Φροντιστήριο
13/11/2014

Τι θα θυμηθούμε....

1. Συμβάσεις καταχωρητών
2. Caller – Callee save καταχωρητές
3. Κλήση συνάρτησης
4. Χρήση στοίβας
5. Δέσμευση Μνήμης
6. Assembly I/O
7. Ascii πίνακας
8. Παράδειγμα

Συμβάσεις καταχωρητών (1/2)

- MIPS
  - 32 Καταχωρητές
- Ονομασία καταχωρητών
  - Απλή ονομασία, π.χ. $1, $2 κτλ
  - Συμβολική ονομασία, π.χ. $t0, $a1, κτλ
- Συμβολική ονομασία➔ υποδηλώνουν την χρήση τους στις συμβάσεις χρήσης

Συμβάσεις καταχωρητών (2/2)

Συμβολική Ονομασία Καταχωρητών	Λειτουργία
$zero	Περιέχει την τιμή 0
$v0, $v1	I/O και επιστρεφόμενη τιμή από συνάρτηση
$a0..$a3	Ορίσματα συναρτήσεων
$t0..$t9	Τοπικοί καταχωρητές
$s0..$s7	Saved καταχωρητές
$sp	Stack pointer
$ra	Δειύθυνσης επιστροφής
$fp	Frame pointer
$k0, $k1	Kernel registers (Interrupts)
$gp	Global pointer
$at	Assembler temporary register

Παραδείγματα χρήσης καταχωρητών

1. $v0, $v1

```
li $v0, 4
la $a0, str1
syscall
```

```
li      $v0, 1          # print integer
la      $a0, 1500       # integer to print
syscall
```

2. $a0, …, $a3

```
li $a0, 4
li $a1, 10
Jal func
```

3. $sp

```
Addi $sp, $sp, -4
Sw $t0, 0($sp)
....
lw $t0, 0($sp)
addi $sp, $sp, 4
```

Caller – Callee Καταχωρητές (1/2)

- Callee-Save καταχωρητές
  - $s0-$s7
- Caller-Save καταχωρητές
  - $t0-$t9
- Η κάθε συνάρτηση δίνει εγγύηση ότι δεν θα αλλάξει τους $s0-$s7
- Η κάθε συνάρτηση μπορεί να αλλάξει τους υπόλοιπους καταχωρητές: $t0-$t7, $a0-$a3, $v0, $v1, $at, $ra

Caller – Callee Καταχωρητές (2/2)

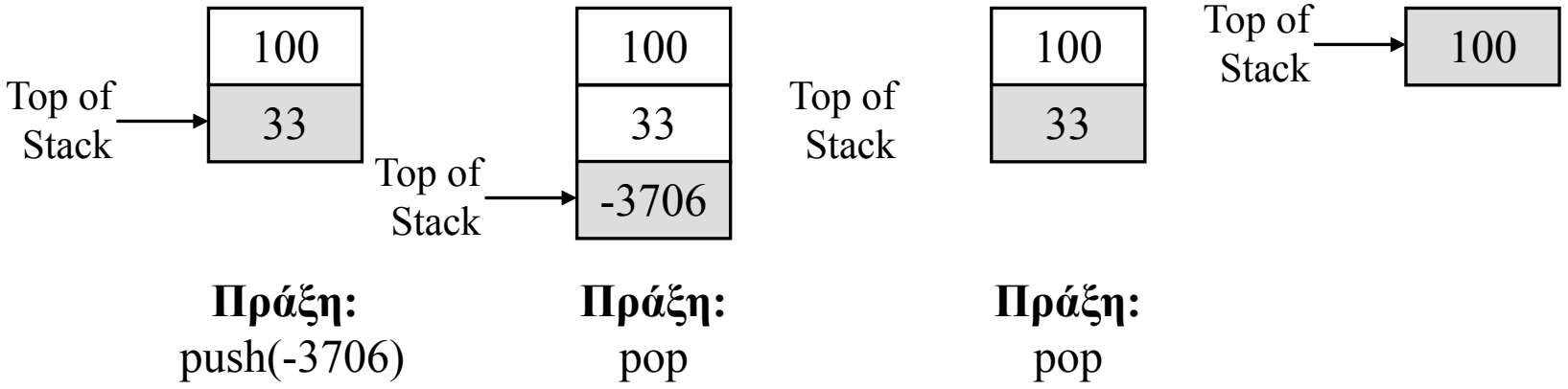
- Λειτουργία
  - Όταν μία συνάρτηση χρησιμοποιεί στο σώμα της τους καταχωρητές **$s** θα πρέπει να τους αποθηκεύουμε στον πρόλογο και να τους επαναφέρουμε στον επίλογο της συνάρτησης
  - Όταν μία συνάρτηση θέλει να διατηρήσει την τιμή για κάποιους από τους καταχωρητές **$t** μετά την κλήση μίας άλλης συνάρτησης τότε θα πρέπει να αποθηκεύουμε τους καταχωρητές πριν την κλήση της άλλης συνάρτησης και να τους επαναφέρουμε αφού επιστρέψουμε.
  - Όμοια για τους καταχωρητές: $ra, $v, $a…

Κλήση συνάρτησης

1. Ορίσματα
   - Καταχωρητές $a0,…,$a3
2. Επιστρεφόμενη τιμή
   - $v0
3. Κληση
   - jal func
   - Τι ακριβώς κάνει η εντολή jal?
4. Return
   - Jr $ra

Χρήση Στοίβας

- Που αποθηκεύουμε τις τιμές των caller-save ή των callee-save καταχωρητών?
  - **ΣΤΟΙΒΑ!!**
- Στοίβα
  - LIFO: Last-In, First-Out.
  - Πράξεις: ***Push*** (εισαγωγή στοιχείου στην στοίβα), ***Pop*** (αφαίρεση στοιχείου από την στοίβα)



Τρόπος γραφής κώδικα

Χρήση Caller-save καταχωρητών	**Χρήση Callee-save καταχωρητών**

Func0:
addi $sp, $sp, -4
sw $ra, 0($sp)
move $t0, $a0
…..
addi $sp, $sp, -4
sw $t0, 0($sp)
jal func1
lw $t0, 0($sp)
addi $sp, $sp, 4
…..
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
	Func0:
	addi $sp, $sp, -8
	sw $ra, 0($sp)
	sw $s0, 4($sp)
	move $s0, $a0
	…..
	jal func1
	…..
	
	lw $s0, 4($sp)
	lw $ra, 0($sp)
	addi $sp, $sp, 8
	jr $ra

Οδηγίες για Δέσμευση Μνήμης

.word w1 [,…, wn]	Δεσμεύει χώρο για n λέξεις όπου και αποθηκεύει τις τιμές w1, …, wn.
.byte b1 [,…, bn]	Δεσμεύει χώρο για n bytes όπου και αποθηκεύει τις τιμές b1, …, bn.
.asciiz str	Όπως και η .ascii αλλά με ένα μηδέν μετά τον τελευταίο χαρακτήρα (για γλώσσες όπως «C»)
.space n	Δεσμεύει χώρο για n bytes (χωρίς αρχικοποίηση)
.align n	Ευθυγραμμίζει την διεύθυνση του επόμενου δεδομένου σε πολ/σιο του 2^n. (.align 2 => διεύθυνση πολ/σιο του 4).

I/O MIPS

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
read character	12	(none)	char in $v0
print character	11	$a0 = integer	(none)

Πίνακας Ascii

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

Source: www.LookupTables.com

Παράδειγμα

Γράψτε ένα πρόγραμμα assembly όπου ο χρήστης θα επιλέγει:

1: Αν θα διαβάζει έναν αριθμό και θα τον αποκρυπτογραφεί σε γράμμα.

2: Αν θα διαβάζει μία συμβολοσειρά και θα την κρυπτογραφεί σε μία αλληλουχία αριθμών.

Η αποκρυπτογράφηση και η κρυπτογράφηση θα γίνεται με τον εξής τρόπο:

Κρυπτογράφηση: a → 0, b → 1, …, A → 26, B→ 27

Αποκρυπτογράφηση: 0 → a, 1 → b, …, 26 → A, 27→ B

```
#include <stdio.h>
char str[100];

char decode_func(int a);
void encode_func(char *str);

int main()
{
	int choice, encoded, decoded;
	printf("Please give your choice (1 or 2): ");
	scanf("%d", &choice);
	if(choice == 1)
	{
		printf("Please give the number:");
		scanf("%d",&encoded);
		decoded = decode_func(encoded);
		printf("The decoded character is: %c", decoded);
	}
	else if(choice == 2)
	{
		printf("Please give the string:");
		scanf("%s", str);
		encode_func( &str[0]);
	}
	return(0);
}
```

Main Function

Function Encoding

```
void encode_func(char *str)
{
	int i;
	int result;
	i = 0;

	while(str[i] != 0)
	{
		if(str[i] >= 97)
			result = str[i] - 97;
		else
			result = str[i] - 65 + 26;

		printf("%d ",result);
		i++;
	}
}
```

Function Decoding

```
char decode_func(int a)
{
    char start;

    if(a > 25)
    {
        start = 65;
        start = start + (a - 26);
    }
    else
    {
        start = 97;
        start = start + a;
    }

    return(start);
}
```