66.70 Estructura del Computador
TP 0
Lenguaje simbólico y simulador de ARC

A. Ejecutar el simulador de ARC

1. El simulador de ARC esta en el archivo ARCToolsv2.0.3.jar. Tiene que estar instalado JAVA, sino se obtiene por la red.
2. Debe estar instalado en las PC de los laboratorios. El icono debe estar disponible.
3. El simulador se puede utilizar libremente.
4. Luego de activar el simulador aparece una ventana. En la parte superior el contador de programa, los indicadores (flags) y los contenidos de los registros del procesador. Luego los botones de comandos: **exit**, **print**, **load**, **reload**, **edit**, **step**, **run**, **stop**, **clear regfile**, **clear breakpts** y **clear memory**; que se utilizan para las distintas operaciones del simulador. También una serie de contenidos que permiten el seguimiento de la ejecución de programas.

B. Sumar dos enteros

1. Para poder utilizar el simulador debe primero ingresarse el código fuente del programa. Se cargará el siguiente programa:

```
!este programa suma dos números
.begin
.org 2048
ld [x], %r1 !cargar x en %r1
ld [y], %r2 !cargar y en %r2
addcc %r1, %r2, %r3 ! %r3 ← %r1 + %r2
st %r3, [z] ! guardar %r3 en z
halt ! parar
x: 15
y: 9
z: 0
.end
```

2. Activar el botón **edit** y se abrirá otra ventana. Activar **File** y ***New***.
Se pueden ingresar con el editor las instrucciones del programa.
Solo esta activo el botón **Assemble**.
3. Al activarlo si hay algún error, como haberse olvidado el .begin aparece:
ARC Parser (ARCTools Version 2.0.3)
ERROR: Syntax Error: Encountered ".end" at line 10, column 1.
.lst and .bin files will not be generated...

Si no hay errores aparece:
ARC Parser (ARCTools Version 2.0.3)
Program assembled successfully.

Y se activan los botones **Show Asm File**, **Show Lst File**, **Show Binary File** y **Bin->Sim**

El **Show Lst File**:
(ARCTools Version 2.0.3)

```
 HexLoc    DecLoc    MachWord   Label      Instruction             Comment
                                .org 2048
00000800  0000002048  c2002814            ld [2068], %r1          ! x en %r1
00000804  0000002052  c4002818            ld [2072], %r2          ! cargar y en %r2
00000808  0000002056  86804002            addcc %r1, %r2, %r3       ! %r3  %r1 + %r2
0000080c  0000002060  c620281c            st %r3, [2076]          ! guardar %r3 en z
00000810  0000002064  ffffffff          halt                  ! parar
00000814  0000002068  0000000f  x:
00000818  0000002072  00000009  y:
0000081c  0000002076  00000000  z:

--- Symbol Table ---
x: 2068
z: 2076
y: 2072
```

El **Show Binary File**

```
00000800
00000800      c2002814
00000804      c4002818
00000808      86804002
0000080c      c620281c
00000810      ffffffff
00000814      0000000f
00000818      00000009
0000081c      00000000
```

4. El archivo fuente puede guardarse con **File** -> ***Save as***, puede enviarse a un diskette, la extensión .asm se agrega automáticamente.

5. El botón **Bin->Sim** carga todo en el simulador, ver el resultado en la otra ventana.

6.Se puede ejecutar el programa paso a paso con el botón **Step**. Notar el cambio de contenido de los registros y memoria.

7. Antes de comenzar la ejecución anotar el valor de los registros:

%r1 _____
%r2 _____
%r3 _____

8. Para observar los datos asociados a este programa ir a la parte de abajo de la pantalla, si todavía no se ejecuto nada Loc contiene 00000000, cambiarlo a la dirección del dato 814. Anotar el contenido de:

814 _____
818 _____
81C _____

9. Ejecutar el primer paso del programa. Indicar el contenido de los registros de memoria después del primer paso:

%r1 _____
%r2 _____
%r3 _____
814 _____
818 _____
81C _____

10. Continuar paso a paso y guardar los contenidos de registros y memorias

Step	**2**	**3**	**4**
%r1			
%r2			
%r3			
814			
818			
81C			

11. Si en vez de seleccionar **step** se elige **run** el programa ejecuta hasta el halt

12. Para finalizar activar el botón de **exit**

13. Para editar o cambiar un programa existente
Correr el simulador
Activar **Edit**
Seleccionar **File *Open***. Seleccionando un archivo existente con la extensión .asm.
Continuar como en los pasos anteriores.

C. Sumar cinco enteros

1. Repetir para el siguiente programa los pasos de **A**.

```
! Este programa suma longitud números
! reserva de registros: %r1 - longitud del arreglo
! %r2 - dirección de comienzo del arreglo a
! %r3 - suma parcial
! %r4 - puntero al arreglo a
! %r5 - contiene elemento de a
            .begin
```

```
            .org 2048 ! comenzar en 2048
a_comienzo      .equ 3000 ! dirección del arreglo a
            ld [longitud], %r1 ! %r1 ← longitud del arreglo a
            ld [direccion],%r2 ! %r2 ← dirección de a
            andcc %r3, %r0, %r3 ! %r3 ← 0
loop:       andcc %r1, %r1, %r0 ! cuantos elementos quedan?
be          done ! terminar cuando longitud = 0
            addcc %r1, -4, %r1 ! decrementar longitud
            addcc %r1, %r2, %r4 ! dirección del siguiente
            ld %r4, %r5 ! %r5 ¬ Memoria[%r4]
            addcc %r3, %r5, %r3 ! sumar el Nuevo elemento a r3
ba          loop ! repetir lazo
done:       halt ! terminar
longitud:   20 ! 5 números (20 bytes) en a
direccion:  a_comienzo
            .org    a_comienzo ! comienzo del arreglo a
a:           25 ! longitud/4 valores siguen
            -10
            33
            -5
            7
            .end
```

2. Cual es la dirección del arreglo a en hexa.
3. Antes de ejecutar el programa pronostique el valor de la suma.
4. Antes de ejecutar pronostique los contenidos de los registros 1 al 5.

Registro	**Contenidos pronosticados (ANTES)**	**Contenidos actuales (DESPUES)**
%r1		
%r2		
%r3		
%r4		
%r5		

D. Funciones lógicas AND, OR y lazos

Escribir un programa en lenguaje simbólico de ARC que:

a. sume los números $1F_{16}$ y $2C_{16}$ dejando el resultado en el registro 1
b. el resultado aplicarle la función AND con CD_{16} y colocarlo en el registro 2
c. utilizando un lazo al registro 2 sumar los números decimales del 5 al 10 y dejar el resultado en el registro 3
d. con el registro 3 hacer una OR con $0D_{16}$ y dejar el resultado en el registro 4.

E.. Escriba las sentencias en ARC para realizar las siguientes operaciones. Para algunas sentencias son necesarias mas de una instrucción.

a) Restar el contenido en %r3 del contenido de %r4 guardar el resultado en %r5. (%r5 ← %r4 - %r3)
b) Colocar el contenido del registro %r1 en la dirección de memoria (rotulo de programa, variable) llamada resultado. (resultado ← %r1)
c) Colocar el valor 17 en %r1 (%r1 ←17)
d) Empujar (push) el valor en %r2 en la pila (stack) (push(%r2))
e) Hacer el bit 2 (el tercero de la derecha) igual a 1 del numero contenido en %r4 sin cambiar el resto del numero. (Si los cinco bits menos significativos en %r4 son 10001, resultara 10101. Solo cambia el bit 2.)
f) Invertir todos los bits del numero en %r3. (Si los cinco bits menos significativos en %r3 son 10101, resultara 01010. Los 32 bits deben invertirse.)
g) Saltar a un subprograma llamado multiplicar.
h) Volver del subprograma a la línea que sigue a la sentencia que lo invoco.
i) Colocar la dirección del rótulo (variable) ***resultado*** in %r1. (%r1 ← &resultado)
j) Sacar el tope de la pila y colocarlo en %r5. (%r5 ← pop())

F. Traducir el siguiente código a lenguaje simbólico ARC. Suponer que se cargo x en %r1, y fue cargada en %r2, y z fue cargada en %r3. (Cada vez que el código utilice x, se utiliza %r1, etc.)
a)

```
while (x  >= y ) {
   y = y  + 2;
   x = x - 2;
}
z = x + y;
```

b)

```
if ( x >= 10)
        x = x - 1;
else
        y = y + 2;
z = y;
```

H.. Mostrar los contenidos hexadecimals de %r3 luego de la ejecución de las siguientes instrucciones. Recordar que los registros son de 32bits.

```
ld      [x], %r1
ld      [y], %r2
```

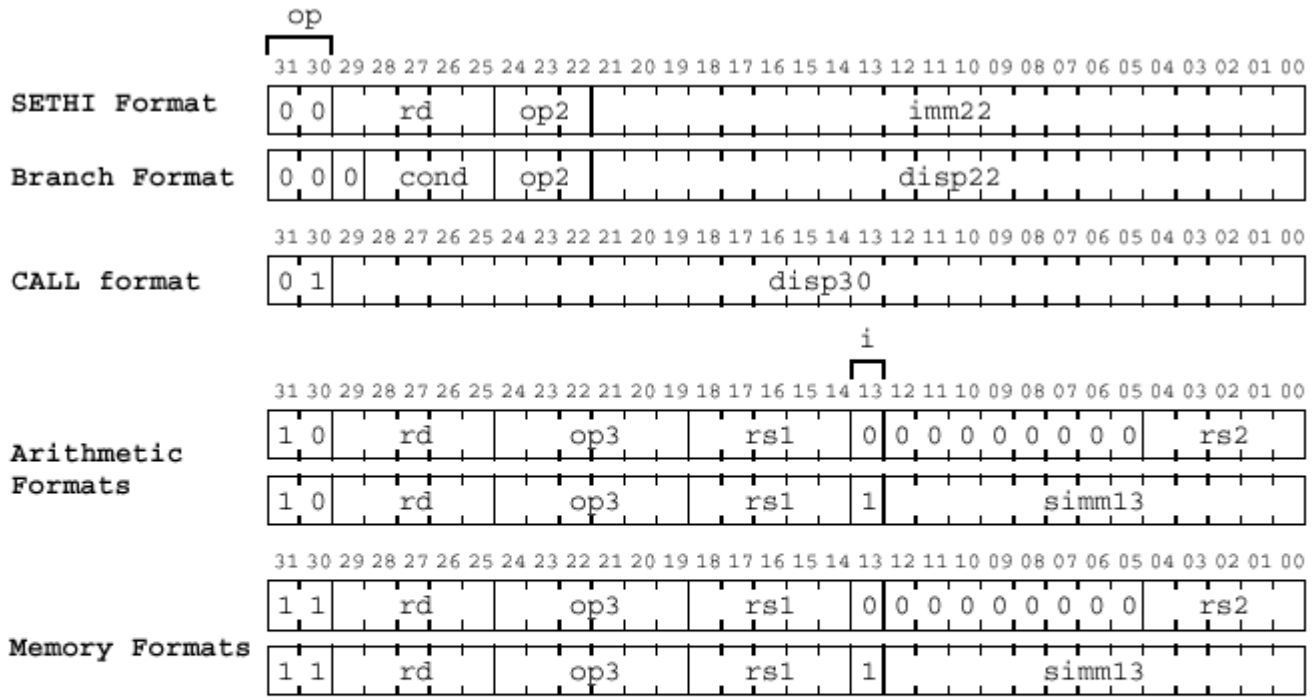
x: 0xc6
y: 0x9a

a. orcc %r1, %r2, %r3 %r3 = _____
b. xorcc %r1, %r2, %r3 %r3 = _____
c. sll %r1, 2, %r3 %r3 = _____
d. orncc %r2, %r0, %r3 %r3 = _____

I. Que procesos se realizan en la primer pasada de un two-pass assembler? .

J. Traducir las siguientes instrucciones de ARC a lenguaje de maquina.

```
HexLoc    DecLoc                Source Code
00000800  0000002048            .begin
00000800  0000002048            .org 0x800
00000800  0000002048    start:  sethi x, %r11
00000804  0000002052            srl   %r11, 10, %r12
00000808  0000002056            ld   %r12, %r13
0000080c  0000002060            addcc   %r13, 25, %r20
00000810  0000002064            orncc   %r20, %r0, %r21
00000814  0000002068            bneg start
00000818  0000002072            st   %r21, [y]
0000081c  0000002076            halt
00000820  0000002080    x:       0xa5
00000824  0000002084    y:       0
```



op	Format
00	SETHI/Branch
01	CALL
10	Arithmetic
11	Memory

op2	Inst.
010	branch
100	sethi

op3 (op=10)	
010000	addcc
010001	andcc
010010	orcc
010110	orncc
100110	srl
111000	jmpl

op3 (op=11)	
000000	ld
000100	st

cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba