**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 formado por solo 5 componentes básicos: procesador, memoria RAM, disco duro, dispositivos de entrada/salida y software.

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

* **Primera Generación** (1951-1958)



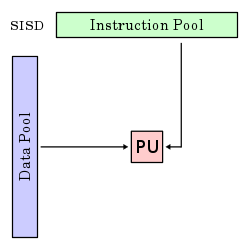
* Usaban tubos al vacío para procesar información.
* Usaban tarjetas perforadas para entrar los datos y los programas.
* Usaban cilindros magnéticos para almacenar información e instrucciones internas.
* Eran sumamente grandes, utilizaban gran cantidad de electricidad, generaban gran cantidad de calor y eran sumamente lentas.
* Se comenzó a utilizar el sistema binario para representar los datos.
* **Segunda Generación** (1958-1964)
* Algunas computadoras se programaban con cintas perforadas y otras por medio de cableado en un tablero.
* Usaban transistores para procesar información.
* Los transistores eran más rápidos, pequeños y más confiables que los tubos al vacío.
* Usaban pequeños anillos magnéticos para almacenar información e instrucciones. cantidad de calor y eran sumamente lentas.
* Se mejoraron los programas de computadoras que fueron desarrollados durante la primera generación.
* Se desarrollaron nuevos lenguajes de programación como COBOL y FORTRAN, los cuales eran comercialmente accsesibles.
* **Tercera Generación** (1964-1971)
* Se desarrollaron circuitos integrados para procesar información.
* Se desarrollaron los "chips" para almacenar y procesar la información. Un "chip" es una pieza de silicio que contiene los componentes electrónicos en miniatura llamados semiconductores.
* Las computadoras pueden llevar a cabo ambas tareas de procesamiento o análisis matemáticos.
* Emerge la industria del "software".
* **Cuarta Generación** (1971-1988)
* Se desarrolló el microprocesador.
* Se colocan más circuitos dentro de un "chip".
* "LSI - Large Scale Integration circuit".
* "VLSI - Very Large Scale Integration circuit".
* Se desarrollan las microcomputadoras, o sea, computadoras personales o PC.
* Se desarrollan las supercomputadoras.
* **Quinta Generación** (1983 al presente)
* En vista de la acelerada marcha de la microelectrónica, la sociedad industrial se ha dado a la tarea de poner también a esa altura el desarrollo del software y los sistemas con que se manejan las computadoras. Surge la competencia internacional por el dominio del mercado de la computación, en la que se perfilan dos líderes que, sin embargo, no han podido alcanzar el nivel que se desea: la capacidad de comunicarse con la computadora en un lenguaje más cotidiano y no a través de códigos o lenguajes de control especializados.
* Japón lanzó en 1983 el llamado "programa de la quinta generación de computadoras", con los objetivos explícitos de producir máquinas con innovaciones reales en los criterios mencionados. Y en los Estados Unidos ya está en actividad un programa en desarrollo que persigue objetivos semejantes, que pueden resumirse de la siguiente manera:
* Se desarrollan las microcomputadoras, o sea, computadoras personales o PC.
* Se desarrollan las supercomputadoras.

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

La taxonomía de Flynn es una clasificación de arquitecturas de computadores propuesta por Michael J. Flynn en 1972

|  |  |  |
| --- | --- | --- |
| **Taxonomía de Flynn** | | |
|  | **Una instrucción** | **Múltiples instrucciones** |
| **Un dato** | [SISD](https://es.wikipedia.org/wiki/SISD) | [MISD](https://es.wikipedia.org/wiki/MISD) |
| **Múltiples datos** | [SIMD](https://es.wikipedia.org/wiki/SIMD) | [MIMD](https://es.wikipedia.org/wiki/MIMD) |

* **Single Instruction, Single Data**: Una Secuencia de instrucciones y una secuencia de datos(SISD) es un único procesador que interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria, no explota el paralelismo en las instrucciones ni en flujos de datos. Ejemplos de arquitecturas SISD son las máquinas con uni-procesador tradicionales como el PC o los antiguos mainframe.

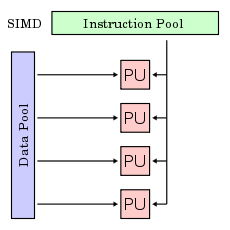


**Características:**

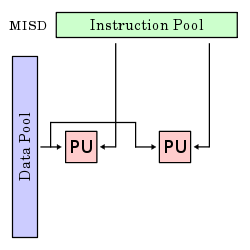
* La CPU procesa únicamente una instrucción por cada ciclo de reloj
* Únicamente un dato es procesado en cada ciclo de reloj
* Es el modelo más antiguo de computadora y el más extendido
* **Single Instruction, Multiple Data:** Una secuencia de instrucciones y múltiples secuencias de datos (SIMD): Una solo instrucción controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, Un computador que explota varios flujos de datos dentro de un único flujo de instrucciones para realizar operaciones que pueden ser paralelizadas de manera natural. Por ejemplo, un procesador vectorial. Es un tipo de arquitectura computacional (particularmente de computación paralela) donde muchas unidades funcionales realizan diferentes operaciones en los mismos datos.

**Características:**

* Todas las unidades ejecutan la misma instrucción
* Cada unidad procesa un dato distinto
* Todas las unidades operan simultáneamente

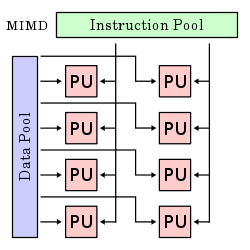


* **Multiple Instruction, Single Data:** Múltiples secuencias de instrucción y una secuencia de datos (MISD): Se transmite una secuencia de datos a un conjunto de procesadores, y cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Poco común debido al hecho de que la efectividad de los múltiples flujos de instrucciones suele precisar de múltiples flujos de datos. Sin embargo, este tipo se usa en situaciones de paralelismo, como por ejemplo en navegación aérea, donde se necesitan varios sistemas de respaldo en caso de que uno falle. También se han propuesto algunas arquitecturas teóricas que hacen uso de MISD, pero ninguna llegó a producirse en masa



**Características:**

* Cada unidad ejecuta una instrucción distinta
* Cada unidad procesa el mismo dato
* Aplicación muy limitada en la vida real
* **Multiple Instruction, Multiple Data:** Consiste un en conjunto de procesadores que ejecutan simultáneamente secuencias de instrucción diferentes con conjuntos de datos diferentes. Los SMP, los clusters y los sistemas NUMA son ejemplos de esta categoría.



**Características:**

* Cada unidad ejecuta una instrucción distinta
* Cada unidad procesa un dato distinto
* Todas las unidades operan simultáneamente

SISD (SINGLE INSTRUCTION SINGLE DATA): Dispositivo que utiliza la arquitectura son los microprocesadores

SIMD (SINGLE INSTRUCTION MULTIPLE DATA): Dispositivo que utiliza la arquitectura es la GPU

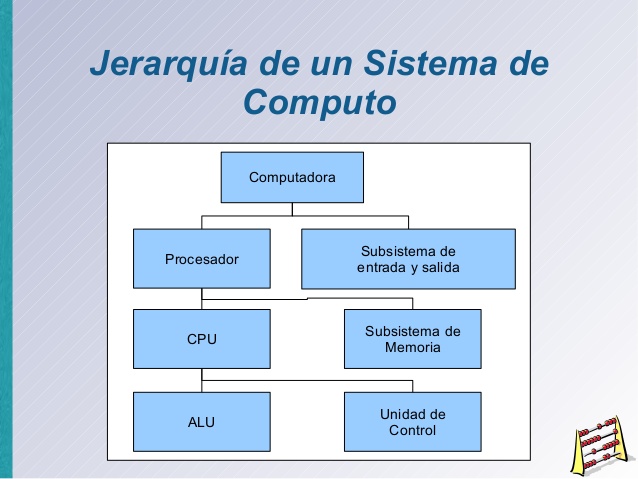
MIMD (MULTIPLE INSTRUCTION MULTIPLE DATA): Dispositivo que utiliza la arquitectura es los sistemas aéreos

MISD (MULTIPLE INSTRUCTION SINGLE DATA): Dispositivo que utiliza la arquitectura es los sistemas distribuidos

1. 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)

1. Muestre la clasificación de la jerarquía de un equipo de cómputo.



1. ¿Qué es un compilador?

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.

1. ¿Defina qué es una instrucción?

"Software de intrusión", que define como Software especialmente diseñado o modificado para evitar la detección por "herramientas de monitoreo", o para derrotar "contramedidas de protección", de una computadora o dispositivo con capacidad de red, y realizar cualquiera de lo siguiente: a. La extracción de datos o información, desde una computadora o dispositivo con capacidad de red, o la modificación de datos del sistema o del usuario; o b. La modificación de la ruta de ejecución estándar de un programa o proceso para permitir la ejecución de instrucciones provistas externamente. Esta nueva clase de software así definida es muy amplia y fundamental. Como demostraremos, no solo cubre software conocido en jerga de seguridad informática como exploits y rootkits, sino que de hecho todos los medios elementales de instrumentación de software, construcción y deconstrucción fuera de las interfaces predefinidas del software para estos fines, a pesar de que se exceptúan explícitamente los "hipervisores". , depuradores o herramientas de ingeniería inversa de software (SRE) "en la nota a la definición anterior. Sin embargo, WA no controla directamente esta nueva clase.

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

1-La simplicidad favorece la regularidad

2-Entre más pequeño es más rápido

3-Hacer el caso común más rápido

4-Buenos diseños demandan grandes compromisos

1. ¿Qué es SPARCV8?

SPARC (Scalable Processor Architecture) es una arquitectura de microprocesador de 32 y 64 bits de Sun Microsystems que se basa en la computación reducida de conjuntos de instrucciones (RISC). SPARC se ha convertido en una arquitectura ampliamente utilizada para hardware utilizado con sistemas operativos basados ​​en UNIX, incluidos los propios sistemas Solaris de Sun. Sun ha hecho de SPARC una arquitectura abierta que está disponible para licencias a los fabricantes de microprocesadores. En su marca más reciente, UltraSPARC, los microprocesadores se pueden construir para placas de PC (ya sea mediante Interconexión de Componentes Periféricos o ATX) así como también para el mercado de estación de trabajo original de SPARC. Como evidencia de la escalabilidad de SPARC, Sun dice que su UltraSPARC III se diseñará para permitir que hasta 1.000 procesadores trabajen juntos.

SPARC es una arquitectura de conjunto de instrucciones de CPU (ISA), derivada de una reducción linaje de conjunto de instrucciones (RISC). Como arquitectura, SPARC permite una espectro de implementaciones de chip y sistema a una variedad de precio / rendimiento puntos para una variedad de aplicaciones, incluida la programación científica / de ingeniería, en tiempo real y comercial.

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

Las instrucciones se dividen en seis categorías básicas:

* 1. Load/store
  2. Arithmetic/logical/shift
  3. Control transfer
  4. Read/write control register
  5. Floating-point opérate
  6. Coprocessor operate

1. ¿Qué tipos de registros se encuentran en SPARC V8?

Un procesador SPARC incluye dos tipos de registros: de propósito general o "de trabajo" registros de datos y registros de control / estado. Registros de uso general de la IU se llaman registros r, y los registros de propósito general de la FPU se llaman f, registros. Los registros de trabajo del coprocesador dependen de la implementación del coprocesador.

Los registros de control / estado de IU incluyen:

* Registro del estado del procesador (PSR)
* 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
* Los registros de control / estado de FPU incluyen:
* Registro Estatal de Punto Flotante (FSR)
* Cola de trampa diferida de punto flotante dependiente de la implementación (FQ)
* Los registros de control / estado del coprocesador (CP), si están presentes, pueden incluir:
* Registro estatal de coprocesador dependiente de la implementación (CSR)
* Coprocesador dependiente de la implementación Deferred-Trap Queue (CQ)

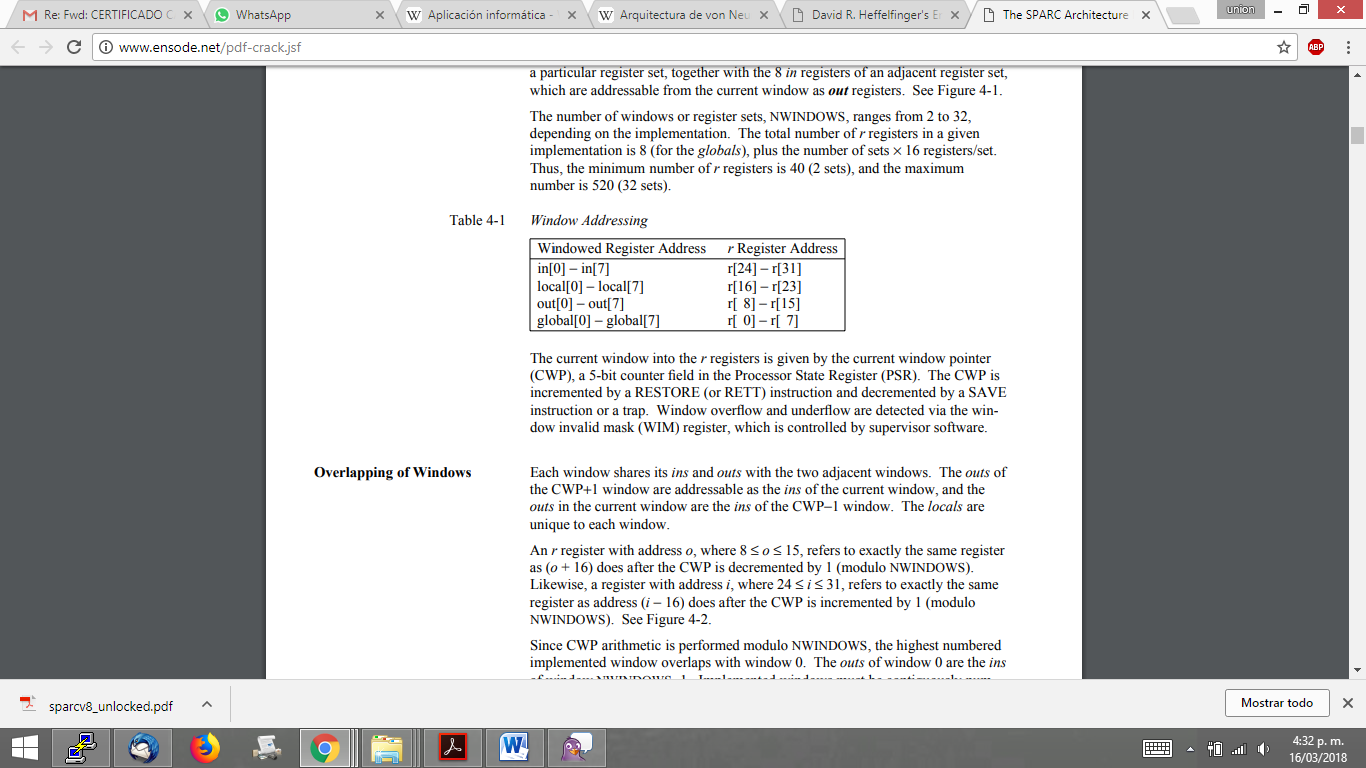
1. ¿Cuál es el número mínimo y máximo de registros que se puede implementar en la arquitectura SPARCV8?

Registros r de 32 bits. Están divididos en 8 registros globales, más un número de 16 registros dependientes de la implementación. Un conjunto de registros es más particionado en 8 en registros y 8 registros locales.

En un momento dado, una instrucción puede acceder a los 8 globales y una ventana de 24 registros en los registros de r Una ventana de registro comprende el 8 en y 8 registros locales de un conjunto de registro particular, junto con los registros 8 en un conjunto de registros adyacente, que son direccionales desde la ventana actual como registros de salida.

La cantidad de ventanas o conjuntos de registros, NWINDOWS, varía de 2 a 32, dependiendo de la implementación. El número total de registros r en un dado la implementación es 8 (para los globales), más el número de conjuntos × 16 registros / conjunto.

Por lo tanto, el número mínimo de registros r es 40 (2 conjuntos), y el máximo el número es 520 (32 juegos).



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

Las instrucciones de SPARC se pueden clasificar en cinco categorías:

**LOAD y STORE** (la única manera de acceder a la memoria). Estas instrucciones usan dos registros o un registro y una constante para calcular la dirección de memoria a direccionar.

* **LOAD:** Las instrucciones de memoria cargan un valor de la memoria o almacenan un valor en la memoria. La dirección de destino se calcula como la adición de un valor base con un valor de desplazamiento opcional.

**Por ejemplo:**

ld [%r4 + 10], %r5

Carga el registro% r5 con el valor en la memoria ubicado en [% r4 + 10]. Si% r4 contiene el número 25, entonces la ubicación de la memoria leída sería [25 + 10] = 35. El parámetro op2, al igual que las instrucciones aritméticas, puede ser un registro o un inmediato. Esto significa que los siguientes son ambos válidos:

ld [%r4 + %r3], %r5

ld [%r4 + 100], %r5

El valor de compensación es el número de bytes compensados, no el número de palabras de máquina o medias palabras. Esto significa que para cargar palabras consecutivas en una máquina de 32 bits, la compensación debe incrementarse en 4:

ld [%r4 + 4], %r5

ld [%r4 + 8], %r6

* STORE: Las instrucciones de STORE son similares a las instrucciones de LOAD, excepto que los operandos están invertidos. Esto significa que tenemos el siguiente formato de instrucción:

st rd, [rs1 + op2]

**Instrucciones Aritméticas/Lógicas/Shift.** Ejecutan operaciones aritméticas, lógicas y de desplazamiento de bits. Estas instrucciones calculan el resultado si es una función de 2 operandos y guardan el resultado en un registro.

Las instrucciones de aritmética / lógica / desplazamiento realizan aritmética, aritmética etiquetada, lógico, y las operaciones de cambio. Con una excepción, estas instrucciones calculan un resultado que es una función de dos operandos fuente; el resultado está escrito en un registro de destino, o descartado.

* **Ejemplo:**

La excepción es una instrucción especializada, SETHI, que (junto con una segunda instrucción) se puede usar para crear un archivo de 32 bits constante en un registro r

Las instrucciones de desplazamiento se pueden usar para cambiar el contenido de un registro r izquierdo o derecho por una distancia dada. La distancia de desplazamiento puede especificarse mediante una constante en la instrucción o por el contenido de un registro r

Las instrucciones de multiplicar entero realizan un signo 32 × 32 → 64-bit sin signo operación. Las instrucciones de división entera realizan un 64 signed firmado o sin firmar 32 → operación de 32 bits. Hay versiones de multiplicar y dividir que establecen el códigos de condición. La división por cero causa una trampa.

Las instrucciones aritméticas etiquetadas suponen que los 2 bits menos significativos d los operandos son "etiquetas" de tipo de datos. Estas instrucciones establecen el código de condición de desbordamiento bit sobre el desbordamiento aritmético, o si alguno de los bits de etiqueta del operando es distinto de cero.

También hay versiones que se atrapan cuando ocurre cualquiera de estas condiciones.

**Operaciones del Coprocesador.** La IU extrae las operaciones de coma flotante desde las instrucciones del bus de datos y los coloca en la cola para la FPU. La FPU ejecuta los cálculos de coma flotante con un número fijo en unidad aritmética de coma flotante (el número es dependiente de la aplicación). Las operaciones de coma flotante son ejecutadas concurrentemente con las instrucciones de la IU y con otras operaciones de coma flotante cuando es necesario. La arquitectura SPARC también especifica una interfaz para la conexión de un coprocesador adicional.

Las instrucciones de operación de coma flotante (FPop) realizan todos los cálculos de coma flotante.

Son instrucciones de registro para registrar que operan en el punto flotante registros. Al igual que las instrucciones aritméticas / lógicas / de desplazamiento, los FPop calculan un resultado que es una función de uno o dos operandos fuente. Operaciones específicas de coma flotante son seleccionados por un subcampo de los formatos de instrucción FPop1 / FPop2.

Las instrucciones de operación de coprocesador (CPop) están definidas por el coprocesador implementado.

Si alguna. Estas instrucciones están especificadas por las instrucciones CPop1 y CPop2 formatos.

**Instrucciones de Control de Transferencia.** Estas incluyen jumps, calls, traps y branches. El control de transferencia es retardado usualmente hasta después de la ejecución de la próxima instrucción, así el pipeline no es vaciado porque ocurre un control de tiempo. De este modo, los compiladores pueden ser optimizados por ramas retardadas.

Las instrucciones de control de transferencia (CTI) incluyen derivaciones y llamadas relativas a PC, saltos de registro indirectos y trampas condicionales. La mayor parte de la transferencia de control las instrucciones son instrucciones retardadas de transferencia de control (DCTI), donde la instrucción inmediatamente después de que se ejecuta el DCTI antes de la transferencia de control a la dirección de destino está completa.

La instrucción que sigue a una instrucción de transferencia de control retrasada se denomina demora instrucción. La instrucción de retraso siempre se obtiene, incluso si el control diferido la transferencia es una rama incondicional. Sin embargo, un poco en el control de transferencia retrasada la instrucción puede causar la anulación de la instrucción de demora (es decir, tener sin efecto) si la rama no se toma (o en la rama siempre caso, si la rama es tomado).

* **Ejemplo:**

Las instrucciones Branch y CALL usan desplazamientos relativos a PC. El salto y La instrucción de enlace (JMPL) usa una dirección de destino de registro indirecto. Calcula su dirección de destino como la suma de dos registros r, o la suma de un registro r y un valor inmediato firmado de 13 bits. La instrucción de bifurcación proporciona un desplazamient de ± 8 Mbytes, mientras que el desplazamiento de palabra de 30 bits de la instrucción CALL permite controlar la transferencia a una dirección de instrucción arbitraria de 32 bits.

**Instrucciones de control de registros Read/Write.** Estas instrucciones se incluyen para leer y grabar el contenido de varios registros de control. Generalmente la fuente o destino está implícito en la instrucción.

Las instrucciones de registro de lectura / escritura leen y escriben los contenidos de software visible registros de estado / estado. También hay un "registro estatal auxiliar" de lectura / escritura instrucciones que el software puede usar para leer / escribir dependientes de la implementación únicos registros del procesador Si cada una de estas instrucciones es privilegiada o no es depende de la implementación.

1. Represente los siguientes números en complemento a 2.
2. 5

Se pasa a binario y quedaría

0000000000101

1111111111010

1

\_\_\_\_\_\_\_\_\_\_\_\_\_

1111111111011

1. 12890

11001001011010

00110110100101

1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

00110110100110

1. 56900

1101111001000100

0010000110111011

1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0010000110111100

1. 11

0000000001011

1111111110100

1

\_\_\_\_\_\_\_\_\_\_\_\_\_

1111111110101

1. 140.

0000010001100

1111101110011

1

\_\_\_\_\_\_\_\_\_\_\_\_\_

1111101110100

1. Explique las instrucciones aritméticas lógicas y su sintaxis en lenguaje ensamblador.

Instrucciones lógicas. Son utilizadas para realizar operaciones lógicas sobre los operandos.

AND-NEG-NOT-OR-TEST-XOR

Instrucciones aritméticas. Se usan para realizar operaciones aritméticas sobre los operandos.

ADC-ADD-DIV-IDIV-MUL-IMUL-SBB-SUB

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

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

|OP|disp30|

Formato 2 - Intrucciones Branch y Sethi:

|op|rd|op2|imm22|con inmediato

|op|a|cond|op2|disp22|instruccion branch

Formato 3: - Instrucciones algebraicas:

Este tipo de instrucciones son las más comunes. Estas son las instrucciones algebraicas.

op|rd|op3|rs1|i=1|UNSUED|RS2

op|rd|op3|rs1|i=0| simm13

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

* FORMATO #1 Lo utilizan las instrucciones de CALL

OP disp30

* FORMATO #2 Lo utilizan las instrucciones SETHI y BRANCH (Bicc,FBfcc,CBcc)

OP RD OP2 imm22

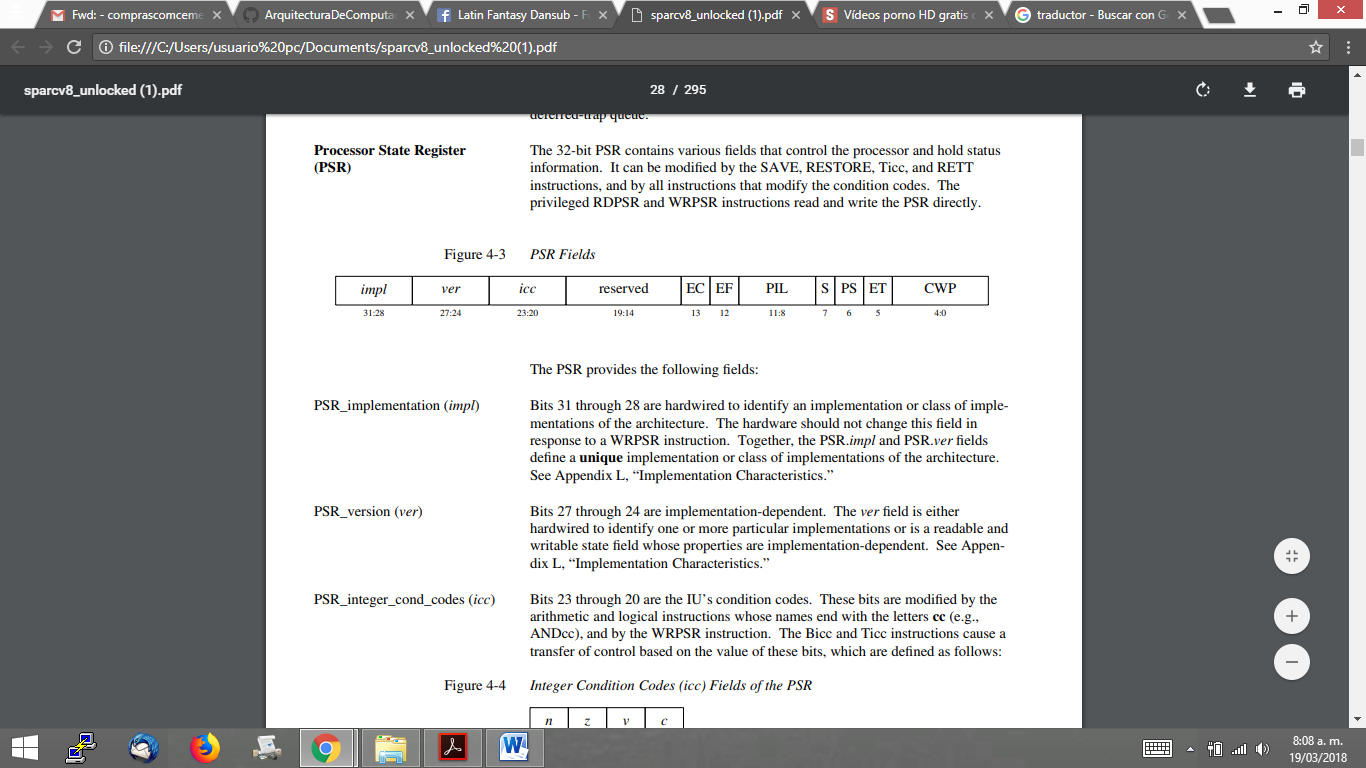
OP a COND OP2 disp22

* FORMATO #3 Lo utilizan las instrucciones memory instructions,arithmetic.logical,shnct and remaining

OP RD OP3 RS1 i UNSUED/ZERO RS2

OP RD OP3 RS1 i UNSUED/ZERO SHNCT

1. ¿Qué 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):**

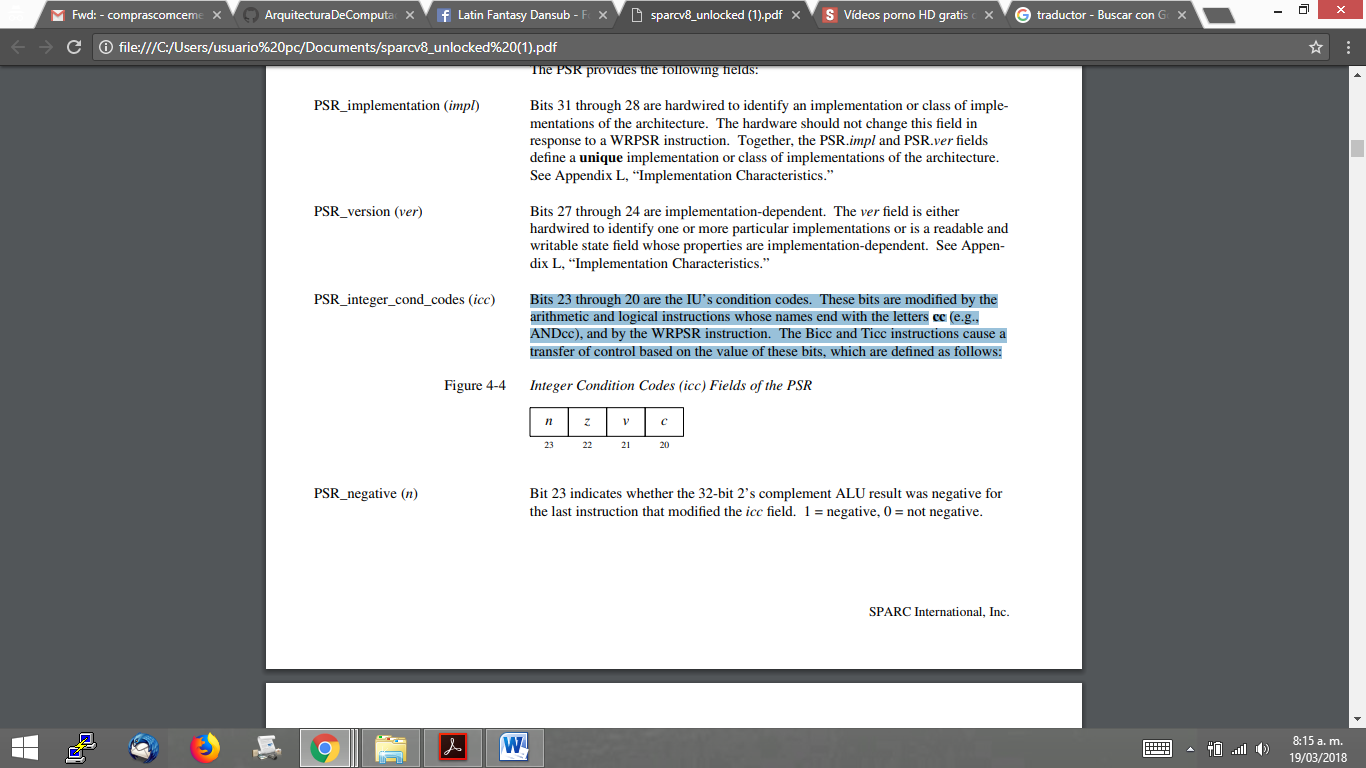
Los bits 31 a 28 están cableados para identificar una implementación o clase de implementaciones de la arquitectura El hardware no debería cambiar este campo en respuesta a una instrucción WRPSR. Juntos, los campos PSR. impl y PSR. ver definir una implementación única o una clase de implementaciones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



* **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).

1. ¿Qué es **ICC** y **CWP**?

* **ICC**

N bit is set if result of operation in negative (MSB = 1)

Z bit is set if result of operation is zero (All bits = 0)

V bit is set if operation produced an overflow

C bit is set if operation produced a carry (borrow on subtraction

E

1. ¿Qué 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

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

El a es un bit en una instrucción branch que anula la ejecución de la siguiente instrucción si la rama es condicional y no tomada o si es incondicional y tomado.

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

El registro %O7 se utiliza como registro temporar para que el CALL le pueda pasar informacion al JMPL

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

10100000000100000010000000000101

10100010000100000011111111111010

10010000000001000100000000010000

COMPLEMENTO A 2

1111111111010

0000000000101

1

0000000000110

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 00000 | 1 | 0000000000101 |

OP RD OP3 RS1 I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 000010 | 00000 | 1 | 1111111111010 |

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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;

}

**24.** Solucione los siguientes programas 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

}

0000000000100

1111111111011

1

\_\_\_\_\_\_\_\_\_\_\_\_\_

1111111111100

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 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 00000 | 1 | 0000000000101 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 000010 | 00000 | 1 | 1111111111100 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10011 | 000000 | 10010 | 1 | 0000001100100 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10101 | 000000 | 10100 | 1 | 0000000010100 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10110 | 000000 | 10000 | 1 | 0000000000010 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10110 | 000100 | 10100 | 1 | 0000000000101 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10111 | 000000 | 10001 | 1 | 0000000000011 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10111 | 000100 | 10100 | 1 | 0000000000100 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10110 | 000000 | 10010 | 1 | 0000000000101 |

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 10111 | 000000 | 10100 | 1 | 0000000000100 |

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10110 | 000000 | 10110 | 0 | 00000000 | 10111 |

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 01000 | 000100 | 10110 | 0 | 00000000 | 10000 |

1. 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;

}

0000 MOV 8,%L0

0004 MOV -14800,%L1

0008 MOV 33,%L2

000C ADD %L0,%L1,%L0

0010 SLL %L1,4,%O0

0014 CMP %L0,%00

0018 BG a OPP

001C SUB %L1,78,%01

OPP

0020 SLL %L1,1,%L1

0024 ADD %L0,%L1,%L2

EXIT

0028 ADD %L0,%L2,%02

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 000000 | 1 | 0000000001000 |

Representacion en Hexadecimal: A0102008

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 00 | 10001 | 100111 | 111111 | 1 | 1111111110001 |

Representacion en Hexadecimal: 233FFFF1

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 000010 | 10001 | 0 | 00000000 | 10001 |

Representacion en Hexadecimal: A2144011

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10010 | 000010 | 00000 | 1 | 00000000100001 |

Representacion en Hexadecimal: A4102011

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000000 | 10000 | 0 | 00000000 | 10001 |

Representación en Hexadecimal: A0040011

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01000 | 100101 | 10001 | 1 | 0000000000100 |

Representación en Hexadecimal: 912C6004

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 010100 | 10000 | 0 | 00000000 | 01000 |

Representación en Hexadecimal: 80A40008

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 11010 | 010000 | 00000 | 0 | 00000000 | 00010 |

Representación en Hexadecimal: 34800002

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 100101 | 10001 | 1 | 00000000000001 |

Representación en Hexadecimal:A32C6001

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10010 | 000000 | 10000 | 0 | 00000000 | 10001 |

Representación en Hexadecimal:A4040011

OP RD OP3 RS I IMM13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01001 | 000100 | 10001 | 1 | 0000001001110 |

Representación en Hexadecimal:9224604E

OP RD OP3 RS I UNSUED RS2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 01010 | 0000000 | 10000 | 0 | 0000000000 | 10010 |

Representación en Hexadecimal:94040012

b.

int main(){

int a = 8;

int b = -10;

if(a!=b){

return c/16;

}

else{

return b\*32;

}

}

c.

int main(){

int a = -21180;

}

00000000000000000101001010111100

11111111111111111010110101000011

1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11111111111111111010110101000100

1111111111111111101011 = 4194283

0101000100 = 324

SETHI HEX

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 10000 | 100 | 1111111111111111101011 |

Representación en Hexadecimal: 213FFFEB

OR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000010 | 10000 | 1 | 0000101000100 |

Representación en Hexadecimal: A0142144

1. 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;

}

x=%I0

Y=%I1

W=%I2

X=%0O

0000 JMPL %07, 8 ,%G0

0004 SUB %I0,%I1,%L0

0008 SLL %I2, 2 ,%L1

000C ADD %L0, %L1, %O1

0010 ADD %O1, 2, %O1

MAIN

0014 MOV 4, %I0

0018 MOV 2 %L1

001C CALL TEST

0020 ADD %O0,45,%O0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 111000 | 01000 | 1 | 000000000001000 |

Representación en Hexadecimal: 81C22008

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01001 | 000010 | 00000 | 1 | 0000101000100 |

Representación en Hexadecimal: 92102144

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10000 | 000100 | 11000 | 0 | 0000000000 | 11001 |

Representación en Hexadecimal: A0260019

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 100101 | 11010 | 1 | 000000000000010 |

Representación en Hexadecimal: A32E1002

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 01001 | 000000 | 10000 | 0 | 000000000 | 10001 |

Representación en Hexadecimal: 92040011

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01001 | 000000 | 01001 | 1 | 000000000000010 |

Representación en Hexadecimal: 92026002

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11000 | 000010 | 00000 | 1 | 00000000000100 |

Representación en Hexadecimal: B0102004

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 11010 | 100 | 1111111111111111110000 |

Representación en Hexadecimal: 353FFFF0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11010 | 000010 | 00000 | 1 | 0001100010000 |

Representación en Hexadecimal: 34102310

|  |  |
| --- | --- |
| 10 | 111111111111111111111111111000 |

Representación en Hexadecimal: 7FFFFFF8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11001 | 000010 | 00000 | 1 | 0000000000010 |

Representación en Hexadecimal: B2102002

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 010000 | 000000 | 00000 | 0 | 00000000 | 101101 |

Representación en Hexadecimal: 9000202D

1. 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 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;

}

0000 MOV 0,%O0

0004 MOV 1,%L0

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

001C MOV %I1,%L1

0020 MOV %L2,1

0024 SUBCC %I2,%L3,G0

0028 BNE a ELSE

002C MOV 1,%I1

0030 SUBCC %L2,I2,GO

0034 BGE SALIR

0038 CALL MUL

003C NOP

0040 BA a ELSE

SALIR

0044 JMPL %O6+8,G0

0048 MOV 4,%I1

004C MOV 8,%I2

0050 CALL POT

0054 MOV 0,O1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01000 | 000010 | 00000 | 1 | 00000000000000 |

Representación en Hexadecimal: OX90102000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01000 | 000010 | 00000 | 1 | 0000000000001 |

Representación en Hexadecimal: OX90102001

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 010100 | 11000 | 0 | 00000000 | 100000 |

Representación en Hexadecimal: 0X80A60010

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 1 | 1011 | 010 | 0000000000000000000010 |

Representación en Hexadecimal: 0X36800002

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 01000 | 000000 | 01000 | 0 | 00000000 | 10010 |

Representación en Hexadecimal: 0X90020012

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 1 | 1000 | 010 | 1111111111111111111101 |

Representación en Hexadecimal: 0X30BFFFFD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 111000 | 01111 | 1 | 0000000001000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 10001 | 000010 | 11001 | 0 | 00000000 | 00000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10010 | 000010 | 10010 | 1 | 0000000000000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 010100 | 11010 | 0 | 00000000 | 10011 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 1 | 1011 | 010 | 0000000000000000000001 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11001 | 000010 | 00000 | 1 | 0000000000001 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 010100 | 10010 | 0 | 00000000 | 11010 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 1 | 1011 | 010 | 0000000000000000000001 |

|  |  |
| --- | --- |
| 01 | 111111111111111111111111110010 |

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 00000 | 100 | 0000000000000000000000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 1 | 1000 | 010 | 1111111111111111111100 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 00000 | 111000 | 01110 | 1 | 0000000001000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11001 | 000010 | 00000 | 1 | 0000000000100 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 11010 | 000010 | 00000 | 1 | 0000000001000 |

|  |  |
| --- | --- |
| 01 | 111111111111111111111111110101 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 01001 | 000010 | 00000 | 1 | 0000000000000 |

1. 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.