Pràctica 4
Aritmètica d'enters (3) i funcions

4.1 Objectius

- Aprofundir en les instruccions aritmètiques del MIPS.
- Entendre el maneig de les instruccions de desplaçament.
- Entendre les funcions de programa i saber escriure i cridar funcions en MIPS.
- Saber fer una multiplicació utilitzant sumes i desplaçaments.
- Entendre la multiplicació i la divisió enteres.

4.2 Material

Simulador MARS i codis fonts de partida.

4.3 Desenvolupament de la pràctica

4.3.1 Instruccions de desplaçament

Les instruccions de desplaçament permeten moure els bits dins d'un registre. Aquestes instruccions són importants especialment si es treballa a baix nivell, per exemple, quan es programa un driver. Una bona raó per a utilitzar instruccions de desplaçament és la major comprensió de la multiplicació i la divisió per una constant.

Podeu fer desplaçament dels bits a la dreta o a l'esquerra; a més, les instruccions del MIPS permeten especificar el nombre de bits a desplaçar. Els desplaçament lògics omplin els espais amb zeros. Els desplaçament aritmètics omplin els espais deixats en els desplaçament a dreta amb el valor del bit de major ordre (el bit situat a l'esquerra del registre) que és el bit del signe en un enter, és a dir, repliquen el bit de signe tantes vegades com bits a desplaçar.

En un desplaçament circular, el bit que ix per un extrem del registre és el que s'introdueix per l'extrem contrari.

El llenguatge d'assemblador MIPS proporciona, a més, dos pseudoinstruccions per a fer les operacions de rotació a l'esquerra i a la dreta, que són *ror* i *rol* (rotate left/right). Aquests operadors s'implementen amb combinacions d'instruccions de desplaçament i l'operació lògica *or*. El nombre de bits a desplaçar en qualsevol de les instruccions de desplaçament està limitat al rang 0..31.

En la taula següent es mostren les instruccions de desplaçament proporcionades pel MIPS:

Instrucció	**Exemple**	**Significat**	**Comentaris**
sll (shift left logical)	sll Rd, Rt, k	Rd ← Rt << k	Desplaçament lògic de k bits a l'esquerra del valor en Rt
sllv (shift left logical variable)	sllv Rd, Rt, Rs	Rd ← Rt << Rs	Desplaçament lògic a l'esquerra del valor en Rt la quantitat de bits en Rs
srl (shift right logical)	srl Rd, Rt, k	Rd ← Rt >> k	Desplaçament lògic de k bits a la dreta del valor en Rt
srlv (shift right logical variable)	srlv Rd, Rt, Rs	Rd ← Rt >> Rs	Desplaçament lògic a la dreta del valor en Rt la quantitat de bits en Rs
sra (shift right arithmetic)	sra Rd, Rt, k	Rd ← Rt >> k	Desplaçament aritmètic de k bits a la dreta del valor en Rt
srav (shift right arith variable)	srla Rd, Rt, Rs	Rd ← Rt >> Rs	Desplaçament aritmètic a la dreta del valor en Rt la quantitat de bits en Rs
rol (rotate left)	rol Rd, Rt, k	Rd[k..0] ← Rt[31..31-k+1], Rd[31..k] ← Rt[31-k..0],	Pseudoinstrucció. Desplaçament circular a l'esquerra del valor en Rt la quantitat de k bits.
ror (rotate fight)	ror Rd, Rt, k	Rd[31-k..k] ← Rt[31..k], Rd[31..31-k+1]←Rt[k-1..0]	Pseudoinstrucció. Desplaçament circular a la dreta del valor en Rt la quantitat de k bits.

Tabla 1. Instruccions de desplaçament del MIPS

Qüestió 1

- Amb quines instruccions traduirà l'assemblador les pseudoinstruccions *rol* i *ror*? Escriviu un codi senzill de prova dels dos operadors i assembleu-lo per a comprovar-ho.

Qüestió 2

- Les instruccions de desplaçament segueixen el format tipus R. Quina es la codificació en hexadecimal de la instrucció `ssl $t2,$t1, 3`? I de la instrucció `srl $t2,$t1, 7` ?. Quin és el valor de cada camp en el format? Ajudeu-vos assemblant un codi de prova.

Qüestió 3

- Descobriu la paraula amagada. Donat el codi següent, completeu-lo de tal manera que mitjançant instruccions lògiques i de desplaçament pugueu escriure en la consola cadascun dels caràcters que es troben emmagatzemats en cada byte del registre $t1.

```
#################################
#                               #
#     Codi de la qüestió 3      #
#     Paraula amagada           #
#                               #
#################################


li $t1, 1215261793

#.......
# A reomplir per l'alumne
#.......

move $a0, $t2
li $v0, 11
syscall

li $v0, 10
syscall
```

4.3.2 Funcions del programa

Igual que en llenguatges en alt nivell, l'assemblador permet programar subrutines o procediments. Una subrutina o subprograma és un fragment de codi dissenyat per a fer una tasca determinada, la qual pot ser invocada des de qualsevol lloc del programa principal i, fins i tot, des d'una altra subrutina. Aquesta subrutina es pot cridar tantes voltes com es vulga. Les subrutines permeten escriure programes més compactes i estructurats i, per tant, més llegibles pel fet d'evitar repeticions de codi innecessaris i obtenir codi reutilitzable. Les subrutines de vegades reben informació a través d'un conjunt de paràmetres amb els valors dels quals faran les operacions o càlculs necessaris. Si el resultat de les operacions és un valor, es retornarà quan finalitze la subrutina. En aquest cas, la subrutina rep el nom de *funció*. Si la subrutina solament

executa una seqüència d'instruccions sense retornar cap valor, s'anomena *procediment*. Els registres $a0, $a1, $a2, $a3 són els utilitzats per a passar els paràmetres a la subrutina i els registres $v0, $v1 són els utilitzats per a retornar valors d'una funció. Si se'n requereixen més s'utilitzarà la pila, però de moment no s'estudiarà, ja la comentarem més endavant.

Les subrutines han de tenir un punt d'entrada i un punt de retorn al programa que els invoca i s'han d'executar de manera independent al programa principal. El llenguatge d'assemblador del MIPS proporciona dues instruccions per a dissenyar subrutines.

La primera instrucció és la *jal* (*jump and link*). Aquesta instrucció salta a l'adreça on es troba la subrutina (modifica el registre PC) i guarda en el registre $ra l'adreça de retorn al programa que crida. És necessari guardar l'adreça de retorn perquè es pot cridar la mateixa subrutina des de diversos llocs del programa. Aquesta instrucció s'escriu: `jal AdreçaDeLaSubrutina`. La instrucció *jal* segueix el format tipus J estudiat en la pràctica anterior a l'introduir la instrucció *j*.

Quan s'escriu una subrutina en MIPS s'etiqueta la primera instrucció amb el nom que es vol donar a la subrutina. Posteriorment es cridarà la subrutina utilitzant el nom de l'etiqueta la qual especificarà l'adreça on es troba.

La segona instrucció és *jr* (*jump register*) i s'escriu `jr $ra`, és la instrucció de retorn de la subrutina i s'hi col·locarà al final. Aquesta instrucció salta a l'adreça emmagatzemada en $ra (modifica el registre PC).

Ara per ara estudiarem exclusivament funcions full que són aquelles no criden una altra funció. Els procediments imbricats, que són aquells que criden altres subrutines, els estudiarem més endavant.

En la figura 1 podeu observar la semàntica d'una crida a una funció:

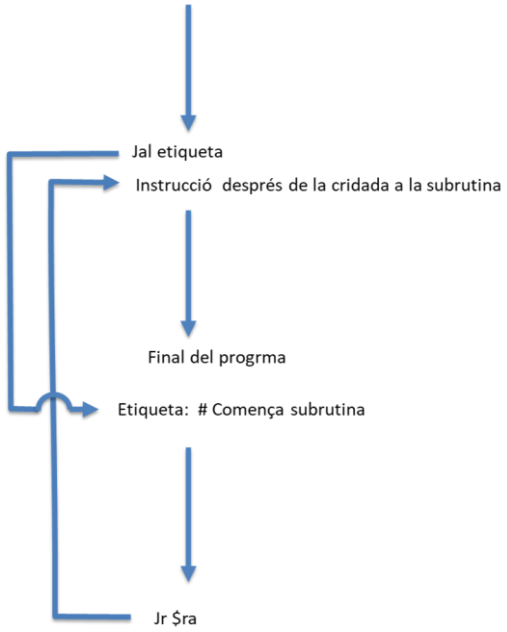

Figura 1. Estructura d'una crida a una funció.

4.3.2.1 Activitat 1

Considereu el següent codi de partida en el qual hi ha una funció que escriu en consola el valor enter passat com argument des del programa principal, a continuació la funció fa un bot de línia.

```
#################################
#                               #
#     Codi de l'activitat 1     #
# Prova de cridada a una funció #
#                               #
#################################

.text

li $a0, '>' #Comença programa principal
li $v0,11
syscall
li $v0,5
syscall        # Llig un enter del teclat

addi $t1,$v0,10
move $a0, $t1     # argument a passar a la
                  # funció en $a0
jal imprim        # crida a la funció imprim

add $t1, $t1,$t1

move $a0, $t1     # argument a passar en $a0
jal imprim        # crida a la funció imprim

li $v0,10      #Acaba el programa
syscall
```

```
          ############################
          #          Funcions        #
          ############################

imprim:   addi $v0,$0,1         # comença la funció
          syscall               # Escriu un valor
          li $a0, '\n'          # salt de línia
          li $v0,11
          syscall
          jr $ra          #Torna al programa principal
```

- Assembleu el programa i observeu en quines adreces es col·loquen les instruccions. Executeu-lo pas a pas. Fixeu-vos en la finestra de registres com van canviant els continguts dels registres PC i $ra.

Qüestió 4

- Escriviu una funció amb instruccions suma que retorne el quàdruple del nombre enter que se li passa. Escriviu el programa principal que lliga el número del teclat i cride la funció quàdruple. Posteriorment el programa principal ha d'escriure el quàdruple en la consola per a la qual cosa aprofiteu la funció *imprim* estudiada en l'activitat 1.

4.3.3 Conveni dels registres MIPS

Una situació que podria ocórrer i cal evitar és que la funció a la qual es crida modifique les dades del programa principal, per exemple, escrivint en registres utilitzats pel programa principal. Una vegada s'ha retornat de la funció, el programa podria necessitar les dades emmagatzemades amb anterioritat en els registres. Per aquesta raó, els dissenyadors del MIPS van establir uns convenis que els programadors haurien de seguir per a evitar esborrar dades erròniament. Les regles afecten els registres $s0,..,$s7 i $t0,...,$t7:

- El programa que crida ha de suposar que la funció cridada pot modificar els registres temporals $t0,...,$t7. A més, el programa que crida ha de saber que les funcions utilitzen els registres $v0 i $v1 per a retornar valors. Per tant, és responsabilitat del programa que crida emmagatzemar en un altre lloc els valors dels registres temporals i de $v0 i $v1 que vulga preservar en la crida a la funció.

- El programa que crida ha d'assumir que els valors que es troben en el registres $s0,..,$s7 no seran modificats per la funció cridada. Això significa que si la funció necessita utilitzar-ne algun haurà de preservar el seu valor abans d'usar-

los i restaurar el valor original abans de retornar al programa que l'ha cridat. El mateix ocorre amb els registres de pas de paràmetres $a0,...,$a3, el programa que crida suposa que la funció no modificarà el seu contingut.

Com a resum, en la taula següent es mostren els convenis i com cal utilitzar els registres implicats en les crides a funcions:

Registre	**Ús**
$s0,..,$s7	Utilitzats pel programa principal. Preservats en les crides.
$t0,...,$t7	Utilitzats en les funcions. No preservat en les crides.
$a0,...,$a3	Pas de paràmetres a les funcions.
$v0, $v1	Valor de retorn de les funcions.

Taula 2. Conveni d'utilització dels registres en MIPS-.

Qüestió 5

- Observeu l'últim codi escrit de la qüestió 4, s'ajusta al conveni MIPS d'utilització de registres? Què hauríeu de modificar? Reescriviu el codi perquè s'ajuste al conveni. D'ara endavant feu servir el conveni d'utilització de registres.

4.3.4 Multiplicació per desplaçaments i sumes

Una bona raó per a utilitzar instruccions de desplaçament és la major comprensió de la multiplicació i la divisió per una constant.

Les instruccions de desplaçament a l'esquerra ens serveixen per a fer multiplicacions per potencies naturals de 2 i les instruccions de desplaçament a la dreta per a fer divisions per potencies naturals de 2. Per exemple, la instrucció `sll $s1, $s1, 1` provoca un desplaçament lògic d'un bit a l'esquerra del registre $s1 amb el resultat de que el seu valor inicial s'haurà multiplicat per dos (2^1). De la mateixa manera, la instrucció `sra $s1, $s1, 2` provoca un desplaçament aritmètic de dos bits a la dreta del registre $s1 amb el resultat que el seu valor inicial s'haurà dividit per quatre (2^2). Les instruccions de desplaçament a l'esquerra utilitzades juntament amb instruccions de sumes ens permeten fer una multiplicació per qualsevol constant.

4.3.4.1 Activitat 3

- Observeu el següent codi de partida en el que es dóna una funció que fa la multiplicació de un enter per 4 utilitzant instruccions de desplaçament. El programa demana l'enter per teclat i escriu la solució en consola.

```
        ##################################
        #                                #
        #     Codi de l'activitat 3      #
        #      Multiplicació per 4       #
        #                                #
        ##################################


        # Codi de partida de l'activitat 3
        #
        # Multiplicació per 4
        #

        .text
        li $a0, '>'    #Comença demanant un enter
        li $v0,11
        syscall
        li $v0,5
        syscall        #Llig un enter

        move $a0, $v0     #paràmetre a passar en $a0
        jal mult4         #cridem la funció mult4

        move $a0, $v0     #paràmetre a passar en $a0
        jal imprim        #cridem la funció imprim

        li $v0,10         #Acaba el programa
        syscall

        ########################
        #                      #
        #        Funcions      #
        #                      #
        ########################

imprim: addi $v0,$0,1     #funció imprim
        syscall           #Escriu el valor en $a0
        li $a0, '\n'      #Salt de línia
        li $v0,11
        syscall
        jr $ra            #Torna al programa principal

mult4:  sll $v0, $a0, 2   #Funció multiplica per 4
        jr $ra
```

- Assembleu i proveu el programa amb diversos valors d'enters.

Qüestió 6

- Modifiqueu el codi de l'activitat 3 perquè ara hi haja una funció mult5 que multiplique per 5 i mostreu el resultat per consola. (Fixeu-vos que al valor obtingut en la multiplicació per 4 haureu de sumar-li una vegada el valor original, x*5=x*4+x). Comproveu que el resultat és correcte.

Qüestió 7

- Modifiqueu el codi de l'activitat 3 per a tenir ara una nova funció mult10 que multiplique per 10. Comproveu que el resultat és correcte.

Acabeu de veure que la multiplicació per un nombre que és potencia de 2 es pot fer senzillament utilitzant l'operador de desplaçament lògic a l'esquerra. La multiplicació per un nombre que no és potencia entera de 2 també resulta senzilla si combinem el desplaçament lògic a l'esquerra amb instruccions de suma. D'aquesta manera podem multiplicar qualsevol nombre per qualsevol constant. El que cal tenir en compte és el desglossament de la constant a multiplicar en potències de 2. Per exemple, el valor 11 s'escriu com $11= 2^3+2^1+2^0$. Una multiplicació d'un valor N per la constant 11 serà $Nx11=Nx2^3+Nx2^1+N$, és a dir, es traduirà en 2 operacions de desplaçaments i dues sumes, en què els exponents indiquen els bits a desplaçar.

Qüestió 8

- Escriviu una funció que multiplique per 60. Escriviu el programa principal que lliga una quantitat de minuts i retorne per consola la quantitat en segons.

Qüestió 9

- Modifiqueu el codi de la qüestió 7 de tal manera que ara el que llija siga una quantitat en hores i mostre per consola la quantitat en segons.

4.3.5 Multiplicació per desplaçaments, sumes i restes

La idea que hem utilitzat fins ara de realitzar la multiplicació per una constant mitjançant instruccions de sumes i desplaçaments la podem estendre a la multiplicació per una constant segons l'algorisme de Booth que ens permet multiplicar tant per valors positius com per valors

negatius. Ara, una multiplicació per una constant constarà de instruccions de desplaçaments, de suma i de resta.

Per exemple, el valor 11 en binari utilitzant 8 bits s'escriu com $11_{10}=00001011_2$. Quan el recodifiquem segons l'algorisme de Booth amb els valors 1, 0 i -1 queda com $11_{10}=000+1-1+10-1_2$. A l'hora d'aplicar l'algorisme el '+' significa una suma i el '–' una resta. Quan l'expressem el valor 11 en potències de 2 recodificat per Booth es transforma en $11_{10}= 2^4-2^3+2^2-2^0$. En aquest cas, la multiplicació per la constant 11 segons Booth es transforma en 3 instruccions de desplaçament, 2 de suma i 2 de resta. Veiem que en aquest cas concret han augmentat el nombre d'instruccions; en altres casos, la multiplicació se simplificarà quan utilitzem l'algorisme de Booth, tot dependrà de l'estructura de bits del valor a recodificar.

Qüestió 10

- Escriviu la funció que multiplique per la constant 11 segons l'algoritme de Booth i comproveu que el resultat es correcte.

4.3.6 Multiplicació i divisió general

En el cas que es requerisca d'operacions de multiplicació o divisió de dos valors enters qualsevol, el llenguatge assemblador del MIPS proporciona diverses instruccions de multiplicació i divisió. El resultat d'aquestes instruccions es col·loca en una parella especial de registres de 32 bits no inclosa en els 32 registres de propòsit general del MIPS. Aquests registres especials s'anomenen *hi* i *lo*.

La interpretació del contingut d'aquests dos registres depèn de l'operació realitzada. En el cas de la multiplicació, la part alta del resultat es deixa en *hi* i la parta baixa en *lo*. Convé notar que la grandària de la paraula és de 32 bits; per tant, si el registre *hi* conté algun valor diferent de zero significa que es pot haver produït desbordament en la multiplicació. Quan és produirà realment desbordament?. En el cas que el resultat haja de ser positiu, si el valor de hi es distint de zero clarament s'haurà produït desbordament. Si el resultat és negatiu no hi haurà desbordament quan tots els bits del registre *hi* són uns i, a més, el bit de major pes de *lo* també ho és.

En el cas de la divisió, en *hi* es col·loca el residu, i en *lo*, el quocient. La divisió per zero no es pot fer en MIPS, és una operació indefinida i no es detecta per l'assemblador, és per tant responsabilitat del programador evitar que es produïsca.

Per a poder extraure els continguts dels registres *hi* i *lo* per a operar amb ells, el MIPS proporciona unes instruccions especials d'accés a aquests

registres. En la taula següent apareixen totes les instruccions relacionades amb la multiplicació i la divisió:

Instrucció	Exemple	Significat	Comentaris
mult	mult Rs, Rt	[hi,lo] ←Rs * Rt	Multiplicació de *Rs* i *Rt*. Deixa resultat en [*hi,lo*]
mflo	mflo Rd	Rd ← lo	Mou el valor de *lo* al registre *Rd*
mfhi	mfhi Rd	Rd ← hi	Mou el valor de *hi* al registre *Rd*
multu	multu Rs, Rt	[hi,lo] ←Rs * Rt	Multiplicació sense signe de *Rs* i *Rt*. Deixa resultat en [*hi-lo*]
div	div Rs, Rt	[hi,lo] ← Rs / Rt	Divisió de Rs per Rt. Deixa resultat en [*hi,lo*]. El quocient en *lo* i el residu en *hi.*
divu	divu Rs, Rt	[hi,lo] ← Rs / Rt	Divisió sense signe de Rs per Rt. Deixa resultat en [*hi,lo*]. El quocient en *lo* i el residu en *hi.*

Taula 3. Instruccions relacionades amb la multiplicació i divisió en MIPS.

Una qüestió important a tenir en compte amb les operacions de multiplicació o divisió és el seu cost temporal. Aquest cost pot arribar a ser molt apreciable i el cost de la divisió és superior al de la multiplicació. Clarament dependrà de la implementació del processador concret i en molts casos de la grandària dels operands. Aquest cost serà molt superior al que resulta de les operacions de multiplicació o divisió amb les instruccions de suma i desplaçament, circumstància que hauríem de tenir present si hem de multiplicar o dividir per constants senzilles.

Qüestió 11

- Escriviu el codi que llig el valor x del teclat i escriviu en la pantalla la solució de l'equació: $5x^2 + 2x + 3$. Les multiplicacions per constants les heu de fer amb instruccions de suma i desplaçament.

4.4 Resum

- S'utilitza una etiqueta en la primera instrucció d'una funció per a donar-li nom.
- Hi ha una sèrie de convenis en l'ús del registres que cal seguir en la programació de funcions del programa.

- La multiplicació i la divisió es pot fer utilitzant instruccions de sumes, restes i desplaçaments o les pròpies que aporta el repertori d'instruccions del MIPS.