



Facultad de Ingeniería

Mattius Alexander Cardona Franco

Daniel Felipe Rivera Arroyave

Tercer informe de laboratorio

2508355-

Arquitectura de computadores y laboratorio

Fredy Alexander Rivera Velez

2018-1

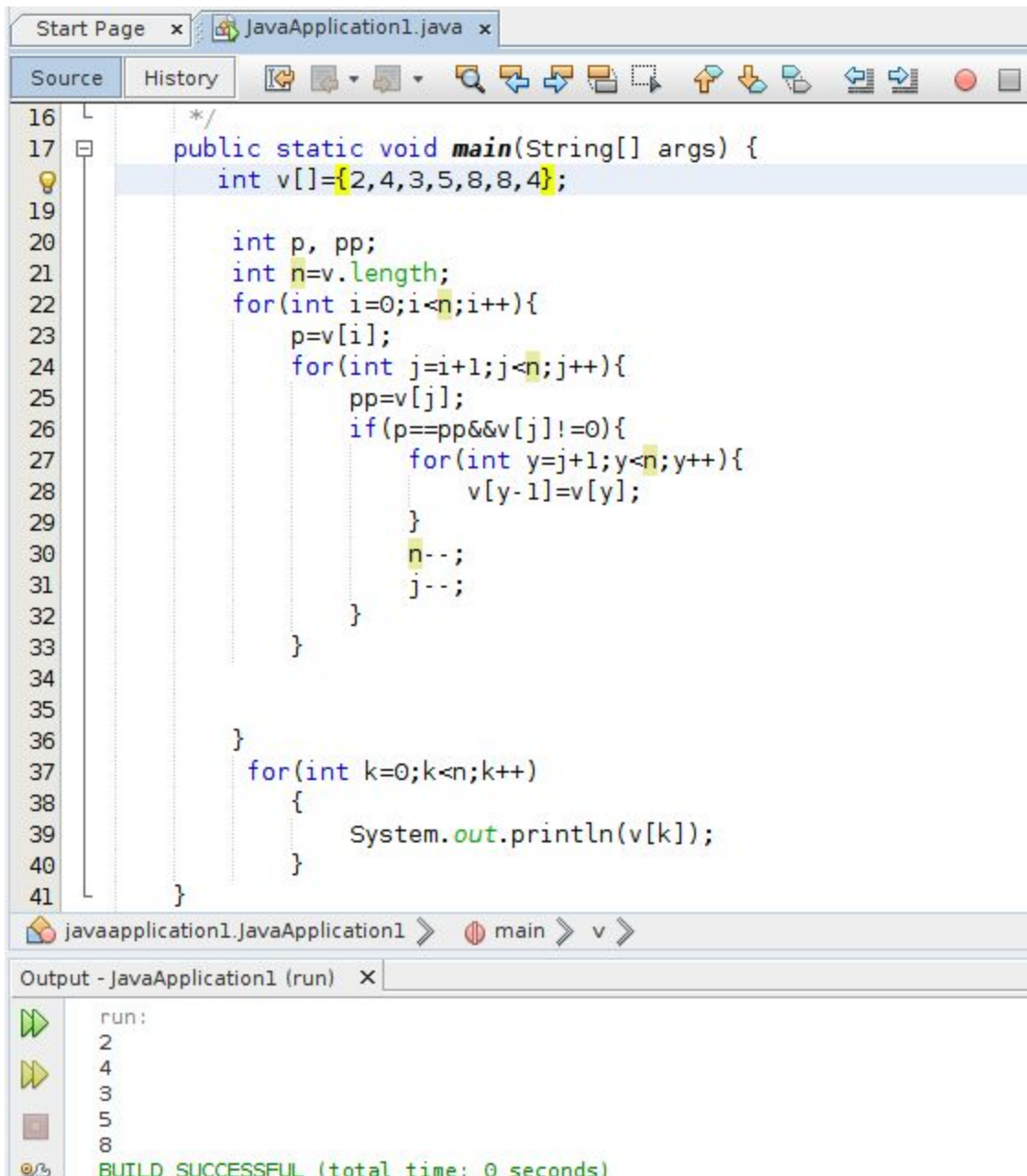
INTRODUCCIÓN

Para el desarrollo de este laboratorio, se pidió realizar en alto nivel (C++, C#, Java, etc.) o pseudo-código de un algoritmo , para luego traducirlos a un lenguaje bajo, que ayuda a entender como el procesador entiende y realiza las instrucciones dadas en el lenguaje de alto nivel, para el desarrollo de este laboratorio se utilizó la herramienta de simulación llamada MARS.

El Algoritmo a desarrollar fue :

“Remueva todos los elementos duplicados de un arreglo desordenado de n elementos y contraiga el arreglo consecuentemente.”

Representación en alto nivel



```
16  */
17  public static void main(String[] args) {
18      int v[]={2,4,3,5,8,8,4};
19
20      int p, pp;
21      int n=v.length;
22      for(int i=0;i<n;i++){
23          p=v[i];
24          for(int j=i+1;j<n;j++){
25              pp=v[j];
26              if(p==pp&&v[j]!=0){
27                  for(int y=j+1;y<n;y++){
28                      v[y-1]=v[y];
29                  }
30                  n--;
31                  j--;
32              }
33          }
34      }
35
36      for(int k=0;k<n;k++)
37      {
38          System.out.println(v[k]);
39      }
40  }
41  }
```

javaapplication1.JavaApplication1 > main > v >

Output - JavaApplication1 (run) x

run:
2
4
3
5
8
BUILD SUCCESSFUL (total time: 0 seconds)

La lógica implementada para este algoritmo es bastante simple: primero se genera un vector random (en alto nivel hemos optado por ingresarlo nosotros y asegurarnos que hayan números repetidos) y en una variable guardamos el tamaño del vector (en bajo nivel es un apuntador hacia el último elemento del vector), utilizamos dos apuntadores para recorrer el vector en 2 ciclos For: un apuntador se planta en una posición y el otro apuntador recorre el vector buscando si hay un elemento igual al del primer apuntador . si encuentra algún elemento igual va a entrar a un tercer ciclo for que sirve para mover a la izquierda todos los elementos que hay después del elemento repetido y luego restar una unidad a la cantidad de elementos(restar al apuntador del último elemento) , y seguir

moviéndose en los ciclos iniciales hasta que terminen todos los elementos comparados con todos.

Posteriormente imprime el vector final después de los cambios realizados.

Explicación código en mars mips y paso a paso de la ejecución.

```
.text
la $s1,vector          #S1=primera posición del vector
addi $v0,$zero,4        # Se le suma 4 al registro $v0 para que el sistema muestre un mensaje
la $a0, primerMenu     # Cargar al registro $a0 el mensaje para mostrar
syscall                #llamada al sistema

addi $v0,$zero,5        # Se le suma 5 al registro $v0 para que el sistema se prepare para recibir un
syscall                #llamada al sistema

addi $s0,$v0,0          #Guarda el tamaño del vector a crear en s0
mul $s0, $s0, 4         #Multiplica s0 x 4 para tener la posición final del vector
add $s0, $s0, $s1       #Apunta con s0 a la posición final del vector
jal crear
```

Se declara la posición donde inicia el vector en el registro \$S1 y en pantalla se imprime un mensaje donde pregunta al usuario cual es el tamaño del vector a generar, y realiza un cálculo para obtener la posición de memoria del último elemento para guardarlo en el registro \$S0; teniendo así a \$S1 como posición inicial y a \$S0 como posición final.

```
crear:
addi $t0,$zero,0
addi $t1,$s1,0          #i=0
                        #Se lleva a t1 la posición inicial del vector

mientras:
    bge $t1,$s0,imprimir #While
    li $a1, 10           #if((Vector en la posición t1)=(Posición final)) salta a imprimir
    li $v0, 42           #Establece el rango del random
    syscall              #genera un número random
                        #llamada al sistema

    sw $a0,vector($t0)   #a0=vector[i]
    addi $t0,$t0,4        #i++
    addi $t1, $t1, 4      #lleva a t1 la siguiente posición en el vector
    j mientras           #reinicia el ciclo
```

Esta sección de código se encarga de generar los números random y guardarlos , haciendo copia del apuntador del primer elemento, e ir sumando 4 para avanzar en posiciones de memoria hasta llegar al último elemento.

En el registro\$a1 se guarda el rango máximo del aleatorio y con la función #42 del syscall hace que se genere el número random y se guarde en \$a0 y luego guardarlo en memoria.

```

imprimir:
    la $t0, vector                                #Carga en el registro t0 la posición inicial del vector

    addi $v0, $zero, 4                            # Se le suma 4 al registro $v0 para que el sistema muestre un mensaje
    la $a0, vecListo                              # Cargar al registro $a0 el mensaje para mostrar
    syscall                                        #llamada al sistema

loop:
    bge $t0, $s0, corregir                        #if((Vector en la posición t0)=(Posición final)) salta a final
    addi $v0, $zero, 1                            # Se le indica al Sistema que se preparó para imprimir un número en pantalla
    lw $a0, ($t0)                                # Se carga en el registro $a0 el valor a mostrar
    syscall                                        # Llamada al Sistema

    addi $v0, $zero, 4                            # Se le suma 4 al registro $v0 para que el sistema muestre un mensaje
    la $a0, espacio                              # Cargar al registro $a0 el mensaje para mostrar, en este caso para generar
    syscall                                        # Llamada al Sistema

    addi $t0, $t0, 4                              #lleva a t0 la siguiente posición en el vector
    j loop                                        #Reinicia el ciclo

```

Una vez que el vector ha sido generado aleatoriamente es necesario que el usuario sepa cuál es el vector generado para tener una vista previa inicial.

Manejamos en el registro \$t0 un apuntador a cada elemento del vector y comienza un ciclo que con la instrucción 1 del syscall nos imprime un número en consola.

```

Por favor ingrese el tamaño del vector a generar: 10
El vector es: 1 - 3 - 5 - 0 - 9 - 7 - 7 - 3 - 9 - 0 -

```

Los fragmentos de código presentados anteriormente sirven principalmente para la interacción del usuario(preguntar, leer e imprimir los datos), el fragmento principal del algoritmo y encargado de cumplir el objetivo final es:

```

corregir:
la $t0, vector
for1:
    bge $t0, $s0, imprimir2
    lw $t1, ($t0)
    addi $t3, $t0, 4
    for2:
        bge $t3, $s0, endFor2
        lw $t4, ($t3)
        beq $t1, $t4, if
        j endIf
        if:
            addi $t5, $t3, 0
            subi $s2, $s0, 4
            for3:
                bge $t5, $s2, endFor3
                lw $t6, 4($t5)
                sw $t6, 0($t5)
                addi $t5, $t5, 4
                j for3
            endFor3:
                subi $s0, $s0, 4
                subi $t3, $t3, 4
            endIf:
                addi $t3, $t3, 4
            j for2
        endFor2:
            addi $t0, $t0, 4
        j for1
    endFor1:

```

```

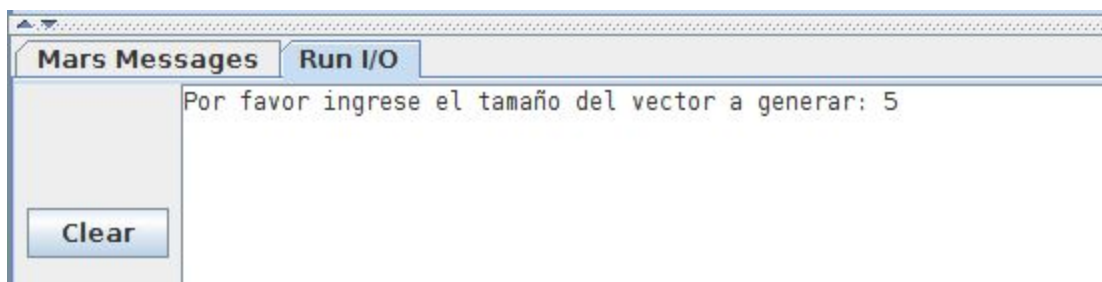
#Carga al registro t0 la primera posición del vector(i=0)
#if((Vector en la posición t0)=(posición final)) salta a imprimir2
#Carga a t1 el dato del vector en la posición t0
#j=i+1
#if((Vector en la posición t3)=(posición final)) salta a enFor2
#Carga a t4 el dato del vector en la posición t3
#
#
#
#siempre que y sea menor que s3 y<n-1
#t6 = y+1
# v[3]=y+1
#aumenta en 1 el apuntador y , y++

```

Esta es la traducción a bajo nivel del algoritmo desarrollado en java y presentado en la primera figura.

Simulación exhaustiva:

1)Pregunta por el tamaño del vector a generar.



2) Guardar en memoria los elementos random e imprimirlos.

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	543966474	1952671094	1663070831	1701999215	1868851559	980641056	1867513888	1634082930
268501024	544370550	1919381097	543519589	1948281957	-245273247	1701060719	1702240364	1919906915
268501056	1730175264	1919250021	540701281	543966464	1952671094	1696625263	2112115	2108704
268501088	5	3	8	8	3	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0

0x10010000 (.data) ☐ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Mars Messages **Run I/O**

Por favor ingrese el tamaño del vector a generar: 5
El vector es: 5 - 3 - 8 - 8 - 3 -

3) Recorrer el vector y lo corrige

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	543966474	1952671094	1663070831	1701999215	1868851559	980641056	1867513888	1634082930
268501024	544370550	1919381097	543519589	1948281957	-245273247	1701060719	1702240364	1919906915
268501056	1730175264	1919250021	540701281	543966464	1952671094	1696625263	2112115	2108704
268501088	5	3	8	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0

0x10010000 (.data) ☐ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Mars Messages **Run I/O**

Por favor ingrese el tamaño del vector a generar: 5
El vector es: 5 - 3 - 8 - 8 - 3 -

4) Recorrer el arreglo corregido e imprimirlo nuevamente con los cambios realizados.

```

Por favor ingrese el tamaño del vector a generar: 5
El vector es: 5 - 3 - 8 - 8 - 3 -
El vector corregido es: 5 - 3 - 8 -
-- program is finished running (dropped off bottom) --

```

Decisiones tomadas

- El rango de los números es hasta el 10 para que si el usuario ingrese la cantidad de elementos sea fácil de identificar cuales son repetidos y se pueda comparar el vector original con el final modificado.
- El tamaño del vector no puede superar la cantidad de posiciones de memoria disponibles.
- Para evitar el uso complejo del stack pointer, sólo se maneja un apuntador al final del vector y uno al inicio, cuando es necesario reducirlo se mueve el último apuntador y así evitamos el uso de un algoritmo recursivo.
- La convención de registros utilizada

Dificultades encontradas:

- En varios segmentos de código nos fue difícil comprender cuál era el tipo de registro adecuado y sin agotarlos para poder evitar el uso del stack.
- El error más simple pero arduo de encontrar se presentó al momento de imprimir un " "(espacio) entre los elementos del vector, que no se imprimía correctamente, la solución fue tan simple como cambiar el orden de línea de código. La palabra de espacio se cargaba en una posición de memoria siguiente a la de donde se cargaba la primera vez el vector, cuando se comenzaba a llenar dicho vector, esta palabra era sobre escrita y lo que imprimía era la representación en ASCII del número en esa posición de memoria.

Conclusiones:

- Es posible concluir que la implementación de bajo nivel se torna un poco más tedioso a comparación de alto nivel ya que las instrucciones básicas del alto nivel se requieren de más líneas de código en el bajo.
- Es imprescindible destacar la importancia de aprender a trabajar de forma básica como mínimo en lenguaje ensamblador y comprender que todos los lenguajes de programación hacen su respectiva traducción a este antes de convertirse en código máquina.
- En caso de no ser muy habilidoso en el desarrollo de lenguaje ensamblador, es altamente recomendable tener las instrucciones hechas en alto nivel o en

pseudo-código para facilitar el trabajo, ir desarrollando secuencialmente el algoritmo y resolviendo las dificultades una por una a la vez.