Universidad Autónoma de Baja California
Ingeniero en computación

Organización y Arquitectura De Computadoras

Practica 8

Nombre Del Alumno: López Mercado Brayan

Matrícula: 1280838

Grupo: 551

Docente: José Isabel García Rocha

Fecha de entrega: 26 de octubre del 2023

Objetivo

Seleccionar las instrucciones lógicas y de manipulación de bits correctas en aplicaciones de sistemas basados en microprocesador mediante la distinción de su funcionamiento, de forma lógica y responsable.

Desarrollo

1- ***Cree un programa llamado P8.asm que contenga la subrutina printHex de la practica 5, la cual recibe en EAX un valor que se quiere imprimir en formato hexadecimal. Agregue a P8.asm las siguientes subrutinas:***

a) ***Crear la subrutina printBin con corrimientos, utilizando la instrucción loop y la bandera de acarreo, no utilizar otros saltos, debe funcionar con valores de 32 bits, en EAX se recibe el valor a imprimir en binario y ESI recibe la dirección de la cadena donde se guardará la conversión.***

```
printBin:
    pushad
    mov edi,eax
    mov ecx,32
    .cycle:
        xor al,al
        shl edi,1
        adc al,'0'
        call putchar
        mov byte[esi+ecx],al
        loop .cycle
    popad
    ret
```

Figura 1: Subrutina printBin.

Descripción: la subrutina printBin funciona de la siguiente manera, se realiza una copia en EDI del registro EAX (contiene el número a imprimir), esto se realiza con el objetivo de no realizar modificaciones erróneas al número a imprimir; ECX contiene la cantidad de bits que se mostraran en la terminal, en la etiqueta cycle es donde se realiza la conversión, primero se “limpia” el registro AL, para posteriormente realizar un corrimiento a la derecha, de esta forma se mueve a la bandera de acarreo el bit menos significativo para luego realizar una suma con acarreo de AL con “cero”, esto puede carecer de sentido al principio, la forma en la que funciona esto es que si la bandera está activa se sumara 1 a cero, es decir, se tendría en AL el bit de la bandera de acarreo, luego se llama a la subrutina putchar para desplegar el digito obtenido para posteriormente guardar el digito en la dirección almacenada por ESI con su respectivo offset, el cual está dado por ECX.

b) ***Crear la subrutina revBit, con la cual se puede conocer el estado de un bit, utilizando corrimientos, la bandera de acarreo, sin utilizar saltos, sin utilizar loop, debe de mostrar en pantalla un '0' si es cero y si es un '1', debe de funcionar con 32 bits, en EAX se le pasa el dato y en CL se le indica que bit se modificara.***

```
revBit:
    pushad
    mov ebx,eax
    shr ebx,cl
    lahf
    and ah,0x1
    movzx eax,ah
    call printBin
    call new_line
    popad
    ret
```

Figura 2: Subrutina revBit.

Descripción: primero se realiza una copia de EAX en EBX, de manera que no se altere el número al cual se quiere conocer el estado de bit seleccionado, la forma en la que se selecciona el bit es por medio de un corrimiento a la derecha dado por CL lo que moverá el bit seleccionado a la bandera de acarreo, para obtener el bit en la bandera de acarreo se hace uso de la instrucción LAHF, y la forma de extraer únicamente la bandera de acarreo es por medio de una máscara, la máscara en cuestión en 0x1, esto se debe a que principalmente la bandera de acarreo es el bit menos significativo del registro de banderas, la extracción se realiza con la operación AND AH,0x1, de esta forma AH solo contendrá el bit de la bandera de acarreo, para desplegarlo en terminal se hace uso de MOVZX para extender con ceros el registro AH y únicamente se muestre el valor del bit en su versión de 32 bits, el cual es desplegado por la subrutina printBin.

c) ***Crear la subrutina setBit: activa un bit del registro EAX, el número de bit a activar está dado por CL.***

```
setBit:
    pushad
    ror eax,cl
    or eax,0x1
    rol eax,cl
    call printBin
    call new_line
    popad
    ret
```

Figura 3: Subrutina setBit.

Descripción: se realiza una rotación hacia la derecha del número a modificar con CL siendo la cantidad bits a rotar, de esta forma se podrá modificar con una máscara fija, la cual es 0x1, luego de aplicar la operación OR a EAX con la máscara, se habrá activado el bit en cuestión, para devolver el bit a su

posición inicial, se realiza un corrimiento a la izquierda dado por CL y luego se imprime el resultado por medio de printBin.

d) Crear la subrutina clearBit: desactiva un bit del registro EAX, el número de bits a desactivar está dado por CL.

```
clearBit:
    pushad
    ror eax,cl
    and eax,0xFFFFFFFE
    rol eax,cl
    call printBin
    call new_line
    popad
    ret
```

Figura 4: Subrutina clearBit.

Descripción: se realiza una rotación hacia la derecha del número a modificar con CL siendo la cantidad bits a rotar, de esta forma se podrá modificar con una máscara fija, la cual es 0xFFFFFFFE, el motivo por que se eligió esta mascara en particular es el hecho de que solo se desea desactivar el bit menos significativo y esto último se logra colocando un cero en el bit menos significativo de la máscara, luego de aplicar la operación AND a EAX con la máscara, se habrá desactivado el bit en cuestión, para devolver el bit a su posición original, se realiza un corrimiento a la izquierda dado por CL y luego se imprime el resultado por medio de printBin.

e) Crear la subrutina notBit: invierte un bit del registro EAX, el número de bits a invertir está dado por CL.

```
notBit:
    pushad
    ror eax,cl
    xor eax,0x1
    rol eax,cl
    call printBin
    call new_line
    popad
    ret
```

Figura 5: Subrutina notBit.

Descripción: se realiza una rotación hacia la derecha del número a modificar con CL siendo la cantidad bits a rotar, de esta forma se podrá modificar con una máscara fija, la cual es 0x1, luego de aplicar la operación XOR a EAX con la máscara, se habrá invertido el bit en cuestión, para devolver el bit a su posición inicial, se realiza un corrimiento a la izquierda dado por CL y luego se imprime el resultado por medio de printBin.

f) Crear la subrutina testBit: copia un registro del registro EAX a la bandera de cero; el bit a copiar está dado por CL, se debe de imprimir en pantalla un cero si la bandera de cero está encendida o un 1 si la bandera de cero está apagada, para esta acción no se deben de utilizar saltos, test o cmp.

```
testBit:
    pushad
    mov ebx,eax
    ror ebx,cl
    and ebx,0x1
    mov ah,bl
    rol ah,6
    sahf
    movzx eax,ah
    shr eax,6
    call printBin
    call new_line
    popad
    ret
```

Figura 6: Subrutina testBit.

Descripción: Primero se realiza una copia del registro EAX a EBX, de esta manera no se modifica el número del cual se quiere copiar el bit a la bandera de acarreo, luego se realiza una rotación a la derecha el cual se encuentra dado por CL, de esa forma se puede colocar el bit deseado en la posición menos significativa y realizar la operación AND EBX,0x1 para solo dejar activo el bit a mover a la bandera de cero el cual se encuentra el registro BL, para luego realizar una copia al registro AH y ese resultado moverlo al byte menos significativo del registro de banderas por medio de la instrucción SAHF, para desplegarlo en terminal se utiliza la instrucción MOVZX, de esta forma se puede lograr que EAX, contenga una versión extendida con ceros de AH y se pueda desplegar en terminal por medio de la subrutina printBin, un detalle a tener en cuenta es que AH, guarda el bit en la posición que representa a la bandera de cero, la forma de posicionarlo como el LSB es por medio de un corrimiento a la derecha, de esta forma al llamar a la subrutina printBin se desplegara un 1 o cero dependiendo del estado de la bandera de cero.

Pruebas

Formato De Salida En Terminal

- **Etiqueta**
- **Numero en formato hexadecimal**
- **Numero original en binario**
- **Numero modificado en binario**

Prueba 1:

EAX=FFFF000A

CL=24

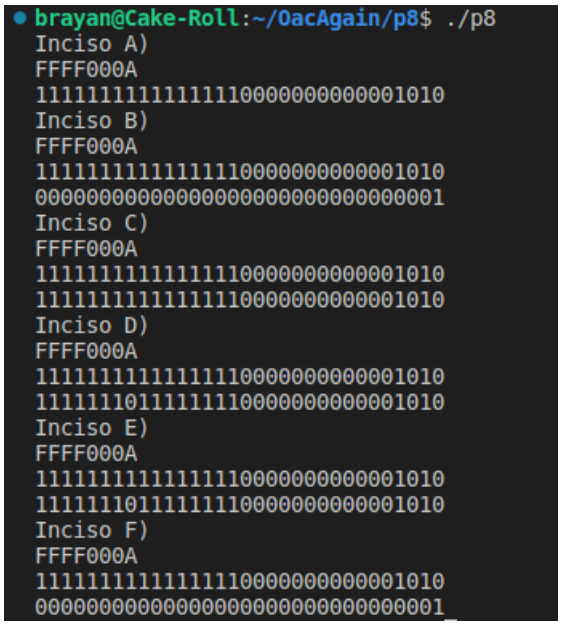

Figura 7: Salida en terminal para la prueba 1.

Prueba 2:
EAX=0x2AEFFF43
CL=5
CL_d=22

```
brayan@Cake-Roll:~/OacAgain/p8$ ./p8
Inciso A)
2AEFFF43
00101010111011111111111101000011
Inciso B)
2AEFFF43
00101010111011111111111101000011
00000000000000000000000000000000
Inciso C)
2AEFFF43
00101010111011111111111101000011
00101010111011111111111101100011
Inciso D)
2AEFFF43
00101010111011111111111101000011
00101010111011111111111101000011
Inciso E)
2AEFFF43
00101010111011111111111101000011
00101010101011111111111101000011
Inciso F)
2AEFFF43
00101010111011111111111101000011
00000000000000000000000000000000
```

Figura 8: Salida en terminal para la prueba 2.

Conclusiones y Comentarios

- Las rotaciones tienen su utilidad cuando se quiere modificar algún bit en específico borrar los demás bits que se encuentren a su alrededor a comparación de un corrimiento, los cuales se pueden utilizar cuando solo se quieran conocer el valor de ciertos bits sin importar los demás.
- La forma en la que se resolvieron algunos de los problemas son un tanto “extrañas” debido al manejo de los bits.
- Esta práctica es igual a la del semestre pasado, los traumas revivieron.

Link a Repositorio De GitHub Con Código Completo

https://github.com/BrayanLMercado/OAC_Practica8_v2.git

Dificultades En El Desarrollo

- Las instrucciones de las practicas siguen siendo igual de abstractas, dejando eso de lado, en el último inciso se sintió un tanto confuso respecto a cómo indicar que la bandera de cero estaba activa o inactivaba.
- La máscara de la subrutina clearBit fue la más confusa de realizar, al para resolver el problema se decidió poner la máscara de 32 bits completa en lugar de un solo bit como las demás.

Referencias