# Architettura dei Calcolatori a.a. 2012/13
# Prova di Laboratorio - assembly MIPS: accesso_array
# 17 Luglio 2013

## Introduzione

Lo scopo di questa prova di laboratorio è lo sviluppo di un semplice programma nel linguaggio assembly del processore MIPS. Non è richiesta una particolare base di conoscenze algoritmiche, ma semplicemente un minimo di dimestichezza con la programmazione assembly.

## Istruzioni

Cominciate facendo il login sulla macchina del laboratorio che vi è stata assegnata. Per il login occorre usare matricola e password dello *student portal*. Sul desktop troverete una cartella contenente i simulatori QtSpim e Mars. Lanciate ed utilizzate quello che preferite. Tutto il vostro codice (sia esso costituito da un singolo file, o da file multipli) andrà salvato nella cartella "`mips`" da creare sul drive `H:` .

Create un file `student-info.txt` con incluso il vostro nome e cognome e numero di matricola nella cartella "`mips`". Per maggior sicurezza, includete anche nome, cognome e matricola come commento, in testa ad ogni file sorgente. Alla fine della prova, i file saranno prelevati automaticamente dalla directory . Tutto quello che lascerete nella cartella `mips` sarà utilizzato per la valutazione. Salvare i vostri file altrove, o non indicare nome e cognome, porterà inevitabilmente all'annullamento della vostra prova. *Tutti i file all'esterno della cartella verranno cancellati automaticamente!!!*

## Le specifiche

Dovete scrivere un programma assembly che allochi un array `arr` di 100 interi in memoria globale. Una funzione `riempi_array` riceve in puntatore all'array come parametro e riempie ogni singola componente `arr[i]` con il valore `i*2+3`. La funzione non ritorna alcun valore al `main`. Una seconda funzione `consulta_array` riceve come parametri l'indirizzo dell'array ed il valore della componente `i`, e ritorna in $v0 e $v1 un codice di errore (0 → tutto ok, 1 → `i` out of bounds) e il valore della componente `arr[i]` (nel caso in cui `i` sia valido, ovviamente).

Il `main` inizializzi l'array con `riempi_array` e poi in loop ne permetta la consultazione richiedendo il numero della componente all'utente. Un input pari a -100 termina il programma. Valori out-of-bounds positivi e negativi vanno opportunamente segnalati. Per maggiori dettagli si veda l'output di esempio mostrato sotto. La chiamata delle due funzioni dovrà rispettare le convenzioni per il salvataggio dei registri. Utilizzate le system call SPIM/MARS per l'I/O e la terminazione del main. ***È fondamentale utilizzare funzioni con scambio dei parametri.***

Per i più bravi: effettuare il test per vedere se l'indice fornito è out-of-bounds (negativo o maggiore di 100, cioè) con un *unico* confronto (cioé con una *unica* coppia di istruzioni *set/branch*).

## Suggerimenti

Potete leggere i numeri in input da tastiera utilizzando la system call 5. Per la stampa di stringhe e di interi potete usare le syscall 4 e 1, rispettivamente. Il seguente è un output di esempio:

```
Riempio l'array.
Fatto.

Che componente dell'array vuoi leggere (-100-> exit): 12
La componente ha il valore 27

Che componente dell'array vuoi leggere (-100-> exit): -2
Indice out-of-bounds. Riprova.

Che componente dell'array vuoi leggere (-100-> exit): 120
Indice out-of-bounds. Riprova.

Che componente dell'array vuoi leggere (-100-> exit): -100

-- program is finished running --
Reset: reset completed.
```

## Valutazione

Scrivere un programma funzionante, che faccia uso di due funzioni come richiesto nelle specifiche e che segua le convenzioni di salvataggio dei registri è strettamente necessario per essere ammessi a sostenere l'orale. In quella sede, si entrerà nel dettaglio della struttura del codice, dando una valutazione migliore a soluzioni "pulite" e ben commentate.