



- R: (3 pt)** La tabella di verità permette quattro coperture di  $4 \times 2$  simboli 1, confermando il risultato dell'esercizio precedente:

<---                      --->

- R: (3 pt)** È sufficiente scegliere un multiplexer a 16 ingressi e 4 controlli collegati rispettivamente ad ABCD, e poi connettere a una sorgente in tensione (per esempio 5V) tutti gli ingressi fuorchè quello associato al controllo 0111, che invece sarà connesso alla massa 0V.

- R: (3 pt)**



- R: (3 pt)** Per esempio:

4: 0110                      9: 1011

- R: (3 pt)** Nel caso peggiore ogni bit può essere in ritardo di 2 ns rispetto a quello trasmesso sulla linea adiacente. Il bus deve quindi garantire un clock non inferiore a  $T = 2 \cdot 10 = 20$  ns, risultando in una frequenza  $f = 1/T = 1/20$  GHz, equivalente a 50 MHz.

- R: (3 pt)** È il canale di comunicazione logico tra il microcontrollore e il mondo esterno. A ogni porta corrisponde un set dei registri di memoria che ne rappresentano lo stato e le relative informazioni di input/output. A ogni pin (piedino fisico) del microcontrollore corrisponde un bit a una determinata posizione, per ogni registro relativo alla porta.

12. È possibile realizzare un'architettura di tipo *load/store* il cui set di istruzioni macchina contenga operazioni aritmetiche prive di argomenti?

**R: (3 pt)** Sì. È sufficiente che tutte le operazioni aritmetiche siano eseguite su registri predefiniti.

13. Sapendo che in una memoria paginata (dimensione locazione = 1 byte) ogni riga della *page table* è lunga 23 bit e il campo *offset* dell'indirizzo è di 8 bit, si dica quante pagine riescono a trovare spazio in memoria principale rispettivamente nei casi in cui il riempimento è ottimale (ovvero non c'è alcuna locazione libera) e pessimo (ovvero c'è il numero massimo di locazioni libere). Nel secondo caso il risultato può essere dato anche in forma approssimata.

**R:** Dai dati si evince che la *page table* contiene indirizzi di 22 bit: di qui la dimensione della memoria principale uguale a  $2^{22} = 4$  MB. Parallelamente la dimensione della pagina è data dal campo *offset* dell'indirizzo:  $2^8 = 256$  byte. Se ne conclude che in condizioni di riempimento ottimale la memoria principale contiene  $2^{22}/2^8 = 2^{14}$  pagine. Viceversa, quando il riempimento è il peggiore possibile allora ogni pagina è preceduta da 255 byte non occupati. In pratica, dunque, è come se ogni pagina occupasse  $255 + 256 = 2^8 - 1 + 2^8 = 2^9 - 1$  locazioni. Di qui il calcolo del numero di pagine, uguale a

$$\frac{2^{22}}{2^9 - 1} = 2^{13} \frac{2^9}{2^9 - 1} = 2^{13} \frac{512}{511} = 8192 \cdot \left(1 + \frac{1}{511}\right) = 2^{13} + 16.031,$$

cioè  $2^{13} + 2^4$ .

14. [INF] Scrivere un programma in assembly per ARM il quale, caricati nei registri **r1** e **r2** rispettivamente due numeri  $n_1$  e  $n_2$  dalla memoria, dapprima verifica che ciascun numero occupi solo la metà meno significativa del registro che lo contiene. Se è così allora esegue l'algoritmo di moltiplicazione in colonna di due numeri positivi, infine depositando il risultato  $n_1 \cdot n_2$  nel registro **r3**.

**R: (9 pt)** Per  $0 \leq i \leq 15$  il contenuto di **r1** traslato di  $i$  bit a sinistra è accumulato in **r3** se l' $i$ -esimo bit di **r2** è uguale a uno.

```
.data
n1:  .word 8
n2:  .word 6
.text
main:
    ldr r1, =n1          ; read n1
    ldr r1, [r1]
    ldr r2, =n2          ; read n2
    ldr r2, [r2]
    tst r1, #0xFF00      ; test magnitude of n1
    bne end
    tst r2, #0xFF00      ; test magnitude of n2
    bne end
    mov r3, #0           ; reset r3
    mov r4, #1           ; bitwise mask
    mov r5, #0           ; iterations counter
loop:
    tst r2, r4, lsl r5    ; r2 AND (r4 << r5)
    beq update           ; if i-th bit is null skip sum..
    add r3, r3, r1        ; ..else add r1
update: mov r1, r1, lsl #1 ; shifts r1 leftward by one bit
    add r5, r5, #1       ; increment counter
    cmp r5, #16          ; if counter == 16..
    beq end              ; ..exit loop..
    b loop               ; ..else repeat
end:
    swi 0x11             ; exit program
    .end
```