cache lines**. Dopo il primo accesso alla MM, tutti i dati relativi alla linea di cache sono disponibili nella cache stessa.

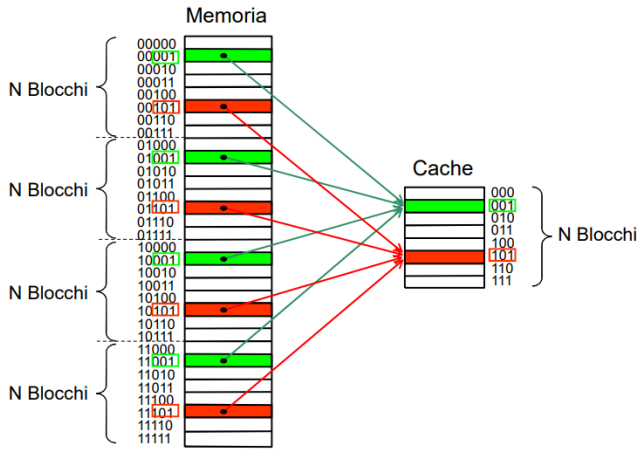
Struttura di base della cache associativa

Ci sono tre strutture di base per le cache:

- Cache ad indirizzamento diretto (cache mappata direttamente);
- Cache Fully Associative (cache completamente associativa);
- Cache Set Associative (cache parzialmente associativa).

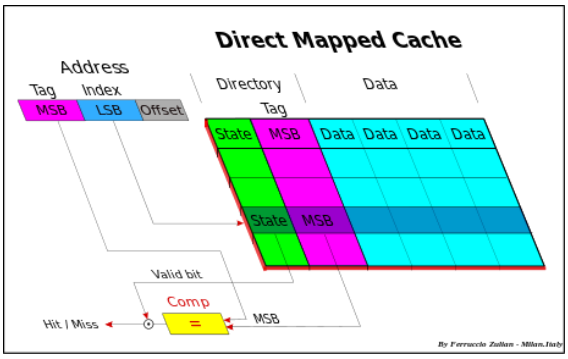
Cache ad indirizzamento diretto (direct mapped)

Nella cache ad indirizzamento diretto ad ogni indirizzo di memoria principale corrisponde **una ed una sola** posizione nella cache come illustrato nella figura:

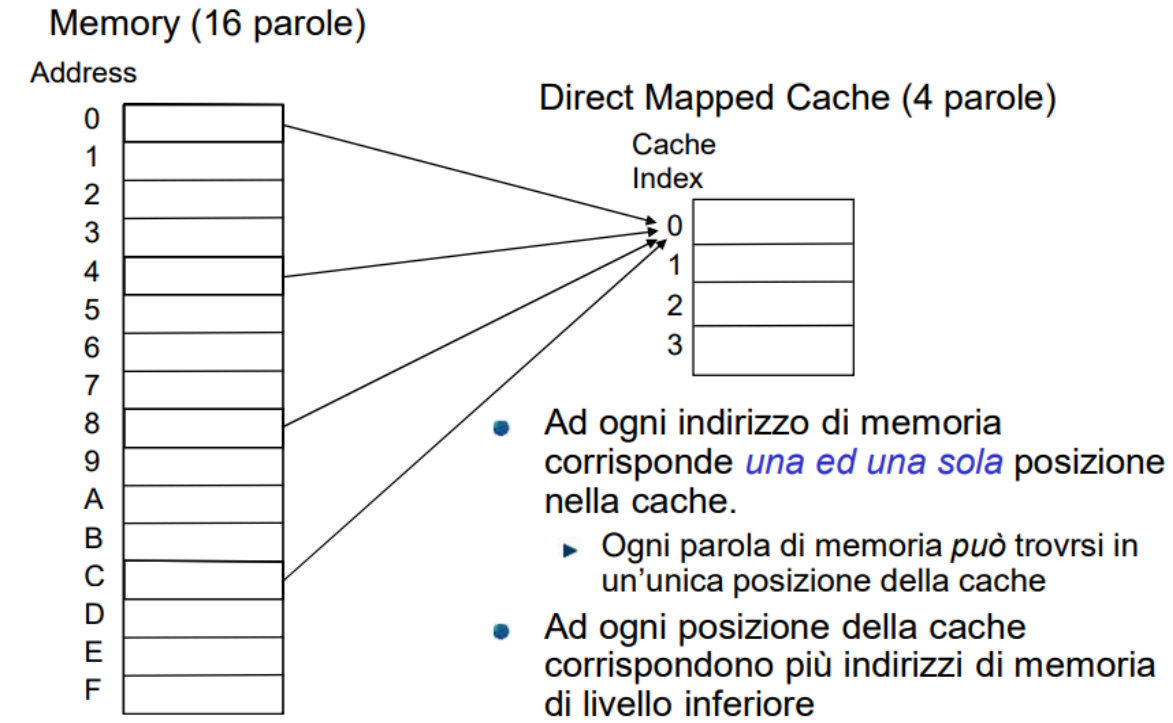


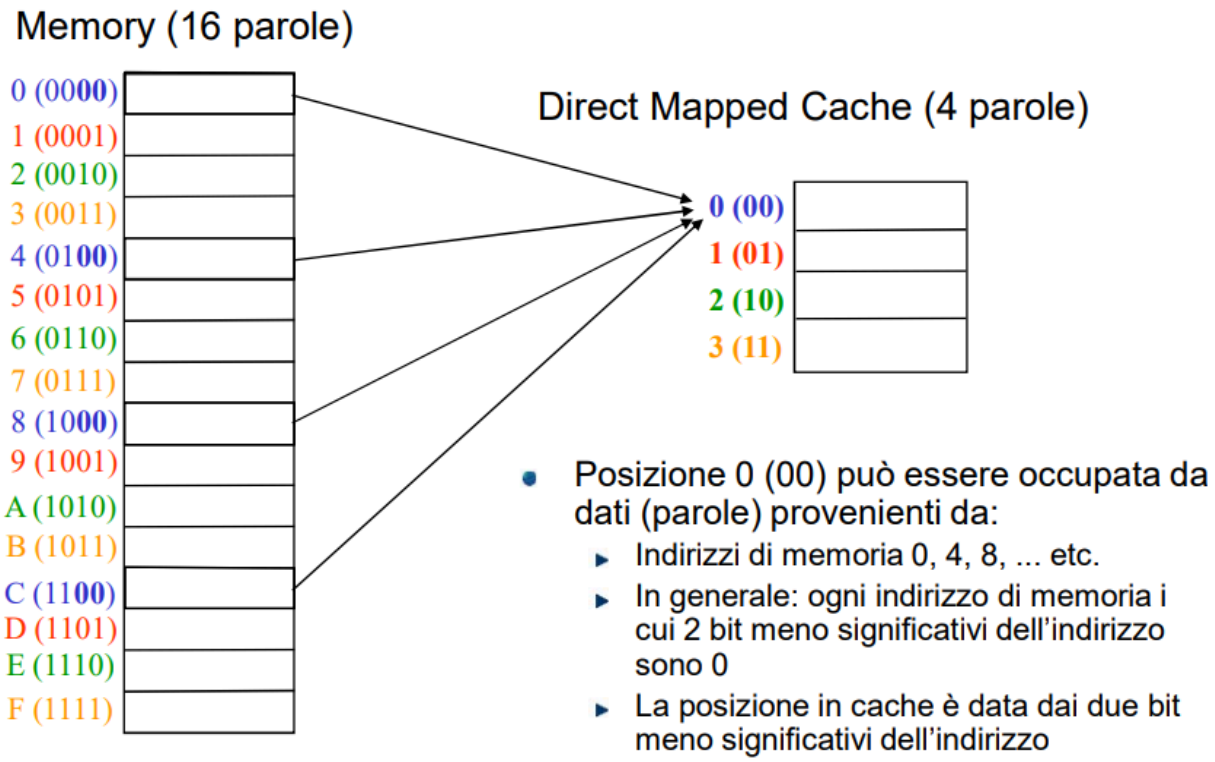
Ogni posizione nella cache include:

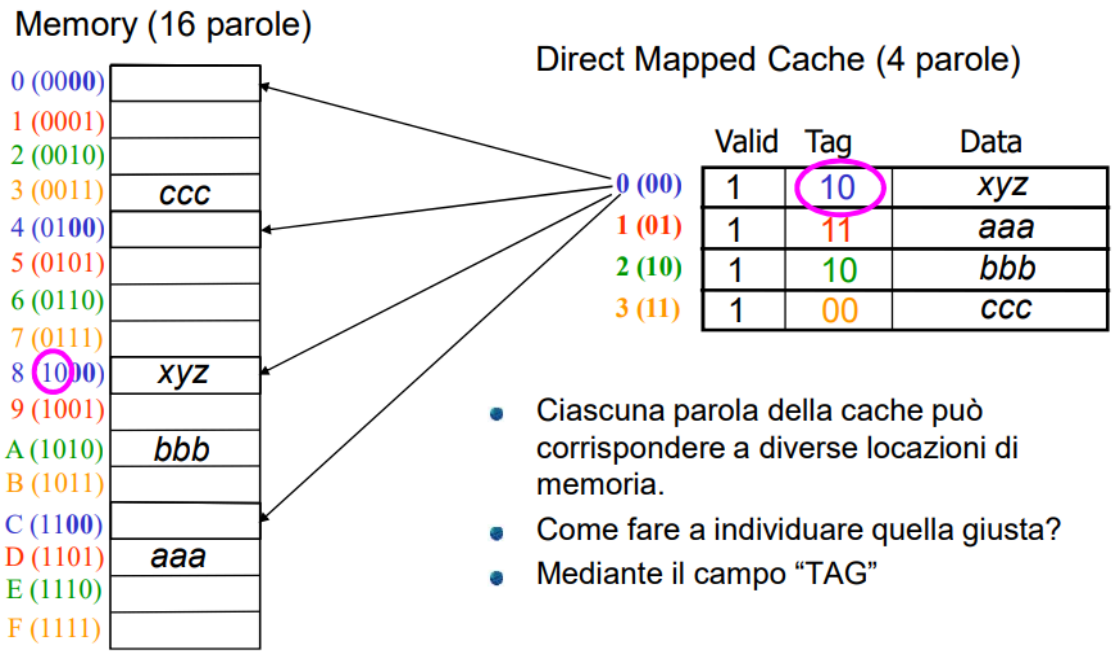
Valid bit	Campo etichetta (tag)	Campo dati
indica se questa posizione contiene o meno dati validi. 0: posizione di cache non utilizzata 1: posizione di cache occupata con dei dati dalla RAM	include l'indirizzo del blocco in RAM, eccetto i k bit meno significativi	contiene una copia del blocco in RAM



Esempio di indirizzamento diretto



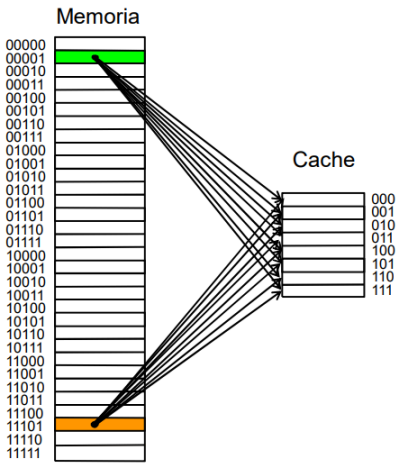




- Ciascuna parola della cache può corrispondere a diverse locazioni di memoria.
- Come fare a individuare quella giusta?
- Mediante il campo “TAG”

Cache completamente associativa

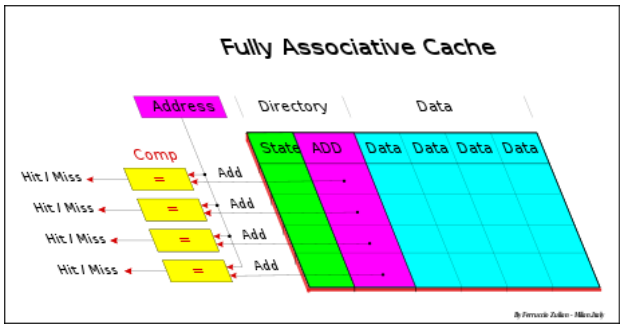
Nella cache completamente associativa un blocco può essere memorizzato **in qualunque posizione** della cache, come illustrato nella figura:



Cercare un dato nella cache richiede il **confronto di tutte le etichette** presenti in cache con l'etichetta dell'indirizzo di memoria richiesto. Nel momento della ricerca, possono avvenire due scenari:

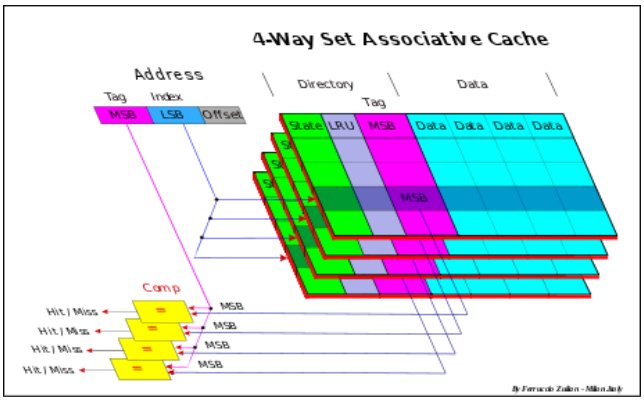
- **Cache hit**: l'etichetta viene trovata;
- **Cache miss**: l'etichetta non viene trovata. si accede alla RAM, il blocco acceduto nella cache viene memorizzato in una posizione che verrá scelta in due modi:
    - Se la **cache non è ancora piena**: in un blocco vuoto qualsiasi di cache, dove verrá poi copiato il blocco della RAM (insieme con il suo tag):
    - Se la **cache è piena**, è necessario sostituire un dato secondo delle politiche (Replacement policy).

La cache Fully Associative ha un'**alta efficienza**. Il dato può essere memorizzato in qualsiasi posizione della cache, ma è **costosa** in termini di circuiti. Necessita di vie di accesso simultanee indipendenti ed un comparatore per ogni entry di cache. Perciò le dimensioni di



questa cache sono in genere molto piccole ed è usata solamente per casi specifici. Normalmente non è mai usata come Cache Memory, ma vengono invece usate le Direct-Mapped e le Set-Associative cache.

Cache completamente associativa a n vie



La cache Set Associative è una combinazione dei due approcci precedenti ed è usata per ridurre il conflitto tra sinonimi.

Questa cache è composta da un **set di cache Direct Mapped identiche**, indirizzate nello stesso identico modo, cosicché per ciascuna entry è disponibile un "set" di alternative linee di cache per memorizzare più di un sinonimo.

I sinonimi possono essere memorizzati in qualsiasi set delle entry selezionate, in funzione dell'algoritmo di "politica di rimpiazzo" utilizzata.

Nella cache Set Associative viene usato un comparatore di indirizzi per ogni set. Una cache Direct Mapped può essere vista come una cache Set Associative con un solo set e una cache Fully Associative con n-linee come una cache Set Associative a n-vie (set) con una sola entry.

Esercizi

Quali locazioni di una memoria principale di 64 kB può contenere la linea 1023 di una cache a 32 byte composta da 1024 entry?

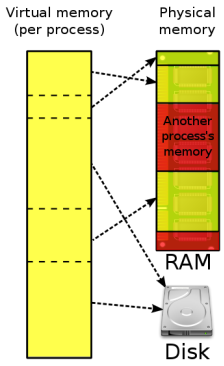
R: Ogni linea di cache è composta da 32 bit, per capire qual è la prima locazione contenuta nella linea 1023 bisogna moltiplicare $32 \cdot 1023 =$ **32736**. A questo risultato, tenendo conto che ogni linea ha 32 bit, è sufficiente aggiungere 31 (non 32 perché il primo bit è quello della locazione già calcolata) per trovare l'ultima locazione $32736 + 31 =$ **32767**.

Essendo la cache una memoria contigua, dopo aver percorso tutta la cache una volta ($32 \cdot 1024$), bisognerà ripercorrerla fino alla linea 1023 ($32 \cdot 1023$). Quindi le successive locazioni andranno dalla $32 \cdot 1024 + 32 \cdot 1023 =$ **65504** fino alla $65504 + 31 =$ **65535**.

Assumendo uno spazio indirizzabile di memoria uguale a 4 GB, qual `e la dimensione di una cache a corrispondenza diretta (cio`e ad accesso diretto) nel caso in cui ogni linea di cache occupa 128 byte e il campo TAG che contiene l'indirizzo di memoria da cui proviene la linea `e lungo 14 bit?

R: Lo spazio indirizzabile in questione è associato a indirizzi di 32 bit. Di questi, i 14 bit più significativi indirizzano la linea e quindi fanno parte del campo TAG. In parallelo, i 7 meno significativi indirizzano il word e l'eventuale byte nella linea. Conseguentemente restano 11 bit per indirizzare ogni riga della cache, che sarà lunga 1 (validita) + 14 (TAG) + 8·27 = 1039 bit. La dimensione della cache in definitiva `e di 1039 · 2 11 = 2127872 bit ≈ 260 kB.

11. Memoria virtuale



È un'architettura di sistema capace di **simulare uno spazio di memoria principale** maggiore di quello fisicamente presente o disponibile, dando l'illusione all'utente di un enorme quantitativo di memoria. Il sistema operativo **mappa** gli indirizzi di memoria usati da un programma, chiamati indirizzi virtuali, in indirizzi fisici utilizzando spazio di memoria secondaria (unità a disco).

Vantaggi

I vantaggi principali di questa architettura sono:

- Una **maggiore sicurezza** dovuta all'isolamento della memoria (si può utilizzare un dato in memoria ram senza esporre tutta la ram);
- La possibilità di **condividere alcune pagine** di memoria **fra diversi processi**;
- Potere usare più memoria di quella disponibile con una tecnica chiamata **swap**.

Swapping

In ambiente POSIX (a.e. su linux), la memoria di massa utilizzata a questo scopo è comunemente chiamata "*swap*" o "*spazio di swap*", mentre, in ambiente Windows, è chiamata "*file di paging*".

Le operazioni di spostamento delle pagine dallo spazio di swap alla memoria fisica sono chiamate "swapping": gli indirizzi virtuali vengono tradotti in indirizzi fisici reali dalla MMU (incorporata nella cpu). La MMU svolge i seguenti compiti:

- **Traduce** l'indirizzo virtuale in indirizzo fisico;
- **Controlla** che l'indirizzo fisico corrisponda a una zona di memoria fisicamente presente nella memoria principale;
- Se invece la zona in questione è nello spazio di swap, la MMU **solleva una eccezione** di page fault e il sistema operativo si occupa di caricarla in memoria centrale, scartando una pagina già presente.

Memoria virtuale paginata

Con questo schema la memoria principale viene **divisa in pagine** tutte della stessa grandezza (4 o 8 kilobyte o comunque una potenza di 2). Se il numero delle pagine è molto grande il meccanismo associativo può diventare troppo complesso, rallentando sensibilmente l'accesso alla memoria (e quindi tutto il sistema).

Gestione della memoria virtuale con paginazione

L'immagine di un processo (codice, variabili e stack) e la ram sono suddivisi in pagine di dimensione fissa e uguale (quelle della ram sono dette page frame). Ad ogni processo è associata una tabella, mantenuta in memoria principale o secondaria a seconda delle dimensioni, detta **tabella delle pagine**. Ogni voce (riga) della tabella delle pagine contiene:

- Il **numero di pagina** per quella riga;
- Il **bit present**:
    - se è 0, la pagina non è presente in memoria principale, si genera quindi un Page Fault (trap) e si attende che la pagina sia caricata in memoria;
    - Se è 1, si calcola l'indirizzo fisico: $indirizzo\ base + offset$;
- Il **bit modified**: indica invece se la pagina è stata modificata o meno. Infatti se una pagina non è stata modificata, al momento di effettuare lo swap nella memoria secondaria, non ha senso riscrivere le pagine su disco. Risparmiando così tempo, e migliorando in parte le prestazioni;
- Il **numero di frame corrispondente**.

Traduzione (mappatura) indirizzo virtuale in indirizzo fisico

Si ha $k = n + m$, con:

- k = lunghezza degli indirizzi fisici e virtuali (bit necessari per indirizzare la mem fisica);
- n = numero di bit destinati al numero di pagina (bit per indirizzare ogni pagina);
- m = bit dedicati all'offset.

Una mem. fisica di 2^k byte è paginata in pagine di 2^m byte. La page table deve possedere 2^{k-m}

Supponiamo ora che volendo tradurre un indirizzo virtuale (nr. pagina, offset), si trovi che il frame corrispondente alla data pagina sia l'i-esimo. Questo però non corrisponde ancora al vero indirizzo fisico (page frame), ma solamente all'indice. Per ottenere l'indirizzo fisico si deve ora moltiplicare $i \cdot 2^m$.

Il vantaggio di questa tecnica consiste nel fatto che, nel sistema binario, questa operazione può essere eseguita concatenando m zeri alla rappresentazione binaria di i. Un'operazione quindi molto veloce, che può essere eseguita direttamente in hardware.

Memoria virtuale segmentata

I programmi che girano su sistemi con memoria segmentata sono strutturati in **segmenti funzionalmente omogenei** gestiti dalla MMU

Vantaggi e svantaggi

Il notevole svantaggio di questo sistema, invece, è il grande spreco di memoria dovuto alla frammentazione esterna: la memoria viene allocata e deallocata in blocchi di varie dimensioni che lasciano un sempre maggior numero di "**buchi**" vuoti, troppo piccoli per poter essere utilmente allocati.

Gestione della memoria virtuale con paginazione

Più precisamente, il meccanismo di gestione della memoria virtuale segmentata è il seguente. Ad ogni processo è associata una **tabella dei segmenti**; ogni voce di questa tabella rappresenta un segmento del processo e contiene almeno i seguenti campi:

- il bit Present
- il bit Modified
- l'indirizzo base del segmento in memoria.

Gli indirizzi logici sono rappresentati dalla coppia (nr. di segmento, displacement). La traduzione avviene in questo modo:

- si trova la voce della tabella corrispondente al nr. di segmento;
- se il bit Present è 0, il segmento non è presente in memoria principale, si genera quindi un Segment Fault e si attende che il segmento sia caricato in memoria.
- Infine si genera l'indirizzo fisico sommando il displacement all'indirizzo base del segmento in memoria.

Esercizi

In una memoria paginata, la page table si compone di righe ciascuna lunga 7 bit. Se la memoria principale paginabile è di 1 Mbyte, di quanti bit si compone il campo offset di ogni indirizzo fisico di memoria? Qual è la dimensione di ogni pagina?

R: Una memoria di 1 Mbyte (2^{20}) necessita di 20 bit per essere indirizzata. Escludendo il bit di presenza/assenza della pagina in memoria, la page table contiene 6 bit per indirizzare ogni pagina. Dunque, restano 20 − 6 = 14 bit per specificare l'offset nella stessa pagina, la quale dunque avrà dimensione 2^{14} = 16 kB.

Una memoria fisica di 1 Gbyte (2^{30}) è paginata in pagine di 64 kB (2^{16}), che possono trovare posto in posizioni casuali di una memoria principale di 4 Mbyte (2^{22}). Non è prevista la presenza di memoria virtuale. Come dev'essere dimensionata in questo caso la page table?

R: La memoria fisica necessita di 30 bit per essere indirizzata. Di essi, 16 individuano l'offset all'interno di ogni pagina. La page table dunque deve possedere $2^{30-16} = 2^{14}$ locazioni corrispondenti alla parte più significativa dell'indirizzo. Ciascuna di esse dovrà contenere 22 bit per indirizzare ciascuna pagina casualmente in 4 MB (2^{22}), più il bit di presenza/assenza della pagina in memoria principale per un totale di 2^{14} locazioni di 23 bit ciascuna.

$k = 30,\ m = 16 \rightarrow n = k - m = 30 - 16 = 14$

12. Arduino

Arduino è una piattaforma hardware composta da una serie di schede elettroniche dotate di un microcontrollore. È dotata di pin (analogici e digitali) che sono internamente collegati a delle porte. Una porta è un canale di comunicazione MCU e l'esterno (tramite un pin).

Timer

Lo scopo dei timer è quello di **contare** da zero fino a $2^{\#\ bit\ timer}$ (a.e. se timer a 8 bit conta da 0 a 255). Ad ogni ciclo di clock il valore del timer viene incrementato di 1, essendo il clock di Arduino UNO pari a 16MHz, il tic più rapido che può ottenere è pari a $62,5ns$, di conseguenza per incrementare tutto il timer0 saranno necessari $63ns \cdot 256$.

Tuttavia è possibile, tramite il **prescaler**, fare in modo che il contatore salga di una unità ogni x cicli di clock. Il prescaler può assumere i seguenti valori:

- **0** se il timer è disattivato;
- **1** (un tic del clock incrementa di 1 il valore del timer, significa che ogni $63ns \cdot 1$ il contatore salirà di 1 valore);
- **8, 64, 256 e 1024** (ogni 1024 cicli di clock, il timer incrementa di 1 il valore del contatore, ciò significa che il contatore incrementa di 1 ogni $63ns \cdot 1024$).

Esistono tre tipi di timer:

- **Timer 0**: timer a 8 bit, conta da 0 a 255 e viene usato dalla funzione *delay()*;
- **Timer 1**: timer a 16bit, conta da 0 a 65535 e viene usato dalle librerie sui servomotori;
- **Timer 2**: timer a 8bit, usato dalle librerie per i buzzer.

Dealy di 2 secondi usando il timer:

$$delay = \frac{1}{clock/prescaler} \cdot periodo \cdot bit_{max}\ timer$$

$$delay = \frac{1}{16 \cdot 10^6/256} \cdot 2 \cdot 65535 = 2s$$

13. Assembly

Simulatore Armsim a 32 bit:

- **Memoria principale** di 2^{32} locazioni da 1 byte (4GB totali)
- **16 registri** da 32 bit denominati $R0 \rightarrow R15$ dove:
    - $R0 \rightarrow R12$ sono registri di uso generale;
    - $R13$ usato come *stack pointer* (contiene l'indirizzo della locazione di memoria occupata dal top dello stack);
    - $R14$ usato come *link register* (contiene l'indirizzo a cui tornare al termine di una chiamata di funzione);
    - $R15$ usato come *program counter* (conservare l'indirizzo di memoria della prossima istruzione da eseguire);
- Un **registro di stato** $CPSR$ (Current Program Status Register)
    - 4 bit esprimono le condizioni (Negative, Carry, Zero e oVerflow);

Instruction set ARM (traslazioni e rotazioni)

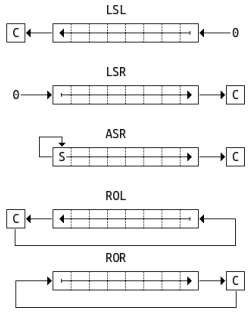
Il linguaggio Assembly è composto da istruzioni che sono gli elementi costitutivi principali. Le istruzioni ARM sono generalmente seguite da uno o due operandi e generalmente utilizzano il seguente modello:

MNEMONIC{S}{condizione} {Rd}, Operand1, Operand2

Lo scopo dei campi nel modello è descritto come segue:

- **MNEMONI**C: Nome abbreviato dell'istruzione;
- **{S}**: suffisso facoltativo. Se viene specificato S, i flag di condizione vengono aggiornati sul risultato dell'operazione;
- **{condizione}**: Condizione che deve essere soddisfatta per eseguire l'istruzione. Il campo condizione è strettamente legato al valore dei registri CPSR, o per essere precisi, ai valori di bit specifici all'interno del registro;

- **{Rd}**: Registro (destinazione) per memorizzare il risultato dell'istruzione;
- **Operand1**: Primo operando. O un registro o un valore immediato;
- **Operand2** - Secondo operando (flessibile). Può essere un valore immediato (numero) o un registro con uno spostamento opzionale. É un registro flessibile in quanto puo essere usato in varie forme:
    - **#123** - Valore immediato (con set di valori limitato);
    - **Rx** - Registro x (come R1, R2, R3 ...);
    - **Rx, ASR n -** Registro x con spostamento aritmetico a destra di n bit (1 = n = 32);
    - **Rx, LSL n -** Registro x con spostamento logico a sinistra di n bit (0 = n = 31);
    - **Rx, LSR n -** Registro x con spostamento logico a destra di n bit (1 = n = 32);
    - **Rx, ROR n -** Registro x con rotazione a destra di n bit (1 = n = 31);
    - **Rx, RRX -** Registra x con ruota a destra di un bit, con estensione.



Esempio

Movle r0, #5 → Sposta il numero 5 (Operand2, perché il compilatore lo considera MOVLE R0, R0, #5) su R0 (Rd) SOLO se la condizione LE (Minore di o Uguale) è soddisfatta;

Mov r0, r1, lsl #1 → Sposta il contenuto di R1 (Operando2 sotto forma di registro con spostamento logico a sinistra) spostato a sinistra di un bit su R0 (Rd). Quindi, se R1 aveva valore 2, viene spostato a sinistra di un bit e diventa 4. 4 viene quindi spostato su R0.

Load e store

In genere, **ldr** viene utilizzato per caricare qualcosa dalla memoria in un registro e **str** viene utilizzato per archiviare qualcosa da un registro in un indirizzo di memoria.