

Trabajo Práctico N° 6

ARQUITECTURA DEL SET DE INSTRUCCIONES

Memoria

- 1.- Una línea de transmisión de datos comunica la computadora "A", que es "little endian", con la computadora "B", que es "big endian". A través de esta línea se transmite desde la computadora A una palabra de 32 bits que contiene el valor numérico 3. Esto se hace byte por byte de modo que en la computadora B el byte 0 se almacena en el byte 0, el byte 1 en el byte1, etc. Una vez terminada la transmisión ¿Cuál es el valor numérico leído en la máquina B?
- 2.- Un programa compilado para una arquitectura SPARC escribe en un archivo el entero sin signo de 32 bits 0xABCD01 y lo recupera correctamente. El mismo programa compilado para una arquitectura x86 (Pentium) también funciona correctamente. Sin embargo, cuando se transfiere el archivo entre máquinas, el programa de la computadora basada en Pentium lee incorrectamente el valor del archivo. ¿Cuál es el problema? ¿Cuál es el valor leído?
- 3.- El set de instrucciones ARC incluye la instrucción "ld" para leer de memoria RAM datos de 32 bits pero también ofrece instrucciones que permiten acceder a memoria para leer datos de 16 bits y de 8 bits. Notar que si bien existe una única instrucción para leer datos de 32 bits en los otros dos casos hay dos instrucciones: una para números con signo y otra números sin signo ("ldsh" y "lduh" para datos de 16 bits con y sin signo respectivamente; "lds" y "ldub" para datos de 8 bits con y sin signo respectivamente). Se pide:
 - (a) Justificar la existencia de instrucciones diferenciadas para leer números con y sin signo.
 - (b) En el caso de escritura de datos en RAM no se hace esta diferenciación entre números signados y no-signados (las instrucciones previstas para escritura en memoria son "st" "sth" y "stb" para escribir 32, 16 y 8 bits respectivamente). Justificar esta diferencia con respecto a lo observado en la operación de lectura.

Código Assembler

Declaración de variables y pasaje de parámetros de subrutina

- 4.- Escriba un programa en código ARC que declare dos variables de 32 bits en memoria RAM y que intercambie el contenido entre ellas. Utilizar el mínimo número de registros que le sea posible.
- 5.- Los tres programas siguientes tienen la misma función. Ordenarlos en función de los tiempos de ejecución esperados.

A	B	C
<pre>.begin value .equ 25 ld %r14,%r2 add %r0,value,%r1 add %r2,%r1,%r2 st %r2,%r14 jmpl %r15, 4, %r0 .end</pre>	<pre>.begin ld %r14,%r2 ld [value],%r1 add %r2,%r1,%r2 st %r2,%r14 jmpl %r15, 4, %r0 value: 25 .end</pre>	<pre>.begin ld %r14,%r2 add %r0, 25,%r1 add %r2,%r1,%r2 st %r2,%r14 jmpl %r15, 4, %r0 .end</pre>

- 6.- Un programa ARC declara dos variables y luego invoca una rutina que obtiene la suma de ambas. Escribir tres versiones del programa principal y de la rutina considerando diferentes convenciones para el pasaje de parámetros (a) por registros (b) por pila (c) por área reservada de memoria La subrutina debe ser declarada en el mismo módulo que el programa principal.
- 7.- Idem el ejercicio anterior pero declarando modulo principal y subrutina en diferentes módulos.
- 8.- Idem ejercicio 6 pero en vez declarar una subrutina realizar la suma utilizando una macro y pasaje de parámetros por registro. Comparar macros y subrutinas con respecto a (a) significado de los parámetros (b) espacio de memoria ocupada por el código de máquina (b) velocidad de ejecución.

66.70 Estructura del Computador

Operaciones aritméticas

- 9.- Proponer una subrutina que recibe por stack un número entero en complemento a 2 y devuelve su valor absoluto.
- 10.- Escribir código que recibe por la pila dos números en complemento a 2 y determina si su suma es representable en 32 bits o no. En el primer caso devuelve un 0 y en caso contrario un -1.
- 11.- Idem anterior pero considerando que recibe números enteros sin signo.
- 12.- La multiplicación no forma parte de las operaciones aritméticas incluidas en el set de instrucciones ARC. Escriba una subrutina que recibe por la pila los números a multiplicar y devuelve por la pila su producto (la multiplicación no necesariamente debe ser implementada siguiendo un método óptimo). En caso de que el producto de ambos números no pueda ser representado en 32 bits, la subrutina devuelve -1.
- 13.- Proponer una rutina que recibe un número entero en el rango de 1 a 10 y devuelve su factorial o bien devuelve -1 en caso de que el número recibido se encuentre fuera del rango indicado.
- 14.- Proponer una rutina que recibe un número entero y devuelve el cuadrado del mismo. En caso de que el resultado no pueda ser representado con 32 bits debe devolver -1.

Operaciones con bits

- 15.- Escribir código que recibe a través de %r10 un número en punto flotante y devuelve su valor absoluto (también en punto flotante).
- 16.- Escribir un programa que recibe a través de la pila dos números en punto flotante simple precisión, los compara y devuelve respectivamente un 1 o un 2 según sea mayor el primero o el segundo.
- 17.- Un programa recibe a través de %r20 una palabra de 32 bits que representa dos números en complemento a 2 en sus dos bytes más significativos y en sus dos bytes menos significativos respectivamente. Proponer un código para ese programa tal que devuelva (también por %r20) la suma de ambos valores recibidos. Considere el manejo de condiciones de fuera de rango si lo considera necesario.
- 18.- Escribir una rutina que lee del stack un valor de 32 bits, cuenta la cantidad de 1's comprendidos en su representación binaria y devuelve ese valor por stack.

Variables estructuradas

- 19.- Escribir un programa que determine la cantidad total de 1's comprendidos en las representaciones binarias de todos los valores guardados en un arreglo cuyo largo y dirección de inicio son conocidos (estos llegan a través de la pila). Se espera que el código invoque la subrutina escrita como respuesta al punto 18.
- 20.- Un programa recibe via stack la dirección de inicio y el largo de un vector y devuelve su valor máximo y su valor mínimo también por stack. El cálculo del máximo se realiza invocando una subrutina declarada en el mismo módulo mientras que el cálculo del mínimo se realiza invocando una subrutina declarada en un módulo externo. Proponer un código para el programa principal y para ambas rutinas.
- 21.- Un programa declara un array de 30 elementos de 32 bits y le pasa a una subrutina su largo y localización en memoria. Esta subrutina determina si la suma de todos los elementos del array es un número par (devuelve un 1) o impar (devuelve un 0). En caso de que la suma excediese el rango de representación del sistema la rutina devuelve -1.
El programa debe inicializar a FFFFh todos los elementos del array en caso de determinar que su suma sea par o bien poner todos en cero si su suma excede el rango de representación.
Proponer un código para el programa principal y para la subrutina considerando las variantes de que ambos están declarados en el mismo módulo o en módulos diferentes.

66.70 Estructura del Computador

Acceso a periféricos

- 22.- Un dispositivo mide simultáneamente 8 valores de temperatura y los almacena en ocho registros de 32 bits c/u los cuales se encuentran accesibles en el mapa de memoria de una computadora ARC mapeados a partir de la dirección B0101002h. Escribir una rutina que lea estos ocho valores y devuelva por stack el mayor de ellos.
- 23.- Un dispositivo de medición esta conectado a una computadora ARC y puede ser controlado a través de dos de sus registros, los cuales están mapeados en las direcciones de memoria D1=90001D00h y D2=90001D04h. El byte menos significativo de D1 refleja la medición de temperaturas realizada por el dispositivo (en el rango de 0 a 255 °C). El bit31 de D2 permite controlar el encendido de un indicador led de ese equipo (un 1 lo enciende y un 0 lo apaga). Escribir un programa que encienda el led sólo en caso de que la temperatura supere los 80 °C. Al escribir el bit31 de D2 no deben alterarse el resto de los bits de esa posición.
- 24.- El ejercicio 36 de la tira de problemas correspondiente a Algebra de Boole plantea un sistema de control que responde a la siguiente consigna:

"El sistema de control habilita la entrada de materia prima si la temperatura del horno es mayor que 300 °C y la presión es inferior a 10 atmósferas, o si se llega a la mínima concentración de sales con una temperatura mayor que 300 °C, o si, siendo la presión mayor o igual que 10 Atmósferas, no hay suficiente concentración de sales."

Diseñe un programa que implemente esta lógica en código ARC considerando que el sistema es controlado por una computadora en la cual están mapeadas las siguientes entradas/salidas:

- las lecturas de presión y temperatura están mapeadas 81000000h y 81000004h.
- la insuficiencia de sales es representada por un 1 en el bit más significativo de 8100008h.
- la entrada de materia prima se habilita con un 1 en el bit0 de la posición 81000008h.

Código de Máquina

- 25.- Las instrucciones de salto condicional dirigen el flujo de programa hacia una instrucción identificada por una etiqueta. Indicar el rango máximo del salto hacia atrás y hacia delante medido en cantidad de instrucciones y en cantidad de bytes.
- 26.- La instrucción 'sethi' asigna una constante a los 22 bits más significativos de un registro. Sin embargo, sería útil que sethi permitiera asignar valores en los 32 bits del registro. Justifique porqué esto último no resulta posible.
- 27.- Proponga tres técnicas alternativas para cargar un registro con la constante 1000 y tres para cargar la constante 50000. Debido al formato del código de máquina, no todas deberían ser factibles en ambos casos. Indique cuáles sí y cuáles no.
- 28.- Las siguientes líneas de código incluyen dos errores. Identifíquelos y proponga formas alternativas para solucionarlos

```
.begin
Dir .equ 10000
.org 2048
main: add %r0,Dir,%r1
      ld [array+4],%r2
      st %r2,%r1
      jmpl %r15+4,%r0
      .org A1010002h
array: .dwb 6
.end
```

- 29.- A continuación se presentan dos secuencias de código escrito en lenguaje C que hacen lo mismo: calcular la sumatoria de i desde 0 hasta n. El de la izquierda sigue un método recursivo mientras que el de la derecha sigue un procedimiento iterativo.

66.70 Estructura del Computador

algoritmo recursivo	algoritmo iterativo
<pre>int sumatorio (int n, int acumulado) { if (n>0) return sumatorio (n - 1, acumulado + n); else return acumulado; }</pre>	<pre>int sumatorio (int n) { int i, acumulado = 0; for (i = n; i > 0; i--) { acumulado = acumulado + n; } return acumulado; }</pre>
$\sum_{i=0}^n i = \sum_{i=0}^{n-1} i + n$	$\sum_{i=0}^n i = 0+1+2+\dots+n$

Aunque ambas secuencias funcionan correctamente y generan el mismo resultado, la iterativa tiene un mejor rendimiento. ¿Por qué? Describa cómo quedan las instrucciones en Assembler y cuál es su comportamiento en tiempo de ejecución.

- 30.- Un programa fue bajado de disco a memoria principal de una computadora ARC. En la tabla siguiente se muestra el contenido de un segmento de memoria dentro del rango ocupado por ese programa.

Dirección	Contenido
00000810	CA006BCC
00000814	CA206BB8
00000818	82206004
0000081C	02800003
00000820	80A06000
00000824	10BFFFFB

Indicar el código Assembler de ese segmento.

TRABAJO PRACTICO N°6

CÓDIGO ASSEMBLED

EJ 4

- BEGIN
- ORG 2048

ST R1, [X]

ST R2, [Y]

LD [X], R2

LD [Y], R1

EJ 5

EJ 5

EL MAS LENTO ES B, PORQUE SE
VECES A MEMORIA

3

LUEGO ENTRE A Y C, ES MAS RAPIDO C YA
QUE TIENE UNA LINEA MENOS

EJ 6

POD REGISTROS (A)

- BEGING
- ORG 2048

LD [X], R1

LD [Y], R2

CALL SUMAR

X: 10

y: 15

• END → Z: 0

} DECLARAR ES GUARDAR
EN MEMORIA

SUMAR: ADD R1, R2, R3

ST R3, [Z]

JMPL R15+4, R0

(B) POR PILA

- BEGIN
- ORG 2048

LD [X], Y.R1

LD [Y], Y.R2

%SP EQU %R14

ST Y.R1, %SP

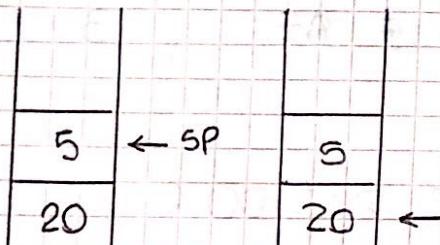
ADDCC Y.SP, -4, Y.SP

ST Y.R2, %SP

CALL SUMAR

X: 20

Y: 5



ACA SIEMPRE HAGO
PRIMEREDO -4 POR SI
YA HABIA ALGO

SUMAR: ANDCC R3, R0, R3

ST SP, R3

ADDCC SP, 4, SP

ASTCC SP, R4

ADDCC R3, R4, R3

ST R3, [Z]

JUMPL R15+4, R0

ACA FALTO
SP +4

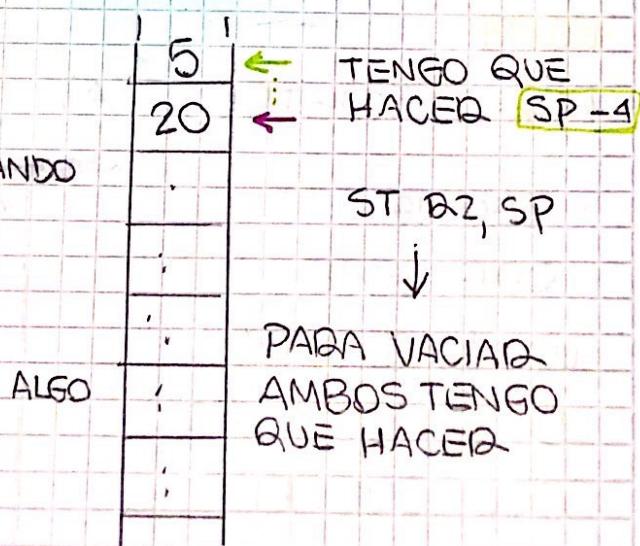
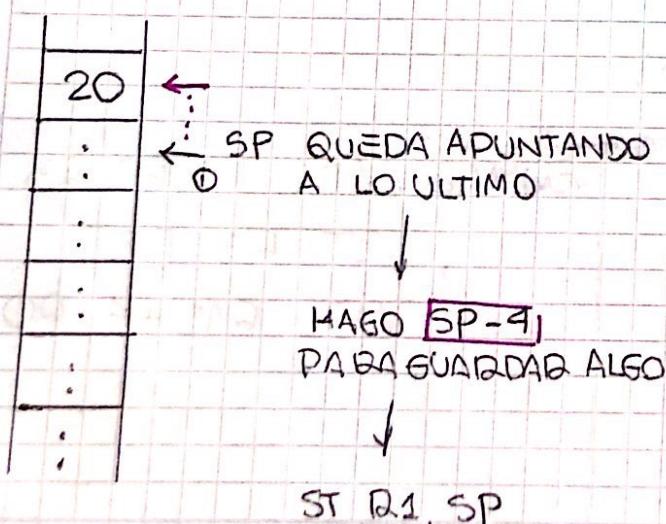
CUANDO USO LA PILA

SI GUARDO ALGO \rightarrow SP -4

SI SACO ALGO \rightarrow SP +4

TENGO QUE DESAD EL

SP COMO ESTABA.



NOTA

EJ 8

- BEGIN
- ORG 2048
- LD [X], R1
- LD [Y], R2

→ ~~SUMAR~~ SUMAR
NO LLEVA
EL CALL X: 15

Y: 3
Z: 0

SUM

- MACRO SUMAR ;A, ;B, ;R3
- ADDCC A, B, ;R3
- ~~MOVCC R3, Z~~
- END MACRO

MACRO

- SE ACCDE EN TIEMPO DE ENSAMBLADO CONVIERTIENDO EN CODIGO EQUIV.
- PARAMETROS SON INTERPRETADOS POR EL ENSAMBLADOR Y REEMPLAZADOS
- DESPISTE CODIGO TANTAS VECES COMO FUE INVOCADA

SUBRUTINA

SE ACCDE POR UN CALL EN TIEMPO DE EJECUCIÓN Y TERMINA CON JMPL

- SUS PARAMETROS LE LLEGAN EN TIEMPO DE EJECUCIÓN
- SU CODIGO ESTA EN UN LUGAR UNICO Y ESPECIFICO

OPERACIONES ARITMETICAS

EJ 10

EJ 9

SE DEBE POER STACK UN NUMERO ENTEO
EN COMP. A 2. DEVUELVE SU VALOR ABSOLUTO

SI TENO NUMEROS SIGNADOS

• BEGIN

$$\begin{array}{r} 1111 \\ + 1010 \\ \hline \end{array} = -6$$

• ORG 2048

$$\begin{array}{r} 0000 \\ + 0101 \\ \hline \end{array}$$

% SP • EQU / R14

$$\begin{array}{r} 0000 \\ + 0110 \\ \hline \end{array} = 6$$

ADDCC R15, R0, R16

LD [X], R1

ADDCC SP, -4, SP

ST R1, SP

CALL ABSOLUTO

LD SP, R3

X: -6

JMP R16 + 4, R0

ABSOLUTO: LD SP, R2

ADDCC SP, 4, SP

ORNCC R2, R0, R2

ADDCC R2, 1, R2

ADDCC SP, -4, SP

ST R2, SP

JMP R15 + 4, R0

CALCULO
ABSOLUTO

• END

NOTA

EJ 10.

RECIBE POR PILA 2 NUMEROS

-2
-5

Y DETERMINA
SI LA SUMA
ES REP. → 0
O NO
↓ -1

V → SABER SI ME FUI DE DANGO

• BEGIN

• ORG 2048

% SP . EQU / R14

LD SP, R1

ADDCC SP, 4, SP

LD SP, R2

ADDCC SP, 4, SP

ADDCC R1, R2, R3

→ BVS NO - REPPRESENTA

ADDCC SP, -4, SP

ST O , SP

JMPL R16 + 4, DO

NO - REPPRESENTA :

ST -1, SP

JMPL R16 + 4, DO

• END

EJ 11 LO MISMO QUE ARRIBA

PEDO CHEQUEO EL CARRY BCS

EJ 12

PARA MULTIPLICAR SUMO, POR EJEMPLO 5×4
SUMO 5 VECES 4.

5
10

←

• BEGIN

• ORG 2048

SP .EQU R14

ADDCC R15, R0, R16

LD SP, R1

ADDCC SP, 4, SP

LD SP, R2

ADDCC SP, 4, SP

LOOP: ANDCC R1, R1, R0

BE FIN

→ ADDCC R1, -1, R1

ADDCC R2, R3, R3

BCS FUERA - DE - RANGO

BA LOOP

FUERA DE RANGO : ADDCC SP, -4, SP

ST -1, SP

JMPL R15+4, R0

FIN : ADDCC SP, -4, SP

ST R3, SP

JMPL R15+4, R0

LD SP, R5

ADDCC SP, 4, SP

JMPL R16+4, R0

• END

EJ 14 LO MISMO SALVO QUE

HAGO LA MULTIPLICACIÓN
CON SI MISMO

NOTA

OPERACIONES CON BITS

EJ 15

- BEGIN
- ORG ZO48

ADDCC R15, R0, R16

LD [X], R10

SETHi 7FFFFh, R1

SLL R1, 2, R1

ADDCC R1, FFFh, R1

ANDCC R10, R1, R10

X: 2,35

- END

1|01-----|-----

011

0111 1111

EJ 17

- BEGIN

- ORG 2048

LD [X], ;.R20

SETHi 11110h, R1

SLL R1, 2, R1

ADDCC R1, 000h, R1

SETHi 00001h, R2

SLL R2, 2, R2

ADDCC R2, 111h, R2

1° 11110000h

2° 00001111h

1 2
16B 16B

↑

PARA EL

1° NUMERO

PARA EL

2° NUMERO

EJ 18

- BEGIN

- ODS 2048

/. SP EQU R14

LD SP, R1

ADDCC SP, 4, SP

ADDCC R0, 32, R2 → ADDCC R10, R0, R10

LOOP : ANDCC R2, R2, R0

BE FIN

ADDCC R2, -1, R2

ANDCC R1, R3 R0

SLL R3, 1, R3

BE NO-HAY-UNO

ADDCC R10, 1, R10

BA LOOP

NO-HAY-UNO : BA LOOP

FIN : ADDCC SP, -4, SP

ST R10, SP

JMPL R16 + 4, R0

- END

CONTADOR²

ADCC

R10,

R0, R10

ADDCC

R3,

1, R3

ANDCC

R2, -1, R2

SLL

R3, 1, R3

BE

NO-HAY-UNO

ADDCC

R10, 1, R10

BA

LOOP

NO-HAY-UNO

: BA

LOOP

FIN

: ADDCC

SP, -4, SP

ST

R10,

SP

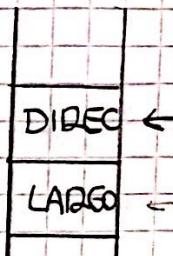
JMPL

R16 + 4,

R0

EJ 19

- BEGIN
- 026 2048
- SP . EQU R14



LD SP, R1 ! R1 ES LA DIREC INICIAL

ADDCC SP, 4, SP

LD SP, R2 ! R2 ES EL LARGO

ADDCC SP, 4, SP

LOOP: ANDCC R2, R2, R0

BE FIN

ADDCC R2, -4, R2

ADDCC SP, -4, SP

ST R1, R2, SP

CALL EJ-18

LD SP, R20

ADCC SP, 4, SP

JMPL R16 + 4, R0

- END

EJ 20

[5, 10, 3, 8]

MAX = 5 → AUX - i = NEG → PISO EL
AUXILIAR

MIN = 5 → AUX - i = POSITIVO → PISO EL
AUX

• BEGIN

• ORG 2048

SP = EQU R14

ADDCC R15, R0, R16

LD SP, R1 ! R1 DIREC

ADDCC SP, 4, SP

LD SP, R2 ! R2 LARGO

ADDCC SP, 4, SP

ADDCC R0, R0, R20 ! AUX MAX

ADDCC R0, R0, R21 ! AUX MIN

LOOP: ANDCC R2, R2, 0

LD R1, R2, R20

LD R1, R2, R21

BE FIN

ADDCC R2, -4, R2

CALL MAXIMO

CALL MINIMO

ADDCC SP, -4, SP

ST R20, SP

ADDCC SP, -4, SP

ST R21, SP

BA LOOP

MAXIMO : LD R1, R2, R3
SUBCC R20, R3, R0
BNEG CAMBIAR
JMPL R15 + 4, R0

CAMBIAR : LD R1, R2, R20
JMPL R15 + 4, R0.

• END

MINIMO : LD R1, R2, R4
SUBCC R21, R4, R0
BNEG NO - CAMBIAR
LD R1, R2, R21
JMPL R15 + 4, R0

NO - CAMBIAR : JMPL R15 + 4, R0-

EJ 21

MASO MENOS LA MISMA IDEA

PERO HAGO UNA → SUMA ACUMULATIVA

- VERIFICO A CADA PASO BVS

- Si NO HAY V

↳ CHEQUEO EL BIS
MENOS SIGNIFICATIVO

0 → PAR

1 → IMPAR

NOTA

ACCESO A PERIFERICOS

EJ 22

DISPOSITIVO MAPEADO \rightarrow B0 101002 h

LOOP LEYENDO TODOS LOS TEMPS

EN UNA AUX GUARDO \rightarrow AUX - ? = NEG \rightarrow DEMPLA

- BEGIN

- ORG Z048

- SP EQU R14

ADDCC R0, R15, R16

SETHI B0101 h, R1 }

SLL R1, 2, R1

ADDCC R1, 002 h, R1 } ! DIREC DE MAPEO

ADDCC R2, 32, R2 } ! LARGO DEL ARDAY 8x4

LD R1, R2, R20 ! AUX

LOOP ANDCC R2, R2, R0

BE FIN

ADDCC R2, -4, R2

LD R1, R2, R21

SUBCC R20, R21, R0

BNEG CAMBIAR - AUX

BA LOOP

CAMBIAR - AUX : ADDCC R21, R0, R20

BA LOOP

FIN : JUMPL R16 + 4, DO

END

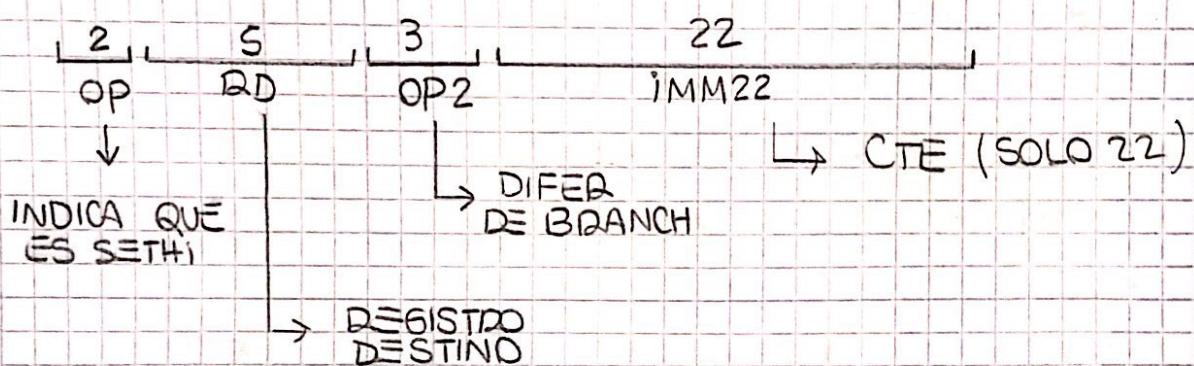
CÓDIGO DE MAQUINA

EJ 25

EL DESPLAZAMIENTO MÁXIMO ES DE 2^{22} BITS
LO CUAL ES EQUIVALENTE A 2^{20} INSTRUCCIONES

EJ 26

NO PUEDE PASAR MAS DE 22 BITS PORQUE
SINO NO DEJA LUGAR PARA EL RESTO DE LA
INSTRUCCIÓN

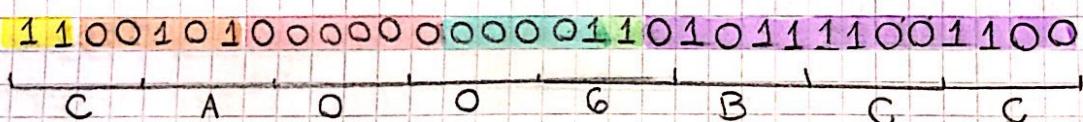


EJ 30

CA006BCC

A = 10 B = 11 C = 12

8 4 2 1



MEMORIA (OP)

REGISTRO DE DESTINO R5 (RD)

ES UN LOAD (OP3)

ES EL REGISTRO R1 (RS1)

i = 1

UNA CTE DE 13 BITS $\rightarrow 010111001100 = 2980$

LA INST \rightarrow LD R1, 2980, R5

Trabajo Práctico N° 7

LOS LENGUAJES Y LA MÁQUINA

Assembler

1) Dado el siguiente código

```

.org 2048
.global main
.extern MiFun
.equ 280000h
.cte22
.cte10
main: sethi cte22, %r12 48
      add %r12, cte10, %r12 52
      add %r15, %r0, %r16 56
      st %r0, %r12 60
      call MiFun 64
      orcc %r2, %r2, %r0 68
      bneg f in 72
      addcc %r2, -1, %r2 76
      bne set 80
      fin: jmpl %r16 + 4, %r0 84
      .org A00004h
Disp: .dwb l

```

- (a) Indicar la tabla de símbolos incluyendo relocalización y referencias externas,
(b) Señalar cuales son las líneas de este código que no tendrán representación en RAM cuando el programa sea cargado para su ejecución.
- 2) Idem problema anterior tomando el código que ha propuesto como respuesta a los problemas 22 y 23 de la tira n°5
- 3) 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')

```

- (a) Escribir su equivalente en lenguaje simbólico de ARC
(b) Obtener el código de máquina y la tabla de símbolos.

4) Dados:

	M1	M2
	<pre> begin .org 2048 main: or %r0, %r15, %r16 48 ld %r14, %r1 52 call separa 56 st %r2, %r14 60 add %r14, -4, %r14 64 st %r3, %r14 68 jmpl %r16, 4, %r0 72 separa: sra %r1, 16, %r2 76 sll %r1, 16, %r3 80 sra %r3, 16, %r3 84 jmpl %r15, 4, %r0 88 .end </pre>	<pre> .begin .org 2048 .macro separa Reg1, Reg2, Reg3 sra Reg1, 16, Reg2 52 sll Reg1, 16, Reg3 56 sra Reg3, 16, Reg3 60 .endmacro main: ld %r14, %r1 48 separa %r1, %r2, %r3 st %r2, %r14 64 add %r14, -4, %r14 68 st %r3, %r14 72 jmpl %r15, 4, %r0 76 .end </pre>

Se pide: (a) determinar el tamaño en bytes del código objeto generado por el ensamblador en cada uno (b) analizar diferencias en su tiempo de ejecución y medirlas a partir de la cantidad de ciclos de reloj insumidos por cada instrucción según el "time model editor" de ARCTools 2.1.2

- 5) Dos archivos de Assembly ARC fueron ensamblados independientemente y el ensamblador generó en cada caso los siguientes listados:

	HexLoc	DecLoc	MachWord	Label	Instruction
	Comment				
A					equ 1
					.org 2048
					.global subrut
	00000800	0000002048	86b0c000	subrut:	orcc %r3, %r0, %r3
	00000804	0000002052	8680e001		addcc %r3, 1, %r3
	00000808	0000002056	81c3e004		jmpl %r15, 4, %r0

	HexLoc	DecLoc	MachWord	Label	Instruction
B	Comment	org 2048			
	.extern subrut				
	00000800 0000002048	c4002810	main:	ld [2064], %r2	
	00000804 0000002052	c6002814		ld [2068], %r3	
	00000808 0000002056	40000000		call subrut	
	0000080c 0000002060	81c3e004		jmpl %r15, 4, %r0	
	00000810 0000002064	00000069	x:	2	
	00000814 0000002068	0000005c	y:	10	

- Indicar las tablas de símbolos generadas en cada caso
- Indicar cuáles son los cambios en el código binario que debe realizar un linker (link editor) para en base a ambos generar un único código objeto ejecutable.
- Indicar cuáles son los cambios en el código binario que debe realizar un linking-loader para la ejecución del código objeto obtenido en el punto (a) si el sistema operativo asigna al programa un segmento de memoria estática que empieza en la dirección 0E28h

Lenguajes de alto nivel

6) Ordenar los siguientes códigos en función de su velocidad de ejecución

A	B	C	D
#define Largo = 25; #define Ancho =10; int main () {int Vol=6*Ancho*Largo;}	#define Cte1 = 25; int main () { int X=6*Cte1*10;}	#define Largo = 25; #define Ancho =10; #define Sup=Ancho*Alto; int main() {int Vol=6*Sup;}	#define Cte1 = 25; #define Cte2 =B001A012h int main () { int X=6*Cte1*Cte2;}

7) Idem ejercicio 6 con:

A	B	C	D
int Largo = 25; byte Ancho =10; int main () { int Sup=Ancho*Largo;}	int Largo = 25; int Ancho =10; int main () { int Sup=Ancho*Largo;}	float Largo = 25; int Ancho =10; int main () {int Sup=Ancho*Largo;}	float Largo = 25; int Ancho =10; int main () {float Sup=Ancho*Largo;}

8) Idem ejercicio 6 con:

A	B	C	D
double Largo = 25; int Ancho =10; int main () {doublé Sup=Ancho*Largo;}	float Largo = 25; double Ancho =10; int main () {double Sup=Ancho*Largo;}	float Largo = 25; float Ancho =10; int main () {double Sup=Ancho*Largo;}	double Largo = 25; double Ancho =10; int main () {double Sup=Ancho*Largo;}

9) Los siguientes códigos son equivalentes en su función pero difieren en su velocidad de ejecución. Justificar esa diferencia y en base a ello discutir costos y ventajas comparativas de utilizar funciones y procedimientos particularmente en caso de que estén anidados.

A	B
long A=1; long B=2; long C; int main () { A=C*(A+B);}	long A=1; long B=2; long C; long Suma(long x, long y) { return x+y; } int main () { A=C*Suma(A,B); return 0;}

TRABAJO PRACTICO N° 7

ASSEMBLED

EJ 1

A

SIMBOLO	VALOR.	DEUB	E/E
MAIN	2048	SI	E
MIFUN	2052	SI	E
CTE22	280000 h	NO	-
CTE10	010 h	NO	-
SET	2068	SSI	-
FIN	2092	SSII	-
DISP	A00004 h	NO	-

NO ES DEUBICABLE PORQUE ESA DIREC
DE MEMORIA CORRESPONDE AL I/O.

ES UN EXTERNO, NO SE DEFINE EN ESTE
MODULO SI ES DEUBICABLE O NO.

B NO TENDRAN REPRESENTACION EN RAM
LAS DIRECTIVAS

EJ 3

EJ 4

A MODULO 1 → 11 PALABRAS

$$11 \times 32 = 352 \text{ BITS}$$

8 Bi

352

1 BY

$$X = 44 \text{ BYTES}$$

CUANDO LLAMA A LA MACRO SOLO SE COPIAN LAS LINEAS DE ADENTRO .MACRO - .ENDMACRO

MODULO 2 → 8 PALABRAS

$$8 \times 32 = 256 \text{ BITS}$$

8 Bi

256 Bi

1 BY

$$X = 32 \text{ BYTES}$$

B EL MODULO 2 VA A SER MAS RAPIDO YA QUE TIENE MENOS LINEAS QUE EJECUTAR

EJ 5

A MODULO A

SIMBOL	VALOR	G/E	R
SUBAUT	2048	G	Si

↓ OJO! HAY QUE TENER EN CUENTA SI ACCEDEN + VECES A RAM

MODULO B

SIMBOL	VALOR	G/E	R
SUBAUT	-	E	-
MAIN	2048	-	Si
X	2064	-	Si
Y	2068	-	Si

NOTA:

B

LINK EDITOR → PRODUCE UNA VERSION LINKEADA
DEL PROGRAMA QUE NORMALMENTE
ES GUARDADA EN ARCHIVO DADA
SEDE EJECUTADA EN OTRO MOMENTO

Trabajo Práctico N° 8

MICROARQUITECTURA

Camino de datos

1. Plantee el esquema de una estructura tipo bus en un sentido general y luego identifique los componentes de la microarquitectura ARC que conforman el "bus de address", el "bus de datos" y el "bus de control" en los buses A B y C. de la misma
2. El registro $\%r0$ puede ser diseñado por medio de buffers tri-state exclusivamente. Proponga el diagrama circuital correspondiente.
3. Proponga un circuito que implemente el registro de instrucciones. Considere FF-D del tipo maestro-esclavo e indique todas las conexiones con el resto de los componentes.
4. Diseñar una unidad aritmético-lógica de 4 bits que tenga capacidad de realizar las operaciones addcc, andcc y orcc. Se piden dos soluciones alternativas:
 - (a) circuitos dedicados (sumador y operadores lógicos)
 - (b) lookup table
5. Plantear el esquema circuital de una microarquitectura ARC basada en 2 buses solamente.

Unidad de control

6. Defina ciclo de búsqueda-ejecución y explique la forma en que este es implementado en la microarquitectura ARC micropogramada. Indique los pasos clave del micropograma.
7. En un procesador ARC el registro Program Counter apunta a la siguiente instrucción guardada en RAM:

addcc %r10, 48, %r1

Detallar los pasos de micropograma que la decodifican y para cada uno de ellos indicar los valores presentes en las entradas y en las salidas de cada uno de los siguientes bloques funcionales:

- multiplexor de direcciones de la memoria de control
- incrementador de direcciones de la memoria de control
- decodificadores de los buses A B y C
- multiplexores que intervienen en la decodificación de los buses A B y C
- multiplexor de datos del bus C
- bus de datos A B y C
- entrada de direcciones del módulo de memoria RAM

Considere que antes de la ejecución de esta instrucción $\%r10 = \%r1 = 0$

8. Idem ejercicio anterior considerando que la instrucción a decodificar es

ld %r10, %r1, %r7

Considere que antes de la ejecución de esta instrucción: $\%r10=3000$, $\%r1=8$, $\%r7=100$

9. Especificar los problemas de funcionamiento que surgirían en el caso de que se conectara a tierra la entrada 'acknowledge' del incrementador de direcciones de la memoria de control
10. Especificar los problemas de funcionamiento que surgirían en caso de que el bit 13 del MIR quedara permanentemente conectado a la tensión de alimentación.
11. Proponer un diagrama circuital para el bloque "lógica de control de saltos" de la microarquitectura ARC
12. Proponer un diagrama circuital para el bloque "multiplexor de la memoria de control" de la microarquitectura ARC

Microcódigo

13. Indique el contenido binario en las posiciones 60 y 61 del MIR a fin de que comprendan las siguientes microinstrucciones:

60: $R[\text{temp}0] = \text{NOR}(R[0], R[\text{temp}0]); \text{ IF } Z \text{ THEN GOTO } 64;$

61: $R[\text{rd}] = \text{INC}(R[\text{rs}1]);$

Complete con 0's los campos que no sean necesarios e indique eventuales consecuencias de completarlos con otro valor.

14. Proponga un patrón de 0's y 1's en los campos C y CMUX del registro de microinstrucciones de manera que ningún registro de propósito general cambie su contenido.

15. Considerando que la siguiente tabla almacena el microcódigo de una unidad ARC microprogramada indicar las tres correspondientes microinstrucciones en forma de mnemónicos

A	B	C	XDR	ALU	COND	JUMP ADDR
M	M	M				
U	U	URW				
11100101 X	B	X	C	XDR	ALU	COND JUMP ADDR
1001010000000000	0100001000000000	0110000000000000				
0000001100001000000000	0100001000000000	010001101110000000				
0000001000000110000000	0110001000000000	010001011110000010				

16. Un microprograma requiere de un salto condicional basado en el bit 13 del registro de instrucciones. Se proponen dos soluciones, los respectivos segmentos de microcódigo se presentan a continuación:

- 1792: $R[\text{temp}0] = \text{ADD}(R[\text{rs}1], R[\text{rs}2]);$

IF $R[\text{IR}[13]] \text{ THEN GOTO } 1794;$

- 1792: IF $R[\text{IR}[13]] \text{ THEN GOTO } 1794;$

$R[\text{temp}0] = \text{ADD}(R[\text{rs}1], R[\text{rs}2]);$

a) Indique eventuales ventajas o inconvenientes de una solución u otra.

b) De ejemplos de microporgramación en que el valor del bit 13 condiciona el flujo de microprograma

17. El siguiente microcódigo corresponde a la instrucción *call* del Assembler ARC:

1280: $R[15] = \text{AND}(R[\text{pc}], R[\text{pc}]);$

1281: $R[\text{temp}0] = \text{ADD}(R[\text{ir}], R[\text{ir}]);$

1282: $R[\text{temp}0] = \text{ADD}(R[\text{temp}0], R[\text{temp}0]);$

1283: $R[\text{pc}] = \text{ADD}(R[\text{pc}], R[\text{temp}0]);$

GOTO 0

Reescribirlo de modo que se usen sólo 3 líneas de microcódigo en vez de 4. Utilizar la operación LSHIF2.

18. Dado el siguiente microcódigo correspondiente a la instrucción *subcc* del Assembly ARC:

1584: $R[\text{temp}0] = \text{SEXT13}(R[\text{ir}]);$

IF $R[\text{IR}[13]] \text{ THEN GOTO } 1586;$

1585: $R[\text{temp}0] = \text{OR}(R[0], R[\text{rs}2]);$

1586: $R[\text{temp}0] = \text{NOR}(R[\text{temp}0], R[0]);$

1587: $R[\text{temp}0] = \text{INC}(R[\text{temp}0]);$

1588: $R[\text{rd}] = \text{ADDCC}(R[\text{rs}1], R[\text{temp}0]);$

GOTO 2047;

Calcule cuántos ciclos de reloj demora el procesador en ejecutar
 $\text{subcc } \%r1, \%r4, \%r6$

19. Indique el contenido binario del Control Store entre las direcciones 1584 y 1588 de acuerdo al microcódigo del ejercicio anterior

20. El standard SPARC incluye la instrucción *addxcc* (*add with carry*) la cual es similar a *addcc* salvo que además de los sumandos estándar suma también el contenido del bit de carry. Se pide:

a) Dar un ejemplo de su aplicación en el contexto de un programa que realiza la suma de números de 64 bits.

b) Detalle los cambios que deberían ser introducidos en la memoria de control de un procesador ARC para que *addxcc* forme parte de su set de instrucciones. Para ello considerar que esta instrucción debe cumplir con el formato de operaciones aritmético-lógicas con un op3 igual a 010011.

21. Según se propone en el libro de Murdocca-Heuring, el microcódigo guardado en la dirección 2047 de la memoria de control es el siguiente:

2047: R[pc] = INCPC(R[pc]); GOTO 0;

Proponga un código alternativo que no incluya GOTO 0 y que cumpla la misma función.

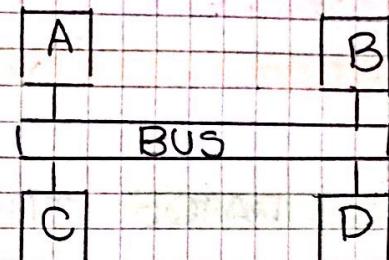
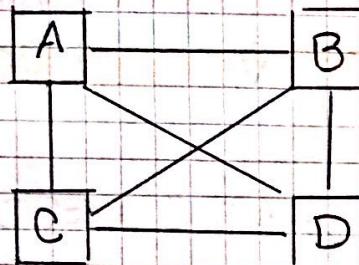
22. El siguiente microcódigo decodifica instrucciones de salto condicional. El mismo incluye intencionalmente un error en la posición 12 en el modo con que el PC es actualizado. Identifique el error y proponga una solución.

```
/ Branch instructions: ba, be, bcs, bvs, bneg
1088: GOTO 2;                                / Decoding tree for branches
      2: R[temp0] ← LSHIFT10(R[ir]);        / Sign extend the 22 LSB's of %temp0
      3: R[temp0] ← RSHIFT5(R[temp0]);       / by shifting left 10 bits, then right 10
      4: R[temp0] ← RSHIFT5(R[temp0]);       / bits. RSHIFT5 does sign extension.
      5: R[ir] ← RSHIFT5(R[ir]);            / Move COND field to IR[13] by
      6: R[ir] ← RSHIFT5(R[ir]);            / applying RSHIFT5 three times. (The
      7: R[ir] ← RSHIFT5(R[ir]);            / sign extension is inconsequential.)
      8: IF R[IR[13]] THEN GOTO 12;         / Is it ba?
          R[ir] ← ADD(R[ir],R[ir]);
      9: IF R[IR[13]] THEN GOTO 13;         / Is it not be?
          R[ir] ← ADD(R[ir],R[ir]);
     10: IF Z THEN GOTO 12;                / Execute be
          R[ir] ← ADD(R[ir],R[ir]);
     11: GOTO 2047;                      / Branch for be not taken
     12: R[pc] ← ADD(R[pc],R[temp0]);     / Branch is taken
          GOTO 0;
     13: IF R[IR[13]] THEN GOTO 16;       / Is it bcs?
          R[ir] ← ADD(R[ir],R[ir]);
     14: IF C THEN GOTO 12;                / Execute bcs
          R[ir] ← ADD(R[ir],R[ir]);
     15: GOTO 2047;                      / Branch for bcs not taken
     16: IF R[IR[13]] THEN GOTO 19;       / Is it bvs?
          R[ir] ← ADD(R[ir],R[ir]);
     17: IF N THEN GOTO 12;                / Execute bneg
          R[ir] ← ADD(R[ir],R[ir]);
     18: GOTO 2047;                      / Branch for bneg not taken
     19: IF V THEN GOTO 12;                / Execute bvs
          R[ir] ← ADD(R[ir],R[ir]);
     20: GOTO 2047;                      / Branch for bvs not taken
2047: R[pc] ← INCPC(R[pc]); GOTO 0;           / Increment %pc and start over
```

TRABAJO PRACTICO N° 8

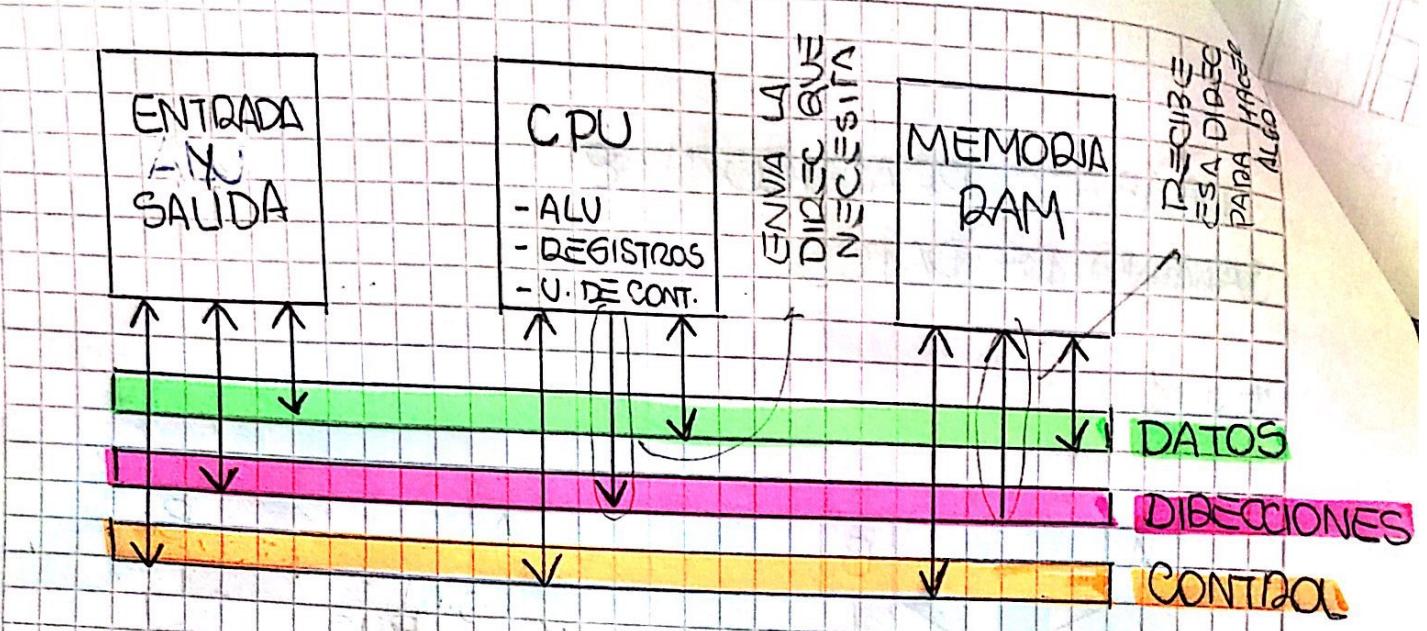
CAMINO DE DATOS

EJ 1



UN BUS PLANTEA UNA SOLUCIÓN EFICIENTE PARA SISTEMAS DONDE MUCHOS MODULOS SE INTERCONECTAN, DONDE SE REDUCE LA CANTIDAD DE CABLEADO REQUERIDO.

UNA VENTAJA ES QUE EL DISEÑO ES SIMPLE Y OCUPA MENOS AREA PERO LA DESVENTAJA ES QUE NO PUEDE CONECTAR TODO AL MISMO TIEMPO, DEQUIERDE DE UN CONTROL QUE HABILITE O DESHABILITE LAS ENTRADAS O SALIDAS.



BUS DE DATOS TRANSPORTA DATOS DESDE/HACIA LA CPU, MEMORIA I/O.

BUS DE DIRECCIONES LA MEMORIA SOLO RECIBE DIRECCIONES Y LA CPU LAS ENVIA.

BUS DE CONTROL TODOS PUEDEN CONTROLAR.

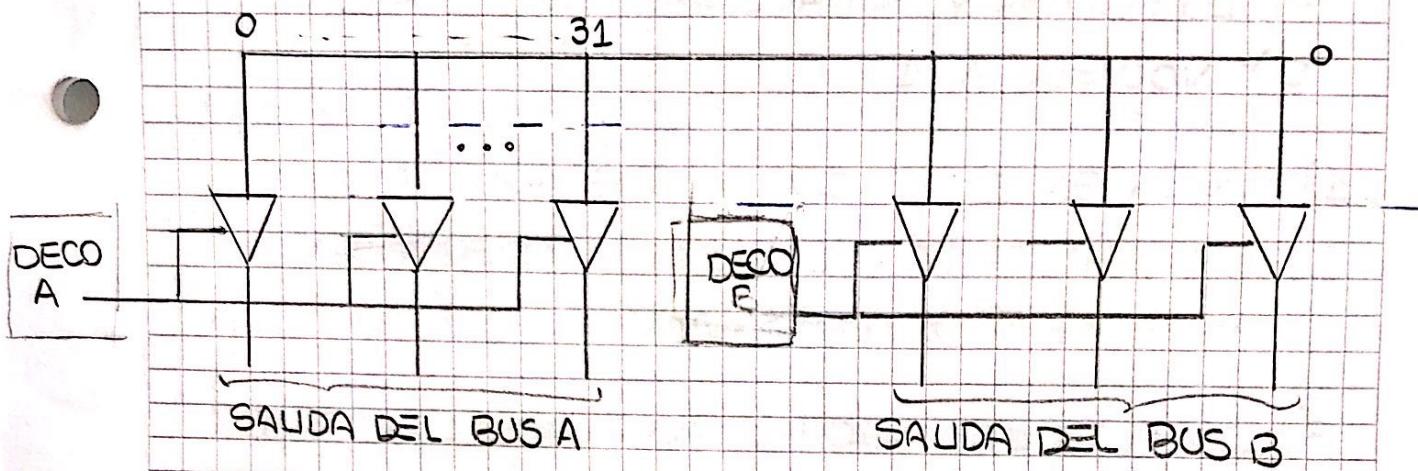
BUS A BUS DE DIRECCIONES
 A MEMORIA

BUS B BUS DE DATOS
 A MEMORIA

BUS C BUS DE DATOS
 DESDE MEMORIA.

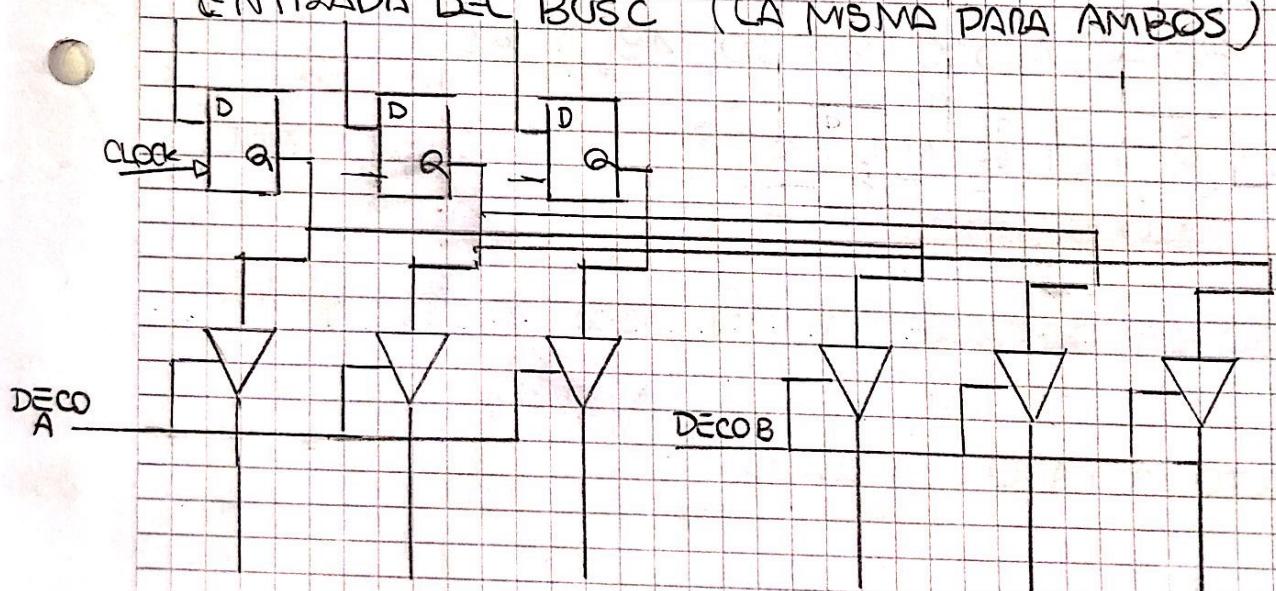
EJ. 2

EL BO NO TIENE ENTRADA DE C, PORQUE
NUNCA SE PUEDE MODIFICAR



EJ. 3

ENTRADA DEL BUS C (LA MISMA PARA AMBOS)



NOTA

UNIDAD DE CONTROL

EJ 6

CICLO DE BUSQUEDA - EJECUCIÓN

- 1) BUSCAR LA PRÓXIMA INSTRUCCIÓN
- 2) DECODIFICARLA
- 3) BUSQUEDA DE OPERANDOS (SI HAY)
- 4) EJECUTAR Y ALMACENAR RESULTADO
- 5) VOLVER A 1

- 1) CARGA EN EL %IR LO QUE APUNTA %PC
- 2) INTERPRETA LA INSTRUCCIÓN (DECODE)
ASÍ SABE A DONDE SALTA
- 3) EJECUTA LA MICROINSTRUCCIÓN

EJ 7

PC → ADDCC %R10, 48, %R1

0: R[R1] ← AND (R[PC], R[PC]); READ

1: DECODE

HAGO LA DECODIFICACIÓN

110010000000
= 1600

SALTO A LA LÍNEA 1600

1600: IF R[12][13] THEN GO TO 1602

M12

BUS A 000000

MUX A 0

BUS B 000000

MUX B 0

BUS C 000000

MUX C 0

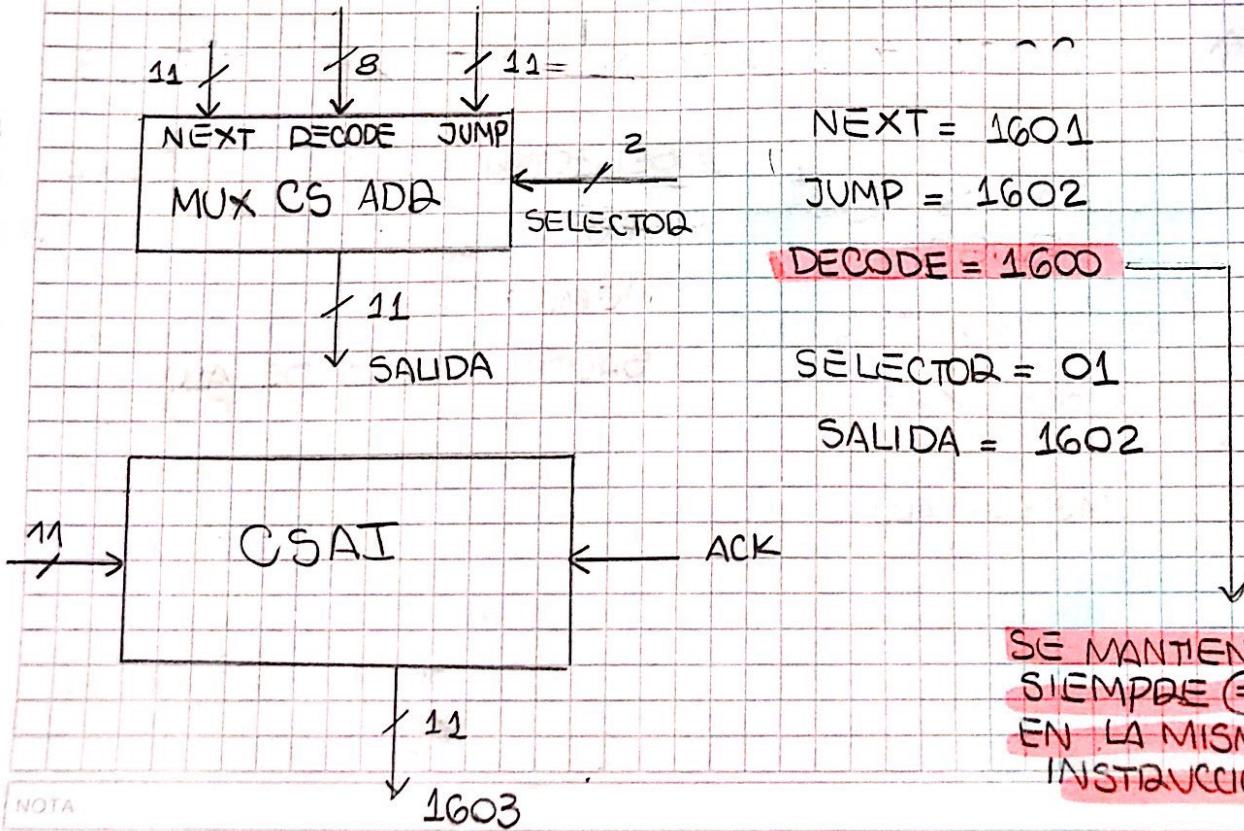
RD 0

WR 0

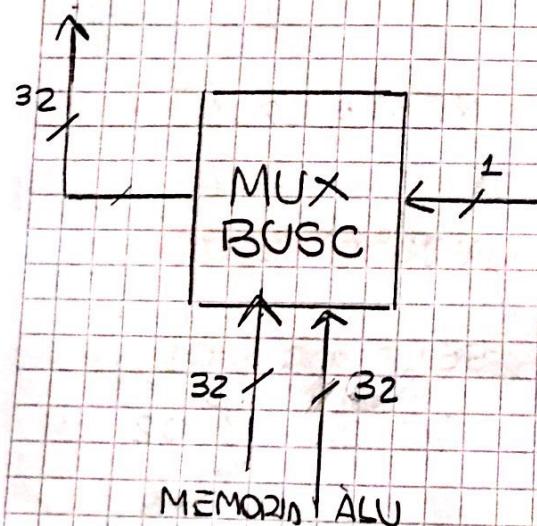
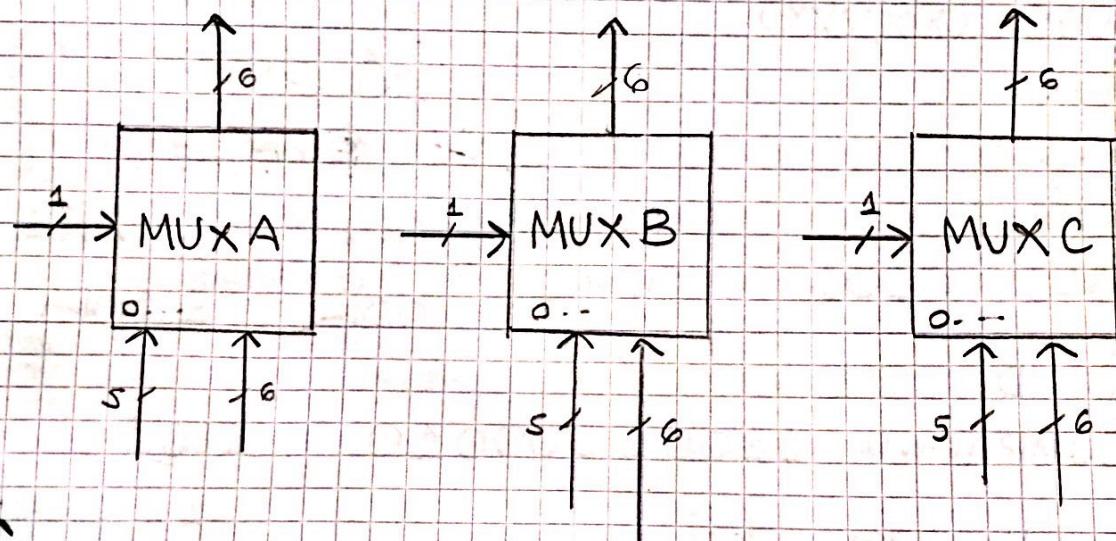
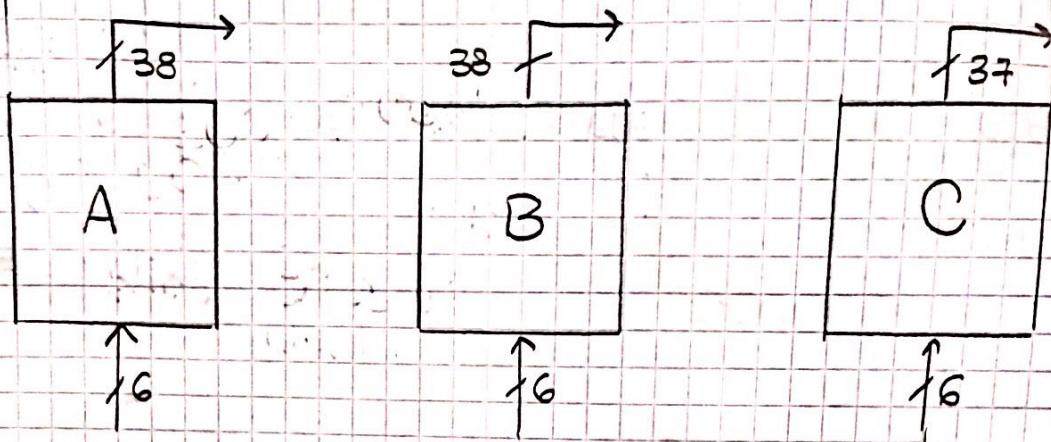
ALU 0101 → PONGO UNA SIEMPRE QUE NO MODIFIQUE NADA

COND 101

JUMPADDR 11001000010 = 1602



DECODIFICADORES



SELECTOR ES 0

ALU :

MEMORIA :

SALIDA ES LO DE ALU

NOTA

AHORA COMO $i = 1$ SALTO A 1602

1602: $Q[TEMPO] \leftarrow SEXT13[RCIR]$

MIR

CAMPO A 100101 (37)

MUX A 0

CAMPO B 000000

MUX B 0

CAMPO C 100001 (33)

MUXC 0

RD 0

W = 0

ALU 1100

COND 000

JUMPADDR 000000000000

1603: $Q[RD] \leftarrow ADDCC(RS1, TEMPO),$
GO TO 2047;

CAMPO A 000000 JUMPADD = 2047

MUX A 1

CAMPO B 100001

MUX B 0

CAMPO C 000001

MUXC 0

ALU 0011

COND 110

EJ 8

LD %R10, %R1, %R7

O: R[IR] \leftarrow ANDCC (R[PC], R[PC]); READ

1. DECODE

DECODE \rightarrow 111000000000 = 1792

1792: R[TEMPO] \leftarrow ADD (R(RS1), R(RS2));

MIR IF R(12(B)); GO TO 1794

CAMPO A 000000

MUXA 1

CAMPO B 000000

MUXB 1

CAMPO C 100001

MUXC 0

RD 0, WR 0, ALU 1000

COND 101 JUMP ADDR 11100000010

(1794)

1793: R(RD) \leftarrow AND (R(TEMPO) R(TEMPO)), READ,
GO TO 2047

CAMPO A 100001

MUXA 0

RD 1

CAMPO B 100001

MUXB 0

WR 0

CAMPO C 000000

MUXC 1

ALU 0101

COND 110

JUMP ADDR = 2047 = 1111111111

NOTA

DATA CLEARED

EJ 9

SI EL ACK SE CONECTARA A TIERRA, ESTO SIGNIFICA QUE SU VALOR ES SIEMPRE 0 ENTONCES NO INCREMENTARIA EL VALOR DE LA LINEA SIGUIENTE, EVENTUALMENTE SI SE NECESITA USAR NEXT EL PROGRAMA FUNCIONARIA MAL YA QUE QUEDO EN LA MISMA LINEA.

EJ 10

SI EL BIT 13 DEL RD QUEDARA SIEMPRE EN 1, FUNCIONARIA MAL CIERTAS INSTRUCCIONES

ADDCC R1, R2, R20

10 - RD - OP3 - BS1 - i → 00000000 - R22

→ SIMM 13 →

Si i = 1 LA LINEA ADDCC R1, R2, R20 LA TOMARIA.

10 - 00011 - 010000 - 00001 - 1 - ?

Sí

BIT 13 DEL MIR, OSEA EL BMUX, SIEMPRE QUEDA EN 1 SE RECIBIDA LA INFO DE RS2 DEL IR Y NO DEL CAMPO C DEL MIR.

EJ 11

EL CBL TIENE 8 ENTRADAS

4 FLAGS - IR 13 - CAMPO COND

V	C	Z	N	i	C1	C2	C3	S1	S2
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	1	-	-	-	-	-
0	1	0	0	0	-	-	-	-	-
1	0	0	0	1	-	-	-	-	-

HAY MUCHOS CASOS Y
MUCHAS DEDUNDANCIAS

ESTA TABLA LA PUEDO
METER EN EL ALGORITMO
DE MCCLUSKEY

Y ME DEVOLVERÁ
EL CIRCUITO MAS
SIMPLIFICADO PARA DE SOLVER
CON COMPUERTAS LÓGICAS

EJ 12

IDEA ARDIBA

MICROCODIGO

EJ 13

GO: $R(\text{TEMPO}) \leftarrow \text{NDR}(R(0), R(\text{TEMPO}))$;
IF \neq GO TO 64

M12

CAMPO A 000000 AMUX 0
CAMPO B 100001 BMUX 0
CAMPO C 100001 CMUX 0
RD 0 WD 0 ALU 0111
COND 010 JUMPADD 00000100000 = 64

61: $R(RD) \leftarrow \text{INC}(R(125))$

M12

CAMPO A 000000 AMUX 1
CAMPO B 000000 BMUX 0
CAMPO C 000000 CMUX 1
RD 0 WD 0 ALU 1101
COND 000 JUMPADD 000000000000

ES REDUNDANTE CON QUE SE COMPLETA

NOTA

EJ 15

- 1) $Q[TEMPO] \leftarrow \text{SEXT13}(Q[IQR])$.
- 2) $Q[TEMPO] \leftarrow \text{ADD}(Q[DS1], Q[TEMPO])$;
GO TO 1793;
- 3) $Q[TEMPO] \leftarrow \text{ADD}(Q[DS1], Q[DS2])$;
IF $Q[I2C13]$ THEN GO TO 1809.

EJ 16

A)

AMBOS CODIGO SON IGUALES, SE OPERAN EN EL MISMO CLOCK

B) LD, ST Y LAS ARITMETICAS CON CTES

EJ 17

- 1280: $Q[15] = \text{AND}(Q[PC], Q[PC])$
- 1281: $Q[TEMPO] = \text{LSHIFT2}(Q[I2])$
- 1282: $Q[PC] = \text{ADD}(Q[PC], Q[TEMPO])$;
GO TO 0

EJ 18

CADA OPERACION ARITMETICA DURA LO MISMO

SI CUENTO 0, 1, 2047 \rightarrow 8 PULSOS DE RELOJ
LO QUE HAY AHI \rightarrow 5 PULSOS

NOTA