Appunti Architettura degli Elaboratori

Domande e Risposte

Capitolo 1 – Rappresentazione dell'Informazione

1) Cosa si intende per informazione?

L'informazione è tutto ciò che viene manipolato da un calcolatore e può essere rappresentato mediante suoni, numeri ed etc. La più piccola unità di informazione viene denominata bit (Binary Digit, 0/1) e tutte le informazioni sono rappresentate in bit. Un bit può assumere soltanto due valori (0 e 1).

2) Cosa si intende per rappresentazione dell'informazione?

La rappresentazione dell'informazione è la modalità con la quale l'informazione viene descritta.
Esempio: il valore "16" può essere rappresentato in diverse modalità:
→ sistema romano: XVI;
→ sistema decimale: 16;
→ sistema binario: 10000;
→ sistema esadecimale: 0x10.
Gli standard di codifica sono delle regole che vengono utilizzate nella rappresentazione dei dati in formato binario che vanno a coprire le più varie necessità di rappresentazione dell'informazione.

3) Cosa si intende per bit?

Il bit (Binary Digit) è l'unità di misura dell'informazione che può assumere soltanto due valori: 0 e 1. Combinando tra loro più bit si ottengono strutture più complesse, in particolare:
→ byte, 8 bit;
→ nybble, 4 bit;
→ word, 32 bit;
→ half word, 16 bit;
→ double word, 64 bit.

4) Quante configurazioni diverse può assumere una sequenza di n bit?

Dati n bit, il numero di configurazioni ottenibili è pari a 2^n bit.
Esempio:

Sia B = {0,1} e a e b appartenenti a B, si ha che:

a	b
0	0
0	1
1	0
1	1

5) Differenza tra sistema numerico posizionale e non posizionale

Il sistema di numerazione posizionale è il sistema di numerazione in cui i simboli usati per scrivere i numeri assumono valori diversi a seconda della posizione che occupano nella notazione (posizione delle unità, delle decine, centinaia ed etc.).
Ad esempio, nel sistema decimale se si prendono le cifre 1 e 2 si ha che:
→ il valore 12 rappresenta una decina e due unità;
→ il valore 21 rappresenta due decine e una unità.
Il valore che ha una cifra cambia a seconda della sua posizione nel numero.
Il sistema di numerazione non posizionale è il sistema di numerazione in cui ogni cifra assume sempre lo stesso valore, come ad esempio il sistema romano.
Ad esempio IV = 4 e non 15.
Il sistema di numerazione posizionale un valore numerico N è caratterizzato dalla seguente rappresentazione:

$$N = d_{n-1}d_{n-2}\ldots d_1 d_0, d_{-1}\ldots d_{-m}$$

$$N = d_{n-1}\cdot r^{n-1} + \ldots + d_0 \cdot r^0 + d_{-1}\cdot r^{-1} + \ldots + d_{-m}\cdot r^{-m}$$

$$N = \sum_{i=-m}^{n-1} d_i \cdot r^i$$

Dove:
→ d rappresenta la singola cifra (digit);
→ r è la radice o base del sistema;
→ n è il numero di cifre della parte intera (sinistra della virgola);
→ m (è il numero di cifre della parte frazionaria).

6) Come viene rappresentato un valore numerico N nel sistema decimale?

Nel sistema decimale, un valore numerico N viene rappresentato mediante la seguente notazione:

$$N = d_{n-1}\cdot 10^{n-1} + \ldots + d_0 \cdot 10^0 + d_{-1}\cdot 10^{-1} + \ldots + d_{-m}\cdot 10^{-m}$$

Nota bene: tutte le volte che si farà riferimento ad un valore senza specificarne la base, lo si considererà in base 10. In caso contrario la base verrà specificata come pedice nella cifra di peso più basso.

Esempio:

$$123{,}45 = 1\cdot 10^{2} + 2\cdot 10^{1} + 3\cdot 10^{0} + 4\cdot 10^{-1} + 5\cdot 10^{-2}$$

7) Come viene rappresentato un valore numerico N nel sistema binario?

Nel sistema binario, un valore numerico N viene rappresentato mediante la seguente notazione:

$$N = d_{n-1}\cdot 2^{n-1} + \ldots + d_{0}\cdot 2^{0} + d_{-1}\cdot 2^{-1} + \ldots + d_{-m}\cdot 2^{-m}$$

Nota bene: Avendo a disposizione n bit, è possibile codificare valori compresi nel range $[0,2^{n-1}]$. È possibile definire 2^{n} codifiche binarie.

Esempio:

$$101_{2} = 1\cdot 2^{2} + 0\cdot 2^{1} + 1\cdot 2^{0} = 5_{10}$$

Il byte è una sequenza di 8 bit consecutivi: 11010010

MSB (Most Significant Bit) = 1 (bit più a sinistra)

LSB (Least Significant Bit) = 0 (bit più a destra)

8) Come viene rappresentato un valore numerico N nel sistema ottale?

Nel sistema ottale, un valore numerico N viene rappresentato mediante la seguente notazione:

$$N = d_{n-1}\cdot 8^{n-1} + \ldots + d_{0}\cdot 8^{0} + d_{-1}\cdot 8^{-1} + \ldots + d_{-m}\cdot 8^{-m}$$

Esempio:

$$127_{8} = 1\cdot 8^{2} + 2\cdot 8^{1} + 7\cdot 8^{0} = 87_{10}$$

9) Come viene rappresentato un valore numerico N nel sistema esadecimale?

Nel sistema esadecimale, un valore numerico N viene rappresentato mediante la seguente notazione:

$$N = d_{n-1}\cdot 16^{n-1} + \ldots + d_{0}\cdot 16^{0} + d_{-1}\cdot 16^{-1} + \ldots + d_{-m}\cdot 16^{-m}$$

Esempio:

$$A1_{16} = A\cdot 16^{1} + 1\cdot 16^{0} = 161_{10}$$

Spesso si usa il pedice $_{H}$ al posto del pedice $_{16}$ per indicare la base esadecimale oppure 0x davanti al numero (0xA1). Il sistema esadecimale viene abbreviato

come hex.

r = 16	r = 10	r = 2
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

10) Come viene convertito un valore di base r ad un valore di base 10?

La conversione da qualsiasi base r a base 10 avviene come segue:

$$N_r = d_{n-1} \cdot r^{n-1} + \ldots + d_0 \cdot r^0 = M_{10}$$

Esempi:

$$1010_2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)_{10} = 10_{10}$$

$$26_8 = (2 \cdot 8^1 + 6 \cdot 8^0)_{10} = 22_{10}$$

$$431_5 = (4 \cdot 5^2 + 3 \cdot 5^1 + 1 \cdot 5^0)_{10} = 116_{10}$$

11) Come viene convertito un valore di base 10 ad un valore di base r?

La conversione da base 10 a qualsiasi base r avviene come segue:

→ si divide il valore numerico N per la base r fino a quando l'ultimo quoziente è minore della base stessa r;

→ si prende l'ultimo quoziente e tutti i resti delle divisioni, e procedendo dall'ultimo resto al primo, li scriviamo da sinistra verso destra.

Esempio 1: convertire il numero 12 base 10 a base 2

Valore	Base	Quoziente	Resto
12	2	6	0
6	2	3	0
3	2	1	1
1	2	0	1

$12_{10} = 1100_2$

Esempio 2: convertire il numero 120 base 10 a base 8

Valore	Base	Quoziente	Resto
120	8	15	0
15	8	1	7
1	8	0	1

$120_{10} = 170_8$

Esempio 3: convertire il numero 19 base 10 a base 16

Valore	Base	Quoziente	Resto
19	16	1	3
1	16	0	1

$19_{10} = 13_H$

12) Come viene convertito un valore di base 2 ad un valore di base 16?

Viene convertito nella seguente maniera:

Dato 111111100011010 si ha che:

111	1111	0001	1010	→	7	15	1	10	→	7	F	1	A

13) Come viene convertito un valore di base p a qualsiasi base q?

La conversione da qualsiasi base p a qualsiasi base q avviene come segue:

→ convertire il numero da base p a base 10;

→ convertire il risultato da base 10 a base q.

È possibile passare dalla base p a q (tale che p e q siano potenze di 2) passare dalla base 2 e non dalla base 10. La conversione tra una base di potenza di 2 e la base 2 è molto più veloce.

Esempio: convertire il numero $AB2_{16}$ in binario.

r = 16	r = 10	r = 2
A	10	1010
B	11	1011
2	2	0010

$AB2_H = 1010101100010_2$

14) Descrivere la rappresentazione dei valori

→ La rappresentabilità dei valori è legata al numero di cifre disponibili e nei sistemi di elaborazione il numero di cifre impiegate nella rappresentazione è limitato.

→ Si ha un overflow quando si è nell'impossibilità di rappresentare il risultato di una operazione (somma o sottrazione) con il numero di cifre a disposizione.

Esempio:

n bit	2^n valori	2^n-1
1	2	1
2	4	3
3	8	7
4	16	15
5	32	31
6	64	63
…	…	…

La rappresentazione dei valori nel sistema binario è diversa rispetto al sistema decimale. Infatti:

→ kilo: in informatica significa $2^{10} = 1024$;

→ mega: significa $2^{20} = 1048576$;

→ giga: significa $2^{30} = 1073741824$.

15) Come viene effettuata la somma nel sistema binario?

La somma tra i bit di pari ordine viene effettuata nella seguente maniera:

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0          con riporto di 1 sul bit di ordine superiore
1 + 1 + 1 = 1      con riporto di 1 sul bit di ordine superiore
```

La somma è definita su 3 elementi:

→ due addendi;

→ il riporto (carry).

Esempio: somma tra 010011 e 010001

1)

Riporto						
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma						

2)

Riporto					1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma						0

3)

Riporto				1	1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma					0	0

4)

Riporto				1	1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma				1	0	0

5)

Riporto			0	1	1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma			0	1	0	0

6)

Riporto		0	0	1	1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma		0	0	1	0	0

7)

Riporto	1	0	0	1	1	
1 addendo	0	1	0	0	1	1
2 addendo	0	1	0	0	0	1
Somma	1	0	0	1	0	0

16) Come viene effettuata la sottrazione nel sistema binario?

La sottrazione tra i bit di pari ordine viene effettuata nella seguente maniera:

```
0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1   con prestito dal bit di ordine superiore
```

Anche la sottrazione opera su gruppi di 3 bit

→ minuendo e sottraendo;

→ prestito (borrow) proveniente dalla cifra di ordine immediatamente superiore.

Esempio: sottrazione tra 11101 e 01110 (29 e 14)

1)

Prestito						
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza						

2)

Prestito						
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza						1

3)

Prestito					2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza					1	1

4)

Prestito					2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza					1	1

Prestito di unità → il minuendo sarà 0 e non 1

5)

Prestito				2	2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza				1	1	1

6)

Prestito			2	2	2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza			1	1	1	1

7)

Prestito			2	2	2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza		0	1	1	1	1

8)

Prestito			2	2	2	
Minuendo		1	1	1	0	1
Sottraendo		0	1	1	1	0
Differenza		0	1	1	1	1

17) Cosa si intende per overflow?

L'overflow è una condizione che rispecchia un errore nella rappresentazione di un certo numero (risultato di una operazione) dovuto al fatto che la quantità di cifre disponibili è minore rispetto a quelle necessarie a rappresentare il numero. Questa situazione si rappresenta quando viene superata la capacità massima del registro aritmetico di un calcolatore, ovvero il risultato

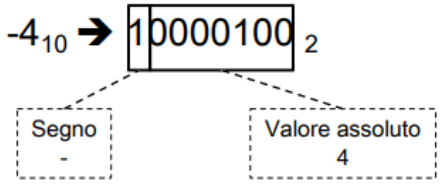
dell'operazione impostata è un numero con tante cifre da eccedere il massimo numero rappresentabile.

18) Come risolvere il problema dell'overflow?

La modalità standard di rappresentazione non permette la rappresentazione di numeri negativi. Per ovviare questo problema è stato definito un metodo di rappresentazione dal nome di Modulo e Segno (MS).
Esistono altre tipologie di rappresentazioni come:
→ complemento a 1 (CA1);
→ complemento a 2 (CA2);
→ eccesso 128.

19) Descrivere il metodo di Modulo e Segno

→ Il metodo di Modulo e Segno (MS) è una rappresentazione dei numeri relativi in base 2, che estende il sistema numerico binario per rappresentare i numeri negativi.
→ Supponendo di avere a disposizione 1 Byte (8 bit) per rappresentare numeri sia positivi che negativi, questo metodo utilizza i primi 7 bit da destra per il valore assoluto del numero e il MSB per rappresentare il segno.
→ Il bit 1 rappresenta il numero negativo e il bit 0 rappresenta il numero positivo.
Esempio:



→ Con n bit totali, si possono rappresentare i numeri interi nell'intervallo

$$[-(2^{n-1}-1)\ ,\ +(2^{n-1}-1)]_{10}$$

20) Quali sono i problemi relativi alla rappresentazione MS?

I problemi relativi alla rappresentazione MS sono:
→ esistono due diverse rappresentazioni dello 0, come ad esempio presi 4 bit totali si ha:

$0000_2 = +0_{10}$ $1000_2 = -0_{10}$

→ un bit tra tutti i bit disponibili viene "speso" per il segno.

21) Descrivere il metodo del complemento a 1 (CA1)

→ Questo metodo si basa sull'operazione di complemento.

→ Il termine complemento indica l'operazione che associa ad un bit (o ad ogni sequenza di bit) il suo opposto, cioè il valore ottenuto sostituendo tutti gli 1 con 0 e tutti gli 0 con 1.
→ Nel metodo CA1 utilizza due strategie:
1) se il numero da codificare è positivo lo si converte in binario con il metodo tradizionale;
2) se il numero da codificare è negativo basta convertire in binario il suo modulo e quindi eseguire l'operazione di complemento sulla codifica binaria effettuata.
Esempi:

$$3_{10} = 0011_2$$

$$-3_{10} = \overline{0011_2} = 1100_2$$

22) Quale problema presenta il CA1?
Il problema relativo al CA1 è il fatto che esso ammette due diverse rappresentazioni dello 0.
0000 0000 (+0)
1111 1111 (-0)
23) Descrivere il metodo del complemento a 2 (CA2)
→ Il complemento a 2 (CA2) è un altro metodo di codifica usato per rappresentare i numeri interi sia positivi che negativi.
→ Nel metodo CA2 codifica il valore x come segue:
1) se il numero x è positivo, esso rimane invariato;
2) se il numero x è negativo, si effettua il complemento a 1 (CA1) sul valore da codificare e si somma +1 al risultato otenuto con CA1.
→ Il metodo CA2 supera il principale difetto del CA1, ossia la presenza di una doppia codifica per lo 0. In CA2 lo 0 ha un'unica rappresentazione.
→ In CA2 i valori negativi hanno MSB = 1.
→ Dati n bit, si possono rappresentare i numeri nell'intervallo
$[-(2^{n-1}) \ , \ +(2^{n-1}-1)]_{10}$
Esempio: dati 5 bit
$-16_{10} \leq X \leq +15_{10}$
Esempi:

Bit	Valore Assoluto	Complemento a 2
0111 1111	127	127
0111 1110	126	126
0000 0010	2	2
0000 0001	1	1
0000 0000	0	0
1111 1111	255	−1
1111 1110	254	−2
1000 0010	130	−126
1000 0001	129	−127
1000 0000	128	−128

Esistono 3 metodi per il calcolo di CA2 di un numero X:
1) Definizione di complemento alla base $CA2(X) = 2^n - X$;
2) Per calcolare CA2 si calcola CA1 e si somma 1 e partendo dalla definizione di complemento alla base – 1

$$CA1(X) = (2^n - 1) - X$$

possiamo definire CA2 in funzione di CA1

$$CA2(X) = CA1(X) + 1$$

3) si parte da destra, si trascrivono tutti gli 0 fino ad incontrare il primo 1 e si trascrive anch'esso e si complementano a 1 (0 → 1 e 1 → 0) tutti i bit restanti.
Esempio: rappresentazione in CA2 su 4 bit il numero -7

```
7(10) = 111(2)

2^4 - 7 = 10000 - 111 = 1001(CA2)

1) Passando dal CA1
111 -(4 bit)-> 0111 -(CA1)-> 1000(CA1) + 1 = 1001(CA2)
 7              +7

111(2) => 0111(2) -> 1001(CA2)
```

Si conclude che:
→ per rappresentare un numero positivo in CA2 non serve applicare l'operazione di CA2;
→ per rappresentare un numero negativo in CA2 è necessario applicare l'operazione di CA2 alla rappresentazione del corrispondente valore positivo.

24) Come viene eseguita la somma dei due valori in Modulo e Segno?

Confronando i bit di segno si valuta se:
a) i bit di segno sono uguali, allora il bit di segno risultante sarà il bit di segno dei due addendi e si esegue la somma bit a bit (a meno di overflow);

b) i bit di segno sono diversi, allora si confrontano i valori assoluti dei due addendi, il bit di segno risultante sarà il bit di segno dell'addendo con valore assoluto maggiore e si esegue la differenza bit a bit.

25) Come viene eseguita la sottrazione dei due valori in Modulo e Segno?
Confronando i bit di segno, si valuta se:
a) i bit di segno sono uguali, allora il bit di segno risultante sarà uguale al bit di segno dell'operando a modulo maggiore e il risultato avrà modulo pari al modulo della differenza dei moduli degli operandi;
b) i bit di segno sono diversi, allora il bit di segno risultante sarà uguale al bit di segno del minuendo e il risultato avrà modulo pari alla somma dei moduli dei due operandi.

```
A - B = A + (-B)
```

Si può avere overflow solo quando:
→ si sommano due operandi con segno concorde;
→ si sottraggono due operandi con segno discorde.

26) Dove si verifica l'overflow nel metodo di somma e sottrazione in Modulo e Segno?
L'overflow si verifica quando c'è un riporto dalla cifra più significativa del modulo, cioè non si è nella condizione di rappresentare il risultato ottenuto.
Nelle operazioni tra valori rappresentati in MS, gli operandi devono essere rappresentati con lo stesso numero di cifre (si aggiungono gli zeri necessari a sinistra del modulo, prima del bit di segno).

27) Come viene effettuata la somma dei due valori in CA2?
La somma dei due valori in CA2 viene effettuata nella seguente maniera:
1) si esegue la somma su tutti i bit degli addendi, segno compreso;
2) un eventuale riporto (carry) oltre il bit di segno (MSB) viene scartato;
3) nel caso gli operandi siano di segno concorde (entrambi positivi o entrambi negativi) occorre verificare la presenza o meno di overflow (il segno del risultato non è concorde con quello dei due addendi).

28) In quali casi si presenta overflow nel metodo della somma CA2?
L'overflow si presenta se:
(+A) + (+B) = -C
oppure
(-A) + (-B) = +C

Una modalità alternativa per verificare la presenza di overflow consiste nel guardare i riporti nelle ultime due posizioni più significative: se sono diversi c'è overflow.

Esempi di CA2

```
3 + (-8)

 (+3) 0000 0011
+(-8) 1111 1000
===============
 (-5) 1111 1011

-2 + (-5)

 (-2)  1111 1110
+(-5)  1111 1011
===============
 (-7)1 1111 1001: scarto carry
```

29) Come viene effettuata la sottrazione dei due valori in CA2?

La sottrazione tra due numeri in CA2 viene trasformata in somma applicando la regola

```
A - B = A + (-B)
```

ovvero:

```
A - B = A + CA2(B)
```

30) In quali casi si presenta overflow nel metodo della sottrazione CA2?

L'overflow si verifica se gli operandi hanno segno concorde e il segno del risultato è discorde con essi.

Esempio di sottrazione

```
 (+8) 0000 1000                    0000 1000
-(+5) 0000 0101 -> Complementa ->  +1111 1011
=====                              ===========
 (+3)                             1 0000 0011 : scarto il carry
```

31) Come è possibile effettuare operazioni tra valori binari con bit diversi?

Nell'ipotesi di avere un valore X in CA2 su n bit (segno incluso) e di volerne ricavare la rappresentazione, sempre in CA2, su m bit (m > n), si attua l'estensione del segno (si replica l'MSB negli (m-n) bit più a sinistra).

Esempio 1: rappresentazione di +18

```
+18 =           0001 0010
+18 = 0000 0000 0001 0010
```

Esempio 2: rappresentazione di -18

```
-18 =      101110
-18 = 1111 101110
```

32) Descrivere l'operazione di shift

L'operazione di shift verso destra (right shift) o verso sinistra (left shift) la posizione delle cifre di un numero, espresso in una base qualsiasi, inserendo uno zero nelle posizioni lasciate libere.

→ Left: equivale a moltiplicare il numero per la base;

→ Right: equivale a dividere il numero per la base.

33) Descrivere la rappresentazione eccesso 2^{n-1}

→ La rappresentazione eccesso 2^{n-1} descrive un numero X come segue:

$X + 2^{n-1}$

→ Avendo a disposizione n bit si rappresenta l'eccesso 2^{n-1} e l'intervallo di rappresentazione è $[-2^{n-1}, +2^{n-1} - 1]$.

→ I numeri in eccesso 2^{n-1} si ottengono da quelli in CA2 complementando il bit più significativo.

34) Descrivere la rappresentazione eccesso 128

→ La rappresentazione eccesso 128 descrive un numero X come segue:

$X + 128$

Esempio:

```
X = 5        5 + 128 = 133    1000 0101(2)
X = -3      -3 + 128 = 125    0111 1101(2)
```

35) Come possono essere rappresentati i numeri reali?

I numeri reali R possono essere rappresentati come segue:

→ virgola fissa;

→ virgola mobile (floating point).

36) Descrivere la rappresentazione della virgola fissa

→ Il metodo di rappresentazione "virgola fissa" è un metodo di rappresentazione binaria dei numeri reali R. Avendo a disposizione N bit, essi vengono usati in questo modo:

→ 1 bit per il segno del numero da rappresentare;

→ I < (N-1) bit per rappresentare la parte intera del numero;

→ D = N – (I+1) bit per rappresentare la parte decimale del numero.

+/-	I-1	I-2	...	0	-1	-2	...	-D
Segno	Parte Intera				Parte Frazionaria			

Le caratteristiche di questo metodo sono:

→ nel sistema di numerazione in virgola fissa è quello in cui la posizione della virgola decimale è implicita e la posizione della virgola decimale è uguale in tutti i numeri;

→ l'intervallo di numeri interi rappresentabili è $[-(2^{I-1}), 2^{I-1}]$;

→ l'intervallo rappresentabile dalla parte decimale è $[0, 1/2^{D}]$

Esempio 1:

Si vuole convertire il numero 5.125 base 10 in base 2 utilizzando il metodo della virgola fissa.

Il procedimento è il seguente:

1 – si converte la parte intera 5 e la si riporta in base 2

```
5(10) = 101(2)
```

2 – la parte decimale viene scomposta per moltiplicazioni successive:

```
0.125 x 2 = 0.25 -> 0 (riporto 0.25)
 0.25 x 2 = 0.5  -> 0 (riporto 0.5)
  0.5 x 2 = 1    -> 1 (stop)

5.125(10) = 101.001(2)
```

Esempio 2:

```
101.01(2) = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^-1 + 1 * 2^-2 = 4 + 0 + 1 + 0 + 0.25 = 5.25
```

37) Quali sono i svantaggi del metodo di rappresentazione virgola fissa?

Gli svantaggi del metodo di rappresentazione virgola fissa sono:

→ rigidità della posizione assegnata alla virgola: sono fissi i bit assegnati per codificare la parte intera e la parte frazionaria;

→ impatto sulla precisione nel codificare i numeri: maggiore è il numero di bit per codificare la parte intera, più bassa sarà la precisione nel codificare i numeri piccoli.

38) Descrivere la rappresentazione della virgola mobile

Il metodo di rappresentazione "virgola mobile" è un metodo di rappresentazione binaria in cui:

→ viene utilizzato un bit per rappresentare il segno s;

→ vengono utilizzati altri bit per rappresentare la mantissa m;

→ altri bit vengono utilizzati per codificare l'esponente e.

s	e	e	...	e	m	m	...	m
Segno	esponente				mantissa			

→ La posizione della virgola è variabile per avere una rappresentazione in notazione scientifica in cui è situata un'unica cifra a sinistra della virgola, una parte frazionaria e un esponente al quale si deve elevare la base del numero.
Esempi:

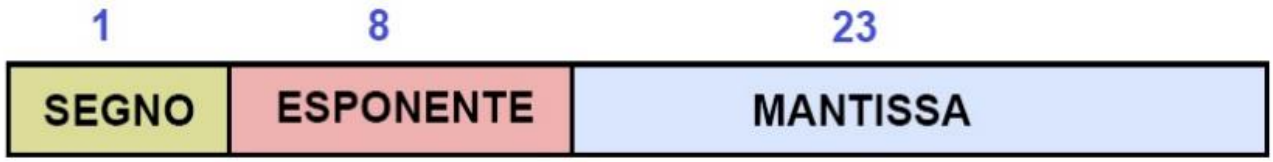
```
546.768(10)   -> 5.46768(10)*10^2
1011.0110(2) -> 1,0110110(2)*2^3
```

→ Questa rappresentazione estende l'intervallo di numeri rappresentati a parità di cifre, rispetto alla notazione in virgola fissa.
I numeri reali R rappresentati da una coppia di numeri composti da:
- mantissa (M);
- esponente (E);
- segno (S).

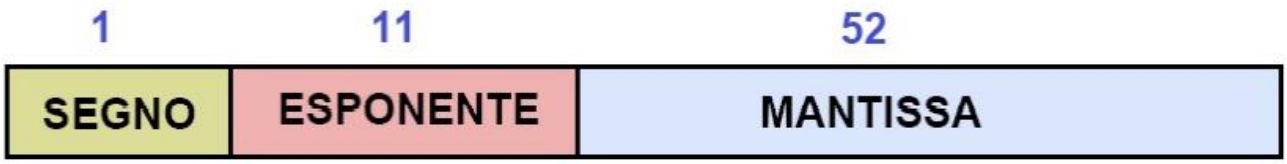
La rappresentazione in virgola mobile è la seguente:

$$1,xx\ldots xx_2 \cdot 2^{yy\ldots yy_2}$$

dove le x rappresentano la parte frazionaria e le y l'esponente a cui elevare la base 2.
Precisione su 32 bit:



Precisione su 64 bit:



39) Come viene convertito un numero reale da base 10 in base 2, in virgola mobile?
Si suppone di convertire 7.5_{10} in base 2:
→ si converte la parte intera $7_{10} = 111_2$;
→ si considera la parte frazionaria $0.5_{10} = 1.0_2$;
→ si considera il numero binario ottenuto convertendo parte intera e parte

frazionaria

```
(111.1)(2) = 1*2^2 + 1*2^1 + 1*2^0 + 1*2^-1
```

Osservazione: l'esponente può assumere valori negativi (quindi numeri in virgola mobile possono essere rappresentati in MS e CA2).
Una rappresentazione comune rappresenta:
→ l'esponente in eccesso 127 (semplicità dei calcoli);
→ la mantissa nell'intervallo [1,2).
Per calcolare il valore di un numero in virgola mobile:
$X = (-1)^S * M * 2^{E-127}$

40) Cos'è lo standard IEEE 754?

→ Lo standard IEEE 754 definisce un metodo per la rappresentazione dei numeri in virgola mobile, o floating point. Il numero reale R viene dapprima rappresentato in binario, convertendo opportunamente la parte intera e la parte frazionaria.
→ Il range è dato dal numero di bit dell'esponente.
→ Il numero reale float più piccolo in valore assoluto si ha quando la mantissa m è composta da tutti 0 e l'esponente e assume il valore minimo, ossia 1:
$N_{min} = 1.(0000000000\ldots0)_2 \cdot 2^{1-127} = 1.0 \cdot 2^{-126}$
→ Il numero reale float più grande in valore assoluto si ha quando la mantissa m è composta da tutti 1 e l'esponente e assume il valore massimo, ossia 254:
$N_{max} = 1.(1111111111\ldots1)_2 \cdot 2^{254-127} = 1.(1111111111\ldots1)_2 \cdot 2^{127} \approx 3.4 \times 10^{38}$

41) Errore assoluto ed errore relativo

→ Rappresentando un numero reale n in virgola mobile si commette un errore di approssimazione.
→ Dato un numero razionale n' con un numero limitato di cifre significative, l'errore assoluto viene rappresentato da $eA = n - n'$, mentre l'errore relativo viene rappresentato da $eR = (n - n')/n$.
→ L'ordine di grandezza dell'errore assoluto dipende dal numero di cifre significative e dall'ordine di grandezza del numero;
→ L'ordine di grandezza dell'errore relativo dipende solo dal numero di cifre significative.

42) Come possono essere rappresentati i caratteri?

I caratteri possono essere rappresentati in:

→ ASCII standard: 1 carattere viene rappresentato con 7 bit per un totale di 128 simboli rappresentabili (quali cifre, lettere maiuscole e lettere minuscole);
→ ASCII estesa: 1 carattere è rappresentato con 8 bit rappresentabili fino a 256 simboli (sono usati i caratteri accentati come caratteri in più);
→ UNICODE: 1 carattere è rappresentato con un numero maggiore di bit (tra 8 e 32 bit).

43) Descrivere le caratteristiche dell'ASCII Standard

L'ASCII standard contiene:
→ 26 + 26 lettere (maiuscole + minuscole);
→ 10 cifre decimali (da 0 a 9);
→ segni di interpunzione;
→ caratteri di controllo.
Inoltre:
→ le cifre sono ordinate per valore;
→ le lettere maiuscole sono ordinate alfabeticamente;
→ le lettere minuscole sono ordinate alfabeticamente (e sono a distanza fissa dalle maiuscole).

```
-> da 0 a 31: caratteri di controllo per le periferiche;
-> da 32 a 47: vari caratteri;
-> da 48 a 57: cifre decimali;
-> da 58 a 64: vari caratteri;
-> da 65 a 90: lettere maiuscole alfabeto;
-> da 91 a 96: ari caratteri;
-> da 97 a 122 lettere minuscole dell'alfabeto;
-> da 123 a 127: vari caratteri.
```

44) Descrivere le caratteristiche dell'ASCII Esteso

L'ASCII esteso è una codifica a 8 bit, in grado di rappresentare molti altri caratteri oltre ai tradizionali 128 dell'ASCII a 7 bit. L'ultimo bit alla descrizione del carattere viene definito parity bit, dedicato al controllo di parità (parity check).

45) Descrivere le caratteristiche della codifica UNICODE

Le caratteristiche della codifica UNICODE sono:
→ evoluzione dello standard ASCII;
→ standard per la rappresentazione di testo;
→ codifica tutti i caratteri utilizzati nelle principali lingue del mondo;
→ indipendenza dalla lingua, dal sistema operativo e dal programma utilizzato;
→ Inizialmente rappresentato come una codifica su 16 bt, ma poi esteso a 24 e

32 bit;
→ continua evoluzione e continua ad aggiungere sempre più caratteri.

46) Che cosa si intende per code point?

Il code point è un codice numerico di 8 cifre esadecimali che rappresenta un carattere UNICODE.

47) Quale problema presenta la codifica UNICODE?

→ UNICODE può codificare 4294967296 caratteri distinti.
→ Essendo che ogni carattere occupa 32 bit (contro gli 8 delle altre codifiche); i documenti richiedono quindi 4 volte lo spazio.
→ Per ovviare a questo problema, e garantire maggiore compatibilità con il sistema operativo e applicazione che non sono in grado di gestire 32 bit per carattere, UNICODE definisce vari formati di codifica più compatti.

48) Descrivere la codifica UTF-8?

→ La codifica UTF-8 (Unicode Transformation Format) è una codifca a lunghezza variabile fra una sequenza di valori a 8 bit e una sequenza di valori a 8 bit e una sequenza di caratteri UNICODE.
→ I primi 128 caratteri di UNICODE (0-7F), equivalenti ai caratteri ASCII, sono codificati con il loro codice "naturale".
→ Tutti gli altri caratteri sono codificati con due, tre o quattro valori a 8 bit (byte).
→ Questa codifica viene utilizzata nei vari linguaggi di programmazione, come Java per codificare le stringhe, nei file system Macintosh, DVD, in alcuni su UNIX per i nomi dei file, negli standard relativi al Web e alla e-mail e nei programmi che trattano testi ASCII.

Capitolo 2 – Circuiti Logici

49) Definizione di circuiti logici e integrati e classificazione circuiti integrati

→ I circuiti logici sono implementati come circuiti integrati che vengono realizzati nei chip di silicio (piastrine).
→ Porte (gate) e fili conduttori depositati nei chip di silicio, sono raggrupati in package e direttamente collegati con insieme di piedini, chiamati pin.
→ I circuiti integrati si suddividono in categorie stabilite in base all'integrazione.
→ I circuiti integrati si distinguono in quattro categorie:
a) SSI (Small Scale Integrated): circuiti integrati costituiti da 1 a 10 porte e sono considerati come piccoli circuiti con poche porte direttamente collegate con pin esterni.
b) MSI (Medium Scale Integrated): circuiti integrati costituiti da 10 a 100 porte e sono circuiti utilizzati per progettare computer.
c) LSI (Large Scale Integrated): circuiti integrati costituiti da 100 a 100000 porte.
d) VLSI (Very Large Scale Integrated): circuiti integrati costituiti da più di 100000 porte e possono contenere una CPU intera o più (esempio microprocessori).

50) Differenza tra circuiti combinatori e sequenziali

I circuiti combinatori sono circuiti in cui lo stato di uscita dipende dalla funzione logica applicata allo stato istantaneo dei valori d'ingresso, mentre i circuiti sequenziali sono circuiti in cui lo stato di uscita non dipende solamente dalla funzione logica applicata ai valori di ingresso, ma anche dai valori pregressi collocati in memoria.

51) Definizione di porta logica

La porta logica è un componente elettronico in grado di effettuare operazioni logiche primitive oltre a quelle direttamente derivate. La porta logica è in grado di eseguire operazioni logiche definite nell'algebra booleana.
Le porte logiche si distinguono in due categorie:
→ porte logiche primitive: svolgono le operazioni di AND, OR e NOT;
→ porte logiche derivate: svolgono le operazioni di NAND, NOR e XOR.

52) Descrivere la porta logica AND

La porta logica AND svolge l'operazione logica di AND, chiamata anche prodotto logico.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

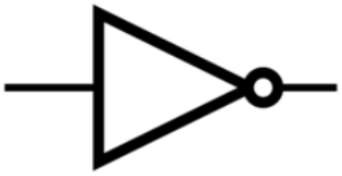
53) Descrivere la porta logica OR

La porta logica OR svolge l'operazione logica di OR, chiamata anche somma logica.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

54) Descrivere la porta logica NOT

La porta logica NOT svolge l'operazione logica di NOT, chiamata anche complemento.



A	¬A
0	1
1	0

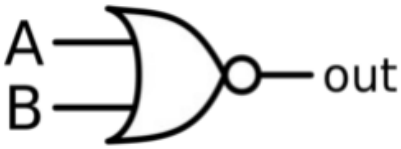
55) Descrivere la porta logica NAND

La porta logica NAND svolge l'operazione logica di NOT sull'operazione logica di AND tra i due valori di ingresso.

A	B	¬(AB)
0	0	1
0	1	1
1	0	1
1	1	0

56) Descrivere la porta logica NOR

La porta logica NOR svolge l'operazione logica di NOT sull'operazione logica di OR tra i due valori di ingresso.



A	B	¬(A+B)
0	0	1
0	1	0
1	0	0
1	1	0

57) Descrivere la porta logica XOR

La porta logica XOR svolge l'operazione logica di XOR, che rappresenta l'operazione di somma logica esclusiva (OR esclusivo).

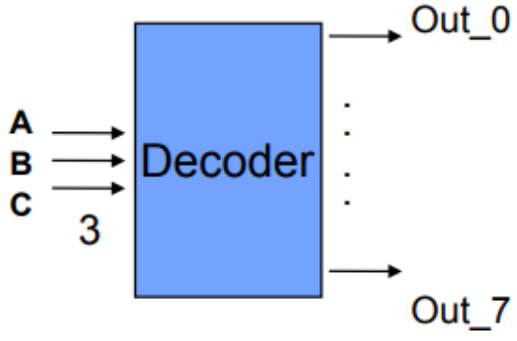
A	B	AxB
0	0	0
0	1	1
1	0	1
1	1	0

58) Definizione di Decoder

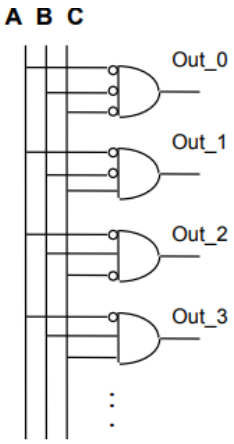
→ Il decoder è un componente elettronico caratterizzato dall'avere n ingressi e 2^n uscite.

→ Il suo scopo è quello di impostare allo stato alto l'uscita corrispondente alla conversione in base 10 della codifica binaria a n bit ricevuta in input.

→ Nel decoder n input sono interpretati come un numero unsigned e se questo numero rappresenta il numero i, allora solo il bit in output di indice verrà posto ad 1 e gli altri verranno posti a 0.



A	B	C	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

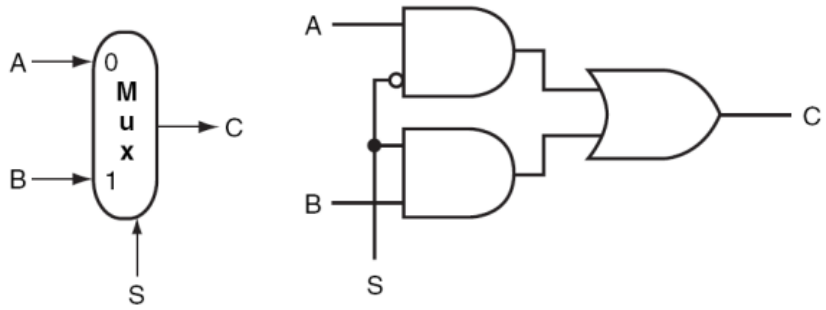


59) Definizione di Multiplexor

→ Un multiplexor è un componente elettronico caratterizzato da 2^n entrate

principali, n entrate di controllo e 1 uscita.
Esempio:

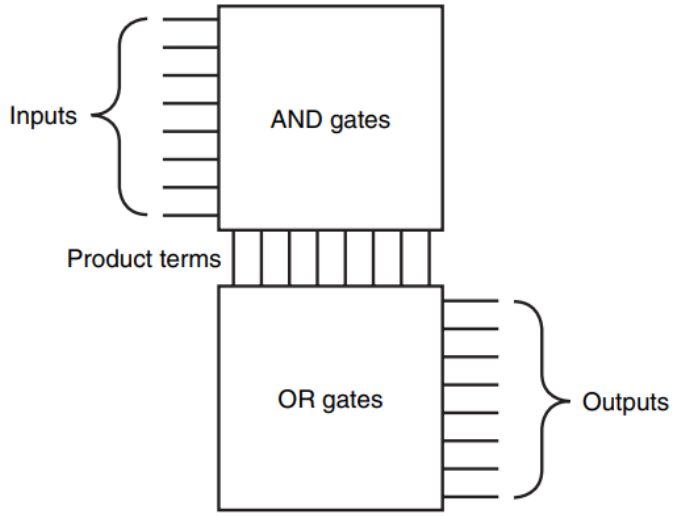
$$C = (A \cdot \bar{S}) + (B \cdot S)$$



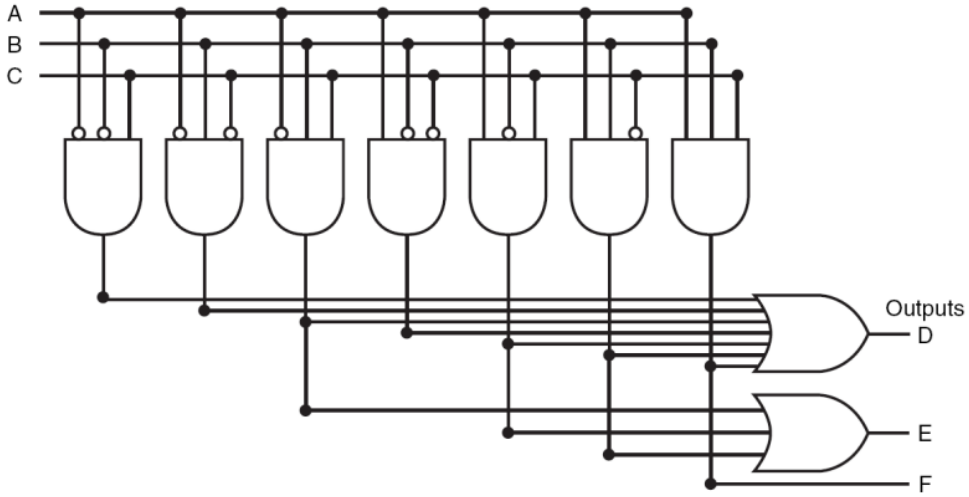
→ Se un multiplexor riceve n segnali in input, esso necessiterà di $\log_2 n$ selettori e consisterà in un decoder che genererà n segnali, un array di n porte logiche AND e un'unica porte logica OR.

$$n = 2^m \quad \longleftrightarrow \quad m = lg_2\, n$$

60) Definizione di PLA (Programmable Logic Array)
PLA (Programmable Logic Array) è un dispositivo logico programmabile usato per implementare circuiti logici combinatori.
Esso è costituito da:
→ insieme di input;
→ insieme di input complementati (mediante inverter) per poter gestire più uscite;
→ logica a due stage in cui il primo stage è costituito da un array di porte logiche AND (prodotto) e un secondo stage è costituito da un array di porte logiche OR (somma).



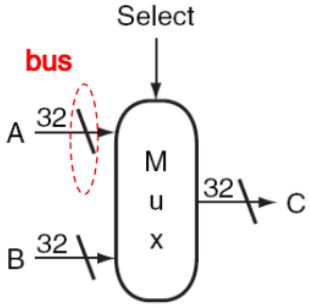
Circuito PLA



Caratteristiche

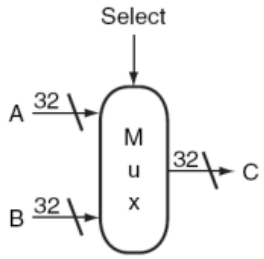
→ la maggior parte delle operazioni vengono svolte su 32 bit, mettendo in luce la necessità di creare array di elementi logici;

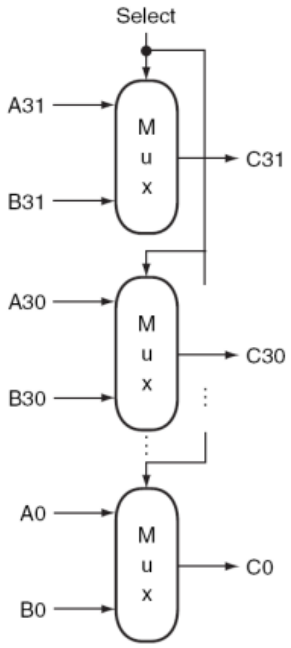
→ un bus è un canale di comunicazione tra vari componenti, considerato come una collezione di linee di input che verranno trattate come un singolo segnale



→ un multiplexor con un bus a 32 bit corrisponde ad un array di 32 multiplexor

ad 1 bit.





61) Definizione di ALU (Aritmethic Logic Unit)

L'ALU (Aritmethic Logic Unit) è una parte della CPU che svolge le operazioni aritmetico – logiche, considerato come un insieme di circuiti combinatori che implementa:

→ operazioni aritmetiche: somma e sottrazione;

→ operazioni logiche: AND e OR

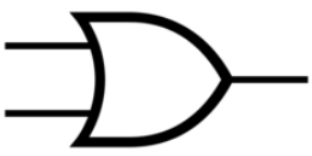
62) Quali sono i componenti dell'ALU (Aritmethic Logic Unit)

I componenti dell'ALU sono:

1) AND gate (c = a*b)

a	b	c = a+b
0	0	0
0	1	0
1	0	0
1	1	1

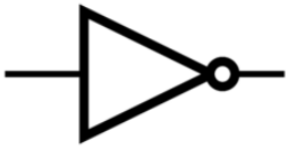
2) OR gate (c = a + b)



A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

3) Inverter (c = ¬a)

A	¬A

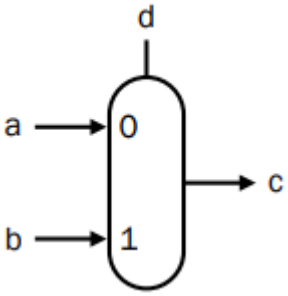


0	1
1	0

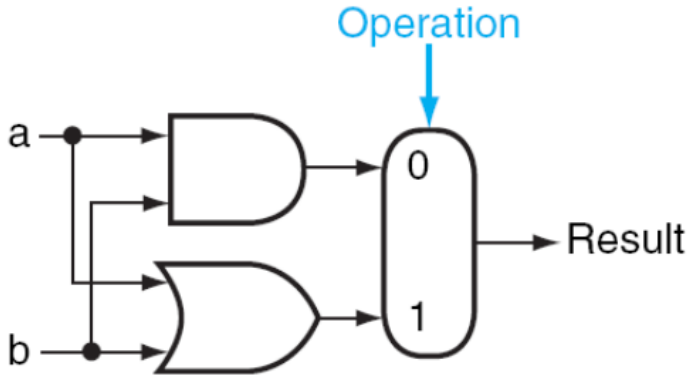
4) Multiplexer

```
if (d==0)
      c = a;
else
      c = b;
```

d	c
0	a
1	b



Esempio: ALU su un 1 bit che implementa AND e OR



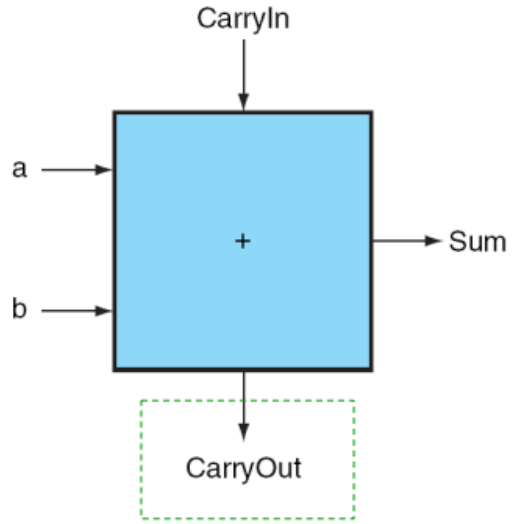
63) Come vengono effettuate le operazioni aritmetiche nell'ALU?

Considero la seguente somma:

```
0000000000000000 0001 1111 0011 0100

0000000000000000 0000 1011 1100 1011 +
0000000000000000 0000 0110 1001 1010 =
====================================
0000000000000000 0001 0010 0110 0101
```

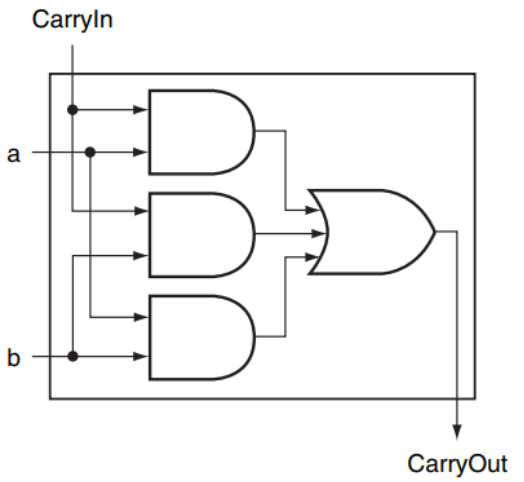
Ovvero



a	b	CarryIn	CarryOut	Somma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	**1**	**1**	**1**	**0**
1	0	0	0	1
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

La formula ricavata per calcolare il CarryOut è la seguente:

```
CarryOut = (b*CarryIn) + (a*CarryIn) + (a*b) + (a*b*CarryIn)
CarryOut = (b*CarryIn) + (a*CarryIn) + (a*b)
```

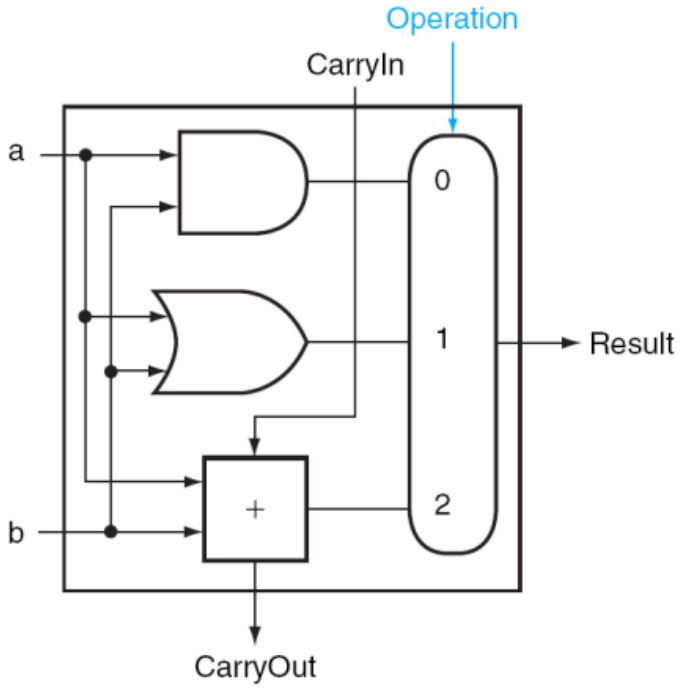


$$CarryOut = (b \cdot CarryIn) + (a \cdot CarryIn) + (a \cdot b)$$

a	b	CarryIn	CarryOut	Somma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	**1**	**1**	**1**	**0**
1	0	0	0	1
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

$$\text{Sum} = (a \cdot \bar{b} \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot \bar{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

64) Rappresentare l'ALU che esegue somma, AND, OR

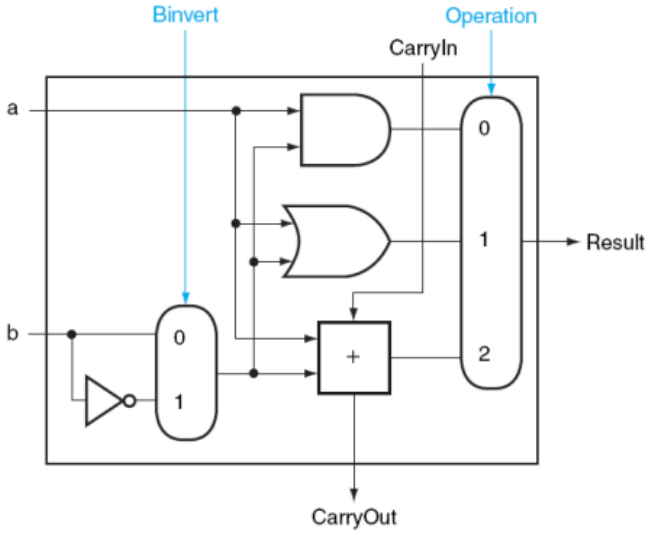


65) Cosa succede se settiamo il CarryIn a 1?

→ Si sommano a + b + 1.

→ Se si nega b, è possibile ottenere una sottrazione (in CA2)

$$a - b = a + (\bar{b} + 1)$$



66) Come si implementa l'operazione di NOR nell'ALU?

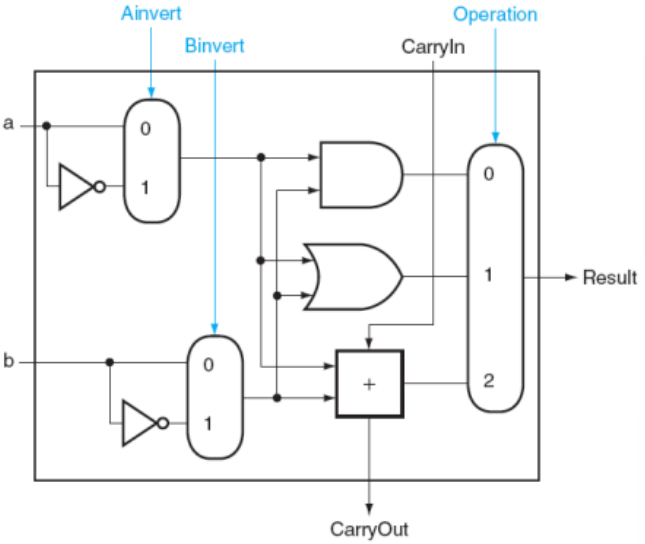
Attraverso ¬(a+b).

67) Leggi di De Morgan

Le leggi di De Morgan affermano che:

$$\overline{(a + b)} = \bar{a} \cdot \bar{b}$$

$$\overline{(a \cdot b)} = \bar{a} + \bar{b}$$

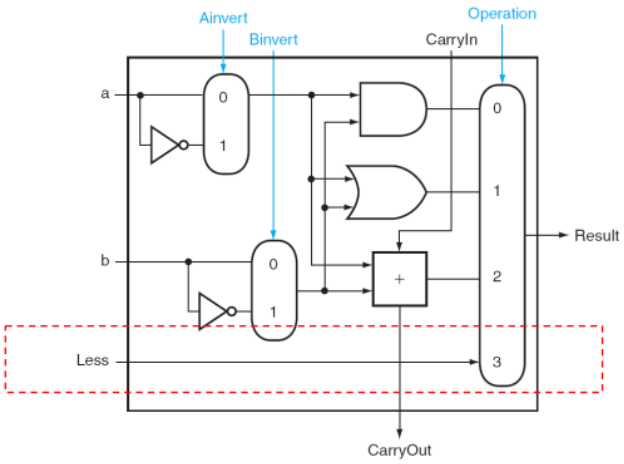


68) Descrivere l'operazione di SLT

→ L'operazione SLT (Set On Less Then) è l'operazione effettuata dall'ALU che setta a 1 se a < b, 0 altrimenti e per eseguire questa istruzione di devono poter azzerare tutti i bit dal bit – 1 al bit – 31 ed assegnare al bit – 0 il valore risultato.

→ Per poter realizzare il confronto si effettua la sottrazione tra a e b e:

a) se a – b è minore di 0, allora a < b (per l'istruzione SLT) e il risultato sarà 00…01;
b) se a – b è maggiore di 0, allora a > b (per l'istruzione SLT) e il risultato sarà 00…00.
→ ALU su 1 bit – si aggiunge Less.



69) Quali sono i casi di overflow?

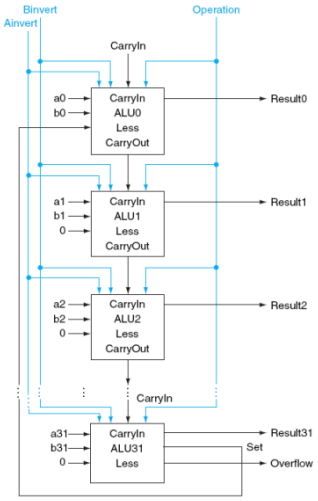
Esistono due casi di overflow:
→ (A – B) > 0 e bit – 31 di (A – B) = 1;
→ (A – B) < 0 e bit – 31 di (A – B) = 0

bit-31(A) = 0, bit-31(B) = 1	bit-31(A - B) = 1
oppure	
bit-31(A) = 1, bit-31(B) = 0	bit-31(A - B) = 0

Il controllo dell'overflow viene rilevato nella maniera seguente:

$$overflow = \bar{a} \cdot b \cdot Result + a \cdot \bar{b} \cdot \overline{Result}$$

→ Le sottrazioni vengono effettuate settando Binvert e CarryIn = 1.
→ Le addizioni e operazioni logiche vengono effettuate settando Binvert e CarryIn = 0.



70) Descrivere l'operazione di BEQ

→ L'operazione BEQ (Branch On Equal) è l'operazione effettuata dall'ALU che verifica l'uguaglianza di a e b attraverso la seguente sottrazione:

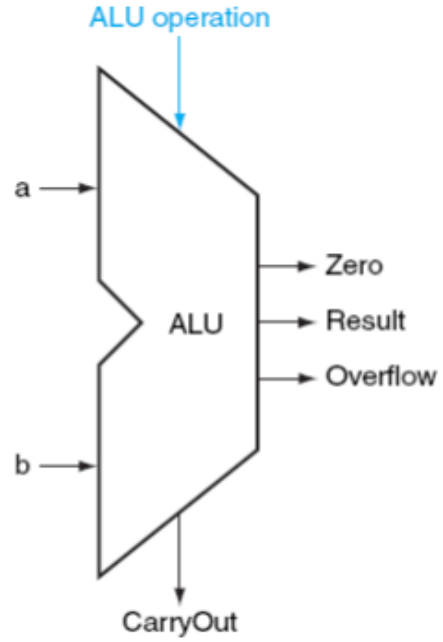
a – b = 0;

a = b

→ L'ALU nega l'OR di tutte le uscite

$$Zero = \overline{(Result31 + Result30 + \cdots + Result0)}$$

71) Rappresentazione ALU

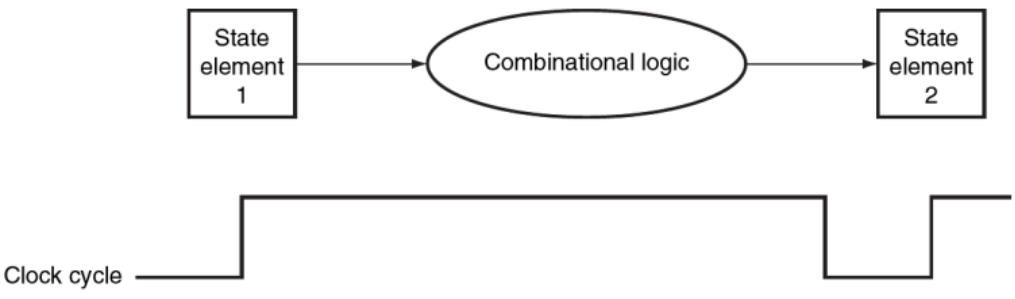


72) Definizione di Clock e quali sono le sue caratteristiche?

→ Il clock è un segnale periodico, generalmente un'onda quadra, utilizzato per sincronizzare il funzionamento dei dispositivi elettronici digitali.

→ Esso è caratterizzato da un periodo T (oppure ciclo) di clock e dalla frequenza F (come 1/T) e misurata in Hertz.

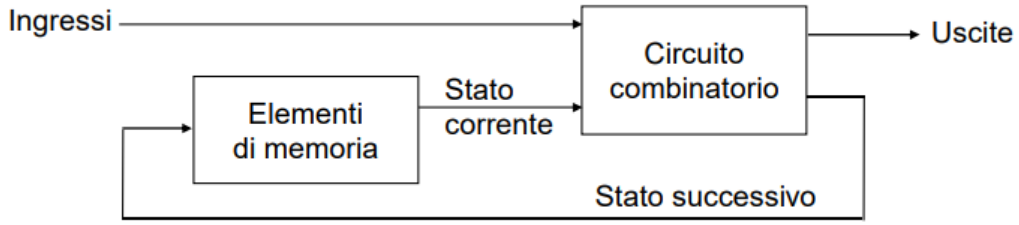
→ Data la sua grandezza accettabile è in grado di assicurare la stabilità degli output fi un circuito e determina il ritmo dei calcoli e delle operazioni i memorizzazione.



→ La funzione calcolata dal circuito sequenziale in un certo momento dipende dalla sequenza temporale dei valori in input. La sequenza temporale determina il valore memorizzato nello stato.

73) Famiglie di circuiti sequenziali e caratteristiche

Esistono due famiglie di circuiti digitali sequenziali:
→ asincroni: in cui non fanno uso di clock, ad esempio SR Latch;
→ sincroni: che necessitano di clock, ad esempio Flip – Flop.
I circuiti sequenziali sono formati da:
→ elementi di memoria, che memorizzano l'informazione;
→ reti combinatorie, che elaborano informazione.
Un circuito sequenziale ha, in ogni dato istante, uno stato determinato dai bit memorizzati.



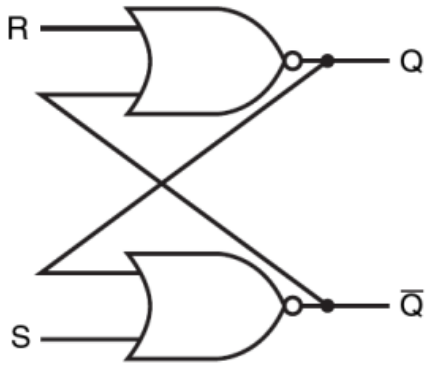
Per realizzare circuiti sequenziali è necessario un elemento di memoria per memorizzare lo stato.

74) E' possibile organizzare le porte logiche in modo da realizzare un elemento di memoria?

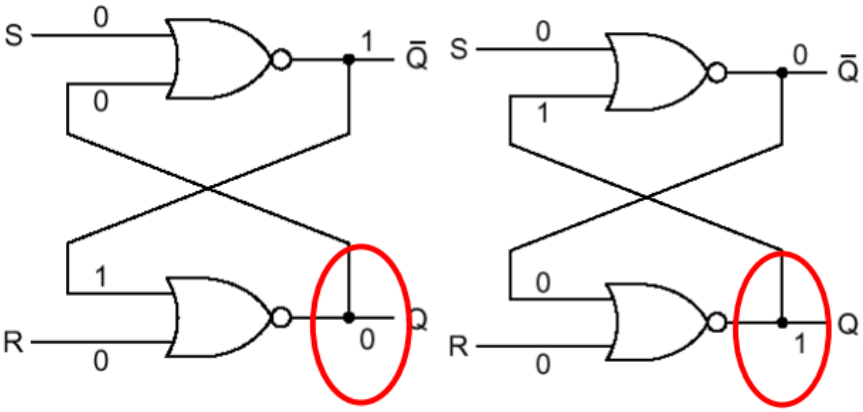
Si, un elemento in grado di memorizzare un singolo bit è il latch.

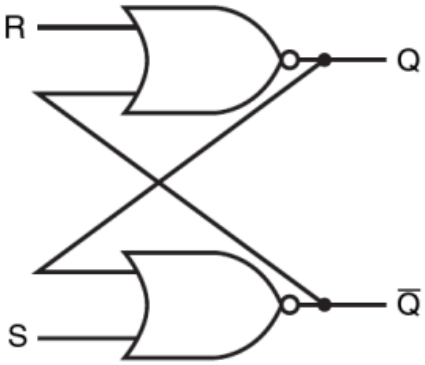
75) S – R Latch: descrizione

S – R Latch (S = Set) e (R = reset) è un circuito, composto da 2 porte NOR concatenate.



A	B	¬(A+B)
0	0	1
0	1	0
1	0	0
1	1	0





A	B	Old Q	Q	¬Q
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

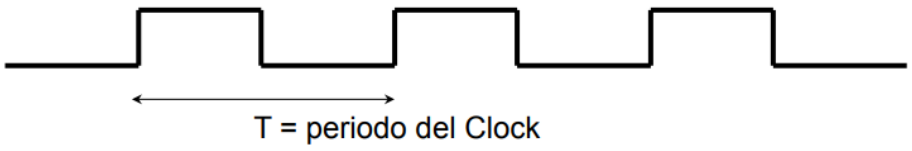
Caratteristiche

→ La combinazione (S,R) = (0,0) viene detta combinazione a riposo, perché mantiene il valore memorizzato in precedenza;

→ La configurazione di S = 1 e R = 1, viola la proprietà di complementarietà di Q e ¬Q, può portare ad una configurazione instabile.

→ (S,R) sono di solito calcolati da un circuito combiantorio e l'output del circuito diventa stabile dopo un certo intervallo di tempo che dipende dal numero di porte attraversate.

→ Per evitare che durante questo intervallo, gli output intermedi del circuito vengano memorizzati è necessario utilizzare ciò che si chiama clock, il quale determina il ritmo dei calcoli e delle relative operazioni di memorizzazione.

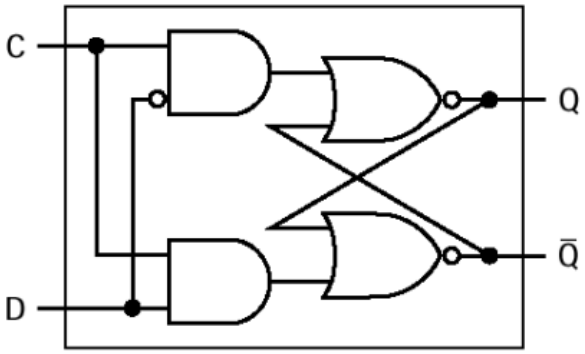


76) D Latch: descrizione

→ Il D Latch è un circuito latch sincronizzato con il clock, che garatnisce che il latch cambi stato solo in certi momenti.

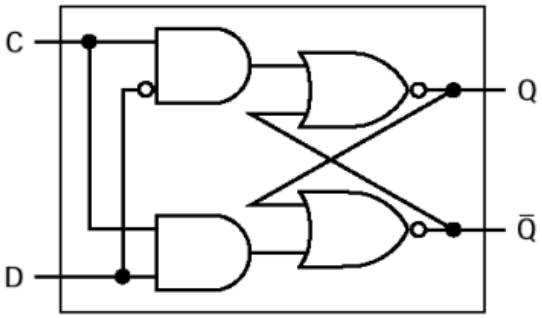
→ D = 1 corrisponde al setting (S = 1 e R = 0)

→ D = 0 corrisponde al resetting (S = 0 e R = 1)



→ Il clock è deasserted quando non viene memorizzato nessun valore (S = 0 e R = 0).

→ Il clock è asserted quando viene memorizzato un valore (in funzione del valore di D).



Il D – Latch è caratterizzato dal sequente comportamento:

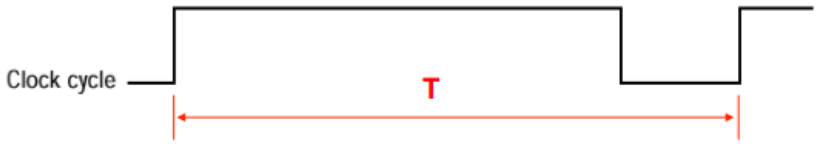
→ durante l'intervallo alto di clock il valore del segnale di ingresso D viene memorizzato nel latch;

→ il valore di D si propaga quasi immediatamente all'uscita Q;

→ anche le eventuali variazioni di D si propagano quasi immediatamente, con il risultato che Q può variare più volte durante l'intervallo alto del clock;

→ solo quando il clock torna a zero Q si stabilizza;

→ durante l'intervallo basso del clock il latch non memorizza. Questo significa che il periodo di clock T deve essere scelto abbastanza lungo affinchè l'output del circuito si stabilizzi.



77) Metodologie di timing

Esistono due metodologie di timing:

→ level triggered: avviene sul livello alto del clock;

→ edge triggered: avviene sul fronte di salita o discesa del clock e:

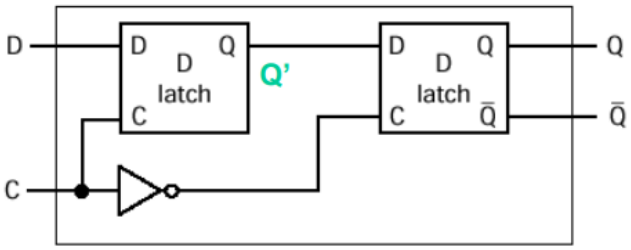
1 – la memorizzazione avviene istantaneamente;

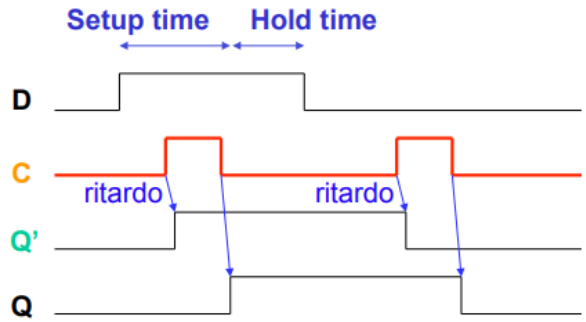
2 – l'eventuale segnale di ritorno "sporco" non fa in tempo ad arrivare a causa dell'istantaneità della memorizzazione.

78) D Flip – Flop

→ Il D Flip – flop è un circuito sequenziale usato come dispositivo di memoria elementare, ed è usato come input e output durante lo stesso ciclo di clock.

→ Questo è realizzato ponendo in serie 2 D – Latch: il primo viene detto master e il secondo slave.





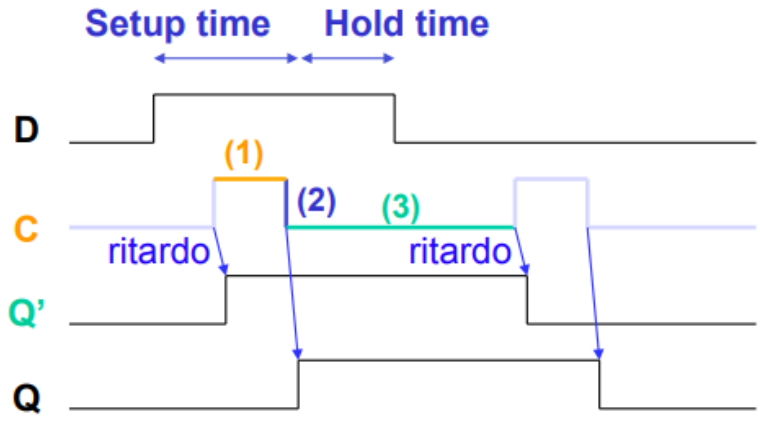
79) Come funziona il D Flip – Flop

Il Flip Flop ha il seguente funzionamento:

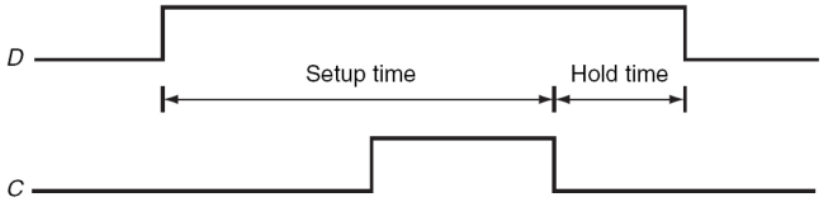
1) Il primo latch è aperto e pronto per memorizzare D. Il valore memorizzato Q' fluisce fuori, ma il secondo latch è chiuso. Nel circuito combinatorio a valle entra ancora il vecchio valore di Q.

2) Il segnale del clock scende, e in questo istante il secondo latch viene aperto per memorizzare il valore di Q';

3) Il secondo latch è aperto, memorizza D (Q'), e fa pulire il nuovo valore Q nel circuito a valle. Il primo latch è invece chiuso, e non memorizza niente.



→ Il segnale D deve essere attivo per un periodo abbastanza lungo: setup time (prima del clock edge) + hold time (dopo il clock edge).



80) Cosa si intende per registro e per MIPS

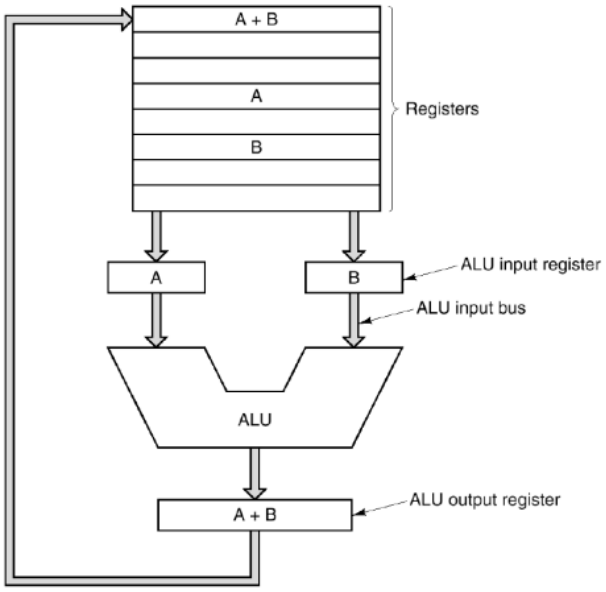
→ Il registro è una cella di memoria, in cui è possibile leggere e scrivere dati. Esso conserva dati e risultati delle operazioni ed è costituito da n flip – flop.

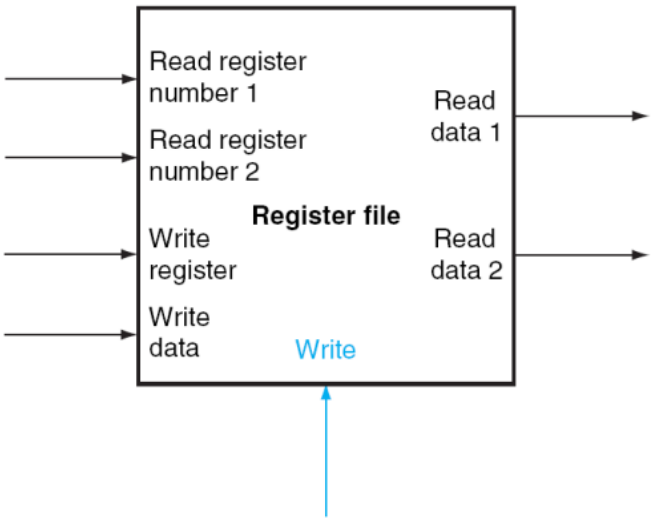
→ Il MIPS rappresenta un microprocessore con architettura RISC prodotto dalla società statunitense MIPS Technologies.

→ Nel MIPS ogni registro è di 1 word = 4 byte = 32 bit.

→ I registri sono organizzati da ciò che si chiamano Register File, che permette la lettura di 2 registri e la scrittura di 1 registro.

→ Il Register File del MIPS ha 32 registri (32 * 32 = 1024 filp – flop).





81) Quali sono le istruzioni del Register File

Le istruzioni del Register File sono:

```
Read Reg1#  (5 bit)     //numero del 1 registro da leggere
Read Reg2#  (5 bit)     //numero del 2 registro da leggere
Read data 1 (32 bit)    //valore del 1 registro, letto sulla base di Read Reg1#
Read data 2 (32 bit)    //valore del 2 registro, letto sulla base di Read Reg2#
Write Reg#  (5 bit)     //numero del registro da scrivere
Write data  (32 bit)    //valore da scrivere nel registro Write Reg#
```

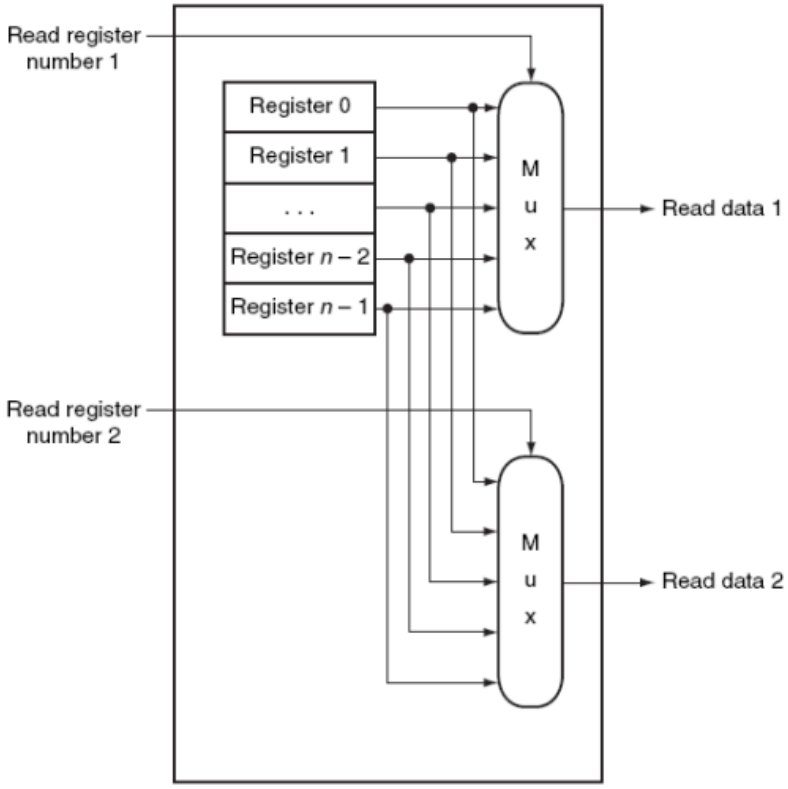
82) Descrivere il processo di lettura dal Register File

Il processo di lettura del Register File è il seguente:

→ utilizza 2 segnali che indicano i registri da leggere (Read Reg1#, Read Reg2#);

→ utilizza 2 multiplexer: ognuno con 32 ingressi e un segnale di controllo;

→ il Register File fornisce sempre in output una coppia di registri.



83) Descrivere il processo di scrittura nel Register File

Il processo di scrittura nel Register File è il seguente:

→ utilizza 3 segnali:

1 – il registro da scrivere (Register Number);

2 – il valore da scrivere (Register Data);

3 – il segnale di controllo (Write);

→ utilizza un decoder che decodifica il numero del registro da scrivere (Write Register);

→ il segnale Write (già in AND con il clock) è in AND con l'output del decoder;

→ se Write non è affermato nessun valore sarà scritto nel registro;

84) Distinzione tipi di memoria e gerarchie

Esistono altri tipi di memorie distinte in base a diversi parametri:

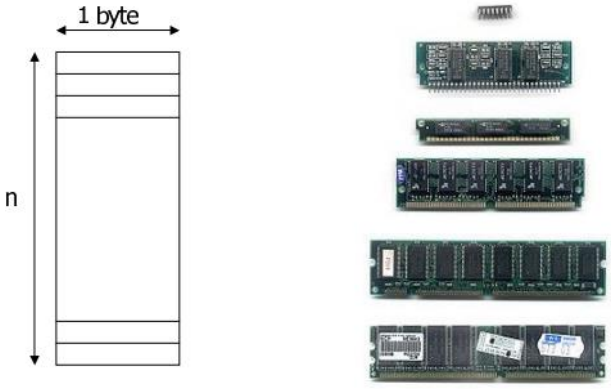
1) dimensione: quantità di dati memorizzabili;

2) velocità: intervallo di tempo tra la richiesta del dato e il momento in cui è

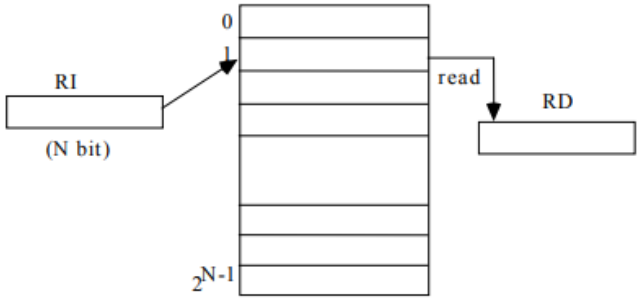
disponibile;
3) consumo: potenza assorbita;
4) costo: costo per bit.
La gerarchia di memoria è la seguente:
→ memorie piccole, più veloci (e costose) sono poste ai livelli alti;
→ memorie ampie, più lente (e meno costose) sono poste ai livelli più bassi.

85) RAM: lettura e scrittura

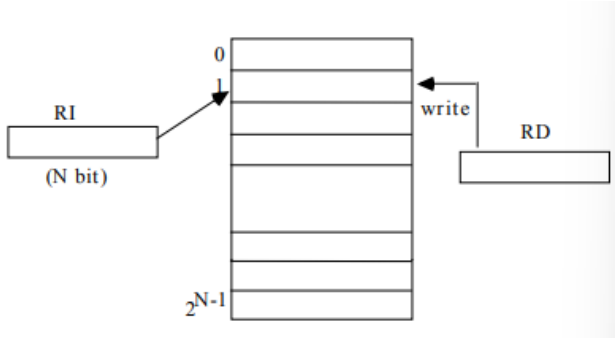
→ La RAM (Random Access Memory) è il componente della memoria principale che memorizza dati e programmi nel momento in cui il sistema è in attività (programma in esecuzione).
→ Insieme di celle (1 byte) ognuna individuata da un indirizzo;
→ In essa viene effettuato quello che si chiama indirizzamento, ovvero l'attività con cui l'elaboratore seleziona una particolare cella di memoria.
→ Con n bit si indirizzano 2^n celle di memoria



→ La lettura (Read) consiste nel copiare il contenuto della cella di memoria indirizzata dal Registro Indirizzi è copiato nel Registro Dati



→ La scrittura (Write) consiste nel copiare il contenuto del Registro Dati nella cella di memoria inidirizzata dal Registro Indirizzi.



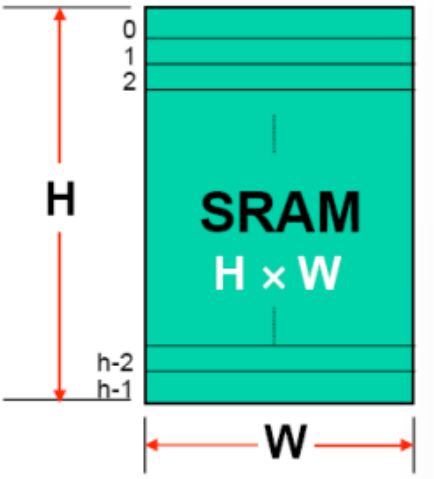
86) SRAM: lettura e scrittura

→ Tipologia di RAM volatile che non necessita di memory refresh, la cui realizzazione è basata sull'utilizzo dei latch e possiede dei tempi di accesso intorno a 0,5 – 2,5 ns.

→ Sia H l'altezza (numero di celle indirizzabili) e W la larghezza o ampiezza (numero di latch per ogni cella), essa viene realizzata come matrice di latch H x W.

→ Un singolo indirizzo viene usato sia per lettura che per scrittura ma non è possibile leggere e scrivere contemporaneamente (come per Register File).

→ Esistono SRAM a basso consumo (5/10 volte più lente).



87) SRAM: componenti

I componenti della SRAM sono:

→ Din e Dout;

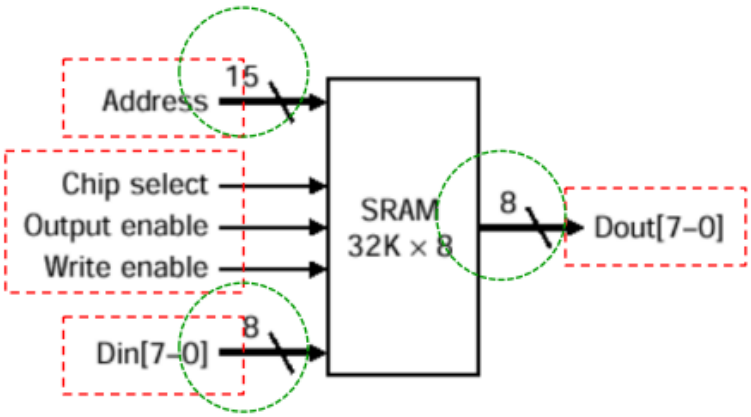
→ Address;

→ Segnali di controllo come:

1) Chip select: deve essere asserito per poter leggere o scrivere.

2) Output enable: deve essere asserito per poter leggere.

3) Write enable: deve essere asserito per poter scrivere.



88) SRAM: come viene realizzata?

→ Dato un numero elevato di celle di memoria, la SRAM utilizza enormi decoder (per selezionare il registro da scrivere) e multiplexer (per selezionare il registro da leggere).

→ Per evitare il multiplexer in uscita, è possibile usufruire una linea di bit condivisa su cui i vari elementi di memoria sono tutti collegati (OR).

→ Il collegamento avviene tramite quello che si chiama buffer a tre stati, che aprono e chiudono collegamenti (se il controllo è asserito o meno).

→ Il buffer ha due ingressi (dato e segnale di Enable) e una uscita:

1 – l'uscita è uguale al dato (0 o 1) se Enable è asserito;

2 – l'uscita viene impedita se Enable non è asserito.

90) DRAM: caratteristiche e componenti

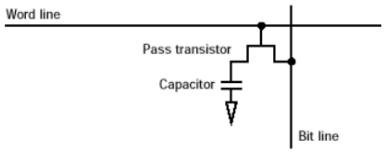
→ La memoria (DRAM) è un componente del computer che consente l'accesso ai dati a breve termine.

→ Per memorizzare le informazioni, la DRAM utilizza i condensatori, i quali vengono ricaricati periodicamente per evitare la perdita dell'informazione in essi contenuti.

→ Gli elementi di memoria di tipo DRAM sono meno costosi e più capienti rispetto al tipo SRAM, ma più lenti.

→ La DRAM è da 5 a 10 volte meno velce della SRAM.

→ Essa è realizzata con un solo transistor per bit (il quale viene chiuso trasferendo il potenziale elettrico del condensatore sulla bit line), e un condensatore (il quale possiede una carica).



90) SSRAM e SDRAM

→ Le Synchronous SRAM e DRAM (SSRAM e SDRAM) permettono di aumentare la banda di trasferimento della mempria sfruttando questa proprietà.

Le caratteristiche delle SSRAM e SDRAM sono:

→ memorie sincrone con segnale di clock;

→ è possibile specificare la volontà di trasferire dalla memoria una sequenza di celle consecutive (burst);

→ ogni burst è specificato da un indirizzo di partenza e da una lunghezza;

→ le celle del burst sono contenute all'interno di una stessa riga, selezionata una volta per tutte tramite decoder;

→ la memoria fornisce una delle celle del burst a ogni ciclo di clock.

91) Macchine a stati finiti e Mealy vs Moore

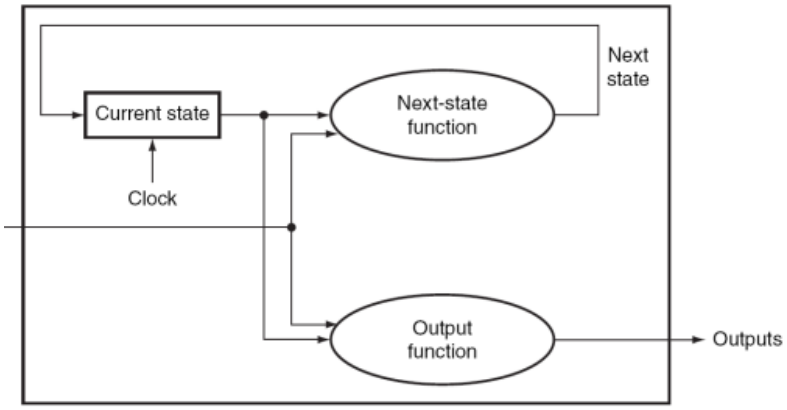
→ Le Finite State Machine (FSM) vengono usate per descrivere i circuiti sequenziali.

→ Sono composte da un set di stati e 2 funzioni:

1 – Next state function: determina lo stato successivo partendo dallo stato corrente e dai valori in ingresso;

2 – Output function: produce un insieme di risultati partendo dallo stato corrente e dai valori in ingresso.

→ Sono sincronizzate con il clock



→ Se dipende dallo stato corrente, Moore è usato come controller.

→ Se dipende dallo stato corrente e dagli input, viene usato Mealy.

→ Moore and Mealy sono equivalenti e si possono trasformare automaticamente tra di loro.

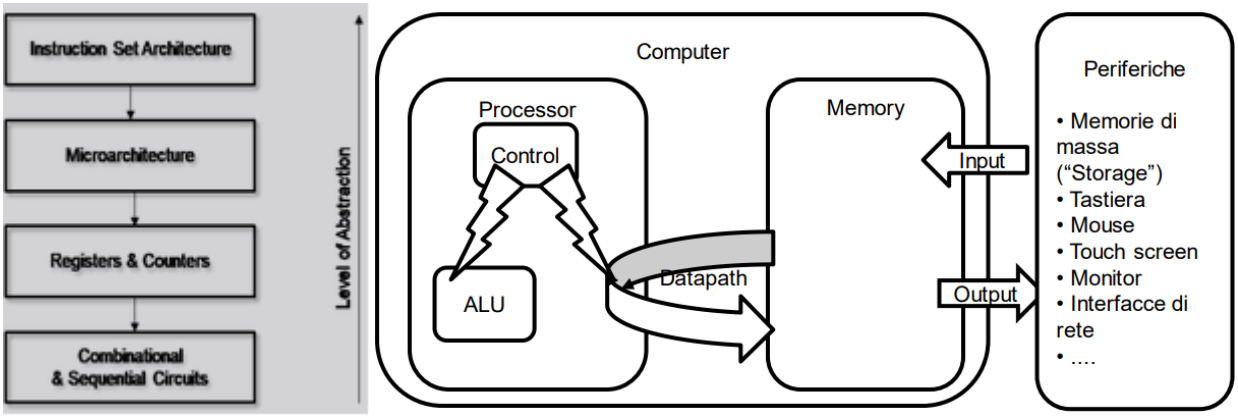
Capitolo 3 – ISA (Istruction Set Architecture)

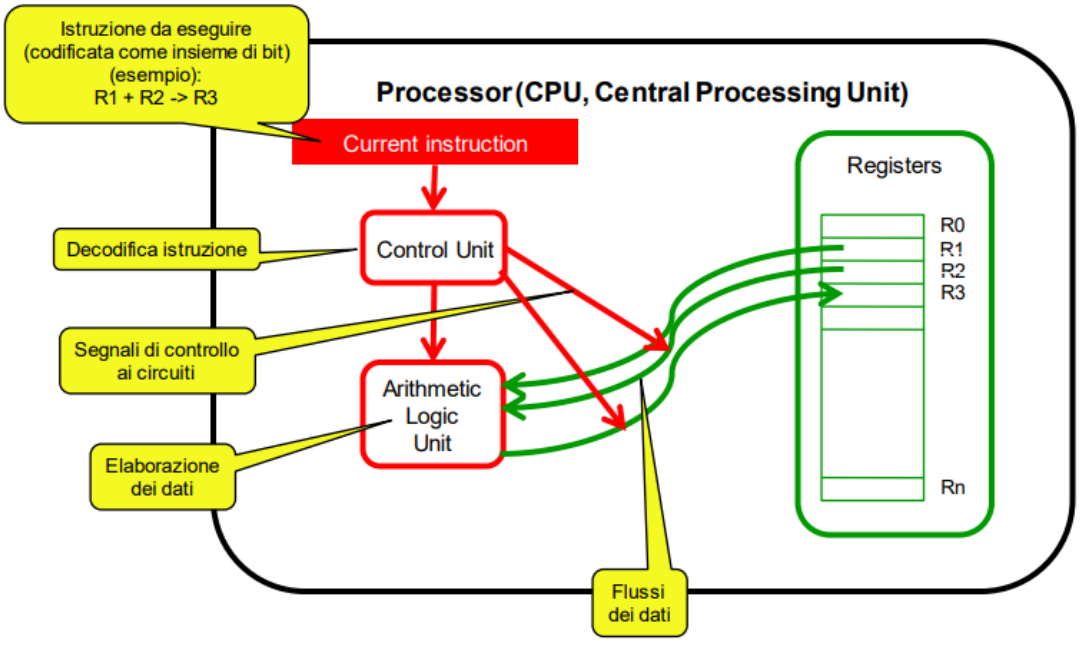
92) Definizione di CPU

CPU (Central Processing Unit): dispositivo hardware interno che esegue le istruzioni di un programma. Il processore (CPU) è in grado di eseguire di eseguire istruzioni molto semplici, come effettuare operazioni aritmetiche di base (somme e/o sottrazioni) o spostare numeri o altri dati presenti in memoria.

93) Istruction Set Architecture

L'ISA (Istruction Set Architecture) è un modello astratto di un computer che definisce come la CPU viene controllata dal software.





94) Differenza tra CISC e RISC

L'architettura RISC è l'architettura per microprocessori, caratterizzata da un set ridotto di istruzioni con tempi di esecuzione brevi e comparabili per facilitarne il parallelismo.

Le caratteristiche di questa architettura sono:

→ include istruzioni single – clock ridotte;

→ non è possibile il trasferimento tra la memoria ma tra i registri;

→ utilizzo di più transistor nei registri.

L'architettura CISC è il tipo di architettura che hanno istruzioni più complesse, più potenti.

Le caratteristiche di questa architettura sono:

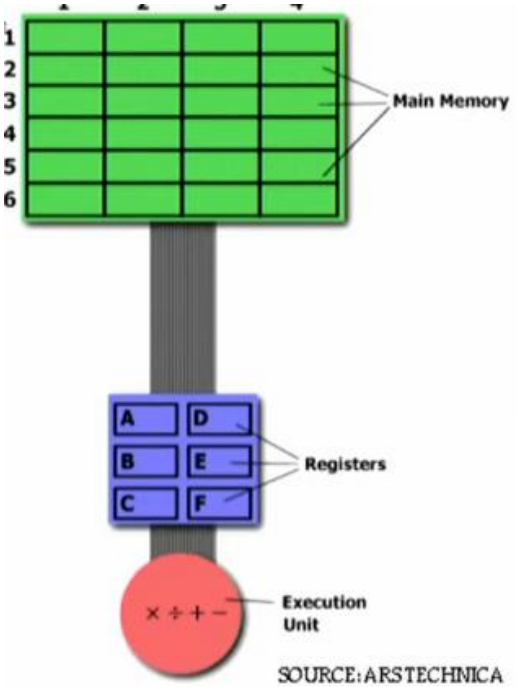
→ possiede 32 registri di 32 bit e le istruzioni sono di 32 bit;

→ include istruzioni multi – clock complesse;

→ Esiste lo spostamento dei dati in memoria;

→ i transistor vengono utilizzati per memorizzare istruzioni complesse.

Esempio: operazioni con matrici



La realtà hardware differisce dalla realtà software, la memoria è organizzata in celle sequenziali.

Memoria RAM

Nell'hardware ridotto (RISC) non esistono alcune operazioni, come il prodotto.

95) MIPS: istruzioni

→ Architettura costituita da 32 registri con 32 bit ciascuna.

Linguaggio Macchina:

```
00000001000010010101000000100000
```

Linguaggio Mnemonico

```
add $10, $8, $9
```

I dati in MIPS sono calcolati solo su registri.

Esempio:

```
add $10, $8, $9
```

Questa istruzione significa che il valore $10 equivale alla somma tra il contenuto di $8 e $9.

```
$10 = $8 + $9
```

Esistono tre tipologie di scomposizione:

→ R – Type: scomposizione su registri (scomposizione in cui si citano i registri)

→ I – Type;

→ J – Type

96) Istruzioni formato R – Type

Data la seguente istruzione:

```
add $10, $8, $9
```

La scomposizione è la seguente:

000000	01000	01001	01010	00000	100000

op.code (6 bit)	rs (5 bit)	rt (5 bit)	rd (5 bit)	shamt (5 bit)	funct (6 bit)

I componenti del formato sono:

→ operation code (6 bit): codifica dell'operazione;

→ rs (5 bit): primo registro sorgente (primo operando);

→ rt (5 bit): secondo registro sorgente (secondo operando);

→ rd (5 bit): registro destinazione (riceve risultato);

→ shamt (5 bit – shift amount): bit spesso irrilevanti e vengono utilizzati per effettuare lo shift e possono contenere qualsiasi bit;

→ funct (6 bit): variante della operazione;

Esempio utilizzo:

`add $10, $8, $9` → add su soli registri

001000	01000	01010	0000000000000100

97) Istruzioni formato I – Type

Nel formato I – type è possibile codificare la somma tra registri e numeri.

op.code (6 bit)	rs (5 bit)	rt (5 bit)	offset (16 bit)

Esempio:

z = x + 9

```
addi rt, rs, imm
```

8	rs	rt	imm

I componenti del formato sono:

→ operation code (6 bit): codifica dell'operazione;

→ rs (5 bit): registro sorgente (operando);

→ rt (5 bit): registro destinazione (riceve risultato);

→ immediate: registra l'offset.

Esempio: `addi $10, $8, 4`

98) Istruzioni formato J – Type

Nel formato J – type permette di saltare all'istruzione il cui indirizzo è $0x00010E8_{16}$

op.code (6 bit)	Jump word address (26 bit)

001000	00000000000000010000111010

I componenti della scomposizione sono:

→ operation code (6 bit): codifica dell'operazione;

→ jump word address (26 bit): rappresenta l'indirizzo a cui si vuole saltare.

Le altre caratteristiche sono:

→ carica nel PC (Program Counter) il valore 00000000000000010000111010;

→ con 26 bit si indirizzano 2^{26} Word. I 4 bit più significativi corrispondono ai 4 bit più significativi di PC.

PC (Program Counter): contiene l'indirizzo dell'istruzione successiva da eseguire.

99) Istruzione load word

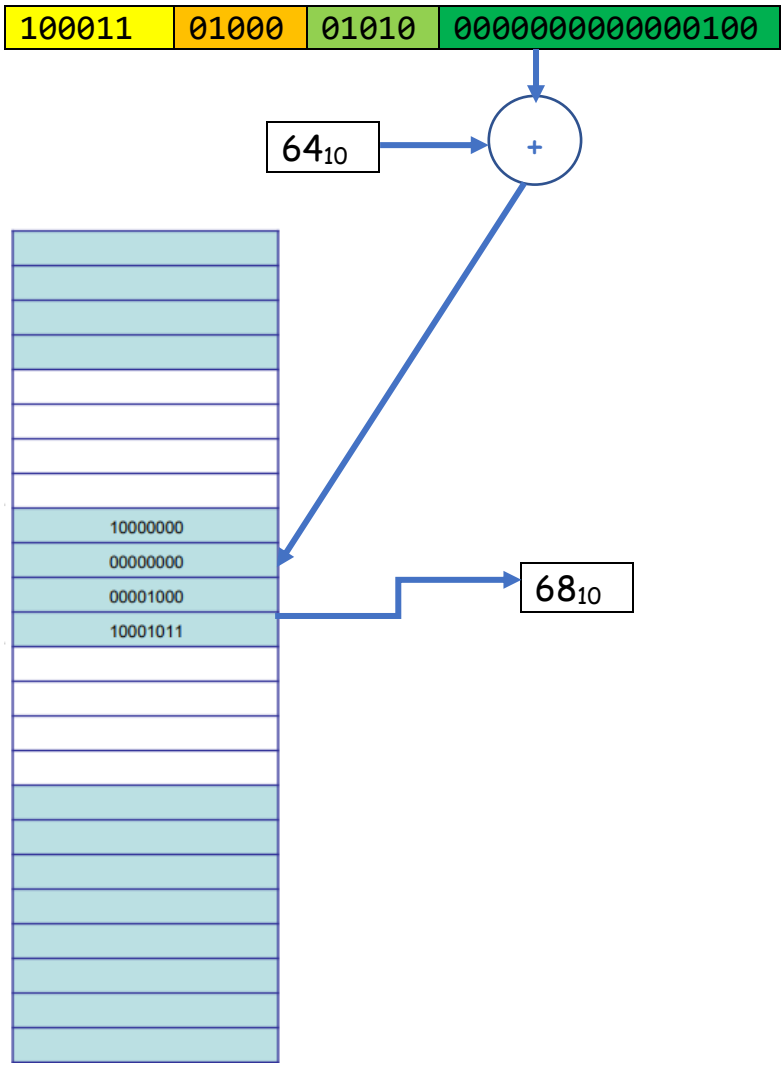
Carica nel registro 10 il contenuto della parola (32 bit) che è all'indirizzo di memria ottenuto come somma del registro 8 e dell'offset immediato 4.

op.code (6 bit)	rs (5 bit)	rt (5 bit)	offset (16 bit)

100011	01000	01010	0000000000000100

→ se rs contiene inizialmente 64 (00000000000000000001000000) e l'indirizzo in memoria della word da caricare è 68 (00000000000000000001000100), in rt vengono trasferiti 32 bit a partire dall'indirizzo così ottenuto.
ASSEMBLY: `lw $10, 4($8)`

Funzionamento



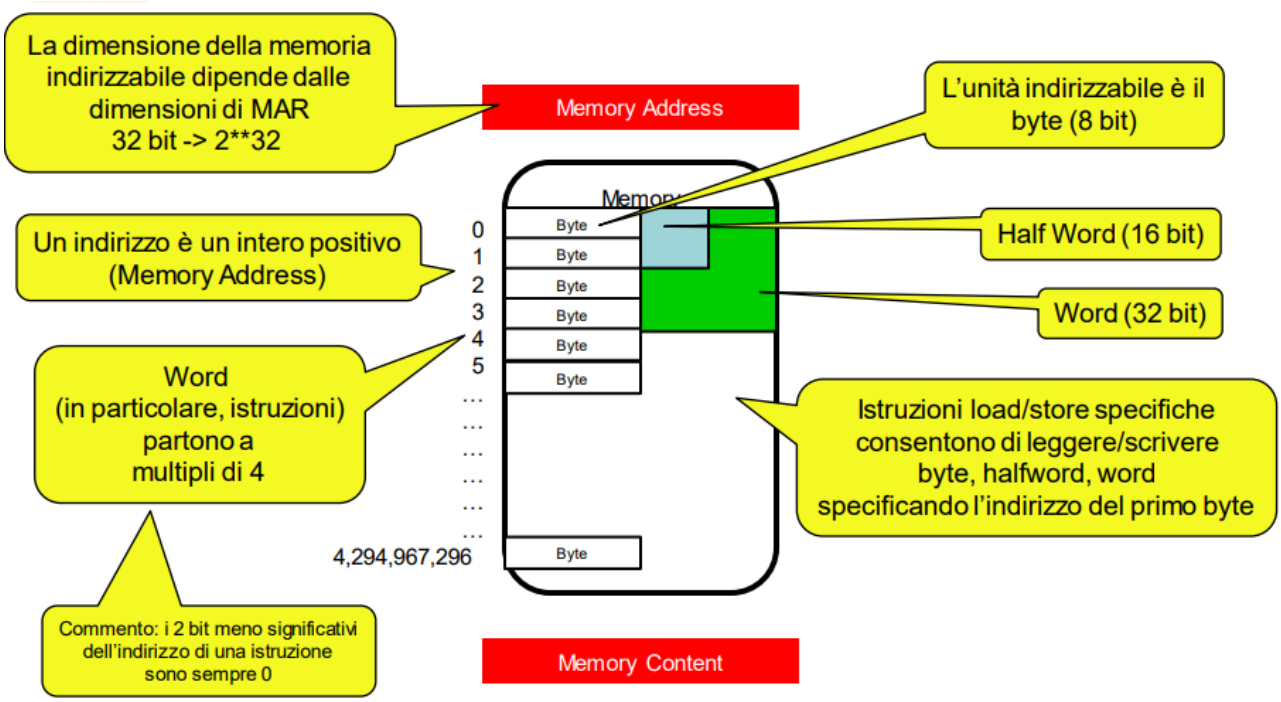
100) Modalità di indirizzamento

Descrive la tipologia in cui si prelevano i dati ed esistono tre modalità di indirizzamento:
→ Registri di CPU: indirizzamento a registri (i dati si muovono tra registri, come nel caso di R – type);
→ Memoria: indirizzamento immediato (come nel caso di I – type).
→ Memoria di massa ("Storage").

101) Descrivere l'indirizzamento di memoria

La realtà MIPS definisce l'indirizzo di memoria a seconda delle seguenti

caratterisitche:
→ la dimensione della memria indirizzabile dipende dalle dimensioni di MAR (32 bit → 2^{32} valori rappresentabili);
→ un indirizzo è un intero positivo (Memory Address);
→ l'unità è indirizzabile è il byte (8 bit);
→ le istruzioni load/store specifiche consentono di leggere/scriverse byte, halfword, word specificando l'indirizzo del primo byte.



102) Istruzione store word

Memorizza il contenuto del registro 10 (32 bit) all'indirizzo di memoria ottenuto come somma del contenuto del registro 8 e dell'offset immediato 4.

101011	01000	01010	0000000000000100

I componenti del formato sono:
→ operation code (6 bit): codifica dell'operazione;
→ rs (5 bit): registro sorgente (operando);
→ rt (5 bit): registro destinazione (riceve risultato);
→ offset immediato: registra l'offset.

ASSEMBLY: `sw $10, 4($8)`

103) Differenza tra store word e load word

→ Load word (lw): carica nel registro 10 il contenuto della parola (32 bit) che è all'indirizzo di memoria ottenuto come somma del registro 8 e dell'offset immediato 4;

`lw $10, 4($8)`

→ Store word (sw): memorizza il contenuto del registro 10 (32 bit) all'indirizzo di memoria ottenuto come somma del registro 8 e dell'offset immediato 4.

`sw $10, 4($8)`

104) Istruzione I – type (branch)

L'istruzione I – type (branch) salta a Branch Address se il contenuto del registro 16 (rs) è diverso dal contenuto del registro 17 (rt).

101011	01000	01010	0000000000000100

I componenti del formato sono:

→ operation code (6 bit): codifica dell'operazione;

→ rs (5 bit): registro sorgente (operando);

→ rt (5 bit): registro destinazione (riceve risultato);

→ Branch Address.

ASSEMBLY: `bne $16, $17, exit` oppure `bne $s0, $s1, exit`

Exit: è un'etichetta che l'assembler traduce in uno spiazzamento.

Il Branch Address salta ad un indirizzo con il seguente range:

valore immediato – 2^{15} <= BA < 2^{15}.

Il problema del "dove saltare" può essere risolto mediante due possibili soluzioni:

→ usare un registro base;

→ usare PC come registro base (CA + 4 + BA*4)

104) Istruzione R – type (shift left)

L'istruzione R – type (shift left) effettua lo shift di 4 bit a sinistra il contenuto del registro 16 e mette il risultato nel registro 10.

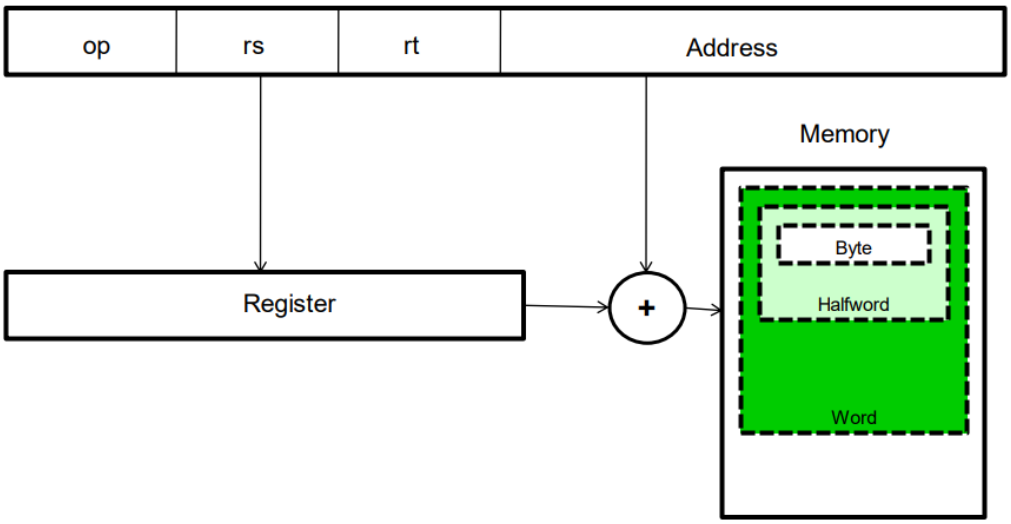
000000	00000	10000	01010	00100	000000

op.code (6 bit)	rs (5 bit)	rt (5 bit)	rd (5 bit)	shamt (5 bit)	funct (6 bit)

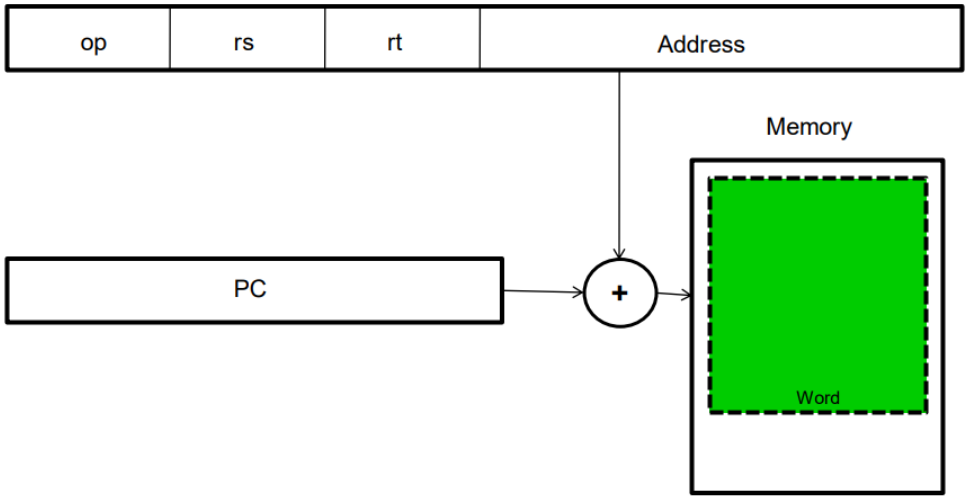
→ se rt contiene inizialmente 9, dopo l'esecuzione rd contiene 144.

ASSEMBLY: `sl $10, $16, 4` oppure `sl $t2, $s0, 4` (shift left logical)

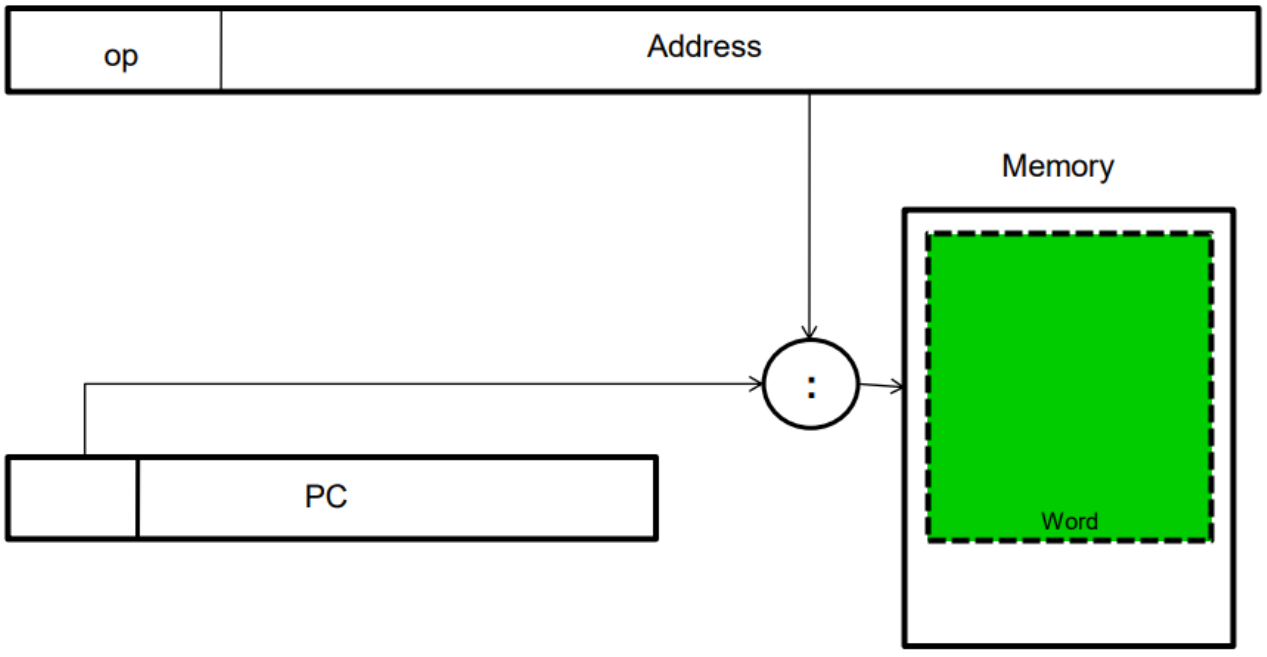
Base addressing



PC - relative addressing



Pseudodirect addressing



Capitolo 4: Assembly e Catena Programmativa

105) Descrivere le caratteristiche di un linguaggio di alto livello

Le caratteristiche di un linguaggio di alto livello sono:
→ notazione vicina al linguaggio corrente e alta notazione algebrica (maggiore espressività e leggibilità);
→ incremento di produttività (svincolata dalla conoscenza dei dettagli architetturali della macchina utilizzata);
→ indipendenza dalle caratteristiche dell'architettura (processore) su cui il programma sarà eseguito (portabilità);
→ ideata per macchine astratte, in grado di effettuare operazioni di più alto livello rispetto alle operazioni dei processori reali;
→ permettono l'uso di librerie di funzionalità già scritte (riusabilità del codice).

106) Cosa si intende per linguaggio macchina?

Per linguaggio macchina si intende il linguaggio con cui vengono scritti i programmi eseguibili da un computer. La grammatica dei linguaggi macchina dipende fortemente dalla tipologia di processore, il quale traduce le istruzioni presenti nel programma e le esegue. I linguaggi macchina sono anche denominati linguaggi di basso livello.

107) Cosa si intende per linguaggio assembly?

Linguaggio di programmazione le cui istruzioni sono composte da stringhe alfanumeriche corrispondenti in modo biunivoco alle istruzioni elementari dell'unità di elaborazione centrale (CPU – Central Processing Unit) di un calcolatore elettronico.
La sintassi è la seguente:

```
MOV AX,BX
ADD DL,AL
CMP AX,DX
```

108) Vantaggi del linguaggio assembler

La dipendenza dall'architettura del calcolatore permette di:
→ ottimizzare le prestazioni (maggiore efficienza);
→ scrivere programmi (potenzialmente) più compatti;
→ sfruttare al massimo le potenzialità dell'hardware sottostante;
→ programmare controller di processi e macchinari (e.g., real-time), o per apparati limitati (e.g., embedded computer, portatili).

109) Svantaggi del linguaggio assembler

Gli svantaggi del linguaggio assembly sono:

→ minore espressività: per esempio, strutture di controllo limitate;

→ necessità di conoscere i dettagli dell'architettura;

→ mancanza di portabilità su architetture diverse;

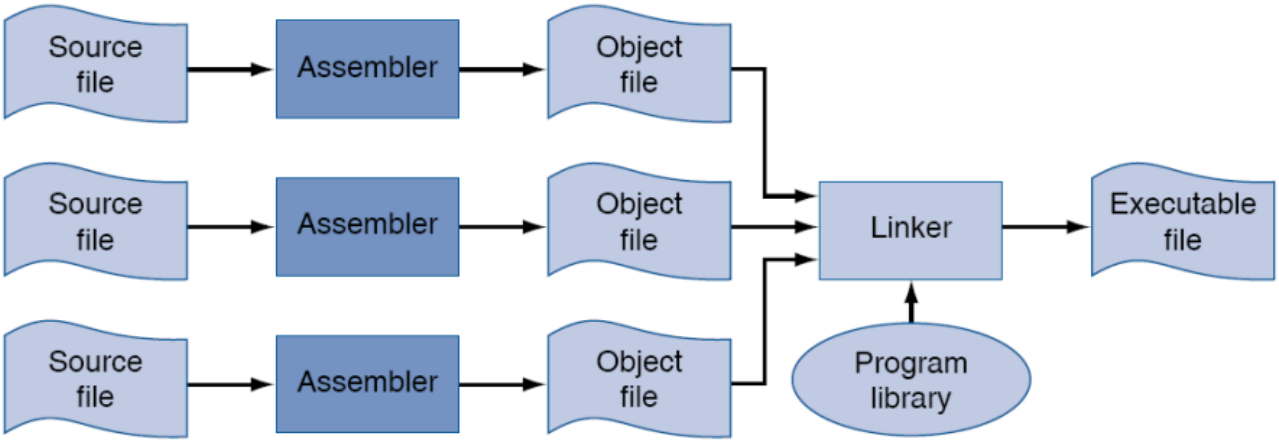
→ difficoltà di comprensione;

→ lunghezza maggiore dei programmi.

110) Compilatore, assembler e linker

111) Catena Programmativa

Il processo utilizzato per eseguire un file assembly è il seguente:
l'assemblatore traduce un file assembly in object file, il quale è collegato (linkato) con altri file e librerie nel file eseguibile.



112) Debugger: definizione e caratteristiche

Il debugger è un programma/software specificatamente progettato per l'analisi e l'eliminazione dei bug (debugging), ovvero errori di programmazione interni al codice di altri programmi.

La funzionalità del debugger è la seguente:

→ esecuzione passo-passo (step-by-step) di un programma;

→ ispezione del valore di variabili ed espressioni;

→ interruzione in punti predefiniti (breakpoint);

→ interruzione in caso di modifica del valore di determinate variabili (watchpoint);

→ visualizzazione degli indirizzi di memoria delle variabili o delle istruzioni.

113) Compilatore: definizione e caratteristiche

Il compilatore è un programma software che traduce le istruzioni di un linguaggio di programmazione ad alto livello in linguaggio macchina. Esso salva le nuove istruzioni in memoria (crea file.exe). Il compilatore traduce prima il codice poi lo esegue; per tale motivo offre una traduzione più veloce ma l'eseguibile creato (esempio.exe) sarà utilizzabile solo sul calcolatore su cui il codice è stato compilato e quindi non sarà portatile.

114) Assemblatore: definizione e caratteristiche

L'assemblatore è un software che trasforma le istruzioni mnemoniche dell'assembly in linguaggio macchina.
Esso gestisce:
→ etichette;
→ pseudo – istruzioni;
→ numeri in base diversi (binario, decimale, esadecimale).

115) Come avviene il processo di assemblaggio?

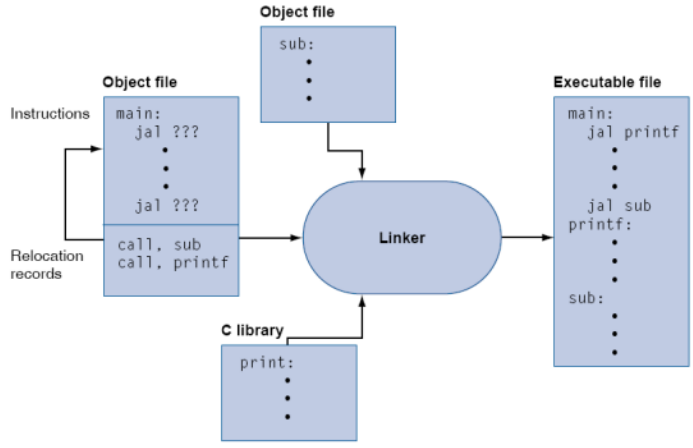
L'assemblaggio è un procedimento sequenziale che esamina, riga per riga, il codice sorgente Assembly, traducendo ciascuna riga in un'istruzione del linguaggio macchina.
Le caratteristiche del processo di assemblaggio sono:
→ viene applicato modulo per modulo al programma e costituisce per ogni modulo la tabella dei simboli del modulo;
→ traduce i codici mnemonici (simbolici) delle istruzioni nei corrispondenti codici binari;
→ traduce i riferimenti simbolici (variabili, registri, etichette di salto, parametri) nei corrispondenti indirizzi numerici.
→ l'etichetta consente al programmatore in assembly di risparmiare lo sforzo nel ricordare la posizione in memoria di ogni singola istruzione;
→ poiché l'etichetta di salto genera il problema dei riferimenti in avanti (ossia, riferimenti ad etichette successive o contenute in altri file), l'assemblatore deve leggere il programma sorgente due volte;
→ ogni lettura del programma sorgente è chiamata passo e l'assemblatore è chiamato traduttore a due passi;
→ ogni modulo assemblato di default parte dall'indirizzo 0;
→ in sistemi dotati di meccanismi di memoria virtuale tali indirizzi sono indirizzi virtuali.

116) Descrivere la tabella dei simboli

La tabella dei simboli contiene i riferimenti simbolici presenti nel modulo da tradurre e, al termine del primo passo, conterrà gli indirizzi numerici di tutti i simboli, tranne quelli esterni al modulo in esame.
Le etichette vengono suddivise in tre categorie:
→ etichette associate a direttive dell'assemblatore che definiscono costanti simboliche nella tabella dei simboli viene creata la coppia <etichetta, valore> e in ogni istruzione che fa riferimento al simbolo viene sostituito il valore;
→ etichette che definiscono variabili (spazio di memoria + eventuale inizializzazione), l'assemblatore riserva spazio, eventualmente inizializza la zona di memoria e crea nella tabella la coppia <etichetta, indirizzo > e in ogni istruzione che fa riferimento al simbolo (all'etichetta), il simbolo viene sostituito con l'indirizzo;
→ etichette che definiscono istruzioni di salto, dove l'assemblatore deve generare un riferimento all'indirizzo dell'istruzione destinazione di salto.

117) Linker (link editor): definizione e caratteristiche

Il linker (link editor) è un software di sistema che raccoglie le procedure separatamente prodotte nella fase di traduzione di un programma scritto in un certo linguaggio di programmazione e provvede a realizzare un collegamento (to link) tra di loro.
Le operazioni de linker sono:
→ inserisce in memoria in modo simbolico il codice e i moduli dati;
→ determina gli indirizzi dei dati e delle etichette che compaiono nelle istruzioni;
→ corregge i riferimenti interni ed esterni e risolve i riferimenti in sospeso (a etichete esterne);
→ genera il file eseguibile.



118) Loader: caratteristiche

Il loader è la parte di un sistema operativo responsabile del caricamento di programmi e librerie.

Le caratteristiche del loader sono:

→ lettura dell'intestazione del file eseguibile per determinare la lunghezza del segmento di testo (cioè delle istruzioni) e del segmento dati (cioè le variabili);

→ creazione di uno spazio di indirizzamento sufficiente a contenere testo e dati;

→ copia delle istruzioni e dati dal file eseguibile in memoria;

→ copia nello stack degli eventuali parametri passati al programma principale;

→ inizializzazione dei registri e impostazione dello stack pointer affinche' punti alla prima locazioni libera;

→ salto a una procedura di startup la quale copia i parametri nei registri argomento e chiama la procedura principale del programma;

→ quando la procedura principale restituisce il controllo, la procedura di startup termina il programma con una chiamata alla funzione di sistema exit.

119) Nomi e utilizzi dei registri

Nome Simbolico	Numero	Uso
$zero	0	Costante 0
$at	1	Assembler temporary
$v0-$v1	2-3	Functions and expressions evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS kernel
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

120) Direttive assemblatore

Le caratteristiche delle direttive dell'assemblatore sono:

→ non corrispondono a istruzioni macchina;

→ sono indicazioni date all'assembler per consentirgli di associare etichette simboliche a indirizzi, allocare spazio di memoria per le variabili, decidere in quali zone di memoria allocare istruzioni e dati ed etc.

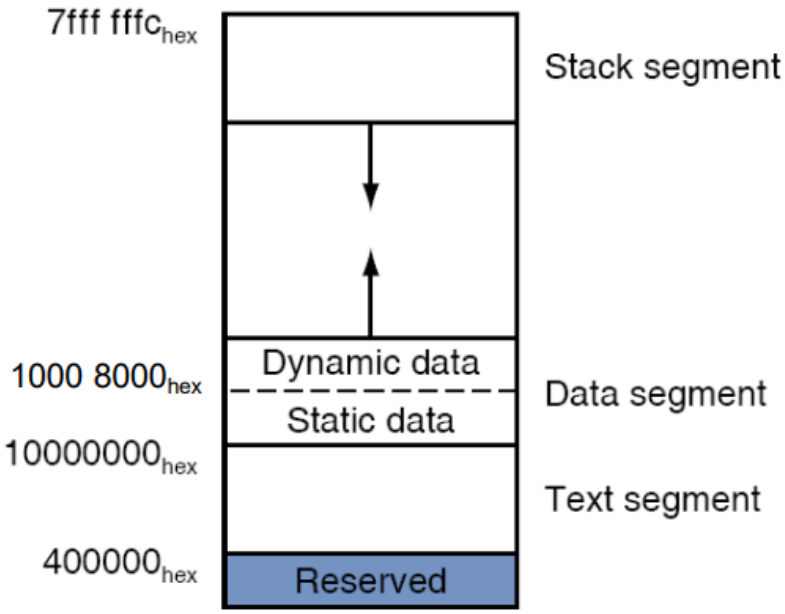
Esempi:

`.data <address>` → inserito nel segmento dati

`.byte b1,b2,..,bn` → inizializza byte successivi

`.word w1,w2,...,wn` → inizializza i valori in word successive;

`.text <addr>` → quel che segue va nel segmento text.



121) Istruzioni aritmetiche e logiche

Le istruzioni aritmetiche e logiche in assembly sono:

→ `add rd, rs, rt`: addizione (rs + rt → rd);

→ `addi rd, rs, imm`: addizione immediata (imm + rs→ rd);

→ `and rd, rs, rt`: and bit a bit di rs e rt → rd;

→ `or rd, rs, rt`: or bit a bit di rs e rt → rd;

→ `ori rt, rs, imm`: or immediato di rs e zero - extended imm → rt;

→ `sll rd, rt, shamt`: shift left rt della distanza shamt → rd.

122) Manipolazione costanti

Le manipolazioni di valori in assembly sono:

→ `lui rt, imm`: load upper immediate e i 16 bit bassi di rt sono 0;

→ `lui $s0, 61 (lui $s0, 0x003d)` : contenuto di `$s0` è 0000 0000 0011 1101 0000 0000 0000 0000;

→ `ori $s0, $s0, 2304 (lui $s0, 0x0900)`: contenuto di `$s0` è 0000 0000 0011 1101 0000 1001 0000 0000

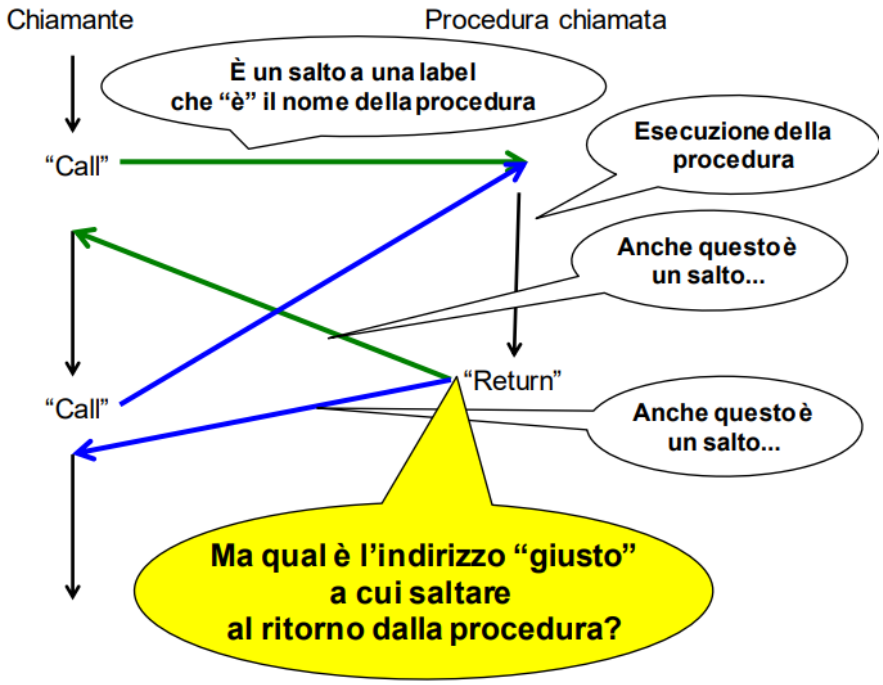
123) Definizione di pseudo istruzione

La psuedo istruzione è l'istruzione assembly che non ha una corrispondente

istruzione macchina e che viene tradotta dall'assembler in una sequenza di istruzioni.
Ad esempio:
`li $s0, 4000000` → tradotta in `lui $s0, 61`;

`move $t0, $t1 -> tradotta in add $t0, $zero, $t1`

124) Flusso di controllo: chiamate procedure



125) Istruzione jal (jump and link)

L'istruzione jal (jump and link) salta a una procedura indicata nell'istruzione e contemporaneamente crea un collegamento a dove deve ritornare per continuare l'esecuzione del chiamante. Salva nel registro $ra (registro 31) ("return address") l'indirizzo a cui tornare dopo l'esecuzione della procedura (l'indirizzo successivo a quello dell'istruzione jal, cioè l'indirizzo in cui si trova la jal + 4). Tale indirizzo si trova nel PC (Program Counter).
Sintassi:

```
jal <indirizzoProcedura>
```

126) Istruzione jr (jump register)

Salta all'indirizzo contenuto in un registro ed è una istruzione di uso generale che consente di saltare a qualsiasi locazione di memoria
Sintassi:

```
jr <registro>
```

`jr $ra` → uno degli indirizzi tipici di jr per realizzare il ritorno da procedura saltando all'indirizzo precedentemente salvato da jal.

127) Come avviene il passaggio dei parametri in assembly

In Assembly il passaggio dei parametri viene gestita nella maniera seguente:
→ `$a0 - $a3`: registri argomento per il passaggio dei parametri;
→ `$v0 - $v1`: registri valore per la restituzione dei risultati;
→ dal punto di vista hw sono registri come tutti gli altri, ma il loro utilizzo per il passaggio di parametri e risultati è una convenzione programmativa che deve essere rispettata per consentire di scrivere procedure che possono essere scritte senza bisogno di sapere come è fatto il programma che le chiama e senza bisogno di sapere come sono fatte dentro;
→ un parametro può essere un dato o un indirizzo!!

128) Procedura: definizione e caratteristiche

→ Una procedura è un meccanismo per organizzare il codice in modo comprensibile e riutilizzabile e consentono ai programmatori di concentrarsi su una parte del problema alla volta.
I 6 passi di una procedura sono:
→ setting dei parametri in un luogo accessibile alla procedura;
→ trasferimento del controllo alla procedura e salvare l'indirizzo dell'istruzione dove tornare dopo la chiamata della procedura (usare l'istruzione jal);
→ acquisizione delle risorse per l'esecuzione della procedura;
→ esecuzione del compito richiesto;
→ mettere il risultato in un luogo accessibile al chiamante;
→ restituire il controllo al punto di partenza (usare l'istruzione jr).

129) Syscall

Le chiamate di sistema (syscall) permettono ai programmi utente di richiamare i servizi del sistema operativo: sono solitamente disponibili come speciali istruzioni assembler o come delle funzioni nei linguaggi che supportano direttamente la programmazione di sistema (ad esempio, il C).

La tabella delle istruzioni è la seguente:

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

130) Se una procedura usa registri, cosa succede del contenuto lasciato nei registri dal chiamante?

Si utilizzano i registri $t e $s.

La differenza tra i registri $t e $s è la seguente:

→ i registri $t ("temporary") non vengono salvati dalla procedura e il chiamante non può aspettare di trovare i contenuti dei registri $t immuatati dopo una chiamata ad una procedura;

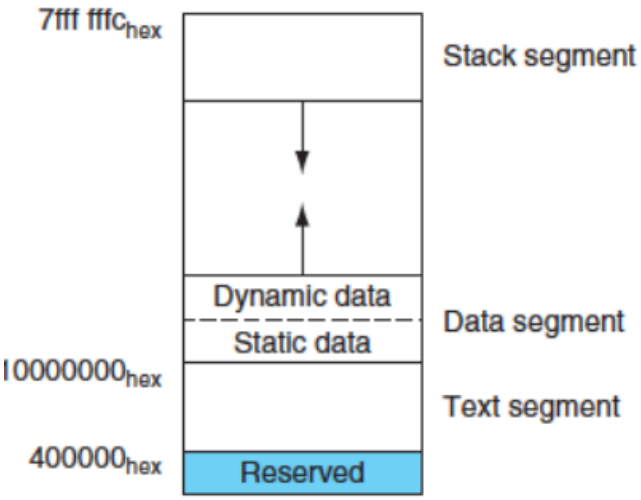
→ i registri $s ("saved") vengono salvati dalla procedura e il chiamante ha diritto di aspettarsi che i contenuti dei registri $s siano immutati dopo una chiamata a procedura.

I registri $s vengono salvati nello stack.

131) Procedura foglia e non foglia

La procedura foglia è la procedura che non chiama altre procedure, mentre la

procedura non foglia chiama altre procedure.



132) Uso dello stack: descrizione

→ Lo stack consiste di un insieme di segmenti logici (stack frame) che vengono impilati (pushed) sullo stack quando viene chiamata una funzione e spilati (popped) quando la funzione ritorna.
→ Questa porzione di memoria utilizza una gestione di dati del tipo LIFO (Last in First out). Normalmente viene utilizzato per il salvataggio temporaneo di dati.
→ Il frame di stack (chiamata di procedura) è un blocco di memoria associato alla procedura.
→ La variabile $sp punta alla prima parola del frame mentre $fp punta all'ultima parola del frame. Un frame è (solitamente) è multiplo della parola doppia (8 byte).

Esempio: frame di 32 byte

```
addi $sp, $sp. -32          # frame di stack di 32 byte
addi $fp, $sp, 28           # imposta il frame pointer
sw $ra, 0($fp)              # salva l'indirizzo di ritorno come primo
                            # word nel frame sullo stack
```

133) Compiti dello stack frame

Prima di chiamare una procedura, lo stack frame i seguenti passi:

→ impostare gli argomenti da passare alla procedura in $a0-$a3; eventuali altri argomenti sono nella memoria o nello stack;

→ salvare eventualmente i registri $a0-$a3 e $t0-$t9 in quanto la procedura chiamata puo' usare liberamente questi registri;

→ chiamare la procedura tramite l'istruzione jal nome_procedura.

Successivamente (dopo la chiamata di una procedura), lo stack frame i seguenti passi:

→ allocare il suo stack frame ($sp = $sp – dimensione frame procedura)

→ salvare i valori disponibili nei registri $s0 – $s7, $fp, $ra se intende usarle tali registri per la sua esecuzione, se per esempio la procedura non chiama un'altra procedura non è necessario salvare il registro $ra;

→ settare il frame pointer (che indica l'indirizzo dell'ultima parola del frame): $fp = $sp – dimensione frame procedura + 4.

Alla fine della procedura, lo stack frame i seguenti passi:

→ mettere il valore di ritorno nel registri $v0, $v1;

→ ripristinare i valori dei registri salvati sullo stack ($s0-$s7, $fp, $ra)

→ liberare lo spazio sullo stack: $sp = $sp + dimensione frame procedura;

→ eseguire l'istruzione jr $ra.