

Corso di Fondamenti di Informatica – 2° modulo  
per Ingegneria Gestionale  
Università degli Studi di Udine - A.A. 2010-11



Docente Ing. Sandro Di Giusto

**Esercitazione di laboratorio n.4 (Martedì 14 gennaio 2011)**

**Obiettivi:**

- Prendere familiarità con il simulatore di CPU RISC SimCPU ed i tool a corredo
- Scrivere, eseguire e testare programmi di media complessità (con uso dello stack) in assembler

**Testo dell'esercitazione:**

1. Si testi al simulatore SimCPU il programma `Funct_A` che si trova nelle pagine seguenti:
  1. si testino tutti (o quasi) i comandi che il simulatore mette a disposizione e verificandone con mano le funzionalità e le caratteristiche (può risultare particolarmente utile o vantaggioso in questo caso l'uso dei *brakpoints*, comando `s b indirizzo` e `e s b indirizzo d`)
  2. si provi a variare un po il codice (ad esempio modificando i valori di `A` e `B`) e si osservi il nuovo comportamento al simulatore, come al punto precedente, cercando di interpretare il codice e capire quindi che funzione logico/matematica viene realizzata dal programma (e come)
2. Si testi al simulatore SimCPU il programma `Funct_B` che si trova nelle pagine seguenti:
  1. si osservi prima di tutto le (minime) differenze che ci sono rispetto al programma `Funct_A` e si provi ad intuire quali differenze si potranno notare poi durante l'esecuzione del programma vero e proprio
  2. Si lanci al simulatore il programma e si verifichi le eventuali intuizioni al punto precedente o si osservi quantomeno il comportamento del programma e si cerchi di dargli una spiegazione logica (suggerimento: si analizzi il contenuto di `SP` e della memoria durante i primi cicli di esecuzione del programma)
3. Si testi al simulatore SimCPU il programma `Funct_C` che si trova nelle pagine seguenti:
  1. si osservi prima di tutto le (minime) differenze che ci sono rispetto al programma `Funct_A` e `Funct_B` e si provi ad intuire quali differenze si potranno notare poi durante l'esecuzione del programma vero e proprio
  2. Si lanci al simulatore il programma e si verifichi le eventuali intuizioni al punto precedente o si osservi quantomeno il comportamento del programma e si cerchi di dargli una spiegazione logica (suggerimento: si analizzi anche in questo caso il contenuto di `SP` e della memoria durante i primi cicli di esecuzione del programma)
4. Si cerchi di scrivere il semplice codice `C` equivalente alle due varianti `Funct_A` e `Funct_C` (come se si trattasse di implementare lo stesso algoritmo con delle funzioni), prestando attenzione a mantenere le modifiche fondamentali tra le due implementazioni
5. Si provi a scrivere la variante del programma `Funct_C` che prende i valori dei 2 dati `A` e `B` non già da memoria (quindi scelti e stabiliti nel codice assembler stesso) ma da input e si visualizzi il risultato ad output (tralasciando per semplicità i casi che portano a problemi di overflow)

## Func\_t\_A

```
A:      byte 4
B:      byte 3
STACK:  word 0F000

START:  LDWA R0 STACK
        SPWR R0
        LDBA R0 A
        LDBA R1 B
        CALL FUNCT
        HLT

FUNCT:  DEC R1
        JMPNZ ELSE
        MV R0 R15
        RET

ELSE:   CALL FUNCT
        ADD R0 R15
        RET
```

## Func\_t\_B

```
A:      byte 4
B:      byte 3
STACK:  word 0F000

START:  LDBA R0 A
        LDBA R1 B
        CALL FUNCT
        HLT

FUNCT:  DEC R1
        JMPNZ ELSE
        MV R0 R15
        RET

ELSE:   CALL FUNCT
        ADD R0 R15
        RET
```

## Func\_t\_C

```
A:      byte 4
B:      byte 3
STACK:  word 0F000

START:  LDWA R0 STACK
        SPWR R0
        LDBA R0 A
        LDBA R1 B
        CALL FUNCT
        HLT

FUNCT:  PUSH R1
        DEC R1
        JMPNZ ELSE
        MV R0 R15
        POP R1
        RET

ELSE:   CALL FUNCT
        ADD R0 R15
        POP R1
        RET
```