Teoría

El diseño de Von Neuman, lo que hace es cargar los programas en memoria. Uno de los problemas que tubo fue de cómo traducir la información a ceros y unos.

De esta manera, surge la codificación de las operaciones, a cada operación le corresponde un numero binario.

Creó el código de operaciones y el de direccionamiento.

Otro problema que tubo fue como darle un orden secuencial de las instrucciones, como ejecutar una instrucción después de la otra, Von Neuman lo que hizo fue colocarlas en forma contigua.

Memoria: es una cantidad de palabras, cada una de las cuales esta identificada por una dirección y posee un contenido.

Existen dos ciclos:

- Lectura: dada la dirección del dato y la instrucción leer y al cabo de un tiempo de acceso nos dará el dato.
- Escritura: se da la dirección y el dato a guardar y al cabo de un tiempo de acceso de escritura se habrá guardado el dato.

Bus:

Es un conjunto de cables que transportan información.

Registro

Lugar donde se puede guardar temporalmente un nro. binario (palabra de memoria), los registros no tienen un direccionamiento, sino que son controlados por una unidad de control. Su tiempo de acceso es mucho más rápido.

Registros específicos:

PC, RDM, RBM, RI, SP, IX

<u>Registros generales:</u> R0.....Rn

Funcionamiento de la CPU

- Realiza un ciclo de maquina. Se divide en dos ciclos:

- C. de Búsqueda de instrucción

Ciclos de Maquina

- C. de Ejecución de la instrucción
    - Cómo se realiza la búsqueda?

El PC tiene la dirección de la primera instrucción , su contenido se copia en el RDM y genera un ciclo de lectura mientras que se incrementa en 1 el PC , el dato leído en memoria se guarda en el RBM y luego se copia en el RI . Fin del ciclo de búsqueda(el PC ya está incrementado).

- La unidad de control decodifica el Código de operaciones del RI y sabe que en el segundo campo están las direcciones de las operaciones y donde debemos guardar el resultado de la operación. Luego la UC transfiere los 2 operandos a la ALU y le dice que debe operarlos según el código de operaciones decodificado, una vez realizada la operación, se guarda el resultado en la dirección que figura en el tercer subcampo del RI. Este resultado se guarda generando un ciclo de escritura.

- Al apagarse la maquina el contenido de los registros se pierde. Para que la maquina funcione a la manera de Von Neuman, es necesario que la maquina tenga un circuito eléctrico, que ponga al PC siempre en la posición 0, al encenderse.

Diferencias entre instrucciones y datos:

Si viene del PC es una instrucción, si viene del campo de direcciones del RI, es un dato.

Esto nos permite tomar instrucciones como dato, si en el campo de direcciones del RI se encuentra un direccionamiento a una instrucción en lugar de una dirección a un dato, la CPU va a operar como si se tratara de un dato.

Esto permite construir programas auto modificables(veneficio, si el programa no es compartido entre varios usuarios). Las modificaciones pueden hacerse en asembler.

Interconexión entre maquinas

Unidades Básicas:

- Memoria
- CPU
- E/S

Existen dos tipos de maquinas, maquinas de 1 bus y maquinas de 2 buses.

En las maquinas de 1 bus, cuando se emite una direccion, esta será resibida tanto por la memoria como por e/s. Como hacemos para distinguirla? Divide el espacio de direcciones pero no el de instrucciones.

Si tenemos una maquina de 2 buses, se tendrán 2 espacios de direcciones y se podrá trabajar simultáneamente sin problemas, pero se deberán diferenciar las instrucciones de E/S de las de Memoria.

- Como reducir la longitud de palabras

Comenzamos sacando el campo de direcciones de la próxima microinstruccion y se devuelve al PC=MPC. La idea de que el código de operaciones del RI sea directamente la direccion de comienzo de la próxima instrucción, esto tiene un inconveniente, las direcciones de comienzo de c/modelo no son contiguas, llevarían mas bits de lo necesario y quedarían sin usar algunos, luego se utiliza una tabla de conversión.

Para compactar aun mas la long. de las palabras debe hacerse una codificación.

Por tener un único bus es imposible que ciertas microoperaciones sé deen simultáneamente (los out, las instrucciones de la ALU).

Lo que hacemos es reunir esas microoperaciones que son mutuamente excluyentes y codificarlas.

El mecanismo de codificacion para reducir la long. de palabras, nos obliga a introducir codificadores. Reducimos la long. de palabra, pero el costo que tenemos es tiempo (para codificar).

Multiplexor: lo contrario a codificador.

Existen tres maneras de modificar el MPC

1)− Cuando termina la ejecución del FECH se hace un LOAD que indica que se debe cargar en el MPC el contenido del código de operaciones del RI que indica la direccion de comienzo del modulo del micro programa que describe la instrucción que se quiere realizar.

2)−Cada vez que se encuentre un END se hace un RESET el MPC poniendo su contenido en 0.

3)− Microsalto: (saltos condicionales desde el punto de vista de las microoperaciones).

Para hacer cualquier tipo de salto en el microprograma es necesario incrementar el MPC.

- Una de las alternativas para ejecutar microsaltos: Consiste en modificar el MPC, forzándolo a una direccion de memoria, dicha direccion se encuentra en el RDM. Solo hay microoperaciones, pero ahora introducimos direcciones. (Esquema de Wilkes), se comparten bits de registro indicando direcciones u operaciones. Si la microoperacion es de salto los bits se interpretan como direcciones. Si no es de salto se interpretan como microoperaciones.

- Salto: toma los bits como direcciones de salto

- Div las Microoperaciones

- No Salto: toma los bits como microoperaciones

Luego se codifican. Las microoperaciones representadas por los bits del campo de direcciones no se tienen que ejecutar nunca con las microinstrucciones de salto, puesto que quedan desactivadas cuando se trata de una microinstruccion de salto.

- Es una necesidad tener codificación de microinstrucciones (que indiquen si es una microinstruccion de salto o no).

La codificación está dada por los 2 bits del campo de direcciones:

C0	C1	Campo de direcciones

Suponemos tener una UAL con 2 flags N y Z.

Si C0 = 0 y C1 = 0 no es una micro instrucción de salto.

C0 = 0 y C1 = 1 es una micro instrucción de salto.

Si C0 = 0 y C1 = 0 con A= 0 => la salida del multiplexor será 0, no modifica el MPC

C0 = 0 y C1 = 1 con B, => si N= 0 (por la UAL)=> la salida será 0 y no se modifica el MPC , se modifica si N=1.

C0 = 1 y C1 = 0 se modifica el MPC cuando Z=1.

- Microsaltos incondicionales: se ejecutan con independencia de los flags de la ALU.

C0 = 1 y C1 = 1 es un microsalto => la salida del multiplexor sera 1 incrementa el MPC.

La codificación de microinstrucciones sirve para manejar el multiplexor. Esta es una de las alternativas para resolver el problema de los microsaltos en la UC microprogramada, y no resulta ser la mas compleja.(existen mas complejas y mas sencillas).

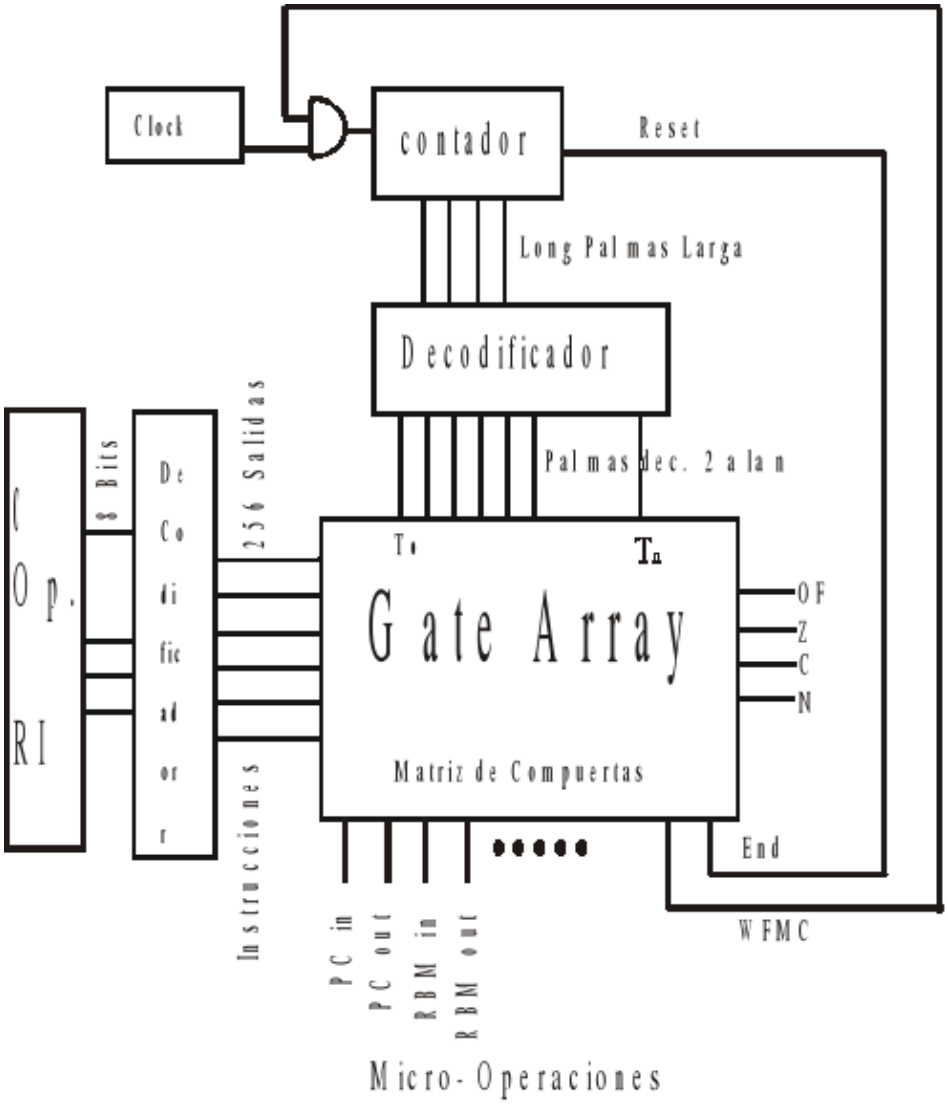
Unidad de control por Hardware

Tendrá que ser capaz de ir siguiendo, paso a paso lo que está descripto en los microprogramas. Saber que cables poner activos y cuales no.

Tendra que saber:

- En que relglon o linea está parda.
- Que instrucción tiene que ejecutar.
- Tener en cuenta los flags de memoria.

- tiene un contador: tiene sieryas entradas que son pulsos de un reloj, los cuales son contados, para decir en que renglon o linea está parada.
- Otra serie de entradas le dira que instrucción ejecutar, otras entradas serán los flags de la ALU.
- Cada microoperacion será un cablecito.
- Cada cablesito es la salida de una compuerta, tendremos tantas compuertas como microoperaciones.

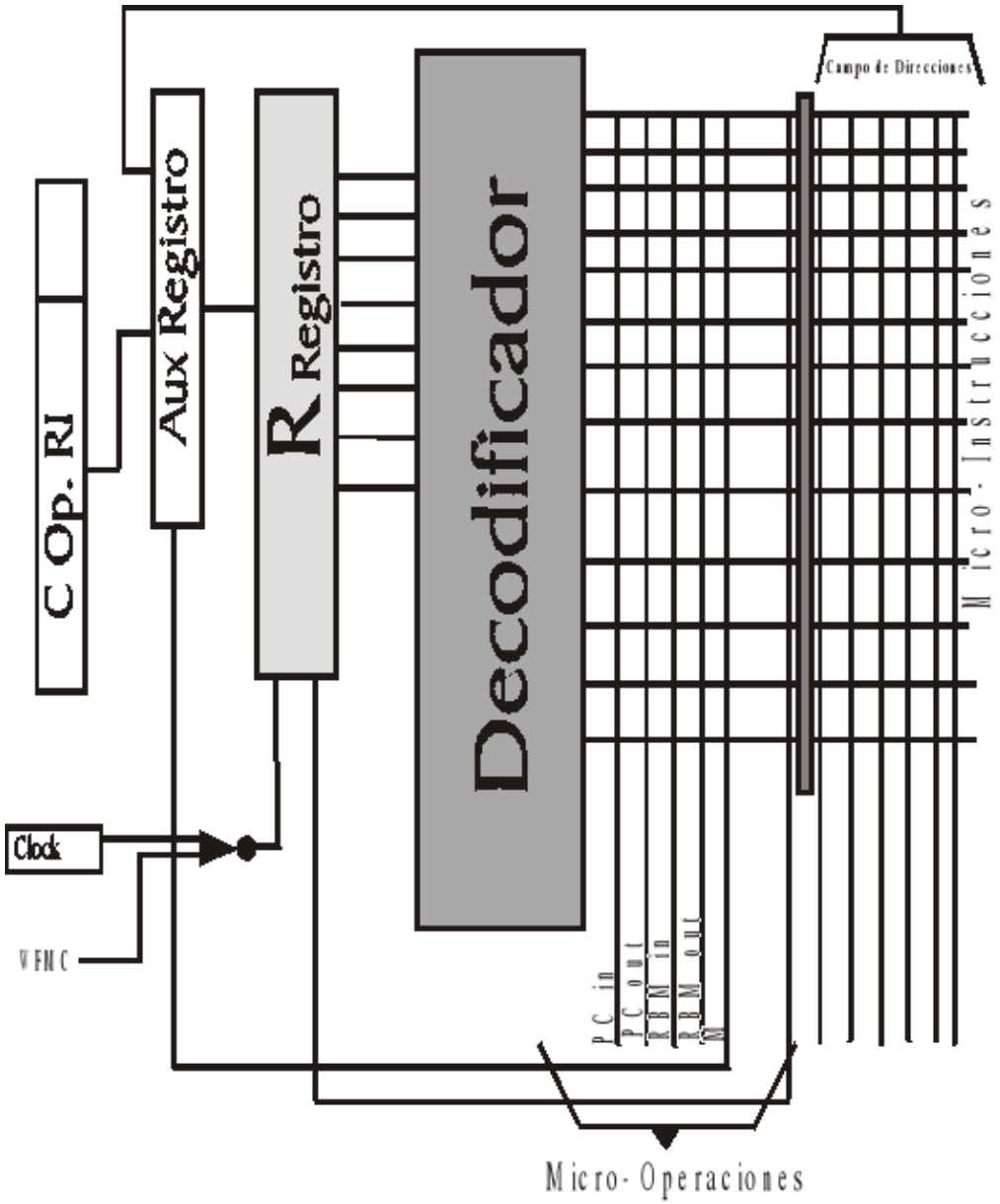


- El contador nos indica donde está parada, es puesto en cero cada vez que se ejecuta un END, o se hace un RESET.
- El tamañ o de dicho contador, debe ser capaz de contener los renglones de las instrucciones mas largas.
- Otra de las enradas será que instrucción debe ejecutar, esta instrucción está en el Campode Operaciones del RI, es un numer binario que indicará la instrucción y el modo de direccionamiento. Hay un decodificador que decodifica el C Op. de RI si tiene 8 entradas tendra 256 salidas 28, que se activará de a una por vez. Un cable activado indicará que instrucción realizar.
- Mediante una ecuacion, usando compuertas AND y OR, podemos describir un circuito para la generacion de los microprogramas. Tenemos una ecuacion para cada microoperacion .
- Lo que se hace es reconocer todas las intrucciones y ver en que renglon están, se escribe en que condicion se puede dar es microoperacion. Es decir el tiempo (Tn) y el nombre de la instrucción (renglon).

RBM out= T2+T4 . LD AC + T3 . ADDDIR+.....

- Caracteristicas: es muy rigida, pero es mas rapida.
- Falencia cada vez que queremos sacarle o agregarle algo, tenemos que cabiar todo.

Unidad de Control Microprogramada



- Tomo todas las instrucciones y las unió con las microoperaciones y formó los microprogramas.
- Todo renglon es una microinstruccion y las lineas verticales las microoperaciones.
- El microprograma se divide en modulos.
- El registro guarda las direcciones de la proxima microinstruccion y se decocifica quedando activada una linea horizontal con una linea vertical automaticamente.
- Las fuentes de direcciones de cada microinstruccion son 2
- El campo que indica la proxima microinstruccion (celda de dir.).
- El codigo de operaciones del RI
- multiplexor contrario a un decodificador, cuatro entradas dos salidas, tiene una llave rotativa que conecta en funcion de las dos entradas.
- C0, C1 a A B C D

- Se colocó para solucionar el problema de los saltos condicionales, se conecto el campo de direcciones del RI al MPC.

- No se pudo llevar a la practica, devido a impedimentos electronicos del momento.

- En realidad este fue el modelo de una memoria de lectura, es decir una memoria ROM.

Comparacion entre UC por Hardware y Microprogramada

- UC H es mas rapida (tiene dos ciclos de retardo).
- UC M para acceder a una microoperacion hay que acceder a memoria (requiere tiempo de acceso de memoria).
- UC H es mas rigida (hay que cambiar la arquitectura de la misma para agregar o sacar algo).
- UC M es mas lenta.

Unidad Aritmetica Logica

Complemento a la Base

- Lo que le falta al numero para llegar a la base.
- Se utiliza para realizar la resta, en si lo que se realiza con un circuito sumador, es hacer la resta sumando el complemento del numero que resta.

A−B = A+CB Base = A+ 2n −B

Si A>=B el numero queda dentro del rango A−B = A+2n −B= 2n+(A−B)

Si A<B no puede ser representado por ser negativo A−B = A+2n −B= 2n+(B−A)

Si el bits de carry es 1 la resta esta bien

es 0 el resultado dio negativo

- A>=B 9−6 = 3

1001 1001

− +

0110 complemento 1010

10011 el bits de carry dio 1 resultado correcto

- A<B 5−7 = −2

0101 0101

− +

0111 complemento 1001

0110 el bits de carry dio 0 resultado negativo

base −(B−A)

16−(7−5)= 14

- 2 = 14
- 14 = 2 = |A−B|

Complemento a la Base−1

- Es lo que le falta al nro para llegar al nro mas grande que se puede escribir en el registro.

CNB−1= 2n −1− A = CNB−1

CNB = CNB−1+1

- Se reemplazan los ceros por unos y los unos por ceros. Sacando el complemento a la base−1 podemos obtener el complemento a la base sumandole 1 al complemento a la base−1.

Resta

A−B = A+ CNB−1 = A+2n−1−B = 2n+(A−B)−1

Si A>= B => A−B = A+ CNB−1 = A+2n+1−B = 2n+(A−B)−1

Si A< B => A−B = A+ CNB−1 = A+2n −1−B = 2n−(B−A)−1

- Si el carry es 1 al resultado debo sumarle 1

es 0 me queda el complemento a la base−1 dando buelta los operandos al resultado siempre le sumo el carry.

- A>=B 10−6 = 4

1010 1010

− +

0110 complemento 1001

a la base−1 **1**0011 le sumo el carry 1

+ **1**

0100 = 4

- A<B 11−13 = −2

1011 1011

− +

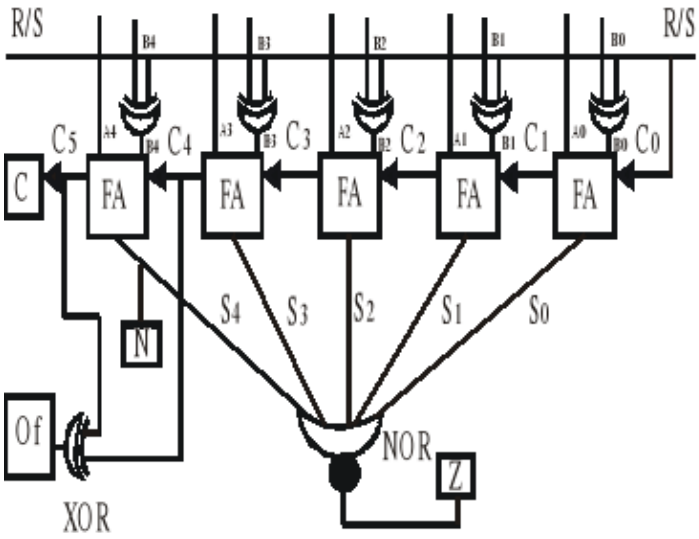
1101 complemento 0010

a la base−1 **0**1101 => **1101** CB−1 **0010**

Overflow: el flags de carry no me sirve para detectar overflows. Podemos decir que para que no haya overflow en los dos transportes, de la penultimaa la ultima etapa y de la ultima etapa hacia fuera deben ser

iguales para que no haya overflow o 0,0 o 1,1 de lo contrario será overflow 0,1 o 1,0.

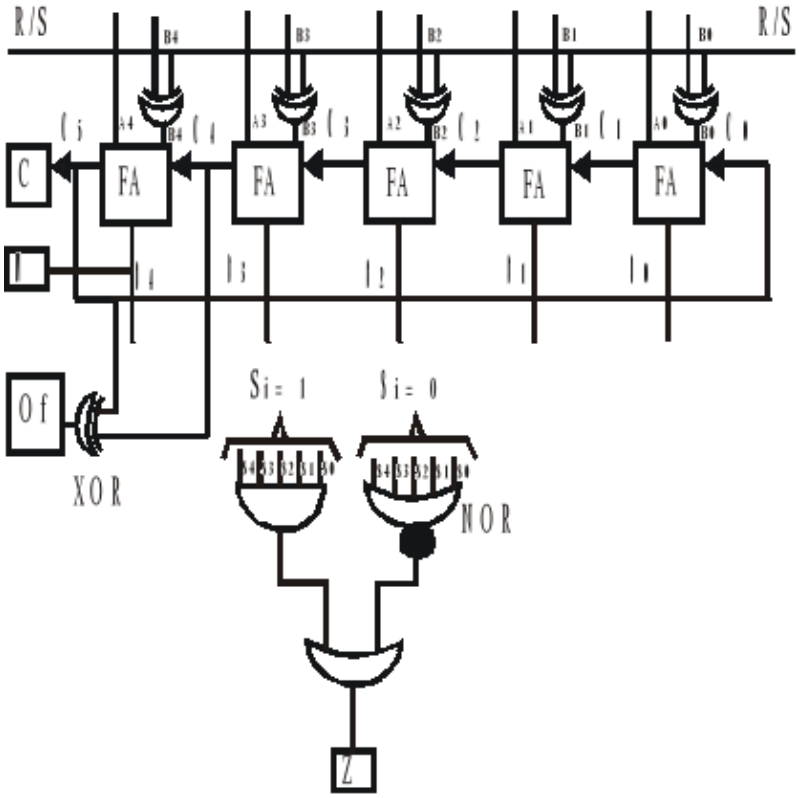
Circuito sumador con complemento a la base de 5 bits:



Sumador Reapple Cary

Tiene un problema demora al producirse la suma ya que los transportes deben

propagarse por los FA. Cuanto mas grande se la longitud de palabra mas lenta es. Si tarda 5 n/s por cada compuerta, lo que tarda en guardar disponible el primer FA son 10 n/s la segunda 20 n/s y así sucecibamente. Si el quinto carry fue un 1 debemos comenzar a modificar todo nuevamente. Esto puedetraer un problema, que esta modificacion del quinto carry no termine nunca. Debemos probar entonces que el carry no produzca mas de una modificacion, haciendo que dicha modificacion o creacion del carry en 1 se propague mientras haya propagaciones hasta llegar al que la generó y all;i se cortará la propagacion y no se sigue modificando.



- la realimentacion implica que debo esperar mas tiempo para estar seguro que en las salidas estará el resultado correcto.
- El de complemento a la base tarda 50 n/s para obtener el resultado.
- El sumador con complemento a la base−1 tarda 90 n/s para obtener resultados correctos.

Sumador con Acelerador de Carry. Lac Carry

Dicho hardware tiene dos funciones, una que crea el carry y otra que lo propaga.

Gi generadora de carry

Pi propagadora de carry

En cada etapa tendremos las entradas y una funcion generadora y propagadorade carry.

C1 = G0+P0.G0

C2 = G1+P1.G1= G1 + P1 (G0 + P0. G0) = G1 + P1.G0 +P1.P0.C0

C3 = G2 + P2.C2+ P2.P1.G0 + P2.P1.P0.G0

Con este hardware tendre 2 retardos uno de generacion y otro de propagacion. Microoperaciones que tardaban 90 n/s tardaran 25 n/s.

- Es mucho mas rapida pero es mucho mas costosa por la gran cantidad de compuertas que demanda el hardware para obtener el carry de antemano.
- El tiempo de retardo no cambia nunca siempre es el mismo por mas que tengamos una ALU de 32, 64, 128, 256. Etc.

Entrada Salida

- Entrada/Salida. Tenemos que tener en cuenta tres aspecto fundamentales.

- Direccionamiento.
- Transferencia de datos.
- Sincronización.
- Direccionamiento:

Se refiere a como se reconocen los dispositivos de E/S. Tiene que ver con la manera en que referenciamos los dispositivos. Estos metodos difieren según la maquina.

- Transferencia de datos DMA:

La transferencia de información se maneja siempre entre los dispositivos de E/S y la memoria. Se trabaja con áreas de memoria designadas por el sistema Operativo, donde cada dispositivo posee su área de memoria.

Así surgen los sistemas de DMA para optimizar el sistema. Lo que hacen es ahorra trabajo a la CPU. La utilidad de estos dispositivos tiene sentido en los ambientes multitarea.

- Sincronización Interrupciones:

Significa coordinar esta transferencia, coordinar CPU y memoria por un lado y otros dispositivos de E/S por otro lado.

La CPU y la memoria están reguladas por un reloj, pero los dispositivos de E/S tienen sus tiempos particulares.

Los mecanismos que realizan la sincronización son los mecanismos de interrupciones. Tener en cuenta que no todas las interrupciones son de E/S (Ej. Interrupciones de CPU).

- Comenzamos analizando un ejemplo:

Analizamos el caso de querer hacer funcionar una impresora, con una CPU tal como la conocemos.

Puede ser que la CPU tenga 1 o 2 buses internos, hablaremos entonces del bus de memoria. Este bus esta compuesto por 2 buses: de direccion de datos; y del bus de control.

Los nombres están vistos siempre desde la CPU, leer significa introducir algo a la CPU, escribir significa mandar algo desde la CPU.

Cuando la CPU debe escribir en el impresor, manda un bits (que puede ser por ej. La letra `a'), y le dice que escriba.

Pero desde el punto de vista del impresor esto es totalmente diferente.

- El pin de Strobe es el que valida los datos. Si está activado significa que los datos son validos, en caso contrario, no lo son. Esta señal permanece activa un tiempo muy corto, es una señal de tipo pulso.

- La señal de busy indica si el impresor estará dispuesto a aceptar el dato que se le envía, ya que el impresor puede estar ocupado imprimiendo otras cosas. Si está en 1 estará ocupado, el impresor no podrá recibir el dato.

Aparentemente el bus de la CPU no tiene nada que ver con los pines del impresor. Cuando la CPU quiere escribir va a poner en el bus de dirección la dirección del impresor, en el bus de datos el o los bits a imprimir, y en el de control la señal para que escriba.

Entonces deberá haber algo en el medio llamado controlador:

Esta controladora debe de tener como mínimo dos registros: Buffers y C. Status. Los dispositivos de E/S a trabes e esta controladora, la CPU ve solo dos registros y los interpreta como si fueran dos palabras de memoria.

Una de las cosas mas importantes que debe de realizar esta controladora es reconocer la dirección del periférico al cual está manejando.

Lo que hace es comparar los bits del bus de dirección con un numero binario que representa al impresor.

Desde el punto de vista de las CPU la entrada de datos al impresor es igual a la entrada de datos a memoria, es decir la CPU ve igual la escritura en el impresor que la escritura en memoria.

Antes de generar el proceso de escritura, debe verificarse el pin de busy pues si el impresor esta ocupado no se puede generar la escritura porque el impresor no va a aceptar los datos.

Se introduce entonces el registro Status (indica por ejemplo si el impresor tiene papel o no, si está ocupado o no, etc), este registro tiene bits que le sirven a la CPU para saber en que estado se encuentra el impresor y otros para controlarlo, es vi direccionable.

¿Que tiene que hacer un programa para que imprima?

El S.O. arma la línea imprimir en el área de memoria que le corresponde, coloca un puntero al primer bits para saber cuando termina.

Entonces lee el reg. C. Status para saber si el impresor está en busy o no, si está ocupado, se que da en un loop preguntando hasta que se desocupe. Cuando se desocupa hace una escritura sobre el registro Buffer, enviando los bits al impresor; luego aumenta en uno el puntero y disminuye lo que le indica la cantidad de bits a imprimir, si está en cero es porque ya terminó de imprimir.

El contenido de los dos registros deberán ser colocados en memoria de la CPU y aquí encontramos diferencias entre las maquinas de 1 y 2 buses.

Maquinas de 1 bus:

En el espacio de direcciones de memoria deberá colocar las direcciones de los dispositivos de E/S pero entonteces deberá diferenciar las direcciones de E/S.

Los 4 k de direcciones de memoria mas alta se usan para colocar los registros de los distintos dispositivos de E/S.

Si tengo una memoria de 64k y emito una dirección entre 60 y 64 k me estaré refiriendo a un dispositivo de E/S. estas maquinas no tienes instrucciones de E/S.

Maquinas de 2 buses:

En la arquitectura de 2 buses habrá un juego de instrucciones para manejarme con memoria y otro para

manejarme con E/S.

Si estoy usando instrucciones de E/S las direcciones que involucran las van a salir por 1. Si estoy usando instrucciones de memoria, las direcciones que involucran van a salir por 2.

Tengo dividido el espacio de memoria . cuando ejecuto una dirección de memoria, lo que hace es referirse a la palabra que está en la parte de memoria no reservada para E/S. en cambio cuando se trata de una instrucción de E/S la dirección sale por 1.En el otro esquema no puedo diferenciar.

Volvamos a los tres puntos a tener en cuenta.

- Direccionamiento:

esto varia según el tipo de CPU ya sea de 1 o 2 buses, pero desde el punto de vista del dispositivo es lo mismo.

- Transferencia de datos:

Supongamos tener una maquina de 2 buses. Para imprimir una línea, el S.O. tiene que armar en al zona de trabajo del impresor la línea a imprimir, es decir los bits que debe mandar uno a uno (el impresor no tiene un registro buffer que le permita guardar varios bits).

Los dispositivos de E/S están manejados por un programita llamado Driver, sirven para la transferencia de datos entre la memoria y E/S.

Es importante en un ambiente multitarea que no sea la CPU quien maneje los dispositivos de E/S, pues ella puede ir ganando tiempo ejecutando otros procesos. Entonces se deben crear otros dispositivos hardware llamados dispositivos DMA, lo que hacen estos controladores es algo parecido a lo que hacen los Driver. Estos controladores difieren entre una maquina de 1 o 2 buses. El DMA lo que hace es la transferencia de datos entre la memoria y los dispositivos de E/S. Luego es aquí donde se fija la diferencia entre las diferentes maquinas.

DMA en maquinas de 1 Bus:

En este tipo de maquinas hay un dispositivo de DMA para cada dispositivo de E/S. Generalmente el DMA estará incluido en la plaqueta controladora, que es quien maneja este dispositivo.

Ahora nuestra controladora tendrá también dos registros **Pprint** y **Count**, nuestra cpu tendrá que reconocer ahora 4 registros, osea que la zona de memoria, si nuestra memoria es de 64k, reservará desde 60 hasta los 64..

Ahora para imprimir. Lo que hace el S.O. es armar la línea a imprimir, realiza una operación de escritura sobre **Pprint,** luego duerme este proceso para seguir ejecutando otros procesos, el próximo que esté en la cola de espera.

El DMA debe ver que el impresor esté busy, si está desocupado el Dma debe realizar un ciclo de lectura en memoria para traer los bits a imprimir, luego incrementa el **Point** y decrementa el **Count**, **notar que aparece alguien mas que accede a memoria.**

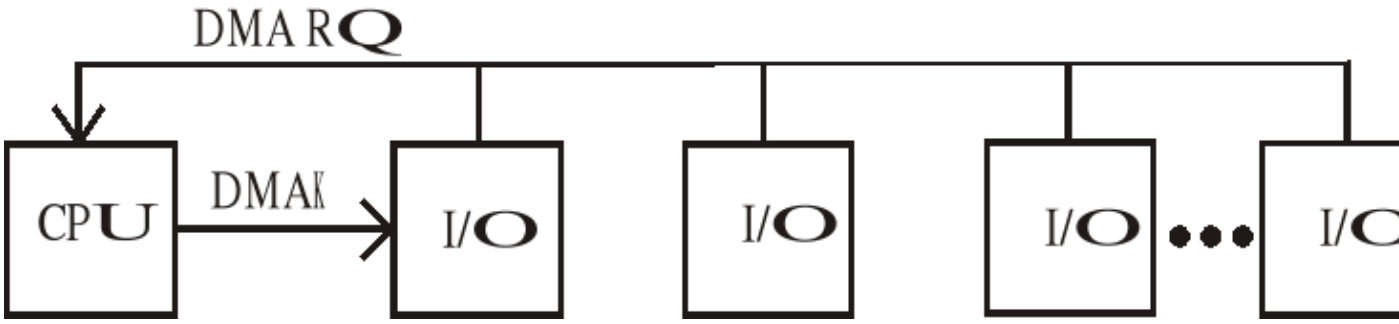
El único problema que esto presenta es que la memoria tiene una única puerta de entrada.

Supongamos que la CPU y el DMA quieran acceder al mismo tiempo al bus de datos. Surge aquí el problema

de competencia. Para resolver este problema debe establecerse una jerarquía, es decir alguien tiene mayor prioridad y esta prioridad se le otorga a la CPU. Como hay barios dispositivos de DMA será la CPU quien tenga la mayor jerarquía.

Surge también el problema de competencia por el bus entre los distintos DMA, que existan, también hay una jerarquía entre los dispositivos. Este probl. se resuelve mediante un sistema llamado **Daisy Chain** (Cadena margarita).

Sistema **Daisy Chain** para una maquina de un único bus:

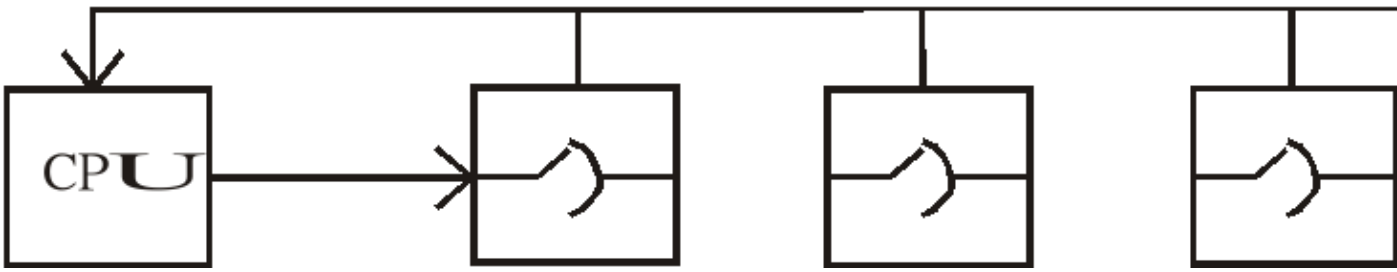


Como la CPU es quien tiene mayor jerarquía sobre el bus, ante una petición del mismo por un dispositivo **DMA RQ**, si el bus está ocupado por la CPU no presta atención a dicha petición. Por ejemplo cuando la CPU termina la ejecución del fetch se fija si algún dispositivo realizo alguna petición del bus, si es así la CPU genera una señal de respuesta positiva y se desentiende **DMAK**.

Lo que le interesa a la CPU es si alguien pide el bus o no.

Solución al problema de jerarquía entre los dispositivos:

Mecanismo **Daisy Chain**



- Si el dispositivo no pidió el bus la llave permanece cerrada, caso contrario se abre.
- El dispositivo mas cercano a la CPU es el que tiene mayor jerarquía.
- Tiene que existir una señal que indique cuando un dispositivo desocupa el bus (DMA GNT).

- Este mecanismo tiene un problema, que dispositivos de menor jerarquía no accedan nunca a memoria. Este es un mecanismo de prioridad estática
- Hay dispositivos de prioridad dimana que evitan que dispositivos queden sin acceder a memoria.

Mascara de interrupción:

La mascara de interrupción soluciona el problema del pedido de interrupción de cualquier dispositivo, en cualquier momento, es decir cuando un dispositivo pide el bus la CPU duerme el proceso en ejecución y acepta el pedido, lo cual no funciona en forma jerárquica, lo9 que queremos es que tenga prioridad el de

mayor jerarquía, para esto coloca un registro mascara con el cual acepto que IRQ acepto y cual no.

Si el que interrumpió es 2 deja en 1 solo el primero y los demás en cero. Como la CPU va a comenzar a barre nuevamente los pedidos de IRQ, vera de ellos solo al que tiene un 1 en el reg. Mascara es decir solo al primero, que es quien tiene mayor jerarquía que 2.

Esta es una forma de resolver la prioridad de que un dispositivo de mayor jerarquía puede interrumpir al proceso de interrupción de otro de menor jerarquía.

Otra solución:

Hay maquinas que no tienen este reg. mascara sino que cuando se pide una interrupción es PSW del dispositivo se carga en el PSW de la CPU y analiza su valor, tomando como referencia ese valor el programa que interrumpe tiene una jerarquía y solo puede ser interrumpido por otro programa de mayor jerarquía. Estas maquinas permiten tener un sistema Daisy Chain en cada nivel.

Interrupciones Internas

Son usadas fundamentalmente por la CPU, hay una o dos interrupciones que están en todas las maquinas. Estas interrupciones evitan que la maquina se plante o cuelgue cuando pasa algo raro, generalmente son errores de la ALU ej. la división por cero.

¿Como funcionan?

Unas son generadas por la ALU y otras por la UC cuando intenta decodificar una operación errónea.

Estas interrupciones deben ser atendidas inmediatamente.

Cuando dividimos por cero por ejemplo, la CPU va salvar el PC en la Pila STACK, salvará también el PSW y va a cargar en el PC el contenido de la dirección 2 y en el PSW el contenido de la dirección 3. en ese momento comienza un programa del S.O. que dará un mensaje por pantalla diciendo que se interrumpió debido a una división por cero. Este tipo de interrupciones no dejan que se ejecute el programa en caso de algún error. A diferencia de las interrupciones externas que esperan que se termine de ejecutar la aplicación en curso.

Interrupciones por Software

- El programador escribe un programa y en una línea puede poner **INT 21** que es la dirección de comienzo del programa de interrupcion. Genera un salto al sistema operativo para devolverle el control.
- Diferencia entre **Jsub, Int #**, la primera queda determinada en tiempo de compilación y la segunda en tiempo de ejecución.

Paralelismo

- Las tendencias que se han seguido son para potenciar el modelo de Von Neuman, para llegar a una optimización de todo el sistema.

El problema que presenta el modelo mencionado es que posee una sola CPU, Memoria, y una única UC, además el accceso a memoria se puede realizar de a un ciclo por ves, quedando ciclos ociosos. Dicho modelo no nos permite realizar tareas en forma simultanea.

Hay dos maneras de hacer paralelismo:

- **Paralelismo temporal:** dicho paralelismo se hace en el tiempo y se apolla en el esquema de pipe−line.
- **Paralelismo espacial:** hay muchas CPU, la simultaneidad se distribuye en el espacio por eso es espacial.

Paralelismo Temporal: Pipe−Line Lineal

La arquitectura pipe−line se aplica en dos lugares de la maquina, en la CPU y en la UAL.

Veamos en que consiste el pipe−line y tratemos de entender porque el pipe−line mejora el rendimiento de todo el sistema.

Veamos una CPU no organizada en pipe−line:

Si se trata de una instrucción a ser ejecutada por la ALU podemos decir que la CPU realiza a lo largo del ciclo de maquina estas 5 tareas.

Una vez que termina de ejecutar una instrucción va a buscar otra y tarda en ejecutarla un tiempo T, es decir cada T segundos ejecuta una instrucción.

¿Qué sucede si dividimos en 5 unidades según las 5 cosas que realiza la CPU?

Supongamos la CPU dividida en 5 unidades, de tal forma que c/u tarde lo mismo en realizar su partecita. Es decir c/u tardará T/5.

Para que una instrucción se ejecute se necesita T segundos entonces para que usar pipe−line.

Si ocurre esto en una CPU normal a una con pipe−line, la cantidad de instrucciones que se hacen por segundo aumenta, es decir aumenta el flujo de instrucciones que se ejecutan por segundo.

¿Qué pasa si el pipe−line no es ideal?

Un pipe−line es ideal cuando todas las sub−unidades tardan lo mismo en realizar su tarea.

Si el pipe −line no es ideal debo coordinar de alguna forma el paso de instrucciones de una sub−unidad a otra, que no llegue una instrucción a una etapa que todavia no a terminado la instrucción anterior.

Para ello se colocan buffers entre las sub unidades y el paso de las instrucciones se rige según la etapa mas lenta (en este caso T/4).

¿Cuáles son las condiciones que me limitan este numero ideal de T/N por etapas siendo N la cantidad de etapas?

- Por un lado el tiempo de cada etapa será igual al de las demás.
- También puede suceder que un dato que se esté calculando en S3, se esté buscando en S4, o escribiendo en S5. esto es un problema de **dependencia de datos.**
- Otro problema es el de los saltos condicionales que hacen que se interrumpa el secuenciamiento normal de un programa interrumpiéndose también el pipe−line comienzan quedar huecos en el diagrama dibujado anteriormente, pues disminuye el numero de instrucciones por segundo que pueden ejecutarse.

¿Cómo solucionar estos problemas?

- **Como el pipe−line no es ideal se colocan buffers entre cada etapa y se hace la transferencia de una sub−unidad a otra simultaneamente aunque ellas no trabajen a la misma velocidad. Para ello se lleva un clock y se cambia de una sub−unidad a otra según la sub−unidad mas lenta.**
- **Para los saltos condicionales se llevan 2 pipe−line uno para las instrucciones que no son de salto y otro para realizar las instrucciones de salto.**
- **Para solucionar la dependencia de datos:**

Puede ocurrir que una instrucción J vaya a buscar un operando que esta siendo calculado por la instrucción I que le presede.

I A = B+C

J D = A+B

Si yo dejo que esto se siga procesando va a leer algo en la dirección A que es incorrecto.

Esto pasa por el solapamiento de tiempo entre una etapa y otra.

Esto ocurre también en el paralelismo espacial.

¿Como solucionarlo?

Debemos analizar los casos de riesgo.

El dominio de una instrucción está formado por todos los datos que intervienen en la ejecución de la misma y todos los datos y todos los datos que una instrucción modifica con su ejecución constituyen el rango de la misma.

Dom I = B, C

Dom J = A, B

Rango I = A

Rango J = D

Existen tres tipos de riesgo:

- **RAW: Es el riesgo de leer antes de haber escrito, entonces habrá una instrucción que tenga que leer después de que otra haya hecho la escrita. Como solucionarlo:**

Buscando que; R(I) " D(J) = "

- **WAW: hay dos instrucciones que deben escribir pero una debe hacerlo antes que la otra:**

I A = B+C

J A = D+H

K L = A+B

Para ver que la A de K no pueda confundirse con la de J o I debe suceder;

R(I) " R(J) = "

Este tipo de riesgo sucede en maquinas de mas de un procesador.

- **WAR: es el riesgo de no escribir después de que se haya leído**

I D = B*F

J E = A+B

Verificamos : D(I) " R(J) = "

- **RAR: este tipo de riesgo no existe ya que no hay modificaciones en la ejecucion de instrucciones, luego no existe tal riesgo.**

Pipe−Line No Lineal

El pipe−line lineal se caracteriza porque las entradas provienen de la salida de la etapa anterior, con este tipo de pipe−line se logra un rendimiento 100% siempre que no surjan saltos.

Un pipe−line no lineal se caracteriza porque las entradas y salidas no están conectadas directamente a las salidas de la etapa anterior, es decir no hay una coneccion lineal, sino que hay otras opciones.

En este caso puede haber realimentación, es decir inyección de las salidas a las entradas.

Se clasifican en:

- **Unifuncionales**: Realizan una única función
- **Multifuncionales**: de acuerdo a la orientación de una llaves desempeñan varias funciones.

Los **Multifuncionales** se subdividen en:

- **Estáticos**: pueden cambiar de función pero deben vaciarse.
- **Dinámicos**: pueden cambiar de función sin necesidad de ser vaciados.

Los tiempos donde podría insertar otro proceso se denomina latencia. Hay latencias prohibidas, y otras permitidas.

Las latencias nos dan idea de cuanto se acerca o se aleja un hipe−line del rendimiento optimo.

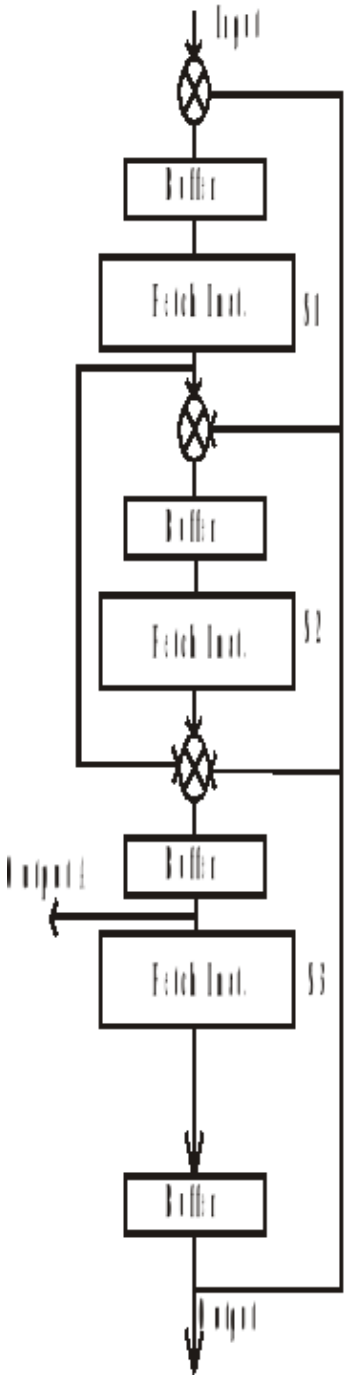
Paralelismo Espacial

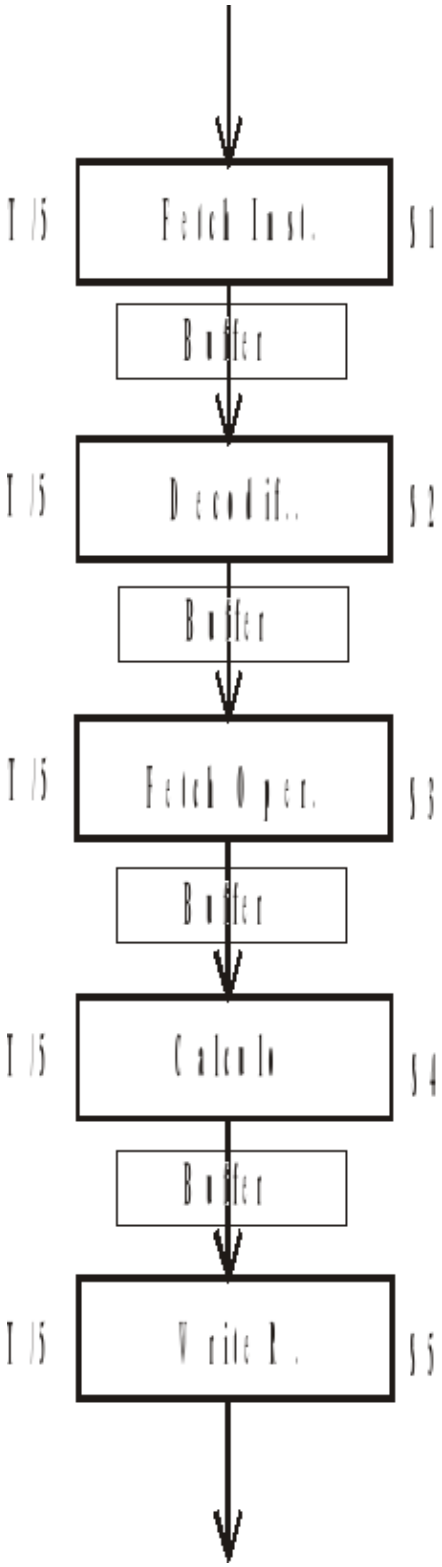
Espacial porque hay una distribucion de tareas no en el tiempo sino en el espacio. En vez de haber una sola CPU tendremos varias, con varias UAL.

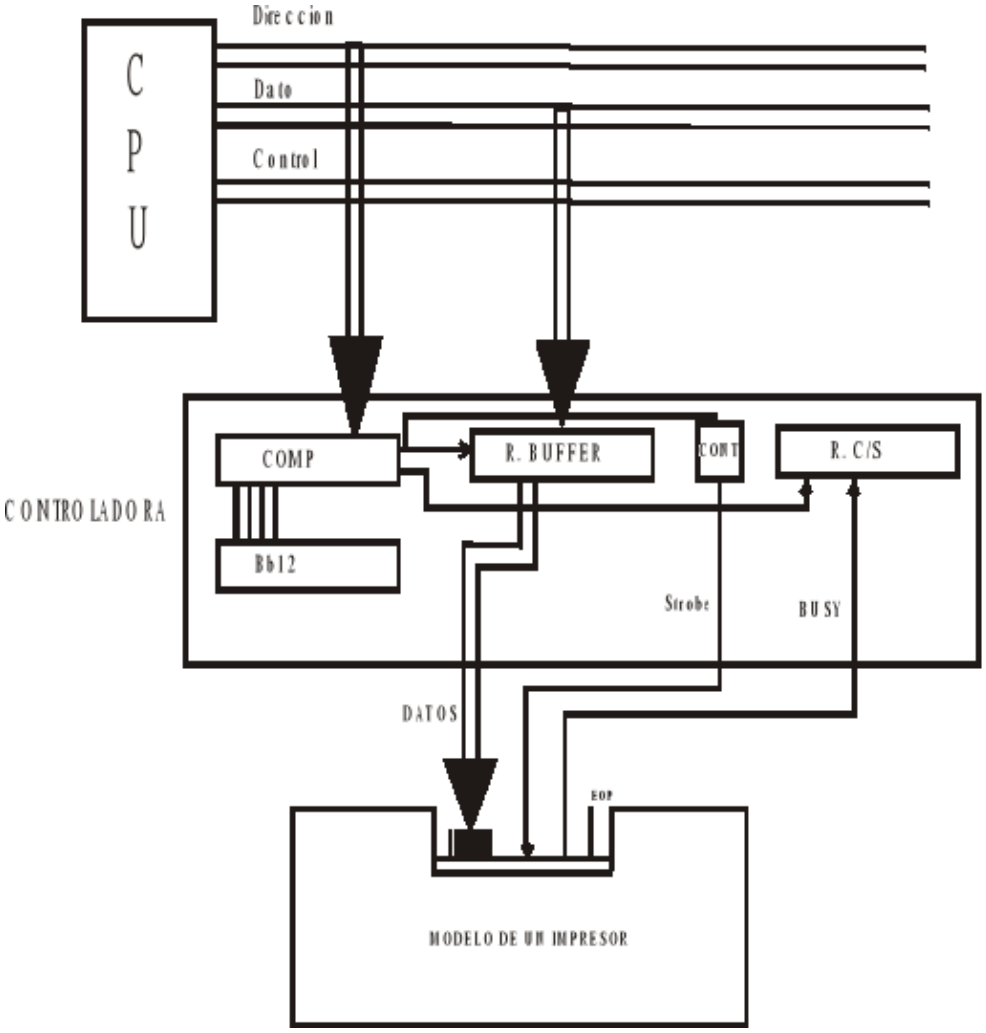
Veremos una clasificación propuesta por Flynn.

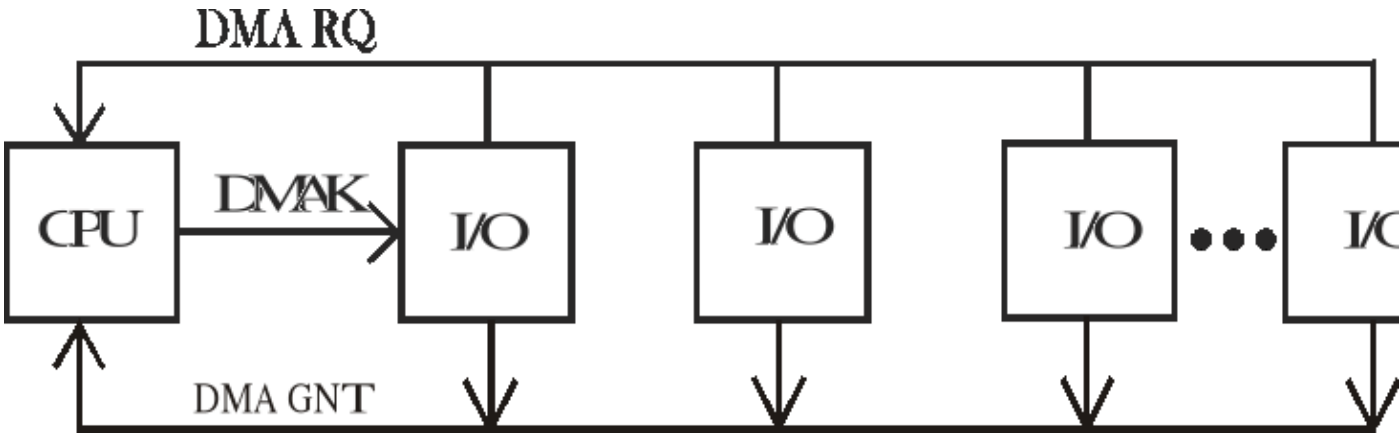
- S.I.S.D. : simple camino de control y simple camino de datos. Una UC, CPU, UAL.
- S.I.M.D. : estas maquinas tienen un único camino de control y varios de datos es decir tendrán varias UC y Memorias.(De nada serviría varias UAL y una única memoria).
- M.I.S.D. : tendremos muchos caminos de control o instrucciones y uno solo de datos. A este tipo de maquinas no se les encontró sentido son maquinas que no existen.

- M.I.M.D. : tendremos muchos caminos de instrucciones, muchos caminos de datos, tendremos muchas UC, Muchas UAL y muchas memorias, estas maquinas si existen.

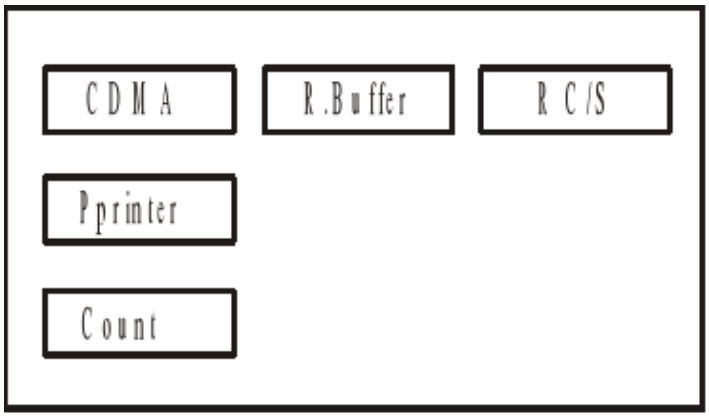


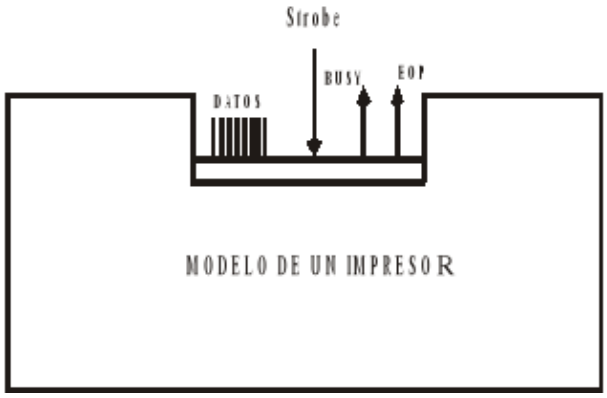

T/5
Fetch Inst.
S1
Buffer
T/5
Decod if..
S2
Buffer
T/5
Fetch Oper.
S3
Buffer
T/5
Calcula
S4
Buffer
T/5
Write R.
S5

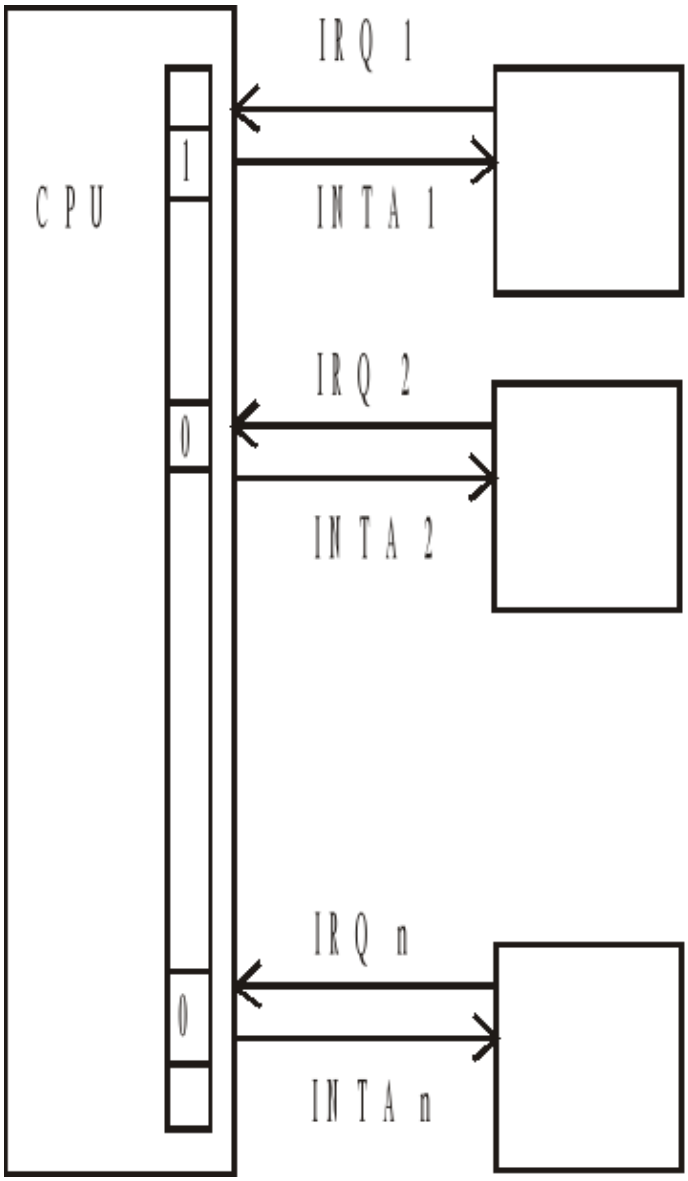

Direccion
C
P
U
Dato
Control
CONTROLADORA
COMP
R. BUFFER
CONT
R. C/S
Bb12
Strobe
BUSY
DATOS
EOP
MODELO DE UN IMPRESOR

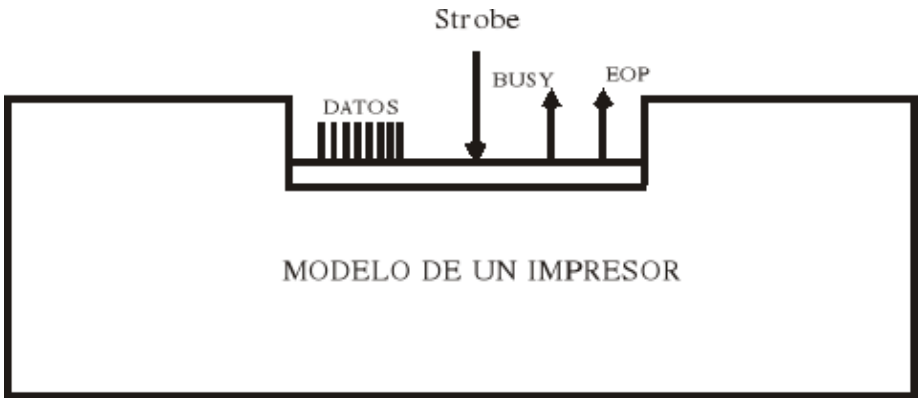


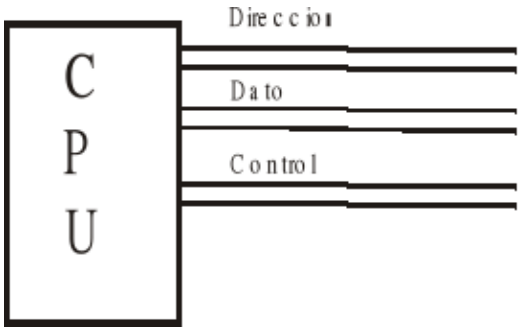
Controlador






IRQ 1
CPU
1
INTA 1
IRQ 2
0
INTA 2
IRQ n
0
INTA n


Strobe
BUSY
EOP
DATOS
MODELO DE UN IMPRESOR



FA

FA

FA

C

C

C