MIPS 32

1) Ejercicio resuelto 2.13: Ordenamiento

* Procedimiento swap

```
swap: sll  $t1, $a1, 2
      add  $t1, $a0, $t1

      lw   $t0, 0($t1)
      lw   $t2, 4($t1)

      sw   $t2, 0($t1)
      sw   $t0, 4($t1)
```

} Cuerpo del Procedimiento

```
      jr   $ra
```

} retorno del procedimiento

* Procedimiento sort

```
sort: addi  $sp, $sp, -20
      sw    $ra, 16($sp)
      sw    $s3, 12($sp)
      sw    $s2, 8($sp)
      sw    $s1, 4($sp)
      sw    $s0, 0($sp)
```

} Guardar registros

```
      move  $s2, $a0
      move  $s3, $a1
```

} Copiar Parámetros

```
      move  $s0, $zero
```

CRISVI

```
for1tst: slt   $t0, $s0, $s3             } for
         beq   $t0, $zero, exit1         } exterior
         addi  $s1, $s0, -1

for2tst: slti  $t0, $s1, 0               }
         bne   $t0, $zero, exit2         }
         sll   $t1, $s1, 2               }
         add   $t2, $s2, $t1             } for
         lw    $t3, 0($t2)               } interior
         lw    $t4, 4($t2)               }
         slt   $t0, $t4, $t3             }
         beq   $t0, $zero, exit2         }
         move  $a0, $s2                  } Paso de parámetros
         move  $a1, $s1                  } y llamada
         jal   swap                      }

         addi  $s1, $s1, -1              } for
         j     for2tst                   } interior

exit2:   addi  $s0, $s0, 1               } for
         j     for1tst                   } exterior

exit1:   lw    $s0, 0($sp)               }
         lw    $s1, 4($sp)               } Restaurando
         lw    $s2, 8($sp)               } Registros
         lw    $s3, 12($sp)              }
         lw    $ra, 16($sp)              }
         addi  $sp, $sp, 20              }

         jr    $ra                       } Retorno del procedimiento
```

2) Ejercicio resuelto 2.14: Tablas frente a punteros

* Inicializar una tabla a ceros utilizando índices:

```
        move    $t0, $zero

loop1:  sll     $t1, $t0, 2
        add     $t2, $a0, $t1
        sw      $zero, 0($t2)
        addi    $t0, $t0, 1
        slt     $t3, $t0, $a1
        bne     $t3, $zero, loop1
```

* Inicializar una tabla a ceros utilizando punteros

```
        move    $t0, $a0

loop2:  sw      $zero, 0($t0)
        addi    $t0, $t0, 4
        add     $t1, $a1, $a1
        add     $t1, $t1, $t1
        add     $t2, $a0, $t1
        slt     $t3, $t0, $t2
        bne     $t3, $zero, loop2
```

CRISVI

* El programa anterior calcula la dirección del final de la tabla en cada iteración del for, a pesar de que no cambia. Mostramos una versión más rápida que traslada este cálculo fuera del for:

```
        move    $t0, $a0
        sll     $t1, $a1, 2
        add     $t2, $a0, $t1

loop2:  sw      $zero, 0($t0)
        addi    $t0, $t0, 4
        slt     $t3, $t0, $t2
        bne     $t3, $zero, loop2
```

3) Ejercicios propuestos:

a)
```
int compare(int a, int b) {
    if( sub(a,b) >=0)
        return 1;
    else
        return 0;
}

int sub(int a, int b){
    return a-b;
}
```

b)
```
int fib_iter(int a, int b, int n) {
    if( n==0)
        return b;
    else
        return fib_iter(a+b, a, n-1);
}
```

* 2.19.1 : Implemente el código de la tabla en ensamblador de MIPS. ¿ Cuántas instrucciones MIPS se necesitan para ejecutar la función

* Continuación en la siguiente hoja

a)
```
compare: subi   $sp, $sp, 12

         sw     $ra, 0($sp)
         sw     $a0, 4($sp)
         sw     $a1, 8($sp)

         jal    sub

         addi   $t0, $v0, 1

         sgt    $v0, $t0, $zero

         lw     $ra, 0($sp)
         lw     $a0, 4($sp)
         lw     $a1, 8($sp)

         addi   $sp, $sp, 12

         addi   $t0, $zero, $zero

         jr     $ra


sub:     sub    $v0, $a0, $a1

         jr     $ra
```

CRISVI

b)
```
fib_iter: subi  $sp, $sp, 16

          sw    $ra, 0($sp)
          sw    $a0, 4($sp)
          sw    $a1, 8($sp)
          sw    $a2, 12($sp)

    recursion: beq  $a2, $zero, base

               add  $t0, $zero, $a0    # aux = A
               add  $a0, $t0, $a1      # A = aux + B
               add  $a1, $zero, $t0    # B = aux

               subi $a2, $a2, 1        # n = n-1

               jal  recursion

          lw    $ra, 0($sp)
          lw    $a0, 4($sp)
          lw    $a1, 8($sp)
          lw    $a2, 12($sp)
          addi  $sp, $sp, 16
          jr    $ra

base: add  $v1, $a1, $zero

      jr   $ra
```

*2.19.2: Los compiladores a menudo implementan las funciones in-line, es decir, el cuerpo de la función se copia en el espacio del programa, eliminando el sobrecoste de la llamada. Implemente una versión in-line de la función de la tabla. ¿Cuál es la reducción del número de instrucciones MIPS necesarias para completar la función? Suponga que el valor inicial de n es 5

a)
```
compare: sub  $t0, $a0, $a1
         bgez $t0, true
         li   $v0, 0
         jr   $ra

true: li $v0, 1
      jr $ra
```

*Al hacerlo de manera in-line me estaría ahorrando 3 instrucciones MIPS sin contar el Guardado y carga de memoria

CRISVI

Escaneado con CamScanner

b)
```
fib_iter: beq   $t2, $zero, base     # Compara n con cero
          add   $t1, $t0, $t1        # a+b
          add   $t0, $t1, $zero      # a = a+b
          sub   $t2, $t2, 1          # n-1

          j fib_iter

base: move  $v0, $t1
      jr    $ra
```

* Al hacerlo de manera in-line me estaría ahorrando, al menos 5 instrucciones MIPS

CRISVI

- Algoritmo de ordenamiento de burbuja

```
*bubblesort: la   $t0, array
             lw   $t1, size
             subi $t1, $t1, 1

outer_loop: move $t2, $zero
            move $t3, $zero

      inner_loop: blt $t3, $t1, compare
                  j   end_inner_loop

            compare: sll  $t4, $t3, 2
                     add  $t4, $t0, $t4
                     lw   $t5, 0($t4)
                     lw   $t6, 4($t4)

                     ble  $t5, $t6, no_swap

                     sw   $t6, 0($t4)
                     sw   $t5, 4($t4)
                     addi $t2, $t2, 1

         no_swap: addi $t3, $t3, 1
                  j    inner_loop

    end_inner_loop: bgtz $t2, outer_loop
```

CRISVI

Escaneado con CamScanner

```
*void bubblasort(int array[ ], int n){
   int i, j, temp;

   for(   i=0   ;   i < n - 1 ; ++i   ){
     for( j=0 ; j< n-i - 1; ++i){
       if(array[j] > array[ j+1 ]){

         temp = array[ j ];
         array [j] = array[ j + 1];
         array[ j ] = temp;
       }
     }
   }
}
```

- Algoritmo de insertion sort

```
*insertionsort: la      $a0, array
                addi    $a1, $0, 5      #base is $0
                addi    $t0, $0, 1      # i=1

     outer_loop:  slt     $t3, $t0, $a1   #size is $a1
                  beq     $t3, $0, exit
                  sll     $t4, $t0, 2     #i*4
                  add     $t4, $t4, $a0
                  lw      $t5, 0($t4)  #t5 = a[i]
                  addi    $t1, $t0, -1    # j=t1=i-1

     inner_loop:  slt     $t4, $t1, $0
                  bne     $t4, $0, exitinnerloop
                  sll     $t4, $t1, 2
                  add     $t4, $t4, $a0
                  lw      $t4, 0($t4)   # a[j]
                  slt     $t6, $t5, $t4
                  beq     $t6, $0, exitinnerloop
                  addi    $t6, $t1, 1
                  sll     $t6, $t6, 2
                  add     $t6, $t6, $a0
                  sw      $t4, 0($t6)
                  addi    $t1, $t1, -1
                  j       inner_loop
```

CRISVI

Escaneado con CamScanner

```
exitinnerloop: addi   $t6, $t1, 1
               sll    $t7, $t6, 2
               add    $t7, $t7, $a0
               sw     $t5, 0($t7)
               addi   $t0, $t0, 1
               j      outer_loop

exit: jr   $ra
```

```
*void insertionsort(int a[], int size) {
    int i, j;
    for(i=1; i < size; i++) {
        int value = a[i];
        for(j=i-1; j>=0 && a[j] > value; j--) {
            a[j+1] = a[j];
        }
        a[j+1] = value;
    }
}
```