CAPÍTULO 1. MARCO TEÓRICO

1.1 Estructura general de una computadora.

La computadora digital es un ordenador numérico, automático, secuencial y universal.

Es numérico porque toda la información que se puede encontrar dentro de la computadora está codificada por un conjunto ordenado de ceros y unos. Esta codificación es de tal naturaleza que un conjunto de ceros y unos, pueden estar representando una letra o un número, por esto decimos que la información dentro de una computadora puede ser alfanumérica.

Se dice que es automática pues puede operar sin la intervención del operador al pasar de una operación a otra en la resolución de un determinado problema.

Es secuencial porque debe seguir una serie ordenada de pasos para la resolución de cada problema.

Y finalmente decimos que es universal porque está capacitada para resolver muchos problemas que se plantee, dependiendo de cómo se haya programado a la máquina.

La computadora es un sistema compuesto de cinco elementos diferenciados: 1.-CPU (unidad central de Procesamiento), 2.-Dispositivos de entrada, 3.-Dispositivos de almacenamiento, 4.-Dispositivos de salida y una 5.-Red de comunicaciones, denominada bus, que enlaza todos los elementos del sistema y conecta a éste con el mundo exterior. A continuación se muestra un diagrama de bloques de una computadora (figura 1.1).

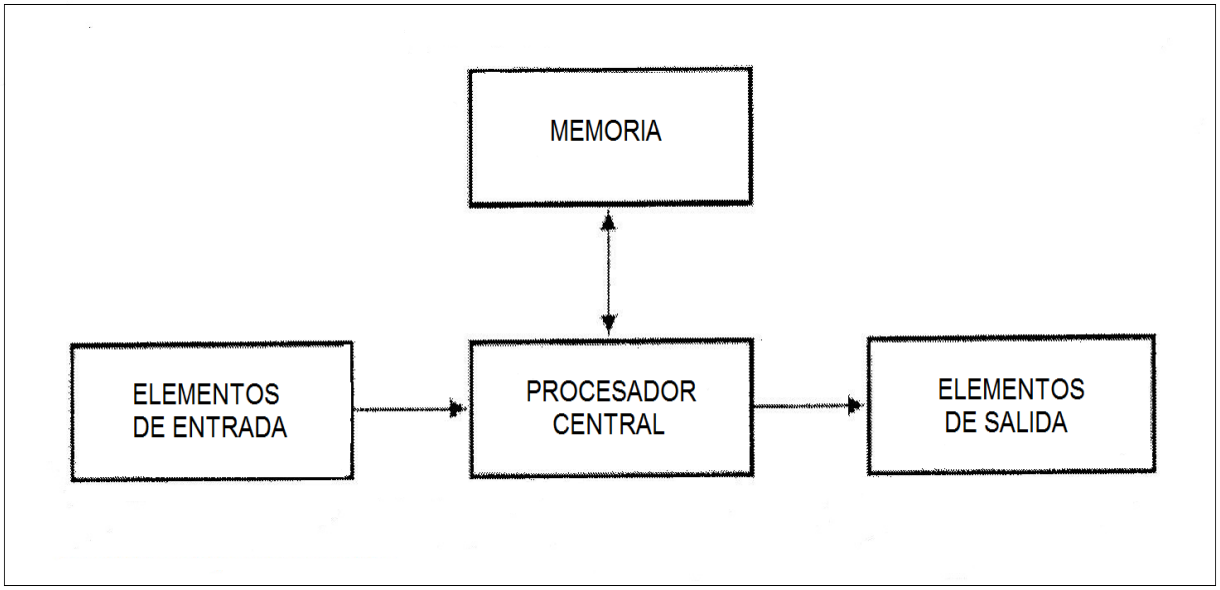


Figura 1.1 Estructura general de una computadora.

1.1.1 –CPU (Unidad Central de Proceso):

Interpreta y lleva a cabo las instrucciones de los programas, efectúa manipulaciones aritméticas y lógicas con los datos y se comunica con las demás partes del sistema. Una CPU es una colección compleja de circuitos electrónicos. Cuando se incorporan todos estos circuitos en un chip de silicio, a este chip se le denomina microprocesador. La CPU, chips y componentes electrónicos se ubican en un tablero de circuitos o tarjeta madre.

La mayoría de los chips del CPU y de los microprocesadores están compuestos de 4 secciones funcionales:

- Una unidad aritmética/lógica que proporciona al chip su capacidad de cálculo.
- Unos registros que son áreas de almacenamiento temporal que contienen datos, realizan seguimiento de instrucciones y conservan la ubicación y los resultados de las operaciones.
- Una sección de control que temporiza y regula las operaciones de la totalidad del sistema informático, lee las configuraciones de datos en un registro designado y las convierte en una actividad e indica en qué orden utilizará la CPU las operaciones individuales y el tiempo que consumirá cada operación.
- Bus interno, red de líneas de comunicación que conecta los elementos internos del procesador y envía también información a los conectores externos que enlazan al procesador con los demás elementos del sistema informático.
- Hay 3 tipos de bus en la CPU: bus de control, bus de dirección y bus de datos.

1.1.2-Dispositivos de entrada:

Son todos aquellos elementos que permiten la interacción del usuario con la unidad de procesamiento central y la memoria.

En esta se encuentran:

- Teclado.
- Mouse o Ratón.
- Escáner o digitalizador de imágenes.
- Micrófonos.

El Teclado:

Es un dispositivo periférico de entrada, que convierte la acción mecánica de pulsar una serie de pulsos eléctricos codificados que permiten identificarla. Las teclas que lo constituyen sirven para entrar caracteres alfanuméricos y comandos a una computadora es similar al de las máquinas de escribir.

Mouse y Joysticks:

Son dispositivos que convierten el movimiento físico en señales eléctricas binarias que permitan reconstruir su trayectoria con el fin de que la misma sea repetida en el monitor.

Escáner o digitalizador de imágenes:

Están concebidos para interpretar caracteres, combinación de caracteres, dibujos gráficos escritos a mano o en maquinas o impresoras y traducirlos al lenguaje que la computadora entiende.

Micrófonos:

Módulos de reconocimiento de voz que convierten la palabra hablada en señales digitales comprensibles para el ordenador.

1.1.3 Dispositivos de Almacenamiento:

En esta se encuentran:

- Disco Duro.
- DVD.
- Cintas magnéticas.

Disco Duro:

Este está compuesto por varios platos, es decir, varios discos de material magnético montados sobre un eje central en el cual giran. Para leer y escribir datos en estos platos se usan las cabezas de lectura / escritura que mediante un proceso electromagnético codifican / decodifican la información que han de leer o escribir. La cabeza de lectura / escritura en un disco duro está muy cerca de la superficie, de forma que casi da vuelta sobre ella, sobre el colchón de aire formado por su propio movimiento. Debido a esto, están cerrados herméticamente, porque cualquier partícula de polvo puede dañarlos.

Este se divide en unos círculos concéntricos cilíndricos (coincidentes con las pistas de los disquetes), que empiezan en la parte exterior del disco (primer cilindro) y terminan en la parte interior (ultimo). Asimismo, estos cilindros se dividen en sectores, cuyo número está determinado por el tipo de disco y su formato, siendo todos ellos de un tamaño fijo en cualquier disco. Cilindros como sectores se identifican con una serie de números que se les asigna, empezando por el 1, pues el numero 0 de cada cilindro se reservan para propósitos de identificación más que para almacenamientos de datos. Estos escritos / leídos en el disco deben ajustarse al tamaño fijado del almacenamiento de los sectores. Habitualmente, los sistemas de discos duros contienen más de una unidad en su interior, por lo que el número de caras puede ser más de dos. Estas se identifican con un número, siendo el 0 para la primera. En general su organización es igual a los disquetes. La capacidad del disco resulta de multiplicar el número de caras por el de pistas por cara y por el de sectores por pista, al total por el número de bytes por sector.

Disco de Video Digital:

Disco de vídeo digital (DVD), un dispositivo de almacenamiento masivo de datos cuyo aspecto es idéntico al de un disco compacto, aunque contiene hasta 15 veces más información y puede transmitirla a la computadora unas 20 veces más rápido que un CD-ROOM. El DVD, denominado también disco de Súper Densidad

(SD) tiene una capacidad de 8,5 gigabytes de datos o cuatro horas de vídeo en una sola cara. En la actualidad, están desarrollándose discos del estilo del DVD regrabables y de doble cara.

Cintas Magnéticas:

Utilizados por los grandes sistemas informáticos.

1.1.4-Dispositivos de Salida:

Estos dispositivos permiten al usuario ver los resultados de los cálculos o de las manipulaciones de datos de la computadora. El dispositivo de salida más común es el monitor, pantalla en la que se ve la información suministrada por el ordenador. Según el tipo se clasifican en VGA Monocromáticos, VGA Color, SVGA Color y UVGA Color.

La resolución se define como el número de puntos que puede representar el monitor por pantalla, en horizontal x vertical. Así, un monitor cuya resolución máxima sea de 1024x768 puntos puede representar hasta 768 líneas horizontales de 1024 puntos cada una, probablemente además de otras resoluciones inferiores, como 640x480 u 800x600. Cuan mayor sea la resolución de un monitor, mejor será la calidad de la imagen en pantalla, y mayor será la calidad (y por consiguiente el precio) del monitor.

Otro de los dispositivos de salida comunes es la impresora es la que permite obtener en un soporte de papel una copia visualizada, perdurable y transportable de la información procesada por un computador.

Las impresoras permiten tener un registro en papel de las operaciones realizadas.

Por último se puede hacer mención a el módem, el cual enlaza dos ordenadores transformando las señales digitales en analógicas para que los datos puedan transmitirse a través de las telecomunicaciones.

1.1.5-Red de Comunicaciones:

Un sistema computacional es un sistema complejo que puede llegar a estar constituido por millones de componentes electrónicos elementales. Esta naturaleza multinivel de los sistemas complejos es esencial para comprender tanto su descripción como su diseño. En cada nivel se analiza su estructura y su función en el sentido siguiente:

Estructura: La forma en que se interrelacionan las componentes

Función: La operación de cada componente individual como parte de la estructura.

Por su particular importancia se considera la estructura de interconexión tipo bus. El bus representa básicamente una serie de cables mediante los cuales pueden cargarse datos en la memoria y desde allí transportarse a la CPU. Por así decirlo es la autopista de los datos dentro del PC ya que comunica todos los componentes del ordenador con el microprocesador. El bus se controla y maneja desde la CPU.

1.1.5 Modelo Von Neumann.

Computadoras digitales convencionales tienen una forma común que es atribuida a John von Neumann (1903-1957). El modelo von Neumann consiste de cinco componentes como los ilustrados en la figura 1.1.5. La unidad de entrada provee instrucciones y datos al sistema, los cuales están almacenados en la unidad de memoria. Las instrucciones y datos son procesados por la Unidad Aritmética y Lógica (ALU) bajo la dirección de la Unidad de Control. Los resultados son enviados a la Unidad de Salida. La ALU y la unidad de control son frecuentemente llamadas en conjunto: Unidad Central de Proceso (CPU). Computadoras más comerciales pueden ser divididas en estas cinco unidades básicas.

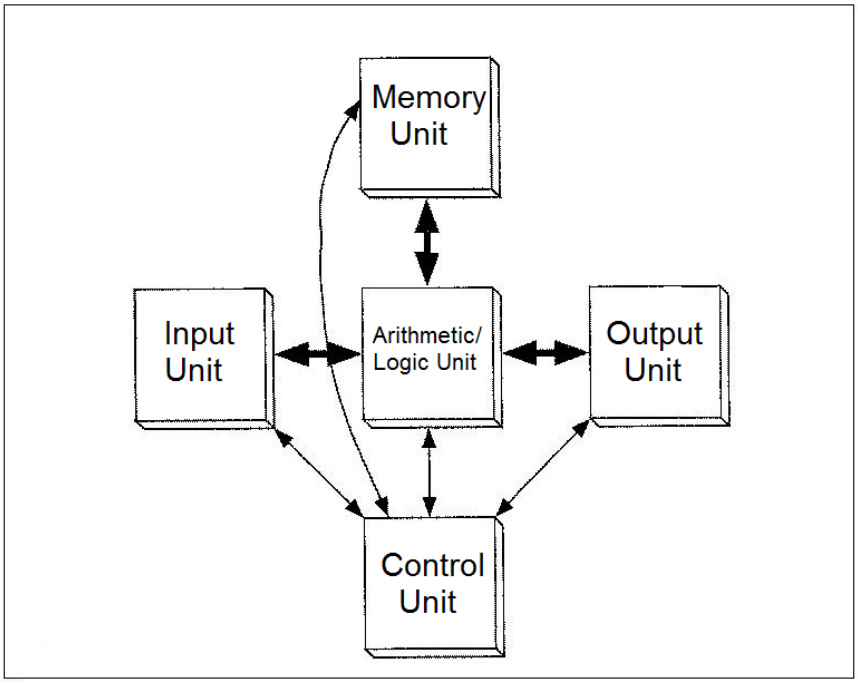


Figura 1.1.5 Modelo Von Neumann.

La ejecución de un programa almacenado es el aspecto más importante del modelo von Neumann. Un programa es almacenado en la memoria de la computadora a través y con los datos a ser procesado. Aunque demos por hecho lo anterior, para el almacenamiento del programa de la computadora, los programas son cargados en un medio externo, como paneles o tarjetas perforadas. El programa puede ser manipulado como si fueran sólo datos, esto brinda un aumento a los compiladores y sistemas de operación y hace posible la gran versatilidad de las computadoras modernas como las conocemos actualmente.

1.2 SECUENCIADORES Y CARTAS ASM.

1.2.1 MAQUINAS DE ESTADOS.

El modelo de máquina de estados contiene los elementos necesarios para describir la conducta de un sistema en términos de entradas, salidas y tiempo.

En el siguiente diagrama se representa un modelo general de una máquina de estados (figura 1.2.1).

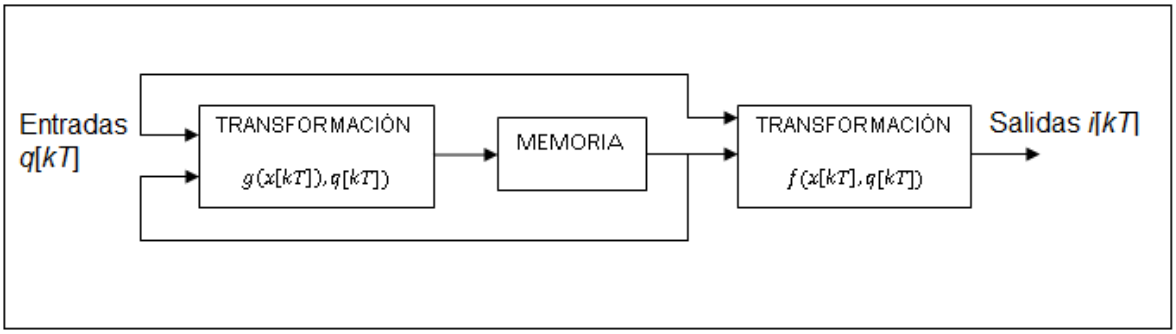


Figura 1.2-1 Modelo general de una máquina de estados.

x[kT], representa el estado en tiempo kT.

x[(k+1)T]=g(x[kT], q[kT]), representa el siguiente estado.

q[kT], representa las entradas en el tiempo kT.

i[kT]=f(x[kT], q[kT]), representa las salidas en el tiempo kT.

T es el periodo de duración de cada estado y k es un contador entero.

1.2.2 REPRESENTACIÓN DE ESTADOS.

El estado de una maquina de estados es la memoria de la historia pasada, suficiente para determinar las condiciones futuras se muestra la representación del estado.

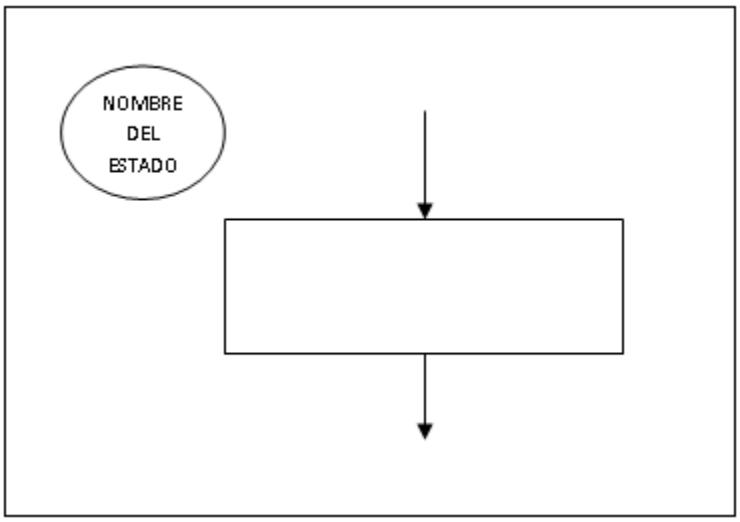


Figura 1.2-2 Representación de un estado.

Un estado se representa con un rectángulo y con su nombre simbólico en el externo superior, encerrado en un círculo.

1.2.3 REPRESENTACIÓN DE DECISIONES.

Las decisiones son usadas para seleccionar el camino que el algoritmo de la máquina de estados debe tomar de acuerdo a la variable o variables de entrada evaluadas. Las decisiones se representan a través de un rombo con el nombre de la variable a probar o una función que evalúe varias variables.

1.2.4 REPRESENTACIÓN DE SALIDAS.

Salidas no condicionales. Sirven para indicar la activación de una variable de salida. Para representarlas, se escriben dentro de un rectángulo de estado, los nombres de las variables de salida que se activan en ese estado. Las salidas no condicionales no dependen de las condiciones de entrada, solo dependen del estado actual.
Salidas condicionales. Estas salidas se presentan solamente cuando ciertas condiciones de entrada existen. Se representan con un óvalo y los nombres de las salidas dentro de él.

1.2.5 CARTAS ASM.

Las cartas AMS son una descripción gráfica del desarrollo de instrucciones en microoperaciones, también se dice que es un grafo orientado y cerrado cuyos nodos son bloques ASM, un bloque ASM es similar a un estado en un circuito secuencial síncrono; todas las acciones asociadas a un bloque ASM tienen lugar en el mismo ciclo de reloj. Un bloque ASM puede estar formado por tres tipos de elementos (gráficos):

- Una caja de estado.
- Cajas de decisión.
- Cajas de acción condicional.

Cajas de estado:
-Existe una y sólo una en cada bloque ASM.
-Especifica las acciones incondicionales del bloque ASM (son aquellas que se activan siempre que se ejecuta el bloque)
-Las acciones son transferidas entre registros= activación de señales de control.
A continuación se presenta un diagrama de Caja de Estado (figura 1.2.5)

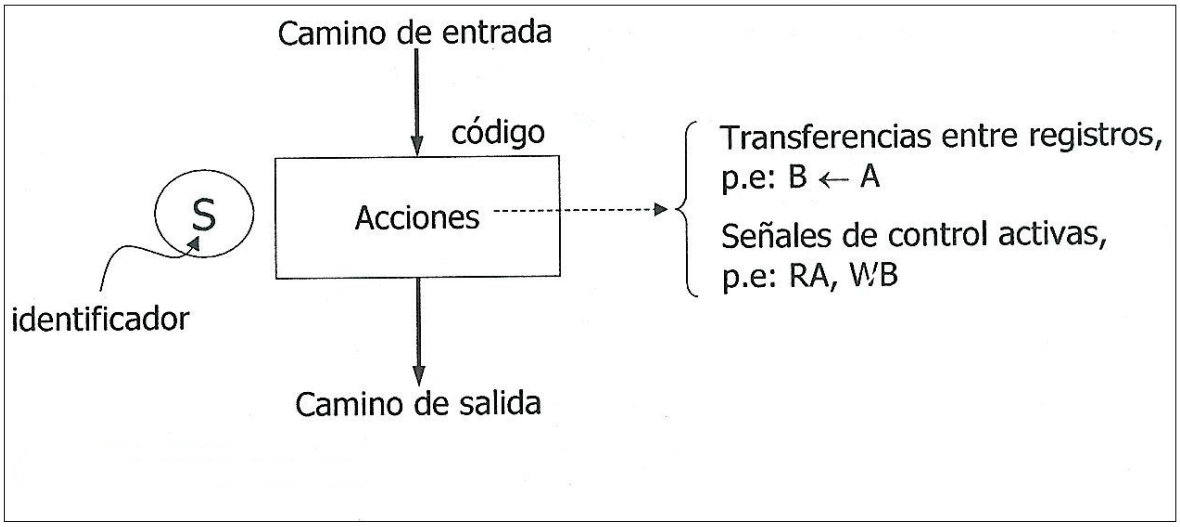


Figura 1.2-5 Cajas de estado.

Cajas de decisión:

-Puede haber varias en un mismo bloque, o ninguna.

-Permiten especificar bifurcaciones dentro del bloque ASM.

-Están caracterizadas por una condición sobre: las entradas de control externas del sistema, y las salidas de control de la unidad de datos. Diagrama de caja de decisión se muestra en la siguiente figura (figura 1.5)

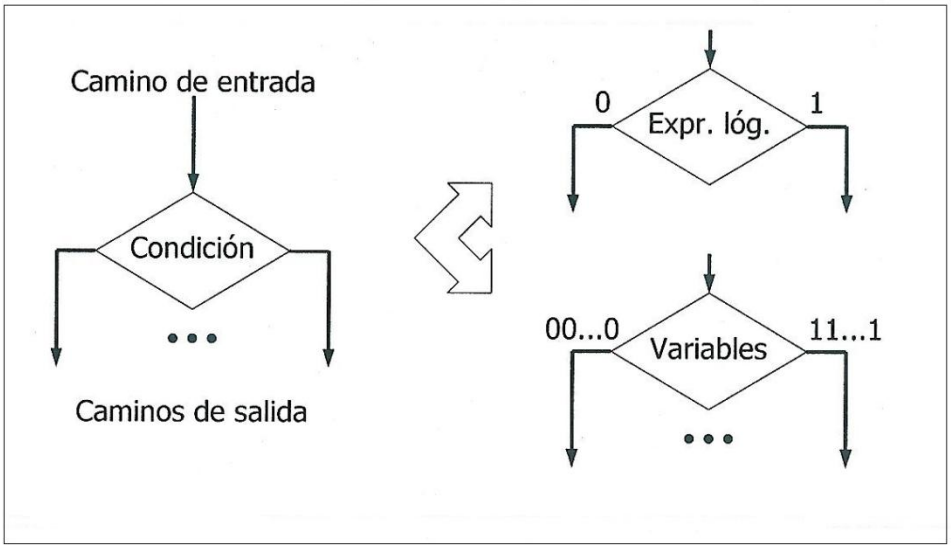


Figura 1.2-6 Caja de decisión.

Cajas de acción condicional:

-Puede haber varias en un mismo bloque.

-Especifica las acciones condicionales del bloque ASM (aquellas que se activan siempre que se ejecuta el bloque y además se cumpla una condición)

-Van siempre asociadas a las cajas de decisión.

-Las acciones son transferidas entre registros= activación de señales de control.

Representación de cajas de acción condicional en la figura 1.6

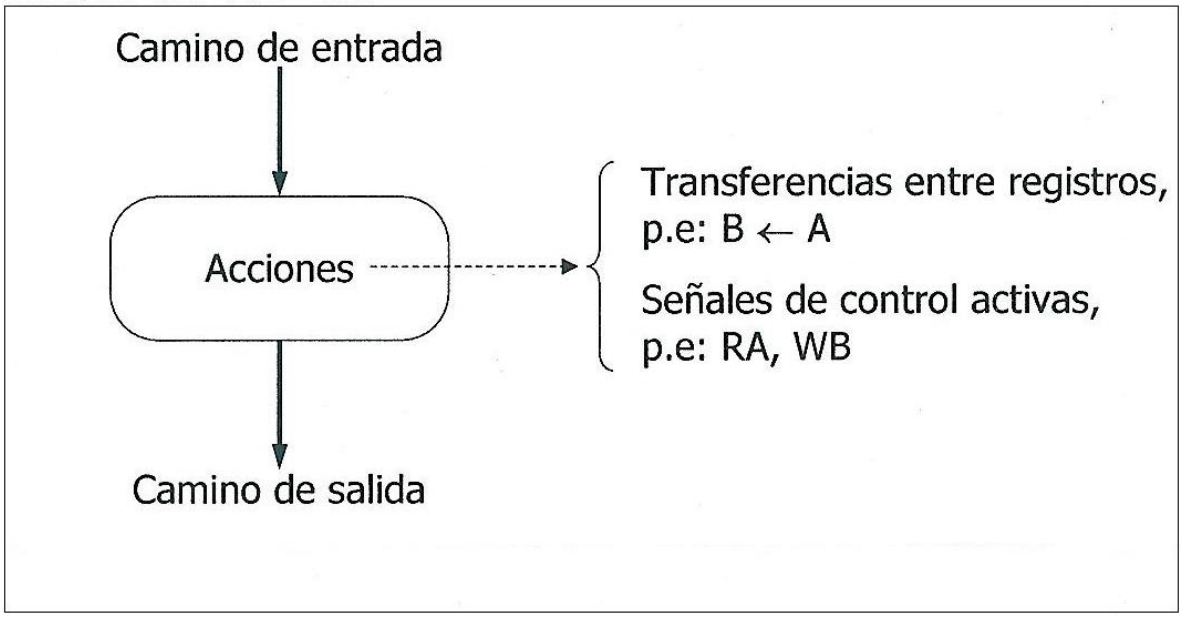


Figura 1.2-7 Caja de acción condicional.

SECUENCIADORES

El Secuenciador Básico.

En el diseño de módulos de control de una computadora es necesario contar con máquinas de estado capaces de ejecutar algoritmos complejos. Para ello se deben hacer modificaciones y agregar componentes a la variante del direccionamiento implícito para crear máquinas de estado que efectúen cartas ASM con llamadas a subrutinas, como las estructuras DO, WHILE, iteraciones tipo FOR, etc. Los dispositivos que son capaces de realizar este tipo de operaciones se llaman secuenciadores.

En la siguiente figura (figura 1.8) se muestra el diagrama de bloques de un secuenciador básico. En él se puede observar la dirección del estado siguiente, dada por el bus y puede proveer de dos lugares posibles.

1.- El registro contador, llamado de micro-programa (µPC), contiene la dirección del estado presente más uno. Es decir, la dirección que se encuentra en la salida del multiplexor es incrementada en una unidad y posteriormente cargada en este registro en el siguiente ciclo de reloj.

2.- En la entrada D se introduce una dirección de salto. Esta dirección puede provenir de tres diferentes lugares: del campo de liga, del registro de transformación o del registro de interrupciones.

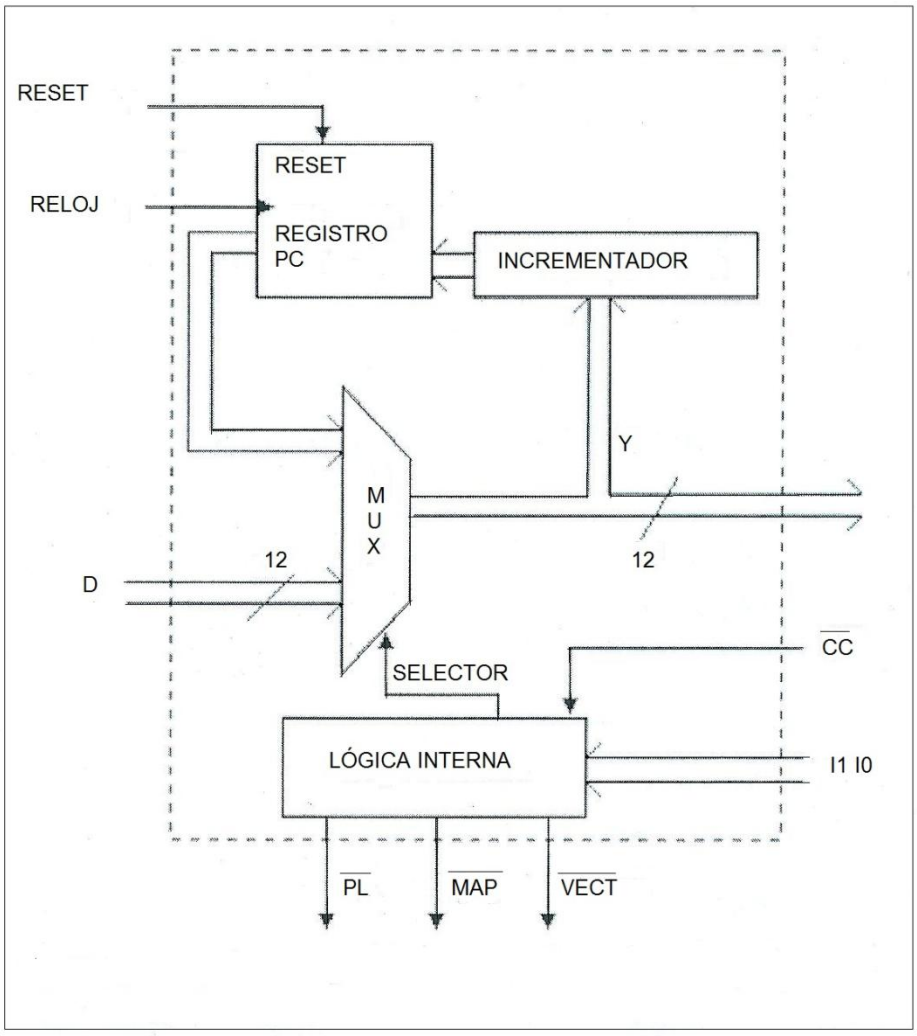


Figura 1.2-8 Diagrama de bloques de un secuenciador básico.

Un secuenciador está formado por una lógica interna que es la que se encarga de general señales que controlan al multiplexor. Dependiendo de la instrucción emitida por las líneas I_1 e I_0 y de la línea $\overline{CC}$, la lógica del secuenciador tiene la capacidad de seleccionar entre la salida del registro µPC o la entrada D. La salida direcciona una memoria que contiene el estado siguiente del algoritmo de la máquina de estados.

Las líneas $\overline{PL}$, $\overline{MAP}$ y $\overline{VECT}$, son generadas por la lógica interna, las cuales seleccionan unos registros en que las salidas están conectadas a la entrada D del secuenciador. Así la dirección de salto podrá venir de tres diferentes lugares. Al diseñar la unidad central de procesos (CPU) se utilizará esta característica.

La figura 1.2-9 muestra las señales de entrada y salida del secuenciador.

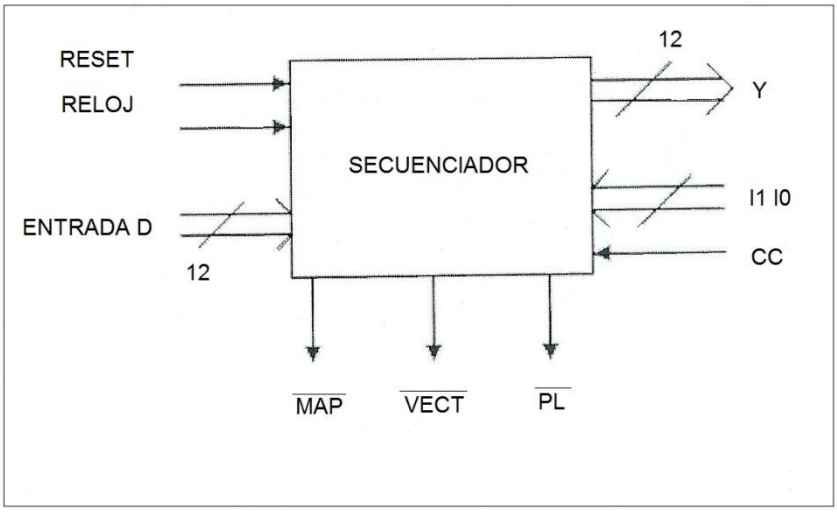


Figura 1.2-9 El secuenciador básico.

Instrucciones para el Secuenciador.

Continua (C)

En esta instrucción, la dirección del estado siguiente la proporciona el registro µPC.

Salto condicional (SCO)

Se realiza una revisión al valor de la línea $\overline{CC}$, cuando es igual a uno, la dirección del estado siguiente es proporcionada por el registro µPC; si es igual a cero, la dirección del estado siguiente, contenida en el registro seleccionado por $\overline{PL}$ se ingresa por la entrada D.

1.3 DATA PATH.

En esta sección se va a considerar el datapath y sus señales de control asociadas. Las instrucciones del conjunto que concentraremos y el formato de instrucciones para el conjunto ARC se mostrarán en la siguiente figura en la que se visualizan quince instrucciones que son agrupadas en cuatro formas. El registro del estado del procesador también es mostrado en la parte inferior de la figura.

Mnemonic	Meaning
ld	Load a register from memory
st	Store a register into memory
sethi	Load the 22 most significant bits of a register
andcc	Bitwise logical AND
orcc	Bitwise logical OR
orncc	Bitwise logical NOR
srl	Shift right (logical)
addcc	Add
call	Call subroutine
jmpl	Jump and link (return from subroutine call)
be	Branch if equal
bneg	Branch if negative
bcs	Branch on carry
bvs	Branch on overflow
ba	Branch always

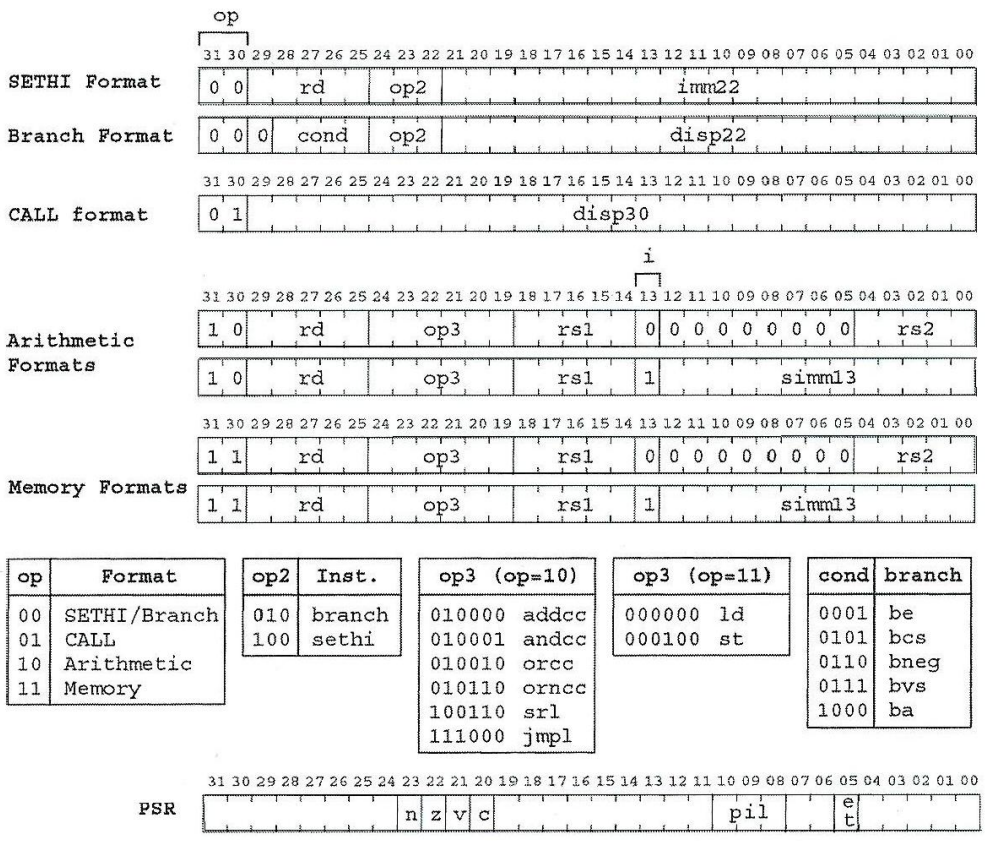


Figura 1.3 Conjunto de instrucciones para ARC, cortesía de Computer Architecture and Organization.

1.3.1 Visión general de datapath.

Un datapath es una colección de unidades funcionales, tales como unidades de aritmética lógica o multiplicadores que realizan el desarrollo de operaciones. La mayoría de las unidades centrales de proceso consistirá en un camino de datos y una unidad de control, con una gran medida de la unidad de control dedicado a la regulación de la interacción entre el camino de datos y la memoria principal.

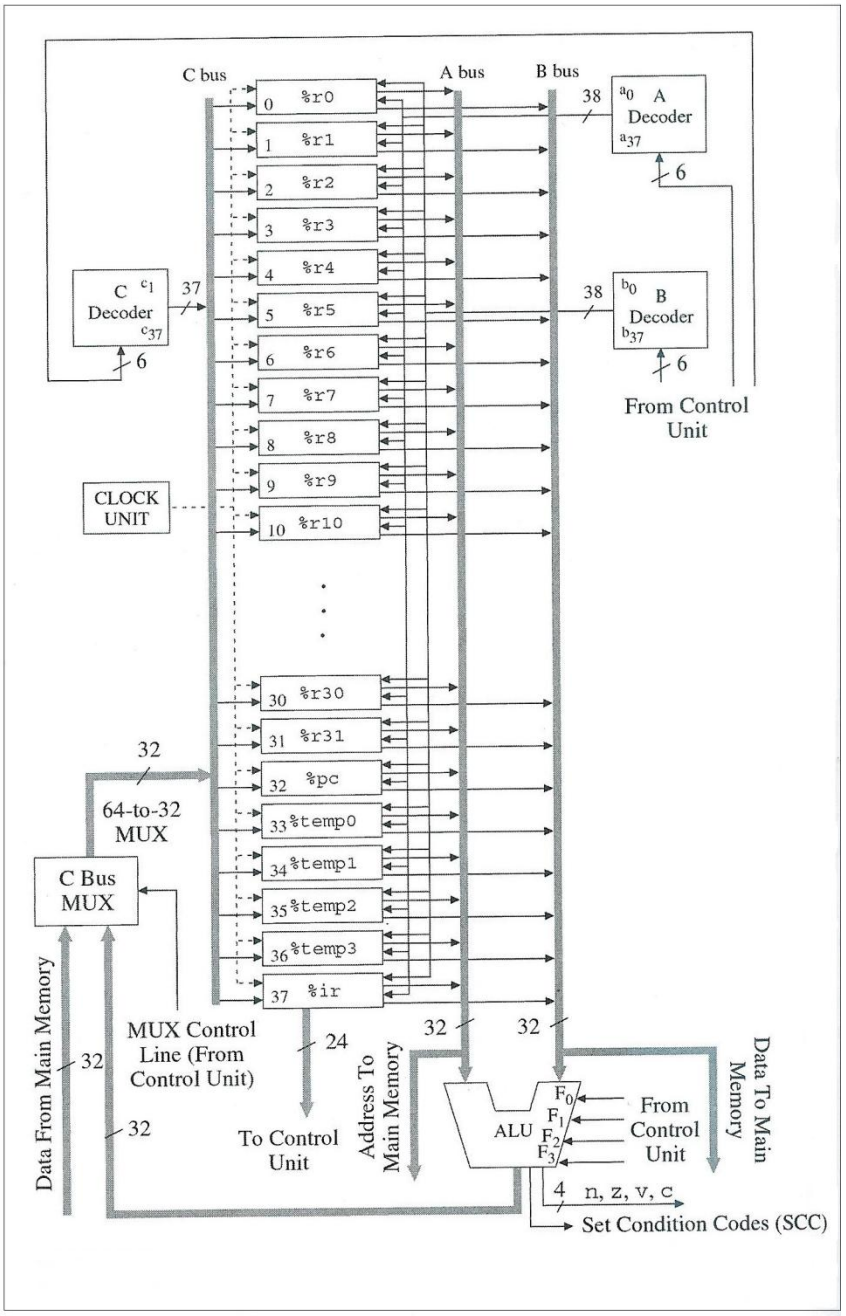


Figura 1.3-1 Datapath de ARC, Cortesía de Computer Architecture and Organization.

1.4 UNIDAD DE CONTROL.

La unidad de control de la computadora tiene como funcionalidad la decodificación de las instrucciones en lenguaje ensamblador y de ejecutar las micro-operaciones que sean necesarias (representadas a través de cartas ASM) para llevarlas a cabo. Para cada instrucción en ensamblador, se activarán secuencialmente una serie de señales que indican a los diferentes componentes de la arquitectura la operación que deben realizar. El secuenciador es la parte fundamental de la unidad de control como se muestra en la figura 1.4

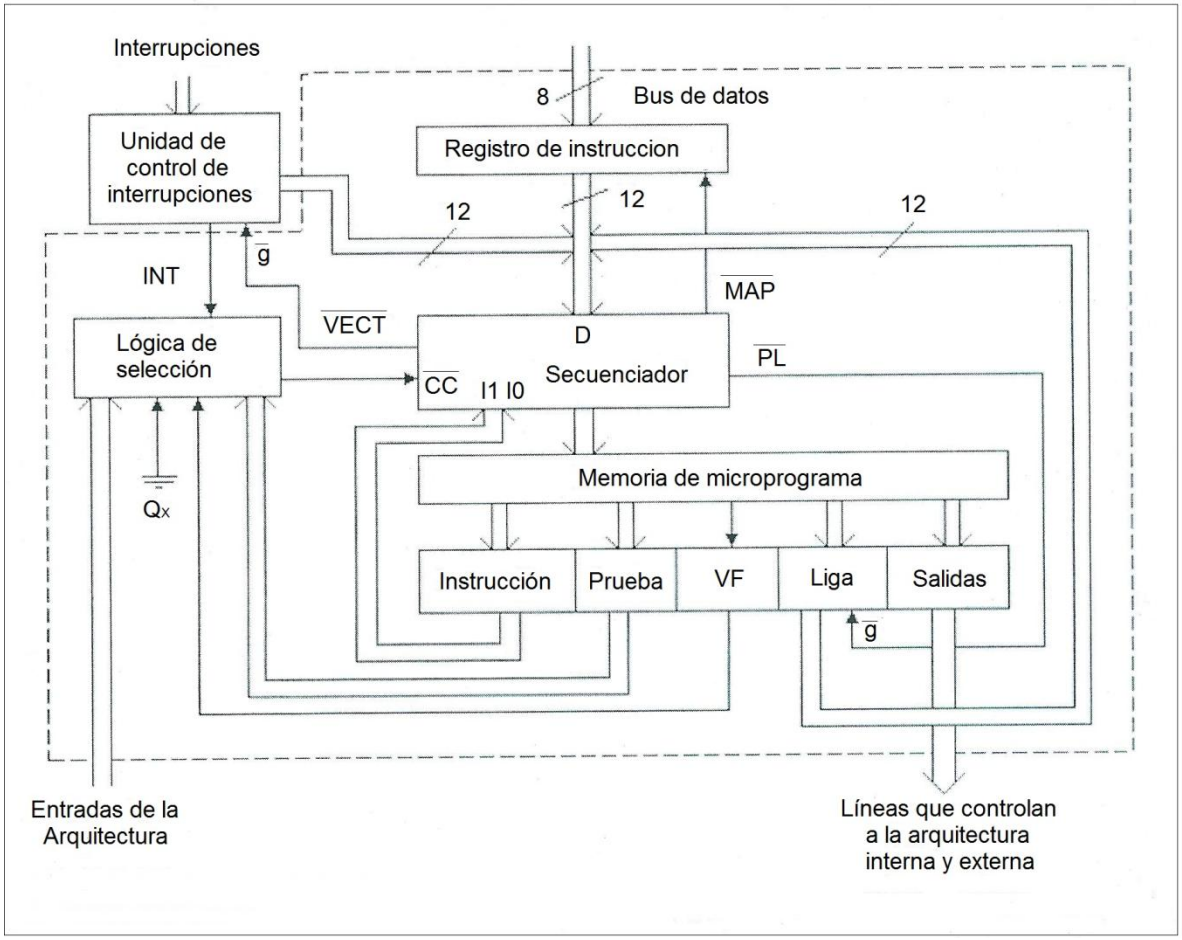


Figura 1.4 Unidad de control de la computadora.

Si el procesador es el núcleo del sistema de computación, la unidad de control lo es del procesador. Tiene tres funciones principales.

- Leer o interpretar instrucciones del programa.
- Dirigir la operación de los componentes internos del procesador.
- Controlar el flujo de programas y datos hacia y desde la RAM.

La unidad de control dirige otros componentes del procesador para realizar las operaciones necesarias y ejecutar la instrucción.

Por otra parte, las entradas de la arquitectura indican el estado en el que se encuentra tanto la arquitectura interna como la externa, y sirven para que el secuenciador pueda tomar decisiones de acuerdo a ciertos criterios. Las entradas son seleccionadas en el bloque de lógica de selección, a través del campo de prueba. La línea de INT también se conecta a la lógica de selección para revisar si existe alguna petición de interrupción.

Existe otra alternativa para diseñar la unidad de control, y es utilizando los lenguajes de descripción de hardware como Verilog HDL, VHDL ó AHDL. Con uno de estos lenguajes y el código de la instrucción que se desea ejecutar, es posible describir los pasos para ejecutar dicha instrucción.

1.5 PIPELINE.

Para lograr paralelismo sin repetir componentes se utiliza un método denominado "línea de ensamblaje" (pipeline). En este caso la computadora se encuentra dividida en varias etapas, en la cual cada etapa realiza operaciones sobre instrucciones diferentes.

La arquitectura en pipeline (basada en filtros) consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.

Esta arquitectura es muy común en el desarrollo de programas para el intérprete de comandos, ya que se pueden concatenar comandos fácilmente con tuberías (pipe).

También es una arquitectura muy natural en el paradigma de programación funcional, ya que equivale a la composición de funciones matemáticas.

El flujo de instrucciones a través de una tubería sigue los pasos tomados cuando la instrucción es ejecutada.

La primera etapa trae la instrucción, la segunda, decodifica la instrucción, la tercera, trae los operandos de la instrucción y por último la cuarta, ejecuta la instrucción (Figura 1.5).

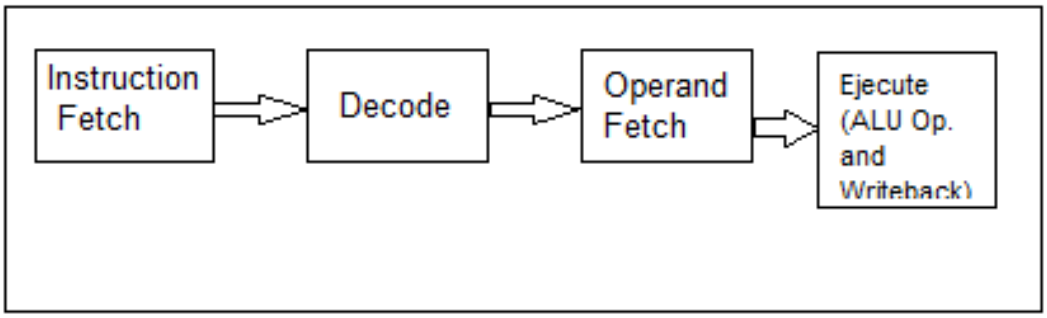


Figura 1.5 Etapas de una instrucción por tubería.

Para cada diseño se debe tener un enfoque de la conducción o tubería desde una perspectiva, dependiendo sobre el particular diseño, objetivos y el set de

instrucciones. Por ejemplo la implementación original SPARC tuvo sólo cuatro tuberías, mientas que algunas tuberías de puntos flotantes pueden tener una docena de etapas.

Cada una de las unidades de ejecución realiza una operación diferente en busca de cerrar el ciclo. Después la instrucción busca que la unidad termine su tarea para que sea llevada a la unidad decodificadora. Hasta este punto la unidad de instrucción buscada puede empezar a buscar la siguiente instrucción, la cual se superpone con la decodificación de la instrucción previa. Cuando la unidad de instrucción y de decodificación completan sus tareas, llevan las otras tareas a la siguiente unidad (el operando es la siguiente unidad a codificar). El flujo de control continúa hasta que todas las unidades están llenadas.

1.5.1 Mantenimiento de Pipeline.

Se debe tomar en cuenta un punto importante: aunque tiene varios pasos para ejecutar una instrucción en este modelo, en promedio, una instrucción puede ser ejecutada por ciclo siempre y cuando la tubería (pipeline) se mantenga llena. La tubería no se queda llena a menos que seamos cuidadosos en cuanto a cómo están ordenadas las instrucciones. En la siguiente tabla (Tabla 1.5.1) se puede visualizar que aproximadamente una de cada cuatro instrucciones es una rama.

Statement	Average Percent of Time
Assignment	47
If	23
Call	15
Loop	6
Goto	3
Other	*1*

Tabla 1.5-1 Frecuencia de aparición de tipos de instrucciones para una variedad de idiomas.

Los porcentajes no suman cien debido al redondeo.

Si bien es cierto, no podemos buscar la instrucción que siga a la rama hasta que ésta haya completado su ejecución. En consecuencia tan pronto como la tubería se llena, la rama es encontrada y luego la tubería tiene que limpiarse llenándola con no operaciones (NOPs). Estas NOPs están a veces referidas como “burbujas”. Una situación similar se presenta con las instrucciones: LOAD y STORE. Ellas generalmente requieren un ciclo de reloj adicional en el cual accesan a la

memoria, que tiene el efecto de ampliar la fase de ejecución de un ciclo a dos ciclos, los ciclos de espera son llenados con NOPs.

En la siguiente tabla (Tabla 1.5.2) se ilustra el comportamiento de la tubería durante la referencia de memoria y también durante la rama del conjunto de instrucción ARC. La instrucción *addcc* entra a la tubería en primer ciclo, para el segundo ciclo la instrucción *ld* que se refiere a la memoria, entra a la tubería y *addcc* se mueve a la etapa de decodificación. La tubería continúa llenándose con las instrucciones *srl* y *subcc* en los ciclos tres y cuatro respectivamente. En el cuarto ciclo la instrucción *addcc* es ejecutada y abandona la tubería. Para el quinto ciclo la instrucción *ld* alcanza el nivel de ejecución, pero no termina la ejecución porque un ciclo adicional es requerido para referencia de memoria. La instrucción *ld* termina de ejecutarse durante el sexto ciclo.

	1	2	3	4	5	6	7	8
Instruction Fetch	adcc	ld	srl	subcc	be	nop	nop	nop
Decode		addcc	ld	srl	subcc	be	nop	nop
Operand Fetche			addcc	ld	srl	subcc	be	nop
Execute				addcc	ld	srl	subcc	be
Memory Reference						ld		

Tabla 1.5-2 Comportamiento de la tubería durante una referencia de memoria.

1.5.2 Riesgos en Pipeline.

Los riesgos en tuberías son los menos problemáticos en el mantenimiento de éstas cuando están llenas. Un riesgo por ejemplo es la situación cuando un resultado es generado por una instrucción no disponible y dicha instrucción es requerida por una instrucción siguiente en la tubería. El potencial de daños en tuberías existe cuando las instrucciones están realizándose, es decir, en algún momento de la ejecución.

1.6 PARALELISMO.

El paralelismo es una forma de computación en que muchos cálculos se llevan a cabo al mismo tiempo, que operan en el principio de que los problemas grandes, se pueden dividir en otros más pequeños, que se resuelven al mismo tiempo ("en paralelo"). Hay varias formas diferentes de paralelismo: a nivel de bits , nivel de instrucción, los datos y paralelismo de tareas. Máquinas paralelas pueden ser clasificadas según el nivel en que el hardware es compatible con el paralelismo en procesadores de computadoras.

Un método para la mejora del rendimiento de un procesador es aumentar la frecuencia de reloj. Esto funciona hasta un punto en que nuestro límite de energía se vuelve incontrolable (mayor frecuencia de reloj consume más energía) para ello se requiere una adecuada ventilación a través de un disipador de calor y un ventilador.

Una alternativa para aumentar la frecuencia de reloj será incrementando el número de procesadores, y se descomponen y distribuyen en un programa único para el procesador. Este enfoque es conocido como procesamiento en paralelo en el cual un número de procesadores trabajan colectivamente, en paralelo para una tarea común.

1.6.1 Midiendo el desempeño.

Una arquitectura paralela se puede caracterizar en términos de tres parámetros: (1) el número de elementos de proceso, (2) la red de interconexión entre los elementos de proceso, y (3) la organización de la memoria. Como ejemplo en las cuatro etapas de una instrucción por pipeline que se muestra en la figura 1.5 existen cuatro elementos de proceso. La interconexión de la red entre los elementos de proceso es un anillo, cabe mencionar que la memoria RAM es una corriente externa al pipeline.

La caracterización de la arquitectura de un procesador en paralelo es una tarea relativamente fácil, pero la medición del rendimiento no es tan sencillo. Aunque podemos medir fácilmente el aumento de la velocidad con un acercamiento al pipeline, la aceleración global depende de los datos: no todos los programas y datos conjuntan el grupo de una arquitectura pipeline.

Las medidas comunes de rendimiento son el tiempo de paralelismo, la aceleración, la eficiencia y el rendimiento. El tiempo de paralelismo es simplemente el tiempo necesario para ejecutar un programa en un procesador en paralelo. El aumento a la velocidad (aceleración) es la relación del tiempo en que el programa se ejecuta en un procesador secuencial (es decir, no paralelo), al tiempo que ese mismo programa se ejecuta en un procesador paralelo. De una forma sencilla representaremos la aceleración (S, en el contexto de procesamiento paralelo) como:

$$S = \frac{T_{secuencial}}{T_{paralelo}}$$

Desde un algoritmo secuencial y una versión paralela del mismo algoritmo, puede ser programado muy diferente para cada máquina, para ello requerimos calificar $T_{secuencial}$ y $T_{paralelo}$ para que se apliquen a la mejor implementación para cada máquina.

Por ejemplo, si queremos un aumento de la velocidad cercana a cien, no será suficiente distribuir cien procesadores en el programa a ejecutar. El problema es que no todos los cálculos son fáciles de descomponer en una forma en que se aprovechen al máximo. Incluso si hay un pequeño número de operaciones secuenciales en un programa paralelo, entonces la velocidad puede ser limitada considerablemente. Esto se resume en la ley de Amdahl en que la aceleración se

expresa en términos del número de procesadores *p* y la fracción *f* de las operaciones que deben realizarse en forma secuencial:

$$S = \frac{1}{f + \frac{1-f}{p}}$$

Por ejemplo, si *f*= 10% de las operaciones deben realizarse en forma secuencial, así, la aceleración puede ser superior a diez, independientemente de cuántos procesadores se utilizan.

$$S = \frac{1}{0.1 + \frac{0.9}{10}} \cong 5.3 \qquad S = \frac{1}{0.1 + \frac{0.9}{\infty}} = 10$$

p=10 procesadores *p*= ∞ procesadores

Esto nos lleva a medir la eficiencia. Eficiencia la definimos como la relación entre el aumento de la velocidad con el número de procesadores que se utilizan. Para un aumento de velocidad de 5.3 usando 10 procesadores, la eficiencia es:

$$\frac{5.3}{10} = 0.53 \text{ ó } 53\%$$

Si doblamos el número de procesadores a 20, entonces el aumento de velocidad se elevará a 6.9 pero la eficiencia será reducida a 34%. Por lo tanto al paralelizar un algoritmo podemos mejorar el rendimiento teniendo como límite la cantidad de operaciones secuenciales.

Así la eficiencia se reduce drásticamente si el aumento de velocidad se acerca a su límite, por lo que no tiene sentido utilizar simplemente más procesadores con la esperanza de obtener una ganancia.

El rendimiento es una medida de cuánto se logra con el tiempo, y es de especial interés para las entradas y salidas de la tubería. Para el caso de una tubería de cuatro etapas que esté llena en el que cada etapa de la tubería completa su tarea en 1ns, el tiempo medio para acompletar una operación es de 1 ns a pesar de que se necesitan 4 ns para ejecutar cualquier operación. El rendimiento general para esta situación es entonces:

$$1.0 \frac{operacion}{ns} = 10^9 operaciones\ por\ segundo$$

1.6.2 LA TAXONOMÍA DE FLYNN.

La arquitectura de computadoras puede ser clasificada en términos del flujo de datos e instrucciones, usando la taxonomía desarrollada por M. J. Flynn (Flynn 1972). Un procesador secuencial convencional encaja en la categoría del flujo de instrucciones y secuencia de datos (SISD) como se ilustra en la figura 1.6-b en que una sola instrucción se ejecuta al momento en un procesador SISD, aunque la tubería puede permitir que varias instrucciones estén en diferentes fases de ejecución en un momento dado.

En un procesador con flujo de instrucciones, flujo múltiple de datos SIMD (single instruction stream), varios procesadores idénticos realizan la misma secuencia de operaciones en diferentes conjuntos de datos, como los ilustrados en la figura 1.6-

b. El sistema SIMD aunque puede ser usado como una sala de llenado, todas las piezas de clasificación diferentes se encuentran en el mismo conjunto de contenedores.

En un flujo de instrucciones múltiples y datos múltiples (MIMD), el procesador realiza diferentes operaciones en distintos conjuntos de datos, pero todas están coordinadas para ejecutarse en un programa paralelo, como se muestra en la figura 1.6-c

El flujo de instrucciones múltiples y datos único (MISD), una secuencia de datos única es operada por varias unidades funcionales, como se ilustra en la figura 1.6-d. El flujo de datos suele ser un vector de varias secuencias relacionadas. Esta configuración es conocida como arreglo o matriz sistólica.

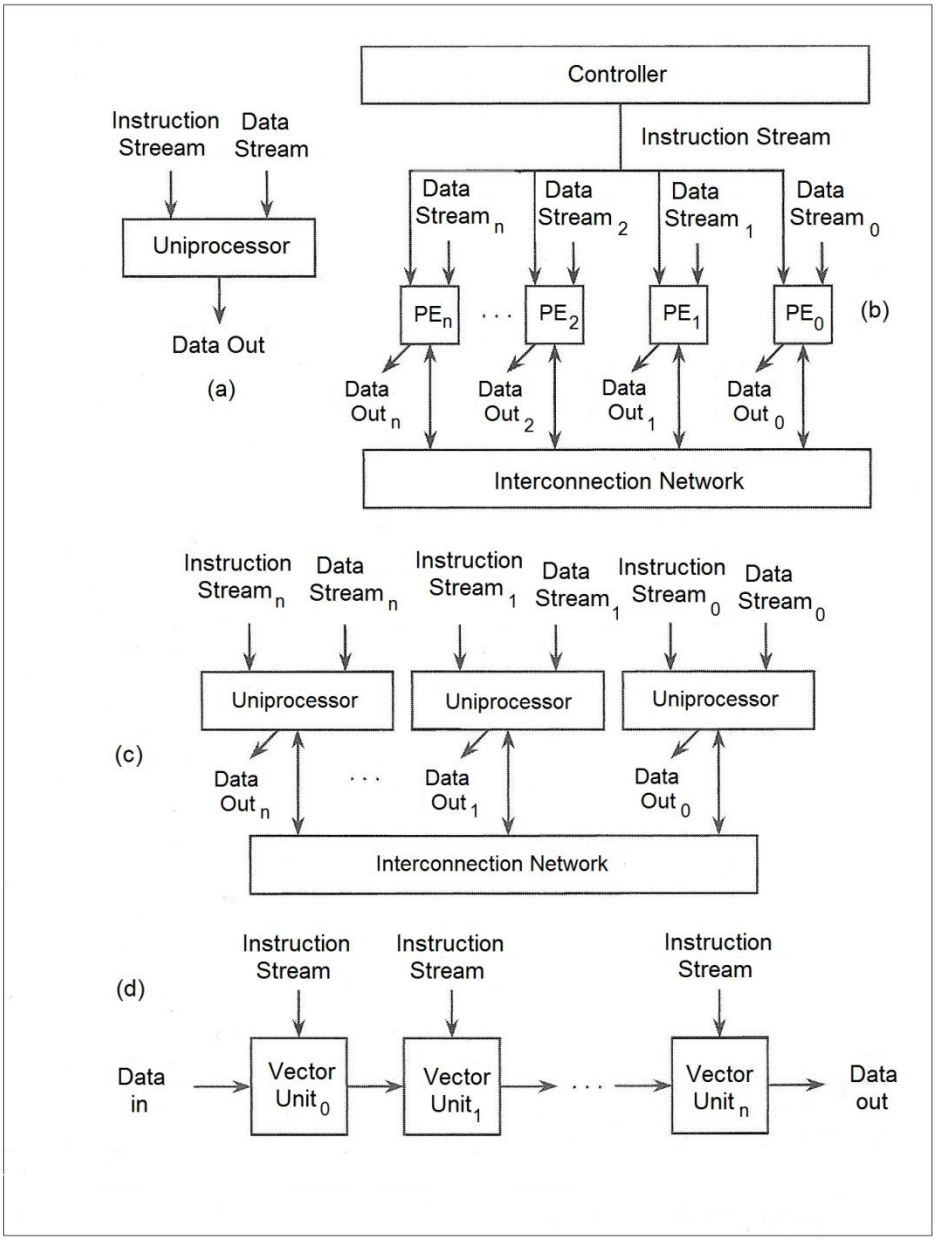


Figura 1.6-1 Clasificación de Arquitecturas de acuerdo a la taxonomía de Fynn (a) SISD; (b) SIMD; (c) MIMD; (d) MISD.

1.6.3 Redes de interconexión.

Cuando la computadora es distribuida por encima de sus elementos de proceso (PEs), ellos necesitan comunicarse con cada uno de ellos a través de una red de interconexión. Existe una variedad de topologías para redes de interconexión, cada una con sus propias características en términos de la complejidad de sus puntos de cruce, diámetro y bloqueo.

Una de las redes más poderosas es la *"crossbar",* en la cual cada elemento de proceso está directamente conectado con los demás elementos de proceso. A continuación se explicarán algunas topologías más representativas para la configuración de redes.

En el otro extremo tenemos la topología *"bus"* la cual es ilustrada en la figura 1.6.1-b, con ella se comparte una parte del ancho de banda entre los elementos de proceso.

En la topología *"ring"* tenemos *N* puntos de cruce para *N* elementos de proceso como se muestran en la figura 1.6.1-c. Cada punto de cruce está contenido dentro de los elementos de proceso.

Para la topología *"mesh"* se tienen *N* puntos de cruce para *N* elementos de proceso pero el diámetro es $2\sqrt{N,}$ como se muestra en la Figura 1.6.1-d. Todos los elementos de proceso pueden estar comunicados simultáneamente en solo $3\sqrt{N}$ pasos.

En la topología *"star"* tenemos un eje central a través del cual todos los elementos de proceso están comunicados (Figura 1.6.1-e).

"tree" es la topología en que tenemos *N* puntos de cruce para *N* elementos de proceso y el diámetro es $2\log_2 N - 1$ (Figura 1.6.1-f). Es efectiva para aplicaciones en las cuales hay una gran cantidad de datos distribuidos.

En la topología *"perfect shuffle"* también tenemos *N* puntos de cruce para *N* elementos de proceso (Figura 1.6.1-g). El diámetro es $\log_2 N$ y pasa a través de la red a conectar cada uno de los elementos de proceso.

Finalmente *"hypercube"* tiene *N* puntos de cruce para *N* elementos de proceso, con diámetro de $\log_2 N - 1$ (Figura 1.6.1-h). El menor número de puntos de cruce con respecto a la topología *perfect shuffle* y está balanceado porque tiene una complejidad mayor en los elementos de proceso.

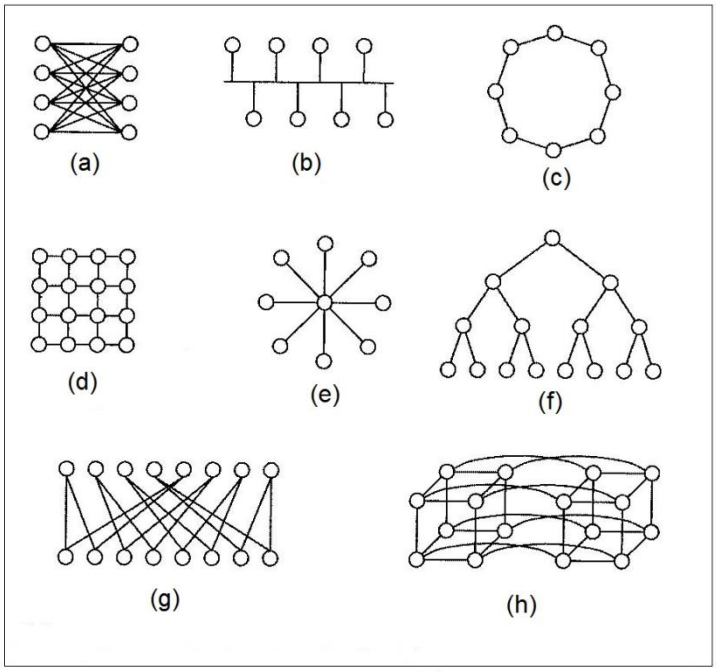


Figura 1.6-2 Topología de redes: (a) crossbar; (b) bus; (c) ring; (d) mesh; (e) star; (f) tree; (g) perfect shuffle; (h) hypercube. Cortesía de Computer Arqchitecture and Organization.

1.6.4 Asignación de Algoritmo en una arquitectura paralela.

El proceso de asignación de un algoritmo en una arquitectura paralela comienza con un análisis de dependencia en que la dependencia de datos entre las operaciones en un programa son identificadas. Consideramos el código C mostrado en la Figura 1.6-3. En un procesador SISD, las cuatro declaraciones numeradas requieren cuatro tiempos para ser completadas, como las ilustradas en la Figura 1.6-4

```
                func(x, y)   /* Compute (x² + y²) × y² */
Número de       int x, y;
operaciones     {
                int temp0, temp1, temp2, temp3;

          0     temp0 = x * x;
          1     temp1 = y * y;
          2     temp2 = temp0 + temp1;
          3     temp3 = temp1 * temp2;

                return(temp3);
                }
```

Figura 1.6-3 Función C en computadoras $(x^2 + y^2)\times y^2$. Cortesía de Computer Arqchitecture and Organization.

El gráfico de dependencias mostrado en la Figura anterior (Figura 1.6-3) expone un paralelismo natural en el control de secuencias. También este gráfico fue creado para la asignación de cada operación en el programa original a un nodo en el gráfico, y luego un trazo se dirige desde cada nodo que produce un resultado al nodo que lo necesita.

La secuencia de control requiere cuatro pasos para completarse, pero el gráfico de dependencias muestra que el programa puede ser completado en solo tres pasos, desde operaciones 0 y 1 no dependen de cada uno y pueden ser ejecutados simultáneamente.

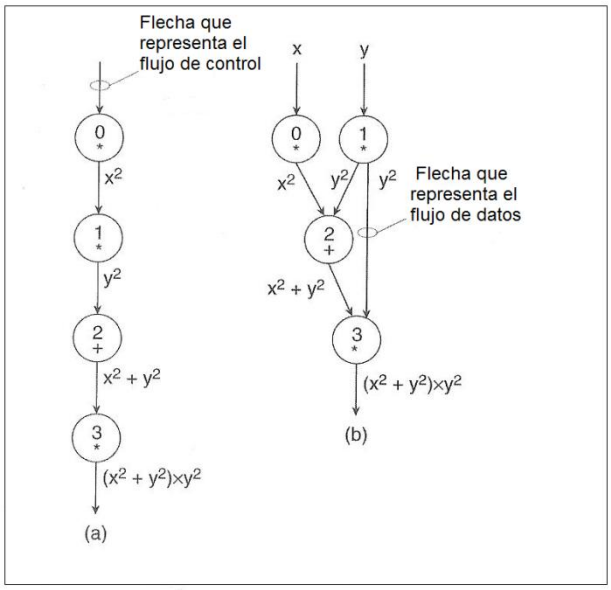


Figura 1.6-4 (a) Control de Secuencia para el programa C; (b) Dependencia gráfica para el programa C. Cortesía de Computer Arqchitecture and Organization.

1.7 CPLD´S, FPGA´S Y VHDL.

1.7.1 CPLD´S.

Un CPLD (Complex Programmable Logic Device) extiende el concepto de un PLD, pero con un mayor nivel de integración, ya que permite implementar sistemas con un mejor desempeño porque utilizan menor espacio, mejoran la confiabilidad en el circuito, y reducen costos. Un CPLD se forma a través de múltiples bloques lógicos, cada uno similar a un PLD. Los bloques lógicos se comunican entre sí utilizando una matriz programable de interconexiones lo cual hace más eficiente el uso del silicio, conduciendo a un mejor desempeño y un costo reducido. A continuación se explican brevemente las principales características de la arquitectura de un CPLD (Figura 1.70).

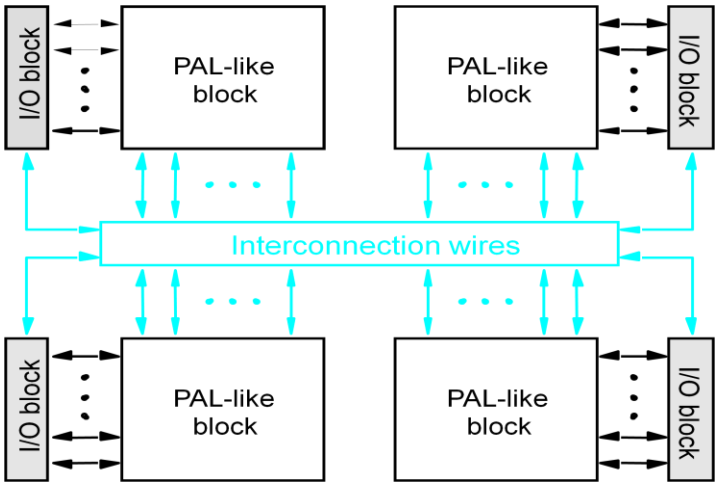


Figura 1.7.1-1 Estructura de un CPLD.

- Alto rendimiento, los dispositivos de lógica programable basados en EEPROM (PLD´s) están basados en la segunda generación de arquitectura MAX.
- Pueden trabajar desde los 2.5 hasta 5 [V].
- El MAX7000S cuenta con 128 o más macrocélulas.
- Retraso lógico de 5ns con una frecuencia máxima de 175.4 MHz

Cuentan con un ahorro de energía programable con una reducción de más del 50% en cada macrocélula, también cuentan con 44 a 208 pines disponibles en el chip. Tiene un soporte de diseño de software y automático en un lugar y ruta proporcionada por Altera en el desarrollo para PC basado en Windows y Sun (Tabla 1.7).

Características	**EPM7032S**	**EPM7064S**	**EPM7128S**	**EPM7160S**	**EPM7192S**	**EPM7256S**
Compuertas Utilizables	600	1250	2500	3200	3750	5000
Macrocelulas	32	64	128	160	192	156
Matriz Lógica de bloques	2	4	8	10	12	16
Pines Máximos I/O	36	68	100	104	124	164
$t_{PD}(ns)$	5	5	6	6	7.5	7.5
$t_{SU}(ns)$	2.9	2.9	3.4	3.4	4.1	3.9
$t_{FSU}(ns)$	2.5	2.5	2.5	2.5	3	3
$t_{CO1}(ns)$	3.2	3.2	4	3.9	4.7	4.7
$f_{CNT}(MHz)$	175.4	175.4	147	149.3	125	90.9

Tabla 1.7.1 Características de la Familia MAX 7000S. Cortesía Altera Corporation.

La familia de alta densidad MAX 7000 tienen PLD´s de alto rendimiento que se basan en la segunda generación de la arquitectura Altera MAX. Fabricados con tecnología avanzada CMOS y basados en EEPROM, la familia MAX 7000 tiene entre 600 y 5000 compuertas utilizables. Su retraso entre pines es de sólo 5ns con velocidades de hasta 151,5 MHz. (Ver tabla 1.7.2).

Dispositivo	Grado de Velocidad									
	-5	-6	-7	-10P	-10	-12P	-12	-15	-15T	-20
EPM7032	√	√	√		√		√	√	√	
EPM7032S		√	√		√					
EPM7064		√	√		√		√	√		
EMP7064S	√	√	√		√					
EPM7096			√		√		√	√		
EPM7128E			√	√	√		√	√		√
EPM7128S		√	√		√			√		
EPM7160E				√	√		√	√		√
EPM7160S		√	√		√			√		

Tabla 1.7.2 Grado de Velocidades de la Familia MAX 7000S. Cortesía Altera Corporation.

Existen versiones mejoradas en que los dispositivos tienen reloj mundial adicional, pueden permitir controles, aumento de interconexión de los recursos, registros de entrada rápida, este tipo de dispositivos soporta TTL al 100%.

La familia de dispositivos MAX 7000 utilizan células CMOS EEPROM para implementar la lógica de funciones. El usuario puede configurar este tipo de familias ya que tiene capacidad en su arquitectura para una variedad de funciones independientes de lógica combinatoria y secuencial. Los dispositivos pueden ser programados para las iteraciones rápidas y eficaces en el desarrollo de diseño y ciclos de depuración, lo mejor es que se puede programar y borrarse hasta 100 veces.

El MAX 7000 proporciona velocidad programable / potencia optimizada. Partes críticas de la velocidad de un diseño puede funcionar como alta velocidad / potencia, mientras que las porciones reducidas trabajan como velocidad / potencia baja. También disponen de una opción que reduce la velocidad de subida de los búferes de salida, reduciendo al mínimo el ruido transitorio cuando la velocidad no es crítica. Los controladores de salida de toda la familia MAX 7000 (excepto dispositivos de 44 pines) se pueden establecer entre 3.3 [V] o 5[V] lo que permite el uso de estos dispositivos en sistemas mixtos de tensión.

La familia MAX 7000 es compatible con los sistemas de Altera, que son paquetes integrados que ofrecen esquemas, textos, incluyendo VHDL, Verilog y el lenguaje de descripción de hardware de Altera (AHDL).

Funciones Principales.

- Bloques lógicos
- Macroceldas
- Expansión o distribución de términos de producto (compatible y paralelos)
- Matriz de interconexión programable
- Entrada y Salida de bloques de control

Un bloque lógico es similar a un PLD, cada bloque está compuesto por un arreglo de compuertas AND y OR en forma de suma de productos. El tamaño del bloque está definido por la capacidad del CPLD y así mismo depende del tamaño de la función booleana que pueda ser implementada dentro del bloque. Los bloques lógicos generalmente cuentan con 4 a 20 macroceldas.

Las macroceldas tienen registros, control de polaridad y buffers para salidas en alta impedancia. Por lo general un CPLD tiene macroceldas de *entrada/salida, macroceldas de entrada* y *macroceldas internas* u *ocultas*.

Para la expansión o distribución de productos tenemos diferencias en cuanto a las matrices de productos, para ello dependemos del CPLD y del fabricante. El tamaño de las sumas es el factor más importante para la implementación de funciones booleanas. Aunque la mayoría de las funciones lógicas se pueden implementar con los cinco términos de productos disponibles en cada macrocélula, la función lógica más compleja requiere condiciones adicionales. Otra macrocélula puede ser usada para suministrar los recursos necesarios de la lógica, así mismo la arquitectura MAX 7000 también permite compartir en términos de expansión de producto (“expansores”) que proveen productos adicionales directamente a cualquier macrocélula en la misma matriz de bloques lógicos. Para ello tenemos la expansión compartida y expansión paralela, en la primera se tienen 16 expansores que pueden ser vistos como un conjunto de términos cada uno con salidas invertidas que se alimentan de nuevo en la matriz lógica y los expansores puden alimentar múltiples macrocélulas, para la expansión paralela, tenemos términos sin usar que se pueden asignar a una macrocélula vecina para aplicar la lógica de funciones complejas, y se tiene permitido hasta veinte términos de productos para alimentar directamente la macrocélula, el diagrama se muestra en la Figura 1.71.

La matriz de interconexiones programables (PIA) se encarga de enlazar los pines de entrada/salida a las entradas del bloque lógico, o las salidas del bloque lógico a las entradas de otro bloque lógico o incluso también a las entradas del mismo. Las configuraciones usadas para esta matriz son: la interconexión mediante arreglo o interconexión mediante multiplexores, de las cuales sólo usan una de ellas. En la interconexión mediante arreglo es una matriz de filas y columnas que contiene una celda programable de conexión en cada intersección y puede ser activada para conectar o desconectar la correspondiente fila y columna. Para la interconexión

mediante multiplexores se tiene un multiplexor por cada entrada al bloque lógico. Las vías de interconexión programables son conectadas a las entradas de un número de multiplexores por cada bloque lógico.

La arquitectura MAX 7000 incluye cuatro entradas (Figura 1.7.1-2) dedicadas que pueden ser utilizadas como insumos de uso general o como de alta velocidad, con control mundial de señales (reloj, clear y dos salidas disponibles).

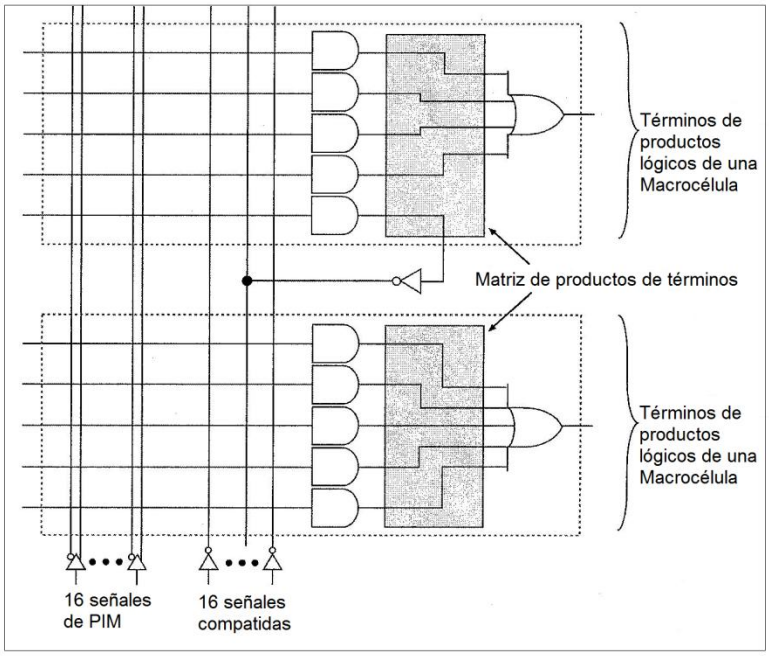


Figura 1.7.1-2 Expansores compartidos. Cortesía Altera Corporation.

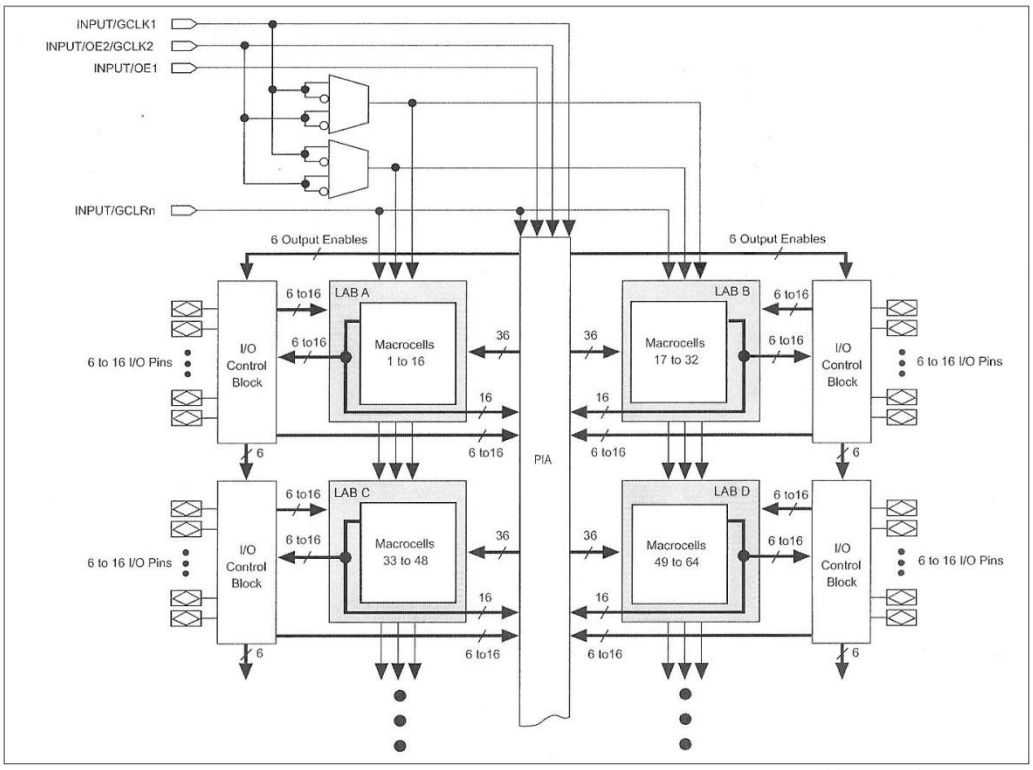


Figura 1.7.1-3 Arquitectura de la familia MAX7000. Cortesía Altera Corporation.

Tiempo de Programación.

El tiempo necesario para poner en práctica cada una de las etapas se menciona a continuación:

- Un tiempo de pulso para borrar, programar, o leer las células EEPROM.
- Un cambio de tiempo basado en la frecuencia de reloj de prueba (TCK) y el número de ciclos de TCK para cambiar las instrucciones, dirección y datos del dispositivo.

Al combinar el pulso y los tiempos de cambio para cada uno de las etapas de los programas, el tiempo puede derivar en función de la frecuencia TCK. Porque diferentes dispositivos con capacidad de ISP tienen un número diferente de células EEPROM, y la duración total de variables fijas son únicas para un solo dispositivo.

El tiempo necesario para programar el dispositivo MAX 7000 se puede calcular a través de la siguiente expresión:

$$t_{PROG} = t_{PPULSE} + \frac{C_{cycle}{}_{PTCK}}{f_{TCK}}$$

Donde:

t_{PROG} = tiempo de programación.

t_{PPULSE} =suma del tiempo fijado para borrar, programar, y verificar las células EEPROM.

$C_{cycle_{PTCK}}$=número de ciclos TCK para programar el dispositivo.

f_{TCK} =frecuencia TCK.

Los tiempos ISP para una verificación independiente de un dispositivo MAX 7000 se pueden calcular de la siguiente manera:

$$t_{VER} = t_{VPULSE} + \frac{C_{cycle_{VTCK}}}{f_{TCK}}$$

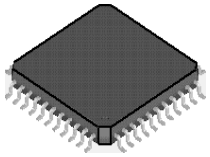
Donde:

t_{VER}= tiempo de verificación.

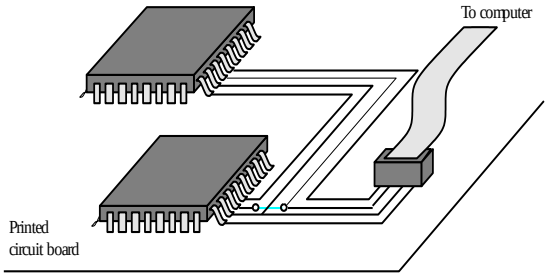
t_{VPULSE}= suma del tiempo fijado para verificar las células EEPROM.

$C_{cycle_{VTCK}}$=número de ciclos TCK para verificar el dispositivo.

Los dispositivos MAX 7000 pueden ser programados en PC basadas en Windows con el programador de tarjetas lógico de Altera, una unidad de programación Maestra (MPU), y el adaptador del dispositivo adecuado. El MPU realiza una prueba de continuidad para asegurar un contacto eléctrico adecuado entre el adaptador y el dispositivo. Ver Figura 1.7.1-3



(a) CPLD in a Quad Flat Pack (QFP) package



(b) JTAG programming

Figura 1.7.1-4 Programación y empaquetado de un CPLD.

A continuación se muestran las condiciones de operación del MAX 7000. Tabla 1.7.3

Símbolo	Parámetro	Condiciones	MIN	MAX	UNIDAD
V_{CC}	Supply voltaje	With respect to ground	-2.0	7.0	V

V_I	DC input voltaje		-2.0	7.0	V
I_{OUT}	DC output current, per pin		-25	25	Ma
T_{STG}	Storage temperature	No bias	-65	150	°C
T_{AMB}	Ambient temperature	Under bias	-65	135	°C
T_J	Junction temperature	Ceramic packages, under bias		150	°C
		PQFP and RQFP packages, under bias		135	°C

Tabla 1.7.3 Valores máximos absolutos del dispositivo MAX 7000.

Consumo de potencia.

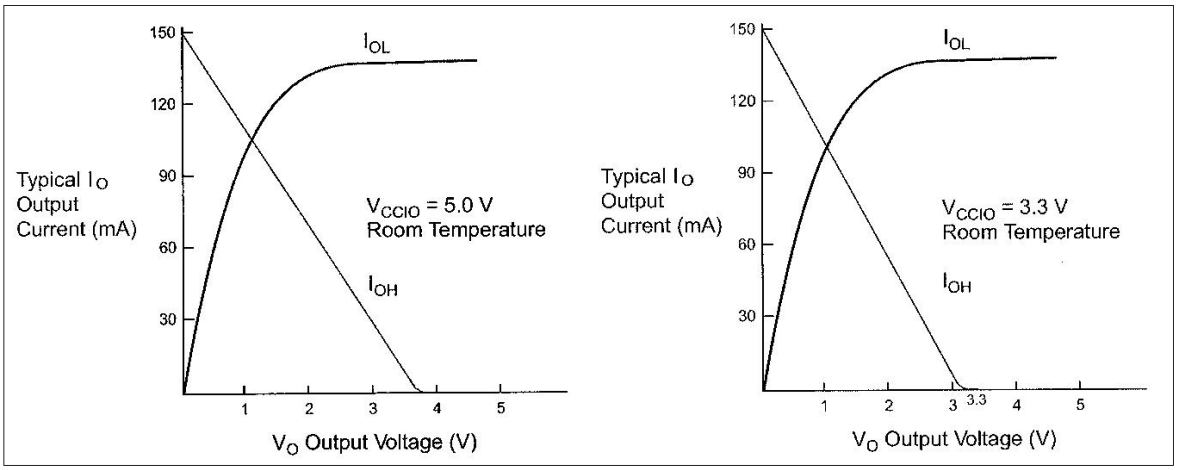
La fuente de energía (P) contra la frecuencia (F_{MAX} en MHz) para dispositivos MAX 7000 se calcula con la siguiente ecuación:

$$P = P_{INT} + P_{IO} = I_{CCINT} \times P_{IO}$$

El valor P_{IO} depende de las características de carga de salida y una frecuencia de conmutación.

El valor I_{CCINT} que depende de la frecuencia de conmutación y la lógica de aplicación.

A continuación se ilustran las características típicas de salida de los dispositivos MAX 7000. Gráfica 1



Gráfica 1. Características típicas de salida de los dispositivos MAX 7000. Cortesía Altera Corporation.

1.7.2 FPGA´S.

Un FPGA (*Field programable Gate Array*) es un dispositivo resultado de la evolución de los CPLD´s, que como ya se mencionó anteriormente, es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser programada. Ambos dispositivos contienen un gran

número de elementos lógicos programables. Haciendo una analogía de la densidad de elementos lógicos programables en compuertas lógicas equivalentes (número de compuertas NAND equivalentes que se pueden programar en un dispositivo) se puede decir que en el CPLD tenemos el orden de decenas de miles de compuertas lógicas, mientas que en un FPGA estaríamos en el orden de cientos de miles o hasta millones de compuertas lógicas programables como se muestra en la Tabla 1.8

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB=Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLB´s						
XC3S50	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400	400K	9,064	32	28	896	56K	288K	16	4	264	116
XC3S1000	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	712	312
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	784	344

Tabla 1.8 Características del Modelo Spartan-3 FPGA. Cortesía XILINX.

Los FPGA´s están diseñados específicamente para satisfacer las necesidades de alto volumen, de costo razonable para aplicaciones en el consumo de electrónica.

La familia Spartan-3 se basa en el éxito de su familia anterior Spartan-IIE al aumentar la cantidad de los recursos de lógica, la capacidad de la memoria interna RAM, el número total de E/S, y el nivel global de rendimiento, así como mejorar las funciones del reloj.

La lógica de elementos flexibles de la arquitectura FLEX incorpora todas las características necesarias para poner en práctica megafunciones en el arreglo de compuertas con un máximo de 250,000 compuertas. La familia FLEX 10K proporciona densidad, velocidad y características para integrar sistemas completos, incluyendo múltiples buses de 32 bits en un solo dispositivos.

Son dispositivos reconfigurables que permiten la inspección al 100% previa a su envío.

Cada dispositivo FLEX 10K contiene una matriz incrustada para aplicar la memoria y la lógica de funciones especializadas, y una gran lógica para aplicar lógica general.

Debido al bajo costo, los FPGA´s son ideales para una amplia gama de aplicaciones de productos electrónicos, como el acceso de banda ancha, redes domésticas, proyecciones y equipos de televisión digital. También son una alternativa superior para enmascarar ASIC´s programados.

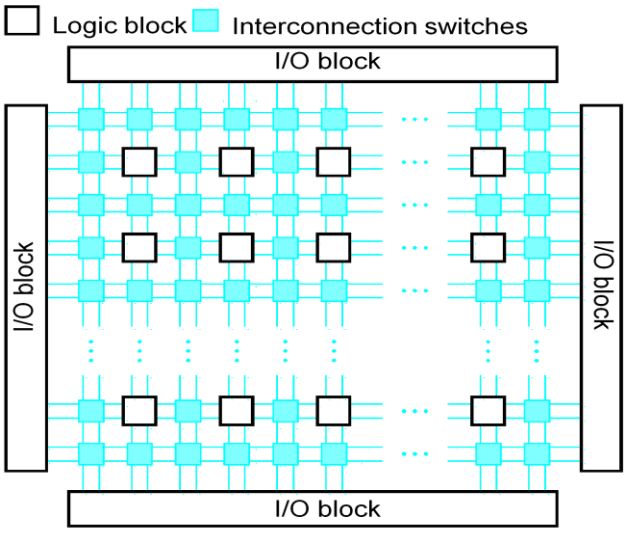


Figura 1.7.2-1 Estructura general de un FPGA.

La arquitectura de la familia Spartan-3 consta de cinco fundamentales elementos programables:

- Contienen bloques lógicos configurables (CBLs) basados en RAM para aplicar la lógica y almacenamiento de elementos que pueden ser utilizados como flip-flops o cerraduras.
- Bloques de E/S (IOB´s) para controlar el flujo de datos entre los pines I/O y la lógica interna del dispositivo. Cada IOB apoya el flujo de datos bidireccional más un tercer estado de operación.
- Bloque de memoria RAM proporciona almacenamiento de datos en forma de 18Kbit.
- Multiplicador de bloques para aceptar dos números binarios de 18 bits como insumos y poder calcular el producto.
- Un bloque administrador de reloj digital (DCM) que proporciona auto calibración, y soluciones completamente digitales para distribuir, retrasar, multiplicar, dividir y cambiar el reloj de fase en señales. Ver Figura 1.8.2

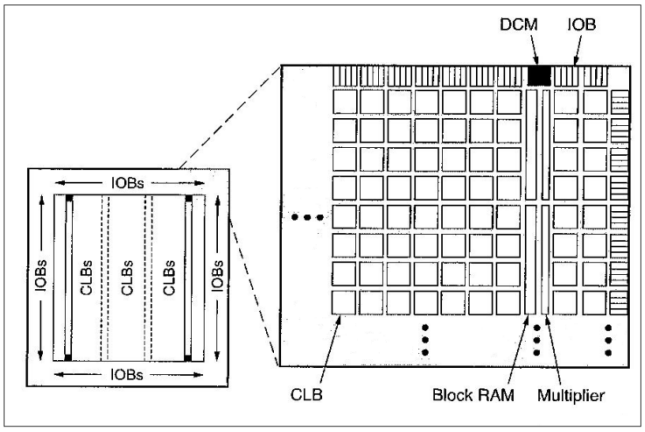


Figura 1.7.2-2 Arquitectura de la Familia Spartan-3. Cortesía Xilinx.

Antes de encender el FPGA, los datos de configuración se almacenan externamente en una PROM o algún otro medio dentro o fuera de la tarjeta.

Dentro de la familia Spartan-3 todos los dispositivos son compatibles pin a pin por encapsulado.

El EAB (Embedded Array Block) es un bloque flexible de RAM con registros sobre los puertos de entrada y salida, y se utiliza para poner en práctica infinidad de arreglo de compuertas.

El LE (Logic Element) es la unidad más pequeña de la lógica en la arquitectura de estos dispositivos, tiene un tamaño compacto que proporciona la utilización de la lógica eficiente, cada uno cuenta con cuatro entradas, que es un generador de funciones que puede calcular rápidamente cualquier función de cuatro variables. Además cada LE contiene un flip-flop programable, una cadena de transporte, además de una cadena en cascada. Ver Figura 1.8.3

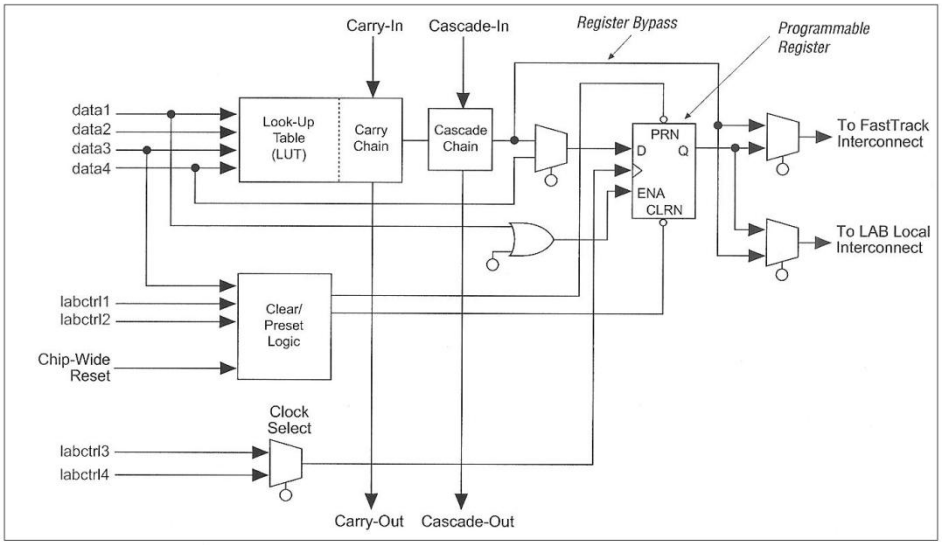


Figura 1.7.2-3 Elemento Lógico (LE). Cortesía Altera Corporation.

La cadena realiza un rápido traspaso de funciones entre ellas. Esta característica permite a la familia FLEX 10K implementar contadores de alta velocidad y comparadores de tamaño de manera eficiente. Ver Figura 1.8.4

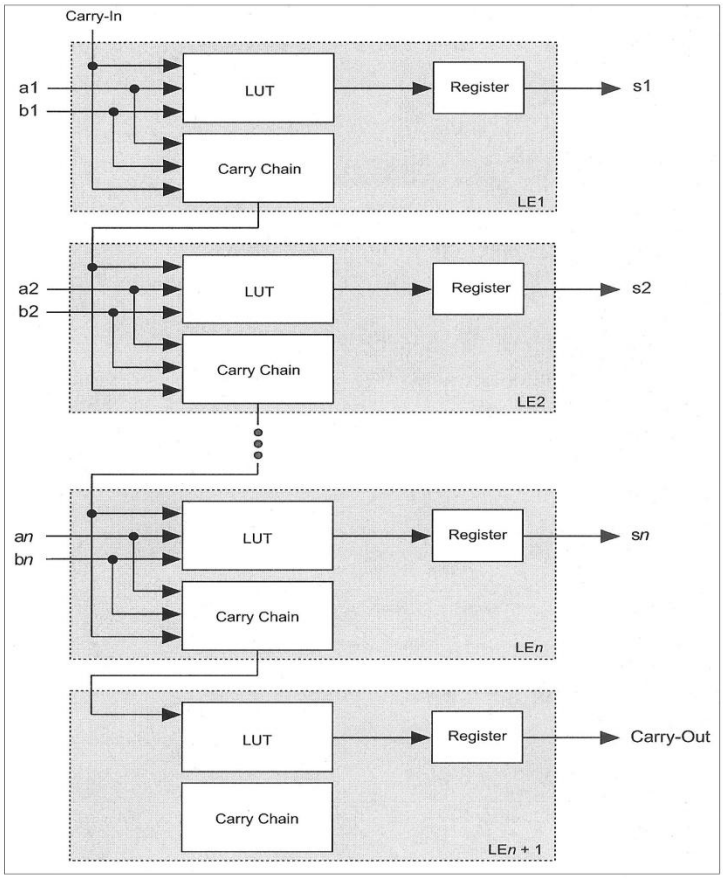


Figura 1.7.2-4 Cadena de Cascadas. Cortesía Altera Corporation.

Con la cadena en cascada se aplican funciones que tienen un amplio abanico de entrada se utilizan para calcular las porciones de la función en paralelo. Pueden utilizar un AND lógico o un OR lógico para conectar las salidas adyacentes. Figura 1.7.2-5

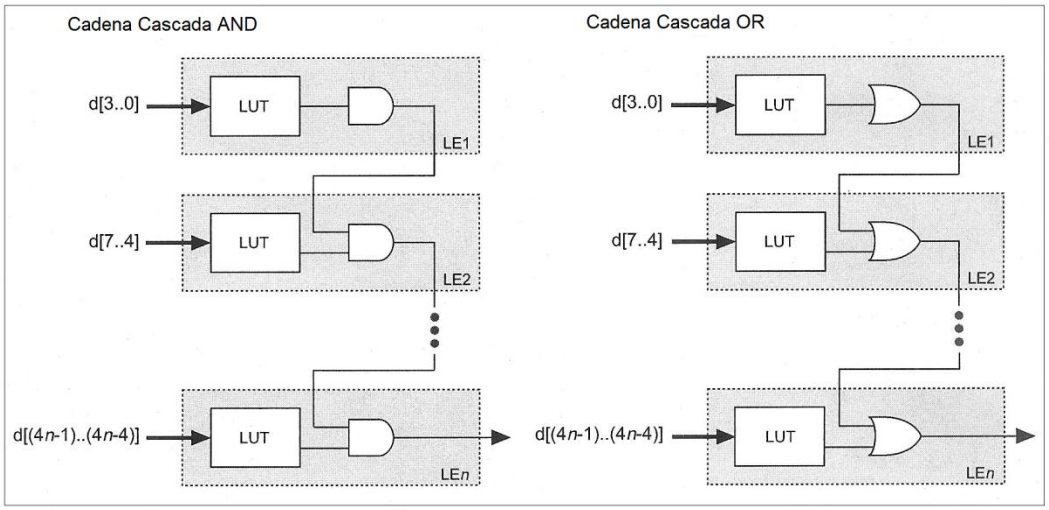


Figura 1.7.2-5 Cadena Cascada AND y OR. Cortesía Altera Corporation.

Los elementos lógicos (LE) pueden operar en los siguientes cuatro modos:

- Modo normal (Normal mode).
- Modo de aritmética (Arithmetic mode).
- Modo contador ascendente (Up-down counter mode).
- Modo contador de limpieza (Clearable counter mode).

Ver Figura 1.8.6

El modo normal es adecuado para aplicaciones de la lógica general y una amplia decodificación de las funciones que pueden tomarle ventaja a una cadena en cascada. En modo normal tenemos cuatro entradas de datos.

En modo aritmético o de cálculo, tenemos tres entradas ideales para la aplicación de complementos, acumuladores y comparadores. Primero se usa el traslado de la señal y dos entradas de datos desde el nivel local de la interconexión de LAB (Logic Array Blocks) para generar una salida.

El contador ascendente (arriba/abajo) ofrece un contador disponible, reloj, y control síncrono arriba/abajo, los cuales controlan las señales generadas por las entradas de datos de la interconexión LAB.

Para apoyar diseños de alta velocidad el fabricante Altera ofrece un reloj que contiene un bucle de enganche de fase que se utiliza para aumentar la velocidad de diseño y reducir el uso de recursos, para ello se hace uso de una sincronización que reduce el retraso del reloj del dispositivo.

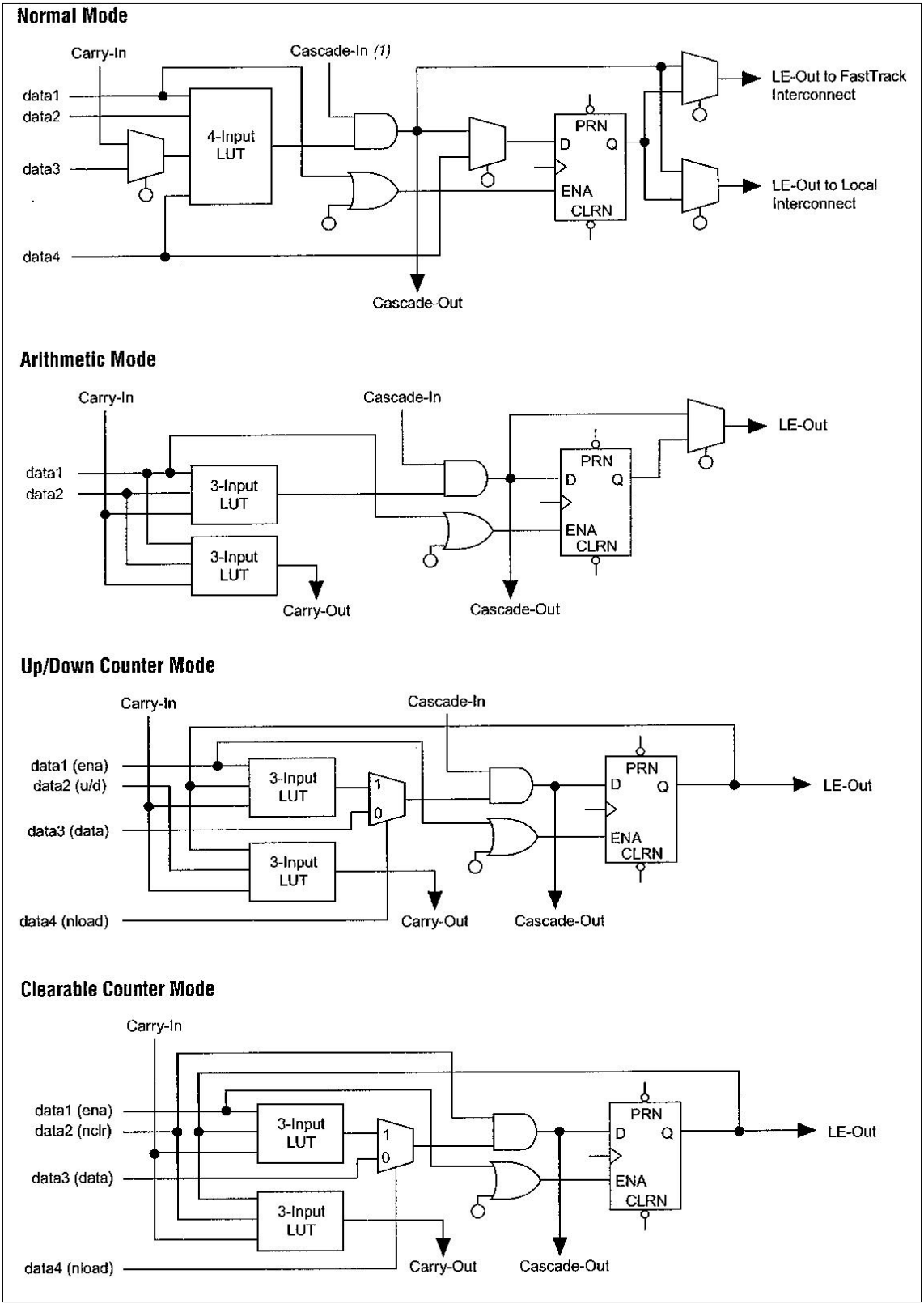


Figura 1.7.2-6 Modos de Operación de Elementos Lógicos (LE). Cortesía Altera Corporation.

A continuación se en la tabla 1.9 se muestran los valores absolutos del dispositivo FLEX 10K.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply Voltage	With respect to ground	-2.0	7.0	V
V_I	DC input Voltage		-2.0	7.0	V
I_{OUT}	DC output Voltage		-25	25	mA
T_{STG}	Storage Temperature	No bias	-65	150	°C
T_{AMB}	Ambient Temperature	Under bias	-65	135	°C
T_J	Junction Temperature	Ceramic packages, under bias		150	°C
		PQFP, TQPF, RQFP packages under bias		135	

Tabla 1.9 Valores absolutos de la Familia FLEX 10K. Cortesía Altera Corporation.

Consumo de potencia.

La fuente de alimentación P de la familia de dispositivos FLEX 10K se calcula a través de la siguiente expresión:

$$P = P_{INT} + P_{IO} = (I_{CCSTANDBY} + I_{CCACTIVE}) \times V_{CC} + P_{IO}$$

El valor de la corriente $I_{CCACTIVE}$ es calculado con la siguiente ecuación:

$$I_{CCACTIVE} = K \times f_{MAX} \times N \times tog_{LC} \times \frac{\mu A}{MHz \times LE}$$

Donde :

f_{MAX} = Frecuencia máxima de operación en MHz.

N = Número total de células lógicas usadas en el dispositivo.

tog_{LC} = Promedio en porciento de células lógicas alternadas en cada reloj.

K = Constante del dispositivo.

A continuación se ilustran las gráficas de salida de los dispositivos FLEX 10K, figura 1.8.7

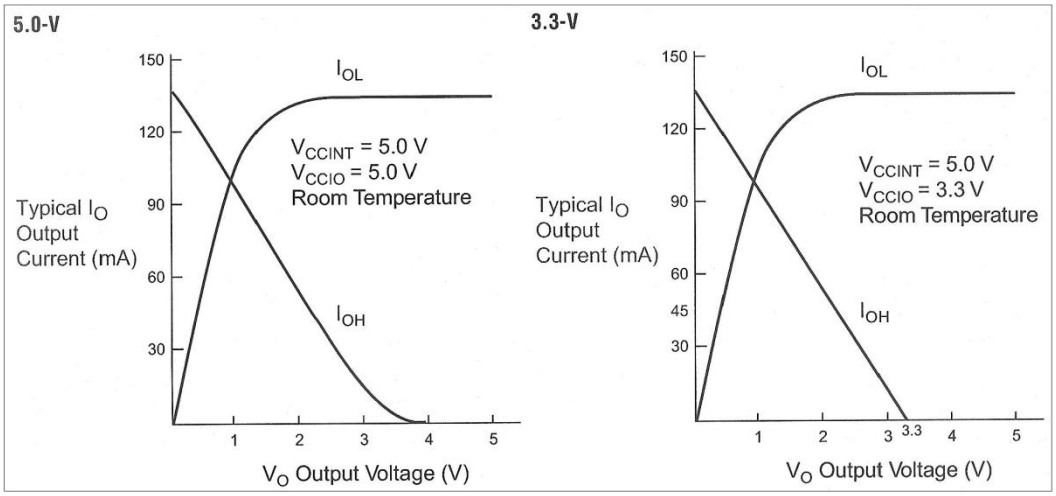


Figura 1.7.2-7 Características de salida de los dispositivos FLEX 10K. Cortesía Altera Corporation.

1.7.3 Lenguaje VHDL.

VHDL proviene de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. El cual es un lenguaje de descripción y modelado diseñado para describir (en una forma que los humanos y las máquinas puedan leer y entender) la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes.
El lenguaje VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automática de circuitos. El VHDL fue desarrollado de forma muy parecida al ADA debido a que el ADA fue también propuesto como un lenguaje puro pero que tuviera estructuras y elementos sintácticos que permitieran la programación de cualquier sistema hardware sin limitación de la arquitectura. El ADA tenía una orientación hacia sistemas en tiempo real y al hardware en general, por lo que se lo escogió como modelo para desarrollar el VHDL.
VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso en distintos estilos del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.
Uno de los objetivos del lenguaje VHDL es el modelado. Modelado es el 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. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía.
Aunque puede ser usado de forma general para describir cualquier circuito se usa principalmente para programar PLD (*Programable Logic Device* - Dispositivo Lógico Programable), FPGA (*Field Programmable Gate Array*), ASIC y similares.
Algunas ventajas del uso de VHDL para la descripción hardware son:

1. VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de compuertas.
2. Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de compuertas.

3. Al estar basado en un estándar (IEEE Std 1076-1987) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
4. VHDL permite diseño Top-Down, esto es, permite describir (modelar) el comportamiento de los bloques de alto nivel, analizándolos (simulación), y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.
5. Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

VHDL describe Estructura y Comportamiento.

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión, de esta manera tenemos especificado un circuito y sabemos cómo funciona; esta es la forma habitual en que se han venido describiendo circuitos y las herramientas utilizadas para ello han sido los esquemas y las descripciones netlist.
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 lo que interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente puede plantear algunos problemas a la hora de realizar un circuito a partir de la descripción de su comportamiento.
El VHDL es interesante puesto que va a permitir los dos tipos de descripciones:

Estructura: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.
Comportamiento: VHDL también se puede utilizar para la descripción funcional de un circuito. Esto es lo que lo distingue de un lenguaje de Netlist. 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 está 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.

Estructura de programa.

VHDL fue diseñado en base a los principios de la programación estructurada. La idea es definir la interfaz de un modulo de hardware mientras deja invisible sus detalles internos. La entidad (ENTITY) en VHDL es simplemente la declaración de las entradas y salidas de un modulo mientras que la arquitectura (ARCHITECTURE) es la descripción detallada de la estructura interna del modulo o de su comportamiento. A continuación se describe el concepto anterior. Muchos

diseñadores conciben la Entity como una funda de la arquitectura dejando invisible los detalles de lo que hay dentro (architecture). Esto forma la base de un sistema de diseño jerárquico, la arquitectura de la entidad de más nivel (top level) puede usar otras entidades dejando invisible los detalles de la arquitectura de la identidad de menos nivel. En la descripción, las entidades B, E y F no utilizan a otras entidades. Mientras que la entidad A utiliza a todas las demás. A la pareja entidad-arquitectura se le llama modelo. En un fichero texto VHDL la entidad y la arquitectura se escriben separadas, por ejemplo a continuación se muestra un programa muy simple en VHDL de una compuerta de 2 entradas. Como en otros programas VHDL ignora los espacios y saltos de líneas. Los comentarios se escribieron con 2 guiones (--) y termina al final de la línea. En la figura siguiente se muestra la estructura de un modelo en VHDL. SINTASIS PARA LA DECLARACION DE LA ENTIDAD VHDL define muchos caracteres especiales llamados “palabras reservadas”. Aunque las palabras reservadas no son sensibles a las mayúsculas o minúsculas, el ejemplo que sigue las utilizaremos en mayúsculas para identificarlas.

```
ENTITY Nombre_entidad IS
PORT ( Nombre de señal: modo tipo de señal;
                . . .
       Nombre de señal:  modo    tipo de señal ) ;
END nombre_entidad ;
```

Ejemplo básico de descripción VHDL.

Ejemplo 1.7 Describir en VHDL un circuito que multiplexe dos líneas (a y b) de un bit, a una sola línea (salida) también de un bit; la señal `selec` sirve para indicar que a la salida se tiene la línea a (selec='0') o b (selec='1').
En la figura 1.7.3 se muestra el circuito implementado con puertas lógicas que realiza la función de multiplexación.
Lo que se va a realizar a continuación es la descripción comportamental del circuito de la figura 1.7.3 y luego se realizará la descripción estructural para ver las diferencias. Más adelante se verá que hay dos tipos de descripción comportamental, pero de momento, el presente ejemplo únicamente pretende introducir el lenguaje VHDL y su estructura.

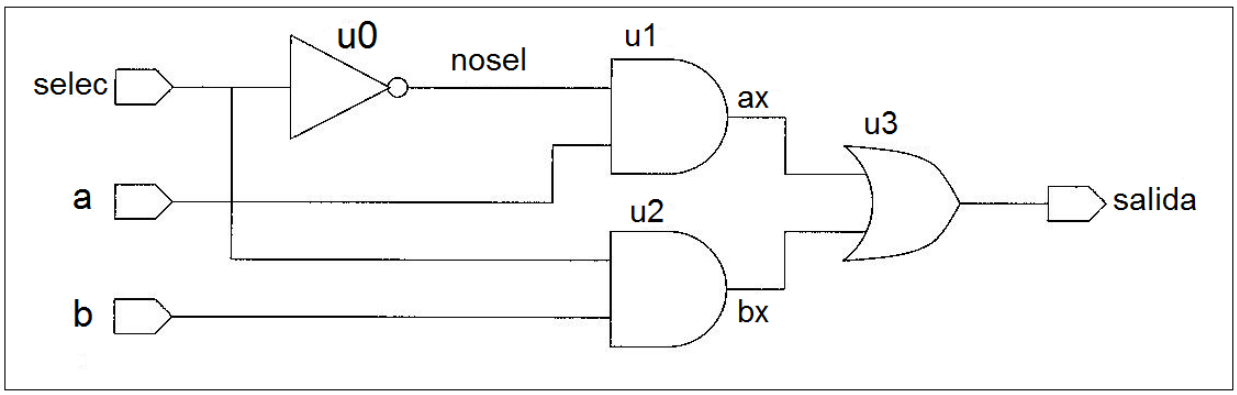


Figura 1.7.3 Esquema del ejemplo básico en VHDL.

La sintaxis del VHDL no es sensible a mayúsculas o minúsculas por lo que se puede escribir como se prefiera. A lo largo de las explicaciones se intentará poner siempre las palabras claves del lenguaje en mayúsculas para distinguirlas de las variables y otros elementos.
En primer lugar, sea el tipo de descripción que sea, hay que definir el símbolo o entidad del circuito. Lo primero es definir las entradas y salidas del circuito, es decir, la caja negra que lo define. Se le llama entidad porque en la sintaxis de VHDL esta parte se declara con la palabra clave ENTITY. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone a continuación:

```
-- Los comentarios empiezan por dos guiones
ENTITY mux IS
PORT ( a: IN bit;
b: IN bit;
selec: IN bit;
salida: OUT bit);
END mux;
```

Esta porción del lenguaje indica que la entidad mux (que es el nombre que se le ha dado al circuito) tiene tres entradas de tipo bit, y una salida también del tipo bit. El tipo bit simplemente indica una línea que puede tomar los valores '0' y '1'.

La entidad de un circuito es única, sin embargo, se mostró que un mismo símbolo, en este caso entidad, podía tener varias vistas o en el caso de arquitecturas VHDL. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra de comportamiento o funcional, ambas son descripciones diferentes, pero ambas descripciones corresponden al mismo circuito, símbolo, o entidad. Veamos entonces la descripción de comportamiento:

```
ARCHITECTURE comportamental OF mux IS
BEGIN
PROCESS(a,b,selec)
BEGIN
IF (selec='0') THEN
salida<=a;
ELSE
salida<=b;
END IF;
END PROCESS;
END comportamental;
```

Elementos sintácticos del VHDL.

El lenguaje VHDL es verdaderamente un lenguaje, por lo que tiene sus elementos sintácticos, sus tipos de datos, y sus estructuras como cualquier otro tipo de lenguaje.
El hecho de que sirva para la descripción hardware lo hace un poco diferente de un lenguaje convencional. Una de estas diferencias es probablemente la posibilidad de ejecutar instrucciones a la vez de forma concurrente.
Algunos de estos elementos sintácticos se muestran a continuación:
Comentarios: Cualquier línea que empieza por dos guiones "`--`" es un comentario.
Identificadores: Son cualquier cosa que sirve para identificar variables, señales, nombres de rutina, etc. Puede ser cualquier nombre compuesto por letras incluyendo el símbolo de subrayado "`_`". Las mayúsculas y minúsculas son consideradas iguales, así que `JOSE` y `jose` representan el mismo elemento. No puede haber ningún identificador que coincida con alguna de las palabras clave del VHDL.
Números: Cualquier número se considera que se encuentra en base 10. Se admite la notación científica convencional para números en coma flotante. Es posible poner números en otras bases utilizando el símbolo del sostenido "`#`". Ejemplo: `2#11000100# y 16#C4#` representan el entero `196`.
Caracteres: Es cualquier letra o carácter entre comillas simples: `'1','3','t'`.
Cadenas: Son un conjunto de caracteres englobados por comillas dobles: "`Esto es una cadena`".
Cadenas de bits: Los tipos `bit` y `bit_vector` son en realidad de tipo carácter y matriz de caracteres respectivamente. En VHDL se tiene una forma elegante de definir números con estos tipos y es mediante la cadena de bits. Dependiendo de la base en que se especifique el número se puede poner un prefijo B (binario), O (octal), o X (hexadecimal). Ejemplo: `B"11101001", O"126", X"FE".`

Operadores y expresiones.

Las expresiones en VHDL son prácticamente iguales a como pudieran ser en otros lenguajes de programación o descripción, por lo que se expondrán brevemente los existentes en VHDL y su utilización.

Operadores varios.

& (Concatenación) Concatena matrices de manera que la dimensión de la matriz resultante es la suma de las dimensiones de las matrices sobre las que opera:
punto<=x&y mete en la matriz punto la matriz x en las primeras posiciones, y la matriz y en las _ultimas.
Operadores aritméticos.

****** (exponencial) Sirve para elevar un número a una potencia: 4**2 es 42. El operador de la izquierda puede ser entero o real, pero el de la derecha sólo puede ser entero.
ABS() (valor absoluto) Como su propio nombre indica esta función devuelve el valor absoluto de su argumento que puede ser de cualquier tipo numérico.
***** (Multiplicación) Sirve para multiplicar dos números de cualquier tipo (los tipos `bit` o `bit_vector` no son numéricos).
/ (División) También funciona con cualquier dato de tipo numérico.
MOD (módulo) Calcula en módulo de dos números. Exactamente se define el módulo como la operación que cumple: a=b*N+(a MOD b) donde N es un entero. Los operandos sólo pueden ser enteros. El resultado toma el signo de b.
REM (resto) Calcula el resto de la división entera y se define como el operador que cumple: a=(a/b)*b+(a REM b), siendo la división entera. Los operandos sólo pueden ser enteros. El resultado toma el signo de a.
+ (suma y signo positivo) Este operador sirve para indicar suma, si va entre dos operandos, o signo, si va al principio de una expresión. La precedencia es diferente en cada caso. Opera sobre valores numéricos de cualquier tipo.
- (resta y signo negativo) Cuando va entre dos operandos se realiza la operación de sustracción, y si va delante de una expresión le cambia el signo. Los operandos pueden ser numéricos de cualquier tipo.

Operadores de desplazamiento.

SLL, SRL (desplazamiento lógico a izquierda y a derecha). Desplaza un vector un número de bits a izquierda (`SLL`) o derecha (`SRL`) rellenando con ceros los huecos libres. Se utiliza en notación infija de manera que la señal a la izquierda del operador es el vector que se quiere desplazar y el de la derecha es un valor que indica el número de bits a desplazar. Por ejemplo `dato SLL 2` desplaza a izquierda el vector dato, es decir, lo multiplica por 4.
SLA, SRA (desplazamiento aritmético a izquierda y derecha).
ROL, ROR (rotación a izquierda y a derecha). Es como el de desplazamiento pero los huecos son ocupados por los bits que van quedando fuera.

Operadores Relacionales.

Devuelven siempre un valor de tipo booleano (`TRUE` o `FALSE`). Los tipos con los que pueden operar dependen de la operación:
`=, /=` (igualdad). El primero devuelve `TRUE` si los operandos son iguales y `FALSE` en caso contrario. El segundo indica desigualdad, así que funciona justo al revés.
Los operandos pueden ser de cualquier tipo con la condición de que sean ambos del mismo tipo.
`<,<=,>,>=` (menor mayor). Tienen el significado habitual. La diferencia con los anteriores es que los tipos de datos que pueden manejar son siempre de tipo escalar o matrices de una sola dimensión de tipos discretos.

Operadores lógicos.
Son `NOT`, `AND`, `NAND`, `OR`, `NOR` y `XOR`. El funcionamiento es el habitual para este tipo de operadores. Actúan sobre los tipos `bit`, `bit vector` y `boolean`. En el caso de realizarse estas operaciones sobre un vector, la operación se realiza bit a bit, incluyendo la operación `NOT`.

Enteros: Son datos cuyo contenido es un valor numérico entero. La forma es que se definen estos datos es mediante la palabra clave RANGE, es decir, no se dice que un dato es de tipo entero, sino que se dice que un dato está comprendido en cierto intervalo especificando los límites del intervalo con valores enteros.
Ejemplos:

```
TYPE byte IS RANGE 0 TO 255;
TYPE index IS RANGE 7 DOWNTO 1;
TYPE integer IS -2147483647 TO 2147483647; -- Predefinido en
el lenguaje
```

Este último tipo viene ya predefinido en el lenguaje aunque no es muy conveniente su utilización, especialmente pensando en la posterior síntesis del circuito.
Físicos: Como su propio nombre indica se trata de datos que se corresponden con magnitudes físicas, es decir, tienen un valor y unas unidades.
Ejemplo:

```
TYPE longitud IS RANGE 0 TO 1.0e9
  UNITS
     um;
     mm=1000 um;
     m=1000 mm;
     in=25.4 mm;
  END UNITS;
```

Hay un tipo físico predefinido en VHDL que es `time`. Este tipo se utiliza para indicar retrasos y tiene todos los submúltiplos, desde `fs` (femtosegundos), hasta `hr` (horas). Cualquier dato físico se escribe siempre con su valor seguido de la unidad: 10 mm, 1 in, 23 ns.

Reales: Conocidos también como coma flotante, son los tipos que definen un número real. Al igual que los enteros se definen mediante la palabra clave RANGE, con la diferencia de que los límites son números reales.
Ejemplos:

```
TYPE nivel IS RANGE 0.0 TO 5.0;
TYPE real IS RANGE -1e38 TO 1e38; -- Predefinido en el
lenguaje
```

Enumerados: Son datos que pueden tomar cualquier valor especificado en un conjunto finito o lista. Este conjunto se indica mediante una lista encerrada entre paréntesis de elementos separados por comas.
Ejemplos:

```
TYPE nivel_logico IS (nose,alto,bajo,Z);
TYPE bit IS ('0','1'); -- Predefinido en el lenguaje
```

Hay varios tipos enumerados que se encuentran predefinidos en VHDL. Estos tipos son: `severity_level, boolean, bit` y `character.`

Subtipos de datos.

VHDL permite la definición de subtipos que son restricciones o subconjuntos de tipos existentes. Hay dos tipos. El primero son subtipos obtenidos a partir de la restricción de un tipo escalar a un rango. Ejemplos:

```
SUBTYPE raro IS integer RANGE 4 TO 7;
SUBTYPE digitos IS character RANGE '0' TO '9';
SUBTYPE natural IS integer RANGE 0 TO entero_mas_alto; --
Predefinido en VHDL
SUBTYPE positive IS integer RANGE 1 TO entero_mas_alto; --
Predefinido en VHDL
```

El segundo tipo de subtipos son aquellos que restringen el rango de una matriz:

```
SUBTYPE id IS string(1 TO 20);
SUBTYPE word IS bit_vector(31 DOWNTO 0);
```

La ventaja de utilizar un subtipo es que las mismas operaciones que servían para el tipo sirven igual de bien para el subtipo. Esto tiene especial importancia por ejemplo cuando se describe un circuito para ser sintetizado, ya que si utilizamos `integer` sin más, esto se interpretará como un bus de 32 líneas (puede cambiar dependiendo de la plataforma) y lo más probable es que en realidad necesitemos muchas menos. Otro caso se da cuando tenemos una lista de cosas y les queremos asignar un entero a cada una, dependiendo de las operaciones que queramos hacer puede resultar más conveniente definirse un subtipo a partir de `integer` que crear un tipo enumerado.

Atributos.

Los elementos en VHDL, como señales, variables, etc, pueden tener información adicional llamada atributos. Estos atributos están asociados a estos elementos del lenguaje y se manejan en VHDL mediante la comilla simple “ ' “. Por ejemplo, `t'LEFT` indica el atributo `'LEFT` de `t` que debe ser un tipo escalar (este atributo indica el límite izquierdo del rango).

Hay algunos de estos atributos que están predefinidos en el lenguaje y a continuación se muestran los más interesantes. Suponiendo que `t` es un tipo escalar tenemos los siguientes atributos:

`t'LEFT` Límite izquierdo del tipo `t`.

`t'RIGHT` Límite derecho del tipo `t` .

`t'LOW` Límite inferior del tipo `t`.

`t'HIGH` Límite superior del tipo `t`.

Para tipos `t`, `x` miembro de este tipo, y `N` un entero, se pueden utilizar los siguientes atributos:

`t'POS(x)` Posición de x dentro del tipo `t`.

`t'VAL(N)` Elemento `N` del tipo `t`.

`t'LEFTOF(x)` Elemento que está a la izquierda de `x` en `t`.
`t'RIGHTOF(x)` Elemento que está a la derecha de `x` en `t`.
`t'PRED(x)` Elemento que está delante de `x` en `t`.
`t'SUCC(x)` Elemento que está detrás de `x` en `t`.

Para a siendo un tipo u elemento de tipo matriz, y `N` un entero de 1 a al número de dimensiones de la matriz, se pueden usar los siguientes atributos:
`a'LEFT(N)` Límite izquierdo del rango de dimensión `N` de `a`.
`a'RIGHT(N)` Límite derecho del rango de dimensión `N` de `a`.
`a'LOW(N)` Límite inferior del rango de dimensión `N` de `a`.
`a'HIGH(N)` Límite superior del rango de dimensión `N` de `a`.
`a'RANGE(N)` Rango del índice de dimensión `N` de `a`.
`a'LENGTH(N)` Longitud del índice de dimensión `N` de `a`.
Suponiendo que `s` es una señal, se pueden utilizar los siguientes atributos (se han escogido los más interesantes):
s'EVENT Indica si se ha producido un cambio en la señal.
s'STABLE(t) Indica si la señal estuvo estable durante el último periodo `t`.
El atributo `'EVENT` es especialmente útil en la definición de circuitos secuenciales para detectar el flanco de subida o bajada de la señal de reloj. Es por esto que es probablemente el atributo más utilizado en VHDL.

Declaración de constantes, variables y señales.

Un elemento en VHDL contiene un valor de un tipo especificado. Para ello existen tres tipos de elementos en VHDL, están las variables, las señales y las constantes. Las variables y constantes son similares a las que encontramos en cualquier lenguaje. A diferencia de las señales, que son elementos cuyo significado es muy diferente y es consecuencia directa de que aunque VHDL es un lenguaje muy parecido a los convencionales, no deja en ningún momento de ser un lenguaje de descripción hardware por lo que se encuentran algunas diferencias.

Constantes.

Una constante es un elemento que se inicializa a un determinado valor y no puede ser cambiado una vez inicializado, conservando para siempre su valor. Ejemplos:

```
CONSTANT e: real :=2.71828;
CONSTANT retraso: time :=10ns;
CONSTANT max_size: natural;
```

En la última sentencia la constante `max_size` no tiene ningún valor asociado. Esto solamente se permite cuando el valor sea declarado en algún otro sitio.

Variables.

Una variable es similar a una constante con la diferencia de que su valor puede ser alterado en cualquier instante. A las variables también se les puede asignar un valor inicial. Ejemplo:

```
VARIABLE contador: natural :=0;
VARIABLE aux: bit_vector(31 DOWNTO 0);
```

Señales.

Las señales son declaradas de la misma forma que las constantes y variables sólo que las señales pueden ser de varios tipos, que son normal, register y bus. Por defecto son de tipo normal. Al igual que en variables y constantes, a las señales se les puede dar un valor inicial si es requerido. Ejemplos:

```
SIGNAL selec: bit := "0";
SIGNAL datos: bit_vector(7 DOWNTO 0)BUS :=B"00000000";
```

Constantes señales y variables se procesan diferente. Las variables, por ejemplo, sólo tienen sentido dentro de un proceso (PROCES) o un subprograma, esto es, que solo tienen un sentido en entornos de programación donde las sentencias son ejecutadas en serie, por lo tanto las variables sólo se declaran en los procesos o subprogramas. Las señales pueden ser declaradas únicamente en las arquitecturas, paquetes (PACKAGE), o en lo bloques concurrentes (BLOCK). Las constantes pueden ser generalmente declaradas en los mismos sitios que las variables y señales.