MODULO 1. ESTRUCTURA DE LA COMPUTADORA

Introducción: Este es el primer curso de software de sistemas en el que se conocerán los principios básicos de una computadora para conocer como se deben de diseñar los programas que controlarán y recibirán datos en un sistema de computo cuyo objetivo es procesar información, de este curso se desprenden los sistemas operativos de una computadora, para poder lograr esto es necesario primero conocer como esta formada una computadora, se mostraran las dos arquitecturas de computadoras que existen, como son el modelo de Von Newmann y Harvard, el proceso de ejecución de programas y así como las diversas modalidades y mecanismos de ensamblado, carga y Macroprocesamiento.

Durante el curso se utilizara como elemento de estudio el Microcontrolador MC68HC12 de FREESCALE para el cual se analizara su modelo de programación, considerando que si se conoce un Microcontrolador será más sencillo emigrar a cualquier otro.

1.1 Arquitectura de von Newmann.

La **arquitectura de computadoras** es el diseño conceptual y la estructura operacional fundamental de un sistema de computadora. Es decir, es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso (CPU) trabaja internamente y accede a las direcciones de memoria.

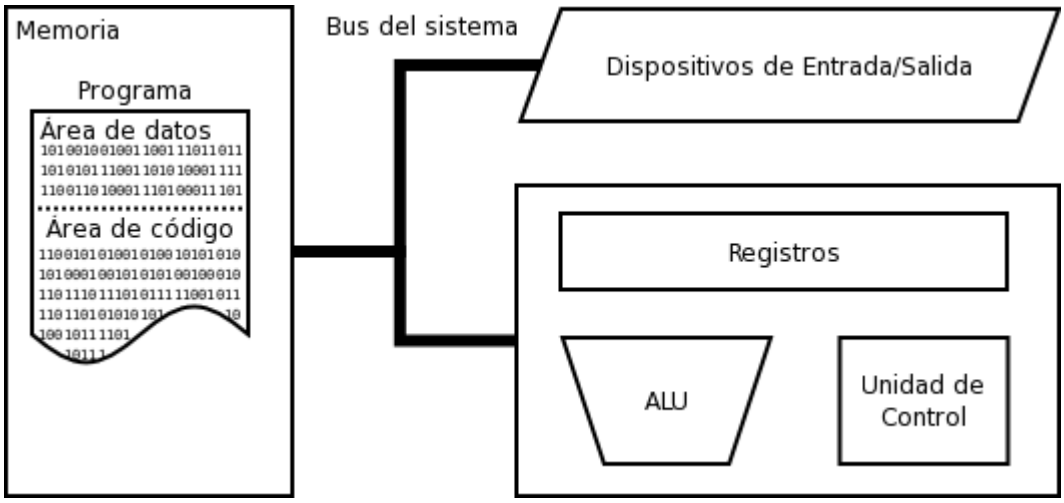
También suele definirse como la forma de seleccionar e interconectar componentes de hardware para crear computadoras según los requerimientos de funcionalidad, rendimiento y costo.

En la actualidad existen dos arquitecturas de computadoras básicas ampliamente utilizadas, la arquitectura de Von Newmann y la arquitectura de Harvard.

La **arquitectura de von Neumann** es una familia de arquitecturas de computadoras que utilizan el mismo dispositivo de almacenamiento tanto para las instrucciones como para los datos (a diferencia de la arquitectura Harvard).

La mayoría de computadoras modernas están basadas en esta arquitectura, aunque pueden incluir otros dispositivos adicionales, (por ejemplo, para gestionar las interrupciones de dispositivos externos como ratón, teclado, etc).

Los ordenadores con esta arquitectura constan de cinco partes: La unidad aritmético-lógica o ALU, la unidad de control, la memoria, un dispositivo de entrada/salida y el bus de datos que proporciona un medio de transporte de los datos entre las distintas partes.




El bloque que contiene los REGISTROS, la ALU y la UNIDAD DE CONTROL en realidad representa a todo el CPU. La memoria tiene dos áreas principales la de datos; en donde se almacenaran todas las variables que requieran los programas y la de código en donde se almacena el programa que ejecutara la computadora en si.

El término **Arquitectura Harvard** originalmente se refería a las arquitecturas de computadoras que utilizaban dispositivos de almacenamiento físicamente separados para las instrucciones y para los datos (en oposición a la Arquitectura de von Neumann). El término proviene de la computadora Harvard Mark I, que almacenaba las instrucciones en cintas perforadas y los datos en interruptores.

Todas las computadoras constan principalmente de dos partes, la CPU que procesa los datos, y la memoria que guarda los datos. Cuando hablamos de memoria manejamos dos parámetros, los datos en sí, y el lugar donde se encuentran almacenados (o dirección). Los dos son importantes para la CPU, pues muchas instrucciones frecuentes se traducen a algo así como "Toma los datos de ésta dirección y añádelos a los datos de ésta otra dirección", sin saber en realidad qué es lo que contienen los datos.

En los últimos años la velocidad de las CPUs ha aumentado mucho en comparación a la de las memorias con las que trabaja, así que se debe poner mucha atención en reducir el número de veces que se accede a ella para mantener el rendimiento. Si, por ejemplo, cada instrucción ejecutada en la CPU requiere un acceso a la memoria, no se gana nada incrementando la velocidad de la CPU - este problema es conocido como 'limitación de memoria'.

Se puede fabricar memoria mucho más rápida, pero a costa de un precio muy alto. La solución, por tanto, es proporcionar una pequeña cantidad de memoria muy rápida conocida con el nombre de caché. Mientras los datos que necesita el procesador estén en la caché, el rendimiento será mucho mayor que si la caché tiene que obtener primero los datos de la memoria principal. La optimización de la caché es un tema muy importante de cara al diseño de computadoras.

La arquitectura Harvard ofrece una solución particular a este problema. Las instrucciones y los datos se almacenan en cachés separadas para mejorar el rendimiento. Por otro lado, tiene el inconveniente de tener que dividir la cantidad de caché entre los dos, por lo que funciona mejor sólo cuando la frecuencia de lectura de instrucciones y de datos es aproximadamente la misma. Esta arquitectura suele utilizarse en DSPs, o procesador de señal digital, usados habitualmente en productos para procesamiento de audio y video.

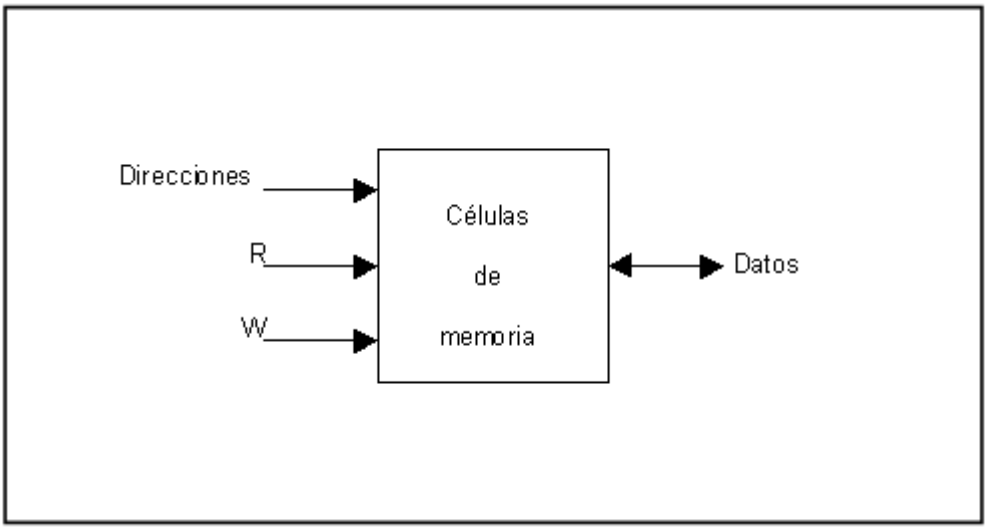
### 1.1.1 Concepto de CPU

La **unidad central de procesamiento,** o **CPU** (por el acrónimo en inglés ***C**entral **P**rocessing **U**nit*), o, simplemente, el **procesador**, es el componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora. Las CPU proporcionan la característica fundamental de la computadora digital (la programabilidad) y son uno de los componentes necesarios encontrados en las computadoras de cualquier tiempo, junto con el almacenamiento primario y los dispositivos de entrada/salida. Se conoce como microprocesador el CPU que es manufacturado con circuitos integrados. Desde mediados de los años 1970, los microprocesadores de un solo chip han reemplazado casi totalmente todos los tipos de CPU, y hoy en día, el término "CPU" es aplicado usualmente a todos los microprocesadores.
La expresión "unidad central de proceso" es, en términos generales, una descripción de una cierta clase de máquinas de lógica que pueden ejecutar complejos programas de computadora. Esta amplia definición puede fácilmente ser aplicada a muchos de los primeros ordenadores que existieron mucho antes que el término "CPU" estuviera en amplio uso. Sin embargo, el término en sí mismo y su acrónimo han estado en uso en la industria de la informática por lo menos desde el principio de los años 1960. La forma, el diseño y la implementación de las CPU ha cambiado drásticamente desde los primeros ejemplos, pero su operación fundamental ha permanecido bastante similar.
Las primeras CPU fueron diseñadas a la medida como parte de una computadora más grande, generalmente una computadora única en su especie. Sin embargo, este costoso método de diseñar los CPU a la medida, para una aplicación particular, ha desaparecido en gran parte y se ha sustituido por el desarrollo de clases de procesadores baratos y estandarizados adaptados para uno o muchos propósitos. Esta tendencia de estandarización comenzó generalmente en la era de los transistores discretos, computadoras centrales, y microcomputadoras, y fue acelerada rápidamente con la popularización del circuito integrado (IC), éste ha permitido que sean diseñados y fabricados CPU más complejos en espacios pequeños (en la orden de milímetros). Tanto la miniaturización como la estandarización de los CPU han aumentado la presencia de estos dispositivos digitales en la vida moderna mucho más allá de las aplicaciones limitadas de máquinas de computación dedicadas. Los microprocesadores modernos aparecen en todo, desde automóviles, televisores, neveras, calculadoras, aviones, hasta teléfonos móviles o celulares, juguetes, entre otros.

### 1.1.2 Concepto de MEMORIA

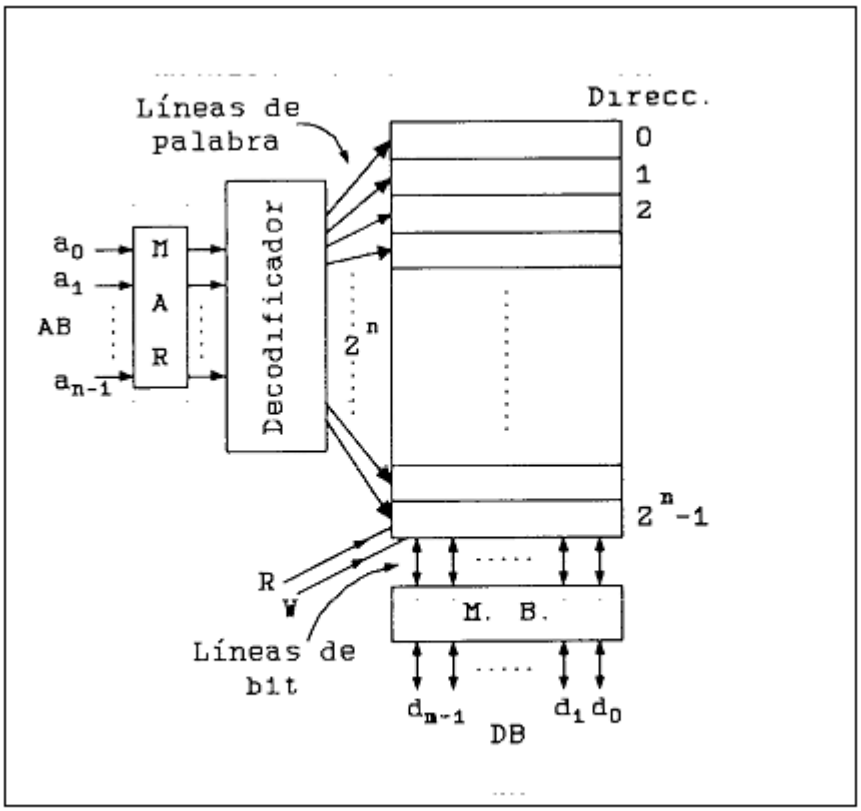
La memoria de un ordenador se puede definir como los circuitos que permiten almacenar y recuperar la información. En un sentido más amplio, puede referirse también a sistemas externos de almacenamiento, como las unidades de disco o de cinta.

Hoy en día se requiere cada vez más memoria para poder utilizar complejos programas y para gestionar complejas redes de computadores.




**Elementos que la componen**

Una memoria vista desde el exterior, tiene la estructura mostrada en la figura de arriba. Para efectuar una lectura se deposita en el bus de direcciones la dirección de la palabra de memoria que se desea leer y entonces se activa la señal de lectura (R); después de cierto tiempo (tiempo de latencia de la memoria), en el bus de datos aparecerá el contenido de la dirección buscada. Por otra parte, para realizar una escritura se deposita en el bus de datos la información que se desea escribir y en el bus de direcciones la dirección donde deseamos escribirla, entonces se activa la señal de escritura (W), pasado el tiempo de latencia, la memoria escribirá la información en la dirección deseada. Internamente la memoria tiene un registro de dirección (MAR, memory address register), un registro buffer de memoria o registro de datos (MB, memory buffer, o MDR, memory data register) y, un decodificador como se ve en la figura de abajo. Esta forma de estructurar la memoria se llama organización lineal o de una dimensión. En la figura cada línea de palabra activa todas las células de memoria que corresponden a la misma palabra.




Como veíamos en las arquitecturas de computadoras podemos clasificar las memorias de acuerdo a su uso dentro de la computadora en memoria de programa y memoria de datos, esta clasificación es muy importante en la arquitectura de Harvard ya que esta usa líneas exclusivas para cada una de estas memorias, en la arquitectura de VON NEWMANN esta separación no es tan evidente ya que el CPU no diferencia entre una memoria y otra. Sin embargo, en un ordenador hay una jerarquía de memorias atendiendo al tiempo de acceso y a la capacidad que normalmente son factores contrapuestos por razones económicas y en muchos casos también físicas. Comenzando desde el procesador al exterior, es decir en orden creciente de tiempo de acceso y capacidad, se puede establecer la siguiente jerarquía:

**- Registros de procesador**: Estos registros interaccionan continuamente con la CPU (porque forman parte de ella). Los registros tienen un tiempo de acceso muy pequeño y una capacidad mínima, normalmente igual a la palabra del procesador (1 a 8 bytes).

**- Registros intermedios**: Constituyen un paso intermedio entre el procesador y la memoria, tienen un tiempo de acceso muy breve y muy poca capacidad.

**- Memorias caché**: Son memorias de pequeña capacidad. Normalmente una pequeña fracción de la memoria principal y pequeño tiempo de acceso. Este nivel de memoria se coloca entre la CPU y la memoria central. Hace algunos años este nivel era exclusivo de los ordenadores grandes pero

actualmente todos los ordenadores lo incorporan. Dentro de la memoria caché puede haber, a su vez, dos niveles denominados caché “on chip”, memoria caché dentro del circuito integrado, y caché “on board”, memoria caché en la placa de circuito impreso pero fuera del circuito integrado, evidentemente, por razones físicas, la primera es mucho más rápida que la segunda.

En el caso de la Arquitectura Harvard Las instrucciones y los datos se almacenan en cachés separadas para mejorar el rendimiento. Por otro lado, tiene el inconveniente de tener que dividir la cantidad de caché entre los dos, por lo que funciona mejor sólo cuando la frecuencia de lectura de instrucciones y de datos es aproximadamente la misma.

**- Memoria central o principal**: En este nivel residen los programas y los datos. La CPU lee y escribe datos en él aunque con menos frecuencia que en los niveles anteriores. Tiene un tiempo de acceso relativamente rápido y gran capacidad. He aquí uno de los inconvenientes de la arquitectura de Harvard, en ella esta memoria principal también se dividirá en memoria de programa y memoria de datos.

**- Extensiones de memoria central**: Son memorias de la misma naturaleza que la memoria central que amplían su capacidad de forma modular. El tiempo de similar, a lo sumo un poco mayor, al de la memoria central y su capacidad puede ser algunas veces mayor. En le modelo de Harvard, esta extensión solo se puede hacer en memoria de datos.

**- Memorias de masas o auxiliares**: Son memorias que residen en dispositivos externos al ordenador, en ellas se archivan programas y datos para su uso posterior. También se usan estas memorias para apoyo de la memoria central en caso de que ésta sea insuficiente (memoria virtual). Estas memorias suelen tener gran capacidad pero pueden llegar a tener un tiempo de acceso muy lento. Dentro de ellas también se pueden establecer varios niveles de jerarquía.

**Clasificación de memorias semiconductoras de acceso aleatorio**

Las memorias se clasifican, por la tecnología empleada y, además según la forma en que se puede modificar su contenido, A este respecto, las memorias se clasifican en dos grandes grupos:

1) **Memorias RAM**: Son memorias en las que se puede leer y escribir, si bien su nombre (Random access memory) no representa correctamente este hecho. Por su tecnología pueden ser de ferritas (ya en desuso) o electrónicas. Dentro de éstas últimas hay memorias estáticas (SRAM, static RAM), cuya célula de memoria está basada en un biestable, y memorias dinámicas (DRAM, dinamic RAM, en las que la célula de memoria es un pequeño condensador cuya carga representa la información almacenada. Las memorias dinámicas necesitan circuitos adicionales de refresco ya que los condensadores tienen muy poca capacidad y, a través de las fugas, la información puede perderse, por otra parte, son de lectura destructiva.

2) **Memorias ROM** (Read 0nly Memory): Son memorias en las que sólo se puede leer. Pueden ser:

a) **ROM** programadas por máscara, cuya información se graba en fábrica y no se puede modificar.

b) **PROM**, o **ROM programable una sola vez**.

c) **EPROM** (erasable PROM) o **RPROM** (reprogramable ROM), cuyo contenido puede borrarse mediante rayos ultravioletas para regrabarlas.

d) **EAROM** (electrically alterable ROM) o **EEROM** (electrically erasable ROM), que son memorias que está en la frontera entre las RAM y las ROM ya que su contenido puede regrabarse por medios eléctricos, estas se diferencian de las RAM en que no son volátiles. En ocasiones a este tipo de memorias también se las denomina **NYRAM** (no volátil RAM).

e) Memoria **FLASH**, denominada así por la velocidad con la que puede reprogramarse, utilizan tecnología de borrado eléctrico al igual que las EEPROM. Las memorias flash pueden borrar-e enteras en unos cuantos segundos, mucho más rápido que las EPROM.

Básicamente las memorias ROM se basan en una matriz de diodos cuya unión se puede destruir aplicando sobre ella una sobretensión (usualmente comprendida ente -12.5 y -40 v.). De fábrica la memoria sale con 1's en todas sus posiciones, para grabarla se rompen las uniones en que se quieran poner 0's. Esta forma de realizar la grabación se denomina **técnica de los fusibles**.

### 1.1.3 Concepto de BUS

En Arquitectura de computadoras, el **bus** es un sistema digital que transfiere datos entre los componentes de un computador o entre computadores. Están formados por cables o pistas en un circuito impreso, dispositivos como resistencias y condensadores además de circuitos integrados.

La función del Bus es la de permitir la conexión lógica entre distintos subsistemas de un sistema digital, enviando datos entre dispositivos de distintos órdenes: desde dentro de los mismos circuitos integrados, hasta equipos digitales completos que forman parte de supercomputadoras.

La mayoría buses están basados en conductores metálicos por los cuales se trasmiten señales eléctricas que son enviadas y recibidas con la ayuda de integrados que poseen una interfaz del bus dado y se encargan de manejar las señales y entregarlas como datos útiles. Las señales digitales que se trasmiten son de datos, de direcciones o señales de control.

Los buses definen su capacidad de acuerdo a la frecuencia máxima de envió y al ancho de los datos. Por lo general estos valores son inversamente proporcionales: si se tiene una alta frecuencia, el ancho de datos debe ser pequeño. Esto se debe a que la interferencia entre las señales (crosstalk) y la dificultad de sincronizarlas, crecen con la frecuencia, de manera que un bus con pocas señales es menos susceptible a esos problemas y puede funcionar a alta velocidad.

Todos los buses de computador tienen funciones especiales como las interrupciones y las DMA que permiten que un dispositivo periférico acceda a una CPU o a la memoria usando el mínimo de recursos.

Hay 3 clases de buses:

- bus de datos, por el que circulan, como el nombre indica, los datos que se transfieren entre los diferentes dispositivos de entrada, salida y almacenamiento, gobernadas por otros dispositivos como los controladores PCI.

- El **bus de dirección** (o direcciones) es un canal del microprocesador totalmente independiente al bus de datos donde se establece la dirección de memoria del dato en tránsito. El bus de dirección consiste en el conjunto de líneas eléctricas necesarias para establecer una dirección. La capacidad de la memoria que se puede direccionar depende de la cantidad de bits que conforman el bus de direcciones, siendo 2^n (dos elevado a la ene) el tamaño máximo en bytes del banco de memoria que se podrá direccionar con n líneas. Por ejemplo, para direccionar una memoria de 256 bytes, son necesarias al menos 8 líneas, pues 2^8 = 256. Adicionalmente pueden ser necesarias líneas de control para señalar cuando la dirección está disponible en el bus. Esto depende del diseño del propio bus.

- y bus de control, Gobierna el uso y acceso a las líneas de datos y de direcciones. Como estas líneas están compartidas por todos los componentes tiene que proveerse de determinados mecanismos que controlen su utilización. Las señales de control transmiten tanto órdenes como información de temporización entre los módulos del sistema.

## 1.2 Estructura lógica del procesador

La operación fundamental de la mayoría de los CPU, es ejecutar una secuencia de instrucciones almacenadas llamadas "programa". El programa es representado por una serie de números que se mantienen en una cierta clase de memoria de computador. Hay cuatro pasos que casi todos los CPU de Eckert-Mauchly usan en su operación: **fetch**, **decode**, **execute**, y **writeback**, (leer, decodificar, ejecutar, y escribir).


Diagrama mostrando como es decodificada una instrucción del MIPS32. (MIPS Technologies 2005)

El primer paso, **leer** (fetch), implica el recuperar una instrucción, (que es representada por un número o una secuencia de números), de la memoria de programa. La localización en la memoria del programa es determinada por un contador de programa (PC), que almacena un número que identifica la posición actual en el programa. En otras palabras, el contador de programa indica al CPU, el lugar de la instrucción en el programa actual. Después de que se lee una instrucción, el PC es incrementado por la longitud de la palabra de instrucción en términos de unidades de memoria.

Frecuentemente la instrucción a ser leída debe ser recuperada de memoria relativamente lenta, haciendo detener al CPU mientras espera que la instrucción sea retornada. Este problema es tratado en procesadores modernos en gran parte por los cachés y las arquitecturas pipeline.

La instrucción que el CPU lee desde la memoria es usada para determinar qué deberá hacer el CPU. En el paso de **decodificación**, la instrucción es dividida en partes que tienen significado para otras porciones del CPU. La manera en que el valor de la instrucción numérica es interpretado está definida por la arquitectura del conjunto de instrucciones (el **ISA**) del CPU. A menudo, un grupo de números en la instrucción, llamado opcode, indica qué operación realizar. Las partes restantes del número usualmente proporcionan información requerida para esa instrucción, como por ejemplo, operandos para una operación de adición. Tales operandos se pueden dar como un valor constante (llamado valor inmediato), o como un lugar para localizar un valor, que según lo determinado por algún modo de dirección, puede ser un registro o una dirección de memoria. En diseños más viejos las porciones del CPU responsables de decodificar la instrucción eran dispositivos de hardware fijos. Sin embargo, en CPUs e ISAs más abstractos y complicados, es frecuentemente usado un microprograma para asistir en traducir instrucciones en varias señales de configuración para el CPU. Este microprograma es a veces re escribible de tal manera que puede ser modificado para cambiar la manera en que el CPU decodifica instrucciones incluso después de que haya sido fabricado.

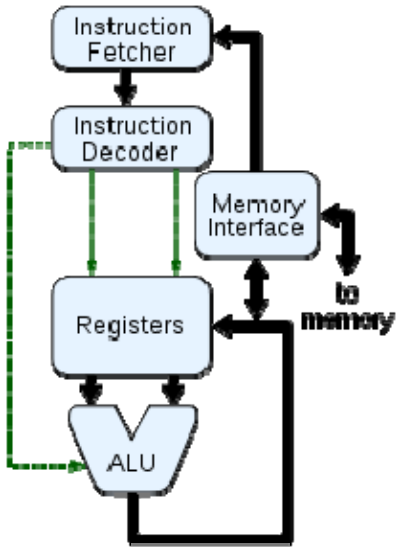



Diagrama de bloques de un CPU simple

Después de los pasos de lectura y decodificación, es llevado a cabo el paso de la **ejecución** de la instrucción. Durante este paso, varias porciones del CPU son conectadas de tal manera que ellas pueden realizar la operación deseada. Si, por ejemplo, una operación de adición fue solicitada, una unidad aritmético lógica (**ALU**) será conectada a un conjunto de entradas y un conjunto de salidas. Las entradas proporcionan los números a ser sumados, y las salidas contendrán la suma final. El ALU contiene la circuitería para realizar operaciones simples de aritmética y lógica en las entradas,

como adición y operaciones de bits (bitwise). Si la operación de adición produce un resultado demasiado grande para poder ser manejado por el CPU,, también puede ser ajustada una bandera (flag) de desbordamiento aritmético localizada en un registro de banderas (ver abajo la sección sobre rango de números enteros).

El paso final, la **escritura** (writeback), simplemente "escribe" los resultados del paso de ejecución a una cierta forma de memoria. Muy a menudo, los resultados son escritos a algún registro interno del CPU para acceso rápido por subsecuentes instrucciones. En otros casos los resultados pueden ser escritos a una memoria principal más lenta pero más barata y más grande. Algunos tipos de instrucciones manipulan el contador de programa en lugar de directamente producir datos de resultado. Éstas son llamadas generalmente "saltos" (jumps) y facilitan comportamientos como bucles (loops), la ejecución condicional de programas (con el uso de saltos condicionales), y funciones en programas. Muchas instrucciones también cambiarán el estado de dígitos en un registro de "banderas". Estas banderas pueden ser usadas para influenciar cómo se comporta un programa, puesto que a menudo indican el resultado de varias operaciones. Por ejemplo, un tipo de instrucción de "comparación" considera dos valores y fija un número, en el registro de banderas, de acuerdo a cual es el mayor. Entonces, esta bandera puede ser usada por una posterior instrucción de salto para determinar el flujo de programa.

Después de la ejecución de la instrucción y la escritura de los datos resultantes, el proceso entero se repite con el siguiente ciclo de instrucción, normalmente leyendo la siguiente instrucción en secuencia debido al valor incrementado en el contador de programa. Si la instrucción completada era un salto, el contador de programa será modificado para contener la dirección de la instrucción a la cual se saltó, y la ejecución del programa continúa normalmente. En CPUs más complejos que el descrito aquí, múltiples instrucciones pueden ser leídas, decodificadas, y ejecutadas simultáneamente. Esta sección describe lo que es referido generalmente como el "entubado RISC clásico" (Classic RISC pipeline).

1.3 Lenguaje Maquina

El **lenguaje ensamblador** es un tipo de lenguaje de bajo nivel utilizado para escribir programas informáticos, y constituye la representación más directa del código máquina específico para cada arquitectura de computadoras legible por un programador.

Fue usado ampliamente en el pasado para el desarrollo de software, pero actualmente sólo se utiliza en contadas ocasiones, especialmente cuando se requiere la manipulación directa del hardware o se pretenden rendimientos inusuales de los equipos.

Programar en lenguaje ensamblador es difícil de aprender, entender, leer, escribir, depurar y mantener, por eso surgió la necesidad de los lenguajes compilados. A pesar de perder rendimiento en un proceso de compilación, en la actualidad la mayoría de las computadoras son suficientemente rápidas.

El lenguaje ensamblador no es portable.

Programar en lenguaje ensamblador lleva mucho tiempo.

Los programas hechos en lenguaje ensamblador son generalmente más rápidos. Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas de 5 a 100 veces más rápidos que con lenguajes de alto nivel.

Los programas hechos en lenguaje ensamblador generalmente ocupan menos espacio. Un buen programa en lenguaje ensamblador puede ocupar casi la mitad de espacio que su contrapartida en lenguaje de alto nivel.

Con el lenguaje ensamblador se pueden crear segmentos de código imposibles de formar en un lenguaje de alto nivel.

Un programa escrito en lenguaje ensamblador consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables que pueden ser cargadas en la memoria de una computadora. Por ejemplo, un microcontrolador hc12 puede ejecutar la siguiente instrucción binaria como se expresa en código de máquina:

- Binario: 10000110 01100001 (Hexadecimal: 0x8661)

La representación equivalente en lenguaje ensamblador es más fácil de recordar:

- ldaa #$61

Esta instrucción significa:

- Asigna el valor hexadecimal 61 (97 decimal) al registro "A".

El mnemónico "ldaa" es un *código de operación* u "opcode", elegido por los diseñadores de la colección de instrucciones para abreviar "load acumulator a" (cargar el acumulador A). El *opcode* es seguido por una lista de argumentos o *parámetros*, completando una instrucción de ensamblador típica.

La transformación del lenguaje ensamblador en código máquina la realiza un programa ensamblador, y la traducción inversa la puede efectuar un desensamblador. A diferencia de los lenguajes de alto nivel, aquí hay usualmente una correspondencia 1 a 1 entre las instrucciones simples del ensamblador y el lenguaje de máquina. Sin embargo, en algunos casos, un ensamblador puede proveer "pseudo instrucciones" que se expanden en un código de máquina más extenso a fin de proveer la funcionalidad necesaria. Los ensambladores más completos también proveen un rico lenguaje de macros que se utiliza para generar código más complejo y secuencias de datos.

Cada arquitectura de computadoras tiene su propio lenguaje de máquina, y en consecuencia su propio lenguaje ensamblador. Los ordenadores difieren en el tipo y número de operaciones que soportan; también pueden tener diferente cantidad de registros, y distinta representación de los tipos de datos en memoria. Aunque la mayoría de las computadoras son capaces de cumplir

esencialmente las mismas funciones, la forma en que lo hacen difiere y los respectivos lenguajes ensamblador reflejan tal diferencia.

Pueden existir múltiples conjuntos de mnemónicos o sintaxis de lenguaje ensamblador para un mismo conjunto de instrucciones, instanciados típicamente en diferentes programas ensamblador. En estos casos, la alternativa más popular es la provista por los fabricantes, y usada en los manuales del programa.

El código máquina, o lenguaje de máquina, está formado por instrucciones sencillas, que -dependiendo de la estructura del procesador- pueden especificar:

- Registros específicos para operaciones aritméticas, direccionamiento o control de funciones.
- Posiciones de memoria específicas (*offset*).
- Modos de direccionamiento usados para interpretar operandos.

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

Las operaciones disponibles en la mayoría de los conjuntos de instrucciones incluye:

- mover
    - llenar un registro con un valor constante
    - mover datos de una posición de memoria a un registro o viceversa
    - escribir y leer datos de dispositivos
- computar
    - sumar, restar, multiplicar o dividir los valores de dos registros, colocando el resultado en uno de ellos o en otro registro
    - realizar operaciones binarias, incluyendo operaciones lógicas (AND/OR/XOR/NOT)
    - comparar valores entre registros (mayor, menor, igual)
- afectar el flujo del programa
    - saltar a otra posición en el programa y ejecutar instrucciones allí
    - saltar si se cumplen ciertas condiciones (IF)
    - saltar a otra posición, pero guardar el punto de salida para retornar (CALL, llamada a subrutinas)

Algunas computadoras incluyen instrucciones complejas dentro de sus capacidades. Una sola instrucción compleja hace lo mismo que en otras computadoras puede requerir una larga serie de instrucciones, por ejemplo:

- salvar varios registros en la pila de una sola vez
- mover grandes bloques de memoria
- operaciones aritméticas complejas o de punto flotante (seno, coseno, raíz cuadrada)

El nivel de lenguaje ensamblador tiene aspectos importantes de los niveles de microarquitectura, en los cuales se encuentra (ISA y sistema operativo) estos dos se utilizan para la traducción en lugar de la interpretación. Algunas características del lenguaje se describen a continuación Los programas que sirven para traducir algún programa para el usuario se llama traductores, el lenguaje en que esta escrito el programa original se llama lenguaje fuente, el lenguaje original que sea modificado se llama lenguaje objeto.

Se usa la traducción cuando se cuenta con un procesador (ya sea hardware o un interprete) para el lenguaje objeto pero no para el lenguaje fuente, Si la traducción se realiza correctamente, la ejecución del programa traducido dará exactamente los mismos resultados que habría dado la ejecución del programa fuente. Hay dos diferencias entre traducción e interpretación, en la traducción no se ejecuta directamente el programa original, en el lenguaje fuente se convierte en un programa equivalente llamado programa objeto o programa binario ejecutable y este funciona solo cuando se ha acabado la traducción.

El código máquina, un simple patrón de bits, es hecho legible reemplazando *valores crudos* por símbolos denominados mnemónicos. Se inventó para facilitar la tarea de los primeros programadores que hasta ese momento tenían que escribir directamente en código binario. Inicialmente el código de ceros y unos (el programa) debía introducirse en una tarjeta perforada. La posición ocupada por cada punto equivalía a un "1" o a un "0" según hubiera o no una perforación. Lo cual suponía una forma casi idéntica en la que hoy se escriben los datos binaros en soportes tales como los CDs y DVDs.

Mientras que una computadora reconoce la instrucción de máquina 8661H

10000110 01100001

para los programadores de microprocesadores HC12 es mucho más fácil reconocer dicha instrucción empleando lenguaje ensamblador :

ldaa #$61

Cada instrucción de la máquina se transforma en una única instrucción en código simbólico.

Pero además, para mejorar la legibilidad del programa, el código simbólico introduce instrucciones adicionales, que no corresponden a ninguna instrucción de la máquina y que proporcionan información. Se llaman "pseudoinstrucciones".

El código simbólico puede parecer de difícil acceso, pero es más fácil de recordar e interpretar que el binario o el hexadecimal.

Los lenguajes simbólicos no resuelven definitivamente el problema de cómo programar un ordenador de la manera más sencilla posible. Para utilizarlos, hay que conocer a fondo el microprocesador, los registros de trabajo de que dispone, la estructura de la memoria, y muchas cosas más.

Además, el lenguaje ensamblador está muy ligado al microprocesador para que sea posible escribir programas independientes de la máquina en que van a ejecutarse.

Este **código simbólico** no puede ser ejecutado directamente por un ordenador, por lo que es preciso traducirlo previamente. Pero la traducción es un proceso mecánico y repetitivo, que se presta a su realización por un programa de ordenador.

Los programas que traducen código simbólico al lenguaje de máquina se llaman ensambladores ("assembler", en inglés), porque son capaces de ensamblar el programa traducido a partir de varias piezas, procedimientos o subrutinas a código binario ("1" y "0") que entiende el procesador.

## 1.4 Conjunto de registros y modelo de programación.

Un modelo de programación provee un marco abstracto en el que entender la sintaxis de los lenguajes concretos que siguen el modelo, y representa la semántica del mismo, así como una filosofía de uso de dicha semántica.

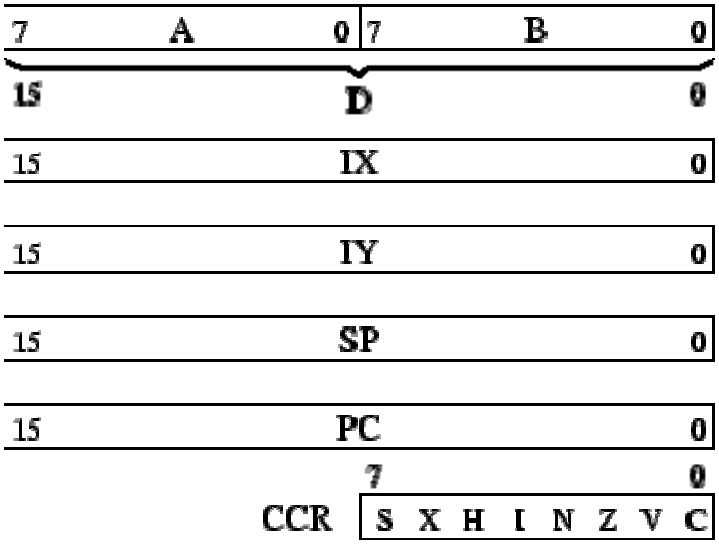
Como ya se menciono anteriormente, el lenguaje ensamblador es específico para cada microprocesador o Microcontrolador, nosotros estudiaremos el modelo de programación del CPU12 de FREESCALE. Su modelo de programación incluye los registros internos que permiten la manipulación de datos, el conjunto de registros, los modos de direccionamiento y el conjunto de instrucciones.

### 1.4.1 Conceptos de modelo de programación

Es una CPU de alta velocidad de 16 bits. Su modelo de programación es idéntico al del MC68HC11. Su repertorio de instrucciones en un superconjunto del de 68HC11

**Registros**

Los registros disponibles son los siguientes:




**Acumuladores A, B y D**

Los acumuladores A y B son registros generales de 8 bits

Salvo en contadas instrucciones, son equivalentes. La suma (ABA), resta (SBA) y comparación (CBA) entre ellos sólo es en una dirección. El ajuste a BCD (DAA) sólo es posible en el acumulador A.

Unidos forman el acumulador D=A:B

**Registros Índices X e Y**

Registros de 16 bits usados para los numerosos direccionamientos indexados.

**Puntero de pila (SP)**

Registro de 16 bits que permite mantener una pila del sistema que se actualiza automáticamente al meter y sacar datos.

Crece hacia abajo en memoria (RAM). SP en cada momento contienen la dirección del primer B libre por debajo de la cabeza.

Debe hacerse apuntar a memoria RAM al iniciarse el sistema.

Se utiliza para: llamadas a subrutinas, guardar estado en interrupciones, salvar datos temporalmente, etc.

**Contador de programa (PC)**

Registro de 16 bits que contiene la dirección de la siguiente instrucción a ejecutar.

**Registro de códigos de condición (CCR)**

Registro de 8 bits que contiene distintos indicadores (banderines) (*flags*):

De control del sistema:

**S**

Inhibición de la instrucción STOP : 1-se ejecuta como NOP, 0-se detiene CPU

**X**

Inhibición de interrupciones no enmascarables (línea ): 1-no se permiten, 0-se permiten

**I**

Inhibición de interrupciones enmascarables: 1-no se permiten, 0-se permiten

de resultados aritméticos:

**H**

medio acarreo: usado para ajustar resultado tras suma con datos en BCD

**N**

negativo: si el bit más significativo está a 1 => dato negativo si está representado en complemento a 2.

**Z**

cero: resultado es 0

**V**

overflow: desbordamiento de una operación si los datos están en complemento a 2

**C**

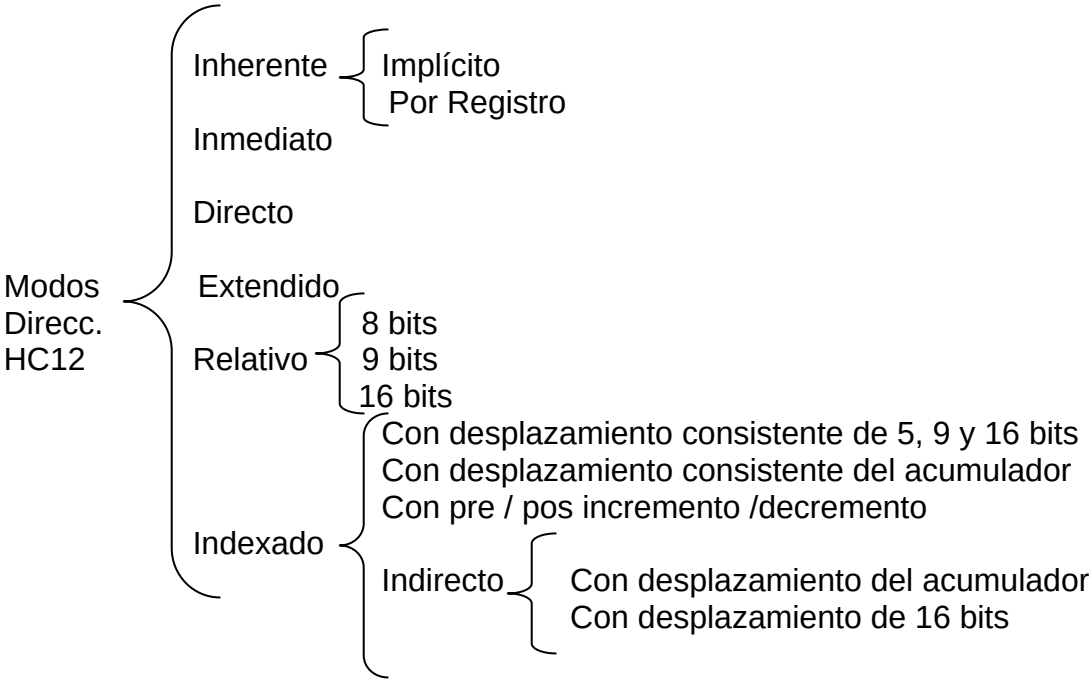
acarreo: desbordamiento de una operación si los datos son sin signo. También se utiliza en las operaciones de desplazamiento y rotación.

Existen instrucciones de salto condicional que permiten comprobar las condiciones < , , =, , O para datos sin signo y con signo.

La mayoría de las instrucciones modifican el CCR. Las excepciones están en las instrucciones de : meter y sacar de la pila, transferencias e intercambios.

## Modos de Direccionamiento microcontrolador 68HC12

En el lenguaje ensamblador, asi como en otros lenguajes de programación, hay diversas formas de acceder a los datos que se pretenden usar en una instrucción, a estas formas se les llamará modos de direccionamiento. Estos tienen por fin, el ayudar a que el usuario del micro pueda realizar programas más flexibles y dinámicos, dependiendo de las exigencias.




Los modos de direccionamiento son la política a seguir por el Microcontrolador o microprocesador a la hora de buscar un operando 'externo' a la instrucción que va a procesar, es decir, encontrar la posición de memoria donde se encuentra el operando. Dicha dirección la denominaremos, de ahora en adelante, EA (Effective Addres, Dirección Efectiva). Excepcionalmente, el modo inherente no tienen operandos 'externos', es decir, no tienen los datos del operando en memoria, sino que se encuentra en los registros de la CPU (Inherente Implícito) o incluido en el contexto de la instrucción (Inherente por registro).

El microcontrolador 68HC12 soporta los siguientes modos de direccionamiento:

| Modo de direccionamiento | Formato fuente | Abreviación |
|---|---|---|
| Inherente | INST | INH |
| Inmediato | INST #opr8i<br>ó<br>INST #opr16i | INM |
| Directo | INST opr8a | DIR |
| Extendido | INST opr16a | EXT |
| Relativo | INST rel8<br>ó<br>INST rel16 | REL |
| Indexado de 5 bits | INST oprx5,xysp | IDX |
| Indexado de 9 bits | INST oprx9,xysp | IDX1 |
| Indexado de 16 bits | INST oprx16,xysp | IDX2 |
| Indexado de acumulador | INST abd,xysp | IDX |
| Indexado de pre incremento | INST oprx3,+xys | IDX |

| | | |
|---|---|---|
| Indexado de pre decremento | INST oprx3,-xys | IDX |
| Indexado de post incremento | INST oprx3,xys+ | IDX |
| Indexado de post decremento | INST oprx3,xys- | IDX |
| Indexado indirecto de 16 bits | INST [oprx16,xysp] | [IDX2] |
| Indexado de acumulador indirecto | INST [D,xysp] | [D,IDX] |

1.4.2 Direccionamientos simples. (Inherente, Inmediato, Directo y Extendido)

**Direccionamiento inherente (INH)**

Las instrucciones de este tipo no tienen una dirección efectiva por que no guardan datos en una dirección de memoria, solo usan la memoria para obtener el código maquina.

Existen 2 tipos de direccionamientos inherentes, los implícitos que no requieren operandos:

**CONLOC COP <etiqueta> INST ; datos en la instrucción**

COP es el código de operación de la instrucción que lo documenta el fabricante el conjunto de instrucciones de las hojas de datos del microprocesador o Microcontrolador (en este caso archivo CPU12RG.pdf).

Y CONLOC es la dirección de memoria que le corresponde a la instrucción.

Y los inherentes por registro, que si llevan operando pero hacen referencia únicamente a registros internos, en otras palabras no ocupan de la memoria:

**CONLOC COP eb <etiqueta> INST r1,r2 ; por registro**

En donde r1 y r2 significan registro 1 y 2 (respectivamente); un ejemplo de esto podría ser la siguiente instrucción en lenguaje ensamblador:

**tfr A,B**

En la cual se indica que se copie lo que contiene el acumulador A en el acumulador B. Otro ejemplo podría ser el siguiente:

**exg X,S**

El cual dice que se cambie lo que contenga el registro X en el registro S.

**Direccionamiento inmediato (INM)**

Se usa cuando se especifica que se quiere asignar específicamente un valor, el cual puede ser tanto de 8 bits como de 16 bits. La EA se encuentra inmediatamente después del código de operación.

**CONLOC COP ii <etiqueta> INST #opr8i**

**CONLOC COP jj kk <etiqueta> INST #opr16i**

Con 0 ≤ opr8i ≤ 255 para el de 8 bits, y 0 ≤ opr16i ≤ 65535 para el de 16 bits. Donde ii es igual a opr8i en formato hexadecimal en 1 byte (completando con cero a su izquierda de ser necesario), y jj kk es opr16i en formato hexadecimal completado a 2 bytes.

Por ejemplo, para el siguiente fragmento de programa que se encarga de sumar el número 30 hexadecimal con el 20 también en hexadecimal, el código maquina se calcularía de la siguiente manera:

***0000 87*** **clra ;INH, no existe EA**

***0000 86*** *30* **ldaa #$30 ;IMM, EA = 0001H**

***0002 8B*** *20* **adaa #$20 ;IMM, EA= 0003H**

***0004 CE*** *00 00* **ldx #$00 ;IMM, EA= 0005H**

**Direccionamiento directo** (DIR)

El operando indica la dirección de memoria de donde se leerá el dato, pero la dirección debe estar en la pagina cero (entre 0000H y 00FFH). La EA = 00dd.

**CONLOC COP dd <etiqueta> INST opr8a**

Donde 0 ≤ opr8a ≤ 255. Con dd=opr8a en formato hexadecimal de 1 byte. Suponiendo que deseamos conocer el código maquina del siguiente extracto de programa que se encarga de sumar 2 números, tenemos:

*0000 87* **clra ; EA no existe**

*0001 86 80* **ldaa $80 ; EA= 0080H**

*0003 8B 81* **adda $81 ; EA=0081H**

*0005 5A 82* **staa $82 ; EA=0082H**

**Direccionamiento extendido (EXT)**

Igual que en el direccionamiento directo el operando especifica la dirección de memoria, con la diferencia de que se puede especificar cualquier dirección de memoria. La EA=hhll.

**CONLOC COP hh ll <etiqueta> INST opr16a**

Con 0 ≤ opr16a ≤ 65535. Donde hh ll = opr16 en formato hexadecimal en 2 bytes.

1.4.3 Direccionamiento Relativo de 8 bits

Hay algunos modos de direccionamiento en que se hace uso de una propiedad muy generalizada de los programas denominada localidad de referencia, esta propiedad consiste en que las direcciones referenciadas por los programas no suelen alejarse mucho unas de otras y, por tanto, suelen estar concentradas en una parte de la memoria. Estas consideraciones nos llevan a la conclusión de que no es necesario utilizar todos los bits de la dirección de memoria en el campo de operando, basta utilizar los bits precisos para cubrir la parte de memoria donde estén incluidas las direcciones a las que el programa hace referencia. Esto puede hacerse tomando corno referencia un punto de la memoria y tomando como campo de operando la diferencia entre ese punto y la dirección efectiva del operando. La dirección que se toma como punto de referencia puede residir en un registro de la CPU y, por tanto, sumando el contenido de ese registro con el campo de operando obtendremos la dirección efectiva. Hay varios direccionamientos basados en esta técnica que reciben diferentes nombres dependiendo de cuál sea el registro en el que radica la dirección tomada como referencia. Todos ellos podrían catalogarse como direccionamientos relativos a un registro.

El direccionamiento denominado habitualmente relativo toma como valor de referencia el registro contador de programa. Cuando se usa este modo de direccionamiento, el campo de operando consiste en un número (normalmente con signo) que expresa la diferencia entre la dirección del dato y la dirección siguiente a la instrucción en curso (contenida en el contador de programa). Si el campo de operando, llamado en este caso desplazamiento u offset, es positivo el operando residirá en una dirección posterior a la de la instrucción y si es negativo, en una dirección anterior.

Este modo de direccionamiento es usado muy frecuentemente en programas cuyo código deba ser independiente de la posición de memoria donde estén situados (programas relocalizables) ya que el desplazamiento es independiente de la localización del programa. También se usa con mucha frecuencia en instrucciones de bifurcación.

La EA es relativa al valor del PC, son saltos cuya dirección de destino estará antes o después del PC. En el relativo de 8 bits la dirección efectiva estará 128 posiciones antes del PC o 127 posiciones

después y se especifica en el operando por medio de rel8 (EA=rel8). EA= PC + rr en donde rr es un numero de 8 bits con signo, mas adelante se explica como calcularlo.

**Direccionamiento relativo (REL)**

Son instrucciones de salto condicional.

**CONLOC COP rr** **<etiqueta>** **INST** **rel8**

**CONLOC COP eb rr** **<etiqueta>** **INST** **abdxys,rel9**

**CONLOC COP qq rr** **<etiqueta>** **INST** **rel16**

Dado que el PC es el registro que especifica la dirección de memoria de la siguiente instrucción, entonces podemos considerar que:

PC = CONLOC + N_bytes

En donde N_bytes es el número de bytes que tiene el código maquina de la instrucción, o visto de otra forma la dirección de memoria de la instrucción siguiente (CONLOC_siguiente).

Entonces:

PC = CONLOC_siguiente.

Y como:

EA= PC + rr

Pero también EA= rel8, sustituyendo tenemos

rel8=CONLOC_siguiente + rr

si despejamos para rr:

rr= rel8 - CONLOC_siguiente

Recordemos que rel8 es el operando de la instrucción y que -128 ≤ rr ≤ 127. Además rr es un solo byte.

1.4.4 Direccionamiento Relativo de 16 bits

Donde -32768 ≤ qq rr ≤ 32767.

En este caso la dirección efectiva es relativa al contador del programa y se calcula de la siguiente manera:

**EA = PC + qq rr**

Con:

**qq rr = rel16 - CONLOC_siguiente**

Por ejemplo, si quisiéramos calcular el código maquina del siguiente programa que se encarga de convertir un dígito o una letra de la A a la F (solo mayúsculas) a su valor en ASCII, tendríamos:

| | | | | |
|---|---|---|---|---|
| *0000* | | | **ORG** | **$4000** |
| *4000* | *96* 00 | | **ldaa** | **$00** |
| *4002* | *80* 00 | | **suba** | **#$30** |
| *4004* | *81 30* | | **cmpa** | **#$11** |
| *4006* | *18 25 00 02* | | **lblo** | **OTRO** |
| *400A* | *80 07* | | **suba** | **$80** |
| *400C* | *5ª 80* | **OTRO** | **staa** | **$80** |
| *400E* | *20 E0* | | **bra** | **$4000** |
| *4010* | | | **END** | |

El relativo de 9 bits por calcularse de acuerdo al valor de cada bit se explicara posteriormente.

1.4.5 Direccionamiento Indexado.

En este modo de direccionamiento, la dirección del operando también se calcula sumando un registro de la CPU al campo de operando, este registro es un registro específico para este uso llamado registro índice. En los ordenadores con organización de registros generales, el registro índice puede ser cualquiera de los registros de la CPU. En los ordenadores en que el contador de programa es considerado como un registro de uso general el modo relativo es un caso particular del direccionamiento indexado. A la cantidad que hay que sumar al registro índice para conseguir la dirección del operando también se le llama desplazamiento u offset. Este modo de direccionamiento es especialmente útil para el direccionamiento de vectores y matrices en bucles ya que, si se quieren direccionar elementos consecutivos del vector o matriz, basta mantener en el desplazamiento la dirección del primer elemento e ir incrementando el registro índice. También sirve para acceder de forma relativa a elementos de vectores cercanos a uno dado, para ello, se carga la dirección del elemento de referencia en el registro índice y después se accede mediante direccionamiento indexado, con el desplazamiento adecuado, al anterior, al siguiente, etc., esto mismo también es aplicable a pilas, en que, en ocasiones, hay que acceder a datos cercanos, por encima o por debajo, al dato señalado por algún apuntador. Una consecuencia de todo esto es una modalidad de direccionamiento indexado de que disponen algunos ordenadores, denominada autoindexación, que hace que el registro índice sea incrementado o decrementado en el tamaño del operando antes o después de acceder al mismo. Los ordenadores que poseen autoindexación

incorporan los modos de direccionamiento descritos en los dos apartados siguientes. En algunos ordenadores existen variantes del direccionamiento indexado en que se obtiene la dirección del operando sumando el contenido de varios registros con el desplazamiento, esto puede servir para especificar el comienzo de un vector mediante un desplazamiento respecto a un registro y el elemento del vector mediante un registro índice.

En este direccionamiento la dirección del dato de la instrucción se obtiene sumando el primer operando al contenido de un registro índice que puede ser X, Y, SP o PC especificado en el segundo operando. EA = ri + desplazamiento, en donde ri es uno de los registros índice.

**Direccionamiento indexado con desplazamiento constante**

En este direccionamiento el desplazamiento (primer operando) es un numero constante de 5, 9 o 16 bits.

**CONLOC COP xb <etiqueta> INST oprx5,xysp**

**CONLOC COP xb ff <etiqueta> INST oprx9,xysp**

**CONLOC COP xb ee ff <etiqueta> INST oprx16,xysp**

Donde -16 ≤ oprx5 ≤ 15, -256 ≤ oprx9 ≤ 255 y 0 ≤ oprx16 ≤ 65535;

el post byte xb se obtiene de la siguiente manera para el modo de 5 bits:

**xb = rr0nnnnn como numero binario.**

No debemos de confundir esta rr con la obtenida para el modo de direccionamiento relativo, aquí la rr es un numero binario de dos bits que se refiere al registro índice que se señala en el operando y que puede ser:

**x 00**

**y 01**

**sp 10**

**pc 11**

Y nnnnn se refiere a oprx5, completados a 5 bits. Una vez que se tienen todos los datos y se sustituyen en la formula se toman los 4 bits más significativos y se transforman a su equivalente en formato hexadecimal, lo mismo se hace con los 4 bits menos significativos, los 2 dígitos hexadecimales se juntan y ellos representan el post byte xb.

Para los modos indizados de 9 y 16 bits el post byte xb se obtiene con la formula:

**xb = 111rr0zs binario**

Z es 0 para el modo de 9 bits y 1 para el modo de 16 bits; s es 0 para el modo de 16 bits y para el modo de 9 bits es igual al bit de signo de la representación binaria de oprx9.

Para el modo indizado de 9 bits, ff se refiere a la representación en 1 byte hexadecimales de los 8 bits menos significativos de oprx9; y para el modo indizado de 16 bits ee es la representación en 1 byte de los 8 bits más significativos de oprx16 y ff la representación en hexadecimal de los 8 bits menos significativos del mismo oprx16.

He aquí un ejemplo:

```
0000                    ORG    $4300
4300   CE 08 00         ldx    #$0800
4303   69 00    LIMPIA clr    0,x
4305   08               inx
4306   8E 09 00         cpx    #$0900
4309   25 F8            blo    LIMPIA
430B   3D               rts
430C                    END
```

**Direccionamiento indexado con desplazamiento del acumulador**

En este direccionamiento el desplazamiento es uno de los registros acumuladores (A, B o D). EA = ri + acumulador.

**CONLOC COP xb  <etiqueta>  INST  abd,xysp**

Donde el post byte xb se calcula con la siguiente fórmula:

**xb = 111rr1aa** también en binario.

El acumulador es el que nos va a dar aa, ya que si este es el acumulador A, aa será igual a 00, si por otra parte es el acumulador B será 01 y por ultimo si se trata del acumulador D, aa será 10.

Veamos pues un ejemplo, el cual convierte un número hexadecimal a su representación en BCD:

```
0000                    ORG    $0E43
0E43   CE 0E 4E         ldx    #TABLA
0E46   96 00    OTRO   ldaa   $00
0E48   84 0F            anda   #$0F
```

```
0E4A  A6 E4          ldaa  A,X
0E4C  20 F8          bra   OTRO
0E4E  00 01 02 03    TABLA DB    %0000, %0001, $02, $03
0E52  04 05 06 07          DB    $04, $05, $06, $07
0E56  08 09 10 11          DB    $08, $09, $10, $11
0E5A  12 13 14 15          DB    $12, $13, $14, $15
0E5E                       END
```

**Indexado con pre/post incremento/decremento**

En este direccionamiento a diferencia de los demás indexados la dirección efectiva se obtiene solo del contenido del registro índice pero este valor se calcula después de incrementar o decrementar el contenido del registro índice, si se trata de pre incremento o decremento respectivamente. Si se trata de post incremento o decremento, primero se calcula la dirección efectiva y luego se incrementa o decrementa el registro índice. El valor en que va a ser incrementado o decrementado el registro índice se especifica en el primer operando y puede ser un valor entre 1 y 8.

```
CONLOC COP xb    <etiqueta>   INST   oprx3,+xys ; pre-incremento
CONLOC COP xb    <etiqueta>   INST   oprx3,-xys ; pre-decremento
CONLOC COP xb    <etiqueta>   INST   oprx3,xys+ ; post-incremento
CONLOC COP xb    <etiqueta>   INST   oprx3,xys- ; post-decremento
```

Esos son los formatos para el pre incremento, pre decremento, post incremento y post decremento respectivamente, donde $1 \leq oprx3 \leq 8$. Donde el post byte xb se obtiene con la formula:

**xb = rr1pnnnn binario como todos los xb**

Como ya hemos visto, rr depende del registro índice al cual se haga referencia en el mismo operando; p es 0 para pre y 1 para post, y por ultimo nnnn representa oprx3 – 1, si se trata de un incremento o –oprx3 para decremento en un formato de 4 bits.

Esto se aprecia mejor en el siguiente ejemplo:

```
0000                ORG   $E00
0E00 CE 08 00       ldx   #$0800
```

*0E03 69* 30 **LIMPIA clr 1,x+**

*0E05 8E 09* 00 **cpx #$0900**

*0E08 25 F9* **blo LIMPIA**

*0E0A 3D* **rts**

*0E0B* **END**

1.4.6 Direccionamiento Indirecto

**Direccionamiento Indirecto por Registro**

Este direccionamiento utiliza el contenido de registros internos como una dirección de memoria. La dirección efectiva es igual al contenido del registro índice, este registro trabaja como apuntador, en el HC12 se puede obtener con el direccionamiento indexado de desplazamiento constante de 5 bits haciendo el desplazamiento igual a 0.

**Direccionamiento Indirecto por Memoria.**

En este otro direccionamiento indirecto es una localidad de memoria la que contendrá el valor de la dirección efectiva, dentro de la memoria se encuentra el apuntador a otra dirección de memoria, en el caso del HC12 existe combinado con el direccionamiento indexado.

**Direccionamiento indexado indirecto**

Existen 2 tipos de direccionamiento indexado indirecto, el de 16 bits:

-Direccionamiento indexado indirecto de 16 bits de desplazamiento constante

Este modo de direccionamiento indexado añade a los 16 bits de la instrucción de desplazamiento a la base del registro índice para formar la dirección de la localidad de memoria que contiene un puntero a la memoria afectada por la instrucción. La instrucción misma no es al puntero a la dirección de memoria la que actúa, mejor dicho a la localidad de memoria del puntero a la dirección a actuar.

**CONLOC COP xb ee ff <etiqueta> INST [oprx16,xysp]**

Donde xb se puede obtener con esta fórmula:

**xb = 111rr011 binario**

Y en la cual rr se refiere al registro índice; y ee ff es oprx16 representado en 2 bytes.

Y también existe el direccionamiento indexado indirecto del acumulador D:

Este direccionamiento indexado añade el valor en el acumulador D al valor en la base del registro índice para formar la dirección de memoria que contiene el puntero de la localidad de memoria afectada por la instrucción.

**<etiqueta>  INST  [D,xysp]**

Y para el cual se calcula su código maquina así:

**COP  xb  [D,xysp]**

Donde xb viene de la siguiente fórmula:

**111rr111**

En la cual (y como hemos visto anteriormente) el post byte xb viene dado por el registro índice.

**1.4.7 Otros direccionamientos**

**Direccionamientos Múltiples**

| | | | |
|---|---|---|---|
| COP dd mm | BCLR | opr8a,msk8 | DIR |
| hh ll mm | | opr16a,msk8 | EXT |
| xb mm | | oprx0_xysp,msk8 | IDX |
| xb ff mm | | oprx9_xysp,msk8 | IDX1 |
| xb ef ff | | oprx16_xysp,msk8 | IDX2 |

| | | | |
|---|---|---|---|
| COP dd mm rr | BRCLR | opr8a,msk8,rel8 | |
| hh ll mm r | | opr16a,msk8,rel8 | |
| xb mm rr | | oprx0_xysp,msk8,rel8 | |
| xb ff mm rr | | oprx9_xysp,msk8,rel8 | |
| xb ef ff rr | | oprx16_xysp,msk8,rel8 | |

| | | | |
|---|---|---|---|
| COP ii hh ll | MOVB | #opr8i,opr16a | INM-EXT |
| xb ii | | #opr8i,oprx0_xysp | INM-IDX |
| hh ll hh ll | | opr16a,opr16a | EXT-EXT |

| | | |
|---|---|---|
| xb hh ll | opr16a,oprx0_xysp | EXT-IDX |
| xb hh ll | oprx0_xysp,opr16a | IDX-EXT |
| xb xb | oprx0_xysp,oprx0_xyps | IDX-IDX |

COP jj kk hh ll MOVW #opr16ii,opr16i

**Ej:**

<Etiqueta> BCLR $01,$10 ⟶ 4D **01 10**<Etiqueta> BCLR $01,$10

<Etiqueta> BRCLR $01, $10,OTRO

5510 4F **01 10 EC.....** <Etiqueta> BRCLR $01,$10,OTRO

5514 **.....** **TABSIM OTRO $5500**

5500 H –

5514 H =

---

FF**EC H**