Microcontroladores

Conceptos Fundamentales

Teresa Orrañanos Guerrero

Agosto **2020**

Ciclo **1208**

Contenido

Sistemas de numeración y operaciones

Los microcontroladores están formados por circuitos lógicos, los cuales pueden tomar dos valores posibles: 0 y 1, conocidos como bits.

La forma más común de representar un número es mediante una cadena de bits que se llama número binario.

Representación de números binarios con signo

En los números binarios se requiere representar tanto números positivos como negativos. Para eso existen tres sistemas de representación diferentes:

- Representación en signo y magnitud
- Representación en complemento a uno
- Representación en complemento a dos

Representación en signo y magnitud

Cuando un número binario cualquiera es negativo únicamente es necesario cambiar el valor del bit más significativo MSB (el que se encuentra más hacia la izquierda) de 0 a 1 para representar los números negativos.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Signo y magnitud				

Cabe hacer notar que cuando se emplea la representación en signo y magnitud, el bit más significativo corresponde sólo al signo y no forma parte del número en sí, es por ello que, cuando se utiliza esta representación se pueden representar valores desde $-2^{n-1}+1$ hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

Representación en complemento a uno

Cuando un número es negativo y se desea representar utilizando complemento a uno, únicamente es necesario obtener el complemento de cada uno de los bits del valor positivo correspondiente.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Complemento uno				

Cabe hacer notar que cuando se emplea la representación en complemento a uno se pueden representar valores desde $-2^{n-1}+1$ hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

Representación en complemento a dos

Cuando un número es negativo y se desea representar utilizando complemento a dos, es necesario calcular el complemento a 1 del número binario sin signo y posteriormente sumarle 1.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Complemento dos				

Si se conoce un número negativo en complemento a dos y se desea saber cuál es su valor, éste puede determinarse sacándole su complemento y sumándole un 1.

Cabe hacer notar que cuando se emplea la representación en complemento a dos se pueden representar valores desde -2^{n-1} hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

El complemento a dos es el modo más eficiente para efectuar operaciones de suma y resta, entre otras cosas debido a que únicamente tiene un cero.

EJERCICIOS

Complete la siguiente tabla

b3 b2 b1 b0	Signo y magnitud	Complemento a uno	Complemento a dos
0111			
0110			
0101			
0100			
0011			
0010			
0001			
0000			
1000			
1001			
1010			
1011			
1100			
1101			
1110			
1111			

Complete las siguientes tablas con los valores correspondientes, **utilizando siempre 8 bits**

Número decimal	-115	-93	67	-200
Binario sin signo				
Signo y magnitud				
Complemento uno				
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud	10010011	01111111	11111011	11001101
Complemento uno				
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud				
Complemento uno	10010011	01111111	11111011	11001101
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud				
Complemento uno				
Complemento dos	**10010011**	**01111111**	**11111011**	**11001101**

Operaciones Aritméticas

Suma de números sin signo

Cuando se suman pares de bits, se empieza siempre por los bits menos significativos LSB (los que se encuentran más a la derecha) lo acarreos se propagan a los de mayor orden.

0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 0 y hay un acarreo de 1

EJERCICIOS

Sume los siguientes números sin signo:

1 0 1 0	1 0 0 0	0 0 0 1	1 0 1 1
+ 0 0 0 1	+ 0 1 1 0	+ 1 1 1 1	+ 0 0 1 1
1 0 1 0 1 1 1	1 0 1 0 0 0 1	0 1 0 1 0 0 1	1 0 0 1 1 0 0
+ 1 0 0 1 1 0 1	+ 0 1 0 0 1 0 0	+ 1 0 1 1 1 0 1	+ 0 0 1 1 0 1 1

Suma de números con signo

Se escriben ambos números en su representación a complemento a dos

El acarreo final de la operación se debe ignorar.

0 0 1 0 (+2)	0 1 0 0 (+4)	1 0 1 1 (-5)	1 1 0 1 (-3)
+ 0 0 1 1 (+3)	+ 1 0 1 0 (-6)	+ 1 1 1 0 (-2)	+ 0 1 1 1 (+7)

EJERCICIOS

Sume los siguientes números en complemento a dos y verifique el resultado

0 1 1 0 (6)	1 0 0 0 (-8)	0 0 0 1 (1)	1 0 1 1 (-5)
+ 1 1 1 1 (-1)	+ 0 1 1 1 (7)	+ 1 0 0 1 (-7)	+ 0 0 1 1 (3)

1 1 1 0 (-2)	0 0 0 1 (1)	0 0 1 1 (3)	1 0 1 1 (-5)
+ 0 1 1 1 (7)	+ 1 1 1 1 (-1)	+ 1 1 0 1 (-3)	+ 1 1 1 0 (-2)

Desbordamiento

Como ya se sabe, en complemento a dos con n bits se pueden representar valores en el rango -2^{n-1} hasta $2^{n-1}-1$. Cuando el resultado de una operación está fuera del rango representable, se dice que se ha producido un desbordamiento aritmético.

El desbordamiento solamente puede ocurrir cuando se suman dos números con el mismo signo (es decir cuando ambos números son positivos y su MSB es 0 o bien cuando ambos números son negativos y su MSB es 1)

Una manera sencilla para detectar el desbordamiento es examinar los bits más significativos. Si el bit más significativo (MSB) del resultado no es igual a los bits más significativos de los sumandos, entonces quiere decir que se produjo un desbordamiento y por lo tanto el resultado NO es correcto.

EJERCICIOS

Realice las siguientes operaciones, verifique sus resultados e indique en el recuadro correspondiente si hubo acarreo al final de la operación y si hubo desbordamiento.

1 1 1 0 (-2) Acarreo ☐
+ 1 0 0 0 (-8) Desbordamiento ☐

0 1 1 1 (7) Acarreo ☐
+ 1 1 0 1 (-3) Desbordamiento ☐

0 0 1 0 (2) Acarreo ☐
+ 0 1 0 0 (4) Desbordamiento ☐

0 1 1 1 (7) Acarreo ☐
+ 0 1 1 1 (7) Desbordamiento ☐

1 1 0 0 (-4) Acarreo ☐
+ 1 0 0 1 (-7) Desbordamiento ☐

0 0 1 1 (3) Acarreo ☐
+ 0 0 0 1 (1) Desbordamiento ☐

Operaciones de desplazamiento

LSR – Logical Shift Right

Corresponde a un desplazamiento lógico hacia la derecha, de forma que el bit menos significativo (LSB) pasa al bit de carry del microprocesador, mientras que todos los bits del registro se desplazan un lugar hacia la derecha y el bit más significativo (MSB) se llena con un 0.

Esta operación equivale a dividir un número sin signo entre 2.

LSL – Logical Shift Left

Corresponde a un desplazamiento lógico hacia la izquierda, de forma que el bit más significativo (MSB) pasa al bit de Carry del microprocesador, mientras que todos los bits del registro se desplazan un lugar hacia la izquierda y el bit menos significativo (LSB) se llena con un 0.

Esta operación equivale a multiplicar un número sin signo por 2.

ROR – Rotate Right trough Carry

Corresponde a un desplazamiento circular con acarreo a la derecha, de forma tal que todos los bits del registro se desplazan un hacia la derecha, y el bit menos significativo (LSB) pasa a ocupar el lugar del bit más significativo (MSB). Dicho bit también queda almacenado en el carry.

ROL – Rotate Left trough Carry

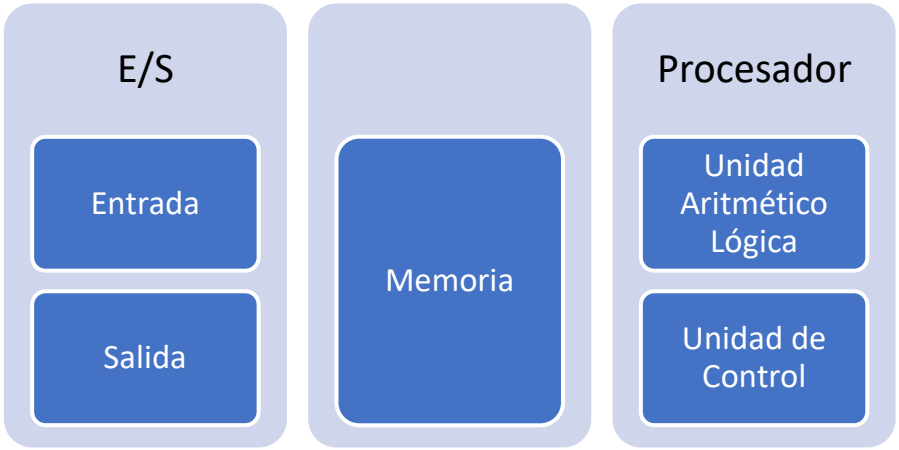
Corresponde a un desplazamiento circular con acarreo a la izquierda, de forma tal que todos los bits del registro se desplazan hacia la izquierda, y el bit más significativo (MSB) pasa a ocupar el lugar del bit menos significativo (LSB). Dicho bit también queda almacenado en el carry.

ASR – Arithmetic Shift Right

Corresponde a recorrer todos los bits de un registro un lugar hacia la derecha, manteniendo siempre constante el valor del bit más significativo (MSB). El bit menos significativo LSB es almacenado en el Carry del microprocesador. Esta operación realiza una división entre dos para números con signo, sin perder el signo en el resultado. La bandera de carry puede emplearse para redondear el resultado.

Estructura básica de los microprocesadores

Unidades funcionales



Unidad de Entrada / Salida (E/S)

La unidad de entrada acepta la información codificada que viene de operadores humanos, de dispositivos electromecánicos tales como teclados u otros dispositivos electrónicos.

La información recibida puede ser almacenada en la memoria para referencias posteriores o bien puede ser utilizada de inmediato por la circuitería de la unidad aritmética lógica (ALU) para realizar las operaciones deseadas. Los pasos del procesamiento son determinados por un programa que se encuentra almacenado en la memoria.

Para terminar, los resultados se envían al mundo exterior mediante la unidad de salida.

Todos los pasos descritos anteriormente son coordinados mediante la unidad de control.

Unidad de memoria

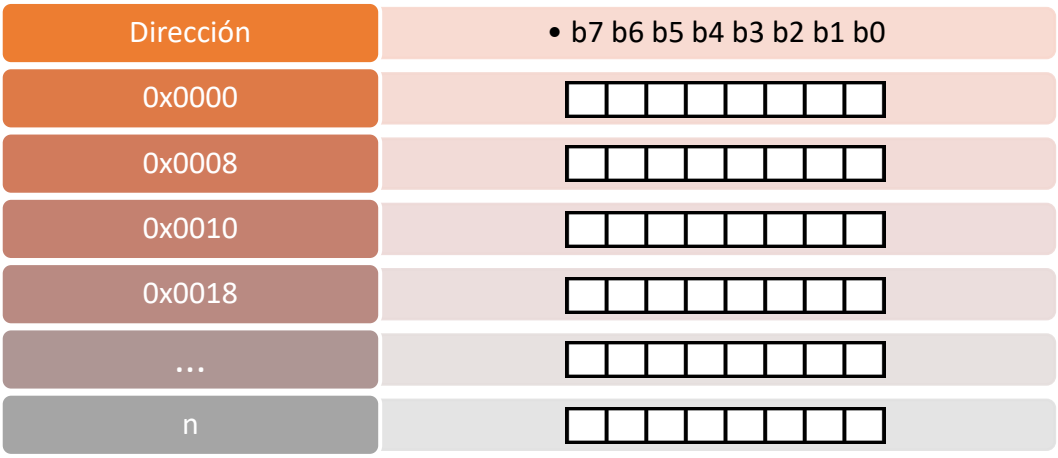
Su función es almacenar programas y datos. Existen dos clases de almacenamiento:

a) Almacenamiento primario (Por ej. Registros, caché, memoria principal)

Es la memoria rápida que funciona a velocidades electrónicas. Aquí se almacenan los programas mientras se ejecutan.

Una memoria tiene un gran número de celdas de almacenamiento, cada una capaz de almacenar un bit de información. Es muy raro que sea necesario leer una sola celda, casi siempre se procesan grupos fijos llamados palabras (Una palabra tiene “n” bits).

Para tener acceso rápido a una palabra, se asocia una dirección a cada una de ellas.



Al número de bits se le conoce como longitud de palabra (en el ejemplo la longitud es de 8 bits). Normalmente la longitud de palabra es entre 8 bits y 64 bits.

Algunos tipos de Almacenamiento Primario son:

- Los registros del procesador. Es el sistema más rápido de los distintos tipos de almacenamientos del microprocesador. Funcionan como "flip-flop" electrónicos.
- La memoria caché. Es un tipo especial de memoria interna usada para mejorar su eficiencia o rendimiento. Parte de la información de la memoria principal se duplica en la memoria caché. Comparada con los registros, la caché es ligeramente más lenta pero de mayor capacidad.
- La memoria principal. Contiene los programas en ejecución y los datos con que operan. Se puede transferir información muy rápidamente entre un registro del microprocesador y localizaciones del almacenamiento principal. En las computadoras modernas se usan memorias de acceso aleatorio (RAM) basadas en electrónica del estado sólido, que está directamente conectada a la CPU a través de buses de direcciones, datos y control.

b) Almacenamiento secundario (Por ej. Memorias USB, CD, DVD, Discos duros)

Es más barato, pero es mucho más lento que el almacenamiento primario, el tiempo necesario para acceder a datos en el almacenamiento primario se encuentra en el orden de los nanosegundos, mientras que en el almacenamiento secundario está en el orden de microsegundos. Sirve para almacenar gran cantidad de datos o programas a los que no se accede con tanta frecuencia. También se llama almacenamiento masivo.

Unidad Aritmético – Lógica (ALU)

Ahí se ejecutan la mayoría de las opciones del microprocesador. Cuando queremos sumar dos palabras localizadas en la memoria, la unidad de control las capta y el ALU (la unidad aritmético lógica) se encarga de realizar la suma. Después de eso el resultado puede almacenarse en la memoria o enviarse a la unidad de salida.

Las unidades de control son mucho más rápidas que otros dispositivos del microprocesador, es por eso que un solo microprocesador puede controlar al mismo tiempo muchos dispositivos externos como teclados, sensores, etc.

Unidad de control

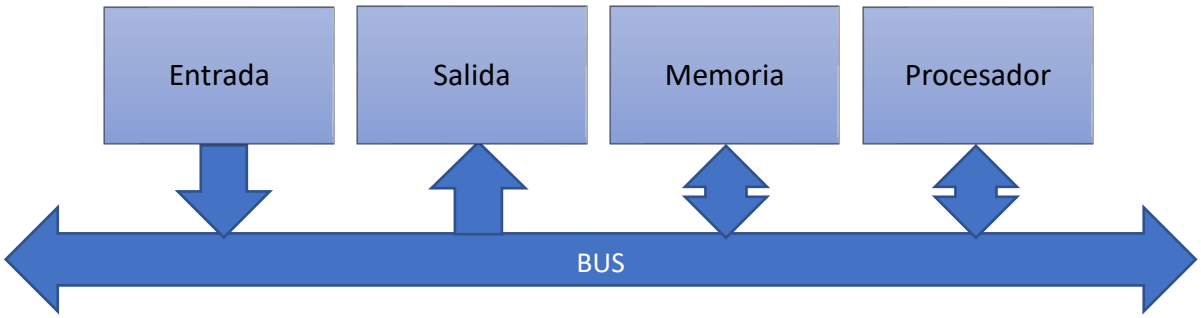
Se encarga de coordinar todas las acciones del procesamiento a través de señales de temporización. Estas señales determinan cuándo debe producirse determinada acción.

Estructura del Bus

Las unidades funcionales del procesador deben estar conectadas de algún modo organizado.

Para alcanzar velocidades de procesamiento razonables, el procesador debe estar organizado de forma tal que sus unidades puedan gestionar una palabra completa de datos (n bits) en un momento dado.

Cuando una palabra de datos de transfiere, todos los bits se transfieren en paralelo (es decir, todos los bits se transfieren al mismo tiempo) por lo tanto se emplea una línea para cada uno de los bits. Al grupo de líneas que sirve como conexión entre los dispositivos se le llama bus. Un bus debe tener líneas de datos, líneas de direcciones y líneas de control.



Algunos dispositivos conectados al bus son más rápidos (Discos, memorias) y hay algunos otros más lentos (teclados, impresoras, etc.)

Para que un dispositivo lento no ocupe el bus por largos periodos de tiempo, es típico incluir registros de buffer que se encargan de almacenar la información durante la transferencia.

Por ejemplo, si un procesador tiene que enviar información a la impresora (que es lenta), entonces la envía por el bus al buffer de la impresora (que es un registro electrónico y por tanto puede funcionar a velocidad rápida) y ahí se almacena. Una vez que el buffer de la impresora se ha cargado, la impresora puede comenzar a trabajar, sin que el procesador intervenga más y sin que el Bus se encuentre ocupado.

Memoria

Posiciones y direcciones de memoria

Cada celda de memoria almacena un bit de información
Una palabra es un grupo de n bits (normalmente entre 8 y 64)
Un byte corresponde exactamente a 8 bits o bien a dos valores hexadecimales
Una instrucción puede requerir más de una palabra para almacenarse
Para tener acceso a una palabra o byte se requiere que la memoria tenga direcciones para cada posición posible.

Direccionamiento de bytes en palabras

En forma similar a cómo ocurre en la escritura de texto, en donde las palabras pueden escribirse de izquierda a derecha o bien de derecha a izquierda, dependiendo del idioma, las máquinas pueden almacenar datos de una palabra en diferente orden.

Supongamos que se desea almacenar el siguiente valor (una palabra de cuatro bytes)

11001010111111101011101010111110

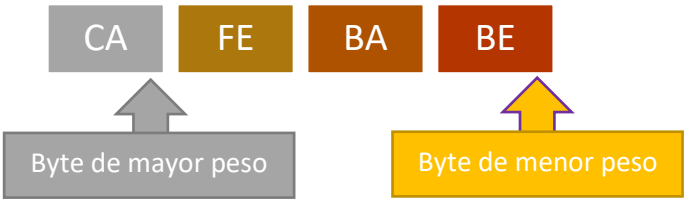
O lo que es lo mismo en forma hexadecimal:

CAFEBABE

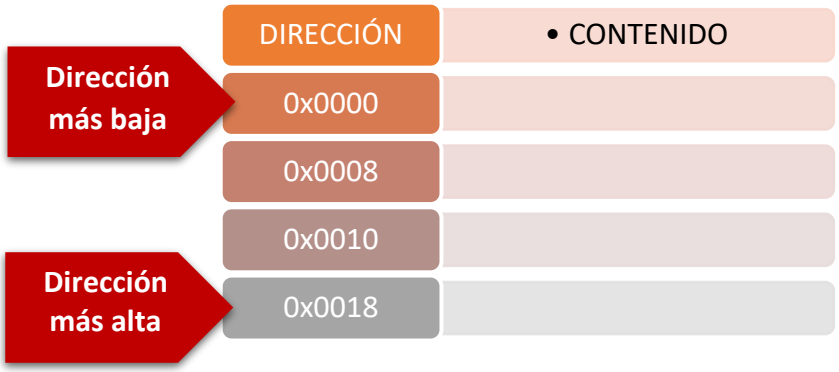
Separándolo por bytes se tiene

11001010 11111110 10111010 10111110

O bien



Y que se tiene el siguiente espacio de memoria para almacenar dichos bytes



Existen dos formas de almacenar los bytes en las direcciones de memoria:

a) Big Endian
b) Little Endian

A) BIG ENDIAN

Significa que el byte de mayor peso se almacena en la dirección más baja de memoria y el byte de menor peso en la dirección más alta.

DIRECCIÓN	• CONTENIDO
0x0000	• CA
0x0008	• FE
0x0010	• BA
0x0018	• BE

B) LITTLE ENDIAN

Significa que el byte de menor peso se almacena en la dirección más baja de memoria y el byte de mayor peso en la dirección más alta.

DIRECCIÓN	• CONTENIDO
0x0000	• BE
0x0008	• BA
0x0010	• FE
0x0018	• CA

Estructuras de datos

Los datos con los que opera un programa se pueden organizar de distintas maneras:

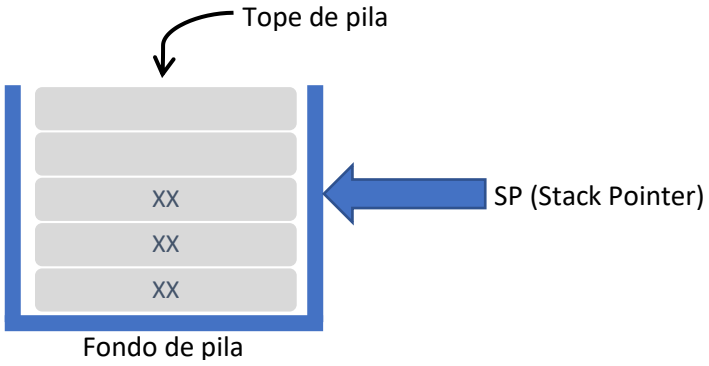
a) Pilas
b) Colas

A) PILAS

Es una lista de elementos en donde éstos sólo pueden añadirse o eliminarse por uno de los extremos de la lista. Se describen con la frase “El último que entra es el primero que sale” – LIFO (last in, first out).

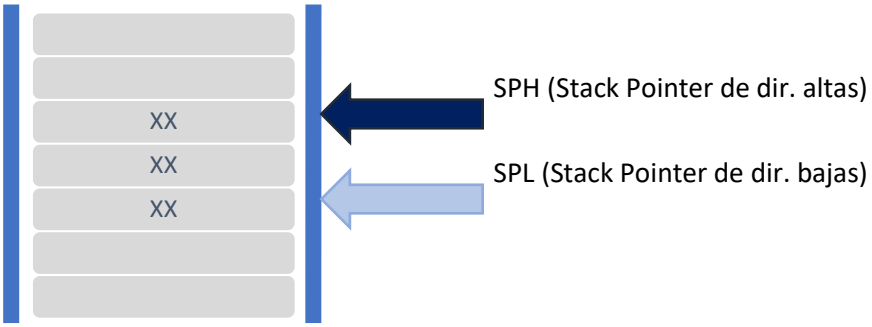
Se emplean los términos Push (insertar) y Pop (extraer) para describir la introducción de los nuevos elementos o la extracción (respectivamente)

Se emplea un puntero de pila, que es un registro del procesador que contiene la dirección del elemento que se encuentra en el tope de la pila en cada momento.



B) COLAS

Los datos se almacenan y se recuperan según el principio “El primero que entra es el primero que sale” – FIFO (First in, first out).



Diferencias en la implementación de una pila y una cola

PILA	COLA
Uno de los extremos está fijo mientras el otro crece o decrece.	Ambos extremos se mueven hacia los dos extremos.
Sólo requiere un puntero SP para apuntar al tope de la pila.	• Se requiere tener dos punteros para hacer referencia a los dos extremos de la cola.

Otra diferencia es que, sin algún tipo de control la cola se movería a través de la memoria. Una forma de limitar la cola a una región fija de memoria es utilizando un buffer circular.

Organización de Entrada / Salida

Usualmente a cada dispositivo de Entrada / Salida se le asigna un conjunto único de direcciones. Cuando el procesador introduce una dirección en la línea de direcciones del bus, el dispositivo que reconoce como propia dicha dirección responde en las líneas de control. Entonces el procesador solicita una operación de lectura o escritura y los datos son transferidos a través de la línea de datos.

Con una Entrada / Salida asignada a memoria, cualquier instrucción que puede acceder a la memoria, puede también transferir datos desde o hacia el dispositivo de Entrada / Salida.

Por ejemplo:

Uno de los puertos de salida del microprocesador tiene asignada la dirección de memoria 0x18 (es decir la dirección 18 en hexadecimal). Por lo tanto, si requiere enviar al puerto una señal que se encargue de prender los LEDs que se encuentran conectados en él, esto se puede hacer directamente mediante la instrucción

OUT 0x18, R16 ***;Que sacaría por el puerto el contenido del registro R16***

Conceptos básicos de funcionamiento

En un microcontrolador las instrucciones:

- Regulan la transferencia de información dentro del procesador, y entre el procesador y los dispositivos de entrada y salida.
- Especifican las operaciones aritméticas y lógicas a realizar.

En la figura de la siguiente página se muestra un diagrama más detallado de las conexiones entre la memoria y el procesador.

a) MAR (Memory Address Register)

Es el registro de dirección de memoria que contiene la dirección de la memoria a la que se desea acceder ya sea para lectura o escritura.

b) MDR (Memory Data Register)

Es el registro de datos de memoria que contiene los datos que se van a escribir en la memoria, o bien los datos que se leyeron de ella.

c) PC (Program Counter)

Es el contador del programa que contiene la dirección de la memoria de la siguiente instrucción a ser ejecutada. Mientras una instrucción se ejecuta, los contenidos del PC se actualizan para apuntar a la siguiente instrucción.

d) IR (Instruction Register)

Es el registro de instrucciones que contiene la instrucción que se está ejecutando en ese momento.

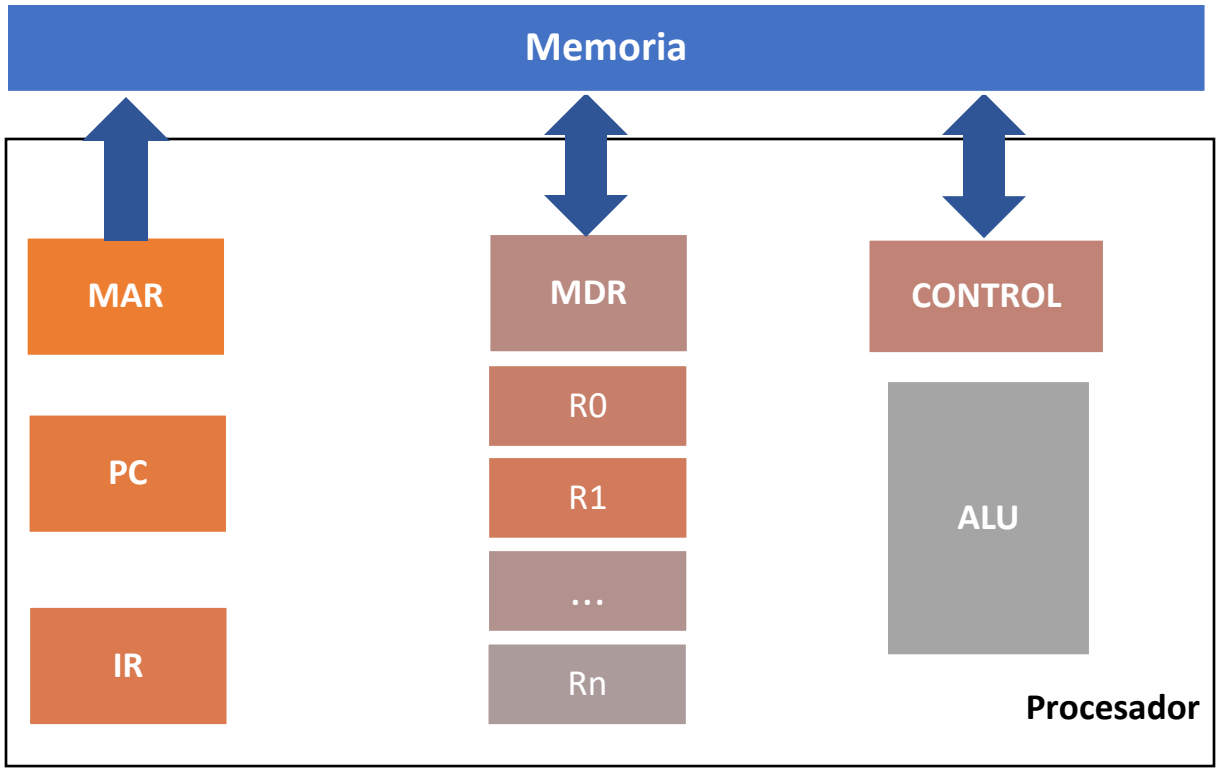
e) RO…Rn (Register)

Son los registros que el microprocesador puede utilizar para almacenar información.

Cuando una instrucción implica traer un dato de una dirección de memoria, primero la dirección que se quiere leer tiene que almacenarse en el registro MAR, entones el CONTROL se encarga de enviar los pulsos necesarios para iniciar un ciclo de lectura, la memoria contesta enviando el dato que se encuentra en la dirección almacenada en MAR al registro MDR y de ahí ese dato puede transferirse a los registros de procesador o al ALU.

Supongamos que la instrucción LDS tiene el siguiente formato:

LDS DESTINO (Registro), FUENTE (Memoria)



Supongamos también que en la memoria del procesador se tiene almacenado el programa que contiene la instrucción LDS, y que a la dirección de memoria 0x1001 se le asigna la etiqueta DirA.

Si queremos guardar el dato almacenado en DirA en el registro R10 del procesador, la instrucción completa se escribiría

LDS R10, DirA

Por tanto la memoria contendrá la siguiente información:

Para ejecutar esa línea del programa los pasos que se llevarán a cabo dentro del procesador serán:

Primero es necesario que el procesador lea la línea del programa donde se encuentra la instrucción, para eso el PC carga un 0000, pues esa es la primera dirección del programa, de esa forma PC "apunta" a 0x0000

PC ← 0000

Después, como lo que se quiere es leer la memoria, es necesario cargar a MAR la dirección de memoria que se quiere leer (esa dirección es el 0x0000 que se encuentra almacenado en el PC)

MAR ← PC

Ya que se tiene la dirección en el MAR, el CONTROL se encarga de mandar los pulsos necesarios a la memoria para que se realice una lectura

READ

Entonces hay que esperar a que la memoria complete la lectura y envíe el contenido de la dirección que se leyó al MDR

RMFC (Read Memory Function Completed)

En este momento ya se tiene en MDR el contenido de la localidad de memoria 0x0000 que fue la que se leyó, es decir, en MDR se encuentra LDS R10, DirA. Ahora, para que el procesador pueda realizar esta instrucción, es necesario que la instrucción se almacene en el registro IR.

IR ← MDR

Con esto se completa la fase de lectura de la instrucción, por lo tanto, automáticamente el CONTROL se encarga de incrementar el contenido del PC, para que se quede apuntando a la localidad de memoria en donde se encuentra la siguiente instrucción, es decir a la localidad de memoria 0x0001

INC PC

Una vez que la instrucción está en el IR, la unidad de CONTROL la reconoce y por lo tanto "entiende" lo que se desea hacer: traer el dato de la memoria de la dirección DirA y guardarlo en el registro R10. Entonces, para leer la memoria, carga en MAR la dirección DirA, es decir un 0x1001

MAR ← DirA (0x1001)

Después de eso el CONTROL se encarga de mandar los pulsos necesarios para realizar la lectura de la memoria.

READ

Y hay que esperar a que se complete la lectura

RMFC

En este momento ya se tiene en el MDR el contenido de la dirección de memoria DirA, es decir, el contenido de la dirección de memoria 0x1001. Por lo tanto en MDR está el dato 0b10101010. Ahora es necesario pasar ese dato al Registro R0

R10 ← MDR

Y con esto se termina de ejecutar la instrucción.

EJERCICIOS

Suponga que la instrucción ADD realiza la suma entre el Operando1 y el Operando2 y guarda el resultado de la operación en el Operando1.

ADD Operando1, Operando2

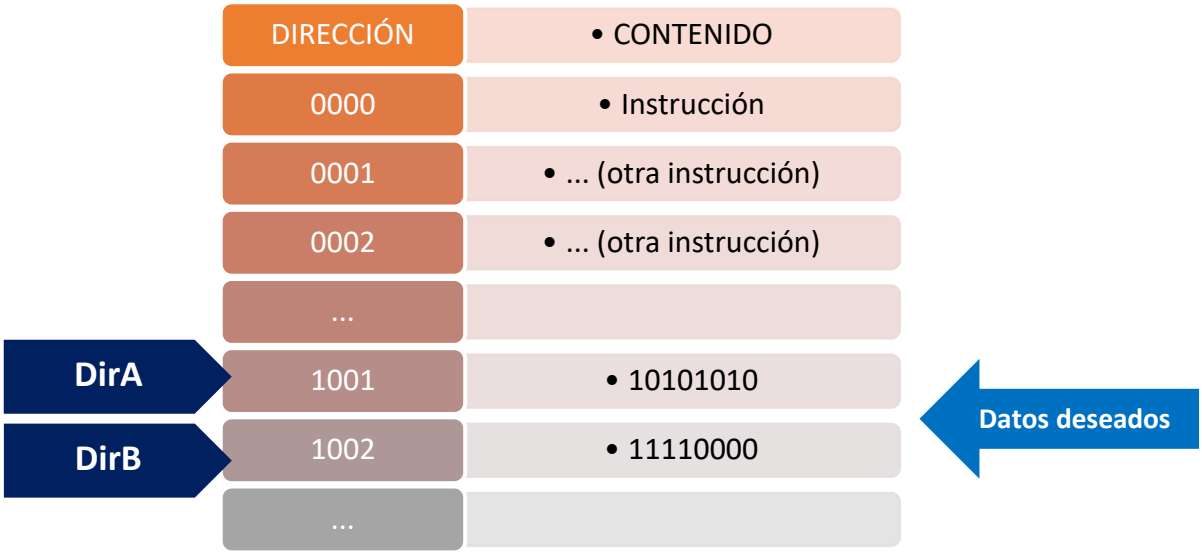
Describa todos los pasos necesarios para ejecutar las siguientes instrucciones (en forma independiente), suponiendo que para cada caso la instrucción se encuentra almacenada en la línea 0000 y la memoria se encuentra como se muestra a continuación:

ADD R0, R1

ADD R0, DirA

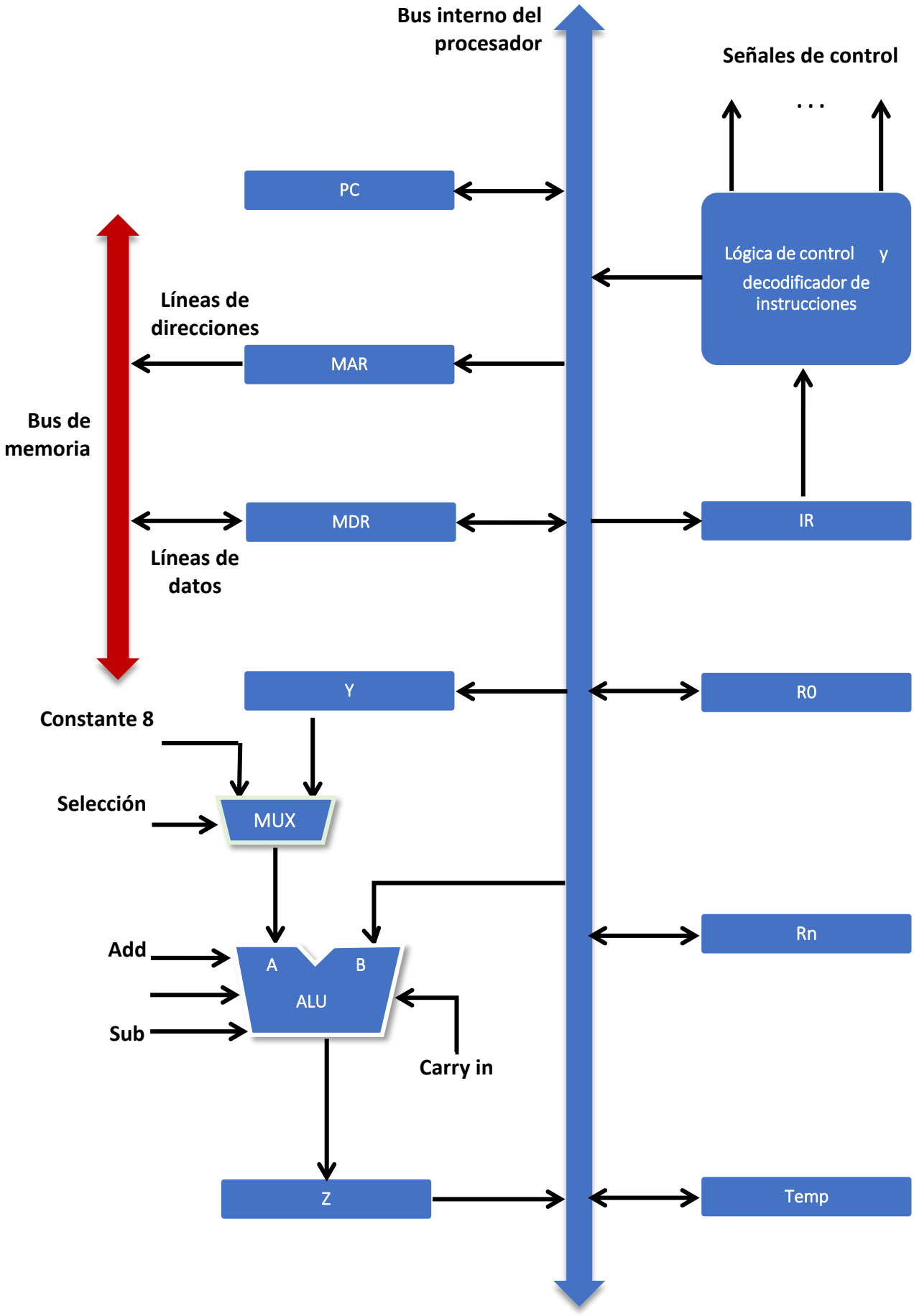
ADD DirA, R0

ADD DirA, DirB



Fases de ejecución

Para llevar a cabo la ejecución de un programa, el procesador toma una instrucción a la vez y realiza la operación especificada. Las instrucciones son tomadas de locaciones de memoria sucesivas hasta que se encuentra un brinco. El procesador lleva un registro de la última dirección de memoria en donde se encuentra la instrucción ejecutada usando el contador de programa. Después de ejecutar una instrucción el PC se actualiza para que apunte a la siguiente.


Bus interno del procesador
Señales de control
PC
Lógica de control y decodificador de instrucciones
Líneas de direcciones
MAR
Bus de memoria
MDR
IR
Líneas de datos
Y
R0
Constante 8
Selección
MUX
Add
A
B
ALU
Sub
Carry in
Rn
Z
Temp

Otro registro importante en el procesador es el IR. Supongamos que cada instrucción emplea 4 bytes que son almacenados en una palabra de memoria. Para llevar a cabo una instrucción, el procesador lleva a cabo los siguientes pasos:

1) El PC apunta a la dirección de memoria. Entonces se determina que el contenido de esta dirección es una instrucción y ésta se almacena en el IR.
2) El contenido del PC se incrementa 4 bytes
3) Se lleva a cabo la instrucción especificada en el IR

En caso de que las instrucciones ocupen más de una palabra, los pasos 1 y 2 deben repetirse las veces necesarias para terminar de cargar la instrucción. Estos dos primeros pasos son conocidos como "fetch phase" y el paso 3 se conoce como "execution phase".

En la figura se muestra la construcción interna de un procesador en el cual, tanto el ALU como los registros, se encuentran interconectados a un bus común. Este bus se encuentra en forma interna dentro del procesador y no debe confundirse con el bus externo que conecta al procesador con los dispositivos de entrada y salida.

Los registros, el ALU y el bus interno usualmente son conocidos en conjunto como "datapath".

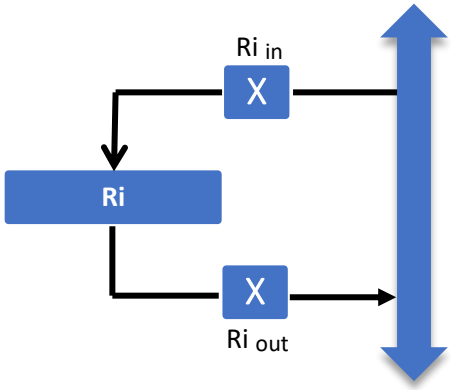
Con muy pocas excepciones, una instrucción puede ser ejecutada llevando a cabo una o más de las siguientes operaciones, en algún momento determinado:

- Transferir una palabra de datos de un registro a otro, o bien al ALU.
- Llevar a cabo una operación lógica o aritmética y guardar el resultado en el procesador.
- Tomar el contenido de una locación de memoria específica y almacenarlo en un registro de procesador.
- Almacenar una palabra de datos de un registro en una localidad de memoria específica.

En los temas siguientes se verá en forma específica como es llevada a cabo cada una de estas operaciones utilizando un procesador simple como el mostrado en la figura.

Transferencia de registros

La ejecución de una instrucción implica una serie de pasos en los cuales los datos deben ser transferidos de un registro a otro. Para cada registro se emplean dos señales: una para transferir el dato del registro hacia bus y otra para permitir que los datos del bus entren en el registro.
Tanto la entrada como la salida de un registro Ri, se encuentran conectadas al bus por medio de switches controlados por las señales Ri_{in} y Ri_{out}



Todas las operaciones y transferencia de datos son llevadas a cabo en periodos de tiempo definidos por el reloj del procesador. Puede suponerse que los registros consisten en flip-flops que se activan por flancos de subida (esta será la forma en que se maneje la explicación, sin embargo también podrían darse otros casos, como que los flip-flops se activen por flancos de bajada).

Operaciones aritmético - lógicas

El ALU no cuenta con almacenamiento interno, únicamente realiza operaciones aritméticas y lógicas a dos operandos aplicados en las entradas A y B.
Para ver un ejemplo de cómo funcionaría la transferencia de registros, en conjunto con la realización de operaciones lógicas vea la figura de la siguiente página.

Obtención de una palabra de la memoria

Para obtener una palabra de información de la memoria, el procesador tiene que especificar la dirección de la localidad de memoria donde la palabra se encuentra almacenada, y también tiene que indicar que desea realizar una operación de lectura.
El procesador transfiere la dirección requerida al MAR, cuya salida se encuentra conectada con líneas de direcciones del bus de memoria; al mismo tiempo el procesador utiliza las líneas de control del bus de memoria para indicar que se requiere una operación de lectura. Cuando el dato solicitado se recibe en la memoria es almacenado en el MDR, de donde puede ser transferido a otro registro en el procesador.
Las conexiones requeridas se muestran en la figura de la siguiente página.
Tanto el proceso de lectura como el de escritura de la memoria, deben encontrarse regulados por ciclos de reloj. El procesador completa una transferencia interna en un ciclo de reloj, sin embargo, cuando se trata de una transferencia a dispositivos externos, la velocidad de respuesta varía dependiendo del dispositivo.
Para compensar la variabilidad en el tiempo de respuesta, el procesador espera hasta que reciba una señal en la que se le indique que la operación de lectura ha sido completada. A esta señal se le conoce como MFC (Memory Function Completed).

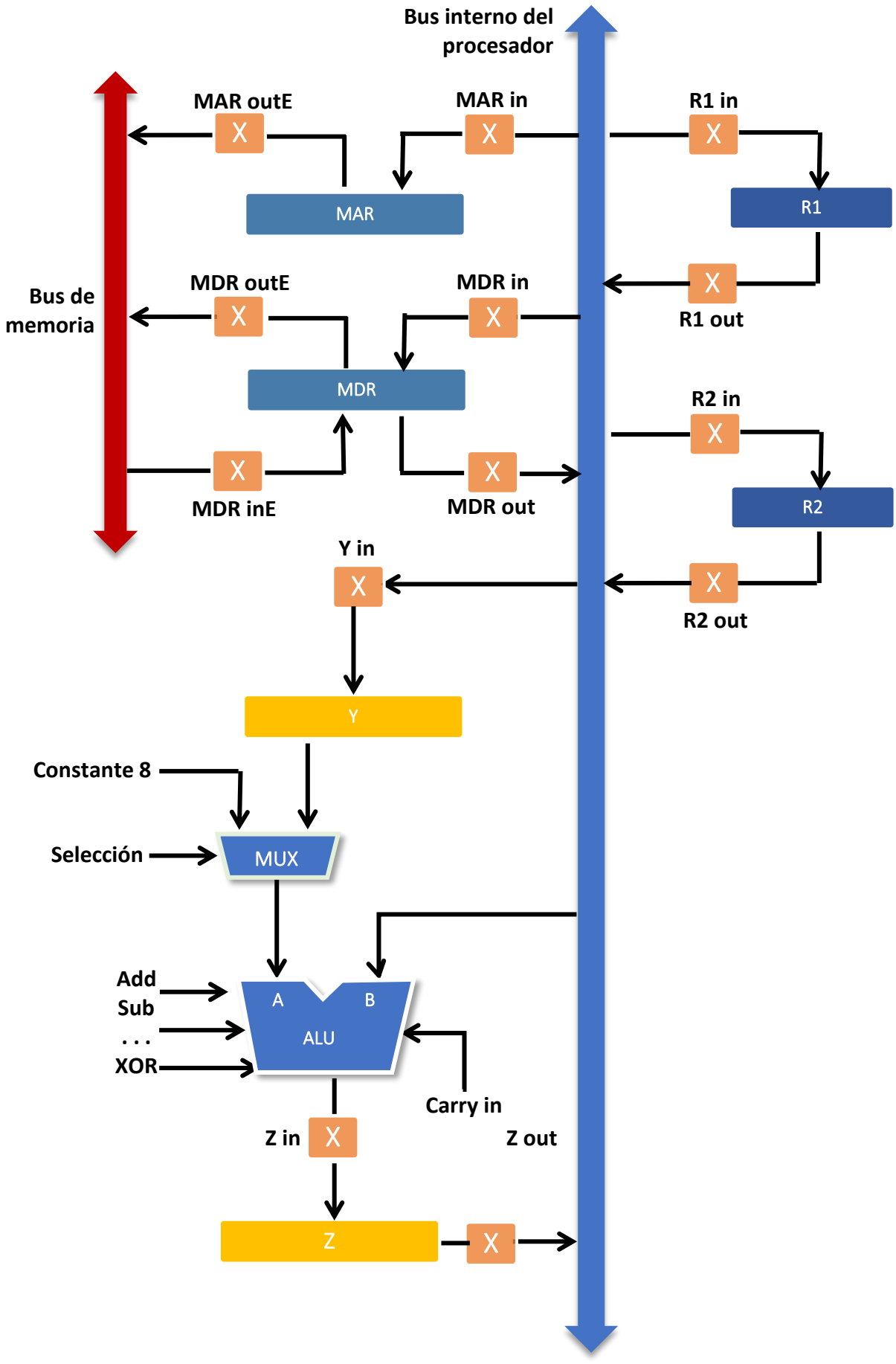
A manera de ejemplo digamos que se tiene la siguiente instrucción:

LD R4, X

La cual debe cargar en R4 el contenido de la dirección a la que apunta el registro X.
Después de haber ejecutado el fetch phase, los pasos necesarios para llevar a cabo esta operación son:

1) X_{OUT}, MAR_{IN}, READ
2) RMFC (Read Memory Function Completed)
3) MDR_{OUT}, $R4_{IN}$

Estos pueden llevarse a cabo en forma separada o bien varios a la vez. Cada uno puede completarse en un ciclo de reloj, excepto el segundo, que depende de la velocidad de respuesta de la memoria.


Bus interno del procesador
MAR outE
MAR in
R1 in
MAR
R1
Bus de memoria
MDR outE
MDR in
R1 out
MDR
R2 in
MDR inE
MDR out
R2
Y in
R2 out
Y
Constante 8
Selección
MUX
Add
Sub
. . .
XOR
A
B
ALU
Carry in
Z in
Z out
Z

Almacenamiento de una palabra en memoria.

Para escribir una palabra en memoria, se lleva a cabo un procedimiento muy similar al que se emplea para obtenerla de ella. El dato a ser escrito debe de ser cargado en MDR, la dirección en donde debe escribirse debe encontrarse en MAR y se envía entonces una señal de escritura.

A manera de ejemplo digamos que se tiene la siguiente instrucción:

ST X, R4

La cual debe cargar en la posición de memoria a la que apunta el registro X, el contenido del registro R4. Después de haber ejecutado el fetch phase, los pasos necesarios para llevar a cabo esta operación son:

1) $R4_{OUT}$, MDR_{IN},
2) X_{OUT}, MAR_{IN}, WRITE
3) WMFC (Write Memory Function Completed)

Al igual que en el caso anterior, después de solicitar la operación de escritura, el procesador espera hasta recibir la señal de respuesta indicando que se ha finalizado el proceso de escritura.

Ejecución completa de una instrucción

Ahora se unirán todos los elementos necesarios para ejecutar una instrucción.

Considere por ejemplo la instrucción: LD R15, X (En donde X es un registro del microprocesador que contiene una dirección de memoria). Esta instrucción se encarga de copiar el contenido de la localidad de memoria a la que apunta X al registro R15. Para llevar a cabo esta instrucción se requieren las siguientes operaciones:

1. Obtener la instrucción
2. Obtener el contenido de la localidad de memoria a la que apunta X
3. Almacenar ese contenido en el registro R15.

Los pasos en forma detallada son:

Paso	Operación
1	PC_{out}, MAR_{in}, SELECT8, ADD, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, READ, RMFC
3	MDR_{out}, IR_{in}
4	X_{out}, MAR_{in}, READ, RMFC
5	MDR_{out}, $R15_{in}$, END

En el paso 1 se inicia la obtención de la instrucción, cargando el contenido del PC en el MAR y enviando una instrucción de lectura a la memoria. Mientras tanto, la señal de SELECT es puesta en SELECT8, lo cual causa que el multiplexor deje pasar la constante 8. Ese valor es sumado al operando que se encuentra en B (que es el contenido del PC). El resultado queda momentáneamente en el registro Z, entonces ese valor es direccionado nuevamente para almacenarse en PC. Todo esto se hace mientras que se aprovecha para esperar que la memoria mande el pulso de respuesta de que ha terminado el proceso de lectura.

En el tercer paso, la palabra obtenida de la dirección de memoria a la que apuntaba PC, es almacenada en IR, de forma que en este punto se conoce la instrucción que se desea ejecutar. Del paso 1 al 3, se lleva a cabo lo que se conoce como "fetch phase", y siempre se lleva a cabo de la misma forma, sin importar la instrucción (pues precisamente estos tres pasos sirven para que el microprocesador pueda cargar dicha instrucción en el IR y así reconocerla).

A partir del paso 4, los circuitos de control se encargan de interpretar el contenido del IR. Para la instrucción que se propuso en el ejemplo, el microprocesador necesita leer el contenido de la dirección a la que apunta el registro X, es por ello que lo primero que hace es transferir el contenido de X a MAR y mandar un pulso de lectura. Después deberá esperar hasta que la memoria termine con la lectura e indique que el dato ya se encuentra en MDR. Entonces se deja pasar el dato de MDR a R15. Después de esto se indica que se ha terminado el ciclo y por lo tanto se puede proceder a realizar otra instrucción comenzando nuevamente con el paso uno.

En esta instrucción se utilizaron todas las señales que se muestran en la tabla de instrucciones, con excepción de Y_{in} en el paso 2, pues no fue necesario almacenar el contenido del PC en el registro Y al ejecutar la instrucción, sin embargo, esto será útil cuando se desean realizar instrucciones de brinco.

Ejecución completa de una instrucción de brinco

Las operaciones de salto se encargan de cargar un valor nuevo al contenido del Program Counter (PC).

Ejemplo	RJMP	Etiqueta	(RELATIVE JUMP)

Hay instrucciones de salto condicional que únicamente provocan un salto cuando se satisface determinada condición.

Ejemplo	BREQ	Etiqueta	(BRANCH IF EQUAL)
	BRNE	Etiqueta	(BRANCH IF NOT EQUAL)

La instrucción RJMP reemplaza el contenido del PC por la dirección a dónde se desea "brincar". Esta dirección usualmente se obtiene sumándole al PC una cantidad X que es indicada por la instrucción. A continuación, se puede ver la secuencia que se encarga de implementar la instrucción RJMP (que es un brinco incondicional).

Paso	Operación
1	PC_{out}, MAR_{in}, SELECT8, ADD, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, READ, RMFC
3	MDR_{out}, IR_{in}
4	Offset-de-IR_{out}, ADD, Z_{in}
5	Z_{out}, PC_{in}, END

El proceso inicia con el fetch phase que es el mismo para cualquier instrucción (pasos del 1 al 3).

En el paso 4, el valor de X que se le agrega al PC (valor de Offset) es extraído del IR. Recordemos que el valor del PC se encuentra entonces en el registro Y, y el valor de Offset se encuentra en el bus y entonces se suma con PC. El resultado queda temporalmente almacenado en la variable Z.

En el quinto paso el valor de Z es enviado al PC y entonces se envía la señal de END para indicar que se puede continuar con la siguiente instrucción.

A manera de ejemplo digamos que, si la instrucción de RJMP se encuentra en la localidad de memoria 0x1000 y el brinco debe ir a la dirección 0x1200, el valor del offset será de 0x200.

Códigos de condición

La información sobre el resultado de diversas operaciones es guardada por el procesador para su uso en posteriores instrucciones de salto condicional. Esta información se guarda en bits individuales que se conocen como indicadores de código de condición o banderas y que se agrupan en un registro especial llamado Registro de Estado.

Normalmente se manejan por lo menos cuatro banderas:

N (NEGATIVO) – Su valor es 1 cuando el resultado de la operación es negativo y 0 cuando es positivo

Z (ZERO) – Su valor es 1 cuando el resultado de la operación es cero y 0 cuando es cualquier otro valor.

C (CARRY) – Su valor es 1 cuando se produce un acarreo durante una operación aritmética.

V (DESBORDAMIENTO) – Su valor es 1 cuando ocurre un desbordamiento en una operación aritmética.

Subrutinas

En los programas a menudo es necesario realizar una sub tarea que se tenga que repetir en varias ocasiones, sin embargo, si esa sub tarea se programa cada vez que se requiere usar, se ocuparían muchos lugares de memoria. Es por ello que, para ahorrar espacio, se guarda solamente una copia de ese conjunto de instrucciones que se repiten constantemente, y de esta forma se les manda a llamar cada vez que sea necesario. A este conjunto de instrucciones se le llama subrutina. Tras la ejecución de una subrutina el programa que lo llamó debe regresar al punto en el que se había quedado.

Para llamar a una subrutina se emplea la instrucción CALL o RCALL, la cual realiza las siguientes operaciones

- Almacena el contenido del Program Counter (PC) en un registro temporal
- Salta a la dirección de la subrutina

Para que la subrutina termine y el programa continúe ejecutándose se emplea la instrucción RETURN o RET, la cual realiza la siguiente operación.

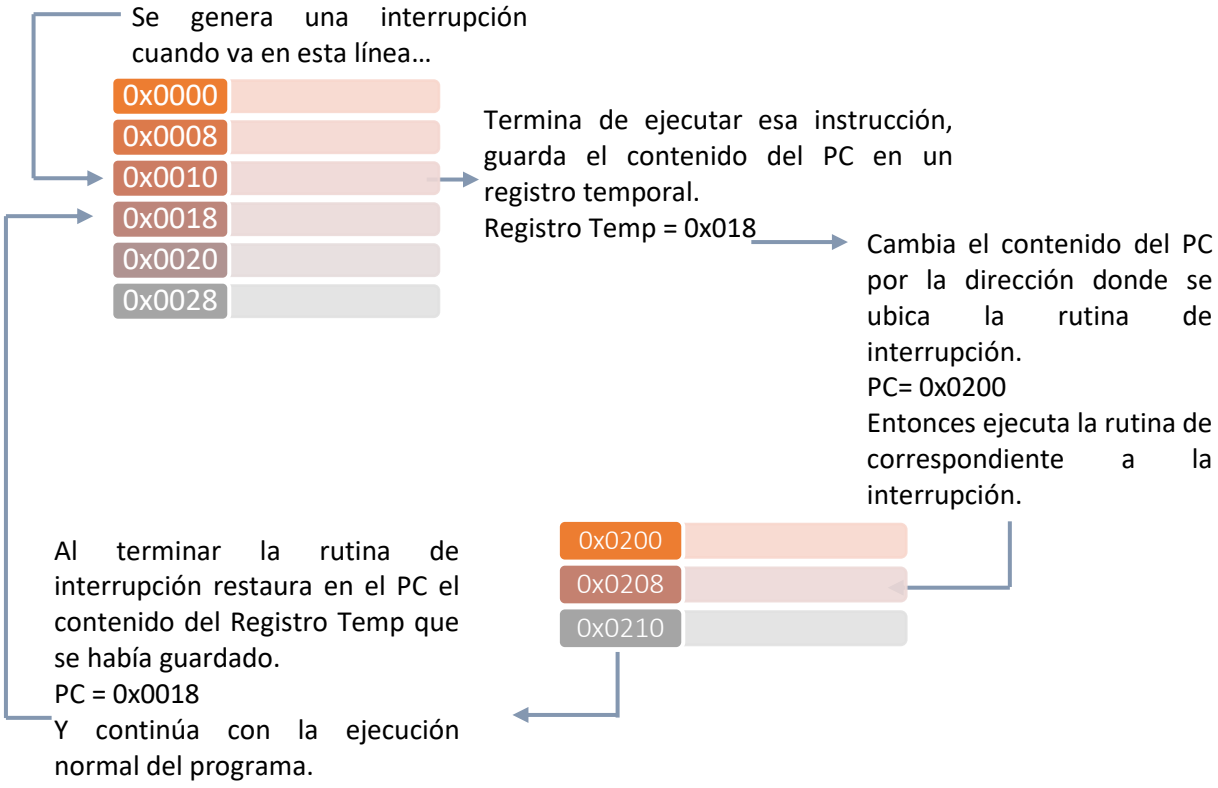
- Le devuelve al PC el valor que se había almacenado en un registro temporal cuando se había ejecutado la instrucción CALL.

Interrupciones

Existen situaciones en las cuales es necesario que el programa deje de hacer la tarea que está realizando, para darle lugar a alguna otra un poco más urgente. Esto lo puede realizar mediante el envío de una señal de hardware al procesador denominada interrupción. Normalmente se dedica una de las líneas del bus de control, llamada "línea de petición de interrupción", para este propósito.

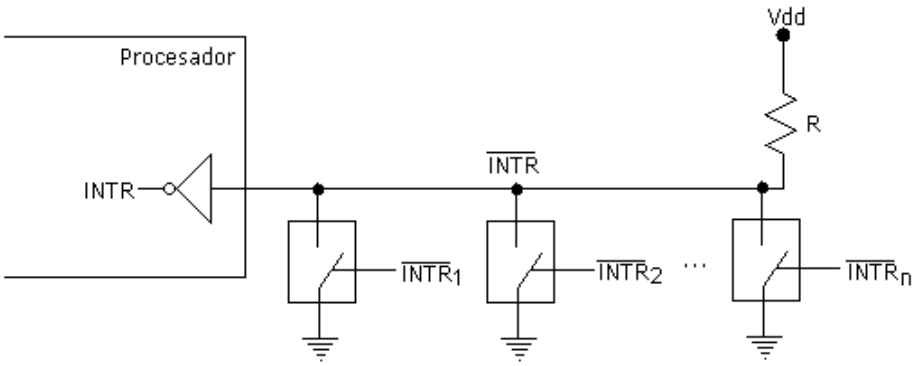
Cuando se ejecuta una interrupción, se realiza un proceso muy parecido a cuando se realiza una subrutina, pero la diferencia está en que una subrutina se realiza cuando el programa la llama, sin embargo, una rutina de interrupción se puede ejecutar en cualquier momento en que se recibe la interrupción, sin importar en qué parte del programa se encuentre en ese momento.

Es importante hacer notar que en procesador ATMEL con el que se trabajará durante el curso, al mandar a llamar una interrupción, los contenidos de los registros no se respaldan, por lo cual, si dentro de la interrupción se modifica el contenido de éstos, el cambio prevalecerá al regresar a ejecutar comúnmente el programa.



INTERRUPCIONES HARDWARE

Habrá ocasiones en las que se desee que múltiples dispositivos tengan la posibilidad de activar una misma interrupción. En la figura se ilustra la manera en la que dichos dispositivos pueden ser conectados.



Para pedir una interrupción el dispositivo cierra la línea correspondiente, así si todos los dispositivos tienen sus interrupciones desactivadas, la línea INTR tendrá Vdd (1 lógico) y el estado de INTR en el procesador será 0 lógico, pero si una de las interrupciones de los dispositivos se activa, entonces el circuito se cerrará, la línea INTR tendrá tierra (0 lógico) y el procesador tendrá un 1 lógico que le indicará que una interrupción ha sido activada.

Habilitando y deshabilitando interrupciones

Un programador debe tener la capacidad de decidir en todo momento si quiere que las interrupciones detengan la ejecución del programa para llevar a cabo la rutina de interrupción, o si no desea reconocerlas en algún momento. Una propiedad importante que tienen los procesadores es la capacidad de habilitar y deshabilitar las interrupciones en la forma que se desee.

Consideremos un caso cuando se envía una interrupción desde un dispositivo, entonces se activará la señal de interrupción y se entrará a la rutina correspondiente, mientras tanto el dispositivo mantendrá activa la señal de interrupción hasta que observe que el procesador la atiende, es decir que la señal estará activa mientras se ejecuta la interrupción. Es esencial asegurar que esta señal de petición activa no conlleva sucesivas interrupciones, causando que el sistema entre en un bucle infinito del cual no podría salir, para ello se emplean diversos mecanismos, a continuación describiremos tres de ellos:

a) Que el hardware del procesador ignore la línea de petición de interrupción hasta que se haya terminado de ejecutar la primera línea de la rutina de interrupción, después, mediante la instrucción de deshabilitar las interrupciones (en la primera línea) se asegura que no se cicle el programa. Generalmente, si se usa este método, la última instrucción de la rutina de interrupción será volver a habilitar las interrupciones.

b) Que el procesador automáticamente deshabilite las interrupciones antes de comenzar una rutina de interrupción, y vuelva a habilitarlas automáticamente cuando termine de ejecutar dicha rutina.

c) Que los circuitos que detectan la interrupción respondan solo por flanco, es decir que sean “disparados por flanco”, en este caso el procesador recibirá una única petición independientemente de cuánto tiempo esté activa la línea.

Modos de direccionamiento

Un programa opera con datos que residen en la memoria del procesador.

Se llama “modo de direccionamiento” a las diferentes formas en que la dirección de memoria de un operando puede ser especificado en una instrucción

OPERACIÓN Destino, Fuente

Direccionamiento modo inmediato

Sintaxis en ensamblador: VALOR
Dirección efectiva: El Operando = VALOR
El operando se proporciona en forma explícita en la instrucción, puede ser en forma decimal (o en binario si se escribe 0b antes del número, o en hexadecimal si se escribe 0x antes del número)

Ejemplos:	LDI R10, 200	(LOAD IMMEDIATE)
	CPI R10, 0Xff	(COMPARE WITH IMMEDIATE)
	ORI R10, 0b11110000	(LOGICAL OR WITH IMMEDIATE)

Direccionamiento modo registro

Sintaxis en ensamblador: Ri
Dirección efectiva: Ri
El operando corresponde al contenido del registro del procesador, el nombre o dirección del registro se proporciona en la instrucción.

Ejemplos:	CP R10, R11	(COMPARE)
	MOV R10, R11	(COPY)

Direccionamiento modo absoluto (modo directo)

Sintaxis en ensamblador: DIRECCIÓN
Dirección efectiva: DIRECCIÓN
También se conoce como modo directo, el operando se encuentra en la dirección de memoria, la dirección se proporciona explícitamente en la instrucción.

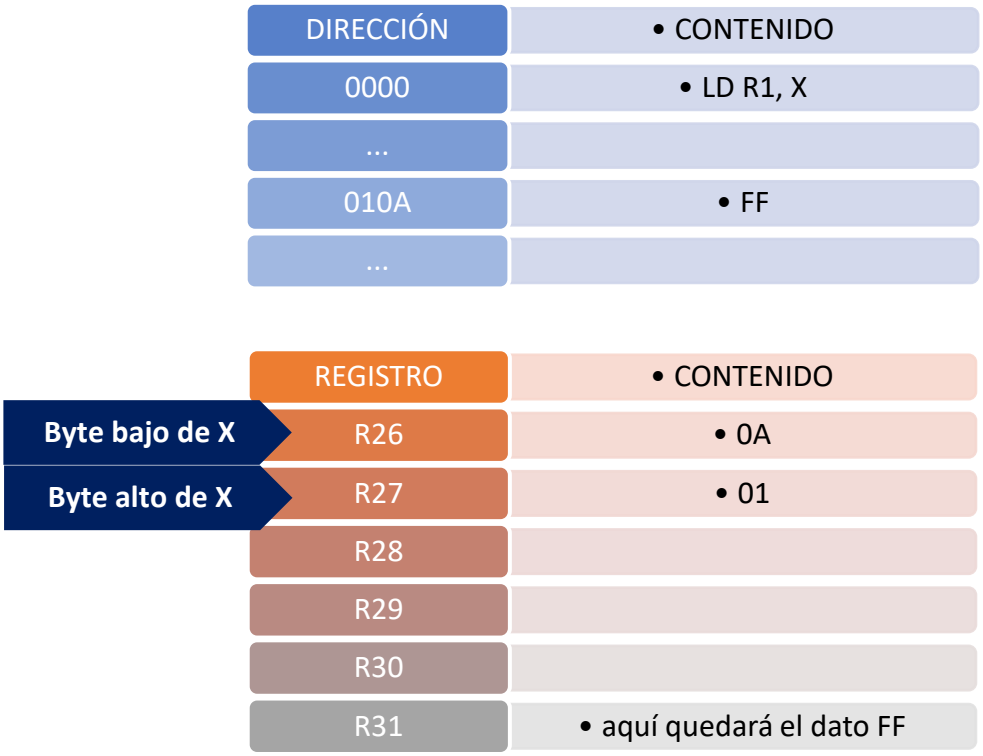
Ejemplos: LDS R2, 0XFF00 (LOAD DIRECT FROM DATA SPACE)
LDS R10, 0x0F0F (LOAD DIRECT FROM DATA SPACE)

Direccionamiento modo indirecto

Sintaxis en ensamblador: X, Y, Z
Dirección efectiva: lo que hay en la localidad de memoria a la que apunta X, Y o Z
La dirección efectiva se encuentra en la posición de memoria cuya dirección se encuentra almacenada en la palabra X, Y o Z

Ejemplos: LD R31, X (LOAD INDIRECT FROM DATA SPACE TO REGISTER USING INDEX X)
ST Y, R1 (STORE INDIRECT FROM REGISTER TO DATA SPACE USING INDEX Y)

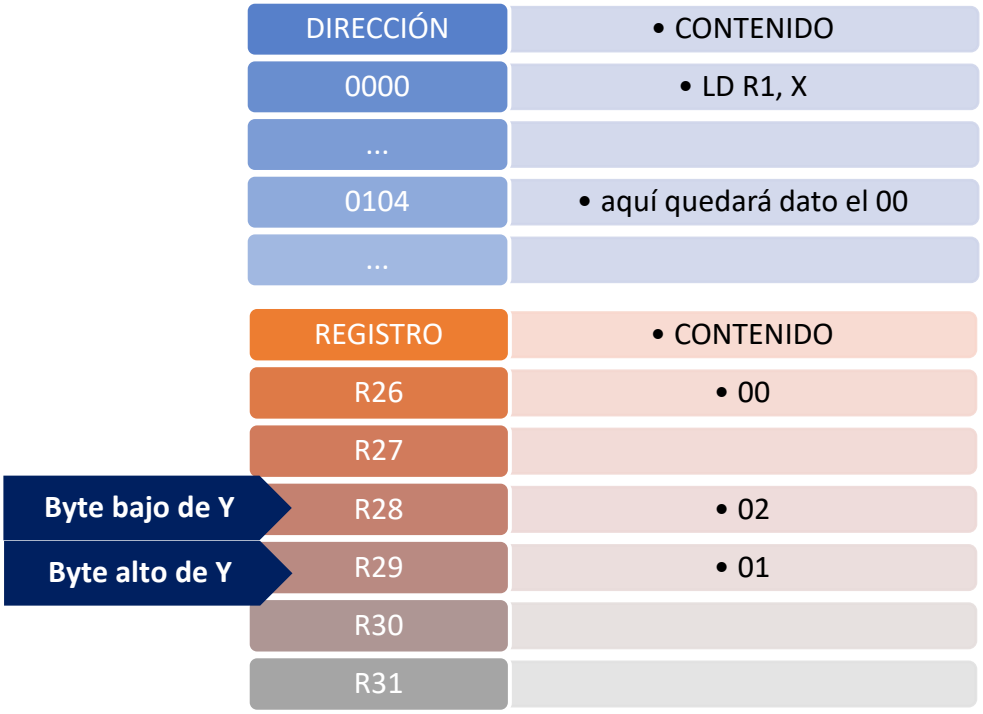
Analizando el primer ejemplo LD R31, X. La instrucción LOAD se encarga de copiar el contenido de la localidad de memoria a la que apunta X al registro R31, así pues, al final de la operación el Registro R31 tendrá guardado el dato al que apunta la dirección de memoria almacenada en X (010A), es decir tendrá un FF



Direccionamiento modo indexado

Sintaxis en ensamblador: X+q, Y+q, Z+q
Dirección efectiva: La dirección a la que apunta el registro Y + q
La dirección efectiva del operando corresponde a la dirección a la que apunta el registro Y más el valor constante q que se conoce como Offset.

STD Y+2, R26 (STORE INDIRECT WITH DISPLACEMENT FROM REGISTER TO DATA SPACE USING INDEX Y)
La instrucción STD que se muestra en el ejemplo sumará el contenido del registro Y, que es 0102 más la constante q, que en este caso es 2. Entonces 0102+2=0104. Y almacenará el contenido de R26 (es decir un 00) en la dirección de memoria a la que apunta Y+2, es decir en la dirección de memoria 0104.



Direccionamiento modo autoincremento

Sintaxis en ensamblador: X+, Y+, Z+
Dirección efectiva: La dirección a la que apunta el contenido del registro X, Y o Z.
La dirección efectiva del operando corresponde a la dirección a la que apunta el registro X, Y o Z, después de realizar la operación incrementa el contenido del registro X, Y o Z de forma que X = X + 1 o Y = Y +1 o Z = Z + 1.

Ejemplo: LD R4, X+ (LOAD INDIRECT FROM DATASPACE TO REGISTER USING INDEX X)
ST Y+, R5 (STORE INDIRECT FROM REGISTER TO DATASPACE USING INDEX Y)

Direccionamiento modo autodecremento

Sintaxis en ensamblador: -X, -Y, -Z
Dirección efectiva: La dirección a la que apunta el contenido del registro X – 1, Y – 1 o z – 1.
Este tipo de direccionamiento decrementa el contenido del registro X, Y o Z, de forma que X = X – 1 o Y = Y – 1 o Z = Z – 1, y la dirección efectiva del operando corresponde a la dirección a la que apunta el registro X, Y o Z ya decrementado.

Ejemplo:	LD R4, -X	(LOAD INDIRECT FROM DATASPACE TO REGISTER USING INDEX X)
	ST –Y, R5	(STORE INDIRECT FROM REGISTER TO DATASPACE USING INDEX Y)

EJERCICIO

En las siguientes instrucciones, en cada operando indica el modo de direccionamiento empleado y explica cuál será el resultado de llevar a cabo dicha instrucción (usando los datos proporcionados, y suponiendo que cada operación es independiente de las demás).

DIRECCIÓN	• CONTENIDO
0x0100	• 0x15
0x0101	• 0x31
0x0102	• 0x89
0x0103	• 0x10
0x0104	• 0x43
0x0105	• 0x32

REGISTRO	• CONTENIDO
R26	• 0X03
R27	• 0x01
R28	• 0x00
R29	• 0x01
R30	• 0x04
R31	• 0x01

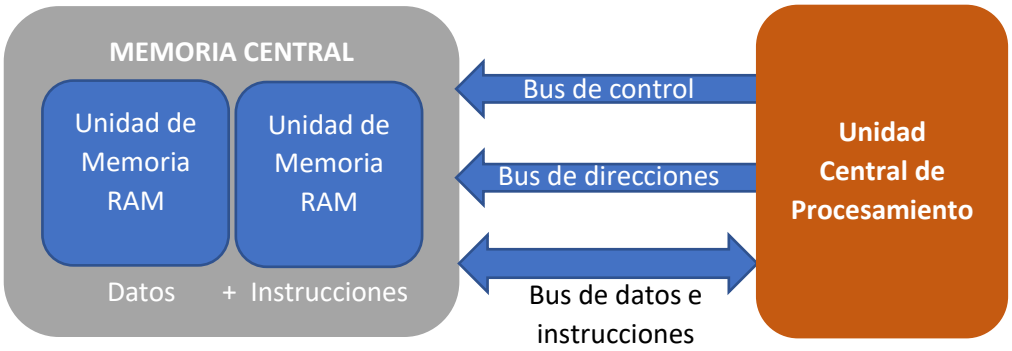
INSTRUCCIÓN	OPERANDO 1	OPERANDO 2	¿QUÉ CAMBIA AL FINAL DE LA INSTRUCCIÓN?
MOV R31, R30			
LD R31, X			
LDS R30, 0x0102			
ST –Z, R26			
STD X+2, R31			
CPI R30, 0x04			
LD R29, Y+			

Clasificación de los microprocesadores

Clasificación de acuerdo con el tipo de arquitectura

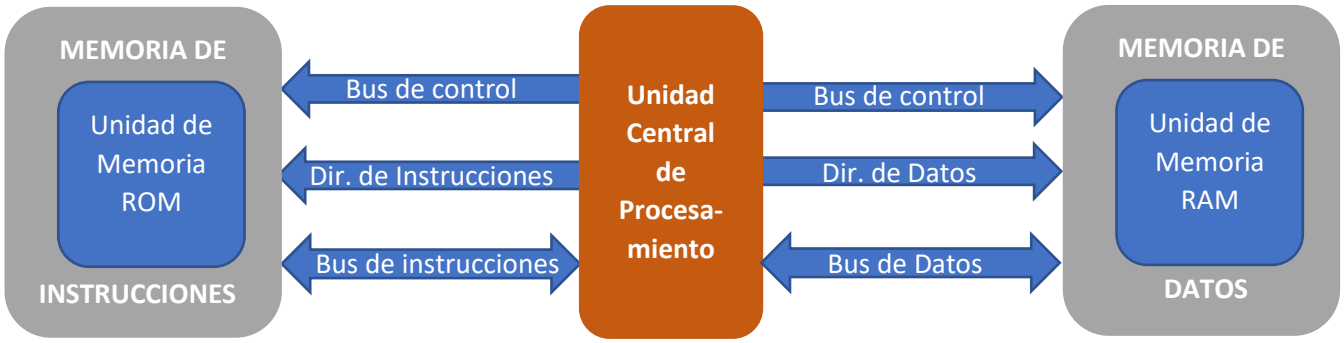
Arquitectura Von Neumann

Esta arquitectura se caracteriza porque utiliza el mismo almacenamiento para datos e instrucciones, conectando la memoria a través de un único bus de datos y de direcciones. Tiene el inconveniente de que solamente se pueden llevar o datos o instrucciones en un momento determinado.



Arquitectura Harvard

En esta arquitectura se utilizan almacenamientos (memorias) separadas para las instrucciones y para los datos, y se tienen dos sistemas completos de buses, uno para datos y otro para instrucciones, de esta forma es posible llevar simultáneamente datos e instrucciones, permitiendo mayores velocidades de procesamiento.



Clasificación de acuerdo con el tipo de instrucciones

Históricamente, los procesadores han sido clasificados también como CISC (Complex Instruction Set Computer) o RISC (Reduced Instruction Set Computer) dependiendo de las instrucciones que son capaces de ejecutar.

Arquitectura CISC

Los microprocesadores CISC se caracterizan por tener un conjunto de instrucciones amplio, que les permite llevar a cabo instrucciones complejas entre operandos que pueden estar situados en los registros internos o en la memoria.

Son procesadores capaces de ejecutar cientos de diferentes instrucciones complejas, dándole así gran versatilidad.

En la actualidad CISC tiene al x86 como su mayor representante, con AMD y sobre todo Intel a la cabeza de su desarrollo. Hay muchos ejemplos históricos como los PDP, Motorola 68000, Intel 4004 o Intel 8086.

Arquitectura RISC

Este tipo de arquitectura se caracteriza por tener instrucciones de tamaño fijo y presentadas en un reducido número de formatos (el término “Reduced” se refiere a la complejidad)

También se caracteriza porque sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos. De esta forma se tiene un conjunto de instrucciones pequeñas y simples que toman menos tiempo en ejecutarse.

Un procesador de tipo RISC es más simple tanto en software (instrucciones) como en hardware (registros de memoria), lo cual hace que sea más barato. En la actualidad el mayor ejemplo de procesador RISC son los productos ARM, utilizados ampliamente en dispositivos móviles.

UNIVERSIDAD PANAMERICANA

Agradecimientos

- A a todos los alumnos que han llevado mi curso de microcontroladores a lo largo ya de varios años, ustedes han sido la motivación para realizar este manual, buscando con el facilitar la adquisición de conocimientos e incentivar su capacidad de investigación para adquirir aún más. De no ser por ustedes no hubiese sido posible la realización de este manual.

- A mi mamá y a mi hija, quien siempre ha sido un gran apoyo para mi al motivarme e impulsarme para buscar siempre crecer tanto a nivel personal como profesional.

- Al Dr. Enrique Arámbula, quien fue mi profesor de microcontroladores al estudiar la carrera de Ingeniería en Electrónica y Sistemas Digitales, gracias por formar en mí el gusto por la electrónica digital y por la investigación.

- A la Universidad Panamericana, campus Aguascalientes, por haber confiado en mí desde el inicio y por brindarme la posibilidad de trabajar con ellos en esta labor que para mí resulta apasionante.

- A mis colegas en la universidad por sus consejos y amistad, gracias por su invaluable apoyo.