Trabajo Práctico Nº 7
LOS LENGUAJES Y LA MÁQUINA

1. Crear una tabla de símbolos para el segmento de programa ARC que se muestra a continuación. Utilizar U para los símbolos indefinidos.

```
X           .equ 4000
            org 2048
            ba main
            .org 2072
main:       sethi x, %r2
            srl %r2, 10, %r2
lab_4:      st %r2, [k]
            addcc %r1, -1, %r1
foo:        st %r1, [k]
            andcc %r1, %r1, %r0
            beq lab_5
            jmpl %r15 + 4, %r0
cons:       .dwb 3
```

2. Traducir a código objeto el código Assembler de ARC del programa siguiente. Asumir la dirección $(4096)_{10}$ como valor inicial de x.

```
k           . equ 1024
            .
            addcc %r4 + k, %r4
            ld %r14, %r5
            addcc %r14, -1, %r14
            st %r5, [x]
            .
            .
            .
```

3. Crear una tabla de símbolos para el siguiente programa.

```
                                ! Realiza una suma en 64-bits: C ← A + B
                                ! Asignación de reg: %r1-32 bits mas significativos de A
                                ! %r2 --32 bits menos significativos de A
                                ! %r3--32 bits mas significativos de B
                                ! %r4 --32 bits menos significativos de B
                                ! %r5--32 bits mas significativos de C
                                ! %r6 --32 bits menos significativos de C
                                ! %r7 –utilizado para reponer el bit de arrastre
            .begin              !comienzo del programa
            .org 2048
main:       ld [A], %r1         ! buscar la palabra mas significativa de A
            ld [A + 4], %r2     ! buscar la palabra menos significativa de A
            ld [B], %r3         ! buscar la palabra mas significativa de B
            ld [B + 4], %r4     ! buscar la palabra menos significativa de B
            call add_64         ! realizar la suma en 64 bits
            st %r5, [C]         ! almacenar la palabra mas significativa de C
            st %r6, [C + 4]     !almacenar la palabra menos significativa de C
            .
            .
            .
            .org 3072           ! la rutina add_64 comienza en 3072
add_64      addcc %r2, %r4, %r6 ! sumas las palabras menos significativas
            bcs lo_carry        ! bifurcar si el arrastre vale 1
            addcc %r1, %r3, %r5 ! Sumar las palabras mas significativas
            jmpl %r15 + 4, %r0  ! volver al programa principal
lo-carry:   addcc %r1, %r3, % r5 ! Sumar las palabras mas significativas
            bcs hi_carry        ! bifurcar si el arrastre vale 1
```

```
            addcc %r5, 1, %r5          ! sumar el arrastre
            jmpl %r15 + 4, %r0         ! volver al programa principal
hi_carry:   addcc %r5, 1, %r5 ! sumar el arrastre
            sethi #3FFFFF, %r7         ! preparar %r7 para el arrastre
            addcc %r7, %r7, %r0        ! generar arrastre
            jmpl %r15 + 4, %r0         ! volver al programa principal
A:          0                          ! 32 bits mas significativos de 25
25                                     ! 32 bits menos significativos de 25
B:          #FFFFFFFF                  ! 32 bits mas significativos de -1
            #FFFFFFFF                  ! 32 bits menos significativos de –1
C:          0                          ! 32 bits mas significativos del resultado
            0                          ! 32 bits menos significativos del resultado
            .end                       ! fin de la traducción
```

4. Traducir a código objeto la subrutina *add_64* del problema anterior, incluyendo las variables A, B y C

5. Un desensamblador es un programa que lee un modulo objeto y recrea el modulo fuente en lenguaje simbólico. Dado el siguiente código objeto, desensamblarlo para obtener las sentencias correspondientes del lenguaje simbólico ARC. Dado que el código objeto no contiene información suficiente para determinar los nombres de los símbolos, se les asignarán ordenadamente las letras del abecedario a medida que sean necesarias.

```
1000 0010 1000 0000 0110 0000 0000 0001
1000 0000 1001 0001 0100 0000 0000 0110
0000 0010 1000 0000 0000 0000 0000 0011
1000 1101 0011 0001 1010 0000 0000 1010
0001 0000 1011 1111 1111 1111 1111 1100
1000 0001 1100 0011 1110 0000 0000 0100
```

6. Dadas las siguientes macros *push* y *pop*, si en un programa se utiliza *push* inmediatamente después de *pop* pueden aparecer instrucciones innecesarias. Expandir las definiciones de las macroinstrucciones mostradas e identificar las instrucciones innecesarias.

```
.begin
.macro push arg1
addcc %r14, -4, %r14
st arg1, %r14
.endmacro
.macro pop arg1
ld %r14, arg1
addcc %r14, 4, %r14
.endmacro
                                   ! comienzo del programa
.org 2048
pop %r1
push %r2
  .
  .
  .
.end
```

7. (a) Cuándo ocurre la expansión de macros, en tiempo de ensamblado o en tiempo de ejecución?
   (b) Ídem con expansión de macro recursiva

8. Suponga tener disponible para su uso la rutina *add_64* del problema 3. Se requiere escribir una rutina ARC llamada *add_128* que sume dos números de 128 bits, haciendo uso de la rutina *add_64*. Los dos operandos se encuentran almacenados en memoria, en

direcciones que empiezan en *x* e *y*, respectivamente, en tanto que el resultado se almacena en la dirección de memoria que comienza en *z*.

9. Escribir una macroinstrucción llamada *subcc* cuya utilidad sea similar a la de *addcc*, que realice la resta del primer operando origen menos el segundo.

10. Para las siguientes sentencias en C encontrar las instrucciones en lenguaje simbólico de ARC

```
for ( i = 0: i < N; i++) sentencia
while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
```

11. Encontrar las instrucciones en lenguaje simbólico de ARC que implementen el siguiente código escrito en lenguaje Pascal

```
 /* Calcula el mínimo de i   j */
function min (i, j: Int): Int
 var m: Int;
   begin
     if i < j then m := i else m := j;
   min := m
 end;
.
```