ESTRUCTURA DE COMPUTADORS
Grau en Enginyeria Informàtica

Sessió de laboratori número 5

ARITMÈTICA D'ENTERS: MULTIPLICACIÓ I DIVISIÓ

Introducció

En aquesta pràctica es treballa amb l'aritmètica entera del MIPS R2000. En particular, aquesta sessió de laboratori se centra en el treball amb les operacions de multiplicació i divisió de nombres enters. Per a això es planteja el disseny d'un conjunt de subrutines que manegen variables que representen l'hora d'un rellotge. La ferramenta de treball és el simulador del processador MIPS R2000 denominat PCSpim.

Objectius

- Entendre el maneig de les operacions de multiplicació i divisió d'enters.
- Mesurar el cost d'execució d'un programa que treballa amb operacions de multiplicació i divisió d'enters.

Material

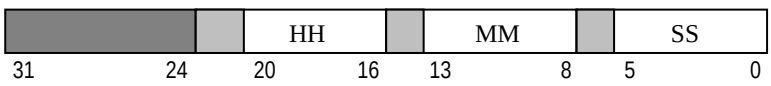
El material es pot obtindre de la carpeta de recursos de PoliformaT.

- Simulador PCSpim del MIPS R2000.
- Arxiu font: `reloj.s`

Codificació d'un format horari i la seua inicialització

En aquesta pràctica treballarem amb una variable que representa l'estat d'un rellotge. Suposarem que el valor del rellotge s'expressa com una tripleta HH:MM:SS (hores, minuts i segons). El rang de representació horària comprén des de 00:00:00 fins a 23:59:59. A tall d'exemple, el rellotge pot contenir valors horaris com ara 13:25:43, 00:10:53, 07:59:32, etcètera. En particular, el camp d'hores pot contenir els valors 0…23, el de minuts 0…59 i el de segons 0…59. En conseqüència, valors com 43:90:21 o 128:40:298 no són vàlids.

Per a manejar aquest tipus de variables en el computador s'utilitza una única paraula de memòria de 32 bits atenent a la distribució de camps que mostra la figura següent. Com s'aprecia en la figura, cada element de la tripleta es codifica en un únic byte i està situat en els bits de menor pes d'este byte.



Ara bé, considerant els valors horaris vàlids, no tots els bits dels 32 que té la paraula s'utilitzen en la codificació de l'estat del rellotge. Els bits no utilitzats han sigut assenyalats gràficament amb un to de gris, i el seu valor, en principi, no està definit. En particular, el byte de major pes (bits 24…31) no s'utilitza. El camp HH necessita 5 bits (2^5=32) ja que pot contenir 24 valors distints. Els camps MM i SS són de 6 bits (2^6=64) perquè poden contenir 60 valors distints.

►Quin valor del rellotge representa la paraula de bits **`0x0017080A`**?

16 : 32 : 10

►Quin valor del rellotge representa la paraula de bits **`0xF397C84A`**?

~~28 : 33 : 10~~ 23 : 08 : 10

►Indica tres codificacions diferents de la variable rellotge per al valor horari 16:32:28.

0XFFF0609C ~~0X00010821C~~

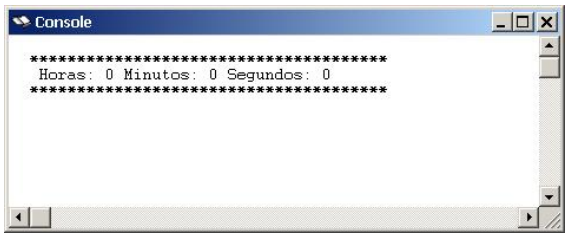
0X E9F060 9C

Considera com a punt de partida el programa en ensamblador contingut en el fitxer `reloj.s`. A continuació referim els elements més significatius de la declaració de variables i del programa principal.

```
                ############################################################
                # Segment de dades
                ############################################################

                .data 0x10000000
rellotge:       .word 0                 # HH:MM:SS (3 bytes de menor pes)

                ############################################################
                # Segment de codi
                ############################################################

                .globl __start
                .text 0x00400000

__start         la $a0, rellotge
                jal imprimeix_rellotge

eixir:          li $v0, 10              # Codi d'exit (10)
                syscall                 # Última instrucció executada
```

El programa disposa en memòria la variable **`rellotge`** per a emmagatzemar una paraula d'acord amb el format horari que hem descrit. La subrutina **`imprimeix_rellotge`** imprimeix en pantalla el valor contingut en la variable horària passada per referència a través del registre **`$a0`**. El programa acaba executant la crida al sistema **`exit`**.

►Carrega el fitxer `reloj.s` i executa-ho en el simulador. Tal com està, el resultat mostrat en la consola ha de ser el següent:



►Per què s'ha imprés l'hora 00:00:00?

perque esta inicialitzat a 0

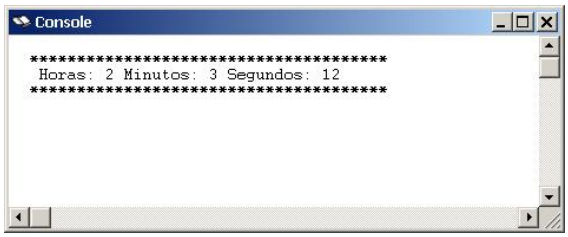
Es vol dissenyar un conjunt de subrutines que inicialitzen la variable **`rellotge`**. En primer lloc dissenyarem una subrutina per a inicialitzar tots els camps del rellotge amb un valor concret d'hores, minuts i segons. La següent taula especifica el seu funcionament.

NOM	ARGUMENTS D'ENTRADA	EIXIDA
`inicialitza_rellotge`	**`$a0`**: adreça del rellotge **`$a1`**: HH:MM:SS	rellotge = HH:MM:SS

Per exemple, per a inicialitzar el rellotge amb l'hora 02:03:12 i veure el resultat en pantalla caldrà executar el codi:

```
la $a0, rellotge
li $a1, 0x0002030C
jal inicialitza_rellotge

la $a0, rellotge
jal imprimeix_rellotge
```

El resultat d'esta execució ha de ser:



►Implementa la subrutina **`inicialitza_rellotge`**.

Ara es vol dissenyar una subrutina alternativa d'inicialització del rellotge amb l'especificació següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
`inicialitza_rellotge_alt`	**`$a0`**: adreça del rellotge **`$a1`**: HH **`$a2`**: MM **`$a3`**: SS	rellotge = HH:MM:SS

Encara que el resultat de la subrutina és idèntic al de la subrutina **`inicialitza_rellotge`** que ja hem dissenyat, el seu funcionament és diferent. Ara els tres camps del rellotge s'especifiquen com a arguments en tres registres distints. A més, es vol que en el disseny d'esta nova subrutina intervinga només una única instrucció d'accés a memòria. Açò significa que el codi ha de construir la paraula de 32 bits que represente el valor HH:MM:SS i escriure-la en memòria fent ús d'una única instrucció. Esta construcció es pot fer utilitzant instruccions lògiques i de desplaçament.

►Implementa la subrutina `inicialitza_rellotge_alt`.

Com a treball a casa es proposa dissenyar un conjunt de subrutines per a inicialitzar per separat cada un dels camps d'una variable rellotge, segons s'especifica en la taula següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
`inicialitza_rellotge_hh`	**`$a0`**: adreça del rellotge **`$a1`**: HH	rellotge.hh = HH
`inicialitza_rellotge_mm`	**`$a0`**: adreça del rellotge **`$a1`**: MM	rellotge.mm = MM
`inicialitza_rellotge_ss`	**`$a0`**: adreça del rellotge **`$a1`**: SS	rellotge.ss = SS

Per exemple, per a inicialitzar el camp MM del rellotge amb el valor 59 es farà:

```
la $a0, rellotge
li $a1, 0x3B
jal inicialitza_rellotge_mm
```

►Implementa el codi de les subrutines **`inicialitza_rellotge_hh`**, **`inicialitza_rellotge_mm`** i **`inicialitza_rellotge_ss`**.

►En principi, un únic valor de rellotge HH:MM:SS pot codificar-se de diferents maneres segons els valors que assignem als bits que no entren a formar part de la codificació dels camps HH, MM i SS. Ara volem obligar que totes les hores es representen d'una única manera fent que els bits del rellotge que no estan definits siguen sempre zero. Per exemple, l'hora 02:03:12 només es codifica només com `0x0002030C`, mentres que altres combinacions com `0x6502030C`, `0x89E203CC` o `0xFFC2038C` no estan permeses. Com serà ara la subrutina **`inicialitza_rellotge`** per a complir amb aquesta condició?

►La subrutina següent opera sobre una variable rellotge la direcció de la qual es passa com a argument en el registre `$a0` i amb un valor X que es passa en el byte menys significatiu de `$a1`. Explica raonadament quin efecte produïx l'execució de la subrutina.

```
subrutina:      lw $t0, 0($a0)
                li $t1, 0x00FFFF00
                and $t0, $t0, $t1
                or $t1, $t0, $a1
                sw $t1, 0($a0)
                jr $ra
```

La multiplicació i la divisió d'enters i el seu cost temporal

Les operacions de multiplicació i divisió d'enters s'implementen en l'arquitectura del MIPS R2000 per mitjà de dos instruccions màquina, `mult` i `div`, respectivament, i `multu` i `divu`, per a nombres enters sense signe. Estes instruccions deixen el resultat en una parella especial de registres denominats `hi` i `lo`.

La interpretació del contingut d'estos registres particulars `hi` i `lo` depén de la instrucció que s'execute. Després d'executar una instrucció de multiplicació, `hi` i `lo` contenen la part alta i baixa, respectivament, del resultat. En aquest cas es considera que hi ha hagut desbordament en la multiplicació quan el resultat necessita més de 32 bits per a ser representat, és a dir, quan `hi` és diferent de zero; la detecció d'este possible desbordament és responsabilitat del programador. D'altra banda, en una operació de divisió, `hi` conté el residu i `lo` el quocient. En l'arquitectura del MIPS R2000 la divisió per zero és una operació indefinida. Per tant, és responsabilitat del programador comprovar que el divisor és diferent de zero abans d'executar una instrucció de divisió.

Per a poder operar amb els valors continguts en estos dos registres especials fa falta traslladar-los prèviament al banc de registres de la unitat aritmeticològica per mitjà de les instruccions de moviment `mfhi` (*move from hi*) i `mflo` (*move from lo*).

Vegem un exemple senzill que il·lustra tot el que s'acaba d'exposar:

```
li $t0, 18        # $t0 = 18
li $t1, 4         # $t1 = 4
mult $t0, $t1     # lo = 18*4 =0x00000048 i hi = 0x00000000
mflo $s0          # $s0 = lo
div $t0, $t1      # lo = 18÷4 = 4 i hi = 18%4 = 2
mfhi $s1          # $s1 = hi
mflo $s2          # $s2 = lo
```

El resultat de la multiplicació (18×4=72) cap en el registre `lo`, per la qual cosa només s'ha mogut el seu contingut a `$s0`. En el cas de la divisió, el quocient (18÷4=4) s'ha emmagatzemat en `lo` i el residu (18%4=2) en `hi`; han fet falta dues instruccions de moviment per a portar estos dos valors als registres `$s1` i `$s2`, respectivament.

Una qüestió molt important a tindre en compte és la complexitat temporal de les operacions de multiplicació i divisió de sencers. Així com les operacions lògiques, de desplaçament o aritmètiques bàsiques com la suma o la resta es poden executar en un sol cicle de rellotge del processador, no ocorre així amb la multiplicació i la divisió. D'estes dos últimes operacions, la segona és generalment la que més tarda. El cost temporal exacte, no obstant això, depén de la implementació

del processador i, en algunes ocasions concretes, de la grandària dels operands. Per a fer-nos una idea, una operació de divisió pot tardar entre 35 i 80 cicles de rellotge, mentres que una operació de multiplicació tarda entre 5 i 32 cicles de rellotge. Si suposem que la multiplicació tarda 20 cicles i la divisió 70, podem estimar el temps d'execució del codi anterior en 1+1+20+1+70+1+1=95 cicles de rellotge. Note's que en aquest càlcul s'ha tingut en compte que, atés que els valors de les constants poden codificar-se amb 16 bits, les pseudoinstruccions `li` del codi es poden traduir per una única instrucció màquina de tipus immediat (per exemple, `li $t0, 18` es pot traduir en `ori $t0, $zero, 18`).

L'operació de multiplicació: conversió de HH:MM:SS a segons

Es vol dissenyar una subrutina que converteix en segons el valor d'una variable rellotge codificada per mitjà de la tripleta amb la forma HH:MM:SS. Per exemple, l'hora 18:32:45 equival a 66765 segons; per al càlcul n'hi ha prou amb fer 18×3600+32×60+45=66765.

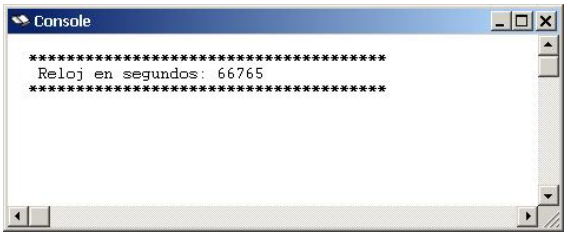
La subrutina s'anomena `torna_rellotge_en_s`. La seua especificació és la següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
`torna_rellotge_en_s`	**`$a0`**: adreça del rellotge	**`$v0`**: segons

El programa proporcionat inicialment en el fitxer `reloj.s` inclou la subrutina `imprimeix_s` per a imprimir segons (es passen com a argument en `$a0`). Per exemple, el codi següent calcula en segons l'hora 18:32:45 i imprimeix el resultat:

```
la $a0, rellotge
li $a1, 0x0012202D
jal inicialitza_rellotge
la $a0, rellotge
jal torna_rellotge_en_s
move $a0, $v0
jal imprimeix_s
```

El resultat mostrat per pantalla és:



►Per a llegir de memòria per separat cada un dels camps del rellotge (HH, MM i SS) es pot usar una instrucció de lectura de byte. Raone si cal utilitzar `lb` (*load byte*) o `lbu` (*load byte unsigned*).

lbu perque necessitem llegir els bytes sense signe

►Implementa la subrutina `torna_rellotge_en_s`.

►Quin tipus d'instruccions de suma han d'utilitzar-se en la subrutina, `add` o `addu`?

addu perque treballem sense signe

►Indica el nombre d'instruccions de multiplicació que s'executen en la subrutina `torna_rellotge_en_s`. 2

►Indica el nombre d'instruccions de moviment d'informació entre els registres del banc d'enters i els registres `hi` i `lo` que s'executen en la subrutina dissenyada. 2

►Es vol completar la subrutina `torna_rellotge_en_s` amb la detecció de desbordament de la multiplicació. Atés que estem treballant amb números positius, cal incloure les instruccions necessàries per a detectar si, després de dur a terme l'operació de multiplicació, el resultat ocupa més de 32 bits. En aquest últim cas, cal botar a l'etiqueta **`eixir`** per a acabar l'execució del programa. Indica les instruccions necessàries per a aquesta detecció de desbordament.

►Suposem que totes les instruccions tarden un cicle de rellotge a executar-se i les de multiplicació tarden 20 cicles. Quant de temps tarda a executar-se el codi de la subrutina de conversió a segons?

50 cicles Es contant

L'operació de divisió: conversió de segons a HH:MM:SS

En aquest apartat plantegem el problema invers considerat en la secció anterior: ara es vol inicialitzar el valor del rellotge partint prèviament d'una quantitat determinada de segons. Així, volem dissenyar una subrutina que, donat un nombre de segons, inicialitze el rellotge amb el valor corresponent però expressat en format HH:MM:SS. Per exemple, un temps de 66765 segons equival al valor 18:32:45, és a dir, 18 hores, 32 minuts i 45 segons. Per a passar de segons a HH:MM:SS fa falta dividir dos vegades per la constant 60, segons s'il·lustra en aquest cas concret:

- 66765 segons ÷ 60 = 1112 minuts (la resta són 45 segons)
- 1112 minuts ÷ 60 = 18 hores (la resta són 32 minuts)

Així, SS correspon a la resta de la primera divisió, MM a la resta de la segona divisió i HH al quocient de la segona divisió.

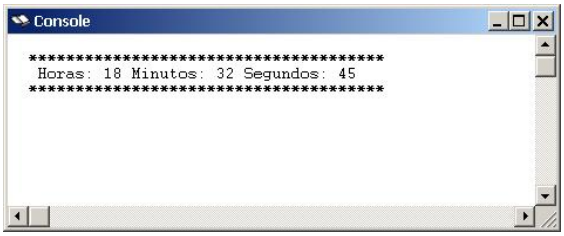
La subrutina que implementa aquesta inicialització s'anomenarà `inicialitza_rellotge_en_s`. La seua especificació és la següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
`inicialitza_rellotge_en_s`	**`$a0`**: adreça del rellotge **`$a1`**: segons	rellotge = HH:MM:SS

Per exemple, el codi següent inicialitza el rellotge amb l'hora corresponent a 66765 segons i imprimeix el resultat:

```
la $a0, rellotge
li $a1, 66765
jal inicialitza_rellotge_en_s

la $a0, rellotge
jal imprimeix_rellotge
```

El resultat mostrat per pantalla és:



►Implementa el codi de la subrutina `inicialitza_rellotge_en_s`.

►Indica el nombre d'instruccions de divisió que s'executen en la subrutina `inicialitza_rellotge_en_s`. 2

►Indica el nombre d'instruccions de moviment d'informació entre els registres del banc d'enters i els registres `hi` i `lo` que s'executen en la subrutina dissenyada. 3

►Suposem que totes les instruccions tarden un cicle de rellotge a executar-se mentres que les de divisió tarden 70 cicles. Quin serà el temps que tarda l'execució del codi de la subrutina `inicialitza_rellotge_en_s`?

►Es volen evitar possibles errors en l'operació de divisió fent que, abans de que es puga produir una divisió per zero, la subrutina ho detecte i bote a l'etiqueta **`eixir`** per a acabar l'execució del programa. Indica les instruccions necessàries per a dur a terme aquesta detecció de la divisió per zero.