

Guía de ejercicios # 9: Máscaras

Organización de Computadoras 2017

UNQ

1 Operaciones lógicas bit a bit

En esta práctica veremos nuevos usos de las operaciones lógicas cuando son aplicadas a un par de cadenas binarias en lugar de a un par de valores de verdad. La idea es poder resolver una operación como la que sigue:

$$\text{AND } \begin{array}{r} 0101 \\ 0011 \\ \hline ????$$

Entonces, sabiendo que una operación lógica (AND, OR) está definida para dos valores de verdad, se replica esta idea para cada columna de manera aislada (bit a bit), resultado en 4 operaciones AND separadas:

$$\text{AND } \begin{array}{r} 0101 \\ 0011 \\ \hline 0--- \end{array} \Rightarrow \text{AND } \begin{array}{r} 0101 \\ 0011 \\ \hline 00-- \end{array} \Rightarrow \text{AND } \begin{array}{r} 0101 \\ 0011 \\ \hline 000- \end{array} \Rightarrow \text{AND } \begin{array}{r} 0101 \\ 0011 \\ \hline 0001 \end{array}$$

Similarmente, desarrollemos un ejemplo para OR:

$$\text{OR } \begin{array}{r} 0101 \\ 0011 \\ \hline 0--- \end{array} \Rightarrow \text{OR } \begin{array}{r} 0101 \\ 0011 \\ \hline 01-- \end{array} \Rightarrow \text{OR } \begin{array}{r} 0101 \\ 0011 \\ \hline 011- \end{array} \Rightarrow \text{OR } \begin{array}{r} 0101 \\ 0011 \\ \hline 0111 \end{array}$$

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

(a)
$$\text{AND } \begin{array}{r} 1101 \\ 0111 \\ \hline ?$$

(b)
$$\text{OR } \begin{array}{r} 0101 \\ 1001 \\ \hline ?$$

(c)
$$\text{NOT } \begin{array}{r} 0100 \\ \hline ?$$

(d)
$$\text{XOR } \begin{array}{r} 1011 \\ 1110 \\ \hline ?$$

(e)
$$\text{AND } \begin{array}{r} 1010 \\ 1100 \\ \hline ?$$

(e)
$$\text{OR } \begin{array}{r} 0101 \\ \hline ?$$

(e)
$$\text{XOR } \begin{array}{r} 1100 \\ \hline ?$$

1.b Complete con el operador adecuado (AND, OR, XOR, NOT) en las siguientes operaciones

- (a) $1000 \dots 1011 = 1011$
- (b) $1011 \dots 1000 = 1000$
- (c) $1101 \dots 1001 = 0100$
- (d) $1111 \dots 0011 = 0011$
- (e) $\dots 0011 = 1100$

1.c Dado una cadena de bits formada por $(x_7x_6x_5x_4x_3x_2x_1x_0)$ ¿qué resultado se obtiene al aplicarle la siguiente operación lógica? Ver el primer ejercicio a modo de ejemplo.

(a)
$$\text{OR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10101010 \\ \hline 1x_61x_41x_21x_0 \end{array}$$

(b)
$$\text{OR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 11111000 \\ \hline ? \end{array}$$

(c)
$$\text{AND } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10101010 \\ \hline ? \end{array}$$

(d)
$$\text{AND } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10001111 \\ \hline ? \end{array}$$

(e)
$$\text{XOR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10101010 \\ \hline ? \end{array}$$

(f)
$$\text{XOR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 00001111 \\ \hline ? \end{array}$$

(g)
$$\text{OR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10000000 \\ \hline ? \end{array}$$

(h)
$$\text{OR } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10000000 \\ \hline ? \end{array}$$

(h)
$$\text{AND } \begin{array}{r} 11110000 \\ \hline ? \end{array}$$

(h)
$$\text{XOR } \begin{array}{r} 00011110 \\ \hline ? \end{array}$$

(i)
$$\text{AND } \begin{array}{r} x_7x_6x_5x_4x_3x_2x_1x_0 \\ 10101111 \\ \hline ? \end{array}$$

(i)
$$\text{OR } \begin{array}{r} 11110000 \\ \hline ? \end{array}$$

(i)
$$\text{XOR } \begin{array}{r} 00011110 \\ \hline ? \end{array}$$

(j) X XOR 10101010, al resultado AND 11110000, y al resultado OR 00011110

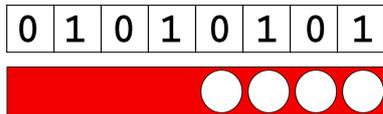
2 Máscaras

A menudo los datos binarios que manejan los programas no son un valor completo (como un número), sino un conjunto de valores, donde cada bit representa un valor (o estado). En estos casos, es útil poder manejar cada bit de manera independiente, por ejemplo para:

- forzar valores en determinadas posiciones
- invertir ciertas posiciones dejando el resto intacto
- conocer el valor de un determinado bit

Para estas tareas se pueden combinar operaciones lógicas adecuadas que reciban como primer operando los bits la secuencia dada y, como segundo operando una secuencia predeterminada, la cual llamaremos “máscara”, y servirá para obtener el resultado deseado.

Gráficamente, lo que sigue es una cadena a analizar. En particular nos interesa rescatar los 4 bits de la derecha. Lo siguiente es una máscara, “anula” las primeras 4 posiciones y deja ver las últimas 4. Al “superponer” la máscara sobre la



cadena se obtiene lo siguiente:



Para implementar las mencionadas máscaras, es necesario agregarle a nuestra arquitectura las instrucciones correspondientes a las operaciones lógicas.

Por un lado, las instrucciones AND y OR:

Operación	Cod Op	Efecto
AND	0100	Dest ← Dest ∧ Origen
OR	0101	Dest ← Dest ∨ Origen

Nota: Estas instrucciones calculan los flags Z y N. El formato de estas dos nuevas instrucciones es ya conocido:

CodOp (4b)	Modo Destino (6b)	Modo Origen (6b)	Destino (16b)	Origen (16b)
---------------	----------------------	---------------------	------------------	-----------------

Por otro lado, la operación NOT es la primera que tiene **un sólo operando destino**. Notar que ya se contaba con instrucciones de un operando (CALL, JMP) pero eran el operando origen.

Operación	Cod Op	Efecto
NOT	1001	Dest ← ¬Dest

El formato de instrucción correspondiente al NOT:

CodOp (4b)	Modo Origen (6b)	Relleno (000000)	Operando Origen (16b)
---------------	---------------------	---------------------	--------------------------

Veamos un ejemplo. Se tiene un valor en R6 y el objetivo es poner el valor 1 en sus primeros 4 bits, dejando el resto sin modificar, se busca algo como:

$$\text{op? } \frac{x_{15}x_{14}x_{13}x_{12}x_{11}x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0}{\text{mascara?}} \\ \frac{1\ 1\ 1\ 1\ x_{11}x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0}{}$$

Para esto, es posible usar una operación or con una máscara con que fije valores 1 en la primera parte y deje ver los valores de la segunda parte:

$$\text{or } \frac{x_{15}x_{14}x_{13}x_{12}x_{11}x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0}{1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0} \\ \frac{1\ 1\ 1\ 1\ x_{11}x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0}{}$$

Esto, en el contexto de un programa, se escribe:

```
OR R6, 0xF000
```

Ejercicios

En los siguientes ejercicios tendrás que pensar que cadena usar como máscara y qué operación lógica aplicar. **NO OLVIDES DOCUMENTAR CADA PROGRAMA**

- Copiar el byte más significativo de la celda 0348 en el registro R1.
- Escribir un programa que invierta los bits de las posiciones pares de una cadena almacenada en R2.
- Determinar si la cadena en R0 es impar, en cuyo caso invocar a la rutina esImpar.
- Si la celda [CCCC] contiene un número par, sumar 30 al valor de R3. En caso contrario sumar 70 al valor de R4.
- Escribí la rutina xor en función de su documentación:


```
;REQUIERE dos valores de 16 bits en R6 y R7
;MODIFICA el registro R4
;RETORNA en R7 el OR exclusivo (bit a bit) entre
R6 y R7
```
- Escribí la rutina absolute que dada una cadena de 16 bits en R0, ponga un 0 en el primer bit, dejando el resto sin modificar. Utilice máscaras y operaciones lógicas.
- Escribí la rutina invimp que dada una cadena de 16 bits en R1, invierta el valor de las **posiciones impares**. Utilice máscaras y operaciones lógicas.
- Escribí la rutina segbit que dada una cadena de 16 bits en R1, ponga un 1 en el segundo bit, dejando el resto sin modificar. Utilice máscaras y operaciones lógicas.
- Escribí la rutina opuesto, que calcule el opuesto aditivo del número almacenado en el registro R2. Dicho número está representado en CA2(16).
- Escribí la rutina equals que determine si el contenido de R2 y R3 son iguales, usando operaciones lógicas
- Escribir la rutina desplazarIzq que desplace los bits de la cadena contenida en R0 un lugar hacia la izquierda, guardando el resultado en R1.

2.l Escribí una rutina `desplazarDer` que desplace los bits de la cadena contenida en R0 un lugar hacia la izquierda, guardando el resultado en R1.

```
MOV R0, 0xFFFF
JMP fin
funcionabien: MOV R0, 0xF000
fin: RET
```

2.m La siguiente es la documentación de la rutina `espar`:

```
;REQUIERE una cadena de 16 bits en el registro R6
;MODIFICA el registro R5
;RETORNA un 1 en registro R1 si R6 es par.
          un 0 en caso contrario
```

Escribí un programa que, **usando la rutina anterior**, sume 30 al valor de R3 si la celda [CCCC] contiene un número par. En caso contrario debe sumar 70 al valor de R4

2.n Escribí un programa que si las celdas CCCC y CCD containen números impares, le reste 0x0001 a ambas. Utilice la rutina `espar`.

2.o La siguiente es la codificación de permisos de acceso sobre archivos en un sistema Linux

- Con 3 bits se indica:
 - ¿puedo leer? (r)
 - ¿puedo escribir? (w)
 - ¿puedo ejecutar? (x)
- Con 3 cadenas se describen permisos de usuario, grupo y otros

Por ejemplo, la cadena 11111111 le da todos los permisos a todos. Escriba la rutina `groupWriting` que, dada una cadena que codifica los permisos de acceso a determinado archivo, determine si otro usuario del grupo lo puede escribir.

Dicha cadena esta almacenada en R4, y el programa debe poner un 1 en R5 si es posible, y 0 en caso contrario.

3 Controlar los programas

Para probar las rutinas de la sección anterior, escribiremos programas de *test* o prueba.

En cada caso se deben proveer valores que cumplan lo requerido por la rutina (campo **requiere**) y luego de invocarla se debe comparar el resultado con el valor esperado, poniendo en R0 el valor F000 en caso de éxito o el valor FFFF en caso de fallo. Por ejemplo, considerar la documentación de la rutina `avg`:

```
;REQUIERE dos valores de 16 bits en R6 y R7
;MODIFICA --
;RETORNA en R6 el promedio ENTERO entre R6 y R7
```

Entonces para probar lo anterior se escribe un programa de test como el que sigue:

```
testAvg: MOV R6, 0x0008
          MOV R7, 0x000A
          CALL avg
          CMP R6, 0x0009
          JE funcionabien
```

- Escribí un programa que haga un control de calidad sobre la rutina `xor`
- Escribí un programa que haga un control de calidad sobre la rutina `absolute`
- Escribí un programa que haga un control de calidad sobre la rutina `invimp`
- Escribí un programa que haga un control de calidad sobre la rutina `segbit`
- Escribí un programa que haga un control de calidad sobre la rutina `opuesto`
- Escribí un programa que haga un control de calidad sobre la rutina `equals`
- Escribí un programa que haga un control de calidad sobre la rutina `desplazarDer`
- Escribí un programa que haga un control de calidad sobre la rutina `desplazarIzq`
- Escribí un programa que haga un control de calidad sobre la rutina `esPar`
- Escribí un programa que haga un control de calidad sobre la rutina `groupWriting`

4 Ejecución de instrucciones

Para cada instrucción de las que se mencionan abajo, se debe:

- Ensamblar la instrucción
- Completar un cuadro de uso de buses como el que sigue, donde se indica la información que viaja por los buses en cada etapa del ciclo de ejecución.

Etapa	Bus de control	Bus de dir.	Bus de datos

- La columna **etapa** debe indicar si es búsqueda de instrucción (B.I), de Operandos (B.O.) o almacenamiento de resultados (A.R)
- En la columna de **bus de control** se debe indicar el estado de las líneas L (Lectura) y E (Escritura).
- En la columna del **bus de direcciones** se debe indicar la dirección de la celda accedida.
- En la columna del **bus de datos** se debe indicar el contenido que se quiere escribir en, o que se leyó de, la celda accedida.

Nota: Asumí que cada instrucción está ensamblada a partir de la celda 0xA000.

- 4.a JLEU salto, sabiendo que salto es la etiqueta de una instrucción en la celda 0xA077
- 4.b AND R5, R7, donde R5=0x5555 y R7=0xA5A5
- 4.c NOT [0x0076], donde la celda 0x0076 tiene el valor 0x5555
- 4.d OR [0x0076], [0xA890], donde la celda 0x0076 tiene el valor 0x5555 y la celda 0xA890 tiene el valor 0BBBB
- 4.e JG salto, sabiendo que salto es la etiqueta de una instrucción en la celda 0x7700
- 4.f JMP [R5], sabiendo que R5=0x2121
- 4.g JMP [0x2121], sabiendo que la celda 0x2121 tiene el valor 0xAAAA