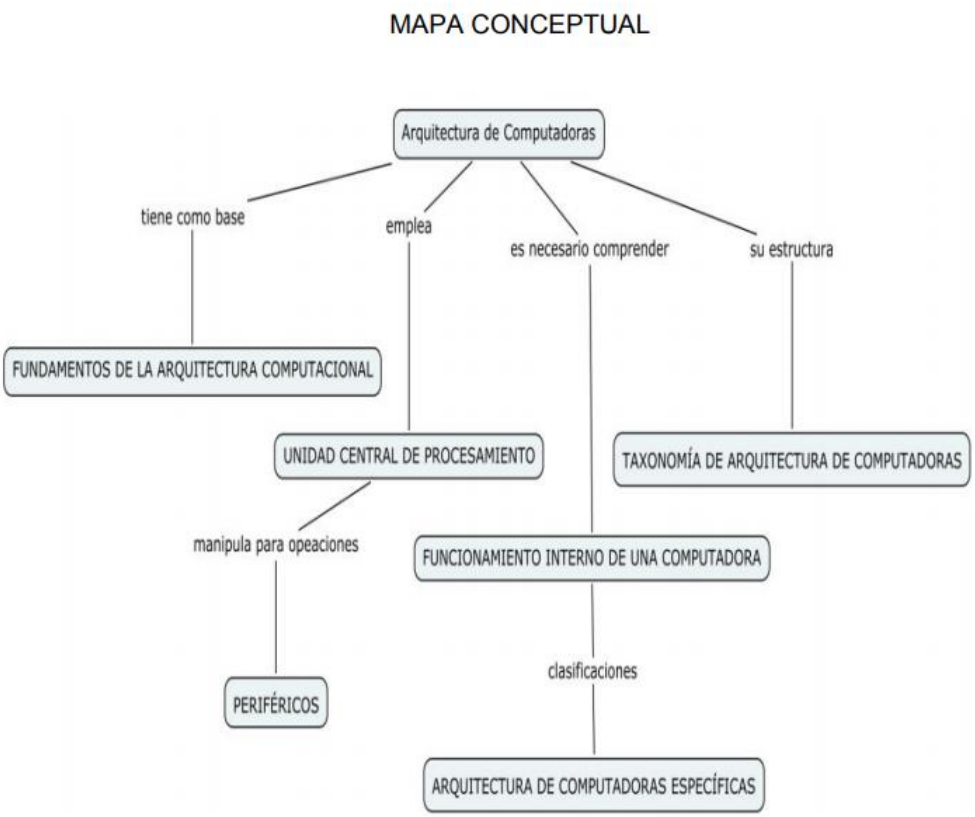
# TALLER (PRIMER PARCIAL)

## ARQUITECTURA DE COMPUTADORES

## 2018

## Jornada Especial.

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




La arquitectura de computadoras es el diseño conceptual y la estructura operacional fundamental de un sistema que conforma una computadora. Es decir, es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso (CPU) trabaja internamente y accede a las direcciones de memoria.

La arquitectura de una computadora explica la situación de sus componentes y permite determinar las posibilidades de un sistema informático, con una determinada configuración, pueda realizar las operaciones para las que se va a utilizar. La arquitectura básica de cualquier ordenador completo está formada por solo 5 componentes básicos: procesador, memoria RAM, disco duro, dispositivos de entrada/salida y software

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?**

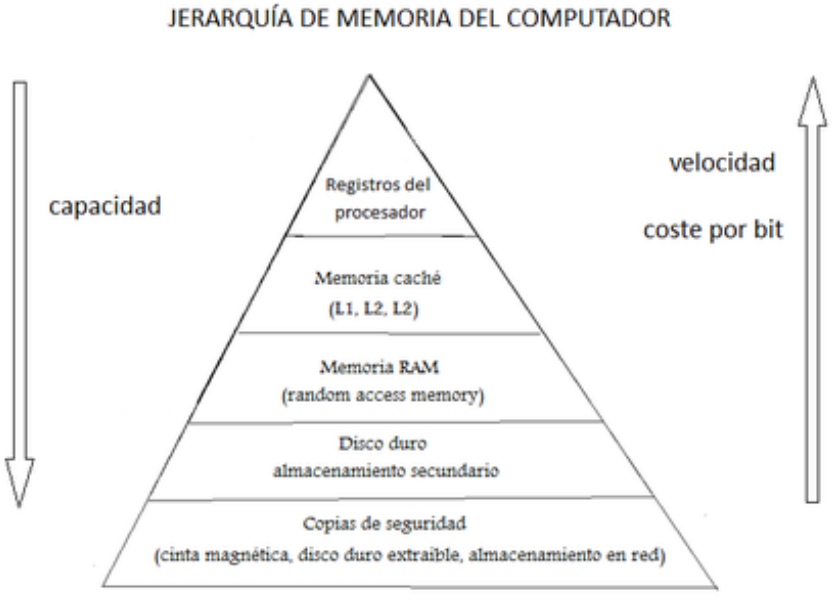
La taxonomía de Flynn se clasifica en 4 tipos:

- **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.**

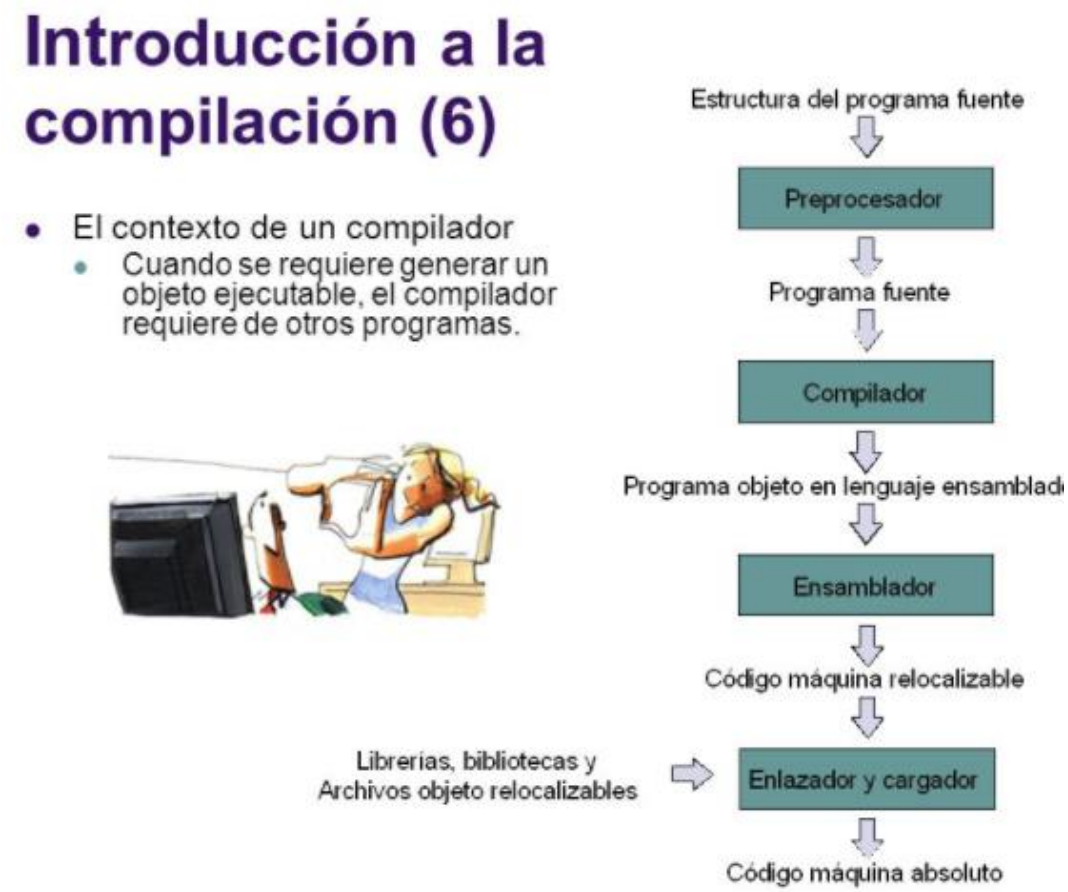
- Equipos de escritorio (PC's)
- Equipos servidores. (Utilizados para almacenar bases de datos y a los cuales acceden muchos usuarios)
- Tablets - dispositivos móviles
- Wearables
- Supercomputadores - HPC (High Performance Computing) - GPU : CPU (Sistemas heterogeneos)

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



## 6. ¿Qué es un compilador?

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio (bytecode), o simplemente texto. Este proceso de traducción se conoce como compilación.
Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje de máquina). De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a cómo piensa un ser humano, para luego compilarlo a un programa más manejable por una computadora.




## 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?

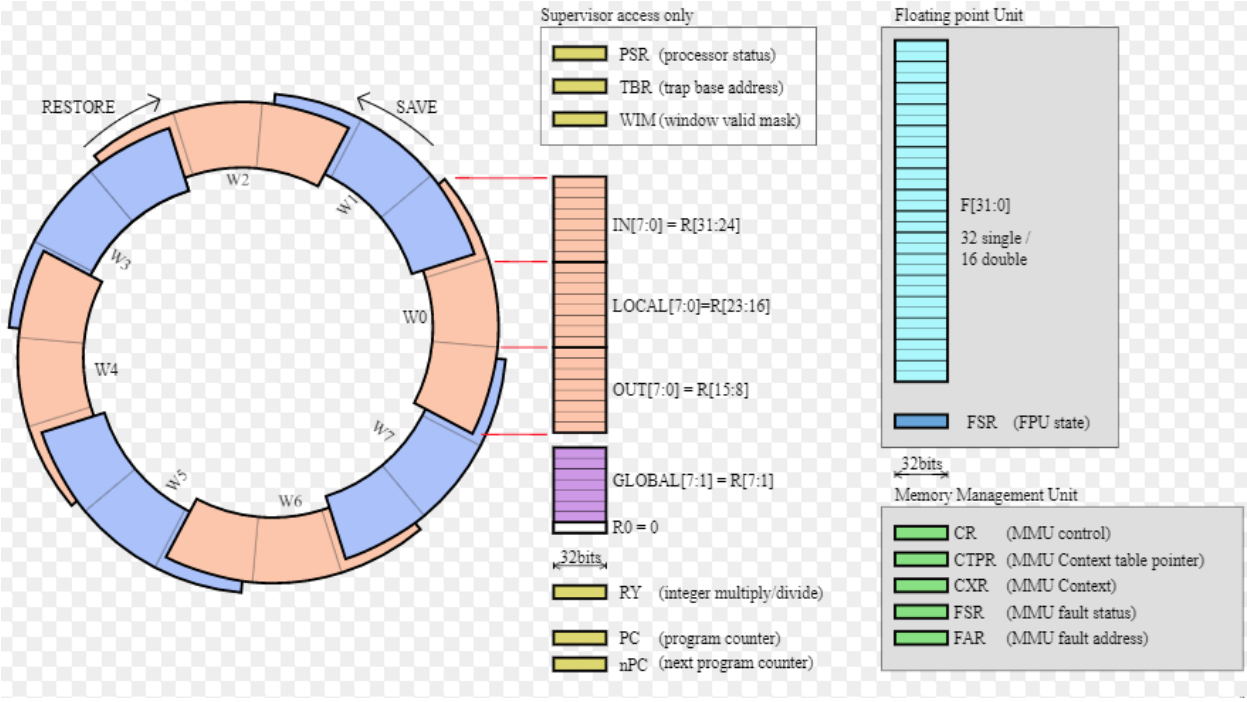
- La simplicidad favorece la regularidad:
  Hacer diseños simples para garantizar que el procesador funcione bien siempre

- 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

## 9. ¿Qué es SPARCV8?

Es una arquitectura de computadores de 32 principalmente diseñada para optimizar compiladores




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

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

## 11. ¿Qué 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
Load [%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
00000000000000000000000000000101
11111111111111111111111111111001

**b.**12890
00000000000000000011001001011010
11111111111111111100110110100110

**c.**56900
00000000000000000110111100100010
11111111111111111001000011011110

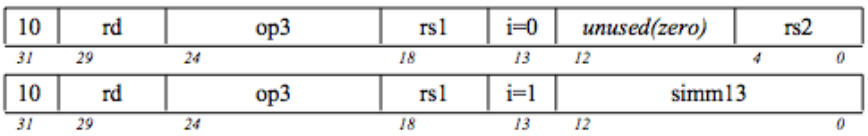
**d.**11
00000000000000000000000000001011
11111111111111111111111111110101

**e.**140
00000000000000000000000010001100
11111111111111111111111101110100

**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* , $reg_{rd}$ |

**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 una 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 instrucción **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 opernando de una instruccion aritmético-lógica, Load/Store, o de corrimiento.

**rs2** : Segundo operando de una instruccion si i = 1.

**imm13** : Segundo operando de una instruccion 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

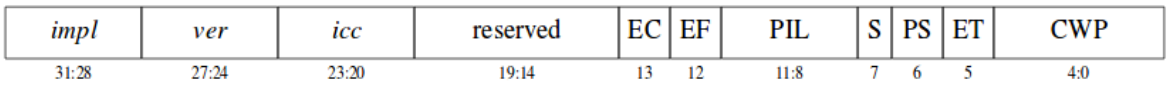
    El op2 es una subclasificación de instrucciones y solo se usa cuando el op es 2 (formato2)

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

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



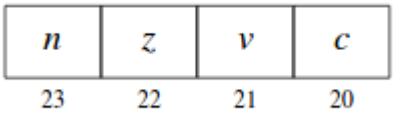

- **impl**: Implementación
- **version**: Vérsión
- **ICC** : Esta dividido en 4 bits: **n**: Negativo cuando una operación da un número negativo, **Z**: se coloca en 1, cuando el resultado de la operación da cero, **v**: Over Flow se activa cuando el resultado de una operación da más de 32 bits y el 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,Targeta 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 iterrupciones llamadas Traps)
- **S**: Supervisor; Indica el modo en el cual se ejecutó el procesador durante la instruccion 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 8, %L0 ---→ esta instrucción ejecuta por debajo un OR entre 8 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

| OR %G0, 5, %L0 | | | | | |
|---|---|---|---|---|---|
| OP | RD | OP3 | RS1 | I | SIMM 13 |
| 10 | 10000 | 000010 | 00000 | 1 | 0000000000101 |

10100010000100000011111111111010

| OR %G0, -6, %L1 | | | | | |
|---|---|---|---|---|---|
| OP | RD | OP3 | RS1 | I | SIMM 13 |
| 10 | 10001 | 000010 | 00000 | 1 | 0000000000010 |

10010000000001000100000000010000

| ADD %L1, %L0, %O0 | | | | | | |
|---|---|---|---|---|---|---|
| OP | RD | OP3 | RS1 | I | UNUSED (ZERO) | RS2 |
| 10 | 01000 | 000000 | 10001 | 0 | 00000000 | 10000 |

```
int main(){

  int x =   5;
  int y =  -6;
  return x + y;                    }

MOV 5,  %L0
MOV -6, %L1
ADD %L0, %L1, %O0
```

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

```
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
}
%L0 →I
%L1 →B
%L2 →C
%L3 →D


0000  MOV  5, %L0
0004  MOV -4, %L1
0008  ADD %L0,2, %L0
```

000C  ST %L0, [%L2 +(5*4)]
0010  ADD %L1, 3, %L1
0014  ST %L1, [%L3 + (4*4)]
0018  LOAD [%L2 +(5*4)], %L4
001C  LOAD [%L3 + (4*4)], %L5
0020D %L4, %L5, %L4
0024 B %L4, %L0, %O0

**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

**0008  ADD %L0,2, %L0**

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

0XA0042002

**000C  ST %L0, [%L2 +(5*4)]**

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

0XE4242014

**D %L1, 3, %L1**

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

0XA2043006

**0014  ST %L1, [%L3 + (4*4)]**

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

0XE6246010

**0018  LOAD [%L2 +(5*4)], %L4**

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

OX E804A014

**001C  LOAD [%L3 + (4*4)], %L5**

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

OX EA04E010

**0020  ADD %L4, %L5, %L4**

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

OX A8050015

**0024 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;
int b = -14800;
int c = 33;
if((a+b)<=b*16){
c=a+(b*2);
}
else{
return b-78;
}
return a+c;
}
```

%L0 →A
%L1 →B
%L2 →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
```

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 | 1111111111111111110001 |

OX 233FFFF1

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 | 0000000000000000101100 |

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 | 00000 | 100 | 0000000000000000000000 | | | |
| OX 01000000 | | | | | | |

**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

0000 OR 8, %G0, %L0

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

0004 OR -10, %G0 %L1

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

0008 SUBICC %L0, %L1, &G0

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

000C BE, A, ELSE

| OP | A | COND | | DISP 22 |
|---|---|---|---|---|
| 00 | 1 | 0001 | 010 | 00000000000000000011000 |

0010 SRL, %L2,4 %O0

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

0014 BA, A, FINPROGRAMA

| OP | A | COND | | DISP 22 |
|---|---|---|---|---|
| 00 | 1 | 1000 | 010 | 00000000000000000011100 |

0018 SLL %L1, 5, %O0

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

001C NOP

| OP | RD | OP2 | IMM 22 |
|---|---|---|---|
| 00 | 00000 | 100 | 0000000000000000000000 |
| OX 01000000 | | | |

**c.**

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

%L0 →A

0000 SETHI  2097131, %L0
0004 OR %L0, 324, %L0

0000 SETHI  2097131, %L0

| OP | RD | OP2 | IMM 22 |
|---|---|---|---|
| 00 | 10000 | 100 | 01111111111111111101011 |

0004 OR %L0, 324, %L0

| OP | RD | OP3 | RS1 | I | SIMM 13 |
|---|---|---|---|---|---|
| 10 | 10000 | 000010 | 10000 | 1 | 0000101000100 |
| OX A0142144 | | | | | |

**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

0004 SLL %I2, 2, %I2

| OP | RD | OP3 | RS1 | I | SIMM 13 |
|---|---|---|---|---|---|
| 10 | 11010 | 100101 | 11010 | 1 | 00000000000010 |

OX B52EA002

0008 ADD %I3, %I2, %L0

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

OX A006C01A

000C ADD %L0, 2, %O0

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

OX 90042002

0010 JMPL %07, 8, %G0

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

OX 81C3E008

0014 NOP

| OP | RD | OP2 | IMM 22 |
|---|---|---|---|
| 00 | 00000 | 100 | 0000000000000000000000 |

OX 01000000

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 | 1111111111111111110000 |

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 | 00000 | 100 | 0000000000000000000000 |

OX 01000000

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 multiplicación 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
    }
```

.

%L0 →A
%L1 →B
%L2 →C

```
0000    MOV 0, %L4
0004    SUB %LO,1,%L5
0008    CMP %L4, %L5
000C    BGE, A, ENDFOR1
0010    ADD %O0, %L0, %O0
0014    ADD %L4, 1, %L4
0018    BA, A, FOR1
001C    JMPL %07, 8, %G0
0020    MOV  %07, %06
0024    MOV %LO, %O0
0028    MOV 0, %L2
002C    SUB %L1, 1, %L3
0030    CMP %L2, %L3
0034    BGE, A, ENDFOR2
0038    CALL MULTIPLICACION
003C    NOP
0040    ADD %L2, 1, %L2
0044    BA, A, FOR2
0048    JMPL %06, 4, %G0
004C    MOV 2, %L0
0050    MOV 5, %L1
0054    CALL POTENCIA
0058    NOP
```

| MULTIPLICACION | | | | | | |
|---|---|---|---|---|---|---|
| 0000 | MOV 0, %L4 | | | | | |
| OP | RD | OP3 | RS1 | I | SIMM 13 | |
| 10 | 10100 | 000010 | 00000 | 1 | 0000000000000 | |
| OX A810200 | | | | | | |
| | | | | | | |
| 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 | 00000000000000001000 | | |
| 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 | 0000000000000001001000 | | |
| OX 36800048 | | | | | | |
| | | | | | | |
| 0038 | CALL MULTIPLICACION | | | | | |
| OP | disp30 | | | | | |
| 01 | 00000000000000000000000000000 | | | | | |
| OX 40000000 | | | | | | |
| | | | | | | |
| 003C | NOP | | | | | |
| OP | RD | OP2 | IMM 22 | | | |
| 00 | 00000 | 100 | 0000000000000000000000 | | | |
| OX 01000000 | | | | | | |
| | | | | | | |
| 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 | 0000000000000000110000 | | |
| 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 | 00000 | 100 | 0000000000000000000000 | | | |
| OX 01000000 | | | | | | |

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.**

```
int factorial (int num) {
int i=0;
    int factorial=1;
    for(i=2;i<=num;i++){
      factorial=factorial*i;
    }
    Printf ( "El resultado es: %d", factorial);
    }

    int main () {

       int numero=5;
      factorial(numero);
    }
```

%L0 →I
%O0 →FACTORIAL
%I0 →NUMERO

```
0000 MOV 0, %L0
0004 MOV 1, %O0
IAMFOR
0008 CMP %L0, %I0
000C BGE A ENDFOR
0010 SLL %O0, %L0, %O0
0014 BA  IAMFOR
0018 ADD %L0, 1, %L0
ENDFOR
001C JMPL %07, 8, %L0
0020 NOP
MAIN
0024 MOV 5, %I0
0028 CALL fact
```

0000 MOV 0, %L0

| OP | RD | OP3 | RS1 | I | SIMM 13 | |
|---|---|---|---|---|---|---|
| 10 | 10000 | 000010 | 00000 | 1 | 0000000000000 | |

OX A0102000

0004 MOV 1, %O0

| OP | RD | OP3 | RS1 | I | SIMM 13 | |
|---|---|---|---|---|---|---|
| 10 | 01000 | 000010 | 00000 | 1 | 0000000000000 | |

OX 90102000

0008 CMP %L0, %I0

| OP | RD | OP3 | RS1 | I | UNUSED (ZERO) | RS2 |
|---|---|---|---|---|---|---|
| 10 | 01000 | 010100 | 10000 | 0 | 00000000 | 11000 |

OX 80A40018

000C BGE A ENDFOR

| OP | A | COND | | DISP 22 | | |
|---|---|---|---|---|---|---|
| 00 | 1 | 1011 | 010 | 00000000000000000100 | | |

OX 36800004

0010 SLL %O0, %L0, %O0

| OP | RD | OP3 | RS1 | I | UNUSED (ZERO) | RS2 |
|---|---|---|---|---|---|---|
| 10 | 01000 | 100101 | 01000 | 0 | 00000000 | 10000 |

OX 912A0010

0014 BA IAMFOR

| OP | A | COND | | DISP 22 | | |
|---|---|---|---|---|---|---|
| 00 | 1 | 1000 | 010 | 11111111111111111100 | | |

OX 30BFFFFC

0018 ADD %L0, 1, %L0

| OP | RD | OP3 | RS1 | I | UNUSED (ZERO) | RS2 |
|---|---|---|---|---|---|---|
| 10 | 10000 | 000000 | 10000 | 0 | 00000000 | 00001 |

OX A0040001

001C JMPL %07, 8, %L0

| OP | RD | OP3 | RS1 | I | SIMM 13 | |
|---|---|---|---|---|---|---|
| 10 | 00000 | 111000 | 01111 | 1 | 0000000001000 | |

OX 81C3E008

0020 NOP

| OP | RD | OP2 | IMM 22 | | | |
|---|---|---|---|---|---|---|
| 00 | 00000 | 100 | 00000000000000000000 | | | |

OX 01000000

0024 MOV 5, %I0

| OP | RD | OP3 | RS1 | I | SIMM 13 | |
|---|---|---|---|---|---|---|
| 10 | 11000 | 000010 | 00000 | 1 | 0000000000101 | |

OX B0102005

0028 CALL fact

| OP | disp30 | | | | | |
|---|---|---|---|---|---|---|
| 01 | 11111111111111111111111110110 | | | | | |

OX 7FFFFFF6