

# Decimale a IEEE754

sabato 20 luglio 2024 09:35

1. [5] Convertire in binario secondo la codifica IEEE754, il numero decimale 16,25.

Convertiamo 16 in binario: 10000

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 10000,01

Normalizzare: Il numero binario 10000,01 può essere scritto come 1.000001×2^4

**SEGNO 0**

**ESPONENTE 4+127=131 -> 10000011**

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 0000010000000000000000000

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 16,25?

Codifica di 17,25

Qual è la **risoluzione della codifica** in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

In **virgola mobile IEEE 754** la **risoluzione variabile**:

Per numeri piccoli la risoluzione è alta.

Supponiamo di rappresentare 1,0000000000000000000000000000000 e successivo 1,0000000000000000000000000000001 = 2^-23

Per numeri grandi la risoluzione diminuisce.

Supponiamo di rappresentare 128 -> 1.0000000 \* 2^7 la risoluzione diminuisce 2^-23 \* 2^7 = 2^-16

Anche per 129 -> 1.0000001 \* 2^7 fino a 255 -> 1.1111111 \* 2^7 incluso sarà sempre 2^-23 \* 2^7 = 2^-16

Se volessi rappresentare 256 -> 1.00000000 \* 2^8 la risoluzione scende 2^-23 \* 2^8 = 2^-15

511 -> 1.11111111 \* 2^8 risoluzione 2^-23 \* 2^8 = 2^-15

512 -> 1.000000000 \* 2^9 risoluzione 2^-23 \* 2^9 = 2^-14

Fino a 1.11111111 \* 2^9 = 1023 risoluzione 2^-23 \* 2^9 = 2^-14

In codifica intera è costante 1 tra un numero e l'altro.

In **virgola fissa** la risoluzione è **costante ma diversa da 1** ed è determinata da quanti bit dedichi alla parte frazionaria.

Risoluzione=2^f dove f è il numero di bit riservati alla parte frazionaria.

Esempio: con 3 bit parte frazionaria, la risoluzione è 2^-3

Scrivere in complemento a 2 su 8 bit la **sottrazione** espressa da numeri in base 10: 7-9 e calcolarne il risultato in binario.

7 -> 00000111

9 -> 00001001

-9 -> 11110111

7+(-9) = -2 ----> 11111110

1. [4] Convertire in binario secondo la codifica IEEE754, il numero decimale 2,25.

Convertiamo 2 in binario: 10

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 10,01

Normalizzare: Il numero binario 10,01 può essere scritto come 1.001×2^1

**SEGNO 0**

**ESPONENTE 1+127=128 -> 10000000**

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 0010000000000000000000000

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 2,25?

Qual è la **risoluzione della codifica** in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

Scrivere in complemento a 2, su 8 bit, la sottrazione espressa da numeri in base 10: 6-7 e calcolarne il risultato in binario.

1. [5] Convertire in binario secondo la codifica IEEE754, il numero decimale 12,25.

Convertiamo 12 in binario: 1100

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 1100,01

Normalizzare: Il numero binario 1100,01 può essere scritto come 1.10001×2^3

**SEGNO 0**

**ESPONENTE 3+127=130 -> 10000010**

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 1000100000000000000000000

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 12,25?

Qual è la **risoluzione della codifica** in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 7-11 e calcolarne il risultato in binario.

2. [5] Convertire in binario secondo la codifica IEEE754, il numero decimale 1,25.

Convertiamo 1 in binario: 1

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 1,01

Normalizzare: Il numero binario 1,01 può essere scritto come  $1.01 \times 2^0$

**SEGNO 0**

**ESPONENTE**  $0+127=127 \rightarrow 01111111$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 0100000000000000000000000

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 1,25?

Quale coppia di numeri decimali ha una distanza pari a 1 LSB di un numero binario codificato in IEEE 754?

Qual è la risoluzione della codifica in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

La prima coppia di numeri decimali consecutivi in formato **IEEE 754 float (32 bit)** che dista esattamente 1,0 è:

$$2^{23} = 8.388.608 \quad \text{e} \quad 2^{23} + 1 = 8.388.609$$

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 8-10 e calcolarne il risultato in binario.

1. [4] Convertire in binario, mediante la codifica IEEE 754, il numero -512,25.

Convertiamo 512 in binario: 1000000000

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 1000000000,01

Normalizzare: Il numero binario 1000000000,01 può essere scritto come  $1.0000000001 \times 2^9$

**SEGNO 1**

**ESPONENTE**  $9+127=136 \rightarrow 10001000$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 00000000010000000000000

La conversione risulterà esatta? Perché? Esistono delle condizioni nelle quali questa conversione non risulta esatta?

Esprimere la risoluzione della codifica binaria IEEE 754 (distanza tra due numeri consecutivi codificati).

**Risoluzione:** La distanza tra due numeri consecutivi in IEEE 754 a precisione singola dipende dall'esponente del numero.

È calcolata come  $2^{\text{esponente}-23}$ , dove 23 è il numero di bit della mantissa.

In generale, più grande è l'esponente, più grande è la distanza tra due numeri consecutivi rappresentabili, mentre per esponenti più piccoli, la distanza tra i numeri consecutivi è più piccola.

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 17-16 e calcolarne il risultato in binario.

1. [4] Codificare in IEEE 754 il numero -128,125.

Convertiamo 128 in binario: 10000000

Convertiamo 0,125 in binario: La parte frazionaria si converte in 0,001 in binario.

Combinare: 10000000,001

Normalizzare: Il numero binario 10000000,001 può essere scritto come  $1.0000000001 \times 2^7$

**SEGNO 1**

**ESPONENTE**  $7+127=134 \rightarrow 10000110$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 00000000100000000000000

E' sempre esatta la conversione da numero decimale in base dieci a numero decimale in base 2 in codifica IEEE754? Perché?

Quale sarà il numero codificato successivo nello standard IEEE754?

A partire dalla codifica binaria, scrivere la codifica esadecimale dello stesso numero.

Esistono delle condizioni in cui la conversione da binario a decimale non risulta esatta?

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 16-9 e calcolarne il risultato in binario.

Qual è il numero intero più grande e il numero intero più piccolo rappresentabile su 8 bit in complemento a 2?

Nel sistema in complemento a 2 su 8 bit, i numeri interi firmati sono rappresentati con:

- 1 bit di segno
- 7 bit di valore

1. [4] Convertire in binario, mediante la codifica IEEE 754, il numero 125,25.

Convertiamo 125 in binario: 1111101

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 1111101,01

Normalizzare: Il numero binario 1111101,01 può essere scritto come  $1.11110101 \times 2^6$

**SEGNO 0**

**ESPONENTE**  $6+127=133 \rightarrow 10000101$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 11110101000000000000000

La conversione risulterà esatta? Perché? Esistono delle condizioni in cui questa conversione non risulta esatta?

Esprimere la risoluzione (distanza tra due numeri consecutivi codificati) della codifica binaria IEEE 754.

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 17-9 e calcolarne il risultato in binario.

1. [4] Codificare in IEEE754 in singola precisione il numero +111,5.

Convertiamo 111 in binario: 1101111

Convertiamo 0,5 in binario: La parte frazionaria si converte in 0,1 in binario.

Combinare: 1101111,1

Normalizzare: Il numero binario 1101111,1 può essere scritto come  $1.1011111 \times 2^6$

**SEGNO 0**

**ESPONENTE**  $6+127=133 \rightarrow 10000101$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 1011110000000000000000000

Quale coppia di numeri codificati consecutivamente in IEEE754 dista esattamente 1 unità? Perché?

Qual è la risoluzione della codifica in virgola mobile, in virgola fissa e della codifica intera? Perché?

Scrivere in complemento a 2 su 8 bit la sottrazione espressa in base 10: 9-14 e calcolare il risultato in binario.

Qual è il numero più grande e il numero più piccolo rappresentabile su 8 bit in complemento a 2?

1. [4] Codificare in notazione binaria IEEE754, il numero decimale -12,75.

Convertiamo 12 in binario: 1100

Convertiamo 0,75 in binario: La parte frazionaria si converte in 0,11 in binario.

Combinare: 1100,11

Normalizzare: Il numero binario 1100,11 può essere scritto come  $1.10011 \times 2^3$

**SEGNO 1**

**ESPONENTE**  $3+127=130 \rightarrow 10000010$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 1001100000000000000000000

La codifica risulterà esatta? Perché?

Esprimere la risoluzione della codifica binaria IEEE754 su 32 bit.

(La risoluzione è la distanza tra due numeri consecutivi codificati).

Convertire il numero binario IEEE754 ottenuto in esadecimale.

Scrivere in complemento a 2 su 8 bit la sottrazione espressa in base 10: -1 - 31 e calcolare il risultato in binario.

1. [4] Convertire in binario, mediante la codifica IEEE 754, il numero 145,25.

Convertiamo 145 in binario: 10010001

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 10010001,01

Normalizzare: Il numero binario 10010001,01 può essere scritto come  $1.001000101 \times 2^7$

**SEGNO 0**

**ESPONENTE**  $7+127=134 \rightarrow 10000110$

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine. 0010001010000000000000000

La conversione risulterà esatta? Perché? Esistono delle condizioni in cui questa conversione non è esatta?

Esprimere la risoluzione (distanza tra due numeri consecutivi codificati) della codifica binaria IEEE 754.

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 15-9 e calcolarne il risultato in binario.

1. [4] Convertire in binario il numero decimale 32,25 e codificarlo in IEEE754.

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 32,25?

E quale nel Google Brain Format?

Quale coppia di numeri codificati sequenzialmente in IEEE754 (variazione del bit meno significativo) dista esattamente una unità?

Qual è la risoluzione della codifica in virgola mobile IEEE 754 in singola precisione, della codifica intera e della codifica in virgola fissa?

Come viene rappresentata la situazione +oo, -oo e NaN in IEEE754?

Scrivere in complemento a 2 su 16 bit la sottrazione espressa da numeri in base 10: 8 – 16, calcolarne il risultato ed esprimere in codifica binaria in complemento a due.

**Quale coppia di numeri codificati consecutivamente in IEEE754 dista esattamente 1 unità? Perché?**

La coppia di numeri che dista esattamente 1 unità in IEEE 754 è quella costituita da due numeri **subnormali**, cioè quelli con esponente 0000000000000000000000000. Questo succede perché il passo tra i numeri è molto piccolo quando l'esponente è zero e la mantissa cambia di uno solo nell'ultimo bit.

**Cosa si intende per codifica denormalizzata?**

In IEEE 754, un numero **normalizzato** in formato a singola precisione (32 bit) è rappresentato come:

$$(-1)^{\text{segn}} \times 1.m \times 2^{(e-127)}$$

Dove:

- **segn** è il bit di segno (1 bit)
- **m** è la mantissa (composta dai 23 bit della parte frazionaria)
- **e** è l'esponente (8 bit) con **bias** 127

Per i numeri **normalizzati**, l'esponente è compreso tra 1 e 254 (inclusi), e la mantissa è sempre preceduta da un **1 implicito** (ossia la rappresentazione della mantissa è sempre di tipo  $1.m$ , dove  $m$  è la parte frazionaria della mantissa).

#### Numeri Denormalizzati

Quando l'esponente è **zero** (ovvero  $e = 00000000$  in binario), i numeri diventano **denormalizzati**. In questo caso:

- L'esponente viene interpretato come un valore di  $1 - 127 = -126$ , ovvero l'esponente effettivo è  $-126$ .
- La mantissa, invece, non ha più il bit implicito di 1, ma è direttamente rappresentata dai 23 bit successivi al punto binario.

Questa rappresentazione consente di rappresentare numeri molto piccoli.

#### Spiegare perché la porta NOR è chiamata porta universale

La porta **NOR** è chiamata **universale** perché è **capace di realizzare qualsiasi operazione logica** (AND, OR, NOT, XOR, ecc.) utilizzando opportunamente la combinazione di più porte NOR. Questo la rende fondamentale in elettronica digitale, dove può essere utilizzata come building block per implementare qualsiasi circuito logico.

#### Operazione 9-10

Per calcolare  $9 - 10$  in complemento a 2 su 8 bit, seguiamo questi passaggi:

##### 1. Rappresentare i numeri in binario

In complemento a 2, prima di eseguire l'operazione di sottrazione, dobbiamo rappresentare i numeri in binario su 8 bit:

- **9 in binario (su 8 bit):**  
9 in decimale è 1001, quindi scriviamo 9 come 00001001 su 8 bit.
- **10 in binario (su 8 bit):**  
10 in decimale è 1010, quindi scriviamo 10 come 00001010 su 8 bit.

##### 2. Calcolare il complemento a 2 di 10

Per sottrarre 10 da 9, possiamo usare il metodo del complemento a 2, che consiste nel sommare il complemento a 2 di 10 a 9. Procediamo con il calcolo del complemento a 2 di 10:

1. Rappresentiamo 10 in binario su 8 bit: 00001010.
2. Invertiamo tutti i bit: 11110101.
3. Aggiungiamo 1 al risultato:  
 $11110101 + 1 = 11110110$ .

Quindi, il complemento a 2 di 10 su 8 bit è 11110110.

##### 3. Sommare il complemento a 2 di 10 a 9

Ora sommiamo il complemento a 2 di 10 a 9 (che è 00001001):

$$\begin{array}{r} 00001001 \quad (9) \\ +11110110 \quad (\text{complemento a 2 di 10}) \\ \hline 11111111 \end{array}$$

Il risultato della somma è 11111111, che è la rappresentazione in complemento a 2 di -1.

##### 4. Interpretazione del risultato

Il risultato 11111111 in complemento a 2 su 8 bit  $\downarrow$  rappresenta -1 in decimale. Questo è il risultato di  $9 - 10$ .

**La conversione in IEEE 754 di un numero risulterà sempre esatta?**

**Perché? Esistono delle condizioni nelle quali questa conversione non risulta esatta?**

No, la codifica IEEE 754 non risulterà sempre esatta. La rappresentazione in virgola mobile utilizzata dalla codifica IEEE 754 è una rappresentazione approssimata dei numeri reali. Questo significa che alcuni numeri non possono essere rappresentati esattamente e possono verificarsi degli errori di arrotondamento.

La codifica IEEE 754 utilizza una determinata quantità di bit per rappresentare la mantissa (parte frazionaria) e l'esponente di un numero in virgola mobile. Tuttavia, poiché il numero di bit è limitato, ci sono limiti alla precisione con cui i numeri possono essere rappresentati.

Ad esempio, si può considerare il numero decimale ricorrente 0,333... Poiché la rappresentazione binaria della frazione 1/3 è una serie infinita di "0,0101..." ricorrenti, la codifica IEEE 754 a precisione singola (float) non può rappresentare esattamente questo numero. Pertanto, quando viene convertito in una rappresentazione in virgola mobile, si verifica un errore di arrotondamento. Quando si eseguono operazioni con numeri approssimati in questo modo, possono verificarsi accumuli di errori di arrotondamento, che possono influire sui risultati dei calcoli.

**Cosa si intende per numero de-normalizzato? Qual'è la condizione**

**necessaria e sufficiente perché la conversione di un numero in virgola**

**mobile in binario sia esatta? Perchè è necessario codificare un numero?**

Un numero de-normalizzato è un numero in virgola mobile che non segue la normale convenzione di rappresentazione, in cui la mantissa è normalizzata (cioè normalmente compresa tra 1 e 2). I numeri de-normalizzati sono utilizzati per rappresentare valori molto piccoli o molto grandi che cadono al di fuori della normale gamma di rappresentazione.

La condizione necessaria e sufficiente perché la conversione di un numero in virgola mobile in binario sia esatta è che il numero sia rappresentabile con il numero di bit disponibili per l'esponente e la mantissa. Se il numero ha una parte frazionaria che non può essere rappresentata completamente, verrà effettuata una troncatura o un arrotondamento, il che può portare a una perdita di precisione.

È necessario codificare un numero per rappresentarlo in una forma specifica che possa essere elaborata da un sistema digitale. La codifica consente di convertire un valore numerico in una sequenza di bit che può essere interpretata correttamente dal sistema, consentendo operazioni matematiche e manipolazioni di dati. La scelta della codifica dipende dall'applicazione specifica e dalle esigenze di rappresentazione, precisione e gamma dei valori.

# IEEE754 a decimale

sabato 20 luglio 2024 08:58

1. [4] Convertire in decimale il numero binario, codificato secondo la codifica IEEE754: 1 1000 0000 0100 0000 0000 0000 0000  
000. -> **-2,5**

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero ottenuto? **1 0111 1111 1000 0000 0000 0000 0000 0000 000**

Quale sarà la rappresentazione binaria del numero decimale ottenuto nel Google Brain Format?

Quale coppia di numeri codificati sequenzialmente in IEEE754 (variazione del bit meno significativo) dista esattamente una unità?

**0011111100000000000000000000000000000000**

**0011111100000000000000000000000000000001**

Qual è la risoluzione della codifica in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

**IEEE754 -> 2^-23**

**INTERA -> 1**

**VIRGOLA FISSA -> 2^F (F NUMERO DI BIT PER PARTE FRAZIONARIA)**

Scrivere in complemento a 2 su 16 bit la sottrazione espressa da numeri in base 10: 12-16 e calcolarne il risultato in binario.

**12 in binario su 16 bit è: 0000 0000 0000 1100**

**16 in binario su 16 bit è: 0000 0000 0001 0000**

**Complemento a 2 di 16 è: 1111 1111 1111 0000**

Sommo 12+(-16) :

**0000 0000 0000 1100 +**

**1111 1111 1111 0000**

-----

**1111 1111 1111 1100**

Il risultato binario **1111 1111 1111 1100** è un numero negativo in complemento a 2.

Per trovare il valore decimale:

- Invertire tutti i bit e aggiungere 1: 0000 0000 0000 0100

• Quindi, il valore decimale è 4. Poiché il bit di segno era 1, il numero originale è **-4**

1. [4] Convertire in decimale, il numero binario in virgola mobile codificato mediante la codifica IEEE 754: 1100 0110 0000 0000 0000 0000 0000 0000. -> **-8192**

Quale sarà la rappresentazione decimale del numero binario in virgola mobile codificato successivamente.

**11000110000000000000000000000001 -> -8192.001**

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 8-10 e calcolarne il risultato in binario.

**8 in binario su 8 bit è: 0000 1000**

**10 in binario su 8 bit è: 0000 1010**

**il complemento a 2 di 1010 è: 1111 0110**

Sommare 8 con il complemento a 2 di 10:

**0000 1000 +**

**1111 0110**

-----

**1111 1110**

Il risultato binario **1111 1110** è un numero negativo in complemento a 2.

Per trovare il valore decimale:

- Invertire tutti i bit e aggiungi 1: 0000 0010

• Quindi, il valore decimale è 2. Poiché il bit di segno era 1, il numero originale è **-2**.

1. [4] Convertire il numero binario 1100 0000 1110 0000 0000 0000 0000 0000, codificato secondo lo standard IEEE754 in un numero decimale. -> **-7**

Quale sarà il numero codificato successivo nello standard IEEE754?

**1100 0000 1110 0000 0000 0000 0000 0001**

A partire dalla codifica binaria, scrivere la codifica esadecimale dello stesso numero.

Passaggi:

Rappresentare 7 in binario su 8 bit. **0000 0111**



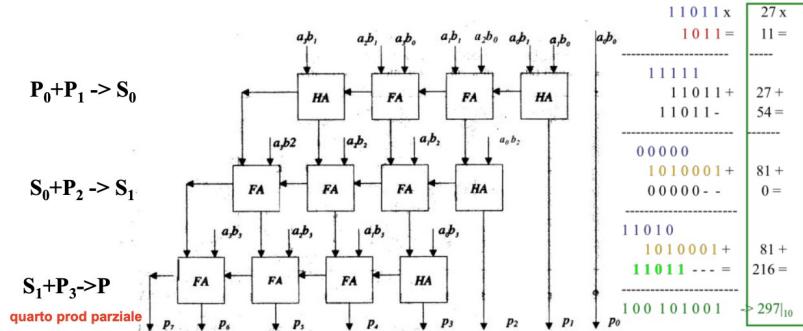
011111101111111111111111111111111 (3.4028235e38)

# Moltiplicazione hardware

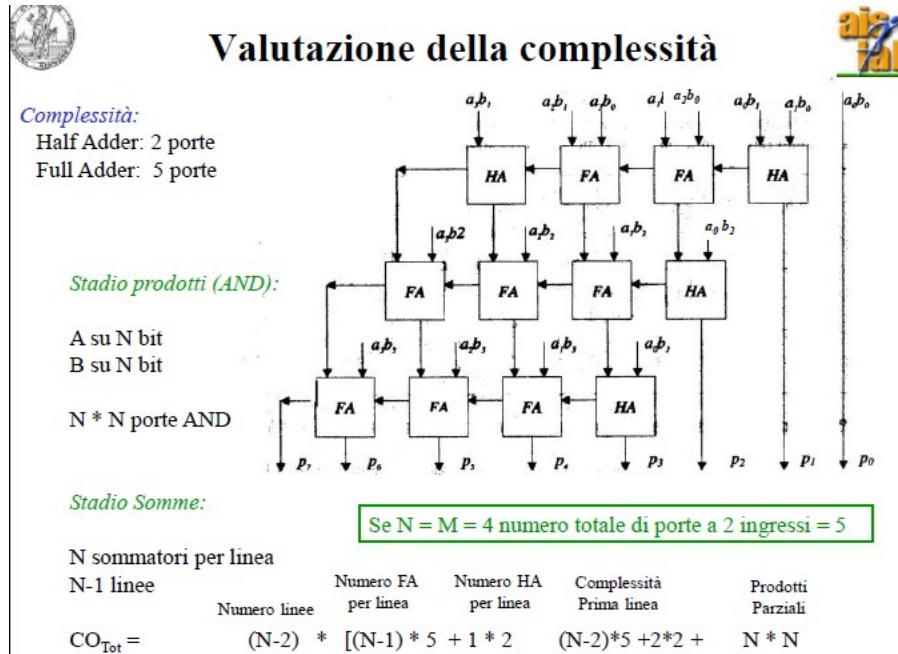
sabato 20 luglio 2024 09:33

## LEZIONE 7

### MOLTIPLICATORE HW A e B SU 4 BIT



Problema: A e B su 4 bit  $\Rightarrow$  P su 8 bit (prodotto su  $2N$  bit)



Complessità

$$t = (N-2) * [(N-1) * 5 + 1 * 2] + (N-2) * 5 + 2 * 2 + N * N$$

Complessità per parole su 4 bit =  $2 * (3 * 5 + 2) + 14 + 16 = 64$  porte a due ingressi



# Valutazione del cammino critico



## Cammini critici:

Half Adder:

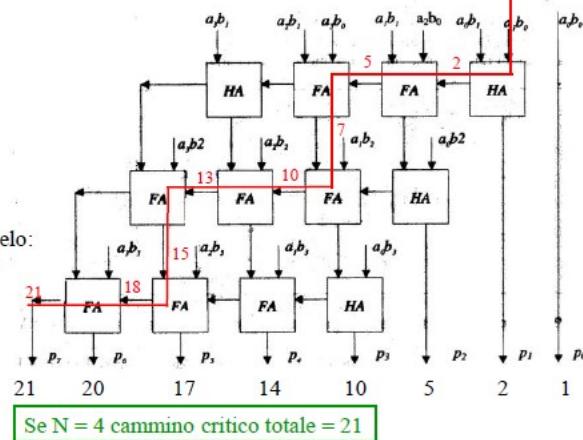
- Somma - 1 porta
- Riporto - 1 porta

Full Adder:

- Somma - 2 porte
- Riporto - 3 porte

Gli AND operano in parallelo:  
ritardo 1.

HA e FA non sono  
equivalenti per il  
cammino critico



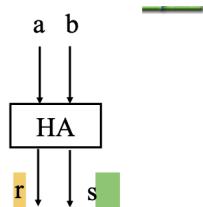
## HALF ADDER

Tabella della verità della somma:

non considero riporto in ingresso

a	b	somma	riporto
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

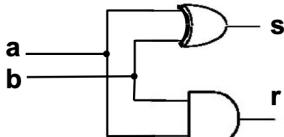
$$a + b =$$



a	b	xor
0	0	0
0	1	1
1	0	1
1	1	0

$$s = a \oplus b \text{ XOR}$$

$$r = ab \text{ AND}$$



La somma è diventata un'operazione logica!

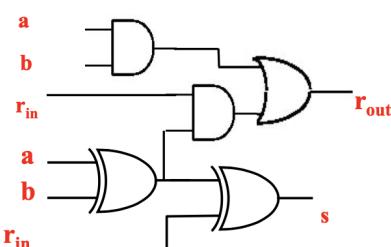
Cammini critici:  
Somma = 1;  
Riporto = 1;

Complessità  
Somma = 1 porta;  
Riporto = 1 porta;

## FULL ADDER

$$s = a \oplus b \oplus r_{in}$$

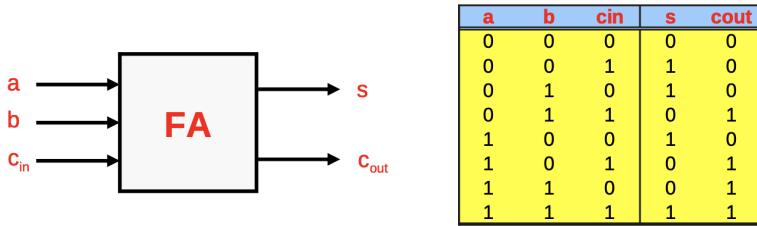
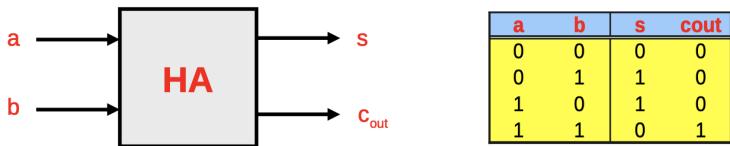
$$r_{out} = ab + (a \oplus b) r_{in}$$



5 porte logiche.  
Cammini critici: s → 2; r\_out → 3

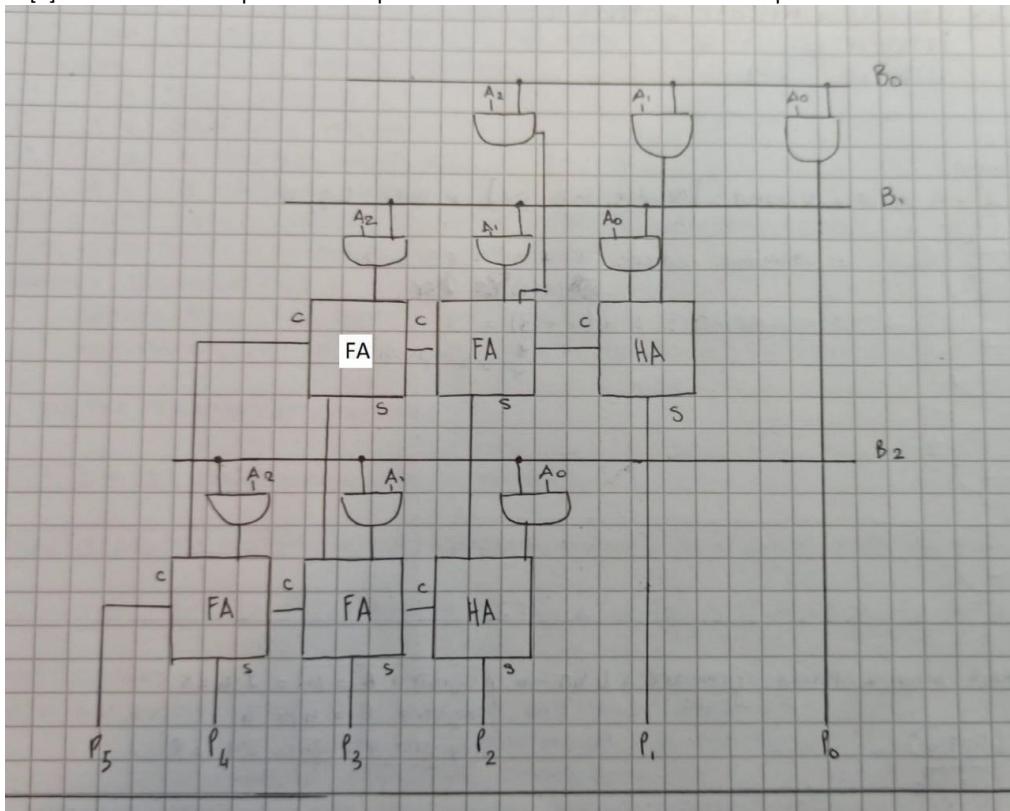
CIRCUITO FINALE  
FULL HADDER

s - rilevatore di (dis)parità VALE 1 QUANDO O 1 O 3 BIT IN INGRESSO SONO UGUALI AD 1  
r\_out - riporto se generato (a = b = 1) o se propagato (a ⊕ b = 1) r\_out = r\_in

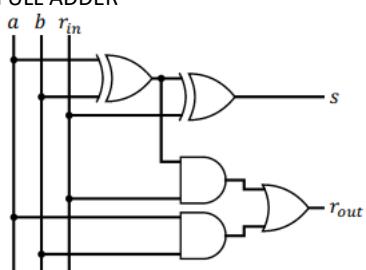


7. [3] Progettare con le porte logiche un moltiplicatore hardware per numeri interi su 2 bit. Calcolare complessità e cammino critico.

6. [2] Costruire un moltiplicatore HW per numeri interi su 3 bit. Calcolare complessità e cammino critico.



FULL ADDER

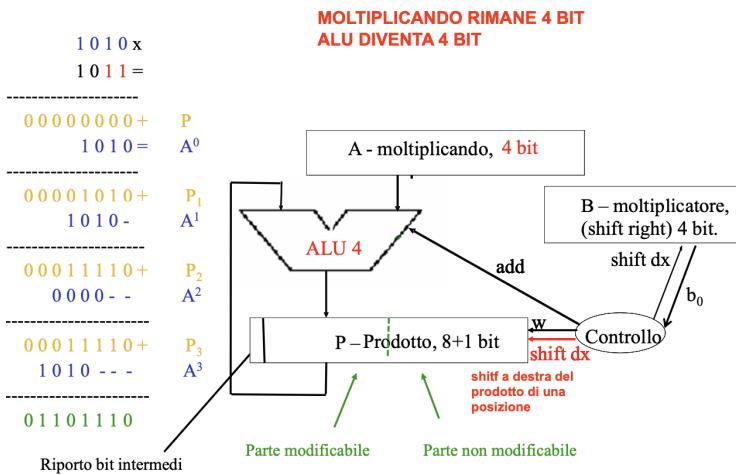


5. [3] Costruire con porte logiche un moltiplicatore hardware a 3 bit. Qual è la sua complessità? E il cammino critico?

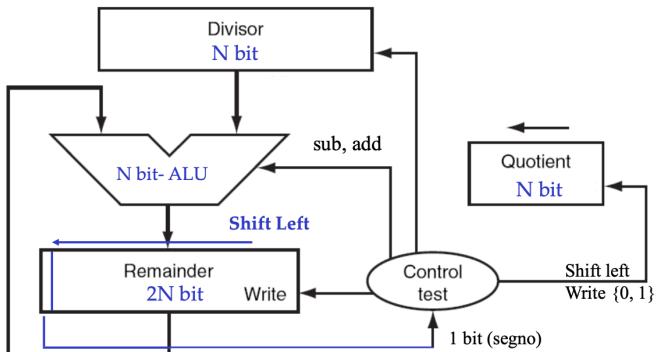
# Moltiplicazione firmware

sabato 20 luglio 2024 09:28

2. [3] Disegnare il diagramma di flusso di un algoritmo della moltiplicazione binaria intera firmware a 4 bit, e implementarlo in un circuito contenente tre registri: 1 registro moltiplicando a 8 bit, 1 registro moltiplicatore a 4 bit e 1 registro risultato a 8 bit. Evidenziare tutti i cammini relativi ai data path, dimensionarli e definire la loro funzione.



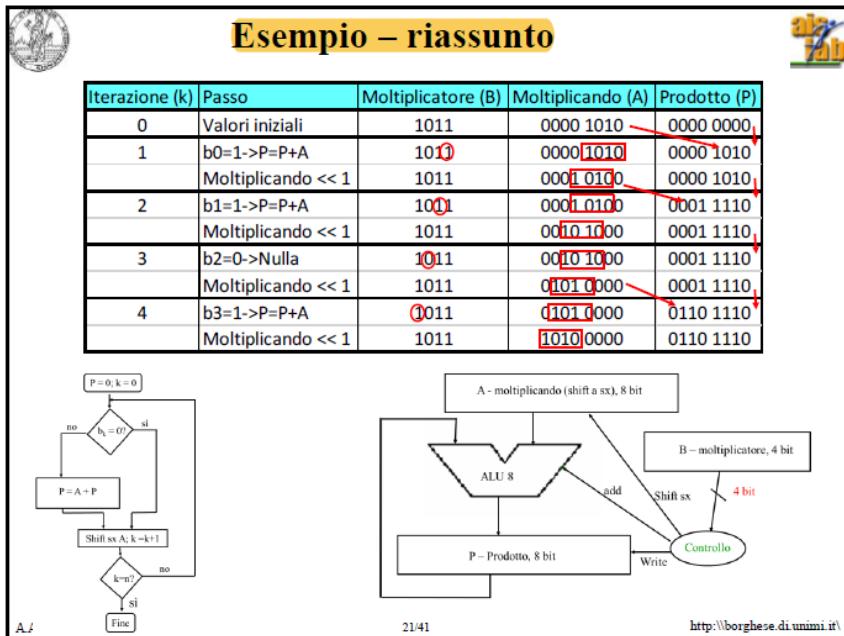
3. [4] Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 2, mantenendo i 3 registri specificati, per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Quali segnali di controllo occorre aggiungere? Motivare le modifiche e definire chiaramente la loro funzione. Mostrare come varia il contenuto di tutti e 3 i registri durante i primi 2 passi di esecuzione della divisione 7:2.



Il quoziente viene riempito un bit alla volta da dx a sx

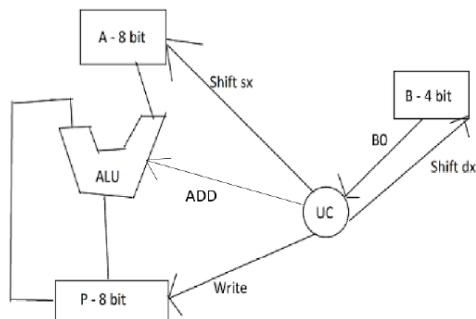
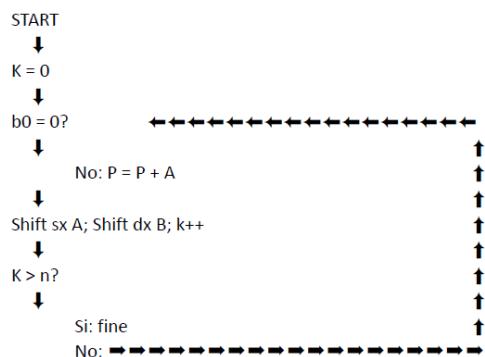
Il dividendo viene spostato da dx a sx di una posizione

0000 1010 X  
1011



4. [5] Costruire il circuito (FSM) che implementa l'unità di controllo del circuito costruito al punto 2. Calcolare complessità e cammino critico. In quanti cicli di clock viene ultimata una moltiplicazione? Semplificare per via analitica le funzioni di stato prossimo.

2. [4] Scrivere un algoritmo della moltiplicazione binaria intera firmware a 4 bit, e implementarlo in un circuito contenente tre registri: 1 registro moltiplicando a 8 bit, 1 registro moltiplicatore a 4 bit e 1 registro risultato a 8 bit. Evidenziare tutti i cammini relativi al data path, dimensionarli e definire la loro funzione.

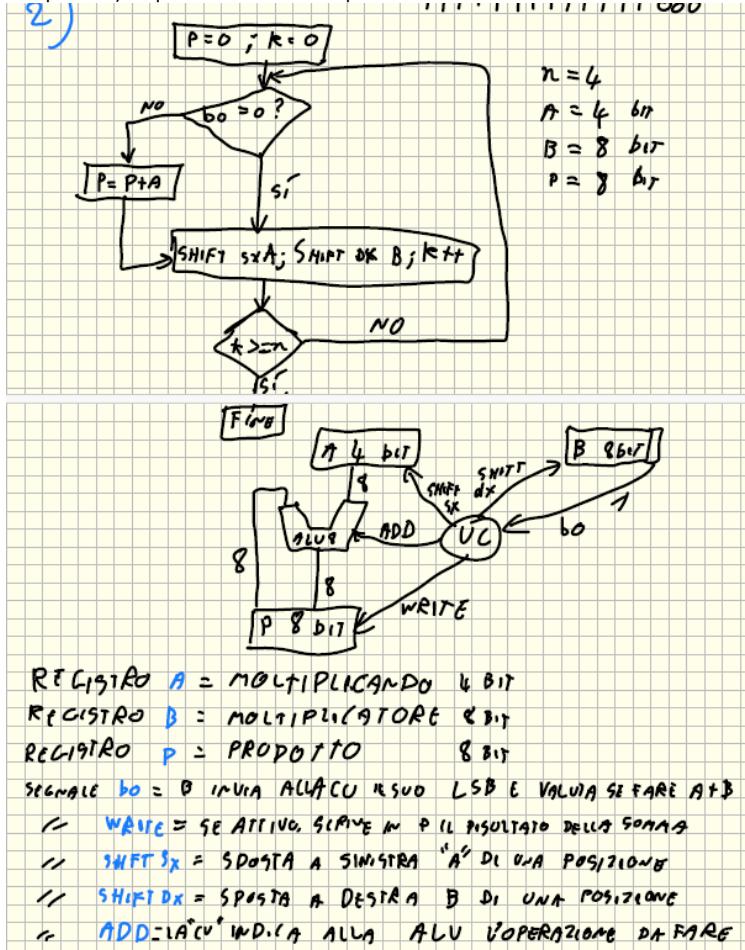


3. [4] Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 3 per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Quali segnali di controllo occorre aggiungere? Motivare le modifiche e definire chiaramente la loro funzione. Riportare l'algoritmo della divisione per questo circuito e, per i primi due passi della divisione 9 : 2, riportare il contenuto dei registri all'inizio e all'fine di ogni passo.

3. [4] Scrivere un algoritmo della moltiplicazione binaria intera firmware a 4 bit, con un circuito contenente tre registri: 1 registro moltiplicando a 4 bit, 1 registro moltiplicatore a 8 bit e 1 registro risultato a 8 bit. Evidenziare tutti i cammini relativi al data path, dimensionarli e definire la loro funzione. Quale sarà il cammino critico di questo circuito? (considerare pari a  $3^*N$  il cammino critico di un sommatore e pari a  $5^*N$  la

complessità). E quale sarà la sua complessità?

2)



4. [4] Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 3 e quali segnali di controllo debbano essere aggiunti per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Motivare le modifiche e definire chiaramente la loro funzione. Riportare l'algoritmo della divisione per questo circuito e, per i primi due passi della divisione  $8 : 2$ , riportare il contenuto dei registri all'inizio e all'fine di ogni passo.

3) REGISTRO A = DIVISORE + n ZERI

REGISTRO B :

REGISTRO P = n ZERI ED n BIT DEL DIVIDENDO  
(RESTO)

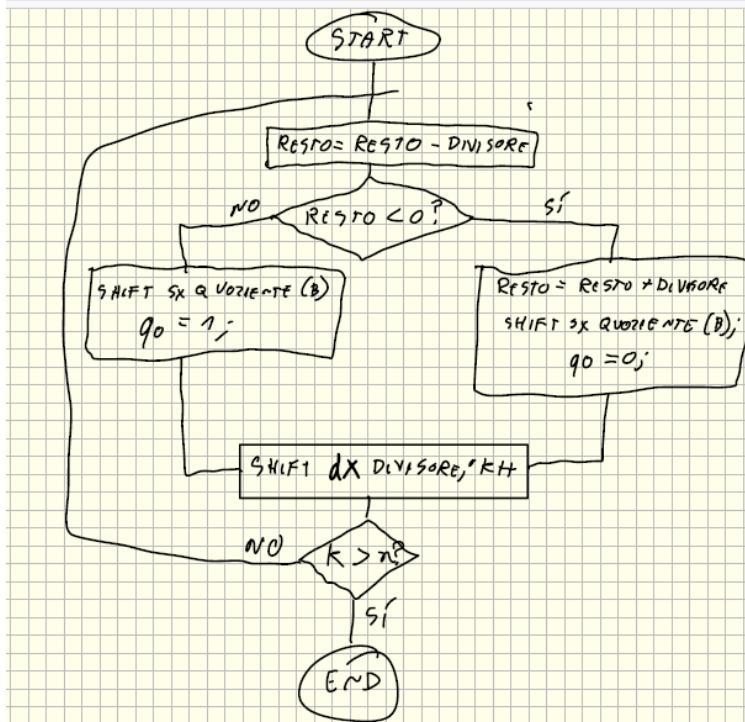
SI AGGIUNGE:

SEGNALI SHIFT DX REGISTRO A: SHIFTA A DESTRA IL DIVISORE

// SHIFT SX REGISTRO B; SHIFTA A SINISTRA IL QUOTIENTE

// SUB ALLA ALU: PERMETTE DI FARE LA SOTTRAZIONE

// DEL BIT DI SEGNO DA "P" ALLA "C": SERVE ALLA CU X SAPERE SE RESTO È NEGATIVO



3. [7] Scrivere un algoritmo della moltiplicazione binaria intera firmware a 4 bit, con registro moltiplicando a 4 bit, registro moltiplicatore a 4 bit e registro risultato a 8 bit. Costruire, con le porte logiche, il circuito firmware associato (datapath e unità di controllo). Evidenziare tutti i cammini, dimensionarli e definire la loro funzione. Quale sarà il cammino critico di questo circuito? (considerare pari a  $3 \times N$  il cammino critico di un sommatore a  $N$  bit). Quale sarà il suo cammino critico? E quale sarà la sua complessità?

4. [2] Descrivere come si possa modificare il datapath e quali segnali di controllo debbano essere aggiunti per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Motivare le modifiche e definire chiaramente la loro funzione.

2. [7] Calcolare mediante un algoritmo firmware la moltiplicazione tra i due numeri interi binari 1100 e 11 su 4 bit. Scrivere l'algoritmo e progettare il circuito firmware associato all'algoritmo scelto. Dimensionare tutti i cammini e definire la loro funzione. Estendere il circuito disegnato per implementare anche la divisione tra numeri interi su 4 bit e spiegarne il funzionamento. E' un circuito sincrono o asincrono? Perché? Progettare l'unità di controllo del circuito firmware che avete disegnato per eseguire la moltiplicazione. Quale sarà il suo cammino critico? E quale sarà la sua complessità?

9. [7] Calcolare mediante un algoritmo binario a vostra scelta, implementabile mediante architettura firmware, la moltiplicazione tra 1001 e 11 rappresentati su parole di 4 bit. Scrivere l'algoritmo utilizzato e progettare il circuito firmware associato all'algoritmo. Estendere il circuito per eseguire anche le divisioni.

3. [8] Scrivere un algoritmo di moltiplicazione firmware binario a piacere e progettare il circuito firmware che implementa l'algoritmo. Costruire la macchina a stati finiti contenuta nell'unità di controllo del circuito disegnato. Come si può estendere il circuito per potere eseguire una divisione? Scrivere l'algoritmo firmware della divisione associata la circuito disegnato precedentemente. Scrivere tutti i passi dell'algoritmo applicato alla divisione 11 : 3, con numeri codificati su 4 bit. Spiegare come si determina il segno del quoziente e del resto in una divisione firmware.

3. [8] Scrivere un algoritmo di moltiplicazione su 6 bit firmware binario a piacere e progettare il circuito firmware che implementa l'algoritmo. Dimensionare tutti i componenti e tutti i bus (cammini) interni. Costruire la macchina a stati finiti contenuta nell'unità di controllo del circuito disegnato. Come si può estendere il circuito per potere eseguire una divisione? Scrivere l'algoritmo firmware della divisione associata la circuito disegnato precedentemente.

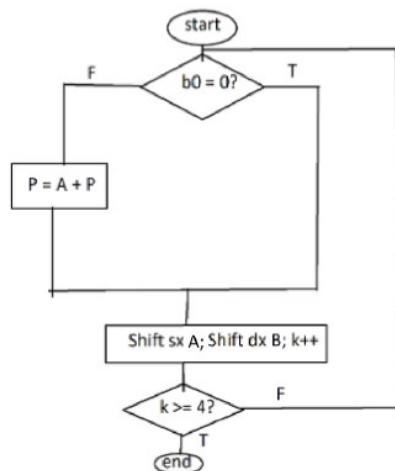
Scrivere tutti i passi dell'algoritmo applicato alla divisione 12 : 4, con numeri codificati su 4 bit. Spiegare come si determina il segno del quoziente e del resto in una divisione firmware.

2. [4] Scrivere un algoritmo di moltiplicazione firmware binario e progettare il circuito firmware che implementa l'algoritmo. Scrivere tutti i passi dell'algoritmo applicato alla moltiplicazione 11 x 3 su 4 bit. Come si può estendere il circuito per potere eseguire una divisione?

2. [3] Disegnare il diagramma di flusso di un algoritmo della moltiplicazione binaria intera firmware a 4 bit, e implementarlo in un circuito contenente tre registri: 1 registro moltiplicando (primo fattore) a 4 bit, 1 registro moltiplicatore (secondo fattore che conterrà il moltiplicatore) a 8 bit e 1 registro risultato a 8 bit. Cosa contengono i 3 registri? Evidenziare tutti i cammini relativi al data path, dimensionarli e definire la loro funzione. Quanti cicli di clock sono necessari per completare l'operazione? Motivare la risposta.

3. [4] Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 2, mantenendo i 3 registri specificati, per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Cosa contengono i 3 registri? Quali segnali di controllo occorre aggiungere? Motivare le modifiche e definire chiaramente la loro funzione. Mostrare come varia il contenuto di tutti e 3 i registri durante i primi 2 passi di esecuzione della divisione 9 : 4.

2. [3] Disegnare il diagramma di flusso di un algoritmo della moltiplicazione binaria intera firmware a 4 bit, e implementarlo in un circuito contenente tre registri: 1 registro moltiplicando a 8 bit, 1 registro moltiplicatore a 4 bit e 1 registro risultato a 8 bit. Evidenziare tutti i cammini relativi al data path, dimensionarli e definire la loro funzione.

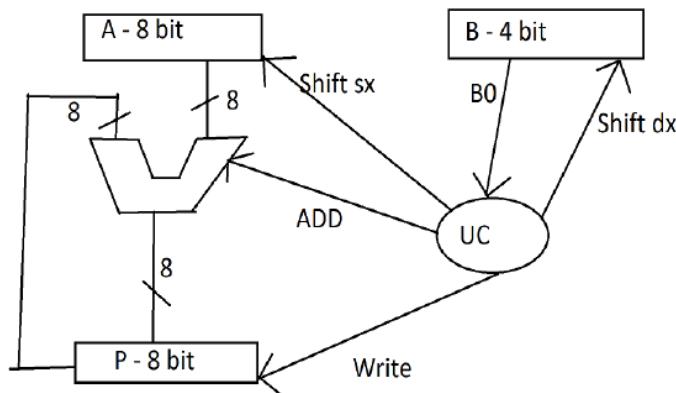


$n = 4$  bit

A = moltiplicando =  $2n$  bit

B = moltiplicatore = n bit

P = risultato =  $2n$  bit



UC: gestisce le operazioni da fare, mandando i segnali opportuni ai componenti.

A: contiene il moltiplicando. Passa alla ALU il suo contenuto.

B: contiene il moltiplicatore. Manda all'UC il bit meno significativo (b0).

P: contiene il prodotto della moltiplicazione. Passa alla ALU il suo contenuto.

B0: è il bit meno significativo del moltiplicatore.

Segnale Write: permette alla ALU di scrivere dentro P.

Segnale ADD: indica alla ALU di fare la somma.

Shift SX: indica al registro B di shiftare a sinistra.

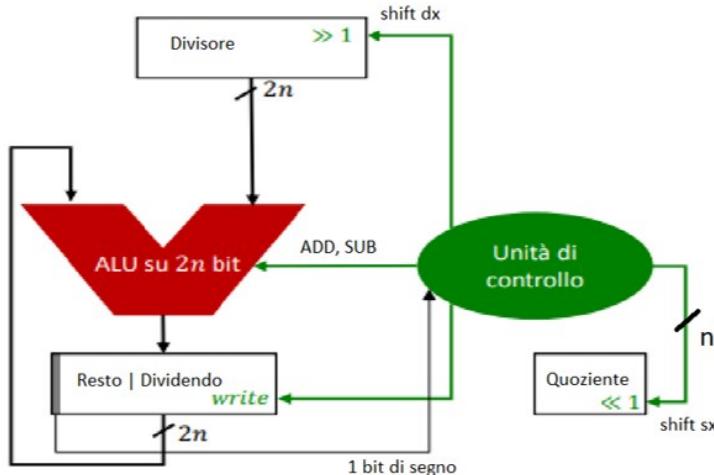
Shift DX: indica al registro A di shiftare a destra.

3. [4] Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 2, mantenendo i 3 registri specificati, per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Quali segnali di controllo occorre aggiungere? Motivare le modifiche e definire chiaramente la loro funzione. Mostrare come varia il contenuto di tutti e 3 i registri durante i primi 2 passi di **esecuzione della divisione 7 : 2**.

**Registro A diventa il divisore|zeri, registro B diventa il quoziente e Registro P diventa resto|dividendo.**

Al datapath bisogna aggiungere:

- shift dx al registro divisore
- shift sx al registro quoziente (serve per poter avere uno zero in fondo a sinistra che in caso sarà sostituito da un 1)
- bit di segno che va dal registro Resto alla UC (serve a sapere se il resto è positivo o negativo)
- segnale di SUB per la ALU (serve a fare la differenza tra divisore e resto)



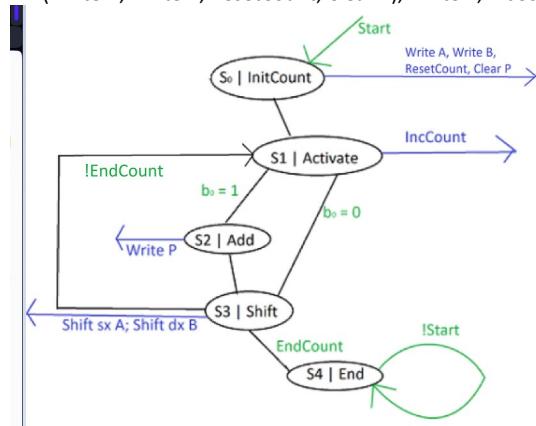
4. [5] Costruire il circuito (FSM) che implementa l'unità di controllo del circuito costruito al punto 2. Calcolare complessità e cammino critico. In quanti cicli di clock viene ultimata una moltiplicazione? Semplificare per via analitica le funzioni di stato prossimo.

X = InitCount, Activate, Add, Shift, End = {000, 001, 010, 011, 100}

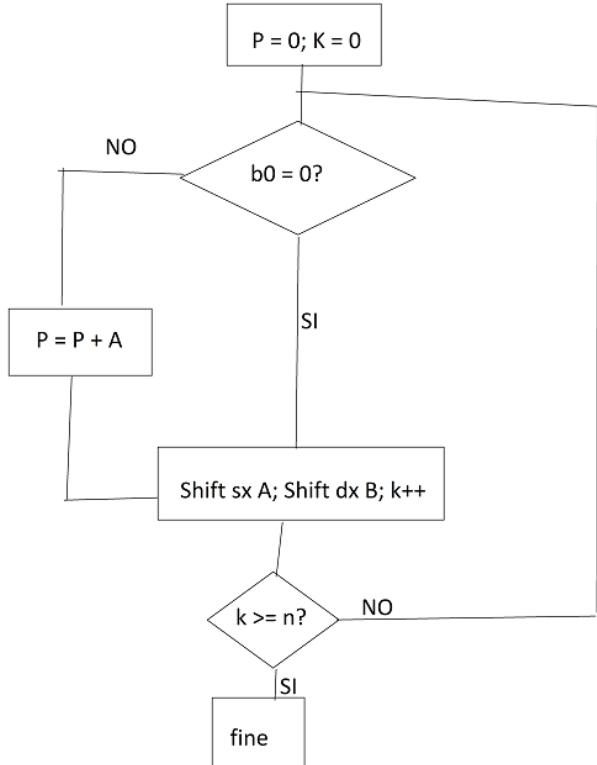
X0 = InitCount

I = b0, EndCount = {00, 01}

Y = (Write A, Write B, ResetCount, Clear P), Write P, IncCount, (Shift sx A, Shift dx B) = {000, 001, 010, 011}



2. Scrivere un algoritmo della moltiplicazione binaria intera firmware a 4 bit, e implementarlo in un circuito contenente tre registri: 1 registro moltiplicando a 8 bit, 1 registro moltiplicatore a 8 bit e 1 registro risultato a 8 bit. Evidenziare tutti i cammini relativi al data path, dimensionarli e definire la loro funzione.

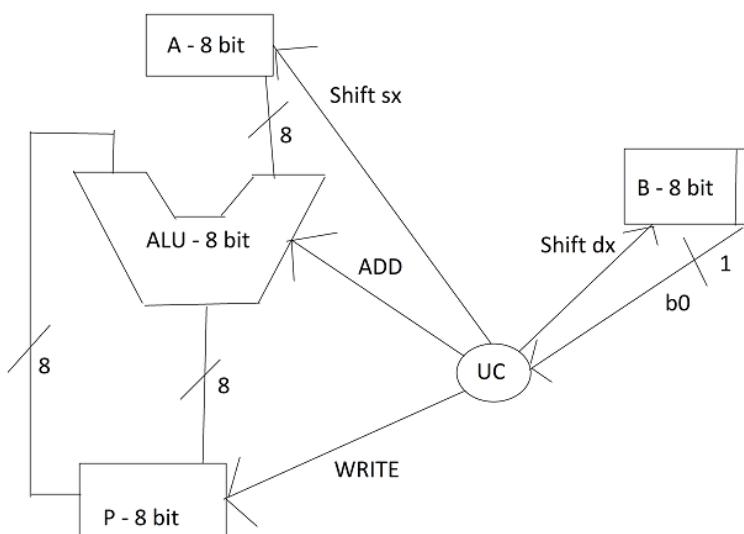


$n = 4$  bit

$A = 2n$  bit

$B = 2n$  bit

$P = 2n$  bit



Registro A =  $2n$  bit, contiene il moltiplicando

Registro B =  $2n$  bit, contiene il moltiplicatore

Registro P =  $2n$  bit, conterrà il prodotto

Segnale  $b_0$  = il registro B invia alla UC il suo bit meno significativo che serve per sapere se fare la somma A+P

Segnale Write = se attivo, consente alla ALU di scrivere nel registro P il risultato della somma

Segnale Shift sx = shifta a sinistra di una posizione il registro A

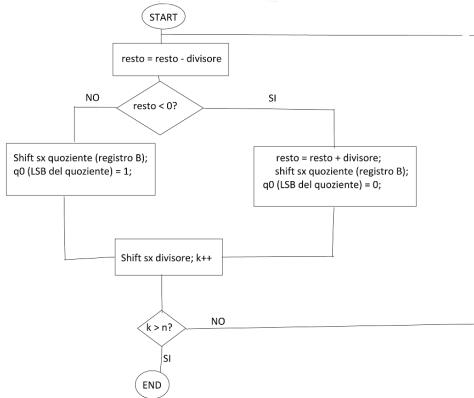
Segnale Shift dx = shifta a destra di una posizione il registro B

Segnale ADD = indica alla ALU che operazione deve fare

3. Descrivere come si possa modificare il datapath del circuito disegnato per l'esercizio 3 per eseguire anche l'operazione di divisione intera di numeri su 4 bit. Quali segnali di controllo occorre aggiungere? Motivare le modifiche e definire chiaramente la loro funzione. Riportare l'algoritmo della divisione per questo circuito e, per i primi due passi della divisione **7 : 2**, riportare il contenuto dei registri all'inizio e alla fine di ogni passo.

Bisogna aggiungere:

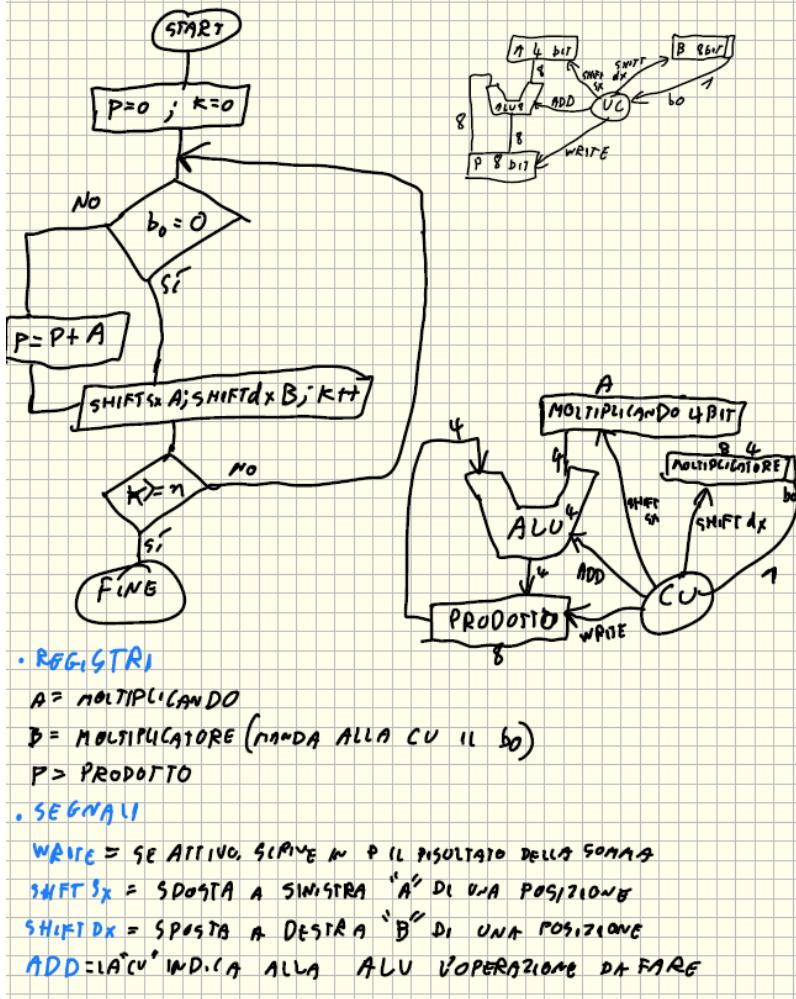
- Segnale shift dx registro A: servira' a shiftare a destra il divisore.
  - Segnale shift sx registro B: servira' a shiftare a destra il quoziente.
  - Segnale di SUB alla ALU: permette di fare la differenza.
  - Segnale del bit di segno dal registro P alla UC: serve alla UC per verificare il resto e' negativo.
- Il registro A conterra' n bit del divisore e n zeri.  
 Il registro B conterra' n zeri all'inizio e poi conterra' il quoziente.  
 Il registro P conterra' n zeri e n bit del dividendo.



Iterazione	Step	Quoziente	Divisore	Resto
0	inizializzo	0000 0000	0001 0000	0000 0111
1	resto = resto - div	0000 0000	0001 0000	1111 0111
	resto < 0 → resto += div; Shift sx quoziante; Q <sub>0</sub> = 0	0000 0000	0001 0000	0000 0111
	shift dx div	0000 0000	0000 1000	0000 0111
2	resto = resto - div	0000 0000	0000 1000	1111 1111
	resto < 0 → resto += div; Shift sx quoziante; Q <sub>0</sub> = 0	0000 0000	0000 1000	0000 0111
	shift dx div	0000 0000	0000 0100	0000 0111

Riassunto:

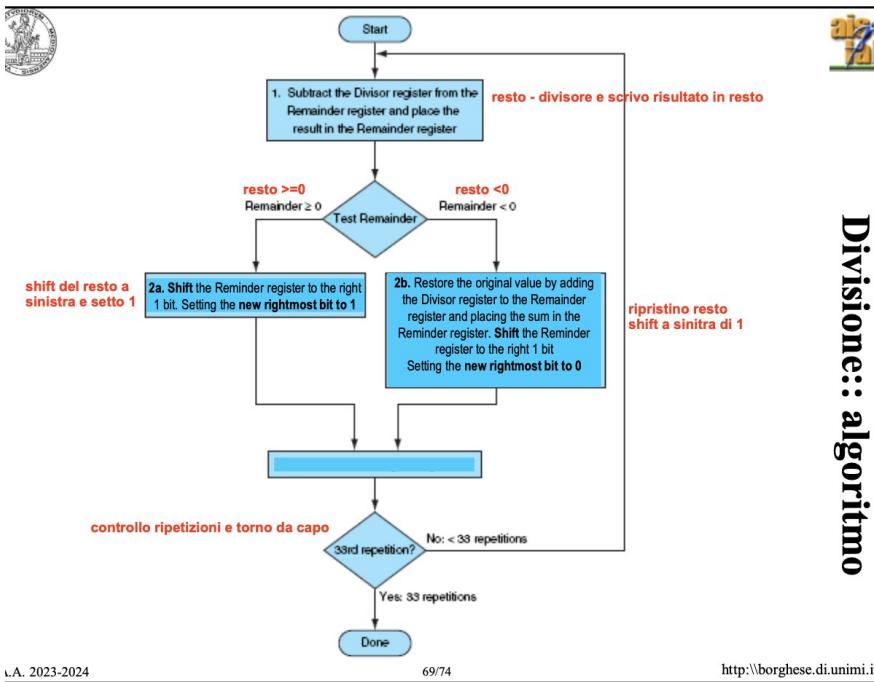
## MOLTIPLICAZIONE



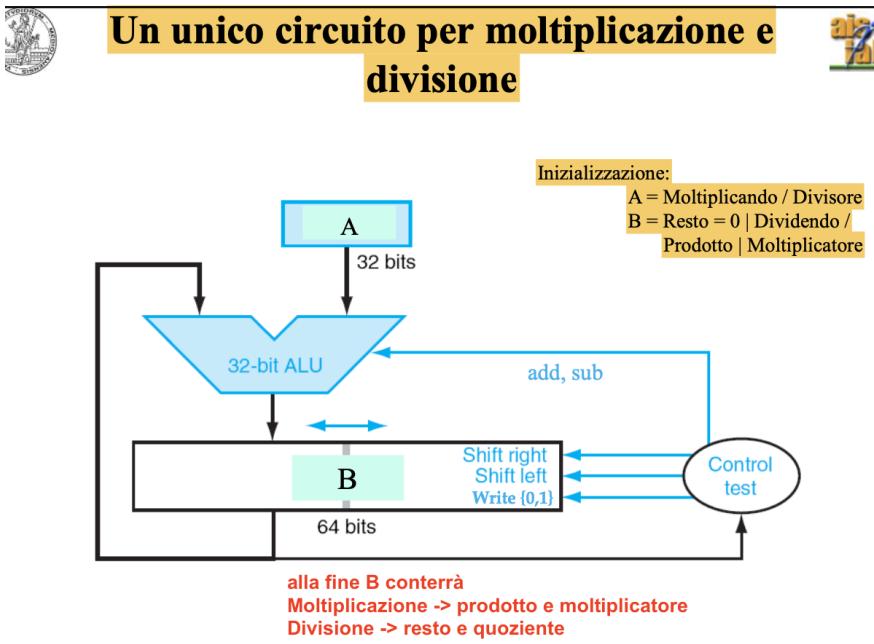
# Divisione firmware

sabato 20 luglio 2024 10:00

## LEZIONE 12



## Divisione:: algoritmo



2. [7] Calcolare mediante un algoritmo binario la divisione tra 1010 e 11 su 4 bit. Scrivere l'algoritmo e progettare il circuito firmware associato all'algoritmo scelto. Dimensionare tutti i cammini. Estendere il circuito disegnato per

implementare anche la moltiplicazione e spiegarne il funzionamento. E' un circuito sincrono o asincrono? Quale sarà il suo cammino critico?

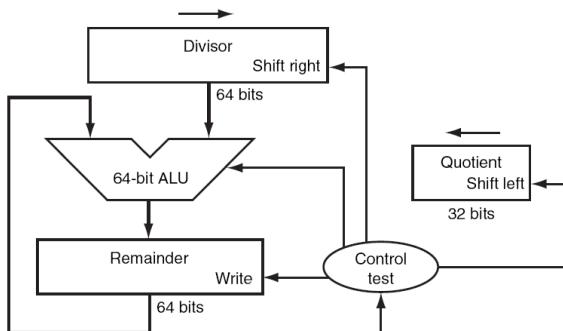
2. [6] Calcolare mediante un algoritmo a vostra scelta la divisione tra 1101 e 101 (13:5) e scrivere l'algoritmo utilizzato. E' possibile stabilire il numero di passi in cui viene eseguita? Perché? Progettare il circuito firmware associato all'algoritmo. Definire la dimensione di tutti i bus interni e dei registri.

3. [3] Estendere il circuito disegnato per eseguire anche le moltiplicazioni. Definire i criteri con cui si progetta l'unità di controllo di questo circuito firmware. Sarà un circuito combinatorio o sequenziale? Perché?

2. [8] Calcolare mediante un algoritmo a vostra scelta la divisione tra 1001 e 101 (9:5) e scrivere l'algoritmo utilizzato. Progettare il circuito firmware associato all'algoritmo. Definire la dimensione di tutti i bus interni. Estendere il circuito disegnato per eseguire anche le moltiplicazioni.

3. [5] Disegnare il circuito di controllo di quest'ultimo (Suggerimento: si tratta di una macchina a stati finiti. Definire gli ingressi, gli stati e le uscite; definire lo state transition graph, la state transition table e da qui la macchina di Huffman). Calcolate il cammino critico e la complessità dell'unità di controllo realizzata.

3. [6] Scrivere l'algoritmo della divisione che può essere implementato nel circuito riportato qui sotto. Riportare il contenuto di tutti i registri in tutti i passi dell'algoritmo definito, nel calcolo della divisione tra 1110 e 11. Il circuito sotto è un circuito sequenziale o combinatorio? Perché? Estendere il circuito per implementare anche la moltiplicazione.



3. [6] Calcolare mediante un algoritmo binario la divisione tra 1010 e 11. Scrivere l'algoritmo e progettare un circuito firmware che consente di eseguire la divisione. Estendere il circuito per svolgere anche una moltiplicazione. Si può estendere il circuito per eseguire anche la somma in virgola mobile? Perché?

2. [7] Calcolare mediante un algoritmo binario la divisione tra 1110 e 11 su 4 bit. Scrivere l'algoritmo e progettare il circuito firmware associato. Dimensionare tutti i cammini. Estendere il circuito per implementare anche la moltiplicazione.

## DIVISIONE 7:2

## DIVISIONE

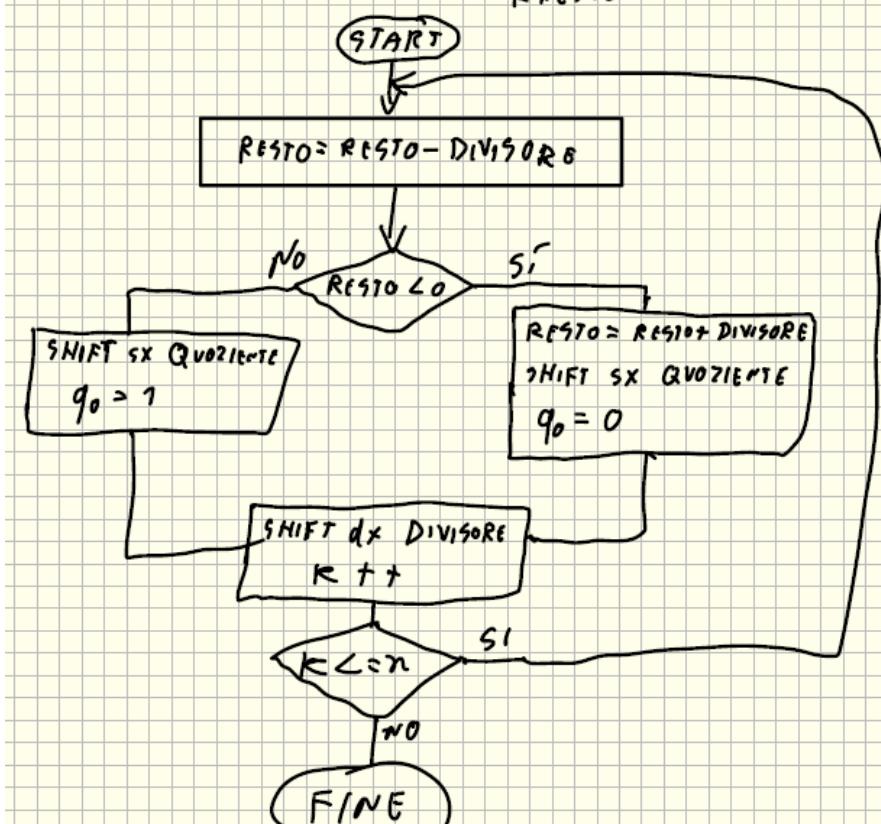
$$7 : 2 = 3$$

↓  
 1  
 ↓  
 0  
 ↓  
 0111

$$0111 - 10 = 011$$

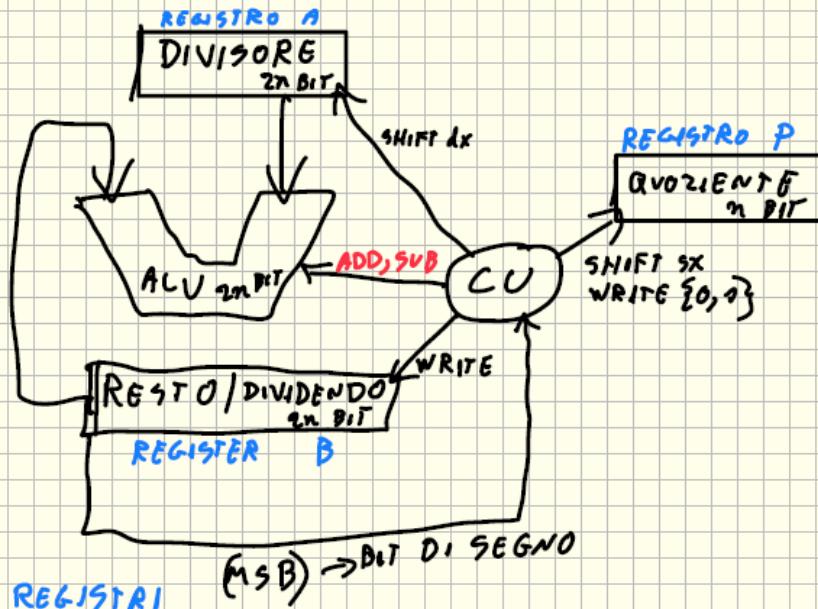
10  
 11 -  
 10  
 11 -  
 10

TERRESTO



ITERAZIONE	STEP	QUOTANTE	DIVISORE	RESTO
0	INITIALIZZAZIONE	0000	0010 0000	0000 0111
	R = R - DIV.	0000	0010 0000	1110 0111
	R < 0 ?			
	↓ T			
1	R = R + DIV			
	SHIFT SX QUOT.			
	Q <sub>0</sub> = 0			
	SHIFT SX DIV	0000	0001 0000	0000 0111
	R = R - DIV.	0000	0001 0000	1111 0111
2	R < 0 ?			
	↓ T			
	R = R + DIV			
	SHIFT SX QUOT.			
	Q <sub>0</sub> = 0			
	SHIFT SX DIV	0000	0000 1000	0000 0111
3	← T	0000	0000 1000	1111 1111
		0000	0000 0100	0000 0111
	R = R - DIV	0000	0000 0100	0000 0011
	R < 0			
	↓ F			
4	SHIFT SX QUOT.			
	q <sub>0</sub> = 1			
	SHIFT SX DIV	0001	0000 0010	0000 0011
5	// F	0001	0000 0010	0000 0001
		0011	0000 0001	0000 0001

## CIRCUITO DIVISIONE E MOLTIPLICAZIONE



### • NEW REGISTRI

$A = \text{DIVISORE} + n \text{ ZERI}$  / MOLTIPLICANDO  
 $B = \text{RESTO} [n \text{ ZERI} + \text{DIVIDENDO}]$  / MOLTIPLICATORE  
 $P = \text{QUOTIENTE}. [n \text{ ZERI ALL'INIZIO}]$  / PRODOTTO

### • NEW SEGNALI

SHIFT AX = AL REGISTRO A

SHIFT SX = AL QUOTIENTE

BIT DI SEGNO = MSB DEL REGISTRO B

SWUB = LA CU COMUNICA ALL'ALU DI FARE LA SOTTRAZIONE

# Somma firmware

sabato 20 luglio 2024 09:39

12. [2] Spiegare come viene effettuata la somma in virgola mobile tra due numeri, in una ALU Firmware e disegnare il diagramma di flusso dell'algoritmo.

## Somma di Numeri in Virgola Mobile

Quando sommi due numeri in virgola mobile, è importante gestire correttamente le loro componenti (segno, esponente e mantissa) e assicurarti che il risultato sia normalizzato. Ecco come avviene il processo:

### Passaggi Dettagliati per la Somma in Virgola Mobile

#### 1. Estrazione delle Parti

Ogni numero in virgola mobile è composto da tre parti:

- **Segno (S)**: Un bit che indica se il numero è positivo (0) o negativo (1).
- **Esponente (E)**: Determina l'ordine di grandezza del numero.
- **Mantissa (M)**: Contiene i bit significativi del numero.

Per esempio, in IEEE 754 a precisione singola (32 bit):

- 1 bit per il segno.
- 8 bit per l'esponente.
- 23 bit per la mantissa (con un bit隐式的 di 1).

#### 2. Allineamento degli Esponenti

Prima di poter sommare le mantisse, gli esponenti dei due numeri devono essere uguali. Se gli esponenti sono diversi, devi fare uno dei seguenti:

- **Shiftare la Mantissa**: Shiftare la mantissa del numero con l'esponente più piccolo verso destra, fino a quando entrambi gli esponenti sono uguali. Questo comporta una perdita di precisione se l'esponente è molto piccolo.
- **Aggiornare l'Esponente**: L'esponente del numero con la mantissa shiftata viene aggiornato.

#### 3. Somma delle Mantisse

Una volta che le mantisse sono allineate, puoi sommarle. Assicurati di considerare il segno dei numeri:

- Se i numeri hanno lo stesso segno, somma direttamente le mantisse.
- Se i numeri hanno segni diversi, esegui una sottrazione.

#### 4. Normalizzazione

Dopo aver sommato le mantisse, il risultato deve essere normalizzato:

- **Normalizzazione**: Sposta il punto binario a sinistra o a destra in modo che ci sia un solo bit non zero a sinistra del punto binario. Aggiorna l'esponente di conseguenza.
- **Overflow**: Se il risultato produce un bit in più a sinistra, shiftare la mantissa a destra e incrementare l'esponente.

#### 5. Gestione del Riporto

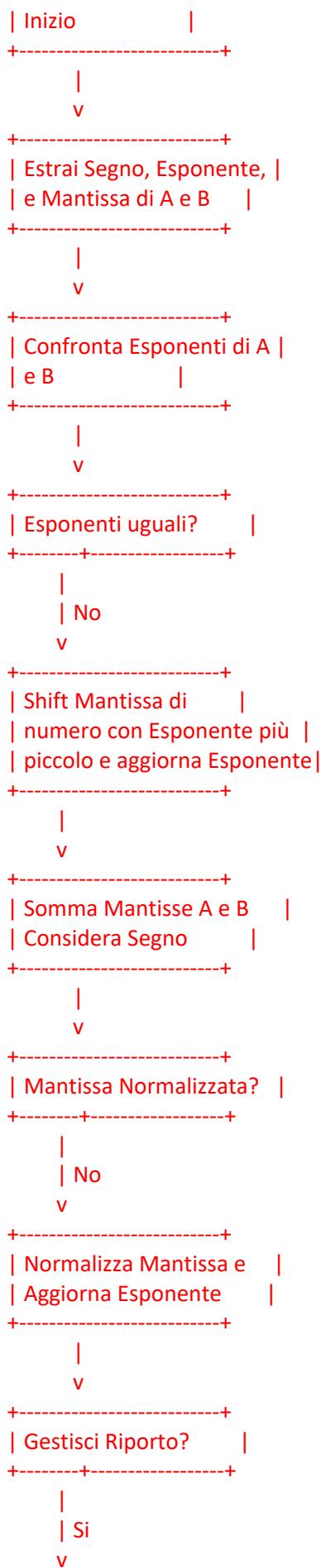
Se la somma delle mantisse produce un riporto (un bit di carry fuori dal range della mantissa):

- **Aggiusta l'Esponente**: Incrementa l'esponente se necessario e gestisci il riporto nella mantissa.

**Esempio**: Se il risultato della somma produce un bit extra, devi normalizzare e incrementare l'esponente.

## Diagramma di Flusso

+-----+



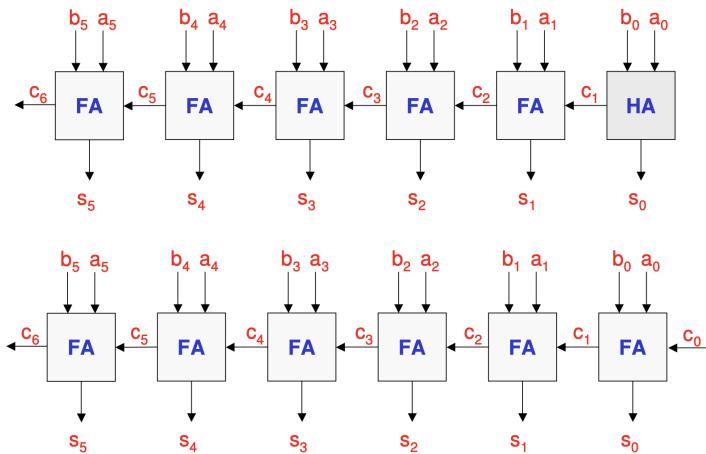
```
+-----+
| Aggiusta Esponente e    |
| Mantissa      |
+-----+
|
| v
+-----+
| Fornisci Risultato    |
| (Segno, Esponente, Mantissa)|
+-----+
|
| v
+-----+
| Fine      |
+-----+
```

# Sommatore propagazione

sabato 20 luglio 2024 09:47

10. [3] Costruire il circuito del sommatore a propagazione di riporto e spiegare chiaramente come il sommatore ad anticipazione di riporto possa velocizzare il calcolo (ridurre il cammino critico) della somma.

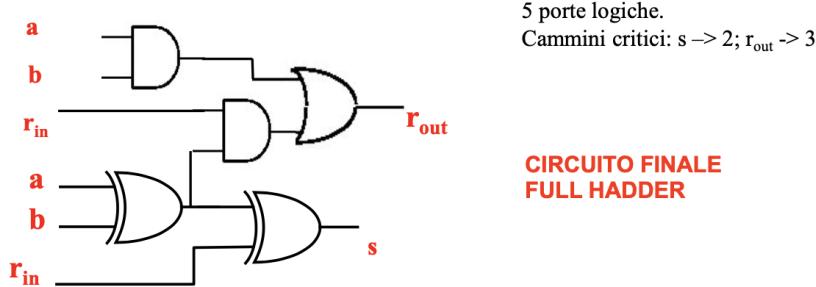
## Sommatore a Propagazione di Riporto



6. [3] Progettare con le porte logiche un sommatore hardware per numeri interi su 2 bit. Calcolare complessità e cammino critico.

$$s = a \oplus b \oplus r_{in}$$

$$r_{out} = ab + (a \oplus b) r_{in}$$



$s$  - rilevatore di (dis)parità **VALE 1 QUANDO O 1 O 3 BIT IN INGRESSO SONO UGUALI AD 1**  
 $r_{out}$  - riporto se generato ( $a = b = 1$ ) o se propagato ( $a \oplus b = 1$ )  $r_{out} = r_{in}$

Il **sommatore a propagazione** di riporto (carry-propagate adder) è un tipo di sommatore in cui i riporti vengono propagati da un bit al successivo. Questo significa che il riporto generato da un bit di somma influenzera il calcolo del riporto per il bit successivo. Il sommatore a propagazione di riporto richiede un tempo di propagazione proporzionale alla lunghezza dei bit di somma, poiché i riporti devono essere propagati attraverso tutti i bit.

# Sommatore anticipazione

sabato 20 luglio 2024 09:33

5. [2] Cosa si intende per sommatore ad anticipazione di riporto e per sommatore a propagazione di riporto.

**Il sommatore a propagazione** È costituito da una serie di sommatori a bit singolo (full adders) collegati in cascata. Ogni sommatore a bit singolo prende tre input: due bit da sommare ( $A$  e  $B$ ) e un bit di riporto in ingresso ( $Cin$ ). Produce due output: la somma del bit ( $S$ ) e un bit di riporto in uscita ( $Cout$ ).

**Il sommatore ad anticipazione** Utilizza una logica più complessa per calcolare il riporto in modo anticipato, senza dover aspettare che il riporto si propaghi attraverso ogni singolo bit. La caratteristica principale di un sommatore ad anticipazione di riporto è che riduce notevolmente il ritardo complessivo dell'operazione di addizione, poiché il riporto viene calcolato in parallelo invece di propagarsi sequenzialmente attraverso tutti i bit.

il sommatore ad anticipazione di riporto (carry-lookahead adder) è un tipo di sommatore che utilizza una tecnica che consente di generare in anticipo i segnali di riporto per i bit di somma.

Invece di propagare i riporti da un bit all'altro, il sommatore ad anticipazione di riporto calcola i segnali di riporto per ciascun bit utilizzando circuiti logici complessi. Questa tecnica riduce il tempo di propagazione, poiché i segnali di riporto sono disponibili in anticipo. Il sommatore ad anticipazione di riporto è in genere più veloce del sommatore a propagazione di riporto, ma richiede più circuiti e risorse per la sua implementazione.

Spiegare chiaramente in base a quali principi cardine siano realizzati.

**Il sommatore a propagazione** di riporto si basa sul principio di calcolo sequenziale del riporto, dove il riporto di ciascun bit viene propagato al bit successivo.

**Il sommatore ad anticipazione** di riporto si basa sul principio di calcolo parallelo del riporto, dove i riporti vengono calcolati in anticipo utilizzando una logica di anticipazione.

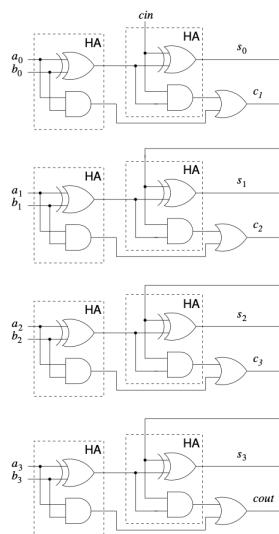
Quali sono i vantaggi e svantaggi di questi due sommatori?

**Propagazione:** È semplice ma è Lento perché ogni sommatore deve attendere il riporto dal sommatore precedente prima di poter completare il proprio calcolo.

**Anticipazione:** implementazione complessa ma molto più veloce.

4. [2] Costruire un sommatore HW per numeri interi su 4 bit.

## Struttura di un n-bit adder (n=4)



Struttura al livello gate di un n-bit adder. Si noti che l'half-adder che somma  $a_0$  e  $b_0$  è stato sostituito da un full-adder in modo da poter utilizzare un carry-in di ingresso.

Spiegare come funziona il meccanismo dell'anticipazione di riporto.

**Il sommatore ad anticipazione di riporto (Carry Look-Ahead Adder)** è progettato per calcolare i riporti in modo parallelo, riducendo il tempo necessario per l'addizione binaria. Ecco come funziona:

1. Concetti di Base:

- **Generate (Gi)**: Un bit genera un riporto se entrambi i bit di ingresso sono 1 ( $Gi = Ai \cdot Bi$ ).
  - **Propagate (Pi)**: Un bit propaga un riporto se almeno uno dei bit di ingresso è 1 ( $Pi = Ai \oplus Bi$ ).
2. **Calcolo Parallelo dei Riporti**: Utilizzando i segnali di generate e propagate, i riporti per tutti i bit vengono calcolati in parallelo. Le equazioni generali per il calcolo dei riporti sono:
- $C1 = G0 + (P0 \cdot C0)$
  - $C2 = G1 + (P1 \cdot C1)$
  - $C3 = G2 + (P2 \cdot C2)$
  - E così via, per tutti i bit.
3. **Calcolo delle Somme**: Una volta calcolati i riporti, la somma di ciascun bit può essere determinata con:
- $S_i = A_i \oplus B_i \oplus C_{i-1}$

I riporti vengono calcolati in parallelo, rendendo l'addizione molto più veloce rispetto al sommatore a propagazione di riporto.

3. [4] Disegnare un circuito hardware in grado di eseguire la somma ad anticipazione di riporto su 3 bit. Calcolare complessità e cammino critico.

Per un sommatore ad anticipazione di riporto a 3 bit:

- **Complessità**: Richiede 3 AND e 3 XOR per  $Gi$  e  $Pi$ , 6 AND e 6 OR per i riporti, e 3 XOR per le somme.
- **Cammino Critico**: Il cammino critico è di 8 livelli logici.

#### **Sommatore ad anticipazione di riporto**

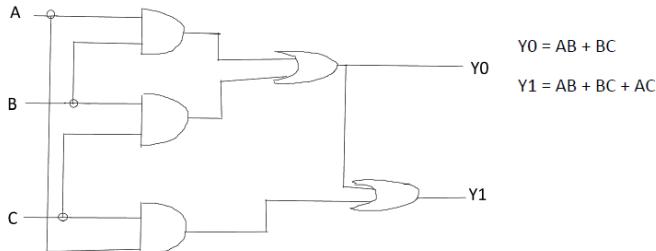
Il sommatore ad anticipazione di riporto si basa sul principio di **anticipare** i riporti a partire dai segnali di **generazione e propagazione** del riporto. In questo modo, si riduce la latenza della somma, permettendo di sommare numeri binari in modo molto più veloce rispetto ai sommatori tradizionali, che calcolano i riporti in modo sequenziale.

**Il sommatore ad anticipazione di riporto è un circuito che calcola rapidamente i riporti per ciascuna posizione di bit, utilizzando una logica combinatoria per "anticipare" i riporti senza dover attendere che il calcolo proceda bit per bit.** Questo permette di ridurre significativamente il tempo totale necessario per eseguire l'operazione di somma.

# funzioni

sabato 20 luglio 2024 09:30

4. [2] Scrivere una funzione logica a piacere, di 3 variabili in ingresso e 2 in uscita, il cui circuito associato abbia cammino critico pari a 3 e complessità pari a 5 [2]



A	B	C	Y0	Y1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

5. [2] Trasformare la funzione logica scritta per l'esercizio precedente nella prima e nella seconda forma canonica.

### 1 forma canonica: SOP

$$Y_0 = !ABC + AB!C + ABC = !ABC + AB(!C + C) = !ABC + AB$$

$$Y_1 = !ABC + A!BC + AB!C + ABC = !ABC + A!BC + AB(!C + C) = !ABC + A!BC + AB$$

### 2 forma canonica: POS

$$Y_0 = (A+B+C)(A+B+!C)(A+!B+C)(!A+B+C)(!A+B+!C)$$

$$Y_1 = (A+B+C)(A+B+!C)(A+!B+C)(!A+B+C)$$

6. [2] Mediante manipolazione algebrica, facendo riferimento alle forme canoniche scritte per l'esercizio precedente, trasformare la prima forma canonica nella seconda o viceversa.

6. [2] Scrivere la funzione logica ottenuta per l'esercizio 5, nella prima e nella seconda forma canonica.

7. [2] Mediante manipolazione algebrica, trasformare la prima forma canonica nella seconda o viceversa.

4. [6] Data l'espressione logica:  $xyz + x!z$ ,

a) Scrivere la tabella della verità [1],

b) Semplificare l'espressione mediante mappe di Karnaugh [1],

c) Sintetizzare la prima e la seconda forma canonica [2].

d) Mediante manipolazione algebrica, trasformare la prima forma canonica nella seconda o viceversa [2].

### a) Tabella della Verità

Costruiamo la tabella della verità per l'espressione  $xyz + x!z$ . Per comodità, indichiamo le variabili  $x, y$ , e  $z$ , e calcoliamo il valore dell'espressione data.

$x$	$y$	$z$	$xyz + x!z$
0	0	0	0
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

### b) Semplificazione mediante Mappe di Karnaugh

Per semplificare l'espressione con le mappe di Karnaugh, creiamo una mappa  $3 \times 2$  per le variabili  $x, y$ , e  $z$ .

	$yz = 00$	$yz = 01$	$yz = 11$	$yz = 10$
$x = 0$	0	1	1	0
$x = 1$	0	1	1	0

Ora vediamo come possiamo semplificare l'espressione.

- Rileviamo che ci sono dei gruppi nella mappa di Karnaugh:

- Il gruppo di 1 nella cella  $(x = 0, yz = 01)$  e  $(x = 1, yz = 01)$  si raggruppa in  $\overline{x} \cdot z$ .
- Il gruppo di 1 nella cella  $(x = 0, yz = 11)$  e  $(x = 1, yz = 11)$  si raggruppa in  $x \cdot y \cdot z$ .

La funzione semplificata è quindi:

$$xyz + x!z = z$$

### c) Prima e Seconda Forma Canonica

#### Prima Forma Canonica (SOMMA DI PRODOTTI)

La prima forma canonica si basa su come scrivere l'espressione come somma di mintermini.

Osserviamo i mintermini per cui l'espressione è vera:

- Quando  $(x = 0, y = 0, z = 1), (x = 0, y = 1, z = 1), (x = 1, y = 0, z = 1), (x = 1, y = 1, z = 1)$ .

Quindi i mintermini sono:

$$m_1 = x!y!z$$

$$m_3 = x!yz$$

$$m_5 = xy!z$$

$$m_7 = xyz$$

La prima forma canonica è:

$$x!y!z + x!yz + xy!z + xyz$$

#### Seconda Forma Canonica (PRODOTTO DI SOMME)

La seconda forma canonica si basa su scrivere l'espressione come prodotto di somme.

Per ottenere la forma canonica, scriviamo gli zeri della tabella di verità come prodotti di somme:

- Per  $(x = 0, y = 0, z = 0), (x = 0, y = 1, z = 0), (x = 1, y = 0, z = 0), (x = 1, y = 1, z = 0)$ , otteniamo le somme che sono  $(x + y + z), (x + y + z!), (x! + y + z), (x! + y! + z)$ .

La seconda forma canonica è quindi:

$$(x + y + z) \cdot (x + y + z!) \cdot (x! + y + z) \cdot (x! + y! + z)$$

#### d) Trasformazione tra la Prima e la Seconda Forma Canonica

Utilizziamo la proprietà distributiva e le leggi algebriche per trasformare una forma canonica nell'altra. In questo caso, abbiamo già trovato la forma semplificata  $zz$ , quindi le forme canoniche che derivano dalla forma semplificata e dalle rispettive trasformazioni sono equivalenti.

Quindi, possiamo dire che:  $xyz+x!z=xyz+x!z=z$  è la forma semplificata.

Per la trasformazione diretta:

- La prima forma canonica (SOMMA DI PRODOTTI) può essere trasformata in una forma di prodotto di somme utilizzando le leggi algebriche.
- La seconda forma canonica (PRODOTTO DI SOMME) può essere semplificata per ottenere la forma finale e ottenere la somma di prodotti equivalenti.

1. [7] Data l'espressione logica:  $xyz + !yz$ ,

a) scrivere la tabella della verità [1],

b) semplificare l'espressione mediante mappe di Karnaugh [1],

c) sintetizzare la prima e la seconda forma canonica [2].

d) Mediante manipolazione algebrica, trasformare la prima forma canonica nella seconda o viceversa [3].

##### a) Tabella della Verità

Per costruire la tabella della verità per l'espressione  $xyz + !yz$ , consideriamo tutte le combinazioni di  $x$ ,  $y$  e  $z$ , e calcoliamo il valore dell'espressione.

$x$	$y$	$z$	$xyz + !yz$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

##### b) Semplificazione mediante Mappe di Karnaugh

Per semplificare l'espressione con le mappe di Karnaugh, costruiamo una mappa  $3 \times 2$  per le variabili  $x$ ,  $y$ , e  $z$ .

	$yz = 00$	$yz = 01$	$yz = 11$	$yz = 10$
$x = 0$	0	1	1	0
$x = 1$	0	0	1	0

Ora, analizziamo la mappa:

1. Nella mappa ci sono due gruppi di 1:

- Un gruppo nella cella  $(x = 0, yz = 01)$  e  $(x = 1, yz = 11)$  si raggruppa in  $x \cdot y \cdot z$ .
- Un gruppo nella cella  $(x = 1, yz = 11)$  si raggruppa in  $x \cdot y \cdot z$ .

La funzione semplificata è quindi:

$$xyz + !yz = yz$$

### c) Prima e Seconda Forma Canonica

#### Prima Forma Canonica (SOMMA DI PRODOTTI)

La prima forma canonica si basa sui mintermini per cui l'espressione è vera. Dalla tabella della verità, abbiamo:

- Quando  $(x = 0, y = 0, z = 1), (x = 0, y = 1, z = 1), (x = 1, y = 1, z = 1)$ .

Quindi i mintermini sono:

$$m_1 = x!y!z$$

$$m_3 = x!yz$$

$$m_7 = xyz$$

La prima forma canonica è:

$$x!y!z + x!yz + xyz$$

#### Seconda Forma Canonica (PRODOTTO DI SOMME)

La seconda forma canonica si basa su scrivere l'espressione come prodotto di somme. Per ottenere la forma canonica, scriviamo i mintermini per cui l'espressione è falsa (le celle con 0) e li combiniamo in somme:

- Per  $(x = 0, y = 0, z = 0), (x = 0, y = 1, z = 0), (x = 1, y = 0, z = 0), (x = 1, y = 1, z = 0)$ , otteniamo le somme:

- $(x + y + z)$
- $(x + y + z!)$
- $(x! + y + z)$
- $(x! + y! + z)$

La seconda forma canonica è quindi:

$$(x + y + z) \cdot (x + y + z!) \cdot (x! + y + z) \cdot (x! + y! + z)$$

### d) Trasformazione tra la Prima e la Seconda Forma Canonica

Per trasformare tra la prima forma canonica e la seconda forma canonica, possiamo seguire le leggi algebriche. In questo caso, la forma semplificata  $yz$  può essere rappresentata direttamente dalle forme canoniche ottenute.

Per la manipolazione algebrica:

- La prima forma canonica  $x!y!z+x!yz+xyz$  si può semplificare direttamente a  $yz$ .
- La seconda forma canonica è una rappresentazione complessa della stessa funzione e può essere semplificata per ottenere la somma dei mintermini e la loro forma complementare.

Se desideri un approfondimento su come fare queste manipolazioni algebriche specificamente, fammi sapere!

10. [2] Semplificare l'espressione logica:  $xyz + !yz$ .

**1. Raggruppamento e Distribuzione:**

Raggruppiamo i termini in base alla presenza di  $yz$ :

$$y = xyz + \neg yz$$

Possiamo estrarre  $yz$  come termine comune:

$$y = yz(x + \neg 1)$$

**2. Legge dell'Identità:**

Dato che  $x + \neg 1 = 1$  (perché  $\neg 1$  è 0 e  $x + 0 = x$ , ma con  $x + 1$  è sempre 1), possiamo semplificare ulteriormente:

$$yz(x + \neg 1) = yz$$

Quindi l'espressione si semplifica a:

$$y = yz$$

La forma più semplice di questa espressione è:

$$y = yz$$

**7. [5] Sintetizzare come SOP e poi semplificare la seguente funzione logica:  $Y_1 = BC + (\neg A)B$ ;  $Y_2 = ABC + \neg A$ . Implementare la funzione con una PLA e ROM e discutete vantaggi e svantaggi dei due approcci. Cos'è un mintermine?**

**3. [3] Semplificare la seguente espressione logica:  $y = abcd + (\neg b)cd + (\neg a)bc$**

Cominciamo raggruppando i termini in base alle variabili comuni:

$$y = abcd + (\neg bcd) + (\neg abc)$$

Possiamo separare i termini con variabili comuni:

$$y = cd(ab + \neg b) + \neg abc$$

## 2. Semplificazione dei Termini:

- Per il termine  $cd(ab + \neg b)$ :

$$ab + \neg b = a + \neg b \text{ (per la legge di assorbimento)}$$

Quindi:

$$cd(ab + \neg b) = cd(a + \neg b) = cda + cd\neg b$$

- Il termine  $\neg abc$  non ha variabili comuni con i termini precedenti.

Uniamo tutto:

$$y = cda + cd\neg b + \neg abc$$

## 3. Ulteriore Semplificazione:

Esaminiamo se possiamo semplificare ulteriormente:

- Notiamo che il termine  $cda$  può essere considerato con  $cd$  come:

$$cda + cd\neg b = cd(a + \neg b)$$

- La funzione diventa:

$$y = cd(a + \neg b) + \neg abc$$

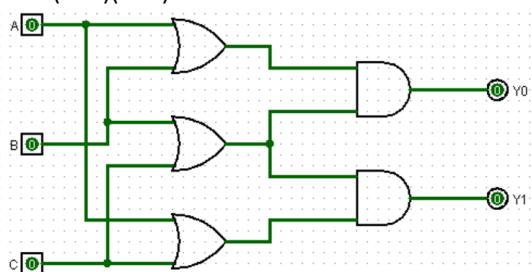
7. [5] Sintetizzare come SOP e poi semplificare la seguente funzione logica:  $Y_1 = AB + B(\neg C)$ ;  $Y_2 = ABC + \neg BC$ . Implementare la funzione con una PLA e ROM e discutete vantaggi e svantaggi dei due approcci. Cos'è un mintermine?

5. [2] Scrivere una funzione logica a piacere, di 3 variabili in ingresso e 2 in uscita, il cui circuito associato abbia cammino critico pari a 2 e complessità pari a 5.

CC = 2, CO = 5

$$Y_0 = (A+B)(B+C)$$

$$Y_1 = (B+C)(A+C)$$



6. [2] Scrivere la funzione logica ottenuta per l'esercizio 5, nella prima e nella seconda forma canonica.

A	B	C	$Y_1$	$Y_0$
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

1 forma canonica:

$$Y_0 = !ABC + !ABC + A!BC + AB!C + ABC = AB(C+!C) + !AB!C + !ABC + A!BC = AB + !AB!C + !ABC + A!BC$$

$$Y_1 = !A!BC + !ABC + A!BC + AB!C + ABC = AB(!C+C) + !A!BC + !ABC + A!BC = AB + !A!BC + !ABC + A!BC$$

2 forma canonica:

$$Y_0 = (A+B+C)(A+B+!C)(!A+B+C)$$

$$Y_1 = (A+B+C)(A+!B+C)(!A+B+C)$$

7. [2] Mediante manipolazione algebrica, trasformare la prima forma canonica nella seconda o viceversa.

# Ciclo istruzione Mips

sabato 20 luglio 2024 09:31

9. [3] Disegnare il ciclo di esecuzione di un'istruzione su un'architettura MIPS a singolo ciclo. Quante fasi si distinguono?

## Fasi del Ciclo di Esecuzione di un'Istruzione MIPS a Singolo Ciclo

1. Fetch: Recupero dell'istruzione dalla memoria.
2. Decode: Decodifica dell'istruzione e lettura dei registri.
3. Execute: Esecuzione dell'operazione ALU (Arithmetic Logic Unit) o calcolo dell'indirizzo.
4. Memory Access: Accesso alla memoria dati (per istruzioni di load/store).
5. Write Back: Scrittura del risultato nei registri.

Quando l'architettura capisce di che istruzione si tratta?

## Decode

I componenti appartenenti a quali fasi vengono configurati dall'unità di controllo? Perché?

Vengono configurati la ALU per fare i calcoli e la memoria per memorizzare i dati

- **Fetch:** Determina l'incremento del PC.
- **Decode:**
  - **Control Signals:** Configura le fasi successive.
- **Execute:**
  - **ALUOp:** Configura l'ALU.
  - **ALUSrc:** Seleziona l'operando per l'ALU.
  - **Branch:** Configura il target di branch.
- **Memory:**
  - **MemRead:** Abilita la lettura dalla memoria.
  - **MemWrite:** Abilita la scrittura nella memoria.
- **Write Back:**
  - **MemtoReg:** Seleziona il dato da scrivere nel registro.
  - **RegWrite:** Abilita la scrittura nel registro.

L'unità di controllo configura i vari componenti del processore in ogni fase dell'esecuzione dell'istruzione attraverso segnali di controllo specifici, garantendo l'esecuzione corretta delle operazioni richieste.

Cos'è un'ISA?

Definisce l'insieme di istruzioni che un processore può eseguire e fornisce una specifica dettagliata delle operazioni che il processore è in grado di compiere. L'ISA funge da interfaccia tra il software e l'hardware, determinando come il software controlla l'hardware.

Possono due CPU avere la stessa ISA?

Sì, due CPU diverse possono avere la stessa ISA. In effetti, questo è un concetto comune nel design dei processori.

Quando due CPU implementano la stessa ISA, sono in grado di eseguire lo stesso set di istruzioni. Questo significa che il software scritto per una CPU può funzionare anche sull'altra senza bisogno di modifiche. Questa compatibilità facilita lo sviluppo del software e permette di avere una varietà di hardware tra cui scegliere, che può eseguire lo stesso codice.

Due CPU diverse devono avere una ISA necessariamente diversa? NO

ISA è l'acronimo di Instruction Set Architecture. Rappresenta l'interfaccia tra il software e l'hardware di un processore. L'ISA definisce il set di istruzioni che un processore può eseguire e le modalità in cui queste istruzioni vengono codificate e interpretate.

L'ISA specifica i tipi di istruzioni supportate dal processore, i formati delle istruzioni, il numero e i tipi di registri disponibili, i modi in cui le istruzioni accedono alla memoria e altre caratteristiche legate all'architettura del processore.

Le istruzioni definite nell'ISA sono rappresentate in linguaggio macchina, che è il formato binario che il processore comprende e può eseguire direttamente. L'ISA fornisce una serie di operazioni di base che il processore può eseguire, come operazioni aritmetiche, operazioni logiche, trasferimenti di dati, operazioni di salto e altro ancora.

Le diverse architetture dei processori, come x86, ARM, MIPS, etc hanno ISA diversi. Ogni ISA ha le sue specifiche istruzioni, formati di istruzione, modalità di indirizzamento e altre caratteristiche uniche. No, non è necessario che due CPU diverse abbiano una ISA (Instruction Set Architecture) necessariamente diversa. È possibile che due CPU diverse abbiano la stessa ISA, specialmente se sono progettate per supportare lo stesso set di istruzioni e lo stesso modello di programmazione.

Tuttavia, è importante notare che pur avendo la stessa ISA, le CPU possono differire in termini di prestazioni, frequenza di clock, organizzazione interna, dimensione della cache e altri aspetti dell'architettura. Quindi, mentre l'ISA può essere la stessa, ci possono essere differenze significative tra le CPU nel modo in cui implementano effettivamente l'ISA e gestiscono le istruzioni.

Definire i diversi formati delle istruzioni MIPS e specificare la funzione dei diversi campi.

**Formato R:**

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6

**Formato I:**

op	rs	rt	immediate		
6	5	5	16		

**Formato J:**

op	address		
6	26		

**Funzioni dei Campi**

- op (Opcode): Indica il tipo di istruzione.
- rs: Registro sorgente 1, utilizzato nei formati R e I.
- rt: Registro sorgente 2 o destinazione, utilizzato nei formati R e I.
- rd: Registro destinazione, utilizzato nel formato R.
- shamt: Specifica lo spostamento nei bit nelle istruzioni di shift.
- funct: Specifica l'operazione esatta da eseguire nel formato R.
- immediate: Valore immediato o offset per operazioni, utilizzato nel formato I.
- address: Indirizzo di salto, utilizzato nel formato J.

Descrivere come viene suddivisa in modo logico per convenzione una memoria principale dai processori MIPS e quali sono le ragioni per la scelta dei confini dei diversi segmenti.

La memoria principale in MIPS è generalmente suddivisa nei seguenti segmenti:

- Segmento delle Istruzioni: Contiene il codice eseguibile del programma.
- Segmento dei Dati: Contiene variabili globali e statiche del programma.
- Segmento di Input/Output: Contiene indirizzi di memoria riservati per la comunicazione con le periferiche.
- Segmento Riservato o Non Utilizzato: Parte della memoria che non è utilizzata o è riservata per scopi futuri o per il sistema operativo.

La suddivisione della memoria principale in un'architettura MIPS è progettata per ottimizzare l'efficienza, garantire la protezione e consentire una gestione flessibile della memoria, mantenendo separate le aree di codice, dati, heap e stack.

7. [2] Disegnare una ALU dell'architettura MIPS a due livelli. Cosa rappresenta ciascun livello? Perché vengono utilizzati due livelli? Quali sono gli ingressi e le uscite del secondo livello?

Ciascun livello rappresenta una fase distinta dell'elaborazione dell'ALU. I due livelli sono il livello combinatorio e il livello sequenziale.

Livello combinatorio: Il primo livello, chiamato anche "combinational level", è responsabile dell'esecuzione delle operazioni logiche e aritmetiche dell'ALU. In questo livello, le operazioni vengono eseguite combinando i segnali di input in base alle istruzioni correnti e alle operazioni richieste. Il livello combinatorio genera quindi i risultati in base ai valori di input, senza memorizzare alcuno stato interno.

Livello sequenziale: Il secondo livello, chiamato anche "sequential level", agisce come un registro che memorizza temporaneamente i risultati prodotti dal livello combinatorio. In questo livello, i

risultati delle operazioni vengono registrati in flip-flop o latch per conservare lo stato dell'ALU tra un ciclo di clock e il successivo. Il livello sequenziale consente di mantenere lo stato e di sincronizzare le operazioni dell'ALU con il ciclo di clock del sistema.

I due livelli sono utilizzati per separare le operazioni combinatorie, che richiedono solo operazioni logiche e aritmetiche senza alcuna memoria interna, dalle operazioni sequenziali, che richiedono la conservazione dello stato e la sincronizzazione temporale.

Le uscite del secondo livello sono i risultati delle operazioni che vengono registrati e conservati nei registri interni dell'ALU. Questi risultati possono essere utilizzati come input per le successive operazioni o per la comunicazione con altre parti del processore.

Gli ingressi del secondo livello includono i segnali di controllo per le operazioni dell'ALU, segnali di clock per la sincronizzazione temporale e i risultati del livello combinatorio come input per il registro interno.

**Disegnare il ciclo di esecuzione di un'istruzione su un'architettura MIPS a singolo ciclo. Quante fasi si distinguono? Quando l'architettura capisce di che istruzione si tratta? I componenti appartenenti a quali fasi vengono configurati dall'unità di controllo? Perché?**

In un'architettura MIPS a singolo ciclo, l'elaborazione di ogni istruzione viene completata in un singolo ciclo di clock. Il ciclo di esecuzione di base comprende le seguenti fasi (per l'esattezza 5 fasi):

Fetch (Recupero): L'istruzione viene recuperata dalla memoria di programma (o dalla cache delle istruzioni) utilizzando il valore del Program Counter (PC) come indirizzo di memoria. L'istruzione viene quindi caricata nel Instruction Register (IR).

Vengono configurati i seguenti componenti:

Program Counter: L'unità di controllo incrementa il valore del PC per indicare l'indirizzo della prossima istruzione da recuperare.

Memoria di programma: L'unità di controllo inizia l'accesso alla memoria di programma per recuperare l'istruzione successiva.

Decode (Decodifica): L'istruzione viene decodificata per determinare il suo tipo e gli operandi coinvolti. Le informazioni di controllo necessarie per l'esecuzione dell'istruzione vengono estratte dall'IR e vengono inviate ai circuiti di controllo appropriati. (è in questa fase che l'architettura capisce di quale istruzione sta trattando)

Vengono configurati i seguenti componenti:

Instruction Register: L'unità di controllo carica l'istruzione recuperata nella IR.

Circuiti di decodifica: L'unità di controllo interpreta i bit dell'istruzione nell'IR per determinare il tipo di istruzione e gli operandi coinvolti.

Execute (Esecuzione): L'istruzione viene eseguita utilizzando i dati presenti nei registri di destinazione o prelevati dalla memoria di dati. Questa fase può includere operazioni aritmetiche, logiche, di accesso alla memoria o di controllo del flusso.

Vengono configurati i seguenti componenti:

Unità di calcolo: L'unità di controllo configura le unità di calcolo appropriate per eseguire l'operazione specifica richiesta dall'istruzione. Ad esempio, un'unità ALU (Arithmetic Logic Unit) può essere configurata per eseguire operazioni aritmetiche o logiche.

Altri componenti di esecuzione: L'unità di controllo può attivare altri componenti specializzati necessari per l'esecuzione dell'istruzione, come unità di moltiplicazione o divisione, unità di gestione delle stringhe, etc.

Memory Access (Accesso alla memoria): Se l'istruzione richiede l'accesso alla memoria, come ad esempio un caricamento o un salvataggio di dati, questa fase coinvolge l'accesso alla memoria dati per ottenere o memorizzare i dati necessari.

Vengono configurati i seguenti componenti:

Memoria dati: Se l'istruzione richiede l'accesso alla memoria di dati, l'unità di controllo configura l'accesso alla memoria appropriata per leggere o scrivere i dati.

Write Back (Scrittura nel registro di destinazione): Il risultato dell'istruzione viene scritto nel registro di destinazione appropriato, come un registro generale o un registro di stato, a seconda dell'istruzione specifica.

Vengono configurati i seguenti componenti:

Registri: L'unità di controllo determina il registro di destinazione in cui deve essere scritto il risultato dell'istruzione e configura l'operazione di scrittura appropriata.

Una volta completate tutte queste fasi, il ciclo di esecuzione dell'istruzione è terminato e il processore

passa al recupero della prossima istruzione. Nel caso di un'architettura MIPS a singolo ciclo, ogni istruzione richiede lo stesso tempo per essere eseguita, indipendentemente dalla sua complessità o dal tipo di istruzione.

**Descrivere come viene suddivisa in modo logico per convenzione una memoria principale dai processori MIPS e quali sono le ragioni per la scelta dei confini dei diversi segmenti.**

Nell'architettura dei processori MIPS, la memoria principale viene suddivisa in modo logico in diversi segmenti, ciascuno dei quali ha uno scopo specifico:

Text Segment (Segmento del codice):

Dove: Solitamente situato all'inizio della memoria principale.

Cosa: Contiene il codice eseguibile, come le istruzioni del programma.

Perchè: La posizione fissa del segmento del codice semplifica l'accesso alle istruzioni del programma. Inoltre, separare il codice dalle altre regioni di memoria può aiutare a proteggerlo da modifiche accidentali o non autorizzate.

Data Segment (Segmento dei dati):

Dove: Segue il segmento del codice.

Cosa: Contiene i dati statici e le variabili inizializzate.

Perchè: Separare il segmento dei dati dal segmento del codice consente una gestione più efficiente dei dati, in particolare quando si effettuano operazioni come la copia o l'inizializzazione dei dati.

BSS Segment (Segmento BSS):

Dove: Segue il segmento dei dati.

Cosa: Contiene le variabili non inizializzate (non inizializzate a un valore specifico).

Perchè: Le variabili non inizializzate occupano spazio di memoria, ma non richiedono un'allocazione fisica fino a quando non vengono effettivamente utilizzate. Il segmento BSS viene utilizzato per allocare dinamicamente spazio per queste variabili durante l'esecuzione.

Heap:

Dove: Solitamente segue il segmento BSS.

Cosa: Fornisce uno spazio di memoria dinamico per l'allocazione e la gestione dei dati a tempo di esecuzione.

Perché: L'heap consente di gestire l'allocazione e la liberazione dinamica della memoria durante l'esecuzione del programma. È particolarmente utile quando le dimensioni dei dati non sono conosciute a priori o variano nel tempo.

Stack:

Dove: Di solito è situato verso la fine della memoria principale.

Cosa: Gestisce le chiamate di funzione, le variabili locali e altre informazioni relative alla gestione delle chiamate.

Perché: Lo stack viene utilizzato per gestire la struttura dati a pila durante l'esecuzione delle funzioni. La sua posizione fissa semplifica l'accesso e la gestione delle chiamate di funzione, dei parametri e delle variabili locali.

La scelta dei confini dei diversi segmenti della memoria è determinata principalmente da motivi di organizzazione e gestione dei dati. La separazione dei segmenti in base alla funzionalità facilita l'accesso, l'allocazione e la gestione dei dati nel corso dell'esecuzione del programma.

**Definire i formati delle istruzioni MIPS specificando la dimensione e la funzione dei diversi campi.**

L'architettura MIPS utilizza diversi formati di istruzioni per rappresentare le diverse operazioni supportate:

R-Format (Register Format):

Dimensione: 32 bit

Campi:

Funzione (Funct): 6 bit (specificano l'operazione specifica all'interno delle istruzioni di tipo R)

Primo operando (rs): 5 bit (specificano il registro sorgente 1)

Secondo operando (rt): 5 bit (specificano il registro sorgente 2)

Terzo operando (rd): 5 bit (specificano il registro destinazione)

Shift amount (shamt): 5 bit (specificano la quantità di shift per le operazioni di shift logico o aritmetico)

I-Format (Immediate Format):

Dimensione: 32 bit

Campi:

Codice operativo (opcode): 6 bit (specificano il tipo di operazione)

Registro sorgente (rs): 5 bit (specificano il registro sorgente 1)

Registro destinazione (rt): 5 bit (specificano il registro destinazione)

Immediate value (valore immediato): 16 bit (specificano un valore immediato o un offset)

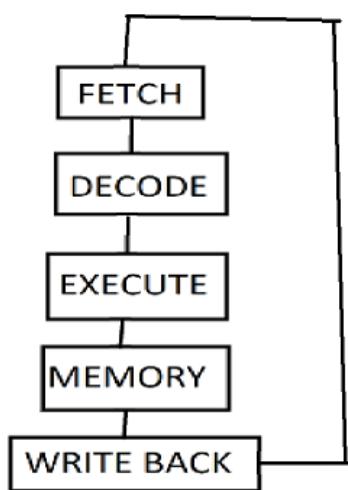
J-Format (Jump Format):

-Dimensione: 32 bit

Campi:

Codice operativo (opcode): 6 bit (specificano il tipo di operazione)

Indirizzo di destinazione (target address): 26 bit (specificano il target address per il salto)



Ci sono 5 fasi. L'architettura capisce di che istruzione si tratta nella fase di Decode.

Componenti configurati dalla UC e la fase a cui appartengono:

- Fetch: la UC indica al MUX da dove prendere il nuovo valore del PC tramite il segnale PCSrc.

- Decode: la UC indica al RF in quale registro scrivere tramite il segnale RegDst.

- Execute: la UC indica alla ALU da dove prendere il secondo operando tramite il segnale ALUSrc. la UC invia alla UC dedicata alla ALU il segnale ALUs che serve per decidere l'operazione della ALU.

- Memory: la UC indica al RF se deve scrivere tramite il segnale RegWrite. la UC indica alla Memoria se deve scrivere o leggere tramite segnali MemWrite/MemRead.

Un'ISA (Instruction Set Architecture) è l'insieme delle istruzioni e la loro codifica di un calcolatore. Comprende il funzionamento (tipologia e meccanismo di funzionamento) e il formato (suddivisione dei gruppi dei bit che formano l'istruzione e il formato dei dati) delle istruzioni. Ogni istruzione deve contenere tutto il necessario per il ciclo di esecuzione: comandi, registri...

Più CPU possono avere la stessa ISA (molte usano la ISA ARM).

Due CPU diverse non hanno necessariamente ISA diverse → avendo la stessa ISA è possibile eseguire il software su più CPU diverse.

Istruzioni R (istruzioni register, aritmetico-logiche):

OPCODE (6) RS (5) RT (5) RD (5) SHAMT (5) FUNCT (6)

OPCODE: 6 bit, indica il tipo di istruzione.

RS: 5 bit, indica il registro che contiene il primo operando dell'operazione.

RT: 5 bit, indica il registro che contiene il secondo operando.

RD: 5 bit, indica il registro in cui scrivere il risultato.

SHAMT: 5 bit, serve nelle operazioni di shift, indica la quantità di shift da eseguire.

FUNCT: 6 bit, indica l'esatta operazione da eseguire (add, sub, ...).

Istruzioni I (istruzioni immediate):

OPCODE (6) RS (5) RT (5) OFFSET (16)

OPCODE: 6 bit, indica il tipo di istruzione.

RS: 5 bit, indica il registro che fa da primo operando.

RT: 5 bit, indica il registro che fa da secondo operando.

OFFSET: 16 bit, contiene una costante che viene usata come operando.

Istruzioni J (istruzioni jump): OPCODE (6) INDIRIZZO (26)

OPCODE: 6 bit, indica il tipo di istruzione.

INDIRIZZO: 26 bit, indica il semi-indirizzo a cui eseguire il jump.

Suddivisione logica memoria principale

- Segmento testo

- Segmento dati

- Heap

- Stack

- Area SO

#### Motivazioni:

Separazione codice e dati: previene la modifica accidentale del codice e ottimizza uso cache.

Gestione dinamica memoria: head e stack crescono in direzioni diverse per evitare collisioni. Permette gestione flessibile e dinamica della memoria durante l'esecuzione del programma.

Protezione SO: garantisce che il codice utente non corrompa il funzionamento del sistema, migliorando la sicurezza e stabilità.

Efficienza accesso memoria: divisione in segmenti consente accesso più efficiente e strutturato.

11. [2] Definire i diversi formati delle istruzioni MIPS e specificare la funzione dei diversi campi. Descrivere come viene suddivisa in modo logico per convenzione una memoria principale dai processori MIPS e quali sono le ragioni per la scelta dei confini dei diversi segmenti

9. [2] Definire i formati delle istruzioni MIPS specificando la dimensione e la funzione dei diversi campi.

8. [3] Definire i campi in cui vengono suddivise le istruzioni di tipo R, I e J e descrivetene l'utilizzo.

Riportare i diversi formati delle istruzioni MIPS e una istruzione di esempio per ogni formato.

#### Formato R (Register)

- Struttura:

- Opcode (6 bit)
- rs (5 bit) - Registro sorgente 1
- rt (5 bit) - Registro sorgente 2
- rd (5 bit) - Registro destinazione
- shamt (5 bit) - Shift amount
- funct (6 bit) - Funzione specifica

- Esempio: add \$t1, \$t2, \$t3

- Significato: Somma il contenuto dei registri \$t2 e \$t3, e memorizza il risultato nel registro \$t1.

#### Formato I (Immediate)

- Struttura:

- Opcode (6 bit)
- rs (5 bit) - Registro sorgente
- rt (5 bit) - Registro destinazione
- Immediate (16 bit) - Valore immediato

- Esempio: addi \$t1, \$t2, 10

- Significato: Somma il contenuto del registro \$t2 con il valore immediato 10, e memorizza il risultato nel registro \$t1.

#### Formato J (Jump)

- Struttura:

- Opcode (6 bit)
- Address (26 bit) - Indirizzo di salto

- Esempio: j 10000

- Significato: Salta all'indirizzo 10000.

È corretto affermare che tutte le operazioni aritmetico logiche sono di tipo R?

No, non è corretto affermare che tutte le operazioni aritmetico-logiche siano di tipo R nell'architettura MIPS. Sebbene

molte operazioni aritmetico-logiche siano effettivamente di tipo R, esistono anche operazioni aritmetico-logiche che utilizzano il formato I come per esempio ADDI.

È corretto affermare che tutte le istruzioni di tipo R sono istruzioni aritmetico logiche?

No, non è corretto affermare che tutte le istruzioni di tipo R sono istruzioni aritmetico-logiche. Le istruzioni di tipo R includono un'ampia gamma di operazioni che vanno oltre le semplici operazioni aritmetiche e logiche. Oltre a queste, esistono altre categorie di istruzioni di tipo R, come le istruzioni di spostamento (shift) e le istruzioni di gestione dei bit come per esempio

**Shift Logico a Sinistra:** sll \$t1, \$t2, 2

- Sposta a sinistra il valore di \$t2 di 2 bit e memorizza il risultato in \$t1.

Che è sempre di tipo R ma non è aritmetico logica.

L'operazione di shift cosa comporta?

Le operazioni di shift (spostamento) comportano lo spostamento dei bit di un registro verso sinistra o verso destra. Le operazioni di shift possono essere utilizzate per moltiplicare o dividere un numero per potenze di due, manipolare singoli bit o campi di bit all'interno di un valore, e altre operazioni di basso livello. In MIPS, le principali operazioni di shift sono:

1. **Shift Logico a Sinistra (SLL) PER MOLTIPLICAZIONE**
2. **Shift Logico a Destra (SRL) PER DIVISIONE**

È equivalente ad una somma, moltiplicazione o divisione?

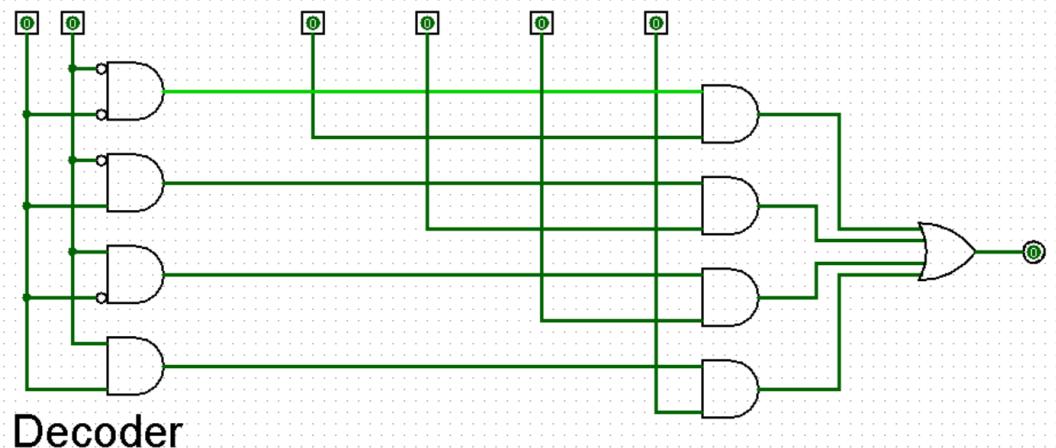
**Shift Logico a Sinistra (SLL) PER MOLTIPLICAZIONE**

**Shift Logico a Destra (SRL) PER DIVISIONE**

# Multiplexer

sabato 20 luglio 2024 09:36

2. [2] Costruire una Mux a 4 ingressi con le porte logiche. Definire complessità e cammino critico



$$CO = 8 \text{ AND e } 1 \text{ OR} = 9$$

$$CC = \text{AND} + \text{AND} + \text{OR} = 3$$

4. [2] Progettare con sole **porte NAND** un multiplexer a due ingressi. Calcolare il cammino critico e complessità del multiplexer così realizzato e confrontarlo con un multiplexer realizzato utilizzando combinazioni di porte AND, OR e NOT.

**NOT usando NAND:**  $!S = S \text{ NAND } S$

**AND usando NAND:**  $(A \cdot B) = !((A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B))$

**OR usando NAND:**  $A+B = !((A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B))$

**Complessità:**

1. Inverter (NOT) per  $!S$ :

- 1 porta NAND

2. AND per  $S \cdot A \cdot A$ :

- 1 porta NAND per  $S \text{ NAND } A$
- 1 porta NAND per l'output U1

3. AND per  $!S \cdot B$ :

- 1 porta NAND per  $!S \text{ NAND } B$
- 1 porta NAND per l'output U2

4. OR per la combinazione finale:

- 1 porta NAND per l'output Y

Totale:  $1 + 2 + 2 + 1 = 6$  porte NAND

**Il cammino critico**

$!S$ : 1 livello (1 porta NAND)

- $S \cdot A$  e  $!S \cdot B$ : 2 livelli ciascuno (1 porta NAND per T e 1 porta NAND per U)
- Combinazione finale: 1 livello (1 porta NAND)

Cammino critico totale:  $1 (\text{NOT}) + 2 (\text{AND}) + 1 (\text{OR}) = 4$  livelli

- **Complessità:**

- Con porte NAND: 6 porte logiche
- Con AND, OR, NOT: 4 porte logiche
- Cammino Critico:
  - Con porte NAND: 4 livelli
  - Con AND, OR, NOT: 3 livelli

6. [3] Progettare con sole **porte NOR** un multiplexer a due ingressi. Calcolare il cammino critico e complessità del multiplexer così realizzato e confrontarlo con un multiplexer realizzato utilizzando combinazioni di porte AND, OR e NOT.

**NOT usando NOR:**  $\text{!S} = \text{S NOR S}$

**AND usando NOR:**  $A \cdot B = (\text{!A} + \text{!B}) \text{ NOR} (\text{!A} + \text{B})$

**OR usando NOR:**  $A + B = \text{!A NOR !B}$

### Complessità

1. **NOT:**
  - $\text{!S}$  richiede 1 porta NOR.
  - $\text{!A}$  e  $\text{!B}$  richiedono ciascuno 1 porta NOR.
2. **AND:**
  - $A \cdot \text{!S}$  richiede 2 porte NOR (1 per la combinazione e 1 per l'inversione).
  - $B \cdot \text{S}$  richiede 2 porte NOR.
3. **OR:**
  - L'OR finale richiede 2 porte NOR.

Totale:  $1+1+1+2+2=9$  porte NOR.

### cammino critico

1. **NOT:**
  - 1 livello logico per calcolare  $\text{!S}$ .
  - 1 livello logico per calcolare  $\text{!A}$  e  $\text{!B}$ .
2. **AND:**
  - 1 livello logico per la combinazione di  $\text{!A}$  e  $\text{!S}$ .
  - 1 livello logico per l'inversione del risultato per ottenere  $A \cdot \text{!S}$ .
  - 1 livello logico per la combinazione di  $\text{!B}$  e  $\text{S}$ .
  - 1 livello logico per l'inversione del risultato per ottenere  $B \cdot \text{S}$ .
3. **OR:**
  - 1 livello logico per la combinazione di  $A \cdot \text{!S}$  e  $B \cdot \text{S}$ .
  - 1 livello logico per l'inversione del risultato per ottenere  $\text{Y}$ .

Totale:  $1+1+1+1+1=6$

### Complessità confronto

- **Con porte NOR:** 9 porte NOR.
- **Con porte AND, OR e NOT:**
  - 1 NOT per  $\text{!S}$  (1 porta NOT).
  - 2 AND per  $A \cdot \text{!S}$  e  $B \cdot \text{S}$  (2 porte AND).
  - 1 OR per il risultato finale (1 porta OR).
  - Totale:  $1+2+1=4$  porte logiche (1 NOT, 2 AND, 1 OR).

### Cammino Critico confronto

- **Con porte NOR:** 6 livelli logici.
- **Con porte AND, OR e NOT:**
  - 1 livello logico per calcolare  $\text{!S}$ .
  - 1 livello logico per ciascuno degli AND ( $A \cdot \text{!S}$  e  $B \cdot \text{S}$ ).
  - 1 livello logico per l'OR finale.
  - Totale:  $1+1+1=3$  livelli logici.

2. [3] Realizzare un multiplexer a due vie, con 1 bit per ciascuna via, utilizzando esclusivamente le porte NAND. Definire complessità e cammino critico.

Un multiplexer a 2 vie con 1 bit per ciascuna via seleziona uno dei due ingressi (A e B) in base al valore di un selettore (S). L'output Y del multiplexer è dato dalla seguente equazione logica:

$$Y = (S \cdot A) + (!S \cdot B)$$

### Complessità:

1. Inverter (NOT) per  $\bar{S}$ :
  - 1 porta NAND
2. AND per  $S \cdot A$  e  $\bar{S} \cdot B$ :
  - 1 porta NAND per  $S$  NAND  $A$
  - 1 porta NAND per l'output  $U_1$
3. AND per  $\bar{S} \cdot B$ :
  - 1 porta NAND per  $\bar{S}$  NAND  $B$
  - 1 porta NAND per l'output  $U_2$
4. OR per la combinazione finale:
  - 1 porta NAND per l'output  $Y$

Totale:  $1 + 2 + 2 + 1 = 6$  porte NAND

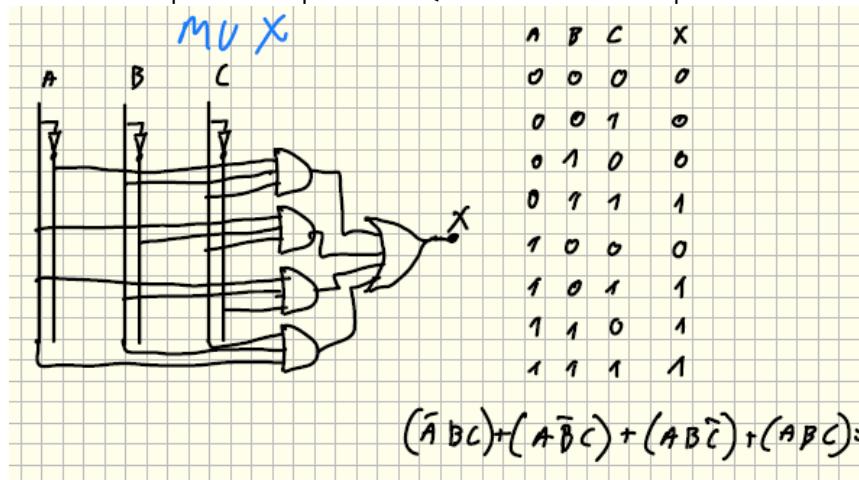
### Il cammino critico

$\bar{S}$ : 1 livello (1 porta NAND)

- $S \cdot A$  e  $\bar{S} \cdot B$ : 2 livelli ciascuno (1 porta NAND per  $T$  e 1 porta NAND per  $U$ )
- Combinazione finale: 1 livello (1 porta NAND)

Cammino critico totale:  $1$  (NOT) +  $2$  (AND) +  $1$  (OR) = 4 livelli

2. [4] Implementare il circuito di un multiplexer a 3 vie e calcolarne cammino critico e complessità. Scrivere l'espressione logica del circuito, trovare una espressione equivalente (mediante manipolazione algebrica) e implementare il circuito associato all'espressione equivalente. Quale dei due circuiti è più conveniente e perché?



L'espressione logica per un multiplexer 3:1 è:

$$Y = (!S_1 \cdot !S_0 \cdot A) + (!S_1 \cdot S_0 \cdot B) + (S_1 \cdot !S_0 \cdot C)$$

### Schema del Multiplexer con Porte AND, OR e NOT

1. NOT Gates per  $S_0$  e  $S_1$ :
  - $\bar{S}_0 = \text{NOT}(S_0)$
  - $\bar{S}_1 = \text{NOT}(S_1)$
2. AND Gates per ciascun termine:
  - $T_1 = \bar{S}_1 \cdot \bar{S}_0 \cdot A$
  - $T_2 = \bar{S}_1 \cdot S_0 \cdot B$
  - $T_3 = S_1 \cdot \bar{S}_0 \cdot C$
3. OR Gate per combinare i termini:
  - $Y = T_1 + T_2 + T_3$

Complessità: 2 NOT + 3 AND + 1 OR = 6 porte

Cammino critico: 1 (NOT) + 1 (AND) + 1 (OR) = 3 livelli

### Manipolazione Algebrica

Riorganizziamo l'espressione per ridurre i termini:

$$Y = \overline{S_1} \cdot (\overline{S_0} \cdot A + S_0 \cdot B) + S_1 \cdot \overline{S_0} \cdot C$$

### Complessità

1. NOT Gates per  $\overline{S_0}$  e  $\overline{S_1}$ :
  - 2 porte NOT
2. AND Gates per ciascun termine:
  - T1, T2, T3, T4 hanno 3 AND Gates
3. OR Gates per combinare i termini interni:
  - 1 porta OR a 2 ingressi per combinare T1 e T2 (che diventa V1)
  - 1 porta OR a 2 ingressi per combinare T3 e T4

Totale: 2 NOT + 4 AND + 2 OR = 8 porte

### Cammino Critico

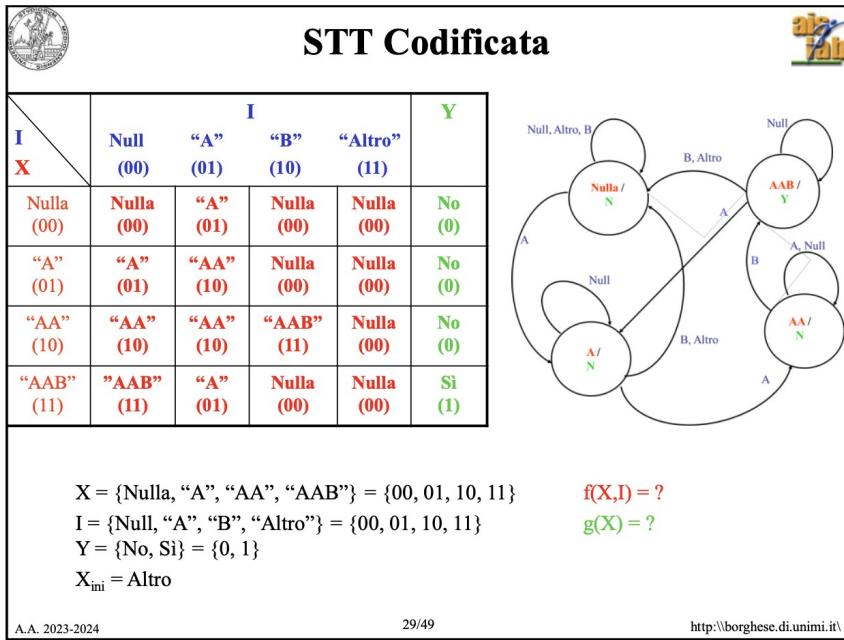
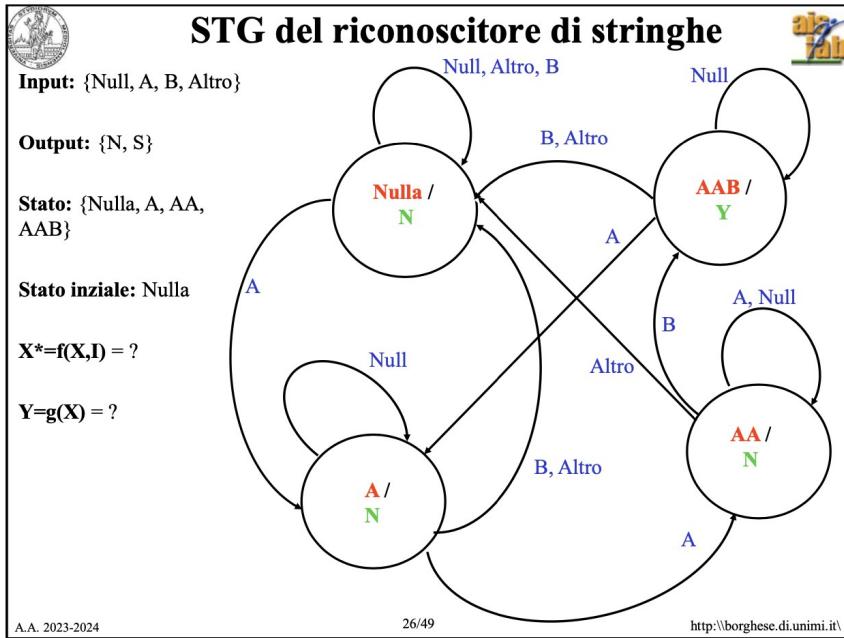
1. Calcolo di  $\overline{S_0}$  e  $\overline{S_1}$ : 1 livello (NOT gates)
2. Calcolo dei termini AND:
  - 2 livelli (AND gates interni e combinazione interna)
3. Combinazione finale con OR:
  - 1 livello (OR gate)

Cammino critico totale: 1 (NOT) + 2 (AND) + 1 (OR) = 4 livelli

Il circuito originale è più conveniente sia in termini di complessità (6 porte contro 8) che di cammino critico (3 livelli contro 4). La manipolazione algebrica, sebbene sembri semplificare l'espressione logica, non riduce la complessità del circuito né il cammino critico, rendendolo meno efficiente rispetto all'implementazione originale.

# MSF stringa

sabato 20 luglio 2024 09:37

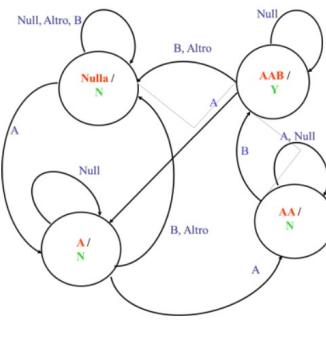




## STT Codificata - sintesi



<b>I</b>	<b>I = {I<sub>1</sub> I<sub>0</sub>}</b>				<b>Y</b>
<b>X=</b> <b>{X<sub>1</sub>X<sub>0</sub>}</b>	Null (00)	"A" (01)	"B" (10)	"Altro" (11)	
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$X = \{\text{Nulla, "A", "AB", "AAB}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

$$X_{\text{ini}} = \text{Altro}$$

$$X^* = f(X, I) = ?$$

$$Y = g(X) = X_1 X_0$$

A.A. 2023-2024

31/49

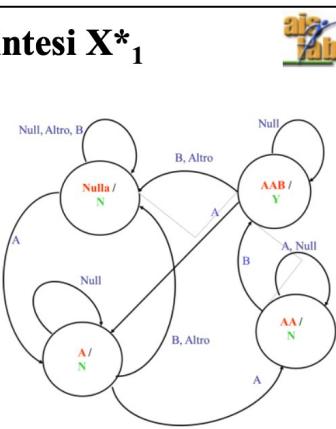
<http://borghese.di.unimi.it/>



## STT Codificata – sintesi X\*<sub>1</sub>



<b>I</b>	<b>I = {I<sub>1</sub> I<sub>0</sub>}</b>				<b>Y</b>
<b>X=</b> <b>{X<sub>1</sub>X<sub>0</sub>}</b>	Null (00)	"A" (01)	"B" (10)	"Altro" (11)	
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$f(X, I) = \begin{cases} X_0^* = f_1(X, I) \\ X_1^* = f_2(X, I) \end{cases}$$

$$g(X) = ?$$

$$X = \{\text{Nulla, "A", "AB", "ABB"}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

$$X_{\text{ini}} = \text{Altro}$$

$$X_1^* = \bar{X}_1 \bar{X}_0 I_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 I_0 + X_1 \bar{X}_0 I_1 \bar{I}_0 + X_1 X_0 \bar{I}_1 \bar{I}_0 = \bar{X}_1 I_1 \bar{I}_0 + X_1 \bar{I}_1 (\bar{X}_0 + \bar{I}_0)$$

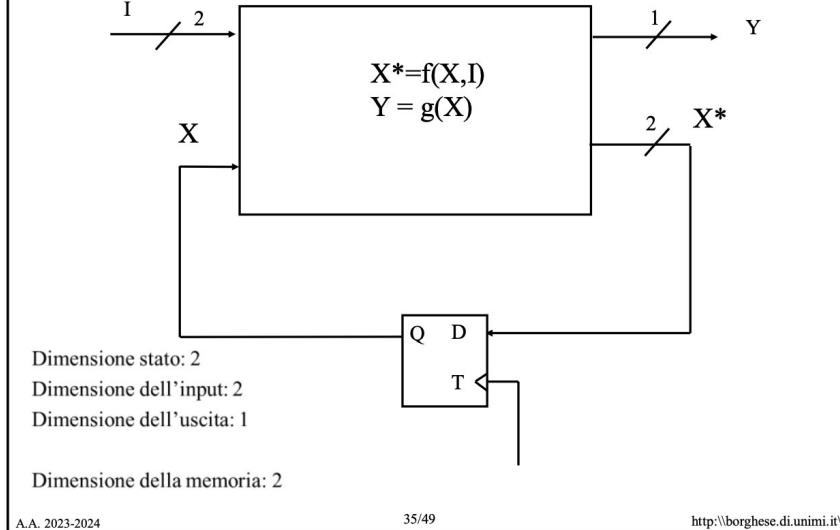
A.A. 2023-2024

34/49

<http://borghese.di.unimi.it/>



## Dimensionamento della MSF



6. [6] Progettare una macchina a stati finiti che scorra un testo (leggendo un carattere alla volta) e riconosca nel testo la stringa " HA " (la coppia di lettere deve necessariamente essere preceduta e seguita da spazi). Progettare la macchina di Huffman associata; calcolarne complessità e cammino critico.

7. [6] Progettare una macchina a stati finiti che scorra un testo (leggendo un carattere alla volta) e riconosca nel testo la stringa "AAA" (la tripletta di lettere NON deve necessariamente essere preceduta o seguita da spazi). Progettare la macchina di Huffman associata; calcolarne complessità e cammino critico.

5. [7] Progettare una macchina a stati finiti che scorra un testo (leggendo un carattere alla volta) e riconosca nel testo la stringa "BBC" (la tripletta di lettere NON deve necessariamente essere preceduta o seguita da spazi). Progettare la macchina di Huffman associata; calcolarne complessità e cammino critico. Implementare le funzioni della macchina utilizzando la prima forma canonica, e semplificare le forme canoniche ottenute. Spiegare cosa sono i maxtermini e come si implementa la seconda forma canonica.

3. [7] Progettare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) tutte le volte che viene trovata la stringa "11". La macchina scorre il testo sequenzialmente, leggendo un carattere alfanumerico alla volta. Il carattere alfanumerico può essere una qualsiasi lettera dell'alfabeto, un numero, uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che lo stato iniziale coincida con il carattere nullo: S0 = "". Viene richiesta la presenza di uno spazio sia prima che dopo "11". Rappresentare la funzione stato prossimo della macchina a stati finiti come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? In caso affermativo calcolare cammino critico e complessità. In caso negativo, spiegare perché non si possono calcolare.

3. [8] Progettare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) tutte le volte che viene trovata la stringa "AZ". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che non è richiesta la presenza di uno spazio né prima né dopo "AZ". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

4. [8] Progettare un contatore in grado di contare da 0 a 3, come macchina a stati finiti. La macchina riceve in ingresso il segnale di reset (riporta a zero il conteggio) e il segnale di conteggio. I segnali di conteggio e di reset sono supposti mutuamente esclusivi e si suppone che il segnale di reset possa arrivare solo al termine del conteggio. Progettare la macchina di Huffman che implementa il contatore. Implementare le funzioni della macchina in tre modi: a) logica distribuita; b) PLA; c) ROM. Per ciascuna delle tre implementazioni definire cammino critico e complessità. Discutere i vantaggi e gli svantaggi di ciascuna implementazione. Definire il mintermine.

4. [8] Progettare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) tutte le volte che viene trovata la stringa "FOO". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che non è richiesta la presenza di uno spazio prima o dopo "FOO". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

1. [8] Progettare e implementare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) quando viene trovata la stringa "BA". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che NON è richiesta la presenza di spazi prima o dopo "BA". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

3. [8] Progettare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) tutte le volte che viene trovata la stringa "AMA". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che non è richiesta la presenza di uno spazio né prima né dopo "AMA". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

3. [8] Progettare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) tutte le volte che viene trovata la stringa "BA". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che NON è richiesta la presenza di spazi prima o dopo "BA". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

2. [8] Progettare e implementare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) quando viene trovata la stringa "HH". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo: S0 = "". Si noti che NON è richiesta la presenza di spazi prima o dopo "HH". Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il

cammino critico e la complessità? Perché? Quanto valgono?

2. [9] Progettare e implementare una macchina a stati finiti che implementa la funzione "Search" in un testo scritto. La macchina scandisce il testo dall'inizio alla fine, leggendo un carattere alla volta, e segnala in uscita (con un 1) quando viene trovata la stringa "AA". La macchina scorre il testo sequenzialmente, leggendo una lettera alla volta. La lettera può essere una qualsiasi lettera dell'alfabeto o uno spazio o un qualsiasi carattere di interpunkzione: "!", "?", ... Si supponga che nello stato iniziale coincida con il carattere nullo:  $S_0 = ""$ . Si noti che NON è richiesta la presenza di uno spazio prima della coppia AA, mentre è richiesto uno spazio dopo la coppia "AA". Quindi occorre riconoscere la stringa: "AA .. Definire la macchina a stati finiti che controlla la macchina. Rappresentare la funzione stato prossimo come SOP, come PLA e come ROM. Semplificare il più possibile tutte le SOP. Quali specifiche occorre dare per le PLA e le ROM? Qual è l'implementazione più vantaggiosa? Cosa sono i mintermini di una funzione? Rappresentare la Macchina a Stati Finiti come Macchina di Huffman. Si possono calcolare il cammino critico e la complessità? Perché? Quanto valgono?

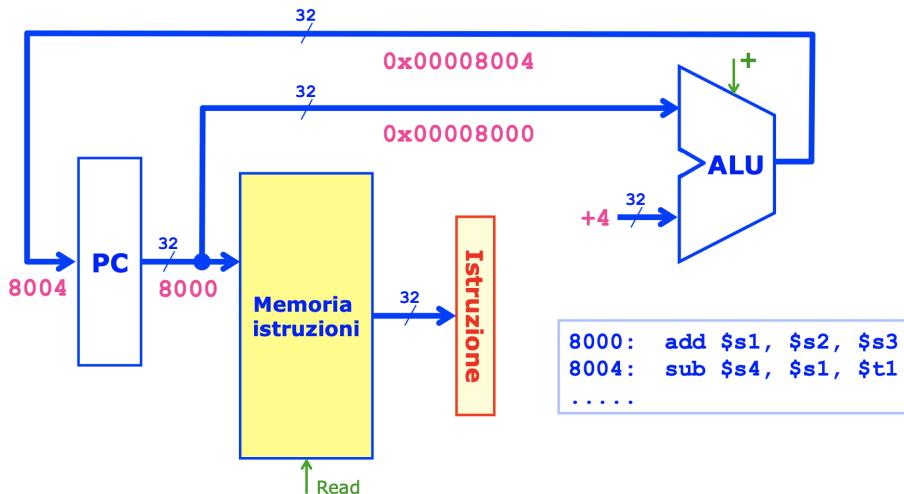
11. [6] Progettare una macchina a stati finiti che scorra un testo (leggendo un carattere alla volta) e riconosca nel testo la stringa "AZ" (la coppia di lettere deve necessariamente essere preceduta da uno spazio, ma NON deve essere necessariamente seguita da uno spazio). Progettare la macchina di Huffman associata; calcolarne complessità e cammino critico.

6. [1] Disegnare il circuito dell'“Estensione del segno” per la CPU in Figura 1.

### Circuito della fase di fetch



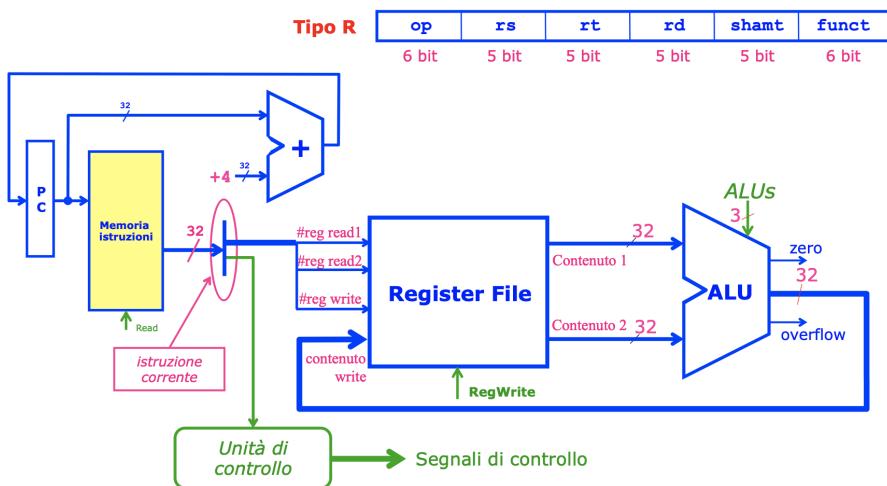
#### Datapath: fase di FETCH (tutte le istruzioni)



### CPU per l'esecuzione di un'istruzione di tipo R



#### Esecuzione istruzione aritmetico/logica tipo R:



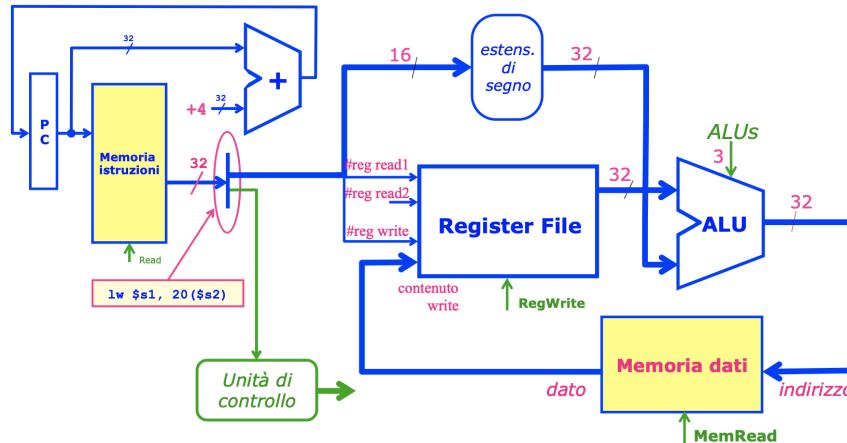
## CPU per l'esecuzione di una **lw**



*Esecuzione istruzione lw:*

8000: **lw \$s1, 20(\$s2)**

6 bit	5 bit	5 bit	16 bit
<b>op</b>	<b>rs</b>	<b>rd</b>	<b>offset</b>



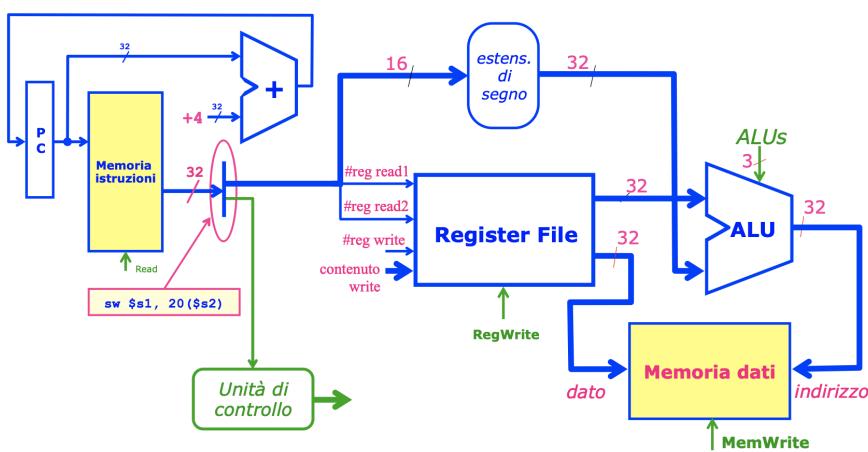
## CPU per l'esecuzione di una **sw**



*Esecuzione istruzione sw:*

8000: **sw \$s1, 20(\$s2)**

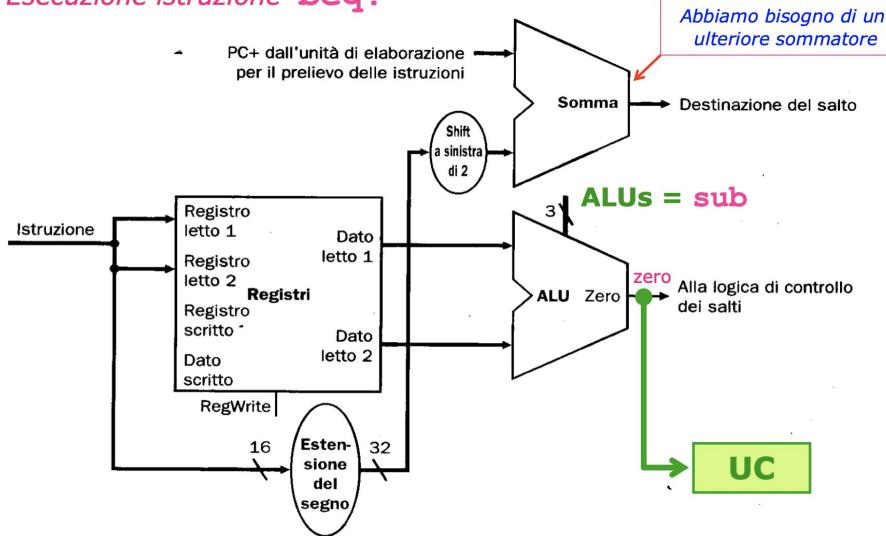
6 bit	5 bit	5 bit	16 bit
<b>op</b>	<b>rs</b>	<b>rt</b>	<b>offset</b>



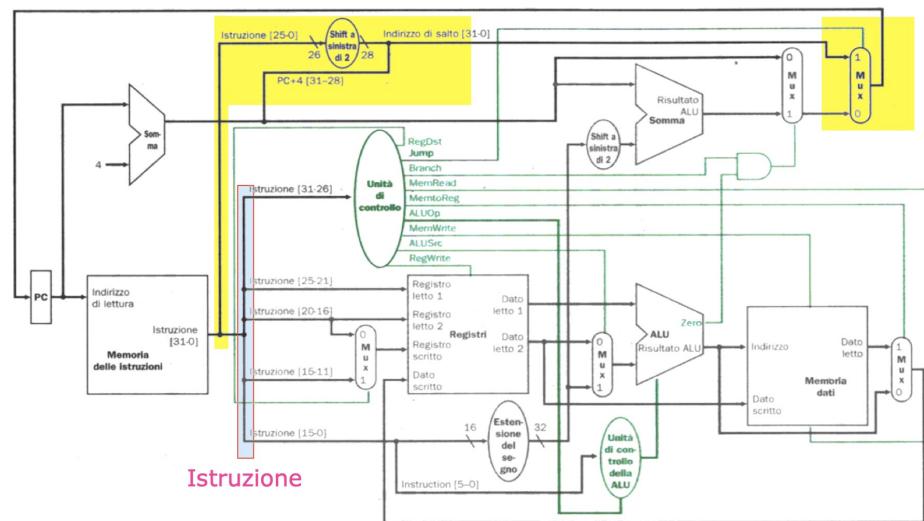
## Circuito di esecuzione: beq



### Esecuzione istruzione beq:



## CPU + UC completa (aggiunta di jump)



# Cpu beq

sabato 20 luglio 2024 09:31

9. [4] Specificare il contenuto di tutti i bus della CPU riportata in Figura 1 quando è in esecuzione l'istruzione di **beq \$1, \$2, ind\_salto**, sapendo che il codice operativo dell'istruzione beq è 4. Si supponga inoltre che l'indirizzo della beq sia 0x4000 0000 e che l'indirizzo di salto sia 0x4000 0040. Evidenziare i bus che trasportano dati utili per l'esecuzione dell'istruzione. Tradurre l'istruzione assembler di beq in linguaggio macchina e rappresentarla in notazione esadecimale. Scrivere in linguaggio macchina l'istruzione di salto incondizionato che consente di saltare allo stesso indirizzo di salto. Il codice operativo dell'istruzione di jump è 2.

**beq \$1, \$2, end\_salto**

Opcode = 4 = 000100

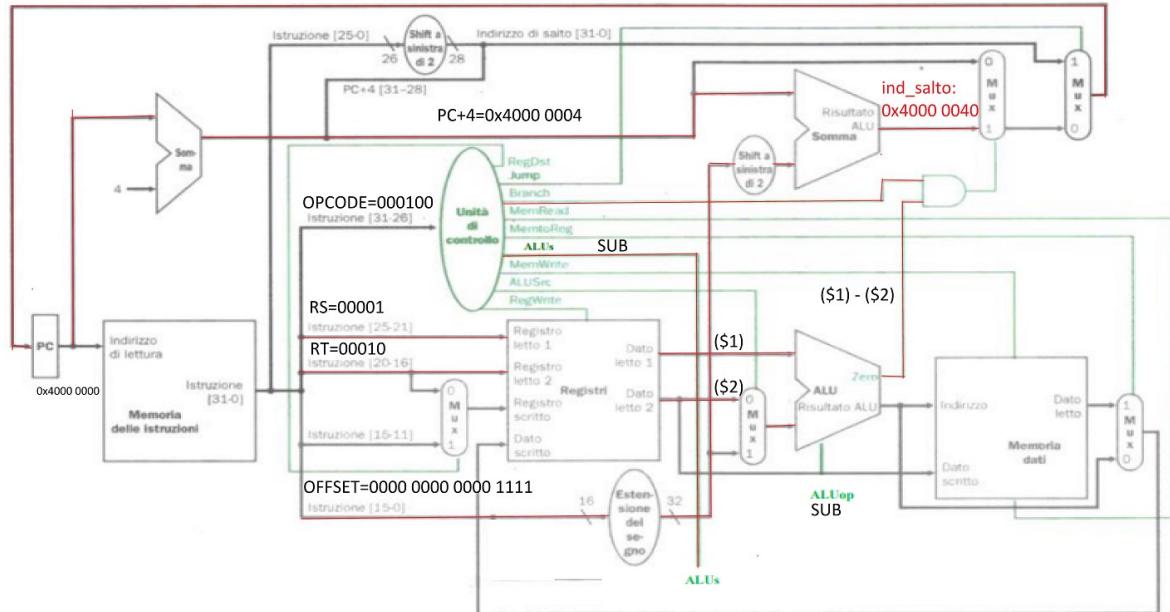
RS = 00001

RT = 00010

OFFSET = indirizzo\_salto - (indirizzo\_attuale+4) = 0x4000 0040 - 0x4000 0004 = 0x3C =  $3 \cdot 16^1 + 12 \cdot 16^0 = 48 + 12 = 60 \cdot 4 = 15$  istruzioni in avanti = 0000 0000 0000 1111

OFFSET shiftato di 2 a sx = 0000 0000 0011 1100 = 0x3C

Nuovo\_indirizzo = (PC+4) + offset\_shiftato = 0x4000 0004 + 0x3C = 0x4000 0040 = indirizzo del salto quindi e' giusto  
Istruzione completa in linguaggio macchina = 000100 000001 00010 0000 0000 0000 1111 = 0x1022000F



Istruzione di JUMP con lo stesso indirizzo di salto:

indirizzo di salto = 0x4000 0040 = 0100 00|00 0000 0000 0000 0100 0000

26 bit meno significativi = 00 0000 0000 0000 0000 0100 0000

26 bit shiftati 2 a sinistra = 0000 0000 0000 0000 0001 0000 0000

Opcode = 2 = 000010

Istruzione JUMP = 000010 0000 0000 0000 0000 0001 0000 0000

11. [2] Scrivere un'istruzione di beq a piacere, specificando tutti i campi e spiegando il significato di ciascun campo. Il codice operativo dell'istruzione di beq è 4.

beq \$1, \$2, 0x0000 0040

beq = opcode 4 = 000100 = indica il tipo di istruzione

\$1 = RS = è il primo operando

\$2 = RT = è il secondo operando

0x0000 0040 = OFFSET = è una costante che indica il numero di istruzioni da saltare.

8. [2] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0300, scrivere l'istruzione assembler di branch on equal, che effettua il salto condizionato all'indirizzo 0x000002C0, e tradurla in linguaggio macchina (il codice operativo dell'istruzione di beq è 4). Si scelgano due registri a piacere.

7. [4] Specificare il contenuto di tutti i bus della CPU riportata in Figura 1 quando è in esecuzione l'istruzione di beq \$0, \$1, ind\_beq, sapendo che il codice operativo dell'istruzione beq è 4 e l'indirizzo di salto (ind\_beq) è 0x0040 0000. Si supponga inoltre che l'indirizzo della beq sia 0x0400 0000. Evidenziare i bus che trasportano dati utili per l'esecuzione dell'istruzione. Tradurre l'istruzione assembler di beq in linguaggio macchina e rappresentare l'istruzione in notazione esadecimale. Quando capisce la CPU se deve saltare o meno? Scrivere in linguaggio macchina l'istruzione di salto incondizionato che consente di saltare allo stesso indirizzo di salto. Il codice operativo dell'istruzione di jump è 2

.

# Cpu SW

sabato 20 luglio 2024 09:38

7. [3] Specificare il contenuto di tutti i bus della CPU riportata in Figura 1 quando è in esecuzione l'istruzione: 0x0000 0048 sw \$1, 16(\$2), sapendo che il codice operativo dell'istruzione sw è 43.

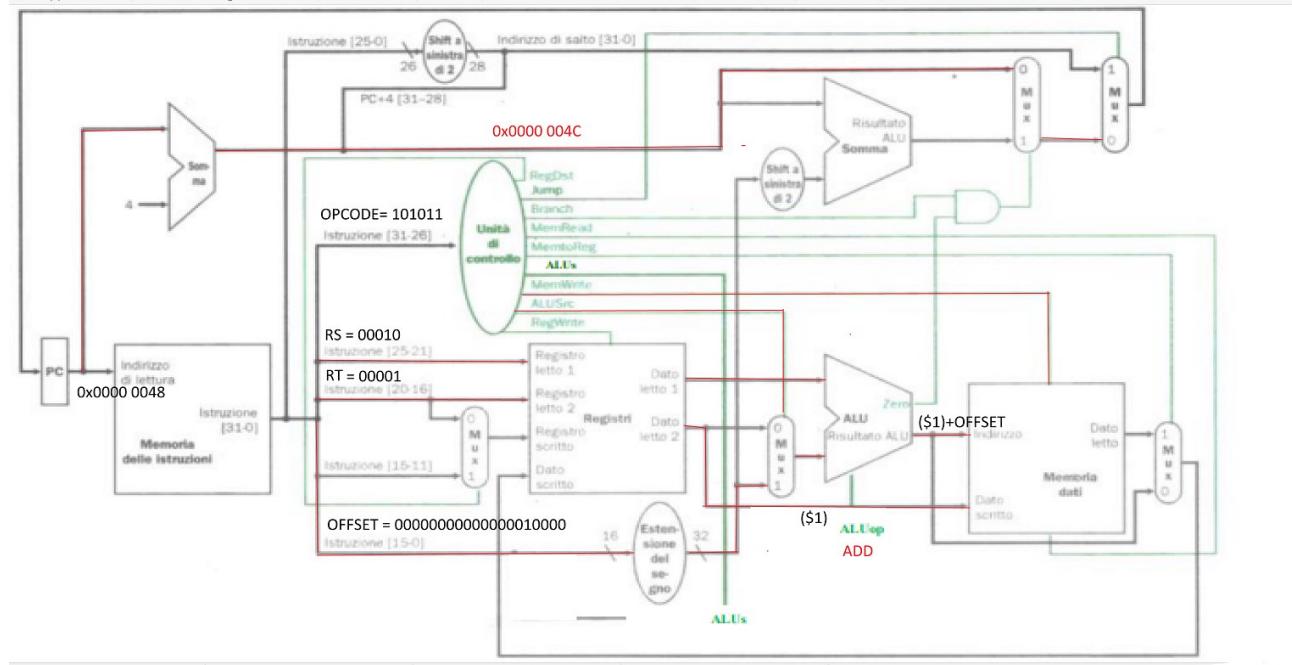
8. [3] Specificare il contenuto di tutti i bus della CPU riportata in Figura 1 quando è in esecuzione l'istruzione di sw \$2, 16(\$1), sapendo che il codice operativo dell'istruzione sw è 43. Evidenziare i bus che trasportano dati utili per l'esecuzione dell'istruzione.

Indirizzo istruzione: 0x0000 0048

Istruzione: sw \$1, 16(\$2) → \$2 e' RS

Codice sw: 43

Istruzione in binario: 101011 00010 00001 00000000000010000



8. [3] Specificare il contenuto di tutti i bus della CPU riportata in Figura 1 quando è in esecuzione l'istruzione di sw \$2, 32(\$1), sapendo che il codice operativo dell'istruzione SW è 43.

# Cpu jump

sabato 20 luglio 2024 10:02

7. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0030, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump) all'indirizzo: 0x0010 0000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di branch che effettua il saldo condizionato all'indirizzo 0x00000028, e tradurla in linguaggio macchina. Si scelgano due registri a piacere.

9. [2] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0048, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump) all'indirizzo: 0x0000 8000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di "branch-on-equal" che effettua il saldo condizionato all'indirizzo 0x00000040, e tradurla in linguaggio macchina, sapendo che il codice operativo dell'istruzione "branch-on-equal" è 4. Si scelgano due registri a piacere.

9. [2] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0044, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump) all'indirizzo: 0x0000 8000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di "branch-on-equal" che effettua il saldo condizionato all'indirizzo 0x00000040, e tradurla in linguaggio macchina, sapendo che il codice operativo dell'istruzione "branch-on-equal" è 4. Si scelgano due registri a piacere.

8. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0034, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump) all'indirizzo: 0x0010 0000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di branch che effettua il saldo condizionato all'indirizzo 0x00000030, e tradurla in linguaggio macchina. Si scelgano due registri a piacere.

7. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0014, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump, codice operativo = 2) all'indirizzo: 0x0000 4000. E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo?

8. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0008, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump, codice operativo = 2) all'indirizzo: 0x0000 4000. E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo?

8. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0028, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump, codice operativo = 2) all'indirizzo: 0x0000 4000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di branch che si trova all'indirizzo 0x0000 0028, e che effettua il saldo condizionato all'indirizzo 0x00000020, e tradurla in linguaggio macchina. Si scelgano due

registri a piacere.

9?. [3] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0048, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump, codice operativo = 2) all'indirizzo: 0x0000 4000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di branch che si trova all'indirizzo 0x0000 0048, e che effettua il saldo condizionato allo stesso indirizzo 0x00000048, e tradurla in linguaggio macchina. Si scelgano due registri a piacere.

9. [2] Supponiamo che l'istruzione corrente si trovi all'indirizzo 0x0000 0028, scrivere l'istruzione assembler e in linguaggio macchina che effettua il salto incondizionato (jump) all'indirizzo: 0x0000 8000.

E' possibile utilizzare un'istruzione di branch per saltare allo stesso indirizzo? Perché?  
Scrivere l'istruzione assembler di branch che si trova all'indirizzo 0x0000 0028, e che effettua il saldo condizionato all'indirizzo 0x00000020, e tradurla in linguaggio macchina. Si scelgano due registri a piacere.

# Register file

sabato 20 luglio 2024 09:56

6. [3] Progettare con le porte logiche la porta di scrittura di un register file con 4 registri da 2 bit e un segnale di reset che consente di portare a zero il numero del registro specificato per la porta di scrittura. E' più opportuno utilizzare flop-flop o latch per i registri di questo register file? Perché? Perché i latch sincroni vengono chiamati "trasparenti"?

5. [6] Progettare con le porte logiche un register file con 1 porta in ingresso, 2 porte in uscita e con 4 registri da 2 bit e un segnale di reset che consente di portare a zero il numero del registro utilizzato dalla porta di scrittura. E' più opportuno utilizzare flop-flop o latch per i registri di questo register file? Perché? Perché i latch sincroni vengono chiamati "trasparenti"?

6. [4] Progettare con le porte logiche un register file a 2 porte in ingresso, 2 porte in uscita con 4 registri da 3 bit. E' più opportuno utilizzare flop-flop o latch per i registri? Perché?

4. [8] Progettare con le porte logiche un register file di 4 registri, ciascuno a 3 bit, completo di 1 una porta di lettura e 1 porta di scrittura. Calcolare complessità e cammino critico.

6. [5] Progettare con le porte logiche un register file a 2 porte in ingresso, 2 porte in uscita con 4 registri da 2 bit. E' più opportuno utilizzare flop-flop o latch per i registri? Perché? Consideriamo un latch di tipo D sincrono, la cui uscita valga 1, quanto deve valere gli ingressi perche' l'uscita vada a zero? Perché i latch sincroni vengono chiamati "trasparenti"?

6. [5] Progettare con le porte logiche un register file a 2 porte in ingresso, 2 porte in uscita con 4 registri da 2 bit. E' più opportuno utilizzare flop-flop o latch per i registri? Perché? Consideriamo un latch di tipo SR, la cui uscita valga 1 e nel quale uno dei due ingressi, inizialmente a 0, passi al valore 1 e questo provoca una commutazione a 0 dell'uscita. Quale dei due ingressi può effettuare questa commutazione? Perché i latch sincroni vengono chiamati "trasparenti"?

6. [6] Progettare con le porte logiche un register file, completo di una porta di lettura e di scrittura, con bistabili three-state costituita da 3 parole di 4 bit. Quale sarà la complessità e cammino critico del circuito? E' più opportuno utilizzare flip-flop o latch per i registri? Perché? Disegnare il circuito di un latch sincrono D e definire cammino critico e complessità.

Consideriamo un latch di tipo SR, la cui uscita valga 1 e nel quale uno dei due ingressi, inizialmente a 0, passi al valore 1 e questo provoca una commutazione a 0 dell'uscita. Quale dei due ingressi può effettuare questa commutazione? Perché i latch sincroni vengono chiamati "trasparenti"?

6. [5] Progettare con le porte logiche un register file a 2 porte in ingresso, 2 porte in uscita con 4 registri da 2 bit.

Quale sarà la complessità e cammino critico del circuito? E' più opportuno utilizzare flip-flop o latch per i registri?

Perché? Disegnare il circuito di un latch sincrono D e definire cammino critico e complessità. Consideriamo un latch di tipo SR, la cui uscita valga 0 e nel quale uno dei due ingressi, inizialmente a 0, passi al valore 1 e questo provoca una commutazione a 1 dell'uscita. Quale dei due ingressi può effettuare questa commutazione? Perché i latch sincroni vengono chiamati "trasparenti"?

6. [3] Progettare con le porte logiche un register file a 2 porte in ingresso, 2 porte in uscita con 3 registri da 2 bit.

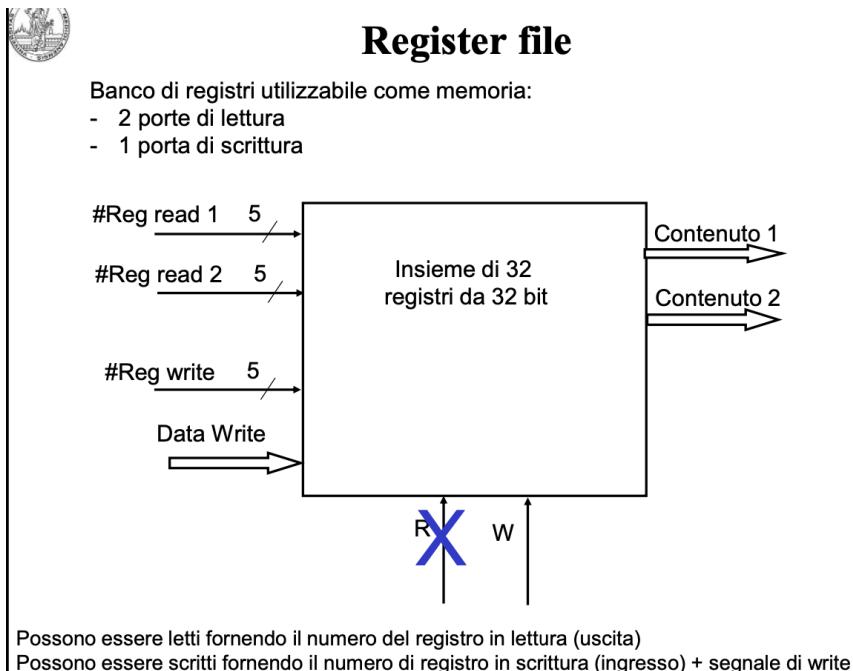
Quale sarà la complessità e cammino critico del circuito? E' più opportuno utilizzare flip-flop o latch per i registri?

Perché?

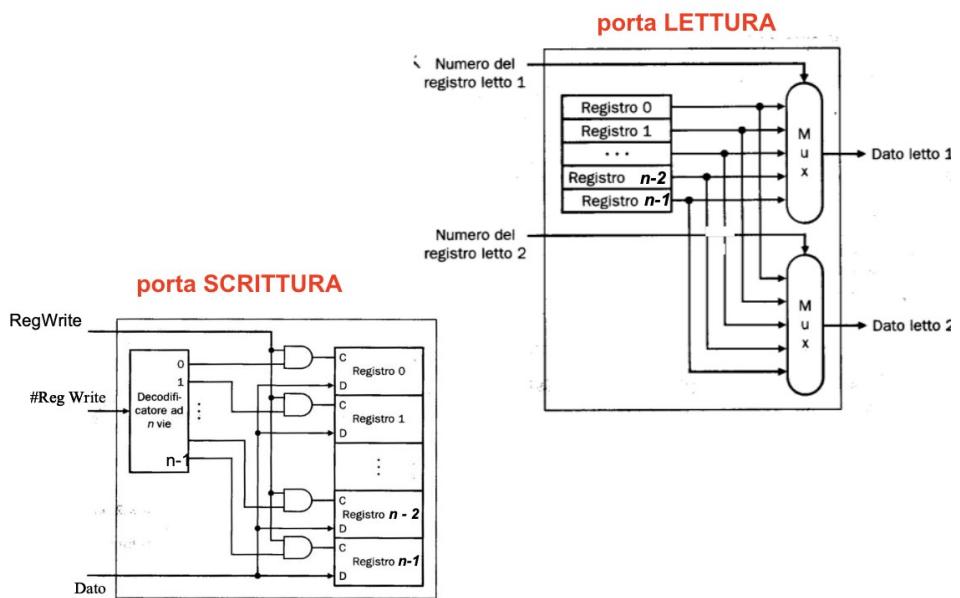
4. [4] Cos'è un register file MIPS? A cosa serve? E' più opportuno utilizzare flop-flop o latch per i registri? Perché? Disegnare il circuito di un latch sincrono e definire cammino critico e la complessità. Scrivere anche la State Transition Table. Perché i latch sincroni vengono chiamati "trasparenti"?

10. [2] Disegnare un banco di registri (numero di registri e ampiezza a vostra scelta) con una porta di lettura e una porta di scrittura.

Mettere solo una porta di lettura



Porta lettura - porta scrittura



# Latch D

sabato 20 luglio 2024 09:41

5. [2] Dato un latch sincrono di tipo D la cui uscita è Q = 0 e l'ingresso e' D= 0.

Cosa succede quando D va a 1?

Quanto tempo viene richiesto per la eventuale commutazione, supponendo che il tempo di commutazione di una porta logica a due ingressi sia 0,1 ps.

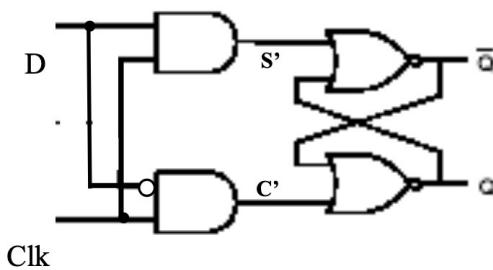
Definire di quali tempi occorre tenere conto per dimensionare il clock di un'architettura. Scrivere la tabella di eccitazione di questo latch.



**Tabella della verità**



$$Q^* = f(T, Q, D)$$



T	D	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$\begin{aligned} Q^* &= \overline{T} \overline{D} Q + \overline{T} D \overline{Q} + T \overline{D} \overline{Q} + T D Q = \\ &= \overline{T} Q + T D \end{aligned}$$

Status quo                       $Q^* = D$

10. [2] Scrivere la tabella di eccitazione di un latch sincroni di tipo D.



**Tabella delle transizioni**



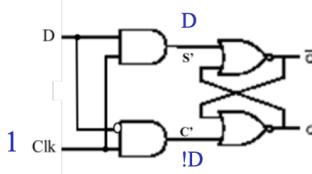
$$Q^* = f(T, Q, D)$$

TQ	D = 0	D = 1
00	0	0
01	1	1
11	0	1
10	0	1

La funzione logica corrispondente è:

$$Q^* = TD + \overline{T}Q$$

$$\begin{array}{ccc} & \nearrow & \searrow \\ Q^* = D & & \text{Status quo} \end{array}$$



$$Q^* = D$$

Q è l'uscita del latch: **stato presente**.

$Q^*$  è il valore dell'uscita al tempo successivo:  
**stato prossimo**.

Come mai qui non si verifica la situazione  $S' = C' = 1$ ?

7. [2]. Consideriamo un latch sincrono di tipo D, la cui uscita valga 1

. Quale deve essere il valore degli ingressi perché l'uscita commuti a 0?

Perché i latch sincroni vengono chiamati "trasparenti"?

I latch sincroni sono spesso chiamati "trasparenti" perché il loro comportamento consente ai dati di passare direttamente dall'ingresso all'uscita quando il segnale di controllo (di solito il clock o il segnale di abilitazione) è in uno stato attivo. In altre parole, quando il latch è "trasparente", i cambiamenti agli ingressi si riflettono immediatamente e direttamente negli output

I latch sincroni vengono chiamati "trasparenti" perché, durante il periodo in cui il segnale di clock è attivo, i dati di ingresso vengono propagati direttamente alle uscite del latch. Ciò significa che le informazioni presenti sulle linee di ingresso del latch vengono trasferite e "rese trasparenti" alle uscite senza alcuna interferenza o blocco.

Durante il periodo di clock attivo, il latch sincrono funziona come un pass-through, consentendo al segnale di ingresso di fluire direttamente alle uscite senza subire modifiche o ritardi significativi. Questa caratteristica "trasparente" rende il latch sincrono particolarmente utile in applicazioni in cui è necessario trasferire dati in modo sincrono e garantire un rapido propagarsi delle informazioni alle uscite.

È importante notare che durante il periodo in cui il segnale di clock è inattivo (ad esempio, quando il clock è basso o in fase di transizione), il comportamento del latch può variare. In questo stato, il latch può mantenere l'ultimo valore di ingresso valido o può essere configurato per mantenere uno stato predefinito specificato. Ciò dipenderà dalla specifica implementazione del latch e dalle impostazioni del circuito associato.

In generale, l'utilizzo di latch sincroni trasparenti semplifica il design dei circuiti sincroni, consentendo un trasferimento rapido dei dati durante il periodo di clock attivo e un funzionamento più prevedibile e stabile nel periodo di clock inattivo.

5. [8] Progettare con le porte logiche una memoria, completa di porta di lettura e di scrittura, con bistabili three-state costituita da 3 parole di 4 bit.

Quale sarà la complessità e cammino critico del circuito?

E' più opportuno utilizzare flip-flop o latch per i registri? Perché?

Disegnare il circuito di un latch sincrono D e definire cammino critico e complessità. Consideriamo un latch di tipo SR, la cui uscita valga 1 e nel quale uno dei due ingressi, inizialmente a 0, passi al valore 1 e questo provoca una commutazione a 0 dell'uscita.

Quale dei due ingressi può effettuare questa commutazione?

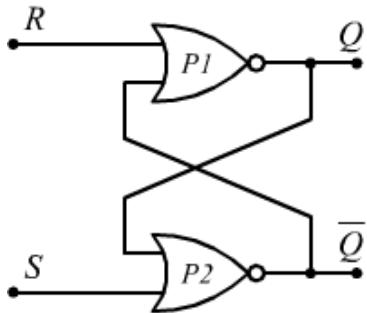
Perché i latch sincroni vengono chiamati "trasparenti"?

Estendere un latch sincrono di tipo SR in modo tale che quando si verifica in ingresso S=R=1, l'uscita commuti.

# Latch sr

sabato 20 luglio 2024 10:07

8. [4] Disegnare il circuito di un **latch asincrono SR** e definire cammino critico e complessità.



Complessità: 2

Cammino critico: 2 tp (dove tp è il tempo di propagazione di una singola porta NOR).

Estendere il circuito in modo tale che quando si verifica in ingresso la condizione  $S = R = 1$ , il circuito commuti l'uscita, cioè  $Q_{t+1} = \bar{Q}_t$ .

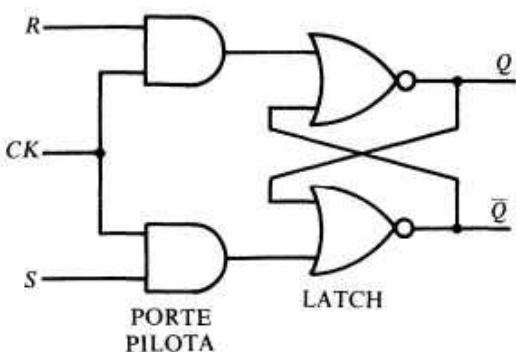
Consideriamo un latch di tipo SR, la cui uscita valga 1 e nel quale uno dei due ingressi, inizialmente a 0, passi al valore 1 e questo provoca una commutazione a 0 dell'uscita.

Quale dei due ingressi può effettuare questa commutazione?

Perché i latch sincroni vengono chiamati "trasparenti"?

I latch sincroni sono spesso chiamati "trasparenti" perché il loro comportamento consente ai dati di passare direttamente dall'ingresso all'uscita quando il segnale di controllo (di solito il clock o il segnale di abilitazione) è in uno stato attivo. In altre parole, quando il latch è "trasparente", i cambiamenti agli ingressi si riflettono immediatamente e direttamente negli output

7. [4] Disegnare il circuito di un latch **SR sincrono** e definire cammino critico e complessità. Supponiamo che entrambi i suoi ingressi valgano 0, se la sua uscita vale 0 e vogliamo portarla a 1, quale degli ingressi deve commutare? Mostrare cosa succede con un diagramma temporale.



Come si può trasformare un latch SR in un latch di tipo D? Perché?

Trasformare un latch SR in un latch di tipo D si ottiene collegando l'ingresso D sia all'ingresso S (Set) che all'ingresso R (Reset) attraverso un NOT gate, garantendo che gli ingressi S e R non siano mai attivi contemporaneamente. Questo elimina lo stato proibito del latch SR e semplifica il design del circuito, rendendo il latch più utile per applicazioni

pratiche nei circuiti digitali.

Come si trasforma un latch SR sincrono in un flip-flop?

### **1. Aggiungere un Clock**

Per trasformare un latch in un flip-flop, dobbiamo sincronizzare l'operazione di memorizzazione con un segnale di clock. Questo viene fatto aggiungendo un circuito di clock che controlla quando il latch deve aggiornare il suo stato.

### **2. Utilizzare Due Latch in Configurazione Master-Slave**

Un flip-flop D può essere costruito utilizzando due latch SR in configurazione master-slave, dove uno dei latch è abilitato durante la fase alta del clock e l'altro durante la fase bassa del clock. Questo assicura che il flip-flop catturi il valore dell'ingresso D solo su un fronte specifico del clock (di solito il fronte di salita).

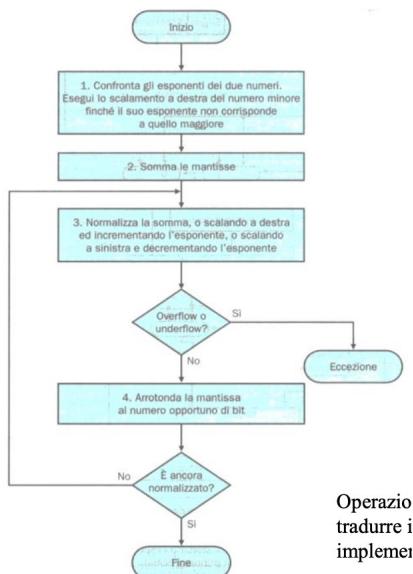
# varie

sabato 20 luglio 2024 10:08

11. [3] Disegnare l'algoritmo e il circuito della somma in virgola mobile.

Sarà un circuito **combinatorio** o sequenziale e perché?

## ALGORITMO



Operazio  
tradurre i  
implemen

## CIRCUITO



## Circuito della somma floating point



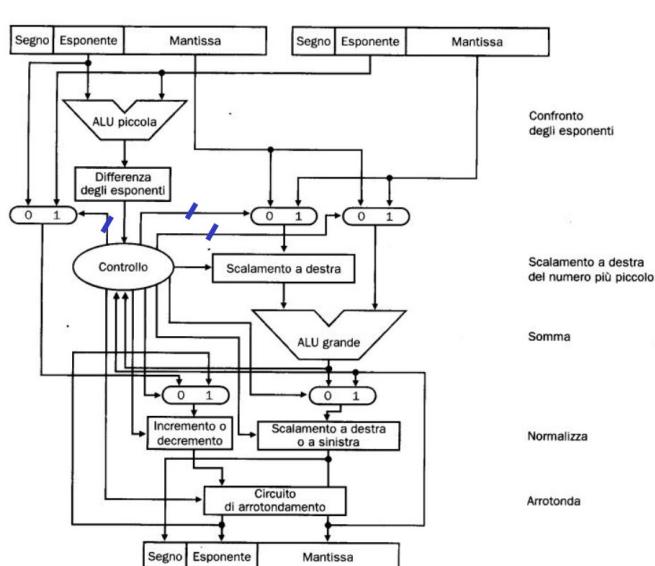
Le tre linee in blu  
contengono lo  
stesso segnale di  
controllo (funzione  
di  $\Delta_{exp}$ ).

Gestisce anche la  
rinormalizzazione:  
 $9,99999 \times 10^2 =$   
 $10,00 \times 10^1$

Gestisce anche i  
numeri negativi.

Problemi?

A.A. 2023-2024



### Circuito Combinatorio:

- L'output dipende solo dagli input attuali.
- Non ha memoria di stati precedenti.
- Esempi: addizionatori, moltiplicatori, comparatori, ecc.

La somma in virgola mobile deve eseguire una serie di operazioni che possono essere tutte realizzate mediante logica combinatoria:

1. **Allineamento degli Esponenti:** Determina quale delle mantisse deve essere shiftata e di quanto. Questo è fatto usando comparatori e shifters (componenti combinatori).
2. **Somma delle Mantisse:** Usa addizionatori che sono circuiti combinatori.
3. **Normalizzazione:** Richiede shifters e addizionatori per aggiustare l'esponente, anch'essi circuiti combinatori.

1. [4] In Figura 1 viene riportato il modulo di una ALU a 1 bit. Collegare 4 moduli per ottenere una ALU a 4 bit che sia in grado di eseguire anche le operazioni di shift a dx e shift a sx di una posizione dell'input A. Indicare e spiegare le modifiche richieste.



archi1tutto.pdf

# Storia elaboratori

giovedì 23 gennaio 2025 11:04

L'evoluzione dei calcolatori ebbe inizio con l'**abaco babilonese** nel X secolo, seguito dalle **prime architetture pneumatiche** con comandi a vapore, come quelle proposte da Charles Babbage, e, verso la fine del 1800, dalle calcolatrici da tavolo meccaniche. Nei primi anni del 1900, con la nascita di IBM, si sviluppò la **prima generazione** di calcolatori, basata sull'uso delle schede perforate lette da macchine elettromeccaniche.

Un momento fondamentale arrivò nel 1945 con la proposta della **macchina di Von Neumann**, che introdusse un'architettura rivoluzionaria per i computer, basata sull'idea di un'unica memoria per i dati e le istruzioni, che potevano essere eseguite sequenzialmente. Questo modello è alla base dei computer moderni e ha posto le fondamenta per lo sviluppo delle generazioni successive.

La **seconda generazione** iniziò nel 1944 con il primo **computer automatico** basato su elettronica a diodi, che incrementò le prestazioni di 1000 volte. Nella seconda metà del 1900 si assistette all'introduzione dell'elettronica allo stato solido e delle memorie ferromagnetiche, aprendo la strada a una nuova era. La **terza generazione**, nel 1964, portò all'adozione dei **circuiti integrati** (LSI), mentre la **quarta generazione** vide l'introduzione del **microprocessore**, realizzato con tecnologia VLSI, e delle **memorie a semiconduttori**.

Nel 1972 fu realizzata la prima calcolatrice tascabile. Con la **quinta generazione** nacquero i **personal computer** (PC). Il primo PC, prodotto da IBM nel 1981, utilizzava il sistema operativo DOS sviluppato da Microsoft e il processore Intel 8086, supportato dal coprocessore matematico Intel 8087. Nel 1987 fu lanciato Windows 1.0, che segnò un'importante evoluzione nell'interfaccia utente.

La **sesta generazione** è caratterizzata da tecnologie come **Arduino**, **microprocessori avanzati** e **schede grafiche potenti**, che hanno ampliato le possibilità applicative dei calcolatori nei campi dell'Internet delle cose, dell'intelligenza artificiale e del calcolo grafico avanzato.

Guardando al **futuro**, i **computer quantistici** rappresentano una nuova frontiera dell'informatica. Questi dispositivi, basati sui principi della meccanica quantistica, sfruttano i qubit per eseguire calcoli paralleli su una scala impossibile per i computer tradizionali. Grazie a questa capacità, i computer quantistici promettono di risolvere problemi complessi in ambiti come la crittografia, la simulazione di sistemi fisici e chimici, e l'ottimizzazione avanzata. Anche se ancora in fase sperimentale, i computer quantistici rappresentano un cambiamento epocale nella storia dell'informatica.

# Dimesionare clock

giovedì 23 gennaio 2025 12:15

## Quali elementi occorre tenere conto per dimensionare il clock di un'architettura?

Per dimensionare il clock di un'architettura, occorre tenere conto di diversi elementi chiave:

**Requisiti di prestazioni:** È necessario considerare le specifiche di prestazioni richieste per l'architettura. Ciò include il tempo massimo consentito per l'esecuzione di istruzioni critiche, il throughput richiesto e la frequenza massima di operazione desiderata.

**Complessità del circuito:** La complessità complessiva del circuito, inclusi il numero e la complessità delle porte logiche, dei registri, delle unità funzionali e dei percorsi critici, influisce sulla frequenza massima del clock. Circuiti più complessi richiedono più tempo per propagare i segnali e quindi richiedono un clock più lento.

**Tecnologia e implementazione:** La tecnologia di fabbricazione utilizzata e la specifica implementazione influenzano la frequenza massima del clock. Fattori come la dimensione dei transistor, la capacità dei cavi di interconnessione, la resistenza e la capacità del carico, la qualità del segnale e la dissipazione di potenza possono influire sulla velocità del circuito e quindi sulla frequenza di clock.

**Interferenze elettriche e rumore:** Le interferenze elettriche, come il rumore del segnale, il ritardo di propagazione e l'accoppiamento capacitivo o induttivo, possono influire sulla frequenza massima del clock. È necessario tenere conto di questi fattori e adottare misure appropriate per ridurre l'effetto del rumore e delle interferenze sull'accuratezza del clock.

**Tolleranza agli errori:** Se l'architettura richiede una maggiore tolleranza agli errori, ad esempio in ambienti ad alta radiazione o con alta variabilità di temperatura, potrebbe essere necessario ridurre la frequenza di clock per garantire la stabilità e l'affidabilità del sistema.

# Crocette

lunedì 18 novembre 2024 11:01

1. Data una funzione con 3 input, U1, U2, U3  
e 2 output: Y1,Y2 determinare se:

$$Y_1 = U_1 (\bar{U}_2) + U_2 U_3$$

$$Y_2 = (U_1 + U_2)(U_1 + \bar{U}_3)$$

Sia una forma canonica della funzione.

A. Si'

**B. No**

C. Solo per Y1

D. Solo per Y2

E. Non si può decidere

2. Il numero  $2^{34}$  non si può codificare in IEEE754.

A. Vero

**B. Falso**

C. Vero se le parole sono a 32 bit

D. Falso se le parole sono a 64 bit

E. E' un numero intero quindi non si può codificare in IEEE754.

3. Può una sottrazione generare overflow aritmetico?

A. No, mai.

B. Si', ma solo se il primo termine è minore di zero.

C. Si', ma solo se il secondo termine e'  
minore di zero.

**D. Si', ma solo se i segni dei due operandi  
sono discordi**

E. Si', ma solo se i segni dei due operandi sono concordi.

4.  $Y = ABC + \bar{A}BC + A\bar{B}C + AB\bar{C} + \bar{A}\bar{B}\bar{C} + A\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$  è  
equivalente a:

A.  $Y = A$

B.  $Y = B$

C.  $Y = C$

**D. Y = 1**

E.  $Y = 0$

5. L'implementazione di una funzione logica  
mediante maxtermini (senza semplificazione)  
ha la stessa complessità  
dell'implementazione mediante mintermini.

A. Si'

- B. No
- C. Si' solo se il numero di variabili in ingresso e' pari.

**D. Si' solo se il numero di mintermini e'  
uguale al numero di maxtermini**

- E. Si' solo se la funzione è dispari

6. Si può calcolare il cammino critico associato a un flip-flop?

**A. Si'**

- B. No
- C. Si' ma solo se è costituito da latch di tipo SR
- D. Si' ma solo se e' costituito da latch di tipo D
- E. Non ha senso perche' il flip-flop commuta sul fronte di clock

7. La frequenza di un segnale (ad esempio il segnale di clock) è l'inverso del periodo

**A. Si'**

- B. No
- C. Si', ma solo se il segnale è una sinusoide
- D. Si', ma solo se il segnale e' una cosinusoide
- E. Si, ma solo se il segnale è il segnale di clock

8. Un moltiplicatore HW svolge una moltiplicazione con un cammino critico che dipende

**A. Dall'ampiezza della parola**

- B. Dal numero di bit diversi da zero del moltiplicatore
- C. Dal numero di bit diversi da zero del moltiplicando
- D. Dalla tecnologia utilizzata per implementare il moltiplicatore
- E. Dall'ampiezza della parola e dal cammino critico di un circuito che calcola l'overflow

9. In una ALU, l'implementazione dell'istruzione di "set on less than", richiede che

- A. La ALU svolga l'operazione di somma e il riporto del sommatore dell'ultimo bit venga portato in ingresso al multiplexer di uscita.
- B. La ALU svolga l'operazione di differenza e il bit di somma del sommatore dell'ultimo bit venga portato in ingresso al multiplexer in uscita.**
- C. La ALU svolga l'operazione di differenza e il riporto del sommatore dell'ultimo bit venga portato in ingresso al multiplexer di uscita.
- D. La ALU svolga l'operazione di somma e il

bit di somma del sommatore dell'ultimo bit venga portato in ingresso al multiplexer di uscita.

E. Mette a 1 l'uscita se il bit di segno dei due operandi e' discorde.

10. La funzione  $Y = (A \text{ XOR } B)$  può essere implementata

- A. Con una SOP come:  $Y = !AB + !A!B$
- B. Con una POS come:  $Y = (!A + !B)(A + B)$
- C. Con una porta NAND come:  $Y = !(A \text{ NAND } B)$
- D. Con una porta NOR come:  $Y = !(A \text{ NOR } B)$
- E. Con un AND negato due volte:  $!( !(A \text{ AND } B) )$

# RIEPILOGO

lunedì 21 luglio 2025 13:34

1. [5] Convertire in binario secondo la codifica IEEE754, il numero decimale 16,25.

Convertiamo 16 in binario: 10000

Convertiamo 0,25 in binario: La parte frazionaria si converte in 0,01 in binario.

Combinare: 10000,01

Normalizzare: Il numero binario 10000,01 può essere scritto come 1,000001 $\times 2^4$

**SEGO 0**

**ESPOENTE** 4+127=131 -> 10000011

**MANTISSA** La mantissa è la parte frazionaria della rappresentazione normalizzata, dopo il punto binario. La parte frazionaria è 000001. Per IEEE 754, la mantissa è di 23 bit, quindi dobbiamo aggiungere zeri alla fine.

0000010000000000000000000

Quale sarà la rappresentazione binaria in IEEE754 del numero decimale ottenuto sommando una unità (1,0) al numero 16,25?

Codifica di 17,25

Qual è la risoluzione della codifica in virgola mobile IEEE 754, della codifica intera e della codifica in virgola fissa?

In virgola mobile IEEE 754 la risoluzione variabile:

Per numeri piccoli la risoluzione è alta.

Supponiamo di rappresentare 1,0000000000000000000000000000000 e successivo 1,0000000000000000000000000000001 = 2^-23

Per numeri grandi la risoluzione diminuisce.

Supponiamo di rappresentare 128 -> 1.0000000 \* 2^7 la risoluzione diminuisce 2^-23 + 2^7 = 2^-16

Anche per 129 -> 1.0000001 \* 2^7 fino a 255 -> 1.1111111 \* 2^7 incluso sarà sempre 2^-23 + 2^7 = 2^-16

Se volessi rappresentare 256 -> 1.00000000 \* 2^8 la risoluzione scende 2^-23 + 2^8 = 2^-15

511 -> 1.1111111 \* 2^8 risoluzione 2^-23 + 2^8 = 2^-15

512 -> 1.00000000 \* 2^9 risoluzione 2^-23 + 2^9 = 2^-14

Fino a 1.11111111 \* 2^9 = 1023 risoluzione 2^-23 + 2^9 = 2^-14

In codifica intera è costante 1 tra un numero e l'altro.

In virgola fissa la risoluzione è costante ma diversa da 1 ed è determinata da quanti bit dedichi alla parte frazionaria.

Risoluzione=2^-f dove f è il numero di bit riservati alla parte frazionaria.

Esempio: con 3 bit parte frazionaria, la risoluzione è 2^-3

Scrivere in complemento a 2 su 8 bit la sottrazione espressa da numeri in base 10: 7-9 e calcolarne il risultato in binario.

7 -> 00000111

9 -> 00001001

-9 -> 11110111

7+(-9) = -2 ----> 11111110

Quale coppia di numeri decimali ha una distanza pari a 1 LSB di un numero binario codificato in IEEE 754?

La prima coppia di numeri decimali consecutivi in formato IEEE 754 float (32 bit) che dista esattamente 1.0 è:

$2^{23} = 8.388.608$  e  $2^{23} + 1 = 8.388.609$

A partire dalla codifica binaria, scrivere la codifica esadecimale dello stesso numero.

Tabella di conversione

binario	esadecimale	binario	esadecimale
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

(0100 1110 1100 1000)<sub>2</sub>

Esistono delle condizioni in cui la conversione da binario a decimale non risulta esatta?

La conversione **non è esatta** ogni volta che il numero decimale ha una **rappresentazione binaria periodica infinita**. Un numero decimale **può essere rappresentato esattamente in binario solo se** può essere scritto come:

$k/2^n$  con k, n appartenenti a Z

Esempi **esatti**:

- 0.5
- 0.25
- 0.75
- 0.125

Esempi **non esatti**:

- 0.1
- 0.2
- 0.3
- 0.15

Qual è il numero intero più grande e il numero intero più piccolo rappresentabile su 8 bit in complemento a 2?

Nel sistema **in complemento a 2 su 8 bit**, i numeri interi **firmati** sono rappresentati con:

- 1 bit di segno
- 7 bit di valore

Più grande 01111111 -> 127

Più piccolo 10000000 -> -128

Come viene rappresentata la situazione +oo, -oo e NaN in IEEE754?

**+∞**

- **Segno**: 0
- **Esponente**: 11111111 (tutti 1)
- **Mantissa**: 000...000 (tutti 0)

**-∞**

- **Segno**: 1
- **Esponente**: 11111111
- **Mantissa**: 000...000

**Nan**

- **Segno**: indifferente
- **Esponente**: 11111111 (tutti 1)
- **Mantissa**: basta un solo bit a 1 in tutta la mantissa

**Google Brain Format?**

Come per ieee754 ma TRONCO LA MANTISSA A 8 BIT

Quindi avrò 1 bit segno - 8 bit esponente - 8 bit mantissa

**Numero Denormalizzato**

In ieee754 un numero è denormalizzato quando ha esponente pari a 00000000  
l'esponente viene interpretato come -126

La mantissa è rappresentata dai successivi 23 bit  
Questa codifica serve per rappresentare numeri molto piccoli

Più piccolo numero positivo normalizzato rappresentabile in formato ieee754:

Il successivo più piccolo:

Numero più piccolo positivo denormalizzato rappresentabile in formato ieee754:

**Successivo:**

**Numero più grande positivo rappresentabile:**

Number using IEEE floating-point representation: