CALCOLATORI

Esercizi

Giovanni Iacca
giovanni.iacca@unitn.it

Lezione basata su materiale preparato dal Prof. Marco Roveri

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Reti logiche

- Individuare tra le opzioni seguenti l'espressione logica equivalente a $x \cdot (y + z) + \overline{x + \overline{y}}$
  - $z \cdot \overline{z} + \overline{x} \qquad = \overline{x}$
  - $x \cdot z + y$
  - $x \cdot y + \overline{x} \cdot z$
  - 1
- Semplifichiamo le diverse opzioni se possibile: $A \cdot \overline{A} = 0$, $0 + A = A$
- Riscriviamo $x \cdot (y + z) + \overline{x + \overline{y}}$ usando De Morgan ($\overline{A + B} = \overline{A} \cdot \overline{B}$, $\overline{A \cdot B} = \overline{A} + \overline{B}$), distributività di $\cdot$ ($A \cdot (B + C) = A \cdot B + A \cdot C$), e doppia negazione ($\overline{\overline{A}} = A$):
  - $x \cdot (y + z) + \overline{x + \overline{y}} = x \cdot y + x \cdot z + \overline{x} \cdot \overline{\overline{y}} = x \cdot y + x \cdot z + \overline{x} \cdot y$
- Sfruttiamo equivalenza $\overline{A} \cdot B + A \cdot B = B$ per semplificare la formula
  - $x \cdot y + x \cdot z + \overline{x} \cdot y = y + x \cdot z$
- Trovato che la seconda opzione è quella corretta (modulo ordine degli operandi)

Reti logiche

- Individuare tra le opzioni seguenti l'espressione logica equivalente a $x \cdot y + \overline{x + \overline{y}} + \overline{\overline{x} + \overline{y}}$
  - $\overline{x + \overline{y}} \quad = \overline{x} \cdot y$
  - $1 + \overline{x + \overline{y}} \quad = 1 + \overline{x} \cdot y = 1$
  - $x \cdot y$
  - y
- Semplifichiamo opzioni usando De Morgan e doppia negazione
- Riscriviamo $x \cdot y + \overline{x + \overline{y}} + \overline{\overline{x} + \overline{y}}$ usando De Morgan, $A + A = A$:
  - $x \cdot y + \overline{x + \overline{y}} + \overline{\overline{x} + \overline{y}} = x \cdot y + \overline{x} \cdot y + x \cdot y = x \cdot y + \overline{x} \cdot y$
- Usando distributività inversa
  - $x \cdot y + \overline{x} \cdot y = (x + \overline{x}) \cdot y$
- Usando $A + \overline{A} = 1$ e idempotenza
  - $(x + \overline{x}) \cdot y = 1 \cdot y = y$
- Trovato che la quarta opzione è quella corretta

Reti logiche

- Individuare tra le opzioni seguenti l'espressione logica equivalente a $x \cdot y + \overline{x + \overline{y}} + \overline{x} + y$
  - $x + \overline{y}$
  - $0 + \overline{x + \overline{y}}$ $= 0 + \overline{x} \cdot y = \overline{x} \cdot y$
  - $\overline{x} + y$
  - $x \cdot y$
- Semplifichiamo opzioni usando De Morgan e doppia negazione
- Riscriviamo $x \cdot y + \overline{x + \overline{y}} + \overline{x} + y$ usando De Morgan, idenpotenza, distributività:
  - $x \cdot y + \overline{x + \overline{y}} + \overline{x} + y = x \cdot y + \overline{x} \cdot y + \overline{x} + y$
  - $x \cdot y + \overline{x} \cdot y + \overline{x} + y = x \cdot y + \overline{x} \cdot (y + 1) + y$
  - $x \cdot y + \overline{x} \cdot (y + 1) + y = x \cdot y + \overline{x} + y$
  - $x \cdot y + \overline{x} + y = (x + 1) \cdot y + \overline{x} = y + \overline{x}$
- Trovato che la terza opzione è quella corretta

Conversione da intero a binario

- Convertire 728_{10} in binario

728	0
364	0
182	0
91	1
45	1
22	0
11	1
5	1
2	0
1	1

Risultato: 1011011000_2

Prova: $2^9 + 2^7 + 2^6 + 2^4 + 2^3$
$= 512 + 128 + 64 + 16 + 8 = 728$

Conversione da intero a binario

- Convertire 3249_{10} in binario

3249	1
1624	0
812	0
406	0
203	1
101	1
50	0
25	1
12	0
6	0
3	1
1	1

Risultato: 110010110001

Prova: $2^{11} + 2^{10} + 2^7 + 2^5 + 2^4 + 2^0$
$= 2048+1024+128+32+16+1 = 3249$

Somma tra numeri interi

- Convertire in numeri binari 623_{10} e 412_{10} e farne la somma in binario.

623	1
311	1
155	1
77	1
38	0
19	1
9	1
4	0
2	0
1	1

412	0
206	0
103	1
51	1
25	1
12	0
6	0
3	1
1	1

$$
\begin{array}{ccccccccccc c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & + \\
 & & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & = \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 &
\end{array}
$$

$$2^{10} + 2^3 + 2^1 + 2^0 = 1024 + 8 + 2 + 1 = 1035$$

Somma tra numeri interi

- Convertire in numeri binari 680_{10} e 378_{10} e farne la somma in binario.

$$
\begin{array}{r|l}
680 & 0 \\
340 & 0 \\
170 & 0 \\
85 & 1 \\
42 & 0 \\
21 & 1 \\
10 & 0 \\
5 & 1 \\
2 & 0 \\
1 & 1
\end{array}
\qquad
\begin{array}{r|l}
378 & 0 \\
189 & 1 \\
94 & 0 \\
47 & 1 \\
23 & 1 \\
11 & 1 \\
5 & 1 \\
2 & 0 \\
1 & 1
\end{array}
$$

$$
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & \\
 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & + \\
 & & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & = \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 &
\end{array}
$$

$$2^{10} + 2^5 + 2^1 = 1024 + 32 + 2 = 1058$$

Moltiplicazione tra numeri interi positivi

- Convertire 21_{10} e 11_{10} in numeri binari e farne la moltiplicazione in binario.

$$
\begin{array}{r|l}
21 & 1 \\
10 & 0 \\
5 & 1 \\
2 & 0 \\
1 & 1
\end{array}
\qquad
\begin{array}{r|l}
11 & 1 \\
5 & 1 \\
2 & 0 \\
1 & 1
\end{array}
$$

$$
\begin{array}{cccccccccc}
 & & & 1 & 0 & 1 & 0 & 1 & * \\
 & & & & 1 & 0 & 1 & 1 & = \\
\hline
 & & & 1 & 0 & 1 & 0 & 1 & \\
 & & 1 & 0 & 1 & 0 & 1 & & \\
 & 0 & 0 & 0 & 0 & 0 & & & \\
1 & 0 & 1 & 0 & 1 & & & & \\
\hline
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 &
\end{array}
$$

$$
2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0 =
$$
$$
128 + 64 + 32 + 4 + 2 + 1 = 231
$$

Moltiplicazione tra numeri interi positivi

- Convertire 15_{10} e moltiplicarlo con se stesso in modo binario.

15	1
7	1
3	1
2	1

				1	1	1	1	*	
				1	1	1	1	=	
				1	1	1	1		
			1	1	1	1			
		1	0	1	1	0	1		Somma parziale
		1	1	1	1				
	1	1	0	1	0	0	1		Somma parziale
	1	1	1	1					
1	1	1	0	0	0	0	1		Somma parziale e risultato

$$2^7 + 2^6 + 2^5 + 2^0 =$$
$$128 + 64 + 32 + 1 = 225$$

Moltiplicazione tra numeri interi positivi

- Eseguire il prodotto tra numeri binari seguente: $1010110_2 * 1001110_2$.

						1	0	1	0	1	1	0	*	
						1	0	0	1	1	1	0	=	
					1	0	1	0	1	1	0	-		
				1	0	1	0	1	1	0	-	-		
			1	0	0	0	0	0	0	1	0	0		Somma parziale
			1	0	1	0	1	1	0	-	-	-		
		1	0	0	1	0	1	1	0	1	0	0		Somma parziale
1	0	1	0	1	1	0	-	-	-	-	-	-		
1	1	0	1	0	0	0	1	1	0	1	0	0		Somma parziale e risultato

$$2^{12} + 2^{11} + 2^9 + 2^5 + 2^4 + 2^2 =$$
$$4096 + 2048 + 512 + 32 + 16 + 4 = 6708_{10}$$
$$1010110_2 = 2^6 + 2^4 + 2^2 + 2 = 86_{10}$$
$$1001110_2 = 2^6 + 2^3 + 2^2 + 2 = 78_{10}$$
$$86_{10} * 78_{10} = 6708_{10}$$

Somma tra numeri interi in complemento a 2

- Convertire -34_{10} e -53_{10} in numeri binari su 8 bit e farne la somma in binario.

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

53	1
26	0
13	1
6	0
3	1
1	1

34		0	0	1	0	0	0	1	0	˜
		1	1	0	1	1	1	0	1	$+\leftarrow$
									1	$+\leftarrow$
53		0	0	1	1	0	1	0	1	˜
		1	1	0	0	1	0	1	0	$+\leftarrow$
									1	$+\leftarrow$
-87	1	1	0	1	0	1	0	0	1	=
		0	1	0	1	0	1	1	0	˜
									1	+
87		0	1	0	1	0	1	1	1	=

Somma tra numeri interi in complemento a 2

- Convertire -50_{10} e -80_{10} in numeri binari su 8 bit e farne la somma in binario.

50	0
25	1
12	0
6	0
3	1
1	1

80	0
40	0
20	0
10	0
5	1
2	0
1	1

50		0	0	1	1	0	0	1	0	˜
		1	1	0	0	1	1	0	1	$+\leftarrow$
									1	$+\leftarrow$
80		0	1	0	1	0	0	0	0	˜
		1	0	1	0	1	1	1	1	$+\leftarrow$
									1	$+\leftarrow$
126	1	0	1	1	1	1	1	1	0	=

Sommo due numeri negativi ottengo un numero positivo

Conversione da decimale a binario

- Convertire 3.5_{10} in binario
  - Parte intera: $3_{10} \rightarrow 011_2$
  - Parte frazionaria: $0.5_{10} \rightarrow 2^{-1} \rightarrow 0.1_2$
  - Risultato: 11.1_2
- Prova inversa:
  - $11_2 \rightarrow 2^1 + 2^0 \rightarrow 2 + 1 \rightarrow 3_{10}$
  - $0.1_2 \rightarrow 2^{-1} \rightarrow 0.5_{10}$
  - Risultato: $3 + 0.5 = 3.5_{10}$

Conversione da decimale a binario

- Convertire 231.71875_{10} in binario

231	1
115	1
57	1
28	0
14	0
7	1
3	1
1	1

0.71875	$\downarrow$
0.43750	1 (sottraggo 1)
0.87500	0
0.75000	1 (sottraggo 1)
0.50000	1 (sottraggo 1)
0.00000	1 (sottraggo 1)

11100111.10111

Conversione da binario a decimale

- Convertire 100110.0011_2 in decimale
  - Parte intera: $2^5 + 2^2 + 2^1 \rightarrow 32 + 4 + 2 = 38$
  - Parte frazionaria: $2^{-3} + 2^{-4} \rightarrow 0.125 + 0.0625 = 0.1875$
  - Risultato: 38.1875_{10}
- Prova inversa:
  - $38_{10} \rightarrow 2^5 + 2^2 + 2^1 \rightarrow 100110$
  - $0.1875_{10} \rightarrow 0.0011$

0.1875	$\downarrow$
0.3750	0
0.7500	0
0.5000	1 (sottraggo 1)
0.0000	1 (sottraggo 1)

  - Risultato: $100110_2 + 0.0011_2 = 100110.0011_2$

Operazioni in virgola fissa

- Riportare in binario la differenza $11000.1011_2 - 111.111101_2$

									0	1		
									0	1		
								0	0	1		
						0	1	0	0	1		
						0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
	0	1	1	1		0	1	0	0	1		
1	1	0	0	0	.	1	0	1	1			-
		1	1	1	.	1	1	1	1	0	1	=
					.							
					.						1	

Conversione IEEE754

- Convertire il numero -10.75_{10} in floating point a 32 bit (singola precisione)

10	0
5	1
2	0
1	1

0.75	$\downarrow$
0.50	1 (sottraggo 1)
0.	1 (sottraggo 1)
0.	

- Quindi $-1010.11 \rightarrow -1.01011_2 * 2^3$ in notazione scientifica
- $(-1)^s * (1 + Mantissa) * 2^{Esponente-127}$
  - $s = 1$
  - Mantissa $= 1.01011 - 1 = 0.01011$
  - $3 = (Esponente - 127)$ quindi Esponente = $130 \rightarrow 10000010$

s	Esp. 8bit	Mantissa 23 bit
1	10000010	01011000000000000000000

Conversione IEEE754

- Convertire il numero 0.1875_{10} in floating point a 32 bit (singola precisione)

$0 \mid 0$

0.1875	$\downarrow$
0.3750	0
0.7500	0
0.5000	1 (sottraggo 1)
0.0000	1 (sottraggo 1)

- Quindi $0.0011 \rightarrow 1.1_2 * 2^{-3}$ in notazione scientifica
- $(-1)^s * (1 + Mantissa) * 2^{Esponente-127}$
  - $s = 0$
  - Mantissa $= 1.1 - 1 = 0.1$
  - $-3 = (Esponente - 127)$ quindi Esponente = $124 \rightarrow 01111100$

s	Esp. 8bit	Mantissa 23 bit
0	01111100	10000000000000000000000

Conversione IEEE754

- Convertire il numero IEEE754 $0x427d0000_{16}$ in decimale virgola mobile
  - $0x427d0000_{16}$ =0100 0010 0111 1101 0000 0000 0000 0000_2
    $0 \mid 10000100 \mid 11111010000000000000000_2$
  - $N = (-1)^s * (1 + Mantissa) * 2^x$ dove $x = Esponente - 127$
    - $s = 0$
    - Esponente = $10000100 = 2^7 + 2^2 = 132$
      $x = Esponente - 127 = 132 - 127 = 5$
    - Mantissa = $11111010000000000000000 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} = 0.9765625$
    - $N = (-1)^0 * (1 + 0.9765625) * 2^5 = 1 * 1.9765625 * 32 = 63.25$

Conversione IEEE754

- Convertire il numero IEEE754 $0x0C000000_{16}$ in decimale virgola mobile
  - $0x0C000000_{16}$ =0000 1100 0000 0000 0000 0000 0000 0000
    $0 \mid 00011000 \mid 00000000000000000000000_2$
  - $N = (-1)^s * (1 + Mantissa) * 2^x$ dove $x = Esponente - 127$
    - $s = 0$
    - Esponente = $00011000 = 2^4 + 2^3 = 24$
      $x = Esponente - 127 = 24 - 127 = -103$
    - Mantissa = $00000000000000000000000 = 0$
    - $N = (-1)^0 * (1 + 0) * 2^{-103} = 2^{-103}$

Esempio RISC-V

```c
typedef long long int int64;
{ // Indirizzo di MemVett in x10
  int64 Ris = 0;
  for (int64 i=0; i<100; i++)
    Ris += MemVett[i];
}
```

$\longrightarrow$

```
      addi x5, x0, 0
      addi x6, x0, 0
      addi x29, x0, 100
CIC:  ld   x7, 0(x10)
      add  x5, x5, x7
      addi x10, x10, 8
      addi x6, x6, 1
      blt  x6, x29, CIC
```

Esempio RISC-V

Quale espressione C corrisponde alle seguenti istruzioni RISC-V? Si assuma che le variabili f, g, h, i, j siano assegnate ai registri x5, x6, x7, x28, x29, rispettivamente, che A e B siano array di double (8 bytes), e che il loro indirizzo base sia nei registri x10 e x11.

```
slli x30, x5, 3
add  x30, x10, x30
slli x31, x6, 3
add  x31, x11, x31
ld    x5, 0(x30)
addi x12, x30, 8
ld   x30, 0(x12)
add  x30, x30, x5
sd   x30, 0(x31)
```

$\longrightarrow$

```
slli x30, x5, 3      // x30 = f*8
add  x30, x10, x30   // x30 = &A[f]
slli x31, x6, 3      // x31 = g*8
add  x31, x11, x31   // x31 = &B[g]
ld    x5, 0(x30)     // x5 = A[f]
addi x12, x30, 8     // x12 = &A[f] + 8 = &A[f+1]
ld   x30, 0(x12)     // x30 = A[f+1]
add  x30, x30, x5    // x30 = A[f+1]+A[f]
sd   x30, 0(x31)     // B[g] = A[f+1]+A[f]
                     // B[g] = A[f+1]+A[f]
```

Esempio RISC-V

```c
// a0 -> x, a1 -> y,
// t0 -> result
// Function computes pow(x,y)
// Direct translation:
typedef long long int int64;
int64 power(int64 x, int64 y) {
  int64 result = 1;
  while (y & y != 0) {
    result *= x;
    y--;
  }
  return result;
}
```

→

```
power: addi t0, x0, 1
Loop:  and t1, a1, a1
       beq t1, x0, Done
       mul t0, t0, a0
       addi a1, a1, -1
       jal x0, Loop
Done:  add a0, t0, x0
       jalr x0, 0(ra)
```

Esempio RISC-V

- Assumendo che il registro `x10` contenga il valore `0x10000000000000FF`. Quale sarà il contenuto del registro `x10` dopo aver eseguito le istruzioni assembly seguenti?

```
addi x11, x0, 15
sll  x11, x11, 28
or   x10, x11, x10
```

- `x10 = 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1111 1111`
- `15` $\rightarrow$ `1111`
- **addi** x11, x0, 15

  `x11 = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1111`
- **sll** x11, x11, 28

  `x11 = 0000 0000 0000 0000 0000 0000 0000 0000 1111 0000 0000 0000 0000 0000 0000 0000`
- **or** x10, x11, x10

  `x10 = 0001 0000 0000 0000 0000 0000 0000 0000 1111 0000 0000 0000 0000 0000 1111 1111`

Esempio RISC-V

- Assumendo che il registro `x10` contenga il valore `255.125` espresso secondo lo standard IEEE754 a 32 bit (esteso con zeri). Quale sarà il contenuto del registro `x10` dopo aver eseguito le istruzioni assembly seguenti?

```
addi x11, x0, 4096
sll  x11, x11, 52
or   x10, x11, x10
```

255	1
127	1
63	1
31	1
15	1
7	1
3	1
2	1

0.125	$\downarrow$
0.25	0
0.50	0
0.	1 (sottraggo 1)

11111111.001

1.1111111001$*2^7$

$(-1)^s * (1+m)^{esp-127}$

s = 0

m = 1111111001

esp-127 = 7 $\rightarrow$ esp = 134

134	0
67	1
33	1
16	0
8	0
4	0
2	0
1	1

0 10000110 11111110010000000000000

0100 0011 0111 1111 0010 0000 0000 0000

- x10 = 0000 0000 0000 0000 0000 0000 0000 0000 0100 0011 0111 1111 0010 0000 0000 0000
- 4096 $\rightarrow$ 1000000000000 $\rightarrow$ 1 0000 0000 0000
- **addi** x11, x0, 4096
  x11 = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0000 0000 0000
- **sll** x11, x11, 52
  x11 = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
- **or** x10, x11, x10
  x10 = 0000 0000 0000 0000 0000 0000 0000 0000 0100 0011 0111 1111 0010 0000 0000 0000

Esempio Intel: somme varie

```
long myFunc(long a, long b, long c) {
    long result = 0;
    result += a;
    result += b;
    result += c;
    return result;
}
```

→

```
myFunc:
    pushq %rbx
/* salva rbx, che usiamo per c */
    pushq %r12
/* salva r12, usato per result */
    movq %rdi, %r8   /* copia a in r8 */
    movq %rsi, %r9   /* copia b in r9 */
    movq %rdx, %rbx  /* copia c in rbx */
    movq $0, %r12    /* result = 0 */
    addq %r8, %r12   /* result += a */
    addq %r9, %r12   /* result += b */
    addq %rbx, %r12  /* result += c */
    movq %r12, %rax  /* ret val = result */
    popq %r12        /* restore old r12 */
    popq %rbx
    ret
```

Versione ottimizzata:

Versione 1

```
myFunc:
    movq $0, %rax
    addq %rsi, %rax
    addq %rdi, %rax
    addq %rdx, %rax
    ret
```

Versione 2

```
myFunc:
    addq %rsi, %rdi          /* v = a + b*/
    leaq (%rdi, %rdx), %rax  /* v += c */
    ret
```

Esempio Intel: ricorsione

```c
long sum(long count) {
  if (count > 0) {
    long p_sum = sum(count - 1);
    return p_sum + count;
  } else {
    return 0;
  }
}
```

→

```
# rdi (arg 1) is count
sum:
    cmpq $0, %rdi
    jle base_case     /* se count <= 0 --> */
                      /*   vai a base_case */
    pushq %rdi        /* salva copia di rdi */
    subq $1, %rdi     /* prepara argomento */
    call sum          /* chiama sum(count-1) */
    popq %rdi         /* ripristina valore */
                      /*   originale di rdi */
    addq %rdi, %rax   /* ret val = sum(count-1) + count */
    ret
base_case:
    mov $0, %rax
    ret
```

Esempio Intel: comparazione stringhe

```
int compare_string (const char *s1,
                    const char *s2) {
  while( (*s1 != '\0') &&
         (*s2 != '\0') &&
         (*s1 == *s2) ) {
    s1++; # increment the pointers to
    s2++; # the next char / byte
  }
  return (*s1==*s2);
}
```

→

```
compare_string:
L2:
   movzbl (%rdi), %eax
   testb %al, %al
   je L3
   movzbl (%rsi), %edx
   testb %dl, %dl
   je L3
   cmpb %al, %dl
   jne L3
   addq $1, %rdi
   addq $1, %rsi
   jmp L2
L3:
   cmpb (%rsi), %al
   sete %al          # Set byte if equal (ZF=1)
   movzbl %al, %eax # return e' intero
   ret
```

Esempio Intel: Fibonacci

```c
long fib(int n) {
    if ((n == 0) || (n == 1))
      return 1;
    return fib(n-1) + fib(n-2);
}
```

$\longrightarrow$

```asm
fib:
    cmpl   $1, %edi
    jbe    .L3
    pushq  %rbp
    pushq  %rbx
    movl   %edi, %ebx      /* copia edi in ebx */
    leal   -1(%rdi), %edi  /* preparo argomento */
    call   fib             /* prima chiamata */
    movq   %rax, %rbp      /* salvo risultato */
    leal   -2(%rbx), %edi  /* preparo argomento */
    call   fib             /* seconda chiamata */
    addq   %rbp, %rax      /* sommo risultati parziali */
    popq   %rbx
    popq   %rbp
    ret
.L3:
    movl   $1, %eax
    ret
```

Esempio Intel: compara memoria

```c
int mem_compare (int nb,
                 char *m1,
                 char *m2){
  while(nb-- > 0) {
    if(*m1++ != *m2++) {
      return 0;
    }
  }
  return 1;
}
```

→

```
main_compare:
.L2:
        movl    %edi, %eax
        subl    $1, %edi
        testl   %eax, %eax
        jle     .L6
        leaq    1(%rsi), %rax
        leaq    1(%rdx), %rcx
        movzbl  (%rdx), %edx
        cmpb    %dl, (%rsi)
        jne     .L5
        movq    %rcx, %rdx
        movq    %rax, %rsi
        jmp     .L2
.L6:
        movl    $1, %eax
        ret
.L5:
        movl    $0, %eax
        ret
```

Esempio ARM: Hailstone sequence

```c
int iter = 0;
int n = 5;
while (n != 1) {
  iter++;
  if (n % 2) {
     n = 3 * n + 1;
  } else {
     n = n / 2;
  }
}
```

$\longrightarrow$

```
        mov   r0, #5               // r0 e' numero corrente (5)
        mov   r1, #0               // r1 conta numero iterazioni
again   add   r1, r1, #1           // incremento numero iterations
        ands  r2, r0, #1           // Verifico se r0 e' dispari
        beq   even
        add   r0, r0, r0, lsl #1   // se dispari, r0 = r0 + (r0 << 1) + 1
        add   r0, r0, #1           //   e ripeto (garantito r0 > 1)
        b     again
even    mov   r0, r0, asr #1       // se pari, r0 = r0 >> 1
        subs  r7, r0, #1           //   e ripeto se r0 != 1
        bne   again
```

$$3 * n = n * 2 + n = (n << 1) + n \qquad\qquad n/2 = n >> 1$$

Esempio ARM: compara memoria

```c
int mem_compare (int nb,
                 char *m1,
                 char *m2){
  while(nb-- > 0) {
    if(*m1++ != *m2++) {
      return 0;
    }
  }
  return 1;
}
```

→

```
mem_compare:
                                //; r0 = nb, r1=m1, r2 = m2
_loop:
    subs r0, r0, #1     ; // Decrement nbytes and set
                          //   flags based on the result
    bmi  _finished      ; // If nb is now negative,
                          //   it was 0, so we're done

    ldrb r3, [r1], #1   ; // Load from the address in r1,
                          //   then add 1 to r1
    ldrb r4, [r2], #1   ; // ditto for r2
    cmp r3, r4          ; // If they match...
    beq _loop           ; //   then continue round the loop
    mov r0, #0          ; //   else give up and return zero
    bx lr
_finished:
    mov r0, #1          ; // Success!
    bx lr
```

Esempio ARM: Fibonacci

```c
long fib(int n) {
    if ((n == 0) || (n == 1))
      return 1;
    return fib(n-1) + fib(n-2);
}
```

$\longrightarrow$

```
fib:
        cmp     r0, #1
        bls     .L3
        push    {r4, r5, r6, lr}
        mov     r4, r0
        sub     r0, r0, #1
        bl      fib
        mov     r5, r0
        sub     r0, r4, #2
        bl      fib
        add     r0, r5, r0
        pop     {r4, r5, r6, lr}
        bx      lr
.L3:
        mov     r0, #1
        bx      lr
```

Esempio ARM: comparazione stringhe

```c
bool compare_string (const char *s1,
                     const char *s2) {
  while( (*s1 != '\0') &&
         (*s2 != '\0') &&
         (*s1 == *s2) ) {
    s1++; # increment the pointers to
    s2++; # the next char / byte
  }
  return (*s1==*s2);
}
```

→

```asm
compare_string:
        b       .L2
.L4:
        add     r0, r0, #1
        add     r1, r1, #1
.L2:
        ldrb    r3, [r0]
        cmp     r3, #0
        beq     .L3
        ldrb    r2, [r1]
        cmp     r2, #0
        beq     .L3
        cmp     r3, r2
        beq     .L4
.L3:
        ldrb    r0, [r1]
        cmp     r3, r0
        movne   r0, #0
        moveq   r0, #1
        bx      lr
```

Esercizio sulla pipeline

- Si consideri una CPU in cui le 5 fasi di esecuzione di un'istruzione impiegrano rispettivamente 100ps, 400ps, 600ps, 300ps e 100ps. Il massimo incremento di prestazioni che ci si può attendere usando una pipeline è:
  - di 2 volte
  - di 3.5 volte
  - di 2.5 volte
  - di 3 volte
  - nessuna delle altre risposte

- Tempo totale = 100ps + 400ps + 600ps + 300ps + 100ps= 1500ps

- Incremento = $\frac{\text{Tempo totale}}{\text{Fase più lenta}} = \frac{1500ps}{600ps} = 2.5$ volte

- La risposta corretta è la terza.

Esercizio sulla pipeline

- Si consideri una CPU che impiega 600ps per la fase di fetch, 600ps per la fase di decodifica, 500ps per eseguire operazioni con la ALU, 400ps per la fase di accesso alla memoria e 700ps per la fase di scrittura nel register file. Il massimo incremento di prestazioni che ci si può attendere usando una pipeline è:
  - di 4 volte
  - di 2.5 volte
  - di 2 volte
  - di 3 volte
  - nessuna delle altre risposte

- Tempo totale = 600ps + 600ps + 500ps + 400ps + 700ps= 2800ps
- Incremento = $\frac{\text{Tempo totale}}{\text{Fase più lenta}} = \frac{2800ps}{700ps} = 4$ volte
- La risposta corretta è la prima.

Esercizio sulla cache set associative

- Si consideri una cache associativa a 2 vie grande 16KB, con blocchi di 32 byte per blocco. In che blocco di cache è mappata la parola che sta all'indirizzo $0x100400_{16}$?
  - Nel primo blocco libero
  - Nessuna delle altre risposte
  - Nel blocco 0 o nel blocco 1
  - Nel blocco 64 o nel blocco 65
  - Nel blocco 16

- # bytes per blocco $= 32 \rightarrow \log_2 32 = 5$ bit (offset)
- # blocchi = $\frac{\text{Cache size}}{\text{\# byte per blocco}} = \frac{16*1024}{32} = \frac{16368}{32} = 512$
- # sets = $\frac{\#blocchi}{\#vie} = \frac{512}{2} = 256 \rightarrow \log_2 256 = 8$ bit (set)

Indirizzo =	$0x100400_{16}$			
=	0001 0000 0000 0100 0000 0000_2			
$\downarrow$		tag	set	offset
=		000100000000	00100000	00000

set = 00100000 = 2^5 = 32

primo blocco = (#set $*$ # vie) $= 32 * 2 = 64 \rightarrow$ Cache a 2 vie, quindi ogni set ha due blocchi

blocchi = 64 oppure 65

Esercizio sulla cache fully associative

- Si consideri una cache fully associative grande 16KB, con blocchi di 64 byte per blocco. In che blocco di cache è mappata la parola che sta all'indirizzo $0x100620_{16}$?
  - Nel blocco 32
  - Nel blocco 24
  - Nel blocco numero 0, o nel blocco numero 1 o nel blocco numero 2, o nel blocco numero 3
  - Nessuna delle altre risposte
  - Nel blocco numero 48 o nel blocco numero 49

- Nella cache fully associative il tag è tutto l'indirizzo del blocco, quindi devo cercare ovunque il dato, e non lo trovo quindi in un blocco preciso calcolabile a priori in base all'indirizzo.
- Quindi l'unica risposta corretta è la 4, ovvero nessuna delle altre risposte!

Esercizio sulla cache direct mapped

- Si consideri una cache direct mapped grande 4KB con blocchi di 16 bytes per blocco. In che blocco di cache è mappata la parola che sta in memoria all'indirizzo $0x1F164_{16}$?
  - Nel blocco numero 6
  - Nel blocco numero 64
  - Nel blocco numero 22
  - Nessuna delle altre risposte
  - Nel primo blocco libero

- # bytes per blocco $= 16 \to \log_2 16 = 4$ bit (offset)
- # blocchi = $\frac{\text{Cache size}}{\text{\# byte per blocco}} = \frac{4*1024}{16} = \frac{4096}{16} = 256 \to \log_2 256 = 8$ bit (block)

Indirizzo = $0x1F164_{16}$

= 0001 1111 0001 0110 0100 $_2$

$\downarrow$

tag	block	offset
00011111	00010110	0100

blocco = 00010110 = $2^4 + 2^2 + 2^1$ = 22

- L'indirizzo è quindi mappato sul blocco # 22, e la risposta corretta è quindi la terza.

Esercizio su CPI della CPU

- Si consideri una CPU dotata di due cache separate per dati ed istruzioni. Il CPI ideale della CPU è 4. La cache istruzioni ha una frequenza di miss dell' 1% e la cache dati ha una frequenza di miss del 4%. Supponiamo che una cache miss richieda 100 cicli di clock per essere servita e che il 20% delle istruzioni assembly accedano a dati in memoria. Il CPI reale (il numero di cicli necessari in media per eseguire un'istruzione tenendo conto degli stalli per accesso alla RAM) è?
  - 3.44
  - 7.44
  - 8
  - Nessuna delle altre risposte
  - 3.72

  CPI Reale=CPI ideale + Penalizzazione per istruzione + Penalizzazione per dati
  =CPI ideale + (Frequenza di miss per istruzioni * Penalità) +
  = (Frequenza di miss per dati * Penalità * Accesso a memoria)
  =4 + (0.01 * 100) + (0.04 * 100 * 0.2)
  =4 + 1 + 0.8
  =5.8
- Il CPI ideale è quindi 5.8, e quindi la risposta corretta è la 4, ovvero nessuna delle altre risposte.