Arquitectura de Computadores
543.426

Ayudante: Antonio Saavedra

Ayudantía No. 2
17 de abril de 2017

Problema 0

El máximo común divisor entre dos números enteros puede ser calculada numéricamente a partir de las siguientes ecuaciones:

$$gcd(x, y) = x, \quad y = 0$$
$$gcd(x, y) = gcd(y, x\,\%y), \quad x >= y, \quad y > 0$$

El operador % denota el resto de la división entera entre los dos números. Escriba una función en assembly MIPS que realice el cálculo de manera recursiva a partir de estas ecuaciones.

Solución

```
1  GCD: bnez $a1, L01
2       move $v0, $a0
3       jr   $ra
4
5  L01: sw   $fp, -4($sp)
6       addi $fp, $sp, -4
7       sw   $ra, -4($fp)
8       addi $sp, $fp, -4
9
10      div  $a0, $a1
11      move $a0, $a1
12      mfhi $a1
13      jal GCD
14
15 L02: addi $sp, $fp, -4
16      lw   $ra, -4($fp)
17      lw   $fp, 0($fp)
18      jr   $ra
```

Problema 1

Esta función calcula el módulo entre dos números (operación % en C), de forma recursiva.

```
int modulo(int m, int n)
{
  if (m<n) return m;
  else return modulo (m-n, n);
}
```

Escriban una función recursiva en assembly MIPS que implemente esta función de C. Utilicen memoria en stacks.

Solución

```
MODULO: bge $a0, $a1, ELSE
        move $v0, $a0
        jr $ra

ELSE:   sw   $fp, -4($sp)
        addi $fp, $sp, -4
        sw $ra, -4($fp)
        addi $sp, $fp, -4
        sub $a0, $a0, $a1
        jal MODULO

        addi $sp, $fp, 4
        lw $ra, -4($fp)
        lw $fp, 0($fp)
        jr $ra
```

Problema 2

La función de Ackermann toma dos números como argumentos y devuelve un único número natural. Se define como sigue:

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \wedge n = 0; \\ A(m-1, A(m,n-1)), & \text{si } m > 0 \wedge n > 0; \end{cases} \tag{1}$$

La implementación en C de la definición anterior es la siguiente:

```
int Ackermann(int m, int n)
{
  if (m == 0) return n+1;
  else if (n == 0) return Ackermann(m-1, 1);
  else Ackermann (m-1, Ackermann(m, n-1));
}
```

Solución

```
ACKER:  bne $a0,$zero,ELSE_IF
        addi $v0,$a1,1
        jr $ra

ELSE_IF:sw $fp,-4($sp)
        addi $fp,$sp,-4
        sw $ra,-4($fp)
        sw $a0,-8($fp)
        addi $sp,$fp,-8
        bne $a1,$zero,ELSE
        addi $a0,$a0,-1
        li $a1,1
        jal ACKER
        addi $sp,$fp,4
        lw $ra,-4($fp)
        lw $fp,0($fp)
        jr $ra

ELSE:   addi $a1,$a1,-1
        jal ACKER
        move $a1,$v0
        lw $a0,-8($fp)
        addi $a0,$a0,-1
        jal ACKER
        addi $sp,$fp,4
        lw $ra,-4($fp)
        lw $fp,0($fp)
        jr $ra
```

Problema 3

Los números Catalán se definen de la siguiente forma:

$$C_0 = 1; \tag{2}$$

$$C_{n+1} = [(4n+2) \cdot C_n]/(n+2) \tag{3}$$

Esto se puede implementar en C de la siguiente manera:

```
unsigned int catalan(unsigned int n)
{
  if (n==0)
    return 1;
  else return ((4*n-2)*catalan(n-1))/(n+1);
}
```

Escriba una función en assembly MIPS que calcule el n-ésimo número Catalán.

Solución

```
CATALAN: bne   $a0, $zero, L1
         li    $v0, 1
         jr    $ra

L1:      sw    $fp, -4($sp)
         addi  $fp, $sp, -4
         sw    $ra, -4($fp)
         sw    $a0, -8($fp)
         addi  $sp, $fp, -8

         addiu $a0, $a0, -1
         jal   CATALAN

L2:      addi  $sp, $fp, 4
         lw    $a0, -8($fp)
         lw    $ra, -4($fp)
         lw    $fp, 0($fp)

         sll   $t0, $a0, 2
         addiu $t0, $t0, -2
         multu $t0, $v0
         mflo  $t0
         addiu $t1, $a0, 1
         divu  $t0, $t1
         mflo  $v0

         jr    $ra
```

Problema 4

La función 91 de McCarthy es una función recursiva, definida por el informático John McCarthy como una prueba de verificación formal dentro de problemas de las ciencias de la computación. Los resultados de evaluar un entero n son $n - 10$ si $n > 100$, y debería retornar luego de calculos recursivos el valor 91 si $n \leq 100$. El siguiente código C implementa la función.

```
unsigned int m91(unsigned int n)
{
  if (n>100)
    return n-10;
  else return m91(m91(n+11));
}
```

Escriba una función en assembly MIPS que implemente esta misma función.

Solución

```
M91: li    $t0, 100
          ble   $a0, $t0, L1
          addiu $v0, $a0, -10
          jr    $ra

L1:       sw    $fp, -4($sp)
          addi  $fp, $sp, -4
          sw    $ra, -4($fp)
          addi  $sp, $fp, -4

          addiu $a0, $a0, 11
          jal   M91

L2:       move  $a0, $v0
          jal   M91

L3:       addi  $sp, $fp, 4
          lw    $ra, -4($fp)
          lw    $fp, 0($fp)

          jr   $ra
```