Pràctica 3
Aritmètica d'enters (2) i pseudoinstruccions

3.1 Objectius

- Entendre el significat del desbordament en les instruccions aritmètiques.
- Conèixer algunes de les pseudoinstruccions del MIPS
- Aprendre a utilitzar funcions del sistema per a l'entrada i eixida de caràcters.
- Introduir el concepte d'etiqueta.
- Introduir la ruptura del flux seqüencial del programa.

3.2 Material

Simulador MARS i un codi font de partida

3.3 Desenvolupament pràctic

3.3.1 Desbordament i instruccions *unsigned*

En un computador, tota la informació està emmagatzemada com a cadenes d'uns i zeros i pot tenir molts significats diferents. Aquestes cadenes poden representar instruccions o dades, poden representar nombres enters, nombres en coma flotant, caràcters, etc. La forma que té el processador d'entendre de què es tracta ve determinada pel mètode de manipulació de la informació. Un nombre pot estar representat en binari amb signe, normalment en complement a 2, o sense signe directament en binari natural. La màquina sols sabrà de quin tipus es tracta per la instrucció que s'utilitze a l'hora de manipular el valor. El MIPS disposa d'instruccions aritmètiques diferents per a operar amb valors amb signe i sense signe (per exemple les instruccions *addi* i *addiu* ja estudiades). La *u* que apareix al final de la instrucció significa

'unsigned', es a dir, suposa que la cadena de bits a la qual està accedint representa un nombre sense signe. La diferència principal entre aquestes dues instruccions és que l'excepció de desbordament (o *overflow*) sols apareix amb les instruccions amb signe. Això significa que no hi ha cap limitació per a utilitzar qualsevol instrucció sense signe (*unsigned*) amb qualsevol valor. Així doncs, el programador és lliure d'utilitzar una instrucció sense signe amb un nombre en representat en complement a 2; ara bé, en aquest cas, el programador ha de ser el responsable de detectar el desbordament, la màquina no ho farà. De vegades pot interessar utilitzar aquesta opció pel menor cost en temps d'execució que suposa al processador l'estalvi de la tasca de detecció del desbordament.

3.3.1.1 Activitat 1

Realitzeu les accions següents per a experimentar amb les instruccions amb signe i sense signe:

- Quin és el major valor positiu que pot contenir un registre del MIPS. N'hi ha prou que ho digueu en hexadecimal.
- Escriviu un programa d'una instrucció que sume *$t0=$t1+$t2* i assemblar-lo. Utilitzeu la instrucció add.
- Poseu en la finestra *Registers* els valors següents en els registres:

  $t1=0x7FFFFFFF
  $t2=0x00000001

- Assembleu i executeu el codi fixant-vos en la finestra *Mars Missatges* i expliqueu què ha passat.
- Substituïu *add* per *addu* i torneu a fer la prova. Expliqueu el resultat.

Qüestió 1

- Dissenyeu un exemple per a provar la diferència de *sub* i *subu*. Assembleu-lo i executeu-lo.

3.3.2 Pseudoinstruccions

Si teniu intenció de inicialitzar un registre, per exemple $t0=10, fins ara heu escrit `addi $t0,$0,10` o `ori $t0,$0,10`. Ara estudiarem una manera alternativa de posar un valor a un registre.

L'assemblador disposa del que s'anomenen pseudoinstruccions, que són pseudooperadors que no existeixen en el conjunt real d'instruccions del MIPS, però que ens permeten escriure d'una manera diferent o d'una manera més fàcil certes operacions molt comunes. El que fa l'assemblador és traduir les pseudoinstruccions en una o més instruccions reals.

Per exemple, la pseudoinstrucció *li* (*load immediate*) serveix per a carregar una dada en un registre. Si per comptes de `addi $t0,$0,10` escriviu `li $t0,10`, l'assemblador traduirà aquesta pseudoinstrucció per `addiu $t0,$0,10`.

En endavant, cada vegada que vulgueu que un programa en assemblador carregue una constant en un registre, feu servir la pseudoinstrucció *li*. És més general i més llegible que *addiu*.

Si voleu copiar el contingut d'un registre en un altre, fins ara heu escrit, per exemple, `addi $t1,$t0,0`. Ara podeu escriure-hi la pseudoinstrucció `move $t1,$t0`.

Si voleu invertir cadascun dels bits del registre $t2 i guardar-lo en el registre $t1, fins ara hauríeu hagut d'escriure `nor $t1, $t2, $zero`. Ara podeu utilitzar la pseudoinstrucció *not* que resulta més intuïtiva i escriure `not $t1, $t2`.

Si voleu fer una resta amb un valor immediat k fins ara hauríeu escrit, per exemple, `addi $t1,$t0,-k`. Ara podeu fer ús de la pseudoinstrucció *subi* i escriure `subi $t1, $t0, k`.

3.3.2.1 Activitat 2

- Comproveu com es tradueixen les pseudoinstrucions quan assembleu el programa següent. Mireu la columna de l'esquerra del codi anomenada *Basic* en la finestra *Text Segment* una vegada assemblat el programa.

```
####################################
#                                  #
#     Codi de l'activitat 2        #
# Traducció de pseudoinstruccions  #
# per l'assemblador                #
#                                  #
####################################

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

3.3.3 Constants grans

Recordeu que el format d'instruccions del MIPS té 32 bits i que les instruccions que permeten operar amb dades immediates (les de format tipus I) sols tenen 16 bits per a allotjar la dada immediata. Això vol dir que, si necessitem carregar un registre amb una dada que ocupe més de 16 bits, no ho podem fer simplement amb una instrucció amb format tipus I. Aquest procés requerirà una sèrie de passos.

3.3.3.1 Activitat 3

- Què fa la instrucció *lui*? Busqueu en la web o en l'ajuda de les instruccions bàsiques del MARS.
- Com faríeu $t0=0x10000000?
- Com faríeu $t1=0x10001000?
- Proveu aquest codi

```
#################################
#                               #
#     Codi de l'activitat 3     #
#     Carregar dades            #
#                               #
#################################


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

- Amb quines instruccions ha traduït l'assemblador la pseudoinstrucció *li*?

Qüestió 2

- Escriviu el codi (3 línies entre instruccions i pseudoinstruccions) que fa aquestes accions:
  $t0=5
  $t1=$t0+10
  $t2=$t1-30
- Assembleu i executeu el programa. Comproveu que el resultat final ($t0=5, $t1=15, $t2=-30) és correcte.

3.3.4 Entrada/eixida de caràcters

Fins ara totes les dades que heu utilitzat han sigut valors enters. Els computadors també permeten manipular dades alfanumèriques fàcilment llegibles pels usuaris. El sistema que permet la codificació

dels caràcters en nombres binaris s'anomena ASCII (*American Standard Code for Information Interchange*). En ASCII tots els caràcters estan representat per un nombre comprès entre 1 i 127 emmagatzemat en 8 bits. Les codificacions ASCII es mostren en la figura 1:

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

Figura 1. Codificacions ASCII

Utilitzant la taula de la figura 1 es poden codificar cadenes de caracters. Per exemple, la cadena "Hola" en caràcters ASCII es codifica amb el nombre hexadcimal 0x486F6C61 (**H** majúscula = 0x48, **o**=0x6F, **l**=0x6C i **a**=0x61). Fixeu-vos que independentment de la gràndaria, per a indicar que un nombre està representat en hexadecimal el comencem amb "0x".

També els nombres es poden representar en com a caràcters en ASCII. Per exemple, el nombre 12 es correspon a $0xC=1100_2$ en binari natural. Si el mateix valor 12 es representa com una cadena de caràcters es correspondria a 0x3132 (**1**=0x31 i **2**=0x32) o escrit en binari natural com a 0011000100110010_2.

Igual que amb el valors enters, el MIPS permet l'entrada i l'eixida de caràcters. Amb la instrucció syscall podem llegir caràcters introduïts per teclat i escriure caràcters per la consola usant les **funcions 11** (`print character`) i **12** (`read character`).

3.3.4.1 Activitat 4

- Proveu el codi següent que escriu en la consola el caràcter llegit del teclat:

```
#################################
#                               #
#     Codi de l'activitat 4     #
#     Programa ECHO             #
#                               #
#################################

li $v0,12      # Funció 12. Read character
syscall        # Caràcter llegit en $v0

move $a0,$v0   # Caràcter a escriure en $a0
li $v0,11      # Funció 11. Print character
syscall

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

Podeu introduir directament un caràcter en un registre posant-lo entre cometes simples i fent servir la pseudoinstrucció *li* com, per exemple, `li $a0,'x'` o també utilitzar el seu valor ASCII, per exemple `li $a0,0x78`.

Qüestió 3

- El caràcter '\n ' és el de salt de línia i provoca que l'eixida per consola des del programa comence en la línia següent. Modifiqueu el codi de l'activitat 4 perquè quan llegim un caràcter per teclat mostre l'eixida en una línia diferent.

Qüestió 4

- Amb la finalitat de facilitar la claredat en l'execució dels programes, modifiqueu el codi de la qüestió 3 perquè mostre '>' o '?' abans de llegir el caràcter del teclat com una manera de demanar-lo.

Qüestió 5

- Escriviu i executeu el codi que imprimeix per consola el valor ASCII en hexadecimal del caràcter llegit.

Qüestió 6

- Escriviu el codi que imprimeix per consola el caràcter corresponent al valor ASCII llegit en decimal des del teclat (recolzat amb la taula ASCII anterior).

3.3.5 Etiquetes

En l'assemblador del MIPS, una cadena de caràcters seguida de ":" és una etiqueta. Una etiqueta és un identificador que representa un valor i pot ser utilitzat en qualsevol línia de codi. Les etiquetes no són equivalents a variables, les etiquetes simplement són marques en el programa. El valor representat per una etiqueta pot ser una adreça de memòria i la podem utilitzar en qualsevol lloc del nostre programa.

En el codi de l'exemple següent podeu observar dues etiquetes *eti1* i *eti2*, cadascuna de les quals pren el valor de l'adreça en què es troba emmagatzemada la instrucció que assenyalen. Així *eti1* pren el valor de l'adreça 0x00400000 que és on es troba la instrucció `addi $v0,$0,5` i *eti2* el de 0x00400008 que és on està `addi $a0,$v0,15`.

```
          ##################################
          #                                #
          #     Codi de l'activitat 5      #
          #     Proves d'etiquetes         #
          #                                #
          ##################################

          .text
eti1:     addi $v0,$0,5     # Funció 5. Read integer
          syscall

eti2:     addi $a0,$v0,15
          addi $v0,$0,1     # Funció 1. Print integer
          syscall

          li $a0, '\n'
          li $v0,11         # Funció 11. Print character
          syscall

          la $t1, eti1   # Carrega en $t1 l'adreça d'eti1
          La $t2, eti2   # Carrega en $t2 l'adreça d'eti2

          move $a0,$t1
          li $v0,34      # Funció 34.Print int en hex
          syscall
          li $a0, '\n'
          li $v0,11      # Funció 11. Print character
          syscall
          move $a0,$t2
          li $v0,34      # Funció 34.Print int en hex
          syscall

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

La pseudoinstrucció *la* (*load address*) serveix per a carregar una dada de 32 bits que especifica una adreça en un registre. En l'exemple anterior `la $t1, eit1` indica que es carregue en el registre $t1 el valor de l'adreça de l'etiqueta eti1.

3.3.5.1 Activitat 5

- Escriviu el codi anterior i executeu-lo. Comproveu el valor correcte de les etiquetes.
- Què fa el codi?
- Amb quines instruccions bàsiques s'ha traduït la pseudoinstrucció *la*?

Una etiqueta és un identificador que també pot representar un valor numèric qualsevol. L'etiqueta rebrà el seu valor una única vegada en el codi font, per a la qual cosa s'emprarà la directiva *.eqv* al principi d'una línia del codi font. Aquesta etiqueta es podrà utilitzar en qualsevol lloc del programa com si fóra el valor que representa. Per exemple:

```
          ##################################
          #                                #
          #     Codi de l'activitat 6      #
          #     Valors de les etiquetes    #
          #                                #
          ##################################

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

3.3.5.2 Activitat 6

- Observeu el significat de les diferents pseudoinstruccions *li* i *la* que s'utilitzen amb les diferents etiquetes.

3.3.6 El primer bucle

La instrucció *j* (*jump*) permet al processador trencar la seqüència d'execució normal del programa i saltar a altres parts del programa modificant el contingut del registre *Comptador del Programa* (*PC*). El que fa aquesta instrucció és carregar en el PC una nova adreça de tal manera que en el següent cicle d'instrucció s'executarà la instrucció que es trobe en la nova adreça assenyalada pel PC. La manera més senzilla de manipular adreces és utilitzant etiquetes.

3.3.6.1 Activitat 7

- Proveu el següent codi executant-lo pas a pas i observeu en la finestra *registers* els valors que va prenent el registre PC:

```
          ##################################
          #                                #
          #     Codi de l'activitat 7      #
```

```
          #     Proves amb el PC          #
          #                               #
          #################################

          .text
eti1:     addi $v0,$0,5     # Funció 5. Read integer
          syscall           #Llig un valor

eti2:     addi $a0,$v0,15
          addi $v0,$0,1     # Funció 1. Print integer
          syscall           # Escriu un valor
          li $a0, '\n'
          li $v0,11         # Funció 11. Print character
          syscall           # Escriu nova línea

          j eti1            # Salta a eti1
```

La instrucció *j* sols té com a paràmetre l'adreça de la instrucció a la qual ha de saltar, la manera més fàcil d'indicar-ho és mitjançant una etiqueta com heu vist en l'exemple anterior. La instrucció *j* utilitza 26 bits per a indicar l'adreça, però com que en el MIPS totes les adreces són de 32 bits, el valor de l'etiqueta que posem en la instrucció j serà també de 32 bits; per tant, a l'obtenir el codi executable l'assemblador s'encarrega de modificar els bits de l'adreça a fi d'aconseguir els 26 bits necessaris per a codificar la instrucció.

La forma general de codificació del format tipus J que és el que utilitza la instrucció *j* es mostra en la figura 2. Amb els format tipus R, tipus I i tipus J es codifiquen totes les instruccions del repertori del MIPS.

Figura 2. Codificació tipus J

Qüestió 7

- Observeu el codi de l'activitat 7 *Proves amb el PC* una vegada s'ha assemblat. Quin és el codi d'operació de la instrucció *j*?

Qüestió 8

- Escriviu un programa que llig del teclat una lletra en majúscula i l'escriviu en minúscula en la consola.

Qüestió 9

- Itereu el codi que acabeu d'escriure. És a dir, utilitzeu la instrucció *j* perquè s'execute indefinidament.

Qüestió 10

- Convertiu caràcters numèrics. És a dir, escriviu el codi que llig del teclat un caràcter numèric (del '0' al '9'), el converteix en un valor numèric (del 0 al 9) i, finalment, l'escriviu en pantalla. Itereu el codi, és a dir, feu servir la instrucció *j* perquè s'execute indefinidament.

3.4 Resum

- L'assemblador ens proporciona pseudoinstruccions per a facilitar-nos la programació i fer més llegible el codi resultant.
- També podem utilitzar caràcters amb la instrucció syscall.
- Les etiquetes poden contenir adreces o valors numèrics, depèn de la instrucció que els utilitza.
- Amb la instrucció *j* podem saltar dins del programa.