

FUNDAMENTOS DE COMPUTADORES

Práctica 7

Ensamblador: Datos en memoria: representación y acceso.

INTRODUCCIÓN Y OBJETIVOS

En la presente práctica se continúa con la programación en ensamblador MIPS haciendo uso del simulador PCSpim. Los objetivos de la misma son:

- Elaborar programas que declaren y manipulen datos almacenados en memoria.
- Comprender la representación de diferentes tipos de datos: enteros, reales y caracteres.
- Introducir el uso de pseudoinstrucciones.
- Utilizar instrucciones para el acceso a datos en memoria.

NOTA: Para que el PCSPIM cargue correctamente las pseudoinstrucciones, se debe configurar (por medio del menú *Simulator* → *Settings*) de la siguiente forma: Desactivar la opción *Bare Machine* y activar la opción *Allow pseudo instructions*.

USO DE LAS DIRECTIVAS DE DATOS

Las directivas de datos son indicaciones al programa ensamblador (el programa que traduce desde lenguaje ensamblador a código máquina) que le serán de utilidad en el momento de generar código ejecutable. Estas directivas no generan código de máquina, es decir, no generan instrucciones. El significado de algunas de las directivas más utilizadas se muestra en la siguiente tabla:

Directiva	Significado
<i>.data <addr></i>	Almacena los elementos declarados en el segmento de datos, a partir de la dirección indicada en <i>addr</i> .
<i>.byte b₁,...,b_n</i>	Almacena <i>n</i> valores enteros en palabras sucesivas de 8 bits.
<i>.half h₁,...,h_n</i>	Almacena <i>n</i> valores enteros en palabras sucesivas de 16 bits.
<i>.word w₁,...,w_n</i>	Almacena <i>n</i> valores enteros en palabras sucesivas de 32 bits.
<i>.float f₁,...,f_n</i>	Almacena <i>n</i> valores reales en palabras sucesivas de 32 bits.
<i>.ascii str</i>	Almacena la cadena <i>str</i> en memoria. Un byte por carácter.
<i>.asciiz str</i>	Almacena la cadena <i>str</i> en memoria. Un byte por carácter y añade al final el carácter NULL.

Los enteros se representan, todos, en complemento a dos. Los reales, en formato IEEE754 de simple precisión. Los caracteres se almacenan en ASCII de 8 bits.

Cuando se cargue un programa que incluya directivas de datos, se verá inmediatamente reflejado el efecto de estas directivas en cuanto a reserva e inicialización de posiciones de

memoria en el segmento de datos. Esto podrá observarse mediante la ventana de segmento de datos del simulador.

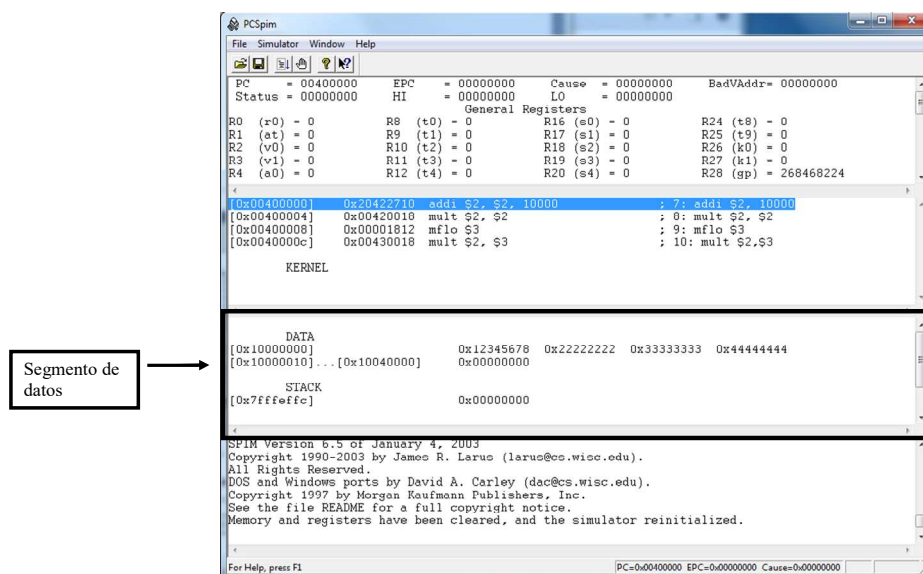


Figura 1. Localización de la ventana de segmento de datos dentro del entorno gráfico del simulador PCSpim.

Ventana de segmento de datos: en esta ventana se muestra el contenido de la memoria. La memoria se despliega mostrando rangos de direcciones y el contenido de este rango. De manera que una línea con la siguiente información:

[0x10000000] 0x12345678 0x22222222 0x33333333 0x44444444

Se interpreta de la siguiente forma:

1. Se muestra el contenido de las cuatro palabras de memoria que están en las direcciones 0x10000000 a 0x1000000C. El tamaño de una palabra es de 32 bits. Por lo tanto, en una línea se muestran $32 \times 4 = 128$ bits.
2. La palabra de memoria 0x10000000 tiene el contenido 0x12345678 (32 bits)
 1. Los bytes que componen esta palabra ocuparán, por lo tanto, las posiciones siguientes según el formato *little endian*:

[0x10000000]	0x78
[0x10000001]	0x56
[0x10000002]	0x34
[0x10000003]	0x12
3. La palabra de memoria 0x10000004 tiene el contenido 0x22222222 (32 bits)
4. La palabra de memoria 0x10000008 tiene el contenido 0x33333333 (32 bits)
5. La palabra de memoria 0x1000000C tiene el contenido 0x44444444 (32 bits)

ALINEACIÓN DE DATOS EN MEMORIA

La alineación de datos en memoria es un requisito de muchos procesadores. Esta alineación de los datos quiere decir que cuando el procesador accede a un dato de un determinado tipo (byte, half o word), lo hará empleando direcciones con unas características concretas y particulares. Por ejemplo, cuando el procesador quiere leer un dato de tipo word, la dirección es necesario que sea múltiplo de cuatro, ya que los words ocupan cuatro bytes. En cambio, las direcciones que hacen referencia a datos de tipo half es necesario que sean múltiplos de dos, mientras que las de tipo byte no tienen ninguna restricción.

Para mantener la relación entre el tipo de dato y su dirección, el programa ensamblador, al cargar el programa y procesar a las directivas, dejará los espacios necesarios entre datos declarados consecutivamente.

Considere el código que aparece a continuación y responda a las cuestiones siguientes:

```
.data 0x10000000
.byte 1,-1,2
.half -2, 3
.byte 4
.word -4
.byte 5
.half 6
```

Pregunta 1. Rellene de forma teórica la tabla siguiente con los contenidos que tendrá la memoria al cargar y procesar a las directivas.

	Contenido (Decimal y hexadecimal)			
Dirección (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x10000000				
0x10000004				
0x10000008				
0x1000000C				
0x10000010				
0x10000014				

Pregunta 2. Una vez en el laboratorio, teclee y cargue el código anterior en el PCSpim, y llene nuevamente la tabla, esta vez copiando los datos tal como aparecen en la ventana del segmento de datos.

	Contenido (Decimal y hexadecimal)			
Dirección (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x10000000				
0x10000004				
0x10000008				
0x1000000C				
0x10000010				
0x10000014				

REPRESENTACIÓN DE REALES

Los números reales se representan en la mayoría de computadores con el formato IEEE754 de simple precisión. En este formato, se utilizan los criterios siguientes para representar un número real:

- El formato es:

1 bit	8 bits	23 bits
Signo	Exponente	Mantisa

- El bit de signo a cero quiere decir que la cantidad representada es positiva. Y con un uno, quiere decir que es negativa.
- El exponente se representa en exceso Z, con $Z=127$, Esto quiere decir que si el exponente de la cantidad que queremos representar es 3, el que realmente se almacenará es 130.
- La mantisa es necesario que esté normalizada de la forma 1.xxx, y se almacena empleando la técnica del bit implícito. Es decir, el primer 1 no se almacena.

Teclee el código siguiente en un editor, cárguelo en el PCSpim, y responda a las siguientes cuestiones:

```
.data 0x10000000
.float 1.0 , -1.0 , 3.23, -3.23
```

Pregunta 3. Llene la tabla siguiente con los contenidos de la memoria mostrada en la ventana del segmento de datos.

	Contenido (Decimal y hexadecimal)			
Dirección (HEX)	31 ... 24	23 ... 16	15 ... 8	7 ... 0
0x10000000				
0x10000004				
0x10000008				
0x1000000C				
0x10000010				

Pregunta 4. Indique los campos de signo, exponente y mantisa (en binario los tres) de los cuatro valores reales definidos en el código anterior:

	Signo	Exponente	Mantisa
1.00			
-1.00			
3.23			
-3.23			

REPRESENTACIÓN DE CARACTERES

Teclee el código siguiente en un editor, cargadlo en el PCSpim, y responda las siguientes cuestiones:

```
.data 0x10000000  
.asciiz "el profe"  
.byte 16  
.ascii "el profe"  
.byte 16
```

Pregunta 5. Llene la tabla siguiente con los contenidos de la memoria mostrada en la ventana del segmento de datos.

	Contenido (Decimal y hexadecimal)			
Dirección (HEX)	31 . . . 24	23 . . . 16	15 . . . 8	7 . . . 0
0x10000000				
0x10000004				
0x10000008				
0x1000000C				
0x10000010				

Pregunta 6. ¿En qué dirección de memoria se encuentran las letras “p”?

Dirección en hexadecimal:

LOCALIZACIÓN DE INSTRUCCIONES Y DE DATOS

Las siguientes preguntas se refieren al código que aparece a continuación. Escríbalo usando un editor de textos, llame al archivo “practica_7_1.s, cárguelo en el PCSPIM y compruebe que se carga correctamente.

```
.globl __start
.data 0x10000000
base:   .word 3
altura: .word 10
area:   .space 4

.text 0x00400000
__start:
    la $10, base
    lw $11, 0($10)
    la $12, altura
    lw $13, 0($12)
    mult $11,$13
    mflo $14
    addi $15,$0,2
    div $14,$15
    mflo $16
    la $17, area
    sw $16, 0($17)
.end
```

Código 1. Cálculo del área de un triángulo (archivo “practica_7_1.s”)

Pregunta 7. Determine las direcciones de memoria asociadas con las siguientes etiquetas:

Dato	Dirección
base	0x10000000
altura	
area	

Pregunta 8. Determine las direcciones de memoria donde se almacenan las siguientes instrucciones:

Instrucción	Dirección
lui \$10, 0x1000	0x00400000
lw \$11, 0(\$10)	
lui \$1, 0x1000	
ori \$12, \$1, 0x0004	
lw \$13, 0(\$12)	
div \$14,\$15	

Pregunta 9. Indique el contenido del segmento de datos antes de iniciarse la ejecución, teniendo en cuenta que los datos se almacenan en formato “little endian”. El contenido debe ponerse en hexadecimal por cada byte de memoria.

31	...	24	23	...	16	15	...	8	7	...	0	Dirección

Pregunta 10. Indique el contenido del segmento de datos una vez finalizada la ejecución, teniendo en cuenta que los datos se almacenan en formato “little endian”. El contenido debe ponerse en hexadecimal por cada byte de memoria.

31	...	24	23	...	16	15	...	8	7	...	0	Dirección

Pregunta 11. Determinar el contenido de los siguientes registros cuando se haya finalizado la ejecución del programa.

Registro	Contenido	Qué representa el contenido
\$10		
\$11		
\$12		
\$13		
\$14		
\$15		
\$16		
\$17		

Pregunta 12. El programa propuesto utiliza muchos registros. Sin embargo, es posible realizar los mismos cálculos con menos registros. Reescribir el programa en un archivo llamado “practica_7_2.s donde se minimice el número de registros empleados

TABLA ASCII

REGULAR ASCII CHART (character codes 0 – 127)

000d	00h	␣	(nul)	016d	10h	▶	(dle)	032d	20h	␣	048d	30h	0	064d	40h	⓪	080d	50h	P	096d	60h	‘	112d	70h	p
001d	01h	␣	(sob)	017d	11h	◀	(dc1)	033d	21h	!	049d	31h	1	065d	41h	A	081d	51h	Q	097d	61h	a	113d	71h	q
002d	02h	␣	(stx)	018d	12h	†	(dc2)	034d	22h	"	050d	32h	2	066d	42h	B	082d	52h	R	098d	62h	b	114d	72h	r
003d	03h	♥	(etx)	019d	13h	‡	(dc3)	035d	23h	#	051d	33h	3	067d	43h	C	083d	53h	S	099d	63h	c	115d	73h	s
004d	04h	♠	(eot)	020d	14h	¶	(dc4)	036d	24h	\$	052d	34h	4	068d	44h	D	084d	54h	T	100d	64h	d	116d	74h	t
005d	05h	♣	(enq)	021d	15h	—	(nak)	037d	25h	%	053d	35h	5	069d	45h	E	085d	55h	U	101d	65h	e	117d	75h	u
006d	06h	♠	(ack)	022d	16h	␣	(syn)	038d	26h	&	054d	36h	6	070d	46h	F	086d	56h	V	102d	66h	f	118d	76h	v
007d	07h	•	(bel)	023d	17h	‡	(etb)	039d	27h	!	055d	37h	7	071d	47h	G	087d	57h	W	103d	67h	g	119d	77h	w
008d	08h	▣	(bs)	024d	18h	†	(can)	040d	28h	(056d	38h	8	072d	48h	H	088d	58h	X	104d	68h	h	120d	78h	x
009d	09h	␣	(tab)	025d	19h	↓	(em)	041d	29h)	057d	39h	9	073d	49h	I	089d	59h	Y	105d	69h	i	121d	79h	y
010d	0Ah	▣	(lf)	026d	1Ah	—	(eof)	042d	2Ah	*	058d	3Ah	:	074d	4Ah	J	090d	5Ah	Z	106d	6Ah	j	122d	7Ah	z
011d	0Bh	♂	(vt)	027d	1Bh	␣	(esc)	043d	2Bh	+	059d	3Bh	;	075d	4Bh	K	091d	5Bh	[107d	6Bh	k	123d	7Bh	{
012d	0Ch	♂	(np)	028d	1Ch	␣	(fs)	044d	2Ch	,	060d	3Ch	<	076d	4Ch	L	092d	5Ch	\	108d	6Ch	l	124d	7Ch	
013d	0Dh	♂	(cr)	029d	1Dh	␣	(gs)	045d	2Dh	-	061d	3Dh	=	077d	4Dh	M	093d	5Dh]	109d	6Dh	m	125d	7Dh	}
014d	0Eh	♂	(so)	030d	1Eh	▲	(rs)	046d	2Eh	.	062d	3Eh	>	078d	4Eh	N	094d	5Eh	^	110d	6Eh	n	126d	7Eh	~
015d	0Fh	♂	(si)	031d	1Fh	▼	(us)	047d	2Fh	/	063d	3Fh	?	079d	4Fh	O	095d	5Fh	_	111d	6Fh	o	127d	7Fh	◊

EXTENDED ASCII CHART (character codes 128 – 255) LATIN1/CP1252

128d	80h	€	90h	144d	90h	‘	160d	A0h	␣	176d	B0h	°	192d	C0h	À	208d	D0h	Ð	224d	E0h	à	240d	F0h	ð
129d	81h	‚	91h	145d	91h	‚	161d	A1h	¡	177d	B1h	±	193d	C1h	Á	209d	D1h	Ñ	225d	E1h	á	241d	F1h	ñ
130d	82h	ƒ	92h	146d	92h	“	162d	A2h	¢	178d	B2h	²	194d	C2h	Â	210d	D2h	Ò	226d	E2h	â	242d	F2h	ò
131d	83h	„	93h	147d	93h	”	163d	A3h	£	179d	B3h	³	195d	C3h	Ã	211d	D3h	Ó	227d	E3h	ã	243d	F3h	ó
132d	84h	…	94h	148d	94h	•	164d	A4h	¤	180d	B4h	´	196d	C4h	Ä	212d	D4h	Ô	228d	E4h	ä	244d	F4h	ô
133d	85h	†	95h	149d	95h	•	165d	A5h	¥	181d	B5h	µ	197d	C5h	Å	213d	D5h	Õ	229d	E5h	å	245d	F5h	ö
134d	86h	‡	96h	150d	96h	-	166d	A6h	¦	182d	B6h	¶	198d	C6h	Æ	214d	D6h	Ö	230d	E6h	æ	246d	F6h	ø
135d	87h	‡	97h	151d	97h	~	167d	A7h	§	183d	B7h	·	199d	C7h	Ç	215d	D7h	×	231d	E7h	ç	247d	F7h	÷
136d	88h	‰	98h	152d	98h	™	168d	A8h	¨	184d	B8h	¸	200d	C8h	È	216d	D8h	Ø	232d	E8h	è	248d	F8h	ø
137d	89h	‰	99h	153d	99h	§	169d	A9h	©	185d	B9h	¹	201d	C9h	É	217d	D9h	Ù	233d	E9h	é	249d	F9h	ù
138d	8Ah	Š	9Ah	154d	9Ah	Š	170d	AAh	ª	186d	BAh	º	202d	CAh	Ê	218d	DAh	Ú	234d	EAh	ê	250d	FAh	ú
139d	8Bh	<	9Bh	155d	9Bh	>	171d	ABh	«	187d	BBh	»	203d	CBh	Ë	219d	DBh	Û	235d	EBh	ë	251d	FBh	û
140d	8Ch	ƒ	9Ch	156d	9Ch	œ	172d	ACH	¬	188d	BCh	¼	204d	CCh	Ì	220d	DCh	Ü	236d	ECh	ì	252d	FCh	ü
141d	8Dh	Ž	9Dh	157d	9Dh	Ž	173d	ADh	™	189d	BDh	½	205d	CDh	Í	221d	DDh	Ý	237d	EDh	í	253d	FDh	ý
142d	8Eh	Ž	9Eh	158d	9Eh	ž	174d	AEnh	®	190d	BEh	¾	206d	CEh	Î	222d	DEh	Þ	238d	EEh	î	254d	FEh	þ
143d	8Fh	Ž	9Fh	159d	9Fh	Ÿ	175d	AFh	—	191d	BFh	¿	207d	CFh	Ï	223d	DFh	ß	239d	EFh	ï	255d	FFh	ÿ

Hexadecimal to Binary

0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Groups of ASCII-Code in Binary

Bit 6	Bit 5	Group
0	0	Control Characters
0	1	Digits and Punctuation
1	0	Upper Case and Special
1	1	Lower Case and Special

© 2009 Michael Goertz
This work is licensed under the Creative Commons
Attribution-NonCommercial-Share Alike 3.0 License.
To view a copy of this license, visit
<http://creativecommons.org/licenses/by-nc-sa/>