Práctica Nº 1

Manejo de interrupciones, reloj y adquisición de datos.

La siguiente práctica debe realizarse utilizando el entorno de programación Borland C sobre el Sistema Operativo Microsoft Windows. Además, para realizar los ejercicios 2 y 4 se necesitará el simulador de la placa adquisidora ADQ12 otorgado por la cátedra.

Ejercicio 1:
Realice una breve investigación sobre el funcionamiento del PIT (Programmable Interval Timer) 8253/8254. Tenga en cuenta los siguientes aspectos:

- función y sub-funciones de cada registro.
- asociación de registros con dispositivos de la PC.
- ubicación en memoria de cada registro.

Ejercicio 2:
Escriba un programa que instale un manejador de interrupción y modifique la frecuencia de interrupción del reloj (PIT 8253/8254). Dicho programa debe recibir un parámetro que indique la frecuencia de interrupciones por segundo e imprimir cada segundo la cantidad de veces que se invocó el manejador de interrupciones. Después de 20 segundos el programa debe finalizar.

Nota: para realizar el ejercicio tenga en cuenta que:

- *nunca debe utilizar operaciones de entrada/salida de alto nivel en una interrupción.*
- *debe salvar para restaurar el manejador de interrupción previo al finalizar el programa.*
- *debe restablecer la frecuencia de reloj al valor por defecto al finalizar el programa.*

Ejercicio 3:
Realice una investigación sobre la placa ADQ12.
a) Describa las características técnicas de la placa.
b) Describa las funciones y sub-funciones de los registros CTREG, INSTR, OUTBR, ADLOW, ADHIGH.
c) Indique el procedimiento de lectura de un valor analógico con interrupciones y sin interrupciones. Explique el porqué de las diferencias.

Ejercicio 4:
Un motor tiene conectado un sensor de temperatura que proporciona 0° para 0V y 120° para 5V. Este sensor mide la temperatura del motor. Cuando esta supera los 65° se activa el sistema de ventilación que está conectado a la salida digital 1. Si la temperatura llegara a alcanzar los 85° el motor debe apagarse utilizando el canal digital 0.
Escriba un programa que adquiera periódicamente la temperatura y controle las salidas digitales de la forma planteada anteriormente. Grafique la señal analógica y las líneas de temperatura (utilizando graphics.h) en un sistema de ejes cartesianos donde el eje "x" indica el momento de la captura y el eje "y" indica el valor adquirido.

Nota: tenga en cuenta que para poder compilar con la biblioteca gráfica debe habilitar la opción desde el menú Options->Linker->Libraries.

Universidad Nacional de La Plata - Facultad de Informática

Ingeniería en Computación

UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas de tiempo real 2018

Informe - Práctica N°1

Configuración de Chip PIT & Placa ADQ12

Alumno: Camino, Lucas Agustín 1277/0

Introducción

En esta práctica se nos presentaron dos componentes de la máquina virtual Bochs: chip timer Intel8253 (Programmable Interval Timer – PIT) y placa ADQ12. Para el ejercicio 2 tuvimos que escribir un programa en C para configurar la cantidad de veces por segundo que se interrumpa a la CPU, informando este número y la cantidad de segundos pasados, al usuario, por pantalla. La ejecución debía terminar una vez pasados veinte segundos. Por otra parte, en el ejercicio 4, la tarea consistió en utilizar la interfaz GUI para simular entradas tanto analógicas como digitales y salidas digitales. El programa a realizar debía recibir el valor de la entrada analógica y en base a su análisis, activar o desactivar salidas digitales.

Dejamos asentado que para el informe sólo vamos a tratar los aspectos cuyo conocimiento sea de vital importancia para la resolución de la práctica.

Ejercicio 1

El PIT está compuesto de tres contadores. Cada contador es un canal independiente que dispone de seis modos de funcionamiento. Para poder configurar los canales es necesario editar un registro de control, el cual recibe el nombre de Control Word Register (CWR). En la máquina virtual, los registros del PIT comienzan en la dirección 0x40h. Las direcciones que permiten acceder a sus registros son: 0x40h para canal0, 0x41h para canal1, 0x42h para canal2 y 0x43h para el CWR. Todos los registros pueden escribirse y leerse, a excepción del CWR que sólo permite la operación de escritura. El PIT es alimentado con un reloj de frecuencia 1193180Hz.

El registro de configuración tiene un tamaño de ocho bits. Cada bit o conjunto de bits tendrán una función determinada:

SC1	**SC0**	CRW1	CRW0	M2	M1	M0	BCD
7	6	5	4	3	2	1	0

Seleccionar Contador (SC)
00: Contador 0
01: Contador 1
10: Contador 2
11: Lectura retardada

Control Read/Write (CRW)
00: Almacenar en buffer para lectura segura
01: Read/Write de byte menos significativo del canal
10: Read/Write de byte más significativo del canal
11: Read/Write de byte menos signif. primero y más signif. después.

Modo (M)
0 0 0: Modo 0
0 0 1: Modo 1
X 1 0: Modo 2
X 1 1: Modo 3
1 0 0: Modo 4
1 0 1: Modo 5

Decimal Codificado en Binario (BCD)
0: Binario
1: BCD

Los modos de funcionamiento de cada canal se detallan a continuación:

0) Ciclo simple: la cuenta empieza al establecer el modo o al escribir un nuevo valor de cuenta inicial. Mediante la entrada GATE se puede controlar el conteo. Si GATE=1 cuenta, sino se pausa. La salida OUT se mantiene en nivel bajo hasta que el contador llega a cero. Después queda en alto.
1) Ciclo simple redisparable: la cuenta se inicia (o reinicia) en el primer flanco de bajada de la señal de reloj con GATE en estado alto. La salida se mantiene a nivel bajo hasta que finaliza la cuenta.
2) Interrupción periódica: la cuenta se inicia al establecer el modo, al escribir un nuevo valor en el contador o cuando el valor del mismo llega a cero. La salida se mantiene en alto si la cuenta es distinta de cero, y en bajo al llegar a cero.
3) Generador de onda cuadrada: Igual que el modo 2, salvo en el valor de la salida. Si el valor de cuenta inicial es N:
    - La salida permanece a nivel alto durante los N/2 primeros ciclos ((N+1)/2 si N es impar)
    - La salida permanece a nivel bajo durante los N/2 últimos ciclos ((N-1)/2 si N es impar)
4) Interrupción sobre fin de cuenta: la cuenta se inicia al establecer el modo o al escribir un nuevo valor de cuenta inicial. La salida se mantiene a nivel alto salvo mientras la cuenta tiene el valor nulo.
5) Interrupción redisparable: la cuenta se inicia con cualquier flanco positivo de la señal GATE. Es, por tanto, redisparable. La salida se mantiene a nivel alto salvo mientras la cuenta tiene el valor nulo.

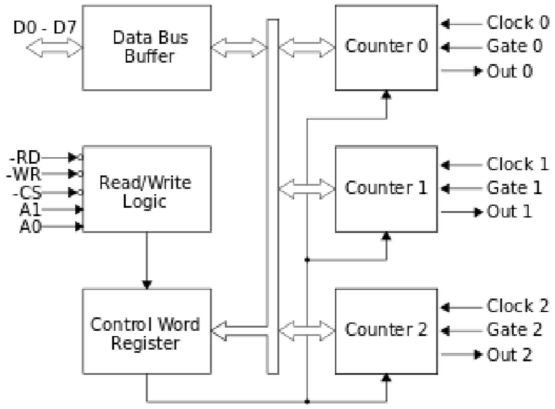


Diagrama de bloques del Intel 8253

Cada contador tiene un tamaño de dos bytes, los cuales se escriben según indique el registro de control. Los contadores funcionan de forma descendente. Dependiendo del valor que se guarde en el contador se puede modificar la frecuencia con que se activa la salida del mismo.

A todo esto, también se debe saber que cada contador tiene asociados usos predefinidos y para nuestro caso nos interesa el canal 0 ya que se usa para llevar la hora del día. Tiene conexión con el chip Controlador de Interrupciones Programable (PIC), ya que al activar su salida también se activa la interrupción 8 (INT8), es decir la interrupción del timer. Esta interrupción de software, según viene configurada la máquina, ejecuta el manejador de interrupciones apuntado por la dirección que se encuentra en la posición 0x1Ch del vector de interrupciones.

Ejercicio 2

```
#pragma inline /* declara que hay instrucciones en ensamblador */
#include <stdio.h>
#include <dos.h>   /* enable(), disable(), getvec(), setvec() */
#include <math.h>  /* ceil(), fmod() */
#include <float.h> /* _clear87() */

#define PORT_CONFIG 0x43
#define PORT_TIMER0 0x40
#define INTER 0x1C

/* Esqueleto de rutina de interrupcion

   Se programa el timer del PC para elegir una frecuencia de interrupcion
   entre 18.2 Hz y 1.193MHz. La rutina de interrupcion se instala en la
   posicion 0x1C del vector para ser ejecutada por la interrupcion del
   timer, es decir, la interrupcion 0x08.
   Lucas Agustin Camino 1277/0
*/

int vecesQueEntroAlHandler = 0;

void interrupt (*viejotimer)(void);
void interrupt far nuevotimer(void)
{
    char *data87[94]; /* Se reserva espacio para los registros del copro */
    asm fsave data87  /* Se salvan los registros del copro */
    _clear87();       /* Se inicializa el copro */

    /* EMPIEZA EL CUERPO DEL MANEJADOR DE INTERRUPCION */
    vecesQueEntroAlHandler++;

    //TERMINA EL CUERPO DEL MANEJADOR DE INTERRUPCION
    asm frstor data87 /* Para salir se recuperan los registros del copro */
}

void main(void)
{
    float frec_base = 1193180.0;            /* La frecuencia de entrada es de
1.193 MHz*/
    unsigned int contador;                  /* contador de 16 bits */
    unsigned char contador_hi, contador_lo; /* 8 bits para cada parte alta y
baja del contador */
    float ts;
    float frecDeseada = 18.3; /* Periodo deseado en segundos. Inicializa el
reloj */
    unsigned char segundos = 0;
```

```
    unsigned int vecesParaUnSegundo = 0;
    char sig;

    printf("\nEmpieza programa\nDarEnter"); //Seguir
    scanf("%c", &sig);
    fflush(stdin);

    printf("\nIngresar frecuencia de interrupcion:\n"); //Elegir frecuencia
    scanf("%f", &frecDeseada);
    fflush(stdin);

    ts = 1 / frecDeseada;                                                    //
calculo ts
    printf("\nLa frecDeseada es %f, entonces ts = %f\n", frecDeseada, ts); //
Seguir
    scanf("%c", &sig);
    fflush(stdin);

    contador = ceil(frec_base * ts); /* se calcula el valor del contador.
                                        La funcion ceil(arg) recibe un decimal
y
                                        devuelve el entero mas cercano, por
arriba o abajo */
    printf("\nEl contador tendra el valor %d, ya que frec_base * ts = %f\n",
contador, ceil(frec_base * ts));
    scanf("%c", &sig);
    fflush(stdin);

    outportb(PORT_CONFIG, 0x36); /* se escribe el registro de control con el
valor
                                    0x36h o 00110110b para elegir:
                                    Contador 0 ||
ParteBajaPrimeroParteAltaDespues || Modo 3 || valor 16 bits */
    printf("\nSe configura el registro de control\n");
    scanf("%c", &sig);
    fflush(stdin);

    contador_hi = contador / 256; /* byte alto del contador */
    printf("\nEl contador_hi tiene %d\n", contador_hi);
    scanf("%c", &sig);
    fflush(stdin);

    contador_lo = contador % 256; /* byte bajo del contador */
    printf("\nEl contador_lo tiene %d\n", contador_lo);
    scanf("%c", &sig);
    fflush(stdin);

    outportb(PORT_TIMER0, contador_lo); /* escribe el bajo */
```

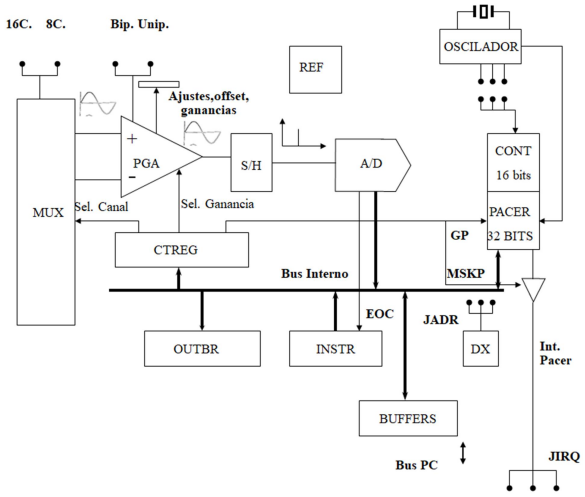
```
    outportb(PORT_TIMER0, contador_hi); /* y luego el alto */
    printf("\nSe cargo el valor del contador, primero parte baja y dps parte
alta\n");
    scanf("%c", &sig);
    fflush(stdin);

    vecesParaUnSegundo = ceil(1 / ts);
    printf("\nVamos a tener una interrupcion cada ts = %f segundos\nVoy a
necesitar contar %d interrupciones", ts, vecesParaUnSegundo);
    scanf("%c", &sig);
    fflush(stdin);

    disable(); /* desactiva interrupciones para cambiar la ISR */
    printf("\nSe deshabilitaron las interrupciones\n");
    scanf("%c", &sig);
    fflush(stdin);

    viejotimer = getvect(INTER); /* guardar direc vieja */
    setvect(0x1C, nuevotimer);  /* setear direc nueva */
    printf("\nHecho el intercambio de direcciones en el vector de
interrupcion\n");
    scanf("%c", &sig);
    fflush(stdin);
    printf("\n\n\n\n\n\n\n\n\nHabilitar interrupciones\n");
    scanf("%c", &sig);
    fflush(stdin);
    enable(); /* habilita interrupciones */

    while (segundos < 20)
    { //funcionar hasta que pasemos los 20 segundos
        if (vecesQueEntroAlHandler >= vecesParaUnSegundo)
        {
            segundos++;
            vecesQueEntroAlHandler = 0;
        }
        printf("\n%d", vecesQueEntroAlHandler);
        printf("\nSegundos pasados: %d", segundos);
    }

    /* Volver a condiciones originales */
    disable();                /* desactiva interrupciones para cambiar la ISR
*/
    setvect(0x1C, viejotimer); /* restaura la vieja ISR */
    outportb(0x43, 0x36);      /* restaura la velocidad del reloj */
    outportb(0x40, 0xFF);      /* cargando el contador 0, con unos */
    outportb(0x40, 0xFF);
    printf("\nTerminar programa\n");
    scanf("%c", &sig);
```

```
    fflush(stdin);
    enable(); /* habilita interrupciones */
}
```

Ejercicio 3

La placa ADQ12 (simulada) nos permite realizar conversiones de señales analógicas a señales digitales, para lo cual cuenta con doce bits. Si bien dispone de entradas tanto analógicas como digitales, sus salidas son digitales.

Diagrama de bloques



Esta placa es flexible, es decir, permite configurar muchos aspectos internos. Dentro de éstos encontramos el tipo de conexión de su amplificador operacional (PGA) o el canal del que se está recibiendo la señal a convertir (MUX).

Para la práctica fue necesario controlar las direcciones (cada una asociadas a registros) indicadas a continuación:

- 0x300h: STINR, CTREG
- 0x304h: OUTBR
- 0x308h: AD_LOW
- 0x309h: AD_HIGH

De nada sirve conocer las direcciones importantes para programar la placa si no se sabe qué funciones tiene cada una asociada. Las contamos ahora.

0x300h

Si se hace una lectura vamos a acceder a recuperar los datos del registro STINR pero si se escribe, se está modificando el contenido del registro CTREG.

El registro STINR contiene ocho bits, donde los cinco menos significativos tienen asociadas las entradas digitales, siendo el LSB capaz de despertar la interrupción 0 del PIC. Los tres bits más significativos sirven para conocer el estado de la salida del conversor analógico/digital (A/D). Un uso frecuente de estos tres bits es mediante polling saber si una conversión finalizó.

El registro CTREG es de control. Es usado para configurar si el amplificador funciona con entrada simple o diferencial. A su vez permite seleccionar el canal que quiere escuchar al momento de convertir. Este registro se pone en cero al reiniciar la computadora.

CTREG:

MSKP	GTP	AX1	AX0	MX3	MX2	MX1	MX0
7	6	5	4	3	2	1	0

MSKP: (1) Habilita interrup. PACER; (0) Enmascara interrup. PACER

GTP: (1) Dispara PACER; (0) Bloquea PACER

AX1-AX0: Seleccionar fondo de escala. (00) Fondo escala = 5v; (01) Fondo escala = 2v; (10) Fondo escala = 1v; (11) Fondo escala = 0v;

MX3-MX2-MX1-MX0: Elegir canal analógico.

0x304

Está permitida la programación de cada uno de los ocho bits del registro asociado a esta dirección. El OUTBR es un puerto de salida. Los tres bits menos significativos se usan para determinar con qué salida digital se quiere trabajar y el bit 3 (de derecha a izquierda) indica el estado de la salida.

0x308

Parte baja del resultado de dieciséis bits producto de la finalización de conversión A/D. Una lectura en esta dirección, además, inicia una conversión.

0x309

Parte baja del resultado de dieciséis bits producto de la finalización de conversión A/D.

Ejercicio 4

```
#pragma inline /* declara que hay instrucciones en ensamblador */
#include <stdio.h>
#include <dos.h>   /* enable(), disable(), getvec(), setvec() */
#include <math.h>  /* ceil(), fmod() */
#include <float.h> /* _clear87() */
```

```
#define STINR 0x300   /* direcc STINR. Registro donde los tres bits de mayor
peso guardan */
                      /* la salida del conversor A/D. El sexto bit tiene un 1
si se terminó la conversión */
#define OUTBR 0x304   // direcc del OUTBR para controlar salidas digitales
#define CTREG 0x300   /* registro de configuración */
                      /* bit 7: (1) habilitar interr PACER */
                      /*        (0) enmascarar interr PACER */
                      /* bit 6: (1) Disparar PACER */
                      /*        (0) Bloquear PACER */
                      /* bits 5-4: Configurar fondo de escala del AmpOp */
                      /*        (00) 5V */
                      /*        (01) 2V */
                      /*        (10) 1V */
                      /*        (11) 0V */
                      /* bits 3-2-1-0: Elección de canal analógico */
#define AD_HIGH 0x309 /* Se guarda el dato convertido por A/D */
#define AD_LOW 0x308  /* Una lectura de este registro inicia una conversión */
#define FE 5          /* Fondo escala = 5V */

#define PORT_CONFIG 0x43
#define PORT_TIMER0 0x40

int vecesQueEntroAlHandler = 0;
void interrupt (*viejotimer)(void);
void interrupt far nuevotimer(void)
{
        char *data87[94]; /* Se reserva espacio para los registros del copro
*/
        asm fsave data87  /* Se salvan los registros del copro */
        _clear87();       /* Se inicializa el copro */

        /* EMPIEZA EL CUERPO DEL MANEJADOR DE INTERRUPCION */
        vecesQueEntroAlHandler++;
        //TERMINA EL CUERPO DEL MANEJADOR DE INTERRUPCION

        asm frstor data87 /* Para salir se recuperan los registros del copro
*/
}

int encenderLed(unsigned char);
int apagarLed(unsigned char);

int main()
{
        /* INICIO Variables locales relacionadas con CONVERSOR_AD */
        unsigned int adValue, i;
        unsigned char ad_high, ad_low, conv;
```

```
        float analogicValue;
        /* FIN Variables locales relacionadas con CONVERSOR_AD */

        /* INICIO Variables locales relacionadas con interrupcion de CONTADOR
*/
        unsigned int vecesParaUnSegundo = ceil(1193180 / 65535);
        unsigned char contador_hi = 0xFF, contador_lo = 0xFF; /* 8 bits para
cada parte alta y baja del contador */
        /* FIN Variables locales relacionadas con interrupcion de CONTADOR */

        /* INICIO Configuracion de interrupcion de CONTADOR */
        outportb(PORT_CONFIG, 0x36);        /* se escribe el registro de
control con el valor
                                        0x36h o 00110110b para elegir:
                                        Contador 0 ||
ParteBajaPrimeroParteAltaDespues || Modo 3 || valor 16 bits */
        outportb(PORT_TIMER0, contador_lo); /* escribe el bajo */
        outportb(PORT_TIMER0, contador_hi); /* y luego el alto */
        /* FIN Configuracion de interrupcion de CONTADOR */

        /* INICIO Configuracion de CONVERSOR_AD */
        outportb(CTREG, 0x1); /*Elegir canal 1 y fondo de escala +5V*/
        /* FIN Configuracion de CONVERSOR_AD */

        /* DESACTIVAR interrupciones para cambiar la ISR */
        disable();
        printf("\nSe deshabilitaron las interrupciones\n");
        viejotimer = getvect(0x1C); /* guardar direc vieja */
        setvect(0x1C, nuevotimer);  /* setear direc nueva */
        printf("\nHecho el intercambio de direcciones en el vector de
interrupcion\n");
        /* HABILITAR interrupciones */
        enable();

        /* INICIO Programa principal */
        conv = inportb(AD_LOW); /*Iniciar conversión*/
        encenderLed(0);
        apagarLed(1);
        for (;;)
        {
                if (vecesQueEntroAlHandler >= vecesParaUnSegundo)
                {
                        /* E QUE PASO UN SEGUNDO*/
                        /* Reinicio el contador */
                        vecesQueEntroAlHandler = 0;
                        /*Puedo mandar a convertir */
                        do
                        {
```

```
                                conv = inportb(STINR);
                                conv = conv && 0x20; /*Quedarme con End Of
Conversion
                                        conv = xxxx xxxx
                                        0x20 = 0010 0000
                                        &&   = --!- ----
                                        Marco con ! el bit que me interesa
conocer */
                        } while (conv == 0x0);
                        /*Leer el dato convertido*/
                        ad_high = inportb(AD_HIGH);
                        ad_low = inport(AD_LOW);
                        adValue = ad_high * 256 + ad_low;
                        analogicValue = (float)FE / 4096 * adValue;

                        for (i = 1; i < 30; i++)
                        {
                                printf("\n");
                        }

                        /* analogicValue = 2.7 --> temp = 65°C */
                        /* analogicValue = 3.5 --> temp = 85°C */
                        if (analogicValue >= 2.7)
                        {
                                /* Activar sistema de ventilacion poniendo un
1 en la salida digital N°1 */
                                encenderLed(1);
                                printf("\nVENTILACION ON. %1.3f >> 2.7 V",
analogicValue);
                        }
                        else
                        {
                                apagarLed(1);
                                printf("\nVENTILACION OFF. %1.3f << 2.7 V",
analogicValue);
                        }

                        if (analogicValue >= 3.5)
                        {
                                /* Apagar motor poniendo un 1 en la salida
digital N°0 */
                                apagarLed(0);
                                printf("\nMOTOR OFF. %1.3f >> 3.5 V",
analogicValue);
                        }
                        else
                        {
                                encenderLed(0);
```

```
                                printf("\nMOTOR ON. %1.3f << 3.5 V",
analogicValue);
                        }
                }
        }
        /* FIN Programa principal */
        /* Volver a condiciones originales */
        disable();                  /* desactiva interrupciones para cambiar
la ISR */
        setvect(0x1C, viejotimer);   /* restaura la vieja ISR */
        outportb(PORT_CONFIG, 0x36); /* restaura la velocidad del reloj */
        outportb(PORT_TIMER0, 0xFF); /* cargando el contador 0, con unos */
        outportb(PORT_TIMER0, 0xFF);
        enable(); /* habilita interrupciones */

        return 0;
}

int encenderLed(unsigned char num)
{

        switch (num)
        {
        case 0:
                outportb(OUTBR, 0x8);
                break;
        case 1:
                outportb(OUTBR, 0x9);
                break;
        case 2:
                outportb(OUTBR, 0xA);
                break;
        case 3:
                outportb(OUTBR, 0xB);
                break;
        case 4:
                outportb(OUTBR, 0xC);
                break;
        case 5:
                outportb(OUTBR, 0xD);
                break;
        case 6:
                outportb(OUTBR, 0xE);
                break;
        case 7:
                outportb(OUTBR, 0xF);
                break;
        }
```

```
        if (num < 0 || num > 7)
        {
                printf("\nValor de led invalido para encender");
        }
        return 0;
}

int apagarLed(unsigned char num)
{
        switch (num)
        {
        case 0:
                outportb(OUTBR, 0x0);
                break;
        case 1:
                outportb(OUTBR, 0x1);
                break;
        case 2:
                outportb(OUTBR, 0x2);
                break;
        case 3:
                outportb(OUTBR, 0x3);
                break;
        case 4:
                outportb(OUTBR, 0x4);
                break;
        case 5:
                outportb(OUTBR, 0x5);
                break;
        case 6:
                outportb(OUTBR, 0x6);
                break;
        case 7:
                outportb(OUTBR, 0x7);
                break;
        }

        if (num < 0 || num > 7)
        {
                printf("\nValor de led invalido para apagar");
        }
        return 0;
}
```