Universidad Autónoma Chapingo

Departamento de Mecánica Agrícola

Ingeniería Mecatrónica Agrícola

Informe 2: MODELOS DE PROGRAMACIÓN DE LOS SISTEMAS EMBEBIDOS

Asignatura:

Sistemas digitales embebidos.

Nombre del profesor:

Luis Arturo Soriano Avendaño

Alumno:

Cocotle Lara Jym Emmanuel [1710451-3]

GRADO:	GRUPO:
6°	7

Fecha de entrega: 27/06/2021

Chapingo, Texcoco Edo. México

Contenido

Introducción

Un Sistema Embebido es un sistema electrónico diseñado para realizar pocas funciones en tiempo real, según sea el caso. Al contrario de lo que ocurre con las computadoras, las cuales tienen un propósito general, ya que están diseñadas para cubrir un amplio rango de necesidades y los Sistemas Embebidos se diseñan para cubrir necesidades específicas.

En un Sistema Embebido la mayoría de los componentes se encuentran incluidos en la placa base (la tarjeta de video, audio, módem) y muchas veces los dispositivos resultantes no tienen el aspecto de lo que se suele asociar a una computadora.

Los Sistemas Embebidos a pesar de no ser muy nombrados están en muchas partes, en realidad, es difícil encontrar algún dispositivo cuyo funcionamiento no esté basado en algún sistema embebido, desde automóviles hasta teléfonos celulares e incluso en algunos electrodomésticos comunes como refrigeradores y hornos de microondas.

Los Sistemas Embebidos suelen tener en una de sus partes una computadora con características especiales conocida como microcontrolador que viene a ser el cerebro del sistema. Este no es más que un microprocesador que incluye interfaces de entrada/salida en el mismo chip. Normalmente estos sistemas poseen una interfaz externa para efectuar un monitoreo del estado y hacer un diagnóstico del sistema [1].

Como sabemos, el único lenguaje que son capaces de entender los microcontroladores es el llamado "Código máquina", el cual se encuentra formado por los ceros y unos del sistema binario. Esto por supuesto dificulta mucho, o imposibilita, mejor dicho, la tarea de programar estos procesadores. Por este motivo existen los lenguajes ensambladores, mediante los cuales el programador puede hacer su tarea gracias a que se trata de un lenguaje con el cual se puede expresar con más naturalidad, pero que a su vez puede ser "entendido" por el microprocesador que se está programando.

Los lenguajes ensambladores llevan mucho tiempo entre nosotros, y aunque en la actualidad existan lenguajes de programación de alto nivel, todavía se sigue utilizando el lenguaje ensamblador para la programación de dispositivos, controladores de hardware y mucho más. Es por ello que en este artículo conoceremos todo acerca de los lenguajes ensambladores, desde sus comienzos hasta sus aplicaciones actuales, siempre con de manera clara y precisa para poder ser entendidos por todos, independientemente de su nivel de conocimientos en estos temas [2].

A través de este informe se pretende dar a conocer las diferencias y características entre el lenguaje de alto nivel y el lenguaje ensamblador, los cuales so utilizados para programar un microcontrolador.

Desarrollo

Lenguaje ensamblador en Sistemas Embebidos.

Los Sistemas Embebidos suelen tener en una de sus partes una computadora con características especiales conocida como microcontrolador que viene a ser el cerebro del sistema. Este no es más que un microprocesador que incluye interfaces de entrada/salida en el mismo chip. Normalmente estos sistemas poseen una interfaz externa para efectuar un monitoreo del estado y hacer un diagnóstico del sistema.

Por lo general, los Sistemas Embebidos se pueden programar directamente en el lenguaje ensamblador del microcontrolador o microprocesador incorporado sobre el mismo, o también, utilizando los compiladores específicos que utilizan lenguajes como C o C++ y en algunos casos, cuando el tiempo de respuesta de la aplicación no es un factor crítico, también pueden usarse lenguajes interpretados como Java [1].

Modos de direccionamiento.

Un modo de direccionamiento se refiere a cómo se dirige a una ubicación de memoria determinada. Hay cinco formas diferentes o cinco modos de direccionamiento para ejecutar esta instrucción, que son las siguientes:

Modo de direccionamiento inmediato

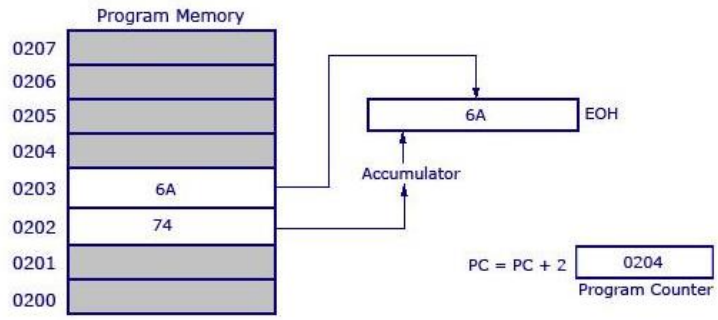
En general, podemos escribir,

```
MOV A, #data
```

Se denomina inmediato porque los datos de 8 bits se transfieren inmediatamente al acumulador (operando de destino).

La siguiente ilustración describe las instrucciones anteriores y su ejecución. El código de operación 74H se guarda en la dirección 0202. Los datos 6AH se guardan en la dirección 0203 en la memoria del programa. Después de leer el código de operación 74H, los datos en la siguiente dirección de memoria del programa se transfieren al acumulador A (E0H es la dirección del acumulador). Como la instrucción es de 2 bytes y se ejecuta en un ciclo, el contador del programa se incrementará en 2 y apuntará a 0204 de la memoria del programa.

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1



1.- Modo de direccionamiento inmediato.

El símbolo '#' antes de 6AH indica que el operando es un dato (8 bits). En ausencia de '#', el número hexadecimal se tomaría como una dirección [3].

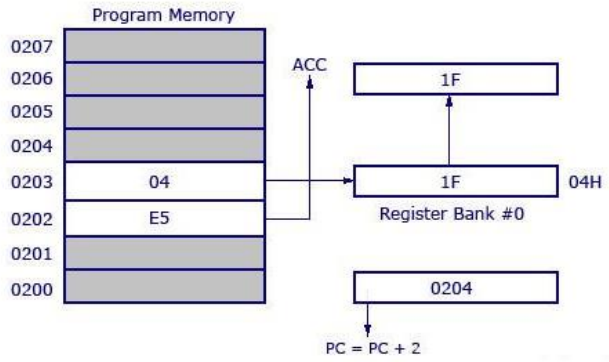
Modo de direccionamiento directo

Esta es otra forma de abordar un operando. Aquí, la dirección de los datos (datos de origen) se proporciona como un operando.

```
MOV A, 04H
```

No hemos usado '#' en modo de direccionamiento directo, a diferencia del modo inmediato. Si hubiéramos usado '#', el valor de datos 04H se habría transferido al acumulador en lugar de 1FH.

Instruction	Opcode	Bytes	Cycles
MOV A, #04H	E5	2	1



2.- Modo de direccionamiento directo

Esta es una instrucción de 2 bytes que requiere 1 ciclo para completarse. La PC se incrementará en 2 y apuntará a 0204. El código de operación para la instrucción MOV A, dirección es E5H. Cuando se ejecuta la instrucción en 0202 (E5H), el acumulador se activa y está listo para recibir datos. Luego, la PC pasa a la siguiente dirección como 0203 y busca la dirección de la ubicación de 04H donde se encuentran los datos de origen (que se transferirán al acumulador). En 04H, el control encuentra los datos 1F y los transfiere al acumulador y, por lo tanto, se completa la ejecución [3].

Registrar el modo de direccionamiento directo

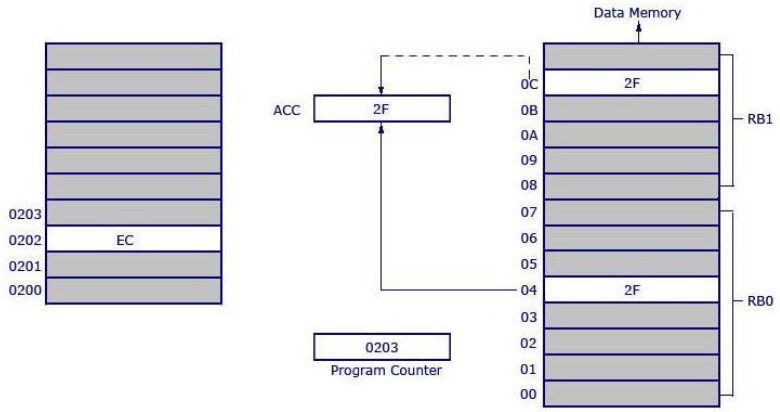
En este modo de direccionamiento, usamos el nombre del registro directamente (como operando de origen).

```
MOV A, R4
```

En un momento, los registros pueden tomar valores de R0 a R7. Hay 32 de esos registros. Para utilizar 32 registros con solo 8 variables para direccionar registros, se utilizan bancos de registros. Hay 4 bancos de registros nombrados de 0 a 3. Cada banco consta de 8 registros nombrados de R0 a R7.

A la vez, se puede seleccionar un solo banco de registro. La selección de un banco de registro es posible a través de un Registro de funciones especiales (SFR) denominado Palabra de estado del procesador (PSW). PSW es un SFR de 8 bits donde cada bit se puede programar según sea necesario. Los bits se designan de PSW.0 a PSW.7. PSW.3 y PSW.4 se utilizan para seleccionar bancos de registro.

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



3.- Registrar el modo de direccionamiento directo

Opcode EC se utiliza para MOV A, R4. El código de operación se almacena en la dirección 0202 y cuando se ejecuta, el control va directamente a R4 del banco de registro respetado (que se selecciona en PSW). Si se selecciona el banco de registro # 0, los datos de R4 del banco de registro # 0 se moverán al acumulador. Aquí 2F se almacena a las 04H. 04H representa la dirección de R4 del banco de registro # 0.

El movimiento de datos (2F) se resalta en negrita. 2F se transfiere al acumulador desde la ubicación de memoria de datos 0C H y se muestra como una línea de puntos. 0CH es la ubicación de la dirección del Registro 4 (R4) del banco de registro # 1. La instrucción anterior es de 1 byte y requiere 1 ciclo para la ejecución completa. Lo que significa es que puede guardar la memoria del programa utilizando el modo de direccionamiento directo de registro [3].

Registrar modo de direccionamiento indirecto

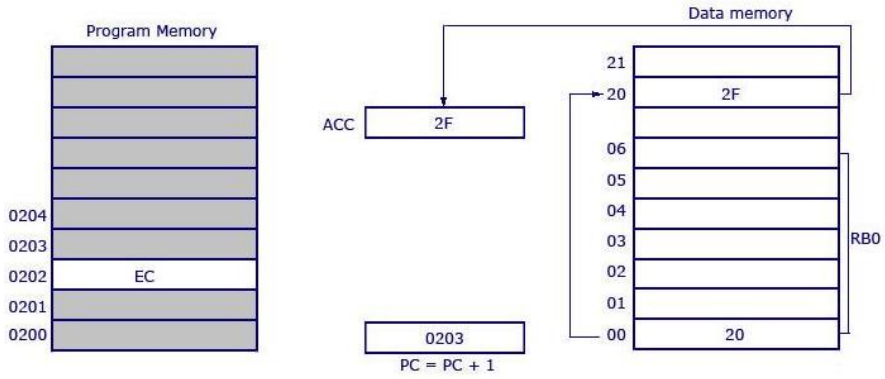
En este modo de direccionamiento, la dirección de los datos se almacena en el registro como operando.

```
MOV A, @R0
```

Aquí el valor dentro de R0 se considera como una dirección, que contiene los datos que se transferirán al acumulador. Ejemplo: si R0 tiene el valor 20H y los datos 2FH se

almacenan en la dirección 20H, entonces el valor 2FH se transferirá al acumulador después de ejecutar esta instrucción.

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



4.- Registrar modo de direccionamiento indirecto.

Entonces, el código de operación para MOV A, @ R0 es E6H. Suponiendo que el banco de registro # 0 esté seleccionado, el R0 del banco de registro # 0 contiene los datos 20H. El control del programa se mueve a 20H donde ubica los datos 2FH y transfiere 2FH al acumulador. Esta es una instrucción de 1 byte y el contador del programa aumenta en 1 y se mueve a 0203 de la memoria del programa.

Solo R0 y R1 pueden formar una instrucción de direccionamiento indirecto de registro. En otras palabras, el programador puede crear una instrucción usando @ R0 o @ R1. Todos los bancos de registro están permitidos [3].

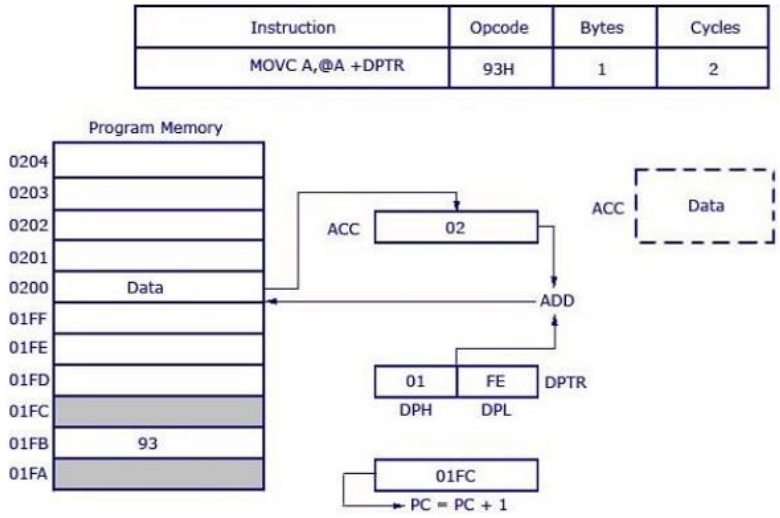
Modo de direccionamiento indexado

Tomaremos dos ejemplos para comprender el concepto del modo de direccionamiento indexado.

MOVC A, @ A + DPTR y MOVC A, @ A + PC

Donde DPTR es el puntero de datos y PC es el contador del programa (ambos son registros de 16 bits).

En el primer caso el operando de origen es @ A + DPTR. Contiene los datos de origen de esta ubicación. Aquí estamos agregando el contenido de DPTR con el contenido actual del acumulador. Esta adición dará una nueva dirección, que es la dirección de los datos de origen. Los datos señalados por esta dirección se transfieren al acumulador.



5.- Modo de direccionamiento indexado.

El código de operación es 93H. DPTR tiene el valor 01FE, donde 01 está ubicado en DPH (8 bits superiores) y FE está ubicado en DPL (8 bits inferiores). El acumulador tiene el valor 02H. Luego se realiza una adición de 16 bits y 01FE H + 02H da como resultado 0200 H. Los datos en la ubicación 0200H se transferirán al acumulador. El valor anterior dentro del acumulador (02H) será reemplazado con los nuevos datos de 0200H.

Esta es una instrucción de 1 byte con 2 ciclos necesarios para la ejecución y el tiempo de ejecución requerido para esta instrucción es alto en comparación con las instrucciones anteriores (que eran todas de 1 ciclo cada una).

El otro ejemplo MOVC A, @ A + PC funciona de la misma manera que el ejemplo anterior. En lugar de agregar DPTR con el acumulador, aquí los datos dentro del contador del programa (PC) se agregan con el acumulador para obtener la dirección de destino [3].

Conjunto de instrucciones.

CISC (Complex instruction set computing)

En los años setenta los procesadores eran diseñados considerando las limitaciones en la transferencia de datos entre el procesador y la memoria. Bajo esa óptica, no tenía mucho sentido el hacer procesadores más rápidos si la velocidad de acceso estaba acotada por la tecnología de las propias memorias.

Los registros eran difíciles y costosos de implementar debido a sus interconexiones. Mínimo número de registros o registros enfocados a tareas específicas Tenía sentido diseñar procesadores que minimizaran la transferencia de información a las localidades de memoria se realizara sin intermediarios.

Los diseñadores se enfocaron en crear conjuntos de instrucciones específicas para cada posible operación o combinación de entrada/salida de registros o memoria.

La meta era proveer cada posible modo de direccionamiento (combinación de registros) para cada instrucción. Un principio conocido como ortogonalidad. Esto condujo a un CPU complejo, pero en teoría capaz de configurar cada posible orden individualmente, haciendo la ejecución más rápida en comparación del uso de un grupo de órdenes simples.

Ya que un set de instrucciones complejo como el CISC dificulta la ejecución de operaciones en paralelo, en la actualidad la mayoría de los sistemas CISC de alto rendimiento implementan un sistema de conversión de instrucciones complejas en conjuntos de instrucciones simples (como las del tipo usado en los procesadores RISC), llamadas microinstrucciones.

El modelo de arquitectura de computadora CISC de conjunto de instrucciones complejo, se caracteriza por tener muchas instrucciones y permitir en operaciones individuales operaciones complejas entre datos situados en la memoria y/o en los registros internos.

Este tipo de arquitectura busca conseguir un aumento de prestaciones con base en el incremento de la complejidad de las instrucciones.

Los CISC pertenecen a la primera corriente de construcción de procesadores. Ejemplos de ellos son: Motorola 68000, Zilog Z80 y toda la familia Intel x86. Los procesadores CISC dominan el mercado de los ordenadores personales, debido a su popularidad y al aumento constante en su capacidad de procesamiento [4].

SISC (Simple Instruction Set Computing)

Es un tipo de arquitectura de microprocesadores orientada al procesamiento de tareas en paralelo la cual aprovecha las tecnologías de fabricación VLSI, con lo que se permite integrar múltiples dispositivos de bajo costo en una sola pastilla [4].

RISC (Reduced instruction set computing)

La arquitectura RISC es un diseño de CPU que está basado en conjuntos de instrucciones simples, con muy pocos modos de direccionamiento, lo que posibilita el paralelismo y la segmentación de rutinas.

Los diseños RISC tienden a utilizar un modelo de arquitectura Harvard, donde los buses de instrucciones y datos están separados, lo mejora el rendimiento, al permitir el acceso a los cachés de datos e instrucciones de manera independiente. Su funcionamiento simple los hace idóneos para aplicaciones de bajo consumo de energía o de tamaño limitado, además permite su fabricación en líneas de producción poco sofisticadas (lo que permite la reutilización de líneas de producción).

Objetivos del desarrollo de los RISC:

- Velocidad: Un número pequeño de instrucciones (opcode) permite describirlas con pocos bits, dejando espacio dentro de la propia palabra o instrucción para incluir en ella datos o constantes, reduciéndose el número de accesos a registros o memoria para completar la información.
- Paralelismo: Instrucciones de longitud fija hace que las instrucciones (opcodes) y operandos siempre estén ubicados en los mismos campos de bits, alineados dentro de los límites de la unidad lógica binaria del procesador (word boundaries), lo que facilita el uso de pipelines (VHDL).
- Simplicidad: Modos de direccionamiento sencillos permiten definir un conjunto de registros casi homogéneo con sólo algunos de ellos dedicados a las operaciones de escritura/lectura con la memoria externa, lo que facilita el diseño de la interfaz de memoria y simplifican el diseño del procesador.

Características RISC

- Conjunto de instrucciones Load/Store (Cargar/Almacenar): Sólo estas instrucciones permiten el acceso a la memoria; las demás operaciones utilizan el conjunto de registros para almacenar los resultados. Lo anterior simplifica el direccionamiento y acorta los tiempos de acceso del CPU. También se facilita la recuperación de fallos en páginas (page faults) cuando se emplea algún esquema de memoria virtual.
- Arquitectura no destructiva de tres direcciones: Los procesadores CISC destruyen la información que existe en alguno de los registros durante la ejecución normal de instrucciones. Las instrucciones RISC (de tres direcciones) contienen dos campos para operandos y uno independiente para el resultado. Esta arquitectura "no destructiva" permite a los compiladores organizar las instrucciones de modo que mantengan llenos los conductos (pipelines) del chip, y por tanto reutilizar los operandos optimizando la concurrencia.
- Ausencia de microcódigo: A diferencia de los procesadores CISC donde el uso de microcódigo obliga al hardware a interpretar cada instrucción de manera dinámica (impidiendo también la implementación de instrucciones de ciclos único); en los procesadores RISC las funciones y el control de registros están implementados por hardware (hardwired) con lo que se obtienen una máxima velocidad y eficiencia.
- Ejecución en conductos (pipelined): El tamaño predeterminado de las instrucciones RISC permite que las distintas etapas del ciclo de operación del CPU (fetch, decoding, execution y result write-back) puedan realizarse en paralelo conforme el contador de programa avanza en el código.

- Ejecución en ciclos únicos (single-cycle): Algunas instrucciones puede ser ejecutadas en un único ciclo de la CPU. La ejecución en ciclos únicos también simplifica la gestión de las interrupciones y de pipelines [4].

Instrucciones aritméticas.

add destino, fuente

Suma los contenidos de fuente y destino y guarda el resultado en destino.

Sintaxis:

add regB, inmB|regB| memB

add memB, inmB|regB

add regW, inmB|inmW| regW|memW

add memW, inmB|inmW| regW

La instrucción add modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

adc destino, fuente

Suma los contenidos de fuente, destino y el valor de la bandera de acarreo y guarda el resultado en destino.

Sintaxis:

adc regB, inmB|regB| memB

adc memB, inmB|regB

adc regW, inmB|inmW| regW|memW

adc memW, inmB|inmW| regW

La instrucción adc modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

Al sumar valores multibytes o multipalabras, use add para sumar los bytes o palabras menos significativos y después use la instrucción adc para sumar los bytes o palabras más significativos y los posibles acarreos generados por las sumas de los bytes o palabras menos significativos.

inc destino

Incrementa en uno el contenido de destino.

Sintaxis:

inc regB| memB

inc regW| memW

La instrucción inc modifica las siguientes banderas: sobreflujo O, signo S, cero Z, acarreo auxiliar A y paridad P.

sub destino, fuente

Resta al valor de destino el valor de fuente y guarda el resultado en destino.

Sintaxis:

sub regB, inmB|regB| memB

sub memB, inmB|regB

sub regW, inmB|inmW| regW|memW

sub memW, inmB|inmW| regW

La instrucción sub modifica las siguientes banderas: sobreflujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

sbb destino, fuente

Resta al valor de destino el valor de fuente tomando en consideración un posible préstamo de una instrucción sbb o sub previa y guarda el resultado en destino.

Sintaxis:

sbb regB, inmB|regB| memB

sbb memB, inmB|regB

sbb regW, inmB|inmW| regW|memW

sbb memW, inmB|inmW| regW

La instrucción sbb modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

Al restar valores multibytes o multipalabras, use sub para restar los bytes o palabras menos significativos y después use la instrucción sbb para restar los bytes o palabras más significativos tomando en cuenta los posibles préstamos de las restas de los bytes o palabras menos significativos.

cmp destino, fuente

Compara los contenidos de fuente, y destino.

Sintaxis:

cmp regB, inmB|regB| memB

cmp memB, inmB|regB

cmp regW, inmB|inmW| regW|memW

cmp memW, inmB|inmW| regW

La instrucción cmp modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

Sólo uno de los operandos puede ser una localidad de memoria. La instrucción cmp trabaja restando el valor de fuente de destino y tirando el resultado, pero afectando las banderas.

dec destino

Decrementa en uno el contenido de destino.

Sintaxis:

dec regB| memB

dec regW|memW

La instrucción dec modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A y paridad P.

mul fuente

Multiplica dos operandos sin signo.

Sintaxis:

mul regB |memB

mul regW|memW

Si la multiplicación es de un byte por un byte, los operandos y el resultado están en:

Multiplicando: AL

Multiplicador: fuente (Un byte)

Producto: AX

Si la multiplicación es de una palabra por una palabra, los operandos y el resultado están en:

Multiplicando: AX

Multiplicador: fuente (Una palabra)

Producto: DX:AX (La palabra menos significativa en AX)

Después de la instrucción mul las siguientes banderas quedan en un estado indefinido: signo S, cero Z, acarreo auxiliar A y paridad P.

Si las banderas de acarreo C y de sobre flujo O son ambas 0 después de la operación entonces la parte

más significativa del resultado es 0. Si las banderas de acarreo C y de sobre flujo O son ambas 1 después de la operación entonces el resultado ocupa todo el (los) registro(s) del producto.

imul fuente

Multiplica dos operandos con signo.

Sintaxis:

imul regB|memB

imul regW|memW

Si la multiplicación es de un byte por un byte, los operandos y el resultado están en:

Multiplicando: AL

Multiplicador: fuente (Un byte)

Producto: AX

Si la multiplicación es de una palabra por una palabra, los operandos y el resultado están en:

Multiplicando: AX

Multiplicador: fuente (Una palabra)

Producto: DX:AX (La palabra menos significativa en AX)

Después de la instrucción imul las siguientes banderas quedan en un estado indefinido: signo S, cero Z, acarreo auxiliar A y paridad P.

Si las banderas de acarreo C y de sobre flujo O son ambas 0 después de la operación entonces la parte más significativa del resultado es simplemente la extensión del signo de la parte menos significativa. Si las banderas de acarreo C y de sobre flujo O son ambas 1 después de la operación entonces el resultado ocupa todo el (los) registro(s) del producto.

div fuente

Divide dos operandos sin signo.

Sintaxis:

div regB|memB

div regW|memW

Si la división es de una palabra entre un byte, los operandos y los resultados están en:

Dividendo: AX

Divisor: fuente (Un byte)

Cociente: AL

Residuo: AH

Si la división es de una palabra doble entre una palabra, los operandos y los resultados están en:

Dividendo: DX:AX (La palabra menos significativa en AX)

Divisor: fuente (Una palabra)

Cociente: AX

Residuo: DX

Después de la instrucción div las siguientes banderas quedan en un estado indefinido: sobre flujo O, signo S, cero Z, acarreo auxiliar A y paridad P y acarreo C.

Si el resultado de la división es mayor que el valor máximo permitido en el registro donde se almacena el cociente o si el divisor vale cero, se genera una interrupción tipo 0: división entre 0. El programa se detiene y despliega el mensaje de división entre 0.

idiv fuente

Divide dos operandos con signo.

Sintaxis:

idiv regB|memB

idiv regW|memW

Si la división es de una palabra entre un byte, los operandos y los resultados están en:

Dividendo: AX

Divisor: fuente (Un byte)

Cociente: AL

Residuo: AH

Si la división es de una palabra doble entre una palabra, los operandos y los resultados están en:

Dividendo: DX:AX (La palabra menos significativa en AX)

Divisor: fuente (Una palabra)

Cociente: AX

Residuo: DX

Después de la instrucción idiv las siguientes banderas quedan en un estado indefinido: sobre flujo O, signo S, cero Z, acarreo auxiliar A y paridad P y acarreo C.

Si el resultado de la división es mayor que el valor máximo permitido en el registro donde se almacena el cociente o si el divisor vale cero, se genera una interrupción tipo 0: división entre 0. El programa se detiene y despliega el mensaje de división entre 0.

cbw

Convierte un byte con signo a una palabra por extensión de signo.

Sintaxis:

cbw

La instrucción cbw no afecta el estado de las banderas.

Utilice cbw para extender un valor de 8 bits con signo en el registro AL a un valor de 16 bits con signo de la misma magnitud en AX. La instrucción trabaja copiando el bit más significativo de AL a todos los bits de AH, esto es, colocando 0FFh en AH si AL es negativo o colocando 0h en AH si AL es positivo.

cwd

Convierte una palabra con signo a una palabra doble por extensión de signo.

Sintaxis:

cwd

La instrucción cwd no afecta el estado de las banderas.

Utilice cwd para extender un valor de 16 bits con signo en el registro AX a un valor de 32 bits con signo de la misma magnitud en DX:AX. La instrucción trabaja copiando el bit más significativo de AX a todos los bits de DX, esto es, colocando 0FFFFh en DX si AX es negativo o colocando 0h en DX si AX es positivo.

neg destino

Obtiene el negativo de destino (forma el complemento de 2).

Sintaxis:

neg regB |memB

neg regW|memW

La instrucción neg modifica las siguientes banderas: sobre flujo O, signo S, cero Z, acarreo auxiliar A, paridad P y acarreo C.

Instrucciones lógicas.

not destino

Obtiene el complemento de 1 de destino.

Sintaxis:

not regB |memB

not regW|memW

La instrucción not deja inalteradas a las banderas.

and destino, fuente

Calcula la intersección lógica entre los bits de fuente y destino y guarda el resultado en destino.

Sintaxis:

and regB, inmB|regB |memB

and memB, inmB|regB

and regW, inmB|inmW |regW|memW

and memW, inmB|inmW |regW

La instrucción and modifica las siguientes banderas: sobre flujo O = 0, signo S, cero Z, paridad P y acarreo C = 0. La bandera de acarreo auxiliar A queda en un estado indefinido.

test destino, fuente

Compara valores mediante la intersección lógica entre los bits de fuente y destino.

Sintaxis:

test regB, inmB| regB| memB

test memB, inmB| regB

test regW, inmW|regW| memW

test memW, inmW|regW

La instrucción test trabaja en forma idéntica a la instrucción and excepto que el resultado de la operación se descarta y sólo se afectan las banderas.

La instrucción test modifica las siguientes banderas: sobre flujo O = 0, signo S, cero Z, paridad P y acarreo C = 0. La bandera de acarreo auxiliar A queda en un estado indefinido.

or destino, fuente

Calcula la unión inclusiva lógica entre los bits de fuente y destino y guarda el resultado en destino.

Sintaxis:

or regB, inmB|regB |memB

or memB, inmB|regB

or regW, inmB|inmW |regW|memW

or memW, inmB|inmW |regW

La instrucción or modifica las siguientes banderas: sobre flujo O = 0, signo S, cero Z, paridad P y acarreo C = 0. La bandera de acarreo auxiliar A queda en un estado indefinido.

xor destino, fuente

Calcula la unión exclusiva lógica entre los bits de fuente y destino y guarda el resultado en destino.

Sintaxis:

xor regB, inmB|regB |memB

xor memB, inmB|regB

xor regW, inmB|inmW |regW|memW

xor memW, inmB|inmW |regW

La instrucción xor modifica las siguientes banderas: sobre flujo O = 0, signo S, cero Z, paridad P y acarreo C = 0. La bandera de acarreo auxiliar A queda en un estado indefinido.

sal | shl destino, cuenta

Corre hacia la izquierda los bits de destino un número de veces dado por cuenta.

Sintaxis:

sal|shl regB |memB, 1|cl

sal|shl regW |memW, 1|cl

Los mnemónicos sal y shl son sinónimos. sal|shl corre hacia la izquierda los bits de destino. Esto es, en cada corrimiento el bit más significativo de destino se mueve a la bandera de acarreo y se inserta un cero como el bit menos significativo de destino, mientras todos los demás bits se mueven una posición a la izquierda. El mnemónico sal se emplea para multiplicar destino por potencias de 2, mientras que el mnemónico shl se emplea para correr los bits de destino a la izquierda. Aunque ambos son intercambiables.

Si el número de corrimientos deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de corrimientos debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción sal|shl modifica la bandera de signo S, cero Z, paridad P y acarreo C. La bandera de acarreo auxiliar queda en un estado indefinido. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobreflujo O, si cuenta es un valor en el registro CL, la bandera de sobreflujo O queda en un estado indefinido.

sar destino, cuenta

Corre hacia la derecha los bits de destino, preservando el signo, un número de veces dado por cuenta.

Sintaxis:

sar regB |memB, 1|cl

sar regW |memW, 1|cl

A diferencia de los mnemónicos sal y shl, sar y shr no son sinónimos. sar corre hacia la derecha los bits de destino preservando el signo. Esto es, en cada corrimiento una copia del bit más significativo de destino se inserta en el bit más significativo. El bit menos significativo se mueve a la bandera de acarreo, mientras todos los demás bits se mueven una posición a la derecha. La instrucción sar se emplea para dividir un valor signado en destino entre potencias de 2.

Si el número de corrimientos deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de corrimientos debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción sar modifica la bandera de signo S, cero Z, paridad P y acarreo C. La bandera de acarreo auxiliar queda en un estado indefinido. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobre flujo O, si cuenta es un valor en el registro CL, la bandera de sobre flujo O queda en un estado indefinido.

shr destino, cuenta

Corre hacia la derecha los bits de destino un número de veces dado por cuenta.

Sintaxis:

shr regB |memB, 1|cl

shr regW |memW, 1|cl

shr corre hacia la derecha los bits de destino. Esto es, en cada corrimiento se inserta un cero en el bit más significativo de destino. El bit menos significativo se mueve a la bandera de acarreo, mientras todos los demás bits se mueven una posición a la derecha. La instrucción shr se emplea para dividir un valor sin signo en destino entre potencias de 2.

Si el número de corrimientos deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de corrimientos debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción shr modifica la bandera de signo S, cero Z, paridad P y acarreo C. La bandera de acarreo auxiliar queda en un estado indefinido. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobre flujo O, si cuenta es un valor en el registro CL, la bandera de sobre flujo O queda en un estado indefinido.

rcl destino, cuenta

Rota hacia la izquierda los bits de destino a través de la bandera de acarreo un número de veces dado por cuenta.

Sintaxis:

rcl regB |memB, 1|cl

rcl regW |memW, 1|cl

rcl rota hacia la izquierda los bits de destino incluyendo a la bandera de acarreo C como parte del valor original. Esto es, en cada rotación el bit más significativo de destino se mueve a la bandera de acarreo y el valor en la bandera de acarreo se inserta como el bit

menos significativo de destino, mientras todos los demás bits se mueven una posición a la izquierda.

Si el número de rotaciones deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de rotaciones debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción rcl modifica la bandera de acarreo C. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobre flujo O, si cuenta es un valor en el registro CL, la bandera de sobre flujo O queda en un estado indefinido.

rcr destino, cuenta

Rota hacia la derecha los bits de destino a través de la bandera de acarreo un número de veces dado por cuenta.

Sintaxis:

rcr regB |memB, 1|cl

rcr regW |memW, 1|cl

rcr rota hacia la derecha los bits de destino incluyendo a la bandera de acarreo C como parte del valor original. Esto es, en cada rotación el bit menos significativo de destino se mueve a la bandera de acarreo y el valor en la bandera de acarreo se inserta como el bit más significativo de destino, mientras todos los demás bits se mueven una posición a la derecha.

Si el número de rotaciones deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de rotaciones debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción rcr modifica la bandera de acarreo C. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobre flujo O, si cuenta es un valor en el registro CL, la bandera de sobre flujo O queda en un estado indefinido.

rol destino, cuenta

Rota hacia la izquierda los bits de destino un número de veces dado por cuenta.

Sintaxis:

rol regB |memB, 1|cl

rol regW |memW, 1|cl

rol rota hacia la izquierda los bits de destino. Esto es, en cada rotación el bit más significativo de destino se inserta como el bit menos significativo de destino, mientras todos los demás bits se mueven una posición a la izquierda. Adicionalmente el bit más significativo del valor original de destino se copia a la bandera de acarreo C.

Si el número de rotaciones deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de rotaciones debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción rol modifica la bandera de acarreo C. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobreflujo O, si cuenta es un valor en el registro CL, la bandera de sobreflujo O queda en un estado indefinido.

ror destino, cuenta

Rota hacia la derecha los bits de destino un número de veces dado por cuenta.

Sintaxis:

ror regB |memB, 1|cl

ror regW |memW, 1|cl

ror rota hacia la derecha los bits de destino. Esto es, en cada rotación el bit menos significativo de destino se inserta como el bit más significativo de destino, mientras todos los demás bits se mueven una posición a la derecha. Adicionalmente el bit menos significativo del valor original de destino se copia a la bandera de acarreo C.

Si el número de rotaciones deseado es de uno, cuenta puede ser el valor inmediato 1. Para otros valores, el número de rotaciones debe asignarse previamente al registro CL y especificar éste como cuenta.

La instrucción ror modifica la bandera de acarreo C. Si cuenta es un uno inmediato, la instrucción también modifica a la bandera de sobre flujo O, si cuenta es un valor en el registro CL, la bandera de sobre flujo O queda en un estado indefinido [5].

Instrucciones de control de programa y manejo de apuntadores.

Saltos incondicionales

La instrucción fundamental de bifurcación en los microprocesadores de la familia 80x86 es la instrucción JMP. La instrucción JMP instruye al 8086 para que ejecute la instrucción en la etiqueta destino como la instrucción que sigue a JMP. Por ejemplo, cuando finaliza la ejecución del siguiente segmento de código.

```
mov    ax,1
```

```
        jmp     AddTwoToAX
AddOneToAX:
        inc     ax
        jmp     AXIsSet
AddTwoToAX:
        add     ax,2
AXIsSet:
```

AX contiene 3, y las instrucciones ADD y JMP que siguen a la etiqueta AddOneToAX nunca son ejecutadas.

```
jmp     AddTwoToAX
```

Instruye al 8086 para que se le asigne al apuntador de instrucciones, el registro IP, el desplazamiento de la etiqueta AddTwoToAX, para que la siguiente instrucción a ejecutar sea:

```
add     ax,2
```

Algunas veces junto con la instrucción JMP se utiliza el operador SHORT. JMP usualmente utiliza un desplazamiento de 16 bits para apuntar a la etiqueta destino; SHORT instruye a TASM para que utilize un desplazamiento de 8 bits, ahorrando un byte por instrucción JMP. Por ejemplo, el último ejemplo es dos bytes más pequeños con las siguientes modificaciones:

```
        jmp     SHORT AddTwoToAX
AddOneToAX:
        inc ax
        jmp SHORT AXIsSet
AddTwoToAX:
        add ax,2
AXIsSet:
```

La desventaja en usar el operador SHORT es que algunos saltos cortos pueden alcanzar a las etiquetas que se encuentran únicamente dentro del alcance de 128 bytes de la instrucción JMP, de tal forma que TASM puede informar que no puede alcanzar una etiqueta determinada debido a que está fuera del alcance de la instrucción JMP que utiliza

un operador SHORT. Unicamente tiene sentido utilizar saltos SHORT cuando se hacen saltos hacia adelante en el código, debido a que TASM inteligentemente hace cortos los saltos hacia atrás cuando están dentro del alcance del destino; los saltos hacia atrás que están más allá de los 128 bytes de la instrucción JMP automáticamente son tomados como saltos con desplazamiento de 16 bits. JMP puede ser utilizado para saltar a otro segmento de código, cargando los registros CS e IP con una sola instrucción.

Saltos condicionales

Los saltos como los descritos en la sección anterior son únicamente parte de lo necesario para escribir programas útiles. La mayor parte de los programas se benefician de los saltos basados en la toma de decisiones. Una instrucción de salto condicional puede saltar o no a una etiqueta destino, dependiendo del estado del registro de banderas. Por ejemplo, considere el siguiente segmento de código:

```
        mov     ah,1            ; #Fn DOS para entrada desde teclado
        int      21h            ; Obtiene la siguiente tecla
        cmp     al,'A'          ; Se presionó la letra 'A'?
        je      AWasTyped       ; Sí, procesa la tecla presionada
        mov     [TempByte],al ; No, almacena el carácter
        ...
AWasTyped:
        push ax
```

Primero, el código obtiene una tecla del teclado a través de la función 1 del DOS. Después, utiliza la instrucción CMP para comparar el carácter leído con la letra 'A'. La instrucción CMP hace que la bandera de cero (ZF) sea 1, si el resultado de la comparación es igual, o sea aclarada (0), si la comparación evalúa a diferente. La instrucción JE es una instrucción de salto condicional que salta a la etiqueta destino sólo si la bandera de cero es 1. De otra forma, se ejecutará la instrucción inmediata a la instrucción JE, en este caso una instrucción MOV. La bandera de cero será 1 sólo cuando se presione la tecla A, y únicamente entonces saltará el 8086 a la instrucción PUSH en la etiqueta AWasTyped.

Nemónico	Significado	Sinónimo	Significado
JA	Salta si es superior	JNBE	Salta si no es inferior o igual
JAE	Salta si es superior o igual	JNB	Salta si no es inferior
JB	Salta si es inferior	JNAE	Salta si no es superior o igual
JBE	Salta si es inferior o igual	JNA	Salta si no es superior

JE	Salta si es igual	JZ	Salta si el resultado es cero
JNE	Salta si no es igual	JNZ	Salta si el resultado no es cero
JG	Salta si es mayor que	JNLE	Salta si no es menor o igual que
JGE	Salta si es mayor o igual que	JNL	Salta si no es menor que
JL	Salta si es menor que	JNGE	Salta si no es mayor o igual que
JLE	Salta si es menor o igual que	JNG	Salta si no es mayor que

Instrucciones de iteración

Una iteración es un bloque de código que finaliza con un salto condicional, de tal forma que el código puede ser ejecutado repetidamente hasta que la condición de terminación sea alcanzada.

El microprocesador 8086 proporciona varias instrucciones especiales para realizar iteraciones.

La instrucción loop

Supóngase que se desea imprimir una cadena de una longitud de 10 caracteres. Esto podría ser realizado con el siguiente código:

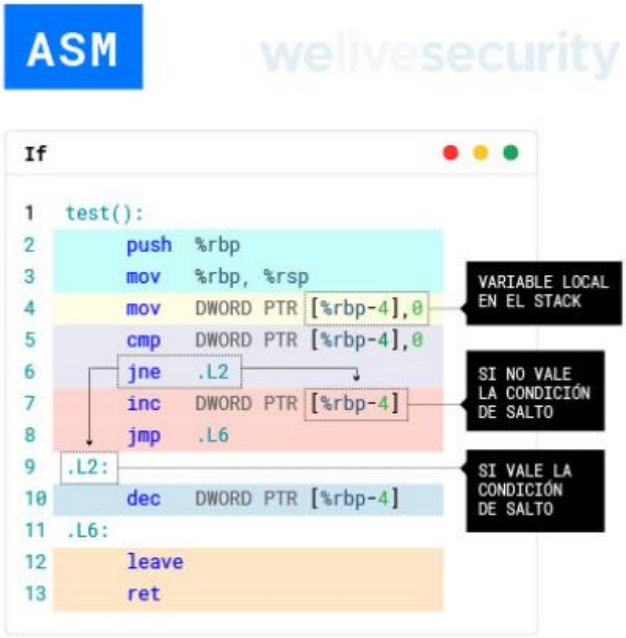
Sin embargo, existe una forma más fácil de hacerlo mediante la instrucción LOOP. La instrucción LOOP decrementa el contenido del registro CX y finaliza la iteración si CX ha alcanzado el valor de cero. Si CX es diferente de cero, entonces se ejecutará la instrucción indicada por la etiqueta destino, operando de la instrucción LOOP.

La instrucción LOOPE hace lo mismo que LOOP, excepto que LOOPE finalizará el ciclo si CX llega a cero o si la bandera de cero es 1. Es importante recordar que la bandera de cero es 1, cuando el resultado de una operación aritmética es cero o si los dos operandos en la última comparación son iguales. La instrucción LOOPNE, por su parte, finalizará el ciclo si CX es cero o la bandera de cero es 0 [6].

Estructura del If, while, for, do, repeat en lenguaje ensamblador.

If

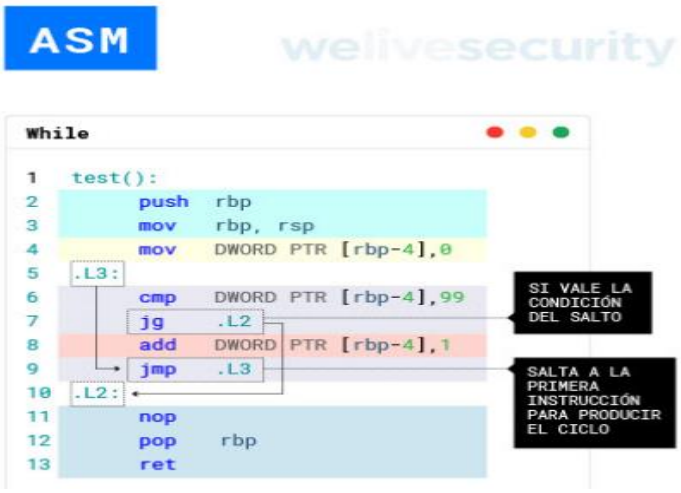
Los condicionales están fuertemente basados en instrucciones de comparación (CMP) y de salto (Jxx). La idea detrás de esto será aplicar una comparación entre dos elementos y, según el resultado de dicha comparación, saltar a otro punto del código o seguir a la siguiente instrucción. Luego, al depender del resultado de una comparación, las instrucciones de salto utilizadas serán las de salto condicional y la instrucción utilizada dependerá del tipo de comparación presente en la condición del if.



6.- Estructura IF.

While/for

Los ciclos son similares a los ifs, ya que contienen una condición que será la que determine durante cuánto tiempo se estará dentro del ciclo. Mientras esta condición no se cumpla (negación de la guarda), no se tomará el salto condicional y simplemente se continuará la ejecución en la siguiente instrucción. Si la condición se cumple, se activará el salto condicional que apuntará a una instrucción por fuera del ciclo. Lo único que falta será la instrucción que permita que la ejecución efectivamente sea un ciclo. Esto se logrará colocando al final del código del cuerpo del ciclo una instrucción de salto no condicional JMP que apunte a la primera instrucción del ciclo, es decir, a la condición inicial [7].



7.- Estructura While.

Do

Algoritmo de cálculo del término n-ésimo de la serie de Fibonacci:

f(n) = f(n–1) + f(n–2), n>1 entero, donde f(0) = 1 y f(1) = 1

➲ MIPS:

```
# $s0 = n
# $s1 = f
# $s2 = fant
# $s3 = i
# $s4 = faux
        li      $s0,5
        li      $s2,1
        li      $s1,1
        li      $s3,2
WHILE:  bgt     $s3,$s0,END
        move    $s4,$s1
        add     $s1,$s1,$s2
        move    $s2,$s4
        addi    $s3,$s3,1
        j       WHILE
END:
```

8.- Estructura Do.

Repeat

La sentencia repeat es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while que podía no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while que está en la parte superior.

Finaliza la ejecución del bloque repetitivo cuando la condición retorna verdadero, es decir en forma inversa al while (repeat/until se lee repetir hasta que sea verdadero) [8].

➲ MIPS:

```
# $s0 = a
# $s1 = b
# $s2 = mcd
# $s3 = resto
        li      $s0,81
        li      $s1,18
REPEAT:
        move    $s2,$s1
        div     $s0,$s1
        mfhi    $s3
        move    $s0,$s1
        move    $s1,$s3
UNTIL:  bnez    $s3,REPEAT
```

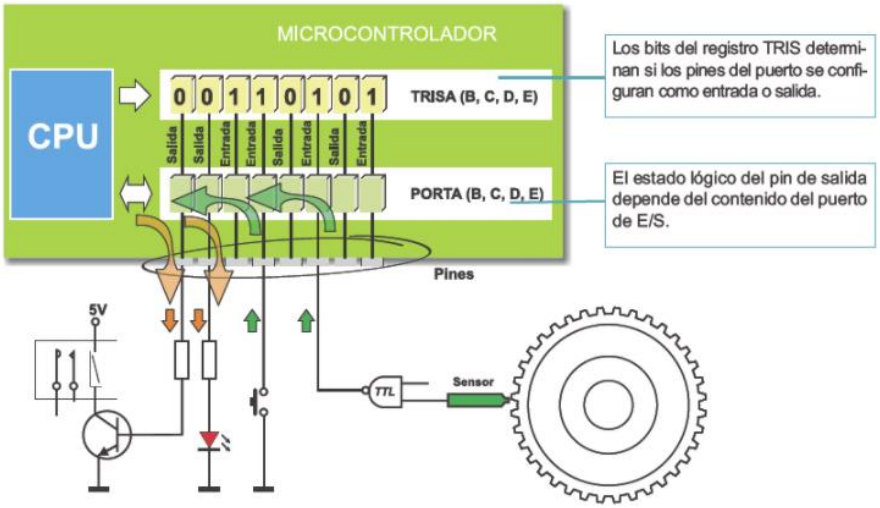
9.- Estructura repeat.

Programación de puertos digitales.

Con el propósito de sincronizar el funcionamiento de los puertos de E/S con la organización interna del microcontrolador de 8 bits, ellos se agrupan, de manera similar a los registros, en cinco puertos denotados con A, B, C, D y E. Todos ellos tienen las siguientes características en común:

- Por las razones prácticas, muchos pines de E/S son multifuncionales. Si un pin re aliza una de estas funciones, puede ser utilizado como pin de E/S de propósito general.
- Cada puerto tiene su propio registro de control de flujo, o sea el registro TRIS correspondiente: TRISA, TRISB, TRISC etc. lo que determina el comportamiento de bits del puerto, pero no determina su contenido.

Al poner a cero un bit del registro TRIS (pin=0), el pin correspondiente del puerto se configurará como una salida. De manera similar, al poner a uno un bit del registro TRIS (bit=1), el pin correspondiente del puerto se configurará como una entrada. Esta regla es fácil de recordar: 0 = Entrada 1 = Salida [9].



10.- Puertos Entrada/Salida.

Programación en lenguaje de Alto Nivel para Sistemas Embebidos.

Cuando hablamos de un lenguaje de alto nivel nos referimos al tipo de lenguaje de programación que no expresa los algoritmos teniendo en cuenta la capacidad que tienen las máquinas para ejecutar órdenes, sino al que se utiliza teniendo en cuenta las capacidades cognitivas de los seres humanos. Existen desde la década de los 50 y nacieron con el objetivo de ir más allá respecto a las limitaciones de los lenguajes de bajo nivel, permitiendo a los usuarios resolver problemas de una forma sencilla y rápida. Desde entonces han aparecido distintos lenguajes de alto nivel.

Estructura del programa con If, while, for, do, repeat en lenguaje de alto nivel

If

if (condición) { bloque de sentencias }

Si la condición mencionada entre paréntesis se cumple, entonces se ejecutará el código que se encuentre en el bloque de sentencias, de lo contrario, la ignorará y continuará el programa ejecutándose.

While

while (condiciones) { bloque de sentencias que ejecutar }

Este bucle lo que evalúa es la condición mencionada entre los paréntesis, si se cumple, ejecuta el bloque de sentencias escritas dentro de los corchetes correspondientes, de lo contrario la ignorará.

Do while

do { bloque de sentencias que ejecutar } while (condiciones);

El bucle se ejecutará mínimo 1 vez, debido a que primero entra en el bloque de sentencias a ejecutar y, una vez ejecutado, después determina si la condición o condiciones se cumplen. Si lo hacen se vuelve a ejecutar nuevamente el código, si no simplemente seguirá el código su curso normal.

For

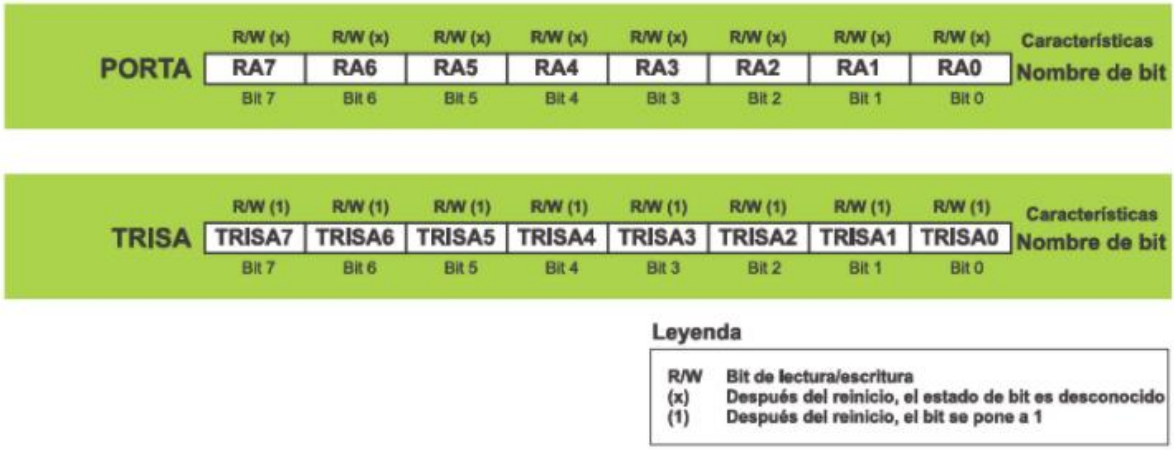
for (valores iniciales; condiciones; actualización) { bloque de sentencias que ejecutar }

For es una estructura iterativa que comienza desde un valor inicial, se ejecuta el bloque de sentencias, y decide si continuar o no basándose en la condición que se le haya impuesto. Si se cumplió va a realizar una actualización del valor que contiene para seguir ejecutándose [10].

Manejo de Puertos de entrada/salida digital.

Puerto PORTA y registro TRISA

El puerto PORTA es un puerto bidireccional, de 8 bits de anchura. Los bits de los registros TRISA y ANSEL controlan los pines del PORTA. Todos los pines del PORTA se comportan como entradas/salidas digitales. Cinco de ellos pueden ser entradas analógicas (denotadas por AN):



11.- Puerto PORTA y registro TRISA.

RA0 = AN0 (determinado por el bit ANS0 del registro ANSEL) RA1 = AN1 (determinado por el bit ANS1 del registro ANSEL) RA2 = AN2 (determinado por el bit ANS2 del registro ANSEL) RA3 = AN3 (determinado por el bit ANS3 del registro ANSEL) RA5 = AN4 (determinado por el bit ANS4 del registro ANSEL) Similar a que los bits del registro TRISA determinan cuáles pines serán configurados como entradas y cuáles serán configurados como salidas, los bits apropiados del registro ANSEL determinan si los pines serán configurados como entradas analógicas o entradas/salidas digitales. Cada bit de este puerto tiene una función adicional relacionada a algunas unidades periféricas integradas, que vamos a describir en los siguientes capítulos. Este capítulo cubre sólo la función adicional del pin RA0, puesto que está relacionado al puerto PORTA y a la unidad ULPWU.

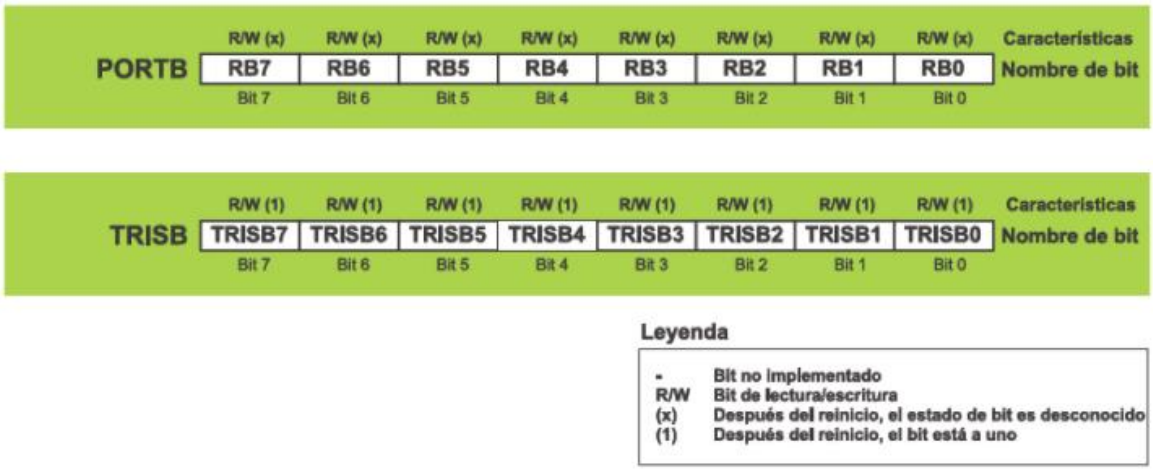
Puerto PORTB y registro TRISB

El puerto PORTB es un puerto bidireccional, de 8 bits de anchura. Los bits del registro TRISB determinan la función de sus pines.

Similar al puerto PORTA, un uno lógico (1) en el registro TRISB configura el pin apropiado en el puerto PORTB y al revés. Los seis pines de este puerto se pueden comportar como las entradas analógicas (AN). Los bits del registro ANSELH determinan si estos pines serán configurados como entradas analógicas o entradas/salidas digitales: RB0 = AN12 (determinado por el bit ANS12 del registro ANSELH) RB1 = AN10 (determinado por el bit ANS10 del registro ANSELH) RB2 = AN8 (determinado por el bit ANS8 del registro ANSELH) RB3 = AN9 (determinado por el bit ANS9 del registro ANSELH) RB4 = AN11 (determinado por el bit ANS11del registro ANSELH) RB4 = AN11 (determinado por el bit ANS11del registro ANSELH) Cada bit de este puerto tiene una función adicional relacionada a algunas unidades periféricas integradas, que vamos a

describir en los siguientes capítulos. Este puerto dispone de varias características por las que se distingue de otros puertos y por las que sus pines se utilizan con frecuencia.

Todos los pines del puerto PORTB tienen las resistencias pull-up integradas, que los hacen perfectos para que se conecten con los botones de presión (con el teclado), interruptores y optoacopladores. Con el propósito de conectar las resistencias a los puertos del microcontrolador, el bit apropiado del registro WPUB debe estar a uno.



12.- Puerto PORTB y registro TRISB

Al tener un alto nivel de resistencia (varias decenas de kiloohmios), estas resistencias "virtuales" no afectan a los pines configurados como salidas, sino que sirven de un complemento útil a las entradas. Estas resistencias están conectados a las entradas de los circuitos lógicos CMOS. De lo contrario, se comportarían como si fueran flotantes gracias a su alta resistencia de entrada.

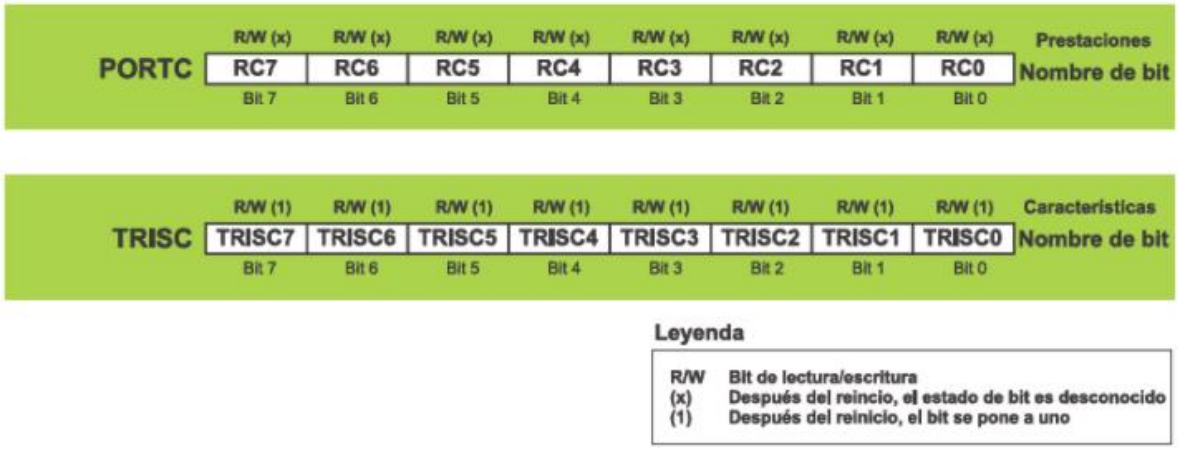
Además de los bits del registro WPUB, hay otro bit que afecta a la instalación de las resistencias pull-up. Es el bit RBPU del registro OPTION_REG.

Al estar habilitado, cada bit del puerto PORTB configurado como una entrada puede causar una interrupción al cambiar su estado lógico. Con el propósito de habilitar que los terminales causen una interrupción, el bit apropiado del registro IOCB debe estar a uno.

Gracias a estas características, los pines del puerto PORTB se utilizan con frecuencia para comprobar los botones de presión en el teclado ya que detectan cada apretón de botón infaliblemente. Por eso, no es necesario examinar todas las entradas una y otra vez.

Puerto PORTC y registro TRISC

El puerto PORTC es un puerto bidireccional, de 8 bits de anchura. Los bits del registro TRISC determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISC configura el pin apropiado del puerto PORTC como entrada.

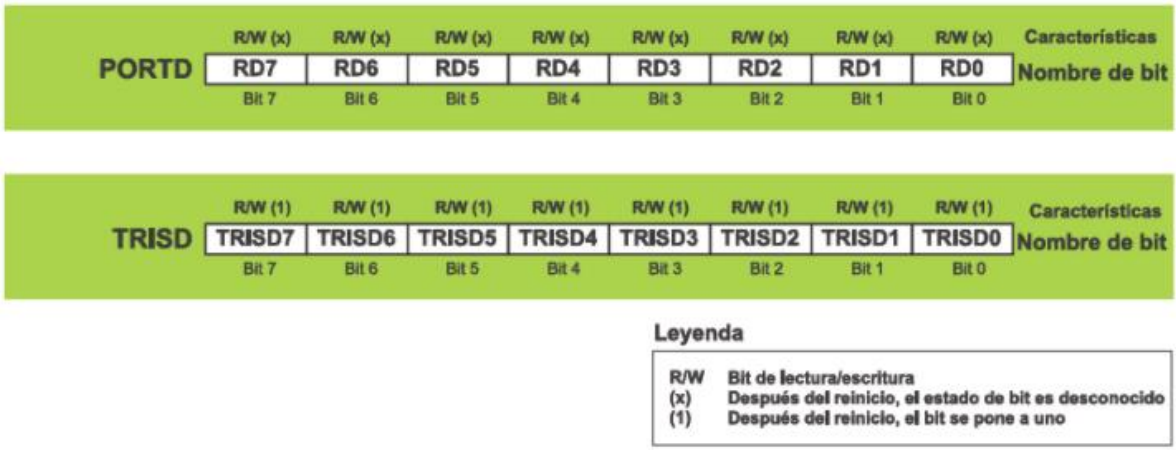


13.- Puerto PORTC y registro TRISC.

Todas las funciones adicionales del puerto PORTC se describen en los siguientes capítulos.

Puerto PORTD y registro TRISD

El puerto PORTD es un puerto bidireccional de 8 bits de anchura. Los bits del registro TRISD determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISD configura el pin apropiado del puerto PORTD como entrada.



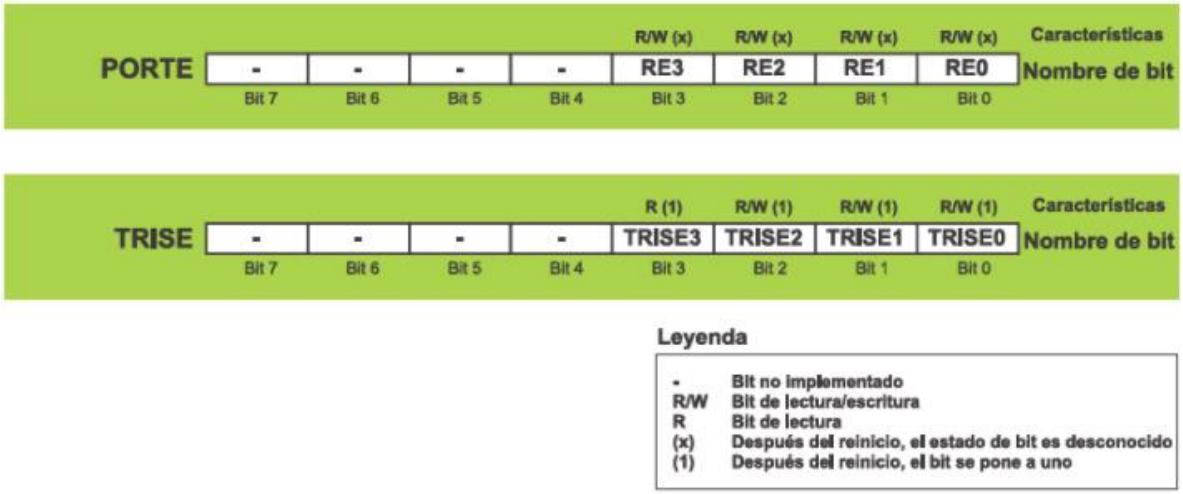
14.- Puerto PORTD y registro TRISD

Puerto PORTE y registro TRISE

El puerto PORTE es un puerto bidireccional, de 4 bits de anchura. Los bits del registro TRISE determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISE configura el pin apropiado del puerto PORTE como entrada.

La excepción es el pin RE3, que siempre está configurado como entrada. Similar a los puertos PORTA y PORTB, en este caso los tres pines se pueden configurar como entradas analógicas. Los bits del registro ANSEL determinan si estos pines serán configurados como entradas analógicas (AN) o entradas/salidas digitales: RE0 = AN5 (determinado por el bit

ANS5 del registro ANSEL); RE1 = AN6 (determinado por el bit ANS6 del registro ANSEL); y RE2 = AN7 (determinado por el bit ANS7 del registro ANSEL) [9].



15.- Puerto PORTE y registro TRISE.

Manejo del Convertidor analógico/digital.

Los convertidores A/D son dispositivos electrónicos que establecen una relación biunívoca entre el valor de la señal en su entrada y la palabra digital obtenida en su salida. La relación se establece en la mayoría de los casos, con la ayuda de una tensión de referencia.

La resolución (R) que tiene cada bit procedente de la conversión, tiene un valor que es función de la tensión de referencia (Vref) de acuerdo con la formula siguiente:

$$R = \frac{Vref+ - Vref -}{1024} = \frac{Vref}{1024}$$

La conversión analógica a digital tiene su fundamento teórico en el teorema de muestreo y en los conceptos de cuantificación y codificación.

Una primera clasificación de los convertidores A/D, es la siguiente:

- Conversores de transformación directa.
- Conversores con transformación (D/A) intermedia, auxiliar.

Un convertidor A/D toma un voltaje de entrada analógico y después de cierto tiempo produce un código de salida digital que representa la entrada analógica. El proceso de conversión A/D es generalmente más complejo y largo que el proceso D/A, y se han creado y utilizado muchos métodos.

Varios tipos importantes de ADC utilizan un convertidor D/A como parte de sus circuitos. En la figura siguiente se muestra un diagrama de bloque general para esta clase de ADC. La oportunidad para realizar la operación es ofrecida por la señal del cronómetro de entrada. La unidad de control contiene los circuitos lógicos para generar la secuen-cia de operaciones adecuada en respuesta al comando "START", el cual inicia el pro-ceso de conversión. El comparador tiene dos entradas analógicas y una salida digital que intercambia estados, según qué entrada analógica sea mayor.

Diagrama en bloques de un ADC

- La operación básica de los convertidores A/D de este tipo consta de los siguientes pasos:
- El comando START pasa a alto dando inicio a la operación
- A una razón determinada por el cronómetro, la unidad de control modifica continuamente el número binario que está almacenado en el registro.
- El número binario del registro es convertido en un voltaje analógico, Va', por el convertido D/A.
- El comparador compara Va' con la entrada analógica Va. En tanto que Va' < Va, la salida del comparador permanece en alto. Cuando Va' excede a Va por lo menos en una cantidad Vt (voltaje umbral), la salida del comparador pasa a bajo y suspende el proceso de modificación del número del registro. En este punto, Va' es un valor muy aproximado de Va y el número digital del registro, que es el equivalente digital de Va' es asimismo el equivalente digital de Va, en los límites de la resolución y exactitud del sistema.

Las diversas variaciones de este esquema de conversión D/A difieren principalmente en la forma en que la sección de control modifica continuamente los números contenidos en el registro. De lo contrario, la idea básica es la misma, con el registro que contiene la salida digital requerida cuando se completa el proceso de conversión [11].

Uso de herramientas como el temporizador, generador de señales, medidor de intervalos, decodificador, entre otros.

Temporizador

Un temporizador es un dispositivo que se utiliza para controlar la conexión o desconexión de un circuito, todo dependiendo del tipo que sea ya que pueden ser eléctricos, neumáticos, hidráulicos, mecánicos, etc.

En cuanto a su funcionamiento se asemeja mucho al de un relevador ya que estos al recibir un pulso inmediatamente cambian la posición de sus contactos y en cuanto a los temporizadores necesita agotarse el tiempo programado para intercambiar sus contactos.

Los temporizadores funcionan bajo el mismo principio y este es: contabilizar un tiempo y cambiar de posición sus contactos.

Generador de señales

Los generadores de funciones son equipos capaces de generar funciones o señales. Si lo que se requiere es hacer una medición o una prueba de algún dispositivo electrónico, es lógico pensar en un osciloscopio para poder adquirir las señales y verificar el comportamiento del dispositivo. Sin embargo, ¿qué sucede cuando el dispositivo que queremos probar o caracterizar no es capaz de producir señales por sí mismo? Un ejemplo de estos equipos puede ser un amplificador, un multiplexor, un receptor de comunicaciones, un convertidor analógico-digital (ADC), un convertidor digital-analógico (DAC), etc.

Las aplicaciones de un generador de funciones podemos dividirla, de manera general, en tres:

- Crear señales: señales creadas desde cero para simular, estimular y probar distintos circuitos y dispositivos.
- Replicar señales: ya sea una anomalía, un error o una señal adquirida por un osciloscopio, podemos recrearla utilizando un generador de funciones en nuestro laboratorio para variar sus parámetros y analizarla en un ambiente controlado.
- Generar señales: señales ideales o funciones ya conocidas para utilizarlas como referencia o como señal de entrada para pruebas.

Medidor de intervalos

Un temporizador programable de intervalos (PIT) es un contador que dispara una interrupción cuando alcanza la cuenta programada. Los contadores PIT pueden ser one-shot o periódicos. Los temporizadores one-shot interrumpen solo una vez, y después paran de contar. Los temporizadores periódicos interrumpen cada vez que alcanzan un valor específico. Esta interrupción es recibida en intervalos regulares desde el temporizador programable de intervalos. Esta interrupción es usada para invocar actividades del kernel que deben ser realizadas en una base regular. Los contadores usualmente son programados con intervalos de incremento fijos que determinan cuanto tiempo el contador cuenta antes de que dispare una interrupción. Por lo tanto, los incrementos del intervalo determinan la resolución para la cual el contador se puede programar para generar su interrupción one-shot o periódica.

Decodificador

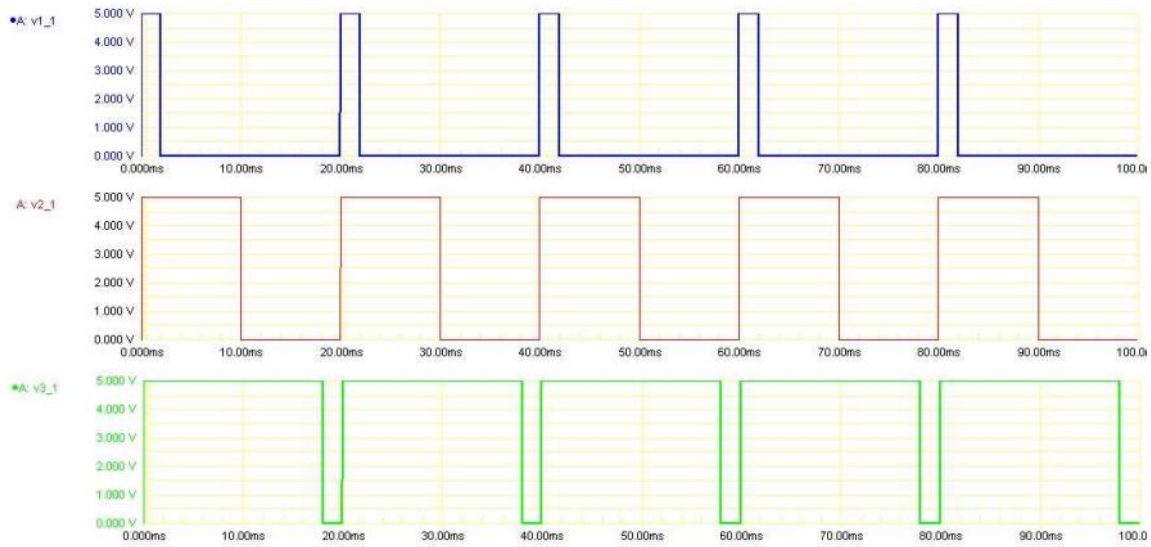
Tienen como función detectar la presencia de una determinada combinación de bits en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. Un

decodificador posee N líneas de entrada para gestionar N bits y en una de las 2 N líneas de salida indica la presencia de una o más combinaciones de n bits.

Para cualquier código dado en las entradas solo se activa una de las N posibles salidas. Un decodificador es un circuito combinacional, que, en su forma más general, posee n entradas y 2^n salidas digitales, donde solamente una de las salidas puede estar activa permaneciendo el resto en reposo [12].

Modulación por Ancho de Pulsos (PWM) para sistemas embebidos.

El control por ancho de pulso (Pulse Wide Modulate) es la forma de control que puede ser empleada también cómo protocolo de comunicación. La principal ventaja es su simplicidad respaldada por la generalidad de sus parámetros. PWM es una señal periódica en la que los datos son representados por el ancho del pulso. Esto quiere decir que se aplica un pulso cada intervalo de tiempo. La duración del pulso ha de ser inferior o igual al intervalo tiempo.



16.- PWM

Las variables a tener en cuenta son el ancho de pulso y la frecuencia de envío de pulso. El ciclo de trabajo se calcula dividiendo el ancho de pulso por la frecuencia.

Las aplicaciones son en el campo del control de motores y parte de un convertidor analógico digital. En caso del control de motores, el ancho de pulso se interpreta cómo velocidad de giro para los motores de corriente continua y cómo posición para los servos. Para las conversiones, se hace la traducción del voltaje a pulsos a través de un filtro.

La flexibilidad de la solución PWM permite a cualquier diseñador de sistemas emplearlo con las más variadas configuraciones: ancho de pulso, forma de onda, frecuencia, todo

ello constituyendo una sencilla y eficaz forma de comunicación exenta de encabezados o redundancias. En cuanto al sistema físico idóneo para el PWM, no hay especificaciones sobre el sistema en el que se puede implementar, dejando al diseñador decidir y especificar las características del bus de transmisión en función de la frecuencia y corriente.

Actualmente el control por ancho de pulso está empleado desde el control de la velocidad de rotación de los ventiladores existentes en los ordenadores, pasando por el posicionamiento gracias a los servomotores y hasta los variadores de intensidad para el control de velocidad de motores de corriente alterna [13].

Simuladores de Sistemas Embebidos.

GPSIM

El gpsim es otra herramienta que se encuentra en el paquete de gputils la cual nos permitirá valorar el comportamiento de nuestro algoritmo antes de llevarlo a la programación en el microcontrolador PIC [14].

LabVIEW

El entorno de LabVIEW constituye una herramienta sumamente eficaz que nos ha permitido dar un enfoque diferente sobre el aprendizaje de las instalaciones térmicas, posibilitándonos su estudio dinámico en el tiempo, sin necesidad de disponer de equipos físicos, que no siempre están disponibles [15].

SPIM

SPIM es un simulador autónomo para programas en lenguaje ensamblador escritos para los procesadores R2000/R3000, los cuales son procesadores de 32 bits de la corporación MIPS. SPIM lee e inmediatamente ejecuta el código en lenguaje ensamblador, proporciona un depurador y un juego simple de servicios del sistema operativo [16].

Conclusión

A través del informe pude conocer las diferencias, características, ventajas y desventajas que existen entre el lenguaje ensamblador, el cual generalmente es utilizado para tareas específicas, lo cual permite economizar la memoria que posee el microcontrolador.

Así mismo tenemos el lenguaje de alto nivel, el cual resulta un tanto mas sencillo de manejar, sin embargo, este consume mayor memoria, aun así, es una excelente alternativa al momento de necesitar programar programas mas complejos, ya que, en lenguaje de alto nivel podríamos ocupar menor número de líneas de código en comparación con el lenguaje ensamblador.

Es importante conocer ambas formas de programación ya que dependiendo de los problemas que puedan surgir al momento de querer realizar un proyecto, poder elegir eficazmente el lenguaje en el cual programar.

Bibliografía

1. Sistemas Embebidos: Innovando hacia los Sistemas Inteligentes. (2021). Consultado el 26 de Junio del 2021, de http://www.semanticwebbuilder.org.mx/es_mx/swb/Sistemas_Embebidos_Innovando_hacia_los_Sistemas_Inteligentes_
2. El lenguaje ensamblador. (2021). Consultado el 26 de Junio del 2021, de https://www.tecnologia-informatica.com/el-lenguaje-ensamblador/
3. Sistemas integrados: modos de direccionamiento. (2021). Consultado el 26 de Junio del 2021, de https://es.it-brain.online/tutorial/embedded_systems/es_addressing_modes/
4. Arquitectura interna y externa del microcontrolador.. (2021). [Libro Electrónico] (pp. 62-74). Consultado el 26 de Junio del 2021, de http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes%20de%20materias/ETD1022_Microcontroladores/1_Arquitectura.pdf
5. INSTRUCCIONES DE TRANSFERENCIA BÁSICAS, ARITMÉTICAS Y LÓGICAS. (2021). [Libro Electrónico] (pp. 4-20). Consultado el 26 de Junio del 2021, de https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r89355.PDF
6. Sánchez González, S. (2021). Capítulo II. Conceptos básicos de Lenguaje Ensamblador 8086. [Libro Electrónico] (pp. 34-41). Consultado el 26 de Junio del 2021, de https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r87220.PDF
7. Estructuras de control de flujo en código Assembly. (2021). Consultado el 26 de Junio del 2021, de https://www.welivesecurity.com/la-es/2020/05/01/estructuras-de-control-de-flujo-en-codigo-assembly/
8. Serrano Sánchez de León, Á., & Rincón Córcoles, L. (2021). Programación en ensamblador [Libro Electrónico]. Consultado el 26 de Junio del 2021, de https://www.cartagena99.com/recursos/electronica/apuntes/tema12_ensamblador_MIPS.pdf
9. puertos-de-entradasalida. (2021). Consultado el 26 de Junio del 2021, de https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/puertos-de-entradasalida
10. Orenga, M., & Manonellas, G. (2021). Programación en ensamblador (x86- 64) [Libro Electrónico]. Consultado el 26 de Junio del 2021, de

https://www.exabyteinformatica.com/uoc/Informatica/Estructura_de_computadores/Estructura_de_computadores_(Modulo_6).pdf
11. Martorell, J. (2003). Práctica 4 – Manejo del convertidor Analógico / Digital. Consultado el 26 de Junio del 2021, de https://jesicagreer.wordpress.com/practica-4-manejo-del-convertidor-analogico-digital/
12. Gastellou, E. (2020). Generadores de Funciones, Todo lo que necesitas saber sobre ellos. Consultado el 26 de Junio del 2021, de https://acmax.mx/que-es-un-generador-defunciones
13. LADARESCU, I. (2021). Sistema de Control Embebido conectable a PC: Diseño y Aplicaciones Prácticas [Libro Electrónico] (pp. 22-24). Consultado el 26 de Junio del 2021, de https://riunet.upv.es/bitstream/handle/10251/9118/DISCA-84.pdf?sequence=1
14. Simulador - Programación de Sistemas Embebidos. (2021). Consultado el 26 de Junio del 2021, de https://sites.google.com/site/sistemasembebidosalpha/home/microcontroladores-pic/simulador
15. Aplicación de LabVIEW en el desarrollo de simuladores para formación profesional. (2021). Consultado el 26 de Junio del 2021, de https://www.ni.com/es-mx/innovations/case-studies/19/labview-application-in-the-development-of-simulators-for-professional-training.html
16. Arquitectura de Computadoras. (2021). Consultado el 26 de Junio del 2021, de https://www.utm.mx/~merg/AC/2009/3.11-simulador.html