Lab 7:

Altri esercizi su procedure

Obiettivi

- Tradurre procedure da C ad assembly
- Far pratica con le "convenzioni di chiamata"
- Far pratica con l'utilizzo dello stack

Procedura: sottoprogramma memorizzato che svolge un compito specifico basandosi sui parametri che gli vengono passati in ingresso.

Esecuzione di una procedura

Per l'esecuzione di una procedura, un programma deve eseguire questi sei passi:

1. Mettere i **parametri** in un luogo accessibile alla procedura;
2. **Trasferire il controllo** alla procedura;
3. **Acquisire le risorse** necessarie per l'esecuzione della procedura;
4. **Eseguire** il compito richiesto;
5. Mettere il **risultato** in un luogo accessibile al programma chiamante;
6. **Restituire il controllo** al punto di origine, dato che la stessa procedura può essere chiamata in diversi punti di un programma.

Parametri e Indirizzo di Ritorno

- registri `a0-a7 (x10-x17)` sono 8 registri per i parametri, utilizzati cioè per passare valori alle funzioni o restituire valori al chiamante
- registro `ra(x1)` contiene l'indirizzo di ritorno

`jal` e `jalr`: Passaggio di Controllo

- L'istruzione **`jal` (jump and link)** serve per la chiamata di funzioni: produce un salto a un indirizzo e salva l'indirizzo dell'istruzione successiva a quella del salto nel registro **`ra`**
  (indirizzo di ritorno, detto appunto link)

```
jal ra, ProcAddress # salta a ProcAddress e salva indirizzo di ritorno in ra
```

[jal ProcAddress]

- Il ritorno da una procedura utilizza un salto indiretto, **jump and link register (`jalr`)**

```
jalr zero, 0(ra) # salta indietro all'indirizzo di ritorno presente in ra
```

[jr ra] oppure [ret]

`jal` e `jalr`: Passaggio di Controllo

Lo schema è quindi il seguente:

- la funzione chiamante mette i parametri in **`a0-a7`** e usa **`jal X`** per saltare alla funzione **`X`**
- la funzione chiamata svolge le proprie operazioni, inserisce i risultati negli stessi registri e restituisce il controllo al chiamante con l'istruzione **`jr ra`**

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

Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere
32	Spazio	48	0	64	@	80	P	096	`	112	p
33	!	49	1	65	A	81	Q	097	a	113	q
34	“	50	2	66	B	82	R	098	b	114	r
35	#	51	3	67	C	83	S	099	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	‘	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Codifica ASCII

Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere	Valore ASCII	Carattere
32	Spazio	48	0	64	@	80	P	096	`	112	p
33	!	49	1	65	A	81	Q	097	a	113	q
34	“	50	2	66	B	82	R	098	b	114	r
35	#	51	3	67	C	83	S	099	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	‘	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44		60	<	76	L	92	\	108	l	124	|

Il linguaggio C termina le stringhe con un byte che contiene il valore 0 (carattere “null” in ASCII, non mostrato nella tabella)

Esercizio 4 – `strcmp`

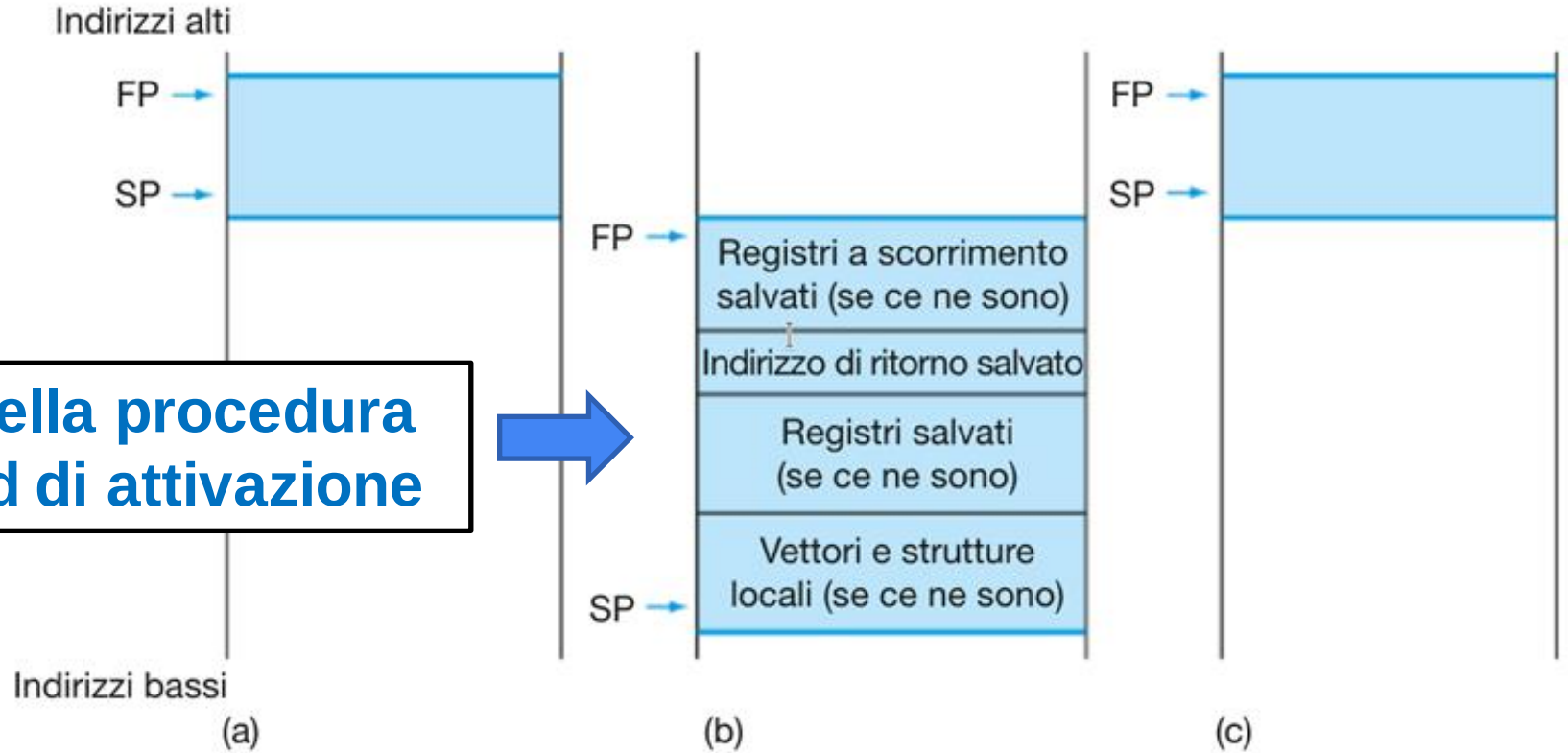
Scrivere una procedura RISC-V `strcmp` per confrontare due stringhe di caratteri. `strcmp(str1, str2)` ritorna 0 se `str1` è uguale a `str2`, 1 nel caso contrario.

risultato atteso = 1

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

Salvataggio sullo stack del contenuto dei registri

Stack



frame della procedura o record di attivazione

- Lo stack ‘cresce’ **da indirizzi di memoria alti verso indirizzi di memoria bassi**
- Quindi quando vengono inseriti dati nello stack il valore dello `sp` **diminuisce**
- `sp` **aumenta** quando i dati sono estratti dallo stack

Convenzioni di chiamata

Per evitare costose operazioni di spilling (salvataggio su stack) e di restore (ri-salvataggio da stack a registri) utilizziamo una convenzione. Dividiamo i registri di uso generale in due categorie: **quelli preservati** nel passaggio fra chiamate di funzione, e quelli **non preservati** fra le chiamate

s0..s11: **registri preservati**. Ovvero: chi chiama una procedura può assumere che la procedura chiamata non ne altererà il valore.
t0..t6: **registri non preservati**. Ovvero: chi chiama una procedura deve considerare la possibilità che la procedura chiamata ne alteri il valore

Esercizio 8 - Inverte Array

Scrivere una procedura **swap(v, x, y)** che scambi i valori di **v[x]** e **v[y]**, dove **v** è l'indirizzo di un array in memoria. Scrivere poi un'altra procedura **invert(v, s)**, che utilizzi **swap** per invertire un array in memoria.

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

- Quante istruzioni RISC-V sono necessarie per implementare la procedura?
- Quante istruzioni RISC-V verranno eseguite per completare la procedura quando l'array contiene 16 elementi?
- Quanti registri sono stati versati in memoria (*registrer spilling*) durante l'esecuzione?

*Bonus: Realizzare un metodo **print(v, s)** che stampa **v** sulla console*

Esercizio 10 - Fibonacci Ricorsivo

Tradurre il seguente frammento di codice C in codice assembly RISC-V.

```
int fib(int n) {
      if (n==0)
            return 0;
      else if (n==1)
            return 1;
      else
            return(fib(n-1) + fib(n-2));
}
```

- Quante istruzioni RISC-V sono necessarie per implementare la funzione?
- Quante istruzioni RISC-V verranno eseguite per completare la funzione quando N=8?
- Per N=8, quanti registri sono stati versati in memoria (*registrer spilling*) durante l'esecuzione?

Esercizi difficili (per chi si annoia)

- Utilizzando la system call 11, che stampa sulla console di output un singolo carattere ASCII, implementare la system call numero 1, ovvero scrivere una procedura **printNum**(dword) che stampa sulla console il numero (su 64 bit) passato come argomento. Attenzione: gestire anche il caso di numero negativo

- Scrivere una procedura **printHex**(dword) che stampa sulla console di output il numero (su 64 bit) passato come argomento in notazione esadecimale, ovvero nella forma: 0x….