TALLER (PRIMER PARCIAL)

ARQUITECTURA DE COMPUTADORES

2018

Jornada Especial.

1. **¿Qué es una arquitectura de computadores?**

Lo que se denomina hardware de computadores consiste en circuitos electrónicos, visualizadores, medios de almacenamiento magnéticos y ópticos, equipos electromecánicos y dispositivos de comunicación. Por lo que la arquitectura de computadoras abarca la especificación del repertorio de instrucciones y las unidades hardware que implementan las instrucciones.

2. **Nombre las generaciones de los computadores y sus características más relevantes.**

Primera Generación - La válvula o tubo de vacío (1946-1954)

- 1943 -> ENIAC -> 30 toneladas -> 140 Kw.
- 1946 -> Máquina de John Von Neumann (Arquitectura utilizada hasta nuestros días)
- 1953 -> IBM crea algunas máquinas (701, 704, 709).

Segunda Generación - El Transistor (1957-1964)

- 1948 -> En los Laboratorios Bell inventaron el transistor y con este desarrollo se ganaron el premio Nobel de física.
- 1961 -> La empresa Digital Equipment Corporation (DEC) lanza un computador conocido como el PDP-1 con una RAM de 120 KB, basado en transistores y la venta al público era de USD $120.000. En este mismo año IBM lanza el IBM 7090 y tenía 32 KB de RAM
- 1964 -> Sale al mercado el CDC 1600 -> Primera Máquina paralela.

Tercera Generación - El Circuito Integrado (1964-1971)

- 1964 -> Aparece el S/360 -> Registros de 32 bits con direccionamiento de memoria de 2^24 posiciones.

- 1959 -> Texas Instruments desarrolla el primer circuito integrado con 12 transistores.

Cuarta Generación - Large Scale Integration (LSI), Very Large Scale Integration (VLSI), Microprocesador (1971-1983)

- Decenas de miles de transistores en un chip (LSI -> Large Scale Integration)
- Cientos de miles de transistores en un chip (VLS -> Very Large Scale Integration)
- Microprocesador.
- Caída de precios por lo que la IBM desarrolla el primer PC.

Quinta Generación - Very High Scale Integration (VHLSI), PC's (1990-????)

- Very High Large Scale Integration -> millones de transistores.

3. **Según Flynn ¿Cuál es la clasificación de las arquitecturas?**

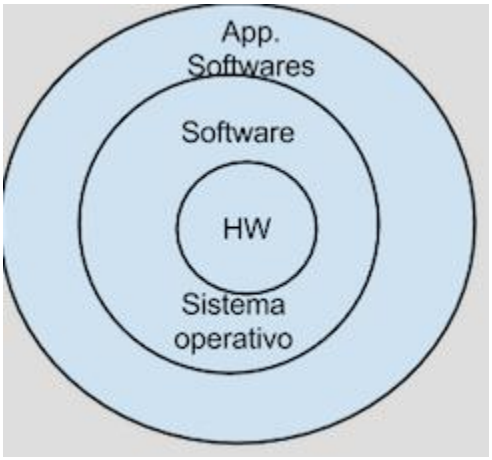
Este tipo de clasificación también es conocida como la taxonomía de Flynn y se clasifica en 4 tipos:

- **SISD - Single Instruction Single Data.** Este tipo de arquitecturas normalmente se ven en celulares de gama baja, generalmente presente en sistemas con un sólo núcleo. Se asigna la ejecución de una sola instrucción sobre un solo dato.
- **SIMD - Single Instruction Multiple Data.** En este caso una sola instrucción es ejecutada sobre un conjunto de datos. Este tipo de arquitecturas tiene como ejemplo a los procesadores vectoriales.
- **MIMD - Multiple Instruction Multiple Data.** Aquí lo que se tiene es que un conjunto de instrucciones diferentes se aplica sobre un conjunto de datos distintos. Un ejemplo de este tipo de arquitecturas puede verse en las GPU, o incluso en sistemas conectados en clúster.
- **MISD - Multiple Instruction Single Data.** No es una arquitectura típica

4. **Nombre las clases de aplicaciones de cómputo.**

   - Equipos de escritorio (PC's)
   - Equipos servidores. (Utilizados para almacenar bases de datos y a los cuales acceden muchos usuarios)
   - Tablets - dispositivos móviles
   - Wearables
   - Supercomputadores - HPC (High Performance Computing) - GPU : CPU (Sistemas heterogeneos)

5. **Muestre la clasificación de la jerarquía de un equipo de cómputo.**



6. **¿Qué es un compilador?**

   Es un programa que recibe como entrada un programa escrito en lenguaje de alto nivel y lo traduce a lenguaje ensamblador y luego a lenguaje máquina

7. **¿Defina qué es una instrucción?**

   Son las operaciones que puede realizar el procesador

8. **¿Cuales son los principios básicos de diseño de hardware de una arquitectura de cómputo, escriba una definición de cada uno?**

   - La simplicidad favorece la regularidad:

     Hacer diseños simples para garantizar que el procesador funcione bien siempre

   - Pequeño es más rápido:

     Hacer nuestros programas con la menor cantidad de recursos posibles para que se ejecuten más rápido

   - Hacer el caso común más rápido:

     Al realizar un diseño favorecer el caso frecuente sobre el infrecuente

   - Buenos diseños implican grandes compromisos:

9. **¿Qué es SPARCV8?**

   Es una arquitectura de computadores de 32 bits creada por Sun MicroSystems en 1985 diseñada principalmente para optimizar compiladores.

10. **¿Cuáles son las categorías de instrucciones de la arquitectura SPARCV8?**

Format	*op*	Instructions
1	1	CALL
2	0	Bicc, FBfcc, CBccc, SETHI
3	3	memory instructions
3	2	arithmetic, logical, shift, and remaining

- Load/Store (carga / almacenamiento)
- Aritmético-lógicas
- CTI (Control Transfer Instruction - Instrucciones de control de transferencia)
- Acceso a registros de estado
- Instrucciones de unidad de punto flotante
- Instrucciones de co-procesador

11. ¿Que tipos de registros se encuentran en SPARC V8?

Windowed Register Address	*r* Register Address
in[0] – in[7]	r[24] – r[31]
local[0] – local[7]	r[16] – r[23]
out[0] – out[7]	r[8] – r[15]
global[0] – global[7]	r[0] – r[7]

- Registros de Entrada: 8 registros de propósito general. Por estándar se sugiere que sean usados para recibir parámetros.
- Registros de Salida: 8 registros de propósito general. Por estándar se sugiere que sean usados para retornar valores.
- Registros Locales: 8 registros de propósito general. Por estándar se sugiere que sean usados para definir variables dentro de una función.
- Registros Globales: 8 registros de propósito general. Por estándar se sugiere que sean usados para almacenar variables globales

12. ¿Cuál es el número mínimo y máximo de registros que se puede implementar en la arquitectura SPARCV8?

De 32 a 520 registros

13. ¿Cuáles son las instrucciones de acceso a memoria de SPARCV8? de un ejemplo de cada uno.

LOAD: Carga un dato de memoria en un registro

Ld [%L0 + (30*4)], %L1

STORE: Escribe un dato en memoria

St %L1, [%O0 + (300*4)]

14. Represente los siguientes números en complemento a 2.

a.5
b.12890
c.56900
d.11
e.140

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

14-a. 5

																								256	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
CA2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1

14-b. 12890

																								256	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1	0
CA2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	0	1	0	0	1	1	0

14-C. 56900

																								256	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	0	1	0	0	0	1	0	0
CA2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	1	1	0	1	1	1	1	0	0

14-d. 11

																								256	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
CA2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

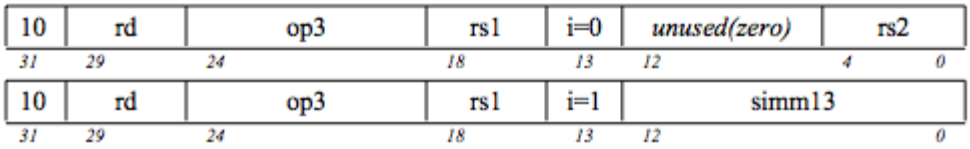
14-e. 140

																								256	128	64	32	16	8	4	2	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0
CA2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0

15. **Explique las instrucciones aritmético lógicas y su sintaxis en lenguaje ensamblador.**

opcode	*op3*	*operation*
AND	000001	And
ANDcc	010001	And and modify icc
ANDN	000101	And Not
ANDNcc	010101	And Not and modify icc
OR	000010	Inclusive Or
ORcc	010010	Inclusive Or and modify icc
ORN	000110	Inclusive Or Not
ORNcc	010110	Inclusive Or Not and modify icc
XOR	000011	Exclusive Or
XORcc	010011	Exclusive Or and modify icc
XNOR	000111	Exclusive Nor
XNORcc	010111	Exclusive Nor and modify icc

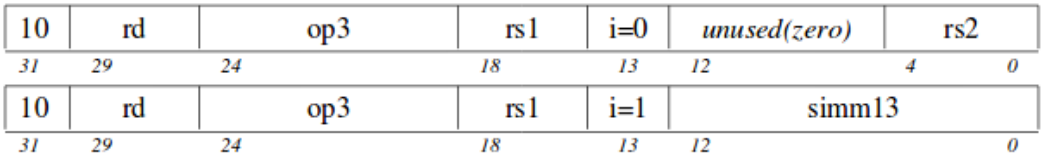
Format (3):



Suggested Assembly Language Syntax	
`and`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`andcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`andn`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`andncc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`or`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`orcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`orn`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`orncc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`xor`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`xorcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`xnor`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`xnorcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}

opcode	*op3*	*operation*
ADD	000000	Add
ADDcc	010000	Add and modify icc
ADDX	001000	Add with Carry
ADDXcc	011000	Add with Carry and modify icc

Format (3):

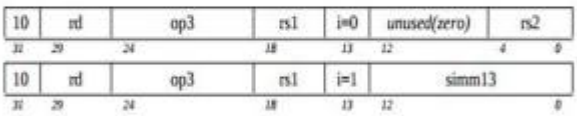


Suggested Assembly Language Syntax	
`add`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`addcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`addx`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`addxcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}

Instrucciones resta (Sub):

opcode	*op3*	*operation*
SUB	000100	Subtract
SUBcc	010100	Subtract and modify icc
SUBX	001100	Subtract with Carry
SUBXcc	011100	Subtract with Carry and modify icc

Format (3):



Suggested Assembly Language Syntax	
`sub`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`subcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`subx`	reg_{rs1} , *reg_or_imm* , reg_{rd}
`subxcc`	reg_{rs1} , *reg_or_imm* , reg_{rd}

16. Explique cada uno de los campos de los 3 formatos de la arquitectura SPARC V8.

Campos de los formatos

- **RD**: Es el registro fuente o el registro destino para unas instrucciones **load/store** ó alguna operación aritmético-lógica.
- **a**: Es un bit de anulación que evita que un salto sea tomado.
- **cond**: Codifican la condición que se evalúa para determinar si un salto se hace o no.
- **imm22**: Son los 22 bits que usa la instrucción **SETHI** para llevarlos a los 22 bits más significativos de un registro destino.
- **disp22 y disp30**: Valores de desplazamiento relativo dentro de la memoria de instrucciones utilizados por **BRANCH** y por las instrucciones **CALL**.
- **op3**: Es un campo de 6 bits que ayuda a codificar todas las instrucciones de formato 3.
- **i**: Es un bit que ayuda a determinar si el segundo operando de una instrucción arimética-lógica, es un valor inmediato o el contenido de un registro.
- **asi**: Estos 8 bits ayudan a codificar un identificador de espacio de direccionamiento (Address Space Identifier)
- **rs1**: El primer operando de una instrucción aritmético-lógica, Load/Store, o de corrimiento.
- **rs2**: Segundo operando de una instrucción si i = 1.
- **imm13**: Segundo operando de una instrucción si i = 0.
- **opf**: Codificación de operaciones de punto flotante.

17. ¿Qué diferencia hay entre el campo op, op2 y op3?

El **op** indica qué tipo de instrucción se va a realizar

El **op2** es una subclasificación de instrucciones y solo se usa cuando el op es 2 (formato2)

El **op3** es una subclasificación de instrucciones y solo se usa cuando el op es 3 (formato3)

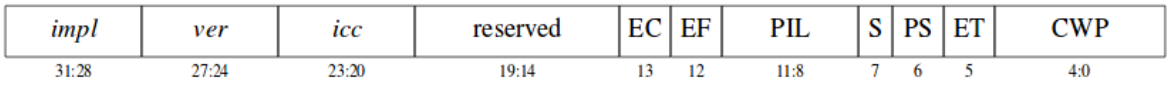
Format	*op*	Instructions
1	1	CALL
2	0	Bicc, FBfcc, CBccc, SETHI
3	3	memory instructions
3	2	arithmetic, logical, shift, and remaining

op2	Instructions
0	UNIMP
1	unimplemented
2	Bicc
3	unimplemented
4	SETHI
5	unimplemented
6	FBfcc
7	CBccc

18. ¿Qué es PSR ?, explique cada uno de sus campos.

Es el registro de estado de procesador con el cual controlo información importante sobre el estado de las operaciones.

Tiene 32 bits y se divide de la siguiente manera

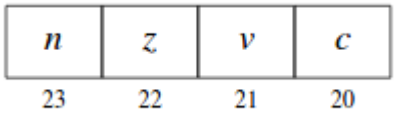


- **impl**: Implementación
- **versión**: Versión
- **ICC**: Esta dividido en 4 bits: **n**: Negativo cuando una operación da un número negativo, **Z**: se coloca en 1, cuando el resultado de la operación da cero, **v**: Over Flow se activa cuando el resultado de una operación da más de 32 bits y el bit **c**: Carry, se activa cuando se tiene acarreo.

- **reserved**: son un conjunto de bit no definidos dentro de la especificación de la arquitectura para que el diseñador los use de manera libre.
- **EC** (Enable co-processor): Indica si el procesador tiene unidad de co-procesamiento (FPGA, Tarjeta gráfica, etc.)
- **EF**: Es un bit que me indica si la arquitectura tiene unidad de punto flotante.
- **PIL**: Processor Interrupt level (la arquitectura SPARC V8 tiene interrupciones llamadas Traps)
- **S**: Supervisor; Indica el modo en el cual se ejecutó el procesador durante la instrucción anterior.
- **ET**: Habilita al procesador para que soporte traps.
- **CWP** Current Windows Pointer, Indica sobre que ventana estoy accediendo o guardando los datos.

19. ¿Qué es ICC y CWP?

ICC: Esta dividido en 4 bits: **n**: Negativo cuando una operación da un número negativo, **Z**: se coloca en 1, cuando el resultado de la operación da cero, **v**: Over Flow se activa cuando el resultado de una operación da más de 32 bits y el bit **c**: Carry, se activa cuando se tiene acarreo.



CWP Current Windows Pointer, Indica sobre que ventana estoy accediendo o guardando los datos.

20. ¿Qué es una instrucción sintética, de dos ejemplos?

Son instrucciones similares a las de la máquina pero que no existen en realidad. Es el compilador el que traduce cada instrucción sintética por otra equivalente a partir del conjunto de instrucciones máquina. Estas instrucciones existen para facilitar la programación y legibilidad de los programas.

Ejemplos:

MOV 5, %L0 ---→ esta instrucción ejecuta por debajo un OR entre 5 y %g0

CMP %L1, %L2--→ esta instrucción ejecuta por debajo una resta SUBicc entre L1 y L2

21. ¿Qué significa el campo a para una instrucción BRANCH?

Es un flag que cuando está en 1 ejecuta un Delay Slot (ejecuta la instrucción por debajo antes de hacer un salto) y cuando está en 0 salta directamente sin hacer la instrucción de abajo

22. ¿Para qué la instrucción CALL utiliza el registro %O7?

El registro %07 que es el registro 15 almacena el valor del program counter desde donde se hace el call para saber a dónde tiene que volver luego de que se termine la función invocada

23. Convertir el siguiente programa en lenguaje de máquina a lenguaje ensamblador y luego a lenguaje de alto nivel:

```
10100000000100000010000000000101
10100010000100000011111111111010
10010000000001000100000000010000

int main(){

  int x =   5;                    MOV 5,  %L0
  int y = -6;                     MOV -6, %L1

  return x + y;                   ADD %L0, %L1, %O0
}
```

24.Solucione el siguiente programas en lenguaje ensamblador, lenguaje de máquina y hexadecimal, además coloque su dirección de memoria.

```
int main(){
int i = 5;                    0000  MOV  5, %L0
int b = -4;                   0004  MOV -4, %L1
int c[100];
int d[20];
c[5] = i + 2;                 0008  ADD %L0,2, %L0
                              000C  ST %L0, [%L2 +(5*4)]

d[4] = b + 3;                 0010  ADD %L1, 3, %L1
                              0014  ST %L1, [%L3 + (4*4)]

Return c[5] + d[4] -i         0018  LOAD [%L2 +(5*4)], %L4
                               001C  LOAD [%L3 + (4*4)], %L5
                               0020  ADD %L4, %L5, %L4
                               0024  SUB %L4, %L0, %O0
}
```

PUNTO 24

OR 5, %G0, %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000010**	**00000**	**1**	**0000000000101**

0XA0102005

OR -4, %G0, %L1

OP	RD	OP3	RS1	I	SIMM 13
10	**10001**	**000010**	**00000**	**1**	**1111111111100**

0XA2103FFC

008 ADD %L0,2, %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000000**	**10000**	**1**	**0000000000010**

0XA0042002

ST %L0, [%L2 +(5*4)]

OP	RD	OP3	RS1	I	SIMM 13
11	**10010**	**000100**	**10000**	**1**	**0000000010100**

0XE4242014

3, %L1

OP	RD	OP3	RS1	I	SIMM 13
10	**10001**	**000000**	**10001**	**1**	**0000000000110**

0XA2043006

ST %L1, [%L3 + (4*4)]

OP	RD	OP3	RS1	I	SIMM 13
11	**10011**	**000100**	**10001**	**1**	**0000000010000**

0XE6246010

LOAD [%L2 +(5*4)], %L4

OP	RD	OP3	RS1	I	SIMM 13
11	**101000**	**000000**	**10010**	**1**	**0000000010100**

OX E804A014

LOAD [%L3 + (4*4)], %L5

OP	RD	OP3	RS1	I	SIMM 13
11	**10101**	**000000**	**10011**	**1**	**0000000010000**

OX EA04E010

20 ADD %L4, %L5, %L4

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**10100**	**000000**	**10100**	**0**	**00000000**	101001

OX A8050015

24 SUB %L4, %L0, %O0

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**01000**	**000100**	**10100**	**0**	**00000000**	10000

OX 90250010

25. Convierta el siguiente código a lenguaje ensamblador, máquina SPARC V8 y hexadecimal.

a.

```
int main(){
int a = 8;                    a→%L0
int b = -14800;               b→%L1
int c = 33;                   c→%L2
if((a+b)<=b*16){
c=a+(b*2);
}
else{
return b-78;
}
return a+c;
}
```

0000 mov 8, %L0

0004 SETHI 4195289, %L1

0008 OR %L1 , 560 , %L1

000C mov 33, %L2

0010 add %L0 , %L1 , %L3

0014 sll %L1 , 4 , %L4

0018 cmp %L3 , %L4

001C bg a, else

0020 sll %L1 , 1 , %L5

0024 add %L5 , %L0 , %L2

0028 ba a , fin

Else

002C sub %L1 , 78 , %O0

0030 ba , a , refin

Fin

0034 add %L0 , %L2 , %O0

Refin

0038 Nop

punto 25-a

0000 OR 8, %G0 , %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000010**	**00000**	**1**	**0000000001000**

OX A0102008

0004 SETHI 4195289, %L1

OP	RD	OP2	IMM 22
00	**10001**	**100**	**11111111111111110001**

OX 433FFFF1

0008 OR %L1 , 560 , %L1

OP	RD	OP3	RS1	I	SIMM 13
10	**10001**	**000010**	**10001**	**1**	**0001000110000**

OX A2146230

000C OR 33, %GO , %L2

OP	RD	OP3	RS1	I	SIMM 13
10	**10010**	**000010**	**00000**	**1**	**0000000100001**

OX A4102021

0010 add %L0 , %L1 , %L3

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**10011**	**000000**	**10000**	**0**	**00000000**	10001

OX A6040011

0014 sll %L1 , 4 , %L4

OP	RD	OP3	RS1	I	SIMM 13
10	**10100**	**100101**	**10001**	**1**	**0000000000100**

OX A92C6004

0018 SUBicc %L3 , %L4 , %G0

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**00000**	**010100**	**10011**	**0**	**00000000**	10100

OX 80A4C014

001C bg a, else

OP	A	COND		DISP 22
00	1	1010	010	00000000000000000101100

OX 3480002C

0020 sll %L1 , 1 , %L5

OP	RD	OP3	RS1	I	SIMM 13
10	**10101**	**100101**	**10001**	**1**	**0000000000001**

OX AB2C6001

0024 add %L5 , %L0 , %L2

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**10010**	**000000**	**10101**	**0**	**00000000**	10000

OX A4054010

0028 ba a , fin

OP	A	COND		DISP 22
00	1	1000	010	00000000000000000110100

OX 30800034

002C sub %L1 , 78 , %O0

OP	RD	OP3	RS1	I	SIMM 13
10	**01000**	**000000**	**10001**	**1**	**0000001001110**

OX 9004604E

0030 ba , a , refin

OP	A	COND		DISP 22
00	1	1000	010	00000000000000000111000

OX 30800038

0034 add %L0 , %L2 , %O0

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**01000**	**000000**	**10000**	**0**	**00000000**	10010

OX 90040012

0038 Nop

OP	RD	OP2	IMM 22
00	**000000**	**100**	**0000000000000000000000**

OX 1000000

b.

```
int main(){
int a = 8;
int b = -10;
if(a!=b){
return c/16;
}
else{
return b*32;
}
}
```

$L0 →A

$L1 →B

$L2 →C

0000 MOV 8, %L0

0004 MOV -10, %L1

0008 CMP %L0, %L1

000C BE, A, ELSE

0010 SRL, %L2,4 %O0

0014 BA, A, FINPROGRAMA

ELSE

0018 SLL %L1, 5, %O0

FINPROGRAMA

001C NOP

25 B.

0000 OR 8, %G0, %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000010**	**00000**	**1**	**00000000001000**

OX A0102008

0004 OR -10, %G0 %L1

OP	RD	OP3	RS1	I	SIMM 13
10	**10001**	**000010**	**00000**	**1**	**1111111110110**

OX A2103FF6

0008 SUBICC %L0, %L1, &G0

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**00000**	**010100**	**10000**	**0**	**00000000**	10001

OX 80A40011

000C BE, A, ELSE

OP	A	COND		DISP 22
00	1	0001	010	0000000000000000011000

OX 22800018

L0 SRL, %L2,4 %

OP	RD	OP3	RS1	I	SIMM 13
10	**01000**	**100110**	**10010**	**1**	**0000000000100**

OX 9134A004

0014 BA, A, FINPROGRAMA

OP	A	COND		DISP 22
00	1	1000	010	0000000000000000011100

OX 3080001C

0018 SLL %L1, 5, %O0

OP	RD	OP3	RS1	I	SIMM 13
10	**01000**	**100101**	**10001**	**1**	**0000000000101**

OX 912C6005

001C NOP

OP	RD	OP2	IMM 22
00	**000000**	**100**	**00000000000000000000**

OX 1000000

c.

```
int main(){
int a = -21180;
}
```

%L0 →A

0000 SETHI 2097131, %L0

0004 OR %L0, 324, %L0

25 C.

0000 SETHI 2097131, %L0

OP	RD	OP2	IMM 22
00	**10000**	**100**	**11111111111111101011**

OX 213FFFEB

0004 OR %L0, 324, %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000010**	**10000**	**1**	**000101000100**

OX 500A1144

26. Convierta el siguiente código a lenguaje ensamblador, máquina SPARC V8 y hexadecimal.

```
int test(int x, int y, int w){
int z;
z = x - y + w*4;
return z + 2;
}

int main(){
int a = 4, b = 2, c = -15600;
int x = test(a,b,c);
return x + 45;
}
```

%I0 →X

%I1 →Y

%I2 →W

%L0 →Z

0000 SUB %I0, %I1, %I3

0004 SLL %I2, 2, %I2

0008 ADD %I3, %I2, %L0

000C ADD %L0, 2, %O0

0010 JMPL %07, 8, %G0

0014 NOP

0018 MOV 4, %I0

001C MOV 2, %I1

0020 SETHI, 4194288, $I2

0024 OR %I2, 784, %I2

0028 CALL TEST

002C NOP

0030 ADD %IO, 45, %O0

0000 SUB %I0, %I1, %I3

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**11011**	**000100**	**11000**	**0**	**00000000**	11001

OX B6260019

004 SLL %I2, 2, %I

OP	RD	OP3	RS1	I	SIMM 13
10	**11010**	**100101**	**11010**	**1**	**0000000000010**

OX B52EA002

008 ADD %I3, %I2, %

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**10000**	**000000**	**11011**	**0**	**00000000**	11010

OX A006C01A

00C ADD %L0, 2, %(

OP	RD	OP3	RS1	I	SIMM13
10	**01000**	**000000**	**10000**	**1**	**0000000000010**

OX 90042002

10 JMPL %07, 8, %

OP	RD	OP3	RS1	I	SIMM13
10	**00000**	**111000**	**01111**	**1**	**0000000001000**

OX 81C3E008

0014 NOP

OP	RD	OP2	IMM 22
00	**000000**	**100**	**0000000000000000000000**

OX 1000000

0018 OR, %G0, 4, %I0

OP	RD	OP3	RS1	I	SIMM 13
10	**11000**	**000010**	**00000**	**1**	**0000000000100**

OX B0102004

001C OR, %G0, 2, %I1

OP	RD	OP3	RS1	I	SIMM 13
10	**11001**	**000010**	**00000**	**1**	**0000000000010**

OX B2102002

0020 SETHI, 4194288, $I2

OP	RD	OP2	IMM 22
00	**11010**	**100**	**11111111111111110000**

OX 353FFFF0

0024 OR %I2, 784, %I2

OP	RD	OP3	RS1	I	SIMM 13
10	**11010**	**000010**	**11010**	**1**	**0001100010000**

OX B416A310

0028 CALL TEST

OP	disp30
01	**00000000000000000000000000000000**

OX 40000000

002C NOP

OP	RD	OP2	IMM 22
00	**000000**	**100**	**00000000000000000000**

OX 1000000

0030 ADD %IO, 45, %O0

OP	RD	OP3	RS1	I	SIMM13
10	**01000**	**000000**	**11000**	**1**	**0000000101101**

OX 9006202D

27. Implemente la función Pot en lenguaje de alto nivel,lenguaje ensamblador SPARC V8 y lenguaje de máquina SPARC V8 que realice la potencia de dos números enteros sin signo realizando llamados a la función multiplicacion hecha en clase.

```
int multiplicar (int a, int b){
        int acumulador = a;
        int i = 0;
        for(i = 0; i < b – 1; i++){
                acumulador = acumulador + a;
        }
        return acumulador;
int potencia (int x, int y){
        int acumulador = x;
        int i = 0;
        for(i=0; i < y -1; i++){
                acumulador = multiplicacion(acumulador, x);
        }
        return acumulador;
}

int main(){
        int a = 2;
        int b = 5;
        int c;
        c = potencia(a,b);
```

return potencia

}

MULTIPLICACION

0000 MOV 0, %L4

OP	RD	OP3	RS1	I	SIMM 13
10	**10100**	**000010**	**00000**	**1**	**0000000000000**

OX 54081000

0004 SUB %LO,1,%L5

OP	RD	OP3	RS1	I	SIMM 13
10	**10101**	**000000**	**10000**	**1**	**0000000000001**

OX AA042001

FOR1

0008 CMP %L4, %L5

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**00000**	**010100**	**10100**	**0**	**00000000**	10101

OX 80A50015

000C BGE, a, ENDFOR1

OP	A	COND		DISP 22
00	1	1011	010	00000000000000000011100

OX 3680001C

0010 ADD %O0, %L0, %O0

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**01000**	**000000**	**01000**	**0**	**00000000**	10000

OX 90020010

0014 ADD %L4, 1, %L4

OP	RD	OP3	RS1	I	SIMM 13
10	**10100**	**000000**	**10100**	**1**	**0000000000001**

OX A8052001

0018 BA, a, FOR1

OP	A	COND		DISP 22
00	1	1000	010	00000000000000000001000

OX 30800008

ENDFOR1

001C JMPL %07, 8, %G0

OP	RD	OP3	RS1	I	SIMM13
10	**00000**	**111000**	**01111**	**1**	**0000000001000**

OX 81C3E008

POTENCIA

0020 MOV %07, %06

OP	RD	OP3	RS1	I	SIMM 13
10	**01110**	**000010**	**01111**	**1**	**0000000000000**

OX 9C13E000

0024 MOV %LO, %O0

OP	RD	OP3	RS1	I	SIMM 13
10	**01000**	**000010**	**10000**	**1**	**0000000000000**

OX 90142000

0028 MOV 0, %L2

OP	RD	OP3	RS1	I	SIMM 13
10	**10010**	**000010**	**00000**	**1**	**0000000000000**

OX A4102000

002C SUB %L1, 1, %L3

OP	RD	OP3	RS1	I	SIMM 13
10	**10011**	**000100**	**10001**	**1**	**0000000000001**

OX A6246001

FOR2

0030 CMP %L2, %L3

OP	RD	OP3	RS1	I	UNUSED (ZERO)	RS2
10	**00000**	**010100**	**10010**	**0**	**00000000**	10011

OX 80A48013

0034 BGE, a, ENDFOR2

OP	A	COND		DISP 22
00	1	1011	010	00000000000000001001000

OX 36800048

0038 CALL MULTIPLICACION

OP	disp30
01	**00000000000000000000000000000000**

OX 40000000

003C NOP

OP	RD	OP2	IMM 22
00	**000000**	**100**	**0000000000000000000000**

OX 1000000

0040 ADD %L2, 1, %L2

OP	RD	OP3	RS1	I	SIMM 13
10	**10010**	**000000**	**10010**	**1**	**0000000000001**

OX A404A001

0044 BA, A, FOR2

OP	A	COND		DISP 22
00	1	1000	010	00000000000000000110000

OX 30800030

ENDFOR2

0048 JMPL %06, 4, %G0

OP	RD	OP3	RS1	I	SIMM13
10	**00000**	**111000**	**01110**	**1**	**0000000000100**

OX 81C3A004

MAIN

004C MOV 2, %L0

OP	RD	OP3	RS1	I	SIMM 13
10	**10000**	**000010**	**00000**	**1**	**0000000000010**

OX A0102002

0050 MOV 5, %L1

OP	RD	OP3	RS1	I	SIMM 13
10	**10001**	**000010**	**00000**	**1**	**000000000101**

OX A2102005

0054 CALL POTENCIA

OP	disp30
01	**00000000000000000000000100000**

OX 40000020

0058 NOP

OP	RD	OP2	IMM 22
00	**000000**	**100**	**0000000000000000000000**

OX 1000000

28. Implemente una función Fact en lenguaje de alto nivel, lenguaje ensamblador SPARC V8 y lenguaje de máquina SPARC V8 que calcule el factorial de un número entero sin signo.

29. Hacer el nPC Y PC en VHDL.