Práctica 3
Aritmética de enteros (2) y pseudoinstrucciones

Objetivos

- Entender el significado del desbordamiento en las instrucciones aritméticas.
- Conocer algunas de las pseudoinstrucciones del MIPS
- Aprender a utilizar funciones del sistema para la entrada y salida de caracteres.
- Introducir el concepto de etiqueta.
- Introducir la ruptura del flujo secuencial del programa.

Material

Simulador MARS y un código fuente de partida

Desarrollo de la práctica

1. Desbordamiento e instrucciones *unsigned*.

En un computador la información está almacenada como cadenas de unos y ceros y puede tener muchos significados distintos, pueden representar instrucciones o datos, pueden representar números enteros, números en coma flotante, etc. La forma que tiene de entender el procesador a que se trata viene determinada por el método de manipulación. Un número entero puede estar representado en binario con signo, normalmente en complemento a 2, o sin signo directamente en binario natural. La máquina sólo lo sabrá por la instrucción que se utiliza para acceder al valor. En MIPS hay instrucciones aritméticas distintas para operar con valores con signo y sin signo (por ejemplo `addi` y `addiu`), la "u" al final de la instrucción significa unsigned, es decir, supone que la cadena de bits a la que está accediendo representa un número sin signo. La diferencia principal entre estas dos instrucciones es que la excepción de desbordamiento (u overflow) sólo aparece con las instrucciones con signo. Esto significa que no hay ninguna limitación para utilizar cualquier instrucción con cualquier valor, que el programador es libre de utilizar una instrucción sin signo con un número en complemento a 2, ahora bien, en ese caso el programador tiene que ser el responsable de detectar el desbordamiento. A veces puede interesar por el menor coste en tiempo de ejecución que implica al procesador el ahorro de la tarea de detección.

Experimenta:

- ¿Cuál es el mayor valor positivo que puede contener un registro del MIPS?. Puedes decirlo en hexadecimal
- Escribe un programa de una instrucción que sume *$t0=$t1+$t2* y ensámblalo.
- Poned en la ventana *Registers* los siguientes valores:
  $t1=0x7FFFFFFF
  $t2=0x00000001
- Ensambla y ejecuta el código fijándote en la ventana *Mars Missatges* y explica lo que ha pasado.
- Sustituye `add` por `addu` y vuelve a hacer la prueba. Explica el resultado.
- Diseña un ejemplo para probar la diferencia de `sub` y `subu`.

2. Pseudoinstrucciones

Si tenéis intención de inicializar un registro $t0=10, hasta ahora habéis escrito `addi $t0,$0,10` o `ori $t0,$0,10`. Ahora estudiaremos una manera alternativa de hacerlo.
El ensamblador tiene lo que se denomina *pseudoinstrucciones*, que son pseudooperadores que no existen en el conjunto real de instrucciones del MIPS pero nos permiten escribir de una manera distinta o de una manera más fácil ciertas operaciones muy comunes. Lo que hace el ensamblador es traducir las pseudoinstrucciones en una o varias instrucciones reales.

Por ejemplo la pseudoinstrucción ***li* (*load immediate*)** sirve para cargar un dato en un registro. Si en lugar de `addi $t0,$0,10` escribís `li $t0,10`, el ensamblador traducirá esta pseudoinstrucción por `addi $t0,$0,10`.

➔ De ahora en adelante cada vez que queráis que un programa en ensamblador cargue una constante en un registro utilizad la pseudoinstrucción *li*. Es más general y más legible que *addiu*.

Si queréis copiar el contenido de un registro a otro, hasta ahora habéis escrito, por ejemplo, `addi $t1,$t0,0`. Ahora podéis escribir la pseudoinstrucción `move $t1,$t0`.

Si queréis invertir cada uno de los bits del registro *$t2* y guardarlo en el registro *$t1*, hasta ahora habríais escrito `nor $t1, $t2, $zero`. Ahora podéis escribir `Not $t1, $t2`.

Si queréis hacer una resta con el valor inmediato k, hasta ahora habríais escrito, por ejemplo, `addi $t1,$t0,-k`. Ahora podéis escribir la pseudoinstrucción
`subi $t1, $t0, k`.

- Comprueba como se traducen las siguientes pseudoinstrucciones al ensamblar el programa. Mira la columna de la izquierda del código nombrado *Basic*.

```
#Traducción de pseudoinstrucción por el ensamblador

li $t1, 4
move $t2, $t1
not $t3, $t2
subi $t4,$t1,1
```

3. Constantes grandes

Recordáis que el formato de instrucciones del MIPS tiene 32 bits y que las instrucciones que permiten operar con datos inmediatos (las de formado tipos I) sólo tienen 16 bits para alojar el dato inmediato. Esto quiere decir que si necesitamos cargar un registro con un dato que ocupa más de 16 bits no podemos hacerlo simplemente con una instrucción con formado tipo I. Este proceso requerirá una serie de pasos.

- ¿Qué hace la instrucción `lui`? Buscad en la web o en la ayuda de las instrucciones básicas del MARS.
- ¿Cómo haríais $t0=0x10000000?
- ¿Cómo haríais $t1=0x10001000?
- Prueba este código

```
li $t0,0x10000000
li $t1,0x10000100
```

- ¿Con qué instrucciones tiene que traducir el ensamblador la pseudoinstrucción `li`?
- Escribid el código (3 líneas entre instrucciones y pseudoinstrucciones) que hace estas acciones:
  $t0=5
  $t1=$t0+10
  $t2=$t1-30
- Ensamblad y ejecutad el programa. Comprobad que el resultado final es ($t0=5, $t1=15, $t2=-15) es correcto.

4. Entrada/salida de caracteres

Hasta ahora todos los datos que habéis utilizado han sido valores enteros. Los computadores también permiten manipular datos alfanuméricos fácilmente legibles por los usuarios. El sistema que permite la codificación de los caracteres en números binarios se denomina ASCII (American Standard Code for Information Interchange). En ASCII todos los caracteres están representados por un número comprendido entre 1 y 127 almacenado en 8 bits. Las codificaciones ASCII se muestran en la siguiente tabla:

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

Utilizando esta tabla se pueden codificar cadenas de caracters. Por ejemplo, “Hola” en caracteres ASCII se codifica con el número hexadecimal 0x486F6C61 (H mayúscula= 0x48, o=0x6F, l=6C y a=61).
También se pueden representar los números en ASCII. Por ejemplo, el número 12 se 0xC=1100_2. Si se representa como una cadena de caracteres sería 0x3132.

Con la instrucción `syscall` podemos tener entrada de caracteres por teclado y salida de caracteres por la consola con las **funciones 11** (`print character`) y **12** (`read character`).

- Probad el siguiente código que escribe un carácter leído:

```
# Programa ECO

li $v0,12           #Función 12. Read character
syscall             #Carácter leído en $v0


move $a0,$v0        #Carácter a escribir en $a0
li $v0,11           #Función 11. Print character
syscall

li $v0, 10          #Función 10. Acaba programa
syscall
```

Podéis cargar directamente un carácter en un registro poniéndolo entre comillas simples, por ejemplo `li $a0,'x'` o con su valor ASCII `li $a0,0x78`.

- El carácter ‘\n‘ es el de nueva línea y provoca que la salida por consola desde el programa comience en una línea nueva. Modifica el código anterior para que al leer un carácter muestre la salida en una línea diferente.

- Con el fin de facilitar la claridad de ejecución, modifica el código anterior para que muestre '>' o '?' antes de leer el carácter como una manera de solicitarlo.

Ejercicios

- Escribe el código que imprime por consola el valor ASCII en hexadecimal del carácter leído.
- Escribe el código que imprime por consola el carácter correspondiente al valor ASCII leído en decimal desde el teclado (ayudado por la tabla ASCII anterior).

4. Etiquetas

En el ensamblador del MIPS una cadena de caracteres seguida ":" es una etiqueta. Una etiqueta es un identificador que representa un valor y puede ser utilizado en cualquier línea de código. Las etiquetas no son equivalentes a variables, las etiquetas simplemente son marcas en el programa. El valor representado por una etiqueta puede ser una dirección de memoria y la podemos utilizar en cualquier lugar de nuestro programa.

En el ejemplo siguiente podéis observar dos etiquetas `eti1` y `eti2`, cada una de ellas toma el valor de la dirección en la que se encuentra la instrucción que señalan. Así `eti1` toma el valor de la dirección 0x00400000 y `eti2` el de 0x00400008.

```
            # Pruebas etiquetas

            .text
eti1:       addi $v0,$0,5
            syscall

eti2:       addi $a0,$v0,15
            addi $v0,$0,1
            syscall

            li $a0, '\n'
            li $v0,11
            syscall

            la $t1, eti1        #Carga en $t1 la dirección de
eti1
            La $t2, eti2        #Carga en $t2 la dirección de
eti2

            move $a0,$t1
            li $v0,34
            syscall
            li $a0, '\n'
            li $v0,11
            syscall
            move $a0,$t2
            li $v0,34
            syscall

            li $v0,10           #Función 10 Exit
            syscall
```

La pseudoinstrucción `la` (`load address`) sirve para cargar un dato de 32 bits que especifica una dirección en un registro. En el ejemplo `la $t1, eit1` indica que se cargue en lo registro *$t1* el valor de la dirección de la etiqueta `eti1`.

- Ejecuta el código y comprueba el valor correcto de las etiquetas
- ¿Qué hace el código?
- ¿En qué instrucciones básicas se ha traducido la pseudoinstrucción *la*?

Una etiqueta es un identificador que también puede representar un valor numérico cualquiera. La etiqueta recibirá su valor en el código fuente, para lo cual se empleará la directiva .eqv al principio de una línea del código fuente. Esta etiqueta se podrá utilizar en cualquier lugar del programa como si fuera el valor que representa.

```
            .eqv eti1 7
            li $s0,eti1
eti2:       la $s1,eti2
```

- Observa el significado de las diferentes pseudoinstrucciones que se utilizan con las diferentes etiquetas

5. El primer bucle

La instrucción `j (jump)` permite en el programa romper la secuencia normal del programa y saltar a otras partes del programa modificando el contenido del registro *Contador del Programa* (PC). Lo que hace esta instrucción es cargar en el PC una nueva dirección de tal manera que en el siguiente ciclo de instrucción se ejecutará la instrucción que se encuentra en el nuevo PC. La manera más sencilla de manipular direcciones es utilizando etiquetas.

- Probad el siguiente código ejecutándolo paso a paso y observad en la ventana *registers* los valores que va tomando el registro PC.

```
            # Pruebas con el PC

            .text
eti1:       addi $v0,$0,5
            syscall             #Lee un valor

eti2:       addi $a0,$v0,15
            addi $v0,$0,1
            syscall             #Escribe un valor
            li $a0, '\n'
            li $v0,11
            syscall

            j eti1              #Salta a eti1
```

La instrucción *j* sólo tiene como parámetro la dirección de la instrucción a la que tiene que saltar. La instrucción *j* utiliza 26 bits y el ensamblador los completa hasta conseguir los 32 bits necesarios para la dirección.

La forma general del formato tipo J es el siguiente:

- Observa el código anterior *Pruebas con el PC* una vez se ha ensamblado. ¿Cuál es el código de operación de la instrucción *j*?

Ejercicios a entregar

- Escribe un programa que lea del teclado una letra en mayúscula y la escríba en minúscula en la consola.
- Itera el código que acabas de escribir.
- Convierte caracteres numéricos. Escribe el código que lea del teclado un carácter numérico (del '0' al '9') y lo convierta en un valor numérico (del 0 al 9) y lo escriba por pantalla. Itera el código.

Resumen

- En el ensamblador se proporcionan pseudoinstrucciones para facilitarnos la programación y hacer más legible el código resultante.
- También podemos utilizar caracteres con la instrucción `syscall`.
- Las etiquetas pueden contener direcciones o valores numéricos, depende de la instrucción que los utiliza.
- Con la instrucción *j* podemos saltar dentro del programa.