Guía de Lenguajes de Interfaz

Antonio Emiko Ochoa Adame

19 de febrero de 2019

1. Importancia del lenguaje Ensamblador

Si se entiende el lenguaje ensamblador se pueden hacer programas eficientes porque puede ver qué se está programando y relacionarlo con ensamblador.

Esto es fácil de ver en lenguajes de medio nivel (como C) donde es fácil de apreciar cómo sería dicho código, pero en ensamblador.

2. Lenguaje Máquina

Es el lenguaje que entiende el procesador, es decir, unos y ceros (1s y 0s).

3. Lenguaje Ensamblador

Es el lenguaje que es más fácil de entender para el programador.
El lenguaje ensamblador está basado en **nemónicos**.

Aprender el código de cada intrucción es muy complicado para el programador, así los **nemónicos** son usados en su lugar.

Nemónicos.- Palabras cortas que dan una idea de la instrucción.

Ejemplos de **nemónicos**: ADD (suma), LD (cargar), sub (restar).

OPCODE (Código de operación).- Lenguaje que entiende el procesador e indica la instrucción a realizar.

Dos procesadores son **compatibles** (equivalentes) si tiene el **mismo número de instrucciones** y el **mismo OPCODE para la misma intrucción**.

4. Memorias

- Secuencial
- No secuencial
- Volátil
- No volátil (ROM)

4.1. Clasificación de memorias

4.1.1. Memoria secuencial y no secuencial

Para acceder a una determinada posición de una localidad un una memoria secuencial , es necesario pasar por las localidades adyacentes.

Ejemplos: Cassettes (se tenía que rebobinar).

¿Cómo funcionan las memorias actuales?

Tienen un bus de direcciones donde se indica la localidad a accesar (no secuencial).

4.1.2. Memoria volátil y no volátil

Memoria volátil.- La información se mantiene siempre y cuando esta se mantenga energizada; La información se pierde si se elimina el voltaje.

Memoria no volátil.- La información se mantiene aún sin voltaje.

4.1.3. Memoria estática y dinámica

La memoria **RAM** se divide en:

- Estática → utiliza **transistores**
- Dinámica → utiliza **capacitores**

Memoria estática

Mientras se encuentre energizada la información se mantiene.

Está basada en transistores.

Memoria dinámica

Además de estar energizada, se requiere refrescar la información para no perderla.

Está basada en capacitores. Un capacitor pierde su carga con el tiempo.

El sistema operativo es el encargado de leer y escribir a información de esta memoria; esto se hace cientos de veces por segundo.

Bit en una memoria dinámica:

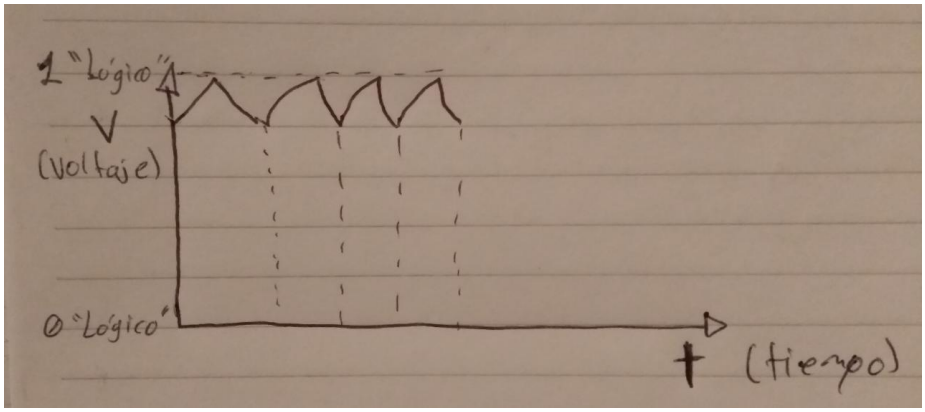

Figura 1: Refresco de bit

En la figura 1 se puede observar cómo el sistema operativo lee el bit de información y lo vuelve a escribir.

4.2. Memoria ROM

- EPROM
- E^2PROM
- Flash

4.2.1. Memoria EPROM (Programable eléctricamente)

Solo puede programar bits en un cambio de **1 a 0**, pero **no** puede convertir de **0 a 1**

Proceso de borrado.- Poner los bits en 1. (se borra con luz UV).

Se caracteriza por tener una ventan de cuarzo transparente para que la luz UV pueda entrar y así borrar la información (poner los bits en 1). Posteriormente, con un programador se ponen los bits deseados a 0.

En las PCs antiguas, el BIOS tenía una memoria EPROM donde se cubría la ventana con cinta negra para evitar la luz UV.

4.2.2. Memoria E^{2}PROM (Programable y borrable eléctricamente)

Con el mismo programador se pueden poner los bits a 0 y a 1 (borrar).

Cualquier localidad de memoria puede programarse de manera individual.

4.2.3. Memoria Flash

Es muy parecida a la E^{2}PROM porque puede borrarse y programarse de manera elétrica.

Permite mayor densidad de información.

Es más económica.

Desventaja:

No se puede programar una localidad de manera independiente; se tiene programar un **bloque (o página)** completo.

Ejemplo de cómo se modifican las memorias:

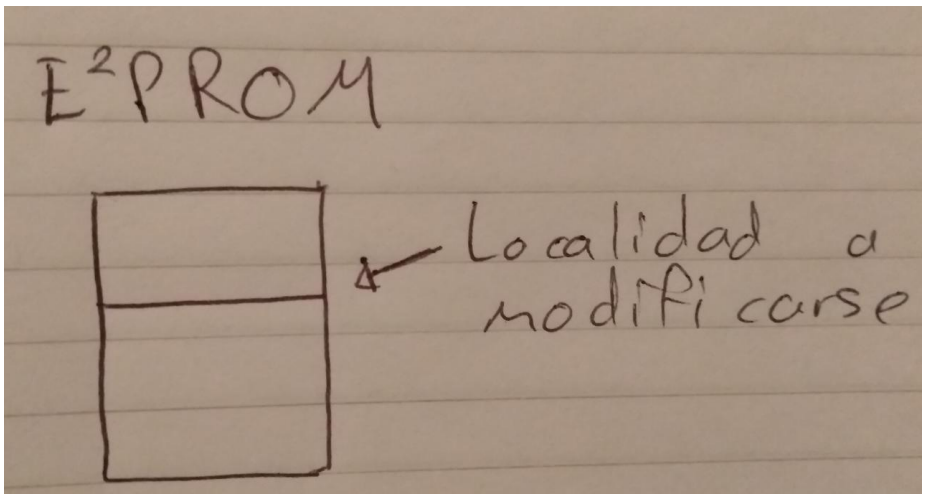

Figura 2: Memoria E^{2}PROM

En la figura 2 se muestra cómo simplemente se modifica la localidad deseada sin ningún problema.

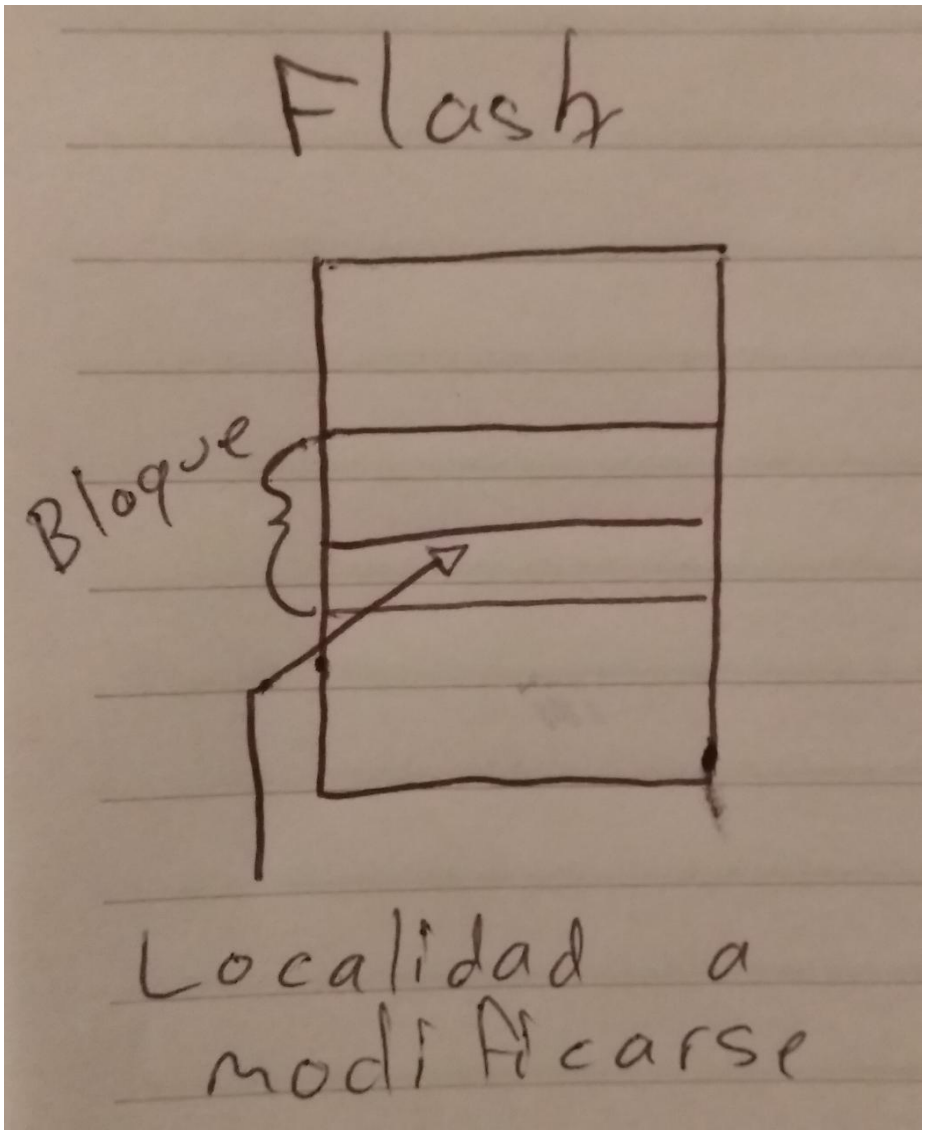

Figura 3: Memoria Flash

En la figura 3 se muestra cómo es el proceso de escribir en este tipo de memoria; hay tres pasos:

1. Se leen 512 bytes
2. Se modifica la localidad deseada
3. Se escribe el bloque completo (los 512 bytes)

5. Arquitectura de un procesador

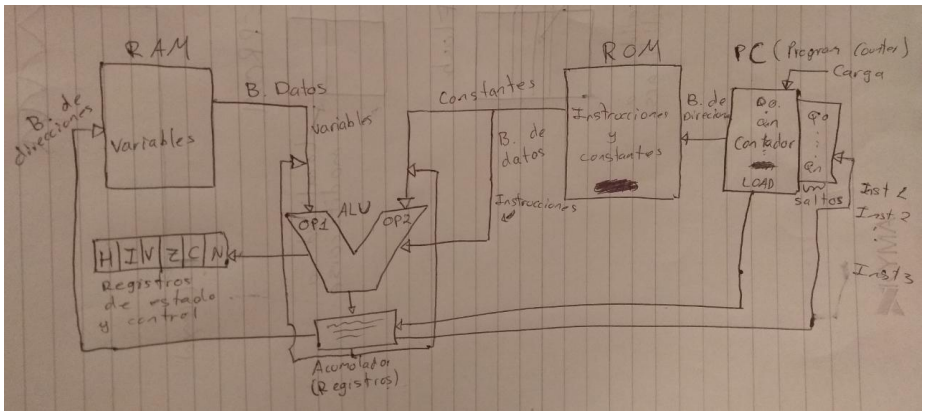

Figura 4: Arquitectura de un procesador

A continuación se explica la arquitectura de un procesador como se muestra en la figura 4:

La ROM guarda tanto las intrucciones como las constantes.

La RAM guarda variables.

Los operandos de la ALU pueden ser variables, constantes y registros, pero siempre uno de los operando debe ser un **registro**.
No se pueden hacer operaciones con 2 variables, 2 constantes o una variables y una constante.

La ALU decodifica la instrucción. Cada instrucción tiene asociado un **OP-CODE** (Código de operación), cuando la ALU detecta el **OPCODE**, sabe qué operación debe realizar con los operandos.

Cuando se hace una operación se modifican las banderas (bits del registro de estados y control) que indican las características del resultado.

6. Números sin signo

El valor más pequeño siempre es 0.

Rango de número sin signo de 8 bits [0, 255], 9 bits [0, 511], 10 bits [0, 1023], etc.

Fórmula: $[0, 2^n - 1]$ donde n es el número de bits del número.

7. Números con signo

Los números con signo son representados con complemento a 2.

Fórmula: $[-2^{n-1}, 2^{n-1} - 1]$ donde n es el número de bits del número.

Cuando un número con signo se representa en complemento a 2, puede mostrar tanto números negativos como positivos.

Se sabe que es negativo si el MSB (Most Significant Bit) es 1 y que es positivo si el MSB es 0.

"Sacar" o "hacer" el complemento a 2 significa que, si el número es positivo, hacerlo negativo y viceversa.
Hacer el complemento a 2 de un número es invertir el signo de dicho número.

Representar el complemento a 2 significa que, si el número es negativo, representarlo negativo y viceversa.

Ejemplo de números con signo de 8 bits:

$1111, 1111_2$: Negativo
$0101, 1111_2$: Positivo
1111_2: Positivo
1010_2: Positivo
$1000, 0000_2$: Negativo

Los números en complemento a 2 son fáciles de determinar cuando son positivos, pero cuando son negativos hay que hacer otros procedimientos.

7.1. Complemento a 2 (método 1)

Pasos:

1. Invertir los bits
2. Sumar una unidad

7.2. Complemento a 2 (método 2)

El MSB tiene una ponderación negativa.

8. Extensión de bit de signo

8.1. Números sin signo

$0101_2 = 5$
$00101_2 = 5$

$000101_2 = 5$
$0000000101_2 = 5$

8.2. Números con signo

$110_2 = -2$
$1110_2 = -2$
$111110_2 = -2$
$1111110_2 = -2$
$111111110_2 = -2$

¿Qué número represetan los siguientes números en complemento a 2 si son de 10 bits?

$a) 11,1111,1010_2 = -6$
$b) 11,1110,0100_2 = -28$
$c) 11,1111,1110_2 = -2$
$d) 11,1000,0010_2 = -126$
$e) 11,1101,1110_2 = -34$

Representar en complemento a 2 de 9 bits:

$-9_{10} = 1,1111,0111_2$
$-17_{10} = 1,1110,1111_2$
$-8_{10} = 1,1111,1000_2$
$-20_{10} = 1,1110,1100_2$

Antes de determinar un número hay que considerar lo siguiente:

1. ¿Están o no en complemento a 2?
2. Si están en complemento a 2, ¿De cuántos bits son?

Sobreflujo.- Indica un cambio indebido de signo.

Ejemplos de operaciones donde se indican los cambios en las banderas:

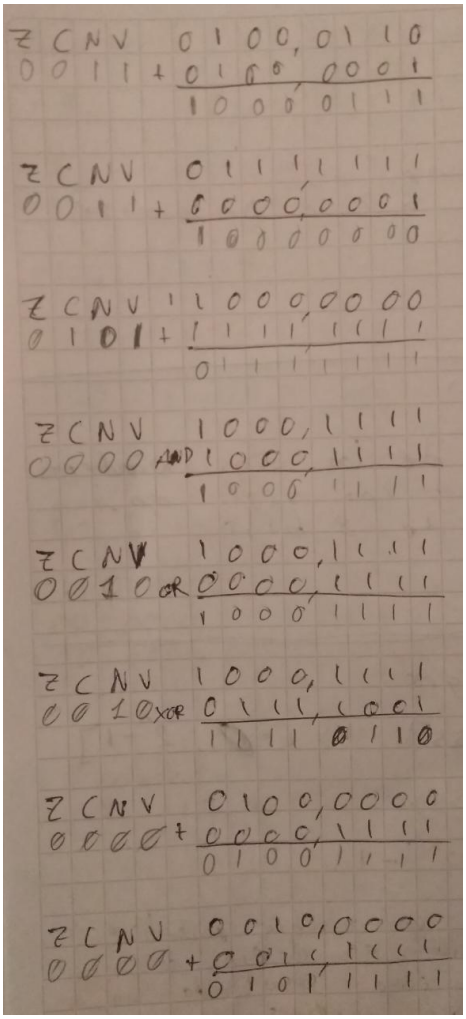

Figura 5: Cambio de banderas debido a las operaciones

Disclaimer: La finalidad de este documento es servir como apoyo de estudio. El autor de la versión original de este documento no se hace responsable del uso indebido del mismo.