ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Programación en ensamblador de MIPS
Ejercicios resueltos

Luis Rincón Córcoles

Departamento de Arquitectura y Tecnología de Computadores y CCIA

Ejercicio 1

Diseñar un programa en ensamblador de MIPS que evalúe la siguiente expresión aritmética, donde todas las variables son de tipo entero:

$$e = a + \frac{b \times c}{4} - d \times (a + 2)$$

Adicionalmente existirá una variable carácter llamada `signo` en la que se escribirá el carácter `"+"` si el contenido final de `e` es positivo o nulo, mientras que si `e` termina siendo negativo, en `signo` se copiará el carácter `"-"`.

A continuación se muestra una posible solución escrita en Pascal:

```
PROGRAM expresion;

VAR
   a,b,c,d,e: INTEGER;
   signo: CHAR;

BEGIN {Programa principal}
   ...
   e := a + (b*c) div 4 - d*(a+2);
   IF e >= 0
      THEN signo := "+";
      ELSE signo := "-";
   ...
END.
```

Solución

Una solución al ejercicio podría ser la siguiente:.

```
######################
#
# Programa que calcula la expresion e := a + (b*c) div 4 - d*(a+2);
# y escribe en signo un caracter segun e sea negativo o no.
#
######################

######################
#
# Seccion de datos del programa principal
#
######################
a:          .space  4              # Se usara $s0
b:          .space  4              # Se usara $s1
c:          .space  4              # Se usara $s2
d:          .space  4              # Se usara $s3
e:          .space  4              # Se usara $s4
signo:      .space  1              # Se usara $s5
```

```
#####################
#
# Seccion de codigo
#
#####################
            .text
main:
# Codigo previo del programa principal ...
# Seria preciso iniciar las variables a, b, c y d
            ...
# Codigo del enunciado
#   e := a + (b*c) div 4 - d*(a+2);
# Esta expresion se evaluara utilizando los registros temporales $t0 ... $t9
# Para las variables se usan registros salvados $s0 ... $s7
# como si fueran una pila
            lw      $s0,a           # $s0 <- a
            lw      $s1,b           # $s1 <- b
            lw      $s2,c           # $s2 <- c
            mult    $s1,$s2         # $t0 <- b*c
            mflo    $t0
            li      $t1,4           # $t1 <- 4
            div     $t0,$t1         # $t0 <- b*c div 4 ($t1 "sale de la pila")
            mflo    $t0
            addu    $t0,$s0,$t0     # $t0 <- a + (b*c div 4)
            lw      $s3,d           # $s3 <- d
            addiu   $t1,$s0,2       # $t1 <- a+2
            mult    $s3,$t1         # $t1 <- d * (a+2)
            mflo    $t1
            subu    $s4,$t0,$t1     # $s4 <- a + (b*c div 4) - d * (a+2)
                                    # Fin de la evaluacion de la expresion
            sw      $s4,e           # e <- $s4 (almacenar resultado final)
# IF e >= 0
if:
            lw      $s4,e           # $s4 <- e (innecesario, e ya esta en $s4)
            bge     $s4,$zero,then
# ELSE signo := "-"
else:
            li      $s5,"-"
            sb      $s5,signo
            b       endif
# THEN signo := "+"
then:
            li      $s5,"+"
            sb      $s5,signo
endif:
#   Fin de IF e >= 0
# Resto del codigo del programa principal ...
            ...
# END.
# Secuencia de final de programa
            li      $v0,10
            syscall
```

Se propone al alumno que realice optimizaciones en el código con objeto de incrementar la velocidad de ejecución de la solución.

Ejercicio 2

Se pretende diseñar un programa en ensamblador de MIPS que, dado un entero que se supone positivo, inspeccione los bits que lo componen y contabilice cuántos de entre ellos se encuentran a 1. Se parte del siguiente fragmento de programa en Pascal:

```
VAR
   dato, unos: INTEGER;
   ...
   unos := 0;
   WHILE dato > 0 DO
      BEGIN
         IF dato MOD 2 <> 0
            THEN unos := unos + 1;
         dato := dato DIV 2;
      END;
  ...
```

Solución

Una solución al ejercicio podría ser la siguiente:

```
#####################
#
# Programa que calcula el numero de bits a 1 de un dato numerico positivo
#
#####################

#####################
#
# Seccion de datos del programa principal
#
#####################
dato:        .space  4              # Se usara $s1
unos:        .space  4              # Se usara $s0

#####################
#
# Seccion de codigo
#
#####################
             .text
main:
# Codigo previo del programa principal ...
# Seria preciso iniciar la variable dato
             ...
# Codigo del enunciado
```

```
#   unos := 0;
            li      $s0,0
# WHILE dato > 0 DO
while:
            ble     $s1,$zero,end_while
# IF dato MOD 2 <> 0
            li      $t0,2
            rem     $t1,$s0,$t0
            beq     $t1,$zero,end_if
# THEN unos := unos + 1;
then:
            addiu   $s0,$s0,1
end_if:
# dato := dato DIV 2;
            li      $t0,2
            div     $s1,$s1,$t0
# END; {del WHILE}
            b       while
end_while:
# Almacenamiento de variables
            sw      $s0,unos
            sw      $s1,dato
# Resto del codigo del programa principal ...
            ...
# END.
# Secuencia de final de programa
            li      $v0,10
            syscall
```

Se propone al alumno que realice optimizaciones en el código con objeto de incrementar la velocidad de ejecución de la solución.

Ejercicio 3

Sea el siguiente fragmento de código escrito en Pascal, que utiliza el algoritmo de Euclides para calcular el máximo común divisor y el mínimo común múltiplo de dos números:

```
VAR
   x, y, mcd, mcm, tmp, resto: INTEGER;
   ...
BEGIN
   ...
   IF (x >= y) THEN BEGIN
      tmp := x;
      mcd := y;
   END;
   ELSE BEGIN
      tmp := y;
      mcd := x;
   END;
   REPEAT
      resto := tmp MOD mcd;
```

```
      IF resto <> 0 THEN BEGIN
         tmp := mcd;
         mcd := resto;
      END;
   UNTIL resto = 0;
   mcm := (x * y) DIV mcd;
   ...
END.
```

Se pretende diseñar un programa en ensamblador de MIPS que realice dicho cálculo.

Es obligatorio que, al terminar la ejecución del algoritmo, los valores de de las variables estén almacenados en memoria. De no ser así, se considerará que el ejercicio está mal realizado.

Solución

A continuación se muestra una posible implementación de la solución:

```
######################
#
# Programa que calcula el mcd y el mcm de dos enteros
#
######################

######################
#
# Seccion de datos del programa principal
#
######################
x:           .space  4              # Se usara $s0
y:           .space  4              # Se usara $s1
mcd:         .space  4              # Se usara $s2
mcm:         .space  4              # Se usara $s3
tmp:         .space  4              # Se usara $s4
resto:       .space  4              # Se usara $s5

######################
#
# Seccion de codigo
#
######################
             .text
main:
# Codigo previo del programa principal ...
# Seria preciso iniciar las variables x e y
        ...
# Codigo del enunciado
# IF (x >= y)
IF1:         lw   $s0,x
             lw   $s1,y
             bge     $s0,$s1,THEN1
# ELSE BEGIN
```

```
ELSE1:
# tmp := y;
             move    $s4,$s1
# mcd := x;
             move    $s2,$s0
# END;
             b    ENDIF1
# THEN BEGIN
THEN1:
# tmp := x;
             move    $s4,$s0
# mcd := y;
             move    $s2,$s1
# END;
ENDIF1:
# REPEAT
REPEAT:
# resto := tmp MOD mcd;
             rem     $s5,$s4,$s2
# IF resto <> 0
IF2:      beq     $s5,$zero,ENDIF2
# THEN BEGIN
THEN2:
# tmp := mcd;
             move    $s4,$s2
# mcd := resto;
             move    $s2,$s5
# END;
ENDIF2:
# UNTIL resto = 0;
UNTIL:    bne     $s5,$zero,REPEAT
# mcm := (x * y) DIV mcd;
             mul     $s3,$s0,$s1
             div     $s3,$s3,$s2
# Almacenar variables
             sw      $s2,mcd
             sw      $s3,mcm
             sw      $s4,tmp
             sw      $s5,resto
# Resto del codigo del programa principal ...
             ...
# END.
# Secuencia de final de programa
             li      $v0,10
             syscall
```

Ejercicio 4

El método de Newton-Raphson sirve para calcular las raíces de una ecuación $f(x) = 0$ de forma iterativa. La fórmula de recurrencia se obtiene a partir de la definición de derivada de una función:

$$f'(x_i) = \frac{f(x) - f(x_i)}{x - x_i}$$

De aquí se obtiene la ecuación de la recta tangente a $f(x)$ en $x = x_i$:

$$t(x) = f'(x_i) \cdot (x - x_i) + f(x_i)$$

Tal como puede verse en la figura 1, dicha recta tangente corta el eje de abscisas en el punto $x = x_{i+1}$ donde:

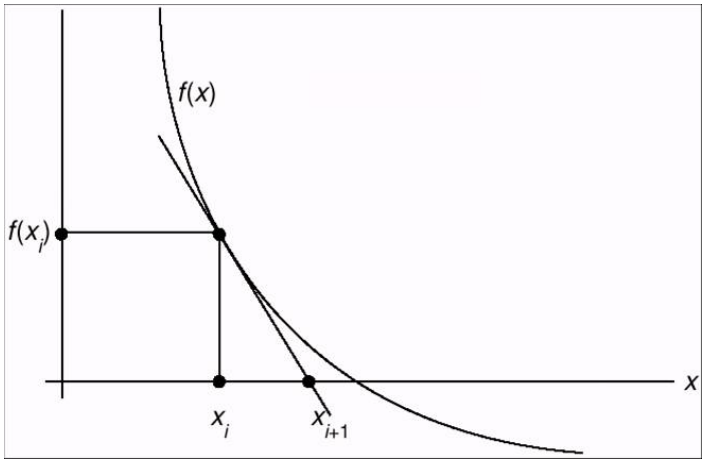

Figura 1: Función $f(x)$ y su recta tangente.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1}$$

La ecuación 1 es la base del método. Para calcular la solución de una función $f(x)$, se tomará inicialmente un valor semilla $x = x_0$ suficientemente próxima a la raíz buscada, y que constituirá la primera aproximación a la solución. A continuación, se aplicará la ecuación 1 para obtener una nueva aproximación $x = x_1$, repitiéndose el proceso sucesivamente hasta que el método converja a una solución $x = x_{i+1}$, cosa que detectaremos cuando sea cierta la expresión $|x_{i+1} - x_i| < \varepsilon$, donde ε es un valor positivo y muy próximo a 0, establecido a priori en función de la precisión buscada.

Se desea diseñar en ensamblador de MIPS un programa que calcule la raíz cuadrada de un número Z mediante el método de Newton-Raphson. Se partirá de la función

$$f(x) = x^2 - Z \tag{2}$$

cuya raíz es

$$x = \sqrt{Z}$$

La derivada de la función es

$$f'(x) = 2x \tag{3}$$

con lo que, sustituyendo 2 y 3 en 1 se obtiene la siguiente ecuación de recurrencia:

$$x_{i+1} = 0,5 \cdot \left(x_i + \frac{Z}{x_i} \right) \tag{4}$$

A continuación se presenta una posible solución en Pascal al problema pedido:

```
PROGRAM raiz_cuadrada;

VAR
   z, raiz_z, epsilon, xi, xi_1: REAL;

BEGIN {Programa principal}
   ...
   xi_1 = 1; {Primera aproximacion a la solucion}
   REPEAT
      xi := xi_1;
      xi_1 := 0.5*(xi-z/xi);
   UNTIL (abs(xi_1-xi) < epsilon);
   raiz_z := xi_1;
   ...
END.
```

Solución

Una solución al ejercicio podría ser la siguiente:

```
######################
#
# Programa que calcula la raiz cuadrada de un numero
# mediante el metodo de Newton-Raphson
#
######################

######################
#
# Seccion de datos del programa principal
#
######################
            .data
z:          .space  4               # Se usara $f22
raiz_z:     .space  4               # Se usara $f24
epsilon:    .space  4               # Se usara $f23
xi:         .space  4               # Se usara $f21
xi_1:       .space  4               # Se usara $f20

######################
#
# Seccion de codigo
#
######################
            .text
main:
# Codigo previo del programa principal ...
```

```
# Seria preciso iniciar las variables z y epsilon
             ...
# Codigo del enunciado
# xi_1 = 1; {Primera aproximacion a la solucion}
             li      $t0,1
             mtc1    $t0,$f4
             cvt.s.w $f20,$f4
# REPEAT
repeat:
# xi := xi_1;
             mov.s   $f21,$f20
# xi_1 := 0.5*(xi+z/xi);
# $f4 <- 0.5
             li      $t0,0x3f000000  # 0x3f000000 es 0.5 en IEEE754 32 bits hex.
             mtc1    $t0,$f4
# $f5 <- z/xi
             l.s     $f22,z
             div.s   $f5,$f22,$f21
# $f6 <- xi+z/xi
             add.s   $f6,$f21,$f5
# xi_1 <- 0.5*(xi+z/xi)
             mul.s   $f20,$f4,$f6
# UNTIL (abs(xi_1-xi) < epsilon);
             sub.s   $f7,$f20,$f21    # $f7 <- xi_1-xi
             abs.s   $f8,$f7          # $f8 <- abs(xi_1-xi)
             l.s     $f23,epsilon
             c.lt.s  $f8,$f23
             bc1f    repeat
fin_repeat:
# raiz_z := xi_1;
             mov.s   $f24,$f20
# Almacenamiento de variables
             s.s     $f20,xi_1
             s.s     $f21,xi
             s.s     $f24,raiz_z
# Resto del codigo del programa principal ...
             ...
# END.
# Secuencia de final de programa
             li      $v0,10
             syscall
```

Se propone al alumno que realice optimizaciones en el código con objeto de incrementar la velocidad de ejecución de la solución.