Apuntes de *Estructura de los computadores*

Eduardo Espuch

Resumen

Este documento agrupara el contenido de la asignatura de EC, mostrando al final de cada apartado un ejercicio practico resuelto ademas de mostrar al final de la parte del temario de EC un ejemplo de como realizar el examen, Viene incluido al final de todo el temario de FC en caso de querer repasar algun contenido, aunque se iran haciendo breves recordatorios durante las explicaciones (tened en cuenta que la aprte de FC puede no estar del todo organiza debido a la introduccion de nuevo contenido).

Índice

1. Estructura de los computadores

1.1. Introducción

Vamos a introducirnos brevemente en definiciones y elementos relacionados con la computacion, para entender como se relacion entre si o estar preparado para siguientes explicaciones.

1.1.1. Conceptos basicos

Un computador(comunmente llamado ordenador) recibe muchas definiciones pero podemos agruparlo todo a que es una maquina capaz de recibir instrucciones a traves de algun medio, entenderlas y finalmente ejecutarlas.

Podriamos decir que **es una maquina digital** (la infomracion se almacena en forma de digitos), **electronica** (construida usando componentes electronicos, circuitos integrados), **programable**(ejecuta instrucciones de una sucesion de ordenes preestablecidas) **para el procesamiento de informacion**.

La informacion hay que entender que son hechos y representaciones que pueden o no estar relacionadas (material crudo) y al procesar dicha informacion con algun objetivo en particular obtenemos datos, que es informacion codificada para ser introducida en el computador.

Distinguimos tres etapas en el procesamiento de la informacion, la entrada (se recogen datos via teclado, codigo de barras, QR,etc), el proceso (se tratan los datos, ya sea ordenandolos, seleccionandolos, calculando con ellos,etc)y salida(obtencion de la informacion resultante, pantalla, impresora,etc).

Asumiremos que los dispositivos de entrada, procesamiento y salida de un computador, ademas de las partes de un computador se conocen, veamos ahora los tipos de computadores segun su potencia de calculo:

1. Supercomputadores: diseñados para aplicaciones cientificas, procesos complejos que requieren gran cantidad de sofisticados algoritmos y de calculos matematicos, son los sistemas mas grandes, costosos y rapidos.
2. Macrocomputadores o mainframes: computadores de grandes dimensiones diseñadas para manejar grandes cantidades de entrada, salida y almacenamiento, son capaces de usar entradas simultaneas, siendo usado por numerosos usuarios a la vez.
3. Minicomputador o servidores de terminales tontos(DAFUQ?): Mas pequeño y economico que un mainframe pero mas potente que un computador personal.
4. Workstation: diseñadas para apoyar a una red de computadoras, permitiendo a los usuarios el compartir archivos, programas y hardware. Basicamente un minicomputador de bajo coste o un microcomputador de altas prestaciones destinado a trabajo tecnico o cientifico.
5. Microcomputadores: pequeños sistemas de proposito general que pueden funcionar en red o de forma independiente. Su tamaño es gracias a los microprocesadores. Los sobremesa, los protatiles, las PDAs,etc son microcomputadores.

Hay que saber, aunque es obvio creo yo, que la tecnologia evoluciona con rapidez, con lo cual lo que un dia es tecnologia puntera, al siguiente ya esta obsoleto.

1.1.2. Funcionamiento de un computador

Veamos el funcionamiento de un computador desde dos perspectivas, la funcional y la estructural.

Si estudiamos el funcionamiento de un computador con una vision funcional, estamos estudiando la operacion individual de los componentes como parte de su estructura. Las funciones que obtenemos las podemos generalizar en:

1. Procesamiento de datos: tratara la informacion recibida
2. Almacenamiento de datos: pone a disposicion la informacion almacenada en el computador
3. Transferencias de datos entre el coomputador y el exterior: se podria entender como el controlador de entrada y salida.
4. Control de las anteriores operaciones: el resto de las funciones estan conectadas al mecanismo de control, el cual indica cuando realizar una operacion determinada.

Tal generalizacion se debe a que la especializacion funcional de un computador ocurre cuando se programa, no cuando se diseña

Por ejemplo, introducir por teclado y que inmediatamente se muestro por teclado realizaria un recorrido que se inicia en el sistema de transferencia de datos (datos del teclado), comprobaria en el mecanismo de control que realizar con dicha informacion y llevaria los datos de nuevo al sistema de transferencias de datos(datos en pantalla). TODA OPERACION PASA POR MECANISMO DE CONTROL

Si estudiasemos el funcionamiento de un computador con una vision estructural, estamos estudiando la forma de como se relacionan los componentes unos con otros. Para ello, tengamos en cuentas los principales componentes estructurales:

1. Procesador,CPU: controla el funcionamiento del computador y procesa los datos (mecanismo de control y recurso de preocesamiento de datos)
2. Subsistema de memoria: almacena datos
3. Subsistema de entrada/salida: transfiere datos entre el computador y el exterior
4. ruta de datos: interconexion entre las diferentes partes

El procesador es un componente muy complejo, compuesto por diversos elemento, veamoslo:

1. ALU, unidad aritmeticologica: lo veremos mas adelante
2. UC,Unidad de Control: lo veremos mas adelante pero conocer por ahora que esta compuesto por logica secuencial, registros y decodificadores y memoria de control.
3. Registros: Unidad completa de información que está asociada a un proceso de entrada o de salida
4. Bus interno: interconexion entre los distintos elementos del procesador

Al describir un computador, diferenciamos entre su arquitectura (atributos del sistemas visibles al programador, es decir, que tienen impacto directo en la ejecucion logica de un programa) y su organizacion o estructura(unidades funcionales que componen al computador y sus interconexiones, las cuales implementan una determinada arquitectura).

Para que nos entendamos, la arquitectura decidira si un computador sera capaz de realizar una funcion y la estructura o organizacion decidira como esa funcion es capaz de realizarse.

La arquitectura del juego de instrucciones ISA describe la estructura del computador desde el punto de vista del programador. Diremos que una familia de computadores tienen la misma arquitectura si tienen el mismo repertorio de instrucciones, es decir, ejecutan el mismo codigo binario.

- Diseño del conjunto de instrucciones
- Interfaz del computador
- Relacion con los compiladores

La organizacion de la maquina se refiere al esquema(layout) e interconexiones de varias unidades funcionales.

- Unidades funcionales y su interconexion
- Transparente al software
- Componentes Hardware

1.1.3. Estructura Arquitectura Von Neumann

Vamos a estudiar la estructura que a dia de hoy es la mas usada, la de la maquina de Von Neumann. Se caracteriza por:

- Realiza un determinado conjunto de operaciones basicas(instrucciones) sobre unos datos
- Los datos y las intrucciones se almacenan en una sola memoria de lectura-escritura
- Los contenidos de la memoria se direccionan indicando su posicion sin considerar el tipo de dato contenido en la misma
- La ejecucion se produce siguiendo una secuencia de instruccion tras intruccion llamada programa

Desde el punto de vista estructural, es exactamente al que hemos visto anteriormente. Para ser mas exactos, la maquina de Von Neumann consta de:

- Memoria principal, almacena tanto datos como instrucciones
- Una unidad aritmeticologico(ALU) capaz de realizar operaciones con datos binarios (elemento de la CPU)
- Unidad de control(UC) que interpreta las instrucciones en memoria y provoca su ejecucion(elemento de la CPU)
- Elementos de entrada-salida(E/S) dirigidos por la UC
- Una ruta de datos, instrucciones, direcciones, informacion de control que permite intercomunicar todos los elementos.

Veamos el funcionamiento de cada parte con mas detalle.

1.1.4. Funcionamiento de la maquina Von Neumann

La memoria principal Compuesta por un conjunto de celdas identicas, cada una asociada a una direccion unica con la que poder seleccionar una determinada celda. Al seleccionar dicha celda, podemos decidir si leer su contenido o escribir/almacenar nuevo contenido. Dicho contenido puede ser tanto datos como instrucciones que forman parte de los programas.

Como no tengo tiempo para hacer un dibujo os comento, la apariencia basica de una unidad de memoria 2^nxk (en bits, se puede hacer la conversion a bytes facilmente) consta en una entrada del bus de direcciones con n bits, en una entrada para determinar si es lectura o escritura (suele ser una entrada E/S y otra entrada desde la UC que activa la unidad) y una entrada/salida desde el bus de datos de k bits. Recordad que Kilo, Mega, Giga, etc al hablar en valores en base binaria son $2^10, 2^20, ...$

Unidad aritmeticologica, ALU Elemento de la CPU que permite realizar unsa serie de operaciones aritemeticas, logicas y desplazamientos, ademas posee registros de almacenamiento temporal de la informacion.

Consta de dos entradas de datos, otra entrada para indicar la operacion a realizar(indicada por la UC) y dos salidas, una el resultado y otra para los registros de estados(por lo general no se usa, pero ahi esta)

Unidad de control, UC Encargada de leer las instrucciones almacenadas en memoria, generar las señales de control necesarias para que el compitador funcione y ejecute las intrucciones leidas de forma correcta y de evaluar el registro de estado.

Consta de dos entradas, una indicando la instruccion y otra el registro de estado, y de 1 salida con numerosos campos(o numerosas salidas, segun como se interprete). Las salidas de la UC representan las señales de control, las cuales estan conectadas a los respectivos componentes del computador.

El registro de estado sera necesario en el caso de una instruccion deba de usar un mismo elemento varias veces y en un unico ciclo no se pueda,

Unidad de Entrada/Salida, E/S Se comunica con unidades exteriores, o perifericos, ademas de permitir, entre otras cosas, cargar datos y programas en la memoria, sacar datos, etc...

Los perifericos constan de una entrada para el bus de direcciones, otra para lectura-escritura (ademas de la señal de control) y una entrada-salida al bus de datos.

Ruta de datos Formada por los caminos de comunicacion entre los diferentes dispositivos, denominados buses. Fisicamente son cun conjunto de conductores electricos paralelos que suelen estar incluidas dentro de la tarjeta del circuito impreso en la que esta implantado el sistema.

El bus que conecta los componentes principales anteriormente visto se denomina bus del sistema, el cual contiene el bus de datos, el bus de direcciones y el bus de control. Todos los dispostivos por lo general deberan de estar conectados a estos tres buses.

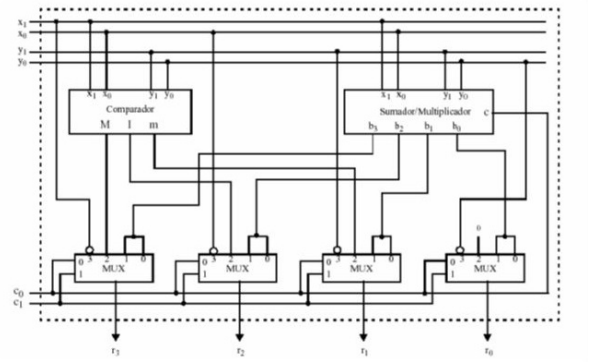
1.2. Unidad aritmeticologica, ALU

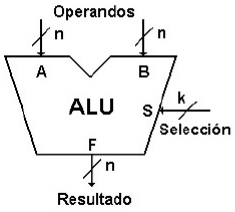
Este es un tema largo, en el cual se repasan conceptos vistos en FC pero aplicados a un dispositivo con unas limitaciones.

1.2.1. Introduccion

Vamos a venirnos arriba y entender que es la unidad artirmeticologica o ALU para abreviar. La complejidad de su estructura viene dada por los tipos de operaciones que puede efectuar y la forma que las ejecuta pero si se usan algoritmos analogos para diferentes operaciones se puede simplificar si diseño.

La estructura basica de una ALU suele consistir en utulizar multiplexores con tantas entradas como operaciones que queremos que realice dicha ALU y en cada entrada colocar el circuito que ha de realizar la operacion en cuestion. Aqui se muestra dos ALU, la primera mostrada a mas detalle podria hacer 4 funciones pero solo hay 3 definidas y la segunda se muestra con la tabla de verdad para las 8 posibles funciones.





Código de Selección			Función	
S2	S1	S0	ALU	
0	0	0	$A+B$	UNIDAD ARITMETICA
0	0	1	$A-B$	
0	1	0	$A+1$	
0	1	1	$A-1$	
1	0	0	$A \bullet B$	UNIDAD LOGICA
1	0	1	$A+B$	
1	1	0	$\overline{A}$	
1	1	1	$A \oplus B$	

Tened en cuenta que la ALU ademas almacena algunos valores en registros, como el acarreo obtenido, la constante cero, indicador de desbordamiento, depende de como se defina la ALU. La tabla a continuacion, para que engañarnos, nos la pasaremos por el forro pero yo la dejo.. tened en cuenta que COmb: combinacional, UC:unidad de control, Copr:coprocesador,prg: programa y secu:secuencual son siglas de como se realizara la operacion respecto a la potencia del calculo.

	Potencia de cálculo			
Operación	Mínima	Baja	Media	Alta
Suma/Resta en binario	Comb	Comb	Comb	Comb
Suma/Resta en coma flotante	Prg/Copr	Prg/UC	UC	Secu
Multiplicación en binario	Prg/Copr	Prg/UC	UC	Comb
Multiplicación en coma flotante	Prg/Copr	Prg/UC	UC	Secu
División en binario	Prg/Copr	Prg/UC	UC	Secu
División en coma flotante	Prg/Copr	Prg/UC	UC	Secu
Operaciones lógicas	Comb	Comb	Comb	Comb
Desplazamientos unitarios	Comb	Comb	Comb	Comb
Desplazamientos múltiples	Prg	Prg/UC	UC	Comb

1.2.2. Unidad logica

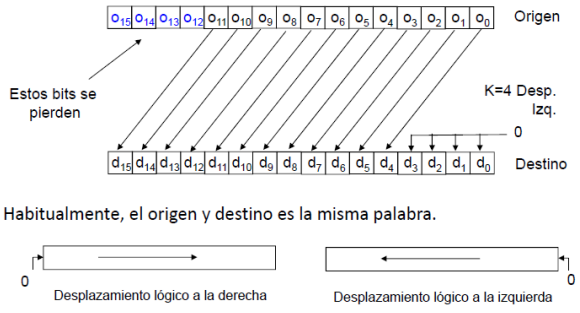
Faciles de implementar ya que corresponden directamente con las puertas logicas usadas en el hardware. Bastaria con usar un multiplexor que controle la salida para la operacion deseada.

y ya ta, te repasas de FC las puertas logicas y los multiplexores si quieres, no te joe

1.2.3. Operaciones de desplazamiento

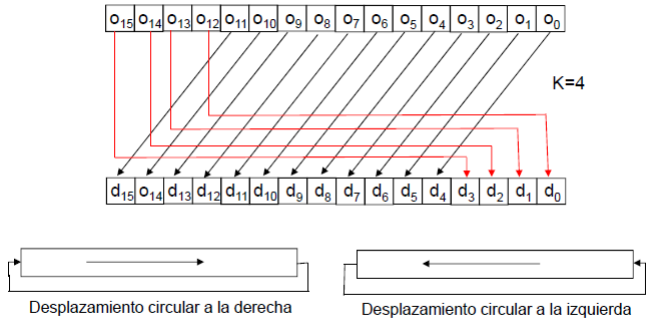
Desplazar bits en un sentido u otro puede parecer sencillo pero, tenemos una cantidad limitada de bits y por lo tanto los extremos de la cadena de bits pueden verse afectados de diversas formas. Trataremos tres tipos de desplazamientos respecto a como interactuan con los extremos:

Desplazamientos logicos El mas sencillo, basicamente desplaza los bits en una direccion y una cantidad indicada y rellena el extremo vacion con 0



Como esta mierda aburre he decidido meter una tonteria, por las risas, perdonarmelo quien lo lea.

Desplazamientos circulares Desplaza los bits en el sentido y la cantidad de veces que se le indique y el extremo que hubiera desaparecido se incluye en el extremo que queda vacio del resultado.



Desplazamiento aritmetico En este caso la cantidad de posiciones que se desplaza un bit y la direccion en la que lo hace tiene gran importancia.

El desplazamiento de n posiciones a la izquierda equivale a multiplicar por 2 n veces, es decir, multiplicar por 2^n, rellenando con ceros a la derecha. Al trabajar en C_2, el bit de mayor peso indica el signo y, si al acabar el desplazamiento, este bit ha cambiado de valor, se ha producido un desbordamiento.

Ala derecha equivaldria la division por 2^n con n las posiciones desplazadas. El bit de mayor peso indicara que valor se introducira al realizarse el desplazamiento, 0 si es positivo 1 si es negativo.

Implementacion de una operacion de desplzamiento Usando un multiplexor, considerando la señal de control dada en C_2 indicando las posiciones a desplazar (positivos a la izquierda negativos a la derecha) o usando puertas logicas AND una entrada siendo el bit que se movera y la otra la señal de control conectadas a una puerta OR cuya salida sera el bit donde se almacenara el valor se puede construir este circuito perfectamente.

1.2.4. Suma y resta

La suma y la resta se realiza mediante el uso de circuitos combinacional si se trata con valores en binario, mas adelante veremos como tratar con coma flotante. Como realizamos circuitos combi-

nacionales los cuales van usando subcircuitos, vamos a ir desde el mas pequeño y simple hasta el mas complejo. Tened en cuenta las puertas logicas XOR (equivalencia negada) XNOR (equivalencia) AND(cierto si ambas lo son) OR(falsa si ambas lo son).

Semisumador binario, H.A. Usa una puerta XOR para calcular la suma y una puerta AND para conocer el acarreo. Las operaciones del XOR se representan con $S = A \oplus B = \overline{A}B + A\overline{B}$ y las de AND con $C = AB$. Este circuito solo depende de los bits de entrada para realizar las dos operaciones con lo cual tardara un ciclo en obtenerse los dos resultados.

Destacar que la puerta XOR se puede descomponer en una puerta OR conectada a dos puertas AND una con A negada y B y la otra con A y B negada.

Sumador completo, F.A. Un sumador el cual tiene en cuenta el acarreo anterior. Tendra 3 entradas las cuales seran el bit correspondiente de los dos sumandos y el bit de acarreo a considerar y dara dos salidas, el resultado de la suma y el acarreo que esta da.

La suma se puede realizar sumando A y B y al valor obtenido sumarle el acarreo mediante puertas XOR, es decir, $S = (A \oplus B) \oplus C_{in} = (\overline{A}B + A\overline{B}) \oplus C_{in}$. El acarreo, por otro lado, consiste en sumar el acarreo de la suma de A y B y el producto del acarreo inicial y el resultado de la suma de A y B, es decir,$C_{out} = AB + C_{in}(A \oplus B)$

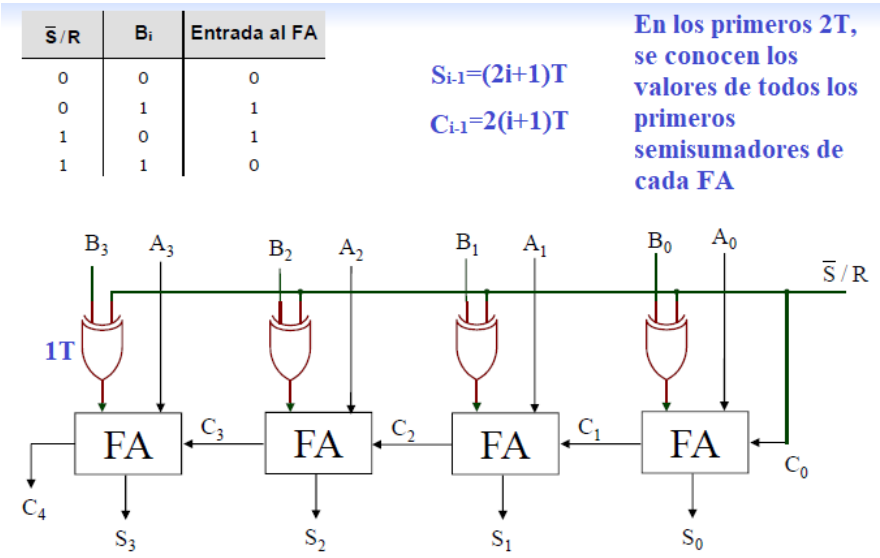
Podemos definir al sumador mediante puertas logicas basicas (puertas NOT $A \rightarrow \overline{A}$, AND $ABC...$ o OR $A + B + C + ...$) si decidimos descomponer las operaciones anteriores o podemos simplificarlo usando dos semisumadores. Realmente no existe diferencia entre el tiempo de realizacion entre ambos, pero es mas comodo expresarlo con dos semisumadores ademas de que para futuras definiciones es mas practico este metodo.

El circuito por cada posicion de bit consiste, tomando el bit correspondiente de A y B, se realiza la operacion de un semisumador como si nada, obteniendo un resultado y un acarreo en 1T. El resultado y el acarreo inicial entran al segundo semisumador dando el resultado final y y otro acarreo, la operacion tarda 1T pero depende del resultado anterior y del acarreo introducido. Por ahora sabemos que el resultado final para el primer bit con el acarreo inicial dado tardara 1T por el primer semisumador y otro 1T por el segundo, 2T en total.

El acarreo final se obtiene con una puerta OR comprobando los dos acarreos obtenidos, el primero se obtiene en el primer semisumador PARA TODOS LOS BITS, es decir, 1T constante y el segundo depende del segundo semisumador.

La explicacion de como obtener el tiempo para obtener cada valor es compleja. Debeis de entender que los resultados para todos los primeros semisumadores para cada bit se obtienen en 1T y el resultado final para el primer sumador completo de entre todos los bits tarda 2T y el acarreo final 3T (este sumador depende de un acarreo conocido en 0T). El siguiente sumador dependera de una funcion que tarda 3T, realizara el segundo semisumador y en 4T dara el resultado final y el acarreo final lo dara en 5T. Para entendernos, $S_i = 2i$ y $C_i = 2i + 1$, recordad que las operaciones se realizan todas a la vez si estan los valores de lo que depenen obtenidos.

Para hacer un restador podemos considerar B dado en C_1, es decir, negado, y hacer un acarreo inicial a 1 ya que $-B = C_2(B) = C_1(B) + 1 = \overline{B} + 1$. Esto se puede hacer de dos formas, con constante asumiendo que el circuito sera estrictamente un restador (B se niega y el acarreo inicial es constante a 1) o con una señal de control en la que 0 equivale suma y 1 resta, siendo este el valor para el acarreo inicial y una puerta XOR que conecta cada bit de B a la señal de control, si la señal de control es 0 pasara el valor de B y si es 1 pasara $\overline{B}$, teniendose entonces, junto el acarreo inicial, $-B = C_2(B) = C_1(B) + 1 = \overline{B} + 1$.



Si le añadimos un XOR conectado a los dos ultimos acarreos podemos detectar si se produce desbordamiento.

CPA, Carry Propagate Adder El CPA vamos a entenderlo como un circuito en el cual una serie de sumadores completos estan construidos en cascada, es decir, lo que hemos realizado hasta ahora pero vamos a ver ahora otra forma de hacerlo.

CLA, Carry Loohahead Adder El CLA es un sumador con anticipacion de acarreo. Hay que tener en cuenta:

1. Trabajamos en grupos de 4 bits para simplificar
2. $A_iB_i = G_i$ sera la señal generador de acarreo(si se activa entonces habra acarreo en el siguiente estado si o si)
3. $A_i \oplus B_i \simeq A_i + B_i = P_i$ es la señal propagadora de acarreo(entenderiamos que es el valor que puede acabar propagando el acarreo)
4. $C_i = G_i + P_iC_{i-1}$ es el acarreo por etapa, si se genera acarreo entonces seguro que habra, y si la señal propagador junto con el acarreo anterior indican que se generia acarreo para esa etapa.
5. Definiendo de forma regresiva, y conociendo $P_i, G_i \forall i \in \mathbb{N}, C_{-1}$

El construir mediante puertas logicas puede ser complejo ya que consta de tres niveles aunque obtenemos el acarreo para todas las etapas a la vez, es decir, permite acortar el tiempo de ejecucion de todos sumadores, dando el acarreo inicial para todos ellos al mismo tiempo y obteniendo el resultado final de todos a la vez.

1.2.5. Multiplicacion

Consistira en un algoritmo de sumas y desplazamientos en el que, si el multiplicando tiene n bits y el multiplicador m bits, el producto tendra maximo n+m bits. Ademas, la multiplicacion binaria es sencilla ya que trabajamos sobre 1 o 0. Vamos a ver paso a paso desde lo mas sencillo hasta lo mas complejo. Se hara una cosa, si $n > m$ se extendera el multiplicador para que tenga n bits y, si $n < m$ entonces se extendera el multiplicando, la idea es que el producto tenga como maximo el doble de bits que los operandos.

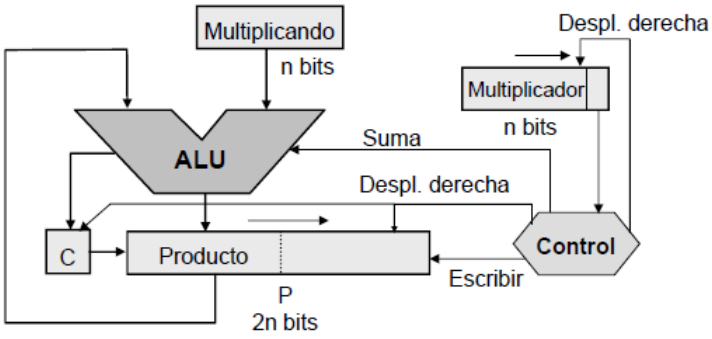
Multiplicacion binaria sin signo Consiste en realizar un algoritmo en el que:

1. Tiene en cuenta el tamaño del multiplicador (B), de n bits
2. Comprueba el valor del bit de menor peso de B
3. Si el valor es 1, suma en los n bits de mayor peso del producto (que es de 2n bits) el valor del multiplicando (A), si fuese 0 entonces sumaria 0
4. Desplaza 1 bit a la derecha el contenido de B y del producto
5. Si no se ha realizado n veces, volver al paso 2

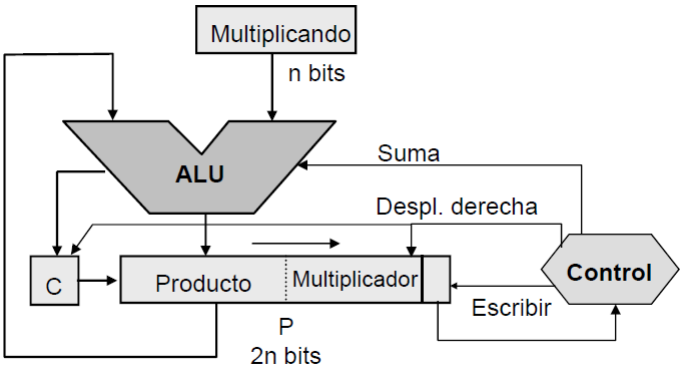
Veamoslo en accion en este circuito:

1. Producto a 0
2. Control comprueba el bit de menor peso del multiplicador
3. Indica a la ALU si sumar a P n ceros o A en los n bits de mayor peso
4. Control indica al multiplicador y a P desplazarse 1 bit a la derecha
5. Si no se ha realizado n veces, volver a 2, en caso contrario, finalizar

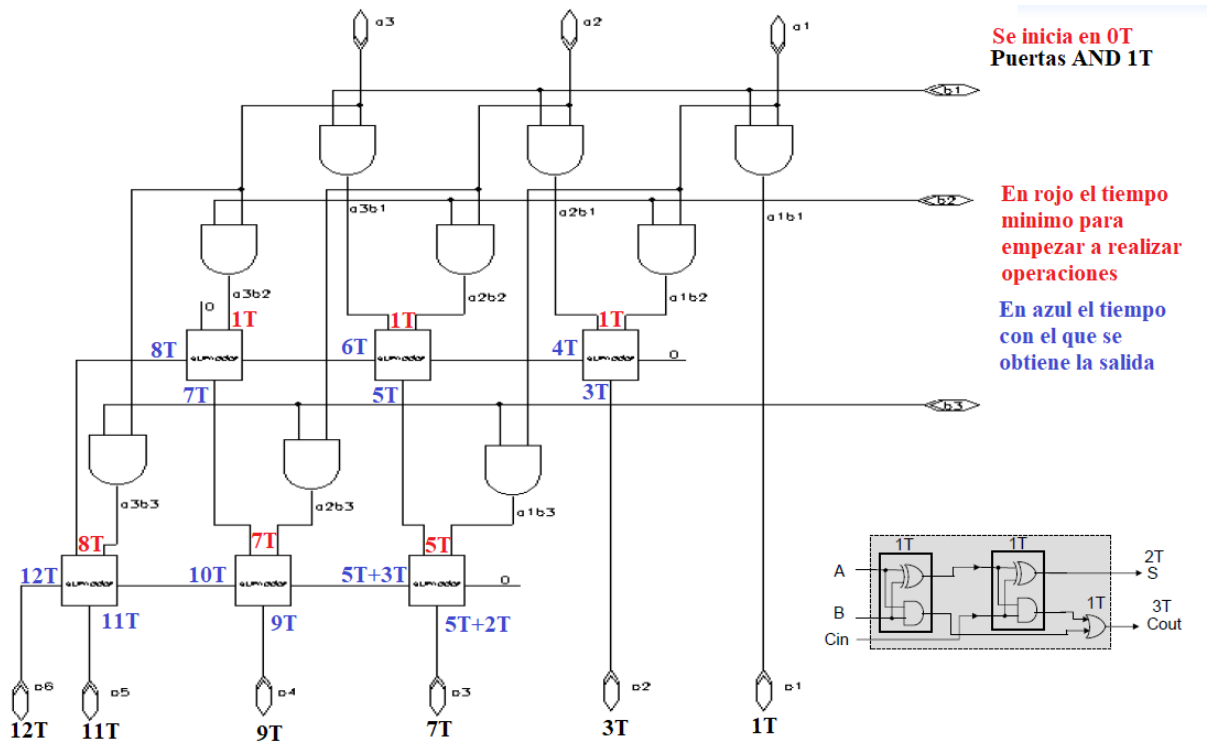
El registro C es para el acarreo, que hay que considerarlo, introduciendolo al desplazar el producto a la derecha



El circuito se puede simplificar si le damos de valor inicial al producto el valor del multiplicador. Al hacer esto, cuando desplazamos a la derecha el producto, tambien lo hacemos con el multiplicador y, ademas, comprobamos el bit de menor peso del producto para realizar la determinada accion. Como se debe de desplazar n veces el producto, el contenido del multiplicador acaba desapareciendo, dejando el producto final.



Si quisieramos hacer una multiplicacion rapida construida por un circuito combinacional deberiamos tener una primera fase en la que se obtiene cada posible producto, es decir, una puerta AND que conecta cada posible par de bits del multiplicador y multiplicando. n-1 circuitos en cascada de sumadores completos con los que sumar el producto de del multiplicando por un bit del multiplicador y el producto de del multiplicando por el siguiente bit del multiplicador. El tiempo que tardaria un circuito de este estilo depende estrictamente de los sumadores completos, en concreto del ultimo sumador, por ejemplo



Multiplicacion binaria con signo Al operar con signo, debemos considerar que trabajamos con numeros en C_2. El algoritmo reacciona de distinta forma segun detecta si el multiplicador es negativo o si lo es el multiplicando. Si lo es el multiplicando, se extendera el signo en todas las iteraciones, si lo es el multiplicador, en la ultima iteracion (en el bit del signo del multiplicador) se añade el valor del multiplicando en C_2, es decir, en lugar de sumarse, se resta.

Antes de seguir, vamos a conocer el algoritmo de Booth, el cual consiste en realizar el menor numero de productos parciales buscando plinomios compuestos por menos potencias de 2.

Para simplificar como funciona el algoritmo de Booth, vamos a considerar esta tabla:

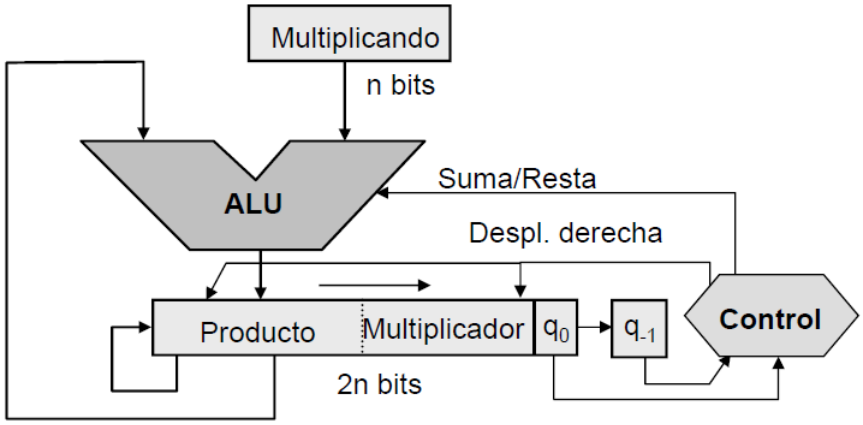
Bit n	Bit n-1	Sustitución
0	0	0 (No hay transición)
1	0	-1 (Transición hacia negativo)
0	1	+1 (Transición hacia positivo)
1	1	0 (No hay transición)

Para n=0, el bit en n-1 es 0

Esta tabla indica los pasos a seguir para recodificar el multiplicador, por ejemplo, si tenemos 7 en binario (0111_2), al recodificarlo podemos leer que para n=0, suponiendo un 0 en n-1, tendriamos 1-0, transicion a negativo. Con n=1, 1-1, no hay transicion. Con n=2, 1-1, no hay transicion. Con n=3, 0-1 transicion hacia positivo. Recodificado seria $(+1)_3 0_2 0_1 (-1)_0 \Rightarrow +2^3 + 0 + 0 - 2^0 = 7$. Mi forma de recordarlo es, desde el bit n, que valor se necesita para obtener el valor en el bit n-1, p.e., teniendo 0 en n y 1 en n-1, se necesitar sumar 1 (transicion positiva. Con 1 en n y 0 en n-1, se necesita restar 1 (transicion negativa).

Multiplicar mediante el algoritmo de Booth consiste en un circuito similar al multiplicador sin signo simplificado, solo que la unidad de control toma el bit de menor peso y el anterior (obtenido al desplazar a la derecha, en la primera iteracion se considera 0) y mandando a la ALU una señal $\overline{S}/R$, si la UC detecta una transicion positiva ($q_n < q_{n-1}$) indicara a la ALU sumar y, si detecta transicion negativa ($q_n > q_{n-1}$), indicara a la ALU restar. A continuacion desplazara a la derecha un bit el valor compuesto por el producto y el multiplicador tantas veces como bits tenga el multiplicador.

Inicialmente q_{-1}=0
Repetir n veces
    Si q_0 = 1 y q_{-1} = 0 entonces
        $Producto_h$ = $Producto_h$ - Multiplicando
    Si q_0 = 0 y q_{-1}=1 entonces
        $Producto_h$ = $Producto_h$ + Multiplicando
    Desplazamiento aritmético a la derecha de Producto y q_{-1}
Fin repetir.



1.2.6. Division

Para realizar la division debemos de tener en cuenta:

1. Dividendo=Cociente x Divisor + Resto
2. El resto sera de igual tamaño que el divisor pero de menor valor, mientras que el dividendo sera el doble de tamaño. Por ahora trabajaremos con operandos positivos

Vamos a ver el algoritmo sin restauracion para obtener la division:

1. Inicializacion: considerando que el divisor sera de n bits, y el dividendo de 2n (si no se dan asi, se extiende), restaremos el divisor a los n bits de mayor peso del dividendo.
2. Comprobamos si los n bits de mayor peso corresponden a un valor positivo o negativo, y desplazaremos a la izquierda los bits, el bit de menor peso queda vacio.
3. Si el valor obtenido es positivo, le restamos el divisor a los n bits de mayor peso, en el caso de ser negativo, se le suma.
4. Se vuelve a comprobar los n bits de mayor peso, si el valor es negativo, se introduce un 0 en el bit de menor peso que se habia quedado vacio, si es positivo se introduce 1.
5. Si no se ha realizado n veces, se vuelve al paso 2. Si se ha finalizado el bucle comprobamos si es necesario realizar una restauracion del resto, el cual se encuentra en los n bits de mayor peso de lo que era el dividendo. Si el resto da negativo, se le suma el divisor, si es positivo, no se hace nada.
6. En los n bits de mayor peso de lo que era el divideno extraemos el resto y en los n bits de menor peso estara el cociente.

Aqui teneis un ejemplo con la restauracion, si el valor de dividendo$_H$, que equivale a la mitad de bits mas pesados del dividenos, fuese positivo al finalizar el bucle entonces no se haria nada mas.

Dividendo	Divisor	Acción	Iteración
00000 01101	00101	Valores iniciales (13÷5)	0
11011 01101	00101	DividendoH - Divisor	0
10110 1101_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	1
11011 1101_	00101	DividendoH + Divisor	1
11011 11010	00101	Dividendo < 0 ⇒ q_0 = 0	1
10111 1010_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	2
11100 1010_	00101	DividendoH + Divisor	2
11100 10100	00101	Dividendo < 0 ⇒ q_0 = 0	2
11001 0100_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	3
11110 0100_	00101	DividendoH + Divisor	3
11110 01000	00101	Dividendo < 0 ⇒ q_0 = 0	3
11100 1000_	00101	Desplazar Izquierda. Dividendo < 0 ⇒	4
00001 1000_	00101	DividendoH + Divisor	4
00001 10001	00101	Dividendo > 0 ⇒ q_0 = 1	4
00011 0001_	00101	Desplazar Izquierda. Dividendo > 0 ⇒	5
11110 0001_	00101	DividendoH – Divisor	5
11110 00010	00101	Dividendo < 0 ⇒ q_0 = 0	5
00011 00010	00101	DividendoH < 0 ⇒ DividendoH + Divisor	5

Resto Cociente

1.2.7. Aritemtica en coma flotante

Vale, ya hemos jugado a las casitas y vamos a lo "divertido", veremos suma, resta, multiplicacion y division con valores en coma flotante asique debemos de darle un repaso, en especial, trabajaremos con el estandar mas empleado, el IE^3.

- Exponente representado en exceso 2^{q-1}-1
  - Exceso a 127 en simple precisión
  - Exceso a 1023 en doble precisión
- Mantisa en valor absoluto; fraccionaria y normalizada con un uno implícito a la izquierda de la coma decimal.
  - Mantisa de la forma 1,XXXXXX
  - El primer uno nunca estará representado
  - Valores posibles entre 1,00000..... y 1,11111....
- S es el signo de la mantisa
- El valor del número representado vendrá dado por:
  - $(-1)^S$ x 1,M x 2^{E-127} simple precisión
  - $(-1)^S$ x 1,M x 2^{E-1023} doble precisión

E	M	Valores
2^q-1	≠0	NaN (no un Número)
2^q-1	0	+∞ y -∞ según el signo de S
0	0	Cero
0	≠0	Números desnormalizados

NaN resultado de operaciones tales como 0/0, $\sqrt{-1}$

El valor cero tiene dos representaciones +0 y –0.

$e_S = E_{10} + S$

e_S:exponente en formato sesgado(8bits)

E_{10}:exponente en decimal

S: Sesgo normalizado o desnormalizado

Simple precisión (32 bits)

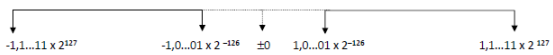
1 bit	8 bits	23 bits
signo	exponente	mantisa

Doble precisión (64 bits)

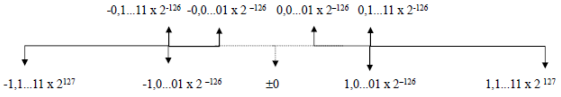
1 bit	11 bits	52 bits
signo	exponente	mantisa

Formato desnormalizado

- 0,M x 2^{-126} simple precisión
- 0,M x 2^{-1022} doble precisión



Sin números desnormalizados



Con números desnormalizados

En el caso de necesitar un repaso, mirarse FC tema 1.

De todas formas, un proceso resumido con el que realizar la conversion de decimal a IE^3

1. Representar en coma fija (parte entera y parte fraccionaria sin exponentes)
2. Pasar las partes entera y fraccionaria decimales a binarias

3. Normalizar la mantisa, es decir, desplazar la coma n veces hasta que quede 1 bit en la parte entera y sea de valor uno. Si el desplazamiento es a la izquierda n sera positivo y si es a la derecha con n negativo,indicandolo con 1.m x 2^n. m sera la mantisa y n el exponente sin sesgar.

4. Normalizamos el exponente, el cual esta en formato sesgado (el sesgo es 2^q-1, q siendo el numero de bits para el exponente) siendo $e_S=n+2^q-1$

5. Representamos el IE[3], signo que obtenemos del principio, exponente sesgado del paso 4 y mantisa del paso 3.

La suma y la resta Debemos de tener en cuenta los siguientes pasos para construir el circuito sumador/restador en coma flotante:

1. Igualar los exponentes de los dos numeros desplazando la mantisa hacia la derecha tantas veces como la diferencia de los exponentes (en valor absoluto?).

2. Operar las mantisas teniendo en cuenta la operacion seleccionada y el signo de ambos numeros.

3. Normalizar el resultado, es decir, dejar la parte entera a la unidad (en binario es facil, desplazas hasta tener un solo uno, en decimal hasta obtener un solo digito) y redondear al numero de bits apropiado.

En esta tabla comprobamos las operaciones a realizar segun el signo de A y B(SA y SB) y si actua como sumador o restador, valores hace referencia al signo del resultado:

SA	SB	Operación	Operación Real	Valores
0	0	Suma	A+B	0
0	1	Suma	A-B	Según resultado
1	0	Suma	B-A	Según resultado
1	1	Suma	A+B	1
0	0	Resta	A-B	Según resultado
0	1	Resta	A+B	0
1	0	Resta	A+B	1
1	1	Resta	A-B	Según resultado

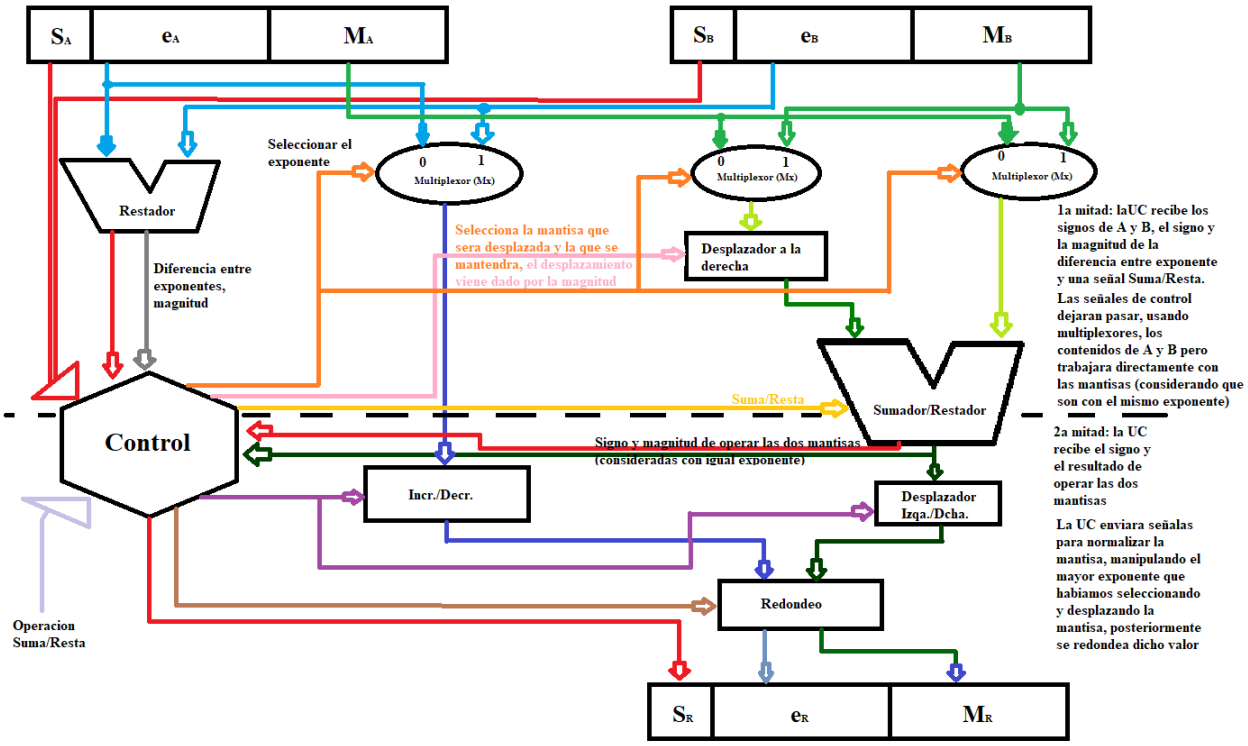
El circuito que podriamos usar para realizar esta operacion seria el siguiente (se han usado colores e indicaciones para que se pueda interpretar que se esta realizando y que contiene cada bus), tengamos en mente que:

1. Se extrae el singo y el exponente de los operandos, se obtiene la diferencia de los exponentes y la UC recibe los signos de los operandos, el resultado de la diferencia (signo incluido) y una señal para indicar si es suma o resta.

   Los signos de los operandos los necesita para conocer que tipo de operacion a realizar en relacion a que se solicita (la tabla anterior es un claro ejemplo), generara la señal de control para la ALU sumador/restador. La diferencia de los exponentes es necesaria para obtener cual es mayor y por cuanto, esto indicara cual exponente debera de pasar por el multiplexor (el mayor en valor absoluto) y la cantidad a desplazar la mantisa del de menor exponente.

2. Se desplazara la mantisa de menor exponente que pasa por un multiplexor y, la mantisa restante, pasara por el otro, estando ambas conectadas a la ALU sumador/restador la cual operara los valores segun la señal recibida por la UC. El resultado dara inicio a la segunda solicitud a la UC, pasandole el signo y la magnitud del resultado.

3. La UC enviara señales de control a un Incr./Decr. para el exponente que hemos dejado pasar y un Desplazador Izqa./Dcha. para el resultado de la ALU con la finalidad de normalizar el resultado y, finalmente, otra señal a redondear. El signo llegara de la UC, exponente y mantisa de redondear.



La multiplicacion y la division En este caso, el circuito se deja para que, a traves de la explicacion, se haga (es decir, me da pereza y me marco la excusa), veamos como multiplicar y dividir en coma flotante y las pautas a seguir para hacer el circuito (el cual yo no hare pero vosotros si xd)

- Multiplicacion

  1. Sumar los exponentes y restar el sesgo ($2^q - 1$,q igual a numero de bits del exponente), esto obtendra el exponente del resultado, es decir, $e_R = e_A + e_B - (2^q - 1)$
  2. Multiplicar las mantisas en el formato dado, recordad que las mantisas normalizadas corresponden a la parte fraccionaria de de una parte entera igual a 1, con lo cual $M_R = 1.M_A \cdot 1.M_B$
  3. Procesamos los signos, recordad el mas por mas, mas... basicamente define una puerta XOR, por lo tanto $S_R = S_A \oplus S_B$
  4. En el caso de ser necesario, normalizar (comprobar signo del resultado y magnitud del resultado y tratar el exponente y la mantisa respecto al valor de la magnitud) y redondear si fuese necesario.

  - $R = (S_A \oplus S_B)(1.M_A \cdot 1.M_B)(2^{e_A + e_B - (2^q - 1)})$

- La division

  1. Restar los exponentes y sumar el sesgo ($2^q - 1$,q igual a numero de bits del exponente), esto obtendra el exponente del resultado, es decir, $e_R = e_A - e_B + (2^q - 1)$
  2. Dividir las mantisas en el formato dado, recordad que las mantisas normalizadas corresponden a la parte fraccionaria de de una parte entera igual a 1, con lo cual $M_R = 1.M_A / 1.M_B$
  3. Procesamos los signos, recordad el mas por mas, mas... basicamente define una puerta XOR, por lo tanto $S_R = S_A \oplus S_B$

4. En el caso de ser necesario, normalizar (comprobar signo del resultado y magnitud del resultado y tratar el exponente y la mantisa respecto al valor de la magnitud) y redondear si fuese necesario.

- $R = (S_A \oplus S_B)(1.M_A/1.M_B)(2^{e_A - e_B + (2^q - 1)})$

- Similitudes:

a) Los exponentes o se suman o se restan y el resultado se resta o se suma, respectivamente, al sesgo para el formato de IE[3] que se usa.

b) Las mantisas se multiplican o se dividen, es una funcion que puede ser controlada por una señal.

c) El procesamiento de signos es el mismo y la manipulacion final del resultado para normalizar y redondear tambien.

Viendo esto, podemos extraer las siguientes pautas para construir el circuito multiplicador/restador:

1. La UC recibe una señal para conocer si se multiplica o se resta. A la vez se extrae el exponente de ambos valores y se mueven a una ALU sumador/restador. La UC indicara a esta ALU que sume si la señal es de multiplicar y restara si es de dividir.

2. El resultado pasara a otra ALU sumador/restador con un operando siengo el sesgo. Actuara opuesta a la anterior ALU, recordad que si multiplicamos los exponentes se suman y al resultado se le resta el sesgo y, en la division, los exponentes se restan y al resultado se le suma el sesgo. El resultado sera el exponente final sin tratar, pasara a la UC y se mantendra en un Incr./Decr. esperando en caso de que se deba de normalizar la mantisa.

3. La UC dejara pasar los valores de las mantisas a una ALU multiplicador/divisor, la cual recibira la señal de la UC para actuar de una de las dos formas, segun la operacion que se le haya indicado al incio. EL resultado de la ALU pasara a un desplazador Izqa./Dcha.

4. Se procesan los signos con una XOR y se almacena en el resultado.

5. El resultado de la ALU pasa por la UC para comprobar si es necesario normalizarse y redondearse, desplazando la mantisa hasta obtener $1.M_R$ e incrementado/decrementando el exponente. La UC indicara que hacer dando señales de control.

6. Al terminar de redondear, se almacenan los resultado en sus campos correspondientes.

1.2.8. Tecnicas de redondeo

Por ultimo para acabar con la teoria de la ALU, despues algun ejemplo, vamos a ver algunos metodos de redondear.

Truncamiento Simplemente eliminas los bits que no quepan (tengo 18 bits y solo puedo almacenar 16, pues elimino 2). Es facil de implementar, el error sera siempre por defecto($\epsilon = |R - C|$, R siendo el redondeo por truncamiento y C el valor original. El error puede crecer rapidamente,p.e., para operaciones consecutivas.

Redondeo al mas proximo Dado dos valores V_i y V_{i-1}, si se cumple que $|V_{i-1} - C| < |V_i - C|$ entonces tomaremos el valor de V_{i-1} como R, en caso contrario sera V_i, siendo $V_{i-1} < C < V_i, i \in I$. Es decir, comprobara de las dos cotas con los bits a representar entre las que se define C, cual representacion generaria el menor error.

Bit menos significativo forzado a 1 Truncar y forzar el bit menos significativo a 1, es rapido y el error es tanto por defecto como por exceso. El error es basicamente igual a lo truncado si no se ha forzado el bit menos significativo a 1 o por exceso por se tiene que restar ya que se a forzado el 1. Recomiendo usar el redondeo al mas proximo, a mi parecer de los posibles valores, cogera el de menor error y es muy sencillo de recordar.

1.2.9. Ejercicio ejemplo

Ejercicios de la ALU consisten en operar en coma flotante, pondre un ejemplo al acabar el temario ya que un ejemplo asi no es estrictamente necesario.

1.3. Unidad de memoria, UM

Vamos a conocer ahora que es la unidad de memoria, sus caracteristicas y cual tiene mayor importancia, distinguiremos los tipos de memoria, haremos especial hincapie en la Cache y conoceremos como diseñar mapas de memoria (mapeado de la memoria).

1.3.1. Conceptos basicos

Con la maquina de Von Neumann, definida al principio del temario, indicamos que la unidad de memoria definida no solo podia almacenar datos, si no intrucciones las cuales definen programas que trabajan a su vez con los datos almacenados en memoria.

Encontraremos diversos tipos de memorias, tecnologias, estructuras, prestaciones y costes. Es mas, si solo tuviesemos un tipo de memoria esta no podria cubrir nuestras necesidades, alomejor necesitamos una memoria que sea capaz de escribir/leer con mucha rapidez u otra con gran capacidad de almacenamiento, es por ello que se describe una jerarquia de memoria, cual tiene mayor importancia.

El computador dispone de dicha jerarquia de elementos de memoria cuya localizacion es diversa, es decir, algunas unidades se encuentra en el interior de la CPU(memoria Cache), otros sobre la placa base(RAM), o externos a ambos (SDD,HDD,flash,etc),etc.

Realizan dos operaciones, escribir y/o leer y las unidades de memoria tienen al menos un mecanismo de direccionamiento y otro para el movimiento de datos (bus de direccion y de datos).

Definamos las caracteristicas de la unidad de memoria respecto:

- Ubicacion: donde se encuentra la unidad de memoria respecto al computador
    1. En el propio procesador
    2. Interna(memoria principal)
    3. Externa(memoria secundaria)
- Capacidad: Teniendo en cuenta
    1. El tamaño de la palabra
    2. El numero de palabras capaz de almacenar
- Unidad de transferencia:
    1. Palabra
    2. Bloque
- Metodo de acceso: como se accede a las celdas de memoria

    1. Secuencial: recorre la memoria hasta encontrar el dato(cintas)
    2. Directo: Acceso a vecinidad+secuencial, accede a un sector y se desplaza hasta encontrar el dato(CD)
    3. Aleatorio: tiempo de acceso constante e independiente del anterior (RAM)
    4. Asociativo: acceso por contenido(algunas cache)

- Rendimiento: por su capacidad para tratar los datos (veremos mas adelante)
    1. Tiempo de acceso
    2. Tiempo de ciclo
    3. Velocidad de transferencia
- Tecnologia:
    1. Semiconductor
    2. Soporte magnetico
    3. Soporte optico
    4. Magnetico-optico
- Persistencia: permanencia de los datos
    1. Volatil/no volatil, es decir, varia el contenido con frecuencia o no (RandomAccessMemory/ReadOnlyMemory)
    2. Borrable/no borrable (ProgrammableROM/ROM)
- Interfaz con el exterior:
    1. Sincronas: necesitan señal de reloj para su funcionamiento
    2. Asincronas: no necesitan señal de reloj
- Organizacion: disposicion o estructura fisica en bits para formar palabras(lo veremos mas adelante)

La memoria estara conectada a un monoprocesador tipo Von Neuman por dos caminos, uno bidireccional de datos (bus de datos) y otro unidireccional de instrucciones(bus de control) para que la UC las interprete.

Organizacion Llamaremos celda al elemento de almacenamiento que contiene 1 bit de informacion, se pueden organizar de diversa forma matricialmente (valores potencias de 2). Veamos algunos conceptos antes de seguir:

1. Unidad de transferencia: numero de bits que se escriben o se leen en memoria a la vez
2. Palabra: el tamaño de la palabra es la unidad basica de organizacion de la memoria, suele coincidir con el numero de bits de representacion de los numeros y de las instrucciones.
3. Unidad direccionable: tamaño minimo en que podemos direccionar la memoria, normalmente coincide con la palabra aunque en ocasiones se permite direccionar a niveles inferiores (como a bytes)
4. Direccion: posicion de la unidad de datos

5. Una memoria se identifica por el numero de palabras que pueda almacenar.

   Por un lado, recordemos que tratamos valores en base dos, con lo cual los terminos Kilo, Mega, Giga, Tera,... se asocian a potencias de 2, siendo $2^{10n}, n \in \mathbb{N} \cup \{0\}$,kilo sera 2^{10} y asi en adelante.

   Por otro, el tamaño de la palabra, compuesta por m bits, siendo n una cantidad definida por el sistema. De ese modo, podemos decir que una memoria tiene $N2^{10n}$ posiciones de un ancho de palabra o $N2^{10n}$xm (bits)

   RECORDATORIO: bits(b),Byte(B)=8bits, palabra (dado por el sistema)

La organizacion interna de la memoria hace referencia a la disposicion fisica de los bits para formar palabras. Para una memoria semiconductora distinguimos 3 tipos: organizacion 2D, $2\frac{1}{2}$D y 3D.

La explicacion de como funciona es un poco compleja ahora mismo, sera mas sencilla de entender en cuanto veamos el mapeado de la memoria pero para que se sepa ya... En 2D, suponemos un eje como las direcciones y el otro como el contenido de la palabra mientras que un decodificador que recibe la direccion indica que linea del eje de direcciones debe tratar con el de palabras. Es como una matriz en la que cada fila esta indicada con un indice y cada columna es uno de los bits de la palabra.

En 3D la idea es similar,solo que el registro se divide en dos campos, la primera mitad para un eje y la segunda para otro, estos pasan por un decodificador que apuntariar a un punto exacto en un mapa. En dicha posicion se encontraria el primer bit de la palabra y el resto se extenderia a lo largo del tercer eje cruzando perpendicularmente el punto del mapa.

Capacidad y rendimiento La capacidad de almacenamiento, el tiempo de acceso, la velocidad de transferencia y el coste por bit son aspectos importantes a la hora de diseñar una memoria. Por un lado trataremos con la capacidad, por otro con el rendimiento y finalmente hablaremos del coste por bit.

La capacidad es la cantidad de informacion binaria que puede almacenar y depende del numero de posiciones y del ancho de la palabra contenida en ella. Esto hace que memorias con distitna organizacion 8x8(8 direcciones de ancho 1 byte), 16x4(16 direcciones de ancho un nible) y 64x1(64 direcciones de ancho 1 bit) contienen 64bits. P.E.:

1. 32TByte= 32x2^{40}Bytes=32x2^{40}x8bits
2. 128Kx1=128·2^{10}x1bits128·2^{10}bits

Para el rendimiento usamos tres parametros:

1. Tiempo de acceso (T_A): depende del tipo de acceso.

   Si es aleatorio sera el tiempo que transcurre desde el instante en el que se presenta una direccion a la memoria hasta que el dato ha sido memorizado(escritura) o esta disponible para su uso(lectura). Para el resto de accesos consiste en el tiempo que se emplea en situar el mecanismo de lectura/escritura sobre la posicion deseada.

2. Tiempo de ciclo de memoria (T_C): el tiempo que transcurre entre la peticion de acceso a la memoria hasta que se pueda dar otra, T_{RES}=T_C-T_A, siendo T_{RES} el tiempo que tarda desde que el dato esta leido o disponible hasta que permite otra peticion. Esto se da en memorias de acceso aleatorio.

3. Velocidad de transferencia (V_T): Es la frecuencia con la que se pueden hacer accesos a la memoria en un seguno.

   Para memorias de acceso aleatorio, se obiente con $V_T = \frac{1}{T_C}$

   Para memorias de acceso no aleatorio sera $V_T = \frac{N}{T_N - T_A}$ siendo N el numero de bits leidos/escritos y T_N el tiempo medio en leer/escribir esos N bits.

Trabajaremos con valores muy pequeños, tened en cuenta esta tabla:

u	mu	μu	nu	pu
unidad	miliunidad	microunidad	nanunidad	picounidad
1	$1x10^{-3}$	$1x10^{-6}$	$1x10^{-9}$	$1x10^{-12}$

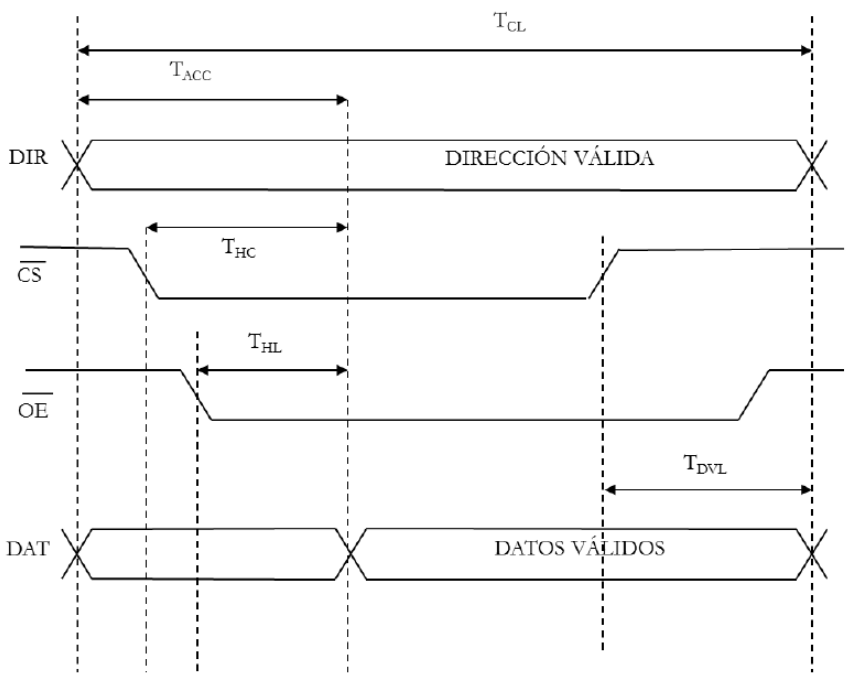
Recordad que un Hz es $s^{-1} = \frac{1}{s}$

Para resolver ejercicios de este tipo, teneis que considerar que os daran tanto rendimiento como capacidad, con lo cual alomejor te dicen que una memoria tiene un tiempo de ciclo relativamente alto pero cada posicion de direccion, es decir, el ancho de palabra esta constituidos por muchos bits, o al reves. Al hacer la velocidad de transferencia la haras siempre respecto a la palabra por direccion por segundo. Despues multiplicaras por el tiempo disonible y, si se te pide dar el resultado en una unidad en concreto, convertir la palabra a dicha unidad.

El coste por bit es el precio que se paga por cada unidad minima de informacion, el bit, en un determinado dispositivo. Basicamente el precio des dispositivo partido el almacenamiento del dispositivo dado en bits. Creo que este es obvio y no saldra.

Operaciones Solo disponemos de lectura y escritura, veamos que sucede en cada una:

- Lectura: primero se coloca la direccion a leer en el registro de direcciones y despues se activa la orden de leer.



T_{CL} Tiempo de ciclo de lectura memoria

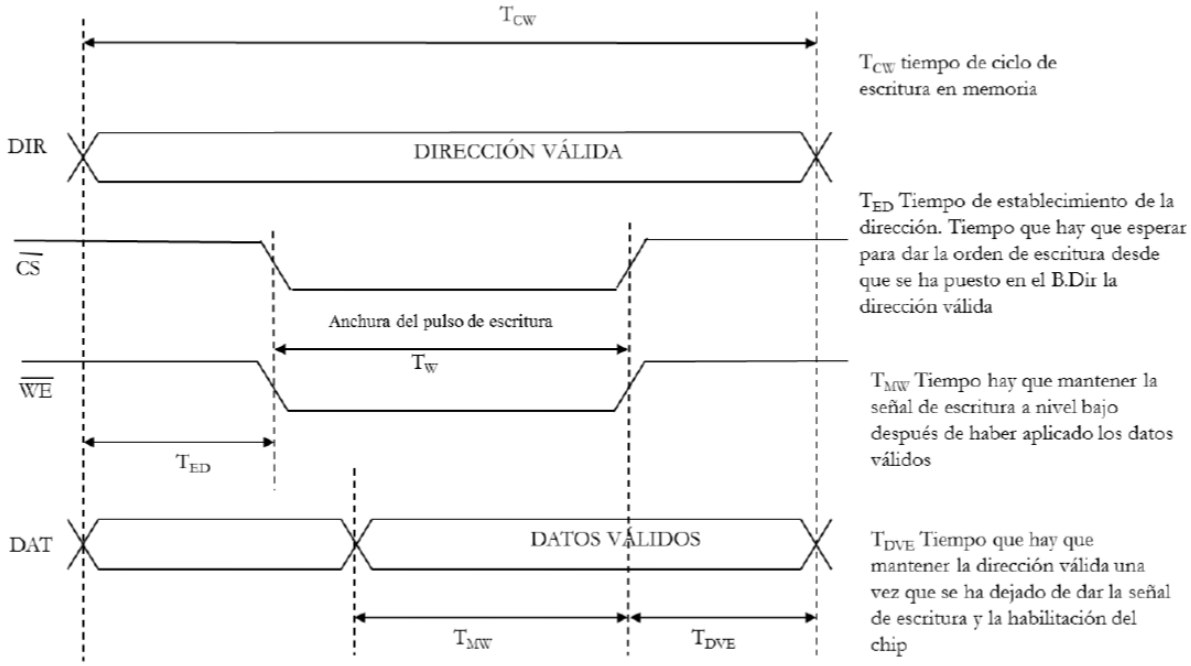
T_{ACC} Tiempo de acceso de acceso. Tiempo que transcurre desde el principio de una dirección válida hasta que los datos válidos aparecen en el B. Dat

T_{HC} Tiempo de acceso de habilitación de chip. Se mide desde que cambia CS hasta que aparecen datos válidos

T_{HL} Tiempo de acceso de habilitación de lectura. Tiempo que transcurre desde que se aplica OE, hasta que aparecen datos válidos

T_{DVL} Tiempo en el que los datos se mantienen válidos aunque se haya deshabilitado el chip

- Escritura: primero se coloca la direccion en la que escribir en el registro de direcciones, colocar el dato a introducir en el registro de datos y finalmente activar la señal de escritura.

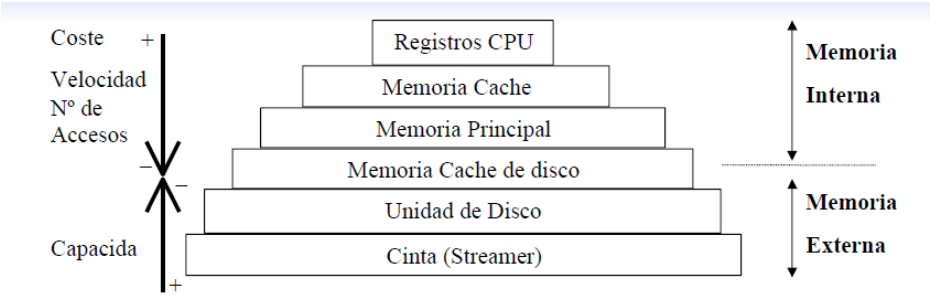


DIR:	bus de direcciones	DAT:	bus de datos
OE:	Output Enable	WE:	Write Enable
CS/CE:	Chip Select/Chip Enable	W/$\overline{R}$:	Write(1) or Read(0)

Las señales de control pueden activar por nivel alto(X:1) o nivel bajo($\overline{X}$:0)
Tambien podrian activarse por flanco de subida o de bajada, en la transicion de 0 a 1 o de 1 a 0

De esto quedaros mas con la tabla y que pasa al activarse y cosas asi, no parece entrar nada tan basto como calcular el tiempo entre cada cosa, si eso los de velocidad de transferencia.

Jerarquia de las memorias Lo ideal seria tener una memoria rapida, de mucho almacenamiento y barata, pero las cosas no son tan bonitas con lo cual, se usa un sistema de jerarquia entre los distintos tipos para obtener un sistema ideal de almacenamiento. Esta piramide la define muy bien:



Tipo	Clase	Borrado	Escritura	Volatilidad
RAM	Lectura/Escritura	Eléctricamente por bytes	Eléctricamente	Volátil
ROM	Sólo lectura	No	Mediante máscaras	No Volátil
PROM			Eléctricamente	
EPROM	Sobre todo lectura	Luz violeta, chip completo		
FLASH		Eléctricamente por bloques		
EEPROM		Eléctricamente por byte		

-RAM: RandomAccessMemory, pueden ser DRAM(Dynamic) o SRAM(Static)
-ROM: ReadOnlyMemory
-PROM:ProgrammableROM

-EPROM:Erasable Programmable Read-Only Memory
-EEPROM: Electrically Erasable Programmable Read-Only Memory

-FLASH: derivado de la memoria EEPROM, permite escritura

1.3.2. Memoria Cache

Unidad de almacenamiento que pertenece a la memoria interna del computador (como la RAM) pero es mucho mas rapida y cercana al procesador, muy costosa y que se suele dividir en tres niveles: L1,L2 y L3.

La estrategia a seguir es:

- Organizar los datos y los programas en memoria de forma que las palabras necesarias esten en la memoria mas rapida y cercana al procesador. De la memoria secundaria, la memoria primaria seleccionara una cantidad de bloques de N palabras, y la cache accedera a un bloque de la memoria primaria.
- Retener copias de las palabras utilizadas recientemente, para no tenerlas que volver a cargar de la DRAM. Mantener del bloque de palabras seleccionadas aquellas que se utilizan con frecuencia.
- Diseñar adecuadamente la cache para aumentar la tasa de aciertos de palabras en la cache, por parte del procesador. El procesador solo lee palabras que le llegaran de la cache.

La idea es que de la memoria principal, tengamos una cantidad de bloques de N palabras, entre las que se encuentra a la que queremos acceder, en la cache se carga el bloque en la que se encuentra esa palabra ademas de mantener palabras usadas frecuentemente y finalmente la CPU lee la palabra de interes en el bloque pasado, pero, si en la cache ya se encontraba la palabra que se esta buscando no hara falta todo el proceso de cargar el bloque, unicamente la palabra. Esto se obtiene aprovechando el fenomeno de la localidad de la referencias.

Caracterisiticas de la memoria cache La cache la podemos caracterizar por:

- Tamaño o capacidad de la cache, se procura mantenerla entre 16KB y 64KB para la L1, la intencion es que sea rapida, no grande. Es mas, el tamaño fisico no tiene un estandar, unicamente que quepa en el chip. En algunos computadores existe caches dedicados a tratar instrucciones por un lado y, por otro, datos.
- Tamaño de linea, cuantos bits se almacenan por direccion. Lo estandar indica que entre 8 y 64bytes, pero computadores mas potentes puede alcanzar 128 bytes.

- Funcion de correspondencia: algoritmo que hace corresponder los bloques de la memoria princial a la lineas de la cache. Distinguimos la correspondencia directa, en la cual cada bloque de la memoria principal se asocia con una linea posible de cache (sencilla pero puede producir fallos), y correspondencia asociativa, cada bloque se identifica con etiquetas unicas y estas se puede cargar en cualquier linea de cache(complicada circuiteria para encontrar la etiqueta pero soluciona los fallos de cache de la correspondencia directa).

- Algoritmo de sustitucion: unicamente para correspondencia asociativa, es el procedimiento a seguir cuando la cache se llena. Se puede sustituir la linea LessRecientlyUse(LRU), segun la FIFO, la usada menos frecuentemente o de forma aleatoria.

- El numero de caches en un computador era de uno en un principio, a dia de hoy disponemos de varias que nos permiten varios diseños: caches multinivel, p.e. L1, L2, L3... que pueden ser on-chip o off-chip(en este caso usan un bus externo a RAM) pero casi todos se hacen on-chip, y las caches unificadas frente a separadas, es decir, se combinaba el funcionamiento de las caches pero, al separarlas, podemos dedicar unas a tratar instrucciones y otras datos permitiendo aprovechar el paralelismo y a la captacion de instrucciones anticipadas.

1.3.3. Diseño de una memoria

Vale, esto es basicamente el ejercicio de pondran asique voy a orientarlo de esa manera. Debemos de tener en cuenta que la memoria tiene una estructura con lo cual primero veremos dicha estructura:

Estructura de una unidad de memoria Estan compuesto por un **circuito de chips de memoria** que determinan no solo el ancho del contenido por direccion, si no la capacidad que puede tener, **un circuito combinacional de dirrecionamiento** en el que se recibira el registro de direccion (ya sea para 2D o 3D, la diferencia es que habra que interpretar campos del registro como direcciones para los ejes) y direccionara al chip correcto, entradas y salidas con el bus de datos mediante **buffers** y una **logica de control** que recibira la señal para activar el chip indicado. En ocasiones, la logica de control depende de la direccion dada, ahora lo veremos con mas calma.

Diseño de la matriz de memoria Para ello, debemos de conocer que chips de memoria disponemos. Los chips de memoria son la unidad de memoria funcional mas pequeña, y que se pueden conectar en serie o en paralelo para construir la malla de almacenamiento de una unidad de memoria de mayor capacidad. Veamos de que esta compuesto cada chip:

- Entrada para el registro de direcciones. Es posible que el bus de direcciones admita N bits pero el chip solo necesita una cantidad de estos bits, digamos que n bits.

  Esto se debe a la cantidad de lineas de un ancho de bits, es decir, 2^nxAbits siendo 2^n el numero de lineas y A el ancho de estas en bits. Con n bits del registro de direcciones (de tamaño N) somos capaces de direccionar la memoria a todas las lineas del chip y, es mas, los bits restantes pueden ser usados como la señal de control CS o CE, que activara el chip.

  Si queremos aumentar el numero de lineas de la unidad apartir del uso de chips que no tienen dicha cantidad, los conectamos en serie. De ese modo, si queremos obtener 2^p lineas y tenemos chips de 2^n lineas (por ahora asumimos que el ancho de la linea no es necesario modificarlo), necesitaremos 2^{p-n} chips conectatos en paralelo para poder direccionar todas las lineas.

  Otra forma de verlo es poniendo ambos en la misma unidad (K,M,G,..) y realizar una division de los coeficientes. El metodo que os propongo obtienes a la vez las lineas a direccionar y la capacidad y si consideras que PK:P$x2^{10} = 2^p x 2^{10} = 2^{10+p}$, realmente puedes hacer ambas.

  Como se ha dicho anteriormente, se usaran los bits de mayor peso en un circuito combinacional para usarlo como señal de control CS o CE.

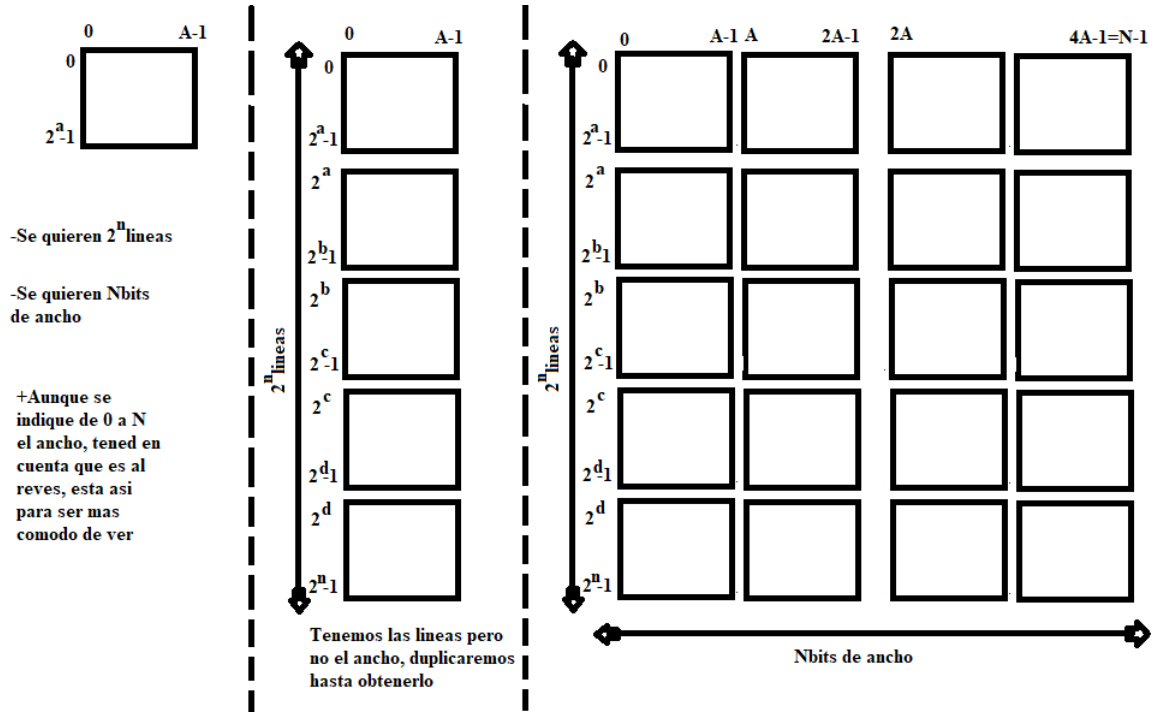
- Entrada-salida del bus de datos. El bus de datos suele indicar el ancho de la palabra del computador, con lo cual es una forma de saber cuantos bits disponemos por parte del computador. Pero no siempre trabajaremos con chips que tiene el mismo ancho, puede que sea mayor o menor que el de bus de datos.

  Pongamos el ejemplo de que tiene un bus con P=2^p bits y disponemos de chips con un ancho de N=2^nbits, podemos obtener la cantidad de chips a usar para poder obtener una unidad de memoria(sin considerar el numero de lineas) con igual ancho de linea que el bus de datos introducido obteniendo $X = P/N = 2^p/2^n = 2^p - n$, siendo X el numero de chips a usar (si $X < 1$ usaras parte de los bits a almacenar de un chip).

  Para aumentar el ancho de linea se conectar chips en serie compartiendo las mismas señales de control.

- Construir la malla de memoria completa. En este caso consideramos tanto el bus de direcciones como el de datos, eso quiere decir que tendremos que ajustar los chips de tal manera que obtenga todas las lineas que deseamos y que todas las lineas sean del ancho indicado.

  La forma mas sencilla para verlo, es usando cuadrados. Obviamente no es un metodo cientifico, eso lo pondre despues, pero si usamos cuadrados en las que el lateral consiste en las lineas de ese chip y las ponemos una encima de otra hasta obtener las lineas deseadas hemos contruido el circuito en paralelo que habra que conectar con las direcciones. Si la columna que hemos creado con el numero de lineas que queremos la vamos duplicando exponencialmente ((((Nx2)x2)...x2)) poniendola al lado, lo que hacemos es aumentar el ancho de todas las lineas, conenctando en serie los chips que direccionen a la misma linea.



  La forma matematica para saberlo consiste en juntar los dos metodos anteriores. Primero obtener las lineas que se piden usando $X = P/N = 2^p/2^n = 2^{p-n}$ siendo p el numero de bits del registro de direcciones y N el numero de lineas de un chip, X sera el numero de chips en paralelo con distintas direcciones. A continuacion usaremos $Y = B/A = 2^b/2^a = 2^{b-a}$ siendo B el numero de bits en el bus de datos que interactua con la memoria y A el ancho de bits de cada linea en un chip(o conjunto de chips en paralelo), Y sera el numero de columnas en serie que habra de hacer, cada fila hara referencia a la misma direccion.

  Podemos saber cuantos chips debemos de usar si multiplicamos

$$X \cdot Y = \frac{P\text{x}B}{N\text{x}A} = \frac{2^{p+b}}{2^{n+a}} = 2^{(p+b)-(n+a)}$$

  PxB sera como indicaremos a la unidad de memoria y NxA al chip usado.

- Como se ha dicho, el chip recibe una señal de contol CS o CE, condicionada por los bits mas pesados del registro de direcciones. Usando un circuito combinacional para estos bits, generaremos señales de control de activacion del chip para cada bloque de lineas segun la direccion recibida. Se suele usar un decodificador cuya salida estandar es 0 si la señal de control se activa a nivel alto (H) o salida estandar a 1 si se activa a nivel bajo (L). Mas adelante veremos a interpretar estas señales.

  Tambien se reciben señales de escritura (WE write enable), de lectura (OE output enable) o de ambas ($W/\overline{R}$ write or read). Lo mas comodo es usar write or read ya que se activaria a nivel alto la escritura y a nivel bajo la lectura, pero si estuviesen WE y OE a la vez, salvo que se indique lo contrario, la entrada para cada uno debe de ser opuesta.

- Recibira aparte una alimentacion V_{CC} y una conexion a tierra V_{SS} pero ignorarlo.

Y realmente, con esto ya somos capaces de hacer el mapeado, no hace falta mas. Vamos a probar con un ejemplo para saber como representar el desarrollo (estara escrito en gran parte a mano porque hacer las tablas y todo con Latex se puede ahcer muy largo):

1.3.4. Ejemplo UM

lo pondre al acabar al menos UC y compruebe un ejercicio (el 1 de las transparencias en la pagina 64) que me parece que esta mal planteado

1.4. Unidad central de procesamiento, CPU

Vamos a comprobar como funciona la CPU, entendiendo un poco el computador, como se construye la ruta de datos y finalmente realizando esquemas de implementacion.

1.4.1. Introduccion

Sabemos del principio del temario que un computador se compone de una unidad de memoria, unidad de E/S, de la CPU y la ruta de datos que los conecta y, a su vez, la CPU se compone de la UC, la ALU, registros de almacenamiento y un bus interno de datos.

Para ser mas concretos, trabajaremos con el procesador de MIPS el cual tiene las siquientes caracteristicas:

- Maquina de arquitectura RISC(Reduced Instruction Set Computing), cuyas instrucciones definidas son de tamaño fijo y presentadas en un reducido numero de formatos y, ademas, solo las instrucciones de carga y almacenamiento acceden a la memoria.
- El ancho de la palabra y el tamaño del bus interno es de 32 bits.
- Habran 32 registro con un ancho de una palabra(32bits).
- La unidad de memoria consta de 4Gx32bits y se puede acceder a ella por byte(B), media palabra(2bytes,H) o una palabra(4bytes,W)
- El ancho de los registro de direcciones (que se denotan con PC) y los de datos(denotados con RI,MDR) sera de una palabra(32bits)

A lo largo de este tema estudiaremos las instrucciones del MIPS, las cuales podemos por ahora agrupar en tres subconjuntos:

- Instrucciones de referencia a memoria: son aquellas de carga y almacenamiento. lw,sw,... Estan compuestas por instrucciones de formato tipo -I.
- Instrucciones aritmetico-logicas: opera dos registos con operadores aritmeticos o logicos. add,or,slt,... Estan compuestas por instrucciones de formato tipo -R.
- Instrucciones de control: permite redireccion a otra instruccion. Si redireccion depende de una condicion sera saltos condicional y este conjunto estara compuesto por instrucciones en formato tipo -I, beq,bltz,... Si la redireccion se produce sin ningun tipo de condicion, tratamos con saltos incondicionales y este conjunto esta compuesto por instrucciones en formato tipo -J, j,eret,...

El proceso, por lo general, sigue unos pasos. Primero usa el contador de programa (PC) para acceder a la direccion de la instruccion en memoria. A continuacion se leeran los registros que la instruccion indica y finalmente se realizara una accion descrita por la instruccion. Se tendra que desplazar PC al menos una palabra (PC< $-$PC+4(bytes, una palabra)) para acceder a la siguiente instruccion. En ocasiones, la instruccion solicita acceder a otra direccion en la memoria de instrucciones y modifica el valor de PC.

1.5. Unidad de entrada/salida, E/S

2. Fundamento de los computadores

Aqui teneis el contenido entero de los fundamentos de los computadores, por si necesitais repsar mas contenido del que ya se ha ido aportando durante las explicaciones de Estructuras de los computadores

2.1. Cuatro conjuntos numéricos ya las equivalencias entre sí

2.1.1. Sistemas de numeración

Los sistemas de numeración en base b son conjuntos que utilizan un alfabeto con b elementos para representar una gran variedad de números. Principalmente trataremos con los siguientes:

1. Sistema binario(b=2), $\mathbb{B} = \{0, 1\}$
2. Sistema octal(b=8), $\mathbb{O} = \{0, 1, 2, 3, 4, 5, 6, 7\}$
3. Sistema decimal(b=10), $\mathbb{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
4. Sistema hexadecimal(b=16), $\mathbb{H} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Para representar estos conjuntos como un número, se aplican diversos métodos de conversión . Por lo general el número que se representa equivale al aportado por el sistema decimal, aunque esto depende respecto al tipo de codificación usada (se vera más adelante):

Ejemplo de equivalencia de los 16 primeros términos:

Binario	Octal	Decimal	Hexadecimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	*A*
1011	13	11	*B*
1100	14	12	*C*
1101	15	13	*D*
1110	16	14	*E*
1111	17	15	*F*

(1)

Notese que el entre el binario, el octal y el hexadecimal, existe una correlacion con el número de caracteres tal que 2^1x$2^3 = 2^4$ o lo que es lo mismo 2x8 = 16, siendo esto de ayuda para realizar conversiones de forma directa (un binario de 3 caracteres puede representar 8 números y un binario de 4, 16).

2.1.2. Sistemas de codificación y representación básicos de números

Los siguientes mecanismos de cambio de base o codificación están conectados, es decir, existe una correlación entre todos ellos con lo cual es posible que un valor tenga varias representaciones en los diversos sistemas.

Binario: Dado un número codificado en sistema binario(b=2), con $\mathbb{B} = \{0, 1\}$, observamos que la conversión a los distintos sistemas se produce de la siguiente forma:

A. A sistema octal(b=8): se agrupan en términos de 3 caracteres de derecha a izquierda en la parte entera y de izquierda a derecha la fraccionaria, rellenando con ceros los espacios necesarios para cerrar un grupo y se cambian por su equivalente. Fijarse en la tabla (1)

B. A sistema decimal(b=10): se entiende al número en binario como una sucesión de términos ordenados de derecha a izquierda con origen en el primer entero($i = 0$) y se considera el siguiente sumatorio $\sum_i x_i \cdot b^i$ donde, si consideramos el numero en binario $x = 1010,011$, podemos descomponerlo y obtener $x_3x_2x_1x_0, x_{-1}x_{-2}x_{-3}$ o, lo que es lo mismo, la sucesión de $x_i \in \mathbb{B}$ con $i = \{-3, 3\}$. Sabiendo esto aplicamos el sumatorio considerando que b es la base con la que hemos estado operando ($b = 2$) y obtenemos un número en representación decimal.

C. A sistema hexadecimal(b=16): se agrupan en términos de 4 caracteres de derecha a izquierda en la parte entera y de izquierda a derecha la fraccionaria, rellenando con ceros los espacios necesarios para cerrar un grupo y se cambian por su equivalente. Fijarse en la tabla (1)

Octal: Dado un número codificado en sistema octal(b=8), con $\mathbb{O} = \{0, 1, 2, 3, 4, 5, 6, 7\}$, observamos que la conversión a los distintos sistemas se produce de la siguiente forma:

A. A sistema binario(b=2): se descompone un término en 3 caracteres respetando el orden de cada término. Fijarse en la tabla (1)
B. A sistema decimal(b=10): Considerando el como descomponer un número en un determinado sistema a términos ordenados de una sucesión (visto en la conversión de binario a decimal) se aplica el siguiente sumatorio $\sum_i x_i \cdot b^i$, considerando que $b = 8$.
C. A sistema hexadecimal(b=16): se obtiene el binario y se aplica la conversión de binario a hexadecimal.

Decimal: Dado un número codificado en sistema decimal(b=10), con $\mathbb{D} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, observamos que la conversión a los distintos sistemas se produce de la misma forma pero cambiando un valor:

A. Parte entera: tomamos la parte entera del número en decimal y lo dividimos por b, el cociente obtenido lo volvemos a dividir por b y asi sucesivamente hasta que el cociente no pueda dividirse más. Tomaremos el número equivalente en base b del número en decimal a aquel que se forma con los restos de las divisiones y el ultimo cociente calculado, siendo el primer resto el valor mas a la derecha y el ultimo cociente el valor mas a la izquierda. El valor usado en b define a que sistema se esta convirtiendo y, usando la tabla 1 se sabe para los casos en el que el resto es un valor superior a 9.
B. Parte fraccionaria: tomando únicamente la parte fraccionaria, la multiplicamos por b, volviendo a multiplicar por b ahora el resultado. La parte entera que se obtenga con el orden usual en los resultados será la parte fraccionaria que corresponderá al número al cual tratamos de obtener representado en el sistema de base b

Hexadecimal: Dado un número codificado en sistema hexadecimal(b=16), con $\mathbb{H} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$, observamos que la conversión a los distintos sistemas se produce de la siguiente forma:

A. A sistema binario(b=2): se descompone un término en 4 caracteres respetando el orden de cada término. Fijarse en la tabla (1)
B. A sistema octal(b=8): se obtiene el binario y se aplica la conversión de binario a octal.
C. A sistema decimal(b=10): Considerando el como descomponer un número en un determinado sistema a términos ordenados de una sucesión (visto en la conversión de binario a decimal) se aplica el siguiente sumatorio $\sum_i x_i \cdot b^i$, considerando que $b = 16$.

Para referirnos al conjunto de valores representable por un sistema numérico de base b hablaremos de rangos de representacion, los cuales tienen en cuenta la cantidad de valores/cifras/bits (n) empleados en representar el total de combinaciones la base en la que se trabaje, obteniendo b^n combinaciones y abordando el conjunto $[0, \ldots, b^n - 1]$.

2.1.3. Aritmética binaria

Con este apartado, consideramos un cuerpo $\mathcal{B} = \{0, 1\}$ en el cual se define una operación asociativa no usual ($0001 \oplus 0001 = 10$ y $0001 \ominus 0011 = -0010$) y la operación multiplicativa usual y dejamos con la siguiente tabla las distintas operaciones definidas en él.

A	B	A+B	A	B	A-B	A	B	AxB	A	B	A/B
0	0	0	0	0	0	0	0	0	10	10	1
0	1	1	0	1	(1)1	0	1	0	10	11	0,111...
1	0	1	1	0	1	1	0	0	11	10	1,1
1	1	(1)0	1	1	0	1	1	1	11	01	11

En la aritmética binaria usual (con binario natural) existe un mecanismo llamado acarreamiento, lo que implica que se suma o resta un 1 en el caso de que este quede fuera de la operación (trabajas con 5 bits pero la operación realizada te saca un 1 para el sexto, dicho 1 suma o resta al número de nuevo).

2.1.4. Sistemas de codificación y representación avanzados de números

Veamos por encima los tres códigos basados en sistemas binarios con los que podremos trabajar:

1. Binario natural: el usado hasta ahora, donde para convertir un binario en decimal se aplican las conversion anteriormente explicadas.
2. Código Gray: similar al binario natural, éste es no ponderado, continuo y ciclico donde dos números sucesivos sólo varían en un bit tal que

2 bits	4 bits	Decimal	Binario natural
00	0000	0	0000
01	0001	1	0001
11	0011	2	0010
10	0010	3	0011
	0110	4	0100
	0111	5	0101
	0101	6	0110
	0100	7	0111
	1100	8	1000
	1101	9	1001
	1111	10	1010
	1110	11	1011
	1010	12	1100
	1011	13	1101
	1001	14	1110
	1000	15	1111

3. BCD (Binary Coded Decimal): de este existen varias ramas pero nos bastara con saber la natural (y la exceso X donde es sumar X a cada valor de forma individual). La codificación en BCD natural consiste en entender codificar cada dígito decimal con una combinación de 4 dígitos en binario natural.

Veamos la representación de números por partes, primero veamos los enteros y después los reales:

Representación de enteros El conjunto de números enteros $\mathbb{Z} = \{-\infty, \ldots, 0, \ldots, +\infty\}$ se caracteriza por la existencia del término negativo, por ende, debes considerarse que trabajando con los números de los distintos sistemas numéricos con términos positivos y negativos.

Antes de empezar, debemos tener claro que trabajaremos con un numero limitado de cifras o bits junto con un signo. Por lo general, usaremos en este curso el sistema binario considerando que b=2.

A. Signo y Magnitud (SM)

   Método que reserva el bit en la posición más alta (a la izquierda) al signo (0 si es positivo y 1 si es negativo) y deja el resto al valor numérico en valor absoluto. Existe doble representación del 0 y su rango de representación abarca $\mathbb{SM} = \{-(b^{n-1}-1), \ldots, 0, \ldots, b^{n-1}-1\}$ con b la base con la que se trabaja y n el número de bits usados. Esta explicación considerarla para el sistema binario, con el resto considerar el uso del signo.

   El volver al sistema numérico natural consiste en fijarse en el primer bit.

B. Complemento a base menos 1($\mathbb{C}_{b-1}$)

   Método que combina el SM y operaciones aritméticas. Los valores positivos se representan en SM, los negativos se obtienen de restar $b^n - 1$ con b la base en la que trabajamos y n el total de digitos

que se hacen uso (el − se cuenta). Esto permite representar en positivo los valores negativos para convertir restas en sumas (existe acarreamiento). En particular, con $\mathbb{C}_1$ basta con intercambiar 1 por 0 y viceversa. Su rango de representación es $\mathbb{C}_{b-1} = \{-(b^{n-1}-1), \ldots, 0, \ldots, b^{n-1}-1\}$

Para volver al sistema numérico natural basta con despejarlo de la operación $C_{b-1} = nt - b^n - 1 \Leftrightarrow C_{b-1} + b^n + 1 = nt$

C. Complemento a base($\mathbb{C}_b$)

Partiendo del $\mathbb{C}_{b-1}$, basta con saber que con los valores positivos es igual pero con los negativos se suma 1 al valor obtenido, es decir, para positivos el sistema en base b natural, el $\mathbb{C}_{b-1}$ y $\mathbb{C}_b$ coinciden(considerando un 0 delante por el término positivos) y los negativos cumplen que, asumiendo que nt es el número en el sistema en base b natural, $\mathbb{C}_{b-1} = nt - b^n - 1$ y $\mathbb{C}_b = C_{b-1} + 1 = (nt - b^n - 1) + 1$ siendo importante el no simplificar esta operación. Su rango de representación es $\mathbb{C}_b = \{-(b^{n-1}), \ldots, 0, \ldots, b^{n-1}-1\}$.

Para el caso del sistema binario, podemos buscar el primer bit de derecha a izquierda con valor 1 y intercambiar los valores a continuación por la izquierda de 1 a 0 y viceversa. La conversión al sistema numérico natural consistiría en despejar la operación. Además, al operar con números en complemento a base no se considera acarreamiento pero si convertimos restas en sumas.

D. Representación sesgada ($\mathcal{S}$)

Considerando que los términos negativos los podemos hacer positivos aplicando aplicando cualquiera de los 3 métodos anteriores y que lo usaremos básicamente sobre el sistema binario, consideremos que la representación sesgada consiste en sumar un sesgo (2^{n-1} con n el total de dígitos en uso) al número en cuestion, sin importar su signo, de tal manera que siendo n el número ya sea en natural, $\mathbb{SM}$,$\mathbb{C}_1$ y $\mathbb{C}_2$, se cumple $\mathcal{S} = n + 2^{n-1}$ pudiendo de esta operación despejar el número natural equivalente.

Los sesgos con 1 al principio suelen ser positivos y los que tienen 0, negativos. Su rango de representación abarca $\mathcal{S} = \{-2^{n-1}, \ldots, 0, \ldots, 2^{n-1}-1\}$.

Representación de reales Con la representación de los números reales incluimos la parte fraccionaria (a la derecha de la coma) a la parte entera, permitiéndonos obtener un número mas especifico que con los enteros. Distinguiremos la representación en coma fija y en coma flotante, pero haremos mayor hincapié ya que en coma fija basta con saber que del total de bit disponibles, una cantidad esta reservada a la parte entera y la otra a la parte fraccionaria, teniendo un rango de $\{-(2^{n-1}-1), \ldots, 0, \ldots, 2^{n-1}-1\}$.

Con la representación en coma flotante debemos distinguir 3 partes, las cuales son:

1. Mantisa o coeficiente

   Representado con $(s)M$ siendo s el signo y M la mantisa, es un número formado por un único dígito significativo en la parte entera seguido del resto de dígitos en la parte fraccionaria.

2. Base

   Representado por b, es la base sobre la que trabajamos

3. Exponente

   De valor estrictamente entero y representado por e, eleva la base obteniendo una potencia.

El resultado final quedaría $\mathcal{N} \equiv (s)M \cdot b^e$.

Esta es la forma general y se nos puede pedir el representar el sesgo y la mantisa con distintos métodos de representación, pero se ha establecido un estándar en el cual la forma de representar cada elemento es única, veamos lo:

- Representación estándar IEEE745 (IE^3).

  Utilizando el modelo SEM (Signo Exponente Mantisa) con la base fijada en 2, la forma mas comun, la de precisión simple, utiliza 32 bits distribuidos de una forma en particular para representar un real en coma flotante. Considerando N el total de bits, nS los bits dedicados al signo, nE los dedicados al exponente y nM los dedicados a la mantisa, vemos que $N = nS + nE + nM$. Debe considerarse que existe la versión con la mantisa normalizada y desnormalizada (pero realmente la diferencia es en el exponente con lo cual lo estudiaremos ahi).

+ Signo

  El primer bit esta destinado a indicar si es positivo (0) o negativo (1).

+ Mantisa

  Son los últimos 23 bits y se entiende que el número dado en el IE^3 es únicamente la parte fraccionaria (nos referiremos a ella como m) pero se considera oculta una parte entera con lo cual $M = [1.m]$ cumpliéndose $1 < M < 2$. m por lo general se representa en binario natural.

+ Exponetne

  Los 8 bits restante situados entre el signo y la mantisa están reservados al exponente, que se representa con el modelo sesgado, de tal manera que $E = e - \mathcal{S}$ siendo E el valor numérico del exponente, e el número representado en sesgado y $\mathcal{S}$ el sesgo, que en formato de mantisa normalizada es $\mathcal{S} = 2^{nE} - 1$.

  Hablaremos de mantisa desnormalizada cuando $e = 0 \dots 0$, en esta caso tomaremos $\mathcal{S}_d = 2^{nE} - 2$ haciendo que el valor del exponente sea ahora $E = e - \mathcal{S}_d$ produciendose que $E = 0 \dots 0 - (2^{nE} - 2) = -2^{nE} + 2$. Se utiliza para números próximos al 0.

Recordar que dada $m = 0 \dots 0$, dará 0 si $e = 0 \dots 0$ y ∞ si $e = 1 \dots 1$, que para redondear un número en IE^3 lo habitual es el redondeo al par (si hay dos 1 consecutivos, se suma 1 en la posción siguiente, si no sucede ésto se truncan) y para representar el rango que abarca veremos:

Nº normalizados		Nº desnormalizados	
$\|b\| = M_{max} 2^{E_{max}}$	$M_{max} = 2 - 2^{-nM}$		
	$E_{max} = 2^{nE-1} - 1$		
$\|a\| = M_{min} 2^{E_{min}}$	$M_{min} = 1$	$\|a'\| = M'_{min} 2^{E'_{min}}$	$M'_{min} = 2^{-nM}$
	$E_{min} = -(2^{nE-1} - 2)$		$E'_{min} = -(2^{nE-1} - 2)$

donde		
$M = \{1, \dots, 1{,}111 \dots\}$	con $1 < M < 2$ y	$nM = 23$ para IE^3
$E = \{-126, \dots, +127\}$	con	$nE = 8$ para IE^3

siendo el rango $\mathcal{N} = [-b, -a'] \cup [a', b]$ y hablaremos de OVERFLOW cuando $|\mathcal{N}| > |b|$ (desborda en dirección de $\pm\infty$) y de UNDERFLOW cuando $|\mathcal{N}| < |a'|$ (desborda en dirección de 0).

2.2. Álgebra de Boole. Fundamentos de los sistemas digitales.

Trabajando sobre un cuerpo en el que predomina la aritmética binaria, definimos un conjunto de propiedades que dará lugar al Álgebra de Boole, permitiéndonos estudiar de forma teórica el funcionamiento de los transistores, un componente esencial en los sistemas digitales que actúa como llave electrónica o conmutador entre dos estados: encendido (1) o apagado(0).

2.2.1. Álgebra de Boole

El Álgebra de Boole es una gran herramienta matemática para analizar y diseñar circuitos digitales, considerando el conjunto con el que trabaja ($\mathbb{B} = \{0, 1\}$) y la operaciones descritas en él (aritmética binaria). Considerando ésto, veamos algunas propiedades básicas.

Conocimientos básicos RECORDATORIO:

Consideramos un cuerpo $(\mathbb{B}, \oplus, \cdot)$ con $\mathbb{B} = \{0, 1\}$ en el cual se define una operación asociativa no usual ($0001 \oplus 0001 = 0010$ y $0001 \ominus 0011 = -0010$) y la operación multiplicativa usual y dejamos con la siguiente tabla las distintas operaciones definidas en él.

A	B	A+B	A	B	A-B	A	B	AxB	A	B	A/B
0	0	0	0	0	0	0	0	0	10	10	1
0	1	1	0	1	(1)1	0	1	0	10	11	0,111...
1	0	1	1	0	1	1	0	0	11	10	1,1
1	1	(1)0	1	1	0	1	1	1	11	01	11

Las propiedades que obtenemos directamente de definir el cuerpo $(\mathbb{B}, \oplus, \cdot)$ son llamadas axiomas, y las que desarrollamos a partir de estos los llamaremos leyes o teoremas, veamos los esenciales y demostremos algunos de ellos:

Identificador	Nombre/Propiedad	Igualdad
Ax.1	Axioma 1 Asociativa	$(a+b)+c = a+(b+c)$
Ax.2	Axioma 2 Conmutativa	$a+b = b+a$ $a \cdot b = b \cdot a$
Ax.3	Axioma 3 Complementación	$a+\overline{a} = 1$ $a \cdot \overline{a} = 0$
Ax.4	Axioma 4 Absorción	$a+(a \cdot b) = a$ $a \cdot (a+b) = a$
Ax.5	Axioma 5 Idempotencia	$a+a = a$ $a \cdot a = a$
Ax.6	Axioma 6 Distributiva	$a \cdot (b+c) = a \cdot b + a \cdot c$ $a+(b \cdot c) = (a+b) \cdot (a+c)$
PD	Teorema 1 Principio de dualidad	Dado un enunciado que es válido, su enunciado dual ($1 \Leftrightarrow 0$ y $+ \Leftrightarrow \cdot$) también lo sera
EN	Teorema 2 Elementos neutros	$a+0 = a$, 0 es el neutro en sumas $a \cdot 1 = a$, 1 es el neutro en productos
EA	Teorema 3 Elementos absorbentes	$a+1 = 1$, 1 es el absorbente en sumas $a \cdot 0 = 0$, 0 es el absorbente en productos
LsM	Teorema 4 Leyes de Morgan	$\overline{a+b} = \overline{a} \cdot \overline{b}$ $\overline{a \cdot b} = \overline{a} + \overline{b}$
TI	Teorema 5 Teorema de involución	$\overline{\overline{a}} = a$
TCON	Teorema 6 Teorema del consenso	$a \cdot b + \overline{a} \cdot c = a \cdot b + \overline{a} \cdot c + b \cdot c$ $(a+b) \cdot (\overline{a}+c) = (a+b) \cdot (\overline{a}+c) \cdot (b+c)$
TCAN	Teorema 7 Teorema de cancelación	$a + \overline{a} \cdot b = a + b$
TSH	Teorema 8 Teorema de Shannon	Toda función representada en suma de productos tiene un equivalente en producto de sumas
Considerando	$\mathbb{B} = \{0, 1\}$, las propiedades se cumplen	$\forall a, b, c \in \mathbb{B}$

Muchos de los teoremas definidos se basan en el principio de dualidad (Leyes de Morgan, involución, Shannon (este ademas considerando la distributiva),...), los de la existencia de elementos neutros y absorbentes es trivial considerando la complementación y la absorción, el de cancelación se obtiene aplicando distributiva y después complementación y el del consenso es aplicar de forma correcta distributiva y complementación para simplificar o ampliar la función. No son complicados de realizar pero debe tenerse claro que se quiere obtener para poder encaminar la operación de forma correcta.

Representación de funciones booleanas Veremos 3 métodos de los que disponemos para representar funciones booleanas:

- Tablas de verdad:

  Estructuras de $2^n + 1$ filas (primera fila para las variables y el resultado y las 2^n restantes para las distintas combinaciones) y $n + 1$ columnas, siendo n el total de variables y la ultima columna la solución respecto a cada combinación posible. Son un método sencillo para estudiar los resultados obtenidos para cada combinación y tienen la siguiente forma:

x_1	x_2	...	x_i	$f(x_1, x_2, \ldots, x_i)$
0	0	...	0	$f(0, 0, \ldots, 0)$
0	0	...	1	$f(0, 0, \ldots, 1)$
⋮	⋮	⋮	⋮	⋮
1	1	...	1	$f(1, 1, \ldots, 1)$

- Representación algebraica:

  Definimos a las expresiones booleanas como variables booleanas ($a, b, c, \ldots$ a la que podremos asignar un valor en concreto dentro del conjunto $\mathbb{B}$), valores constantes (0 y 1) o la combinación de los anteriores con un operador, y serán dichas expresiones con las que representaremos algebraicamente a las funciones booleanas. Según la combinación de valores dados por las expresiones mínimas (las variables) obtenemos un valor equivalente a la función, pero cabe destacar que una misma función booleana tiene varias representaciones algebraicas, salvo que ducha función sea una expresión mínima.

  Debemos destacar dos componentes importantes en las representaciones algebraicas:

  · Maxitérmino o término suma canónico (M_i): término suma en el que intervienen todas las variables disponibles, de forma directa o negada/inversa disponibles en la función. $f(a, b, c) = \overline{a}b + \underline{a + \overline{b} + c}$

  · Minitérmino o término producto canónico(m_i): término producto en el que intervienen todas las variables disponibles, de forma directa o negada/inversa disponibles en la función. $f(a, b, c) = \overline{a} + b + \underline{a\overline{b}c}$

- Representación numérica:

  Para entender como se hace la representación numérica hay que entender que son las funciones estándar o canónica, las funciones incompletas, la conversión a binario para términos productos y términos suma y el funcionamiento del sumatorio y productorio no usuales, comencemos:

  · Función estándar o canónica: funciones formadas exclusivamente por términos canónicos (suma o producto). Podemos definir una función en la que interviene una función canónica (considerando que se mantienen el numero de variables en ambas).

  · Función incompleta: funciones cuyo resultado para las combinaciones dadas es indeterminada, es decir, una incógnita.

  · Conversión a binario: podemos interpretar los términos productos y suma como un numero binario y este a su vez como un numero decimal, veamos como:

a	b	c	d	Binario	Decimal	Equivalencia
0	0	0	0	0000_2	0_{10}	$\overline{a+b+c+d} = \overline{a}\overline{b}\overline{c}\overline{d} = 0$
0	0	0	1	0001_2	1_{10}	$\overline{a+b+c+\overline{d}} = \overline{a}\overline{b}\overline{c}d = 0$
0	0	1	0	0010_2	2_{10}	$\overline{a+b+\overline{c}+d} = \overline{a}\overline{b}c\overline{d} = 0$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	0	1110_2	14_{10}	$\overline{a}+\overline{b}+\overline{c}+d = \overline{abc\overline{d}} = 0$
1	1	1	1	1111_2	15_{10}	$\overline{a}+\overline{b}+\overline{c}+\overline{d} = \overline{abcd} = 0$

La columna de equivalencias esta partida por una separación de puntos para dar a entender no solo la simplificacion de las numerosas interpretaciones disponibles, también el observar que para leer las variables las consideramos siempre con valor 1 y la igualdad la obtenemos aplicando el principio de dualidad. Por ese motivo tenemos que $\overline{a+b+c+d} = \overline{a}\overline{b}\overline{c}\overline{d} \Leftrightarrow 0000_2$, considerando que las variables valen 1 en todo momento.

· Funcionamiento del sumatorio y productorio no usuales: ya que estamos en el cuerpo $(\mathbb{B}, \oplus, \cdot)$ con $\mathbb{B} = \{0, 1\}$, es normal que el sumatorio y prodcutorio no funcionen de la misma forma que en la álgebra usual. Las explicaciones pueden variar y la forma de como representarlas también (en algunos casos se usan los maxitérminos en el productorio (que correspondería al producto de términos suma canónicos o POS) y minitérminos en el sumatorio (que correspondería a la suma de términos producto canónicos o SOP) pero veremos lo más básico.

Para simplificar el proceso de explicación, definiremos $\Xi_i := \{\sum_i, \prod_i\}$, es decir, con Ξ_i nos referiremos a ambos casos pero sabiendo que el resultado del sumatorio dará 1 y el del productorio dará 0. La expresión básica es $\Xi_i(x_1, x_2, \ldots, x_n)$ donde i sera el total de variables que intervienen o un 0 o el vacío $\varnothing$ (para funciones incompletas, se explicara que significa a continuación) y los valores $(x_1, x_2, \ldots, x_n)$ son las representaciones en decimal de los términos producto o suma canónicos en los que se cumple que da el resultado correspondiente (términos suma $\Leftrightarrow$ productorio con resultado igual a 0 y términos producto $\Leftrightarrow$ sumatorio con resultado igual a 1). Para las funciones incompletas, tanto el productorio como sumatorio, el resultado es una incógnita. Veamos ahora como se representarían y, un ejemplo rápido de resolver una función incompleta:

$\sum_3 m(1,2,3) = m\overline{a}\overline{b}c + m\overline{a}b\overline{c} + m\overline{a}bc = 1$ con m=1

$\prod_3 M(0,4) = M(a+b+c)M(\overline{a}+b+c) = 0$ con M=0

$\sum_\varnothing x_i(5,6) = x_1\overline{a}\overline{b}c + x_2ab\overline{c} = x$ con $x = x_1 + x_2$ y $x_i \in \{0,1\}$

$\prod_\varnothing x_i(5,6) = x_1(\overline{a}+b+\overline{c})x_2(\overline{a}+\overline{b}+c) = x$ con $x = x_1 + x_2$ y $x_i \in \{0,1\}$

en los casos donde coexistan funciones incompletas con estándares, habrá que despejar las x_i teniendo como premisas las funciones estándares.

Simplificación de funciones booleanas

1. Simplificación algebraica:

   Consiste en aplicar los axiomas y teoremas de forma correcta para obtener una expresión booleana con el menor número de términos posible, cabe destacar que puede que no sea única con lo cual, lo más cómodo, es comprobar su veracidad haciendo uso de tablas de verdad o, la mejor opción, con las tablas o mapas de Karnaugh.

2. Tablas de Karnaugh:

   Las tablas de Karnaugh nos permiten simplificar las expresiones booleanas estudiando la distribución sistemática de los valores sobre ésta. Están constituidas por $2^n + 1$ filas y $2^m + 1$ columnas, donde $n + m =$ total de variables. Por ahora aplicaremos estas tablas a expresiones con 3 ($n = 1$ y $m = 2$), 4 ($n = 2$ y $m = 2$) y 5 ($n = 2$ y $m = 3$) variables.

La primera fila y columna están reservadas a las posibles combinaciones entre las variables ordenados según el Código Gray (dos números consecutivos difieren en una sola variable), pero la celda asignada a cada posible combinación se le asigna un decimal según el valor en binario natural (mirar imagen de ejemplo). Dicha representación decimal nos sera de ayuda para asignar los valores de las representaciones numéricas donde corresponda.

Veamos la tabla para 5 variables con los representantes numéricos incluidos para poner un ejemplo:

ab \ cde	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

Según los datos iniciales que tengamos (una tabla de verdad, una representación numérica en POS o en SOP o una representación algebraica), introducimos 1 o 0 donde corresponda. Si se nos da una tabla, bastaría con introducir el resultado respecto a los valores de la variable con las que se obtiene.

Con las SOP $\sum_i(x_1, x_2, \ldots, x_m) = x_1 + x_2 + \cdots + x_m = 1$, las celdas con valor numérico $x_1, x_2, \ldots, x_m$ serán donde se cumple la SOP y, por lo tanto, hay situado el valor 1 (si i =total de variables) o una incógnita (si $i = \varnothing$ o $i = 0$, siendo una función incompleta). Cuando se da una función incompleta, la función cambiara para despejar los posible valores de las incógnitas pero respetando las premisas y rellenando las celdas restantes con 0.

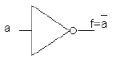
Por otro lado, las POS $\prod_i(x_1, x_2, \ldots, x_m) = x_1 \cdot x_2 \cdot \ldots \cdot x_m = 0$ podríamos considerarlo el enunciado dual de SOP, es decir, si i =total de variables las celdas con valor numérico $x_1, x_2, \ldots, x_m$ serán donde introduciremos un 0, si es $i = \varnothing$ o $i = 0$ pasaría igual. Se despejará la función incompleta, si hay, y se rellena de 1 las celdas restantes.

A la hora de introducir una función algebraica, lo que realmente haremos será introducir una función para que la simplifique. La tabla agrupara elementos del mismo valor en grupos, respetando una simetría, de 2^n con $n \in \mathbb{N} \cup \{0\}$. Formaremos términos producto si los grupos son respecto al valor 1 y suma si el valor es 0.

2.2.2. Sistemas digitales

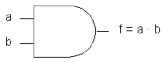
Puertas lógicas digitales e implementación de funciones booleanas Veamos las puertas básicas junto con la expresión booleana asociada a éstas:

- Operació negació: porta NOT, inversor o negador



a	f
0	1
1	0

- Operació producte: porta AND

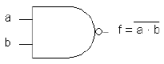


a	b	f
0	0	0
0	1	0
1	0	0
1	1	1

- Operació suma: porta OR

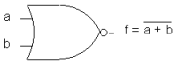
a	b	f
0	0	0
0	1	1
1	0	1
1	1	1

- Porta NAND



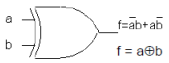
a	b	f
0	0	1
0	1	1
1	0	1
1	1	0

- Porta NOR



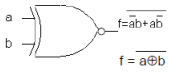
a	b	f
0	0	1
0	1	0
1	0	0
1	1	0

- Porta OR exclusiva o XOR



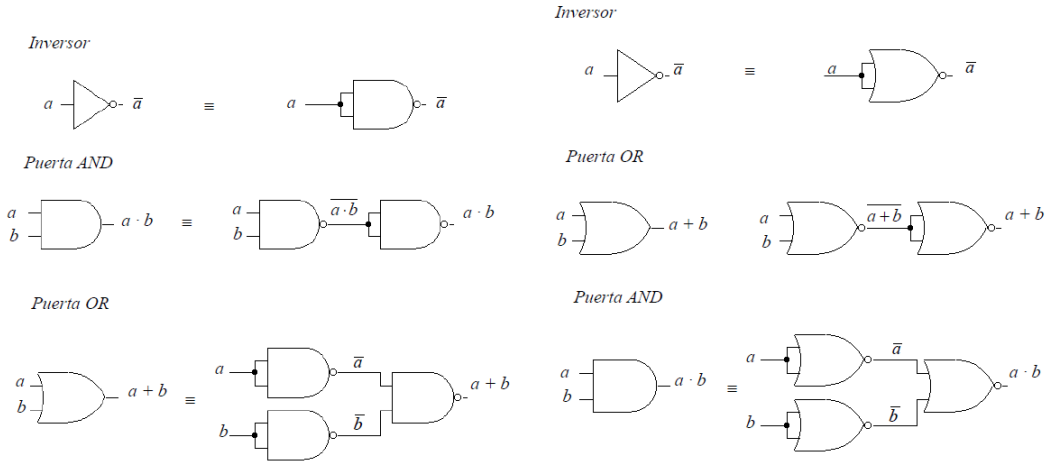
a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

- Porta NOR exclusiva o XNOR



a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

Con éstas definidas podemos observar como definir una misma puerta haciendo uso de otras (la puerta NAND y NOR están representadas a continuación):



Llamaremos conjunto de puertas completo al conjunto compuesto por un tipo de puerta mínimo que nos permite implementar cualquier función booleana en un circuito digital.

2.3. Circuitos combinacionales

Los sistemas digitales se calsifican en dos grandes grupos, diferenciados por como interacciona el tiempo sobre los valores de las variables de salida. En este tema nos centraremos en el circuito combinacional y en el siguiente veremos los sistemas secuenciales.

Lllamaremos circuito o sistema combinacional al conjunto de dispositivos lógicos en el que las salidas dependen exclusivamente del valor actual de las entradas. Podemos asignarle una función lógica y se puede formar un sistema combinación compuesto por otros, llamandose sistema combinacional integrado. Veamos algunos tipos:

2.3.1. Codificadores

Los codificadores pueden venir con 3 elementos adicionales ademas de las entradas y salidas, siendo estos elementos el Enable Input(EI), Enable Output (EO) y el Group Selection (GS), luego veremos su funciones pero entendamos el funcionamiento basico del codificador:

- Llamados codificadores m:n con m entradas y n salidas, clasificados en con y sin prioridad (que depende POR LO QUE PARECE de cuando varias entradas activan una misma salida, en este caso tomara la de mayor peso).
- Se tiene que cumplir que $m \leq 2^n$, si se da que $m = 2^n$, sera un codificador completo, si no se cumple sera incompleto.
- El circuito se puede construir haciendo uso de puertas OR cuyas entradas son los componentes con los que se activaria la salida.
- EI vale 1 de forma predeterminada. Si valiese 0, la salida, aunque las entradas fueran el codigo correcto, estaria desactivada. Actua como un interruptor ON/OFF del circuito.
- EO y GS muestran si hay o no una combinacion correcta que activa la salida. EO sera 1 y GS 0 si ningun codigo es correcto, y viceversa. La unica ocasion donde ambos valdran o es cuando EI es 0.
- Deben entenderse las entradas y salidas como componentes de un numero binario, es decir, el codificador leera un numero de m bits y si el las secuencia correcta te devolvera un numero de n bits correspondiente a dicho codigo.
- La prioridad, recalcando la diferencia otra vez, sera cuando un conjunto de entradas con valores similares o que se pueden agrupar (1XXX por ejemplo) denota al mismo codigo.

2.3.2. Decodificadores

Un decodificador es lo opuesto al codificador, dado un valor en binario en particular habra una salida dedicada a dicho valor. Existe un unico elemento adicional, un EI que actua como con el codificador.

- Dicho de otra manera, los decodificador actuan como una criba en la que para cada combinacion de las entradas, existe una unica salida (con el codificador varias entradas podian hacer referencia a la misma salida debido a que hay mas entradas que salidas).
- Llamados decodificadores m:n, se debe de cumplir que $m \geq 2^n$, y recordando que EI es un habilitador o interrumptor del circuito.
- Se puede construir un circuito decodificador usando puertas AND donde las entradas seras la secuencia correcta(p.e. $a\bar{b}$ correspondiente a S_2 con un habilitador en el circuito, las entradas de la pueta AND para esta salida serian a, (NOT)b y EI).
- Al igual que con el codificador, tratamos con los componentes de numeros binarios de distintos bits, para resolverlo tener claro cuales son las entradas y que combinacion activa que salida. La forma mas sencilla para implementarlo seria para una funcion booleana en SOP, donde los valores representado numericamente serian las posiciones o el valor correspondiente donde se deben poner las salidas.

Un ejemplo sencillo de 2 bits donde la salida corresponde directamente al valor por orden seria:

a	b	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

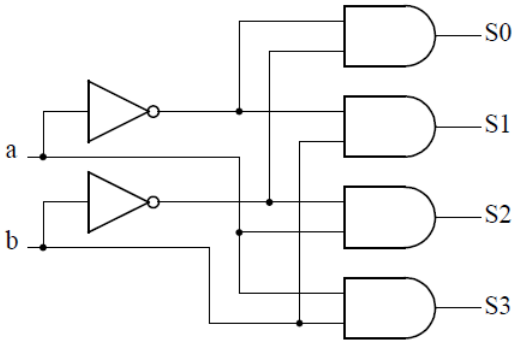
$S_0 = \bar{a}\bar{b}$

$S_1 = \bar{a}b$

$S_2 = a\bar{b}$

$S_3 = ab$

Esquema:



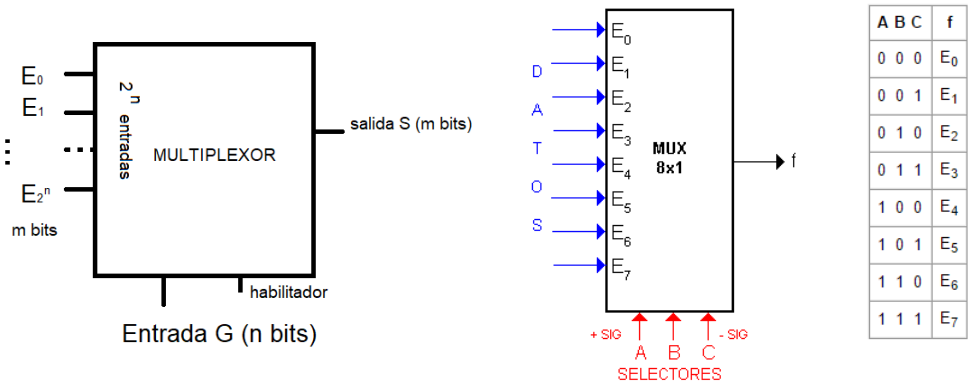
2.3.3. Multiplexores

El multiplexor es un circuito que permite simplificar tablas de verdad y viene caracterizado por 2^n entradas, n señales de controles o entradas de seleccion y una unica salida, ademas un habilitador EI, pero podemos entender a todo el circuito como un problema de n+1 entradas para realizar la resolucion.

- Llamados multiplexor 2^n:1, la salida tendra el valor de la entrada si se da que la señal de control para dicha posicion esta activa. La tabla a continuacion es un ejemplo de como se simplificaria un tabla de verdad de 3 valores (abc) dejandola en funcion de una de ellas.

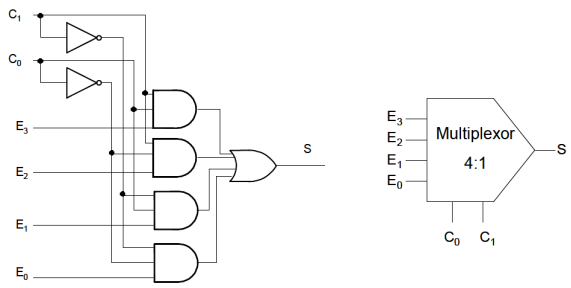
Señales de control		f(c)		f simplificada
a	b	c		f
		1	0	
0	0	0	0	0
0	1	1	0	c
1	0	0	1	$\overline{c}$
1	1	1	1	1

- Las entradas del multiplexor serian los valores de f en las posciones correspondientes a las señales de control (0 o menor peso arriba, 2^n o de mayor peso abajo). El dispositivo se veria de la siguiente forma:



A B C	f
0 0 0	E_0
0 0 1	E_1
0 1 0	E_2
0 1 1	E_3
1 0 0	E_4
1 0 1	E_5
1 1 0	E_6
1 1 1	E_7

- Se puede construir como un conjunto de codificador cuyas salidas estan conectadas a una puerta OR y las entradas de estas serian las señales de control y los habilitadores serian la entrada del multiplexor. En esta imagen se puede ver facilmente:



- Sera util para simplificar las tablas de verdad de circuitos con muchas entradas de bits. Recordar el uso del habilitador, puedes aprovechar la salida de un multiplexor para usarlo como habilitador de otro.

2.3.4. Desmultiplexores

El desmultiplexor es el opuesto al multiplexor, dispone de una unica entrada y n señales de controles para seleccion cual de las 2^n salidas se activara. Practicamente la idea del desmultiplexor o distribuidor es un decodificador donde el habilitador que corresponderia a la entrada del desmultiplexor y las entradas de por si serian las señales de control. Se podria plantear como un multiplexor para desarrollarlo. No creo que se pregunte pero es practicamente un decodifcador.

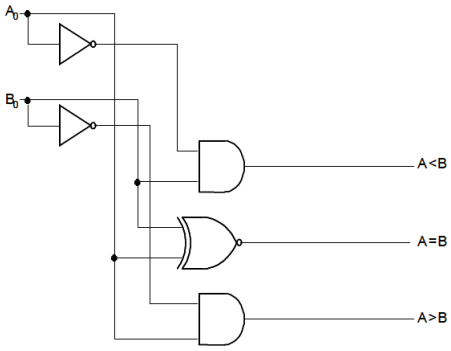
2.3.5. Circuitos comparadores

Un comparador es un circuito en el que se introducen dos numeros binarios de n bits ambos e ira probando si tienen igual valor los bits en la misma posicion o si uno es mayor que otro. Para probar la realacion entre los valores en la siguiente posicion bastaria con encadenarlos con el estado anterior que permita seguir avanzando (cuando valgan los mismo).

Seria complicado de dibujar pero bastaria con hacer tres ramas donde la central fuera la cadena que comprueba la igualdad de los bits en una posicion y los extremos de cada subrama la desigualdad entre estos. Puede ser abstracto de primeras pero con el ejemplo para un bit creo que sera suficiente:

A_0	B_0	$S_{A<B}$	$S_{A=B}$	$S_{A>B}$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

$$S_{A=B}=\overline{A_0 \oplus B_0}$$
$$S_{A<B}=\overline{A}_0 \cdot B_0$$
$$S_{A>B}=A_0 \cdot \overline{B}_0$$

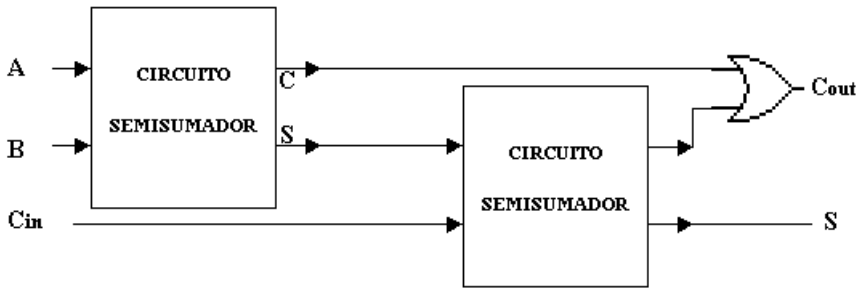


2.3.6. Circuitos aritmeticos

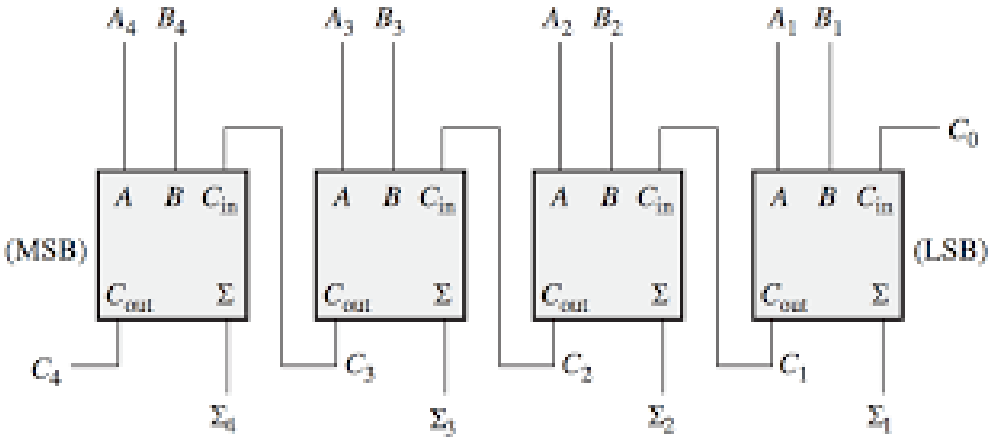
Sumadores El sumador completo(considera acarreo anterior) y el semisumador(no considera directamente un acarreo anterior, solo suma dos elementos que se le de) son consecuencia directa del implementar las puertas XOR y AND (la puerta XOR equivale a $\overline{a}b + a\overline{b} = a \oplus b$ y la puerta AND a $a \cdot b$ con $a, b \in \mathcal{B}, \mathcal{B} = [0, 1]$). Con la puerta XOR realizamos la suma y con la puerta AND obtenemos el posible acarreo.

Son estas operaciones las que realizan ambos circuitos pero, en especial, el sumador completo que podemos entenderlo como dos semisumadores.

$$S = A \oplus B \oplus C_{in} = (\overline{A}B + A\overline{B}) \oplus C_{in} = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$



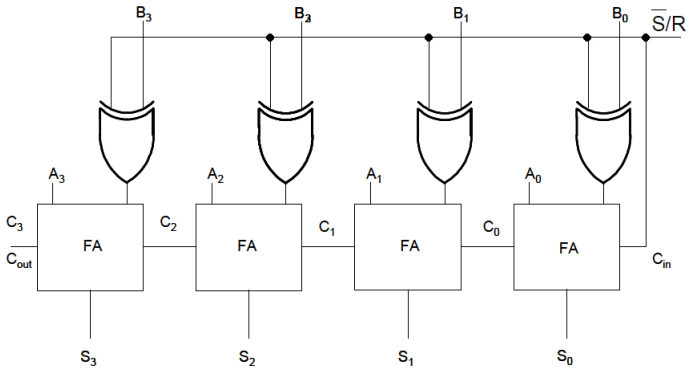
En la imagen anterior observamos como la suma $A+B$ acompañada por un posible acarreo (C_{in})de un etapa anterior aporta como resultado final un valor S y otro C_{out}, pero debemos comprender que trabaja con 1 bit, si se desease con más lo que realmente haria sería una sucesión de sumadores completos. Veamos un ejemplo con 4 bits.



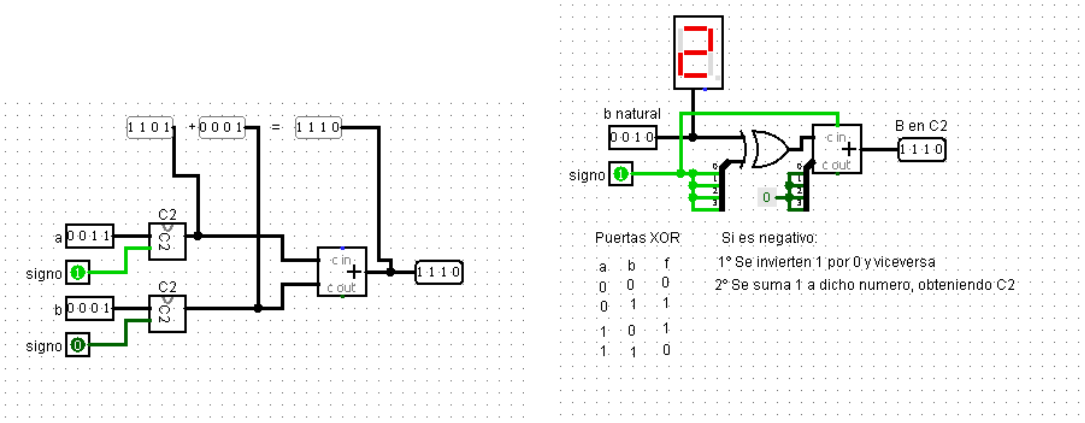
En este ejemplo vemos las suma de $A+B$ acompañado por un posible acarreo C_0, siendo estas nuestras entradas, cuya salida nos dará el acarreo de la ultima etapa (C_4) y la suma, $\sum$.

Restadores Un restador completo y un semirestador, cuya diferencia es la misma que la de los sumadores, siguen los mismos principios que el sumador, lo unico que cambia es el acarreo que en lugar de ser $C_{out+} = ab$ como en el sumador (situacion en la que la suma genera escedente), sera $C_{out-} = \overline{a}b$ (situacion en la que el positivo es menor que el negativo en valores de la misma posicion y se aplica la resta al valor de la siguiente posicion.

Tambien, podemos considerar las propiedas de los complemento a 1 y a 2 para entender la resta como una suma. Recordad que un numero en C1 es el opuesto del binario natural negativo y el C2 seria el complemento a 1 +1. Con el sumador se veria como las entradas del numero negativo con puertas NOT y un acarreo con valor a 1 si fuese con C2. Es mas, en el siguiente ejemplo se observa como se utilizan puertas XOR y un activador S/R para indicar que funcion se realiza (usado en C2, solo considera el segundo numero posible de ser negativo):



Este seria un ejemplo realizado por logisim de un sumador que considera si un numero es negativo o positivo y los suma.

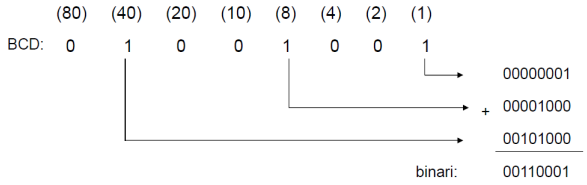


2.3.7. Más sistemas combinacionles (dado en valenciano)

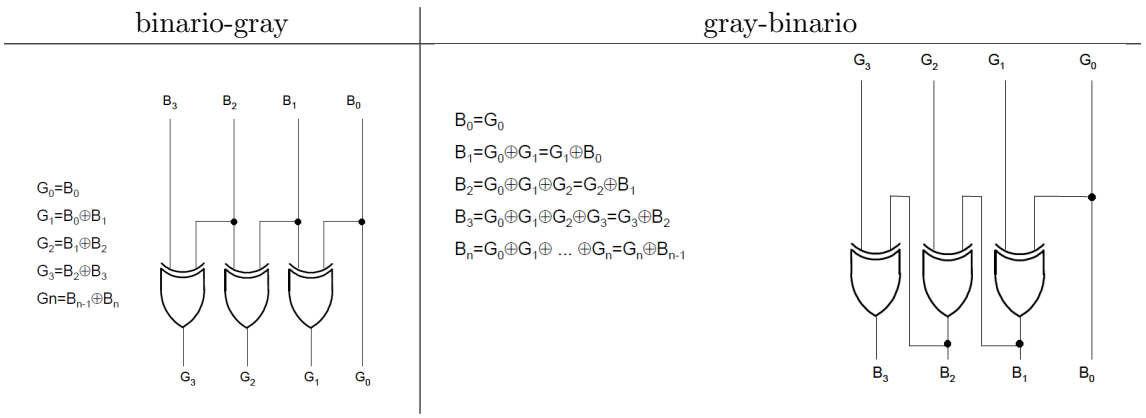
Estos conceptos se explicara por encima ya que solo los ha introducido una profesora sustituta y no esta en los apuntes que se nos dio al principio, son faciles de sacar con tablas de verdad y los circuitos anteriormente vistos ademas de ciertas propiedades, veamoslos:

Conversores de código Dsipositivo que permite pasar de un tipo de binario a otro (recordemos que esta el binario natural, el BCD, el codigo Gray.. vistos en el 1.4).

- De natural a BCD solo consistira en tener en cuenta que el BCD son grupos de 4 bits cuyo valor numerico no supera el 9. Si realizamos un comprobador de valor superior a 9 y que este active dos cosas, un 0001 que sera el bit de mayor peso y un sumador de 6 (ya que es el equivalente de 9 en C1).
- De BCD a natural creo yo que con este ejemplo bastaria para entenderlo:



- Binario natural-Gray y Gray-Binario natural



Generadores/detectores de paridad PAAAAAAAAAAAAAAASO

Hoja de caracteristcas (data sheet) Es teoria, un Data sheet es un documente que explica detealladamente el funcionamiento y caracteristicas de un dispositivo.

2.4. Sistemas secuenciales

Los sistemas digitales cuyas salidas no dependan unicamente de las entradas en ese instante, tambien de los valores en instantes anteriores reciben el nombre de sistemas secuenciales. Se estudian dos modelos, el de Mealy y el de Moore, pero antes debemos conocer algunos dispositivos de almacenamiento de memoria:

2.4.1. Biestables

Circuitos lógicos en los que se distinguen dos estados estables los cuales pueden mantenerse indefinidamente aun habiendo desaparecido la señal que activa el estado. Los podemos clasificar por funcionalidad o por la forma de su activación (asíncronos, es decir, sin señal de reloj o sincrono, depende de una señal de reloj y hay por nivel o por flanco).

Diremos que es de nivel alto cuando se encuentra con valor 1, nivel bajo cuando tiene valor 0, flanco de subida en el momento exacto que se desplaza de 0 a 1 y flanco de bajada en el momento exacto que se desplaza de 1 a 0. Esta terminología también se puede aplicar a otros elementos pero sera mas frecuente con las señales de reloj. Para hacer por nivel bastaria con recibir una señal constante (las entradas dependen del valor del reloj) pero por flanco se tendria que realizar un conjunto de puertas logicas en la que la señal de reloj de una valor en una pequeña fraccion de tiempo (se vera un ejemplo mas adelante).

Por funcionalidad distinguimos los biestables RS, JK, D y T:

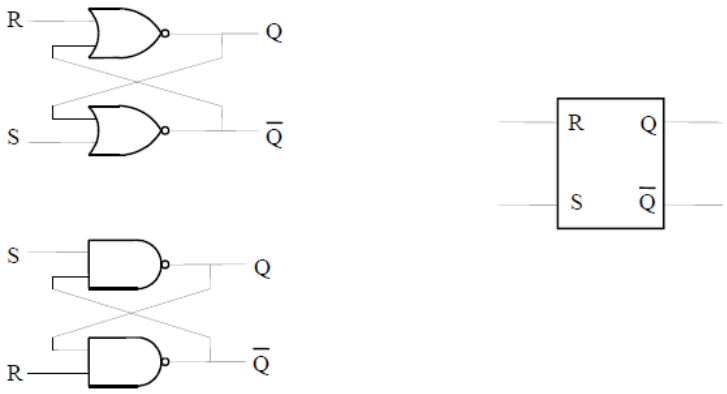
Biestable RS (reset y set) Sin importar si es sincrono o asíncrono o sincrono con entradas asíncronas, este circuito actúa como un Reset-Set, es decir, la entrada R resetea la salida Q y la entrada S únicamente la puede activar. Que estén ambas encendidas no esta permitida y si están las dos apagadas la salida es la que estaba en el estado anterior. Se ha nombrado el caso de que sea sincrono con estadas asíncronas, éste es cuando se usan las entradas P y C (Preset y Clear), siendo la función del Preset la de activar la salida Q y la del Clear es despejar la salida dándole el valor de 0. La función de estos elementos es darle un valor a la salida directamente sin tener que dar una pulsación de reloj.

R	S	Q(t+1)	Acción
0	0	Q(t)	No cambia
0	1	1	Se activa
1	0	0	Se resetea
1	1	-	No definida

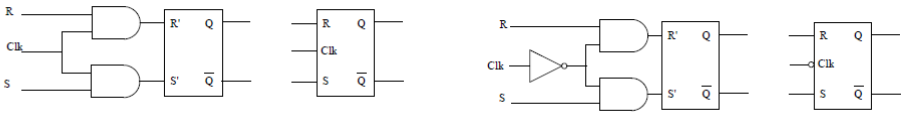
Prst	Clr	R	S	Q(t)	Acción
0	0	0	0	Q(t-1)	No cambia
0	0	0	1	1	Set
0	0	1	0	0	Reset
0	0	1	1	-	No definido
1	0	x	x	1	Activa la salida
0	1	x	x	0	Despeja la salida
1	1	x	x	0	No definida/Clear tiene preferencia

Se puede escribir de las siguiente formas: (meter fotos del tema)

· Asincrono con puertas NOR y con puertas NAND

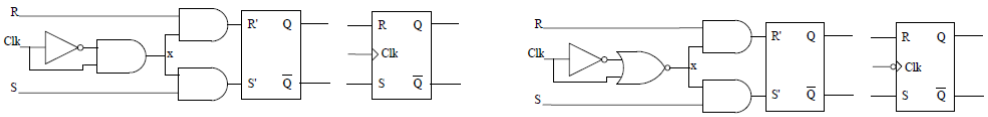


· Sincrono por nivel con puerta AND y usando asincrono.



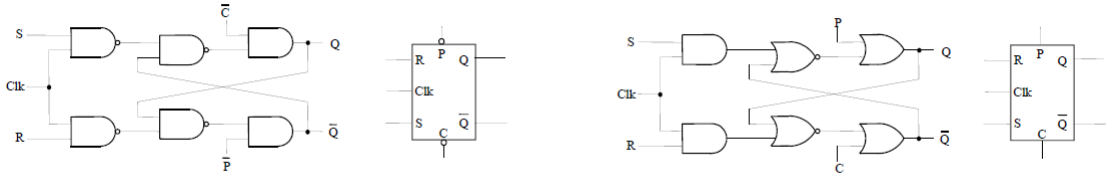
Nivel alto Nivel bajo

· Sincrono por flanco con puerta AND y usando asincrono.



Flanco de subida Flanco de bajada

· Con entradas asincronas (por nivel bajo y alto, respectivamente)



Se pueden darse mas formas pero estos son ejemplos .

NOTA: recordad que si usais todo puertas NAND con los sincronos, la salida de la puerta NAND sera R y S negadas para una señal de reloj de nivel alto.

Biestable JK Este biestable con entradas J y K actúa de tal forma que si únicamente se activa K, Q se resetea, si solo se activa J, Q se activa, si J y K no se activan Q se mantiene en el estado anterior y si ambas entradas se activan entonces la salida sera la opuesta al estado anterior (basculación), tambien consideraremos entradas de Preset y Clear.

Aunque aparentemente utiliza señales de reloj, solo las utilizas para considerar los estados... mas adelante se vera como, construido con un biestable RS, la señal de reloj da igual si es de nivel alto o bajo que no afectara realmente a las entradas.

Debemos considerar dos tablas de verdad que hacer, el funcionamiento del biestable JK y la interacción de las entradas Preset y Clear:

J	K	Q(t+1)	Acción
0	0	Q(t)	No cambia
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q(t)}$	Basculación

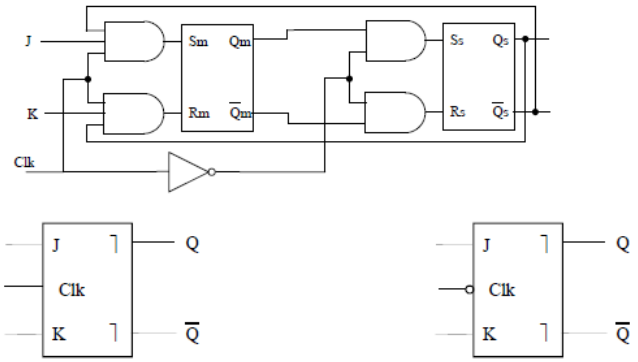
Funcionamiento del biestable JK sin uso alguno de las entradas prst y clr

Prst	Clr	Q(t)
0	0	Q(t)
1	0	1
0	1	0
1	1	0

En este caso, Q(t) se sobrescribe ya que no depende de la señal del reloj y si $Prst = Clr = 1$ entonces parece tener preferencia Clr

Prst	Clr	J	K	Q(t)	Acción
0	0	0	0	Q(t-1)	No cambia
0	0	0	1	0	Reset
0	0	1	0	1	Set
0	0	1	1	$\overline{Q(t-1)}$	Basculación
1	0	x	x	1	Activa la salida
0	1	x	x	0	Despeja la salida
1	1	x	x	0	No definida/Clear tiene preferencia

En esta imagen observamos como construimos un biestable JK a partir de uno RS, con la configuracion llamada Master-Slave (Master hace las cosas y Slave las copia).



Biestable D (Data) Con una sola entrada D (Dato) ademas de la señal de reloj, la acción de este biestable es almacenar brevemente la información aportada por la entrada D. Prácticamente repite la información dada.

Se podría construir con un biestable JK siendo D la entrada J y $\overline{D}$ la entrada K.

J	K	Q(t+1)	Acción
0	0	Q(t)	No cambia
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q(t)}$	Basculación

D	$\overline{D}$	Q(t+1)	Acción
0	0	-	No se da
0	1	0	Reset
1	0	1	Set
1	1	-	No se da

D	Q(t+1)	Acción
0	0	Elimina
1	1	Guarda

Biestable T (Toggle) Usando una sola entrada, T (Toggle), junto con las señales de reloj, la función de este biestable consiste en alternar la salida si la entrada T se activa, recordando el estado actual.

Usando un biestable JK, sin consideramos las entradas J y K como la entrada T, podemos construir este elemento.

J	K	Q(t+1)	Acción
0	0	Q(t)	No cambia
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q(t)}$	Basculación

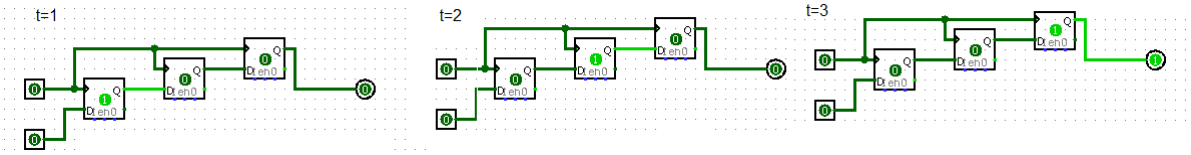
T	T	Q(t+1)	Acción
0	0	Q(t)	No cambia
0	1	-	No se da
1	0	-	No se da
1	1	$\overline{Q(t)}$	Basculación

T	Q(t+1)	Acción
0	Q(t)	No cambia
1	$\overline{Q(t)}$	Cambia

2.4.2. Registros y contadores

Existen otros elementos como los registros (circuitos secuenciales con funciones de almacenamiento o desplazamiento de información) y contadores (sistema secuencial que memoriza el numero de pulsaciones aplicadas a una entrada de reloj), veamoslos por encima:

- Registro: Almacenan o desplazan datos por unidad de tiempo, se pueden construit con biestables D en serie, teniendo tantos como unidades de tiempo que quieres que se mantenga el datos ya que todos ellos estaran conectados a la misma señal. En este ejemplo por logisim vemos como se almacena un dato durante tres señales de reloj



- Contadores: Son circuitos que cuentan el numero de pulsaciones de una señal de reloj, se pueden clasificar en asincronos (muy faciles) o sincronos (no veo ningun ejemplo so sudo). El asincrono utiliza biestable JK con ambas entradas con valor 1 constante conectadas en cascada, donde la señal de reloj sera la salida del biestable anterior salvo el primero, que sera la propia señal de reloj. Con n biestable podemos contar hasta 2^n ya que la salida de cada biestable seran los bits de menor a mayor peso que genere el numero de pulsaciones.

2.4.3. Diseño de sistemas secuenciales, modelos de Moore y de Mealy

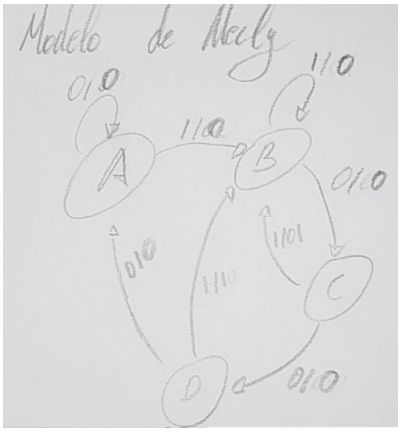
Con esto visto, podemos ver los distintos modelos de sistemas secuenciales delos que disponemos, el modelo de Mealy y el de Moore. La principal diferencia es de que depende la salida del circuito, si esta depende del estado anterior y de las entradas hablamos del modelo de Mealy, si solo depende del estado anterior entonces hablamos de Moore. Veamos como obtener la resolucion de un problema aplicando ambos modelos, ya que hay un modelo de Moore equivalente a uno de Mealy y viceversa, y siguiendo las siguientes pautas:

1. Obtener un diagrama de estado y una tabla de estados partiendo de las especificaciones del problema, es decir, distinguimos que posibles estados se crearan y que provoca que se cambie de uno a otro.
2. Codificamos los estados para obtener una tabla de transición de estados y de salida.
3. Seleccionar el biestable adecuado y realizar una tabla de excitactión del biestable a partir de la tabla de transición de estado.
4. Obtener las ecuaciones de entrada de los biestables y las ecuaciones de salida del circuito a partir de estas tablas, ésto se puede realizar usando Karnaugh.
5. Dibujar el circuito usando las entradas, salidas y biestable que hayas obtenido.

Veamos el siguiente ejercicio, donde se nos pide que se encienda una bombilla para cuando entra una secuencia de datos en particular (1001) permitiendo que exista solapamiento, es decir, aprovecha el ultimo 1 de la secuencia para iniciar la siguiente.

Veamoslo primero por Mealy y por pasos indicados anteriormente:

1. Obtener un diagrama de estado y una tabla de estados partiendo de las especificaciones del problema. Con el modelo de Mealy, habra que considerar que lo que conecta los estados no es unicamente la entrada, sino la salida.



Z_0	Accion
0	Bombilla apagada
1	Bombilla encendida

Estado	Explicacion
A	Estado inicial o cuando se cancela la secuencia con x:0
B	Se inicia la secuencia, X:1 reinicia y X:0 continua
C	Primer cero, X:1 reinicia y X:0 continua
D	Segundo cero, X:1 fin secuencia y X:0 cancela

Estado anterior	Estado nuevo en funcion de x	
	x=0	x=1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

con X,Z_1Z_0, X estado nuevo y Z_i las salidas segun la entrada y X.

2. Codificamos los estados para obtener una tabla de transición de estados y de salida. Con el modelo de Mealy, estas tablas estan unidas.

Estado anterior	Estado nuevo en funcion de x	
	x=0	x=1
A=00	00,0	01,0
B=01	11,0	01,0
C=11	10,0	01,0
D=10	00,0	01,1

3. Seleccionar el biestable adecuado y realizar una tabla de excitactión del biestable a partir de la tabla de transición de estado. Lo mas comun es usar como elemento de memoria el biestable JK, ya que puede actuar como todos los demas.

   Se usaran tantos biestables como numero de bits hayan para distinguir los estados codificados, en este caso 2 bits. En este caso, es mejor hacer la tabla de excitacion del biestable considerando tambien el estado anterior tal que

J	K	q	Q	Acción
0	0	0	0	No cambia
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Basculación
1	1	1	1	

$\Leftrightarrow$

J	K	q	Q	Acción
0	X	0	0	No cambia o Reset, depende de J
1	X	0	1	Set o basculacion, depende de J
X	0	1	1	No cambia o Set, depende de K
X	1	1	0	Reset o basculacion, depende de K

   Aunque parezca que dependa de la incognita, realmente depende del valor constante, ya que da igual el valor que tenga la incognita, lo que importa es el valor de la entrada que sea constante si queremos obtener Q partiendo de q. Esto es importante para entender el siguiente paso.

4. Obtener las ecuaciones de entrada de los biestables y las ecuaciones de salida del circuito a partir de estas tablas, ésto se puede realizar usando Karnaugh. Es el paso mas complicado, ya que se deben de considerar que elementos usar de entrada en un circuito combinacional para obtener la salida adecuada.

   Como hemos dicho antes, tendremos 2 biestables JK, con lo cual habra que considerar 4 entradas a realizar: J_0, K_0, J_1 y K_1. Debemos hacer que para cada una de las entradas entre la correcta combinacion de entradas y estados anteriores que de el siguiente estado, veamos como seria para este caso:

q_1q_0\X	0	1
00	0	0
01	1	0
11	X	X
10	X	X

J_1

Nos fijamos en $\overline{q_1}$

q_1q_0\X	0	1
00	X	X
01	X	X
11	0	1
10	1	1

K_1

Nos fijamos en q_1

q_1q_0\X	0	1
00	0	1
01	X	X
11	X	X
10	0	1

J_0

Nos fijamos en $\overline{q_0}$

q_1q_0\X	0	1
00	X	X
01	0	0
11	1	0
10	X	X

K_0

Nos fijamos en q_0

   Nos fijamos en las regiones en las que las entradas JK dara la salida correspondiente a su peso que deseamos, por ejemplo, estando en 01 y entra x=0, pasariamos al estado 11. La salida $Q_0 = 1$ depende de $q_0 = 1$ y x=0 (No hay cambio o se hace Set) y la salida $Q_1 = 1$ depende de $q_1 = 0$ y x=0 (se produce una basculacion o un Set). Mirando la tabla auxiliar del biestable vemos que cuan q=0, nos fijamos en la salida deseada para poner J, y si q=1 nos fijaremos en K.

En ese caso, con $q_0 = 1$ y x=0 y sabiendo por la tabla de transicion que $Q_0 = 1$, nos fijaremos en que valor de K provoca esta funcion, siendo K=0. Ahora, con $q_1 = 0$ y x=0 y sabiendo que $Q_1 = 1$, que valor de J provoca esta funcion, siendo J=1. Repitiendo este proceso obtenemos las tablas de excitacion de los biestables, que despejando por Karnaugh obtendremos las ecuaciones de entrada que son:

$$J_1 = q_0\overline{x} \quad K_1 = x + \overline{q_0 x}$$
$$J_0 = x \quad K_0 = q_1\overline{x}$$

Para obtener las de salida debemos considerar la tabla de transicion de estados al principio (en el caso de modelo de Mealy). Pero podemos si no lo vemos claro hacer una tabla para comprobar las salidas, siendo la de transicion pero omitiendo los estados nuevos y si se prefiere considerar una tabla por cada bit, tal que:

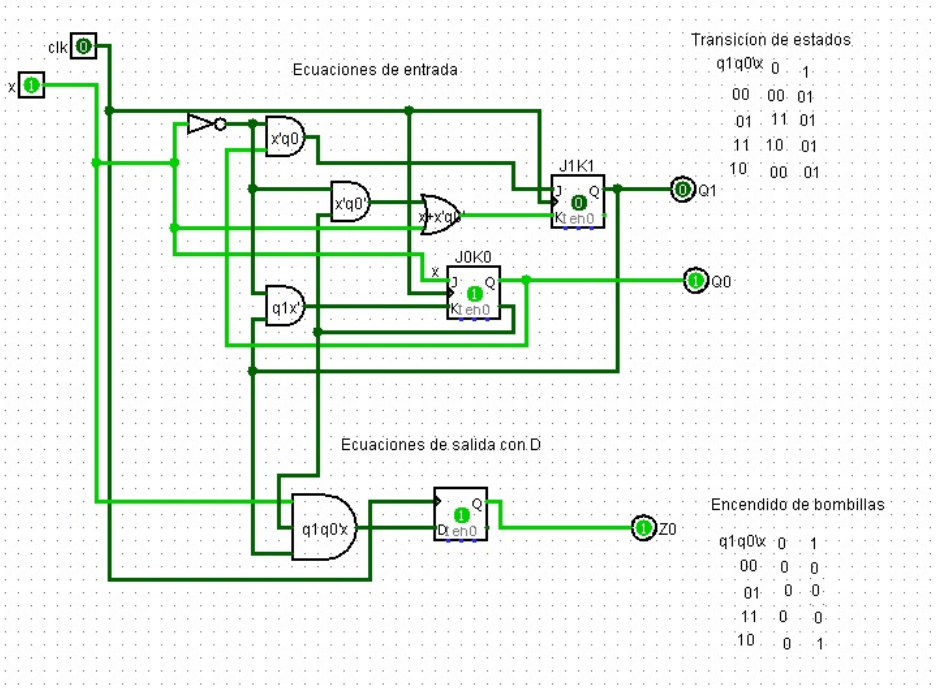
Estado anterior	Salida para Z_0	
	x=0	x=1
A=00	0	0
B=01	0	0
C=11	0	0
D=10	0	1

Y facilmente por Karnaugh despejamos: $Z_0 = q_1\overline{q_0}x$

5. Dibujar el circuito usando las entradas, salidas y biestable que hayas obtenido.

   Facil de obtener en cuanto se tienen las ecuaciones, recordad que sumas son puertas OR y multiplicaciones puertas AND, siendo estas las mas sencillas, puertas XOR de utilidad.

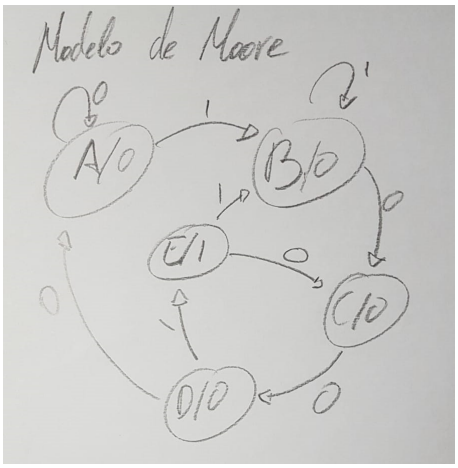
   En ocasiones se pueden dar problemas de sincronizacion debido a que depende de la entrada y esta no va a la par con el pulso, con lo cual es posible que se necesita añadir un biestable D para desplazar la x. Sinceramente, esto cuesta verlo y mas por escrito, con lo cual no creo que os vayan a matar mucho por ello.



La bombilla permanecera encendida cuando, estando en el estado D=10, recibe un x=1, no se de la siguiente señal de reloj. Si no estuviese el biestable D, se encenderia un instante que seria justo el instante en el que recibe las condiciones necesarias.

Veamoslo ahora para el modelo de Moore:

1. Obtener un diagrama de estado y una tabla de estados partiendo de las especificaciones del problema. Con el modelo de Moore, cada estado tiene una salida asociada.



Z_0	Accion
0	Bombilla apagada
1	Bombilla encendida

Estado	Explicacion
A	Estado inicial o cuando se cancela la secuencia con x:0
B	Se inicia la secuencia, X:1 reinicia y X:0 continua
C	Primer cero, X:1 reinicia y X:0 continua
D	Segundo cero, X:1 fin secuencia y X:0 cancela
E	Fin secuencia, X:1 reinicia y X:0 continua

Estado anterior	Salida correspondiente
A	0
B	0
C	0
D	0
E	1

Estado nuevo en funcion de x	
x=0	x=1
A	B
C	B
D	B
A	E
C	B

2. Codificamos los estados para obtener una tabla de transición de estados y de salida. Con el modelo de Moore, seran dos tablas distintas que parten de la tabla anterior.

Estado anterior	Salida correspondiente
A=000	0
B=001	0
C=011	0
D=010	0
E=110	1

	Estado nuevo en funcion de x	
	x=0	x=1
000	000	001
001	011	001
011	010	001
010	000	110
110	011	001

Con Mealy teniamos un estado menos que era justo $2^2 = 4$ estados, los que podiamos clasificar con las combinaciones de 2 bits, pero en este caso, con Moore, tenemos 5 con lo cual tendremos que coger 3 bits, que nos dan hasta $2^3 = 8$ combinaciones.

3. Seleccionar el biestable adecuado y realizar una tabla de excitactión del biestable a partir de la tabla de transición de estado. Lo mas comun es usar como elemento de memoria el biestable JK, ya que puede actuar como todos los demas.

   Como se ha dicho durante la explicacion de Mealy, usaremos tantos biestables como bits hayan para distinguir los estados, es decir, 3 bits y por lo tanto usamos 3 biestables, cada uno para los bits de entrada del mismo peso o que estan en la misma posicion que los de salida.

   Recordemos de nuevo la tabla del biestable JK (que puede actuar como todos) considerando q y constantes:

J	K	q	Q	Acción
0	0	0	0	No cambia
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Basculación
1	1	1	1	

$\Leftrightarrow$

J	K	q	Q	Acción
0	X	0	0	No cambia o Reset, depende de J
1	X	0	1	Set o basculacion, depende de J
X	0	1	1	No cambia o Set, depende de K
X	1	1	0	Reset o basculacion, depende de K

Sera comodo escribir dos tablas para facilitar el siguiente paso:

J	q	Q	Acción
0	0	0	No cambia o Reset
1	0	1	Set o basculacion

K	q	Q	Acción
0	1	1	No cambia o Set
1	1	0	Reset o basculacion

4. Obtener las ecuaciones de entrada de los biestables y las ecuaciones de salida del circuito a partir de estas tablas, ésto se puede realizar usando Karnaugh. Es el paso mas complicado, ya que se deben de considerar que elementos usar de entrada en un circuito combinacional para obtener la salida adecuada.

   Veamos ahora las ecuaciones de entradas para los tres biestables JK que usaremos, teniendo en cuenta que los estados 111, 101 y 100 no estan definidos:

$q_2q_1q_0$\X	0	1
000	0	0
001	0	0
011	0	0
010	0	1
110	X	X
111	X	X
101	X	X
100	X	X

J_2

Nos fijamos en $\overline{q_2}$

$q_2q_1q_0$\X	0	1
000	X	X
001	X	X
011	X	X
010	X	X
110	1	1
111	X	X
101	X	X
100	X	X

K_2

Nos fijamos en q_2

$q_2q_1q_0$\X	0	1
000	0	0
001	1	0
011	X	X
010	X	X
110	X	X
111	X	X
101	X	X
100	X	X

J_1

Nos fijamos en $\overline{q_1}$

$q_2q_1q_0$\X	0	1
000	X	X
001	X	X
011	0	1
010	1	0
110	0	1
111	X	X
101	X	X
100	X	X

K_1

Nos fijamos en q_1

$q_2q_1q_0$\X	0	1
000	0	1
001	X	X
011	X	X
010	0	0
110	1	1
111	X	X
101	X	X
100	X	X

J_0

Nos fijamos en $\overline{q_0}$

$q_2q_1q_0$\X	0	1
000	X	X
001	0	0
011	1	0
010	X	X
110	X	X
111	X	X
101	X	X
100	X	X

K_0

Nos fijamos en q_0

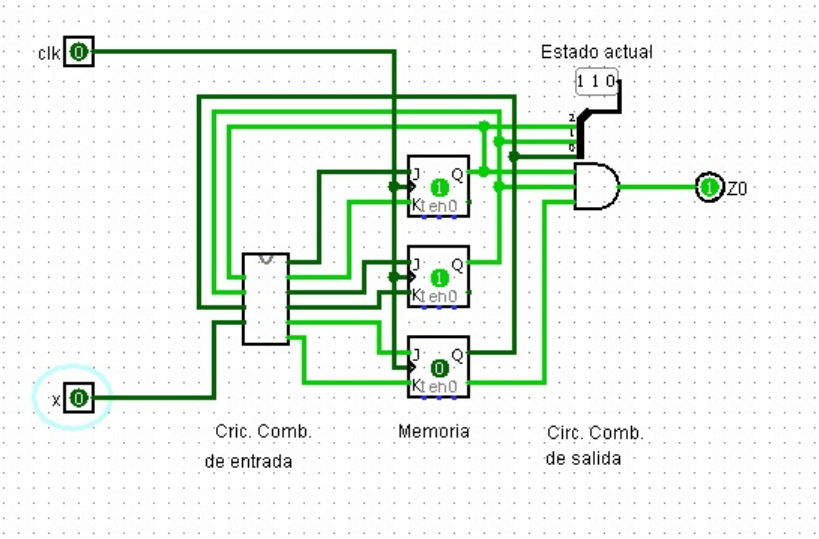
   Veamos las ecuaciones de entrada que se obtienen de realizar Karnaugh(que por cierto, la tabla de Karnaugh no corresponde a las mostradas pero se sacaria facilmente tanto la tabla como las operaciones en estas usando el mismo principio) sobre estas tablas:

$$J_2 = q_1\overline{q_0}x \qquad J_1 = q_0\overline{x} \qquad J_0 = q_2 + \overline{q_1}x$$
$$K_2 = 1 \qquad K_1 = \overline{q_2q_0x} + q_0x + q_2x \qquad K_0 = q_1\overline{x}$$

   Para obtener las salidas nos fijaremos en la tabla anteriormente de salidas para cada estado, pero en este caso en sencillo ya que hay un estado exclusivo para activar la salida, siendo $Z_0 = E = q_2q_1\overline{q_0}$

5. Dibujar el circuito usando las entradas, salidas y biestable que hayas obtenido.

   Al igual que con Mealy, usamos las ecuaciones para definir puertas logicas para cada ecuacion, en este caso al ser mas numeros resumimos el circuito combinacional pero el contenido es el mismo.



   En este caso, no hace un biestable para controlar la salida ya que este viene dado por los estados que se obtiene al dar una señal de reloj.

2.5. Recomendaciones

Pues ya mas no se que poner, si a alguno se le ocurre alguna cosa de ultima hora la anoto:

· Recordad los trucos para pasar de dinario natural a los complementos y SM, la representacion de numero reales era algo mas complicada con el sesgo pero en el tema 1 esta bien explicado.

· Una resta se puede convertir en suma usando complementos.

· Codigo Gray lo usamos para las tablas de karnaugh, BCD es agrupar de 9 en 9(en grupos de 4 bits que no superar 1001).

· Revisar las reglas logicas del tema 2, viene muy bien para simplificar cuando por Karnaugh no lo ves

· Memorizar JK y tener claro que J=D=T y K=$\overline{D}$=T, todos los biestables estan relacionados con el, RS es un componente del JK.

· Las tablas de excitacion estan relacionadas directamente con la posicion del bit de salida y de entrada y su valor, si q=0 trabajas con J, si q=1 con K y segun la salida que quieras obtener para Q, J y K tomaran valores distintos (ej. en desarrollo por Moore).

Si se quieren los archivos de logisim habladme @eduespuch (y seguirme por insta y todo eso)