

Sessió de laboratori número 6

ARITMÈTICA D'ENTERS: SUMES, RESTES I DESPLAÇAMENTS

Introducció

En aquesta pràctica es treballa amb l'aritmètica d'enters del MIPS R2000. En particular, aquesta sessió de laboratori se centra en el treball amb les operacions de suma, resta i desplaçament 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.
- Substituir instruccions de multiplicació d'enters per un conjunt de sumes, restes i desplaçaments.
- Quantificar la millora del temps d'execució degut a la substitució de les instruccions de multiplicació d'enters per un conjunt equivalent d'instruccions de suma, resta i desplaçament.
- Comprendre l'operació de suma per a variables temporals i tractar el transport (carry).

Material

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

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

El format horari i la seua inicialització

En aquesta pràctica treballarem novament amb la 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) en una única paraula de memòria de 32 bits atenent a la distribució de camps que mostra la figura següent.



En la sessió de laboratori anterior s'ha treballat amb aquest format horari i s'han implementat algunes subrutines que treballen amb ell. En aquesta sessió necessitarem alguna d'estes subrutines; en particular, la que inicialitza una variable rellotge i també la que converteix en segons un valor

horari expressat per la tripleta HH:MM:SS. La taula següent mostra el funcionament d'estes dos subrutines.

NOM	ARGUMENTS D'ENTRADA	EIXIDA
<code>inicialitza_rellotge</code>	<code>\$a0</code> : adreça del rellotge <code>\$a1</code> : HH:MM:SS	rellotge = HH:MM:SS
<code>torna_rellotge_en_s</code>	<code>\$a0</code> : adreça del rellotge	<code>\$v0</code> : segons

Considera com a punt de partida el programa en ensamblador contingut en el fitxer `reloj.s`, al qual el alumne li ha d'haver afegit, almenys, les dos subrutines anteriors implementades en la sessió anterior del laboratori. 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 que se li passa per referència a través del registre `$a0`. El programa acaba executant la crida al sistema `exit`.

Multiplicació per mitjà de sumes i desplaçaments

Atés que l'execució d'una instrucció de multiplicació sol ser molt costosa en nombre de cicles de rellotge, en les ocasions en què algun dels operands ho permet, els compiladors poden optar per reemplaçar-les per un conjunt de sumes, restes i desplaçaments que, en termes globals, suposen un temps d'execució inferior. Recordem que una instrucció de multiplicació pot tardar, segons la implementació del processador, entre 5 i 32 cicles de rellotge. Les operacions de suma, resta i desplaçament, no obstant això, tarden un sol cicle a executar-se.

La multiplicació per un número que és potència sencera de dos pot fer-se utilitzant desplaçaments cap a l'esquerra. Per exemple, per a multiplicar un nombre enter per 4 ($4 = 2^2$) basta de desplaçar-ho cap a l'esquerra dos posicions. Per exemple, considerem el producte $7 \times 2^2 = 28$. Si codifiquem el número 7 en un byte i ho expressem en binari obtenim 00000111; si desplaçem els bits dos posicions cap a l'esquerra i omplim els buits generats amb zeros obtenim 00011100, byte que correspon a la codificació binària del número 28.

Però també podem basar-nos en la propietat anterior i aprofitar-nos d'ella per a multiplicar per constants que no són potències senceres de dos. Per exemple, imaginem que volem multiplicar el contingut del registre \$a0 per la constant 15. En principi, podríem utilitzar directament la instrucció de multiplicació:

```
li $t0, 15
mult $a0, $t0    # lo = $a0*15
mflo $v0         # $v0 = lo
```

Si la instrucció de multiplicació tarda 20 cicles de rellotge i la resta d'instruccions tarda un cicle, llavors el codi anterior tarda a executar-se un total d' $1+20+1=22$ cicles de rellotge.

D'altra banda, el número 15 es pot expressar com a suma de potències senceres de dos: $15 = 2^3+2^2+2^1+2^0$. Per tant, el producte $\$a0 \times 15$ es transforma de forma equivalent en $\$a0 \times (2^3+2^2+2^1+1)$. D'acord amb l'expressió anterior, cal fer una sèrie de desplaçaments i sumes atenent als uns que tinga la constant ($15 = 00001111_2$). En aquest cas concret podem transformar l'operació de multiplicació per tres sumes i tres desplaçaments (el terme 2^0 no genera cap desplaçament). Així doncs, el codi anterior es pot substituir completament pel següent codi alternatiu i d'idèntic resultat:

```
sll $v0, $a0, 3    # $v0 = $a0*2^3
sll $t0, $a0, 2    # $t0 = $a0*2^2
addu $v0, $v0, $t0 # $v0 = $a0*(2^3 + 2^2)
sll $t0, $a0, 1    # $t0 = $a0*2^1
addu $v0, $v0, $t0 # $v0 = $a0*(2^3 + 2^2 + 2^1)
addu $v0, $v0, $a0 # $v0 = $a0*(2^3 + 2^2 + 2^1 + 2^0)
```

Este codi alternatiu de 6 instruccions de desplaçaments i suma tarda a executar-se 6 cicles de rellotge, la qual cosa suposa una millora del temps d'execució respecte del primer de $20/6=3.33$ vegades, açò és, el codi anterior s'executa 3.33 vegades més ràpidament que el primer.

Per a utilitzar aquesta nova forma de dur a terme la multiplicació en el programa que ens ocupa de conversió del rellotge en segons hem de considerar els productes per les constants 3600 i 60. Estes dos constants es poden expressar com a potències senceres de dos de la manera següent:

$$\begin{aligned} 3600 &= 0000\ 1110\ 0001\ 0000_2 = 2^{11} + 2^{10} + 2^9 + 2^4 \\ 60 &= \quad\quad\quad 0011\ 1100_2 = 2^5 + 2^4 + 2^3 + 2^2 \end{aligned}$$

Així doncs, per a multiplicar un valor per qualsevol d'estes constants fan falta quatre desplaçaments cap a l'esquerra (instruccions sll) i tres sumes (instruccions addu). Noteu que en aquest cas, atés que no apareix el terme 2^0 , totes les potències de dos generen una operació de desplaçament.

► Escriu el codi necessari per a multiplicar el contingut del registre \$a0 per la constant 36 i tornar el resultat en el registre \$v0 utilitzant sumes i desplaçaments.

Fent ús de la idea que s'acaba d'exposar, dissenyarem una versió alternativa de la subrutina torna_rellotge_en_s denominada torna_rellotge_en_s_sd en la qual les instruccions de multiplicació se substitueixen per un conjunt equivalent d'instruccions de suma i desplaçament. El funcionament de la subrutina és el següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
torna_rellotge_en_s_sd	\$a0: adreça del rellotge	\$v0: segons

► Implementa el codi de la subrutina torna_rellotge_en_s_sd.

► Suposem que totes les instruccions tarden un cicle de rellotge a executar-se i les de multiplicació tarden 20 cicles. Quant de temps tarda en executar-se el codi de la subrutina torna_rellotge_en_s_sd? Quantes vegades és més ràpida aquesta última subrutina que torna_rellotge_en_s?

Multiplicació per mitjà de sumes, restes i desplaçaments

La tècnica de multiplicació per descomposició d'una constant en suma de potències que hem usat en l'apartat anterior es pot adaptar a l'escriptura del multiplicador segons la codificació de Booth. En aquesta codificació el pes de cada dígit és idèntic al de la codificació normal, però ara els dígit són tres, 0, +1 i -1.

Per exemple, la constant 15 es pot expressar com el byte 0000 1111₂ en binari natural, i com 000+1000-1_{Booth} segons la codificació de Booth. Dit d'una altra manera, la constant 15 es pot descompondre, amb la primera opció, en $2^3+2^2+2^1+2^0 = 8+4+2+1$ i amb la segona opció, en $2^4-2^0 = 16-1$.

Es pot entendre que, per a aquest cas particular, la constant 15 té una descomposició en potències de dos més curta, que ara inclou l'operació de resta, i que per tant, pot ajudar a simplificar la generació

de codi. Així, la multiplicació del registre \$a0 per 15 que analitzàvem en l'apartat anterior es pot implementar amb el codi següent:

```
sll $v0, $a0, 4      # $v0 = $a0*24
subu $v0, $v0, $a0    # $v0 = $a0*(24 - 20)
```

És evident que aquesta tècnica només resulta d'utilitat pràctica quan la codificació de Booth de les constants senceres proporciona un número reduït de dígit +1 i -1 comparat amb el nombre d'uns de la codificació en binari natural.

Tornant novament al nostre programa de càlcul de segons, podem considerar la codificació de Booth de les constants 3600 i 60 per a veure si ens permet un estalvi en el nombre de sumes i desplaçaments. La codificació de Booth d'estes dos constants és:

$$\begin{array}{lcl} 3600 = 000+1 & 00-10 & 00+1-1 & 0000_{\text{Booth}} = 2^{12} - 2^9 + 2^5 - 2^4 \\ 60 = & & 0+100 & 0-100_{\text{Booth}} = 2^6 - 2^2 \end{array}$$

En conseqüència, veiem que només la constant 60 permet una reducció del nombre d'operacions, ja que de quatre desplaçaments i tres sumes passem a dos desplaçaments i una resta. La complexitat associada a la constant 3600 roman igual perquè només hi ha hagut una substitució de dos sumes per dos restes.

En qualsevol cas recordem que en tot moment estem tractant d'implementar la mateixa operació (multiplicació) però de distint mode (sumes, restes i desplaçaments) ajudant-nos de les diferents codificacions d'un dels operands de la multiplicació:

$$\begin{array}{lcl} \$a0*3600 & = & \$a0*(2^{11}+2^{10}+2^9+2^4) = \$a0*(2^{12}-2^9+2^5-2^4) \\ \$a0*60 & = & \$a0*(2^5+2^4+2^3+2^2) = \$a0*(2^6-2^2) \end{array}$$

► Escriu el codi necessari per a multiplicar el contingut del registre \$a0 per la constant 31 i tornar el resultat en el registre \$v0 utilitzant sumes, restes i desplaçaments.

Fent ús de la idea que s'acaba d'exposar, dissenyarem una altra versió alternativa de la subrutina torna_rellotge_en_s denominada torna_rellotge_en_s_srd en la qual se substitueixca només la instrucció de multiplicació per la constant 60 per un conjunt equivalent d'instruccions de suma, resta i desplaçament. El funcionament de la subrutina és el següent:

NOM	ARGUMENTS D'ENTRADA	EIXIDA
torna_rellotge_en_s_srd	\$a0: adreça del rellotge	\$v0: segons

► Implementa el codi de la subrutina torna_rellotge_en_s_srd. Comprova que el resultat de l'execució és el mateix que en el cas anterior.

► Suposem que tots 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 `torna_rellotge_en_s_srd`? Quantes vegades és més ràpida aquesta última subrutina que `torna_rellotge_en_s`?

Increment del rellotge

En aquest últim apartat considerarem el problema d'incrementar el valor del rellotge. En primer lloc considerem la subrutina `passa_hora` inclosa en el fitxer `reloj.s`.

NOM	ARGUMENTS D'ENTRADA	EIXIDA
<code>passa_hora</code>	<code>\$a0</code> : adreça del rellotge	rellotge = HH:MM:SS + 1 h

El codi d'esta subrutina es mostra a continuació:

```
passa_hora:    lbu $t0, 2($a0)        # $t0 = HH
               addiu $t0, $t0, 1      # $t0 = HH++
               li $t1, 24
               beq $t0, $t1, H24      # Si HH==24 és posa HH a zero
               sb $t0, 2($a0)        # Escribe HH++
               j fi_passa_hora
H24:          sb $zero, 2($a0)        # Escribe HH a 0
fi_passa_hora: jr $ra
```

Com es pot apreciar, la subrutina comprova si l'increment del camp HH fa que supere el seu valor màxim; si és així, el camp HH passa a valdre 0. Per exemple, si el rellotge amb valor 07:48:21 s'incrementa en una hora passa a valdre 08:48:21. Açò és mostra en el codi següent:

```
la $a0, rellotge
li $a1, 0x00073015    # Hora 07:48:21
jal inicialitza_rellotge

la $a0, rellotge
```

```

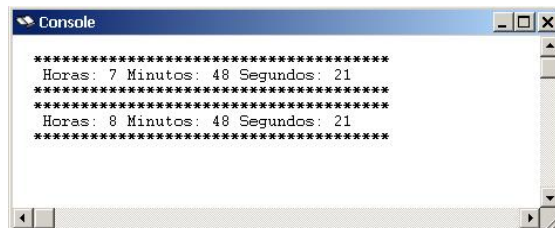
jal imprimeix_rellotge

la $a0, rellotge
jal passa_hora          # Incrementa el rellotge en una hora

la $a0, rellotge
jal imprimeix_rellotge

```

i el seu resultat en pantalla:



No obstant això, considerant que el rellotge funciona de forma cíclica, quan el rellotge tinga el valor 23:48:21 i s'incrementa en una hora llavors el seu nou valor passa a ser 00:48:21 i no 24:48:21.

Considerem ara el disseny d'una subrutina que incrementa l'hora del rellotge en un segon. El seu nom és `passa_segons` i el seu funcionament s'especifica en la següent taula.

NOM	ARGUMENTS D'ENTRADA	EIXIDA
<code>passa_segons</code>	<code>\$a0</code> : adreça del rellotge	rellotge = HH:MM:SS + 1 s

La subrutina hi ha d'accedir al valor del rellotge emmagatzemat en memòria i, com indica el seu nom, incrementar-lo en un segon. Per exemple, el rellotge 19:22:40 passa a valdre 19:22:41. No obstant això, cal tindre en compte un parell de casos especials. En primer lloc, quan l'increment d'un segon involucra al seu torn l'increment dels minuts. Per exemple, l'hora 19:22:59 passaria a valdre 19:23:00. En segon lloc, l'increment del camp dels minuts també poden donar lloc a l'increment del camp de les hores; aquest és el cas de l'hora 23:59:59, que després de l'increment d'un segon passa a valdre 00:00:00. A manera d'exemple, l'execució del codi següent parteix del rellotge amb aquest valor horari i el incrementa en dos segons:

```

la $a0, rellotge
li $a1, 0x00173b3b      # Hora 23:59:59
jal inicialitza_rellotge

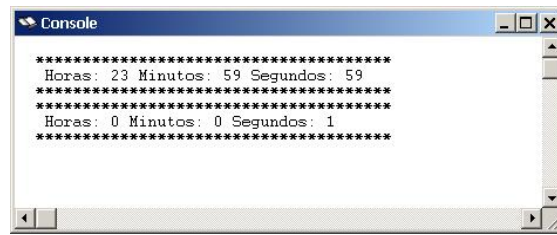
la $a0, rellotge
jal imprimeix_rellotge

la $a0, rellotge
jal passa_segons         # Incrementa el rellotge en un segon
jal passa_segons         # Incrementa el rellotge en un segon

la $a0, rellotge
jal imprimeix_rellotge

```

El resultat que ofereix aquest codi ha de ser el següent:



```
*****
Horas: 23 Minutos: 59 Segundos: 59
*****
Horas: 0 Minutos: 0 Segundos: 1
*****
```

► Implementa el codi de la subrutina `passa_seg`.

► Escriu el codi necessari per a inicialitzar una variable rellotge amb el valor 21:15:45 i incrementar en tres hores i 40 segons. Utilitza les subrutines `inicialitza_rellotge`, `passa_hora` i `passa_seg`. Quin és el valor final del rellotge?