Organización de Computadoras 2003

Apunte 3: Sistemas de Numeración: Operaciones Lógicas

Para comprender este tema, me parece apropiado que repasen el tema de cálculo proposicional introducido en el curso de ingreso (en este apunte hay una breve introducción al respecto). De esta manera, con el concepto de conectivos lógicos firme, vamos a entender los distintos usos que podemos dar a las operaciones lógicas en Informática. Como siguiente paso voy a describir el concepto básico de operador, para finalizar con las operaciones lógicas que podemos realizar en lenguaje Assembler. El apunte finaliza con una serie de ejercicios prácticos sobre el tema.

Espero que este apunte les sea de utilidad, y si tienen dudas sobre los temas expuestos o quieren profundizar en alguno de ellos, al final se agrega la bibliografía utilizada.

Introducción: "Cálculo Proposicional"

El cálculo proposicional es el estudio de las relaciones lógicas entre objetos llamados proposiciones, que generalmente pueden interpretarse como afirmaciones que tienen algún significado en contextos de la vida real. Para nosotros, una proposición será cualquier frase que sea verdadera o falsa, pero no ambas.

Recordemos del curso de ingreso y de programación de computadoras que en el cálculo proposicional se utilizan letras minúsculas (ej: p, q, r) para simbolizar proposiciones, que se pueden combinar utilizando *conectivos lógicos*:

$\neg$	para "no" o negación
$\wedge$	Para "y"
$\vee$	para "o"
$\rightarrow$	para "entonces" o implicación condicional
$\leftrightarrow$	para "si y sólo si" o la bicondicional

Repasemos con un ejemplo. Proposiciones:

p = "está lloviendo"
q = "el sol está brillando"
r = "hay nubes en el cielo"

simbolizamos las siguientes frases:

Proposición	*Simbolización*
Está lloviendo y el sol está brillando	$p \wedge q$
Si está lloviendo, entonces hay nubes en el cielo	$p \rightarrow r$
Si no está lloviendo, entonces el sol no está brillando y hay nubes en el cielo	$\neg p \rightarrow (\neg q \wedge r)$
El sol está brillando si y sólo si no está lloviendo	$q \leftrightarrow \neg p$

La suposición fundamental del cálculo proposicional consiste en que los valores de verdad de una proposición construida a partir de otras proposiciones quedan completamente determinados por los valores de verdad de las proposiciones originales. Para ello se establecen los valores de verdad según las posibles combinaciones de valores de verdad de las proposiciones originales, basándonos en las siguientes tablas:

Negación ("no"):

p	¬ p
V	F
F	V

Evidentemente, si la proposición p es verdadera, su negación será falsa y viceversa.

Conjunción ("y"):

p	q	p ∧ q
V	V	V
V	F	F
F	V	F
F	F	F

La tabla indica que, el conectivo lógico "y" sólo será verdadero cuando ambas proposiciones p y q sean verdaderas.

Disyunción ("o"):

p	q	p ∨ q
V	V	V
V	F	V
F	V	V
F	F	F

La tabla indica que, si al menos una de las proposiciones es verdadera, la proposición formada por el conectivo "o" será verdadera.

Condicional ("entonces"):

p	q	p → q
V	V	V
V	F	F
F	V	V
F	F	V

Naturalmente, si el antecedente es verdadero y el consecuente es falso, la proposición formada por el conectivo "→" será falsa.

Bicondicional ("si y sólo si"):

p	q	p ↔ q
V	V	V
V	F	F
F	V	F
F	F	V

El bicondicional establece que, para que la proposición formada por el conectivo "↔" sea verdadera ambas proposiciones deben tener el mismo valor de verdad.

Observaciones importantes:

- Nótese que, el operador "¬" es unario (se utiliza con una única proposición), mientras que los demás operadores son binarios (utilizan dos proposiciones).
- Otra observación importante es que los operadores condicionales "→" y bicondicionales "↔" en realidad no son necesarios, ya que pueden ser reemplazados por combinaciones de "¬", "∧" y "∨ " a través de las siguientes equivalencias lógicas:

p → q	es equivalente a	(¬ p) ∨ q
p ↔ q	es equivalente a	(p → q) ∧ (q → p)

(donde los condicionales pueden reemplazarse por la equivalencia anterior)

Relacionemos este tema con la Informática:

La lógica utilizada en informática a bajo nivel admite sólo dos estados para cada unidad mínima de información: (1,0), (on, off) o (Verdadero, Falso). Podemos reemplazar los valores V y F de las tablas anteriores por los valores 1 y 0 respectivamente, para formar el álgebra booleana, consistente de dos operaciones binarias "∧" y "∨ ", y una operación unaria "¬".

De esta manera, reconstruimos las tablas de los conectivos lógicos considerando que las operaciones se efectúan a nivel de bits:

b1	b2	b1 AND b2
1	1	1
1	0	0
0	1	0
0	0	0

b1	b2	b1 OR b2
1	1	1
1	0	1
0	1	1
0	0	0

b1	NOT b1
1	0
0	1

Se puede observar que el conectivo lógico OR es inclusivo (la operación retorna 1 donde al menos uno de los operandos sea 1). También resulta de utilidad el conectivo lógico OR exclusivo (XOR), donde la operación retorna 1 en caso que uno de los operandos sea 1, pero no ambos. A continuación se observa la tabla del XOR:

b1	b2	b1 XOR b2
1	1	0
1	0	1
0	1	1
0	0	0

Operaciones comunes en lenguaje Assembler

Podemos categorizar las operaciones más comunes que realiza la unidad Aritmético/Lógica (ALU) en la siguiente tabla:

Desplazamientos:	Lógicos
	Circulares
	Aritméticos
Lógicas:	NOT
	AND
	OR
	XOR
Aritméticas:	Negación
	Suma
	Resta
	Multiplicación
	División

En este apunte nos centramos en las operaciones lógicas, las que por su propia naturaleza son a nivel de bit. Aunque se pueden realizar en forma paralela, no existe ninguna interacción entre bits de posiciones diferentes. Las operaciones lógicas sobre dos secuencias de bits realizan la operación sobre cada par de bits de igual posición entre ambas secuencias, de manera independiente.

Detallemos cómo funcionan las operaciones lógicas con grupos de bits mediante ejemplos:

	1101
AND	
	1011
	1001

	1001
OR	
	1011
	1011

	1101
XOR	
	1011
	0110

NOT 1011
0100

Observación: Estos ejemplos consideran para el AND, OR y XOR dos secuencias de cuatro bits c/u.

Se puede notar que la operación NOT equivale al complemento lógico (complemento a 1 -Ca1).

Implementación de máscaras

Dada una secuencia de bits, a veces resulta útil "jugar" con ciertas posiciones (ej: forzar los bits de posiciones impares a 1, o averiguar el estado del 3er bit en la secuencia, o invertir los valores de ciertas posiciones dejando intactas las demás, etc). Para esta tarea se pueden utilizar operaciones lógicas adecuadas que reciban como primer secuencia de bits la secuencia dada y, como segunda secuencia de bits, una secuencia predeterminada, denominada "máscara", que servirá como segundo operando para obtener el resultado deseado.

Tengamos en cuenta lo siguiente para definir el operador adecuado y la máscara:

- De la tabla del AND se deduce que:
  - Al utilizar un operando con valor 1, el resultado de la operación coincidirá con el valor del otro operando (1 AND 1 = 1, 1 AND 0 = 0). Genéricamente: 1 AND X = X.
  - Al utilizar un operando con valor 0, el resultado de la operación será 0 independientemente del valor del otro operando. (0 AND 0 = 0, 0 AND 1 =0). Genéricamente: 0 AND X = 0.
- De la tabla del OR se deduce que:
  - Al utilizar un operando con valor 1, el resultado de la operación será 1 independientemente del valor del otro operando (1 OR 0 = 1, 1 OR 1 = 1). Genéricamente: 1 OR X = 1.
  - En caso de realizar un OR entre 0 y cualquier operando, el resultado coincidirá con el valor de ese operando (0 OR 0 = 0, 0 OR 1 = 1). Genéricamente 0 OR X = X.
- De la tabla del XOR se deduce que:
  - Al utilizar como primer operando un 0, el resultado coincidirá con el valor del segundo operando (0 XOR 0 = 0 , 0 XOR 1 = 1). Genéricamente: 0 XOR X = X.
  - En caso de ralizar un XOR entre 1 y cualquier operando, el resultado será el opuesto del operando (1 XOR 0 = 1, 1 XOR 1 = 0). Genéricamente: 1 XOR X = NOT X.

A modo de ejemplo, se plantean los siguientes ejercicios:

Ej. 1) Dada una secuencia de 4 bits, forzar el 1er bit a cero, dejando el resto sin modificar.

Solución: Usamos el operador AND y como máscara definimos una que tenga el valor 0 en la primer posición (dado que 0 AND X = 0) y completamos la máscara con 1 (dado que 1 AND X = X).

```
        b3 b2 b1 b0
AND
      1  1  1  0     (máscara = 1110)
-------------------
        b3 b2 b1 0
```

Ej. 2) Dada una secuencia de 4 bits, invertir el valor de las posiciones impares, dejando el resto sin modificar.

Solución: Usamos el operador XOR y como máscara definimos una que tenga el valor 1 en las posiciones que queremos invertir (1 XOR X = NOT X) y un 0 en las posiciones que deben permanecer iguales (0 XOR X = X).

```
        b3 b2 b1 b0
XOR
      1  0  1  0     (máscara = 1010)
-------------------
      ¬b3 b2 ¬b1 b0
```

Ej. 3) Dada una secuencia de 4 bits, forzar a uno el segundo bit, dejando el resto sin modificar.

Solución: Usamos el operador OR y como máscara definimos una que tenga el valor 1 en la segunda posición (1 OR X = 1) y un 0 en las posiciones que deben permanecer iguales (0 OR X = X).

```
        b3 b2 b1 b0
OR
      0  0  1  0    (máscara = 0010)
      ---------
        b3 b2 1 b0
```

En ciertos casos, puede ser necesario pasar la secuencia de bits de entrada por mas de una máscara y operador:

Ej. 4) Dada una secuencia de 4 bits, invertir el valor de las posiciones impares y forzar a uno las posiciones pares.

Solución: En el 2do ejemplo observamos que el operador XOR resulta adecuado para invertir dígitos binarios utilizando como máscara un 1 en las posiciones que deseamos invertir y un 0 en las posiciones que deseamos queden intactas. Luego realizamos un OR sobre el resultado parcial para forzar a 1 las posiciones pares, utilizando así una segunda máscara con un 1 en las posiciones que deseamos forzar a 1 y un 0 en las posiciones que deseamos se mantengan intactas:

```
          b3 b2 b1 b0  (secuencia de entrada)
XOR
        1  0  1  0     (1er máscara = 1010)
      ---------------
      ¬b3 b2 ¬b1 b0    (resultado parcial, se invirtieron las posiciones impares)
OR
        0  1  0  1     (2da máscara = 0101)
      ---------------
      ¬b3 1  ¬b1 1     (resultado final)
```

Bibliografía consultada para elaborar este apunte

- "Matemáticas Discretas", Kenneth A. Ross, Charles R.B. Wright. Ed. Prentice Hall.
- "Arquitectura de Computadores", Pedro de Miguel, José Angulo. Ed. Paraninfo.
- "Fundamentos de los Computadores", Pedro de Miguel Anasagasti. Ed. Paraninfo.
- "Programmer's Technical Reference: The Processor and Coprocessor", Robert Hummel. Ed. ZD Press.

Ejercicios prácticos

Ej. 1) ¿Cuál es el resultado de las siguientes operaciones?.

1101 AND 0111 = ...
0101 OR 1001 = ...
NOT 0100 = ...
1011 XOR 1110 = ...
(((1010 AND 1100) OR 0101) XOR 1100) = ...

Ej. 2) Completar los bits *X*. Aclaración: puede haber más de una combinación posible.

1001 AND *XXXX* = 1000
0110 OR *XXXX* = 1110
1010 XOR *XXXX* = 1010
NOT *XXXX* = 0110

Ej. 3) ¿Tienen solución las siguientes operaciones?. ¿Por qué?.

1011 OR *XXXX* = 0001
0101 AND *XXXX* = 1100

0011 XOR *XXXX* = 0000

Ej. 4) Dada una secuencia de 4 bits (b_3 b_2 b_1 b_0), encuentre las máscaras apropiadas para:

- Poner en 0 el bit mas significativo, dejando el resto sin modificar.
- Poner en 1 las posiciones impares, dejando el resto sin modificar.
- Invertir todos los bits.
- Invertir las posiciones pares, dejando el resto sin modificar.
- Poner en 1 el bit b_0 y en 0 el bit b_3,dejando el resto sin modificar.
- Invertir las posiciones impares y forzar a 0 las posiciones pares.

Ej. 5) Complete con el operador adecuado (AND, OR, XOR, NOT) las siguientes operaciones:

1000 ..... 1011 = 1000
0110 ..... 1000 = 1110
1101 ..... 1001 = 0100
1111 ..... 0011 = 1100