Lab 8:

Supporto hardware alle procedure

Esercizio 1 – `strupr` (str uppercase)

Scrivere una funzione RISC-V `strupr` che converta in maiuscolo tutti i caratteri di una stringa ASCII presente in memoria.

Si supponga che la stringa sia composta solo da lettere (a-z).

Esercizio 2 – `digit` ()

Scrivere una funzione RISC-V `digit` che verifichi se un byte passato come parametro nel registro `a0` rappresenta un carattere cifra (0-9) nella codifica ASCII. *Verificare* vuol dire: restituire 1 se la condizione è vera, 0 altrimenti.

Esercizio 3 – `isnumber()`

Scrivere una funzione RISC-V `isnumber` che controlli se una stringa ricevuta come parametro è la rappresentazione di un numero intero positivo in ASCII. *Controllare* vuol dire: restituire 1 se la condizione è vera, 0 altrimenti.

`isnumber` deve utilizzare la funzione `digit` realizzata nell'esercizio precedente.

```
int isnumber(char *s) {
  for (int i = 0; s[i] != '\0'; i++)
    if (!digit(s[i]))
      return 0;
  return 1;
}
```

Esercizio 4 – `atoi()`

- Scrivere una funzione RISC-V `atoi` che converta una stringa ASCII con un numero intero positivo in una variabile numerica (intero in complemento a 2). Il valore ottenuto deve essere restituito al chiamante nel registro `a0`. Realizzare `atoi` usando l'algoritmo ricorsivo riportato nella prossima slide.
- Realizzare poi il «main» (riportato nella prossima slide) che utilizza `isnumber` e `atoi` per stampare a schermo il quadrato di un numero presente in una stringa ASCII (se la stringa contiene un numero valido).

Esercizio 4 – `atoi()`

```c
unsigned long atoi(char *str, unsigned long n) {
  if (n == 1)
    return str[0] - '0';
  return (10*atoi(str, n-1) + str[n-1]-'0');
}
```

```c
int main(void) {
  char str1[] = "11";

  if (isnumber(str1)) {
    unsigned long x = atoi(str1, strlen(str1));
    printf("%d\n", x*x);
  }
  return 0;
}
```

Esercizio 5 – `is_sorted`

Scrivere una funzione RISC-V che verifichi se un array di N numeri in memoria (word contigue) contiene una sequenza ordinata di numeri interi (crescente). *Verificare* vuol dire: restituire 1 se la condizione è vera, 0 altrimenti.

Esercizio 6 – `minarray`

Scrivere una funzione `minarray(v, s)` che restituisca l'indice del valore minimo presente nell'array **v.**

Nota: L'indirizzo di **v** deve essere passato come parametro a `minarray` insieme a **s** (size), che rappresenta il numero di word in **v.**

Esempi:

```
minarray([0,1,2,3,4], 5) = 0
minarray([1,1,1,1,1], 5) = 0
minarray([5,4,3,2,1], 5) = 4
```


In questi casi, restituire
l'indice del primo tra i minimi

Esercizio 7 – `selection_sort`

Usando `minarray(v, s)` e la procedura `swap(v, x, y)` vista nel Lab 7 (che scambia i valori di `v[x]` e `v[y]`), scrivere una funzione ricorsiva per ordinare un array di numeri interi chiamata `selection_sort`:

```
void selection_sort(int v[], int s) {
  if (s == 0)
    return;
  swap(v, 0, min_array(v, s));
  selection_sort(v+1, s-1);
}
```

Realizzare anche il main che effettui la chiamata di `selection_sort` e poi usi `is_sorted` per confermare che l'array ottenuto sia correttamente ordinato.

Esercizio 8 – RISC-V & GCC

- Sul Linux/Ubuntu, è possibile installare il toolchain del compilatore GCC per RISC-V tramite il seguente pacchetto: **https://packages.ubuntu.com/search?keywords=gcc-riscv64-linux-gnu**

- Per compilare un file .c in assembly RISC-V, serve eseguire il commando:

```
riscv64-linux-gnu-gcc -O1 -o- -S FILE.c
```

- Scaricare i file **selection.c** e **selection.asm** da Moodle. Comparare la vostra implementazione di `swap`, `min_array` e `selection_sort` con quella prodotta dal compilatore GCC.
- Guardare la funzione main generata da GCC. Il codice assembly è come quello che vi sareste aspettati? Cosa fa il compilatore di diverso?