

**SECONDA PROVA INTERMEDIA DEL MODULO DI
CALCOLATORI ELETTRONICI
CORSI DI LAUREA IN
INGEGNERIA ELETTRICA, ELETTRONICA ED INFORMATICA
INGEGNERIA BIOMEDICA
IMMATRICOLATI A.A. 2015/16 e precedenti
1 giugno 2017**

NOME:

COGNOME:

MATRICOLA:

ESERCIZIO 1 (12 punti)

Motivando ciascuna istruzione con opportuni commenti, scrivere una funzione Assembly MIPS scomponi che effettui la scomposizione in fattori di un numero intero non negativo fornito in ingresso. Se il numero passato vale 0 o 1, la funzione restituisce zero, altrimenti, restituisce la sequenza dei fattori (numeri primi) a partire da un'area di memoria `fattori` e i relativi esponenti a partire dall'area di memoria `esponenti`. Le due aree di memorie sono espresse in modo simbolico, mentre il valore da scomporre è passato attraverso il registro \$4. La funzione scrive nel registro \$5 il numero di fattori complessivi (escluso il valore 1).

Per esempio, se il valore da scomporre è 12, passato in \$4, la funzione porrà a 2 il valore di \$5 e le aree `fattori` ed `esponenti` saranno caratterizzate come segue:

Indirizzo	Contenuto	Indirizzo	Contenuto
<code>fattori</code>	2	<code>esponenti</code>	2
<code>fattori+4</code>	3	<code>esponenti+4</code>	1

In modo da significare che $12 = 2^2 \cdot 3^1$.

Nell'implementare la funzione, si usi la funzione `dividi(x,y)` che, ricevendo due interi non negativi `x` e `y` in \$4 e \$5, restituisce quoziente e resto in \$6 e \$7, rispettivamente.

La modalità di salvataggio/ripristino del contesto per `scomponi` e `dividi` è *callee save*.

Un possibile pseudocodice per la funzione `scomponi` è:

```
int scomponi(int x)
{
    int i,f,e;

    if(x<2) return 0;
    f=2;
    i=0;
    while ((x>0) && (f<=x))
    {
        e=0;
        quoziente, resto = dividi(x,f);
        while (resto!=0)
        {
            e=e+1;
            x=quoziente;
            quoziente, resto=dividi(x,f);
        }
        if (e!=0)
        {
            fattori[i]=f;
            esponenti[i]=e;
            i=i+1;
        }
        f=f+1;
    }
    return i;
}
```

Note:

- (1) Implementazione del solo ciclo while interno in modo corretto: 5 punti
- (2) Implementazione dei soli ciclo while + if interni in modo corretto: 8 punti
- (3) Implementazione del solo salvataggio e ripristino in modo corretto: 2 punti

ESERCIZIO 2 (7 punti)

Si consideri la seguente rappresentazione di numeri binari in virgola mobile: campo complessivo di 16 bit, con parte frazionaria di 10 bit rappresentata in segno e valore (1.M) ed esponente a 5 bit rappresentato in eccesso 15.

1. (2 punti) Esprimere il minimo ed il massimo valore rappresentabili in valore assoluto, escludendo lo zero.
2. (3 punti) Rappresentare il valore 45.25.
3. (2 punti) Se la mantissa venisse ridotta a 6 bit, a favore dell'esponente, sarebbe possibile rappresentare esattamente il numero dato? In caso contrario, cosa sarebbe rappresentato in luogo di 45.25?

ESERCIZIO 3 (6 punti)

Si progetti una ALU per ingressi a N bit che abbia il funzionamento descritto nella seguente tabella. F è la funzione logico-aritmetica realizzata dalla ALU. s1 e s0 sono due variabili di controllo da cui dipende la funzione realizzata dalla ALU.

s1	s0	F
0	0	A
0	1	A-B
1	0	B-A
1	1	B

Per la realizzazione della suddetta ALU si utilizzi un parallel adder e delle opportune reti logiche. Si disegni lo schema della ALU a livello di blocchi e porte logiche.

ESERCIZIO 4 (8 punti)

L'ampiezza della linea dati del bus sincrono di un calcolatore è pari a 32 bit. La frequenza del clock della CPU è di 2 GHz.

- 1) (5 punti) Ipotesizzando che il bus sincrono abbia la stessa frequenza di clock della CPU e la durata di una trasmissione sul bus sia pari a 4 cicli di clock, e che il tempo di ciclo della memoria sia pari a 20 cicli di clock, illustrare chiaramente il protocollo di lettura di una parola da memoria a CPU su bus sincrono utilizzando l'opportuno grafico, indicando il tempo complessivo di trasferimento di una parola di 32 bit.
- 2) (3 punti) Si supponga di disporre della seguente funzione Assembly MIPS per la lettura di caratteri dalla tastiera e la loro visualizzazione su monitor.

```
acquire-from-keyboard:  addi $29, $29, -4
                        sw $10, 0($29)
                        lw $10, keyboard($0) #acquisisce
                        sw $10, monitor($0) #mostra su monitor
                        lw $10, 0($29)
                        addi $29, $29, 4
                        jr $31
```

Indicare chiaramente a quale tecnica di gestione dell'I/O si sta facendo riferimento, qual è la tecnica di indirizzamento delle periferiche adottata ed illustrare in quale punto del ciclo di esecuzione delle istruzioni sia possibile utilizzare la funzione `acquire-from-keyboard`.

Se ciascun carattere è rappresentato con 8 bit, quanti caratteri alla volta sono trasferibili con questo metodo?

Soluzioni

ESERCIZIO 1

Dall'analisi dello pseudo-codice, è possibile decidere i registri che possono essere usati:

\$8 → i; \$9 → f; \$10 → e

\$12 → copia di \$4 per ripristinarlo prima dell'uscita dalla funzione

\$11 → condizioni di verità

\$14 → i*4

\$6 → quoziente, \$7 → resto (come da funzione dividi)

```
scomponi:    addi $29, $29, -28    #salvataggio del contesto
             sw $8, 0($29)
             sw $9, 4($29)
             sw $10, 8($29)
             sw $11, 12($29)
             sw $12, 16($29)
             sw $14, 20($29)
             sw $31, 24($29)
             move $12, $4
             move $8, $0          #i=0
             slti $11, $4, 2      #if (x<2)...
             bne $11, $0, exit     #...return 0
             addi $5, $0, 2       #f=2
             move $6, $4

while:       beq $6, $0, exit      #while (x>0)...
             slt $11, $6, $9
             bne $11, $0, exit     #...&& (x<f)...
             move $10, $0         #e=0
             move $4, $6          #passaggio parametri per dividi
             jal dividi

while_r:     bne $7, $0, exit_while_r #while(r==0)...
             addi $10, $10, 1      #aggiorno l'esponente
             move $4, $6          #x=d
             jal dividi           #d=dividi(x,f)
             j while_r

exit_while_r: beq $10, $0, updf     #if e!=0...
             muli $14, $8, 4       #calcolo i*4
             sw $5, fattori($14)   #fattori[i]=f
             sw $10, esponenti($14) #esponenti[i]=e
             addi $8, $8, 1        #i=i+1

updf:       addi $5, $5, 1         #f=f+1
             j while

exit:       move $5, $8            #numero di fattori
             move $4, $12         #ripristino x iniziale
             lw $8, 0($29)        #ripristino contesto
             lw $9, 4($29)
             lw $10, 8($29)
             lw $11, 12($29)
             lw $12, 16($29)
             lw $14, 20($29)
             lw $31, 24($29)
             addi $29, $29, 28
             jr $31              #ritorno al chiamante
```

ESERCIZIO 2

Soluzione alla domanda 1.

La rappresentazione data è nella forma:

$$1.M \cdot 2^e = \left(2^0 + \sum_{i=-1}^{-10} b_i \cdot 2^i \right) \cdot 2^e$$

Con $M = \{b_{-1}, \dots, b_{-10}\}$, i dieci bit di parte frazionaria ed e esponente.

Il minimo valore si ha in corrispondenza di una mantissa tutta di 0, con l'1 in corrispondenza del bit implicito. Il minimo esponente si ricava dall'eccesso k , e poiché l'eccesso è 15 il minimo esponente è pari a -15.

Il massimo valore si ha in corrispondenza di una mantissa tutta di 1, alla quale va aggiunto il bit implicito. Il massimo esponente si ricava a partire dal massimo valore ottenibile con cinque bit, pari a 31, al quale va sottratto l'eccesso 15, ottenendo così 16.

Quindi:

Minimo valore: $1.0 \dots 0 \cdot 2^{-15} = 2^{-15}$

Massimo valore: $1.1 \dots 1 \cdot 2^{16} = \left(1 + \sum_{i=-1}^{-10} 2^i \right) \cdot 2^{16} = (2 - 2^{-10}) \cdot 2^{16}$

Soluzione alla domanda 2.

Convertendo parte intera e parte frazionaria del valore dato 45.25 si ha:

$$45.25_{10} = 101101.01_2 = 1.0110101 \cdot 2^5$$

La conversione di 5 in eccesso 15 è semplice: $5 + 15 = 20 \rightarrow 10100$

La rappresentazione del numero dato in virgola mobile è dunque, ipotizzando il valore 0 per un segno positivo: 0 10100 0110101000 dove il primo è il bit di segno, seguito da cinque bit di esponente ed infine i dieci bit di mantissa. Il bit implicito non è mai rappresentato.

Soluzione alla domanda 3.

Dal calcolo di sopra si ottiene che il valore non sarebbe più rappresentabile esattamente perché si perderebbe l'uno meno significativo in mantissa. In altre parole, si avrebbe un troncamento che, se in difetto, ovvero semplicemente rilevando il bit meno significativo, porterebbe al valore $1.011010 \cdot 2^5 = 45.0_{10}$ con una perdita di precisione dello 0.55%.

ESERCIZIO 3

Per il circuito e le funzioni circuitali si vedano le dispense del corso. In funzione delle operazioni richieste si hanno i valori seguenti per gli operandi e il riporto in ingresso:

s1	s0	F	A	B	C _{in}
0	0	A	A	0	0
0	1	A-B	A	B'	1
1	0	B-A	A'	B	1
1	1	B	0	B	0

s1s0 \ A _i	00	01	11	10
0				1
1	1	1		

$$A_i^{\text{new}} = \bar{s}_1 A_i + s_1 \bar{s}_0 \bar{A}_i$$

s1s0 \ B _i	00	01	11	10
0		1		
1			1	1

$$B_i^{\text{new}} = s_1 B_i + \bar{s}_1 s_0 \bar{B}_i$$

$$c_{\text{in}} = s_1 \oplus s_0$$

L'esercizio per essere completato richiede il disegno su porte logiche dei tre circuiti di cui sopra, la cui realizzazione si lascia allo studente.

ESERCIZIO 4

Soluzione alla domanda 1.

La durata di un ciclo di clock è pari a $1/2 = 0.5 \text{ ns}$

La lettura su un bus sincrono avviene secondo il protocollo seguente:

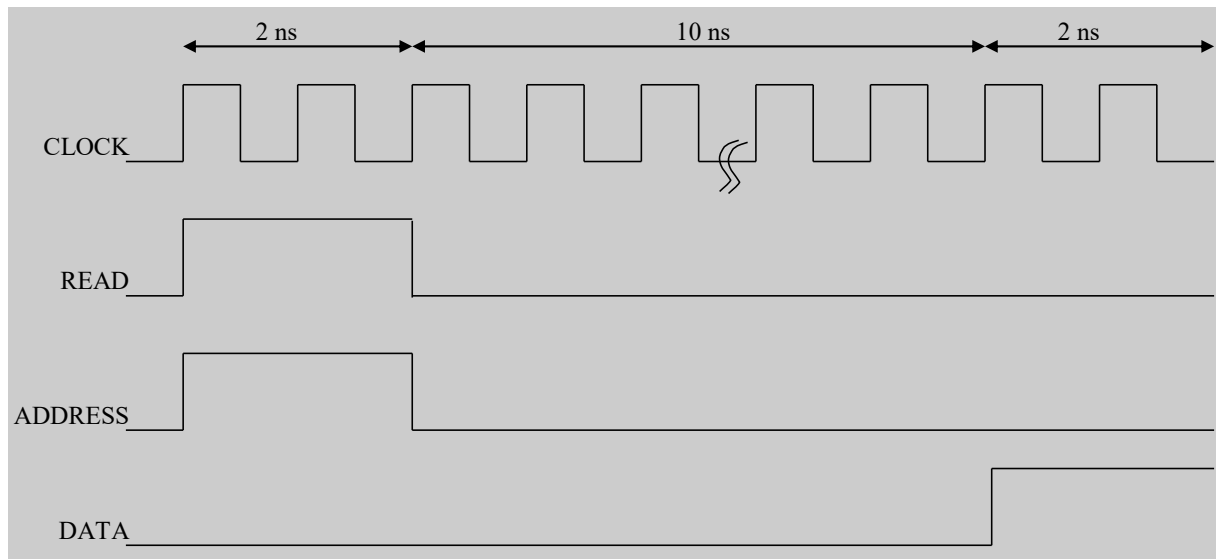
- Segnale di READ sulla linea di controllo e contemporaneamente l'indirizzo della locazione in cui risiede il dato sulla linea indirizzi:

4 cicli di clock = 2 ns

- Lettura della parola dalla memoria: $0.5 \cdot 20 = 10 \text{ ns}$
- Trasferimento della parola dalla memoria:

trasferimento della parola = 2 ns

Tempo totale per leggere una parola dalla memoria = $(4 + 10) \text{ ns} = 14 \text{ ns}$



Nota: la durata delle trasmissioni è 4 colpi di clock, nel grafico esemplificate con 2.

Soluzione domanda 2.

La presenza di una funzione, o subroutine, per la gestione dell'I/O, non può che riferirsi ad un I/O pilotato da interruzioni.

Un'interruzione è un segnale presente lungo la linea "interrupt" del bus di controllo. Quando essa è attiva, viene inviato al PC l'indirizzo della subroutine di gestione dell'interruzione che permette di capire quale sia la periferica che richiede un trasferimento dati (la tastiera).

Tale subroutine, come si vede dal codice MIPS, si compone di una parte di salvataggio del contesto, attraverso lo stack (il cui puntatore è in $\$29$), seguito dall'acquisizione di una parola dalla tastiera e dalla stampa su monitor. In questo caso è evidente che l'I/O sia mappato in memoria, visto che non sono definite istruzioni specifiche per le azioni di trasferimento dati, attraverso l'uso delle istruzioni lw e sw già note.

Il punto del ciclo di esecuzione delle istruzioni in cui viene valutata la linea "interrupt" per l'eventuale esecuzione di `acquire-from-keyboard` è quello immediatamente successivo alla fase "store-results", quindi precedente il prelievo dell'istruzione successiva a quella in corso, in modo tale che il PC venga salvato in $\$31$ prima di essere aggiornato con l'indirizzo della subroutine.

Infine, considerando che una parola MIPS è composta da 4 byte, ciascuno di essi accessibili, il sistema andrà a prelevare dalla tastiera fino a 4 caratteri alla volta, dal momento che essi sono composti appunto da 1 byte.