Taller Parcial 1

1)que es una arquitectura de computadores?

Una arquitectura de computadores es la estructura conceptual y también operacional con la cual un computador puede funcionar dado que sin dicha arquitectura el centro de cómputo no podría operar ya que no constaría por decirlo así con su lenguaje primario de como hacer y representar sus operaciones internas tales como operaciones lógicas matemáticas e instrucciones de memoria, la arquitectura esta formada por 5 componentes fundamentales los cuales son, procesador, memoria RAM , disco duro, dispositivos de entrada y salida y software

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

Primera generación 1951-1958

-Se usaban los tubos de vacío para procesar la información que manejaba el computador
-Se utilizaban tarjetas perforadas para introducir los datos y los operandos al equipo de cómputo.
-se utilizaban cilindros magnéticos para poder guardar la información
-eran muy grandes y pesadas, también eran muy lentas y su consumo energético era muy grande por lo cual también generaba gran cantidad de calor
-se empieza a usar el sistema binario para su representación de datos

Segunda generación 1958 – 1964

-Ya se podían programar algunas computadoras con cintas perforadas o bien con un tablero cableado
-empezaron a usar transistores envés de tubos de vacío y así poder reducir el tamaño
-ya no se usaban los cilindros magnéticos sino anillos lo cual también redujo mucho su tamaño
-aunque seguían siendo lentas, su rendimiento mejor dado que se mejoraron mucho los programas hechos en la primera generación
-se crearon nuevos lenguajes de programación como el COBOL FORTRAN

Tercera generación 1964-1971

-Se desarrollaron circuitos integrados para el procesado y asi también redujeron mucho el tamaño
-se desarrollaron los chips para almacenar información, un chip es una pieza muy pequeña hecha de cilicio en la cual van varios componentes los cuales se denominan semi conductores.
-Gracias a todos estos componentes nuevos se había creado el multiprocesamiento dado que las computadoras ya eran capaces de hacer tareas de procesamiento y análisis matemático de forma paralela
-Emerge la industria del software

Cuarta generación 1971-1988

-se desarrolla el microprocesador
-el circuito integrado logra introducir aun mas circuitos en un chip
-LSI large scale integration circuit
-VLSC very large-scale integration circuit
-se desarrolla el computador personal
-Se desarrollan las super computadoras

Quinta generación 1988- a la actualidad

-se genera una gran competencia por crear microelectrónica para asi poder mantener el poderoso hardware que se esta implementando
-se trata de hacer la comunicación con un computador de una manera mas cotidiana no con códigos o lenguajes de control especializados
-países dedican sus fondos a la verdadera innovación de la computación dado que sus posibilidades no parecen tener limites

3)Según Flynn cual es la clasificación de las arquitecturas

La clasificación o más bien conocida como la taxonomía de Flynn es una matriz que cruza numero de instrucciones con su respuesta en un dato o múltiples datos teniendo así 4 clasificaciones la matriz es

	Una instrucción	Múltiples instrucciones
Un dato	SISD	MISD
Multiples datos	SIMD	MIMD

SISD, que es decir single instruction single data, quiere decir que la arquitectura trabajaría dándole una sola instrucción y esa misma arquitectura devolverá un solo resultado

MISD, que quiere decir multiples instructions single data, que quiere decir que esta arquitectura recibe muchas instrucciones pero solamente arrojara un solo resultado

SIMD, que dice single instruction multiple data, o bien que la arquitectura toma muchas instrucciones y regresa un solo dato

MIMD, que quiere decir multiple instructions multiple data, y dicta que esta arquitectura recibe muchas instrucciones y su respuesta serian muchos datos

4) Nombre las clases de aplicaciones de cómputo.

-Aplicaciones de Sistema de control y automatización industrial

-Aplicaciones ofimáticas

-Software educativo

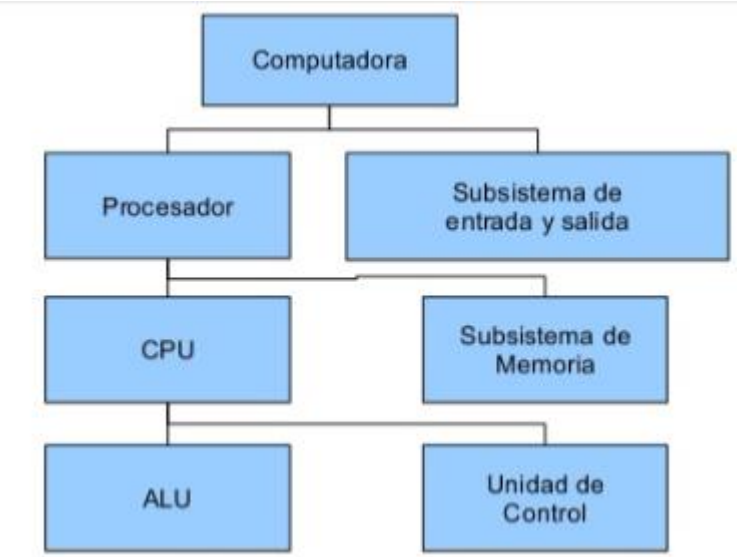
-Software médico

-software de Cálculo Numérico

-Software de Diseño Asistido (CAD)

-Software de Control Numérico (CAM)

5)Muestre la clasificación jerárquica de un equipo de computo



6)Que es un compilador?

Un compilador hace papel de traductor entre lo que se programa en la maquina y el procesador de la maquina haciendo posible que lo que sea programado en el computador sea llevado a lenguaje de maquina el cual es legible para el procesador, el cual es representado en código binario.

7)Defina que es una instrucción

Se le denomina instrucción al conjunto de datos insertados en una secuencia estructurada o especifica que el procesador lee, ejecuta y muestra un resultado, se debe tener una gran variedad de instrucciones que se usan como medio para controlar el procesador o CPU

8) ¿Cuáles son los principios básicos de diseño de hardware de una arquitectura de cómputo, escriba una definición de cada uno?

-La simplicidad favorece la regularidad

-Entre más pequeño es más rápido

-Hacer el caso común más rápido

-Buenos diseños demandan grandes compromisos

9)que es Sparc V8?

Sparc(Scalable processor Architecture), es una arquitectura de microprocesadores que trabajar a 32 y 64 bits, dicha arquitectura se basa en la computación reducida de conjuntos o bien llamada por sus siglas en ingles RISC, esta arquitectura a tomado mucha fuerza dado que se puede usar libremente y también gracias a su poder para el diseño de hardware eh incluso se ha usado en programación científica con muy buenos resultados y Sparc como arquitectura permite la creación de espectros de chip a un gran rendimiento y precio.

10)cuales son las categorías de instrucciones de la arquitectura sparc v8?

1) Load/store

2) instrucciones Aritmeticas/logica/shift

3) instrucciones de control de transferencia

4)insturcciones de control de registro Read/write

5) operación de punto flotante

6) operaciones de coprocesador o coprocesamiento

11)que tipos de registros se encuentran en la arquitectura spark v8?

Registros de estado del procesador

-Ventana máscara inválida (WIM)
-Registro de la base de trampas (TBR)
-Registro de multiplicación / división (Y)
-Contadores de programa (PC, nPC)
-Registros de estado auxiliar (ASR) dependientes de la Implementación
-IU Deferred-Trap Queue dependiente de la implementación

Registros de estados de la FPU

-registro estatal de punto flotante

-cola de trampa difereida de punto flotante dependiente de la implementación FQ

Registros de estado del coprocesador

-Registro estatal del de coprocesamiento dependiente de la implementación (CSR)
-Coprocesador dependiente de la implementación Deferred trap Queue(CQ)

12) cual es el mínimo y máximo numero de registros que se puede implementar en la arquitectura sparc v8?

Se puede implementar un minimo de 40 registros y un máximo de 520 registros

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

Las instrucciones de memorio de la arquitectura sparc v8 son las instrucciones LOAD/STORE, con Load se puede acceder a alguna dirección de memoria en la cual se almacena un dato y sacar dicho dato y guardarlo en otro registro, y con Store se puede tomar el dato de un registro y guardarlo en una dirección de memoria deseada

Ejemplo load

C[8]+2;

Se le asigna un registro L0 a la letra C, y se accede a la dirección de memoria 8, este numero se debe multiplicar por 4 dado que la memoria esta divida de a 4 bits, entonces tendremos la dirección 8*4 de dicho registro para asi poder cargar (load) el dato que se encuentra almacenado allí, para después poder guardarlo en otro registro por decir L1

Ld [%L0+(8*4)],L1

Ejemplo store

C[5] = 7;

Se le asigna un registro L0 a la letra C, y se apunta a la dirección de memoria 5, este numero se debe multiplicar por 4 dado que la memoria esta divida de a 4 bits, entonces tendremos la dirección 5*4 de dicho registro para asi poder apuntar y almacenar(Store) el dato que queremos almacenar en dicha dirección de memoria

ST 7,[L0+(5*4)]

14)Represente los siguientes números en complemento a 2

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

a)5

0101

1010
1

1011 = complemento a dos para 5

b)12890

11001001011010

00110110100101
1

00110110100110 = complemento a dos para 12890

c) 56900
1101111001000100

0010000110111011
1

0010000110111100 = complemento a dos para 56900

d)11

1011

0100
1

0101 = complemento a dos para 11

e)140

10001100

01110011
1

01110100 = complemento a dos para 140

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

ADD-SUB-AND-OR
Syntax ADD -- >ADD rs1, rs2, rd
Syntax SUB -- >SUB rs1, rs2, rd
Syntax AND -- >AND rs1, rs2 ó inm, rd
Syntax OR -- >OR rs1, rs2 ó inm, rd

16)Explique cada uno de los formatos de la arquitectura spark V8

Formato 1 - Instrucciones de salto:
Sólo hay una instrucción en la máquina SPARC que es de la forma número uno llamada (CALL) o instrucción de llamada.

```
FRTO 1
OP   DISP 30
```

01	

Formato 2 - Intrucciones Branch y Sethi:

```
FRTO 2
OP   a  COND  OP2  DISP22
```

00				

```
OP   RD     OP2     OPERATION
```

00		100	

Formato 3: - Instrucciones algebraicas:
Este tipo de instrucciones son las más comunes. Estas son las instrucciones algebraicas.

OP RD OP3 RS1 I SIMM13/RS2

10	10000	000010	00000	1	0000000000101	
10	01000	000000	10001	0	00000000	10000

17)que diferencia hay entre op, op2, op3?

Op define los tipos de operaciones los cuales se van a realizar y todo esto también depende del formato le cual se este manejando

Op se encuentra en cada uno de los formatos y este cambia dependiendo de cual operación se vaya a realizar

Op2: solamente se encuentra en el formato 2 y nos dice que tipo de instrucción va a usar dicho formato ya sea sethi o Branch

Op3: solo se utiliza en el formato y cambia dependiendo de la operación que se vaya a realizar, inckuso hay operaciones con el mismo op3 pero el formato 3 se ayuda con el op ya que puede cambiar dependiendo si se requiere utilizar una instrucción aritmético lógica o una instrucción de meoria.

18)que es PSR, explique cada uno de sus campos

El PSR de 32 bits contiene varios campos que controlan el procesador y mantienen el estado información. Se puede modificar mediante SAVE, RESTORE, Ticc y RETT instrucciones, y por todas las instrucciones que modifican los códigos de condición. Las instrucciones privilegiadas RDPSR y WRPSR leen y escriben el PSR directamente.

PSR_implementation (impl):

Los bits 31 a 28 están cableados para identificar una implementación o clase de implementaciones de la arquitectura El hardware no debería cambiar este campo en respuesta a una instrucción WRPSR. Juntos, los campos PSR. impl y PSR. ver definir una implementación única o una clase de implementaciones de la arquitectura.

PSR_version (ver):

Los bits 27 a 24 dependen de la implementación. El campo ver es cualquiera cableado para identificar una o más implementaciones particulares o es legible y campo de estado modificable cuyas propiedades dependen de la implementación.

PSR_integer_cond_codes (icc):

Los bits 23 a 20 son los códigos de condición de IU. Estos bits son modificados por instrucciones aritméticas y lógicas cuyos nombres terminan con las letras cc (p. ej., ANDcc), y por la instrucción WRPSR. Las instrucciones Bicc y Ticc causan transferencia de control basada en el valor de dichos bits

PSR_negative (n):

El bit 23 indica si el resultado ALU del complemento de 2 de 32 bits fue negativo para la última instrucción que modificó el campo icc. 1 = negativo, 0 = no negativo.

PSR_zero (z):
El bit 22 indica si el resultado ALU de 32 bits fue cero para la última instrucción eso modificó el campo icc. 1 = cero, 0 = distinto de cero.

PSR_overflow (v):

El bit 21 indica si el resultado de ALU estaba dentro del rango de (era representable in) notación de complemento de 2 de 32 bits para la última

instrucción que modificó la campo icc 1 = desbordamiento, 0 = desbordamiento.

PSR_carry (c):

El bit 20 indica si se realizó un complemento de 2 (o un préstamo) para el última instrucción que modificó el campo icc. Carry está configurado además si hay un llevar a cabo el bit 31. Carry se establece en la resta si hay préstamos en el bit 31. 1 = llevar, 0 = sin llevar

PSR_reserved:

Los bits 19 a 14 están reservados. Cuando se lee mediante una instrucción RDPSR, estos bits entregar ceros. Para compatibilidad futura, el software supervisor solo debe emitir Instrucciones WRPSR con valores cero en este campo.

PSR_enable_coprocessor (EC)

El bit 13 determina si el coprocesador dependiente de la implementación está habilitado.
Si está deshabilitado, una instrucción de coprocesador se bloqueará. 1 = habilitado, 0 = deshabilitado. Si una implementación no es compatible con un coprocesador en hardware, PSR.EC debería siempre se lee como 0 y se debe ignorar.

PSR_enable_floating-point (EF)

El bit 12 determina si la FPU está habilitada. Si está deshabilitado, un punto flotante la instrucción atrapará. 1 = habilitado, 0 = deshabilitado. Si una implementación no lo hace admite una FPU de hardware, PSR.EF siempre debe leer como 0 y escribe en ella ser ignorado.

PSR_proc_interrupt_level (PIL)

Los bits 11 (el bit más significativo) hasta 8 (el bit menos significativo) identifican el nivel de interrupción por encima del cual el procesador aceptará una interrupción.

PSR_supervisor (S)

El bit 7 determina si el procesador está en modo supervisor o
usuario. 1 = supervisor
modo, 0 = modo de usuario.

PSR_previous_supervisor (PS)

El bit 6 contiene el valor del bit S en el momento de la captura más
reciente

PSR_enable_traps (ET)

El bit 5 determina si las trampas están habilitadas. Una trampa
reinicia automáticamente ET a 0.
Cuando ET = 0, se ignora una solicitud de interrupción y una captura
de excepción causa la IU detener la ejecución, lo que generalmente
resulta en una trampa de reinicio que reanuda la ejecución en
dirección 0. 1 = trampas habilitadas, 0 = trampas deshabilitadas.

PSR_current_window_pointer (CWP)

Los bits 4 (el MSB) a 0 (el LSB) comprenden el puntero de la
ventana actual, un contador que identifica la ventana actual en los
registros r. El hardware disminuye el CWP en las trampas y
GUARDA las instrucciones, y lo incrementa en Instrucciones
RESTORE y RETT (módulo NWINDOWS).

19)Que es el ICC y el CWP?

ICC quiere decir integer conditional code, y es el bit que se activa después de una comparación tenemos los bits N Z V C, N se activa si la coparacion da un numero negativo, Z se activa si la comparación da cero, V se activa si la operación da un overflow o bn un numero que no soporte la arquitectura, y el C se activa en caso de que se lleve una cantidad para añadir al siguiente numero es decir un carrie.

CWP es el current Windows pointer, y nos dice en que ventana estamos operando actualmente dado que sparc es un arquitectura con ventaneo.

20)Que es una instrucción sintética de dos ejemplos.

Es la forma corta para representar una instrucción en SPARV8
Ejemplo 1
OR %G0 5 %L0 -- -> MOV 5 %L0
Ejemplo 2
SUBCC %L0 0 %G0 -- > CMP %L0 0

21)que significa el campo a para una instrucción Branch?

Significa que el bit a esta activado y cuando el Branch salte a su destino no se llera la línea de instrucción que tiene debajo del Branch

22)para que la instrucción call utiliza el registro 07?

La utiliza para guardar la dirección de memoria a la cual debe regresar después de haber finalizado el llamado que la función call hizo.

23)

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	10000	000010	00000	1	0000000000101	
10	10001	000010	00000	1	1111111111010	
10	01000	000000	10001	0	00000000	10000

LENGUAJE ENSAMBLADOR

OR %G0, 5, %L0

OR %G0, -6, %L1

ADD %L1,%L0,0O

LENGUAJE ALTO NIVEL

```
INT (){
        INT A = 5;
        INT B = -6;
        C = (-6+5);
        RETURN C;
        }
```

COMPLEMENTO A 2 PARA -6

1111111111010

0000000000101

1

00000000000110

24)

```
int main(){
        int i = 5;
        int b = -4;
        int c[100];
        int d[20];
        c[5] = i + 2;
        d[4] = b + 3;
        return c[5] + d[4] -i
}

0000 MOV 5, %L0
0004 MOV -4, %L1
0008 LD [%L2 +(100*4)],L3
000C LD [%L4 +(20*4)],L5
0010 ADD %L0,2,%L6
0014 ST %L6,[%L2+(5*4)]
0018 ADD %L1,3,%L7
001C ST %L1,[%L4+(4*4)]
0020 LD [%L2 +(5*4)],L6
0024 LD [%L4 +(4*4)],L7
0028 ADD L6,L7,L6
002C SUB L6,L0,O0
```

OP	RD	OP3	RS1	I	SIMM13/RS2
10	10000	000010	00000	1	0000000000101
10	10001	000010	00000	1	1111111111100
11	10011	000000	10010	1	0000001100100
11	10101	000000	10100	1	0000000010100
10	10110	000000	10000	1	0000000000010
11	10110	000100	10100	1	0000000000101

10	10111	000000	10001	1	0000000000011
11	10111	000100	10100	1	0000000000100
11	10110	000000	10010	1	0000000000101
11	10111	000000	10100	1	0000000000100

10	10110	000000	10110	0	00000000	10111
10	01000	000100	10110	0	00000000	10000

Complemento a dos para el -6

0000000000100

1111111111011

1

1111111111100

25)

A)

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

LENGUAJE ENSAMBLADOR

```
       0000 MOV 8,%L0
       0004 STEHI -29,%L1
       0008 OR %L1,48,L1
       000C MOV 33,%L3
       0010 ADD %L0,%L1,%L4
       0014 SLL %L1,4,%L5
       0018 SUBcc %L4,%L5,G0 = CMP %L4,L5
       001C BGE a SUMA
       0020 SUB %L1,78,%L1
       0024 BA a EXIT
SUMA   0028 SLL %L1,1,L6
       002C ADD %L0,%L6,%L3
EXIT   0030 ADD %L0,%L3,O0
```

LENGUAJE MAQUINA

FRTO3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	10000	000010	00000	1	0000000001000	0XA0102008

FRTO2

OP	RD	OP2	OPERATION	
00	10001	100	11111111111111100011	0X2300001D

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	10001	000010	10001	1	0000000110000	0XA2146030
10	10011	000010	00000	1	0000000100001	0XA6102021
10	10101	100101	10001	1	00000000	00100	0XAB2C6004
10	00000	010100	10100	0	00000000	10101	0X80A50015

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000010	0X36800002

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	10001	000100	10001	1	0000001001110	0XA224604E

FRTO 2
OP a COND OP2 DISP22

00	1	1000	010	00000000000000000010

0X30800002

FRT 3
OP RD OP3 RS1 I SIMM13/RS2

10	10110	100101	10001	1	00000000	00001

SUMA 0XAB2C6001

10	10011	000000	10000	0	00000000	10110

0XA6040016

10	01000	000000	10000	0	00000000	10011

EXIT 0X90040013

COMPLEMENTO A DOS PARA 14800
11100111010000

00011000101111
1

00011|000110000

11100
1

11101 = 29

B)

```
int main(){
   int a = 8;
   int b = -10;
   int c = 64;
   if(a!=b){
        return c/16;
   }
   else{
        return b*32;
   }
}
```

LENGUAJE ENSAMBLADOR:

```
       0000 MOV 8,%L0
       0004 MOV -10,%L1
       0008 MOV 64,%L2
       000C SUBcc %L0,%L1,G0
       0010 BNE a DIV
       0014 SLL %L1,5,O0
       0018 BA a EXIT
DIV    001C SRL %L2,4,O0
EXIT   0020 NOP
```

LENGUAJE DE MAQUINA:

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	10000	000010	00000	1	0000000001000	0XA0102008
10	10001	000010	00000	1	1111111110110	0XA2103FF6
10	10010	000010	00000	1	0000001000000	0XA4102040
10	00000	010100	10000	0	00000000	10001 0X80A40011

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1001	010	00000000000000000010	0X32800002

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	01000	100101	10000	1	0000000000101	0X912C2005

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1000	010	00000000000000000001	0X30800001

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	01000	100110	10010	1	0000000000100	DIV 0X9134A004

FRTO2

OP	RD	OP2	OPERATION	
00	00000	100	00000000000000000000	EXIT 0X01000000

COMPLEMENTO A 2 PARA -10

01010

10101
+ 1
10110

C)

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

LENGUAJE ENSAMBLADOR

```
0000 STEHI -21,%L0
0004 OR %L0,324,%L0
```

LENGUAJE DE MAQUINA

00	10000	100	11111111111111111011

0X213FFFEB

10	10000	000010	10000	1	0000101000100

0XA0142144

COMPLEMENTO A DOS PARA -21180

101001010111100

010110101000011
1

01011|0101000100

10100
1

10101 = -21

26)

```
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;
}
```

```
LENGUAJE ENSAMBLADOR


TEST    0000 SUB %I0,%I1,%I0
        0004 SLL %I3,2,%I3
        0008 ADD %I3,%I0,%O0
        000C JMPL %07+8,%G0
        0010 ADD %O0,2,%O0

MAIN    0014 MOV 4,%I0
        0018 MOV 2,%I1
        001C SETHI -31,%I3
        0020 OR %I3,272,%I3
        0024 CALL TEST
        0028 MOV 0,%L0
        002C ADD %L0,45,%01

LENGUAJE MAQUINA
```

TEST:

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	11000	000100	11000	0	00000000	11001	0XB0260019
10	11011	100101	11011	1	00000000	00010	0XB72EE002
10	01000	000000	11011	0	00000000	11000	0X9006C018
10	00000	111000	01111	1	0000000001000		0X81C3E008
10	01000	000000	01000	1	0000000000010		0X90022002

MAIN:

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	11000	000010	00000	1	0000000000100	0XB0102004
10	11001	000010	00000	1	0000000000010	0XB2102002

FRTO 2

OP	RD	OP2	OPERATION	
00	11011	100	11111111111111100001	0X373FFFE1

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	11011	000010	11011	1	0000100010000	0XB616E110

FRTO 1
OP DISP 30

01	11111111111111111111111110111	0X7FFFFFF7

FRTO3
OP RD OP3 RS1 I SIMM13/RS2

10	10000	000010	10000	1	0000000000000	0XA0142000

10	01000	000000	10000	1	0000000101101	0X9004202D

COMPLEMENTO A DOS PARA LOS DISP DEL CALL -9

1001

0110
1

0111

COMPLEMETNO A 2 PARA -15600

11110011110000

00001100001111
1

00001|100010000

11110
1

11111 =-31

27)

```
int mul(int c, int d){
        int z=0;
        for(int cont=1;cont<=d;cont+=1){
                z=z+c;
        }

        return z;

}

int pot(int a,int b){
        int c = a;
        if(b==0){
        return a=1;
        }
        else{
        for(int i=1;i<b;i++){
                a=mul(a,c);
                }
        return a;
}
}
Int main(){
        Int a=4, b=8;
        Int x= pot(a,b);
        Return x;
        }
```

LENGUAJE ENSAMBLADOR:

```
MUL     0000 MOV 0,%O0
        0004 MOV 1,%L0
FOR     0008 SUBCC %I0,%L0,G0 == CMP %I0,%L0
        000C BGE a SALIR
        0010 ADD %O0,%L2,%O0
        0014 BA a FOR
SALIR   0018 JMPL %O7+8,%G0
POT     001C MOV %I1,%L1
        0020 MOV %L2,1
        0024 SUBCC %I2,%L3,G0
        0028 BNE a ELSE
        002C MOV 1,%I1
ELSE    0030 SUBCC %L2,I2,GO
        0034 BGE SALIRE
        0038 CALL MUL
        003C NOP
        0040 BA a ELSE
SALIRE  0044 JMPL %O6+8,G0
MAIN    0048 MOV 4,%I1
        004C MOV 8,%I2
        0050 CALL POT
        0054 MOV 0,O1
```

LENGUAJE MAQUINA

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	01000	000010	00000	1	0000000000000		0X90102000
10	10000	000010	00000	1	0000000000001		0XA0102001
10	00000	010100	11000	0	00000000	10000	0X80A60010

FRTO2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000010	0X36800002

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	01000	000000	01000	0	00000000	10010	0X90020012

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1000	010	11111111111111111101	0X30BFFFFD

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	00000	111000	01111	1	0000000001000		0X81C3E008--
10	10001	000010	11001	0	00000000	00000	0XA2164000
10	10010	000010	10010	1	0000000000000		0XA414C000
10	00000	010100	11010	0	00000000	10011	0X80A68013

FRTO2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000001	0X36800001

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	11001	000010	00000	1	0000000000001		0XB2102001
10	00000	010100	10010	0	00000000	11010	0X80A4801A

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000001	0X36800001

```
FRTO 1
OP   DISP 30
```

01	11111111111111111111111110010	0X7FFFFFF2

FRTO2

```
OP   RD     OP2     OPERATION
```

00	00000	100	00000000000000000000	0X01000000

FRTO2

```
OP   a  COND  OP2  DISP22
```

00	1	1000	010	11111111111111111100	0X30BFFFFC

FRTO 3

```
OP    RD     OP3     RS1    I    SIMM13/RS2
```

10	00000	111000	01110	1	0000000001000	0X81C3A008

10	11001	000010	00000	1	0000000000100	0XB2102004

10	11010	000010	00000	1	0000000001000	0XB4102008

```
FRTO 1
OP   DISP 30
```

01	11111111111111111111111110101	X07FFFFFF6

FRTO 3

```
OP    RD     OP3     RS1    I    SIMM13/RS2
```

10	01001	000010	00000	1	0000000000000	0X92102000

28)

```
int mul(int c, int d){
        int z=0;
        for(int cont=1;cont<=d;cont+=1){
                z=z+c;
        }

        return z;

}
```

```
int fact (int a){
  int b,fac=1;
  for (b=1 ; b<=a ; b++)
    {
         fac=mul(b,fac);
    }
    return fac;

}
```

```
Int main(){
        Int a=7
        Int x= fact(a);
        Return x;
        }
```

LENGUAJE ENSAMBLADOR:

```
MUL     0000 MOV 0,%O0
        0004 MOV 1,%L0
FOR     0008 SUBcc %I0,%L0,G0 == CMP %I0,%L0
        000C BGE a SALIR
        0010 ADD %O0,%L2,%O0
        0014 BA a FOR
SALIR   0018 JMPL %O7+8,%G0
FACT    001C MOV %L1,1
        0020 MOV %L2,1
FOR1    0024 SUBcc %L1,%I1,G0’
        0028 BGE a SALIR1
        002C CALL MUL
        0030 NOP
        0034 BA a FOR1
SALIR1  0038 JMPL %O6+8,%G0
MAIN    003C MOV 7,%I1
        0040 CALL FACT
        0044 MOV 0,%01
```

LENGUAJE MAQUINA

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	01000	000010	00000	1	0000000000000	0X90102000
10	10000	000010	00000	1	0000000000001	0XA0102001
10	00000	010100	11000	0	00000000	10000 0X80A60010

FRTO2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000010	0X36800002

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2		
10	01000	000000	01000	0	00000000	10010	0X90020012

FRTO 2

OP	a	COND	OP2	DISP22	
00	1	1000	010	11111111111111111101	0X30BFFFFD

FRTO 3

OP	RD	OP3	RS1	I	SIMM13/RS2	
10	00000	111000	01111	1	0000000001000	0X81C3E008
10	10001	000010	00000	1	0000000000001	0XC2102001
10	10010	000010	00000	1	0000000000001	0XC4102001
10	00000	010100	10001	0	00000000	11001 0X80A44019

FRTO2

OP	a	COND	OP2	DISP22	
00	1	1011	010	00000000000000000011	0X36800003

FRTO 1

OP	DISP 30	
01	11111111111111111111111110101	0X7FFFFFF6

FRTO2

OP	RD	OP2	OPERATION	
00	00000	100	00000000000000000000	0X01000000

FRTO2

OP a COND OP2 DISP22

00	1	1000	010	11111111111111111001

0X30BFFFF9

FRTO 3
OP RD OP3 RS1 I SIMM13/RS2

10	00000	111000	01110	1	0000000001000

0X81C3A008

10	11001	000010	00000	1	0000000000111

0XB2102007

FRTO 1
OP DISP 30

01	11111111111111111111111110111

0X7FFFFFF7

FRTO 3
OP RD OP3 RS1 I SIMM13/RS2

10	01001	000010	00000	1	0000000000000

0X92102000