EL PROCESADOR

Se llama CPU o Unidad Central de Proceso a la unidad donde se ejecutan las instrucciones de los programas y se controla el funcionamiento de los distintos componentes del computador. Suele estar integrada en un chip denominado microprocesador.

En la Figura 1 podemos ver una CPU convencional, la cual es el corazón de todo computador y es un microchip que aloja millones de transistores en su interior que a su vez forman una serie de circuitos lógicos que permiten ejecutar una determinada variedad de instrucciones.

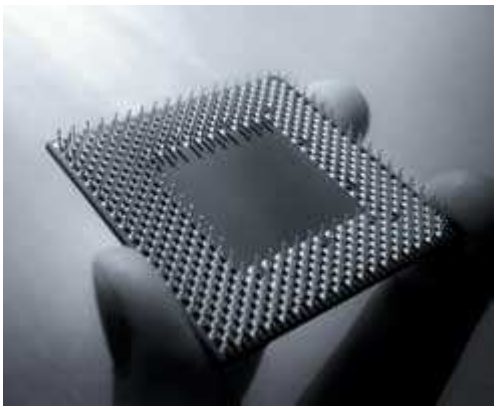

Figura 1. Procesador

fuente: hardwarezone.com

El microprocesador en términos funcionales es una caja negra que recibe como entrada instrucciones y datos, produciendo como salida nuevos datos. Los datos son elaborados en su interior de acuerdo a un algoritmo expresado mediante las instrucciones. El procesador ejecutando las instrucciones secuencialmente genera como resultado los nuevos datos de salida. Los datos de salida pueden ser ligeras modificaciones de los de entrada o incluso copias de los mismos, aunque normalmente la salida serán datos nuevos creados a partir de la entrada y modificados de acuerdo a algoritmos.

Finalmente, los datos de salida producidos por el procesador se almacenan en el sistema de memoria o se envían a los dispositivos periféricos que los necesiten o lo requieran.

En un microprocesador de propósito general, la tarea a realizar se especifica en un programa.

Definición de un programa

Un programa consiste en una secuencia de instrucciones, codificadas (código máquina) de acuerdo a un formato interpretable por el procesador.

El microprocesador sólo será capaz de ejecutar un conjunto básico de instrucciones, cada una de las cuales realiza una operación elemental muy simple. La tarea a realizar se debe especificar de acuerdo a estas instrucciones elementales. El programa por tanto será la secuenciación de las instrucciones elementales de tal manera que lleven a cabo el algoritmo que describe la tarea u operación que se desea realizar.

Ejecución de las instrucciónes

El microprocesador secciona en varias fases de ejecución la realización de cada instrucción:

- Fetch, lectura de la instrucción desde la memoria principal.
- Decodificación de la instrucción, es decir, determinar qué instrucción es y por tanto qué se debe hacer.
- Fetch de los datos necesarios para la realización de la operación.
- Ejecución.
- Escritura de los resultados en la memoria principal o en los registros.

Cada una de estas fases se realiza en uno o varios ciclos de CPU dependiendo de la estructura del procesador, la duración de estos ciclos viene determinada por la frecuencia de reloj la cual es generada desde un oscilador de cuarzo capaz de generar pulsos a un ritmo constante de modo que genera varios ciclos (o pulsos) en el lapso de un segundo.

Partes de un Procesador

En el interior de un microprocesador encontramos claramente definidas las siguientes partes:

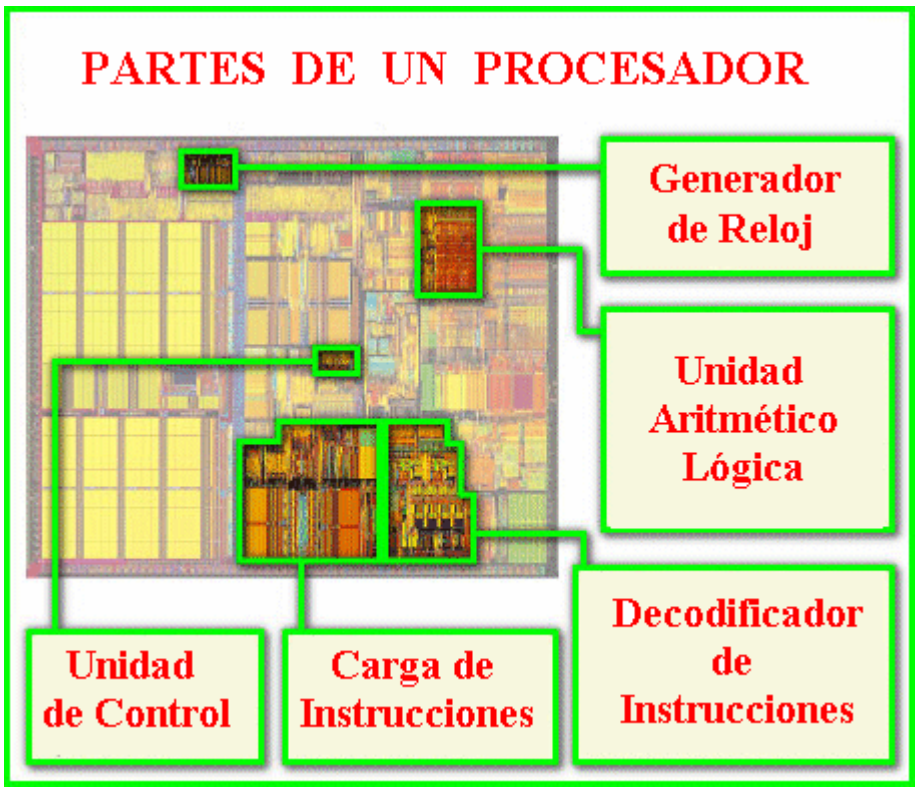

Figura 2. Partes de un Procesador

Fuente: es.wikipedia.org

Unidad de control

Es la encargada de realizar el control del proceso, generando las señales necesarias para activar los componentes de la unidad de proceso que actuarán sobre los datos en el instante de tiempo que corresponda. Sus tareas principales son:

- Controlar la secuencia de instrucciones a ser ejecutadas.
- Controlar el flujo de datos entre las diferentes partes que forman un computador.
- Interpretar las instrucciones.
- Regular tiempos de acceso y ejecución en el procesador.
- Enviar y recibir señales de control de periféricos externos.

Elementos que forman parte de la unidad de control y que desempeñan tareas específicas son:

- Decodificador de Instrucciones: Unidad que interpreta el contenido del registro de instrucciones y permite generar las señales adecuadas para ejecutar la instrucción.
- Decodificador de Direcciones: Unidad que interpreta la dirección en el registro de direcciones de Memoria (MAR) y selecciona la posición de memoria a ser accedida.
- Registros en la unidad de control: Estos son elementos de almacenamiento, donde se guardan temporalmente valores durante la ejecución de un programa.

En la unidad de control se dispone generalmente de los siguientes registros:

- Contador de Programa: Guarda la dirección de la siguiente instrucción a ser ejecutada.
- Registro de Instrucciones: Guarda la instrucción en curso de ejecución.
- Registro de Estado: Mantiene información "bits de estado" o "flags" con información sobre lo que ha pasado en la operación realizada por la ALU.
- Registro de Direcciones de Memoria (MAR): Guarda la dirección del dato que va a ser accedido en la memoria.

Unidad de proceso

Es un conjunto de recursos en los cuales son procesados los datos. En estos recursos se realizan operaciones sobre los datos y se obtiene un resultado. El control de la operación a realizar y que recursos intervienen o no para realizar una determinada tarea se controla mediante las señales que provienen de la unidad de control.

Los elementos que forman parte de la unidad de proceso son:

- Unidad Aritmético Lógica: Es la unidad encargada de realizar las operaciones matemáticas, operaciones lógicas y comparaciones. Internamente esta formada por circuitos lógicos elementales para realizar estas operaciones: sumadores, incrementos, operadores lógicos, desplazamientos, rotaciones, comparaciones...
- Registros en la unidad de proceso: Tienen la función de almacenar temporalmente datos durante la ejecución del programa.
    - Acumulador: Almacena los resultados parciales y el resultado final de la operación realizada por la ALU.
    - Registros de propósito general: Permiten guardar información temporalmente durante la ejecución del programa.

En los procesadores actuales pueden existir múltiples unidades de cada uno de los recursos para aumentar el grado de paralelización al ejecutar un proceso, y de esta manera será posible ejecutar dos instrucciones simultáneamente. Por otro lado, existen procesadores que no son de propósito general y que llamaremos procesadores de propósito específico, los cuales son diseñados para realizar una tarea concreta muy especializada.

En cada ciclo de reloj la unidad de control debe generar todas las señales de control, es decir, asignar valores a determinados bits que controlan la unidad de tratamiento. La unidad de control es básicamente una máquina secuencial. (Ver Anexo 1.)

Diseño de la Memoria

La memoria del computador consta de circuitos que retienen datos durante algún intervalo tiempo. Es uno de los componentes fundamentales de un computador y va acoplada al procesador.

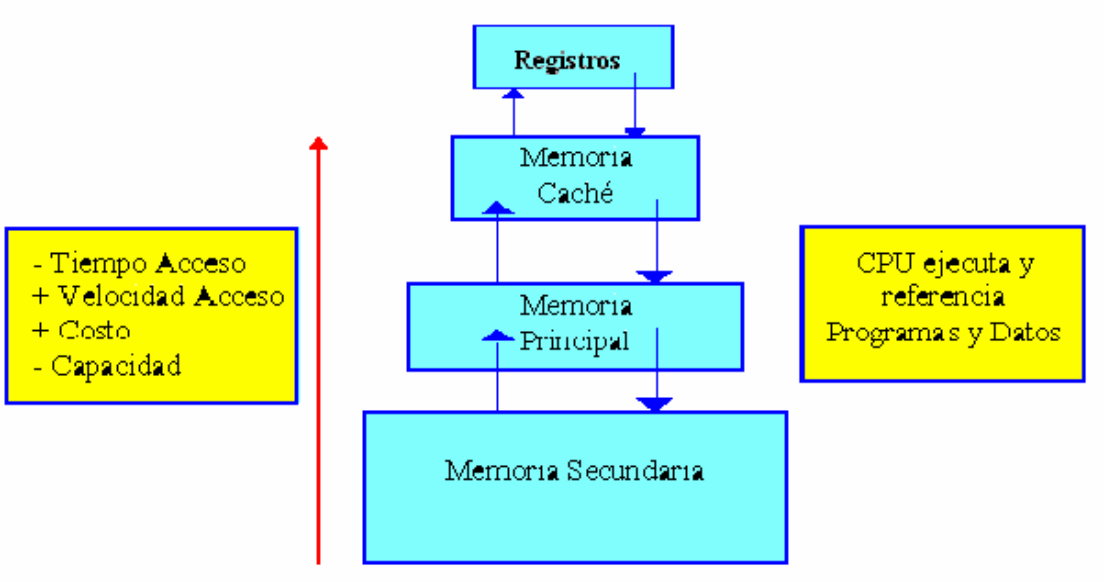

Figura 3. Jerarquía de memoria

Fuente: cosmos.ujaen.es

En el contexto en que estamos, nos referiremos a la memoria de almacenamiento principal.

El almacenamiento primario está directamente conectado al microprocesador para que este funcione correctamente. El almacenamiento primario consiste en tres tipos de almacenamiento:

- **Los registros del procesador:** Un registro es una memoria de alta velocidad, poca capacidad y está integrada al procesador, que permite guardar y acceder a valores muy usados, generalmente en operaciones matemáticas. Los registros están en la parte más alta de la jerarquía de memoria, y son la manera más rápida que tiene el sistema de almacenar datos. Estos se miden generalmente por el número de bits que almacenan; por ejemplo, un registro de 8 bits o un registro de 32 bits. Los procesadores tienen varios registros que son usados con propósitos específicos, como el contador de programa, de los cual sabremos mas adelante. Por ejemplo, en la arquitectura IA32, el conjunto de instrucciones define 8 registros de 32 bits los cuales son cuatro de propósito general y cuatro de índices de memoria.
- **La memoria caché:** es un tipo especial de memoria interna usada en muchos microprocesadores para mejorar su eficiencia o rendimiento. Parte de la información de la memoria principal se duplica en la memoria caché. Comparada con los registros, la caché es ligeramente más lenta pero de mayor capacidad. Sin embargo, es más rápida pero de mucha menor capacidad que la memoria principal.
- **La memoria principal**: contiene los programas en ejecución y los datos con que operan. La unidad aritmético-lógica puede transferir información muy rápidamente entre un registro del procesador y direcciones de memoria.
- **La pila**: La pila es simplemente un conjunto de datos que se almacenan en una zona de la memoria principal, y en la cual los datos se guardan de la

siguiente manera: LIFO, es decir, Last In, First Out (último en entrar, primero en salir). Su uso principalmente es para almacenar datos, para llevar control del programa, para pasar variables entre funciones y para almacenar variables locales. El programador es el responsable de reservar el espacio apropiado para la pila.

Representación de los números

Tabla 1. Representación de los números

Binario (base 2)	Octal (base 8)	Decimal (base 10)	Hexadecimal (base 16)
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Estándar de números de puntos flotante IEEE

El estándar de la IEEE para operaciones de punto flotante (IEEE 754) es el más ampliamente usado para este fin, y es seguido por muchas de las implementaciones de CPU y FPU. El estándar define formatos para la representación de números en punto flotante (incluyendo el cero) y valores desnormalizados, así como valores especiales como infinito y NaNs. Conjuntamente con un conjunto de operaciones en

punto flotante que opera sobre estos valores, también especifica cuatro modos de redondeo y cinco excepciones (incluyendo cuando dichas excepciones ocurren, y que sucede en dichos momentos).

IEEE 754 especifica cuatro formatos para la representación de valores en punto flotante: precisión simple (32 bits), precisión doble (64 bits), precisión simple extendida (≥ 43 bits, no usada normalmente) y precisión doble extendida (≥ 79 bits, usualmente implementada con 80 bits). Solo los valores de 32 bits son requeridos por el estándar, los otros son opcionales. Muchos lenguajes especifican que formatos y aritmética de la IEEE implementan, a pesar de que a veces son opcionales.

Modos de Direccionamiento

La potencia y capacidad del juego de instrucciones de un procesador viene determinada no sólo por el número de instrucciones, sino por los distintos modos de direccionamiento de que disponga.

Los modos de direccionamiento son las diferentes maneras de especificar un operando dentro de una instrucción y cómo se especifican e interpretan las direcciones de memoria según las instrucciones.

A continuación se describen los modos de direccionamiento más característicos:

a) **Direccionamiento inmediato**: En la instrucción está incluido directamente el operando o dato.
b) **Direccionamiento directo:** El campo de operando en la instrucción contiene la dirección en memoria donde se encuentra el operando.
c) **Direccionamiento indirecto:** El campo de operando contiene una dirección de memoria, en la que se encuentra la dirección efectiva del operando.
d) **Direccionamiento absoluto**: El campo de operando contiene una dirección en memoria, en la que se encuentra la instrucción..

e) **Direccionamiento por registro:** Sirve para especificar operandos que están en registros.
f) **Direccionamiento indexado o de desplazamiento:** Combina el modo directo e indirecto mediante registros.

Instrucciones del procesador

La potencia de un microprocesador dependerá de su velocidad en la ejecución de las instrucciones pero también de la variedad y tipo de operaciones que sea capaz de ejecutar. Cada microprocesador dispone de un repertorio de instrucciones propio, que se conoce como juego de instrucciones del microprocesador, este juego depende de la circuitería interna con la cual ha sido diseñado. La programación más básica y directa del procesador debe hacerse usando ese juego de instrucciones de la máquina concreta con la que estemos trabajando.

Dentro de la lista de instrucciones del microprocesador existirán un conjunto asociado a las distintas operaciones que puede realizar la ALU. Cuando la Unidad de Control identifica un código de instrucción que indica una determinada operación, generará las señales oportunas para activar la operación correspondiente dentro de la ALU.

El formato general de las instrucciones que son ejecutadas por el microprocesador se ajusta al formato:

<Código de operación> [<operando1> [<mas operandos>]]

Supongamos un microprocesador que puede realizar la siguiente operación:
Sumar el contenido de un registro con un determinado valor y guardar el resultado en el propio registro.

Consideremos que el código de dicha operación es 0001. El registro se llamará AX, y el valor a sumar es 3 (11 en binario). La instrucción que debería recibir el microprocesador para sumar el contenido de AX con el valor 2 es:

0001 0011 (código de operación, y primer y único operando)

Al entrar en un ciclo de búsqueda la unidad de control extraerá desde la posición de memoria señalada por IP el código de la siguiente instrucción a ejecutar. Al recibir el código de instrucción 0001 se enviarán las señales de control necesarias para:

1. Hacer llegar los operándos implicados en la suma a la ALU (contenido del registro AX, y valor 0011)
2. Indicarle a la ALU que debe realizar una operación de SUMA.
3. Almacenar el resultado de la suma efectuada en el registro AX

Estas señales son una simplificación de todas las señales elementales que habrían de ser generadas.

El lenguaje ensamblador

Es un tipo de lenguaje de bajo nivel utilizado para escribir programas y sus instrucciones están formadas a partir de opcodes o códigos de operación constituyendo la representación más directa del código de máquina.

Cuando abstraemos los opcodes y los sustituimos por una palabra que sea una clave de su significado, a la cual comúnmente se le conoce como mnemónico, tenemos el concepto de Lenguaje Ensamblador. Para mas adelante generar el código objeto que entiende el procesador se requiere un compilador.

Como se puede ver, el lenguaje ensamblador es directamente traducible al lenguaje de máquina, y viceversa; simplemente, es una abstracción que facilita su uso para los programadores.

La importancia del lenguaje ensamblador radica principalmente que se trabaja directamente con el microprocesador; tiene la ventaja de que en él se puede realizar cualquier tipo de programas que en los lenguajes de alto nivel no lo pueden realizar además que los programas en ensamblador ocupan menos espacio en memoria ya que pueden ser mas eficientes si son generados por un programador, al contrario que cuando son generados por un compilador el cual genera mucho código redundante.

También tiene desventajas como mayor tiempo de programación, ya que como es un lenguaje de bajo nivel requiere más instrucciones para realizar el mismo proceso, en comparación con un lenguaje de alto nivel. Por otro lado, requiere de más cuidado por parte del programador, pues es propenso a que los errores de lógica se reflejen más fuertemente en la ejecución o con el peligro que se afecten recursos inesperadamente.

El lenguaje de máquina está formado por instrucciones sencillas, que pueden especificar:

- Posiciones de memoria específicas.
- Registros específicos para operaciones aritméticas, direccionamiento o control de funciones.
- Modos de direccionamiento usados para interpretar operándos.

Las operaciones más complejas se realizan combinando estas instrucciones sencillas, que pueden ser ejecutadas secuencialmente o mediante instrucciones de control de flujo.

Entre las operaciones disponibles en la mayoría de los conjuntos de instrucciones se incluye:

- Mover
    - Llenar un registro con un valor constante
    - Mover datos de una posición de memoria a un registro o viceversa
    - Escribir y leer datos de dispositivos
- Calcular
    - Sumar, restar, multiplicar o dividir los valores de dos registros, colocando el resultado en uno de ellos o en otro registro.
    - Ejecutar operaciones binarias, incluyendo operaciones lógicas (AND/OR/XOR/NOT)
    - Comparar valores entre registros (mayor, menor, igual)
- Afectar el flujo del programa
    - Saltar a otra posición en el programa y ejecutar instrucciones allí
    - Saltar si se cumplen ciertas condiciones (IF)
    - Saltar a otra posición, pero guardar el punto de salida para retornar (llamada a subrutinas)

Algunos microprocesadores incluyen instrucciones mas complejas que con una sola instrucción hace lo mismo que en otros microprocesadores puede requerir una larga serie de instrucciones, por ejemplo:

- Salvar varios registros en la pila de una sola vez
- Mover grandes bloques de memoria
- Operaciones aritméticas complejas o de punto flotante

SIMUPROC

SimuProc es el nombre que se le ha dado a la aplicación y surge de Simulador de Procesador.

Arquitectura Interna

SimuProc tiene implementado un conjunto de operaciones y es lo que se denomina un procesador hipotético. Estas operaciones se denominan instrucciones del procesador y permitirán abordar un gran número de tareas, empleando para realizarlas una combinación de las instrucciones básicas.

Para diseñar un procesador se debe indicar como funcionan las operaciones básicas para tratar la información y llevar a cabo una tarea concreta. Para que un procesador realice sucesivamente cada una de estas operaciones se necesitan:

1. Un algoritmo con los pasos a seguir para realizar cada operación.
2. Los recursos necesarios para implementar el algoritmo: elementos de memoria y tratamiento de datos.

Un procesador se diseña con una arquitectura que le permita efectuar las funciones de búsqueda y ejecución de instrucciones. Para eso cuenta con los siguientes elementos:

Decodificador de instrucciones: Elemento que recibe la instrucción en código binario e interpreta su significado. Consta de un decodificador y de una ROM de microinstrucciones en la que residen los pasos que se deben realizar para ejecutar cada una de las instrucciones.

Unidad de Control: Bloque encargado de generar y transmitir las señales de gobierno y sincronismo a todo el sistema, para ejecutar la instrucción previamente

decodificada. Una vez que recibe las microinstrucciones de la ROM interna, genera las señales necesarias para poner al resto de los bloques en el modo que corresponda a la instrucción a ejecutar.

Unidad aritmético-lógica (ALU): Es la encargada de realizar las operaciones de carácter lógico y aritmético de los datos que poseen el acumulador y el otro registro usado para la operación.

Acumulador (AX): Registro que contiene uno de los operándos que intervienen en la operación que realiza la ALU, así como el resultado una vez realizada dicha operación.

Registros de propósito general: Se emplean como memorias temporales y pueden operar de modo independiente o unidos por parejas para aumentar así la longitud de palabra.

Registro puntero de pila (SP, stack pointer): Su misión es la de controlar una zona de la memoria, con estructura LIFO (Last In First Out), en la que se salva temporalmente el contenido de algunos registros del procesador o contenido de la memoria.

Registro de estado: Registro en el que se depositan las condiciones o resultados de una determinada operación realizada. Cada bit de este registro se utiliza para guardar información específica, los cuales se denominan banderas (flags). Son los siguientes:
C: Arrastre aritmético.
Z: Indica si la operación realizada ha dado cero.
N: Indica el signo de la operación realizada. también conocido como S (Sign)
O: registro de overflow o desbordamiento.

Número de Registros y Puertos de E/S

SimuProc cuenta con 5 registros y 4 puertos de E/S repartidos así:

Registros de procesador

3 registros de propósito general

2 registros de manejo de Pila.

Puertos de E/S

1 puerto de Entrada/Salida

2 puertos de Entrada

1 puerto de Salida

Estos registros y puertos cuentan con 5 bits = 32 posibles valores. Calculados así:

Desde 00 hasta 1F

Siendo los primeros 16 para los registros de procesador y los siguientes 16 para los puertos de E/S.

Tabla 2. Códigos asignados a los registros y puertos

Desc.	**Registro**	**Cod. Hexa**	**Binario**
Registro A	AX	01	00001
Registro B	BX	02	00010
Registro C	CX	03	00011
Base Pila	BP	0A	01010
Puntero Pila	SP	0B	01011

Desc.	**Puertos**	**Cod. Hexa**	**Binario**
Teclado y Pantalla.	1	11	10001
Reloj	8	18	11000
Switches	9	19	11001
PC-Speaker	13	1D	11101

El *teclado* ha sido implementado para aceptar ingreso de valores numéricos, los valores textuales como mensajes de texto se pueden ingresar desde el código fuente para ser desplegados durante la ejecución de los programas.
La *pantalla* muestra los mensajes de salida de la ejecución de los programas.
El *reloj* retorna los segundos del sistema, tiene varios usos como por ejemplo generar números aleatorios.
El *Switch* puede ser usado para modificar los bits de ese puerto y realizar programas que simulen la operación de dispositivos de hardware.
El *PC-Speaker* puede generar frecuencias entre 7Hz y 32767Hz dependiendo de los parámetros que se le den, el tono y la duración.

Tamaño de Memoria

La capacidad de la memoria Simulada es de 4096 posiciones de 16 bits cada una; las posiciones en hexadecimal: Desde **000** hasta **FFF**.

Esta memoria es suficiente para ejecutar gran variedad de programas desde algunos simples hasta otros más complejos.

El simulador trabaja con constantes y variables en binario y direcciones (posiciones de memoria) en hexadecimal.

Los 16 bits pueden ser usados para almacenar datos o instrucciones con sus parámetros.

Los **números enteros** que se pueden representar van desde 0 hasta 65535 y el signo lo determina el estado de la bandera N al momento de hacer la operación con dicho numero.

Los **números de punto flotante** se representan usando el estándar IEE754 descrito anteriormente, el cual usa 32 bits; para esto en SimuProc usamos dos posiciones consecutivas de memoria estando los bits más significativos en la primera.

La Pila

La pila es un buffer usualmente implementado como un bloque de n bytes o words consecutivos.
Un problema con las pilas es que el programador las puede hacer muy pequeñas y puede que ocurra un stack overflow o desbordamiento de pila si se esta intentando usar la instrucción PUSH para enviar algo cuando ya la pila esta llena; o stack underflow si se usa la instrucción POP estando la pila vacía.
Por cada PUSH debe de haber luego un POP y es responsabilidad del programador de que se cumpla de lo contrario el programa podría fallar.

Operaciones Soportadas

Una de las primeras decisiones a la hora de diseñar un procesador es decidir cual será su juego de instrucciones. Este conjunto de instrucciones (órdenes) es el lenguaje que realmente entiende el procesador y constituye lo que se conoce como lenguaje ensamblador o lenguaje-máquina.

La decisión es importante por dos razones. Primero: el juego de instrucciones decide el diseño del procesador. Segundo: cualquier operación que deba ejecutarse con el procesador deberá poder ser descrita en términos de este "lenguaje" elemental; recordar que los compiladores e intérpretes son en realidad traductores desde el lenguaje de alto nivel a este lenguaje-máquina.

Se puede decir que frente a esta decisión caben dos filosofías de diseño. La primera conduce a máquinas denominadas CISC ("Complex Instruction Set Computer"); las máquinas construidas según el otro criterio se denominan RISC ("Reduced Instruction Set Computer").

Como puede deducirse de estos nombres, las máquinas CISC utilizan instrucciones muy complejas, las cuales son muy descriptivas y específicas, lo que se traduce en varias consecuencias:

1. El lenguaje debe contener un amplio surtido de ellas (una para cada circunstancia distinta).

2. Son instrucciones de ejecución rápida pero la circuitería del procesador es compleja.

3. Para un trabajo específico se requieren pocas instrucciones (siempre hay una que resuelve el problema).

Las máquinas RISC representan el enfoque opuesto. Utilizan instrucciones muy simples, que deben ser cuidadosamente escogidas, porque cualquier operación debe ser expresada como una secuencia de estas pocas instrucciones. Las consecuencias son justamente opuestas a las anteriores:

1. El lenguaje contiene un conjunto pequeño de instrucciones.

2. Las instrucciones son muy complejas, por tanto de ejecución rápida. La circuitería es más simple que en los procesadores CISC.

Para cualquier operación se requieren varias instrucciones elementales.

Naturalmente cada criterio tiene sus pros y sus contras en lo que a rendimiento se refiere. En las máquinas RISC, lentitud de cada instrucción frente a poca cantidad de ellas; en las CISC, rapidez individual aunque hay que ejecutar un mayor número.

Tabla 3. Juego de instrucciones que puede ejecutar SimuProc

Código hexadecimal	**Instrucción y parámetros**	**Descripción**
03	XAB	Intercambia los valores de los registros AX y BX. Esta instrucción no necesita parámetros.
04	CLA	Hace AX = 0
06	PUSH [registro]	Envía el valor del registro especificado a la pila.
07	POP [registro]	Trae de la Pila el ultimo Valor llevado por PUSH (indicado por el registro SP) y lo almacena en el registro especificado.
08	INC [dest]	Incrementa en 1 el destino especificado, el parámetro puede ser una dirección de memoria o un registro. Si en la posición de memoria EB esta el valor 1001, al ejecutar INC EB se obtiene que ahora el valor de EB es 1010.
09	DEC [dest]	Decremento en 1 el destino especificado, Si el destino queda = 0, se vuelve Z = 1
10	MOV [dest,orig]	Copia el valor almacenado en el origen al destino. El destino y/o origen pueden ser registros o direcciones de memoria o combinación de estos. Para copiar lo que esta en la posición de memoria 12E a la

		posición D2 se usa la instrucción MOV D2,12E
11	AND [dest,orig]	Y lógico, hace un Y lógico entre todos los bits de los dos operándos escribiendo el resultado en el destino. Los parámetros pueden ser direcciones de memoria o Registros. La siguiente regla aplica: 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0 Si en AX se tiene el número 1001101 y en la pos 3F tengo el numero 11011. al ejecutar la instrucción AND AX,3F obtendré en AX el resultado 1001. El Y lógico lo que hace es dejar los bits en común que tengan los dos números.
12	NOT [destino]	NO lógico, invierte los bits del operando formando el complemento del primero. NOT 1 = 0 NOT 0 = 1 Si en AX se tiene 10011 al ejecutar NOT AX obtengo AX=1111111111101100
13	OR [dest,orig]	O inclusive lógico, todo bit activo en cualquiera de los operándoos será activado en el destino. La siguiente regla aplica: 1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1

		0 OR 0 = 0 Si en 3A se tiene el numero 1001101 y en la pos 3B se tiene el numero 11011. al ejecutar la instrucción OR 3A,3B obtendré en 3A el resultado 1011111.
14	XOR [dest,orig]	O exclusivo, realiza un O exclusivo entre los operándoos y almacena el resultado en destino. La siguiente regla aplica: 1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1 0 XOR 0 = 0 Si en 3A se tiene el numero 1001101 y en la pos 3B se tiene el numero 11011. al ejecutar la instrucción XOR 3A,3B obtendré en 3A el resultado 1010110.
15	ROL [dest,veces]	Rota los bits a la izquierda las veces especificadas(en decimal), los bits que salen por la izquierda re-entran por la Derecha. En el Carry Flag queda el ultimo bit rotado. Supongamos que en la posición 7E se tiene el numero 101110 Al Ejecutar ROL 7E,2 obtengo 7E=10111000 C=0 Al Ejecutar ROL 7E,7 obtengo 7E=1011100000000 C=0 Al Ejecutar ROL 7E,13 obtengo 7E=1100000000000101 C=1
16	ROR [dest,veces]	Rota los bits a la derecha las veces

		especificadas(en decimal), los Bits que salen por la derecha re-entran por la izquierda. El Carry Flag guarda el ultimo bit rotado.
17	SHL [dest,veces]	Desplaza los bits a la izquierda el numero de veces especificado(en decimal), agregando ceros a la derecha, el Carry Flag guarda ultimo bit desplazado.
18	SHR [dest,veces]	Desplaza los bits a la Derecha el numero de veces especificado(en decimal), agregando ceros a la izquierda, el Carry Flag guarda ultimo bit desplazado.
20	ADD [mem]	Sumar: AX = AX + el contenido de la dirección de memoria. Si el resultado de la suma supera los 32 bits, el resultado queda así: en AX los bits mas significativos y en BX los menos, también se activa el Overflow flag.
21	SUB [mem]	Restar: AX = AX - el contenido de la dirección de memoria.
22	MUL [mem]	Multiplicar: AX = AX * el contenido de la dirección de memoria. Si el numero resultante supera su longitud en binario de 32 bits, este resultado se parte almacenando los bits mas significativos en el Registro BX.
23	DIV [mem]	AX = AX / el contenido de la dirección de memoria, BX=AX % el contenido de la dir de memoria (BX = modulo o residuo).
24	CLN	Limpia el Negative Flag. N = 0
25	CLC	Limpia el Carry Flag. C = 0

26	STC	Pone el Carry Flag. C = 1
27	CMC	Complementa (invierte) el Carry Flag. Si C = 1 vuelve C = 0 y viceversa
29	LOOP [mem]	Decrementa CX y salta a la Pos de memoria si CX no es cero.
30	JMP [mem]	Salto incondicional. PC = dirección de memoria donde esta la siguiente instrucción a ejecutar.
31	JEQ [mem]	Saltar si son iguales. Es decir, Si Z = 1, PC = contenido de la memoria.
32	CMP [mem]	Compara AX con [mem], si AX es mayor, Z=0 N=0, si es igual Z=1 N=0, si es menor Z=0 N=1
33	JME [mem]	Saltar si es Menor. Es decir, Si N = 1, PC = contenido de la memoria.
34	JMA [mem]	Saltar si es Mayor. Es decir, Si Z = 0 y N = 0, PC = contenido de memoria
35	JC [mem]	Saltar si el Carry Flag esta activado. Si C = 1, PC = contenido de memoria.
36	JNC [mem]	Saltar si el Carry Flag no esta activado. Si C = 0, PC = contenido de memoria.
37	JO [mem]	Saltar si el Overflow Flag esta Activado. Si O = 1, PC = contenido de memoria
38	JNO [mem]	Saltar si el Overflow Flag no esta activado. Si O = 0, PC = contenido de memoria
39	JNE [mem]	Saltar si no son iguales. Si Z = 0, PC = contenido de memoria
40	LDT	Lee un valor del Teclado y lo lleva al registro AX

41	EAP	Escribe en Pantalla el contenido del registro AX
42	MSG	Muestra un mensaje en pantalla
50	LDB [mem]	La instrucción carga en AX el contenido de memoria almacenado en [mem] + BX
51	STB [mem]	Guarda el contenido de AX en la dirección [mem] + BX
55	LDF [mem]	Carga en BX y AX el número de 32 bits almacenado en [mem] y [mem+1] En BX quedan los bits mas significativos
56	STF [mem]	Guarda el número de 32 bits almacenado en BX y AX e la [mem] y [mem+1]
60	ADDF [mem]	Suma números de 32 bits: en BX y AX, queda el resultado de la suma de estos mas el contenido de [mem] y [mem+1]
61	SUBF [mem]	Resta el número de 32 bits: BX y AX = BX y AX – [mem] y [mem+1]
62	MULF [mem]	BX y AX = BX y AX * [mem] y [mem+1]
63	DIVF [mem]	BX y AX = BX y AX / [mem] y [mem+1] En CX queda el residuo en entero de 16 bits
64	ITOF	Convierte el número entero (16 bits) almacenado en AX en un número Real (32 bits) El resultado queda en BX y AX
65	FTOI	Convierte un número Real (32 bits) BX y AX en un entero (16 bits) El resultado queda en AX
80	IN registro,puerto	Lleva al Registro el valor retornado por el puerto especificado. Ej.: IN AX,8 ;lleva a

		AX el valor retornado por el puerto 8 (Reloj: los segundos del sistema).
81	OUT puerto,registro	Escribe en el puerto especificado, el valor del registro
90	NOP	Esta operación no hace nada. Útil para cuando se modifica la memoria para parchar código y desactivar instrucciones
99	HLT	Terminar Programa Todo Programa lleva esta instrucción para indicarle al simulador que el programa ha terminado su ejecución

Los siguientes códigos serán reservados para mantener compatibilidad con versiones anteriores de SimuProc

- 01 - LDA [mem] : Carga en AX el contenido de la dirección de Memoria especificada.
- 02 - STA [mem] : Guarda el contenido de AX en la dirección de Memoria especificada.

La Simulación

Básicamente la simulación es un intento de imitar las funciones de un dispositivo y una de esas formas es interpretando. Así lo hace SimuProc.

El simulador lee las instrucciones de la memoria de a una posición a la vez y las decodifica, luego ejecuta los comandos apropiados en los registros simulados, memoria y dispositivos de entrada/salida. El algoritmo general que describe este simulador es el siguiente:

```
Mientras Que (CPUestaCorriendo)
      {
            HacerFetch()
            InterpretarCodigo()
      }

HacerFetch()
  {
      Va al PC
      Va a la dirección que apunta el PC
      Hace IR = MEM[PC]
      Incrementa PC
  }

InterpretarCodigo()
  {
      Si tiene que ir a Memoria
      va a Memoria
      switch(OpCode)
      almacena resultados.
  }

switch(OpCode)
  {
    case OpCode1:
    case OpCode2:
    ...
  }
```

Las virtudes de este modelo incluyen la facilidad de depurar, la portabilidad y la facilidad de sincronización entre los diferentes dispositivos del procesador.

Implementando el bucle principal como: *"Mientras Que (CPUestaCorriendo)"* donde CPUestaCorriendo es una variable booleana, se obtienen varias ventajas como terminar la simulación en cualquier momento colocando la variable en falsa.

Ahora lo primero que hacemos cuando estamos dentro del bucle es leer el siguiente opcode y modificar el PC (Program Counter)

Cuando llegamos al momento de interpretar el Opcode habían dos opciones, una es "*switch(OpCode)*" la cual se compila en una tabla de saltos la cual es bastante eficiente y no como se suele creer que se genera una cadena de "*si ()...entonces...si()...entonces*". Otra manera es hacer una tabla de funciones y llamar la apropiada.

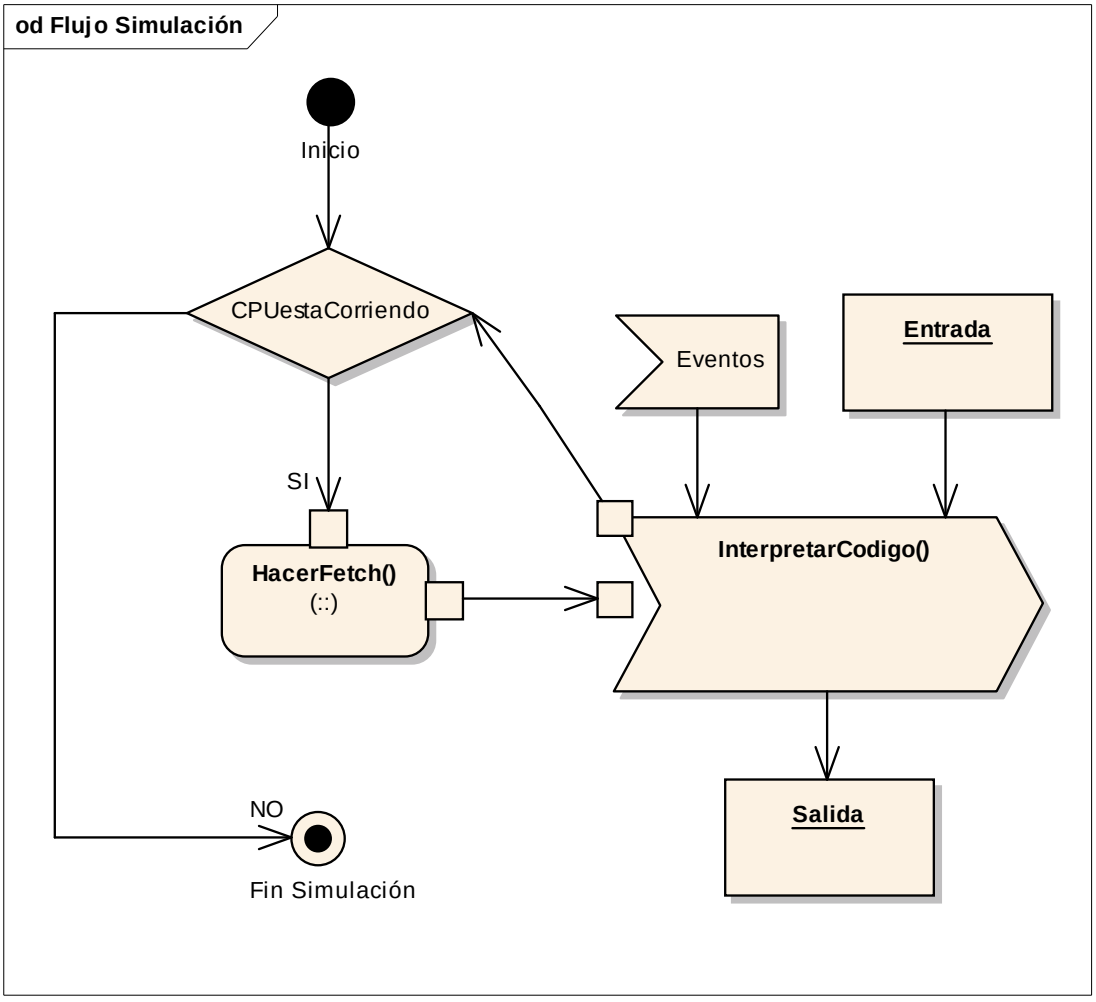

Figura 4. Diagrama de Flujo de la Simulación

CONSTRUCCION DEL SOFTWARE

Metodología

Por la forma en que fue concebida la idea de SimuProc y la manera que evolucionó, este ha sido desarrollado usando la metodología de Programación Extrema.

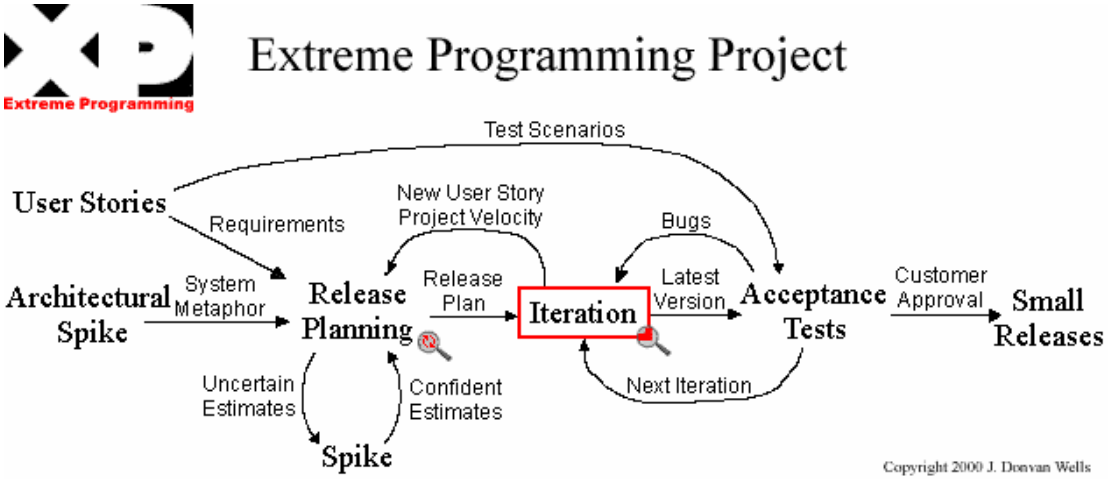

Figura 5. Metodología de programación extrema.

Fuente: http://www.extremeprogramming.org

La programación extrema o en ingles eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck en 1999. De los procesos ágiles de desarrollo de software es el más sobresaliente. La programación extrema se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de los proyectos, ya que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. Se puede

considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- Frecuente interacción del programador con el usuario.
- Corrección de todos los errores antes de añadir nuevas funcionalidades.
- Se hacen entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Simplicidad en el código: es la mejor manera para que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Mientras más simple es el sistema, menos tendrá que comunicar sobre este, lo que lleva a una comunicación más completa.

Construcción

En conjunto con el asesor, se decidió que era lo que se quería hacer. Para ello se utilizaron varias frases en las que se decía que debe hacer el software. Estas frases

son mas largas que un requisito y menos que un caso de uso; se evaluó para cada requisito cuanto tiempo podría tomar, que debía ser corto, de aproximadamente una semana. Se podía estimar con cierta fiabilidad un trabajo que lleve unos días, pero la estimación es menos fiable si es de un plazo superior a una semana. Cuando era más largo, se partía el requisito en otros más pequeños. Luego se ordenaron en el orden en que se iban a desarrollar y se establecieron las mini-versiones, de forma que cada mini-versión implementa varias de estos requisitos o necesidades.

Toda esta planificación no fue muy exacta. No se puede saber todo lo que va a ser necesario ni evaluar los tiempos correctamente. La planificación se revisó y modificó continuamente a lo largo del proyecto. Muchos de los requisitos se modificaron, añadieron o eliminaron sobre la marcha.

Siempre que se terminaba una mini versión se hacia una prueba para ver si la versión cumplía perfectamente con el requisito. Estas pruebas se hacían manualmente.

Este ciclo se va repitiendo una y otra vez hasta que se está satisfecho con la aplicación. De esta manera en paralelo se iba haciendo la documentación.

Herramientas

Para el desarrollo de SimuProc se utilizó principalmente Borland C++ Builder 5 el cual es un lenguaje de programación muy poderoso y con grandes posibilidades.

Requisitos Funcionales

Organización	**Universidad Eafit**
Dirección	Carrera 49 N°7 Sur - 50 | Medellín - Colombia - Suramé rica
Teléfono	(57) (4) - 2619500

Fax	
Comentarios	Ninguno

Participante	**Vladimir Yepes Bedoya**
Organización	Universidad Eafit
Rol	Desarrollador
Es desarrollador	Sí
Es cliente	Sí
Es usuario	Sí
Comentarios	Ninguno

Participante	**José Luís Montoya Pareja**
Organización	Universidad Eafit
Rol	Asesor de proyecto
Es desarrollador	No
Es cliente	Sí
Es usuario	Sí
Comentarios	Asesor

FRQ-0001	**Simular un Procesador Hipotético**
Versión	1.4 (01/08/2000)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja
Dependencias	• [FRQ-0004] Trabajar con números en binario y hexadecimal
Descripción	El sistema deberá *ejecutar instrucciones en ensamblador, haciendo todos los pasos que hace un procesador como obtención de datos en memoria y luego interpretándolos mostrando la ejecución de estas paso a paso donde se pueda ver el funcionamiento de un microprocesador.*
Importancia	vital
Urgencia	inmediatamente
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0002	**Editor de Programas**
Versión	1.2 (14/02/2004)
Autores	Vladimir Yepes Bedoya

Fuentes	José Luís Montoya Pareja
Dependencias	Ninguno
Descripción	El sistema deberá *permitir el ingreso de los programas desde una interfaz grafica donde se resalte la sintaxis para facilitar la lectura del código que allí se digite.*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0003	**Manipulación de la ejecución**
Versión	1.1 (14/10/2002)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *permitir que la simulación pueda ser iniciada, pausada, detenida en cualquier momento.*
Importancia	quedaría bien
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0004	**Trabajar con números en binario y hexadecimal**
Versión	1.1 (14/11/2001)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético
Descripción	El sistema deberá *trabajar con números en binario y hexadecimal*
Importancia	importante
Urgencia	inmediatamente
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0005	Control de Velocidad de simulación
Versión	1.0 (14/05/2007)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	• [FRQ-0003] Manipulación de la ejecución
Descripción	El sistema deberá *permitir que la velocidad de simulación pueda ser controlada.*
Importancia	quedaría bien
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0006	Dispositivos de Entrada y Salida
Versión	1.3 (14/02/2004)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético
Descripción	El sistema deberá *contar con dispositivos de entrada como un teclado y salida como un monitor para que el usuario pueda interactuar con el. Además con dispositivos adicionales como Reloj, Switches y PC Speaker para aumentar la funcionalidad del simulador*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0007	Salvar y recuperar programas grabados
Versión	1.1 (17/08/2002)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya

Dependencias	• [FRQ-0002] Editor de Programas
Descripción	El sistema deberá *permitir que los programas puedan ser grabados en disco para luego ser vueltos a usar.*
Importancia	importante
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0008	**Conversor de Bases**
Versión	1.1 (23/10/2003)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *permitir que un usuario pueda convertir bases para facilitar su entendimiento ya que el simulador trabajara con binario y hexadecimal*
Importancia	quedaría bien
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0009	**Estadísticas**
Versión	1.0 (09/10/2003)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético
Descripción	El sistema deberá *generar estadísticas de la simulación las cuales podrán ser mostradas durante y después de terminada esta.*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

FRQ-0010	**Configurar Simulador**

Versión	1.1 (14/11/2004)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético
Descripción	El sistema deberá *permitir que el usuario pueda configurar y personalizar el simuproc de acuerdo a sus gustos, como colores, apariencia, etc.*
Importancia	quedaría bien
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

Requisitos No Funcionales

NFR-0001	Uso de recursos
Versión	1.0 (11/06/2001)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético
Descripción	El sistema deberá *usar recursos de manera adecuada, ser una aplicación liviana.*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

NFR-0002	Validación de Datos
Versión	1.0 (02/03/2003)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *validar todos los datos que le sean entrados ya sea por su interfaz grafica o por archivos de programas y garantizar su continuo funcionamiento teniendo un manejo adecuado de excepciones.*
Importancia	vital
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

NFR-0003	Interfaz grafica
Versión	1.4 (02/02/2007)

Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *tener una interfaz de usuario agradable y de fácil manejo.*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

NFR-0004	**Mantenimiento y actualizaciones**
Versión	1.1 (11/02/2007)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *contar con una pagina Web donde los usuarios puedan obtener mas información de la herramienta y descargar las nuevas versiones o programas adicionales para el simulador.*
Importancia	importante
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	Pagina disponible en http://simuproc.tk/

NFR-0005	**Facilidad de instalación**
Versión	1.1 (04/02/2004)
Autores	Vladimir Yepes Bedoya
Fuentes	Vladimir Yepes Bedoya
Dependencias	Ninguno
Descripción	El sistema deberá *contar con un instalador y desinstalador automático para que el usuario no requiera seguir pasos complicados para usar el simulador.*
Importancia	quedaría bien
Urgencia	puede esperar

Estado	validado
Estabilidad	alta
Comentarios	Ninguno

NFR-0006	**Desempeño**
Versión	1.1 (22/02/2003)
Autores	Vladimir Yepes Bedoya
Fuentes	José Luís Montoya Pareja Vladimir Yepes Bedoya
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético • [FRQ-0005] Control de Velocidad de simulación
Descripción	El sistema deberá *tener un buen desempeño a la hora de simular los programas.*
Importancia	quedaría bien
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	Ninguno

Casos de Uso

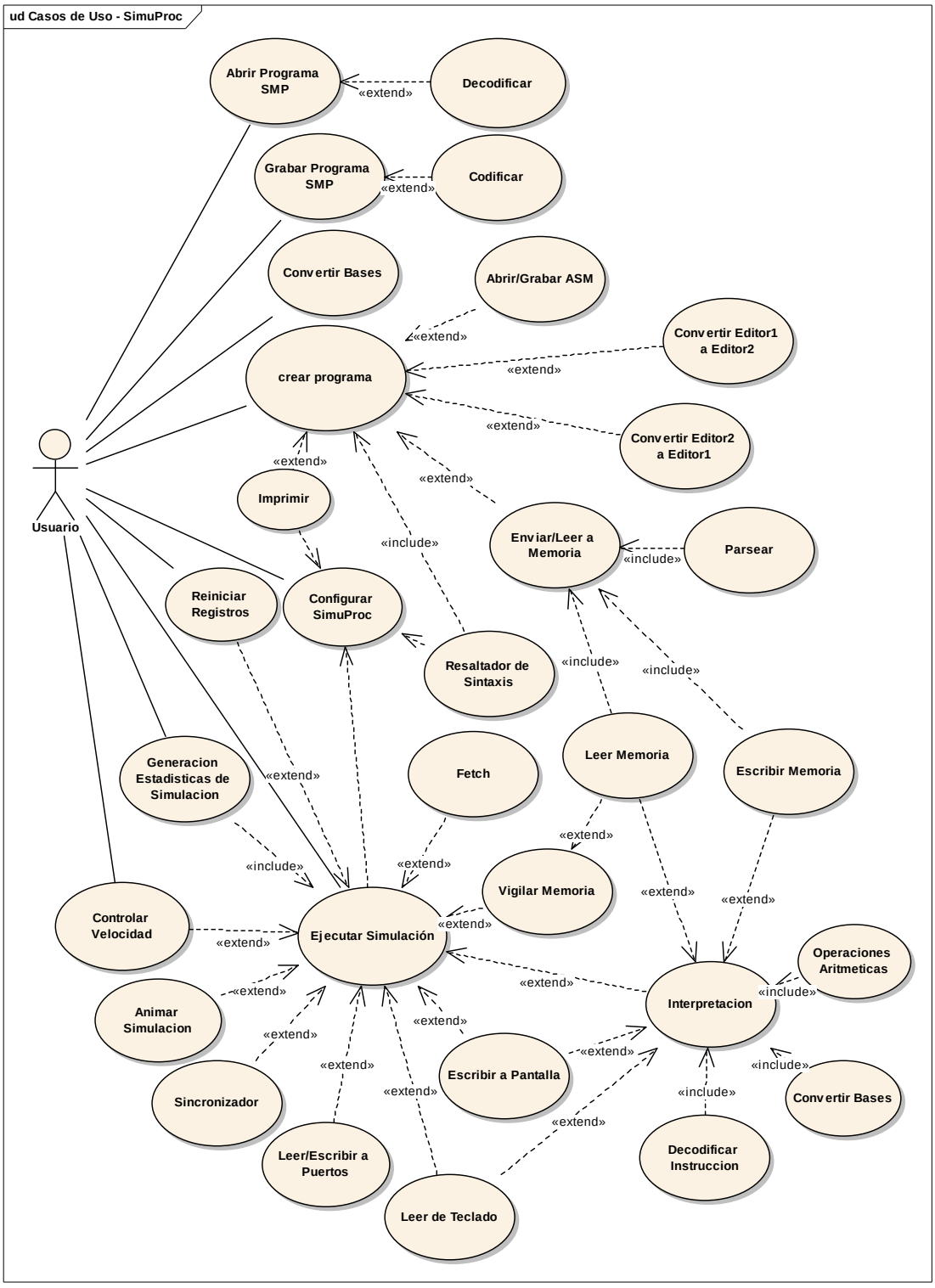

Figura 6. Casos de Uso SimuProc

Definición de Casos de Uso

UC-0001	**Ejecutar Simulación**	
Versión	1.0 (26/02/2001)	
Autores	Vladimir Yepes Bedoya	
Fuentes	José Luís Montoya Pareja	
Dependencias	• [FRQ-0001] Simular un Procesador Hipotético	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *presionan el botón Ejecutar*	
Precondición	Debe haber un programa en ensamblador en la memoria del simulador	
Secuencia normal	**Paso**	**Acción**
	-	-
Postcondición	La simulación termina exitosamente o se ha abortado.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Tiempo máximo**
	-	-
Frecuencia esperada	20 veces por hora(s)	
Importancia	vital	
Urgencia	inmediatamente	
Estado	validado	
Estabilidad	alta	
Comentarios	Ninguno	

UC-0002	**Crear Programa**	
Versión	1.1 (06/05/2003)	
Autores	Vladimir Yepes Bedoya	
Fuentes	José Luís Montoya Pareja	
Dependencias	• [FRQ-0002] Editor de Programas	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *se presiona el botón de editor de programas. Se carga la ventana de edición.*	
Precondición	Se dispone a ingresar o editar un programa.	
Secuencia	**Paso**	**Acción**

normal	-	-
Postcondición		
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Tiempo máximo**
	-	-
Frecuencia esperada	5 veces por hora(s)	
Importancia	importante	
Urgencia	hay presión	
Estado	validado	
Estabilidad	alta	
Comentarios	Ninguno	