UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA
UNAN – MANAGUA
RECINTO UNIVERSITARIO "RUBEN DARIO"
FACULTAD DE CIENCIAS E INGENIERIAS
DEPARTAMENTO DE COMPUTACIÓN

Tema: Lógica Programable.

Subtema: Diseño de una unidad Aritmética Lógica.

Seminario de Graduación.

Tutor:

Msc. Eman Hussein Yousif.

Integrantes:

Br. Alina Ramírez Reyes.
Br. Georgina Blanco Gómez.
Br. Elsa Gabriela Santana Sirias.

Fecha: **17/03/2011**

Índice

I. Introducción

El microprocesador, circuito electrónico que actúa como unidad central de proceso de un ordenador, proporcionando el control de las operaciones de cálculo.

Un microprocesador consta de varias secciones diferentes. La unidad aritmético-lógica (ALU, siglas en inglés) efectúa cálculos con números y toma decisiones lógicas; los registros son zonas de memoria especiales para almacenar información temporalmente; la unidad de control descodifica los programas; los buses transportan información digital a través del chip y de la computadora.

En todo sistema computacional, existen unidades funcionales encargadas de realizar las operaciones aritméticas, éstas son llamadas unidades de aritmética y lógica (ALU). Comúnmente trabajan a numeración de punto fijo o punto flotante, utilizando la representación binaria de los números de base decimal. El funcionamiento de una unidad aritmética convencional se basa en la utilización de sumadores binarios, que están compuestos por compuertas que cumplen funciones lógicas sencillas.

Las operaciones que se pueden desarrollar con una ALU son sumas, con la representación adecuada de los dígitos binarios, con la misma unidad sumadora se pueden implementar las restas. Para el desarrollo de multiplicaciones es necesaria la implementación de un algoritmo y de hardware adicional. Caso similar ocurre con la implementación de las divisiones. De esta manera se utilizan estas unidades en la solución de sistemas complejos que involucran un uso mayor que el usual de instrucciones aritméticas.

Se pretende con este Trabajo explicar la definición funcional de una ALU, a través de su simulación con un lenguaje de alto nivel llamado VHDL.

II. Justificación

Se diseñó una ALU (Unidad Aritmética Lógica) para mostrar por medio de ejemplos diversas operaciones Aritméticas, lógicas, explicando así su funcionamiento y estructura, para su diseño se utilizara un lenguaje de alto nivel llamado VHDL.

Este proyecto de diseño se elaboró con el propósito de finalizar los estudios universitarios de la carrera Ciencias de computación en la modalidad de seminario de graduación y obtener el titulo.

Lógica programable es el tema principal de este Proyecto; Este trabajo se elabora por primera vez en el departamento de computación, y puede servir de ejemplo, a los futuros estudiantes de la carrera.

III. Objetivos

3.1 Objetivo General.

- Diseñar una unidad aritmética lógica que realice diferentes operaciones utilizando el lenguaje VHDL.

3.2 Objetivos Específicos.

- Describir de manera física y teórica la ALU, cuales son sus componentes básicos y como funcionan.

- Realizar el diseño una unidad aritmética lógica utilizando operadores combinacionales capaz de efectuar diferentes operaciones.

- Mostrar a través de ejemplos las operaciones Aritméticas (Suma, resta, Multiplicación) y lógicas (And, Or, Xor, Comparación, Notb) que simula la ALU.

- Verificar el funcionamiento de la unidad aritmética lógica a través de los procedimientos empleados por medio del lenguaje de descripción VHDL.

Capitulo I

1.1 Lógica programable

La lógica programable, como el nombre implica, es una familia de componentes que contienen conjuntos de elementos lógicos (AND, OR, NOT, LATCH, FLIP-FLOP) que pueden configurarse en cualquier función lógica que el usuario desee y que el componente soporte. Hay varias clases de dispositivos lógicos programables: ASICs, FPGAs, PLAs, PROMs, PALs, GALs, y PLDs complejos.

Los dispositivos lógicos programables permiten reemplazar grandes diseños digitales que antes se implementaban con componentes discretos (como compuertas y flip-flops) por un solo integrado. Debido a la gran capacidad lógica que tienen los dispositivos modernos, sistemas completos pueden desarrollarse sobre un solo circuito integrado. Esto da lugar a las denominaciones System-on-a-Programmable Chip, o SoPC (sistema sobre un circuito integrado programable) y System-on-a-Reprogammable Chip (SoRC).

Los dispositivos actuales (CPLD y FPGAs) tienen una capacidad lógica de hasta millones de compuertas, incluyen interfaces programables para varios estándares de interfase eléctrica y tienen bloques de funciones especiales embebidos entre la lógica programable tales como memoria, multiplicadores o CPUs completas. Esta gran capacidad y variedad de los dispositivos los hace sumamente útiles a la hora de crear prototipos, desarrollo rápido de nuevos productos, para los productos que deben ser reconfigurables por naturaleza o productos que se producen en bajos volúmenes y para los cuales no es económicamente viable crear un integrado a medida.

Uno de los factores más importantes para el auge de las FPGA es el conjunto de herramientas disponibles. Entre los puntos a destacar de las herramientas de software disponibles en el mercado pueden nombrarse los siguientes:

- La practicidad de usar lenguajes de descripción de hardware (HDL o Hardware Description Language) para sintetizar circuitos.

- La integración de los flujos de diseño en un solo ambiente que hacen posible el prototipeado rápido con poco tiempo de uso de las herramientas.
- La integración de los simuladores con el ambiente de desarrollo que permiten una rápida verificación de los sistemas diseñados
- Las interfases cada vez más amigables, que incluyen interfases gráficas, sistemas de desarrollo integrados, la capacidad de desarrollar sistemas utilizando diversas metodologías (esquemático, maquina de estado, HDL, etc.).
- El soporte de diseño para FPGA por parte de las herramientas clásicas utilizadas para el diseño de circuitos electrónicos (Orcad, etc.).
- La disponibilidad de herramientas gratuitas o a bajo costo por parte de los vendedores de PLDs
- Las librerías de componentes integradas con las herramientas de software, que permiten hacer diseños de manera rápida.
- Guías y "Templates" automáticos para el diseño de circuitos básicos.

En resumen, la facilidad de uso de herramientas de desarrollo para PLDs ayuda a que una gran cantidad de usuarios puedan rápidamente desarrollar un sistema utilizando PLDs.

1.2 Evolución de los Primeros Dispositivos Lógicos Programables

Los primeros dispositivos lógicos programables fueron las memorias PROM. Para ello se utilizan como entradas las líneas de dirección a la memoria. De esta manera la memoria provee una tabla de decodificación completa de las entradas, y puede utilizarse para implementar cualquier función lógica combinacional.

Los primeros dispositivos diseñados específicamente para implementar funciones digitales programables fueron los PLA, introducidos al mercado en la década de 1970 por Philips.

La base teórica de estos dispositivos es que cualquier ecuación lógica puede reducirse a una suma de productos. El corazón electrónico de estos dispositivos consiste en un bloque que permite negar las entradas y dos niveles de compuertas: un nivel de compuertas AND y un nivel OR. Mediante fusibles pueden programarse las interconexiones entre las entradas

y la lógica. Así los PLA permiten implementar cualquier suma de productos de sus entradas.

Debido a las limitaciones en la velocidad de las conexiones programables y los problemas en el proceso de manufactura de las PLA, surgieron dispositivos con un solo nivel de lógica programable denominados PAL (recordar que los PLA tienen dos niveles programables: un nivel AND y un nivel OR). Los PAL se basan en el mismo principio que las PLA, pero tienen un nivel de lógica AND programable mientras el nivel de lógica OR es fijo. Esto limita la cantidad de funciones que pueden definirse con un dispositivo dado, así que los fabricantes de PAL los ofrecían en una variedad de configuraciones para adecuarlos a las necesidades del usuario. Muchos PAL además incorporaron registros sobre las salidas y realimentación para implementar circuitos secuenciales.

Los dispositivos denominados GAL son un poco más complejos que los PAL mencionados anteriormente. Incluyen un nivel de lógica AND a la entrada y luego un bloque lógico denominado macrocelda a la salida, en reemplazo del nivel OR. La denominación GAL fue utilizada en principio por Lattice, y luego licenciada a otros fabricantes.

1.2.1 CPLDs

Los CPLDs son dispositivos que combinan varios bloques lógicos similares a las PAL o GAL con una matriz de interconexión programable. Estos dispositivos se fabrican con distintas capacidades para adaptarlos a las necesidades del usuario.

1.2.2 FPGAs

La arquitectura de una FPGA consiste en arreglos de bloques lógicos que se comunican entre si a través de canales de conexión verticales y horizontales La principal diferencia entre las FPGA y CPLDs es que en general los bloques lógicos de las FPGA no implementan la lógica usando compuertas sino con generadores de funciones.

El FPGA contiene cinco elementos principales:

- Bloque de entrada-salida (IOB o Input-Output Block): estos bloques proveen la interfase entre las patitas o "pines" del integrado y la lógica interna.

- Bloque lógico configurable (CLB o Configurable Logic Block): Estos son los bloques básicos que se utilizarán en la implementación de un circuito digital.
- Bloque de distribución y compensación de reloj (DLL o Delay Locked Loop): Estos bloques controlan los dominios de reloj dentro del integrado y compensan por retardos que pueda haber entre el reloj externo y el interno.
- Bloque de memoria (BLOCK RAM): Estos bloques son memorias dedicadas integradas dentro de la lógica programable.
- Estructura de interconexión: Es una estructura versátil y multi-nivel de interconexión entre los otros componentes de la FPGA.

1.2.2.1 Bloque Lógico Programable

Todas las FPGA tienen algún tipo de bloque lógico programable. Este es el corazón de la FPGA, y permite implementar las diferentes funciones lógicas.

Cada CLB esta compuesto por dos bloques iguales denominados "slices". Cada "slice" contiene dos generadores de funciones y un multiplexor MUXF5. El multiplexor combina los resultados de los generadores de funciones dentro de cada "slice" del CLB.

Las dos "slices" están unidas por un multiplexor MUXF6, que puede seleccionar la salida de una u otra "slice" hacia la salida del CLB. Esto permite implementar cualquier función de 6 entradas, un multiplexor de 8:1 o determinadas funciones lógicas de hasta 19 entradas. Además de poder implementarse lógica combinacional, cada "slice" contiene recursos para implementar circuitos secuenciales y operaciones aritméticas eficientes.

1.2.2.2 Bloque de Control de Reloj

El sistema de control del reloj consiste en bloques de control integrados a la red de distribución de reloj. La red de distribución de reloj en las FPGA asegura retardos parejos a todos los bloques lógicos de la FPGA. Cada fabricante utiliza una arquitectura diferente para el control y distribución de reloj.

La familia Spartan IIe de Xilinx tiene bloques específicos para control de reloj denominados DLL (Delay Locked Loop). Estos bloques sincronizan el reloj interno al reloj externo del sistema, controlan el desplazamiento de fase entre los relojes, sincronizan los diferentes dominios de reloj y aseguran un retardo de distribución del reloj pareja para la lógica interna de la FPGA.

1.2.2.3 Memoria

La memoria es un componente muy utilizado en diseños digitales. Varias familias de FPGA contienen bloques de memoria embebida integrados con la lógica programable. En general estos bloques básicos de memoria pueden utilizarse en diferentes configuraciones para generar RAMs y ROMs de diferentes tamaños. Además de memorias embebidas, las FPGAs basadas en memoria SRAM pueden usar las tablas LUT de los bloques lógicos como memoria.

Estos bloques de memoria pueden usarse como memorias de puerto dual, puerto simple, RAMs o ROMs. Para entender la versatilidad de estos bloques de memoria; Las interfaces de dirección y datos (ADDRA, ADDRB, DIA, DOB, DOA) se pueden configurar para diferentes tamaños de memoria.

Además de poder configurar cada bloque, varios bloques pueden conectarse utilizando lógica externa para implementar memorias de otros tamaños y colas FIFO o FILO.

1.2.2.4 Bloque de Procesamiento de Señal

Varias FPGA contienen bloques específicos que optimizan en hardware ciertas funciones especiales. Las FPGA de la familia Stratix de Altera, por ejemplo, contienen uno o más módulos de procesamiento de señal entre los bloques de lógica programable de propósito general. Estos bloques permiten desarrollar ciertas funciones específicas típicas de las aplicaciones de procesamiento de señal de manera muy eficiente. Pueden configurarse de varias maneras diferentes según las necesidades del usuario. Este bloque contiene lógica para implementar operaciones de multiplicación-acumulación que requerirían de muchos recursos y ciclos de reloj si se implementaran utilizando lógica de propósito general. Al igual que los otros bloques, los bloques específicos pueden

interconectarse a los demás bloques utilizando la matriz de interconexión programable de la FPGA.

1.2.2.5 Matriz de Interconexión

Para poder implementar circuitos lógicos, los elementos lógicos mencionados en las secciones anteriores no solo deben configurarse adecuadamente sino que también deben conectarse entre sí. La estructura de interconexión interna de un PLD consiste en un conjunto de alambres o trazas que pueden conectarse mediante elementos de conexión programables. Las herramientas de "localización e interconexión" (place and route) son las encargadas de decidir en que elementos lógico se implementará la lógica diseñada por el usuario y como deben programarse las interconexiones para que el diseño funcione según las especificaciones de tiempo y retardos que se han definido.

Los FPGA tienen dos niveles de interconexión. Por un lado tienen una interconexión de propósito general a través de la matriz de interconexionado general o GRM por sus siglas en inglés. Por otro lado contienen recursos de interconexión local. Además de los ya mencionados, las FPGA Spartan IIe contienen recursos de interconexión dedicados a señales de tres estados, de entrada-salida y recursos de interconexión global para la distribución de reloj y señales específicas.

Los recursos de interconexión local, permiten hacer las conexiones entre los elementos internos de un bloque lógico o CLB, como las tablas de búsqueda (LUT), los flip-flop y las realimentaciones dentro del CLB. Además, el interconexionado a este nivel provee conexiones a la matriz de interconexionado general y a los CLB adyacentes. Las conexiones a los CLB adyacentes permiten optimizar los diseños al evitar los retardos y la utilización de recursos de la matriz general de interconexionado.

1.3 Granularidad de los Dispositivos Lógicos Programables (PLDs)

La granularidad de un dispositivo lógico programable está dada por la funcionalidad básica que provee cada bloque de configuración lógica, o la relación entre las celdas lógicas y los recursos de interconexión.

Algunos parámetros para definir la granularidad de un dispositivo programable son:

1. Número de funciones que puede implementar cada celda lógica.
2. Número de compuertas NAND de dos entradas equivalente por cada celda lógica.
3. Número total de transistores por celda lógica.
4. Área total normalizada de lógica configurable (relación de área lógica/área de interconexión).
5. Número de entradas y salidas por celda lógica.

Algunos dispositivos tienen arquitecturas con celdas lógicas que pueden implementar funciones lógicas complejas, de varias entradas/salidas. A estas se las denomina de granularidad gruesa. Otras arquitecturas están formadas por celdas básicas que solo permiten implementar lógica muy simple. A estas se las denomina de granularidad fina.

La granularidad de un dispositivo lógico influirá en la manera que se implemente una función lógica dada, en su frecuencia máxima de funcionamiento y en la utilización que se hace del dispositivo.

Un bloque lógico grande puede implementar lógica más compleja y por lo tanto se necesitan menos bloques para una función dada. Por otro lado cada bloque consume más área que puede desaprovecharse.

Una arquitectura de granularidad fina será típicamente más densa y tendrá menor retardo de interconexión entre celda y celda. Sin embargo, para una función dada, se deberán conectar un mayor número de celdas. En general la mejor granularidad dependerá de la aplicación y las restricciones que hay sobre el diseño. Para una FPGA basada en tablas de búsqueda (Look Up Table o LUT) la mejor relación entrada salida del bloque lógico es 4 a 1. Esto es lo que utilizan los dos fabricantes principales de FPGAs basados en tecnología SRAM.

Algunas características de cada tipo de arquitectura son las siguientes:

1.3.1 Arquitecturas con granularidad fina

- Celdas implementan funciones lógicas parciales de n entradas
- Relación de entradas a registros por celda muy baja (2/4 a 1)
- Ejemplo: ProAsicPlus de Actel

1.3.2 Arquitecturas con granularidad media

- Celdas implementan funciones completas de n entradas
- Relación de entradas a registros por celda de 6/8 a 1
- Buenas para implementar máquinas de estado y lógica secuencial compleja
- Ejemplo: Xilinx Spartan 3:

1.3.3 Arquitecturas con granularidad gruesa

- Celdas estilo PLD de arreglos de términos
- Relación de entradas a registros de 32 a 1
- Buenas para lógica combinacional como decodificación de direcciones, funciones aritméticas,
- Pobres para arquitecturas con colas (pipelines)
- Ejemplo: Altera MAX 3000

1.3.4 Arquitecturas mixtas

- Combinan celdas con granularidad fina y gruesa
- Muchas veces tienen recursos de interconexión dedicados para cada tipo de celda

1.4 Tecnología de Configuración de los PLDs

La tecnología .utilizada para configurar los elementos de un PLD, ya sean los bloques lógicos o las interconexiones, dependen del fabricante y el dispositivo. Entre los dispositivos reprogramables, se destacan las tecnologías EEPROM, SRAM y Flash. Los dispositivos que solo pueden configurarse una vez en general utilizan tecnologías de antifusible. Cada tecnología tiene sus ventajas y desventajas respecto a las otras.

En la siguiente tabla se resumen algunas de estas características.

Tecnología	Ventajas	Desventajas
EEPROM	○ No volátil ○ Puede probarse en un 100% ○ Software simple ○ Tecnología base muy conocida	○ Requiere tensiones altas para configurar y borrar ○ Requiere programador ○ Mayor consumo de potencia ○ Tolerancia media a la radiación ○ Requiere mucha área para implementarse (integrados grandes)
SRAM	○ Reconfigurable. Puede cambiarse el diseño solo cambiando la PROM de configuración en la placa final ○ No requiere programador ○ Proceso de manufactura igual que la lógica ○ Puede probarse en un 100% ○ Tecnología integrada con la lógica (manufactura fácil, menor costo)	○ Volátil. Se debe reprogramar después de cortar la alimentación ○ Requiere memoria (PROM) externa ○ Tiempo de encendido "lento" (debe configurarse) ○ Mayor utilización de área en el integrado ○ No muy tolerante a la radiación
Flash	○ No volátil ○ Tiempo de encendido rápido	○ Requiere programador ○ Requiere tensiones altas ○ La misma velocidad que

	○ Menor potencia que SRAM ○ Menor área que SRAM y antifusible ○ Programable sobre la placa (no tan fácil)	SRAM ○ Menor cantidad de reprogramaciones
Antifusible	○ No volátil ○ Mayor densidad ○ Mucha tolerancia a la radiación (aplicaciones espaciales) ○ Tiempo de encendido muy rápido ○ Baja potencia	○ No pueden reconfigurarse ○ Requiere programador ○ No puede evaluarse. Si se comete un error, se pierde el integrado

A continuación se dan algunos ejemplos de dispositivos que utilizan cada tecnología.

SRAM: Virtex II, Virtex4, Spartan2e, Spartan 3 de Xilinx. Stratix, Cyclone y Apex de Altera
Flash: Actel ProAsic, ProAsicPlus
Antifusible: Actel A54SX, Quicklogic pAsic
EEPROM: Altera MAX7000, Cypress Ultra37000
EEPROM + SRAM: Lattice Semiconductor ispXPGA

1.5 Tendencias Actuales y Futuras

A continuación se nombran algunas tendencias que pueden observarse en la industria de de las lógicas programables, no solo de los dispositivos en sí sino de las herramientas y usos de estos dispositivos.

- Desarrollo de FPGAs con mayor capacidad, más velocidad, menos consumo de potencia y menor costo.
- Integración y mezcla entre FPGAs y circuitos integrados de propósito específico (ASICs). Componentes que pueden pasar directamente de un prototipo sobre una FPGA a un ASIC sin esfuerzo de diseño adicional.
- Enfoque sobre la reutilización de propiedad intelectual. Modelo de negocios en el que no solo se provee el hardware (PLDs) sino también propiedad intelectual por un costo.
- Cambios en las herramientas de software para tener mayor integración entre los diferentes usuarios y en los distintos pasos del proyecto. Esto incluye el manejo de grupos de trabajo, el manejo de la propiedad intelectual, la documentación de proyectos integrada a las herramientas de diseño, la transferencia de los diseños para FPGAs a ASICs, etc.
- Integración con herramientas y otros software de desarrollo de mayor nivel, como son Matlab y herramientas para el diseño integrado de software y hardware. (HW-SW co-design).
- Integración del manejo de configuración de FPGAs con las herramientas de diseño de plaquetas (PCBs), para poder configurar el interconexionado interno del SoPC de manera integral con el conexionado hacia los componentes externos.
- Instrumentos, tales como analizadores lógicos, que permiten observar la lógica interna de las FPGA en tiempo real al mismo tiempo que se observan otras señales, externas a la FPGA.
- Sistemas con aplicaciones que utilicen la reconfiguración o reprogramación de las FPGAs directamente en el producto final según la necesidad del momento.
- Integración de bloques lógicos programables alrededor de lógica fija en ASICs o integrados estándares.

En los próximos años probablemente se verá una mayor incorporación de las lógicas programables en los diseños, junto a una mayor variedad, diversidad, nuevas ideas y diseños basados en lógicas y plataformas programables.

Capitulo II

2.1 Microprocesadores

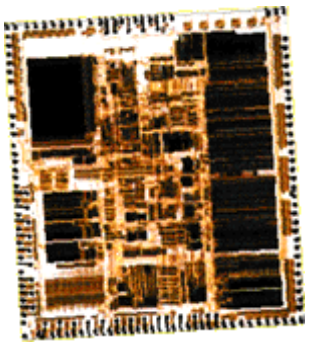
2.1.1 Historia de los primeros microprocesadores

Han pasado más de 25 años desde que Intel diseñara el primer microprocesador, siendo la compañía pionera en el campo de la fabricación de estos productos, y que actualmente cuenta con más del 90 por ciento del mercado. Un tiempo en el que todo ha cambiado enormemente, y en el que desde aquel 4004 hasta el actual Pentium II se ha visto pasar varias generaciones de máquinas que han entretenido y ayudado en el trabajo diario a muchos estudiantes.

Según el Dr. Albert Yu, vicepresidente de Intel y responsable del desarrollo de los procesadores desde el año 1984, para el año 2011 utilizaremos procesadores cuyo reloj irá a una velocidad de 10 GHz (10.000 MHz), contendrán mil millones de transistores y será capaz de procesar cerca de 100 mil millones de instrucciones por segundo. Un futuro prometedor, que permitirá realizar tareas nunca antes pensadas.

2.1.1.1 Los inicios

Sin embargo, para que esto llegue, la historia de los procesadores ha pasado por diferentes situaciones, siguiendo la lógica evolución de este mundo. Desde aquel primer procesador 4004 del año 1971 hasta el actual Pentium II ha llovido mucho en el campo de los procesadores. Tanto, que no estamos seguros si las cifras que se barajan en Intel se pueden, incluso, quedar cortas. Aquel primer procesador 4004, presentado en el mercado el día 15 de noviembre de 1971, poseía unas características únicas para su tiempo. Para empezar, la velocidad de reloj sobrepasaba por poco los 100 KHz, disponía de un ancho de bus de 4 bits y podía manejar un máximo de 640 bytes de memoria. Realmente una auténtica joya que para entonces podía realizar gran cantidad de tareas, pero por desgracia no tiene punto de comparación con los actuales micros.



Entre sus aplicaciones, se puede destacar su presencia en la calculadora Busicom, así como dotar de los primeros tintes de inteligencia a objetos inanimados.

Poco tiempo después, sin embargo, el 1 de abril de 1972, Intel anunciaba una versión mejorada de su procesador. Se trataba del 8008, que contaba como principal novedad con un bus de 8 bits, y la memoria direccionable se ampliaba a los 16 Kb. Además, llegaba a la cifra de los 3500 transistores, casi el doble que su predecesor, y se le puede considerar como el antecedente del procesador que serviría de corazón al primer ordenador personal. Justo dos años después, Intel anunciaba ese tan esperado primer ordenador personal, de nombre Altair, cuyo nombre proviene de un destino de la nave Enterprise en uno de los capítulos de la popular serie de televisión Star Trek la semana en la que se creó el ordenador. Este ordenador tenía un costo de entorno a los 400 dólares de la época, y el procesador suponía multiplicar por 10 el rendimiento del anterior, gracias a sus 2 MHz de velocidad (por primera vez se utiliza esta medida), con una memoria de 64 Kb. En unos meses, logró vender decenas de miles de unidades, en lo que suponía la aparición del primer ordenador que la gente podía comprar, y no ya simplemente utilizar.

2.1.1.2 Introducción de IBM

Sin embargo, el ordenador personal no pasó a ser tal hasta la aparición de IBM, el gigante azul, en el mercado. Algo que sucedió en dos ocasiones en los meses de junio de 1978 y de 1979. Fechas en las que respectivamente, hacían su aparición los microprocesadores 8086 y 8088, que pasaron a formar el denominado IBM PC, que vendió millones de unidades de ordenadores de sobremesa a lo largo y ancho del mundo. El éxito fue tal, que Intel fue nombrada por la revista "Fortune" como uno de los mejores negocios de los años setenta. De los dos procesadores, el más potente era el 8086, con un bus de 16 bits, velocidades de reloj de 5 MHz, 8 MHz y 10 MHz, 29000 transistores usando la tecnología de 3 micras y hasta un máximo de 1 Mega de memoria direccionable. El rendimiento se había vuelto a multiplicar por 10 con respecto a su antecesor, lo que suponía un auténtico avance en lo que al mundo de la informática se refiere. En cuanto al

procesador 8088, era exactamente igual a éste, salvo la diferencia de que poseía un bus de 8 bits en lugar de uno de 16, siendo más barato y obteniendo mejor respaldo en el mercado.

En el año 1982, concretamente el 1 de febrero, Intel daba un nuevo vuelco a la industria con la aparición de los primeros 80286. Como principal novedad, cabe destacar el hecho de que por fin se podía utilizar la denominada memoria virtual, que en el caso del 286 podía llegar hasta 1 Giga. También hay que contar con el hecho de que el tiempo pasado había permitido a los ingenieros de Intel investigar más a fondo en este campo, movidos sin duda por el gran éxito de ventas de los anteriores micros. Ello se tradujo en un bus de 16 bits, 134000 transistores usando una tecnología de 1.5 micras, un máximo de memoria direccionable de 16 Megas y unas velocidades de reloj de 8 MHz, 10 MHz y 12 MHz. En términos de rendimiento, se puede decir que se había multiplicado entre tres y seis veces la capacidad del 8086, y suponía el primer ordenador que no fabricaba IBM en exclusiva, sino que otras muchas compañías, alentadas por los éxitos del pasado, se decidieron a crear sus propias máquinas. Como dato curioso, basta mencionar el hecho de que en torno a los seis años que se le concede de vida útil, hay una estimación que apunta a que se colocaron en alrededor a los 15 millones de ordenadores en todo el mundo.

2.1.1.3 Intel

El año de 1985 es clave en la historia de los procesadores. El 17 de octubre Intel anunciaba la aparición del procesador 80386DX, el primero en poseer una arquitectura de 32 bits, lo que suponía una velocidad a la hora de procesar las instrucciones realmente importante con respecto a su antecesor. Dicho procesador contenía en su interior en torno a los 275000 transistores, más de 100 veces los que tenía el primer 4004 después de tan sólo 14 años. El reloj llegaba ya hasta un máximo de 33 MHz, y era capaz de direccionar 4 Gigas de memoria, tamaño que todavía no se ha superado por otro procesador de Intel dedicado al mercado doméstico. En 1988, Intel desarrollaba un poco tarde un sistema sencillo de actualizar los antiguos 286 gracias a la aparición del 80386SX, que sacrificaba el bus de datos para dejarlo en uno de 16 bits, pero a menor coste. Estos procesadores irrumpieron con la explosión del entorno gráfico Windows, desarrollado por Microsoft unos años antes, pero que no había tenido la suficiente aceptación por parte de los usuarios.

Y si esto parecía la revolución, no se tuvo que esperar mucho para que el 10 de abril de 1989 apareciera el Intel 80486DX, de nuevo con tecnología de 32 bits y como novedades principales, la incorporación del caché de nivel 1 (L1) en el propio chip, lo que aceleraba enormemente la transferencia de datos de este caché al procesador, así como la aparición del co-procesador matemático, también integrado en el procesador, dejando por tanto de ser una opción como lo era en los anteriores 80386. Dos cambios que unido al hecho de que por primera vez se sobrepasaban el millón de transistores usando la tecnología de una micra (aunque en la versión de este procesador que iba a 50 MHz se usó ya la tecnología .8 micras), hacía posible la aparición de programas de calidad sorprendente, entre los que los juegos ocupan un lugar destacado. Se había pasado de unos ordenadores en los que prácticamente cualquier tarea compleja requería del intérprete de comandos de MS-DOS para poder ser realizada, a otros en los que con mover el cursor y pinchar en la opción deseada simplificaba en buena medida las tareas más comunes

2.1.1.4 El Pentium

Sin embargo, Intel no se quedó contemplando la gran obra que había creado, y rápidamente anunció que en breve estaría en la calle una nueva gama de procesadores que multiplicaría de forma general por cinco los rendimientos medios de los 80486. Se trataba de los Pentium, conocidos por P5 en el mundo de la informática mientras se estaban desarrollando, y de los que la prensa de medio mundo auguraba un gran futuro, tal y como así ha sido. Estos procesadores pasarán a la historia por ser los primeros a los que Intel no los bautizó con un número, y sí con una palabra. Esto era debido a que otras compañías dedicadas a la producción de procesadores estaban utilizando los mismos nombres puesto que no se podía registrar una cadena de ellos como marca, y por lo tanto, eran de dominio público. De modo que a Intel no le quedó más remedio que ponerle una palabra a su familia de procesadores, que además, con el paso del tiempo, se popularizó en los Estados Unidos de tal forma, que era identificada con velocidad y potencia en numerosos cómics y programas de televisión. Estos procesadores que partían de una velocidad inicial de 60 MHz, han llegado hasta los 200 MHz, algo que nadie había sido capaz de augurar unos años antes.

Con una arquitectura real de 32 bits, se usaba de nuevo la tecnología de .8 micras, con lo que se lograba realizar más unidades en menos espacio. Los resultados no se hicieron esperar, y las compañías empezaron aunque de forma tímida a lanzar programas y juegos exclusivamente para el Pentium, hasta el punto que en este momento quien no posea un procesador de este tipo, está seriamente atrasado y no puede trabajar con garantías con los programas que actualmente hay en el mercado. Algo que ha venido a demostrar la aparición del nuevo sistema operativo de Microsoft Windows 95, que aunque funciona en equipos dotados de un procesador 486, lo hace sin sacar el máximo partido de sus funciones.

2.1.1.5 Pentium Pro y Pentium II

La aparición, el 27 de marzo de 1995, del procesador Pentium Pro supuso para los servidores de red y las estaciones de trabajo un aire nuevo, tal y como ocurriera con el Pentium en el ámbito doméstico. La potencia de este nuevo procesador no tenía comparación hasta entonces, gracias a la arquitectura de 64 bits y el empleo de una tecnología revolucionaria como es la de .32 micras, lo que permitía la inclusión de cinco millones y medio de transistores en su interior. El procesador contaba con un segundo chip en el mismo encapsulado, que se encargaba de mejorar la velocidad de la memoria caché, lo que resultaba en un incremento del rendimiento sustancioso. Las frecuencias de reloj se mantenían como límite por arriba en 200 MHz, partiendo de un mínimo de 150 MHz.

2.2 El futuro de los microprocesadores

La evolución que están sufriendo los procesadores es algo que no parece escapar a la atención de millones de personas, cuyo trabajo depende de hasta dónde sean capaces de llegar los ingenieros de Intel a la hora de desarrollar nuevos chips. El último paso conocido ha sido la implementación de la nueva arquitectura de 0.25 micras, que viene a sustituir de forma rotunda la empleada hasta el momento, de 0.35 micras en los últimos modelos de procesador. Esto va a significar varias cosas en un futuro no muy lejano. Para empezar, la velocidad se incrementará una media del 33 por ciento con respecto a la generación de anterior. Es decir, el mismo procesador usando esta nueva tecnología puede ir un 33 por ciento más rápido que el anterior. Para hacerse una idea del tamaño de esta tecnología, se

dice que el valor de 0.25 micras es unas 400 veces más pequeño que un pelo de cualquier persona. Y este tamaño es el que tienen los transistores que componen el procesador. El transistor, permite el paso de la corriente eléctrica, de modo que en función de en qué transistores haya corriente, el ordenador realiza las cosas (esto es una simplificación de la realidad, pero se ajusta a ella más o menos). Dicha corriente eléctrica circula entre dos puntos, de modo que cuanto menor sea esta distancia, más cantidad de veces podrá pasar pues el tiempo de paso es menor. Aunque se habla de millonésimas de segundo, se tiene en cuenta que un procesador está trabajando continuamente, de modo que ese tiempo que parece insignificante cuando es sumado a lo largo de las miles de millones de instrucciones que realizar, puede dar una cantidad de tiempo bastante importante. De modo que la tecnología que se utilice puede dar resultados totalmente distintos incluso utilizando el mismo procesador. Por el momento, en un futuro cercano además de contar con la arquitectura de 0.25 micras, se podría disfrutar de una de 0.07 para el año 2011, lo que supondrá la introducción en el procesador de mil millones de transistores y alcanzando una velocidad de reloj cercana a los 10000 MHz, es decir, 10 GHz.

2.3 El microprocesador

El microprocesador es el chip más importante de cualquier placa madre. Sin él, un ordenador no podría funcionar. A menudo a este componente se le denomina CPU (Central Processing Unit, Unidad de procesamiento central), que describe a la perfección su papel dentro del sistema. El procesador es realmente el elemento central del proceso de tratamiento de datos.

La CPU gestiona cada paso en el proceso de los datos. Actúa como el conductor y el supervisor de los componentes de hardware del sistema. Asimismo, está unida, directa o indirectamente, con todos los demás componentes de la placa principal. Por lo tanto, muchos grupos de componentes reciben órdenes y son activados de forma directa por la CPU.

El procesador está equipado con buses de direcciones, de datos y de control, que le permiten llevar a cabo sus tareas. Estos sistemas de buses varían dependiendo de la categoría del procesador.

También durante el desarrollo de los ordenadores personales han ido variando las unidades funcionales internas de los procesadores, evolucionando drásticamente. Se ha incorporado un número de transistores y circuitos integrados cada vez mayor, y dentro de un espacio cada vez más reducido, a fin de satisfacer las demandas cada vez más exigentes de mayores prestaciones por parte del software.

Por todo lo expuesto, se hacen lógicamente necesarios unos procesos de fabricación también complejos y especiales. Esta técnica permite construir elementos casi microscópicos (un micrómetro, o la millonésima parte de un metro). Esta técnica desarrollada por Intel se conoce como CHMOS-IV .Para apreciar la miniaturización en cuestión, se piensa que un solo pelo humano tiene una anchura que se extendería sobre 100 unidades de este tipo.

La configuración y capacidad de este procesador son los criterios fundamentales que determinan el rendimiento de todo el ordenador.

La unidad central de proceso (CPU), procesador o microprocesador, es el verdadero cerebro del ordenador. Su misión consiste en controlar y coordinar todas las operaciones del sistema. Para ello extrae, una a una, las instrucciones del programa que está en la memoria central del ordenador (memoria RAM), las analiza y emite las órdenes necesarias para su completa realización.

Para entender cómo funciona un microprocesador, se debe tener en primer lugar una clara idea acerca de las partes que lo componen. De otro modo, será prácticamente imposible hacerse una idea sobre su funcionamiento.

Se profundizará en las diferentes partes que componen un microprocesador. El esta compuesto por las dos siguientes unidades:

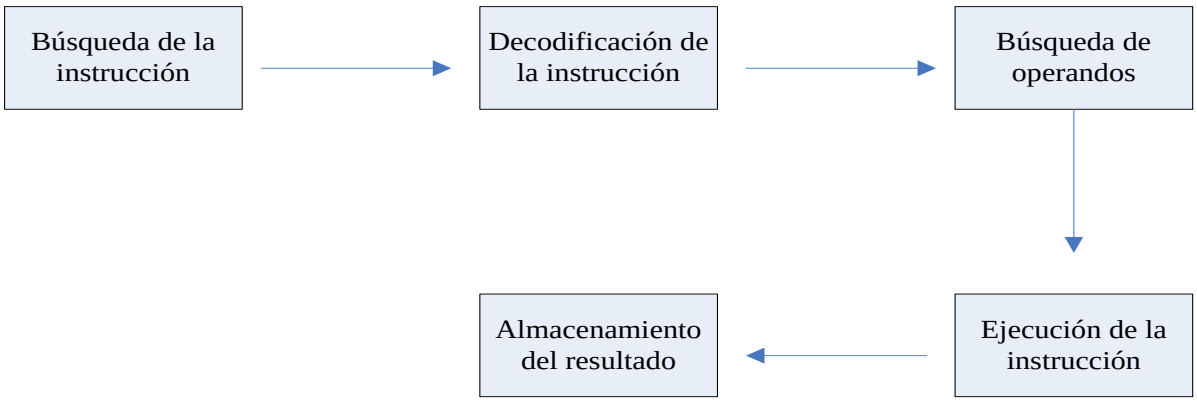
- ❖ Unidad de Control.
- ❖ Unidad aritmético-lógica (ALU).

Unidad de Control

Es el centro nervioso del ordenador, ya que desde ella se controlan y gobiernan todas las operaciones. Cómo funciones básica tiene:

- tomar las instrucciones de memoria
- decodificar o interpretar las instrucciones
- ejecutar las instrucciones (tratar las situaciones de tipo interno (inherentes a la propia CPU) y de tipo externo (inherentes a los periféricos).

Esquema de la búsqueda de las instrucciones



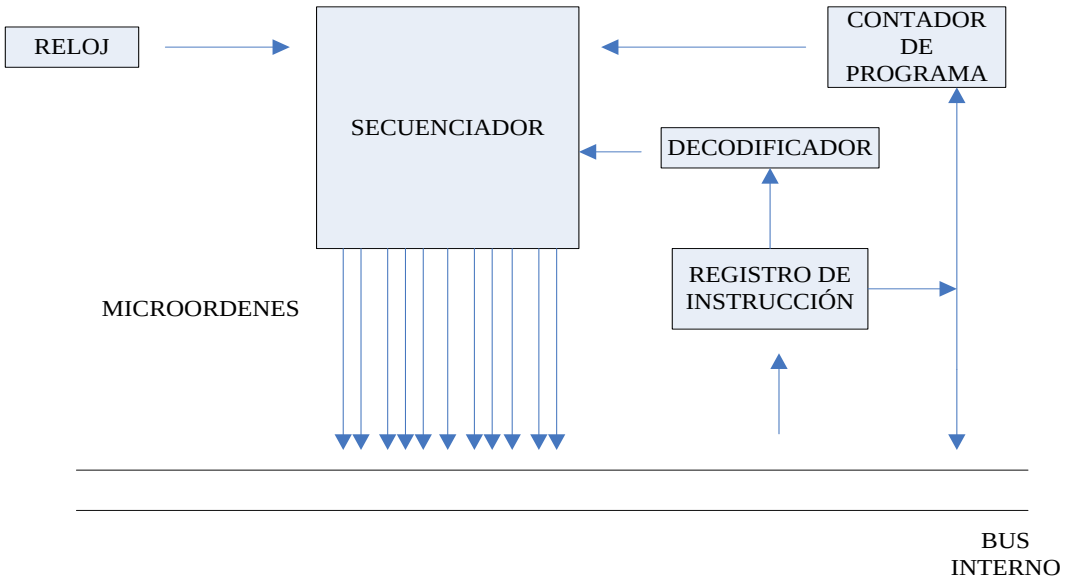
Esquema que muestra como trabaja la unidad de control al tratar los datos.

Tras interpretar la unidad de control el código de la instrucción en curso, localiza los operandos de entrada, selecciona un operador del camino de datos par procesarlos y finalmente almacena el resultado.

Para realizar su función, la unidad de control consta de los siguientes elementos:

- Contador de programa
- Registro de instrucciones
- Decodificador
- Reloj
- Secuenciador

Esquema de los Elementos de la Unidad de control



Explicación del esquema:

Contador de programa: Contiene permanentemente la dirección de memoria de la siguiente instrucción a ejecutar. Al iniciar la ejecución de un programa toma la dirección de su primera instrucción. Incrementa su valor en uno, de forma automática, cada vez que se concluye una instrucción, salvo si la instrucción que se está ejecutando es de salto o de ruptura de secuencia, en cuyo caso el contador de programa tomará la dirección de la instrucción que se tenga que ejecutar a continuación; esta dirección está en la propia instrucción en curso.

Registro de instrucción: Contiene la instrucción que se está ejecutando en cada momento. Esta instrucción llevará consigo el código de operación (un código que indica qué tipo de operación se va a realizar, por ejemplo una suma) y en su caso los operandos (datos sobre los que actúa la instrucción, por ejemplo los números a sumar) o las direcciones de memoria de estos operandos.

Decodificador: Se encarga de extraer el código de operación de la instrucción en curso (que está en el registro de instrucción), lo analiza y emite las señales necesarias al resto de elementos para su ejecución a través del secuenciador.

Reloj: Proporciona una sucesión de impulsos eléctricos o ciclos a intervalos constantes (frecuencia constante), que marcan los instantes en que han de comenzar los distintos pasos de que consta cada instrucción.

Secuenciador: En este dispositivo se generan órdenes muy elementales (microórdenes) que, sincronizadas por los impulsos de reloj, hacen que se vaya ejecutando poco a poco la instrucción que está cargada en el registro de instrucción.

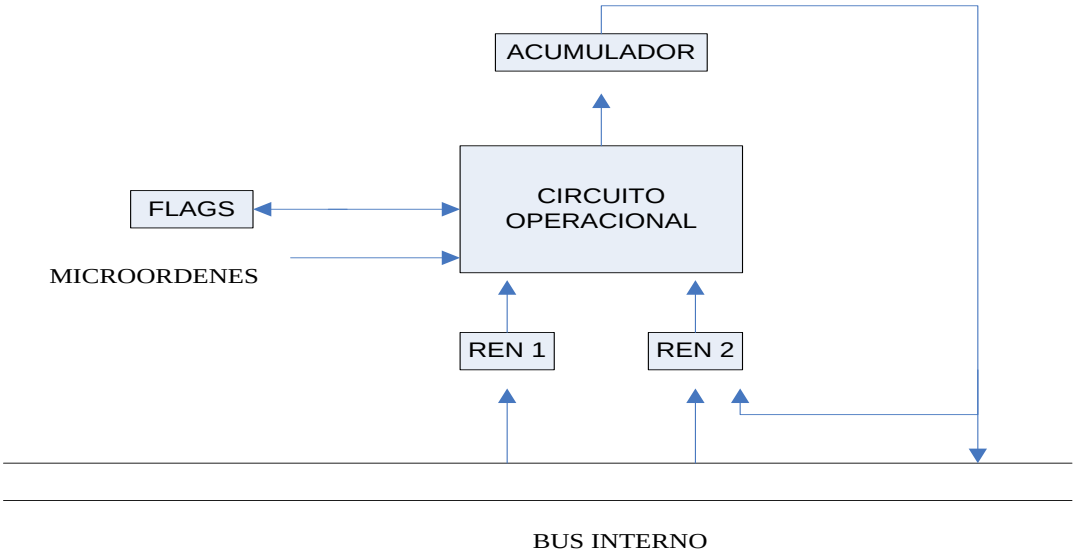
Unidad aritmético-lógica (ALU)

Esta unidad se encarga de realizar las operaciones elementales de tipo aritmético (sumas, restas, productos, divisiones) y de tipo lógico (comparaciones). A través de un bus interno se comunica con la unidad de control la cual le envía los datos y le indica la operación a realizar.

La ALU está formada a su vez por los siguientes elementos:

- Circuito operacional
- Registros de entrada (REN)
- Registro acumulador
- Registro de estado (flags)

Esquema de los elementos que forman la unidad aritmética Lógica.



Explicación del esquema:

Circuito operacional: Contiene los circuitos necesarios para la realización de las operaciones con los datos procedentes de los registros de entrada (REN). Este circuito tiene unas entradas de órdenes para seleccionar la clase de operación que debe realizar en cada momento (suma, resta, etc.).

Registros de entrada (REN): En ellos se almacenan los datos u operandos que intervienen en una instrucción antes de la realización de la operación por parte del circuito operacional. También se emplean para el almacenamiento de resultados intermedios o finales de las operaciones respectivas.

Registro acumulador: Almacena los resultados de las operaciones llevadas a cabo por el circuito operacional. Está conectado con los registros de entrada para realimentación en el caso de operaciones encadenadas. Asimismo tiene una conexión directa al bus de datos para el envío de los resultados a la memoria central o a la unidad de control.

Registro de estado (flags): Se trata de unos registros de memoria en los que se deja constancia algunas condiciones que se dieron en la última operación realizada y que habrán

de ser tenidas en cuenta en operaciones posteriores. Por ejemplo, en el caso de hacer una resta, tiene que quedar constancia si el resultado fue cero, positivo o negativo.

El registro de estado se encarga de contener los señalizadores, banderas o flags, que son bits que informan si se ha producido o no una condición especial la realizar la ALU una operación. Los señalizadores más usuales son:

Acarreo (C): Si se pone a 1 el bit correspondiente a este señalizador, indica que se ha producido acarreo al sumar los 2 bits de mas peso de los operandos. También se pone a 1 cuando hay llevada en dichos bits al proceder a restar operandos.

Cero (z): Este bit se pone a 1 si el resultado es 0.

Paridad (P): Si el procesador trabaja con paridad par, este bit se pone a 1 cuando el número de 1 del resultado es par.

Se conoce como set de instrucciones al conjunto de instrucciones que es capaz de entender y ejecutar un microprocesador.

En función del tipo de microprocesador, concretamente si es más avanzado o no, podrá entender y ejecutar más o menos instrucciones.

Las instrucciones se clasifican según su función en:

- Instrucciones de transferencia de datos
- Instrucciones de cálculo
- Instrucciones de transferencia del control del programa
- Instrucciones de control

Instrucciones de transferencia de datos: Estas instrucciones mueven datos (que se consideran elementos de entrada/salida) desde la memoria hacia los registros internos del microprocesador, y viceversa. También se usan para pasar datos de un registro a otro del microprocesador. Existen algunas instrucciones que permiten mover no sólo un dato, sino un conjunto de hasta 64 KBytes con una sola instrucción.

Instrucciones de cálculo: Son instrucciones destinadas a ejecutar ciertas operaciones aritméticas, como por ejemplo sumar, restar, multiplicar o dividir, o ciertas operaciones lógicas, como por ejemplo AND, OR, así como desplazamiento y rotación de bits.

Instrucciones de transferencia del control del programa: Permiten romper la secuencia lineal del programa y saltar a otro punto del mismo. Pueden equivaler a la instrucción ***goto*** que traen muchos lenguajes de programación.

Instrucciones de control: Son instrucciones especiales o de control que actúan sobre el propio microprocesador. Permiten acceder a diversas funciones, como por ejemplo activar o desactivar las interrupciones, pasar órdenes al coprocesador matemático, detener la actividad del microprocesador hasta que se produzca una interrupción, etc.

Prácticamente todas las instrucciones están formadas por dos elementos:

- código de operación que indica el tipo de operación se va a realizar
- operandos, que son los datos sobre los que actúa.

Por ejemplo, una instrucción que sume dos números está formado por:

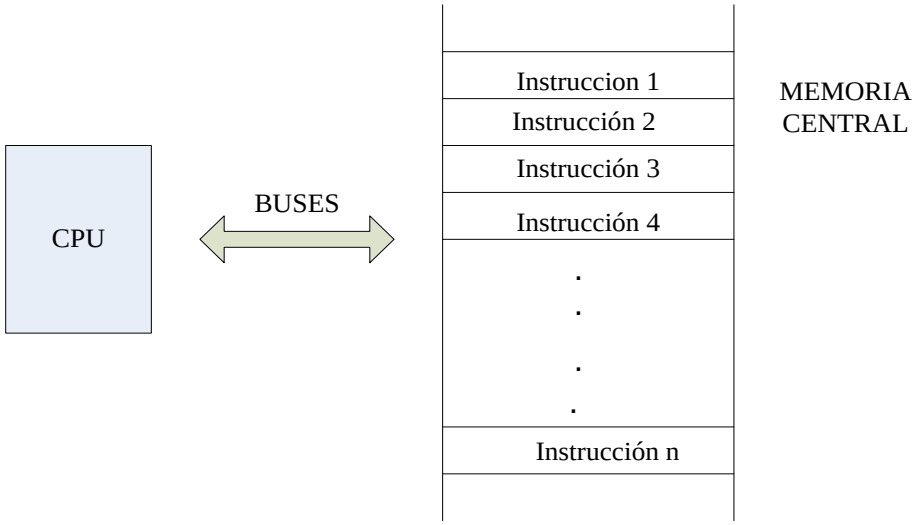
- código de operación que indique "sumar"
- primer número a sumar
- segundo número a sumar

Existen instrucciones que sólo tienen un operando o incluso que no tienen ninguno, estando formadas solamente por el código de operación.

2.4 Ejecución de las instrucciones

Para que un programa pueda ser ejecutado por un ordenador, ha de estar almacenado en la memoria central (memoria RAM). El microprocesador tomará una a una las instrucciones que lo componen e irá realizando las tareas correspondientes.

Esquema de las instrucciones



Se denomina ciclo de instrucción al conjunto de acciones que se llevan a cabo en la realización de una instrucción.

Se compone de dos fases:

- Fase de búsqueda
- Fase de ejecución

Fase de búsqueda. En esta fase se transfiere la instrucción que se va a ejecutar desde la memoria central a la unidad de control.

Fase de ejecución. Consiste en la realización de todas las acciones que conlleva la propia instrucción.

2.4.1 El camino de datos.

El camino de datos, también llamado Unidad de proceso, es la sección del computador encargada de efectuar las operaciones lógicas y aritméticas que implican las instrucciones que efectúan. Su misión es de manipular y transformar los datos procedentes de la Memoria o de los registros internos.

La labor del camino de datos es soportar el conjunto de operaciones que precisan las instrucciones del repertorio que es capaz de interpretar la unidad de control.

El camino de datos está formado por tres secciones diferentes:

1- Unidad Aritmética Lógica.

2- Registros.

3- Buses o líneas de interconexión.

Cabe mencionar que en el camino de los datos existen diferentes tipos de registros según la información que almacenan, pudiéndose distinguir los siguientes grupos:

1- Registros de la ALU.

2- Banco de registros generales.

3- Registros especiales.

Banco de registros generales:

En el camino de los datos existe un conjunto de registro de propósito general cuya función principal es la de guardar datos y direcciones que se usan mucho como el tiempo de acceso a la memoria principal externa es mayor que a dichos registros, su correcta utilización permite mejorar la velocidad de procesamiento de los programas si se almacena en ellos información de uso muy frecuente.

Registros especiales:

Para unificar la descripción de los registros se incluye en el camino de Datos algunos registros muy específicos como: contador de programa, apuntador de pila y el registro de direcciones de datos.

Una forma de clasificar los microprocesadores es en función de las instrucciones que son capaces de ejecutar. Podemos encontrar dos tipos: microprocesadores: con tecnología CISC y RISC.

CISC Complex Instructions Set Computer, Ordenador con un conjunto de instrucciones complejo.

RISC Reduced Instructions Set Computer, Ordenador con un conjunto de instrucciones reducido.

Anteriormente se definió el conjunto de instrucciones como el conjunto de instrucciones que es capaz de entender y ejecutar un microprocesador. Si ese microprocesador entiende y ejecuta muchas instrucciones (cientos de ellas), se trata entonces de un microprocesador CISC. En cambio, si el microprocesador entiende y ejecuta muy pocas instrucciones (decenas de ellas), se trata entonces de un microprocesador RISC.

En principio, parece que la tecnología CISC es mucho más ventajosa que la RISC. Pero no es así: un micro CISC tarda mucho tiempo en ejecutar cada una de esas instrucciones. En cambio un micro RISC, como sólo entiende unas cuantas, su diseño interno le permite ejecutarlas en muy poco tiempo, a una gran velocidad, mucho más rápido que un microprocesador CISC.

Cuando se desee que un microprocesador RISC ejecute cierta instrucción que no entiende, ésta se descompondrá en varias instrucciones de las sencillas que sí entiende. Aún así, descomponiendo una instrucción compleja en varias sencillas, es capaz de operar mucho más rápido que el microprocesador CISC, el cual no tiene que descomponer esa instrucción porque la entiende directamente.

Prácticamente, todos los microprocesadores que se utilizan en la fabricación de ordenadores personales (microprocesadores fabricados por Intel) son de tecnología CISC. Intel, poco a poco, va abandonando la tecnología CISC y la sustituye por tecnología RISC. Así por ejemplo, un Pentium, sin dejar de pertenecer a la categoría CISC incorpora algunas características de los micros RISC. Es de esperar que en un futuro, los micros fabricados sean de tecnología RISC; entonces los ordenadores serán muchísimo más rápido de lo que hoy los conocemos.

Intel no fabrica microprocesadores completamente RISC para no perder la compatibilidad con los microprocesadores anteriores.

Capitulo III

3.1 Unidad aritmética lógica

Como ya se mencionó en el capitulo anterior, una unidad aritmética lógica es un circuito digital que realiza el cálculo ejecutando las operaciones (suma, resta, multiplicación etc.) y operaciones lógicas (AND, OR, XOR, NOT, igual, mayor que, menor que etc.) además de conocer sus diferentes componentes.

3.1.1. Concepto de operador:

Se llama operador a todo circuito capaz de realizar una operación aritmética o lógica; un operador es por tanto una unidad funcional capaz de realizar una o varias operaciones, tales como suma, resta, AND, OR, etc.

Los operadores se pueden clasificar por su función en:

1. Ámbito de aplicación.
2. Realización.
3. Numero de operación.
4. Paralelismo.
5. Operación.
6. Tecnología empleada.

1. Ámbito de aplicación del operador:

Según su ámbito de aplicación, los operadores pueden dividirse en generales y especializados. Los operadores generales pueden realizar distintas clases de operaciones, mientras que los especializados se restringen a una sola clase.

2. Realización del operador:

Dependiendo de su realización, los operadores pueden clasificarse en combinacionales y secuenciales.

Operador combinacional: es un circuito digital combinacional diseñado para que realice una o varias operaciones este no tiene elementos de memoria por lo que suministra la respuesta siempre que los operandos se mantengan fijos en sus entradas. Si se elimina un operando, o se le cambia el resultado se modifica.

Características de los operandos combinacionales

- No tiene elementos de memoria.
- Mientras las entradas fijas, la salida fija.
- Si se modifica una entrada, se modifica la salida
- El tiempo que tarda en dar el resultado es la suma de los retardos.
- Si se quiere guardar el resultado se debe utilizar un elemento de memoria.

El comportamiento de los circuitos combinacionales sólo depende de las señales de entrada en un instante determinado, y no de la secuencia de entradas, es decir, de la historia del circuito. Este hecho no quiere decir que el comportamiento temporal no sea importante, de hecho una de las principales características de los circuitos que se tienen en cuenta es la velocidad de operación o el retraso de propagación. En función de este retraso, podemos encontrar dos zonas temporales de operación bien diferenciadas: estado estacionario y estado transitorio. Una posible definición de estos estados sería la siguiente:

El estado transitorio: Es aquel espacio temporal que va desde el cambio de las entradas hasta que la salida se estabilice.
En este estado, tanto las señales internas como las de salida pueden sufrir cambios (no necesariamente uno solo, sino que pueden ser varios), aunque las señales de entrada no cambien. Estos posibles cambios son los necesarios para que el circuito busque su estabilización.

El estado estacionario: Es aquel espacio temporal que va desde la estabilización del circuito lógico hasta que las entradas vuelvan a cambiar.

En este estado, ninguna de las señales del circuito puede sufrir ningún cambio, a no ser que sean las señales de entrada. Es decir, en el estado transitorio se producen todos los cambios necesarios en las señales de salida (e internas) hasta conseguir la estabilización del circuito.

En cambio, en el estado estacionario, las señales de salida (e internas) están estables a su valor correcto. Por lo tanto, el comportamiento lógico hay que observarlo en el estado estacionario, en el cual no se producirá ningún cambio adicional debido al cambio actual de las señales de entrada.

La aplicación que se diseña en este trabajo se realizará con operadores combinacionales.

Operador secuencial: es un operador que requiere de varias fases para obtener el resultado, debiendo contar con los elementos de memoria necesarios para almacenar la información que debe transmitirse entre fases, y con un contador para estas.

En total se pueden realizar una operación por cuatro mecanismos distintos:

- Con un circuito combinacional (se tendría una sola fase).
- Con un circuito secuencial que genere sus propias fases.
- Generando las fases mediante la unidad de control del computador.
- Mediante un programa.

3. Número de operandos del operador:

Existen operaciones monádicas, que solamente emplean un operando, tal como la negación, y operaciones diádicas, que requieren dos operandos, tal como la suma o la multiplicación.

4. Paralelismo del operador:

Según el número de dígitos con los que opera simultáneamente, un operador puede clasificarse como paralelo o serie. Un operador paralelo realiza la operación sobre todos los dígitos de los operandos al mismo tiempo, mientras que uno en serie trabaja digito a digito. Por ello, el operador paralelo es un operador de palabra o vectorial, mientras que el operador serie es un operador de digito.

5. Operación realizada por el operador:

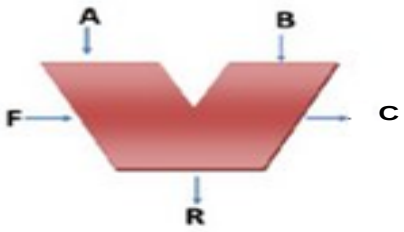
Los operadores pueden diseñarse para que hagan una gran variedad de operaciones entre las más frecuentes tenemos:

- ✓ Aritméticas: Negación, suma, resta, multiplicación, División.
- ✓ Lógicas: NOT, AND, OR, XOR.
- ✓ Desplazamiento: Logicos, circulares, aritméticos.

6. Tecnología Empleada:

Los operadores pueden fabricarse con cualquier tipo de tecnología MOS o bipolar, la velocidad de funcionamiento de los operadores depende de la tecnología de fabricación.

3.2 Estructura y operaciones de una unidad Aritmética.



Un típico símbolo esquemático para una ALU: A y B son operandos; R es la salida; F es la entrada de la unidad de control; C es un estado de la salida.

Los operadores son, casi siempre, de tipo combinacional y de tipo paralelo, encargándose la unidad de control del computador de llevar a cabo el control de los algoritmos de las operaciones complejas.

3.3 Operaciones típicas de la unidad aritmética lógica

Suma/resta en binario.

Suma/resta en coma flotante.

Suma/resta en BCD.

Multiplicación en binario.

División en binario.

División en coma flotante.

Operaciones lógicas.

Desplazamiento Unitario

Comparaciones Múltiples.

3.3.1 Operaciones de suma y resta

La operación de suma es el pilar sobre el que se construyen las unidades aritméticas de la mayoría de los computadores.

Sumador Elemental: Es un circuito combinacional capaz de sumar dos dígitos binarios, más el posible acarreo de la etapa anterior, produciendo el digito de suma y el de acarreo a la etapa siguiente.

El sumador elemental se emplea normalmente como bloque constructivo para formar sumadores paralelos.

Figura de sumador elemental:

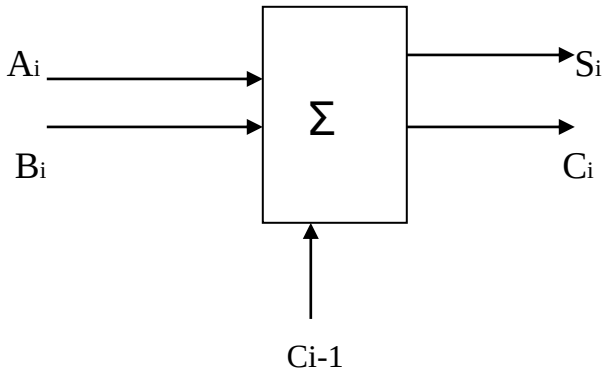
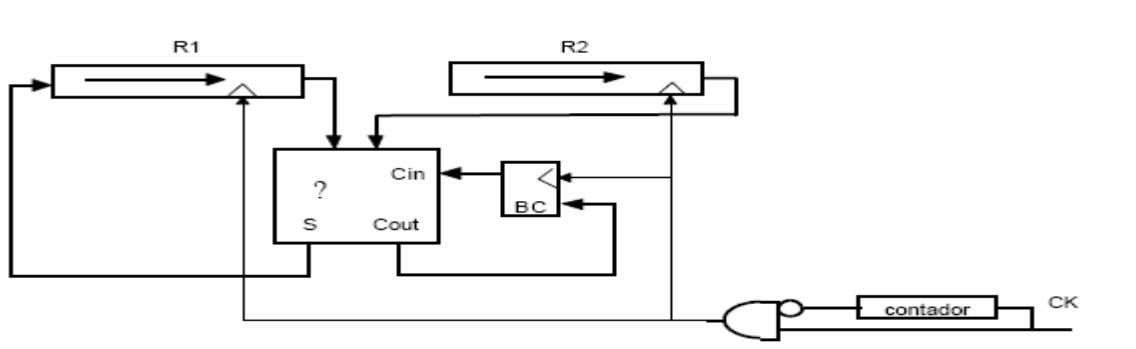


Tabla de verdad del sumador Completo.

a	b	C_{in}	C_0	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figura de un sumador secuencial:

Se supone que se quiere sumar el contenido de los Registros R1 y R2 y que los registros permiten desplazamientos a la derecha.



Se inicia el biestable de acarreo (BC) a cero y el contador con el número de fases.

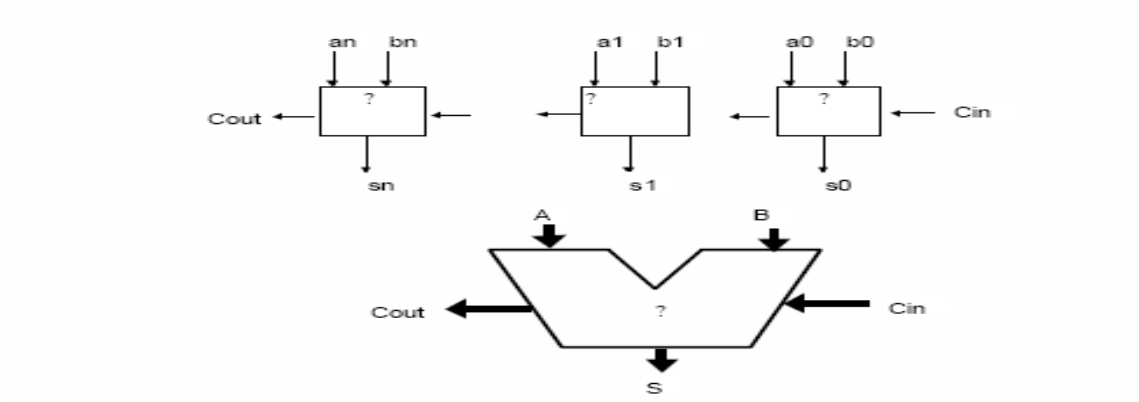
Ventaja: Barato en área porque con un sumador de 1 bit implementas un sumador de n bits

Desventaja: Bajo rendimiento. Puesto que se necesitan n ciclos de reloj para obtener el resultado final.

En este ejemplo aparecen claramente determinadas las ventajas y desventajas de la implementación combinacional (aumento de área y de rendimiento) frente a la implementación secuencial (disminución de área y rendimiento).

Sumador Paralelo: Se construye mediante N sumadores elementales.

Figura de un sumador paralelo con propagación de arrastre



El acarreo de entrada Cin depende del tipo de representación usada para binario sin signo Cin = 0.

El Cout se utiliza para detectar el desbordamiento.

El retardo total viene dado por el máximo número de puertas que deben atravesar las señales: el camino más largo es el de los Acarreos.

Restador Paralelo: El restador paralelo puede hacerse siguiendo el procedimiento empleado con el sumador mediante n circuitos restadores elementales.

Sumadores Rápidos: Existen varios procedimientos para aumentar la velocidad de la suma. Un procedimiento obvio, para acelerar la suma es generar los acarreos directamente, sin esperar a que tengan que propagarse entre las etapas sumadoras. Para ello, se tendrá que

diseñar un circuito específico que haga una generación adelantada de acarreo, circuito que se suele denominar anticipador de acarreo.

El anticipador de acarreo consiste en calcular directamente todos los acarreos con una lógica especial dedicada.

Estos sumadores son mas rápidos que los que van propagando el acarreo entre los sumadores elementales, porque las señales que anticipan el valor de los acarreos circulan a través de menos puertas en serie. Sin embargo, aumenta el costo al emplear mayor número de puertas en paralelo.

Para el cálculo del acarreo previo de la etapa i se calculan los valores de dos parámetros llamados Generación (G_i) y propagación (P_i).

G_i hace referencia a la generación del acarreo de salida en función de los valores de los bits de entrada y de forma independiente del valor del acarreo de entrada.

$$G_i = A_i \ B_i$$

Pi hace referencia a la posibilidad para que el acareo de entrada se propague al de salida, por eso la condición es:

$$P_i = A_i \oplus B_i$$

Utilizando Gi y Pi se pueden definir las ecuaciones lógicas de los acarreos que se generan en las diferentes etapas de un sumador.

Para el acarreo de la primera etapa

$$C_1 = G_1 + P_1 * C_0$$

Para el acarreo de la segunda etapa

$$C_2 = G_2 + P_2 * C_1$$

Sustituyendo en la ecuación de C2 el valor de C1

$$C_2 = G_2 + P_2 (G_1 + P_1 * C_0) = G_2 + P_2 * G_1 + P_2 * P_1 * C_0$$

Se notara que C2 se expresa únicamente en función de los valores de G, P y C_0

Este hecho es muy importante porque va a suponer que todos los acarreos dependen exclusivamente de A B y C_0.

Al igual que $\mathbf{C_2}$ se puede deducir la ecuación para $\mathbf{C_3}$

$$\mathbf{C_3 = G_3 + P_3 * G_2 + P_3 * P_2 * G_1 + P_3 * P_2 * P_1 * C_0}$$

Generalizando se llega a la siguiente expresión:

$$\mathbf{C_k = G_k + P_k * G_{k-1} + P_k * P_{k-1} * G_{k-2} + \ldots + P_k * P_{k-1} \ldots P_1 * C_0}$$

La ecuación de $\mathbf{C_k}$ indica que es posible calcular todos los acarreos que hay que aplicar a las etapas del sumador a partir del valor de los operandos y de $\mathbf{C_0}$ ya que los términos G y P dependen solamente de los mismos.

Ejemplo:

Si se introducen al sumador de la figura siguiente los operandos A= 011 y B= 010, calcular los valores obtenidos en las líneas de salida y de las señales G y P. Se supone que $\mathbf{C_0} = 0$.

Solución:

$\mathbf{P_1} = \mathbf{A_1} \oplus \mathbf{B_1} = 1$

$\mathbf{G_2} = \mathbf{A_2} * \mathbf{B_2} = 1$

$\mathbf{P_2} = \mathbf{A_2} \oplus \mathbf{B_2} = 0$

$\mathbf{G_3} = \mathbf{A_3} * \mathbf{B_3} = 0$

$\mathbf{P_3} = \mathbf{A_3} \oplus \mathbf{B_3} = 0$

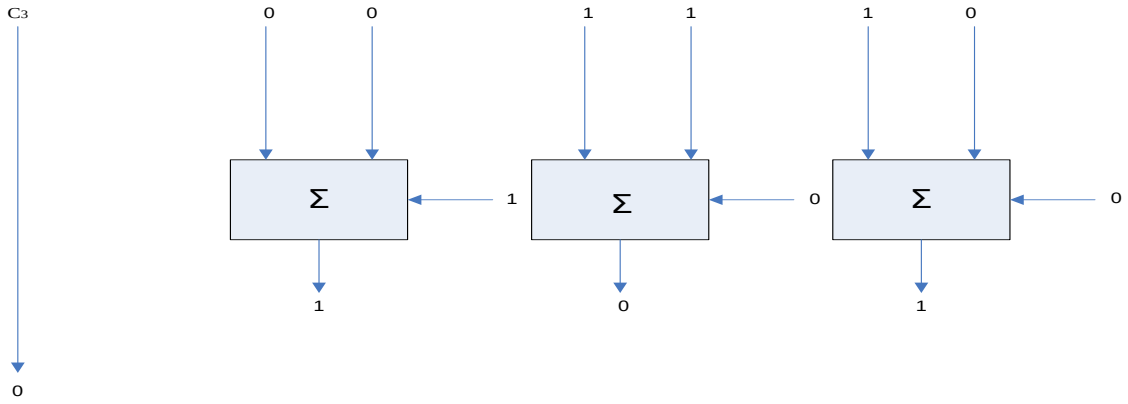
Con los valores de G P y $\mathbf{C_0}$ se pueden calcular los acarreos de cada etapa.

$\mathbf{C_1 = G_1 + P_1 * C_0} = 0 + 1 * 0 = 0$

$C_2 = G_2 + P_2 * G_1 + P_2 * P_1 * C_0$ = 1 + 0 * 0 + 0 * 1 * 0 = 1

$C_3 = G_3 + P_3 * G_2 + P_3 * P_2 * G_1 + P_3 * P_2 * P_1 * C_0 = 0$

Figura de representación del valor de las entradas y salida a las etapas del semador



Existen diferentes modelos de sumadores restadores de acuerdo con las diversas formas con las que se representan los números negativos, para poder efectuar la resta mediante la suma de un número positivo con otro negativo.

Los formatos más importantes para representar números negativos, con los que se obtiene el diseño de un sumador restador específico, son los siguientes:

- Complemento a uno.
- Complemento a dos.
- En signo de magnitud.

3.3.2 Sumador / restador en complemento a uno.

Para realizar la resta de dos números mediante una suma hay que sumar al minuendo el complemento a uno del sustraendo y al resultado hay que añadirle el acarreo final Cn-1.

Resta decimal	**Resta Binaria**	**Resta con complemento a uno**
7	0111	0111
-3	- 0011	+ 1100 **(complemento a uno)**
4	0100	10011
		+1 **(se le suma el acarreo)**
		0100

Aquí se utiliza la técnica del complemento a uno para las restas, se consigue añadir el acarreo final al resultado de la suma del minuendo con el complemento a uno del sustraendo, recirculando el acarreo de salida hasta la entrada.

El esquema general no se puede utilizar directamente. Hay que introducir modificaciones, teniendo en cuenta que la suma de dos números en complemento a uno se realiza en dos pasos: Se suman y luego Se suma el carry al resultado inicial. La negación del complemento a uno sólo exige la negación lógica.

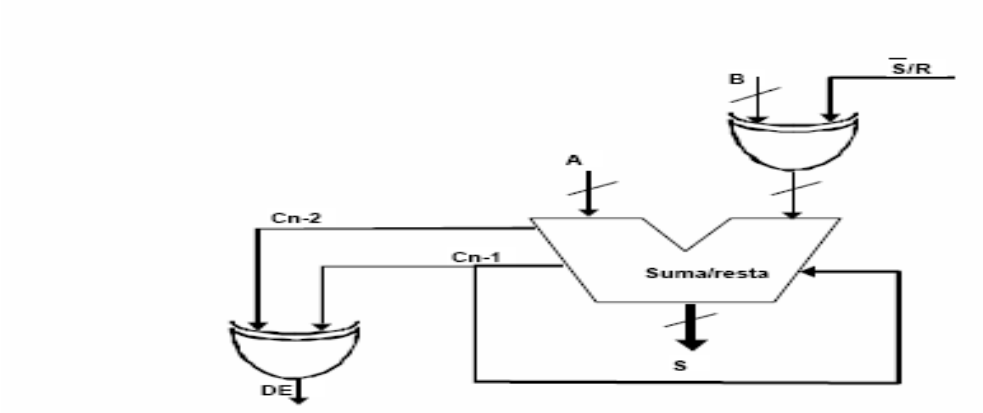


Figura de complemento a uno

Las condiciones de desbordamiento se obtienen con un razonamiento idéntico al anterior.

3.3.3 Sumador/restador en complemento a dos

A diferencia de la técnica en complemento a 1, aquí se desprecia el acarreo final al realizar una resta sumando al minuendo el complemento a2 del sustraendo.

Ejemplo:

Realizar la resta 0110-0010 utilizando la técnica del complemento a dos.

Solución:

0110

+ 1110 (complemento a dos).

10100 (se desprecia el acarreo).

Resultado: 0100

3.3.4 Sumador /restador en signo de magnitud.

En el sistema de representación binaria en signo y magnitud, tanto los números positivos como los negativos se expresan en forma no complementada. Al valor binario se antepone el bit de signo, que es un 0 para los números positivos y un 1 para los negativos.

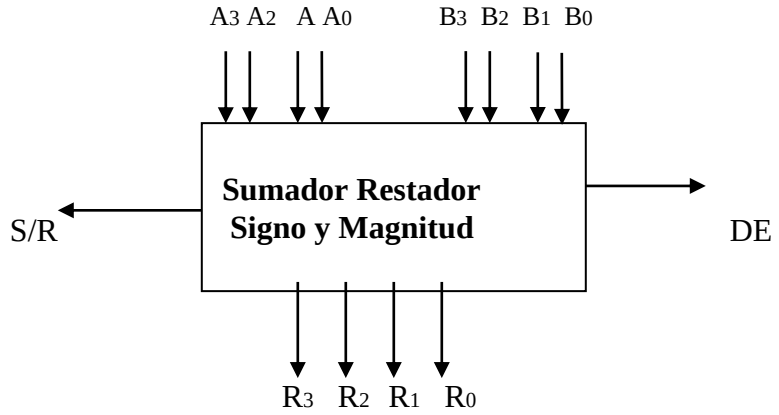


Figura simplificado de un sumador _Restador en signo de magnitud para números de 4 bits.

3.4 Números enteros sin signo

A continuación se detallan la manera como se pueden realizar operaciones aritméticas con números sin signo y enteros. Existen maneras para realizar la suma, resta, multiplicación y división.

Suma

Si se suman números decimales es común la expresión “cinco más siete es igual a doce, entonces son dos y llevo uno”. Este “llevo uno” se conoce como el acarreo. De la misma manera se realiza una suma binaria de varios bits. Únicamente se necesita recordar las sencillas reglas de la suma binaria de un bit.

1+1 = 0 y llevo 1

1+0 = 1

0+1 = 1

0+0 = 0

1+1+1 = 1 y llevo 1

Con estas reglas tenemos por ejemplo.

$$\begin{array}{rr} \text{Acarreo -->} & \begin{array}{r} 1\ \ 11 \\ 10010111 \\ +\ 01010110 \\ \hline 11101101 \end{array} \end{array}$$

Resta

La resta se lleva de manera similar en números decimales y binarios. Si en la suma existe el acarreo, en la resta existe el “préstamo”, igual que en decimales.

1-1 = 0

1-0 = 1

0-0 = 0

.0-1 = 1 prestando un uno al bit siguiente.

Con estas reglas tenemos por ejemplo.

$$\begin{array}{r} 11001001 \\ -\ 00110101 \\ \hline 10010100 \end{array}$$

Multiplicación

La multiplicación no es, en lo absoluto más complicada que las dos operaciones anteriores que hemos visto. Simplemente se trata de multiplicar con unas sencillas reglas y después sumar. Cuando realizamos multiplicaciones decimales de varios números debemos multiplicar cada uno de los números del multiplicador por cada uno de los del multiplicando corriendo el resultado un lugar hacia la izquierda para después sumar los resultados.

Las reglas de la multiplicación son:

1x1 = 1

1x0 = 0

0x1 = 0

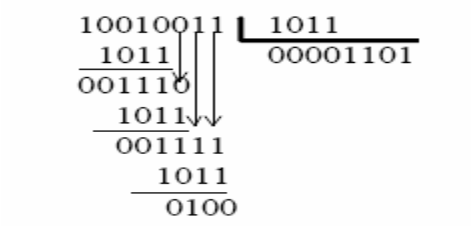
0x0 = 0

De esta manera tenemos:

1231	1101
*x*45	*x*101
..6155	1101
4924	..0000
55395	1101
	1000001

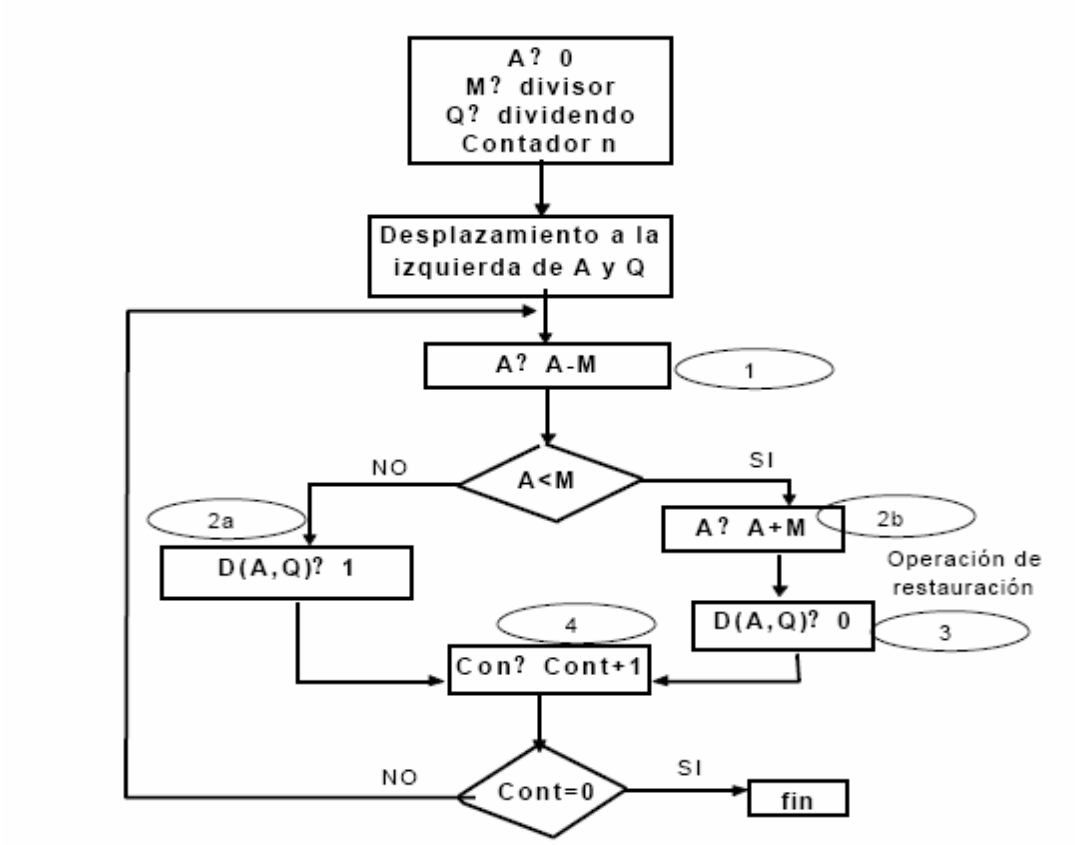
División.

Es muy simple realizar una división de números binarios pues es sumamente similar a la división con números decimales. Hay un cociente como resultado con un posible residuo, hay un dividendo y un divisor y se trata de encontrar múltiplos e irlos restando uno por uno. No creo necesario extenderse en la explicación paso a paso de la división de números enteros positivos en binario. Basta con un pequeño ejemplo y recordar las reglas para la división de números decimales ya que son iguales.



La razón de cargar inicialmente en A todo ceros es para que en el primer desplazamiento quede cargado en A el primer bit del dividendo y así comenzar el algoritmo A y Q están concatenados de manera que cada desplazamiento a la izquierda carga en A un bit del dividendo en el resto parcial. Esto se hace para comprobar si el resto parcial "cabe o no cabe". La forma de comprobar si cabe es restar el divisor del resto parcial, si el resultado es negativo (SN-1 = 1) la resta no es valida, y para restaurar el resto parcial se suma el divisor.

El algoritmo se puede ver en la siguiente figura.



3.5 Números enteros con signo.

Complemento a dos

Es interesante lo que se puede hacer con números binarios. Como ya se realizo anteriormente hay maneras de realizar sumas, restas y multiplicaciones, pero se han realizado operaciones con números enteros y positivos. ¿Acaso se puede hacer operaciones con números binarios que representen números negativos? Sí se puede y no es tan

complicado como pareciera ser. Para esto se llegó a una notación especial denominada complemento a 2.

Es más sencillo explicarlo si se toma en cuenta registros de 8 bits. Cuando queremos representar números positivos, el bit más significativo servirá de signo: si es 0 es positivo y si es 1 es negativo. Esto deja únicamente 7 bits para el número o bien, hasta 64 números se pueden formar. Lo importante es que de cada número positivo se puede encontrar su negativo por medio del complemento a dos. Esto se hace sacando el complemento de dicho número y sumándole un uno.

Número positivo: 00000100 = 4

Complemento: 11111011

Se le suma uno: 11111011+1=11111100 = -4

Si alguna de las operaciones aritméticas arroja un resultado negativo, dicho resultado estará en representación complemento a dos. Para leer mejor el resultado solo se tiene que convertir dicho número negativo a un número binario normal. Esto se hace calculando el complemento a dos de dicho numero (que ya esta en complemento a dos). Por ejemplo, si el resultado de una suma o una resta nos da –18 (11101110) tendríamos que complementarlo primero (00010001) y después sumarle 1 (10001 + 1) lo cual daría 10010 = 18. Es claro que al convertirlo a notación binaria normal el numero da positivo, pero esto solo se hace con fines de poder leer mejor el resultado. Se sabe que el resultado es negativo, solo se necesitaba averiguar la magnitud que es 18.

Suma

La suma en complemento a2 es sencilla ya que lo único que tenemos que hacer es convertir a negativo cualquier número que tenga signo negativo. Por ejemplo: Si tenemos 6+9 no es complicado ya que ambos son positivos y sabemos realizar este tipo de sumas.

$$\begin{array}{r} 00001110 \\ +\ 00001001 \\ \hline 00011111 \end{array} \approx \begin{array}{r} 6 \\ +\ 9 \\ \hline 15 \end{array}$$

Pero si la suma es –6+9, entonces debemos calcular el complemento a2 del 6 que en binario es 110. Dicho complemento se obtiene negando primero (11111001) y sumándole 1 después (11111001 + 1) para lo cual nos queda 11111010. La suma quedará como sigue:

$$\begin{array}{r} 11111010 \\ +\ 00001001 \\ \hline 00000011 \end{array}$$

Si hay dos números negativos simplemente hay que convertir en complemento a dos ambos números. La suma -9 + (-9) seria:

$$\begin{array}{r} 11110111 \\ +\ 11110111 \\ \hline 11101110 \end{array}$$

3.6 Suma y resta en BCD.

Los Dígitos Decimales Codificados en Binario (BCD) se representan con 4 bits.

Números	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Los sumadores y restadores BCD siguen las mismas reglas que los binarios, si bien precisan de circuitos de corrección, pues en binario o hexadecimal con 4 bits se puede alcanzar el valor máximo 1111, que en decimal equivale al 15 y se expresa con dos dígitos. Cuando el resultado de sumar dos números BCD sobrepasa el 9(1001) hay que aplicar un

circuito corrector que transforme la magnitud binaria de 4 bits en una correspondiente a BCD formada por dos dígitos de 4 bits cada uno.

Ejemplo:

Realizar la siguiente suma con los operandos expresados en código BCD.

$$\begin{array}{r} 386 \\ +\ 243 \\ \hline 629 \end{array}$$

Solución:

En BCD se deben realizar tres sumas, uno para cada digito: centenas, decenas y unidades

$$\begin{array}{rrr} 0011 & 1000 & 0110 \\ +\ 0010 & +\ 0100 & +\ 0011 \\ \hline 0101 & 1100 & 1001 \end{array}$$

Los tres dígitos del resultado corresponden con los valores decimales 5, 12 y 9. El numero 12 no corresponde con ninguno de los 10 posibles códigos BCD, lo que significa que hay que corregirlo.

El método para la corrección de los números que sobrepasan el valor 9 consiste en restarles 10 y añadir el acarreo a la etapa siguiente. En este ejemplo, el numero 12 quedaría en 12 – 10 = 2 y al siguiente digito se le añadiría una unidad de acarreo convirtiéndose en 5 + 1 = 6. Así, el resultado corregido seria 01100010 1001, equivalente al valor decimal 629.

Para corregir el resultado de cualquier etapa sumadora BCD cuyo valor supere al 9 en lugar de restar 10 (en binario1010), se suma el complemento a dos del 10, que en binario es el 0110 y que corresponde con el valor decimal 6 esta corrección precisaría un segundo

sumador que, en el caso de que el primero superase al valor 0, sumaria el valor 0110, mientras que, en caso contrario, no modificaría el valor sumando 0000.

Se precisa un circuito detector de números que superan al 9, el cual, consiste en un conjunto de tres puertas lógicas: una OR y dos AND.

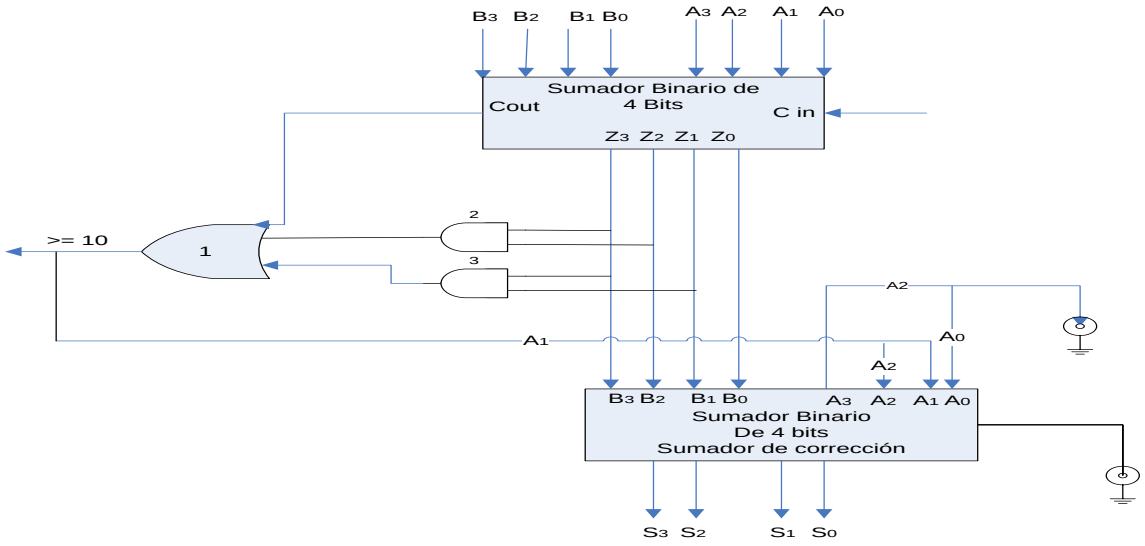
1ro. Si la entrada a la OR correspondiente a la salida C vale 1, significa que el valor de la suma supera el numero 1111, que según es el 15_{10}.

2do. Si vale 1 la entrada a la OR procedente de la AND-2, significa que los bits Z3 y Z2 valen 1, lo que detecta los números: 1100(12), 1101(13), 1110(14), 1111(15).

3ro. Finalmente la salida de la OR detecta que el resultado de la suma es superior a 9 cuando vale 1 la entrada procedente de la AND -3, lo que significa que $Z_3 = Z_1 = 1$, es decir, la suma es el numero 1010(10), o 1011(11).

A continuación se muestra el esquema de una etapa sumadora BCD, con la lógica de detección de números superiores al 9 en el primer sumador y el segundo sumador de corrección, que añade el valor 0110 cuando el primer sumador produce un resultado igual o mayor que 1010.

Figura_BCD de la etapa de un sumador BCD formada por un primer sumador de los operandos BCD. Una lógica de resultados a 9 de dicho sumador y un segundo sumador para la corrección de los resultados iguales o superiores a 1010.



Figura_BCD

3.7 Operadores básicos en coma Flotante.

Con la finalidad de aumentar el rango y la precisión de los números no enteros usando una cantidad concreta de bits, se utiliza el sistema de representación en coma flotante, también llamado notación científica.

Como los computadores trabajan con 1 y 0, se hace referencia a expresiones binarias en coma flotante. Un número en coma flotante dispone de un número fijo de bits, que suele corresponder con el tamaño de la palabra que usa la máquina y que se divide en tres campos:

1) Campo: Bit de signo (s)
2) Campo: Bits destinados al exponente (E)
3) Campo: Bits destinados a la mantisa (M).

El valor del exponente determina la posición de la coma que indica el lugar de comienzo de la parte fraccionaria. De aquí el nombre de “Coma flotante”

Se describen las características fundamentales del formato correspondiente al estándar IEEE 754 de 1985, debido a su implantación en todos los sistemas informáticos.

Según la norma IEEE 754, la mantisa M es la parte fraccionaria del numero binario que hay que añadir a la unidad. Sus bits asignan el peso de las sucesivas potencias negativas de la base 2. El valor de la fracción de una mantisa M (m_1, m_2, m_3,…) es $m_1 * 2^{-1} + m_2 * 2^{-2} + m_3 * 2^{-3} +$…

El exponente E se trata de un numero entero con signo expresado en exceso a $2^{n-1}-1$, siendo n el numero de bits que tiene el exponente. La base de exponenciacion es 2. Ejemplo:

Averiguar a que valor corresponde el numero X dado en formato de coma flotante según norma IEEE 754 de simple precisión.

X = 01000001111000000000000000000000

Solución:

Bit de Signo: 0

Exponente: 10000011 Mantisa: 11000000000000000000000

a) Al ser el bit de signo s = 0, el valor de la expresión es positiva ya que $(-1)^s = +1$
b) Como el exponente esta expresado en exceso a $2^{n-1}-1$, o sea, en exceso a $2^{8-1}-1 = 127_{10} = 1111111_2$, el valor del exponente E será

$$E = 10000011_2 - 1111111_2 = 131_{10} - 127_{10} = 4_{10}$$

c) La mantisa es la parte fraccionaria que hay que sumar a la unidad y sus bits representan las sucesivas potencias negativas de la base 2.

$$11000000000000000000000 = 1 * 2^{-1} + 1 * 2^{-2} = 0{,}75_{10}$$

De donde el valor decimal del número X será:

$$X = + 1{,}75 * 2^4 = + 28_{10}$$

3.8 Operadores para la Multiplicación

El diseño de operadores específicos para realizar la multiplicación resulta complicado y costoso, razón por la que solo los procesadores mas potentes, como el Pentium pro, dispongan de los mismos.

La multiplicación se suele desarrollar empleando sumadores/restadores y un algoritmo adecuado. A continuación se describen los algoritmos y esquemas de operadores de multiplicación más importantes.

3.8.1 Algoritmo de suma y desplazamiento

Este algoritmo se basa en el método manual para efectuar las multiplicaciones por lo que también se le conoce con el nombre de algoritmo de lápiz y papel.

El método que se emplea para multiplicar dos números, A y B, manualmente puede resumirse en las siguientes etapas:

1) Se inspeccionan sucesivamente los bits del multiplicador B.
2) Si Bi = 1 se suma al resultado el multiplicando A, desplazado a la izquierda i-1 posiciones.
3) Si Bi = 0, no se hace nada.

En lugar de desplazar a la izquierda el multiplicando, los operadores de multiplicación desplazan a la derecha el resultado parcial, lo que supone el mismo efecto. Por cada bit del multiplicador cuyo valor sea 1, se desplaza el resultado una posición a la derecha y se suma el multiplicando.

3.8.2 Multiplicador rápido

Este operador se caracteriza por generar simultáneamente todos los sumandos que se forman al realizar manualmente el producto. Luego los suma directamente, añadiendo cada acarreo a la etapa siguiente.

3.9 Operadores de división.

Como sucede con la multiplicación, los procesadores potentes que disponen de operadores especializados para realizar divisiones binarias utilizan un sumador-restador y un algoritmo adecuado para implementar la operación secuencialmente.

3.10 Operadores de desplazamiento

Hay numerosas situaciones en el proceso de la información que requieren desplazar de posición los datos almacenados en los registros. Es el caso de algunas operaciones aritméticas como la multiplicación y división por potencias de dos. También cuando se quiere situar uno de los bits del registro en una determinada posición del mismo.

La ALU de un procesador puede disponer de registros capaces de efectuar determinado número de desplazamiento de sus bits tanto a la derecha como a la izquierda el gobierno de estos operadores de desplazamiento se realiza mediante las adecuadas señales de control.

En este tipo de operadores de desplazamiento a veces el contenido desplazado del registro origen se almacena en el registro destino, pero es frecuente que el contenido desplazado de un registro se deposite en el mismo actuando como origen y destino.

Los operadores de desplazamientos se clasifican en:

1) Operadores de desplazamiento lógicos.
2) Operadores de desplazamiento aritméticos.
3) Operadores de desplazamiento circulares.

3.10.1. Los operadores de desplazamiento lógicos

La **figura 1** muestra como se pierden bits del dato inicial por el extremo derecho o izquierdo según el sentido del desplazamiento. Por el extremo opuesto a donde salen bits

entran ceros, aunque hay algunos operadores que introducen uno a las posiciones que se vacían.

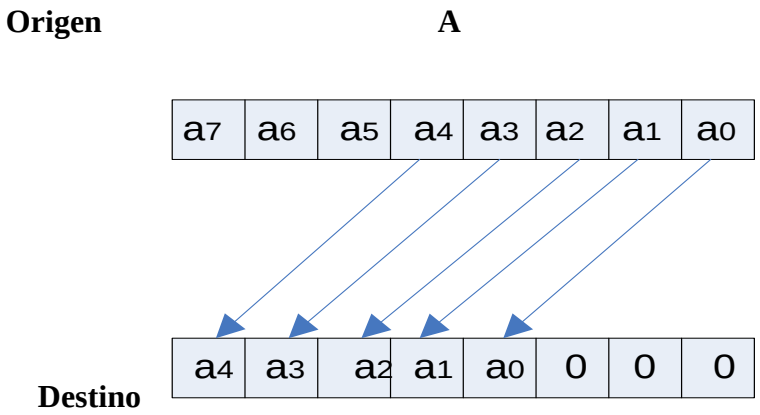


Figura1

Figura 1 Actuación de un operador lógico de desplazamiento de tres posiciones a la izquierda**.** Los tres bits de peso de A se pierden y en B se rellenan con los tres bits de menos peso.

Ejemplo 1:

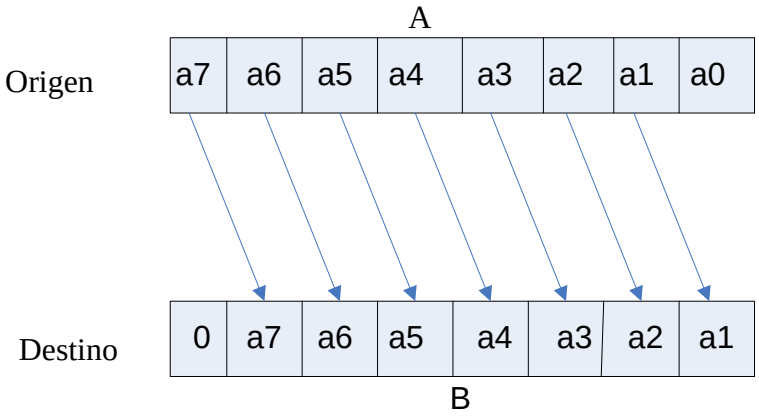


Figura 2

Figura 2 Actuación de un operador lógico de desplazamiento de una posición a la derecha**.**

Dibujar el contenido de un registro destino al aplicarle un operador lógico de desplazamiento de dos posiciones a la izquierda al contenido de un registro origen de 8 bits cuyo contenido es 8DH.

Solución:

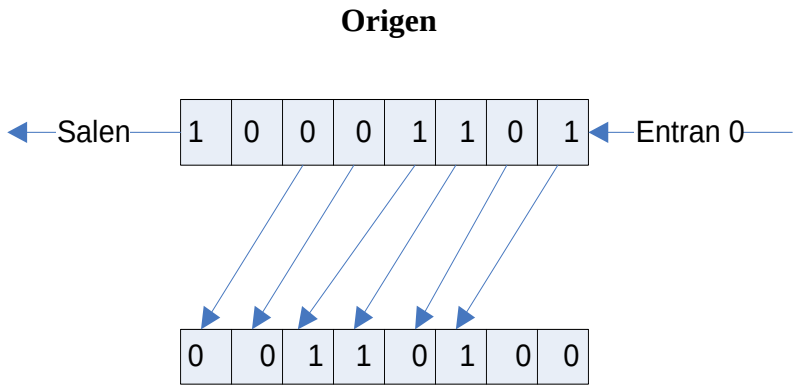


Figura 3 **Destino**

Figura 3 Actuación de un operador lógico de desplazamiento de dos posiciones a la izquierda. Se pierden los dos bits de más peso del registro origen.

3.10.2. Los operadores de desplazamiento aritméticos

Estos son similares a los lógicos pero mantienen invariable el bit de más peso que representa el signo de la magnitud.

En la práctica, estos desplazadores suponen una multiplicación (izquierda) o división (derecha) por 2^n, siendo n el número de bits que se desplazan.

Si el valor que contiene el registro origen es negativo y esta representado en el formato de complemento a uno, cuando se multiplica se debe introducir uno por la derecha.

En los desplazamientos aritméticos a la derecha, además de mantener su valor el bit de signo también se desplaza.

Ejemplo: Indicar el resultado que produce un operador de desplazamiento aritmético para multiplicar por 4 el contenido del registro origen.

Solución:

Para multiplicar por dos elevado a dos hay que desplazar dos posiciones a la izquierda el valor inicial, pero manteniendo invariable el bit de signo. En la siguiente figura.

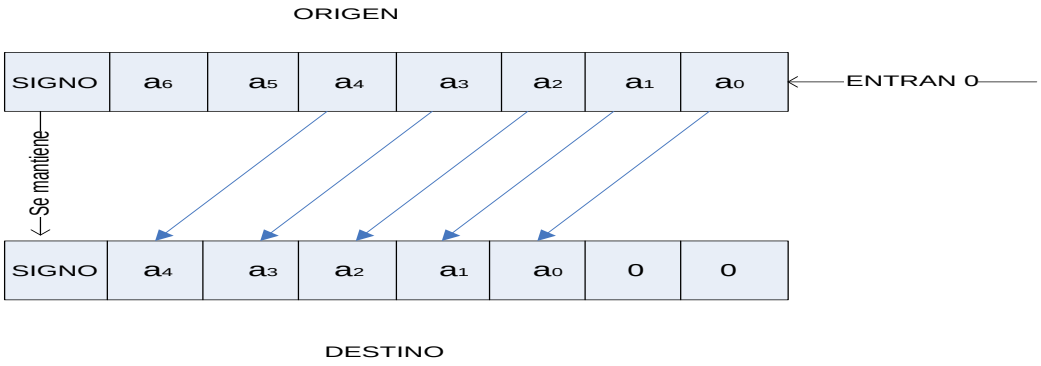


Figura 4

Figura 4 Actuación de un operador de desplazamiento aritmético que multiplica por 4 el valor del registro origen. Los bits a_6 y a_5 del registro origen se pierden.

3.10.3. Los operadores de desplazamiento circulares

Los operadores de desplazamiento aritmético y lógico se denominan “abiertos”, por que se pierden los bits que rebosan. Reciben el nombre de operadores “cerrados” los circulares por que los bits que rebosan por un extremo los introducen por el opuesto. No se pierde información solo cambia de lugar.

Ejemplo:

Exponer el comportamiento de un operador de desplazamiento circular de dos posiciones a la derecha.

Solución:

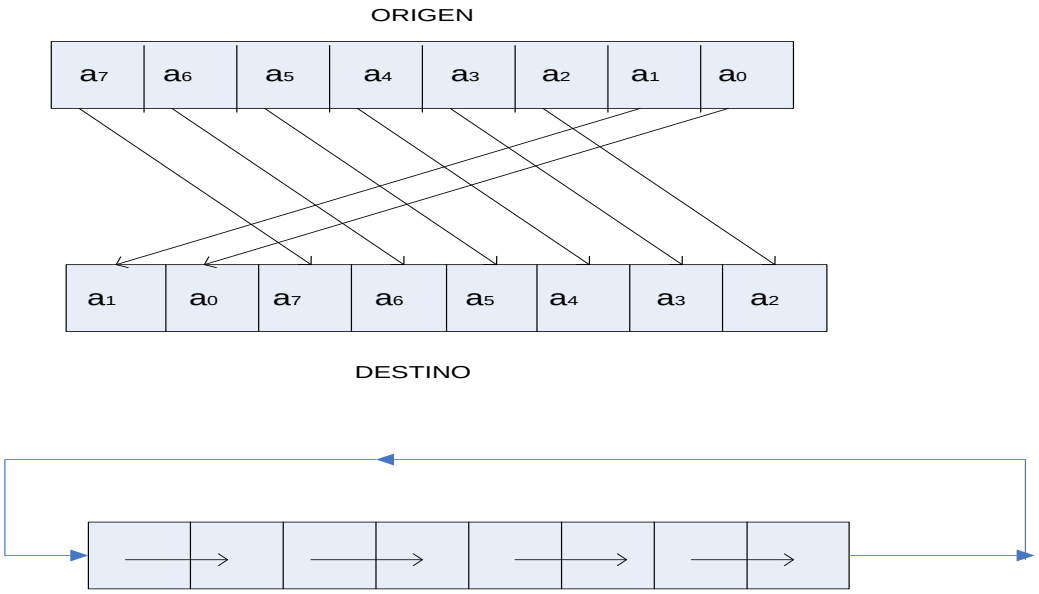


Figura 5 Actuación de un operador de desplazamiento circular de dos posiciones a la derecha

3.11 Operaciones Lógicas.

Las operaciones lógicas, que generalmente se encuentran en los computadores, son las cuatro siguientes:

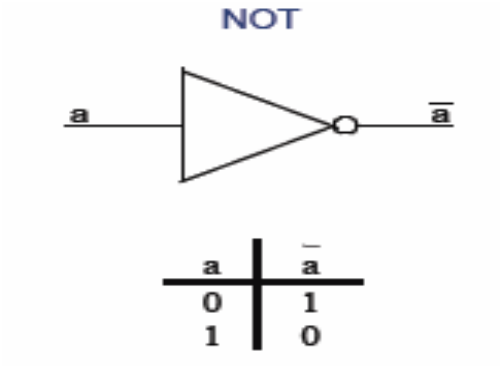
- Negación NOT.
- Suma OR.
- Producto AND.
- Suma Exclusiva XOR.

La primera es una operación monádica, mientras que las otras tres son diádicas. La realización de estas operaciones es inmediata, englobándose generalmente en un solo operador combinacional, en unión a las operaciones aritméticas elementales.

Negación NOT: la puerta lógica NO (NOT en ingles) realiza la función booleana de inversión o negación de una variable lógica. Una variable lógica A la cual se le aplica la negación se pronuncia como “no A” o “A negada”.

La ecuación característica que describe el comportamiento de la puerta NOT es:

$F=\overline{a}$

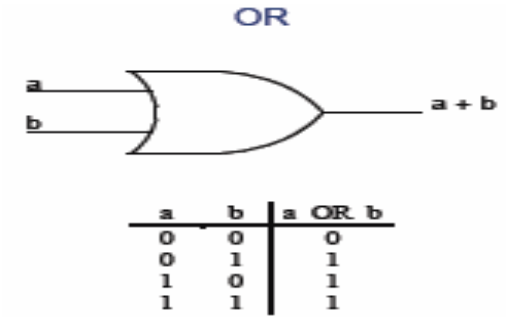


Figura_NOT

Suma OR: La puerta lógica O, más conocida por su nombre en ingles OR realiza la operación de suma lógica.

La ecuación característica que describe el comportamiento de la puerta OR es:

F= a+b



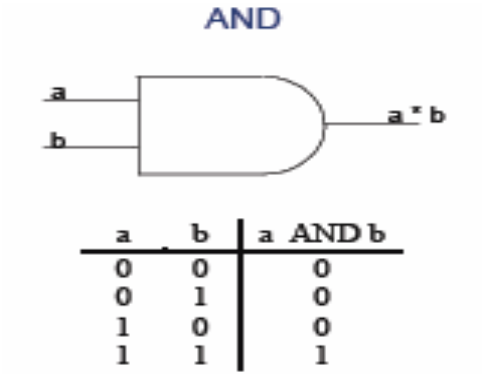
Figura_OR

Se Puede definir la puerta O como aquella que proporciona su salida un 1 lógico si almenos una de sus entradas está a 1.

Producto AND: la puerta lógica Y, más conocida por su nombre en ingles AND, realiza la función booleana de producto lógico. Su símbolo es un punto (.), aunque se suele omitir. Así, el producto lógico de las variables a y b se indica como ab, y se lee a y b o simplemente a por b.

La ecuación característica que describe el comportamiento de la puerta AND es:

F= (a). (b)



a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

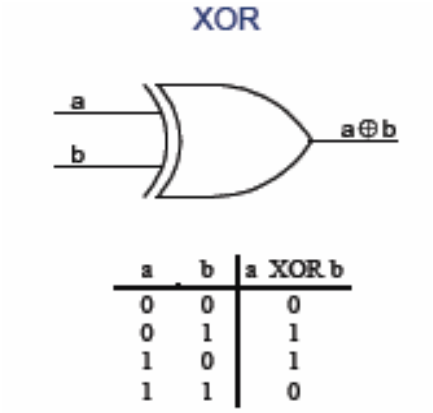
Figura_AND

Suma Exclusiva XOR: la puerta lógica O-exclusiva, mas conocida por su nombre en ingles XOR, realiza la función booleana a´b + ab´. Su símbolo es el (+) inscrito en un circulo.

La ecuación característica que describe el comportamiento de la puerta XOR es:

F= a $\oplus$ b

Se puede definir esta puerta como aquella que da por resultado uno, cuando los valores en las entradas son distintos. Ej: 1 y 0, 0 y 1 (En una compuerta de dos entradas).



a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Figura_XOR

Operaciones de comparación

Son sistemas combinacionales que reciben dos datos de entrada A y B y los comparan en binario puro, devolviendo 3 salidas que indican si A>B, A=B ó A<B.

En la **figura _6** muestra las operaciones de comparación (igual, mayor, menor).

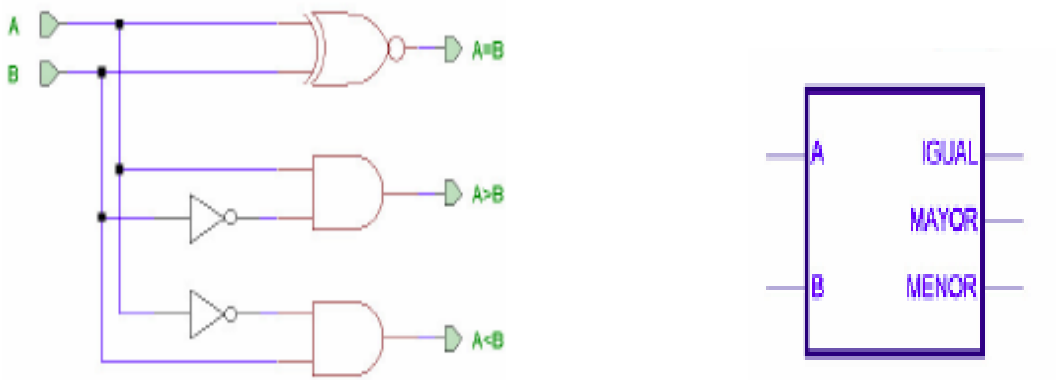


Figura_ 6

3.12 Técnicas de redondeo

Existen varias formas de redondeo pero se abordara las más corrientes:

- Truncamiento.
- Redondeo propiamente.
- Bit menos significativo a uno.

Truncamiento: Este consiste en cortar simplemente los bits de la derecha que no caben en la representación. Esta es una función muy fácil de ejecutar.

Redondeo propiamente a veces llamado al redondeo más próximo toma el valor más próximo al que se quiere representar. El redondeo minimiza los errores puesto que toma el valor más próximo. Además los errores cometidos son tanto por defecto como por exceso, lo que tiende a evitar su acumulación.

El tercer método consiste en truncar y forzar el bit menos significativo a uno. Este método es tan rápido como el truncamiento simple y presenta la ventaja de que sus errores son por defecto y no por exceso, lo que tiende a limitar su acumulación.

	Truncamiento	Redondeo	Forzado 1
1100110111 111	1100110111	1100111000	1100110111
1000100010 111	1000100010	1000100011	1000100011
1000000000 100	1000000000	1000000001	1000000001
1111111111 000	1111111111	1111111111	1111111111

Capitulo IV

4.1VHDL

4.1.1 Breve reseña histórica

VHDL es un lenguaje utilizado para describir circuitos en un nivel alto de abstracción el cual está siendo rápidamente aceptado como un medio estándar de diseño. VHDL es producto del programa Very High Speed Integrated Circuit (VHSIC) desarrollado por el Departamento de Defensa de los Estados Unidos a finales de la década de los 70. El propósito era hacer un estándar para diseñar, modelar, y documentar circuitos complejos de manera que un diseño desarrollado por una empresa pudiera ser entendido por otra, además, que pudiera ser procesado por software para propósitos de simulación

En diciembre de 1987 VHDL se estableció como el estándar IEEE-1076. En 1993 el estándar IEEE-1076 se actualizó y un estándar adicional, el IEEE-1164, fue adoptado. En 1996, el estándar **IEEE-1076.3** se convirtió en un estándar de **VHDL para síntesis** siendo este el que se utiliza en el diseño de sistemas digitales.

En la actualidad VHDL es un estándar de la industria para la descripción, modelado y síntesis de circuito digitales. El mercado entero de síntesis ha alcanzado aproximadamente los 100 millones de dólares, con un crecimiento del 20% al 30% al año. Por esto, los ingenieros de la mayoría de las áreas de electrónica, si no es que todas, deben aprender a programar en VHDL para incrementar su eficiencia.

Lo que ha hecho que VHDL sea en un tiempo tan corto el lenguaje de descripción de hardware más utilizado por la industria electrónica, es su independencia con la metodología de diseño utilizada por cada programador, su capacidad de descripción a diferentes niveles de abstracción, y en definitiva la posibilidad de poder reutilizar en diferentes aplicaciones un mismo código.

4.2 Características del lenguaje

VHDL corresponde a un lenguaje de descripción de hardware de gran versatilidad derivado de un lenguaje de alto nivel conocido como ADA. VHDL es un lenguaje con una sintaxis amplia que permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital y el desarrollo de modelos de simulación del mismo. Uno de los objetivos del lenguaje VHDL es el modelado. Se entiende por modelado al desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la simulación.

Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. La síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño yendo hacia el más bajo nivel de la jerarquía.

Existen distintas formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y la interconexión entre los mismos, es decir, se describe la estructura del circuito.La segunda forma consiste en describir un circuito indicando lo que hace o como funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador, puesto que lo que realmente le interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente, algunos problemas a la hora de la síntesis pueden aparecer.

VHDL soporta distintos tipos de descripciones:

Estructural. VHDL puede ser usado como un lenguaje de descripción estructural, donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.

Comportamental. VHDL también se puede utilizar para la descripción comportamental o funcional de un circuito. Sin necesidad de conocer la estructura interna

de un circuito, es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación ya que permite simular un sistema sin conocer su estructura interna, pero este tipo de descripción se esta volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

Una descripción en VHDL puede ser mixta, es decir puede comprender varios estilos. Por ejemplo, en una descripción estructural, los elementos más bajos de la jerarquía de diseño pueden definirse en forma de flujo de datos, debido a que el lenguaje incluye un conjunto de operadores predefinidos, como es el caso de los operadores lógicos, que pueden emplearse en la descripción.

4.3. Fundamentos del lenguaje

Toda descripción de un sistema en VHDL esta constituida por, al menos, tres tipos de elementos: Entidades, Arquitecturas, Bibliotecas.

Generalmente, se realiza la descripción de un diseño definiendo una entidad que represente el mismo a través de una arquitectura que especifique su funcionamiento o la interconexión de componentes que lo integran utilizando elementos almacenados en bibliotecas.

En esta sección se pretende dar una idea de los elementos que forman una descripción VHDL.

4.3.1. Entidades y Arquitecturas.

Independientemente del tipo de descripción que se utilice, todo sistema digital debe poseer un nombre que lo identifique de forma univoca dentro del diseño. En una descripción VHDL, esto se conoce como entidad de diseño y la misma no solo define el nombre que identifica al mismo, sino también las entradas y salidas que el sistema posee.

Toda entidad de diseño debe tener asociado un cuerpo de arquitectura donde se especifique la funcionalidad del mismo o bien los componentes que integran el diseño de acuerdo al tipo de descripción que se utilice.

Una entidad de diseño puede tener más de una arquitectura. Por ejemplo, se puede definir una arquitectura orientada a la simulación del diseño y otra orientada a la síntesis.

4.3.2. Bibliotecas y paquetes.

En descripciones complejas de un circuito o programa, se hace necesaria una organización que permita al diseñador trabajar con grandes cantidades de información. La forma de organizar un diseño a fin de darle una jerarquía al mismo es a través de las bibliotecas y los paquetes.

Las bibliotecas (libraries) almacenan distintos elementos a utilizar en una descripción VHDL. Las mismas pueden incluir componentes, tipos de datos, procedimientos, funciones, etc.

Estos elementos se agrupan en una biblioteca determinada en unidades denominadas paquetes (packages).

4.3.3. Configuraciones.

Las configuraciones son descripciones encargadas de que una arquitectura, de entre todas las existentes, se asocie con una entidad dada. Cuando se describe un circuito complejo, normalmente se realizan varias arquitecturas de un componente representado por una única entidad. Las configuraciones constituyen una herramienta muy útil porque definen la arquitectura que se desea asociar a una entidad y a su vez el conjunto entidad arquitectura que se asocia con cada componente del circuito.

4.4. Elementos de la sintaxis.

En primer lugar se debe mencionar que VHDL no discrimina entre el uso de mayúsculas o minúsculas (case insensitive), por lo tanto “ejemplo”, “EJEMPLO” y “EjEmPlO” son interpretados de la misma forma.

4.4.1. Comentarios

Los comentarios en VHDL son cadenas de caracteres que comienzan con dos guiones y se extienden hasta el final de la línea. Son ignorados por el lenguaje ya que su propósito es precisamente el de introducir aclaraciones.

−− Comentario

4.4.2. Identificadores

Los identificadores en VHDL representan los elementos del lenguaje. Se distinguen dos tipos de identificadores: básicos y extendidos.

Un identificador básico es cualquier conjunto de letras, números y el carácter “_ ” donde el primer carácter sea una letra del alfabeto ingles, el último carácter no puede ser “_ ”, y no se encuentren dos caracteres “_ ” seguidos. Tampoco pueden utilizarse como identificadores las palabras reservadas del lenguaje.

Los identificadores extendidos se introdujeron en la revisión del lenguaje de 1993, los cuales pueden contener cualquier carácter estándar. Un carácter extendido se delimita con caracteres “\”.

Así por ejemplo, identificadores validos son los siguientes:

Entrada
Salida _1
\ signal_1 %123\
\ signa l2 \\123\

Nótese que dentro de un identificador extendido puede utilizarse el carácter "\" siempre que se coloquen dos consecutivos, lo cual es interpretado como uno solo.

4.4.3. Literales

Los literales son símbolos cuya representación indica directamente su valor.

4.4.3.1. Números

Los números se pueden representar en base diez o en cualquier base que sea potencia de dos entre dos y dieciséis. Sin embargo, esto debe indicarse explícitamente por medio del carácter "#". Si el número incluye un punto representa un número real, de lo contrario representa un número entero. Los dígitos de un numero entero pueden separarse por medio del carácter " " de forma tal de mejorar la legibilidad.

Los números reales pueden contener un exponente al final, precedido por la letra E.

Ejemplos: 0 6 12 345 890 47.987 9 −− Números en base diez

2. 89E−8 −− Número real en donde el punto se desplaza a 8
−− Lugares hacia la izquierda
1. 77E+6 −− Número real en donde el punto se desplaza a 6
−− Lugares hacia la derecha

2#11000010# 2#1.000 111# −− Números en binario
4#2312# 4#23.122# −− Números en base cuatro
16#4BAC# 16#4F. 8 9# −− Números en hexadecimal

4.4.3.2. Caracteres

Los caracteres están constituidos por un carácter ASCII encerrado entre comillas simples.

Ejemplos: 'a' 'A' '0' 'z '

4.4.3.3. Cadenas de caracteres (strings)

Los strings se construyen encerrando una secuencia de caracteres ASCII entre comillas dobles. Para incluir una comilla dentro del string, se deben incluir dos comillas seguidas.

Ejemplos:

” Esto es un string”

4.4.3.4. Cadenas de bits

Las cadenas de bits se utilizan para representar arreglos compuestos por los caracteres “0” y “1”. Las cadenas de bits se encierran entre comillas dobles. Además es posible definir cadenas de bits en base dieciséis, ocho o dos precediendo el valor a codificar con las letras X, O B respectivamente.

Ejemplos:

X” 4BFF” O” 2320 ” B” 10011110 ”

4.4.3.5. Literal nulo

Un literal nulo identifica a un puntero nulo, es decir un puntero que no apunta a ningún objeto (similar al lenguaje C). Se indica por medio de la palabra reservada null.

4.5. Tipos de datos

VHDL no proporciona ningún tipo de dato pero si la forma de definirlo. En el paquete Standard de la biblioteca std se encuentran predefinidos los tipos de datos básicos los cuales incluyen números, magnitudes físicas y enumerativas.

4.5.1. Tipo entero

Un dato de tipo entero toma valores de tipo entero en un rango especificado. Los rangos pueden ser ascendentes o descendentes.

Ejemplos:

```
Type   byte is range 0 to 25 5;          −− Declara un tipo byte con
                                         −− Rango ascendente de 0 a 255

Type index   is range 31 downto 0;   −− Declara un tipo index
                                         −− Con rango descendente de 31 a 0
```

Existe un tipo predefinido en la biblioteca std denominado Integer el cual puede asumir un valor dentro el rango -214783648 a 214783647.

Ejemplo:

```
Type integer is range −214783648 to 214783647;
```

4.5.2. Tipo real

Un dato de tipo real se utiliza para representar números reales. Se definen de la misma forma que un tipo entero, con la diferencia que los limites del rango son números reales.

Ejemplo:

```
Type x is range     −7.25 to 1 0. 5;
```

La biblioteca std define el tipo real cuyo rango se extiende desde -1.0E38 hasta 1.0E38.

4.5.3. Tipos físicos

Un tipo de datos físico es un tipo numérico que se puede utilizar para representar alguna magnitud física como por ejemplo tiempo, tensión eléctrica, distancia, etc. La declaración de un tipo físico incluye una magnitud denominada base y un conjunto de subunidades que son múltiplos de la unidad base.

Ejemplo:

```
Type l ong is
unts

mm;
```

cm = 10 mm;

m = 10 cm;

end units;

4.5.4. Tipo enumerado

Un tipo enumerado es un conjunto ordenado de identificadores. Cada miembro de ese conjunto ordenado debe ser distinto dentro de una misma declaración.

Ejemplos:

Type Bit is (0, 1);

Type Boolean is (true, false);

type Hexdigit is (0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ,A,B,C,D, E , F) ;

4.5.5. Tipo arreglo

Un arreglo es una colección homogénea de datos indexados. Los arreglos pueden ser de una sola dimensión o varias. Además pueden ser restringidos, lo que significa que los límites se fijan en el momento de la declaración, o irrestrictos, en cuyo caso los límites se determinan en tiempo de compilación. Un arreglo esta formado por un rango discreto y un tipo de datos que indica el tipo de los elementos por los cuales esta formado el arreglo.

Ejemplos:

Type vector_ int is array (1 to 100) of integer;

Type matriz_ bit is array (integer range <>, i n t e g e r range <>) of bit;

En el primer caso, se define un arreglo de 100 enteros, en el segundo una matriz de bits de rango irrestricto de enteros. Por ejemplo, un objeto puede ser declarado como tipo matriz bit de la siguiente forma:

Matriz: matriz _ bit (1 to 10, 1 to 10);

Lo que declara una matriz de 10 filas por 10 columnas donde cada elemento debe ser tipo bit. La biblioteca std presenta dos tipos arreglo predefinidos irrestrictos, el tipo string y el tipo bit vector:

Type string is array (natural range <>) of character;

Type bit_ vector is array (natural range <>) of bit;

Un elemento de un arreglo puede ubicarse indexando el nombre del arreglo. Si se supone que A y B son arreglos de una y dos dimensiones respectivamente, entonces los objetos indexados A(3) y B(1,2) se refieren a elementos indexados de los arreglos. De hecho, se puede acceder a un subconjunto de elementos de un arreglo utilizando la técnica que se conoce como slicing. Si A es un arreglo de 10 elementos, se pueden obtener los elementos de las posiciones 5 a 8 mediante A (5 to 8). De la misma manera, se pueden asignar valores a las posiciones 5 a 8 utilizando un aggregate. Si se supone que A es un arreglo declarado como:

Type vect is array (1 to 10) of integer;

Y se desean asignar valores a los elementos del mismo, se puede utilizar un agrégate posicional en la que los elementos están listados en el orden en que serán asignados al arreglo de izquierda a derecha:

(1 , 4 , 0 , 0 , 3 , 3 , 0 , 0 , 0 , 0)

Alternativamente, se puede utilizar un aggregate con posicionamiento nombrado que tiene el mismo efecto que el caso anterior. En este caso, el índice de cada elemento se indica explícitamente:

(1 ,4 ,5=>3 ,6=>3, o t h e r s=>0)

4.5.6. Tipo registro

Un registro es una colección de datos heterogéneos. Los datos son nombrados, lo que significa que a diferencia de los arreglos se deben acceder por el nombre del campo. Un campo es un identificador que se asocia con un tipo de datos determinado dentro del registro. Los registros se declaran utilizando la palabra reservada record. Para acceder a los campos de un registro se utiliza el nombre del objeto de tipo registro separado por un punto del nombre del campo.

Ejemplo:

```
Type Persona is record
Nombre: string (1 to 30);
Edad: natural;
End  record;
```

Si se desea acceder al nombre de una persona P se debe hacer P.Nombre. Para asignar valores a los registros, se pueden utilizar aggregates como ocurre con arreglos. Por ejemplo, ("Juan Perez", 28) da valores a los campos de un objeto registro Persona.

4.5.7. Tipo puntero

VHDL soporta la declaración de punteros mediante la palabra reservada access. Estas variables pueden crearse dinámicamente, reservando memoria en tiempo de simulación. Los punteros son una estructura de alto nivel, por lo que se utilizan solo para realizar la descripción algorítmica del sistema y pueden ser utilizados solo con variables.

Ejemplo:

```
Type  pEntero is acces  integer; −− Declara un puntero a integer.
```

La creación de un puntero se hace mediante la palabra clave new y el espacio se libera mediante la sentencia deallocate.

Ejemplos:

```
Ep: pEntero;          −− Declara un puntero integer.
p:= new  integer;     −− Crea el puntero dinámicamente
Deallocate (p);   −− Libera el espacio.
```

4.5.8. Tipo archivo

El tipo de datos file es una estructura de alto nivel de VHDL, la cual es apropiada para almacenar datos durante la simulación o realizar bancos de prueba. Un archivo puede almacenar cualquier tipo de datos. La forma de declarar un archivo difiere entre los distintos estándares de VHDL, tal como se muestra a continuación:

```
Type nombre tipo is  file  of  integer; −− Comun a ambos
File nombre logico: nombre tipo i s [modo]" archivo. Ext"
File  nombre logico: nombre tipo [open modo] i s" archivo. Ext"
```

La segunda declaración corresponde al estándar 87, mientras que la tercera al estándar 93. El nombre lógico de un archivo es el nombre por el cual se lo referencia en el código. En VHDL 87, el modo puede ser in u out, indicando que se va a leer o escribir el archivo. En VHDL 93, el modo puede ser write mode, read mode o append mode. Read mode se toma por omisión. El tipo de datos archivo como tipo de dato abstracto, posee las siguientes operaciones asociadas:

```
Procedure read (variable nombre: inout nombre tipo; valor: out  Integer);
Procedure wr i t e (nombre: inout nombr e t ipo; valor: in  integer);
Function end file (variable nombre: in nombre tipo) return boolean;
```

4.6. Subtipos

Los subtipos son restricciones o subconjuntos de tipos existentes. Existen dos clases de subtipos. La primera clase se utiliza para restringir tipos de un tipo escalar a un rango especifico. La otra clase de subtipos se utiliza para restringir el rango de un tipo arreglo

irrestricto especificando valores para sus índices. Se declaran mediante la palabra reservada subtype.

Ejemplos:

Subtype natural is integer range 0 to HIGHEST INTEGER;
Subtype digitis i s character range 0 to 9;
Subtype mayusculas is character range A to Z;
Subtype nombre is string (1 to 30);
Subtype bus16 is bit vector (15 downto 0);

4.7. Objetos del lenguaje

Un objeto en VHDL es un elemento nombrado que tiene asociado un valor de un tipo de datos especifico. Los objetos en VHDL se clasifican en tres grupos: constantes, variables y señales. Las constantes se declaran mediante la palabra reservada constant, las variables mediante variable y las señales mediante signal. Las variables son elementos secuenciales de alto nivel, por lo que pueden aparecer únicamente dentro de construcciones secuenciales o procedurales. Las variables se utilizan en descripciones algorítmicas y no se pueden mapear en hardware.

Una señal especifica una conexión entre dos entidades, módulos o componentes y representa la interconexión entre los dispositivos físicos de un sistema. Pueden aparecer en construcciones secuenciales o concurrentes. Una señal se declara de la siguiente forma1:

Signal nombre señal: topo señalal [:= valor omi sion] ;

En donde tipo señal es el conjunto de valores que transporta la señal.

Ejemplos:

Constant PI: Real:= 3.1416;
Constant E: Real:= 2.7182;
Variable a: Integer;

```
Variable  b : Real := 0 ;   −− A las  variables  se las puede
                            −− inicializar  con un valor  por  defecto
 signal   bus : Bi t_ v e c t o r (31 downto 0) ;
```

4.8. Operadores

Los operadores se utilizan para combinar expresiones formando expresiones más complejas.

Existen distintos tipos de operadores en VHDL los cuales son mencionados a continuación.

4.8.1. Operadores lógicos

Los operadores lógicos que incluye el lenguaje son: and, or, nand, nor, xor y not, y operan sobre datos de tipo bit, bit vector, boolean, o arreglos de dichos tipos elemento a elemento. Para datos de tipo bit o boolean, los operadores binarios evalúan únicamente la expresión de la derecha si la de la izquierda no alcanza para determinar el resultado.

4.8.2. Operadores de desplazamiento

Operan sobre datos de tipo bit vector. Se utilizan en forma infija, es decir que el objeto a desplazar se ubica a la izquierda del operador y la cantidad a su derecha:

Sll, srl: Desplazamiento lógico a izquierda y a derecha. Desplazan un vector un número de bits a izquierda o derecha, ingresando un cero por el lado opuesto al sentido de desplazamiento.

Sla, sra: Desplazamiento aritmético a izquierda y a derecha. Efectúan el mismo desplazamiento que el caso anterior pero conservando el signo.

Rol, ror: Rotación a izquierda y a derecha. Realiza la rotación del operando a izquierda o derecha. Los bits que se extraen por un extremo, ingresan nuevamente al operando por el lado opuesto.

Ejemplos:

Dato sll 2; −− Desplaza dato dos lugares a la izquierda a
Dato sra 1; −− De plaza dato un lugar hacia la derecha.

−− Conser vando el signo
Dato ror 4; −− Rotandato cuatro veces a la derecha

4.9. Operadores relacionales

Esta clase de operadores requieren datos del mismo tipo y retornan siempre un valor booleano. =,/= Operadores de igualdad y desigualdad. Tienen el significado habitual. <,<=,>,>= Operadores de comparación. Comparan por mayor, menor, mayor o igual y menor o igual, y tienen el significado habitual.

4.10. Operador de concatenación

El operador de concatenación (&) se usa para unir operandos del mismo tipo. Por ejemplo, dados dos vectores A y B, se define el vector concatenación C:= A & B como el vector que posee como primeros elementos a los elementos del vector A y como últimos a los del vector B.

4.11. Operadores aritméticos

Los operadores aritméticos son los mismos que se presentan usualmente en todos los lenguajes de programación. Operan sobre datos de tipo numérico. Los más comunes son:

**	Potencia
Abs ()	valor absoluto
*	Multiplicacion
/	Division
Mod	resto modulo
Rem	resto
+	Suma
-	resta

4.11.1. Atributos

Los tipos y objetos en VHDL pueden tener información asociada denominada atributo. Existe un número predefinido de atributos asociados a un tipo determinado. Si t es un tipo enumerativo, entero, flotante o físico, se definen los siguientes atributos:

t'left - Lıımite izquierdo del tipo t.

t'right - Lıımite derecho del tipo t.

t'low - Lıımite inferior del tipo t.

t'high – Lımite superior del tipo t.

Si x es un miembro de un tipo t como el anterior y n un entero, se definen los siguientes atributos:

t'pos(x) - Posición de x dentro del tipo t.

t'val(n) - Elemento n dentro del tipo t.

t'leftof(x) - Elemento que esta a la izquierda de x en t.

t'rightof(x) - Elemento que esta a la derecha de x en t.

t'pred(x) - Elemento que esta delante de x en t.

t'succ(x) - Elemento que esta detrás de x en t.

Por medio de la palabra clave attribute se puede definir un nuevo atributo:

−− Declaración:
Attribute nombre _atributo: tipo;
−− Especificación:
Attribute nombre_attributo of id_ elemento: clase_elemento is valor;

4.12. Declaraciones básicas

4.12.1. Entidades

Un sistema digital se diseña como un conjunto de componentes interconectados. Cada componente tiene un conjunto de puertos que constituyen su interfaz con el entorno. En VHDL la declaración de un componente se denomina entidad y se especifica como sigue:

Sintaxis Básica de la Declaración de Entidad

```
ENTITY {nombre del dispositivo} IS

PORT (

{lista de puertos de entrada}:        IN {tipo de dato};
{lista de puertos bidireccionales}:  INOUT {tipo de dato};
{lista de puertos de salida}:         OUT {tipo de dato};
{lista de puertos de salida}:         BUFFER {tipo de dato});
END ENTITY;
```

Obviamente, el identificador que sigue a la palabra clave ENTITY es el nombre del dispositivo. Los puertos del dispositivo se especifican, entre paréntesis, en la lista de interfaz de la declaración de entidad, que es el campo señalado por la palabra clave PORT. Para cada puerto hay que declarar:

1. Su nombre: Los nombres de los puertos son etiquetas definibles por el usuario. No pueden utilizarse palabras reservadas del lenguaje.

2. Su dirección: La dirección del puerto se determina a partir de las características del terminal del dispositivo que se desea modelar: los pines de entrada se definen como de tipo IN, los de salida como OUT y los bidireccionales como INOUT.

3. El tipo de datos que maneja el puerto: En los modelos VHDL hay que definir el tipo de datos de los objetos. La elección del tipo de datos es muy importante, pues determina el

conjunto de valores que puede tomar el objeto declarado, así como las operaciones (lógicas, aritméticas y de conversión de tipos) que se le pueden aplicar, cuestión fundamental a la hora de describir el funcionamiento de los dispositivos.

4.12.2. Arquitecturas

Con la declaración de entidad, se especifica únicamente la interfaz del módulo. El comportamiento del módulo puede ser descrito de varias formas en los cuerpos de la arquitectura. Cada arquitectura representa una vista distinta de la entidad. Se puede realizar la descripción de una arquitectura de manera puramente funcional, utilizando técnicas de programación convencionales.

De otro modo se puede implementar una arquitectura como una colección de componentes interconectados de manera estructural. Un cuerpo de arquitectura se especifica como:

Cuerpo de la arquitectura:

ARCHITECTURE {nombre_de_arquitectura} OF {nombre_de entidad} IS
{zona de declaración}
BEGIN
{zona de descripción}
END {nombre de arquitectura};

Las dos etiquetas que forman parte de la cabecera nombran a la propia Arquitectura y a la Declaración de Entidad a la que esta asociada. La primera vuelve a aparecer en la línea que cierra el Cuerpo de Arquitectura. Por lo demás, se distinguen dos áreas para la inclusión de código con dos propósitos distintos:

1. La comprendida entre la cabecera y la palabra BEGIN está destinada a la declaración de objetos que se precisen para realizar la descripción del funcionamiento del dispositivo.

2. Entre la palabra BEGIN y END {arquitectura} se describe el funcionamiento. En el ejemplo de la puerta and no se declara ningún objeto y la descripción del funcionamiento se realiza utilizando una sentencia concurrente de asignación de valor a señal

4.13. Tipo de descripciones

El lenguaje VHDL no solo sirve para la descripción estructural de un circuito sino también para su descripción funcional. En VHDL existen dos aproximaciones a la descripción comportamental de un circuito. Por un lado, se pueden especificar las ecuaciones de transferencia entre diferentes objetos en VHDL. A esta posibilidad de descripción de un circuito se le llama descripción de flujo de datos, o también, refiriéndose al nivel de abstracción, descripción a nivel de transferencia entre registros, conocido por las siglas RTL (Register Transfer Level).

Existe otra forma de describir circuitos en un nivel de abstracción todavía mas elevado. A este nivel se le conoce como descripción comportamental propiamente dicha. Esta segunda posibilidad incluye a la primera y permite al diseñador de circuitos describir la funcionalidad en un nivel de abstracción alto. La diferencia más importante entre un estilo de descripción y el otro, es que la ejecución, o interpretación de sentencias en el nivel RTL es concurrente, es decir, las sentencias indican conexiones o leyes que se cumplen y por lo tanto es como si se ejecutaran continuamente.

4.14. Estructuras de la ejecución concurrente RTL

La instrucción básica de la ejecución concurrente es la asignación entre señales que es gobernada por el operador de asignación (<=). Para facilitar la tarea de realizar asignaciones algo complejas, VHDL introduce algunos elementos de alto nivel como son instrucciones condicionales, de selección, etc. Se verán a continuación otras estructuras propias de la ejecución concurrente o descripción RTL.

4.14.1. Asignación condicional: when...else

Es importante, en toda expresión condicional que describa hardware de forma concurrente, incluir todas las opciones posibles y contemplar todos los casos posibles de variación de una variable.

En este sentido es obligatorio siempre acabar esta expresión condicional con un else. Se pueden anidar varias condiciones en una misma asignación.

Ejemplo:

```
s <= ’1’ when a = b else
’0’ when a > b else
’X’;
```

4.14.2. Asignación con selección: with...select...when

La asignación se hace según el contenido de cierto objeto o resultado de cierta expresión. Es una ampliación de la asignación condicional.

Ejemplo:

```
With estado  select
      Semáforo <=” rojo” when”01”,
        ” verde ” when ”10” ,
        ” amarillo ” when ”11” ,
        ”no funciona ” when  others ;
```

Es obligatorio, al igual que ocurre con la asignación condicional, incluir todos los posibles valores que pueda tomar la expresión. Por lo tanto, si no se especifican todos los valores en las cláusulas when, hay que incluir la cláusula when others. De lo contrario la expresión podría tomar valores frente a los cuales no se sabe que respuesta dar.

4.14.3. Bloque concurrente

En muchas ocasiones es interesante agrupar sentencias de ejecución concurrente en bloques. Estos bloques son el mecanismo que tiene VHDL para la realización de diseños modulares.

Dichos bloques permiten además subdividir un mismo diseño en una jerarquía de módulos, ya que estos pueden estar unos dentro de otros. Los bloques están definidos dentro de la arquitectura en entornos de ejecución concurrente y, de alguna manera, son equivalentes a entidades, ya que se les puede definir entradas y salidas, aunque quizás su uso más común es el agrupamiento de instrucciones para separar el diseño en módulos. La estructura general de la declaración de bloque se muestra a continuación:

```
block _i d: block (expresion  de vigilancia )
[ cabecera ]
[ declaraciones ]
Begin
[Sentencias   concurrentes]
End block   block_ i d ;
```

El nombre block id es opcional y su uso sirve para nombrar a diferentes bloques en un mismo diseño y así ayudar en la depuración, simulación y sobre todo en la legibilidad del programa.

La cabecera puede tener cláusulas de tipo genérico, declaraciones de puertos de entrada, salida, etc., es decir, es equivalente a la declaración de entidad y su alcance es dentro del bloque.

4.15. Descripción estructural

VHDL dispone de diferentes mecanismos de descripción estructural:

Descripción de componentes.

Enlace entre componentes y entidades.

Definición de señales

Referencia de componentes

Configuración

4.15.1. Componentes

Normalmente, un componente se corresponde con una entidad declarada en algún sitio dentro del diseño. Por lo tanto es necesario enlazar el componente con la entidad correspondiente. Esto generalmente se hace dentro de una biblioteca donde se declaran todos los componentes del sistema.

La declaración de un componente se realiza de la siguiente forma:

Component nombre componente is

[Generic lista parametros]

[Port l ista puertos]

End component [nombre componente];

El nombre del componente, identifica al mismo, lista parámetros indica los parámetros genéricos que pueden ser pasados al componente y lista puertos hace referencia a las señales de entrada y salida que tiene el componente. La instanciación de un componente dentro de una arquitectura se realiza de la siguiente forma:

inst_ id : [component] { nombre_componente | entity nombre_entidad [(nombre_a rquitectura)] | configuración nombre_ configuración } [generic map (pa r´ame t ros)]

[por t map(puertos)] ;

4.15.2. Enlace entre componentes y entidades:

Especificación de configuración

Un componente especifica una interfaz sin ninguna funcionalidad asociada. Para que tenga sentido, debe ser asociado con una entidad y una arquitectura, lo cual se realiza a través de la sentencia for:

for: lista_ inst: nombre componente

use entity nombre _ent idad (n ombre_ arquitectura) |

use configuration nombre_ configuración

[generic map (parámetros)]

[port map (puertos) ;

Las instancias del componente nombre componente se especifican por medio de una lista, la cual es indicada por lista inst. En esta lista se pueden usar las palabras reservadas all u others para seleccionar todas las instancias del componente o todas las que queden por enlazar con una entidad, respectivamente.

La directiva use permite dos posibilidades. Lo normal es enlazar una entidad especificando opcionalmente la arquitectura que se desea enlazar si posee más de una. La otra posibilidad es enlazar directamente con un bloque de configuración donde estará especificada la entidad a enlazar.

4.15.3. Repetición de estructuras

En los casos donde un sistema esté compuesto por estructuras repetitivas, la sentencia generate provee la capacidad de describir estructuras regulares como vectores de bloques, procesos o instancias de componentes dentro del cuerpo de una arquitectura. Su utilización es la siguiente:

```
Etiqueta_generate:
for  rango_generate  generate
[ i f   condici o n gene rat e
{ instancia_ de_ component  |
      bloque |
      proceso}
     end i f ]
       { instancia_de_ componente |
          bloque |
           proceso}
      end  generate
```

El rango generate describe la cantidad de veces que el esquema se va a generar y puede depender de una variable que puede ser utilizada en los esquemas. La sentencia if se utiliza para casos excepcionales dentro de los patrones de repetición.

4.15.4. La unidad de configuración

Hasta ahora se ha puesto de manifiesto la importancia que tiene el enlazar un componente con una entidad, y por lo tanto, con una arquitectura específica dentro de la entidad. Esto se llama especificación de configuración. Sin embargo, existe otra posibilidad de enlazar componentes y entidades, y es lo que se conoce como declaración de configuración. La declaración de configuración se realiza en un bloque especial de VHDL denominado Unidad de Configuración. En este bloque se pueden especificar los mismos enlaces de componentes con entidades tal como ocurre en la especificación de configuración. La forma general de una declaración de configuración es la siguiente:

```
Configuration  nombre _configuración  of  nombre_entidad  is

{Sentencia_ use |

definición _atributo|

definición _grupos}
```

bloque_ configuración

End nombre_ configuracion;

La unidad de configuración está siempre asociada a una entidad. Todo el contenido de una configuración se refiere a la entidad indicada por nombre entidad.

La definición de grupos se utiliza para definir una colección de elementos mediante la palabra reservada group. Se utiliza normalmente para pasar información acerca de una entidad a la herramienta de síntesis del circuito si esta lo requiere.

El bloque configuración es la única parte obligatoria dentro del cuerpo de una unidad de configuración. Se refiere a las posibles estructuras presentes en una arquitectura.

4.16. Palabras reservadas

Las siguientes son las palabras reservadas en VHDL. No deberán utilizarse como nombres de ningún tipo que no sea como instrucciones de VHDL.

abs	else	nand	return
access	elsif	new	select
after	end	next	severity
alias	entity	nor	signal
all	exit	not	subtype
and	file	null	then
architecture	for	of	to
array	function	on	transoprt
asser	generate	open	type
attribute	generic	or	units
begin	guarded	others	until
block	if	out	use
body	in	package	variable
buffer	inout	port	wait
bus	is	procedure	when
case	label	process	while
component	library	range	with
configuration	linkage	record	xor

constant	loop	register
disconnect	map	rem
downto	mod	report

4.17. Símbolos Especiales

Algunos símbolos especiales usados por VHDL y ejemplos se dan en la siguiente tabla.

Los operadores pueden sobrecargarse y tener diferentes significados para diferentes tipos en diferentes librerías.

Símbolos especiales de VHDL

Símbolo	**Significado**	**Ejemplos de uso**
“	Agrupa valores de bits Forma cadenas de texto	std_logic_vector := “001000” “Una cadena” “una cadena con “”comillas””.”
#	Divisor para números en base 2 a 16	2#1001000# -- número binario 16#af125f# -- número hexadecimal
&	Operador de concatenación	“001” & “111” -- equivalente a “001111”
‘ (comilla simple)	Atributos Calificación de tipo ambiguo Caracteres Valores de dígitos de bit	Clock’event A’left B’high unsigned’(“10001”) ‘A’ ’c’ ‘ ’ ‘0’ ‘1’ ‘Z’ (alta impedancia)
( )	Subprogramas, para agrupar operadores y dar prioridad en expresiones, etc.	function () (A + B) * 4
*	Multiplicación	A := A * 4;
+	Suma o identidad	A + B Cuenta_REG + 1
-	Resta o negación	A – B Cuenta_REG – 1
, (coma)	Separa listas de parámetros	port map(A => A, B => B)

. (punto)	Punto decimal Jerarquía en librerías	0.5 2#0.100# use ieee.std_logic_1164.all
/	División	A **:=** A / 4;
:	Especificación de tipo	signal A : std_logic; constant C : integer;
;	Fin de instrucción	A_REG <= ‘1’**;**
<	Menor en comparaciones	**if** (A < B) **then**
=	Igual en comparaciones	**if** (A = ‘0’) **then**
>	Mayor en comparaciones	**if** (A > 0) **then**
[ ]		
| (barra vertical)	Múltiples opciones en una condición	**when** 1 | 2 | 3 => -- Cuando la expresión sea -- 1 o 2 o 3
=>	Opciones case, when Mapeo de puertos en arquitecturas estructurales Para dar valor a bits no asignados de vectores	**when** “000” => **port map**(A => A, B => B) (**others** => ‘0’)
**	Exponencial	
:=	Asignación para variables	VariableA := 1
/=	No igual en comparaciones	**if** (A /= ‘0’) **then**
>=	Mayor o igual en comparaciones	**if** (A >= 2) **then**
<=	Asignación para señales Menor o igual en comparaciones	A <= ‘1’; -- A toma el valor ‘1’ **if** (A <= 3) **then**
<>	Usado para indicar rango indefinidos	(natural **range** <>)

\	Usado para definir identificadores extendidos	\C:\\Xilinx\CarpetaProyecto\
_ (Guión bajo)	Usado para separar números o cadenas de bits largas	124_321 2#0110_1100# b"0100_0110" X"12FD_C46B_4567"

Tipos más utilizados en VHDL

Tipo	Comentario
bolean	Tipo booleano, toma los valores TRUE o FALSE
carácter	Contiene todos los caracteres ISO de 8 bits
integer	Valores enteros. Como mínimo deben incluir los valores desde el $(- 2^{31} + 1)$ hasta el $(2^{31}-1)$
real	Usado para representar valores fraccionarios
time	Usado para controlar tiempo en los modelos. Especialmente importante durante las simulaciones.
std_logic	Usada para señales
std_logic_vector	Definido a partir de std_logic, permite tener señales de un ancho de varios bits

4.18. Paquetes de Lógica Estándar del IEEE

El lenguaje VHDL tiene una gran capacidad para extender los tipos y definir y sobrecargar operadores y funciones. Para estandarizar el IEEE desarrolló un paquete lógico estándar denominado **std_logic_1164**. Aunque no es realmente parte de VHDL, este paquete es tan utilizado que casi podría considerarse un aparte del lenguaje. Se recomienda siempre usar los tipos definidos en estos paquetes.

Además de definir el paquete **std_logic_1164**, el IEEE definió un paquete con operaciones aritméticas para números representados por vectores de bits. Este paquete se denomina **numeric_std** y es el recomendado para diseños sintetizables.

Para utilizar la lógica 1164 del IEEE o los paquetes numéricos estándar con las operaciones aritméticas sobre la lógica estándar se debe agregar el uso de los paquetes adecuados.

-- Utilización de los paquetes lógicos estándar del IEEE
library ieee;
use ieee.std_logic_1164.**all**; -- Lógica estándar
use ieee.numeric_std.**all**; -- Operaciones aritméticas
Ej: Uso de los paquetes estándar IEEE

- Lógica estándar std_logic y std_ulogic definidos en std_logic_1164

El paquete std_logic_1164 dentro de la librería ieee define un tipo **std_logic** con nueve valores posibles.

Valores posibles para lógica std_logic

Valor	Descripción	Valor	Descripción
'U'	Sin inicializar	**'W'**	Desconocido débil
'X'	Desconocido forzado	**'L'**	0 débil
'0'	0 forzado	**'H'**	1 débil
'1'	1 forzado	**'-'**	No importa
'Z'	alta impedancia		

A partir de esta lógica también se define el subtipo **std_logic**. Este tipo es lo que se denomina lógica resuelta ya que incluye funciones que definen que pasa cuando varias salidas se conectan a una señal.

Los valores débiles y forzados se incluyen para poder modelar hardware que tiene diferentes capacidades de controlar una señal. Por ejemplo, si una señal tiene un buffer de tres estados que la maneja (posibles valores '0', '1' y 'Z') y además una resistencia de "pull-up" a la tensión de alimentación con un valor alto (valor 'H'). Si la salida del buffer es alta impedancia ('Z') el valor de la señal estará dado por la resistencia, pero si el buffer tiene una salida lógica, esta forzará el valor de la señal más allá de la resistencia de "pull-up".

Resumen de tipos de las operaciones aritméticas

Oper.	Operación	Tipo del operando izquierdo	Tipo del operando derecho	Tipo del resultado	Longitud del resultado
abs -	valor absoluto negación		signed	signed	Longitud del operando
+ - * / rem mod	suma resta multiplicación división resto módulo	unsigned signed unsigned natural signed integer	unsigned signed natural unsigned integer signed	unsigned signed unsigned unsigned signed signed	La mas grande de los dos operandos Suma de las longitudes Longitud del izquierdo Longitud de operando derecho
= /= < <= > >=	igualdad diferente menor menor o igual mayor mayor o igual	unsigned signed unsigned natural signed integer	unsigned signed natural unsigned integer signed	boolean boolean boolean boolean boolean boolean	
sll	corrimientos	unsigned	integer	unsigned	Longitud del operando

srl rol ror	lógicos rotaciones	signed	integer	signed	del tipo vector de bits
not	negación		unsigned signed	unsigned signed	Longitud del operando
and or nand nor xor xnor	operaciones lógicas	unsigned signed	unsigned signed	unsigned signed	Longitud de los operandos. Ambos deben ser de igual longitud.

V Diseño Metodológico

5.1 Tipo de Estudio o Investigación:

La investigación es un proceso metódico y sistemático dirigido a la solución de problemas o preguntas científicas, mediante la producción de nuevos conocimientos, los cuales constituyen la solución o respuesta a tales interrogantes. La investigación puede ser de varios tipos se puede clasificar de distintas maneras, sin embargo es común hacerlo en función de su nivel, su diseño y su propósito.

Esta investigación se caracteriza por ser **Exploratoria y descriptiva**. Es exploratoria porque se efectúa sobre un tema u objeto desconocido o poco estudiado, en este caso Logica Programable y el Lenguaje VHDL; por lo que sus resultados constituyen una visión aproximada de dicho objeto, es decir, un nivel superficial de conocimiento. Las investigaciones exploratorias son útiles por cuanto sirve para familiarizar al investigador con un objeto o tema, que hasta el momento le era totalmente desconocido, sirve como base para la posterior realización de una investigación descriptiva.

Investigación Descriptiva: consiste en la caracterización de un hecho, fenómeno, individuo o grupo, con el fin de establecer su estructura o comportamiento.

5.2. Recopilación de la información

En la primera etapa del desarrollo del trabajo asignado se procedió a realizar la documentación, a través de libros e Internet, la cual se basó en la recopilación de información sobre los dispositivos de lógica programable, los microprocesadores, lenguajes de descripción y programación, y el software xilinx y su simulador modelsim que va ser utilizado, para finalmente realizar una evaluación y selección de las herramientas más adecuadas para este proyecto.

Se recopiló información acerca del lenguaje de descripción de hardware VHDL, el cual fue escogido desde el inicio del proyecto para el diseño de la ALU.

Posteriormente se realizó una etapa de estudio; donde se analiza y selecciona la información para luego comenzar a realizar el desarrollo del trabajo; aquí se definieron los objetivos, se redacto la introducción y el resto del contenido del trabajo.

Una vez que se obtuvo el software se procedió a instalarlo y realizar muchas pruebas como ejemplos de una ALU pequeña para familiarizarse con su plataforma, simulador y sintaxis del lenguaje, aprendiendo así su funcionamiento.

5.3 Herramientas

El comportamiento de cada elemento se diseñó mediante *HDL* y el Schematic Editor, módulo del lenguajeVHDL. Familia spartan3, dispositivo *XC3S200* de Xilinx versión 9.1c. Durante la etapa de diseño, para comprobar que la descripción que se ha hecho es correcta, es necesario verificar la sintaxis de este archivo. Enseguida, para verificar su funcionamiento se sintetiza el diseño.

Se utilizaron las herramientas de Microsoft office Vicio 2003, Word 2003, PowerPoint 2003.

Se decidió seleccionar el lenguaje VHDL porque es un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción y reduce el ciclo de diseño además es Portable; el mismo código VHDL puede ser simulado y usado en las herramientas de las diferentes etapas del proceso de diseño.

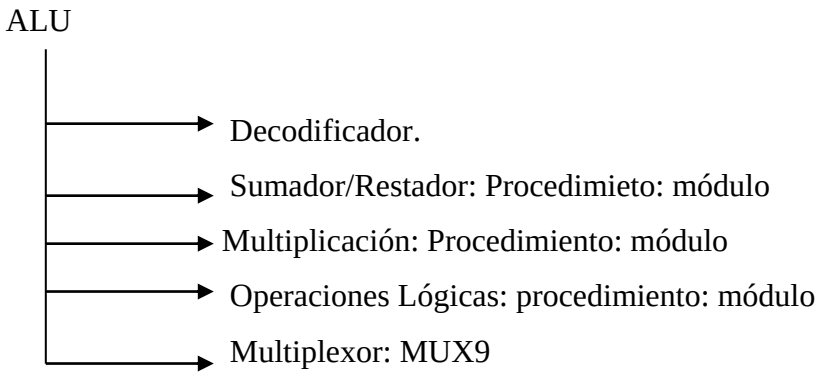
El lenguaje VHDL es un lenguaje de descripción que especifica los circuitos electrónicos en un formato adecuado para ser interpretado tanto por máquinas como por personas. Se trata además de un lenguaje formal, es decir, no resulta ambiguo a la hora de expresar el comportamiento o representar la estructura de un circuito.

Está normalizado, o sea, existe un único modelo para el lenguaje, cuya utilización está abierta a cualquier grupo que quiera desarrollar herramientas basadas en dicho modelo, garantizando su compatibilidad con cualquier otra herramienta que respete las indicaciones especificadas en la norma oficial.

5.4 Diseño de los Módulos o circuitos.

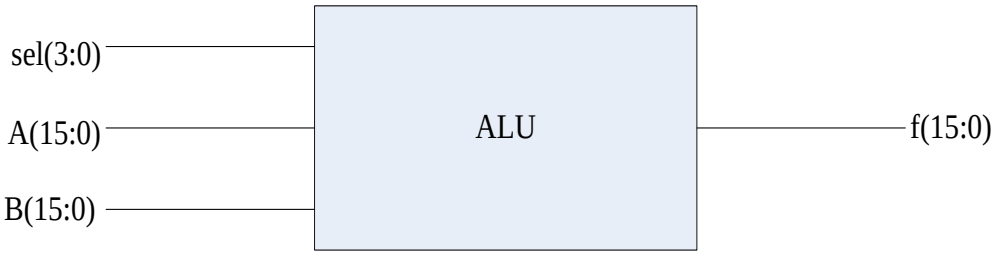
La ALU fue descrita en el lenguaje de descripción VHDL, en el anexo se observa el código de descripción de esta aplicación. La estructura es una entidad llamada "alu3.vhd" integra todas las unidades o módulos de la ALU, presta la conectividad de las señales externas hacia las unidades funcionales y sirve de interfaz hacia el sistema usuario.

La figura muestra un diagrama jerárquico de los componentes en la ALU.



5.4.1 Diseño Físico de la Alu.

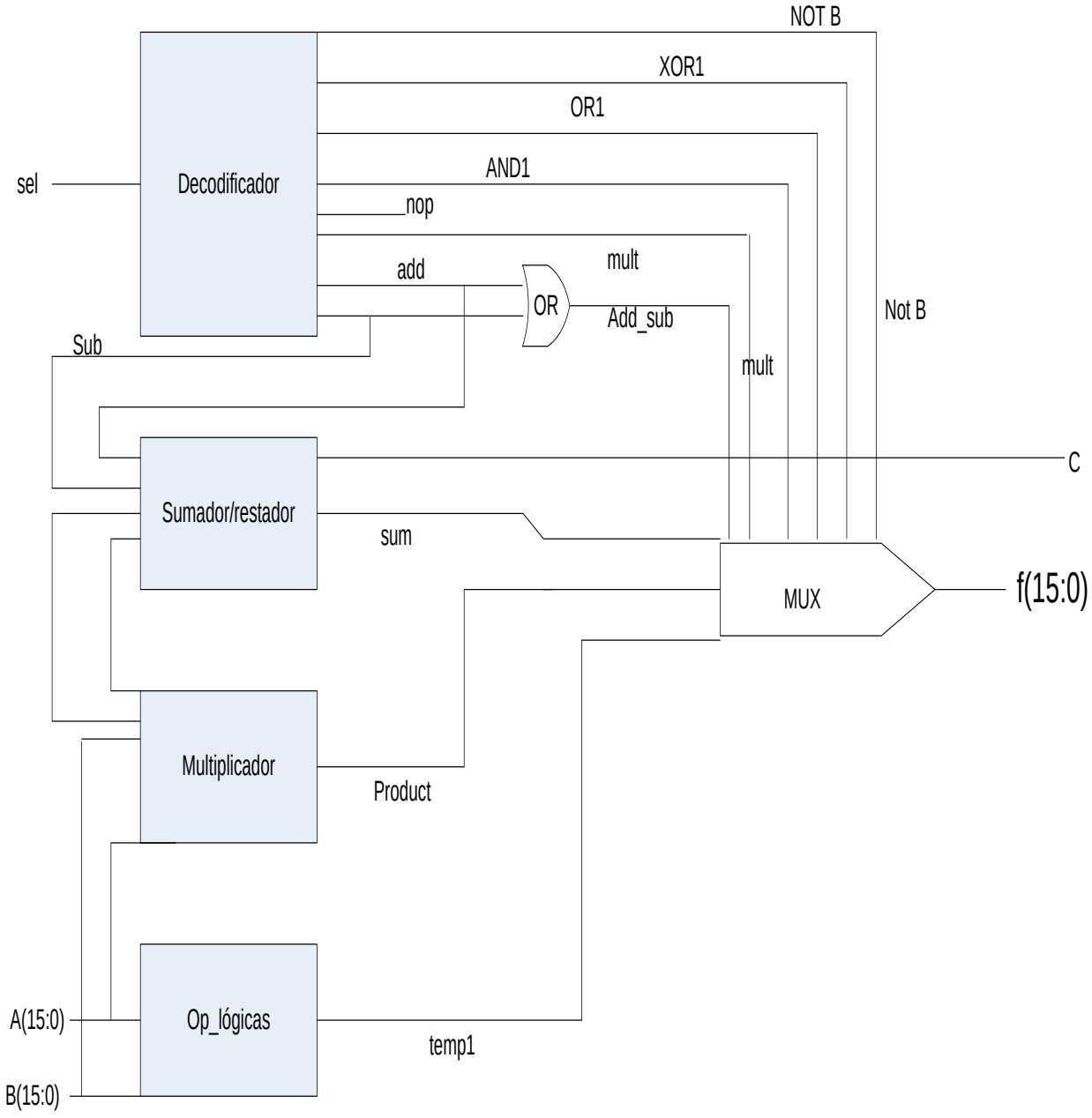
Estructura física de la ALU



La ALU tiene tres entradas generales: una llamada **Sel** que sirve para selección del tipo de operación, los operandos de entrada, son dos (**A, B**), presenta la variable **f** que mostrará el resultado final de la operación en curso.

Como ya se mencionó antes la ALU contiene todos los módulos descritos anteriormente, para unificar el trabajo se realizó a través de componentes que es una herramienta de VHDL.

Estructura del diseño interno de la ALU



Esta estructura tiene 4 módulos y dos entradas de datos (A y B) de 16 bits. El primer módulo es el **decodificador** presenta una sola variable de entrada **(sel**) que contiene el código del tipo de operación que puede realizar la ALU y presenta 6 líneas de salida que llegan al multiplexor cada linea representa una operación diferente.

El segundo módulo es un **sumador/restador** que se conecta al multiplexor por medio de la variable sum, esta tendrá el resultado del sumador/restador y se usa la compuerta OR para conectarse al decodificador y con la variable add_sub se activa el sumador/restador envíando la señal de operación (suma o resta) al multiplexor. Notese que sólo para el sumador/restador se hace uso de la variable **c** para el acarreo.

El **módulo multiplicador** usa la variable product para almacenar temporalmente el resultado del producto y conectarse al multipxeor.

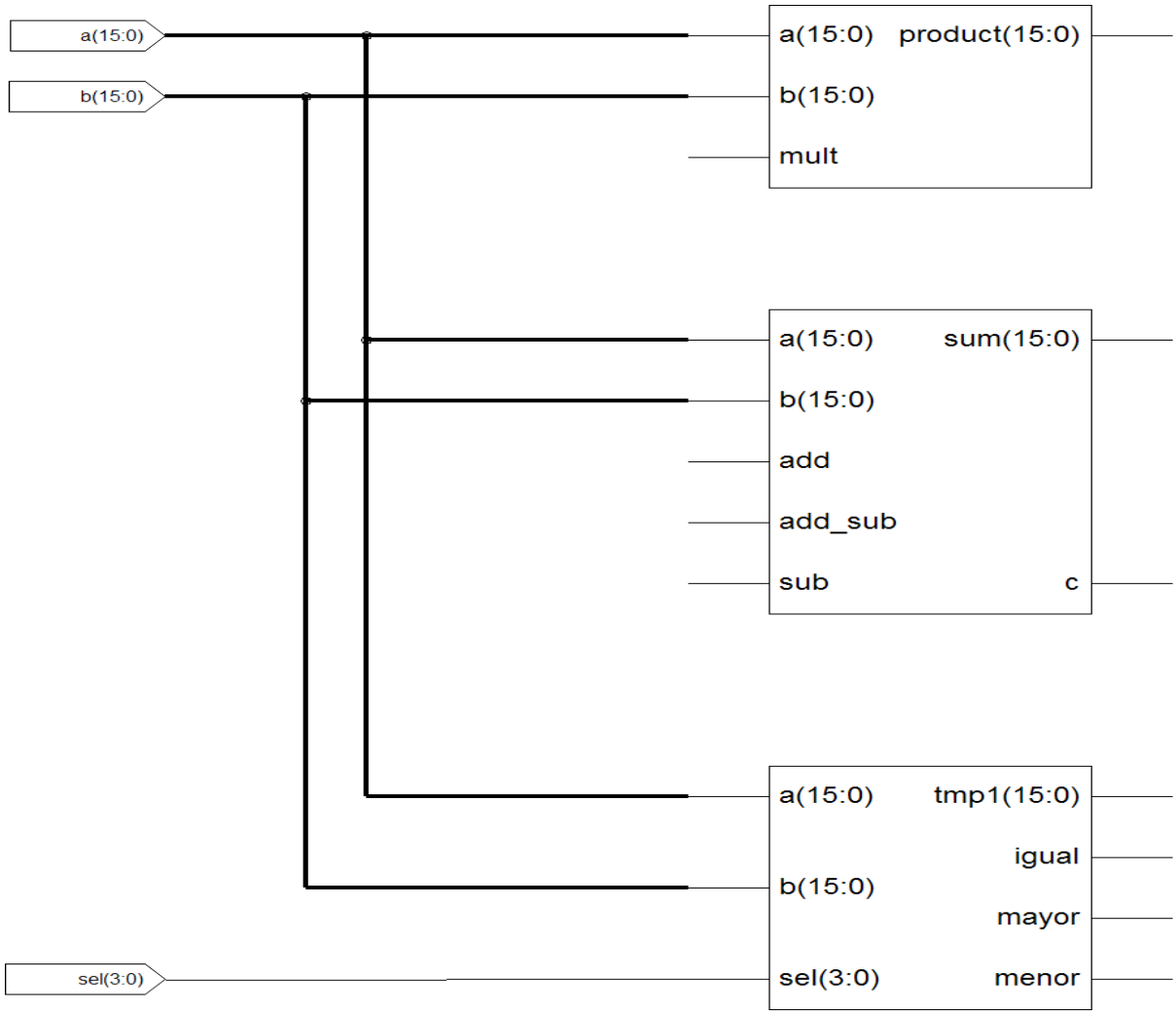
El módulo para las **operaciones lógicas** usa la variable temp1, para conectarse al multiplexor y almacena el resultado temporalmente.

Finalmente muestra el **multiplexor:** acepta N entradas en este caso son nueve y presenta una salida representada por la variable " **f** " en la que aparecerá el valor lógico asociado a una de estas entradas. La selección de cuál de esas entradas es la que aparece en la salida se determina por un conjunto de M señales de control, cumpliéndose que $N = 2M$. La salida presentará aquella entrada cuyo índice concuerda con el número codificado en binario en las líneas de control (o selección).

Se realizaron numerosos cambios en el diseño con el propósito de poder presentar las operaciones lógicas. Cada modificación en el código o en la estructura de los dispositivos hizo necesario repetir el ciclo de simulación funcional.

En la **figura_módulos** se puede observar los tres módulos de la ALU, se puede notar como están conectados entre sí el primer módulo es el multiplicador segundo es el sumador/restador y por ultimo el de las operaciones lógicas. Y todos envían el resultado de la operación a través de sus variables de salida al multiplexor.

Figura_módulos



5.5 Descripción de la aplicación.

Se diseñaron cinco módulos empezando a programar con el lenguaje VHDL. Hay una sola posibilidad de introducir los datos y esta, es en forma Binaria; La información de los datos son enviados a la ALU y el código de operación, lo envía el decodificador a los módulos de la aplicación enviando una señal del tipo de operación a realizar.

Cuando se obtiene respuesta a una operación determinada, la interfaz muestra el resultado, a través del multiplexor de la unidad aritmética lógica.

Decodificador: Un decodificador es un circuito lógico con N entradas y dos elevado a la n salida que funciona de forma tal que en cada instante se encuentra activa la salida correspondiente con la codificación binaria codificada en la entrada.

El decodificador selecciona el código de cada operación que se realizara y a través de una señal de activación de entrada la envía a los módulos de suma/resta, multiplicador, multiplexor y al módulo de las operaciones lógicas, funciona como una unidad de control que dice que tipo de operación realizará la ALU; las operaciones que puede realizar la ALU son suma, resta, multiplicación y operaciones Lógicas (AND, OR, XOR, NOT B, y comparaciones entre dos datos.)

Este circuito tiene una sola variable de entrada llamada **sel** que es la del código de operación y este envía la señal de la operación seleccionada por el usuario, al módulo correspondiente; y presenta 8 variables de salida **add**, **and1, mult, nop, notb, or1, sub xor1** de todas estas variables solo una se activa en cada salida.

Acontinuación su algoritmo y estructura:

Algorimo_decodificador

{Variables

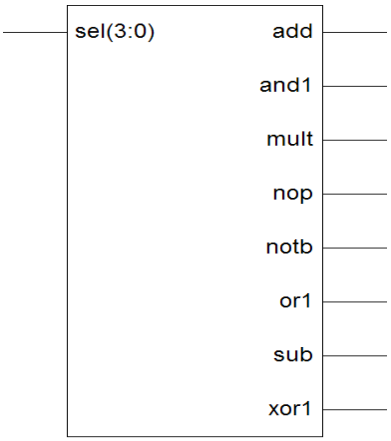
Variable de entrada: sel para indicar el código de operación.

Variables de salida: xor1, sub, or1, notb, nop, mult, and1, add.

Mult: multiplicación, nop: activa la no operación, add: para la suma, sub: para la resta.

And1, Xor, OR, notb: para las operaciones Logicas.}

Decodoficador



Inicio
Leer sel
Si sel = “0001” Entonces
add ⟵ ‘1’;
sub ⟵ 0;
mult ⟵ 0;
nop ⟵ 0;
xor1 ⟵ 0;
notb ⟵ 0;
or1 ⟵ 0;
De lo contrario
Sel =” 0000”
Sub ⟵ ‘1’;
add ⟵ 0;
mult ⟵ 0;
nop ⟵ 0;
xor1 ⟵ 0;
notb ⟵ 0;
or1 ⟵ 0;
De lo contrario
Sel = “0100”
mult ⟵ ‘1’;
sub ⟵ 0;
add ⟵ 0;
nop ⟵ 0;
xor1 ⟵ 0;
notb ⟵ 0;
or1 ⟵ 0;
De lo contrario
Sel = “0110”
nop ⟵ ‘1’;
sub ⟵ 0;
mult ⟵ 0;
add ⟵ 0;
xor1 ⟵ 0;
notb ⟵ 0;
or1 ⟵ 0;
Fin del si;
Fin del algoritmo.

En el algoritmo solo se incluyeron las operaciones aritméticas, para mayor comprensión de la selección se presenta a continuación abajo una tabla con el menú de las operaciones que realiza y el código asignado a cada operación.

Códigos de operación

Menú de operaciones de la ALU	
Selección	**función**
Sel= 0001	Suma
Sel= 0000	Resta
Sel= 0100	Multiplicación
Sel= 0110	No operación
Sel= 0010	AND
Sel= 0011	OR
Sel= 0111	XOR
Sel= 1111	A NOT B

Un sumador /restador: llamado Adder, tiene dos operandos de entrada (a y b) cada uno de 16 bits, para realizar las operaciones aritméticas (suma o resta) se utiliza la variable de entrada **add** para habilitar la suma, y la variable **sub** para habilitar la resta y la variable **add_sub** para activar el sumador/restador y para conectarse al multiplexor. Para realizar el cálculo de la suma y resta se utilizo compuertas XOR.

La programación de este módulo no es tan sencilla, ya que al calcular la resta se tiene que realizar el complemento a uno y luego el complemento a dos.

En el diseño de este operador sumador/restador, se utilizan la técnica de complemento a uno para la resta, añadiendo el acarreo final al resultado de la suma del minuendo con el complemento a uno del sustraendo.

Para mostrar la salida del sumador/restador se utiliza la variable sum tanto para la suma como para la resta y para propagar el acarreo se usan las variables **c** y **carry**.

A continuación su algoritmo y estructura:

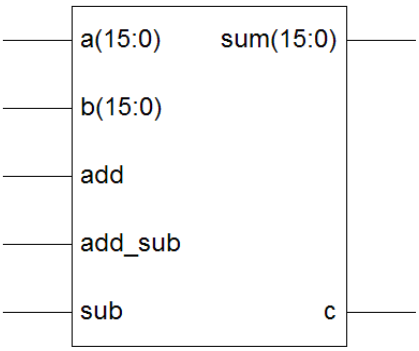
Algoritmo_Sumador/restador:

{Variables:

Variables de entrada: a y b que representan los operandos de tipo vector de 16 bits, add, sub, add_sub son variables de entrada se habilita según la operación ya sea suma o resta. sum_reg, temp_reg: variable temporales de tipo arreglo, i: representan los índices del arreglo, carry: variable para almacenar el acarreo.

Variable de salida: sum almacena el resultado final de la suma o resta, c: almacena el resultado de acarreo.}

Módulo Sumador/Restador




```
Inicio
Leer a, b
sum_reg ⟵ [0...15];
temp_reg ⟵ [0...15];
    Si (add='1') y (sub='0') entonces
        Carry ⟵'0';
        For i ⟵ 0 hasta 15 hacer
            sum_reg⟵ a(i) xor b(i) xor carry;
            carry⟵ (a(i) and b(i)) or (a(i) and carry) or( b(i) and carry);
        fin de para;
       sum ⟵ sum_reg
        c ⟵ carry;
  De lo contrario
      (Add = '0') y (sub = '1') entonces
          tmp_reg ⟵"1111111111111111";
          tmp_reg ⟵ b xor tmp_reg;
          carry ⟵ '1';
      For i ⟵ 0 hasta 15 hacer
```

sum_reg (i)⟵ a (i) xor tmp_reg(i) xor carry;
carry⟵ (a (i) y tmp_reg(i)) or ((a(i) y carry)) or ((tmp_reg(i) y carry));
fin de si
sum ⟵ sum_reg;
c ⟵ carry;
fin del si
Fin del Algoritmo.

Multiplicador: Se diseño para elaborar la operación de multiplicación, tiene dos operandos de entrada (a y b) de 16 bits cada una, se usa un registro temporal del producto parcial de 18 bits, otro registro para la suma de 9 bits, y otro registro para los primeros 9 bits del primer operando este módulo es uno de los más complicados de programar, ya que en la multiplicación se realiza la suma binaria para obtener el resultado de la operación.

Se programó utilizando compuertas lógicas XOR, AND, y se diseño un multiplicador de 8 bits, donde se divide el primer y segundo operando en 2, teniendo 8 bits para el primer y segundo operando esto se hace en la programación, se hacen una serie de asignación de bits, tanto para el arreglo parcial del producto, como para el arreglo de la suma, y para calcular el resultado y obtener el producto final, la variable **product** almacena temporalmente el resultado.

Su algoritmo y estructura:

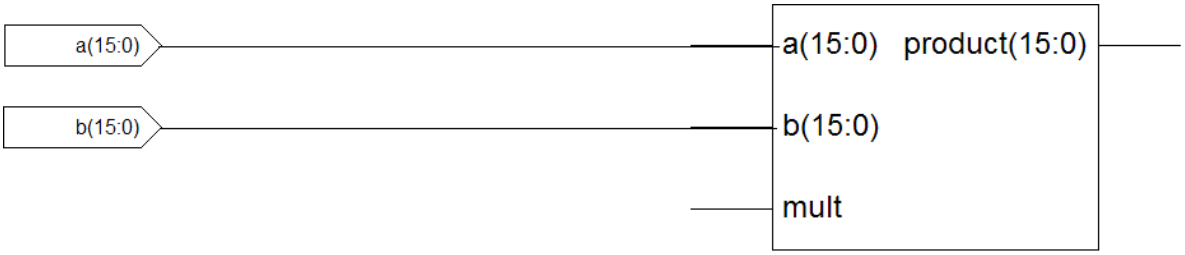
Algoritmo_ Multiplicación

{Variables de entrada: a y b de 16 bits, que representan los operandos y mult para cativar la operación.

Variable de salida: product.

Variables temporales: a_reg, product_reg, carry_reg: variable de tipo arreglo, i: representan los índices del arreglo.}

Módulo Multiplicador



```
Inicio
Leer  a y b
1. Si mult   =  '1' entonces
      a_reg  ←—— '0'  y  a (7… 0);
      product_reg ←——  "0000000000"  y  b (7... 0)
      psum  ←—— "000000000"
      carry_reg  ←——  '0'

    1.1    for  j  ←——    1 hasta  9  hacer
             1.1.1   Si  product_reg (0)  = '1' entonces
               1.1.1.1   for i  ←—— 0 hasta 8 hacer
                        psum (i) ←——    product_reg (i+9) xor a_reg (i) xor carry_reg
                       carry_reg  ←——(product_reg (i+9) y a_reg (i)) or (product_reg (i+9) y
                        carry_reg) or (a_reg (i) y carry_reg)
                      fin del for  de i

                 1.1.1.2  product_reg (17... 9)  ←——   psum (8... 0)
                 1.1.1.3    carry_reg←———  '0'
               fin del si
        1.2  product_reg (17... 0)  ←—— '0' y  product_reg (17... 1);

   Fin de for de j
    product ←——   product_reg (15... 0)
Fin del si
Fin del Algoritmo.
```

Opera raciones Lógicas: En este módulo se uso dos variables de entradas (a y b) cada una de 16 bits, calcula operaciones AND, XOR, OR, NOT B.

Opera raciones de comparación: En el operador lógico XOR además de efectuar su función exclusiva, también asume la función de comparación. Este elemento de diseño puede verificar si dos cantidades de 16 bits son iguales, mayor o menor que la otra, para ello se usan las variables **menor, mayor e igual**. Al introducir dos cantidades iguales la variable igual tomará el valor de uno, y si el primer operando es mayor que el segundo operando, la variable mayor tendrá el valor de uno, de lo contrario la variable menor tomara el valor de uno.

El comparador utilizado realiza una relación aritmética (mayor, menor o igual) entre las palabras. Estos dispositivos se denominan comparadores de magnitud.

A continuación su algoritmo:

Algoritmo operaciones _ lógicas y su estructura.

{Variables:

Variables de entrada: a y b de 16 bit cada una, sel, es la variable que me indica el tipo de operación.

Variables de salida: mayor, menor, igual son variables que se usan para hacer comparaciones aritméticas, temp1 es la que almacena temporalmente el resultado.}

Módulo Operaciones Lógicas

Inicio
Leer a y b
En caso sel de
cuando sea “0111”
tmp1 ← a xor b
Si (a < b) Entonces
menor ← '1'
igual ← '0'
mayor ← '0'
De lo contrario
(a=b)
menor ← '0'
igual ← '1'
mayor ← '0'
De lo contrario
menor ← '0';
igual ← '0'
mayor ← '1'
Fin del Si
cuando sea "1111"
tmp1 ← a + (not b) + 1
menor ← '0'
igual ← '0'

```
            mayor ←—— '0'
      cuando sea  "0010"
            tmp1 ←——  a and b
            menor ←—— '0'
            igual ←——  '0'
           mayor←——  '0'
      cuando sea  "0011"
          tmp1←——  a or b;
          menor ←—— '0'
           igual ←—— '0'
         mayor ←—— '0'
      cuando otra operación   ←—— nulo
   fin  case
Fin del Algoritmo.
```

Multiplexor: Módulo que contiene las señales de la multiplicación, suma y resta, operaciones lógicas; tiene 9 entradas con las variables (**product**, para la multiplicación, sum, para mostrar el resultado de la suma o resta, **tmp1** para las operaciones lógicas, **add_sub**, se utiliza para activar el sumador/restador, **mult** para activar la señal de la multiplicación), también para las operaciones lógicas presenta **notb, xor1, or1, and1 y nop**. La variable **f** muestra la salida del resultado final de todas las operaciones.

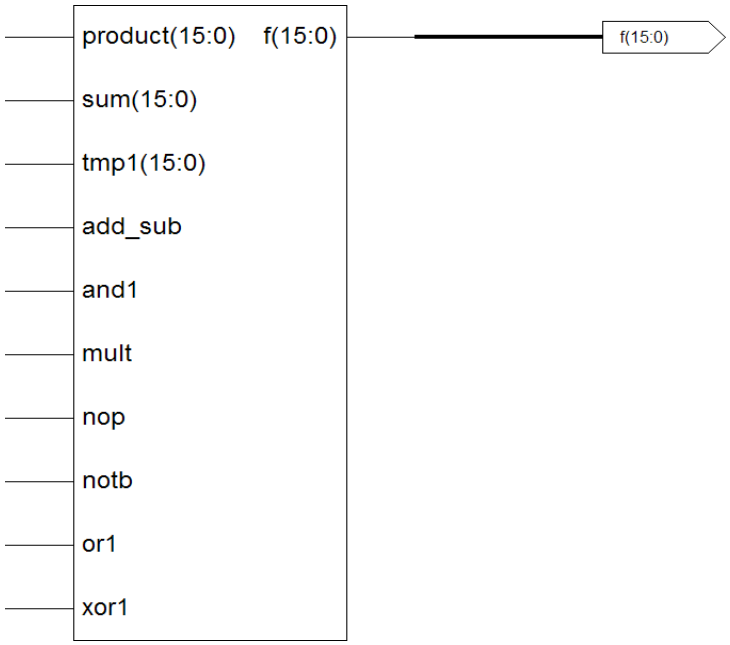
A continuación su algoritmo y estructura:

Algoritmo_multiplexor:

{Variables:

Variables de entrada: product, sum, tmp1, and1 add_sub, and1, mult, nop, notb, or1, xor1, y una única variable de salida f.}

Módulo Multiplexor



Inicio
Leer Sum, product, tmp1
Si (mult='1')and(add_sub='0')and(and1='0')and(or1='0')and(xor1='0')and(notb='0') and(nop='0') Entonces
f ⟵ product
De lo contrario
(mult = '0')and(add_sub='1')and (and1='0')and(or1='0') and(xor1='0') and(notb='0') and(nop='0'
f ⟵ sum
De lo contrario
(mult = '0')and (add_sub = '0')and (and1='1')and(or1='0')and (xor1='0')and (notb='0') and(nop='0')
f ⟵ tmp1
De lo contrario
(mult = '0') and (add_sub = '0')and (and1='0')and(or1='1')and (xor1='0')and (notb='0') and(nop='0')
f ⟵ tmp1
De lo contrario
(mult = '0') and (add_sub = '0')and (and1='0')and(or1='0')and (xor1='1')and(notb) and(nop='0')
f ⟵ tmp1
De lo contrario

(mult = '0') and (add_sub = '0')and (and1='0')and(or1='0')and (xor1='0')and(notb='1') and(nop=’0’)

f ⟵ tmp1

De lo contrario

(mult = '0') and (add_sub = '0')and (and1='0')and(or1='0')and (xor1='0')and(notb=’0’) and(nop='1')

f ⟵ "0000000000000000"

De lo contrario

f ⟵ "0000000000000000"

Fin del si

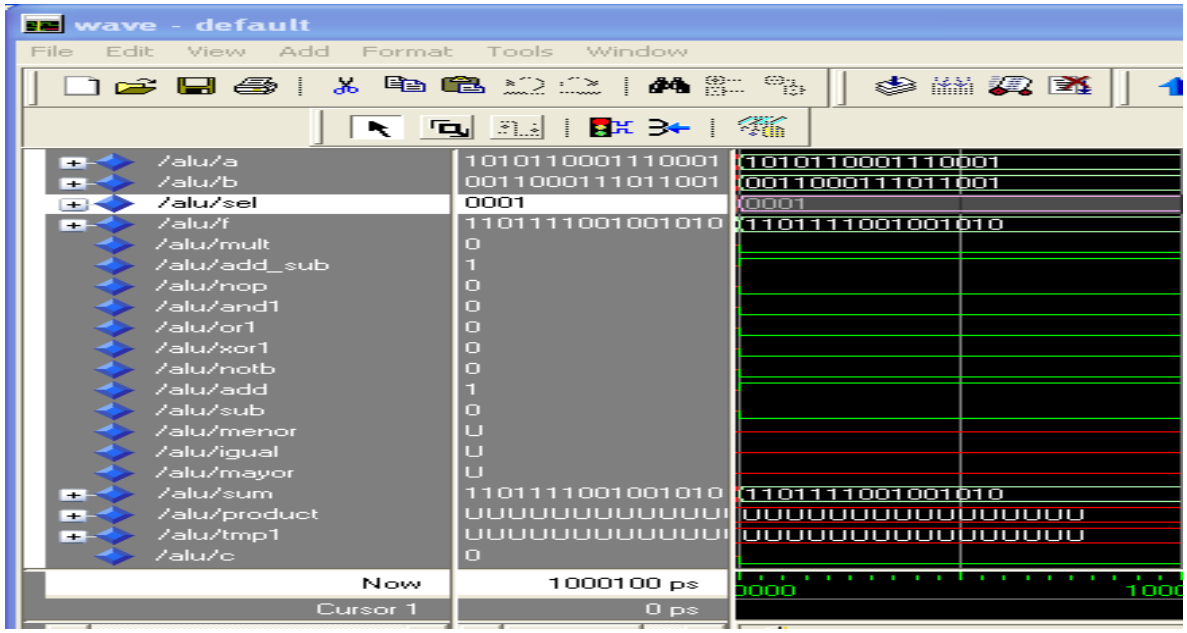
Fin del algoritmo.

5.6 Pruebas realizadas con la simulación funcional del modelsim.

Operación de Suma:

En la **figura1_suma** el código de operación de la suma es 0001 y se activa **add** y el sumador/restador con la variable add_sub enviando su respuesta por la variable sum, también muestra el acarreo de la suma en la variable C, y el multiplexor, a la vez toma su resultado y lo muestra por la variables f para el resultado final y C para el acarreo.

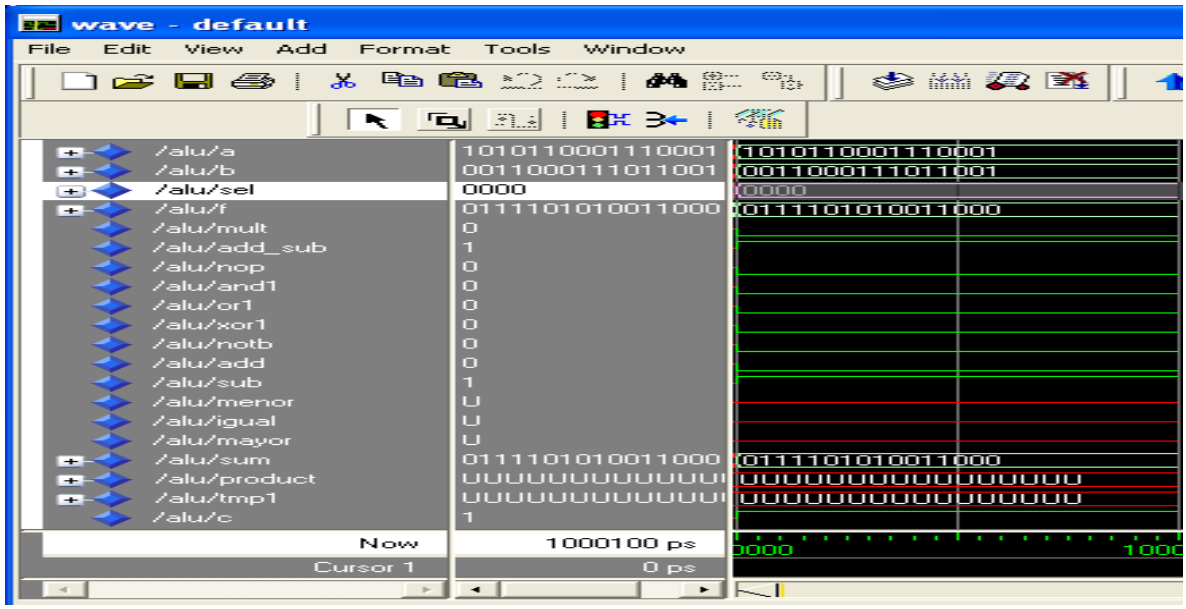
Figura1_suma



Operación de resta:

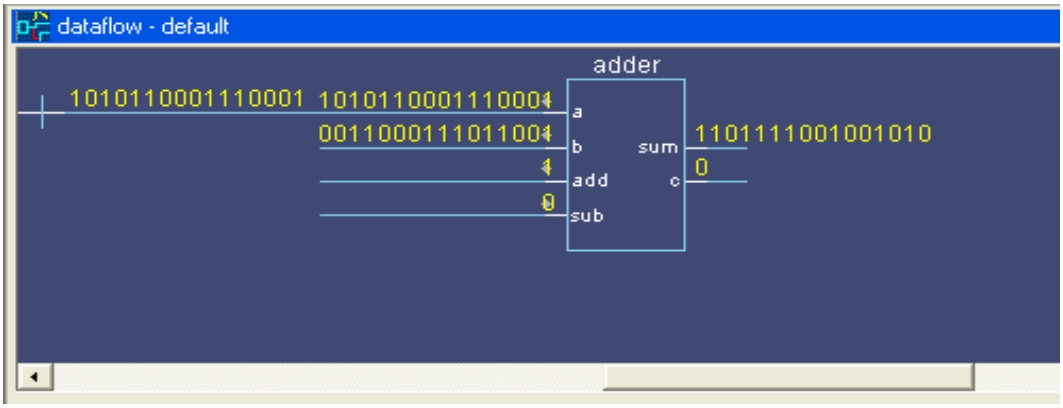
En la **figura1_resta** el código de operación de la resta es 0000 y se activa **sub** y el sumador/restador con la variable add_sub enviando su respuesta por la variable sum, también muestra el acarreo de la suma en la variable C, y el multiplexor, a la vez toma su resultado y lo muestra por la variables f para el resultado final y C para el acarreo.

Figura1_resta



En la **fig_suma/resta** se observa el esquema del sumador restador, con sus dos datos de entrada, aquí se activa la operación suma y envía su resultado a través de la variable **sum** y su acarreo presenta cero.

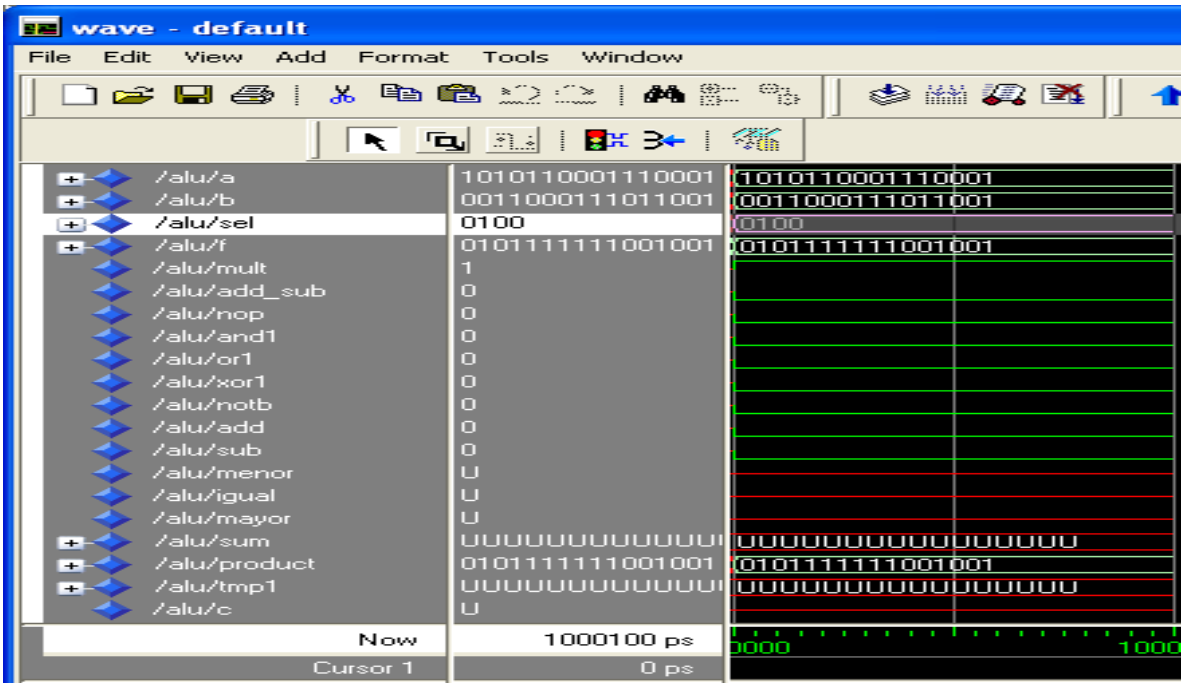
Fig_suma/resta



Operación de multiplicación:

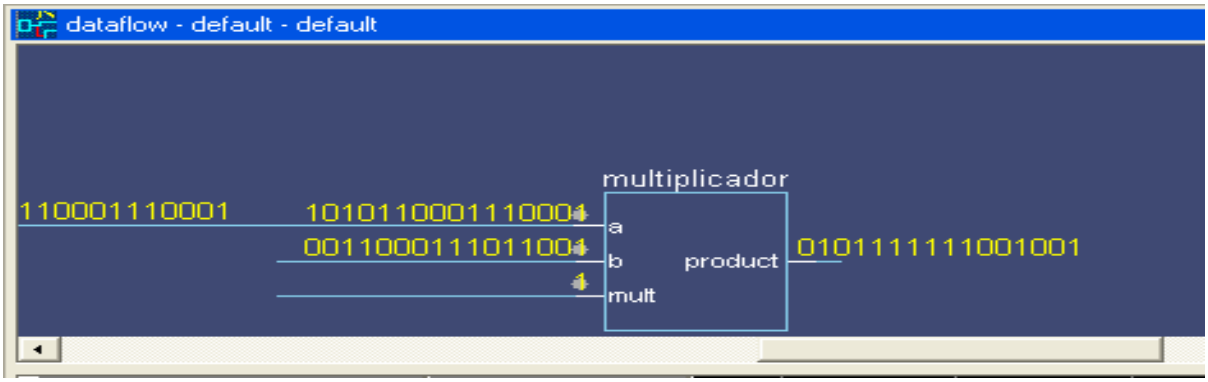
En la figura_mult el código de operación de la multiplicación es 0100 y se activa **mult** como señal de operación enviando su respuesta por la variable product y el multiplexor, a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivadas porque no es su tuno para operar.

Figura_mult



En **la fig_mult** se muestra el esquema del módulo multiplicador, mostrando el resultado a través de la variable product.

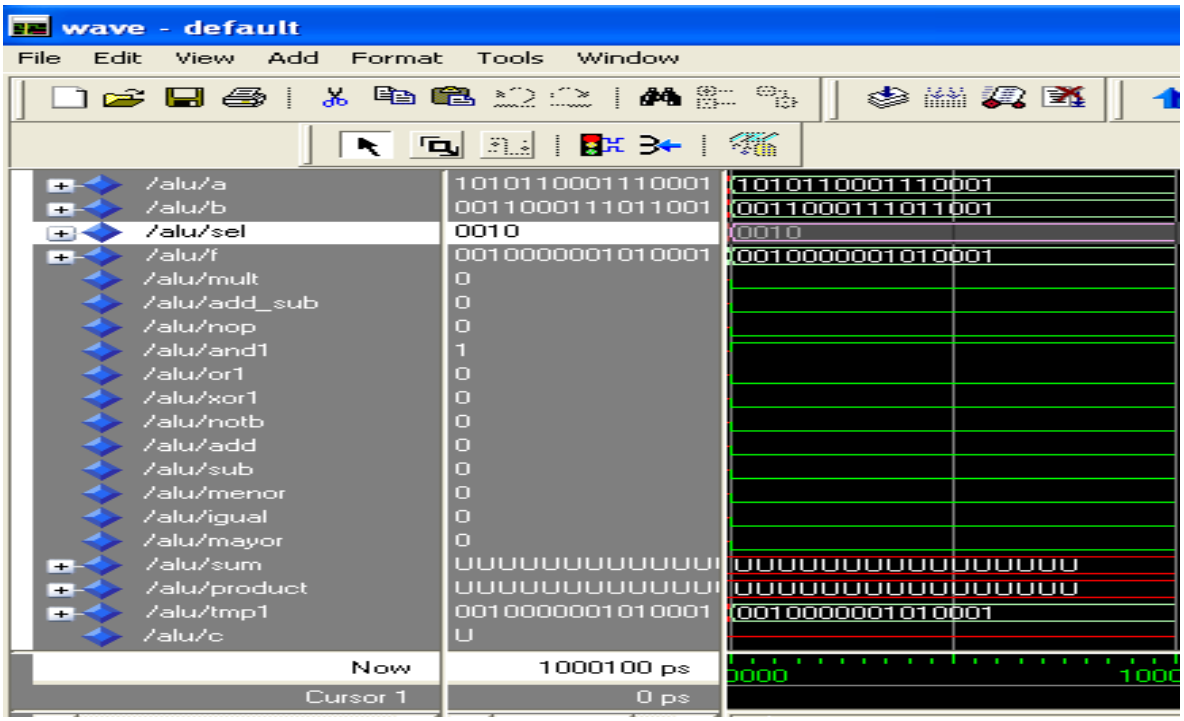
Fig_mult



Operación Lógica: AND

En la **figura1_and** el código de al operación AND es 0010 y se activa and1, como señal de operación enviando su respuesta por la variable tmp1 y el multiplexor, a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivada porque no es su turno para operar.

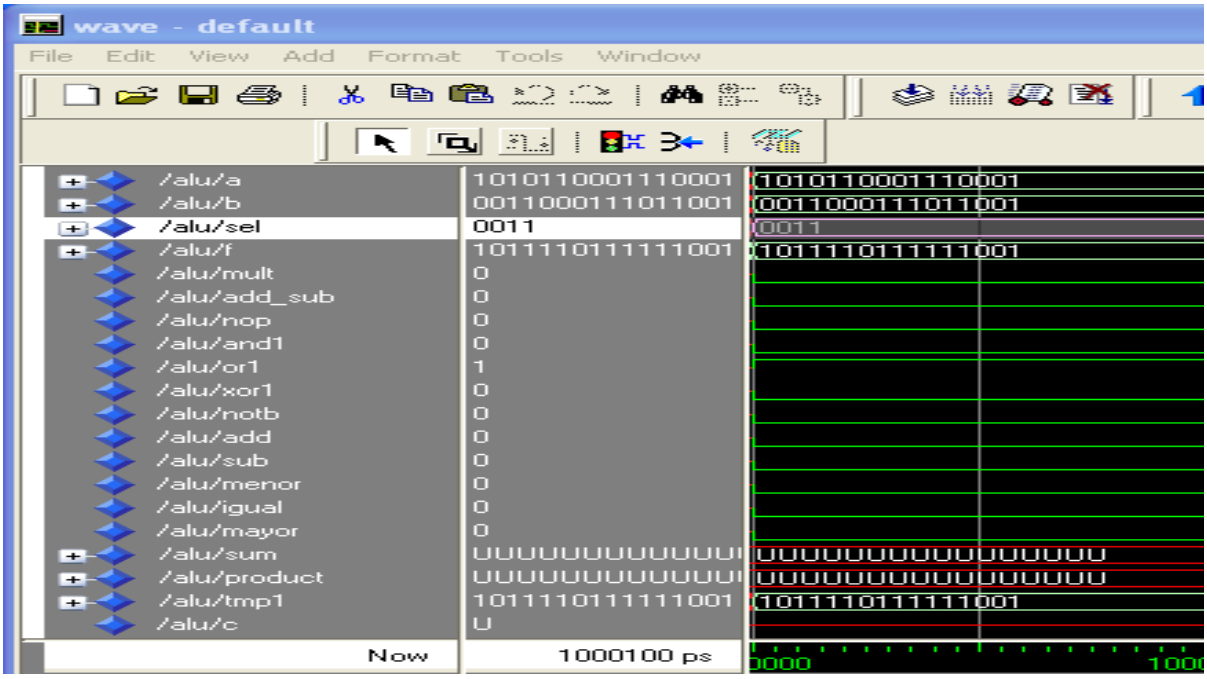
Figura1_and



Operación Lógica: OR

En la **figura _OR**, el código del operando OR es 0011 y se activa **or1** como señal de operación enviando su respuesta por la variable tmp1 y el multiplexor, a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivadas porque no es su turno para operar.

Figura_OR



Operación Lógica: XOR

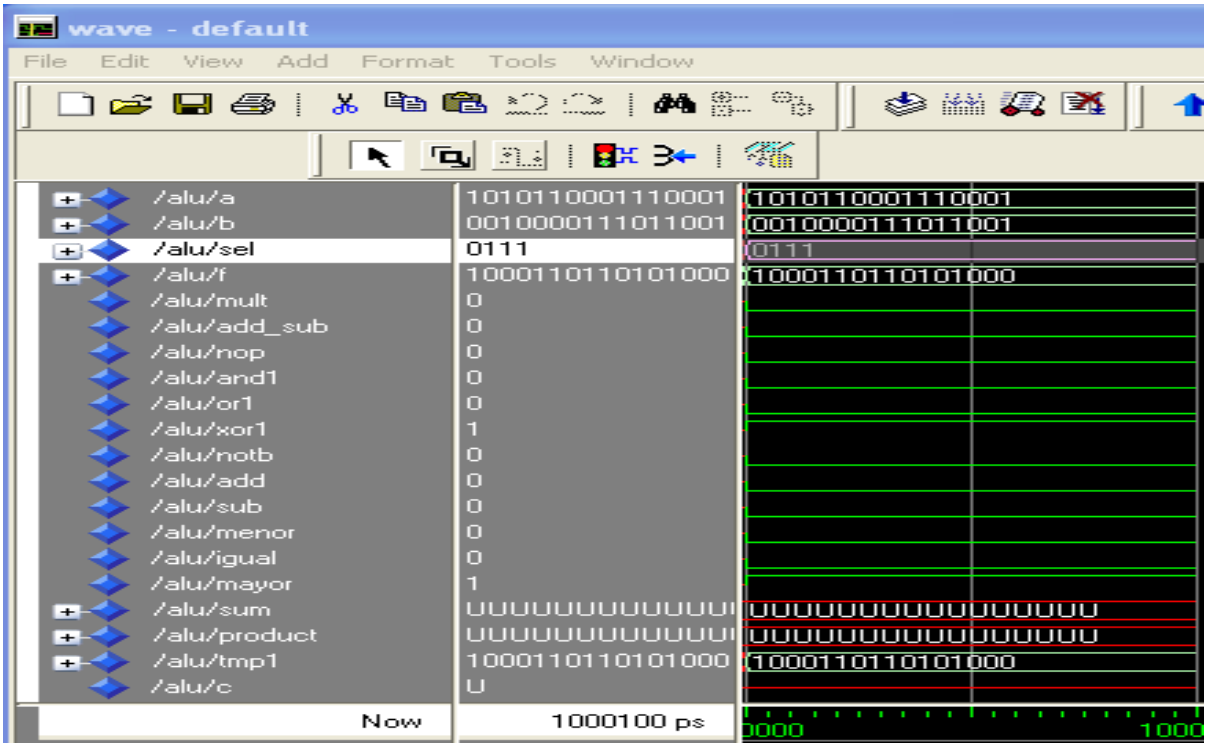
En la **figura1_XOR** el código del operando XOR es 0111 y se activa **xor1,** como señal de operación enviando su respuesta por la variable tmp1 y el multiplexor a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivadas porque no es su turno para operar.

Además este operador lógico tiene la función de comparar, por eso se activa la variable mayor que significa que el operando a > b esto se muestra en la **figura1_XOR**.

De igual manera en la **figura2_XOR** se muestra la operación lógica xor, diciendo que el operando A<B.

En la **figura3_XOR** se muestra la operación lógica xor, diciendo que el operando A=B,

Figura1_XOR



En la **fig_log_menor** se muestra la operación xor, muestra su resultado por la variable tmp1 Y se activa la operación de comparación mayor.

Fig_log_menor

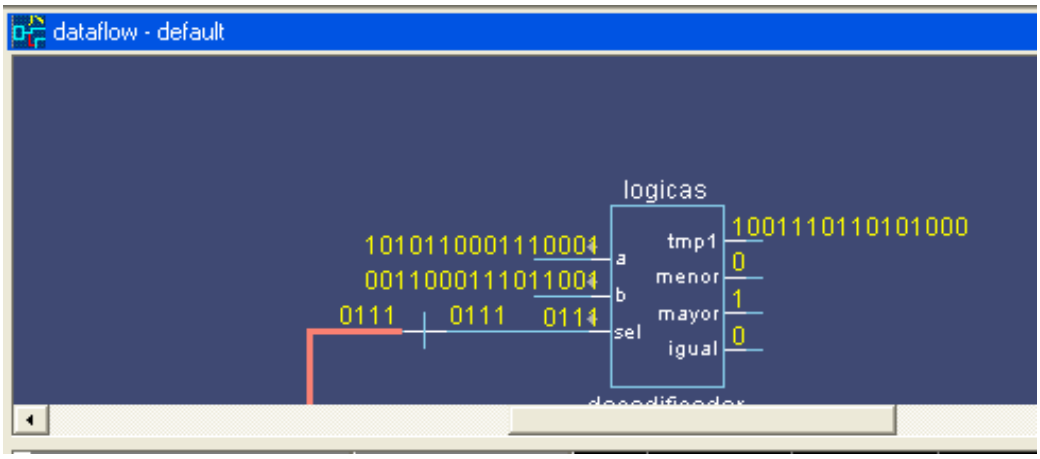


Figura2_XOR

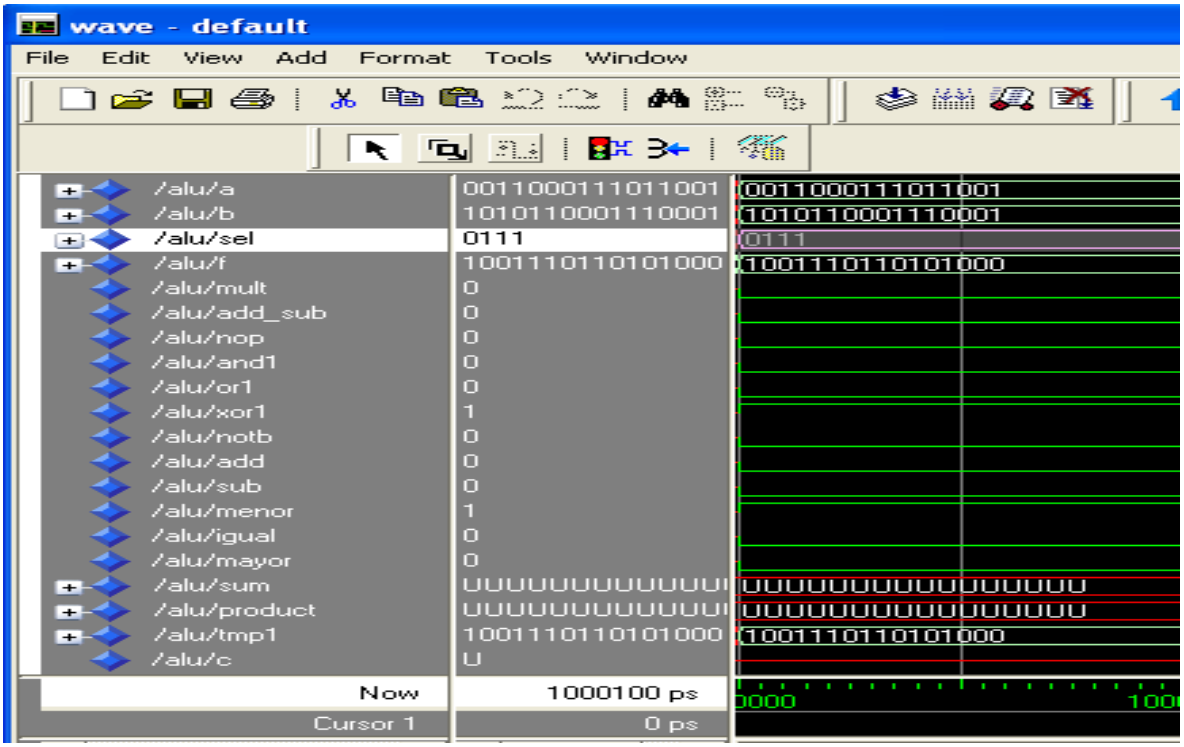
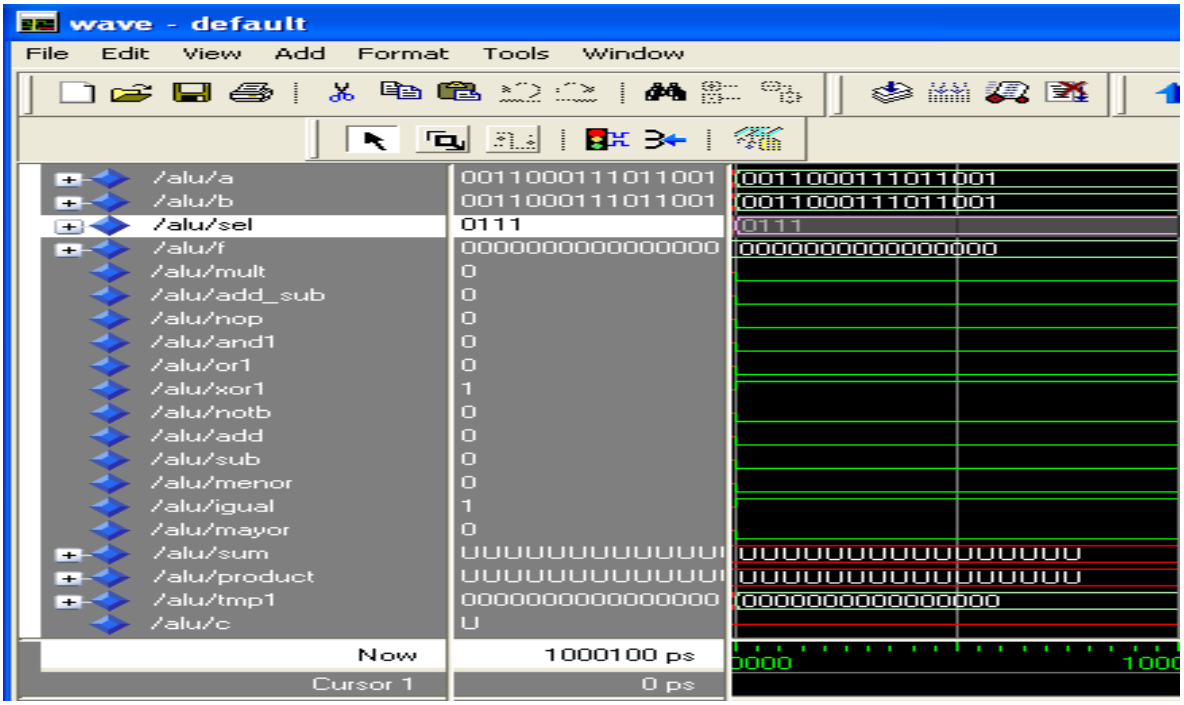
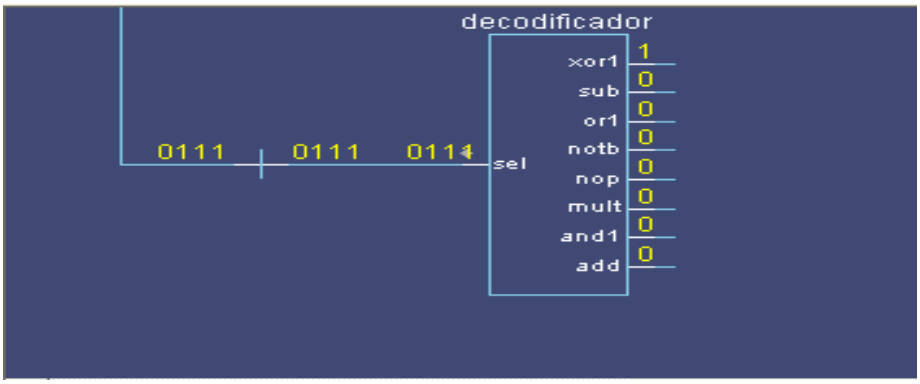


Figura3_XOR



Figxor1

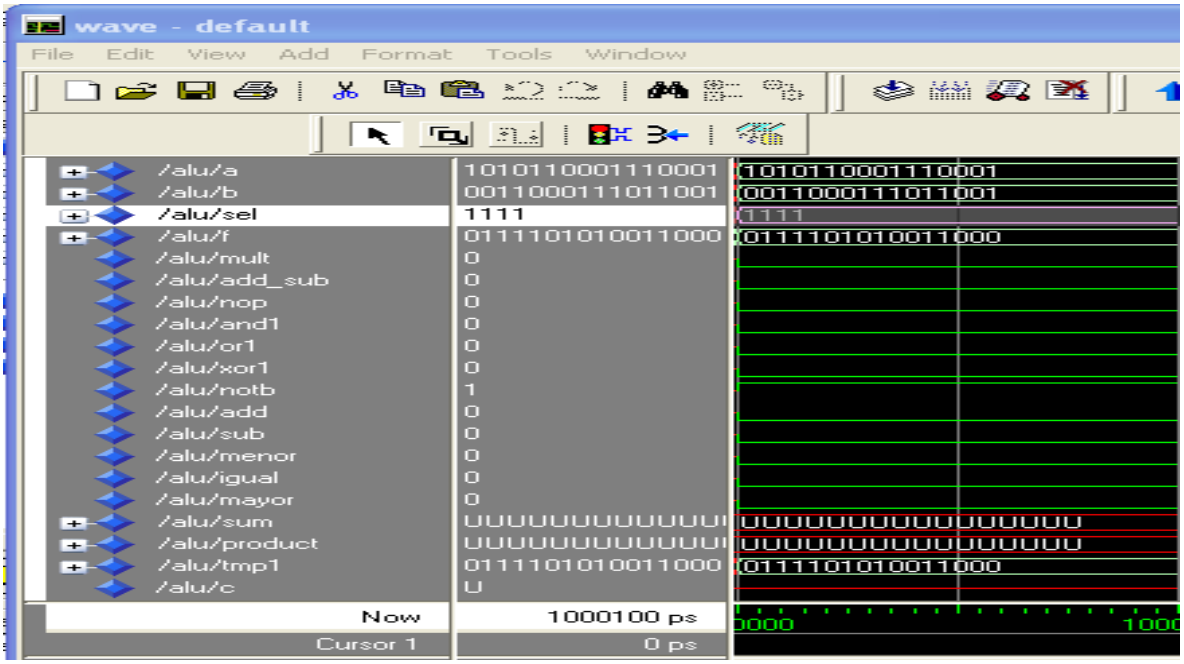


En la **figxor1** se puede observar el decodificador activando la operación xor, y para ello se utiliza la variable xor1 y se puede notar que las demás variables están desactivadas.

Operaciones Lógicas: A not B

En la **figura_not b**, el código del operando OR es 1111 y se activa **not b** como señal de operación enviando su respuesta por la variable tmp1 y el multiplexor, a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivadas porque no es su turno para operar.

Figura_not b

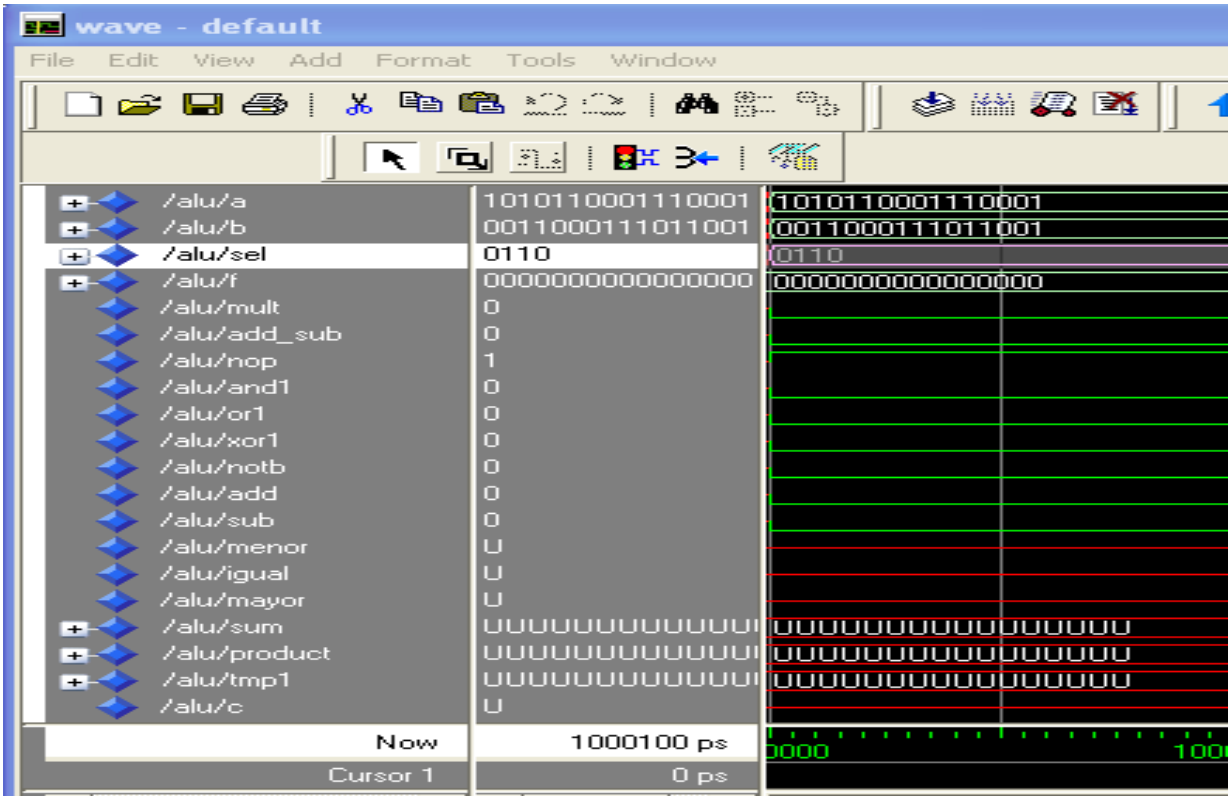


Operacion: Nop

En la **figura_nop**, el código de la operación es 0110 y se activa **nop** como señal de operación enviando su respuesta por la variable tmp1 y el multiplexor, a la vez toma su resultado y lo muestra por la variables f; todas las demás variables están desactivadas porque no es su turno para operar.

El resultado de esta operación la salida será cero lo que indica que no se realiza ninguna operación.

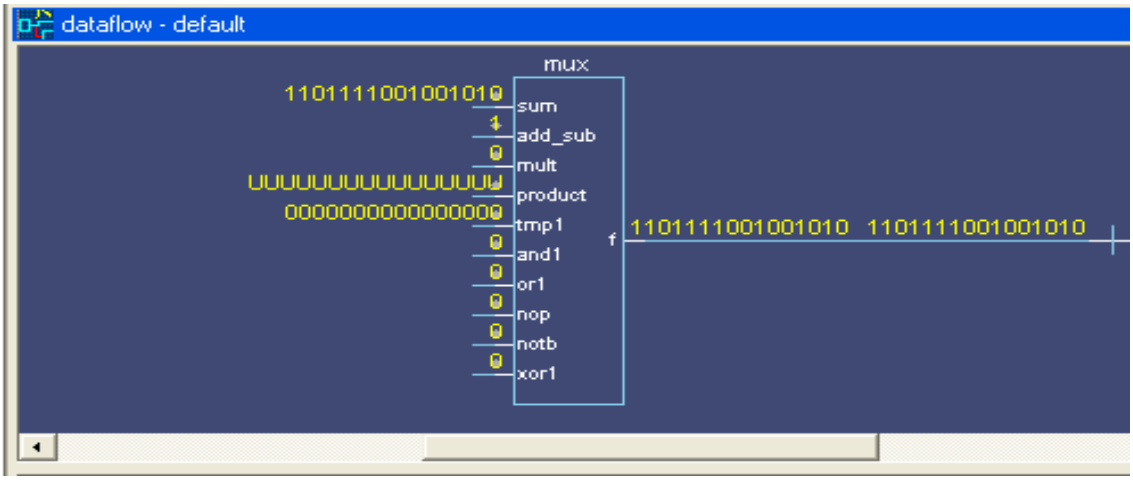
Figura _nop



Esquema del multiplexor:

En **la fig_multiplexor** presenta el esquema del multiplexor mostrando el resultado de una suma, a través de la variable f.

Fig_multiplexor



Conclusiones de las pruebas de la simulación

- Las pruebas se realizaron una y otra vez corrigiendo errores de análisis y diseño.

- Se elaboro el diseño apropiado para la realización de las operaciones.

- Se compiló con el simulador modelsim la Aplicación, obteniendo una buena simulación funcional.

- Por ultimo se sintetizó y se implementó el código con el simulador Xilinx de VHDL y fue satisfactoriamente.

- Se puede concluir que la aplicación combi nacional funciona bien realizando satisfactoriamente todas las operaciones propuesta.

VI Conclusiones

- Se diseñó una unidad aritmética lógica utilizando operadores combinacionales que no tienen elementos de memoria pero que realizan varias operaciones típicas de la ALU.
- Se desarrolló una aplicación para el diseño de la UAL en VHDL y se implementó en una plataforma de Xilinx versión 9.1c, simulándose con Modelsim versión XE III 6.2c Para llegar a ello fue necesario hablar de las características de los microprocesadores.

- Se realizaron 5 módulos para lograr el diseño de la UAL, realizando operaciones aritméticas (Suma, resta, multiplicación y negación) y lógicas (And, or, Xor, comparación, notb) con dígitos binarios.

- Se realizó la simulación de todo el diseño y se probó su buen funcionamiento.

VII Recomendaciones

- Mejorar la aplicación utilizando operadores secuenciales, que si tienen elementos de memoria.

- Ampliar la aplicación para que esta realice todas las operaciones de una unidad aritmética lógica en dígitos binarios como en hexadecimal.

- Adquirir documentación de lógica programable que facilite el conocimiento a los estudiantes en esta área.

- Incluir en una de las asignaturas de la carrera el tema de lógica programable y el lenguaje VHDL.

VIII Bibliografía

1. Estructuras de computadores II adición actualizada 1997.

José Mario Angulo Usategui.

2. Fundamentos de computadores, séptima edición.

Pedro de miguel Anasagasti.

3. Fundamentos de computadoras Digitales

Thomas C. Baratee

Mc. Graw Hill Quinta edición

4. Organización y Arquitectura de Computadoras

William Stallings

Printice Hall Quinta edición

Webgrafia

www.monografias.com

www.xilinx.com

http://canalhanoi.iespana.es/hardware/microprocesadores.htm.

http://es.wikipedia.org/wiki/Sumador

http://es.wikipedia.org/wiki/Unidad_aritm%C3%A9tico_l%C3%B3gica

http://es.wikipedia.org/wiki/Serie_7400

http://es.wikipedia.org/wiki/Multiplicador

http://es.wikipedia.org/wiki/Circuito_digital

http://html.rincondelvago.com/computacion_2.html

http://boards5.melodysoft.com/app?ID=foroiri3&msg=101&DOC=321

http://www.mailxmail.com/curso-arquitectura-ordenadores/unidad-central-proceso

http://www.mflor.mx/materias/comp/cursohar/hardwar1.htm

http://es.kioskea.net/contents/pc/processeur.php3

http://html.rincondelvago.com/arquitectura-de-computadoras_2.html

http://canalhanoi.iespana.es/hardware/microprocesadores.htm

http://www.pdf-search-engine.com/unidad-aritmetica-logica-pdf.html

http://www.sindominio.net/~apm/articulos/apuntes/uned_2/etciii/

www.reduaz.mx/eninvie/CD2k6/Inst/28.pdf

Anexos

Código fuente de una ALU de 16 bits

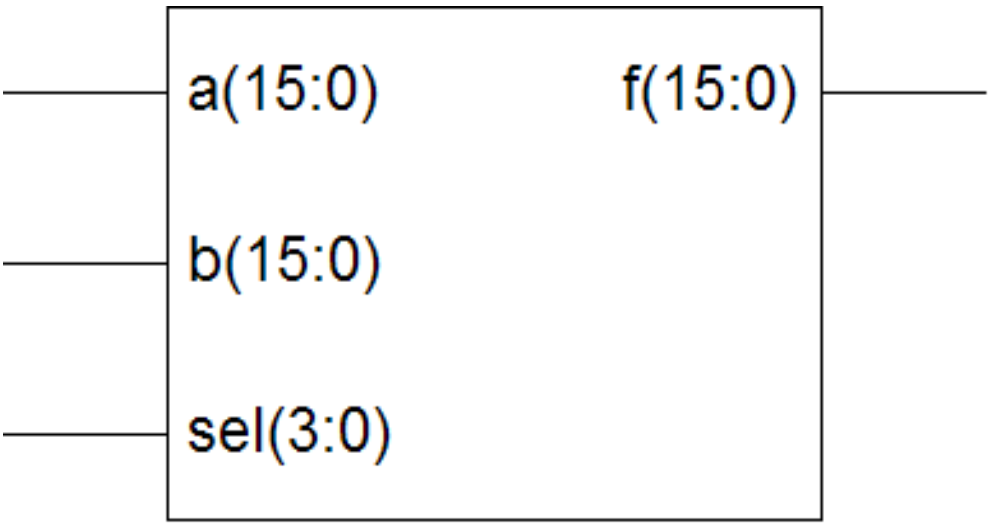
```
-- El modulo de la ALU llama a todos los demás módulos que la forman y la ayudan
--a realizar las diferentes operaciones, para ello se utilizo componentes.
--operaciones que realiza la ALU:
--1-Aritmeticas: Suma, resta, Multiplicación, y no operación.
--2-Logicas: and, or, Xor y comparison, a + Not (b).
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity alu is port
    (   a,b:            in std_logic_vector(15 downto 0);      --operandos.
        sel:            in std_logic_vector(3 downto 0);       --selección de la operación.
        f:              out std_logic_vector (15 downto 0 );
           end alu;
 architecture structural of alu is
 signal mult,add_sub  : std_logic;              --selection de entradas para el  mux
 signal nop  : std_logic;       --no operation
 signal and1,or1,xor1,notb : std_logic;         --op logica.
  signal add,sub :std_logic;                    --op aritméticas(suma,resta)
  signal menor, igual, mayor :      std_logic;
```

```
  signal sum, product,tmp1 :      std_logic_vector(15 downto 0);    --op de multiplicación
                          signal c  :       std_logic;
  componente decodificador
port  (sel: in std_logic_vector(3 downto 0);
        mult: out std_logic ;
         nop:  out std_logic ;
         add:  out std_logic ;
         sub:  out std_logic;
         and1, or1, notb :      out std_logic       );
    end component;
  component adder
  port (                          a,b:   in std_logic_vector (15 downto 0);
              add_sub, sub, a dd:  in std_logic;
                                  sum:   out std_logic_vector (15 downto 0);
                                    c:    out std_logic);
     end component;
component multiplicador
 port (a,b    : in std_logic_vector (15 DOWNTO 0);
        mult  : in std_logic ;
      product : out std_logic_vector (15 DOWNTO 0) );
end component;
component mux
     port( product, sum, tmp1:   in std_logic_vector (15 downto 0);
```

```
            add_sub, mult:        in std_logic ;
      and1,or1, xor1 ,notb, nop:  In std_logic;
                                  f:   out std_logic_vector (15 downto 0) );
end component;
component logicas
    port  (          a,b:   in std_logic_vector (15 downto 0);          --operands
                     sel:     in std_logic_vector(3 downto 0);
                   tmp1:      out std_logic_vector(15 downto 0);        --function output
                  menor:       out std_logic;
                   igual:       out std_logic;
                  mayor:       out std_logic );
end component;
begin
   add_sub <= add or sub; --transmite la señal al multiplexor
p0: decodificador port map
   (     sel => sel,
         mult  => mult,
          nop  => nop,
           add  => add,
           sub  => sub,
           and1 =>and1,
            or1  =>or1,
                 xor1 =>xor1,
```

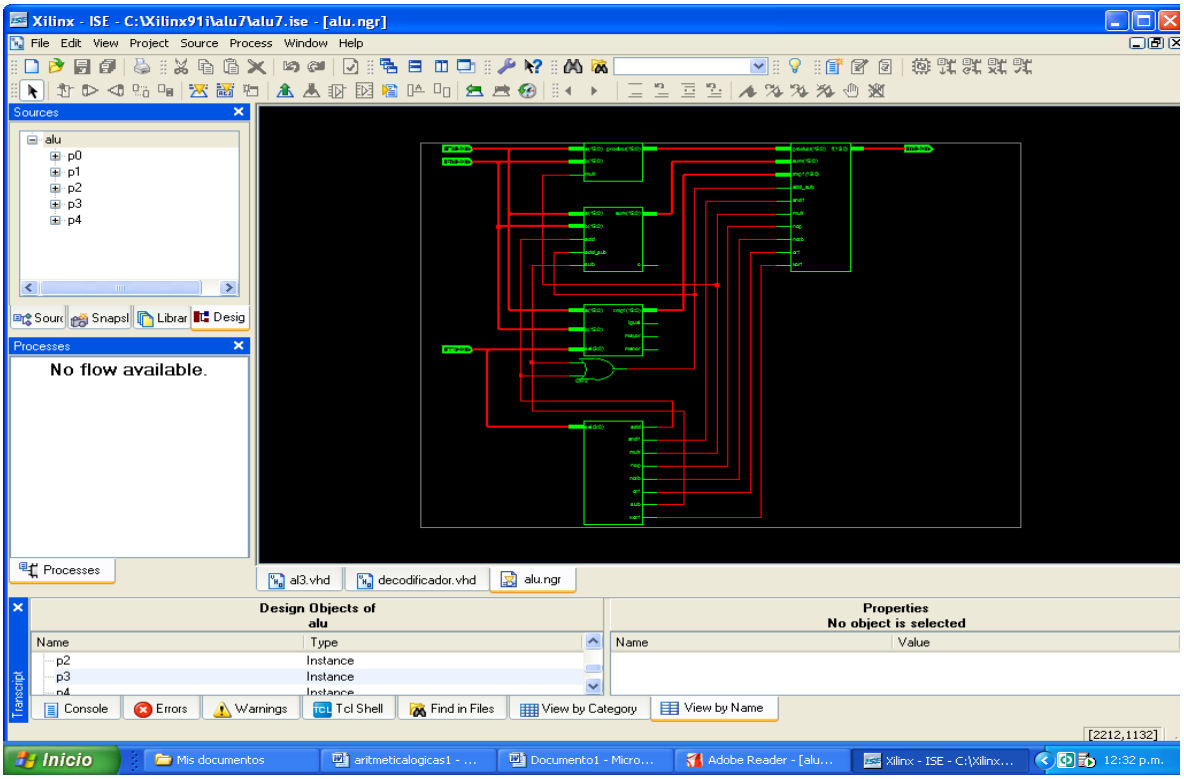
```
                notb =>notb   );
p1: adder port map
          (    a => a,  b => b,  c => c,
            add_sub => add_sub,
            add => add,  sub => sub,   sum => sum );
p2 : multiplicador port map
   (    a => a,  b => b,  mult => mult,
        product => product );
p3 : mux port map
   ( product => product,
              sum => sum,  add_sub => add_sub,
              mult => mult, and1 => and1,
               or1 => or1, nop => nop,
               xor1 => xor1, tmp1 => tmp1,
                notb => notb,  f => f  );
p4 : lógicas port mal
        (   a => a,   b => b,  sel => sel,  tmp1 =>tmp1,
           menor =>menor,  igual =>igual,   mayor => mayor  );
   end structural;
```

Estructura general de la ALU:

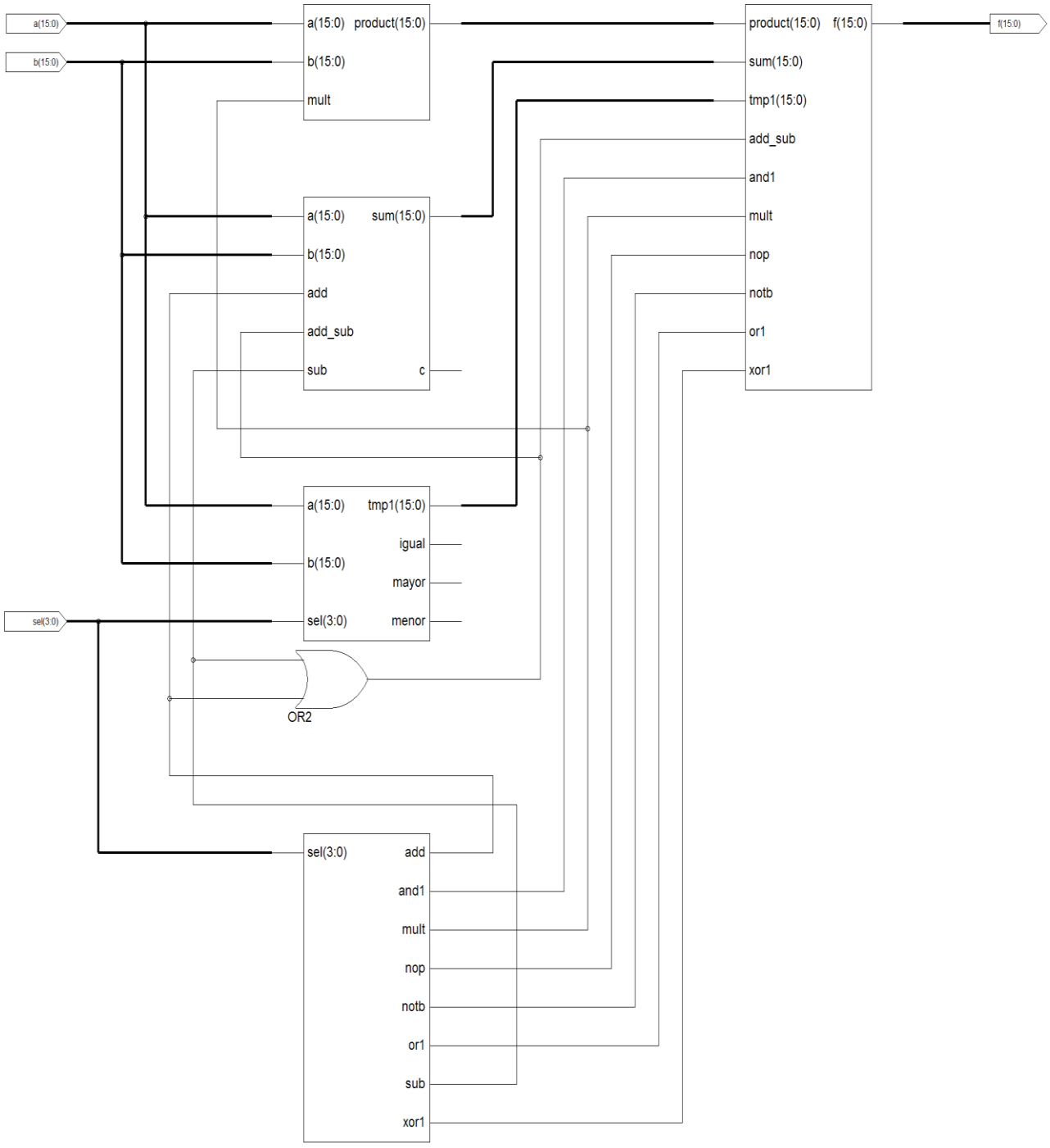


En la fig1 se muestra la pantalla de salida con el xilin, contiene la estructura interna de la ALU con los cinco módulos

Fig1



Estructura interna de la Alu



****
Reporte del reloj
****

Clock Net	Resource	Locked	Fanout	NetS kew (ns)	Max Delay(ns)
p0/add	Local		51	0.190	2.692
p4/menor_not0001	Local		16	0.043	2.449
p0/notb_not0001	Local		4	0.699	1.413
p0/mult	Local		17	0.072	2.570

Glosario

ALU: Unidad Aritmético Lógica: Componente básico de un microprocesador, consistente en una serie de puertas lógicas que realizan las operaciones aritméticas y lógicas.

Ancho de bus: El ancho de bus está determinado por el número de bits que se pueden transmitir simultáneamente por el bus.

Contador de programa (PC): Registro de un procesador cuyo contenido es la dirección de memoria de la siguiente instrucción que se va a ejecutar.

Coprocesador: Junto a los diversos microprocesadores se utilizan coprocesadores especiales, los cuales colaboran, sobre todo, en las operaciones aritméticas. Por ello también son llamados coprocesadores matemáticos. Éstos son denominados 8087,80287, 80387, 80487, de acuerdo al microprocesador que sea utilizado.

CPU: Unidad Central de Proceso (UCP).
La porción del computador encargada de la búsqueda y ejecución de las instrucciones. Básicamente consiste en las unidades funcionales, la unidad de control y los registros. A menudo se refiere simplemente como procesador.

Microprocesador: Es un circuito integrado que contiene un procesador. Es casi el corazón de cada ordenador y consta de miles de transistores y otros componentes, integrados en una pastilla o chip.

Palabra: Son dos bytes contiguos, los cuales comienzan en una frontera de bytes definida. Contiene en total 16 bits, los cuales están consecutivamente numerados de 0 hasta 15.

Unidad funcional: Cualquiera de las unidades aritmético lógicas de un procesador.

Arquitectura: Se refiere a la estructura lógica de un circuito. Al hablar de FPGAs se llama arquitectura a las características que comparten los dispositivos de una familia; por ejemplo, tenemos a las familias XC4000, XC4000XL, Virtex, Spartan

CLB El Bloque de Lógica Configurable (Configurable Logic Block por sus siglas en ingés) es el elemento básico del FPGA, ya que es ahí donde se almacena la información relativa al comportamiento del circuito que hemos diseñado. Puede contener, por ejemplo: dos generadores de funciones de 16 bits, un generador de función de 8 bits, 2 registros y un multiplexor.

Componente: Un componente es un dispositivo que puede ser instanciado o generado dentro de una entidad. Una entidad VHDL está formada por una declaración de señales y por una arquitectura. Al inicio de la arquitectura es posible declarar también las señales de un componente, con el objetivo de utilizarlo. En el cuerpo de la arquitectura se realiza el mapeo de las señales de la entidad con las del componente, cuyo comportamiento se ha descrito previamente en otro archivo *.HDL.

Editor de Restricciones – Constraints Editor Esta herramienta de Xilinx permite agregar restricciones de tiempo a un diseño, para después proceder a ejecutar la implementación.

Editor HDL – HDL Editor El ambiente de desarrollo de Xilinx contiene un editor que nos permite crear y modificar un archivo en VHDL, Verilog o Abel. Además es posible verificar la sintaxis y sintetizar tal archivo. En este proyecto únicamente empleamos VHDL.

Entidad: En VHDL una entidad es un modelo de comportamiento sintetizable. La entidad incluye una declaración de puertos, es decir, de señales de entrada y de salida y una descripción de la función que se ejecutará. Un diseño puede estar compuesto de una o más entidades interrelacionadas.

FPGA: Un FPGA (*Field Programmable Gate Array*) es un tipo de circuitos integrados con arquitectura basada en RAM, que pueden reprogramarse después de haber sido fabricados. Estos circuitos pueden tener múltiples aplicaciones en el área de sistemas digitale.

Simulación funcional: A través de este tipo de simulación se puede probar su funcionamiento antes de implementar un diseño, pues asigna retardos unitarios.

HDL: Este tipo de lenguajes de alto nivel (*Hardware Description Language*) permiten describir el comportamiento de un circuito de forma textual.

Asistente del lenguaje: El HDL Editor de Xilinx posee un asistente que contiene plantillas con las instrucciones más comunes del VHDL, por ejemplo: *if... else... end if, case*, operadores lógicos y aritméticos.

Mapeo: Durante el mapeo se asigna de forma automática un la lógica de un diseño a elementos físicos que componen el FPGA elegido.

Síntesis: A través del proceso de síntesis se convierte una descripción realizada en un lenguaje de alto nivel como VHDL correspondiente al FPGA seleccionado.

LÓGICA PROGRAMABLE

DISEÑO DE UNA UNIDAD ARITMÉTICA LÓGICA

DEDICATORIA

Alina Ramírez Reyes

A Jehová Dios: Por permitirme calificar en dicha carrera, y darme
Sabiduría, Entendimiento para poder culminar dichos estudios.
A mi familia: Mi esposo Rene Palacios por apoyarme
siempre. A mi hija Alison Palacios por entenderme
y a mi Madre Rosalina Reyes. Y a mí amiga
Luz Marina Obando.

Georgina Blanco Gómez

A Dios por guiarme en este largo camino, por darme
Entendimiento y sabiduría.
A mis padres José Félix Blanco y Silvia
Elena Gómez por su esfuerzo y apoyo incondicional.

Elsa Santana Sirias

A Dios por darme vida y fortaleza para seguir adelante.
A mis padres Bernardo Santana y Isabel Sirias por
su esfuerzo y siempre confiar en mí, A mi esposo
Donald Acevedo por su paciencia y
apoyo incondicional.

AGRADECIMIENTO

Alina Ramírez Reyes

Primeramente a Jehová Dios por darme vida hasta el día de hoy
Y haberme permitido culminar mis estudios universitarios
a pesar de todas las dificultades que se me presentaron.

A mi esposo por ayudarme económicamente
Y brindarme su apoyo incondicional para
cumplir mis metas profesionales.

A Msc. Eman Hussein Yousif. Por su paciencia y eficiente
Colaboración y constante Preocupación en la elaboración
del presente estudio. Al Profesor Walter Pastran
y Reinero Bermúdez Por su incondicional apoyo brindado.

A mí amiga Luz Marina Obando que me brindo apoyo
Moral, emocional y económico para seguir adelante.

Georgina Blanco Gómez

Primeramente a Dios por darme la fortaleza
Día a día de continuar con los estudios
Pese a las adversidades y obstáculos del camino.

A mis padres José Félix Blanco y
Silvia Elena Moya por ayudarme
Económica y emocionalmente para
Cumplir mis metas profesionales.

A Msc. Eman Hussein Yousif por su apoyo
Incondicional y constante colaboración en la
Elaboración del presente estudio. Al Profesor
Walter Pastran y Reinero Bermúdez por su
Paciencia y Compromiso por transmitirme
Parte de sus conocimientos.

Elsa Santana Sirias

A mi familia por su cariño y apoyo, por siempre darme ánimos a seguir adelante.

A mis profesores y profesoras de quienes siempre recibí un gran apoyo y a quienes admirare por el resto de mi vida.

A la Msc. Eman Husein Yousif a quien debo agradecer su paciencia y confianza para estas tres humildes servidoras.

RESUMEN

El presente trabajo de defensa en la modalidad de seminario de Graduación es el desarrollo de una Unidad Aritmética Lógica, tomando como tema general Lógica Programable, utilizando un lenguaje de alto nivel llamado VHDL con el cual se podrán desarrollar todos los módulos de la aplicación utilizando la plataforma de Xilinx versión 9.1c y posteriormente los simularemos con Modelsim versión XE III 6.2c.

El proyecto es un sistema digital combinacional que realiza operaciones aritméticas; Las operaciones que puede realizar la ALU son: multiplicación, suma, resta y operaciones Lógicas (AND, OR, XOR, Comparación, NOT) entre datos, donde cada dato es representado por dos operándos de entrada y una salida. De esta manera se facilitan algunas operaciones complejas, como ejemplo, la de la multiplicación.

El presente documento se divide en cuatro capítulos que tratan de algunos temas relacionados directamente con el desarrollo del proyecto.

En el capitulo uno se plantea de forma muy general el tema a desarrollar lógica programable, una breve reseña de sus inicios y la evolución de los dispositivos; a continuación en el capitulo dos se mencionan como iniciaron los microprocesadores, su evolución y como están divididos. Posteriormente el capitulo tres habla de la unidad aritmética lógica como esta estructurada y las operaciones que realiza. Otro capitulo es el cuatro aquí se describe en sí al lenguaje a utilizar en el proyecto VHDL. Y a continuación se muestra todo lo concerniente al diseño metodológico del proyecto; se describe como se realizaron los cuatro módulos de la aplicación y como se simulo cada uno de ellos para lograr su buen funcionamiento.