FUNDAMENTOS DE COMPUTADORES
Práctica 8 y 9
Ensamblador: Datos en memoria. Análisis de programas. Codificación de instrucciones.

Apellidos y nombre	Grupo	DNI
Lurbe Sempere Manel	1A1	

Durante la realización de la práctica **en el laboratorio** deberás responder las preguntas de un **examen** de Poliformat. Es conveniente que visualices y resuelvas este examen al mismo tiempo que avanzas el trabajo de la práctica. Si estáis realizando la práctica en pareja tenéis dos opciones:
Opción a: primero realizáis el examen con uno de los usuarios, y una vez finalizado el trabajo y el examen, cambiáis de usuario y volvéis a realizar el examen.
Opción b: utilizáis dos navegadores diferentes al mismo tiempo, por ejemplo IE y Firefox. Esto os permite abrir dos sesiones diferentes en poliformat y responder los dos exámenes al mismo tiempo.
Es **importante que resuelvas antes de acudir** al laboratorio los ejercicios que se plantean en este boletín, ya que las preguntas del examen están muy relacionadas con ellos, y no tendrás tiempo para hacerlo todo en el laboratorio.

INTRODUCCIÓN Y OBJETIVOS

En esta última práctica se vuelve a hacer uso del simulador PCSpim para la ejecución de programas en ensamblador. Los objetivos que se persiguen son los siguientes:

- Comprender la representación de datos en memoria.
- Elaborar programas que declaren y manipulen datos almacenados en memoria.
- Utilizar instrucciones para el acceso a datos en memoria.
- Analizar programas con estructura iterativa, que incluyen instrucciones de salto condicional e incondicional.
- Estudiar programas que procesan cadenas de caracteres.
- Analizar la codificación de instrucciones de diferente tipo.

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 máquina, es decir, no generan instrucciones. Algunas de las directivas más utilizadas se muestran en la siguiente tabla:

Directiva	Significado
.data <addr>	Almacena los elementos declarados en el segmento de datos, a partir de la dirección indicada en addr.
.byte $b_{1,..,}bn$	Almacena *n* valores enteros en palabras sucesivas de 8 bits.
.half $h_{1,..,}hn$	Almacena *n* valores enteros en palabras sucesivas de 16 bits.
.word $w_{1,..,}wn$	Almacena *n* valores enteros en palabras sucesivas de 32 bits.
.float $f_{1,..,}fn$	Almacena *n* valores reales en palabras sucesivas de 32 bits.
.ascii str	Almacena la cadena str en memoria. Un byte por carácter.
.asciiz str	Almacena la cadena str 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.

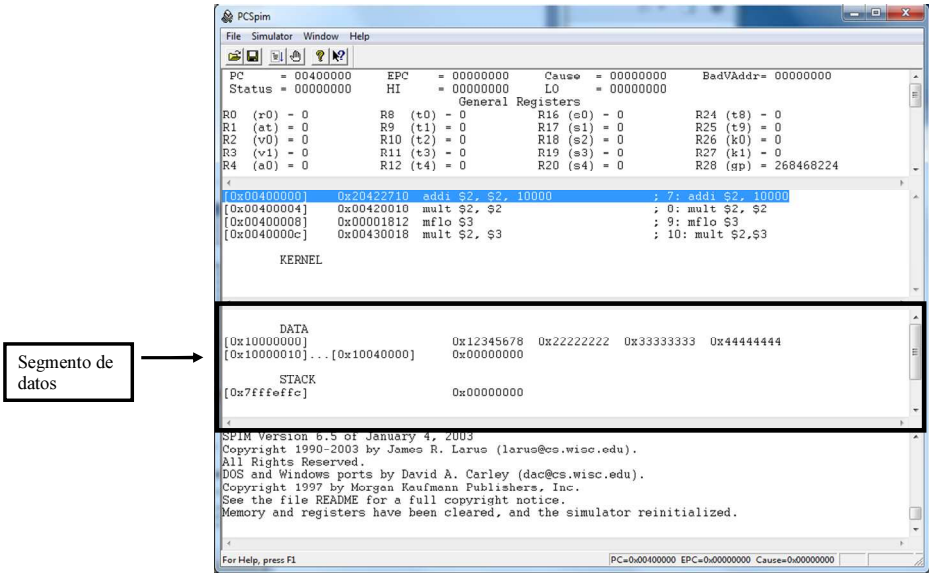

Figura 1. Localización de la ventana de segmento de datos dentro del entorno gráfico del simulador PCSpim.

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 0x3333333 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 32x4 = 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		0x02	0xff	0x01
0x10000004	0x00	0x03	0xff	0xfe
0x10000008				0x04
0x1000000C	0xff	0xff	0xff	0xfc
0x10000010	0x00	0x06		0x05
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	0x00	0x02	0xff	0x01
0x10000004	0x00	0x03	0xff	0xfe
0x10000008	0x00	0x00	0x00	0x04
0x1000000C	0xff	0xff	0xff	0xfc
0x10000010	0x00	0x06	0x00	0x05
0x10000014	0x00	0x00	0x00	0x00

Marque la sentencia que corresponda:

- No hice la tabla teórica en casa.
- Las dos tablas no coinciden.
- Las dos tablas me han salido idénticas.
- No entiendo de qué estamos hablando

Examen (si estas en el laboratorio):

Ahora es el momento de acudir a poliformat y responder la primera pregunta del examen.

REPRESENTACIÓN DE CARACTERES

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

```
.data 0x10000000
.asciiz “el profe”
.byte 16
.ascii “el profe”
.byte 16
```

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	0x70	0x20	0x6c	0x65
0x10000004	0x65	0x66	0x6f	0x72
0x10000008	0x6c	0x65	0x10	0x00
0x1000000C	0x6f	0x72	0x70	0x20
0x10000010	0x00	0x10	0x65	0x66

Pregunta 4. ¿En qué dirección de memoria se encuentran las letras “p”?

Dirección en hexadecimal: se encuentra en el registro numero 3

TRUNCAMIENTO DE UNA CADENA DE CARACTERES

El código que se va a emplear en las siguientes preguntas es el siguiente. Escríbalo en un archivo y llámelo “practica_9_1.s”, cárguelo en el PCSPIM y compruebe que se carga correctamente.

```
.data 0x10000000
ini:  .asciiz "Hola"
sel:  .word 3
res:  .space 4
tot:  .word -1

.text 0x00400000
.globl __start
__start:
      la $8, ini
      la $9, sel
      lw $9,0($9)
      la $10, res
      add $11,$0,$0
bucle:
      beq $9,$0,fin
      lb $12,0($8)
      sb $12,0($10)
      addi $8,$8,1
      addi $9,$9,-1
      addi $10,$10,1
      addi $11,$11,1
      j bucle
fin:
      la $12, tot
      sw $11,0($12)
.end
```

Código 1. Truncamiento de una cadena de caracteres ASCII (archivo "practica_9_1.s")

Pregunta 5. 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
0x61	0x6c	0x6f	0x48	0x10000000
0x00	0x00	0x00	0x00	0x10000004
0x00	0x00	0x00	0x03	0x10000008
0x00	0x00	0x00	0x00	0x1000000c
0xff	0xff	0xff	0xff	0x10000010

Pregunta 6. 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
0x61	0x6c	0x6f	0x48	0x10000000
0x00	0x00	0x00	0x00	0x10000004
0x00	0x00	0x00	0x03	0x10000008
0x00	0x6c	0x6f	0x48	0x1000000c
0x00	0x00	0x00	0x03	0x10000010

Pregunta 7. Determinar el contenido de los siguientes registros cuando se haya ejecutado por primera vez la instrucción j bucle.

Registro	Contenido
$8	10000001
$9	00000002
$10	1000000d
$11	00000001
$12	00000048

Pregunta 8. Determinar el contenido de los siguientes registros cuando se haya finalizado la ejecución del programa.

Registro	Contenido
$8	10000003
$9	00000000
$10	1000000f
$11	00000003
$12	10000010

Escriba el código que aparece en el PCSPIM de las instrucciones que se detallan, tanto en hexadecimal como en binario, separando (en el caso del binario) cada campo.

Pregunta 9. Instrucción tipo R.

Instrucción:	`add $11, $0, $0`	Código hexadecimal:	0x00005820		
Código binario:					
CO	rs	rt	rd	Numdesp	Función
000000	00000	00000	01011	00000	100000

Pregunta 10. Instrucción tipo I.

Instrucción:	`lb $12, 0($8)`	Código hexadecimal:	0x810c0000
Código binario:			
CO	rs	rt	Desp/Inm
100000	01000	01100	0000000000000000

Pregunta 11. Localice en el PCSPIM los siguientes códigos de instrucciones y determine con qué instrucciones se corresponden:

Código	Instrucción
0x3c011000	00001111 0x0f lui
0x11200008	00001100 0x04 beq
0x08100007	00000010 0x02 j

Examen (si estas en el laboratorio):
Ahora es el momento de acudir a poliformat y responder las preguntas relacionadas con el programa "práctica_9_1.s".

FUNCIONALIDAD DE PROGRAMAS

Escriba el siguiente código en un archivo "practica_9_2.s", cárguelo en el PCSPIM y compruebe que se carga correctamente. A partir de la ejecución del mismo, responda a las siguientes preguntas.

```
.globl __start
.data 0x10000000
vector:     .word 3,4,5
ncomp:      .word 3
result:     .space 4

.text 0x00400000
__start:
      la $8, vector
      la $9, ncomp
      lw $9, 0($9)
      addi $10, $0, 0
bucle:
      beq $9, $0, fin
      lw $11, 0($8)
      add $10, $10, $11
      addi $8, $8, 4
      addi $9, $9, -1
      j bucle
fin:
      la $12, result
      sw $10,0($12)
.end
```

Código 2. practica_9_2.s

Pregunta 12. De las siguientes funciones, determine cuáles de ellas implementa el código anterior y describa cómo se ha llegado a esa respuesta, teniendo en cuenta que la posición del primer elemento de un vector comienza por 0.

$f = \sum_{i=0}^{2} vector[i]$	$f = \sum_{i=0}^{2} i + vector[i]$	$f = \sum_{i=0}^{2} i \cdot vector[i]$
(a)	(b)	(c)

Código	Función
practica_9_2.s	

Justificación de la respuesta

Examen (si estas en el laboratorio):
Ahora es el momento de acudir a Poliformat y terminar el examen

TABLA ASCII

REGULAR ASCII CHART (character codes 0 – 127)

000*d*	00*h*	␀	(nul)	016*d*	10*h*	►	(dle)	032*d*	20*h*	␣	048*d*	30*h*	0	064*d*	40*h*	@	080*d*	50*h*	P	096*d*	60*h*	`	112*d*	70*h*	p
001*d*	01*h*	☺	(soh)	017*d*	11*h*	◄	(dc1)	033*d*	21*h*	!	049*d*	31*h*	1	065*d*	41*h*	A	081*d*	51*h*	Q	097*d*	61*h*	a	113*d*	71*h*	q
002*d*	02*h*	☻	(stx)	018*d*	12*h*	↕	(dc2)	034*d*	22*h*	"	050*d*	32*h*	2	066*d*	42*h*	B	082*d*	52*h*	R	098*d*	62*h*	b	114*d*	72*h*	r
003*d*	03*h*	♥	(etx)	019*d*	13*h*	‼	(dc3)	035*d*	23*h*	#	051*d*	33*h*	3	067*d*	43*h*	C	083*d*	53*h*	S	099*d*	63*h*	c	115*d*	73*h*	s
004*d*	04*h*	♦	(eot)	020*d*	14*h*	¶	(dc4)	036*d*	24*h*	$	052*d*	34*h*	4	068*d*	44*h*	D	084*d*	54*h*	T	100*d*	64*h*	d	116*d*	74*h*	t
005*d*	05*h*	♣	(enq)	021*d*	15*h*	§	(nak)	037*d*	25*h*	%	053*d*	35*h*	5	069*d*	45*h*	E	085*d*	55*h*	U	101*d*	65*h*	e	117*d*	75*h*	u
006*d*	06*h*	♠	(ack)	022*d*	16*h*	▬	(syn)	038*d*	26*h*	&	054*d*	36*h*	6	070*d*	46*h*	F	086*d*	56*h*	V	102*d*	66*h*	f	118*d*	76*h*	v
007*d*	07*h*	•	(bel)	023*d*	17*h*	↨	(etb)	039*d*	27*h*	'	055*d*	37*h*	7	071*d*	47*h*	G	087*d*	57*h*	W	103*d*	67*h*	g	119*d*	77*h*	w
008*d*	08*h*	◘	(bs)	024*d*	18*h*	↑	(can)	040*d*	28*h*	(	056*d*	38*h*	8	072*d*	48*h*	H	088*d*	58*h*	X	104*d*	68*h*	h	120*d*	78*h*	x
009*d*	09*h*		(tab)	025*d*	19*h*	↓	(em)	041*d*	29*h*	)	057*d*	39*h*	9	073*d*	49*h*	I	089*d*	59*h*	Y	105*d*	69*h*	i	121*d*	79*h*	y
010*d*	0A*h*	◙	(lf)	026*d*	1A*h*	→	(eof)	042*d*	2A*h*	*	058*d*	3A*h*	:	074*d*	4A*h*	J	090*d*	5A*h*	Z	106*d*	6A*h*	j	122*d*	7A*h*	z
011*d*	0B*h*	♂	(vt)	027*d*	1B*h*	←	(esc)	043*d*	2B*h*	+	059*d*	3B*h*	;	075*d*	4B*h*	K	091*d*	5B*h*	[	107*d*	6B*h*	k	123*d*	7B*h*	{
012*d*	0C*h*		(np)	028*d*	1C*h*	∟	(fs)	044*d*	2C*h*	,	060*d*	3C*h*	<	076*d*	4C*h*	L	092*d*	5C*h*	\	108*d*	6C*h*	l	124*d*	7C*h*	|
013*d*	0D*h*	♪	(cr)	029*d*	1D*h*	↔	(gs)	045*d*	2D*h*	-	061*d*	3D*h*	=	077*d*	4D*h*	M	093*d*	5D*h*	]	109*d*	6D*h*	m	125*d*	7D*h*	}
014*d*	0E*h*	♫	(so)	030*d*	1E*h*	▲	(rs)	046*d*	2E*h*	.	062*d*	3E*h*	>	078*d*	4E*h*	N	094*d*	5E*h*	^	110*d*	6E*h*	n	126*d*	7E*h*	~
015*d*	0F*h*	☼	(si)	031*d*	1F*h*	▼	(us)	047*d*	2F*h*	/	063*d*	3F*h*	?	079*d*	4F*h*	O	095*d*	5F*h*	_	111*d*	6F*h*	o	127*d*	7F*h*	⌂

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

128*d*	80*h*	€		144*d*	90*h*			160*d*	A0*h*		176*d*	B0*h*	°	192*d*	C0*h*	À	208*d*	D0*h*	Đ	224*d*	E0*h*	à	240*d*	F0*h*	ð
129*d*	81*h*			145*d*	91*h*		‘	161*d*	A1*h*	¡	177*d*	B1*h*	±	193*d*	C1*h*	Á	209*d*	D1*h*	Ñ	225*d*	E1*h*	á	241*d*	F1*h*	ñ
130*d*	82*h*	‚		146*d*	92*h*		’	162*d*	A2*h*	¢	178*d*	B2*h*	²	194*d*	C2*h*	Â	210*d*	D2*h*	Ò	226*d*	E2*h*	â	242*d*	F2*h*	ò
131*d*	83*h*	*ƒ*		147*d*	93*h*		“	163*d*	A3*h*	£	179*d*	B3*h*	³	195*d*	C3*h*	Ã	211*d*	D3*h*	Ó	227*d*	E3*h*	ã	243*d*	F3*h*	ó
132*d*	84*h*	„		148*d*	94*h*		”	164*d*	A4*h*	¤	180*d*	B4*h*	´	196*d*	C4*h*	Ä	212*d*	D4*h*	Ô	228*d*	E4*h*	ä	244*d*	F4*h*	ô
133*d*	85*h*	…		149*d*	95*h*		•	165*d*	A5*h*	¥	181*d*	B5*h*	µ	197*d*	C5*h*	Å	213*d*	D5*h*	Õ	229*d*	E5*h*	å	245*d*	F5*h*	õ
134*d*	86*h*	†		150*d*	96*h*		–	166*d*	A6*h*	¦	182*d*	B6*h*	¶	198*d*	C6*h*	Æ	214*d*	D6*h*	Ö	230*d*	E6*h*	æ	246*d*	F6*h*	ö
135*d*	87*h*	‡		151*d*	97*h*		—	167*d*	A7*h*	§	183*d*	B7*h*	·	199*d*	C7*h*	Ç	215*d*	D7*h*	×	231*d*	E7*h*	ç	247*d*	F7*h*	÷
136*d*	88*h*	ˆ		152*d*	98*h*		˜	168*d*	A8*h*	¨	184*d*	B8*h*	¸	200*d*	C8*h*	È	216*d*	D8*h*	Ø	232*d*	E8*h*	è	248*d*	F8*h*	ø
137*d*	89*h*	‰		153*d*	99*h*	™		169*d*	A9*h*	©	185*d*	B9*h*	¹	201*d*	C9*h*	É	217*d*	D9*h*	Ù	233*d*	E9*h*	é	249*d*	F9*h*	ù
138*d*	8A*h*	Š		154*d*	9A*h*	š		170*d*	AA*h*	ª	186*d*	BA*h*	º	202*d*	CA*h*	Ê	218*d*	DA*h*	Ú	234*d*	EA*h*	ê	250*d*	FA*h*	ú
139*d*	8B*h*	‹		155*d*	9B*h*	›		171*d*	AB*h*	«	187*d*	BB*h*	»	203*d*	CB*h*	Ë	219*d*	DB*h*	Û	235*d*	EB*h*	ë	251*d*	FB*h*	û
140*d*	8C*h*	Œ		156*d*	9C*h*	œ		172*d*	AC*h*	¬	188*d*	BC*h*	¼	204*d*	CC*h*	Ì	220*d*	DC*h*	Ü	236*d*	EC*h*	ì	252*d*	FC*h*	ü
141*d*	8D*h*			157*d*	9D*h*			173*d*	AD*h*		189*d*	BD*h*	½	205*d*	CD*h*	Í	221*d*	DD*h*	Ý	237*d*	ED*h*	í	253*d*	FD*h*	ý
142*d*	8E*h*	Ž		158*d*	9E*h*	ž		174*d*	AE*h*	®	190*d*	BE*h*	¾	206*d*	CE*h*	Î	222*d*	DE*h*	Þ	238*d*	EE*h*	î	254*d*	FE*h*	þ
143*d*	8F*h*			159*d*	9F*h*	Ÿ		175*d*	AF*h*	¯	191*d*	BF*h*	¿	207*d*	CF*h*	Ï	223*d*	DF*h*	ß	239*d*	EF*h*	ï	255*d*	FF*h*	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