Trabajo Práctico 0: Arquitectura de computadoras

Versión 2C 2023

Carrera: INGENIERIA EN INFORMATICA

Asignatura: 3638-Arquitectura de computadoras

Tema: Arquitectura de computadoras

Unidad: 0-Arquitectura de computadoras

Objetivo: Entender diferentes arquitecturas de computadoras, haciendo hincapié en la arquitectura RISC-V y comprender parte de su funcionamiento mediante la creación de programas para la misma. Asimilar los conceptos de eficiencia y el impacto del uso de pipelining.

Competencia/s a desarrollar:

Genéricas:

- Desempeño en equipos de trabajo.
- Comunicación efectiva.
- Actuación profesional ética y responsable.
- Aprendizaje continuo.
- Desarrollo de una actitud profesional emprendedora.

Específicas:

- Especificación, proyecto y desarrollo de software.
- Establecimiento de métricas y normas de calidad de software.
- Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
- Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
- Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
- Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.

Descripción de la actividad:

1-Tiempo estimado de resolución: 6 semanas
2-Metodología: Planteo y resolución de problemas modelo. Utilización de las herramientas de software recomendadas por la cátedra (Logisim-Evolution, Ripes).
3-Forma de entrega: La entrega no es obligatoria.
4:Metodología de corrección y feedback al alumno: Presencial y por Miel.

A- Arquitecturas de Computadora

A.01 *Defina los bloques funcionales de una computadora y los componentes internos que componen estos bloques (ej: flip flops, registros, compuertas)*

A.02 *Complete la siguiente tabla indicando cuántas palabras pueden ser direccionadas en computadoras con las siguientes cantidades de líneas (bits) en el bus de direcciones (sin esquema de multiplexado). Indique también la capacidad máxima de almacenamiento para los distintos tamaños de palabra. Recuerde que 8 bits son un byte. Si una palabra es de 16 bits, entonces esa palabra es de 2 bytes.*

Bus de direcciones	Palabras direccionables	Palabra 8 bits	Palabra 16 bits	Palabra 32 bits
4	(2^4)16 palabras	16 B (bytes)	32 B	64 B
8				
10				
12				
14				
16				
20				
24				
28	256 M palabras	256 MB	512 MB	1 GB
30				
31				
32				
40	1 T palabras	1 TB	2 TB	4 TB

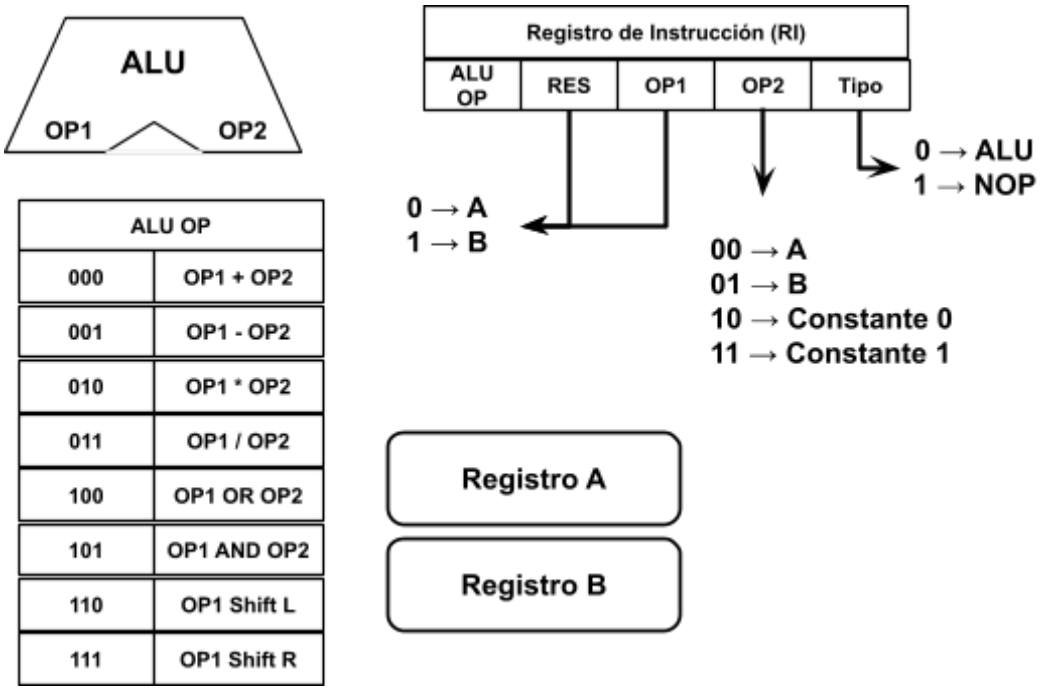
A.03 *Si se desea direccionar la cantidad de palabras indicadas a continuación, indique cuantas líneas son necesarias en el bus de direcciones.*

Palabras a direccionar	Log_2 (Palabras)	Líneas bus de direcciones
140	7,1293	8
65536	16	16
65537	16,000022	17
130000		
4294967296		
4294967297		

A.04 *Indique en hexadecimal la dirección más alta (comenzando en 0) para computadoras cuyos buses de direcciones poseen:*

Líneas bus de direcciones	Dirección más alta en hexadecimal
8	FF
10	3FF
11	
12	
15	
16	
19	
20	
22	
29	
30	
32	
33	

A.05 *Dada la siguiente arquitectura genérica , con dos registros de propósito general (A y B) y una ALU que soporta 8 operaciones:*



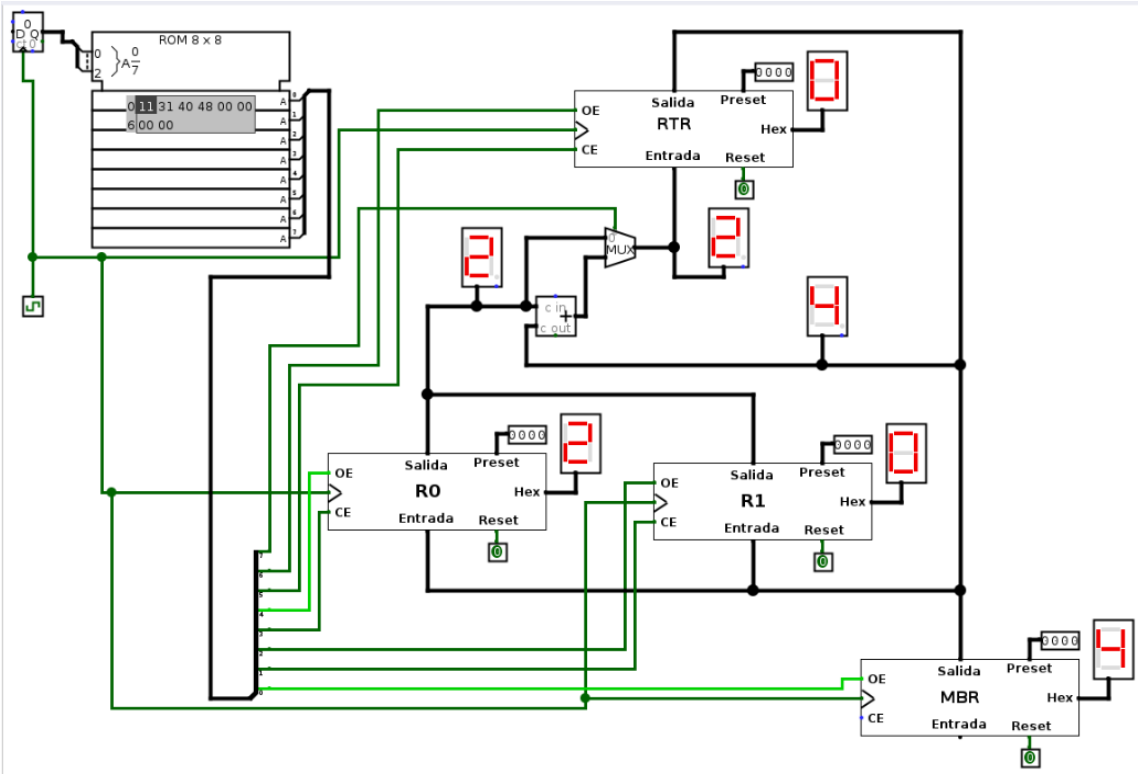
ALU OP	
000	OP1 + OP2
001	OP1 - OP2
010	OP1 * OP2
011	OP1 / OP2
100	OP1 OR OP2
101	OP1 AND OP2
110	OP1 Shift L
111	OP1 Shift R

Nota: **NOP** deja pasar el tiempo y no realiza ninguna acción. **Shift L** desplaza OP1 a izquierda un bit. **Shift R** desplaza a derecha de forma aritmética un bit el valor de OP1.

Indique el valor que debe tomar la instrucción (teniendo en cuenta el formato de RI) para resolver las siguientes operaciones:

- A=B+1 (ej: 00001110)
- A=B (ej: 00001100)
- A=0
- A=A^2
- B=A
- B=A^2
- B=B^2
- B=B*2 (en binario correr la coma divide o multiplica por dos)
- B=A/2 (en binario correr la coma divide o multiplica por dos)
- A=1 si B es impar (mirar el bit menos significativo de B y recordar AND).
- B=1 (sabiendo que A!=0)
- A=A/B
- A=B/A
- A=A (usando NOP o usando alguna operación aritmética)
- Indique lo que ocurre cuando RI= 01101100

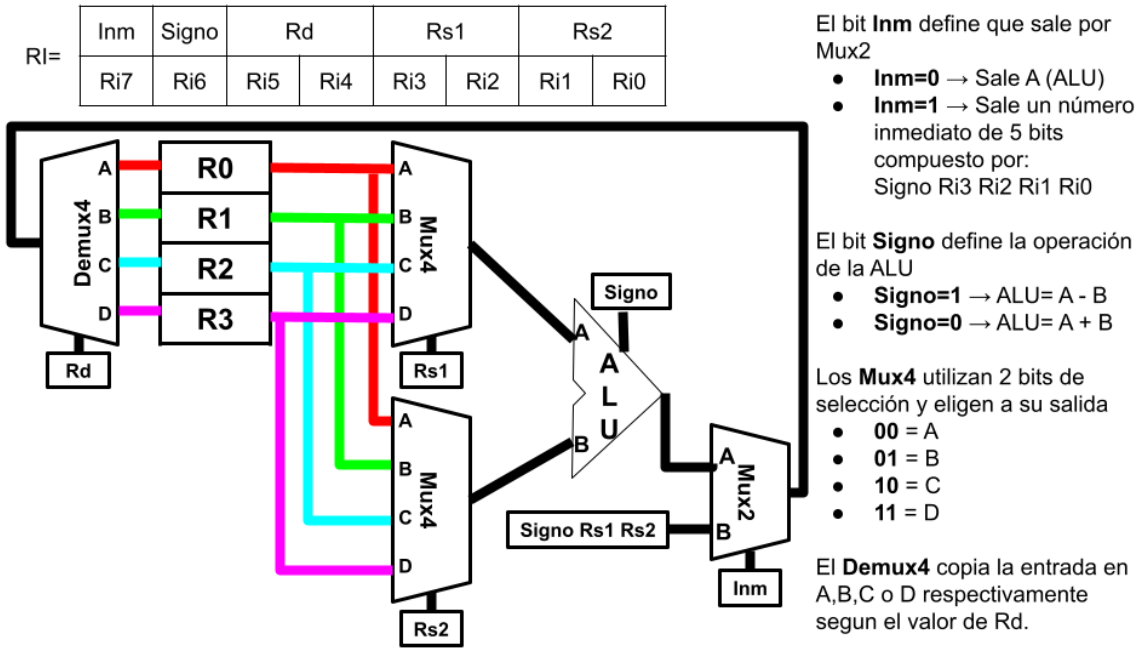
A.06 *Dada la siguiente arquitectura (microprogramada.circ) , basado en el modelo teórico de arquitectura microprogramada, escribir microcódigos para resolver las siguientes operaciones.*



- *R0 = R1 + MBR*
- *R1 = R0*
- *R0 = R0 + R0*
- *R1 = R0 + MBR*
- *R0 = R0*
- *R1 = R1*
- *R0 = R1 + R1 + R1 + R1*
- *R0 = R0 (usando ALU = A+0)*
- *R1 = R1 + MBR*
- *R0 = R0 + MBR*
- *R1 = 2 * R0*
- *R0 = R1 + 2*MBR*

Nota: *Utilice logisim-evolution para verificar los programas. Utilizando Preset puede dar valores iniciales a los registros.*

A.07 *Dada la siguiente arquitectura (cableada) que opera con palabras de 5 bits (negativos en C.B.), indique* ***el conjunto de instrucciones*** *necesario (como valores en el RI) para realizar las siguientes operaciones. (El resultado no necesariamente será válido, ej: sumar dos números cuyo resultado no puede ser representado en 5 bits).*



Realizar estos ejercicios utilizando el modelo Cableada.circ en Logisim-evolution. La forma de verificarlos es darle valores a los registros. Ej: R2=R1-R0, darle valores a R1 y R0 y verificar el resultado en R2 es efectivamente R1-R0.

- R2 = R1-R0 (ej: RI= 0 1 10 01 00)
- R3 = 2 (ej: RI= 1 0 11 0010)
- R3 = -2 (ej: RI= 1 1 11 1110)
- R1 = R2 +2 (ej: Ri= 1 0 01 0010 ; Ri = 0 0 01 10 01)
- R0 = -1 + R2
- R3 = (-16) + R2
- R1 = 4
- R2 = 3 + R3 - R1
- R3 = R2 + 16 (recuerde la regla de signos)

A.08 *Se dispone de una arquitectura Registro-Registro con 4 registros (R0, R1, R2, R3) y soporta 6 operaciones:*

- **SUMA Rd , Rs1 , Rs2** : Realiza la operación Rd = Rs1 + Rs2
- **RESTA Rd , Rs1 , Rs2** : Realiza la operación Rd = Rs1 - Rs2
- **MULT Rd , Rs1 , Rs2** : Realiza la operación Rd = Rs1 * Rs2
- **DIV Rd , Rs1 , Rs2** : Realiza la operación Rd = Rs1 / Rs2 (si Rs2=0 entonces Rd=1)
- **LOAD Rd , VAR** : Lee VAR de memoria y carga su valor en Rd
- **STORE Rs, VAR** :Toma el valor de Rs y lo almacena en VAR en memoria

Nota: Dividir por cero genera un 1 en el registro destino. La resta de un registro contra sí mismo siempre es cero.

Ejemplo: **A = 2 - B** → DIV R0, R0, R0 (R0=1) ; SUMA R0,R0,R0 (R0=2); LOAD R1,B (R1=B) ; RESTA R0,R0,R1 (R0=R0-R1), STORE R0,A (A = R0).

Resuelva las siguientes operaciones. Puede usar los registros de la forma que crea más conveniente.

$$A = \frac{(2-B)^2}{4-C} * D$$

$$A = \frac{B+2}{(C+4)^2} * D$$

$$A = (B+4)^2 - (1+C)^2$$

A.09 *Describa la arquitectura de programación y la estructura de un sistema de computación. Enuncie la diferencia entre ambos conceptos.*

A.10 *Asumiendo los direccionamientos al byte, indique para los siguientes tamaños de palabra si la dirección accedida se encuentra alineada o no.*

Tamaño de palabra [bytes]	Dirección	Alineada
4	0x00001200	Si
4	0x00001201	No
4	0x00001202	
2	0x00001202	
2	0x00001201	
1	0x00001200	
1	0x00001201	
4	0x00001204	
4	0x00001205	
2	0x00001205	

A.11 *Dada una computadora que direcciona 65536 bytes, la misma trabaja con palabras de 32 bits pero direcciona al byte. La computadora utiliza una instrucción para cargar el acumulador A con una palabra(32 bits) ubicada en una posición de memoria (LOAD 0xYYYY). Indique si las siguientes instrucciones se encuentran alineadas:*

- LOAD 0xF000
- LOAD 0xF001
- LOAD 0xF002
- LOAD 0xF003
- LOAD 0xF004
- LOAD 0xF005

Existe otra instrucción que carga el acumulador A con media palabra (half word) LOADHW 0xYYYY. Indique si las siguientes instrucciones se encuentran alineadas (considerando que solo se accede a 16 bits).

- LOADHW 0xF000
- LOADHW 0xF001
- LOADHW 0xF002
- LOADHW 0xF003
- LOADHW 0xF004
- LOADHW 0xF005

Por último indique si la instrucción LOADB 0xYYYY que carga un byte en el acumulador A genera accesos no alineados

- LOADB 0xF000
- LOADB 0xF001
- LOADB 0xF004
- LOADB 0xF007

A.12 *El siguiente vector se almacena en memoria a* ***partir de la dirección de memoria 0x12345678****. Se utiliza la instrucción LW (load word) para leer direcciones de memoria. La computadora direcciona con* ***32 bits al byte****, es decir, cada dirección de memoria apunta a un byte dentro de palabras de memoria de 32 bits.*

int vector[ ] = { 0x80516502 , 0x11223344, 0x00000908, 0x12345678, 0x20181209}

Escriba qué instrucción (lw r2, 0xYYYYYYYY) debe ejecutar la CPU para acceder a los elementos del vector.
Ejemplo: Para acceder a **vector[1]** se debe ejecutar: **lw r2, 0x1234567C.**
Complete para vector[0], vector[2], vector[3],vector[4].

A.13 *Almacene el número 0x20181209 en formato big endian y little endian a partir de la posición de memoria 0x12345678:*

Dirección	DatoBE	DatoLE
0x12345678		
0x12345679		
0x1234567A		
0x1234567B		

Repita el ejercicio para los siguientes datos: 0x00112233 , 0x44556677, 0x89ABCDEF, 0x6502, 0x8051.

B- ASM RISC-V

B.01 *Escriba la instrucción necesaria en Ripes para lograr que el registro x5 tenga el valor hexadecimal 0x000003FF (note que 3FF se codifica con menos de 12 bits)*

B.02 *Escriba las instrucciones necesarias en Ripes para lograr que el registro x5 tenga el valor hexadecimal 0x12345678*

B.03 *Luego de resolver B.02, almacene en x6 el valor de x5 más uno.*

B.04 *Luego de resolver B.02, almacene en x7 el valor de x5 menos uno.*

B.05 *Escriba el valor 100 en x7 y luego programe un bucle utilizando el registro x8 como contador de 10 a 0. En cada iteración del bucle debe incrementar en uno el valor de x7. Indique con qué valor comienza x7 y en qué valor termina luego del bucle.*

B.06 *Escriba un programa que almacene el número 2 en x5 , el número 3 en x6 y 0 en x7. Realice un bucle de forma tal que "x7 = x7 + x5" mientras que x6>0 (decrementando x6 en cada iteración). Indique el valor que queda en x7 luego del bucle. ¿Qué sucedería si el mismo programa se ejecuta cambiando los valores de x5=10 y x6=8 ? Almacene el programa en su computadora como "mult_x5_por_x6.s".*

B.07 *Escriba un programa que dado un número almacenado en x5 (ej: x5=234), el mismo sea dividido por 10, dejando en x7 la cantidad de veces que entra 10 en ese número, y en x8 el resto. NO utilizar la instrucción de división, se puede resolver con un bucle contando cuantas veces se le pueden restar 10 a x5 y que el mismo sea mayor que 10. Almacene el programa en su computadora como "div_x5_por_10.s"*

B.08 *Dadas las siguientes instrucciones en Ripes,* indique qué valor se muestra en la salida "Console" luego de ejecutar las instrucciones. Relacione estos valores con la tabla ASCII. Indique qué modificaciones debe hacer a las instrucciones para lograr que imprima "5678".

```
addi a0,x0,0x30
addi a7,x0,11
ecall
addi a0,x0,0x31
addi a7,x0,11
ecall
addi a0,x0,0x32
addi a7,x0,11
#Continua en la columna
#siguiente
```

```
ecall
addi a0,x0,0x33
addi a7,x0,11
ecall
addi a0,x0,0x34
addi a7,x0,11
ecall
```

B.09 *Escriba un programa llamado "ITOAREVERSE.s". El mismo debe grabar en el registro x5 el valor decimal 1234. Luego debe dividir el mismo por 10 (utilizando el método de restas sucesivas de B.07), mostrar el valor del resto por consola (recuerde que el ASCII del carácter 0 es 0x30 y el 9 es 0x39), y almacenar el resultado nuevamente en x5. Continuar ejecutando esta secuencia hasta que x5 sea cero.*

Ejemplo: en la primera iteración, se imprime 4 en consola (resto), y el número queda 123. En la segunda iteración, se imprime 3 en consola (resto), y el número queda 12. En la tercera iteración, se imprime 2 en consola y el número queda 1. En la cuarta iteración se imprime 1 en la consola y el programa termina.

B.10 *Escriba un programa llamado "DIV8.s". El mismo debe tomar un valor en el registro x5 y dividirlo por 8. NO se puede usar el método de restas sucesivas, y el mismo debe funcionar para valores positivos o negativos. Ej: si x5 = -25, el resultado es -3. Repase la instrucción SRAI.*

B.11 *Escriba el programa "ITOA1024.s" basado en el punto B.09, de forma tal que el número impreso por consola sea "1234" y no "4321". Recomendamos almacenar cada dígito en memoria e ir recorriendo la memoria en orden inverso para imprimir por consola. Puede crear en memoria .data una variable de 4 bytes que almacene ceros inicialmente y utilizar este espacio de memoria para invertir el número.*

B.12 *Escriba el programa "ITOA.s" basado en los puntos anteriores pero ahora el número a convertir no es solamente el 1024 en x5 sino que puede tener cualquier valor positivo en x5. Ejemplo: Si x5=0xBC614E , entonces debe imprimir por consola el 12345678.*

B.13 *Dado el siguiente programa:*

```
.data
numero: .string "127"
.text
  la   x5,numero
  lb   x6,0(x5)
  lb   x7,1(x5)
  lb   x8,2(x5)
```

Indique qué valores quedan almacenados en x6,x7 y x8. ¿Qué ocurre si en vez de usar lb se utiliza lh o lw?

B.14 *Basado en el ejercicio B.13, realice un programa que imprima por consola el valor del número en orden inverso (ej: 127 se imprime como 721). Luego considerando que los strings terminan con un carácter nulo (0x00) diseñe un programa que almacene el string* ***"YVAN EHT NIOJ"*** *y lo muestre por consola en orden inverso buscando primero la ubicación del caracter nulo y recorriendo en sentido inverso el string.*

B.15 *Se define un vector de 10 elementos (cada uno de 32 bits) llamado centena de la siguiente forma:*

```
.data
centena: .word 0,100,200,300,400,500,600,700,800,900
.text
  la x5,centena
  addi x6,x0,8
```

*Dado que x5 posee la dirección de comienzo del vector, y x6=8, escriba las instrucciones necesarias para que x7 obtenga el número 800 (almacenado en el vector). Recuerde que el direccionamiento en RISC-V es al byte, por ende la palabra 8 de un vector se encuentra almacenada en la dirección 8*4 relativa al comienzo del vector.*

B.16 *Analice lo que ocurre cuando se ejecuta el siguiente programa.*

Indique qué valor queda almacenado en x9 luego de la ejecución del mismo. Indique qué valor obtendría x9 si se define numero: .string “987”.
Describa en forma de comentario cual es la utilidad de este programa.

```
.data
numero: .string "127"
centena: .word 0,100,200,300,400,500,600,700,800,900
decena:  .word 0,10 ,20, 30, 40, 50, 60, 70, 80, 90
unidad:  .word 0,1  ,2  ,3,  4,  5,  6,  7,  8,  9
.text
  #Lectura
  la    x5,numero
  lb    x6,0(x5)
  addi  x6,x6,-48
  lb    x7,1(x5)
  addi  x7,x7,-48
  lb    x8,2(x5)
  addi  x8,x8,-48

  #Calculo Centena
  addi  x9,x0,0
  la    x10,centena
  slli  x6,x6,2 #direccion al byte a palabra
  add   x10,x10,x6
  lh    x11,0(x10) #centena almacena valores mayores al rango [-128:127]
  add   x9,x9,x11

  #Calculo Decena
  la    x10,decena
  slli  x7,x7,2 #direccion al byte a palabra
```

#---------->Continúa en la página siguiente<----------

```
  add   x10,x10,x7
  lb    x11,0(x10)
  add   x9,x9,x11
  #Calculo Unidad
  la    x10,unidad
  slli  x8,x8,2 #direccion al byte a palabra
  add   x10,x10,x8
  lb    x11,0(x10)
  add   x9,x9,x11
```

B.17 *Modifique el programa B.16 para que pueda aceptar números de* ***exactamente*** *4 dígitos.*

B.18 *Modifique el programa B.17 para que pueda aceptar números de* ***hasta*** *4 dígitos. A diferencia de los anteriores, en este programa son válidos los números "1234", "1", "6502","543","999","0001","0023","0123","0000", etc. No siempre serán de 4 dígitos. Tenga en cuenta que los strings terminan con un carácter nulo (0x00). Sugerimos pensar esto como 4 casos posibles (1 dígito, 2 dígitos, 3 dígitos, 4 dígitos), y dependiendo del caso saltar al caso correspondiente.*

B.19 *Dado un vector con valores enteros signados, escriba un programa que encuentre cuál es el valor mínimo almacenado y lo informe por consola. El vector posee 10 elementos. Una vez completado el programa modifique los valores del vector con distintos casos de prueba.*

```
.data
vector: .word 100,-20,40,-365,400,65536,1,0,3,-10
.text
  #Complete el programa
```

B.20 *Dado un vector con valores enteros signados, escriba un programa que encuentre cuál es el valor mínimo almacenado y lo informe por consola. El vector posee N elementos, pero se sabe que el último elemento del mismo siempre es 0. Una vez completado el programa modifique los valores del vector con distintos casos de prueba.*

```
.data
vector: .word 100,-20,40,-365,-400,65536,1,0
.text
  #Complete el programa
```

B.21 *Modifique el programa de B.20 para encontrar el valor máximo.*

B.22 *Dado un vector con valores enteros sin signo, escriba un programa que encuentre cuál es el valor mínimo almacenado y lo informe por consola. El vector posee N elementos, pero se sabe que el último elemento del mismo siempre es 0 (el cual no debe ser tenido en cuenta en el resultado). Una vez completado el programa modifique los valores del vector con distintos casos de prueba.*

```
.data
vector: .word 100,20,40,365,400,65536,1,0
.text
  #Complete el programa
```

B.23 *Escriba una subrutina llamada Max, la cual debe recibir en a0 y en a1 (x10 y x11 respectivamente) dos números signados y devolver cual de los dos es el mayor en a0.*

B.24 *Escriba una subrutina llamada MaxUnsigned, la cual debe recibir en a0 y en a1 (x10 y x11 respectivamente) dos números sin signo y devolver cual de los dos es el mayor en a0.*

B.25 *Modifique el programa B.21 para que utilice la subrutina de B.23. Tenga en cuenta recorrer el vector copiando el valor de cada elemento al registro a1, el valor máximo actual en a0, y la subrutina dejará en a0 el valor del máximo entre esos dos elementos. Luego es cuestión de ir cargando en a1 el valor del elemento actual del vector hasta llegar al final.*

B.26 *Escriba una subrutina llamada MaxDir. La misma va a recibir en a0 la dirección de un dato (número signado) y en a1 la dirección de otro dato (número signado). La subrutina debe retornar en a0 el valor 1 si el dato almacenado en la dirección de memoria apuntada por a0 es mayor que el dato almacenado en la dirección de memoria apuntada por a1.*

B.27 *Escriba una subrutina llamada Swap. La misma va a recibir en a0 la dirección de un dato (número signado) y en a1 la dirección de otro dato (número signado). La subrutina debe intercambiar el contenido de ambos. Ej: Si a0=0x10000000 , quiere decir que en la dirección de memoria 0x1000000 existe un dato (supongamos -10). Luego si a1=0x10000004, quiere decir que en 0x10000004 existe otro dato (supongamos 5). Si se llama a la subrutina con esas referencias a memoria la misma debe retornar habiendo guardado 5 en 0x10000000 y -10 en 0x10000004.*

B.28 *Dado un vector de 10 elementos:*

.data
vector: .word 100,-20,40,-365,400,65536,1,0,3,-10
Escriba una subrutina llamada PrintVector, la cual reciba en a0 la dirección de comienzo del vector y en a1 la cantidad de elementos del mismo. Imprima por consola los elementos del vector separados por coma (ej: 100, -20, 40, -365, 400, 65536, 1, 0, 3, -10).

B.29 *Dado un vector de 10 elementos:*

.data
vector: .word 100,-20,40,-365,400,65536,1,0,3,-10
Utilizando las subrutinas MaxDir y Swap, recorra el vector desde el primer elemento hasta el último comparando direcciones consecutivas (ej: vector[0] vs vector[1], luego vector[1] vs vector[2], luego vector[2] vs vector[3]....). En cada iteración si el primer elemento es mayor que el segundo, utilice swap para intercambiarlos. Al final de las iteraciones debe quedar el elemento mayor ubicado al final del vector. Utilice entonces un llamado a PrintVector para ver el mismo por consola.

B.30 *Transforme el ejercicio B.28 en una subrutina llamada PushMax, la cual reciba en a0 la dirección de comienzo del vector y en a1 la cantidad de elementos del mismo. Cuando la*

subrutina retorna debe haber "empujado" el elemento mayor a la última posición del vector. Verifique luego del retorno llamando a PrintVector.

B.31 *Escriba un programa llamado Burbujeo, que utilice PushMax 10 veces consecutivamente sobre el vector. El mismo debería quedar ordenado de menor a mayor luego de las 10 iteraciones. En cada iteración llame a PrintVector para ver como se ordena.*

B.32 *Modifique el programa anterior para que el mismo ordene de forma descendente.*

C- GCC RISC-V

C.01 *Dado el siguiente programa escrito en C:*

```
void main(){

}
```

//Note que el programa no hace nada más que declarar una función que retorna sin hacer nada.

Se genera el siguiente código ASM para RISC-V (compilado con GCC)

```
00010054 <main>:
   10054:      ff010113      addi x2 x2 -16
   10058:      00812623       sw x8 12 x2
   1005c:      01010413       addi x8 x2 16

   10060:      00000013       addi x0 x0 0

   10064:      00c12403       lw x8 12 x2
   10068:      01010113       addi x2 x2 16
   1006c:      00008067       jalr x0 x1 0
```

Asumiendo que x2=0x7FFFFFF0 y x8=0x00000000 al momento del inicio del programa, Represente el valor que tiene x2 , x8 y las posiciones de memoria: 0x7FFFFFEC y 0x7FFFFFE0 en cada instrucción.
Note que la instrucción addi x0 x0 0 es similar a NOP (no hacer nada). Por ende se puede entender que las primeras 3 instrucciones son ejecutadas por el compilador para guardar el valor de x8 en el stack, y las últimas 3 instrucciones recuperan este valor original y retornan a la dirección almacenada en x1.

C.02 *Dado el siguiente programa escrito en C:*

```
void main(){
   int a=5;
}
void funcion(){
   int a=10;
}
```

1)Indique qué diferencia existe entre el código ASM entre ambas funciones.
2)Indique en cada función cual es el offset de la variable con respecto al fp.

C.03 *Dado el siguiente programa escrito en C:*

```
void main(){
  int A=5;
  int B=10;
  int C;
  C=funcion(A,B);
}
int funcion(int a, int b){
  return a+b;
}
```

Se genera el siguiente código ASM:

```
00010054 <main>:
  10054:    fe010113    addi x2 x2 -32
  10058:    00112e23    sw x1 28 x2
  1005c:    00812c23    sw x8 24 x2
  10060:    02010413    addi x8 x2 32
  10064:    00500793    addi x15 x0 5
  10068:    fef42623    sw x15 -20 x8
  1006c:    00a00793    addi x15 x0 10
  10070:    fef42423    sw x15 -24 x8
  10074:    fe842583    lw x11 -24 x8
  10078:    fec42503    lw x10 -20 x8
  1007c:    01c000ef    jal x1 28 <funcion>
  10080:    fea42223    sw x10 -28 x8
  10084:    00000013    addi x0 x0 0
  10088:    01c12083    lw x1 28 x2
  1008c:    01812403    lw x8 24 x2
  10090:    02010113    addi x2 x2 32
  10094:    00008067    jalr x0 x1 0
00010098 <funcion>:
  10098:    fe010113    addi x2 x2 -32
  1009c:    00812e23    sw x8 28 x2
  100a0:    02010413    addi x8 x2 32
  100a4:    fea42623    sw x10 -20 x8
  100a8:    feb42423    sw x11 -24 x8
  100ac:    fec42703    lw x14 -20 x8
  100b0:    fe842783    lw x15 -24 x8
  100b4:    00f707b3    add x15 x14 x15
  100b8:    00078513    addi x10 x15 0
  100bc:    01c12403    lw x8 28 x2
  100c0:    02010113    addi x2 x2 32
  100c4:    00008067    jalr x0 x1 0
```

Identifique en el código ASM:
1) Lugar en el “frame pointer” (offset de x8 (fp)) donde se almacenan las variables A, B y C de main().
2) Registros utilizados para pasar como argumento los valores de A y B a la funcion().
3)Registro utilizado por funcion() para retornar el valor de a+b.

C.04 *Dado el siguiente programa escrito en C:*

```
unsigned char funcion(unsigned char,unsigned char);
void main(){
  unsigned char A=5;
  unsigned char B=10;
  unsigned char C;
  C=funcion(A,B);
}
unsigned char funcion(unsigned char a, unsigned char b){
  return a+b;
}
```

El mismo es una modificación del programa de C.03 donde los tamaños de las variables son de 8 bits codificados sin signo.
El código ASM generado para la función es:

```
000100a4 <funcion>:
   100a4:       fe010113        addi x2 x2 -32
   100a8:       00812e23        sw x8 28 x2
   100ac:       02010413        addi x8 x2 32
   100b0:       00050793        addi x15 x10 0
   100b4:       00058713        addi x14 x11 0
   100b8:       fef407a3        sb x15 -17 x8
   100bc:       00070793        addi x15 x14 0
   100c0:       fef40723        sb x15 -18 x8
   100c4:       fef44703        lbu x14 -17 x8
   100c8:       fee44783        lbu x15 -18 x8
   100cc:       00f707b3        add x15 x14 x15
   100d0:       0ff7f793        andi x15 x15 255
   100d4:       00078513        addi x10 x15 0
   100d8:       01c12403        lw x8 28 x2
   100dc:       02010113        addi x2 x2 32
   100e0:       00008067        jalr x0 x1 0
```

1) Indique por qué motivo se ejecuta la instrucción **“100d0: 0ff7f793 andi x15 x15 255”** luego de realizar la suma en 100cc.
2) Indique qué resultado se obtiene en C si A=200 y B=100;
3) Modifique el programa en C para que el resultado anterior sea válido.

C.05 *El siguiente programa en C*

```
void main(){
  int a=10;
  while(1){
     a=a+1;
  }
}
```

Genera el siguiente código ASM

```
00010074 <main>:
  10074:      fe010113      addi x2 x2 -32
  10078:      00812e23       sw x8 28 x2
  1007c:      02010413       addi x8 x2 32
  10080:      00a00793       addi x15 x0 10
  10084:      fef42623      sw x15 -20 x8
  10088:      fec42783       lw x15 -20 x8
  1008c:      00178793       addi x15 x15 1
  10090:      fef42623      sw x15 -20 x8
  10094:      ff5ff06f      jal x0 -12
```

1) Indique en qué posición del frame pointer se almacena la variable a.
2) A diferencia de ejercicios anteriores, el programa no termina ejecutando jalr x0, x1, 0. Explique por qué ocurre esto.

C.06 *El siguiente programa en C*

```
void main(){
  int a=10;
  int b=0;

  if (a>=5){
     b=2;
  } else if (a>=3){
     b=3;
  } else {
     b=4;
  }
}
```

Genera el siguiente código ASM

```
00010074 <main>:
   10074:       fe010113        addi x2 x2 -32
   10078:       00812e23         sw x8 28 x2
   1007c:       02010413         addi x8 x2 32
   10080:       00a00793         addi x15 x0 10
   10084:       fef42623        sw x15 -20 x8
   10088:       fe042423         sw x0 -24 x8
   1008c:       fec42703        lw x14 -20 x8
   10090:       00400793         addi x15 x0 4
   10094:       00e7d863         bge x15 x14 16
   10098:       00200793         addi x15 x0 2
   1009c:       fef42423        sw x15 -24 x8
   100a0:       0240006f         jal x0 36
   100a4:       fec42703        lw x14 -20 x8
   100a8:       00200793         addi x15 x0 2
   100ac:       00e7d863         bge x15 x14 16
   100b0:       00300793         addi x15 x0 3
   100b4:       fef42423        sw x15 -24 x8
   100b8:       00c0006f         jal x0 12
   100bc:       00400793         addi x15 x0 4
   100c0:       fef42423        sw x15 -24 x8
   100c4:       00000013         addi x0 x0 0
   100c8:       01c12403         lw x8 28 x2
   100cc:       02010113        addi x2 x2 32
   100d0:       00008067         jalr x0 x1 0
```

1) Indique en qué posición del frame pointer se almacena la variable a y en qué posición la variable b.

2) Explique que se está comparando (lógicamente) en la línea:

10094: 00e7d863 bge x15 x14 16

3) Explique que se está comparando (lógicamente) en la línea:

100ac: 00e7d863 bge x15 x14 16

C.07 *El siguiente programa en C*

```
void funcion(int ,int , int *);
int main(){
   int x=10;
   int y=20;
   int z=0;

   funcion(x,y,&z);
   return z;
}
void funcion (int a,int b, int *c){
   *c = a+b;
}
```

Genera el siguiente código ASM

```
00010074 <main>:
   10074:    fe010113    addi x2 x2 -32
   10078:    00112e23    sw x1 28 x2
   1007c:    00812c23    sw x8 24 x2
   10080:    02010413    addi x8 x2 32
   10084:    00a00793    addi x15 x0 10
   10088:    fef42623    sw x15 -20 x8
   1008c:    01400793    addi x15 x0 20
   10090:    fef42423    sw x15 -24 x8
   10094:    fe042223    sw x0 -28 x8
   10098:    fe440793    addi x15 x8 -28
   1009c:    00078613    addi x12 x15 0
   100a0:    fe842583    lw x11 -24 x8
   100a4:    fec42503    lw x10 -20 x8
   100a8:    01c000ef    jal x1 28 <funcion>
   100ac:    fe442783    lw x15 -28 x8
   100b0:    00078513    addi x10 x15 0
   100b4:    01c12083    lw x1 28 x2
   100b8:    01812403    lw x8 24 x2
   100bc:    02010113    addi x2 x2 32
   100c0:    00008067    jalr x0 x1 0
```

```
000100c4 <funcion>:
100c4:    fe010113    addi x2 x2 -32
100c8:    00812e23    sw x8 28 x2
100cc:    02010413    addi x8 x2 32
100d0:    fea42623    sw x10 -20 x8
100d4:    feb42423    sw x11 -24 x8
100d8:    fec42223    sw x12 -28 x8
100dc:    fec42703    lw x14 -20 x8
100e0:    fe842783    lw x15 -24 x8
100e4:    00f70733    add x14 x14 x15
100e8:    fe442783    lw x15 -28 x8
100ec:    00e7a023    sw x14 0 x15
100f0:    00000013    addi x0 x0 0
100f4:    01c12403    lw x8 28 x2
100f8:    02010113    addi x2 x2 32
100fc:    00008067    jalr x0 x1 0
```

1) Indique en qué posición del frame pointer se almacenan las variables locales x, y ,z.
2) Explique que se envía a la función usando el registro s2 (x12) en la secuencia:

```
   10098:    fe440793    addi x15 x8 -28
   1009c:    00078613    addi x12 x15 0
   100a0:    fe842583    lw x11 -24 x8
   100a4:    fec42503    lw x10 -20 x8
   100a8:    01c000ef    jal x1 28 <funcion>
```

3) Identifique dentro de la función en que posición del frame pointer se almacenan a, b y c.
4) Indique en qué línea del programa ASM se asigna el valor de a+b en *c.

C.08 *El siguiente programa en C*

```
int main(){
  int b=2;
  int c=0;
  for (int i=0;i<10;i++){
      c=c+b;
  }
  return c;
}
```

Genera el siguiente código ASM

```
00010054 <main>:
   10054:       fe010113        addi x2 x2 -32
   10058:       00812e23         sw x8 28 x2
   1005c:       02010413         addi x8 x2 32
   10060:       00200793         addi x15 x0 2
   10064:       fef42223        sw x15 -28 x8
   10068:       fe042623         sw x0 -20 x8
   1006c:       fe042423         sw x0 -24 x8
   10070:       0200006f         jal x0 32
   10074:       fec42703         lw x14 -20 x8
   10078:       fe442783         lw x15 -28 x8
   1007c:       00f707b3         add x15 x14 x15
   10080:       fef42623        sw x15 -20 x8
   10084:       fe842783         lw x15 -24 x8
   10088:       00178793         addi x15 x15 1
   1008c:       fef42423        sw x15 -24 x8
   10090:       fe842703         lw x14 -24 x8
   10094:       00900793         addi x15 x0 9
   10098:       fce7dee3        bge x15 x14 -36
   1009c:       fec42783        lw x15 -20 x8
   100a0:       00078513         addi x10 x15 0
   100a4:       01c12403         lw x8 28 x2
   100a8:       02010113         addi x2 x2 32
   100ac:       00008067         jalr x0 x1 0
```

1) Indique en qué posición del frame pointer se almacenan las variables locales b,c e i.
2) Resalte en el código donde se realiza la comparación i<10 (tenga en cuenta que no necesariamente compara con 10, pero el resultado de la comparación será el mismo)

C.09 *El siguiente programa en C*

```
void main(){
   while(1);
}
```

se genera el siguiente código ASM

```
00010054 <main>:
   10054:       ff010113        addi x2 x2 -16
   10058:       00812623         sw x8 12 x2
   1005c:       01010413         addi x8 x2 16
   10060:       0000006f        jal x0 0
```

Indique la función de la línea jal x0 0.

C.10 *Dado el programa en C del ejercicio* ***C.03*** *y el ASM que genera.*

Considere que el SP(x2)=0x7ffffff0 , FP(x8)=0 cuando comienza el programa.
Indique en qué posiciones reales de memoria se encuentran las variables A, B y C (de main) y las variables a y b de la función. Recomendamos primero calcular la dirección del frame pointer y luego calcular las direcciones usando esto de referencia.

D- Eficiencia

D.01 *Dada una arquitectura que posee los siguientes ciclos por tipo de instrucción:*

- *Load - 5 ciclos*
- *Store - 4 ciclos*
- *Registros - 4 ciclos*
- *Salto - 3 ciclos*

Calcule el CPI para un programa compuesto de la siguiente forma:

- *25% Loads*
- *10% Stores*
- *40% Registros*
- *25% Saltos*

D.02 *Dada una arquitectura que posee los siguientes ciclos por instrucción*

- *Load - 3 ciclos*
- *Store - 5 ciclos*
- *Registros - 2 ciclos*
- *Salto - 4 ciclos*

Calcule el CPI para un programa compuesto de la siguiente forma:

- *10% Loads*
- *5% Stores*
- *15% Registros*
- *70% Saltos*

D.03 *Dada una arquitectura que posee los siguientes*

- *Load - 5 ciclos*
- *Store - 4 ciclos*
- *Registros - 1 ciclo*
- *Salto - 3 ciclos*

Calcule el CPI para un programa compuesto de la siguiente forma:

- *30% Loads*
- *10% Stores*
- *40% Registros*
- *20% Saltos*

D.04 *Indique cuantos MIPS puede ejecutar una CPU operando con CPI=4 y una frecuencia de 200MHz.*

D.05 *Indique cuantos MIPS puede ejecutar una CPU operando con CPI=1 y una frecuencia de 50MHz.*

D.06 *Una CPU ejecuta 5 MIPS si opera a una frecuencia de 50 MHz, indique CPI.*

D.07 *Una CPU ejecuta 50 MIPS si opera a una frecuencia de 50 MHz, indique CPI.*

D.08 *Una CPU resuelve instrucciones con CPI=2. Si la misma ejecuta 10 MIPS indique cual es la frecuencia de operación.*

D.09 *Una CPU resuelve instrucciones con CPI=2. Si la misma ejecuta 5 MIPS indique cual es la frecuencia de operación.*

D.10 *El siguiente programa enciende un led durante un tiempo definido por el valor del registro x5. El programa enciende el led escribiendo un 1 en la posición de memoria 0x20000000, luego espera un tiempo relativo al valor de la variable "demora" y cuando ese tiempo se acaba apaga el led escribiendo un cero en la posición 0x20000000. Escriba la fórmula Clocks(demora) donde se puedan calcular cuántos pulsos de clocks toma la ejecución de este programa sabiendo que las instrucciones consumen 5 pulsos de clock. Nota: recuerde que* ***li*** *se puede traducir a una instrucción (lui o addi) o dos instrucciones (lui y addi) dependiendo del valor constante. También recuerde que* ***la*** *se traduce a dos instrucciones (auipc y addi).*

```
1  .data
2      demora:  .word 20
3  .text
4      la x5,demora
5      lw x5,0(x5)
6      li x6, 0x20000000
7      #encendemos el led
8      addi x7,x0,1
9      sw x7 , 0(x6)
10 loop:
11     #demora el valor de x5
12     addi x5,x5,-1
13     bge x5,x0,loop
14
15     #Apagamos el led
16     sw x0,0(x6)
```

D.11 *El siguiente programa calcula el largo en caracteres del string llamado texto. Asumiendo que todas las instrucciones consumen 4 ciclos de reloj, indique la fórmula para obtener clocks en base a la cantidad de caracteres de texto, o sea: Clock(strlen(texto))*

```
.data
    texto: .string "Hola Mundo"
    salida: .string "<-- strlen()="
.text
    la x5,texto
    addi x6,x0,-1

loop:
    lb x7,0(x5)
    addi x6,x6,1
    addi x5,x5,1
    bne x0,x7,loop

    #Imprime cantidad
    la a0,texto
    addi a7,x0,4
    ecall
    la a0,salida
    ecall
    addi a0,x6,0
    addi a7,x0,1
    ecall
```

D.12 *En una CPU con 5 fases de instrucción (IF-Fetch, ID-Decode, EXE-Execute, Mem, WB) donde cada fase consume un ciclo de clock y el formato de todas las instrucciones es uniforme, se incorpora un pipeline. Calcule si se produce una falla de datos e indique si es posible solucionarlo sin demorar el pipeline.*

a)

lw x5,0(x6)
addi x6,x5,1
addi x9,x9,1
slli x7,x7,2
addi x0,x0,0

Ejemplo:

Clock	1	2	3	4	5	6	7	8
lw x5,0(x6)	IF	ID	EXE	MEM	WB			
addi x6,x5,1		IF	ID					
addi x9,x9,1								
slli x7,x7,2								
addi x0,x0,0								

Se produce un problema cuando se decodifica addi x6,x5,1 ya que x5 no se calcula hasta el ciclo 5. Se puede solucionar adelantando las siguientes dos instrucciones.

Clock	1	2	3	4	5	6	7	8
lw x5,0(x6)	IF	ID	EXE	MEM	WB			
addi x9,x9,1		IF	ID	EXE	NOP	WB		
slli x7,x7,2			IF	ID	EXE	NOP	WB	
addi x6,x5,1				IF	ID	EXE	NOP	WB
addi x0,x0,0					IF	ID	EXE	NOP

b)

lw x5,0(x6)
addi x6,x5,1
addi x7,x7,1
slli x7,x7,2
addi x0,x0,0
jalr x0,x1,0

c)

lw x5,0(x6)
addi x9,x9,1
slli x7,x7,2
addi x6,x5,1
addi x0,x0,0