Lab 7:

Supporto hardware alle procedure

Codifica ASCII

- *American Standard Code for Information Interchange*
- Utilizza 8 bit (1 byte) per rappresentare i caratteri

- `load byte unsigned (lbu)` prende un byte dalla memoria mettendolo negli 8 bit di un registro, collocati più a destra
- `store byte (sb)` prende il byte corrispondente agli 8 bit di un registro, collocati più a destra, e lo salva in memoria

```
lbu x12, 0(x10) // Leggi un byte dall'indirizzo sorgente
sb  x12, 0(x11) // Scrivi il byte all'indirizzo di destinazione
```

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

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
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

Il linguaggio C termina le stringhe con un byte che contiene il valore 0 (carattere “NULL” in ASCII)

Esercizio 3 – `strlen` (String Length)

Scrivere una procedura RISC-V per calcolare la lunghezza di una stringa di caratteri in C, escluso il carattere terminatore. Le stringhe di caratteri in C sono memorizzate come un array di byte in memoria, dove il byte '\0' (`0x00`) rappresenta la fine della stringa.

```
unsigned long strlen(char *str) {
    unsigned long i;
    for (i = 0; str[i] != '\0'; i++);
    return i;
}
```

```
.globl _start
.data
      src: .string "This is the source
string."
```

Esercizio 3 – `strlen` (String Length)

```
.globl _start

.data
      src: .string "This is the source string."

.text
_start:

      # call strlen
      la   a0, src
      jal  ra, strlen

      # print size, ret in a0
      li    a7, 1
      ecall
```

Main

Esercizio 4 – `strcmp`

Scrivere una procedura RISC-V `strcmp` per confrontare due stringhe di caratteri. `strcmp(str1, str2)` ritorna 0 se `str1` è uguale a `str2`, 1 nel caso contrario.

risultato atteso = 1

```
.globl _start
.data
      str1: .string "first"
      str2: .string "second"
```

Esercizio 4 – `strcmp`

```
.globl _start

.data
      str1: .string "first."
      str2: .string "second."

.text
_start:

      # call strcmp
      la   a0, str1
      la   a1, str2
      jal  ra, strcmp
...
```

Main

Esercizio 5 – `strchr`

Scrivere una procedura RISC-V `strchr(str, char)` per restituire l'indirizzo in memoria della prima occorrenza di `char` in `str`.
`strchr(str, char)` ritorna 0 se `char` non è presente in `str`.

```
.data
      str: .string "my long string"
      char: .string "g"
.text
_start:
      la   a0, str
      la   t1, char
      lbu  a1, 0(t1)
      jal  ra, strchr
```

str è all'indirizzo = `0x0000000010010000`
risultato atteso = `0x0000000010010006`

Esercizio 6 – `strrchr`

```
# a0 = const char *str
# a1 = char
strrchr:
      add  t2, zero, zero          # return address

strrchr_loop:
      lbu  t1, 0(a0)               # dereference str[i]
      beq  t1, zero, strrchr_end   # if str == \0 done
      bne  t1, a1, strrchr_cont
      add  t2, a0, zero            # if str[i] == char, update t2
strrchr_cont:
      addi a0, a0, 1
      j    strrchr_loop

strrchr_end:
      add  a0, t2, zero            # return t2
      ret
```

Esercizio 7 – `strcpy` (String Copy)

Scrivere una procedura RISC-V per copiare una stringa in un'altra (`strcpy`). Assumere che `dst` abbia spazio sufficiente in memoria per ricevere i byte di `src`, cioè che `strlen(dst)>=strlen(src)`

Nota: `strcpy` deve utilizzare `strlen`, come in questo codice in C:

```
void strcpy(char *dst, char *src) {
    unsigned long i;
    unsigned long n;
    n = strlen(src);
    m = strlen(dst);
    for (i = 0; i < n; i++)
        dst[i] = src[i];
    for ( ; i < m; i++)
        dst[i] = '\0';
    return;
}
```

```
.data
      src: .string "source"
      dst: .string "------------"
```

Esercizio 9 - Somma Array

Scrivere due versioni per una procedura che calcoli la somma di un array di word in memoria:

una iterativa (cfr. Lab 5, Esercizio 4)
una ricorsiva → somma := v[1] + somma(v[2:s])

- Quante istruzioni RISC-V sono necessarie per realizzare le procedure?
- Quante istruzioni RISC-V verranno eseguite per completare le procedure quando l'array contiene 16 elementi?
- Quanti registri sono stati versati in memoria (*registrer spilling*) durante l'esecuzione delle due versioni?