Universidad Autónoma de Baja California
Ingeniero en computación

Organización y Arquitectura De Computadoras

Practica 12

Nombre Del Alumno: López Mercado Brayan

Matrícula: 1280838

Grupo: 551

Docente: José Isabel García Rocha

Fecha de entrega: 30 de noviembre del 2023

Objetivo

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

Desarrollo

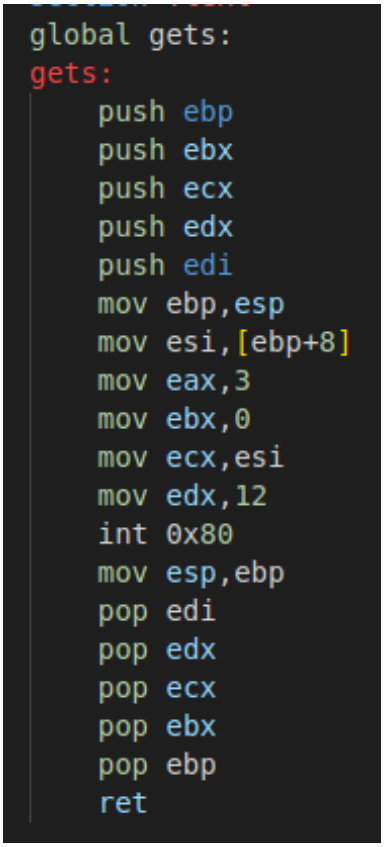
1- Cree un programa llamado p12_enC.c que realice los siguiente:
Crear un programa en el cual el usuario ingrese una cadena de caracteres numéricos terminados con el símbolo *, posteriormente se llame a la subrutina puts para mostrar la cadena en pantalla, luego se llame a la subrutina Atoi para convertir la cadena a un número decimal, para finalmente llamar a la subrutina printHex para mostrar el valor contenido en EAX y posteriormente se muestra en pantalla con la subrutina printDec.

```c
# include <stdio.h>
void writeTag();
extern int* gets(int* cad);
extern unsigned int atoi(int* cadena);
extern int printDec(unsigned int val);
extern int printHex(unsigned int val);
int main(){
    int cad;
    int *pcad=&cad;
    for(int i=0;i<=2;i++){
        writeTag();
        int* cadena=gets(pcad);
        int num= atoi(cadena);
        printf("Salida En Terminal De PrintHex:\n");
        printHex(num);
        printf("\nSalida En Terminal De PrintDec:\n");
        printDec(num);
        printf("\n");
        printf("\n");
    }
    return 0;
}
void writeTag(){
    printf("Ingresa La Cadena:");
    printf("\n");
}
```

Figura 0: Código en C Utilizado.

El programa debe de contener las siguientes subrutinas en un archivo llamado p12_enASM.asm:

- Gets: Subrutina que captura en memoria direccionada por ESI, una cadena de caracteres y esta termina de almacenarlos hasta que se ingrese un símbolo '*', al final de la cadena almacenada debe de colocar un 0.
  Void gets (int* direccion)



```
global gets:
gets:
    push ebp
    push ebx
    push ecx
    push edx
    push edi
    mov ebp,esp
    mov esi,[ebp+8]
    mov eax,3
    mov ebx,0
    mov ecx,esi
    mov edx,12
    int 0x80
    mov esp,ebp
    pop edi
    pop edx
    pop ecx
    pop ebx
    pop ebp
    ret
```

Figura 1: Subrutina gets para ser llamada desde C.

Descripción: La manera en la que se obtiene la dirección de la cadena donde que quiere guardar la cadena, es enviando a ESI la dirección que se envió como parámetro desde la función en C, luego se captura la cadena por medio del servicio 3 del NASM cuya captura será guardada en la dirección contenida por ESI con una longitud máxima de 12 dígitos, para capturar el dato como tal se hace uso de la interrupción 0x80, una vez finalizada la captura se mueve a ESP el valor del puntero de base (EBP) para dejar el ESP en su posición inicial, luego de restaurar todos los registros se sale de la subrutina, donde EAX retorna la dirección donde se guardó la cadena .

- Atoi: Subrutina que convierte una cadena de caracteres que representan un número decimal en un número decimal, recibe en ESI la dirección de la cadena y lo retorna en EAX, el máximo valor decimal a convertir es 4,294,967,295.
  Unsigned int Atoi (int* cadena)

```
global atoi:
atoi:
    push ebp
    push ebx
    push ecx
    push edx
    push esi
    mov ebp,esp
    xor ebx,ebx
    mov esi,[ebp+8]
    call len
    .next:
        movzx eax,byte[esi]
        inc esi
        sub al,'0'
        imul ebx,10
        add ebx,eax
        loop .next
    mov eax,ebx
    mov esp,ebp
    pop esi
    pop edx
    pop ecx
    pop ebx
    pop ebp
    ret

len:
    push esi
    xor ecx,ecx
    .check:
        cmp byte[esi],'*'
        jz .stlen
        inc ecx
        inc esi
        jmp .check
    .stlen:
        mov byte[esi],0
    pop esi
    ret
```

Figura 2: Subrutina Atoi y su auxiliar len para ser llamada desde C.

Descripción: En esta variante de la subrutina Atoi se realiza una copia de los contenidos de ESP en EBP, de esta manera se pueden acceder a los parámetros enviados desde C de manera más sencilla, en EBX se utilizara para ir almacenado el resultado de las conversiones ,en ESI se recibe la

dirección de la cadena donde se almacena la cadena a convertir a numero decimal, aunque antes de eso se realiza un llamado a la subrutina len para obtener la cantidad de dígitos que existen antes del * ; dentro de esta subrutina se realiza una comparación de byte en byte sobre los dígitos guardados en la dirección contenida por ESI hasta encontrar el * a la vez que se incrementa el contador de dígitos (en este caso se usa ECX), una vez encontrado el * este se remplazara por un 0 para indicar el fin de cadena y retornara a la subrutina Atoi, después de obtener la longitud de la cadena, se moverá a EAX el primer byte contenido en la dirección almacenada por ESI a través de la instrucción MOVZX y se incrementara ESI para la siguiente iteración, luego se le restara a AL (lugar real donde se encuentra el carácter) el carácter '0' de la tabla ASCII para pasarlo de tipo cadena a número, por medio de la instrucción IMUL se multiplicaran el contenido de EBX por 10 y este guardara en EBX, y ese resultado se le sumara lo que se obtuvo en EAX y así hasta completar todas la iteraciones; una vez realizado lo anterior se moverá a EAX el resultado final el cual será retornado como un entero para ser utilizado en C como número.

- PrintDec: Subrutina que imprime los valores en formato decimal, ejemplo, se ingresa en EAX un 0x64 y muestra en pantalla un 100 decimal como cadena, el número máximo a mostrar debe ser 4,294,967,295.
  Void printDec (unsigned int val)

```
global printDec:
printDec:
    push ebp
    push eax
    push ebx
    push edx
    mov ebp,esp
    mov ebx,10
    xor edx,edx
    xor ecx,ecx
    mov eax,[ebp+8]
    .convert:
        div ebx
        add dx,'0'
        push dx
        inc ecx
        xor edx,edx
        cmp eax,0
        jne .convert
    .print:
        pop dx
        call print
        loop .print
    mov esp,ebp
    pop edx
    pop ebx
    pop eax
    pop ebp
    ret
```

Figura 3: Subrutina printDec ser llamada desde C.

Descripción: primero se realizan la operación con los registros ECX y EDX para ponerlos en cero, de manera que no afecten el resultado, el registro ECX se utiliza como contador, específicamente la cantidad de dígitos que tendrá el numero decimal a mostrar y EDX es el registro que contiene el residuo de las divisiones, EBX se utiliza únicamente para almacenar el numero 10 decimal; dentro de la etiqueta convert se realizan las divisiones entre EAX (el número que se desea mostrar en pantalla) y EBX cuyo residuo será enviado a la pila para almacenarlo de manera temporal, esto se realizó de esta manera ya que al importar funciones de ensamblador a C no permite guardar en variables (puede que si se pueda, pero se desconoce cómo hacerlo), regresando al código, una vez enviado el residuo a la pila se realiza la operación XOR EDX, EDX para ponerlo en 0, en caso de que el cociente (el registro EAX) aun no sea 0, se realizara un salto a la etiqueta convert para repetir el proceso, en caso de que el cociente sea 0, se procederá a llamar a la subrutina print para ir mostrando en terminal los dígitos que se tienen almacenados en la pila.

```
print:
    push eax
    push ebx
    push ecx
    push edx
    mov eax,4
    mov ebx,1
    mov ecx,esp
    mov edx,1
    int 0x80
    pop edx
    pop ecx
    pop ebx
    pop eax
    ret
```

Figura 4: Subrutina auxiliar print de printDec y printHex.

Descripción: La única finalidad de esta subrutina es mostrar en pantalla el digito extraído de la pila por medio del servicio 4 de NASM, en este caso ESP contiene la dirección del digito que se envió a la pila, el cual al ser mostrado en pantalla será el MSB del número decimal que se intenta mostrar y el ultimo digito que “saque” desde printDec será el LSB.

2- Cree un programa llamado p12.asm que contenga las siguientes subrutinas para el intérprete 80x86:

a) SetBit: Activa un bit del registro AX. El número de bit a activar está dado por CL.

```
44 setBit proc
45 push ax
46 push cx
47 ror ax,cl
48 or ax,1h
49 rol ax,cl
50 call printBin
51 call newLine
52 pop cx
53 pop ax
54 ret
55 endp
```

Figura 5: Subrutina setBit para ensamblador de 16 bits.

Descripción: Al igual que su variante de 32 bits, primero se debe de realizar una rotación a la de derecha de CL veces para tener en la posición menos significativa el bit que se quiere manipular, una vez realizada la rotación se le aplica la operación OR haciendo uso de una máscara de 0x1, se esa manera se activara el bit menos significativo de AX, posteriormente se realiza una rotación a la izquierda de CL veces para regresar el bit a su posición original, para mostrar el resultado se llama a la subrutina printBin para mostrar el resultado en la consola de MTTTY junto a un salto de línea.

b) ClearBit: Desactiva un bit del registro AX. El número de bit a desactivar está dado por CL.

```
clearBit proc
push ax
push cx
ror ax,cl
and ax,0FFFEh
rol ax,cl
call printBin
call newLine
pop cx
pop ax
ret
endp
```

Figura 6: Subrutina clearBit para ensamblador de 16 bits.

Descripción: Al igual que su variante de 32 bits, primero se debe de realizar una rotación a la de derecha de CL veces para tener en la posición menos significativa el bit que se quiere manipular, una vez realizada la rotación se le aplica la operación AND haciendo uso de una máscara de 0xFFFE, de esa manera se desactivara el bit menos significativo de AX, posteriormente se realiza una rotación a la izquierda de CL veces para regresar el bit a su posición original, para mostrar el resultado se llama a la subrutina printBin para mostrar el resultado en la consola de MTTTY junto a un salto de línea.

c) NotBit: Invierte un bit del registro AX. El número de bit a invertir está dado por CL.

```
notBit proc
push ax
push cx
ror ax,cl
xor ax,1h
rol ax,cl
call printBin
call newLine
pop cx
pop ax
ret
endp
```

Figura 7: Subrutina clearBit para ensamblador de 16 bits.

Descripción: Al igual que su variante de 32 bits, primero se debe de realizar una rotación a la de derecha de CL veces para tener en la posición menos significativa el bit que se quiere manipular, una vez realizada la rotación se le aplica la operación XOR haciendo uso de una máscara de 0x1, de esa manera se invertirá el bit menos significativo de AX, posteriormente se realiza una rotación a la izquierda de CL veces para regresar el bit a su posición original, para mostrar el resultado se llama a la subrutina printBin para mostrar el resultado en la consola de MTTTY junto a un salto de línea.

3- Realizar las siguientes subrutinas en ensamblador para la placa T-Juino.

- **SetBitPort:** Manipula la información de un puerto dado por DX para activar un determinado bit, es decir, mediante ella se puede activar (Hacer 1) un bit del puerto. El número del bit está en el rango de 0 a 7 siendo 7 el bit más significativo y está dado por CL.

```
setBitPort proc
push ax
push cx
push dx
ror al,cl
or al,1h
rol al,cl
call outportC
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp
```

Figura 8: Subrutina setBitPort para ensamblador de 16 bits (limitada a 8 bits).

Descripción: Como se puede apreciar en la figura 8, la subrutina setBitPort tiene el mismo comportamiento que su contraparte que no hace uso de puertos (setBit), los únicos cambios presentes son que el registro AL recibe el dato a manipular y que se agregó la llamada a la subrutina outPortC, la cual tiene la finalidad de "Sacar" el resultado por el puerto C (0x42) cuyo resultado es indicado a través de LEDS y la subrutina printBin8.

- **ClearBitPort:** Manipula la información de un puerto dado por DX para desactivar un determinado bit, es decir, mediante ella se puede desactivar (hacer 0) un bit del puerto. El número del bit está en el rango de 0 a 7 siendo 7 el bit más significativo y está dado por CL.

```
clearBitPort proc
push ax
push cx
push dx
ror al,cl
and al,0FEh
rol al,cl
call outportC
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp
```

Figura 9: Subrutina clearBitPort para ensamblador de 16 bits (limitada a 8 bits).

Descripción: Como se puede apreciar en la figura 8, la subrutina clearBitPort tiene el mismo comportamiento que su contraparte que no hace uso de puertos (clearBit), los únicos cambios presentes son que ahora el registro AL recibe el dato a manipular, la máscara se cambió a 0xFE y se agregó la llamada a la subrutina outPortC, la cual tiene la finalidad de “Sacar” el resultado por el puerto C (0x42) cuyo resultado es indicado a través de LEDS y la subrutina printBin8.

- **NotBitPort:** Manipula la información de un puerto dado por DX para invertir un determinado bit, es decir, mediante ella se puede invertir un bit del puerto. El número del bit está en el rango de 0 a 7 siendo 7 el bit más significativo y está dado por CL.

```
notBitPort proc
push ax
push cx
push dx
ror al,cl
xor al,1h
rol al,cl
call outportC
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp
```

Figura 10: Subrutina notBitPort para ensamblador de 16 bits (limitada a 8 bits).

Descripción: Como se puede apreciar en la figura 8, la subrutina setBitPort tiene el mismo comportamiento que su contraparte que no hace uso de puertos (notBit), los únicos cambios presentes son que el registro AL recibe el dato a manipular y que se agregó la llamada a la subrutina outPortC, la cual tiene la finalidad de “Sacar” el resultado por el puerto C (0x42) cuyo resultado es indicado a través de LEDS y la subrutina printBin8.

Subrutinas Auxiliares

```
printBin proc
push ax
push dx
push cx
push bx
mov bx,ax
mov cx,16
.convert:
    xor al,al
    shl bx,1
    adc al,'0'
    call putchar
    loop .convert
pop bx
pop cx
pop dx
pop ax
ret
endp
```

```
printBin8 proc
push ax
push dx
push cx
mov ah,al
mov cx,8
.convert2:
    xor al,al
    shl ah,1
    adc al,'0'
    call putchar
    loop .convert2
pop cx
pop dx
pop ax
ret
endp
```

Figura 11: Subrutinas para desplegar un número en Binario (16 bits y 8 bits respectivamente)

```
delay proc
push cx
mov cx,0ffffh
.delay:
    nop
    loop .delay
pop cx
ret
endp
```

Figura 12: Subrutina delay.

Descripción: La subrutina de la figura 12 tiene la finalidad de provocar un delay en la ejecución del programa, esto se realiza repiendo la instrucción nop un total de 65,535 veces; la instrucción tiene la particularidad de no afectar a los registros a excepcion del puntero de instrucciones (IP), el tener que realizar esta instrucción que aparentemente no hace nada la sufiente cantidad de veces, provocara la ilusión de que programe se “Detenga” por algunos milisegundos.

```
global printHex:
printHex:
    push ebp
    push eax
    push ebx
    push edx
    mov ebp,esp
    mov ebx,10h
    xor edx,edx
    xor ecx,ecx
    mov eax,[ebp+8]
    .convert:
        div ebx
        cmp dx,9
        jbe .menor
        add dx,7
    .menor:
        add dx,'0'
        push dx
        inc ecx
        xor edx,edx
        cmp eax,0
        jne .convert
    .print:
        pop dx
        call print
        loop .print
    mov esp,ebp
    pop edx
    pop ebx
    pop eax
    pop ebp
    ret
```

Figura 13: Subrutina printHex.

Descripción: Como se puede observar en la figura 13, la subrutina printHex tiene el mismo funcionamiento que la subrutina printDec de la figura 3, con la diferencia de que ahora en vez de dividir en un 10 decimal ahora divide entre un 10 hexadecimal, y que en ocasiones es necesario hacer un ajuste al carácter a mostrar en pantalla, es decir, debido a que los caracteres que representan a las letras A, B, C, D, E y F tienen una separación de 7 espacios respecto a los que representan a los números del 0 al 9 en la tabla ASCII, se le debe sumar un 7 al residuo en caso de que este sea mayor que 9, por medio de ese ajuste se puede mostrar números hexadecimales que contengan las letras desde A hasta F de manera correcta, en caso de que el residuo sea menor o igual a 9, no se necesita ningún ajuste y se puede convertir a carácter de manera directa.

Pruebas

Ensamblador y C.

Prueba 1

```
brayan@Cake-Roll:~/OacAgain/p12$ ./run_all.sh
Ingresa La Cadena:
123*
Salida En Terminal De PrintHex:
7B
Salida En Terminal De PrintDec:
123

Ingresa La Cadena:
459*
Salida En Terminal De PrintHex:
1CB
Salida En Terminal De PrintDec:
459

Ingresa La Cadena:
1456*
Salida En Terminal De PrintHex:
5B0
Salida En Terminal De PrintDec:
1456
```

Figura P1: Prueba de C con Ensamblador.

Prueba 2

```
brayan@Cake-Roll:~/OacAgain/p12$ ./run_all.sh
Ingresa La Cadena:
4294967295*
Salida En Terminal De PrintHex:
FFFFFFFF
Salida En Terminal De PrintDec:
4294967295

Ingresa La Cadena:
666777999*
Salida En Terminal De PrintHex:
27BE398F
Salida En Terminal De PrintDec:
666777999

Ingresa La Cadena:
4578*
Salida En Terminal De PrintHex:
11E2
Salida En Terminal De PrintDec:
4578
```

Figura P2: Prueba de C con Ensamblador.

Prueba 3

```
brayan@Cake-Roll:~/OacAgain/p12$ ./run_all.sh
Ingresa La Cadena:
455*9
Salida En Terminal De PrintHex:
1C7
Salida En Terminal De PrintDec:
455

Ingresa La Cadena:
7896549*
Salida En Terminal De PrintHex:
787DE5
Salida En Terminal De PrintDec:
7896549

Ingresa La Cadena:
555666*555
Salida En Terminal De PrintHex:
87A92
Salida En Terminal De PrintDec:
555666
```

Figura P3: Prueba de C con Ensamblador.

Pruebas De SetBit, ClearBit, NotBit

Prueba 1 SetBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0AEh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,14
    call setBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010101110
0100000010101110

0000000010101110
0100000010101110
```

Figura P4: Código y Salida en MTTTY de la prueba 1 de SetBit.

Prueba 2 SetBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0AEh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,9
    call setBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010101110
0000001010101110

0000000010101110
0000001010101110
```

Figura P5: Código y Salida en MTTTY de la prueba 2 de SetBit.

Prueba 3 SetBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0AEh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,7
    call setBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010101110
0000000010101110

0000000010101110
0000000010101110
```

Figura P6: Código y Salida en MTTTY de la prueba 3 de SetBit (Sin Cambios).

Prueba 1 ClearBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0FFh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,7
    ;call setBit
    call clearBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000011111111
0000000001111111

0000000011111111
0000000001111111
```

Figura P7: Código y Salida en MTTTY de la prueba 1 de clearBit.

Prueba 2 ClearBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0FFh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,15
    ;call setBit
    call clearBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
0000000011111111
0000000011111111

0000000011111111
0000000011111111
```

Figura P8: Código y Salida en MTTTY de la prueba 2 de clearBit (Sin cambios).

Prueba 3 ClearBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0FFh
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,0
    ;call setBit
    call clearBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000011111111
0000000011111110

0000000011111111
0000000011111110
```

Figura P9: Código y Salida en MTTTY de la prueba 3 de clearBit.

Prueba 1 NotBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0A7h
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,0
    ;call setBit
    ;call clearBit
    call notBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010100111
0000000010100110

0000000010100111
0000000010100110
```

Figura P10: Código y Salida en MTTTY de la prueba 1 de notBit.

Prueba 2 NotBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0A7h
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,13
    ;call setBit
    ;call clearBit
    call notBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010100111
0010000010100111

0000000010100111
0010000010100111
```

Figura P11: Código y Salida en MTTTY de la prueba 2 de notBit.

Prueba 3 NotBit

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    ;call outportC
    mov dx,PC
    mov al,0A7h
    call printBin
    ;call outportC
    call newLine
    call delay
    mov cl,9
    ;call setBit
    ;call clearBit
    call notBit
    ;call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
0000000010100111
0000001010100111

0000000010100111
0000001010100111
```

Figura P12: Código y Salida en MTTTY de la prueba 3 de notBit.

Pruebas De SetBitPort, ClearBitPort, NotBitPort

Prueba 1 SetBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0Eh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,6
    call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
00001110
01001110

00001110
01001110
```

Figura P13: Código y Salida en MTTTY de la prueba 1 de SetBitPort.

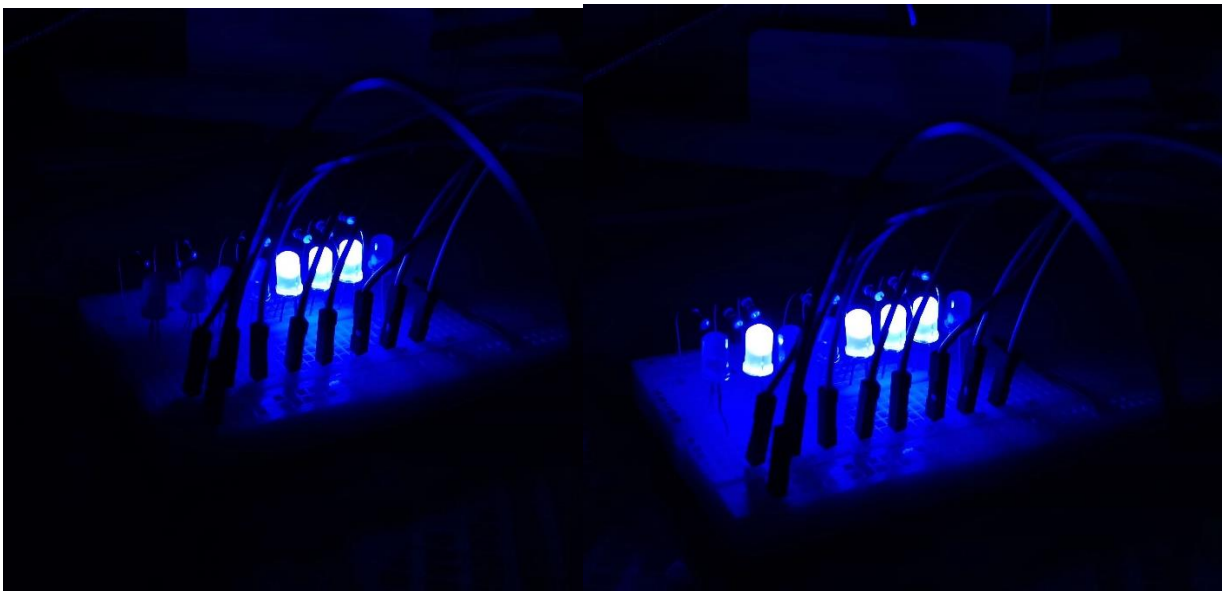


Figura P14: Numero Original (Izquierda) y Numero con bit 6 Activado (Derecha).

Prueba 2 SetBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0Eh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,0
    call setBitPort
    ;call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
00001110
00001111

00001110
00001111
```

Figura P15: Código y Salida en MTTTY de la prueba 2 de SetBitPort.

Figura P16: Numero Original (Izquierda) y Numero con bit 0 Activado (Derecha).

Prueba 1 ClearBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0FFh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,3
    ;call setBitPort
    call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
11111111
11110111

11111111
11110111
```

Figura P17: Código y Salida en MTTTY de la prueba 1 de clearBitPort.

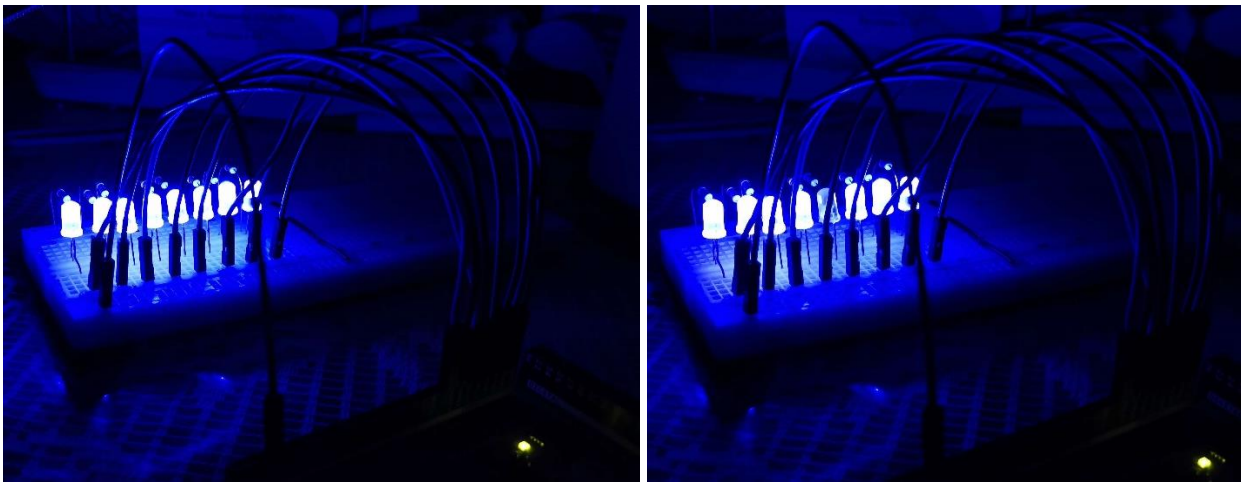


Figura P18: Numero Original (Izquierda) y Numero con bit 3 desactivado (Derecha).

Prueba 2 ClearBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0FEh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,0
    ;call setBitPort
    call clearBitPort
    ;call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
11111110
11111110

11111110
11111110
```

Figura P19: Código y Salida en MTTTY de la prueba 2 de clearBitPort.

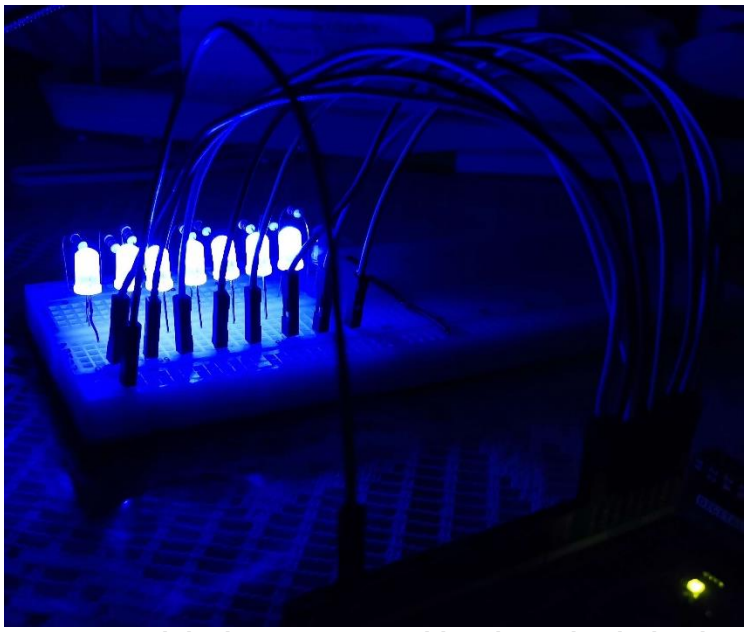


Figura P20: Numero Original y Numero con bit 0 desactivado (No hay cambios).

Prueba 1 NotBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0AEh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,0
    ;call setBitPort
    ;call clearBitPort
    call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
10101110
10101111

10101110
10101111
```

Figura P21: Código y Salida en MTTTY de la prueba 1 de notBitPort.

Figura P22: Numero Original (Izquierda) y Numero con bit 0 invertido (Derecha).

Prueba 2 NotBitPort

```
Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0AEh
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,4
    ;call setBitPort
    ;call clearBitPort
    call notBitPort
    call newLine
    call delay
    jmp start
    ret
    endp
```

```
>g
10101110
10111110

10101110
10111110
```

Figura P23: Código y Salida en MTTTY de la prueba 2 de notBitPort.

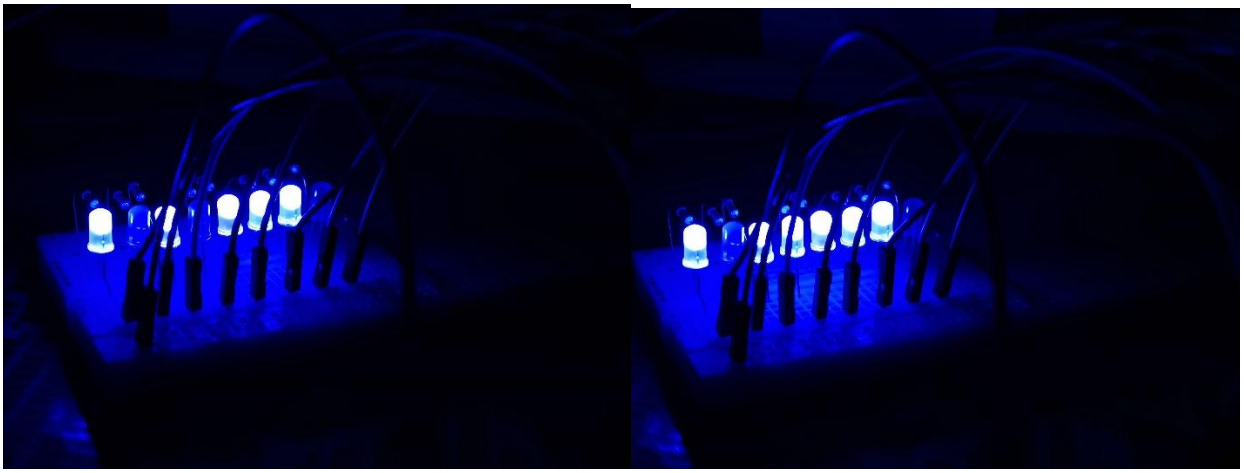


Figura P24: Numero Original (Izquierda) y Numero con bit 4 invertido (Derecha).

Circuito Utilizado

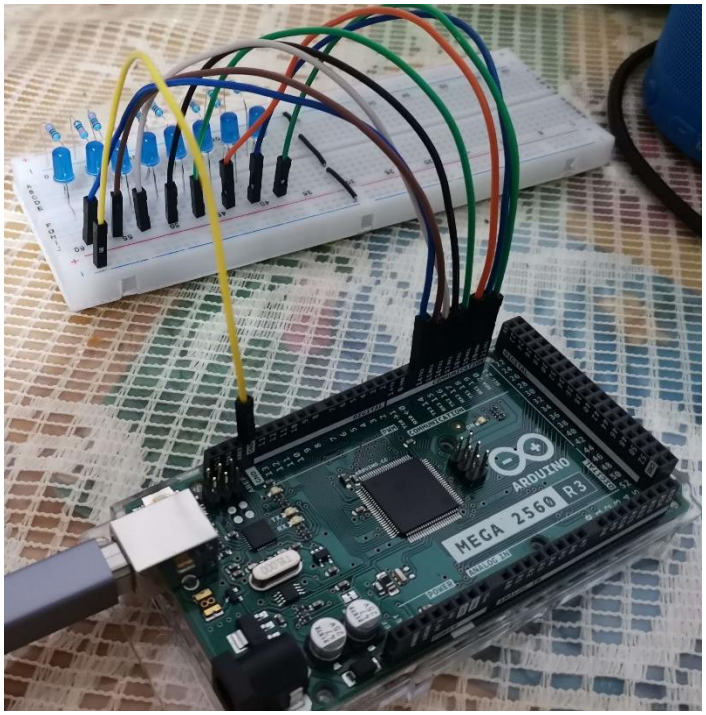


Figura C1: Circuito Utilizado

Código

Archivo P12_enC

```
# include <stdio.h>
void writeTag();
extern int* gets(int* cad);
extern unsigned int atoi(int* cadena);
extern int printDec(unsigned int val);
extern int printHex(unsigned int val);
int main(){
    int cad;
    int *pcad=&cad;
    for(int i=0;i<=2;i++){
        writeTag();
        int* cadena=gets(pcad);
        int num= atoi(cadena);
        printf("Salida En Terminal De PrintHex:\n");
        printHex(num);
        printf("\nSalida En Terminal De PrintDec:\n");
        printDec(num);
        printf("\n");
        printf("\n");
    }
```

```
    return 0;
}
void writeTag(){
    printf("Ingresa La Cadena:");
    printf("\n");
}
```

Archivo p12_enASM

```
section .data
section .bss
section .text
global gets:
gets:
    push ebp
    push ebx
    push ecx
    push edx
    push edi
    mov ebp,esp
    mov esi,[ebp+8]
    mov eax,3
    mov ebx,0
    mov ecx,esi
    mov edx,12
    int 0x80
    mov esp,ebp
    pop edi
    pop edx
    pop ecx
    pop ebx
    pop ebp
    ret

global atoi:
atoi:
    push ebp
    push ebx
    push ecx
    push edx
    push esi
    mov ebp,esp
    xor ebx,ebx
    mov esi,[ebp+8]
    call len
    .next:
        movzx eax,byte[esi]
```

```
        inc esi
        sub al,'0'
        imul ebx,10
        add ebx,eax
        loop .next
    mov eax,ebx
    mov esp,ebp
    pop esi
    pop edx
    pop ecx
    pop ebx
    pop ebp
    ret

len:
    push esi
    xor ecx,ecx
    .check:
        cmp byte[esi],'*'
        jz .stlen
        inc ecx
        inc esi
        jmp .check
    .stlen:
        mov byte[esi],0
    pop esi
    ret


global printDec:
printDec:
    push ebp
    push eax
    push ebx
    push edx
    mov ebp,esp
    mov ebx,10
    xor edx,edx
    xor ecx,ecx
    mov eax,[ebp+8]
    .convert:
        div ebx
        add dx,'0'
        push dx
        inc ecx
        xor edx,edx
```

```
        cmp eax,0
        jne .convert
    .print:
        pop dx
        call print
        loop .print
    mov esp,ebp
    pop edx
    pop ebx
    pop eax
    pop ebp
    ret
print:
    push eax
    push ebx
    push ecx
    push edx
    mov eax,4
    mov ebx,1
    mov ecx,esp
    mov edx,1
    int 0x80
    pop edx
    pop ecx
    pop ebx
    pop eax
    ret

global printHex:
printHex:
    push ebp
    push eax
    push ebx
    push edx
    mov ebp,esp
    mov ebx,10h
    xor edx,edx
    xor ecx,ecx
    mov eax,[ebp+8]
    .convert:
        div ebx
        cmp dx,9
        jbe .menor
        add dx,7
    .menor:
        add dx,'0'
```

```
        push dx
        inc ecx
        xor edx,edx
        cmp eax,0
        jne .convert
    .print:
        pop dx
        call print
        loop .print
    mov esp,ebp
    pop edx
    pop ebx
    pop eax
    pop ebp
    ret
```

Archivo p12

```
.model tiny
locals
.data
    PA dw 0040h
    PB dw 0041h
    PC dw 0042h
    RCtr dw 0043h
    PTOs_all_out db 80h
.code
   org 100h

Main proc
    start:mov sp,0fffh
    mov dx,RCtr
    mov al,PTOs_all_Out
    call outportC
    mov dx,PC
    mov al,0A7h
    call printBin8
    call outportC
    call newLine
    call delay
    mov cl,5
    ;call setBit
    ;call clearBit
    ;call notBit
    ;call setBitPort
    ;call clearBitPort
    call notBitPort
```

```
    call newLine
    call delay
    jmp start
    ret
    endp

outportC proc
out dx,al
ret
endp

; Parte 2 (Sin Puertos)
setBit proc
push ax
push cx
ror ax,cl
or ax,1h
rol ax,cl
call printBin
call newLine
pop cx
pop ax
ret
endp

clearBit proc
push ax
push cx
ror ax,cl
and ax,0FFFEh
rol ax,cl
call printBin
call newLine
pop cx
pop ax
ret
endp

notBit proc
push ax
push cx
ror ax,cl
xor ax,1h
rol ax,cl
call printBin
call newLine
```

```
pop cx
pop ax
ret
endp

; Parte 3 (Con Puertos)
setBitPort proc
push ax
push cx
push dx
ror al,cl
or al,1h
rol al,cl
call outportC
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp

clearBitPort proc
push ax
push cx
push dx
ror al,cl
and al,0FEh
rol al,cl
call outportC
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp

notBitPort proc
push ax
push cx
push dx
ror al,cl
xor al,1h
rol al,cl
call outportC
```

```
call printBin8
call newLine
pop dx
pop cx
pop ax
ret
endp

; Subrutinas Auxiliares
delay proc
push cx
mov cx,0ffffh
.delay:
    nop
    loop .delay
pop cx
ret
endp

printBin proc
push ax
push dx
push cx
push bx
mov bx,ax
mov cx,16
.convert:
    xor al,al
    shl bx,1
    adc al,'0'
    call putchar
    loop .convert
pop bx
pop cx
pop dx
pop ax
ret
endp

printBin8 proc
push ax
push dx
push cx
mov ah,al
mov cx,8
.convert2:
```

```
    xor al,al
    shl ah,1
    adc al,'0'
    call putchar
    loop .convert2
pop cx
pop dx
pop ax
ret
endp

newLine proc
 push ax
 mov al,10
 call putchar
 mov al,13
 call putchar
 pop ax
 ret
endp

putchar proc
 push ax
 push dx
 mov dl,al
 mov ah,2
 int 21h
 pop dx
 pop ax
 ret
 endp

end Main
```

Link De GitHub Con Código Completo

https://github.com/BrayanLMercado/OAC_Practica12_v2.git

Script De Bash Utilizado

```
$ run_all.sh
1   nasm -f elf p12EnASM.asm
2   gcc -m32 -c p12_enC.c
3   gcc -m32 p12EnASM.o p12_enC.o -o p12
4   ./p12
```

Figura 14: Script Para La ejecución Del Programa

Conclusiones y Comentarios

- El diseñar funciones en ensamblador para ser llamadas desde C requiere uso de pila para almacenar información en caso de que se necesita almacenar fragmentos de cadenas para luego ser utilizadas más tarde, lo que en ensamblador de 32 bits es un mas enredoso debido a que se necesitan 4 registros para mostrar una cadena en terminal a diferencia del ensamblador de 16 bits que solo requiere 2 registro para lo mismo.

- Las subrutinas requeridas para la parte 2 de la practica casi solo fue tomar las subrutinas de la practica 8 y reducir el tamaño de los registros a sus versiones de 16 bits y hacer ajustes en las máscaras correspondientes.

- Como posiblemente se habrá dado cuenta, las capturas de pantalla de la parte 2 se ven más borrosas de lo que deberían estar, esto es por un problema con GUI Assembler, aún se desconoce la forma de solucionar este problema.

Dificultades En El Desarrollo

- El hacer uso de la subrutina puts para imprimir cadenas en C o la versión de ensamblador no funcionaban por algún motivo desconocido, por lo que tuvo que dejar de lado la idea de crear esa subrutina.

- La manera de implementar el delay en la parte 2 fue más tardado de lo que debería debido a que no se sabía cuál instrucción utilizar para provocar el delay sin terminar afectando los registros, por lo que se tuvo que realizar una mini investigación en internet acerca de ello, para el final resultar que esa instrucción estaba en una de las presentaciones de la clase, después de conocer la instrucción necesaria, únicamente se puso en loop la suficiente cantidad de veces para provocar el delay de casi 1 segundo, en la segunda referencia incluye link a la pregunta de Stack Overflow donde se encontró el documento de 2522 páginas donde estaba la instrucción, específicamente se encuentra en la página 4-165.

Referencias

Guide to x86 Assembly. (s/f). Recuperado el 2 de noviembre de 2023, de https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

M. Burge, “What’s the difference between the x86 NOP and FNOP instructions?”, Stack Overflow. Consultado: el 23 de noviembre de 2023. [En línea]. Disponible en: https://stackoverflow.com/q/25008772