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).

Codifica ASCII

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

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()`

```
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');
}
```

```
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

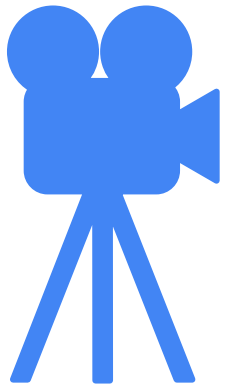
Esercizio 7 – `selection_sort`

Scrivere una funzione per ordinare un array di numeri interi chiamata `selection_sort`

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.

```
void selectionSort(int ar[],int size){
  int i, j, min_idx;

  for (i = 0; i < size-1; i++) {
    min_idx = i;
    for (j = i+1; j < size; j++)
      if (ar[j] < ar[min_idx])
        min_idx = j;
    swap(&ar[min_idx], &ar[i]);
  }
}
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
```



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?