Tema 4: Operaciones Aritmeticas y Logicas

HISTORIAL DE REVISIONES			
NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2018 September 12		CA

Índice

1. Temario

1. Aritmética y lógica
    a. Operaciones aritméticas y lógicas sobre enteros en binario
    b. Redondeo y propagación de error en números reales

2. Objetivo

- Operaciones aritméticas de suma y resta con datos representados en código binario.
- Operaciones lógicas de datos representados en código binario.
- Libro de texto
    - Parte 3ª, Capítulo 10 : Aritmética del Computador

3. Introduccion

- La Unidad Aritmetico Lógica (ALU) es la unidad hardware básica encargada de realizar las operaciones de cálculo aritmético como la suma y resta y de realizar operaciones lógicas de tipo booleano como las operaciones not, or, and, etc

3.1. Aritmetica Binaria

3.1.1. Suma-Acarreo

- Suma de datos binarios
    - Interpretación modular $\rightarrow$ módulo 100.000 $\rightarrow$ Interpretación gráfica mediante la circunferencia.Qué ocurre en el cuentakilómetros parcial del coche cuando llegamos a 99.999.
    - Suma en módulo 2. El **acarreo** se produce al llegar o pasar el valor *2*.
    - 10011011+00011011 = 10110110

3.1.2. Overflow-Desbordamiento

- Ocurre cuando:
    - El valor a representar está fuera de rango
    - El resultado de la operación aritmética tiene un tamaño superior al permitido por la palabra de memoria o registro donde se almacena.
- Provoca errores: al inicializar variables de tipo entero, en las operaciones aritméticas suma y resta.
- Ejemplos:
    - Representar el valor 8 en binario de dos bits. Con dos bits el valor máximo es el 3 y las operaciones se realizan en módulo 4
        - 3+1 = 0 $\rightarrow$ suma modular. Representar gráficamente mediante la rueda.
        - 8 = 8 - 4*n = 0 donde n es cualquier entero positivo
    - 10011011+10011011 = 100110110 = Error overflow o desbordamiento= suma modular = 00110110

3.1.3. Resta

- Resta de datos binarios. Resta en módulo 2. Interpretación gráfica.
  - 10110110 - 10011011 = 00011011
  - Suma de datos con signo. Codificación en complemento a 2.
  - 10011011+00011011 = 10110110
    - El primer sumando es negativo y el segundo es positivo, es decir, su suma es una resta.
    - El primer sumando convertido a positivo : Complementar 10011011 sumar 1 = 01100100+1 = 01100101 = 64+32+4+1 = 101
    - El segundo sumando es 00011011 = 16+8+2+1 = 27
    - La operación en decimal es -101+27=-74
    - El resultado convertido a positivo : complemento a 2 = 01001001+1 = 01001010 = 64+8+2 = 74

3.1.4. Overflow C2

- Errores en las operaciones aritméticas suma y resta.
  - 10000000+10000000 = 00000000 = Error Overflow
    - Si los dos sumandos son negativos el resultado no puede ser positivo
    - Si los dos sumandos son positivos el resultado no puede ser negativo
- Intelx86 activa el error de overflow cuando en el resultado de una operación aritmética con signo el acarreo del bit MSB afecta al valor del resultado.

---

nota
Observar que al realizar operaciones aritméticas de suma y resta, el código del resultado es idéntico en números sin signo y en complemento a 2. El código es idéntico pero su valor asociado no lo es.

3.1.5. Suma Hexadecimal

- Suma en módulo 16. El acarreo se produce al llegar o pasar el valor del dígito F.
  - 0xF+0x1 = 0x10
  - 0x3AF+0xA = 0x3B9
  - 0x3A1F+0xF4E1=0x12F00
- El código binario complemento a 2 se puede codificar como hexadecimal
  - -1 en decimal -$\longrightarrow$ 11111111 en C2 -$\longrightarrow$ 0xFF en hexadecimal

3.1.6. Complemento a 2 en hexadecimal

- 0xEC+0xAB=0x97
  - En binario el bit MSB es 1 significa que el valor es negativo
  - Los dos sumandos y el resultado son negativos
  - La suma de dos números negativos da overflow si el resultado es positivo, por lo que no hay overflow
  - C2 de 0xEC $\rightarrow$ 0xEC negado es 0x13 y sumando 1 $\rightarrow$ 0x15
  - C2 de 0xAB $\rightarrow$ 0x54+1 $\rightarrow$ 0x55
  - C2 de 0x97 $\rightarrow$ 0x68+1 $\rightarrow$ 0x69

3.1.7. Suma Octal

- Suma en módulo *8*. El acarreo se produce al llegar o pasar el valor del dígito *8*.
  - 08+01 = 010
  - 0377+06 = 0305

3.2. Tipos de variables en C

- Enteros
  - char
  - short
  - int
  - long
- Reales
  - float
  - double
- Operador sizeof()
- Conversión de tipos
  - casting

4. Operaciones Logicas

4.1. Operadores BITWISE

- Bitwise: operaciones bit a bit
  - not,and,or,xor

4.1.1. Lenguaje C

- https://www.salesforce.com/us/developer/docs/apexcode/Content/langCon_apex_expressions_operators_understanding.htm
- Algebra Boole
- algebra symbols
  - Bitwise operator: and &, or |, xor ^, not ~
  - Shift operator: left <<,right signed >>, right unsigned >>>

Operador	Algebra	C
NOT	¬ ~	~
OR	∨	|
AND	∧	&
XOR	⊕ ⊻	^
NOR	⊽	
NAND	⊼	
Left SHIFT		$x << m$
Right SHIFT signed		$x >> m$
Right SHIFT unsigned		$x >>> m$

4.1.2. Tablas de la Verdad

x	y	z=x∨y	z=x∧y	z=x⊕y
0	**0**	**0**	**0**	**0**
0	**1**	**1**	**0**	**1**
1	**0**	**1**	**0**	**1**
1	**1**	**1**	**1**	**0**

4.1.3. Expresión Lógica

- z=¬x·y+x·¬y
  - Si desarrollamos la tabla de la verdad comprobamos su equivalencia con el operador XOR

5. Multiplicación

- Multiplicación 0xFF x 0x6
  - Realizarla en Binario

- Observar que al multiplicar por una potencia de 2 hay un desplazamiento del multiplicando hacia la dcha
- multiplicar = sumar y desplazar

6. Programación

6.1. funciones matemáticas

- http://bisqwit.iki.fi/story/howto/bitmath/
  - El código fuente está escrito en lenguaje C
- Librería libm.so del standard de C

6.2. Aplicación

- Desarrollar un programa que multiplique números enteros con signo.

7. Hardware

7.1. Circuitos Digitales

7.1.1. Básicos:Puerta lógicas

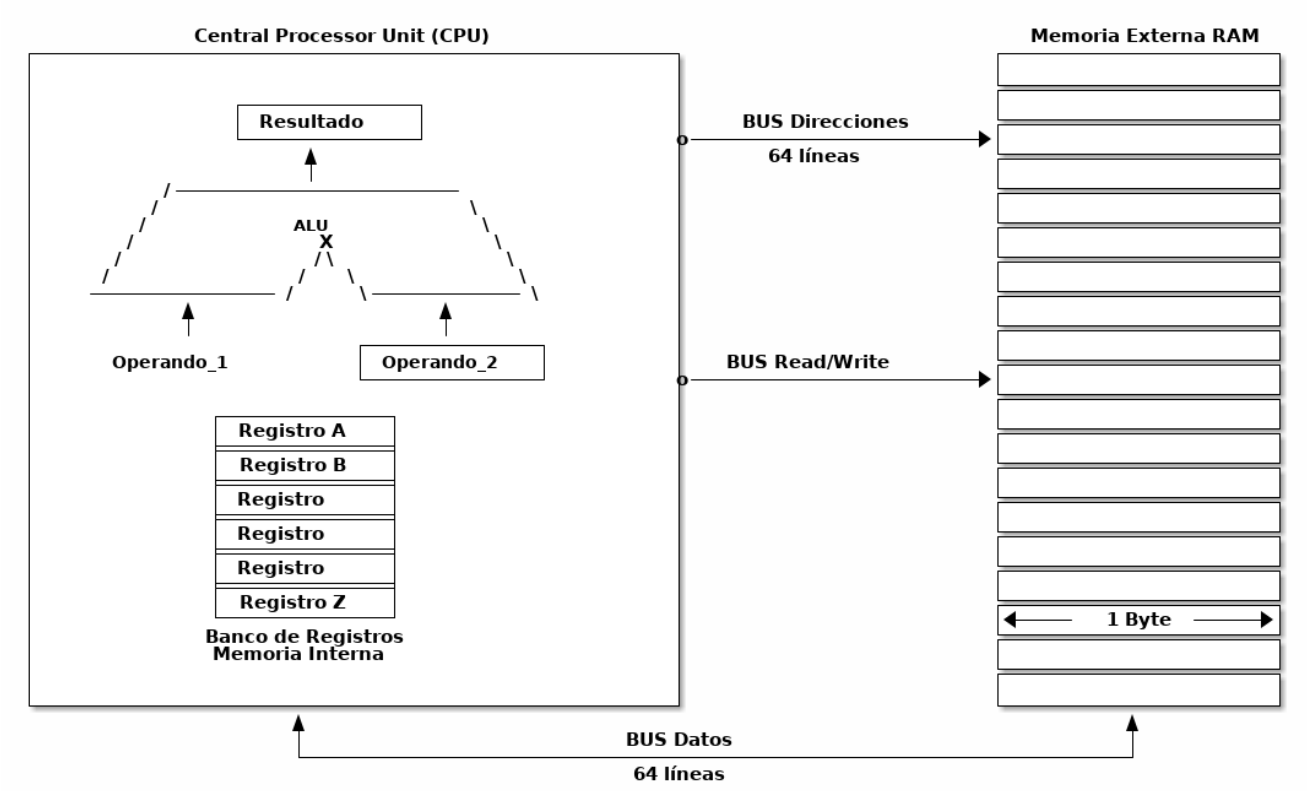
- Puertas lógicas
  - not, and, or, xor

7.1.2. Complejos

- half adder, full adder
- multiplicador
  - circuito combinacional formado por puertas lógicas
  - acumulador y registro desplazador

7.2. Unidad Aritmetico Lógica (ALU)

- Arithmetic logic unit (ALU)
- Circuito Digital
- Conexión CPU-DRAM
  - Transferencia de Instrucciones y Datos
  - La ALU es interna a la CPU y procesa datos numéricos enteros almacenaddos en los registros de propósito general.



7.3. Registro de flags EFLAG

- El registro de flags EFLAGS es un registro de memoria interno a la CPU Intel x86
- Cada bit del registro de 32 bits es un banderín o flag que se activa en función del resultado de la operación realizada por la última instrucción máquina ejecutada.

Cuadro 1: RFLAG Register

Flag	Bit	Name
CF	0	Carry flag
PF	2	Parity flag
AF	4	Adjust flag
ZF	6	Zero flag
SF	7	Sign flag
OF	11	Overflow flag

- Carry flag:
  - se activa si la llevada afecta a una posición de bit mayor que del ancho de palabra (word size) de la ALU en una operación aritmética de números enteros sin signo o con signo
- Overflow flag:
  - se activa si teniendo en cuenta el bit de mayor peso MSB (aunque esté fuera el word size) indicase error en la operación aritmética con números enteros con signo. Si no se tiene en cuenta el MSB fuera del word size, la operación es correcta.

- Parity Even flag:
  - indica si el número de bits del byte LSB del resultado de la última operación ha sido par.
- Sign flag:
  - se activa si el resultado de la última operación ha sido negativo.
- Adjust flag:
  - se activa si hay llevada en el nibble LSB del resultado de la última operación
- Ejemplos

```
  11111111
+ 00000001
**********
 100000000 -> hay acarreo y también overflow ya que si consideramos el MSB=1 de la  ↩
    posición 8ª el resultado sería erróneo, es decir, si realizasemos una    extensión de ↩
     signo el resultado sería erroneo y por lo tanto hay que rechazar el dígito fuera de  ↩
    rango con lo que la operación es correcta. El bit de signo es el de la posición más  ↩
    alta del "word size" (posición 7ª)


 A :    11110000
-B :   +11101100
****************
A-B:   111011100 -> hay acarreo pero no overflow ya que el MSB=1 en una posición fuera del ↩
    "word size" no afecta al signo del resultado, ya que las posiciones del MSB (8ª) y la ↩
    anterior (7ª) del bit de signo tienen el mismo dígito de valor 1


 A :    10000000
-B :   +10000000
****************
A-B:   100000000 -> hay acarreo. Hay también overflow ya que el bit MSB de valor 1 es  ↩
   diferente de la posición anterior ( 7ª, bit de signo) de valor 0.  Los dos sumandos  ↩
   son negativos (bit de signo posición 7ª) y el bit de signo del resultado (bit posición ↩
    7ª) es positivo luego el resultado es erróneo.
```

- Se ve nuevamente en el próximo capítulo Programación en Lenguaje Ensamblador (x86)

7.4. Float Point Unit-FPU

- Unidad de procesamiento de datos en coma flotante
- Antiguamente era una unidad no integrada en la CPU denominada coprocesador matemático
- Utiliza registros específicos denominados SSE distintos de los Registros de Propósito General utilizados por la ALU para realizar operaciones con números enteros.