

**PROVA SCRITTA DEL CORSO DI**  
**CALCOLATORI ELETTRONICI**  
**NUOVO ORDINAMENTO DIDATTICO**  
 27 Gennaio 2009

NOME:

COGNOME:

MATRICOLA:

**ESERCIZIO 1 (8 punti)**

- (4 punti) Progettare una rete logica che calcoli il complemento a 2 di un operando a tre bit.
- (2 punti) Indicare a quale tipo di reti logiche appartiene la rete progettata e spiegarne il motivo.
- (2 punti) Proporre uno schema alternativo per la realizzazione della rete logica, utilizzando un parallel adder e porte not.

**Soluzione.**

- Tabella di verità della rete in oggetto :

Ingressi			Uscite		
A	B	C	A'	B'	C'
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	0	1	0
1	1	1	0	0	1

AB \ C	00	01	11	10
0		1		1
1	1	1		

$$A' = \overline{A}B + \overline{A}C + A\overline{B} \cdot \overline{C}$$

AB \ C	00	01	11	10
0		1	1	
1	1			1

$$B' = B\overline{C} + \overline{B}C$$

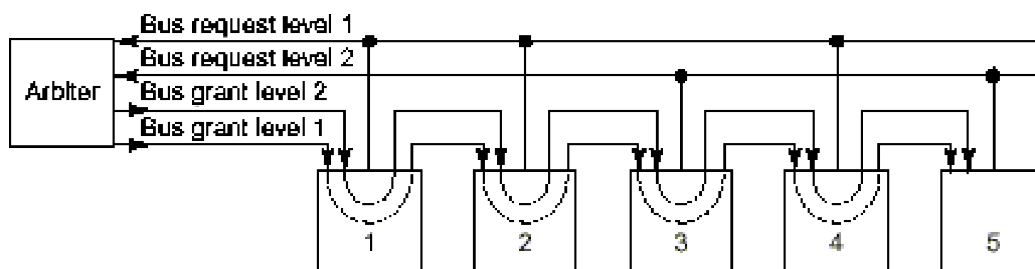
AB \ C	00	01	11	10
0				
1	1	1	1	1

$$C' = C$$

- Si tratta ovviamente di una rete logica combinatoria, in quanto le uscite dipendono soltanto dai valori degli ingressi ad un dato istante.
- E' sufficiente negare A,B,C (complemento a 1 del numero) prima di inviare i bit al parallel adder e impostare il riporto in ingresso al parallel adder a 1 (per realizzare il complemento a 2).

## ESERCIZIO 2 (8 punti)

Si consideri lo schema di arbitraggio del bus rappresentato in figura. Le linee di "bus request" e "bus grant" a 2 livelli vengono impiegate per gestire una diversa priorità delle periferiche.



1. (4 punti) A quali schemi, fra quelli studiati nel corso, si ispira quello mostrato in figura? Spiegare come può essere gestita in questo caso la gerarchia di priorità fra le periferiche.
2. (4 punti) Illustrare la sequenza di segnali di controllo che vengono scambiati fra periferica e arbitro del bus prima di iniziare un trasferimento di dati attraverso il bus.

### Soluzione

1. Lo schema in figura è un ibrido fra la tecnica di arbitraggio a richieste indipendenti e quella con collegamenti in catena "daisy chaining". Rispetto ai livelli 1 e 2 le periferiche sono divise in due gruppi indipendenti. All'interno di ciascun gruppo le periferiche sono collegate a catena. Pertanto vi è una gerarchia di priorità gestita direttamente dall'arbitro del bus che, in caso di richieste contemporanee da parte dei due livelli, può essere programmato per servire l'una o l'altra richiesta. All'interno di ciascun gruppo vi è un'ulteriore gerarchia di priorità "cablata", data dalla successione delle periferiche nella catena: ottiene il bus la prima periferica della catena, fra quelle che hanno fatto richiesta, che riceve il grant.
2. Una periferica che debba trasmettere dati sul bus, ne fa richiesta all'arbitro attraverso la linea di bus request. L'arbitro del bus risponde a questa richiesta inviando un segnale di bus grant sulla linea appropriata. Il bus grant attraversa serialmente le periferiche (nota: il grant relativo a un certo livello di priorità viene fatto passare da periferiche con livello di priorità diverso) e viene bloccato dalla prima periferica che aveva fatto richiesta del bus. A questo punto la periferica invia il segnale di bus busy, per indicare che il bus è impegnato e inizia il trasferimento dei dati.

### ESERCIZIO 3 (7 punti)

Sia dato il seguente codice Assembly.

1. (4 punti) Indicare eventuali errori di sintassi/semantica presenti nel codice.
2. (3 punti) Analizzare il codice spiegando chiaramente qual è la funzione svolta dallo stesso, quali sono i parametri in ingresso e quali in uscita.

```
funzione: subi $29, $29, -4
           sw $8, $29(0)
           addi $6, $0, 1
           move $8, $0
a:         beq $8, $5, b
           muli $6, $6, $4
           add $8, $8, 1
           jr a
b:         sw $8, $29(0)
           addi $29, $29, 4
           j $31
```

#### Soluzione.

1. Nel codice sono presenti evidenti errori di sintassi, laddove ad esempio nell'istruzione `sw` vengono permutate la posizione del secondo e terzo operando, e di semantica, laddove un'istruzione con indirizzamento immediato (`addi`) viene usata con indirizzamento a registro (`add`), e viceversa. Un altro errore consiste nell'uso errato dell'istruzione `j`, che prevede come operando un indirizzo di memoria, e dell'istruzione `jr`, che invece richiede un registro. Un ultimo errore è presente nella sezione di ripristino del contesto: viene usata l'istruzione di memorizzazione `sw` invece di quella di prelievo `lw`.

Funzione data con errori (in neretto)	Funzione corretta
funzione: <b>subi</b> \$29, \$29, -4 sw \$8, <b>\$29(0)</b> addi \$6, \$0, 1 move \$8, \$0 a: beq \$8, \$5, b <b>muli</b> \$6, \$6, \$4 <b>add</b> \$8, \$8, 1 <b>jr</b> a b: <b>sw</b> \$8, <b>\$29(0)</b> addi \$29, \$29, 4 <b>j</b> \$31	funzione: addi \$29, \$29, -4 sw \$8, 0(\$29) addi \$6, \$0, 1 move \$8, \$0 a: beq \$8, \$5, b mul \$6, \$6, \$4 addi \$8, \$8, 1 j a b: lw \$8, 0(\$29) addi \$29, \$29, 4 jr \$31

2. Una volta che la funzione è stata emendata dagli errori, si può notare che essa, dopo aver inizializzato \$6 ad 1, lo aggiorna moltiplicandolo per il contenuto di \$4. Ciò viene fatto finché il valore presente in \$8 è minore di quello presente in \$5. La presente funzione realizza quindi l'elevamento a potenza, con base in \$4 ed esponente in \$5. Il risultato viene memorizzato in \$6.

#### ESERCIZIO 4 (10 punti)

Si consideri un calcolatore che dispone di una memoria principale di 256 Mbyte e di una memoria cache di 512 Kbyte. E' possibile accedere al singolo byte e la memoria è suddivisa in blocchi da 16 byte.

- (4 punti) Spiegare, precisando il significato e la funzione dei diversi campi, come vengono interpretati gli indirizzi logici per recuperare l'informazione contenuta nella cache, nel caso venga usata la modalità di indirizzamento:
  - diretto;
  - associativo su insiemi, in cui ciascun insieme contenga due blocchi.
- (6 punti) Si consideri la cache di cui alla domanda precedente. Ipotizzare che il processore acceda in sequenza ai byte dall'indirizzo 0000000 a 00007FF e da 0180000 a 01807FF in questo ordine, e ripeta questa sequenza di accesso per 5 volte consecutive. Si ipotizzi inoltre che la cache sia inizialmente vuota. Calcolare il numero di miss sia nel caso di modalità ad indirizzamento diretto sia nel caso di modalità ad indirizzamento associativo su insiemi spiegata nel punto precedente. Calcolare quindi l'hit ratio per i due casi.

#### Soluzione:

- Per indirizzare 256 Mbyte occorre un indirizzo di almeno 28 bit. Per indirizzare il singolo byte all'interno di un blocco occorrono 4 bit ( $16 = 2^4$ ), che coincidono con i 4 bit meno significativi dell'indirizzo di memoria primaria. I restanti 24 bit costituiscono l'indirizzo del "block frame". Per indirizzare la cache, il "block frame" viene interpretato diversamente a seconda che l'indirizzamento sia di tipo "diretto" o "associativo su insiemi".

- Indirizzamento diretto. In questo caso devo poter indirizzare ciascuno dei 32K blocchi contenuti nella cache ( $512\text{Kbyte}/(16\text{byte}/\text{blocco})$ ). Occorrono 15 bit che coincidono con i 15 bit meno significativi del "block frame". Pertanto i 28 bit di indirizzo della memoria primaria vengono interpretati come:

tag	cache index	Offset
9 bit	15 bit	4 bit

- Indirizzamento "associativo su insiemi". In questo caso devo poter indirizzare ciascuno dei 16 insiemi in cui sono suddivisi i blocchi contenuti nella cache ( $\frac{512\text{Kbyte}}{\frac{2\text{blocchi}}{\text{insieme}} \cdot \frac{16\text{byte}}{\text{blocco}}} = 16\text{K insiemi}$ ).

Occorrono 14 bit che coincidono con i 14 bit meno significativi del "block frame"

Pertanto i 28 bit di indirizzo della memoria primaria vengono interpretati come:

tag	cache index	offset
10 bit	14 bit	4 bit

- La memoria è divisa in blocchi di 16 byte ciascuno in modo che la richiesta di un dato non presente in cache causa il trasferimento del blocco a cui appartiene il dato richiesto dalla memoria principale alla cache. Nel caso proposto i dati richiesti sono così suddivisi (N.B. per semplicità i valori di index sono riportati in formato decimale, mentre gli indirizzi in esadecimale):

indirizzi 0000000 ÷ 000000F ⇒ index 0; indirizzi 0000010 ÷ 000001F ⇒ index 1; ... ; 00007F0 ÷ 00007FF ⇒ index 127; 0180000 a 018000F ⇒ index 0; ... ; 01807F0 a 01807FF ⇒ index 127. (N.B. in questa sequenza l'index è identico nel caso di indirizzamento diretto e associativo su insiemi).

La prima volta che viene richiesta la parola 0000000 avremo un "cache miss" che provoca il caricamento del blocco 0 nella linea di cache di indirizzo 0. Le successive richieste dei dati di indirizzo 0000001 ÷ 000000F vengono quindi soddisfatte dalla cache ("cache hit"). Analogamente avviene per tutti i blocchi fino al blocco 127, che vengono allocati in linee consecutive della cache fino alla 127. Quando viene richiesta la parola 0180000 (index 0), nel caso di indirizzamento diretto questa sovrascrive il blocco 0000000 ÷ 000000F e così via per tutte le richieste consecutive. Nel caso di indirizzamento associativo su insiemi a due vie uno stesso index corrisponde a un insieme di due blocchi. Pertanto le parole da 0180000 a 01807FF non

sovrascrivono quelle precedentemente inserite, ma vengono memorizzate nel secondo blocco disponibile in ciascun insieme memorizzato.

Nei cicli successivi al primo il processore richiede nuovamente tutti i dati, a partire da 0. Nel caso di indirizzamento diretto avremo un miss per il caricamento della prima parola di ciascun blocco e 15 hit per le parole dello stesso blocco, per tutti i 256 blocchi. Nel caso di indirizzamento associativo su insiemi si hanno solo hit perché tutti blocchi sono presenti in cache.

In sintesi:

- nel caso di indirizzamento diretto avremo per ciascun ciclo 256 miss, quindi 1280 miss in totale ( $256 \cdot 5$ );
- nel caso di indirizzamento associativo su insiemi a due vie avremo soltanto 256 miss in totale, tutti relativi al primo ciclo.

Poiché il numero totale di accessi è sempre  $256 \cdot 5 \cdot 16$ , si ricava:

$H(\text{diretto}) = 0,9375$ ;

$H(\text{associativo su insiemi}) = 0,9875$ .