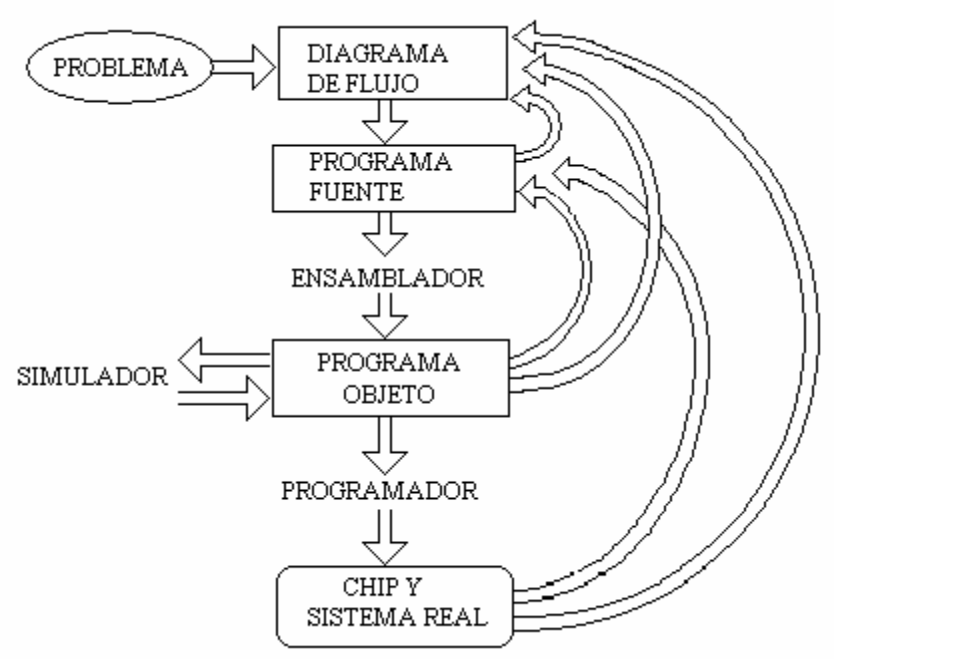
III) PROGRAMACIÓN

El programa es la solución a un problema donde se quiere aplicar el micro.
El diseño de un programa para un micro parte de un diagrama de flujo. Éste expresa la solución a aplicar.
De ahí se extrae el programa fuente, que es el listado de instrucciones.
Si es necesario se vuelve al diagrama y se le modifica.
Del programa fuente se saca el programa objeto, el cual será el cargado en el chip.



El ensamblador (assembler) se encarga de extraer el programa objeto del programa fuente. Y el programador se ocupa de grabar el chip.
El programa objeto antes de ser cargado en el chip, puede ser probado por el simulador. Este es un programa de PC que virtualmente simula las entradas y salidas del PIC siguiendo las instrucciones del programa fuente.
En cualquier punto del proceso se puede volver al diagrama de flujo o a editar el programa fuente.

El programa fuente es solo un archivo de texto muy simple. Se escribe con cualquier procesador de texto (EDIT.COM, WORDPAD, BLOCK de NOTAS, WRITE, WORD, etc.).
Los PICmicro® tienen incluida una interfase serial de programación. Por lo tanto se puede leer y programar con un circuito simple y un PC con puerto paralelo (LPT1). En este curso se usa el más simple encontrado: el NOPPP, de Michael Covington. Que posee varias versiones de hardware y de software.
Se pueden descargar del sitio http://www.covingtoninnovations.com/noppp/index.html.
Su versión para DOS en español tiene fáciles pasos para usarlo.
Este programador es compatible con el software y hardware del TOPIC de David Taite.
Hay una versión del mismo para W95 y W98, pero no funciona para XP, ni ME. Con esos sistemas operativos es necesario usar un disquete de arranque en DOS o W95/98.
El ensamblador que se usa será el suministrado por Microchip. Para MS-DOS es el MPASM.EXE y para W95/98 el MPASMWIN.EXE.
El simulador que funciona en MS-DOS es el SIM84 o el SIMUPIC.
Microchip suministra una herramienta llamada MPLAB que reúne todos estos elementos.

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

El programa es, para el micro, el resultado de estudiar un problema para hallarle solución. Toda situación en donde se aplica un procesador se debe resumir a elementos de entrada y salida. Las salidas son las respuestas del sistema o acciones que desarrolle. Las entradas son las variables que alteran o determinan las respuestas.
Para el procesador las entradas son valores digitales (estas entradas son datos del exterior convertidos por los periféricos para que el procesador las pueda manejar), y las salidas son también valores digitales, pero que los periféricos convierten para que el mundo exterior las interprete.
El programa gobierna las acciones del micro para que éste lea los datos de entrada, los procese y entregue datos de salida.
Varios programas diferentes pueden relacionar de la misma manera los datos de entrada y salida. El proceso se limita por las características físicas del sistema.

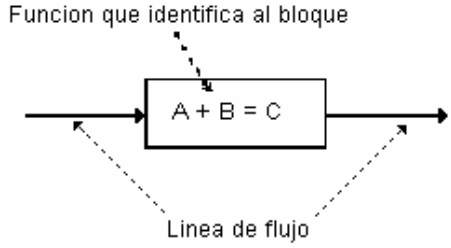
El diagrama de flujo muestra en forma gráfica y de fácil interpretación el proceso desde la adquisición de los datos, pasando por el tratamiento de los mismos, hasta la entrega de del resultado.

Del diagrama de flujo se procede a escribir el programa fuente, el cual es un listado de mnemónicos o instrucciones para el micro, además de información sobre la posición del programa en la memoria del sistema, etc. En él se permite el uso de etiquetas. El programa fuente es una forma que tiene el programador para indicar, en lenguaje entendible por las personas, lo que tiene que hacer el microprocesador.

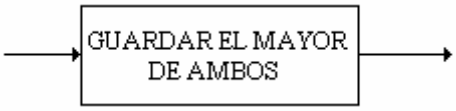
El programa fuente es entregado al ensamblador (Assembler) quien extrae el programa objeto. El programa objeto es el listado del código operativo de las instrucciones del programa fuente. El programa objeto es escrito en la memoria de programa (ROM) del sistema para no desaparecer al apagar la fuente.

Diagrama De Flujo

Son representaciones gráficas de cualquier proceso (industrial, programación, trámite burocrático, detección de fallas, etc.). Interesa a este curso solo el aplicado a los procesadores y su programación.
Los diagramas de flujo se componen por elementos llamados bloques. A cada bloque llega y/o salen líneas que indican la dirección por donde sigue el proceso.
Cada bloque puede tener varias salidas o ninguna, pero una entrada sola o ninguna.
En cada bloque se describe la función que realiza, con que parámetros y sobre que variable la aplica.



Por ejemplo en este bloque, se realiza una suma entre 2 variables. El resultado se carga en una tercera variable. Las líneas de flujo solo indican cual bloque se ejecutara a continuación.
En los bloques no solo pueden ir funciones lógico-matemáticas sino también una descripción del tratamiento que se le hace a los datos.

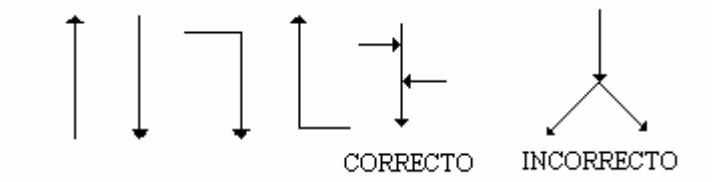


Por ejemplo: Un bloque puede manejar más de un dato de entrada pero solo tiene una línea de llegada de información.
Las líneas de flujo no indican cuantos o cuales datos hay en proceso o son procesados, sino de que bloque salir y a cual entrar. Por eso indican el flujo de información y no el número de datos en proceso.
Todo programa tiene más de un diagrama de flujo que lo represente. Se elige y prioriza que aspectos del programa se desea mostrar.
Se realiza primero un diagrama general donde se muestra una idea del proceso a llevar a cabo. Este diagrama es de fácil entendimiento. Sus bloques representan funciones complejas o son el resumen de varias acciones. De este primero se procede a realizar otro mas detallado donde se representen menos acciones por cada elemento. O sea se convierte cada bloque de funciones complejas en varios elementos con funciones simples, básicas y específicas.
Se debe seguir desglosando hasta que solo queden bloques con funciones básicas que son las que puede realizar el micro.
Esto es personalizar el diagrama de flujo, se realiza para un procesador en particular y se limita por las características físicas del sistema.
Las respuestas, que esperamos nos entregue el sistemas, serán las salidas. El micro las entrega como valores digitales que los periféricos convierten en señales entendibles para el exterior.

ES necesario discernir que elementos del problema son relevantes y alteran las salidas, es decir que se deben considerar para seleccionar las salidas.
Los periféricos se encargan de convertir estos elementos de entrada en valores digitales que los representen.
A los datos de entrada el micro les aplica el proceso que le indica el programa.
Generar la respuesta es entregar datos a los periféricos o calcular un resultado, lo cual constituye el o los elementos de salida.

Símbolos Gráficos

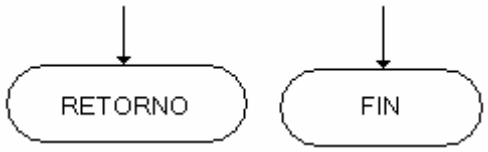
Para realizar los diagramas de flujos se usan símbolos, los cuales tienen su forma según la clase de función que tengan.



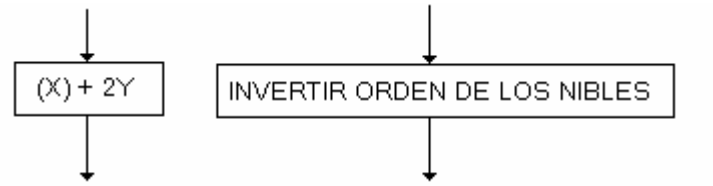
Las flechas son ramas del diagrama de flujo o línea de flujo. Indican de qué bloque salen y a cual entran. Los codos indican saltos en el programa. O sea al salir de un bloque, no se va al inmediato sino que se saltea al menos uno posterior o se vuelve a entrar en otro anterior. Una rama puede confluir en otra pero una rama no puede bifurcarse o abrirse en dos.

Símbolo de inicio de programa o subrutina. En su interior se coloca el nombre del programa o la palabra INICIO. Nótese solo tiene una flecha de salida o partida del proceso. En el caso de inicio representa la salida del estado de RESET (interno o externo) del micro

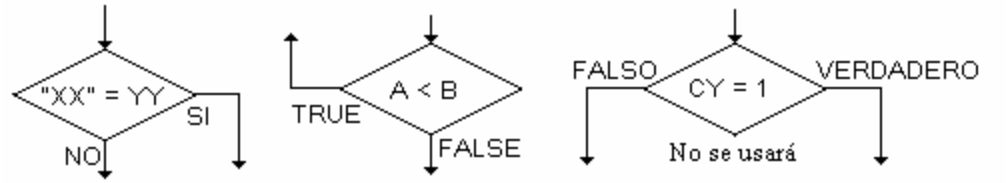
Símbolo de fin de programa o subrutina. Indica que el programa terminó y se detiene el micro, en caso de ser una subrutina indica volver al programa principal. Se le escribe FIN o RETORNO. Nótese que solo tiene flecha de llegada de información.



Bloque de ejecución o proceso. Tiene una flecha de entrada y una de salida. En su interior se escribe la acción que realiza, ya sea la descripción de dicha acción o su representación con símbolos y funciones. Tiene una línea de entrada y una de salida

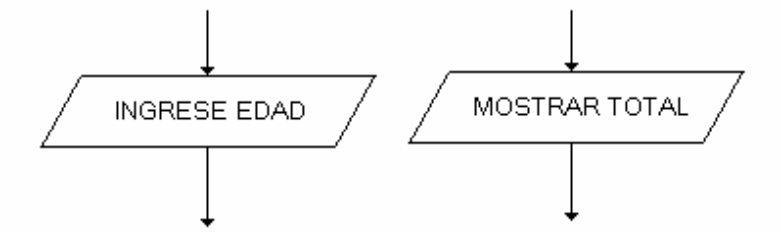


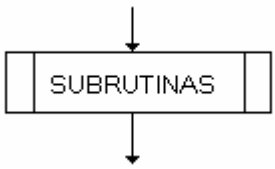
Bloque de decisión o selección. En este elemento no se modifica la información pero se decide en base a un parámetro elegido que trayectoria seguir. Puede tener más de una salida, pero en el uso con



microprocesadores solo tienen dos salidas según si la condición es cierta o falsa, o si se cumple o no. Tiene una entrada y dos salidas representando los posibles resultados. (SI-NO, falso-verdadero, true-false). Siempre se asocia la condición a chequear a alguna operación lógico-matemática. En la práctica se verifica algún flag o bit. Es la única manera de hacer una bifurcación en la información y se le une comúnmente a un salto en el programa.

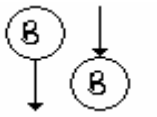
Bloque de ingreso o despliegue de datos. Aquí es donde el programa toma datos desde un periférico o los saca al mundo exterior. En general al trabajar personalizando los programas se sustituye este bloque por otros de ejecución donde se realiza asignaciones de valores o movimientos de datos.



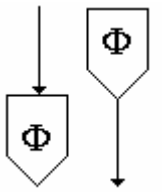


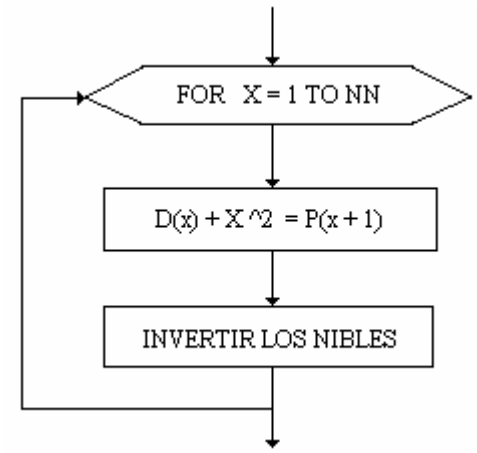
Bloque de llamado de subrutina. Indica que se saltará al comienzo de la subrutina, su nombre se indica dentro del bloque. Al terminar dicha subrutina se continúa con el programa principal. Permite simplificar programas pues se sustituye una rutina de varios elementos por un bloque que la invoca

Conectores dentro de página. Son etiquetas que se colocan para indicar que una rama del diagrama de flujo continúa en otro lado, o proviene de otro lugar, dentro de la misma página. Siempre se dibujan de a pares. El conector de salida es al que llega la flecha y el de entrada es del que sale la rama. Ambos deben tener el mismo símbolo en el interior para saber que se corresponden. Así se evitar dibujar líneas que atraviesen toda la hoja.



Conectores fuera de página. Son etiquetas que indican que la rama del diagrama de flujo continúa en, o proviene de, una pagina diferente. Se dibujan de a pares, uno en cada página. El de partida es al que llega la línea y el de llegada es del que sale la rama. Ambos deben tener el mismo símbolo para saber que están asociados.





Este símbolo FOR IT indica un lazo hasta que se cumpla una condición determinada. Es un bloque complejo que solo se usa con lenguajes de alto nivel. No se usa al programar microprocesadores con assembler. Mientras la variable de lazo no alcance el valor prefijado no se sale del lazo. El valor de la variable de lazo se actualiza con cada pasada, automáticamente.

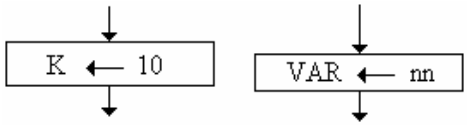
Variables, Etiquetas, Punteros y Vectores

Además de los bloques, se usan elementos de programación que van escritos dentro de ellos.
Variable es un elemento lógico que representa un valor numérico y que puede ser actualizado a lo largo del programa. Esta compuesta de dos partes:

VALOR: es el número que asume y el cual puede variar a lo largo del programa

NOMBRE: cadena de caracteres (sin espacios en blancos ni signos ortográficos) que identifica la variable y es invariable durante todo el programa.

Se les puede dar cualquier nombre pero con cuidado de no repetirlo, ni usar mnemónicos del micro. Se buscan nombres que ayuden a recordar su función. Por ejemplo si se usa como contador se le puede llamar CONT. Puede actualizar su valor mediante una asignación ya sea de un valor numérico, un valor genérico o el resultado de una operación. El uso de un valor genérico *nn* nos permite

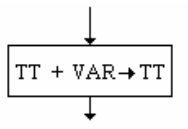


decidir mas tarde que valor tendrá.
Se asocia a la variable con un elemento fisico del micro que tenga identicas propiedades. Primero debe ser un elemento que pueda ser escrito y leido, eso se aplica a la RAM estatica. Para que no altere el funcionamiento del micro se descartan los SFR. Solo quedan los GPR.
Consideremos un registro GPR, este tiene dos partes:

CONTENIDO: valor numérico que almacena y puede variar en cualquier momento.

DIRECCIÓN: valor numérico que indica su posición, esta prefijado por el fabricante y es inamovible.

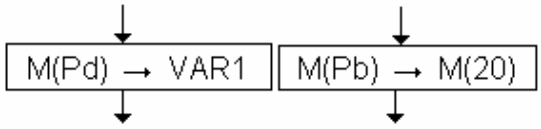
La asociación es: VALOR <=> CONTENIDO y NOMBRE <=> DIRECCIÓN
Es aconsejable al principio del programa asignarle un valor, así no genera ninguna indeterminación posible. Comúnmente con los microprocesadores se puede solo operar con 2 operandos, y el resultado se guardará en uno de ellos . Las variables en deben tener el mismo número de bit que los registros del micro. Los micros más sencillos siempre usan un registro especial (TRABAJO o ACUMULADOR) para suministrar uno de los operandos a las operaciones.



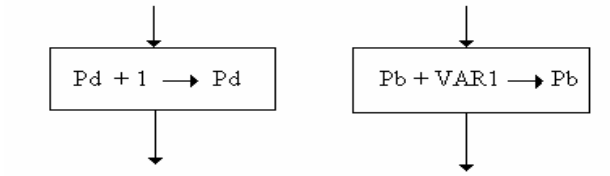
La variable tiene dos niveles. El lógico que es la capacidad de variar su valor pero no su nombre. El físico que es capaz de variar su contenido pero no su posición.
El nombre de la variable se le asocia como etiqueta al valor numérico de la posición de memoria del registro. El valor que le asignamos a la variable es el contenido que se carga en el registro.
Consideremos la variable de nombre VAR1, la asociamos con el registro de RAM de la posición 21h.
Hacer valer E7h a VAR1 es cargar el valor E7h en la posición de memoria 21.
No hay que confundir el valor que se carga en la variable con el valor de posición de memoria que se asocia a la variable.

Puntero o **Vector** es una variable que permite direccionar una posición de memoria o una dirección de puerto. Se usa para direccionar distintas posiciones de memoria con la misma instrucción. O sea almacenar la dirección del operando. Cambiar de operando representa solo modificar el contenido de la variable, pero la instrucción siempre es la misma.

Si tenemos un grupo de valores en posiciones continuas de memoria se les puede agrupar en un conjunto, y llamar a ese grupo serie o matriz. Toda matriz implica un registro que le sirve de puntero para direccionar los elementos del mismo. Así en el programa se trabaja con la matriz, que implica varios valores posibles y se invocan con el puntero.



Su nombre puede ser cualquiera. Comúnmente se le comienza con la letra **P** o **V.**
Por ejemplo: Pdest, Porig, Vmatriz, Vec1. Su número de bit depende de la capacidad de direccionar posiciones diferentes. Si direcciona 256 posiciones será de 8 bit, si direcciona 1024 será de 10 bit, si direcciona 65535 será de 16 bit. Se le debe asignar un valor antes de usarlo por primera vez, y puede ser actualizado por una asignación, un incremento, un decremento u otra operación.



Formas de direccionar posiciones de memoria, datos o registros:

Direccionar es la forma de indicar en la instrucción donde se halla el operando a tratar. Se trata de la identificación de la posición en memoria donde la CPU debe buscar el dato a operar.

W← 3B Direccionamiento Inherente. La instrucción indica el registro a utilizar sin incluir su dirección física. Lo invoca nombrándolo, se usa solo para registros internos especiales. En este ejemplo se carga el registro de trabajo con 3Bh.

```
movlw   3Bh
```

F3 + W → W Direccionamiento Inmediato. La instrucción incluye el dato a operar. En este ejemplo el valor literal F3h se suma al registro W y se guarda el resultado en W.

```
addlw   F3h
```

M(17) ← W Direccionamiento Directo. En la instrucción esta incluido el valor de la posición de memoria requerida, o sea se da el valor de la dirección de memoria deseado. El contenido de W se copia en la posición de memoria 17h

VAR1 → W Direccionamiento Directo. Se le asigna un nombre a la posición de memoria

para representarla. Se asocia una posición de memoria RAM con una variable. El contenido de la posición de memoria VAR1 se copia en el W. Previamente se asocio VAR1 con alguna posición de memoria RAM.

```
movwf   17h
movf    VAR1,W      ;Aquí W es una etiqueta para 0
```

M(Po) ← 2C Direccionamiento Indirecto. En la instrucción se menciona el puntero que indica la posición del dato deseado. Se identifica el puntero sin dar su ubicación. La dirección del operando esta contenido en el registro puntero. Po es registro usado como puntero. El PIC16F84 no tiene esta clase de direccionamiento.

M(VAR1)→ W Direccionamiento Indirecto por Registro.
M[M(04)]→ W Direccionamiento Indirecto por Registro. La instrucción contiene la dirección de memoria del registro usado como puntero. El puntero puede estar formado por una o dos variables concatenadas. Se usa indicando el valor de la posición de memoria o el nombre de la variable asociada. El PIC16F84 tiene un solo registro de RAM que tiene estas características. Es el FSR en la posición 04h. La posición de memoria direccionada por VAR1 (que esta en 04h) se copia en el W.

```
movf    00h,W
```

M(Pd + k) Direccionamiento indexado. La instrucción incluye como parámetros el vector (Po) a utilizar y el valor (k) que se le debe sumar al mismo para direccionar el dato a operar. Al vector se le suma un valor para determinar cual elemento de la serie se selecciona, después de la operación el vector índice queda inalterado. El PIC16F84 no tiene esta característica.

M(PC + VAR1) Direccionamiento Relativo. Idéntico al indexado, pero el puntero a utilizar es el PC. El PIC16F84 tiene posibilidad de este direccionamiento pero es necesaria una pequeña subrutina.

Los puertos se identifican con la letra P y el sistema es igual que el de las memorias.
P(nn) -Direccionamiento directo, se selecciona el puerto de la dirección nn
PA -Direccionamiento directo, se selecciona el puerto mediante su valor real en forma directa. También se le puede nombrar con una cadena de caracteres o etiqueta Ej.: PB, MOTORES, LEDS
P(VAR1) -Direccionamiento por registro indirecto, la variable contiene la posición del puerto

Diseño De Diagramas De Flujo

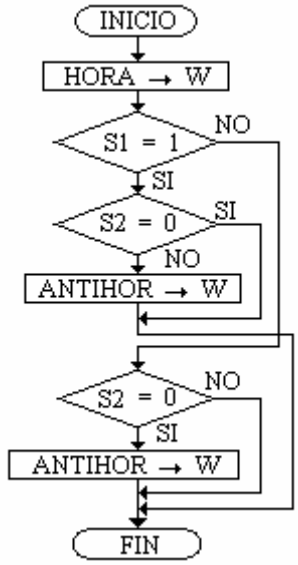
Mostrados la mayoría de los elementos que intervienen en un diagrama de flujo se procede a su construcción.

Los bloques se unen mediante las líneas de flujo permitiendo saber el camino a seguir en el programa. Se comienza realizando un diagrama que muestre lo que deseamos hacer. Este es un diagrama con pocos bloques y muy generales. A cada bloque se le va convirtiendo en varios, que ejecutan acciones más básicas. Eso nos permite personalizar el diagrama para el micro donde lo apliquemos. No todos los micros tienen iguales capacidades ni flexibilidad. Por eso las acciones que para un dispositivo son un solo bloque para otros insumen varios.

Pueden tener varias estructuras, pero en programación de microprocesadores se usan dos:

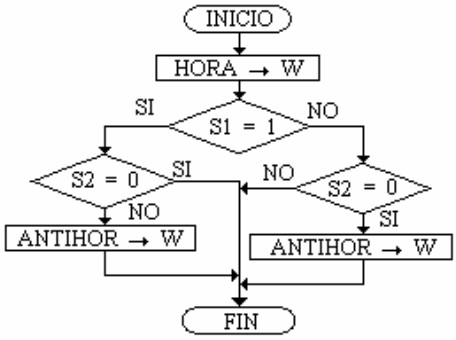
i) Diagrama Alineado

En esta estructura todos los bloques se dibujan sobre una línea vertical. Demanda mayor cantidad de páginas, no es tan fácil de entender su funcionamiento, pero en cambio es más fácil realizar el programa fuente y determinar cuando hay un salto.



ii) Diagrama Sábana o Extendido

Los bloques se distribuyen libremente dentro de la página. Se necesita menor número de páginas y es más fácil de entender. Se dificulta determinar el número correcto de saltos y extraer el programa fuente.



En este curso se utiliza preferentemente el diagrama alineado, excepto que se le explique el programa a un lego.

El programa se desarrolla sobre una línea vertical de arriba hacia abajo, que comienza con el bloque de inicio de programa y en caso de finalizar lo hará con el bloque de fin de programa.

Si el diagrama corresponde al programa principal el bloque de inicio corresponde a la salida de reset del micro. Si es de una subrutina, dicho bloque marca el inicio de la misma.

Los bloques se ejecutan en orden descendente a medida que transcurre el tiempo. Se unen con líneas de flujo. La línea de flujo que avanza de un bloque al siguiente consecutivo se dibuja sobre la línea vertical de guía. Si se produce un salto hacia un bloque posterior la línea de flujo baja por la derecha del eje, y, si el salto es a un bloque anterior lo hace por la izquierda del eje.

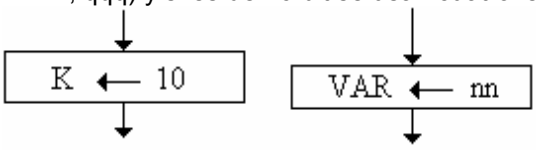
Recordar: ***por la derecha se avanza y por la izquierda se retrocede.***

Siempre a un bloque se llega por arriba y se sale por abajo o el costado.

Es muy útil hacer aclaraciones al diagrama de flujo o comentarios al borde de cada elemento así se da una idea más clara de lo que hace. El comentario más importante es indicar la etiqueta de posición a cada llegada de un salto.

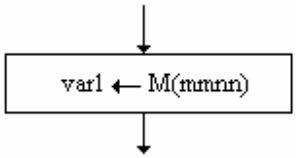
Movimientos y asignaciones:

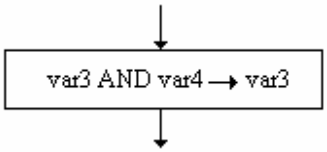
La operación más simple es la asignación de valores a una variable. Si es un valor genérico de 8 bit se representa con dos letras en minúscula (nn, xx, yy), si es de 12 bit con tres minúsculas (nnn, mmm, qqq) y si es de 16 bit se usan cuatro letras (nnmm, ppqq, yyyy, vvzz)



El destinatario de la asignación es una variable o puntero. El valor que se le asigna puede ser un valor directamente, venir de una posición de memoria, o ser el resultado de una operación.

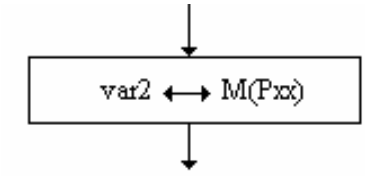
Se hacen asignaciones para cargar valores en las posiciones de memoria, para el valor de los punteros, o para a enviar datos por los puertos. Las asignaciones son movimientos de datos.



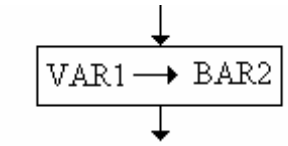


Los movimientos de datos significan copiar el contenido de un elemento en otro. Solo se puede hacer entre elementos de igual clase. Por ejemplo el valor de un puntero de 10 bit no puede ser copiado en una variable de 8 bit. El elemento que suministra la información es llamado **origen** y el que la recibe **destino**. El dato residente en el origen no se altera, solo se destruye el dato anterior al movimiento en el registro destino.

El intercambio es el movimiento que no destruye ninguna de las dos informaciones. Pocos micros pueden realizar esa acción en una sola instrucción



Cuando un dato es traído de memoria es necesario tener su posición de alguna manera. Ya sea el valor de su dirección en forma directa, almacenada en alguna variable o puntero o por el resultado de una operación.

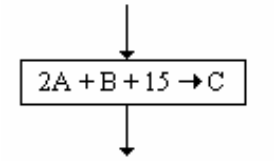


Una variable de muchos bits puede ser formada por la concatenación de dos variables más cortas.
Ejemplo Pdest = VAR1:VAR2
Los dos puntos en medio significa que se concatenan, aportando VAR1 los bit más significativos y VAR2 los bit menos significativos.

El mover el contenido de un registro a otro, significa en realidad, ***copiar el valor que almacena un registro llamado origen o fuente en el registro llamado destino***.

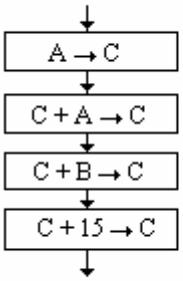
Operaciones:

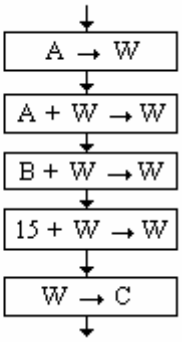
Las operaciones pueden ir en un solo bloque, en general los micros solo pueden operar con 2 variables y una de ellas será usada como destino del resultado si lo hay. Por lo tanto las operaciones complejas o multi-variables solo van en los diagramas generales y no en los personalizados. Además solo hacen operaciones lógico-matemáticas simples y básicas.



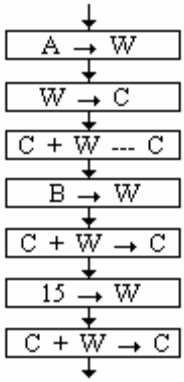
Es necesario descomponer este bloque en varios simples.

Y aun personalizarlo, pues el micro solo opera con un registro y el W.



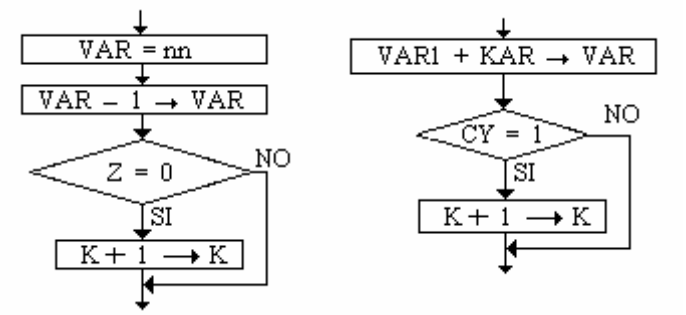


Otra forma de personalizarlo es la siguiente



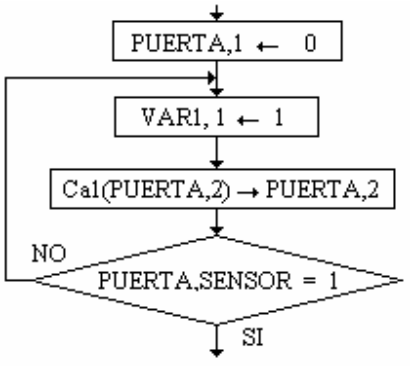
Decisiones y Manejo Bit a Bit:

En los bloques de decisión se puede consultar por cualquier condición.
Los micros solo pueden decidir si un bit determinado perteneciente a un registro en especial cumple o no una condición, es decir si vale 0 o 1. Cuando se personaliza el diagrama de flujo hay que expresar esa condición de manera que se refleje en algún indicador o en el valor de algún bit en particular.
Según la capacidad del micro, este puede chequear cualquier bit dentro de un grupo de registros o solo los bits especiales llamados indicadores, banderas (flag) o banderines (pennant). Se representan en un bit de la palabra de estado del programa (PSW, PSR o STATUS). Su valor depende siempre de la última operación realizada y están asociados a la ALU.
Se verifica si determinada bandera tiene valor 0 o 1. Los indicadores imprescindibles son CY (indicando si hubo acarreo en una suma o borrow en una resta) y Z (indica si el resultado de una instrucción es cero). Actualmente los microcontroladores chequean cualquier bit de un grupo de registros determinados.
Para el PIC16F84A los indicadores son bit del registro STATUS (posición 0x03)



El primero de estos ejemplos se verifica cuando se activa el flag Z indicando que K alcanzo el valor cero. El segundo detecta si hay un desborde de la suma.
Algunos micros manejar álgebra booleana, setear (poner a 1) o resetar (poner a 0) un bit determinado dentro de una variable. Resetear se le llama también limpiar (clear) el bit. Otra operación es el complemento a 1 del bit. Otras operaciones lógicas como OR o AND son más escasas.
Los bits dentro de la variable se numeran según su orden. La notación es el nombre de la variable seguido de una coma y del número o nombre del bit elegido.
Ejemplos
VAR,1 identifica al bit de VAR por el número de orden
PUERTA,SENSOR identifica al bit mediante la etiqueta SENSOR

Los bloques de decisión chequean algunos bit y se selecciona el camino según sean 1 o 0 el bit en cuestión.



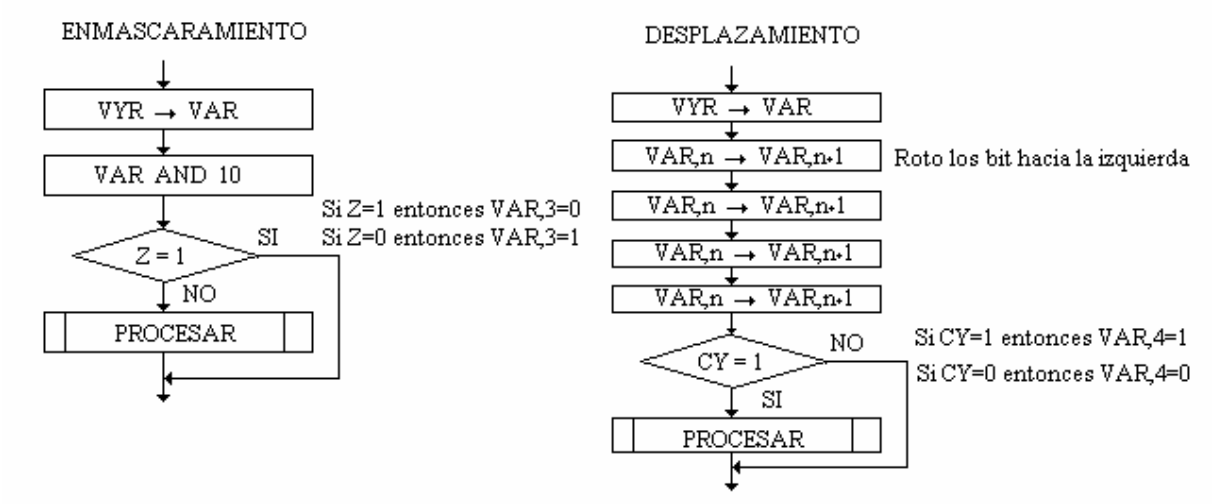
Hay micros que solo pueden chequear los bit de las banderas de resultado. Por lo tanto se utilizan técnicas que permitan verificar cualquier bit de cualquier variable. Hay 2 formas principales y ambas son destructivos de la información de la variable.
Modo de enmascaramiento.

Se considera este ejemplo: queremos verificar el bit 4 de una variable VYR.
Primero se copiar esta información en un lugar seguro.
Se hace una operación AND con el valor 00010000_B.
El resultado se resume a dos posibles: es 00_H o es 10_H.
Solo se detecta si el resultado es cero o no, verificando el flag Z.

Modo de desplazamiento.

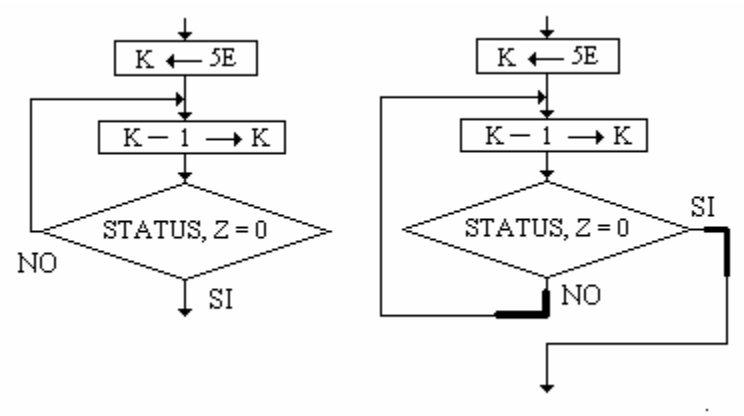
Se considera el ejemplo anterior.
Primero se copiar esta información en un lugar seguro.
Se giran los bits de la variable hacia la izquierda (o la derecha) hasta que el bit deseado quede en el flag de CY.

Solo se detecta si el flag de CY vale cero o uno.



Hay otros métodos pero incluyen la combinación de algunas de estas formas.
Estas son las maneras más prácticas de detectar un bit dentro de una variable. Si se trata del bit más significativo de la variable se puede sumar 80_H y el valor de ese bit se copia en el CY.
La detección de bit en el PIC se hace sobre cualquier bit de los registros SFR y GPR es decir sobre todo el mapa de memoria RAM. La bifurcación solo pueden saltar una instrucción, se ejecuta o no la siguiente instrucción.
Al personalizar los diagramas de flujo se necesita considerar este punto para contar correctamente los saltos de programa.

Considere el siguiente ejemplo:

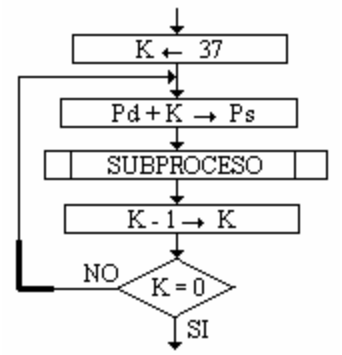


En el diagrama de la derecha se observa el detalle de los saltos para un diagrama personalizado.
Más adelante cuando se vea como convertir el diagrama de flujo en el programa fuente se retomara este problema.

Lazos, loop o iteraciones:

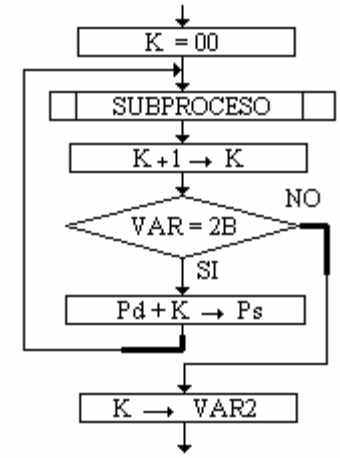
La capacidad de repetir acciones del micro y su implementación con series y matrices hace de los **lazos** (loop) o **iteraciones** una estructura lógica muy usada, permite con pocas instrucciones procesar muchos datos.
Los lazos pueden tener asociados una variable de lazo. Ésta es un elemento que modifica su valor, ya sea dependiendo del número de veces de la repetición o de la actualización de algún dato de entrada. Cuando la variable de lazo alcanza un valor determinado o se cumple una condición esperada se sale del lazo y se continúa con el programa. Cuando la iteración no tiene variable de lazo, comúnmente es un lazo sinfín o sin salida. O sea se repite constantemente mientras funcione el sistema.
Se necesita conocer el valor con que cada variable llega al lazo. Se debe asegurar de no entrar en el lazo con alguna variable teniendo un valor que perjudique el desarrollo del mismo.
Si su valor se asigna independiente del valor anterior no es tan importante, pero si su valor depende de valores anteriores como en los incrementos o decrementos es necesario tener certeza de su valor al comienzo de la primera vuelta de la iteración. Por lo tanto antes de entrar al lazo se le debe

asignar un valor de inicio.



En este ejemplo el puntero Pd no tiene valor conocido al entrar en el lazo. Afecta el desempeño del programa pero no la salida de la iteración.

La variable de lazo se puede chequear en cualquier lugar del lazo. Si se hace al final permite ahorrar un salto, de lo contrario se necesitan al menos dos.

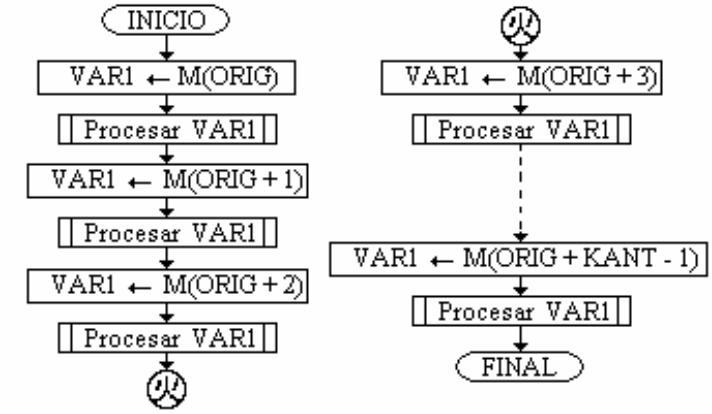


Para salir del lazo se puede chequear más de una variable, la misma varias veces o diferentes condiciones de la variable. Es decir tener más de un bloque de decisión que permite salir del mismo.

Hay programas que son un lazo infinito, se entra en la iteración pero como no hay variable de lazo no se sale de ella.

En los lazos se hacen muy útiles los punteros. Cuando se tratan datos, se toman de algún lado. Si es siempre de la misma posición de memoria se puede usar el direccionamiento directo del operando. Pero si se trabaja con una serie o matriz se usa un puntero, al variarlo en cada vuelta cambia la posición del dato seleccionado. Se utiliza el direccionamiento indirecto por registro.
Ejercicio: Aplicar una subrutina PROCESO a cada uno de los elementos de la serie cuyo comienzo es *ORIG* y tiene una extensión de *KANT* elementos.

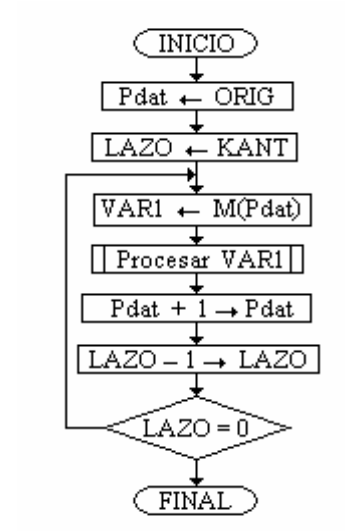
Para tratar cada dato se lo coloca en un variable VAR1 y se invoca la subrutina PROCESO.
Es necesario hacer esto para cada bloque. Por lo tanto las acciones de traer un dato e invocar la subrutina se realizan *KANT* veces. El primer valor se ubica en la posición *ORIG* y el último o sea el kant-ésimo valor en la posición *ORIG + KANT – 1.*



En este ejemplo no se especifica el número de elementos a tratar. Asi que no se puede determinar cuantas veces repetir los bloques de escritura y llamado a subrutina.

En el diagrama se nota lo repetido de la estructura lógica de traer de memoria e invocar a subrutina. Esta repetición se puede ejecutar con un lazo finito. Se identifica lo que varia en cada iteración, en

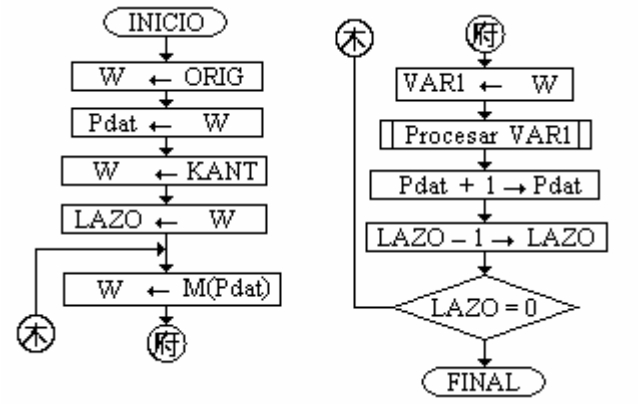
este caso es la posición de memoria usada para buscar el dato. En el programa anterior se usa direccionamiento directo. En el lazo se debe usar direccionamiento indirecto por registro. Se incluye entonces un puntero que asumirá el valor de cada una de las posiciones de memoria, se inicializa con el valor *ORIG* y se incrementa en 1 para cada vuelta del lazo. Todo lazo finito necesita una variable de lazo, se utiliza la variable LAZO que comienza con el valor *KANT*. Agregar un lazo finito implica agregar una variable de lazo.



El bloque VAR1 ← M(Pdat) toma el dato de una posición diferente en cada vuelta del bucle. Así no es necesario ir nombrando cada registro a usar en forma directa.
Este lazo se repite *KANT* veces, desde que la variable de lazo LAZO vale *KANT* hasta que vale 1. Al final del último lazo asume el valor cero, por lo que el bloque de decisión termina la iteración.

Se observa que el puntero y la variable de lazo adquieren un valor diferente al de entrada en la iteración. En particular el puntero termina con un valor correspondiente al del último elemento de la serie más uno. Si se va a usar de nuevo el puntero Pdat deberíamos asignarle el valor de inicio de la serie (*ORIG*).
Personalizado para el PIC16F84A este programa queda:
El puntero se asocia al registro FSR, que es el único puntero de la memoria RAM y todos los movimientos se hacen a través del registro W.

Este programa es más corto en cantidad de bloques a usar, pero demora más tiempo en terminar que la solución anterior. También utiliza más variables que la primera solución, por lo que consume más recursos del sistema.
Siempre al elegir un programa se debe sopesar en que aspecto se lo optimizará.



Cuando un programa es complejo y extenso, no siempre el primer diagrama es el mejor. Se le debe revisar para mejorarlo u optimizarlo.
Optimizar un programa es mejorarlo en alguno de estos aspectos:

*Bajar el número de instrucciones total.
*Disminuir posibilidades de error.
*Mayor rapidez del proceso total.
*Usar menos recursos del micro. O sea usar el menor número posible de variables, registros internos, periféricos internos o externos.

Programa Fuente Para El Pic16f84

El siguiente paso es escribir el programa fuente para el micro. Este se escribe en un lenguaje inteligible para el operador humano, aunque no lo es para el microcontrolador.
Se deduce del diagrama de flujo, de esta manera solo es necesario ir convirtiendo cada bloque en las instrucciones equivalentes. Si se observan posibles mejoras del programa se vuelve al diagrama de flujo y se le altera. Se vuelve a deducir el programa fuente, y se verifica si es necesaria alguna otra mejora. De ser así se vuelve a modificar el diagrama de flujo. A lo largo del diseño del programa, y aun cuando el sistema este funcionando, se debe revisar todo para optimizarlo.
Cada bloque se sustituye por una o varias líneas de instrucciones
El programa fuente contiene:

Instrucciones en el orden a ser ejecutadas por el micro
Reglas de sintaxis muy estrictas, para que sea comprensible para el compilador.
Comandos para el compilador: posición de los datos en el sistema, concatenar con macros, etc.
Mensajes o comentarios del programador sobre el programa

El Assembler es un lenguaje de programación de bajo nivel, es decir trabaja con las instrucciones del micro propiamente dichas. Con los Mnemónicos del micro en particular se genera el programa fuente, que es un archivo de texto. Después se convierte a código maquina, donde cada instrucción corresponde a un número binario.

Los lenguajes de alto nivel usan sintaxis más elaboradas, donde las líneas se decodifican no en una instrucción en código maquina sino en una subrutina en código maquina, o sea cada instrucción es un pequeño programa.

Programación De Bajo Nivel

Todas las instrucciones contienen la información de que operación realizar y donde encontrar los operandos involucrados. En la mayoría de las instrucciones del PIC16F84A uno de los operando es el registro W, o el resultado se guarda en él. El origen de los operando puede ser de un registro RAM (movf STATUS,W), de un periférico (movf PORTA,W) , el propio registro W (movwf TRISB) o un dato que viene en la instrucción (movlw 0x23). El destino puede ser igual de variado que el origen.

Según como se indique la posición de los datos, en la instrucción, se reconocen las siguientes formas de direccionamiento:

INHERENTE: La instrucción incluye el nombre del registro, no su dirección.
INMEDIATO: La instrucción incluye el valor de unos de los operando.
DIRECTO: La instrucción contiene la dirección donde se ubica el operando.
INDIRECTO: La posición la da un puntero identificado por su nombre.
INDIRECTO POR REGISTRO: La posición del operando la da un puntero identificado por su dirección.
RELATIVO: La instrucción contiene un número el cual se suma al registro PC. Si la instrucción contiene 8 significa que el operando esta en PC+8 y si es -8 esta en PC-8. Son instrucciones de salto
INDEXADO: Es similar al relativo pero se usa un registro llamado Registro Índice, diferente al PC. Capturar el operando no implica la alteración del índice.

Estas son las formas de direccionamiento usuales en programación. Una instrucción puede tener mas de una forma de direccionar, una para cada operando y otra para el destino.
El PIC tiene dos instrucciones que no usan operando, ni tienen un registro indicado aparentemente: NOP y SLEEP.
El PIC tiene solo las siguientes formas de direccionamiento:
Inherente, inmediato, directo, indirecto por registro e indexado.

Para direccionar distintas posiciones de memoria pero con la misma instrucción (en un lazo), se usa una variable que almacena la dirección del operando. Para cambiar de operando se modifica el contenido de la variable pero la instrucción siempre es la misma, a estas variables se les llama punteros.

Si tenemos un grupo de valores en posiciones continuas de memoria se les puede agrupar en un conjunto, y llamar a ese grupo serie. Toda serie implica también un registro que le sirve de puntero (vector) para direccionar los elementos de la misma. Así en el programa se trabaja con el vector, que

implica varios valores posibles y se invocan con el puntero. Si tenemos la posibilidad de hacer direccionamiento indexado, el puntero pasa a ser el índice de la serie.

Los datos también se pueden ordenar en grupos de 256, los cuales si están continuos forman una página, debido a eso, con un byte se les puede direccionar a todos. Es decir, el byte bajo del puntero indica la posición del registro, y el byte alto del puntero selecciona de cual página es el registro. Es como que todos los datos se encuentran en una grilla con su posición vertical y horizontal particular. A esta forma de ver los datos se les llama matriz, array o arreglo. Se indica el número de grupos por el número de registros por grupo. Por ejemplo si se usan 10 páginas, se dice matriz de 10 x 256, o sea se consideran 10 grupos de 256 elementos cada grupo. Se eligen páginas de 256 registros solo por comodidad

Todo programa esta diseñado para solucionar un problema.
Primero se debe considerar las partes importantes o de peso del problema. La mayor cantidad de partes consideradas permiten hacer un programa al cual hay que modificar y corregir menos veces.
Una vez considerados los puntos de peso, vemos cuantas variables son necesarias suministrar al sistema y como darles el valor adecuado.
Después se realiza un primer diagrama de flujo que describe el proceso en forma clara y concisa, que solo muestre los pasos más importantes.
Entonces se manipula desglosando cada bloque en varios más simples. Se eliminan redundancias y si hay procesos repetitivos se resumen en lazos. Estos lazos implican agregar más variables.
Se verifica y se realiza el programa fuente.
A continuación se optimiza o depura el programa fuente. En caso de ser necesario se vuelve al diagrama, se le modifica y se re-escribe el archivo fuente.
Puede ser probado con un simulador virtual en un PC. Si se detectan errores se vuelve a modificar el diagrama de flujo o el programa fuente.
Después solo queda compilar el programa objeto para colocarlo en la memoria ROM del sistema físico.

Asociación Diagrama De Flujo - Programa Fuente

Bloque de ejecución:
Según sea la acción desarrollada en él, se determina la instrucción equivalente.
Cada variable debe asociarse a una posición de memoria RAM. Los nombres de las variables representan posiciones de memoria de los registros de propósito general (GPR) o de los especiales (SFR)
Los punteros como Pdat debe asociarse con el registro FSR de posición 04h. Cuando el programa contiene más de un puntero lógico, se debe guardar cada puntero en un registro. Para usar cada puntero se lo coloca en el FSR.
Los valores nombrados en forma directa se escriben precedidos por un cero y una x para indicar que es número hexadecimal.

Movimientos y Asignaciones

VAR1 → 35
El único registro que permite cargar un valor en forma inmediata es W, así que primero cargamos el valor en W y después lo movemos a la posición de VAR1
Los nombres de las variables aquí serán usados como etiquetas para determinar su posición

```
        movlw   0x35        ;Se carga W con 35
        movwf   VAR1        ;Mover W al VAR1
```

VAR1 → VAR2
No hay instrucción que mueva los datos de un registro GPR o SFR a otro. El dato debe ir primero al W y después al registro destino.

```
        movf    VAR1,W      ;mover VAR1 al W
        movwf   VAR2        ;mover W al VAR2
```

El dato se copia en el registro destino y en el W.

VAR1 → VAR1
Aunque no modifica al registro, el flag Z es activado si el valor de VAR1 es cero

```
        movf    VAR1,F      ;Se verifica si VAR1 es cero
```

M(Pdat) → 45

Cargar una posición de memoria seleccionada por puntero. Solo se puede utilizar el FSR de 8 bit ubicado en 04h. Este basta para direccionar todos los registros RAM. Para utilizarlo se trabaja con el registro INDF

```
movlw   0x45    ;Cargar 45 en W
movwf   INDF    ;Mover W a M(Pdat)
```

M(Pdat) → VAR2
Cargar el valor de una variable en un registro direccionado por puntero.

```
movf    VAR2,W  ;Colocar VAR2 en el registro de trabajo W
movwf   INDF    ;Cargar el registro M(Pdat) con el valor de VAR2
```

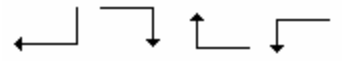
El registro INDF permite usar el direccionamiento a través del puntero FSR.

M(Porig) → M(Pdest)
Mover la información entre dos posiciones direccionadas por puntero. El dato debe pasar por el W. Se indican dos punteros pero solo esta el FSR. Se deben usar registros auxiliares para almacenar el valor de los punteros cuando no son usados.
Estos van a ser VORIG y VDEST. Para guardar momentáneamente el dato se usa AUX.

```
movf    VORIG,W     ;Traer valor actual de Porig
movwf   FSR         ;Se coloca Porig en el registro FSR
movf    INDF,W      ;M(Porig) → W
movwf   AUX         ;Se guarda momentáneamente
                    ;el dato en un registro auxiliar
movf    VDEST,W     ;Traer valor actual de Pdest
movwf   FSR         ;Se coloca Pdest en el registro FSR
movf    AUX,W   ;Traer el dato desde el registro auxiliar.
movwf   INDF        ;Acomodo dato en la posición direccionada
                    ;por Pdest
```

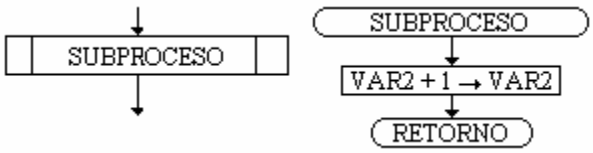
Saltos, Subrutinas y Decisiones

La llegada de una rama a otra implica colocar una etiqueta que marque esa posición en el programa fuente.
Son nombres que van colocados sobre el margen izquierdo, sin espacios en blanco.



Los codos implican saltos hacia algún bloque del programa que no es el consecutivo. Se asocian con la instrucción goto, y con un argumento que representa una posición como una etiqueta de posición..

```
goto    ETIQUETA_DE_POSICION
```



Las subrutinas se desarrollan antes o después del programa principal. En el caso de los PIC es necesario que sea al principio de la página de memoria.

```
SUBPROCESO          ;Cada subrutina comienza con la etiqueta
                    ;de su nombre
    incf    VAR2,f  ;Incrementar registro VAR2
    return          ;Retornar al bloque siguiente al llamado de
                    ;subrutina
```

Otra forma puede ser

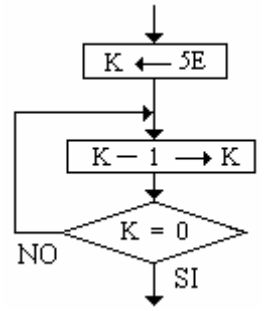
```
SUBPROCESO          ;Cada subrutina comienza con la etiqueta
                    ;de su nombre
    incf    VAR2,f  ;Incrementar registro VAR2
    retlw   PRONTO  ;Retornar con el valor PRONTO en W

PROGRAMA_PRINCIPAL
    ….
    call    SUBPROCESO;Se invoca a la subrutina llamada SUBPROCESO
    ….              ;Al volver de la subrutina se sigue en esta línea
    ….
    call    SUBPROCESO;Se invoca de nuevo la subrutina SUBPROCESO
```

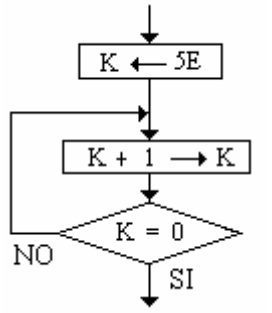
```
….                              ;Al volver de la subrutina se sigue en esta línea
….
```

La PILA permite guardar 8 direcciones de vuelta de subrutinas. Eso implica que se pueden anidar hasta 8 subrutinas una dentro de otras. Si se supera este número se perderá la posición del primer regreso por sobre-escritura.

Se convierte de a un bloque:

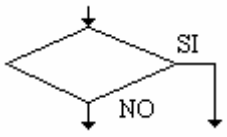


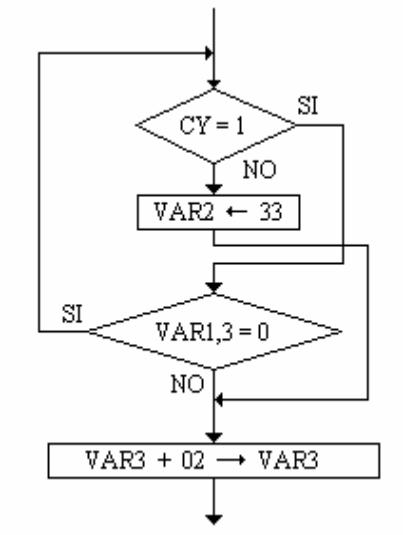

```
        movlw   0x5E
        movwf   K           ;Cargar K con el valor 5Eh
LAZO                        ;La llegada de una rama
                            ;indica una etiqueta. Esta se
                        ;escribe sin espacios en blanco.
        decfsz  K,F         ;Decrementar K y si es cero
                            ;entonces PC + 1 → PC.
        goto    LAZO        ;Si no es cero salto a donde
                            ;esta la etiqueta LAZO
        …                   ;Si es cero sigo de largo con el
        …                   ; programa
```



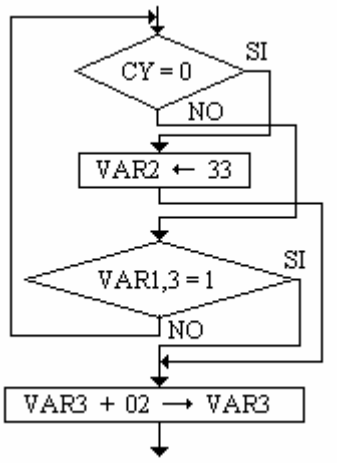

```
        movlw   0x5E
        movwf   K           ;Cargo K con el valor 5E
AUMENT                      ;Etiqueta que marca la llegada
                            ;de una rama
        incfsz  K,F         ;Incrementar y si K desborda
                            ;saltar la próxima instrucción
        goto    AUMENT      ;Mientras no sea cero
        …                   ;ejecuta este goto
```

Los bloques de decisión saltan una instrucción cuando se cumple la condición. Si no se cumple ejecutan la instrucción siguiente. Se debe realizar la consulta de forma que la respuesta afirmativa genere el salto. La forma de clarificar los diagramas es utilizar esta forma para los bloques de decisión:





Considere el primer bloque de decisión. Si la repuesta es afirmativa se salta un bloque compuesto de dos instrucciones y una instrucción goto. Saltar 3 instrucciones no se puede con una sola consulta. Es necesario incluir de un codo en el diagrama. Respecto al segundo bloque de decisión, no puede solo saltar hacia atrás. Se debe utilizar un codo y preguntar de forma que la afirmativa genere el salto de una instrucción. El diagrama se convierte en el de la derecha



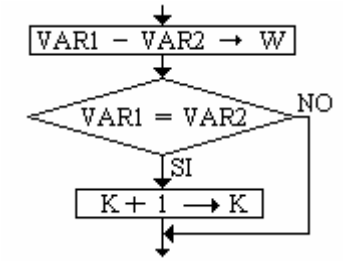

```
LLEGAR1
        btfsc   STATUS,CY       ;Si CY es cero  PC + 1 → PC
        goto    LLEGAR2         ;Si es uno saltar a LLEGAR2
        movlw           0x33
        movwf           VAR2    ;Cargar 33h en VAR2 si CY = 0
        goto    LLEGAR3
LLEGAR2
        btfsc           VAR1,.3   ;Chequear el bit 3 de VAR1
        goto    LLEGAR1         ;Si es 1 salto a LLEGAR1
LLEGAR3
        movlw           0x02    ;Sumar 02h a VAR3
        addwf           VAR3,F;
```

Para comparar dos variables primero debemos restarlas entre si. Y que el resultado no afecte a ninguna de las dos. Después se verifican los flag de CY (STATUS,0) y Z (STATUS,2). En la resta el CY representa el negado del borrow. Por lo tanto si la resta lleva borrow entonces CY=0

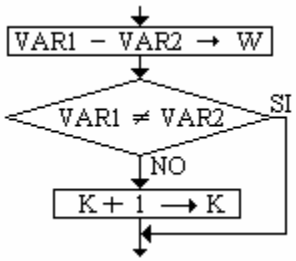
VAR1 – VAR2

VAR1 = VAR2	;	Z=1	CY=X
VAR1 ≠ VAR2	;	Z=0	CY=X
VAR1 < VAR2	;	Z=X	CY=0
VAR1 ≥ VAR2	;	Z=X	CY=1
VAR1 > VAR2	;	Z=0	CY=1
VAR1 ≤ VAR2	;	Z=1	CY=X o Z=0 CY=0

Observar que la repuesta que genera el salto es la afirmativa

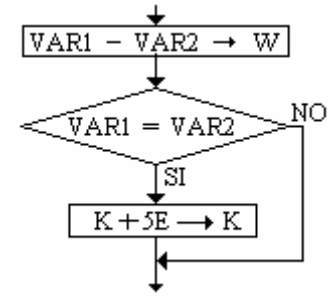


Se transforma en



Se hacen por bloques

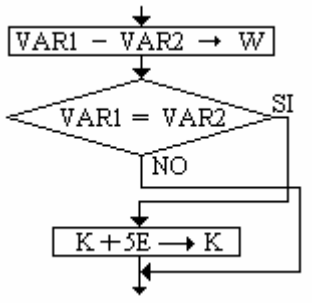
```
        movf        VAR2,W
        subwf       VAR1,W      ;Resto para comparar
        btfsc       STATUS,Z    ;Si la bandera Z es cero
                                ;salto una instrucción,
                                ; Si Z=1 ⇒ PC+1 → PC
        incf        K,F
        ….
```



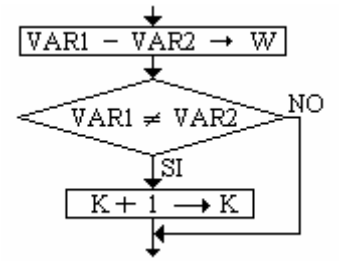
La diferencia entre este diagrama y el anterior radica en el bloque de suma. Si es solo un incremento se puede representar por una sola instrucción, pero si es un valor diferente a 1 se necesitan dos o más.

Las instrucciones de decisión solo saltan una posición de memoria, se debe usar una instrucción de salto.

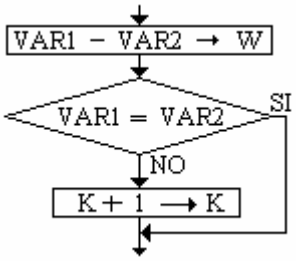
Se transforma en el diagrama de la derecha.



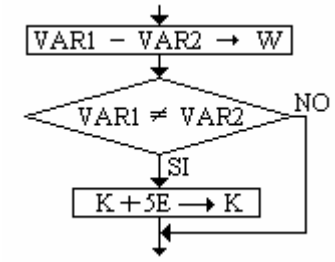
```
        movf    VAR2,W
        subwf   VAR1,W          ;Restar para comparar
        btfss   STATUS,Z        ;Si flag Z es uno saltar una instrucción
                                ;  Z=1 ➔ PC + 1 → PC
        goto    NOIGUAL
        movlw   0x5E
        addwf   K,F             ;Sumo K + 5E y guardo en K
NOIGUAL
        …
```



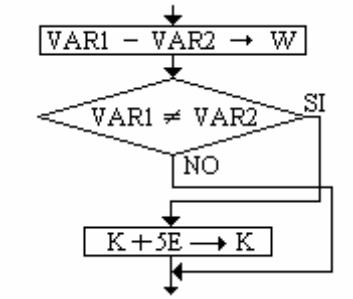
Se transforma en



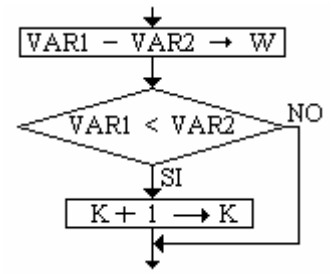

```
        movf    VAR2,W
        subwf   VAR1,W      ;Restar para comparar
        btfss   STATUS,Z    ;Si la bandera Z es uno se salta la instrucción
                            ; inmediata.  Si Z=0 ➔ PC + 1 → PC
        incf    K,F
        ….
```



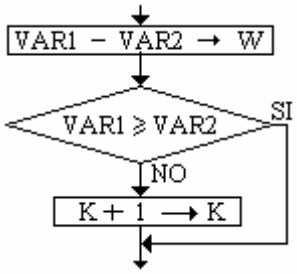
La diferencia entre este diagrama y el anterior radica en el bloque de suma. Un incremento se puede representar por una sola instrucción, pero si es un valor diferente a 1 se necesitan más instrucciones. Para avanzar más de una posición se usa un salto. Lo que implica una instrucción goto.



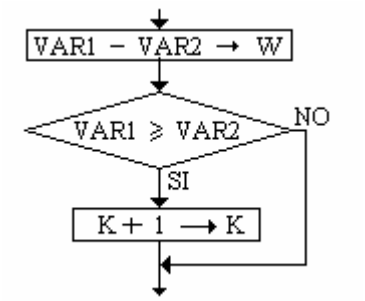

```
        movf    VAR2,W
        subwf   VAR1,W          ;Restar para comparar
        btfsc   STATUS,Z        ;Si Z es cero saltar una instrucción,
                                ;Si  Z=0 ➔ PC + 1 → PC
        goto    IGUALES
        movlw   0x5E
        addwf   K,F             ;Sumar K + 5E y guardar en K
IGUALES
        …
```



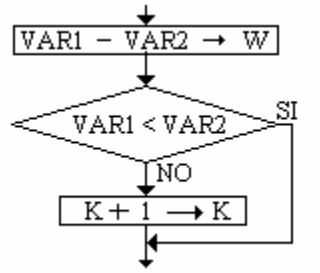
Se convierte en el siguiente , así el salto se produce con la afirmativa



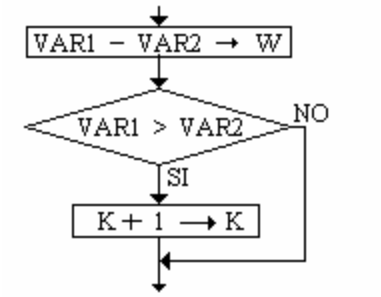

```
        movf    VAR2,W
        subwf   VAR1,W          ;Restar
        btfss   STATUS,CY       ;Si BW=0  ➔ PC+1 → PC
        incf    K,F
        ….
```



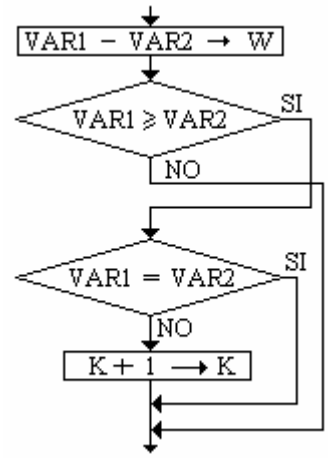
Se transforma en




```
        movf    VAR2,W
        subwf   VAR1,W          ;Restar
        btfsc   STATUS,CY       ;Si CY=1 ➔PC+1 → PC
        incf    K,F
        ….
```

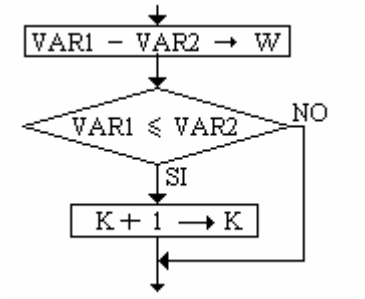


En este diagrama el bloque de decisión es complejo por lo que se debe convertir en otros más simples. Su complejidad radica en que es necesario consultar dos indicadores para determinar si es verdadero o falso.

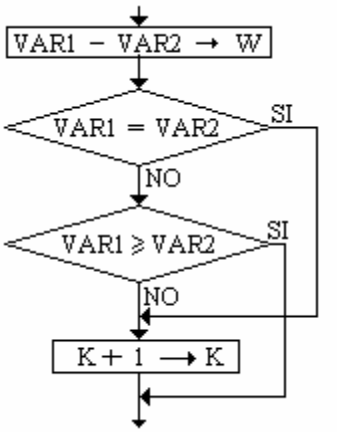


Estos dos bloques de decisión son más simples y cada uno verifica un banderín. Observar que la primera decisión tiene que saltar más de una instrucción.

```
        movf    VAR2,W
        subwf   VAR1,W          ;Restar para
comparar
        btfss   STATUS,CY       ;Si es mayor o igual saltar la siguiente
                                ; instrucción
        goto    NOMAYOR
        btfss   STATUS,Z        ;Si es igual saltar la siguiente instrucción
        incf    K,F             ;Incrementar K
NOMAYOR
….
```

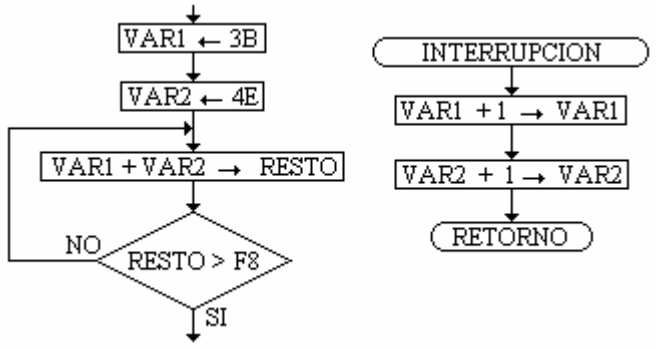


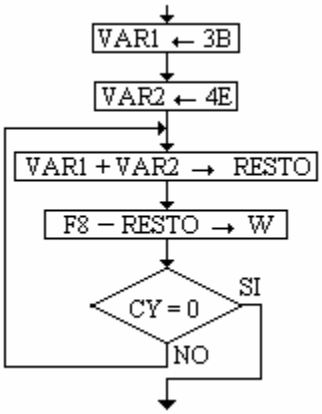
En este diagrama el bloque de decisión es complejo por depender de dos banderas, así que se debe convertir en dos más simples.




```
        movf    VAR2,W
        subwf   VAR1,W          ;Comparar
        btfss   STATUS,Z        ;Si son iguales altar y realizar incremento
        btfss   STATUS,CY       ;Si no es menor salto
        inca    K,F
        ….
```

La subrutina de atención a una petición de interrupción es una subrutina particular. Se le llama interrupción o subrutina de interrupción. Como toda subrutina debe ir al comienzo de la página.





El bloque de decisión no es simple. Se le puede descomponer en los siguientes bloques.

Se debe convertir un diagrama a continuación del otro.

```
INTERRUPCION
        incf        VAR1,f
        incf        VAR2,f
FIN_DE_INTERRUPCION
        retfie                  ;Retornar de la subrutina de interrupción
                                ;y habilitar las demás interrupciones.
PROGRAMA_PRINCIPAL
        ....
        ….
        movlw       0x3B
        movwf       VAR1        ;Setear la primera variable
        movlw       0x4E
        movwf       VAR2        ;Cargar la segunda variable
LAZO
        movf        VAR2,W
        addwf       VAR1,W      ;Sumar ambas variables
        movwf       RESTO
        sublw       0xF8        ;Restar para comparar con F8
        btfsc       STATUS,CY   ;Cuando CY=0 RESTA será mayor
        goto        LAZO
        ….
        ….
        ….
```

Sintaxis Del Compilador

Para indicar el sistema de numeración en que se escribe un número se usa esta notación

Hexadecimal	0xHH , 0xHHH , 0xHHH
Binario	b'xxxxxxxx'
Decimal	.d , .dd , .ddd

Los comandos para el compilador más importantes son:

EQU Declara las etiquetas a usar en el programa. Asocia cadena de caracteres (strings) con valores numéricos

Su sintaxis es

```
<nom_etiq>   EQU   0xnn   ; Asigna a la etiqueta llamada nom_etiq el valor nn (hexa).
```

Ejemplos:

```
Status       EQU   0x03   ;Se asocia el valor 03h con la cadena "Status". Así se identifica
                          ; este nombre con la posición del registro indicado.
CY           EQU   .0     ;Se usa esta etiqueta para indicar la posición del CARRY
VAR1         EQU   0x2F   ;Esta variable VAR1 se asociada con un GPR y se le asigna el
                          ; valor correspondiente a la posición de dicho registro
FSR          EQU   0x04   ;Se asocia esta cadena de caracteres al valor de la posición
                          ;de este registro.
```

ORG Sirve para ubicar el programa que se escriba a continuación en una posición determinada de la memoria.

Su sintaxis es

```
ORG   0xnnnn ; El programa a partir de aquí se ubicara desde la
                          ;posición nnnn (hexa) en adelante.
```

Su uso permite escribir el programa como quede más cómodo y que el compilador lo ordene.
El PIC16F84 al ser reseteado pone el PC a cero. Por lo tanto se obtiene el **vector de inicio** en 000h.
Al atender una petición de interrupción, el programa principal, la busca en la posición 004h. Por lo tanto se tiene el **vector de interrupción** en 004h.
Entre el vector de inicio y el vector de interrupción hay pocas posiciones de memoria. Para poder desarrollar un programa más o menos completo es necesario saltar por arriba de la subrutina de interrupción. O sea colocar una instrucción "goto" hacia donde comienza el programa principal.
Las subrutinas en este micro deben comenzar entre 000h y 0FFh.

```
            ORG   0x000
VECT_INIC
            goto  INICIO

            ORG   0x004
VECT_INT
            ….                  ;Comienzo de la subrutina de atención a una
                                ; interrupción
            ….
            ….
            retfie              ;Termina la subrutina y se vuelve al programa
                                ; principal

SUBRUTINA                       ; Comienza de subrutina antes de la posición 100h
            ….                  ;Al no indicar un valor determinado, el propio
                                ;compilador le signa la posición.
            ….                  ; Cuenta una posición por cada instrucción
            ….                  ; desde el último comando "org".
            return

INICIO                          ; Inicio del Cuerpo del programa principal.
            ….                  ;
            ….
            ….
```

END ;Indica el final del programa a compilar. Todo lo que se escriba a continuación no será compilado por el ensamblador. Después de este comando debe haber un carácter de retorno de carro (enter) o renglón en blanco.

LIST p=16f84a ;Indica para que dispositivo se realizara la compilación. Antes de comenzar con el programa fuente es necesario elegir para qué micro se hará esta asociación. Cada micro tiene su propio set de instrucciones. Inclusive dentro de la misma familia debido a sus características personales puede variar. En este curso se elige el PIC16F84A.

__CONFIG 0xnn ; El byte nn se guarda en la posición 2007 para el pic16f84a y configura si se habilita o no la protección de código, habilitación de WatchDog, si o no Power Up Timer y clase de oscilador.

__IDLOCS **h'***xxxx***'** ;Graba los nibles IDLOCS de las posiciones 2000 a 2003.

El programa fuente debe ser escrito dividido en 4 columnas llamadas campos:
La primera de más a la izquierda se llama **campo de etiquetas**. Se escriben sin dejar espacio a la izquierda. Puede tener una longitud de hasta 32 caracteres dependiendo del ensamblador a usar. Las etiquetas aparecen una sola vez esta columna, pero puede repetirse en cualquiera de las otras. Pueden usarse para designar valores numéricos o marcar posiciones de memoria en el programa. Si hay que saltar, no se indica la posición sino el nombre de la etiqueta que lo marca. Para cada llegada de una rama a otra se debe colocar una etiqueta, excepto en el caso de saltos predefinidos por el fabricante.
La segunda es el **campo de comandos**. Ahí van las instrucciones del micro y los comandos para el ensamblador.
La tercera es el **campo de operandos**. Se colocan los operandos de cada instrucción, estos valores numéricos pueden sustituirse por etiquetas declaradas previamente. El uso de una etiqueta evita escribir el valor numérico que representa.
La cuarta columna (la de más a la derecha), es el **campo de los comentarios**. Comienza con el símbolo " **;** ", todo lo escrito a su derecha es ignorado por el compilador. Es solo para orientar al programador. Cuantos más comentarios incluya un programa más fácil es entenderlo.

Etiquetas	Comandos	Operandos	Comentarios
INICIAR	EQU	0x000	;Seleccionar inicio al principio de la
			;memoria ROM
PORTB	EQU	0x05	;Indica posición del registro del puerto B
VAR1	EQU	0x2A	;Indica posición de la variable VAR1
INICIO			;Es una etiqueta de posición. Una marca
			;en el programa.
	ORG	INICIAR	;Comando para indicar donde se colocara
			;el programa
	movlw	0x35	;Cargar el valor 35h en el W
	movf	VAR1	;Cargo este valor en la posición 2A
	…	…	
	…	…	
FINAL	goto	INICIO	;Saltar al principio del programa
	END		;Finaliza el programa a compilar

Al escribir un programa fuente se debe incluir 3 comandos al principio.
Se debe incluir en el programa fuente información sobre el modelo de PIC elegido, clase de oscilador a usar, si se habilita el WDT, el modulo de PWRT, código de protección y el valor para los números de ID.

Se usa un comando para el ensamblador al principio del programa

LIST p=16f84

Ejemplo para compilar para el pic16F84 o pic16F84A. Le dice al compilador para que micro esta diseñado el programa. Se debe colocar primero este comando sino no reconoce los demás.

__CONFIG b'*xxxxxxxxxxxxxx***'**

Bits de Configuración o Palabra de Configuración:
Al momento de quemar el micro se puede setear parámetros de funcionamiento como modo de Oscilador (OSC), retardo de encendido (PWRT), habilitación del Vigilante (WDTE) y protección del

programa (CP).
La palabra de configuración tiene 14 bit al igual que las instrucciones.

b13													b0
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRT	WDTE	OSC1	OSC0

bit 14-4 CP: bits de Protección de Código
Alguno 1 = No protección de código, permite la lectura de la ROM y la EEPROM por el programador
Todos 0 = Protección de código habilitado. Impide la lectura de la ROM por el programador. Solo se leen los bits de IDLOCS y el resto de la ROM y EEPROM como ceros.
bit 3 PWRTE: bit de habilitación de Temporización de Encendido
1 = Temporizador de Encendido deshabilitado
0 = Temporizador de Encendido habilitado
bit 2 WDTE: bit de habilitación del WDT
1 = WDT habilitado
0 = WDT inhabilitado
bit 1,0 FOSC1,FOSC0 :bits de selección de oscilador
11 = Oscilador RC
10 = Oscilador HS
01 = Oscilador XT
00 = Oscilador LP

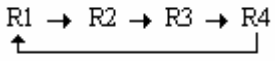
```
__CONFIG 0x3FF1          ;Configura No protección de código, PWRTE habilitado,
                         ; WDT no habilitado y oscilador a cristal XT
```

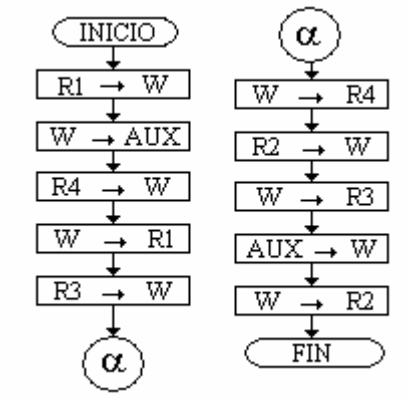
__IDLOCS h'*xxxx*' ;El PIC 16F84 tiene la posibilidad de grabar los nibles bajos de las posiciones 2000 a 2003 con un valor que puede ser usado como número de serie, identificador o cualquier otro uso. Solo estos dos comandos pueden grabarse en las posiciones de memoria de configuración.

Al principio del programa fuente se debe declarar todas las etiquetas de las variables que se usan. Se asocia cada variable con una posición de memoria RAM o registro GPR. Se debe observar si al registro elegido se le pueden aplicar las operaciones y movimientos que sufre la variable.
A cada registro de función especial (SFR) se le asocia el nombre con su posición.
El registro más versátil es el de trabajo (W). No se debe asociar ninguna variable a este registro en forma permanente, sino siempre temporal. Esto se debe a que muchas veces al operar se necesita mover la variable del registro elegido al W y después nuevamente al registro.
Si la variable tiene más bit que un registro, hay que asociarla a más de una posición de memoria RAM.
Solo hay dos punteros accesibles al operador, el de direccionamiento indirecto FSR para la RAM y el contador de programa PC para la ROM.
Se puede direccionar un dato solo de estas formas:
Direccionamiento inmediato. El dato o literal esta incluido en la instrucción.
Direccionamiento inherente. La instrucción define el registro a usar sin incluir su dirección.
Direccionamiento directo. La instrucción incluye la dirección donde esta el operando o dato.
Direccionamiento indirecto por registro. El registro a usar como puntero es el FSR. La instrucción direcciona al registro INDF (00h), pero busca el dato en la posición direccionada por FSR.

A cada valor de posición ROM (Vector de Interrupción, Vector de Inicio, Subrutinas, etc.) se le asocia con una etiqueta.
Si no se hace esto ultimo, el compilador asigna a cada instrucción una posición de ROM y calcula para cada etiqueta de posición su valor. El compilador cuenta desde el último comando ORG y sino lo hubiera por defecto comienza en 0000h.

Ejercicio:
Cambie el contenido de los registros R1, R2, R3, R4
Mover R1 a R2, R2 a R3, R3 a R4 y R4 a R1
A partir del diagrama de flujo diseñado como solución se escribe el programa fuente correspondiente.






```
;----------Ejercicio Rotación de Registros ----------
;Es aconsejable llenar el programa de comentarios que guíen en
;la comprensión del mismo
;Primero definir el micro a usar y las condiciones de uso

                LIST        p=16f84a        ;Selecciona para que
                                            ;debe compilar el ensamblador
                __CONFIG    h'0011'         ;Con este valor configura
                                            ;Oscilador XT , No WDT
                                            ;Si PWRT, No CP
                __IDLOCS    0x0001          ;Grabar 4 nibles
                                            ;en los registros ID
;Se debe definir los valores de las variables a usar.
;No usa ningún modulo interno.
AUX             EQU         0x20
R1              EQU         0x21
R2              EQU         0x22
R3              EQU         0x23
R4              EQU         0x24

;----------  Definir los SFR a usar  ----------
F               EQU         .1              ;Sirve para indicar registro como destino
W               EQU         .0              ;Sirve para definir a W como destino
;No se necesitan ningún otro registro de función especial
                ORG         0x0000
INICIO                                      ;Se debe colocar los comandos org antes
                movf        R1,W            ; de la etiqueta de posición
                movwf       AUX             ; R1 → AUX
                movf        R4,W
                movwf       R1              ; R4 → R1
                movf        R3,W
                movwf       R4              ;  R3 → R4
                movf        R2,W
                movwf       R3              ; R2 → R3
                movf        AUX,W
                movwf       R2              ; R1 → R2

                END                         ;Fin del programa a compilar
```

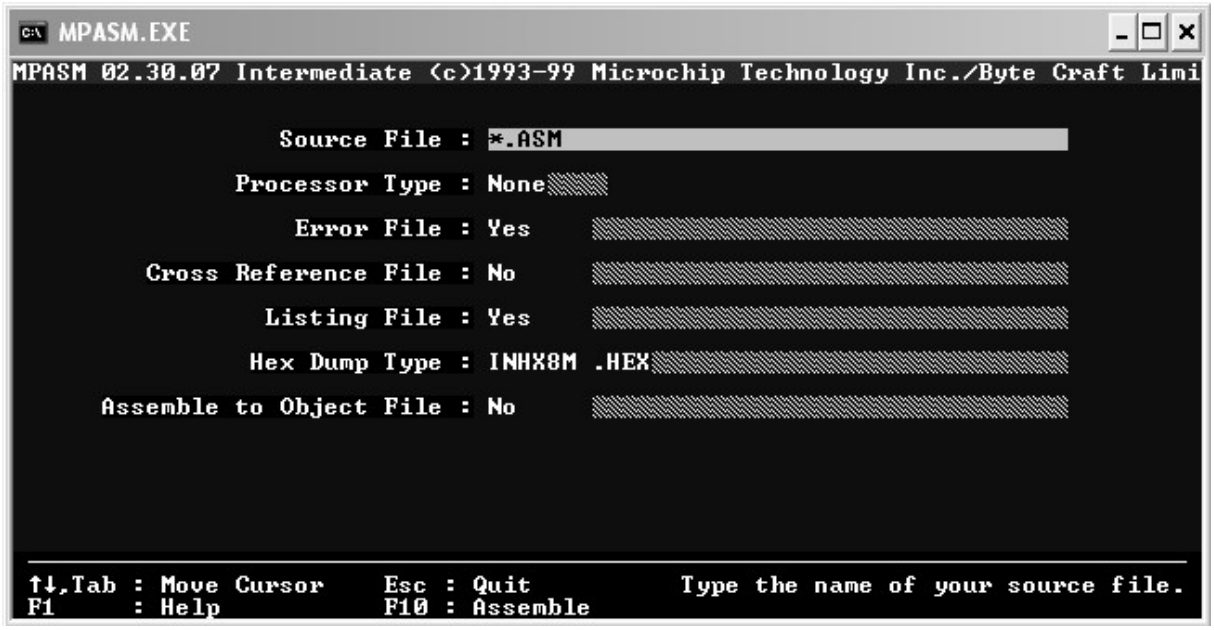
Herramientas de Programación

EDITOR

La primera herramienta a usar será el editor de texto. Por ser una utilidad de los sistemas operativos se obvia la explicación de su manejo.
Se escribe el programa fuente y se guarda como un archivo de extensión .ASM
Se almacena en la misma carpeta que el compilador y programador cuando se use el sistema operativo DOS.
Lo aconsejable es crear una carpeta en el directorio raíz llamado por ejemplo PIC y en ella tener el simulador, el compilador y el grabador.
Eso permite llegar fácilmente si tenemos que arrancar la maquina en modo DOS.

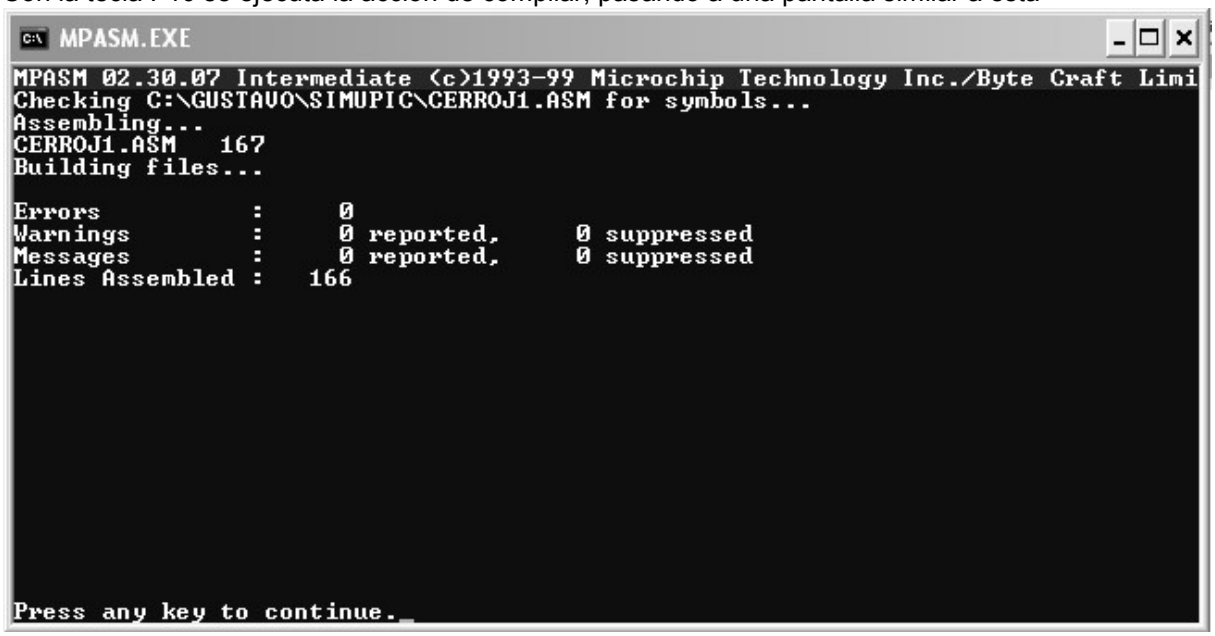
Ensamblador o Compilador

La siguiente herramienta es el compilador. Genera a partir del programa fuente el programa objeto y el archivo listado. El programa objeto se cargara en la memoria de programa del sistema. El archivo listado nos muestra información sobre la versión del compilador, modelo de microprocesador, cantidad de líneas del programa fuente.
En DOS usaremos el MPASM.EXE y en Windows se usa el MPASMWIN.EXE
Esta es la pantalla de control para el MPASM.EXE:



En la parte inferior se ven las teclas para manejarlo, y una explicación del cuadro en que estamos parados.
Source File: Escriba el nombre del archivo del programa fuente o presione <enter> para mostrar la lista de archivos .ASM de la carpeta actual. Solo se pueden buscar los archivos en esa carpeta o subcarpetas, no subir hacia el directorio raíz.
"*Type the name of your source file*" Escriba el nombre de su archivo fuente.
Processor Type: Presione enter hasta seleccionar el procesador adecuado o deje "None" si el programa fuente tiene el comando *LIST*
"*Press Enter to change value*" Presione <Enter> para cambiar el valor. Esta versión contiene más de 100 modelos de chips.
Error File: Seleccione "Yes" para generar archivo con los errores, avisos y mensajes <SourceFile>.ERR, por defecto es "YES".
"*Press Enter to change value*" Presione <Enter> para cambiar el valor.
Cross Reference File: Seleccione "Yes" para generar <SourceFile>.XRF, por defecto es "No". Es el listado de todos los macros y librerías invocadas en el archivo fuente.
"*Press Enter to change value*" Presione <Enter> para cambiar el valor.
Listing File: Seleccione "YES" para generar <SourceFile>.LST, archivo que contiene el programa fuente, el programa objeto e información sobre errores y la compilación. Por defecto es "Yes".
"*Press Enter to change value*" Presione <Enter> para cambiar el valor.
Hex Dump File: Seleccionar "INHX32", "INHX8S" o "INHX8M" para generar

<SourceFile>.<Extensión>, archivo usado por el programador para grabar en el sistema. Por defecto es "INHX8M .HEX".
"*Press Enter to change value*" Presione <Enter> para cambiar el valor.
Assembler to Object File: Seleccione "Yes" para generar <SourceFile>.O Por defecto es "No"
"*Press Enter to change value*" Presione <Enter> para cambiar el valor.
Con la tecla F10 se ejecuta la acción de compilar, pasando a una pantalla similar a esta



En este caso se observa que esta ensamblando un archivo llamado EJEMPLO.ASM
Durante la primera pasada (checking) chequea la ortografía y sintaxis de cada línea y calcula el valor de las etiquetas de posición.
Durante el ensamblado (2nda pasada Assembling) convierte las instrucciones en código máquina. Indica el número de líneas que van siendo compiladas del archivo fuente.
En la 3ra etapa (Building Files) escribe, en la misma carpeta en que esta el archivo EJEMPLO.ASM, los siguientes archivos: de listado (EJEMPLO.LST), de errores (EJEMPLO.ERR) y el programa objeto (EJEMPLO.HEX)
Los ERRORS (errores) indican que hay palabras que no son instrucciones o etiquetas no declaradas.
Los WARNING (avisos) indican errores de sintaxis por omisión de algún dato en la instrucción. El ensamblador asume un valor por defecto para el parámetro sin definir.
Los MESSAGES (mensajes) son avisos sobre posibles confusiones del programador. Ayudan a prestar atención sobre ese punto en particular del programa fuente.
Recordemos los pasos hasta ahora:

* Escribir el programa fuente como archivo sin formato y guardarlo con extensión .ASM en la misma carpeta que este compilador.

*Después invocamos el compilador con la línea

C:\>mpasm

Si este archivo tiene la el comando LIST p=<modelo_pic> invocamos de una el compilador con la línea

C:\>mpasm ejemplo.asm

*Ya en la pantalla de control presionamos la tecla <enter> y seleccionamos el archivo a compilar.

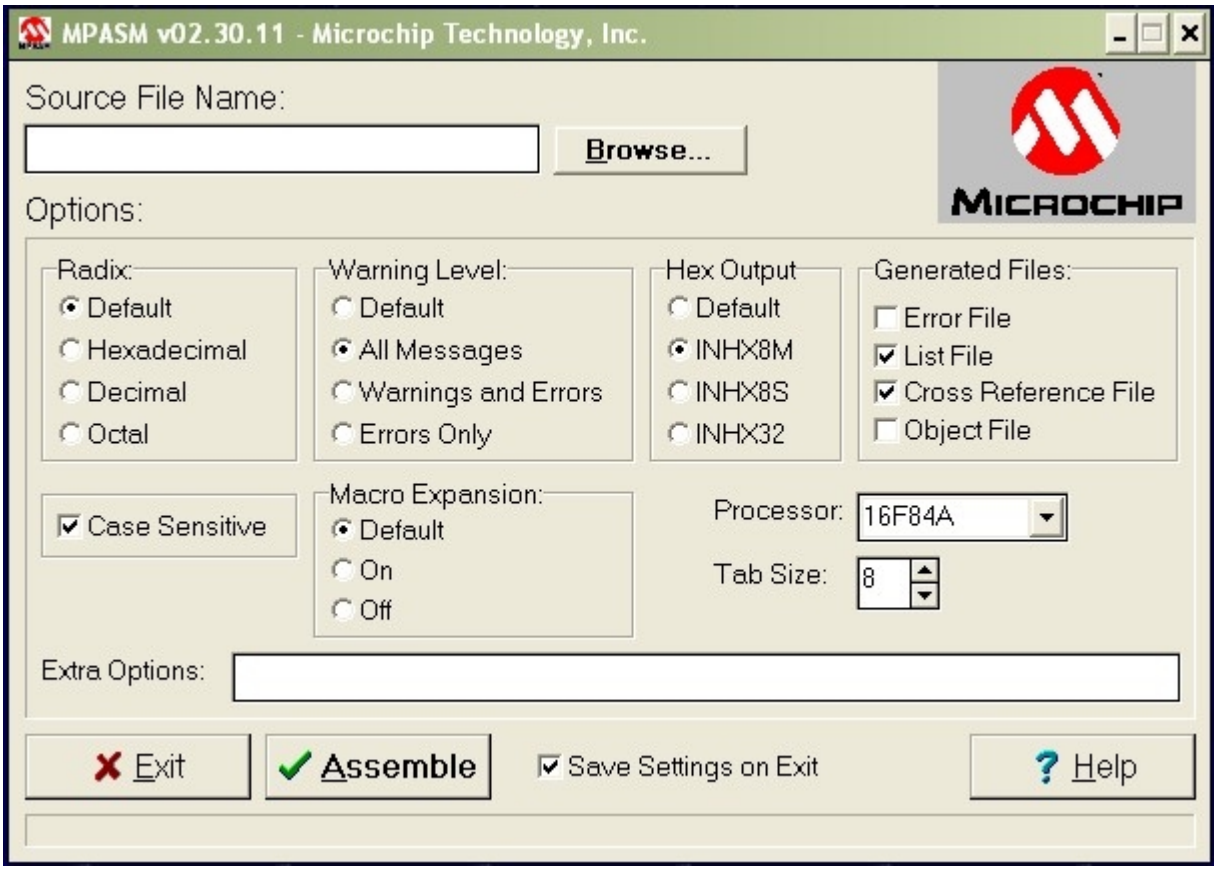
*Cambiamos de cuadro con el cursor descendente y presionamos <enter> hasta hallar el micro deseado

*Para que el ensamblador compile solo es necesario presionar la tecla <F10>

*Cuando termina de presentar la segunda pantalla presionamos cualquier tecla para terminar (Press any key to continue…).

Una vez terminado de ensamblar podemos pasar al simulador o al programador.

Esta es la pantalla de control para el MPASWIN.EXE



El botón BROWSE nos permite navegar por todas las carpetas y todas las unidades para ubicar nuestro archivo fuente. Esta configuración que se muestra es la que se usara para compilar durante el curso.

Este ensamblador tiene una casilla de *CASE SENSITIVE*, lo cual indica que puede diferenciar mayúsculas y minúsculas.

Lo usaremos como esta configurado en la imagen.

Al presionar el botón *ASSEMBLE* comienza el proceso de compilado, se muestra el proceso mediante el cuadro siguiente:

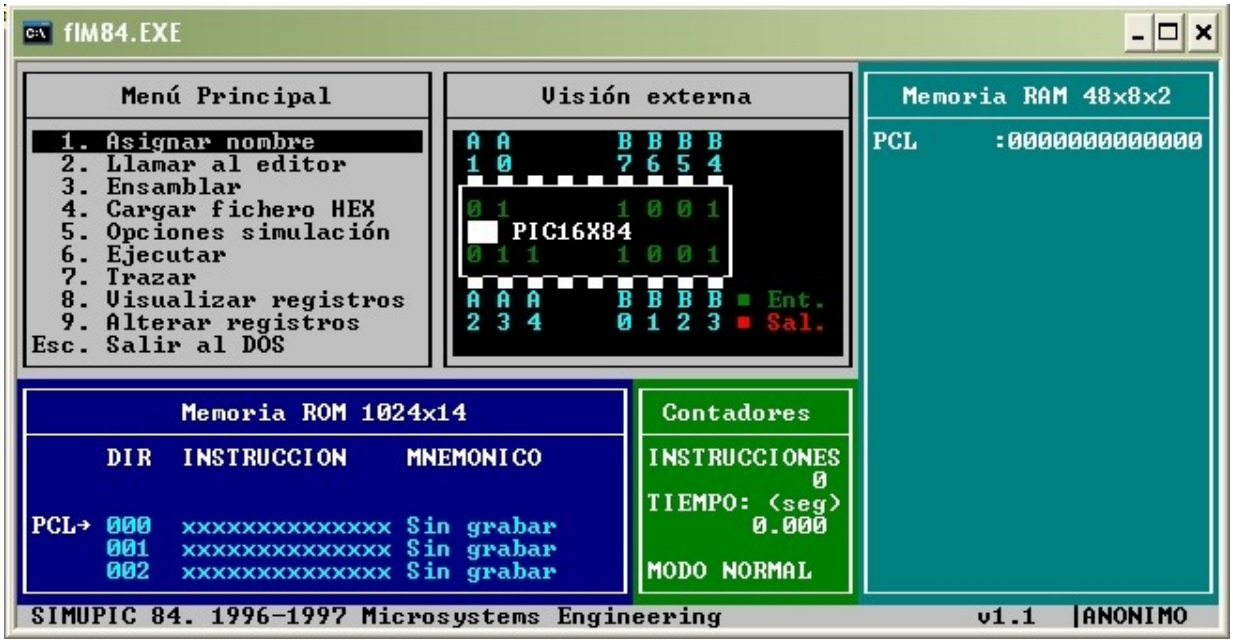
USO DEL SIMULADOR

El programador elegido es el sim84.
Permite simular el PIC16C84 o el PIC16F84. Su ventaja es funcionar en entorno MS-DOS. Pero tiene muchas carencias.
Sus limitaciones son:

No incrementa el TIMER0
No produce interrupciones, es necesario alterar los indicadores INTF, T0IF, EEIF y RBIF manualmente.
Solo maneja las primeras 2F posiciones de cada banco de datos.
No acepta los comandos __CONFIG ni __IDLOCS en el programa fuente.
Es necesario detener la ejecución para alterar los registros de los puertos.
No simula la memoria de datos EEPROM

Tendiendo en cuenta estas limitaciones igual es útil para desarrollar los programas de este curso.
Se invoca con la línea

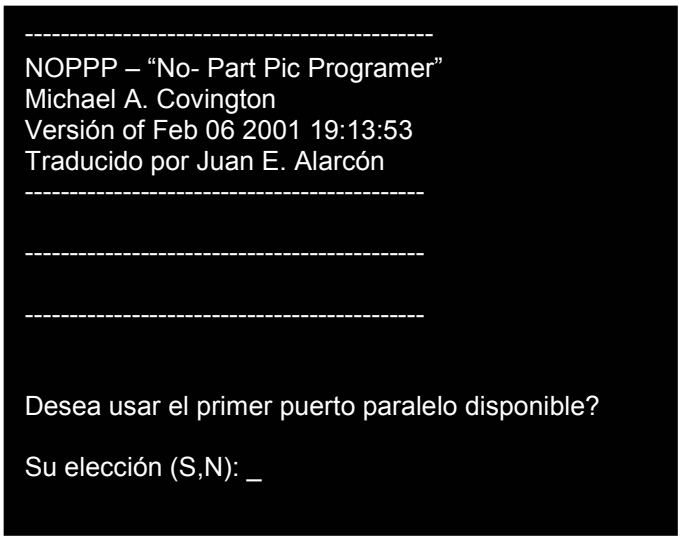
```
c:\pic>sim84
```



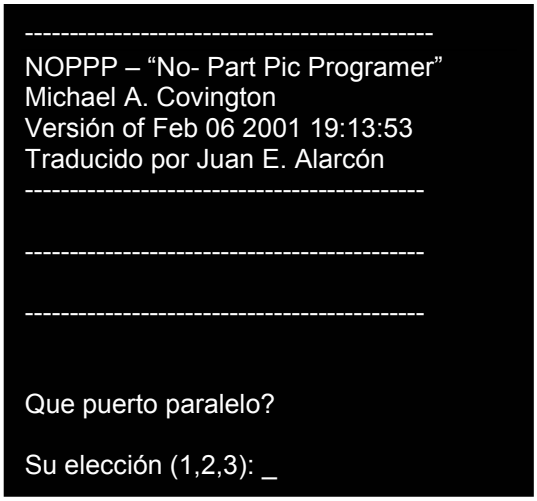
Se despliega esta pantalla
Si tenemos entorno grafico acepta manejarse con el ratón. Cada ítem cuenta con ventanas para seleccionar la función.
El ítem [1. Asignar nombre] permite seleccionar el programa a utilizar. Se carga el archivo fuente de extensión .ASM. Tiene un buscador que le permite recocer todas las carpetas de todas las unidades de disco o disquete.
El ítem [2. Llamar al editor] abre el editor EDIT.COM, permitiendo alterar el programa fuente.
El ítem [3. Ensamblar] invoca al compilador, mpasm.exe, ubicado en la misma carpeta que el ejecutable.
El ítem [4. Cargar fichero HEX] Lo ubica en forma automática a partir de la posición del archivo fuente. Si contiene los comandos __CONFIG o __IDLOCS no lo acepta y da error.
El ítem [5. Opciones Simulación] Permite resetear el PIC, resetar los contadores (de tiempo, de instrucciones, PC), examinar el programa en memoria y configurar el pic.
El ítem [6. Ejecutar] Comienza a ejecutar cada una de las instrucciones.
El ítem [7. Trazar] Ejecuta de a una instrucción por vez con cada pulsación de la tecla <enter>
El ítem [8. Visualizar registros] Permite agregar en el cuadro de la derecha el registro interno que se desee ver cuando varia con el programa.
El ítem [9. Alterar registros] Permite modificar registros internos bit a bit.

Programador o Grabador

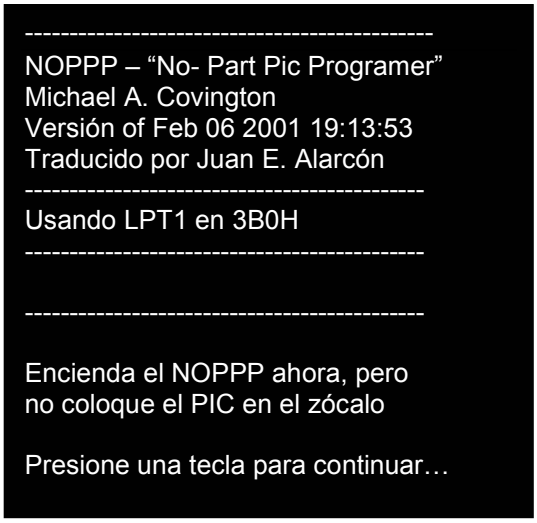
La siguiente herramienta es el programador de pic diseñado por Michael Covington para el sistema operativo DOS
Se invoca con el comando
c:\>nopppes
Se entra en una pantalla así



Aquí se selecciona con que puerto se trabajara
Si se selecciona S el propio programa investiga y utiliza el primer puerto disponible.
En caso de tener, más de un puerto LPT, escribimos N.
Presentando la siguiente pantalla



Una vez elegido un puerto se pasa a la siguiente pantalla



Al presionar una tecla el software verifica si
detecta el hardware. En caso de no detectarlo avisa. Eso a veces sucede, pero igual puede que funcione bien.

En la siguiente pantalla permite seleccionar que modelo de PIC se usara o testear el hardware del programador.

```
Dispositivos soportados:
        C       PIC16C84
        F       PIC16F84
        3       PIC16F83

        P       Probar circuito NOPPP

Su Elección (C, F, 3, P): _
```

La elección, en este curso será < F >, con lo cual se presenta esta pantalla

Nos permite colocar el PIC sin temor a daños por exceso de tensión en el zócalo del programador.

```
---------------------------------------------
NOPPP – "No- Part Pic Programer"
Michael A. Covington
Versión of Feb 06 2001 19:13:53
Traducido por Juan E. Alarcón
---------------------------------------------
Usando LPT1 en 3B0H
---------------------------------------------
PIC16F84
---------------------------------------------


Ahora puede colocar el PIC en el zócalo


Presione una tecla para continuar…
```

Después de presionar una tecla se despliega el siguiente menú:

```
  C     Cargar un archivo HEX
  E     Elegir el tipo de PIC
  B     Borrar PIC
  P     Programar PIC
  V     Verificar PIC
  A     Palabra de Configuración

  S     Salir del programa
Su elección (C,E,B,P,V,A,S): _
```

Explicación de cada comando:

C Permite cargar el archivo objeto a grabar en el chip. Se debe escribir el nombre completo del archivo, incluida la extensión que esta deberá ser .HEX. Recordemos que solo se podrán cargar archivos que estén ubicados en el mismo directorio o carpeta que el NOPPPES.

E Si deseamos cambiar de modelo de PIC, nos regresa a la pantalla anterior.

B Borra la memoria ROM, IDLOCS y Palabra de Configuración del chip.

P Graba el archivo cargado al chip. Quema la ROM, EEPROM, IDLOCS y Palabra de Configuración del PIC. A medida que va grabando una posición la lee para verificar si se quemo adecuadamente. En caso contrario se detiene e indica error.

V Compara lo grabado en el PIC con el archivo cargado. Permite verificar si este quedo bien quemado. Va leyendo cada posición y si hay algún error señala donde se produce la primera diferencia.

A Permite seleccionar los Bit de Configuración si el programa no los define o modificarlos si se desea.

S Habilita a sacar el chip con seguridad y sale al sistema operativo.

Ejemplo de uso

-Con el comando **C** cargamos el archivo objeto, el cual deberá estar previamente en la misma carpeta. Se debe escribir nombre y la extensión (*.HEX) del archivo objeto y el camino si esta en subcarpetas u otra unidad, no permite subir hacia el directorio raíz. Una vez cargado indica la cantidad de instrucciones, IDLOCS, Configuración y EEPROM

```
Archivo a cargar: walk.hex
Memoria de programa cargada:          17 word(s)
Configuración cargada:                1 word(s)
Memoria de Identificación cargada     4 word(s)
Memoria de datos cargada:             29 word(s)
Carga completa.

Presione una tecla para continuar...
```

La memoria de identificación son los IDLOCS, configuración son los Bit de Configuración, la memoria de datos son los que se cargaran en la EEPROM.
En caso de no tener cargada la configuración del sistema se puede cargar o modificar con el comando **A**.
Se vuelve a la pantalla anterior de control.

-Ya se puede quemar el PIC, usando el comando **P**. Mientras es quemado muestra la dirección donde esta trabajando, el valor que esta grabando y el valor que leyó después de grabarla.

-Después se puede verificar la grabación con el comando **V**.

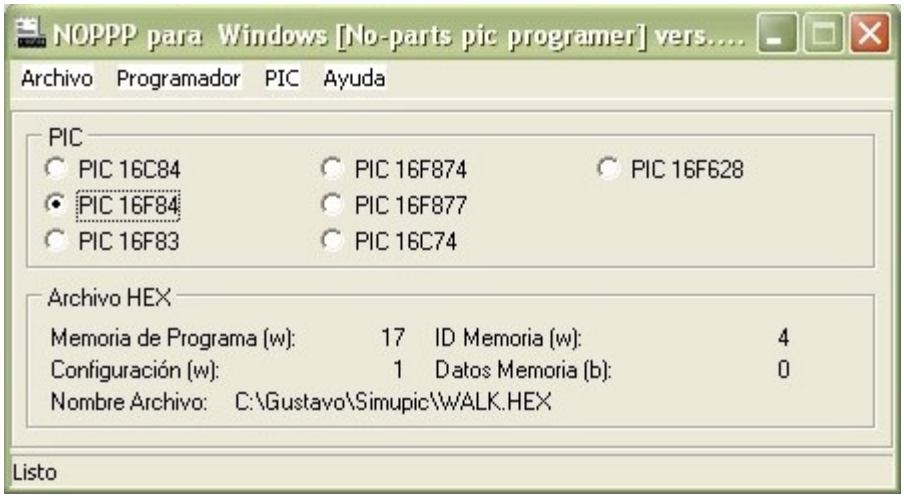
-Solo falta salir adecuadamente del programa. Al presionar el comando **S** permite que se saque el PIC con seguridad. Después de terminar es aleatorio el estado de la tensión en el zócalo del programador.

```
---------------------------------------------
NOPPP – "No- Part Pic Programer"
Michael A. Covington
Versión of Feb 06 2001 19:13:53
Traducido por Juan E. Alarcón
---------------------------------------------
Usando LPT1 en 3B0H
---------------------------------------------
PIC16F84            walk.hex
---------------------------------------------
C       Cargar un archivo HEX
E       Elegir el tipo de PIC
B       Borrar el PIC
P       Programar PIC
V       Verificar PIC
A       Palabra de configuración

S       Salir del programa
```

Ahora se esta en condiciones de colocar el chip en el sistema y realizar la prueba de campo.

La versión NOPPPWIN.EXE para Windows presenta la siguiente pantalla.



La búsqueda del programa objeto de extensión *.HEX s puede realizar en todas las carpetas y unidades. Tiene mayor número de chip a programar con el hardware del noppp. Indica del programa cargado la cantidad de palabras de programa, configuración ,IDLOCS y la cantidad de bytes de la memoria de datos.
Ambas versiones permiten detectar y probar el hardware.