

# The VGA Standard

viernes, 11 de julio de 2014

19:43

- DAC to convert RGB into analogic signal.
- 256K of video memory, divided into four 64K maps (bit planes)
- Supports MDA,CGA and EGA
- Supports up to 640 x 480 x 16
- Support text modes up to 720x400x16 and up to 3 fonts.
- Access to the VGA registers and video memory is through the microprocessor.

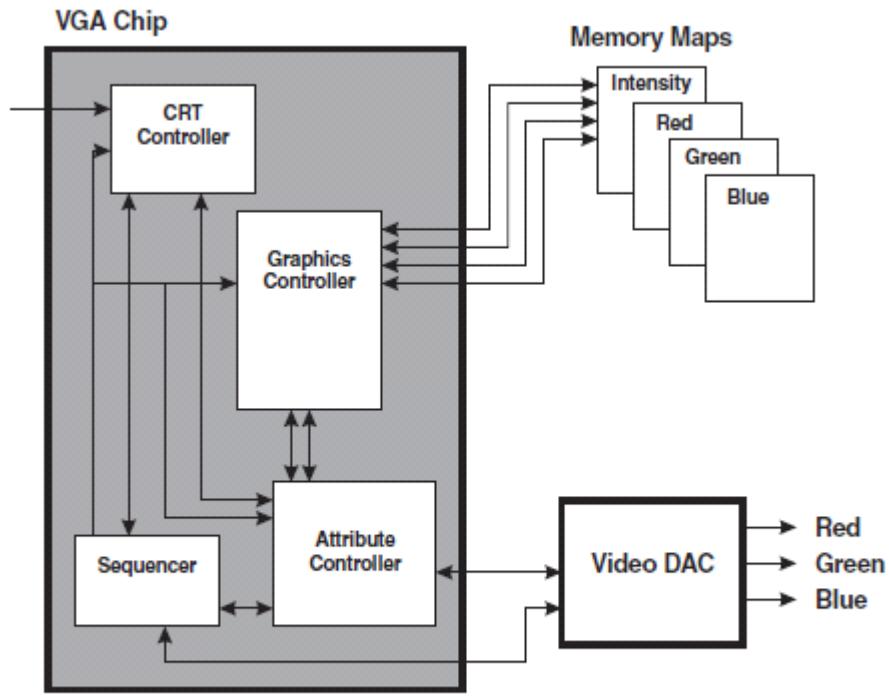
VGA has low performance, as the video operations are performed by the microprocessor.

TABLE OF VIDEO MODES

MODE	COLORS	TYPE	TEXT COLS/ROW	TEXT PIXEL BOX	SCREEN PAGES	BUFFER ADDRESS	SCREEN PIXELS
0,1	16	Alpha	40x25	8x8 8x14* 9x16+	8	B8000H	320x200 320x350 360x400
2,3	16	Alpha	80x25	8x8 8x14* 9x16	8	B8000H	320x200 320x350 360x400
4,5	4	GRA	40x25	8x8	1	A0000H	320x200
6	2	GRA	80x25	8x8	1	A0000H	640x200
7		Alpha	80x28	9x14 9x16	8	B0000H	720x350 720x400
13	16	GRA	40x25	8x8	8	A0000H	320x200
14	16	GRA	40x25	8x8	4	A0000H	640x200
15		GRA	40x25	8x14	2	A0000H	640x350
16	16	GRA	40x25	8x14	2	A0000H	640x350
17	2	GRA	40x30	8x16	1	A0000H	640x480
18	16	GRA	40x30	8x16	1	A0000H	640x480
19	256	GRA	40x25	8x8	1	A0000H	320x200

## VGA COMPONENTS

- VGA Chip
- Video Memory
- DAC



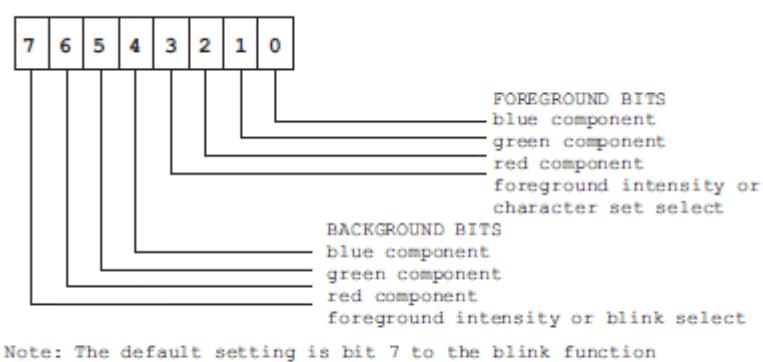
**Figure 7-2 VGA System Components**

Recorte de pantalla realizado: 11/07/2014 19:58

All VGA systems contain the 256K of video memory that is part of the hardware. This memory is logically arranged in four 64K blocks that form the video maps (labeled blue, green, red, and intensity in Figure 7-2). The four maps are sometimes referred to as bit planes 0 to 3.

In the alphanumeric modes 0, 1, 2, 3, and 7 (see Table 7-1) the VGA video buffer is structured to hold character codes and attribute bytes. The organization of the video buffer in the alphanumeric modes was discussed in Part I of this book. The default functions of the bits in the attribute byte can be seen in Figures 1.11 and 1.12. However, the VGA standard allows redefining two of the attribute bits in the color alphanumeric modes:

bit 7 can be redefined to control the background intensity and bit 3 can be redefined to perform a character-set select operation. Figure 7-3 shows the VGA attribute byte, including the two redefinable bits.



**Figure 7-3 Attribute Byte Bitmap in VGA Systems**

The programmer can toggle the functions assigned to bits 3 and 7 of the attribute byte by means of BIOS service calls or by programming the VGA registers. These operations are performed by the VGA graphics library on that is part of the book's software.

## Graphics Modes

One of the problems confronted by the designers of the VGA system was the limited memory space of an IBM microcomputers under MS DOS. Recall that in VGA mode number 18 (see Table 7-1) the video screen is composed of 480 rows of 640 pixels per row, for a total of 307,200 screen pixels. If 8 pixels are encoded per memory byte, each color map would take up approximately 38K, and the four maps required to encode 16 colors available in this mode would need approximately 154K. The VGA designers were able to reduce this memory space by using a latching mechanism that maps all four color maps to the same memory area. Figure 7-4 is a diagram of the video memory structure in VGA mode number 18.

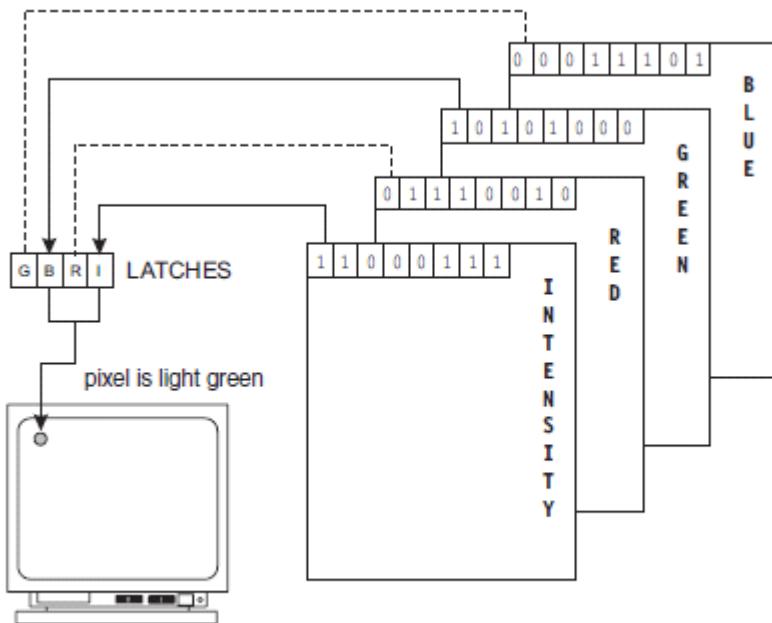
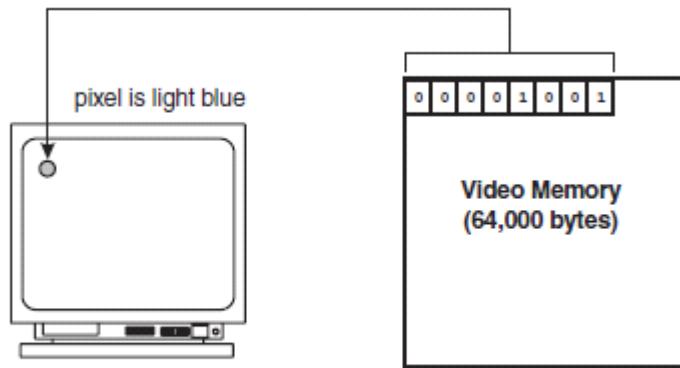


Figure 7-4 Video Memory Mapping in VGA Mode 18

Many VGA graphics modes were created to insure compatibility with previous video systems. Specifically, VGA graphics modes numbers 4, 5, and 6 are compatible with modes in the CGA, EGA, and PCjr; modes numbers 13, 14, 15, and 16 are compatible with EGA; and graphics mode number 17 (a two-color version of mode number 18) was created for compatibility with the MCGA standard. This leaves two proprietary VGA modes: mode number 18 with 640-by-480 pixels in 16 colors, and mode number 19, with 320-by-200 pixels in 256 colors. It is in these two most powerful VGA modes that we will concentrate our attention.





**Figure 7-5** Video Memory Mapping in VGA Mode 19

Recorte de pantalla realizado: 11/07/2014 20:07

# VESA Programming notes

sábado, 17 de mayo de 2014  
20:23

## **Listing 1: VBE wrapper functions**

```
#include <string.h>
#include <dos.h>
#include "vbe.h" /* exports and data types */
/*----- local data -----*/
static VbeInfo_t _VbeInfo;
static ModelInfo_t _ModelInfo;
static int _Width; /* scan line width in bytes */
static long _Window; /* memory window size in bytes */
static int _Segment; /* mem window segment */
/*----- exported functions -----*/
int VbeGetVbeInfo(VbeInfo_t far *p)
/* fetches the vbe info block; returns 0 if no vbe. */
{
    union REGS r={0x4f00,0,0,0,0,FP_OFF(p)};
    struct SREGS s={FP_SEG(p)};
    if(!p) return 0;

    _fmemset(p,0,sizeof(VbeInfo_t));
    _fmemcpy(p->VbeSignature,"VBE2",4);
    int86x(0x10,&r,&r,&s);
    if(_fmemcmp(p->VbeSignature,"VESA",4)) return 0;
    return r.x.ax==0x4f;
}

int VbeGetModeInfo(int mode,ModelInfo_t far *p)
/* fetches the mode info block for the specified mode
   number; returns 0 if mode unsupported by vbe. */
{
    union REGS r={0x4f01,0,mode,0,0,FP_OFF(p)};
    struct SREGS s={FP_SEG(p)};
    if(!p) return 0;

    _fmemset(p,0,sizeof(ModelInfo_t));
    int86x(0x10,&r,&r,&s);
    return r.x.ax==0x4f;
}

int VbeSetMode(int mode)
/* initializes the requested video mode; returns 0 if
   there's no vbe or the mode is unavailable. */
{
    union REGS r={0x4f02,mode};
    if(!VbeGetVbeInfo(&_VbeInfo)) return 0;

    if(mode>=0x100) /* svga mode */
    {
        if(!VbeGetModeInfo(mode,&_ModelInfo) ||
           !(_ModelInfo.ModeAttributes&1)) return 0;

        _Width=_ModelInfo.BytesPerScanLine;
        _Window=1024L*_ModelInfo.WinSize; /* convert to bytes */
    }
}
```

```

    _Segment=_ModelInfo.WinASegment;
}
int86(0x10,&r,&r);

    return r.x.ax==0x4f;
}

void VbeSetPalette(const char far *p,int start,int n)
/* loads the dac palette registers; uses bios on vbe
   versions before 2.0 */
{
    if(_VbeInfo.VbeVersion<0x200) /* use bios */
    {
        union REGS r={0x1012,start,n,FP_OFF(p)};
        struct SREGS s={FP_SEG(p)};
        int86x(0x10,&r,&r,&s);
    }
    else /* use vbe */
    {
        union REGS r={0x4f09,0,n,start,0,FP_OFF(p)};
        struct SREGS s={FP_SEG(p)};
        int86x(0x10,&r,&r,&s);
    }
}

void VbeSetWindow(int window,int position)
/* repositions the indicates memory window to the new
   position (in WinGranularity units). */
{
    union REGS r={0x4f05,window,0,position};
    int86(0x10,&r,&r);
}

void VbeWrite(int x,int y,int bytes,const char far *buffer)
/* copies the contents of the buffer (<64k) to display,
   starting at pixel (x,y). */
{
    long absolute=x+(long)y*_Width; /* absolute offset */
    long position=absolute/_Window; /* of window */
    long offset=absolute%_Window; /* of window */
    char far *vram=MK_FP(_Segment,0); /* to window */
    VbeSetWindow(0,(int)position);

    if(offset+bytes>_Window) /* data overruns window */
    {
        int n=(int)(_Window-offset); /* bytes left */

        _fmemcpy(vram+offset,buffer,n); /* display 1st part */
        VbeSetWindow(0,(int)++position); /* move window */
        _fmemcpy(vram,buffer+n,bytes-n); /* display rest */
    }
    else _fmemcpy(vram+offset,buffer,bytes);/* no overrun */
}
/* End of File */

```

Pegado de <<http://www.drdobbs.com/programming-with-vesa-bios-extensions/184403213?pgno=4>>

# PCX C Functions

jueves, 10 de julio de 2014  
14:45

## Listing 3: PXC file functions

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include "vbe.h" /* vesa vbe interface */
/* ----- Local Data -----*/
/* (Abridged) header record of a PCX file */
typedef struct
{
    char Manufacturer; /* pcx file sig: 10 */
    char Version; /* pc paintbrsh ver */
    char Encoding; /* 1: rle */
    char BitsPerPixel; /* color resolution */
    int Xmin,Ymin; /* image origin */
    int Xmax,Ymax; /* image extent */
    char NotUsed1[53]; /* not used by this
                           program */
    char Planes; /* number of color
                   planes */
    int BytesPerLine; /* (per plane) */
    char NotUsed2[58]; /* by this program */

} pcx_t;
/* Error codes; these also index ErrorText
   array below */
enum
{
    PCX_OK,PCX_EOPEN,PCX_EVIDEO,
    PCX_EREAD,PCX_EFORMAT
};
static const char *ErrorText[]=
{
/* PCX_OK */ "Ok",
/* PCX_EOPEN */ "File not found",
/* PCX_EVIDEO */ "Unsupported video mode",
/* PCX_EREAD */ "File read error",
/* PCX_EFORMAT */
    "Invalid or unsupported format"
};
/* PCX image line buffer */
static char Line[640];
/* DAC palette data */
static char Palette256[768];
/* PCX file header record */
static pcx_t Header;

/*----- local pcx file functions -----*/
static int ReadHeader(FILE *f)
```

```

/* reads the header of the open file into the
global Header structure; returns 0 on a
file read error, or if the file is not a
valid pcx file, or if the pcx file does
not contain a 640x480x256-compatible
image. */
{
    pcx_t *h=&Header;

    fread(h,sizeof(pcx_t),1,f);
    return !(ferror(f)) ||
        h->Manufacturer!=10 ||
        h->Encoding!=1 ||
        h->Planes!=1 ||
        h->BitsPerPixel!=8 ||
        h->Ymax-h->Ymin>479 ||
        h->Xmax-h->Xmin>639);
}
static int ReadPalette(FILE *f)

/* reads the 256-color palette of the open
file into the global buffer Palette256,
and converts it to 6-bit DAC palette
register data; returns 0 on a file read
error or an invalid palette block. */
{
    int i;
    fseek(f,-769L,SEEK_END); /* to palette */
    if(fgetc(f)!=12) /* check sig byte */
        return 0;
    /* read the 256 triplets */
    fread(Palette256,768,1,f);
    if(ferror(f)) return 0;
    fseek(f,128L,SEEK_SET); /* bk to image */
    for(i=0;i<768;i++)
        Palette256[i]>>=2; /* to 6-bit rgb */
    return 1;
}
static void ReadLine(FILE *f)

/* reads and decompresses the next scanline
from the current pcx file into the global
Line buffer. */
{
    int c,i=0;

    while(i<Header.BytesPerLine)
    {
        c=fgetc(f);
        if((c&0xc0)==0xc0) /* is rle hdr */
        {
            c=&~0xc0; /* =repeat count */
            /* copy repeated pixel data */
            memset(Line+i,fgetc(f),c);
            i+=c;
        }
        else Line[i++]=c; /* is pixel data */
    }
}

```

```

    }
    return;
}
/*----- general functions -----*/
int PcxShowFile(const char *file)
/* reads and displays a pcx file; returns 0
   on success, PCX_error code on failure. */
{
    int e=PCX_OK;
    FILE *f=fopen(file,"rb");
    if(!f) e=PCX_EOPEN;
    else if(!ReadHeader(f)) e=PCX_EFORMAT;
    else if(Header.Version==5 &&
           !ReadPalette(f)) e=PCX_EFORMAT;
    else if(!VbeSetMode(0x101)) e=PCX_EVIDEO;
    else
    {
        int y,ymax=min(Header.Ymax,479);
        int n=min(Header.Xmax,640)-
            Header.Xmin+1;
        if(Header.Version==5)
            VbeSetPalette(Palette256,0,256);
        for(y=Header.Ymin;y<ymax;y++)
        {
            ReadLine(f);
            VbeWrite(Header.Xmin,y,n,Line);
        }
    }
    if(f) fclose(f);
    return e;
}
void PcxDeinit()
/* clears the image and resets the video to
   standard 80-column text mode */
{
    VbeSetMode(3);
}
void main(int argc,char **argv)
/* displays the pcx file specified by the
   first argument to the program
   (usage: PCX <filename>) */
{
    int e=PcxShowFile(argv[1]);

    if(!e)
    {
        getch();
        PcxDeinit();
    }
    else puts(ErrorText[e]);
}
/* End of File */

```

Pegado de <<http://www.drdobbs.com/programming-with-vesa-bios-extensions/184403213?pgno=6>>

# Programming With VESA BIOS Extensions

jueves, 10 de julio de 2014

14:46

## Programming With VESA BIOS Extensions

By Chris Krehbiel, August 01, 1996

[Post a Comment](#)

VESA standardizes the software interface to many "super" VGA cards, but nobody said it made programming effortless.

When I first started coding in BASIC on the Apple IIe many years ago, I made a fantasy adventure game called "Fubar's Jubilant Sword," or something. I sat down with a sheet of engineering paper and a pen, marked off the screen dimensions on the page, sketched the graphic for my title screen, and then diligently mapped every little square of the grid that had ink on it to a pixel array in my program -- my very first bitmap.

Things change. Today, a standard VGA-equipped IBM PC can display 256 colors simultaneously, selected from a palette of 16.8 million. It can also achieve a 640x480 pixel screen resolution -- but not while displaying 256 colors. Thus, the standard VGA is a tantalizing beast: if it could only give both color and resolution at once, it could display lifelike images on a 15-inch monitor. But it can't. Fortunately, a host of "Super" VGA cards appeared on the scene, with resolutions of 640x480x256, 800x600x256, and beyond. Photorealism on a PC

was no longer a fantasy. But these new cards, unlike the IBM-standard VGA, varied from one manufacturer to the next. Writing a program with Super VGA graphics meant writing system-level code for every make of video card your software was likely to encounter.

Someone needed to set a standard for SVGA cards, so the Video Electronics Standards Association (VESA) was formed in 1989 to write one. They created the VESA BIOS Extension (VBE), of which the two major releases are VBE 1.2 (1991) and VBE 2.0 (1994). The VBE specifies that manufacturers bundle their cards with driver software that extends the existing BIOS of an IBM PC. This driver provides a set of VESA-defined extended video BIOS system calls to isolate the programmer from the manufacturer-dependent quirks of the hardware. With the VBE, you have a single interface that works no matter who made the video card.

As the VBE catches on, more and more commercial programs using 640x480x256 display resolutions and higher are appearing on the shelves, awing the public with their photorealistic images and surrealistic renderings. Forget those boring 16-color user interfaces and ugly 320x200 games; I'm going to give you a peek under the hood and see how the VBE works, finishing up with a program that will display a photorealistic image on the screen.

### Talking To The VBE

As the term BIOS extension implies, programs access the VBE functions through the IBM PC's BIOS video interface. They make a PC/MS-DOS system call to interrupt 0x10 with the CPU's AH register set to 0x4f, the AL register set to the VBE function number, and any other registers loaded according to the function's requirements. Most of the VBE.C functions ([Listing 1](#)) are little more than shells over these system calls. [Figure 1](#) shows the core function list.

The VBE will return the value 0x004f in the AX register if all went well. Any value other than 0x4f in the AL register means the driver doesn't support the function. AH = 0 represents success, AH = 1 is failure, AH = 2 means the hardware doesn't support the function (VBE 2.0+ only), and AH = 3 means the function is invalid in the current video mode (also VBE 2.0+ only).

### Detecting The VESA VBE

Function 0 (Return VBE Controller Information) is the first function any VESA-aware application should call. The application calls this function to see if the VESA driver is present, which version it is, what modes are available, and so on. The C code to call this function looks like this:

```
union REGS r;
struct SREGS s;
r.x.ax = 0x4f00;
r.x.di = FP_OFF(VbeInfo);
```

```
s.es = FP_SEG(VbeInfo);
int86x(0x10,&r,&r,&s);
```

(The int86x function, or its equivalent, is supplied with PC C compilers to enable programs to call the MS-DOS interrupt functions from C. The REGS and SREGS structures are defined in DOS.H to hold copies of the 80x86 register contents.)

VbeInfo must point to a free block of memory at least 256 bytes long; it must be 512 bytes long if you want the 2.0 extended information as well. The function will treat the block of memory like a VbeInfo\_t structure as shown in VESA.H ([Listing 2](#)). To get the function to return the extended fields, initialize the VbeSignature field to the characters "VBE2" before calling the VBE function

When the call returns, the AX register should equal 0x004f, indicating success, and the VbeSignature field should contain the characters "VESA." There are a few other important fields to know in this structure:

- VbeVersion: This is a BCD (binary coded decimal) value representing the VESA version of the VBE. 0x102 is version 1.2, 0x200 is 2.0, and in general, (unsigned)VbeVersion>>8 yields the major version number, and VbeVersion&0x00ff yields the minor version number.
- VideoModePtr: This is a far (32-bit) pointer to a 16-bit integer array containing the numbers of all the VESA-defined SVGA video modes this driver supports. The last element in the list is always -1 (0xffff).
- TotalMemory: This function sets this field to the number of 64Kb blocks of memory installed on the video card. For instance, if you see a 0x10 in this field, you have  $16 * 64\text{Kb} = 16 * 65,536 = 1\text{Mb}$  of video memory installed. TotalMemory<<6 will yield the memory in Kb, and (unsigned) TotalMemory>>4 yields the memory in Mb.
- OemStringPtr: Upon return this will point to an ASCIIZ (null-terminated) string containing hello-world information about the VBE manufacturer.

The VBE.C function VbeGetVersion forms a wrapper over this system call, returning a Boolean on whether or not the VBE was detected.

### Detecting SVGA Mode Support

The previous function indicated which modes the VGA board would support by setting the VideoModePtr field to point to an array of supported mode numbers. But after obtaining this mode list from the previous function, the application still must check to see if a particular mode from the list really can be used. Function 1 (Return VBE Mode Information) answers this question. The call looks like this:

```
r.x.ax=0x4f01;
r.x.cx=Mode;
r.x.di=FP_OFF(ModelInfo);
s.es=FP_SEG(ModelInfo);
int86x(0x10,&r,&r,&s);
```

The Mode passed into the CX register is one of the mode numbers from the array of supported modes. [Figure 2](#) lists the VESA-defined modes. As with function 0, the memory block pointed to by ModelInfo must be at least 256 bytes long. The VBE function will treat this block as a ModelInfo\_t structure. (This structure is not shown here, but is available electronically. See p.3 for details.)

There's a lot of information in this structure, but here are some of the most important fields:

- int ModeAttributes: Bit 0 (LSB) is set if the requested mode is available, and clear if the hardware doesn't support the mode. Bit 3 is set for color modes, clear for monochrome.
- int Xresolution, Yresolution: These fields indicate the screen resolution in pixels (graphics modes) or character cells (text modes).
- char BitsPerPixel: This field gives the number of bits used to represent a single pixel, or the color resolution of the mode.  $1 << \text{BitsPerPixel}$  yields the number of colors that can be displayed simultaneously on the screen: you get 16 colors with four bits, 256 colors with eight bits, 16.8 million with 24, and so on.
- char MemoryModel: The memory model of the mode refers to the way the video segment is mapped to the display; more on this later.
- int BytesPerScanline: This represents the length of a logical scan line in display memory. (Note that this may or may not be larger than what Xresolution may imply!)
- int WinASegment, WinBSegment: These fields tell which memory segment(s) have been mapped to the video display area.

- int WinGranularity, WinSize: These show the size and granularity of the video segment -- subjects I'll also tackle later on.

Here's a typical example of how a VESA mode can turn up on the mode list but not actually be available: your VBE may support a mode like 1,280x1,024x256 (most do), but this mode needs one byte/pixel \* 1,280 pixels \* 1,024 pixels = 1,310,720 bytes, or 1.25 Mb of display memory (1Mb = 1,048,576 bytes). If your card has only has 1Mb video memory installed, the ModeAttributes bit 0 will be clear because you don't have enough memory to run the mode.

The code disk and online services contain a program CHECK.C that uses the VBE.C functions VbeGetVersion and VbeCheckMode to return information about the VBE on the current machine, and then to report on each of the VESA modes supported by the driver.

### Initializing the VESA Modes

Now that we've determined that the mode we want to use is available, we can use function 2 (Set VBE Mode) to initialize the mode. Calling function 2 is accomplished as follows:

```
r.x.ax=0x4f02;
r.x.bx=Mode;
int86(0x10,&r,&r);
```

This is what the VESA.C function VbeSetMode does. This function can be used to set any of the standard VGA modes as well. If Mode == 3, for example, the code above would restore the computer to text mode.

You get a couple of extras with the VBE function: if you pass Mode|0x8000 into the BX register, you can set the mode without clearing display memory (and the screen); and (for VBE 2.0+ only) if you pass Mode|0x4000 you can initialize the mode to use the linear frame buffer memory model.

### Of Memory Models and Windows

Okay, we can detect the driver, check the video modes, and initialize them; time to plot some high-res SVGA pixels. How, exactly, does a program plot hi-res SVGA pixels? The answer, unfortunately, is "it depends." The modes do have one thing in common: each of them map the display to a "window" in main memory; but there are numerous ways of organizing that map. In other words, different video modes use different memory models.

In this article I'll deal with the most common SVGA mode, 0x101 (640x480x256). This mode uses an 8-bit packed pixel memory model, a fairly easy layout to deal with. Display memory begins at the start of the memory window (typically located at A000:0000), with each consecutive byte mapping to each consecutive pixel, going from left to right, top to bottom. The value contained in the byte is the color number for the pixel. These color numbers reference a palette that can contain 256 entries (the number of possible values in a byte). This is where the term "eight-bit color" comes from -- eight bits is one byte.

Other common memory models are as follows: planar is a counterintuitive layout used by VGA mode 0x12; non-chain 4 256-color (or planar 256) is an even more counterintuitive scheme used in VGA mode X; direct color bypasses the palette and writes the color data itself to the memory window, and is used by 15-, 16-, 24-bit color modes, and up.

The ModelInfo block's MemoryModel field tells which memory model the specified mode uses. Most SVGA graphics modes return either 4 (packed pixel) or 6 (direct color) in this field.

Packed pixel is almost as easy as it gets, but that's a relative assessment: there's still a lot to think about. Remember the ModelInfo fields, WinASegment and WinBSegment? These fields indicate what segment or segments of main memory have been mapped to the display. Typically, WinASegment == 0xa000, the standard video segment for color graphics modes on the IBM PC. In the packed pixel memory model (for mode 0x101), each byte in the video segment corresponds to one pixel on the screen, beginning with pixel (0,0) in the upper-left corner. In general, pixel (x,y) and its corresponding memory offset are related as follows:

$$\begin{aligned} \text{Offset}(x,y) &= \text{Xresolution} * y + x \\ x(\text{Offset}) &= \text{Offset} \bmod \text{Xresolution} \\ y(\text{Offset}) &= \text{Offset} \div \text{Xresolution} \end{aligned}$$

Where Xresolution == 640 in mode 0x101. On an IBM PC, there are 65,536 bytes in any segment, the last one at offset 65,535. But:

$$x(0xffff) = 65,535 \bmod 640 = 255$$

$$y(0xffff) = 65,535 \div 640 = 102$$

So you can only plot pixels from (0,0) to (255,102) before overrunning the video segment, even

though the mode 0x101 screen goes all the way to (639,479). To plot pixels beyond (255,102) the memory window must be remapped to a different part of the video display area. The function that remaps the display is function 5 (VBE Display Window Control), subfunction BH = 0, Set Memory Window. (Subfunction BH = 1 fetches the current window position in the DX register.) Here's how you call function 5:

```
r.x.ax=0x4f05;
r.h.bh=0;
r.h.bl=Window;
r.x.dx=Position;
int86(0x10,&r,&r);
```

The BL register tells the VBE which window to remap: BL = 0 means window A, and BL = 1 means window B. The window position passed into the DX register must be in WinGranularity units. WinGranularity is the smallest increment (in Kb) by which a memory window can be shifted. If WinGranularity == 64, then the memory window can start at the first pixel or the 65,536th, but not between them.

The ugly part of all this is that when we move the window from position 0 to position 1, offset 0 in the video segment now corresponds to pixel (256,102), which isn't even the first pixel in the scan line. We need to rethink the relationship between pixel coordinates, window positions, and memory offsets. It comes down to this:

$$\begin{aligned} \text{Offset}(x,y) &= (x+y*\text{Xresolution}) \bmod \text{WinGranularity} \\ \text{Position}(x,y) &= (x+y*\text{Xresolution}) \div \text{WinGranularity} \\ x(\text{Offset},\text{Position}) &= (\text{Position} * \text{WinGranularity} + \text{Offset}) \bmod \text{Xresolution} \\ y(\text{Offset},\text{Position}) &= (\text{Position} * \text{WinGranularity} + \text{Offset}) \div \text{Xresolution} \end{aligned}$$

Assuming Xresolution == 640, WinASize == 64Kb, and WinGranularity == 64Kb, then it takes about four and a half windows to cover the whole screen:

- Position 0: (0,0) to (255,102)
- Position 1: (256,102) to (511,204)
- Position 2: (512,204) to (127,307)
- Position 3: (128,307) to (383,409)
- Position 4: (384,409) to (639,479)

One more thing: the ModelInfo block contains a field called WinFuncPtr. WinFuncPtr is a far pointer to the window control function (function 5) just discussed; it's there because calling it this way is faster than calling the function through the VBE interface. Because its arguments are passed in the CPU registers and not pushed on the stack, it must be called from the inline assembler:

```
_asm
{
    sub bh,bh      ;subfunction bh=0: set memory window
    mov bl,Window  ;bl=window a or b
    mov dx,Position ;dx=position in WinGranularity units
    call WinFuncPtr ;call window control function directly
}
```

Note that this code didn't have to load the AX register the VBE function number. We don't need it here because WinFuncPtr already references function 5 directly. The VESA.C function VbeSetWindow uses the interrupt version for clarity, but when speed counts, use the direct call instead.

### Setting the DAC Palette Registers

I mentioned earlier that in an eight-bit, packed pixel mode like VESA mode 0x101, the value of the byte written to memory dictates the color of the pixel on the screen. What happens, roughly, is this: this byte gets fed to the video card's DAC (digital-to-analog converter) chip, which uses it to index a table of 256 RGB triplets (the red, green and blue components that actually make up the color). The chip converts this data to an analog signal that the video monitor can use.

This table, called the DAC color register set or the 256-color palette, can be and frequently is preloaded with a custom color map before anything is drawn to the screen. A photographic-quality

256-color image achieves the smooth gradations in hue and shadow by loading all 256 palette entries with the colors it needs most. A program can load the palette through the VBE or through the BIOS. Since VBE function 9 (Load/Unload Palette Data) only exists in VESA version 2.0+, I'll mention both methods. There isn't much difference between them anyway.

The code to load Count palette registers beginning at register offset First looks like this, using function 9:

```
r.x.ax=0x4f09;  
r.h.bl=0;  
r.x.cx=Count;  
r.x.dx=First;  
r.x.di=FP_OFF(PaletteData);  
s.es=FP_SEG(PaletteData);  
int86x(0x10,&r,&r,&s);
```

The BL register specifies subfunction 0, Load Palette Data. The PaletteData array needs to be at least 3\*Count bytes long, since the data for each entry is a triplet of red, green, and blue components, in that order. If you set all 256 palette entries, Count == 256, First == 0, and sizeof(PaletteData) == 768. The standard BIOS version of loading the palette uses video function 0x1012:

```
r.x.ax=0x1012;  
r.x.bx=First;  
r.x.cx=Count;  
r.x.dx=FP_OFF(PaletteData);  
s.es=FP_SEG(PaletteData);  
int86x(0x10,&r,&r,&s);
```

So the VBE and standard BIOS techniques are pretty close. The VESA.C function VbeSetPalette will use the VBE call under 2.0, and the BIOS call under prior versions.

### The Moment of Truth

Check out PCX.C ([Listing 3](#)) . This (very) scaled-down program uses the VESA.C functions to display a 640x480x256 (mode 0x101) PCX image file. I've extracted this code out of a more comprehensive PCX file viewer I wrote that handled CGA, EGA, MCGA, VGA, as well as eight-, 16-, and 24-bit SVGA pictures. I've tried to make this example as small and concise as I could, so this program will only handle PCX files that can be displayed in mode 0x101.

A quick word about the PCX file format: it was developed by Zsoft Corporation for their PC Paintbrush software, and was a de facto standard for a long time. Today it's being displaced by the GIF (which offers better 256-color compression) and JPEG (which offers excellent true color compression) standards. Though PCX is a poor choice of formats for SVGA images, it is the simplest. A PCX file begins with a header record, which I've typedef'd as ppx\_t. I've abridged the structure of the block a bit to make it as self-explanatory as possible; essentially, the PCX file will have Manufacturer == 10 (i.e., a PCX file), Encoding == 1 (run-length encoded), Planes == 1 (packed pixel), BitsPerPixel == 8 (256 colors), Xmax-Xmin<640, and Ymax-Ymin<480 (no larger than 640x480 pixels). The PCX.C function ReadHeader checks for these values in the header to ensure the viewer can display the requested file.

If the header's Version field is 5, a 256-color custom palette resides at the end of the file. (Just about every 256-color PCX file will have its own palette). The palette record consists of the signature byte 0x0c followed by a 768-byte block of palette RGB data. This block is similar to what gets passed into VBE function 9 (LoadPaletteData), with one exception: the DAC uses only 6-bit RGB components, and the PCX palette contains 8-bit components. To convert from a PCX palette component to a DAC palette component, the viewer just drops the two least significant bits with a right shift. The PCX.C function ReadPalette shows how to read and translate the palette data.

The compressed image lies between the header and the palette data. Decoding the data is simple: fetch a byte and check the two most significant bits. If they're not both set, then the byte represents a single pixel value. If both bits are set, then the low six bits represent the number of following consecutive pixels that take the value of the next byte fetched. This is a simple form of run length encoding. The PCX.C function ReadLine demonstrates the decoding algorithm.

The viewer's logic is pretty straightforward. The program opens the requested file, verifies the header, and translates the palette data. Then it sets the video mode and loads the palette into the

DAC. Finally, the viewer enters the read/display loop, which decompresses one line of the image into a buffer and then copies the buffer to the display. When the viewer is done, it uses the VBE to set the video mode back to text mode (VGA mode 3). The code is not meant to be terribly efficient, just illustrative. (When I yanked the code from my file viewer I did a lot of de-optimizing in favor of simplicity.)

### **Is That All?**

Heck, no. I've ignored a bunch of stuff in order to nail down the basics without writing a book. There's plenty of territory left to explore: linear frame buffers and DPMI; other memory models and color models; multiple window segments, scrolling displays, and page flipping...you get the idea. Graphics technology is a fleet-footed beast. Somewhere in my garage, buried at the bottom of a disintegrating cardboard box, on a crumbling sheet of engineering paper, is my very first bitmap. Big pixels. No color. But back then it added a touch of class to "Fubar's Jubilant Sword."

Or whatever the game was called. It's been about 10,000 years since then.

### **Standards Information**

Programmer's Toolkit: VBE/Core 2.0. Video Electronics Standards Association. This is the most current version of the official VBE standard, can be purchased from VESA (Voice: 408-435-0333). Zsoft PCX File Format Technical Reference Manual, Revision 5. Zsoft Corporation. This is available from Zsoft's BBS at 404-427-1045 (9600/2400,N81). Also available from Zsoft is the file SHOW\_PCX.PAS, a Pascal program that displays CGA, EGA, MCGA, and VGA files.

*Chris Krehbiel is a systems programmer with Logicon Syscon Corporation. He has a degree in computer science from Thomas Edison State College, and lives in Williamsburg, Virginia. He tinkers with games and graphics and can be reached at ckrehbiel@logicon.com.*

Pegado de <<http://www.drdobbs.com/programming-with-vesa-bios-extensions/184403213?pgno=1>>

# VESA - INT 10 LIST

jueves, 10 de julio de 2014

14:48

## Figure 1: VESA VBE Function List (INT 10H)

AX=0x4f00 Return VBE Controller Information  
AX=0x4f01 Return VBE Mode Information  
AX=0x4f02 Set VBE Mode  
AX=0x4f03 Return Current VBE Mode  
AX=0x4f04 Save/Restore State

DL=0: Return save/restore buffer size  
DL=1: Save state  
DL=2: Restore state  
AX=0x4f05 VBE Display Window Control

BH=0: Set memory window  
BH=1: Get memory window  
AX=0x4f06 VBE Set/Get Logical Scan  
Line Length

BL=0: Set Scan Line Length In Pixels  
BL=1: Get Scan Line Length  
BL=2: Set Scan Line Length In Bytes  
BL=3: Get Maximum Scan Line Length  
AX=0x4f07 VBE Set/Get Display Start Control

BL=0: Set Display Start  
BL=1: Get Display Start  
BL=0x80: Set Display Start  
During Vert.Retrace

AX=0x4f08 VBE Set/Get DAC Palette Format

BL=0: Set DAC Palette Format  
BL=1: Get DAC Palette Format  
AX=0x4f09 Load/Unload Palette Data  
(2.0+ only)

BL=0: Set Palette Data  
BL=1: Get Palette Data  
BL=2: Get Secondary Palette Data  
BL=3: Set Secondary Palette Data  
BL=0x80: Set Palette Data During Vertical Retrace

AX=0x4f0a VBE 2.0 Protected Mode Interface  
(2.0+ only)

BL=0: Return Protected Mode Table

VESA VBE Function Status Word (AX)

AH Result:

0: success  
1: failure  
2: unsupported in hardware (2.0+ only)  
3: invalid in current video mode  
(2.0+ only)

AL Support:

0x4f: function supported in VBE  
Other: function unsupported

**Figure 2: VESA-defined Super VGA Modes**

Number	Resolution
0x100	640x480x256
0x101	640x480x256
0x102	800x600x16
0x103	800x600x256
0x104	1024x768x16
0x105	1024x768x256
0x106	1280x1024x16
0x107	1280x1024x256
0x108	80x60x16
0x109	132x25x16
0x10A	132x43x16
0x10B	132x50x16
0x10C	132x60x16
0x10D	320x200x32K
0x10E	320x200x64K
0x10F	320x200x16M
0x110	640x480x32K
0x111	640x480x64K
0x112	640x480x16M
0x113	800x600x32K
0x114	800x600x64K
0x115	800x600x16M
0x116	1024x768x32K
0x117	1024x768x64K
0x118	1024x768x16M
0x119	1280x1024x32K
0x11A	1280x1024x64K
0x11B	1280x1024x16M

Pegado de <<http://www.drdobbs.com/programming-with-vesa-bios-extensions/184403213?pgno=3>>

# Vesa C structures & Drawing

jueves, 10 de julio de 2014

14:49

## Whats VESA?

VESA is a standard which provides programmers with a single method for using many different video cards. With VESA you need not worry about the specific hardware in different cards, you only need to understand the VESA functions. Anyone old enough will remember having to install a VESA driver for their video card, these days it's implemented in hardware though - so that's another thing you need not worry about. We **highly recommend** that anyone attempting this tutorial gets a copy of the official VESA 2.0 specification. This tutorial should get you up and running even if you don't understand a word of it though!

## Vesa Infomation Structures

To begin with you need 2 structures to store information returned to you by the video card. Just copy ours. VESA\_INFO will store information about your video card. MODE\_INFO will hold information about the graphics mode you are using. The use of many fields should be fairly obvious:

```
typedef struct VESA_INFO
{
    unsigned char VESASignature[4] __attribute__ ((packed));
    unsigned short VESAVersion __attribute__ ((packed));
    unsigned long OEMStringPtr __attribute__ ((packed));
    unsigned char Capabilities[4] __attribute__ ((packed));
    unsigned long VideoModePtr __attribute__ ((packed));
    unsigned short TotalMemory __attribute__ ((packed));
    unsigned short OemSoftwareRev __attribute__ ((packed));
    unsigned long OemVendorNamePtr __attribute__ ((packed));
    unsigned long OemProductNamePtr __attribute__ ((packed));
    unsigned long OemProductRevPtr __attribute__ ((packed));
    unsigned char Reserved[222] __attribute__ ((packed));
    unsigned char OemData[256] __attribute__ ((packed));
}VESA_INFO;
typedef struct MODE_INFO
{
    unsigned short ModeAttributes __attribute__ ((packed));
    unsigned char WinAAttributes __attribute__ ((packed));
    unsigned char WinBAttributes __attribute__ ((packed));
    unsigned short WinGranularity __attribute__ ((packed));
    unsigned short WinSize __attribute__ ((packed));
    unsigned short WinASegment __attribute__ ((packed));
    unsigned short WinBSegment __attribute__ ((packed));
    unsigned long WinFuncPtr __attribute__ ((packed));
    unsigned short BytesPerScanLine __attribute__ ((packed));
    unsigned short XResolution __attribute__ ((packed));
    unsigned short YResolution __attribute__ ((packed));
    unsigned char XCharSize __attribute__ ((packed));
    unsigned char YCharSize __attribute__ ((packed));
    unsigned char NumberOfPlanes __attribute__ ((packed));
    unsigned char BitsPerPixel __attribute__ ((packed));
    unsigned char NumberOfBanks __attribute__ ((packed));
    unsigned char MemoryModel __attribute__ ((packed));
    unsigned char BankSize __attribute__ ((packed));
    unsigned char NumberOfImagePages __attribute__ ((packed));
    unsigned char Reserved_page __attribute__ ((packed));
```

```

unsigned char RedMaskSize __attribute__ ((packed));
unsigned char RedMaskPos __attribute__ ((packed));
unsigned char GreenMaskSize __attribute__ ((packed));
unsigned char GreenMaskPos __attribute__ ((packed));
unsigned char BlueMaskSize __attribute__ ((packed));
unsigned char BlueMaskPos __attribute__ ((packed));
unsigned char ReservedMaskSize __attribute__ ((packed));
unsigned char ReservedMaskPos __attribute__ ((packed));
unsigned char DirectColorModelInfo __attribute__ ((packed));
unsigned long PhysBasePtr __attribute__ ((packed));
unsigned long OffScreenMemOffset __attribute__ ((packed));
unsigned short OffScreenMemSize __attribute__ ((packed));
unsigned char Reserved[206] __attribute__ ((packed));
}MODE_INFO;

```

Pegado de <<http://www.retro-games.co.uk/downloads/progvesa.htm>>

### Testing for VESA

We will use the VESA\_INFO structure when we first check for valid VESA infomation with this function:

VESA\_INFO vesa\_info;//info will be stored in here

```

int get_vesa_info()
{
    __dpmi_regs r;
    long dosbuf;
    int c;
    dosbuf = __tb & 0xFFFF;
    for (c=0; c<sizeof(VESA_INFO); c++)
        _farpokeb(_dos_ds, dosbuf+c, 0);
    dosmemput("VBE2", 4, dosbuf);
    r.x.ax = 0x4F00;
    r.x.di = dosbuf & 0xF;
    r.x.es = (dosbuf>>4) & 0xFFFF;
    __dpmi_int(0x10, &r);
    if (r.h.ah)
        return -1;
    dosmemget(dosbuf, sizeof(VESA_INFO), &vesa_info);
    if (strcmp(vesa_info.VESASignature, "VESA", 4) != 0)
        return -1;
    return 0;//OK
}

```

Here we are using the dos interrupt 10 Hex. All VESA funtion are called though this combined with a value loaded into the AX register. This function returns zero if valid VESA info is found.

### Setting VESA modes

Once you have validated VESA compatability you are ready to set and check for the various resolutions available:

```

int set_vesa_mode(int mode)
{
    __dpmi_regs r;
    r.x.ax = 0x4F02;
    r.x.bx = mode;
}

```

```

__dpmi_int(0x10, &r);
if (r.h.ah)
return -1;
return 0;
}

int get_vesa_mode()
{
__dpmi_regs r;
r.x.ax = 0x4F03;
__dpmi_int(0x10, &r);
if (r.h.ah)
return -1;
return r.x.bx;
}

```

The following get\_mode\_info function can be used in conjunction with these to get specific details about a mode:

```

int get_mode_info()
{
__dpmi_regs r;
long dosbuf;
int c;
dosbuf = __tb & 0xFFFF;
for (c=0; c<sizeof(MODE_INFO); c++)
_farpokeb(_dos_ds, dosbuf+c, 0);
r.x.ax = 0x4F01;
r.x.di = dosbuf & 0xF;
r.x.es = (dosbuf>>4) & 0xFFFF;
r.x.cx = 257;
__dpmi_int(0x10, &r);
if (r.h.ah)
return 0;
dosmemget(dosbuf, sizeof(MODE_INFO), &mode_info);
return 1;
}

```

It may be worth going through [this example](#) which includes calls to all the code so far.

### Drawing

Initially you may draw to a buffer as in the VGA examples earlier in this tutorial. When we come to display that buffer there are new considerations. The video memory for high resolutions is split into a number of banks (the reason for this lies in limitations of PC architecture). The 640\*480 buffer will have to be copied in (probably 4) chunks. The following code will do this:

```

void copy_to_vesa_screen(char *memory_buffer)
{
int bank_number=0;
int todo=307200;
int copy_size;
while (todo>0)
{
set_vesa_bank(bank_number);
if (todo>65536)
copy_size=65536;
else
copy_size=todo;
}

```

```
dosmemput(memory_buffer, copy_size, 0xA0000);
todo-=copy_size;
memory_buffer+=copy_size;
bank_number++;
}
}
```

Note the use of set\_vesa\_bank function detailed here:

```
void set_vesa_bank(int bank_number)
{
    __dpmi_regs r;
    r.x.ax = 0x4F05;
    r.x.bx = 0;
    r.x.dx = bank_number;
    __dpmi_int(0x10, &r);
}
```

Pegado de <<http://www.retro-games.co.uk/downloads/progvesa.htm>>

# The Super VGA Graphics Card

jueves, 10 de julio de 2014

14:51

The Super VGA Graphics Card is a step up from the plain old VGA. It offers both increased resolution and in some modes, a larger number of colours on screen at once. If you are programming in Real Mode on a PC, it can be a bit inconvenient, however, since I have never done any programming in Protected Mode, you will have to wait till I can ask my friend about programming it in a flat memory model.

This document describes VESA specification version 1.2.

## The basics:

Resolution	Colours	Memory Required
640 x 400	256	250Kb
640 x 480	256	300Kb
800 x 600	256	469Kb
1024 x 768	256	768Kb
1280 x 1024	256	1.28Mb
640 x 400	65536	500Kb
640 x 480	65536	600Kb
800 x 600	65536	930Kb
1024 x 768	65536	1.5Mb
1280 x 1024	65536	2.5Mb
640 x 400	16.7 million	750Kb
640 x 480	16.7 million	900Kb
800 x 600	16.7 million	1.4Mb
1024 x 768	16.7 million	2.25Mb
1280 x 1024	16.7 million	3.75Mb

The VESA specification of the SVGA graphics card has many different modes available. As you can see from the table, some of these modes require quite a lot of graphics memory.

There are many different chipsets used by graphics cards, which would make it very inconvenient to program for them all. The VESA specification makes it easy to program for any graphics card which supports it, and most do. You can quickly and easily identify the make of the card, how much memory it has, and what modes it is capable of.

You can ask the BIOS for all this information by calling interrupts, as well as set any graphics mode.

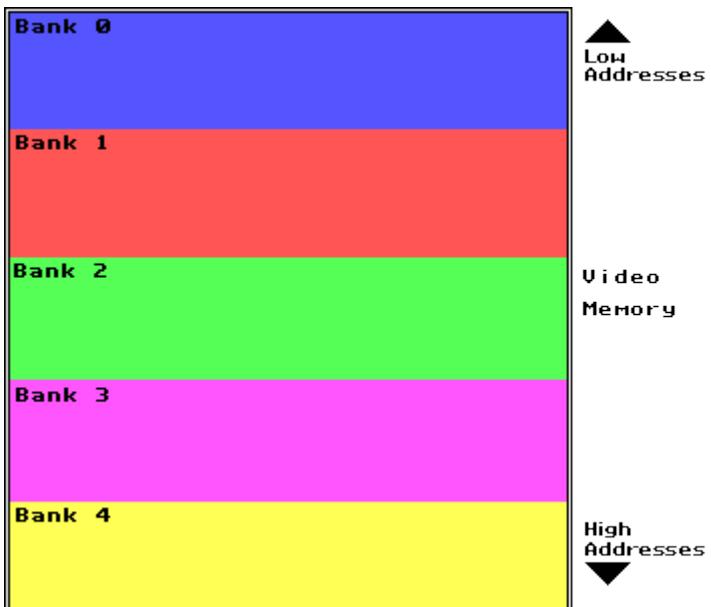
The other thing I have found is that the BIOS does not seem to support printing text to screens with more than 256 colours. This means that you will have to write your own routines to do this trivial task. I will try and explain how to do this later.

Be warned. Programming the SVGA card requires some knowledge of interrupts, data structures, and some assembler can really help.

I will start by explaining the structure of the video memory when in the various modes. I suggest you have a look at the page on programming the VGA card if you are not familiar with it.

## Memory Banks:

One thing you may have noticed is that, although the graphics modes require several hundred Kb of memory, the PC only lets you write to 64Kb of graphics memory, starting at segment A000h. This is of course a problem, and is solved by bank switching.



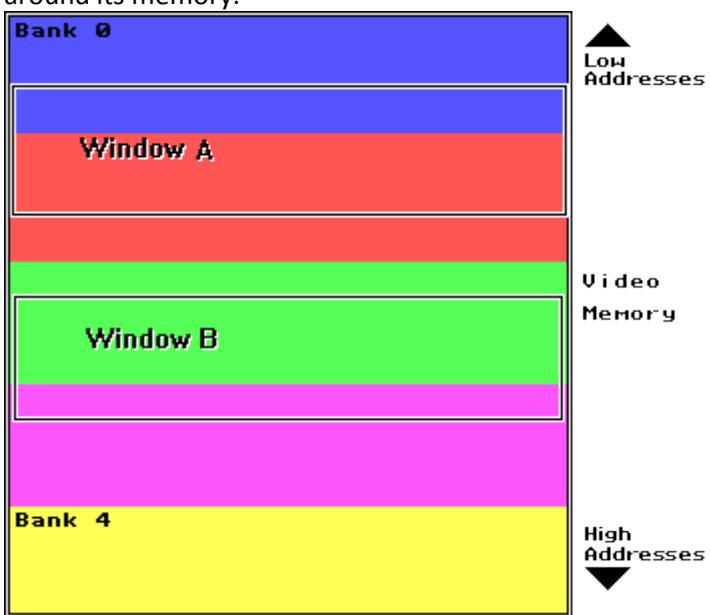
The graphics memory is split up into several banks of 64Kb each. To access some graphics memory, you first need to tell the graphics card which bank you want access to.

The diagram shows memory split up into discrete banks (ie the banks are 65536 bytes apart). However, this is not always the case. Some graphics cards allow you to address banks 4096 bytes apart. I will explain this in more detail later, but all you need to know for now is that you cannot address all the memory at once.

One more important thing to keep in mind is that the end of a bank does not always come at the end of a scanline. This can be inconvenient, but there are easy and efficient ways to get round the problem.

## Windowing

This has nothing to do with Microsoft Windows. This is how you access the banks. The graphics card lets you access different portions of its memory by allowing you to move a window around its memory.



The BIOS may provide you with up to 2 windows, called A and B. They may be either readable or writable or both. Often, though, one will be readable, the other writable. You can move these windows around the memory. The picture shows them in arbitrary places in memory, but often, you will only be able to move them in 4Kb or 64Kb steps.

This is, of course, very annoying if you want to draw something very big to the screen, or something that crosses a window boundary. In these cases, you will have to draw it in more than one step. i.e. draw the first part, then move the window, then draw the next part.

This gets even more inconvenient if you want to draw large circles. An optimised circle drawing routine will plot eight pixels at once. This is not possible in this case, because, chances

are, the pixels will be in different windows.

The end of a window will not fall at the end of a scanline unless the screen width is a power of 2. Again, this is seriously pissing off when drawing sprites. I will explain about getting round this later.

## 256 colour modes (8 - bit):

These modes are very similar to the normal VGA 256 colour mode. Every pixel on the screen is represented by one byte. This allows 256 colours to be displayed on the screen at once, selected from a palette of 262,144.

## 32768 colour modes (15 - bit):

There is a real difference between the 8 bit and 15 bit modes. The 15 bit mode allows you to directly specify the red, green and blue content of every pixel. You no longer have to set up a palette, and you are not restricted to a limited selection of 256 colours.

Although this is called a 15 bit mode, every pixel has 2 bytes assigned to it for convenience, and the most significant bit (bit 15) is unused.

-	R	R	R	R	R	G	G	G	G	B	B	B	B	B
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Each colour is assigned 5 bits. So every pixel can have a choice of 32 levels of red green and blue. Giving a total of 32768 colours.

## 65536 colour modes (16 - bit):

This is the same as the 15 bit mode except for the fact that the green value is assigned one more bit.

R	R	R	R	R	G	G	G	G	G	B	B	B	B	B
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Giving 65536 colours.

## 16.7 million colour modes (24 - bit):

In this mode, each pixel is assigned 3 bytes. One each for red, green and blue. This lets you specify 256 levels of each, giving a total of 16777216 different colours. Thats more colours than there are pixels on screen.

## Getting Started:

Ok, here I will explain how to go about setting up the data structures and calling the interrupts necessary to set up the SVGA card. I am going to have to assume that you know about structures and interrupts because this is not really the place to explain such things.

[Extended VGA BIOS Interrupt list](#)

## An Example Program

Right, now you know all that, (don't bother learning it all off by heart), Its time to see an example of all this in action, as it were.

This example application will be a bare-bones kind of thing. Not really doing anything useful. I will write most of it in English and pseudo code, with some small bits of it in Assembler maybe. For clarity, I am going to split the program up on several different pages, listed below.

### What will this program do?

0. [Data Structures for the application.](#)
1. [Detect to see if a VESA BIOS is present.](#)
2. [Look for all the colour modes, and ask the user to choose a resolution.](#)
3. [Save the state of the video card.](#)
4. [Switch to that resolution.](#)
5. [Set the virtual screen to be twice the screen width.](#)
6. [fill the screen with random pixels.](#)
7. [Scroll and pan around the screen a bit.](#)
8. [Restore the screen to its former state.](#)

# Some of the finer points

I'm sitting here in the computer room, writing this document, and listening to Radio 1. (that's an English radio station for those of you on far away bits of this planet). They're playing the Alabama 3, Live! If you haven't heard them they are so cool. Kind of Acid Blues, really mellow, but funky at the same time. There's nothing like it.

Enough of the useless banter. On with the information. OK, so now you know how to setup and use a SVGA screen. That's not all there is to it though. As they say, it's not what you do, its how you do it. There are many ways to use the above functions and procedures. I have't really had much practice with this, so my advice will be limited. No doubt many of you will be able to figure out far superior methods and algorithms, but here are a few tips.

## 1. Avoid Bank Switching

ok, so you're not surprised. Bank switching takes time. If you are going to be writing lots of individual pixels to the screen, say in a star field, there are many ways to avoid excessive bank switching.

Try to write all the pixels in one bank before switching to the next. ie, write all those pixels in bank 0, then switch to bank 1 and write all the pixels in that bank.

If this is not possible, then keep a record of the current bank and don't switch if you don't need to.

The VESA specification says this:

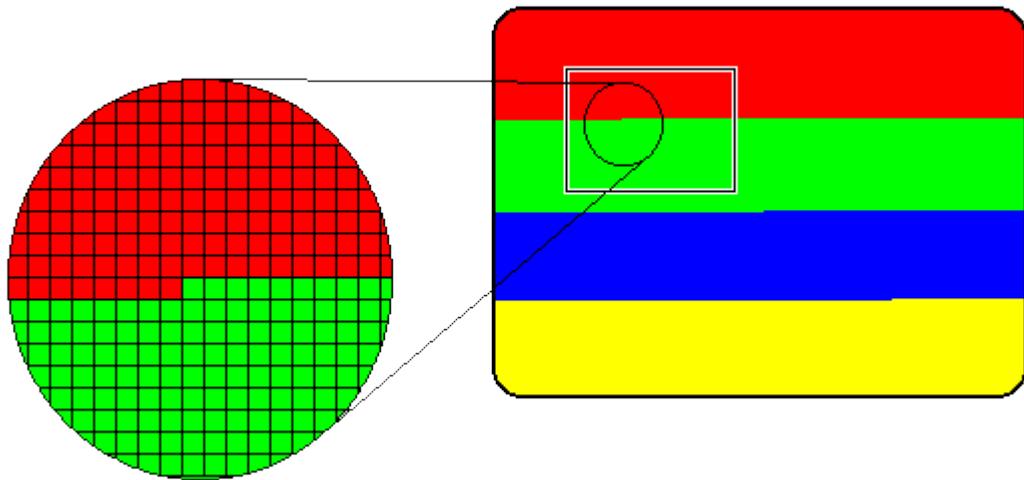
*The organization of most software algorithms which perform video operations consists of a pair of nested loops: and outer loop over rows or scan lines and an inner loop across the row or scan line. The latter is the proverbial inner loop, which is the bottle neck to high performance software.*

*If a target rectangle is large enough, or poorly located, part of the required memory may be within the video memory mapped into the CPU address space and part of it may not be addressable by the CPU without changing the mapping. It is desirable that the test for remapping the video memory is located outside of the inner loop.*

*If the granularity is equal to the length of the CPU address space, i.e. the least significant address bit of the hardware mapping function is more significant than the most significant bit of the CPU address, then the inner loop will have to contain the test for crossing the end or beginning of the CPU address space. This is because if the length of the CPU address space (which is the granularity in this case) is not evenly divisible by the length of a scan line, then the scan line at the end of the CPU address will be in two different video memory which cannot be mapped into the CPU address space simultaneously.*

OK, so that's a bit of a mouthfull. However what they say is only partly true. Indeed, if the end of a bank occurs in the middle of a scan line, it can make things a little bit inconvenient, but it should never be necessary to check if the end of the bank has been reached after every pixel write.

It may be a little hard to visualise exactly what is happening here, so I think a picture is in order.



Right, so the big stripy thing is meant to be a representation of the screen, showing the banks. If you look closely, you can see that the end of the banks fall somewhere in the middle of the scanlines.

Now, lets say that we wanted to fill in that rectangle which falls right over that bank boundary. The VESA specification reckons that, as we plot each pixel, we have to check to see if we have crossed into the next bank, and, if so, switch.

This is not so. Imagine you were the rectangle filling routine, you personally. And you were going to fill in the rectangle. You would be smart about it wouldn't you? So the algorithm can be made smart.

So the filling would be done a scanline at a time. Now, if the end of the bank does not occur in any particular scanline, then there is absolutely no point in checking while it is being drawn. If it does occur, then why not fill up to the bank boundary, switch bank, then continue filling.

A good way to keep track of all these banks is to keep an array specifying which bank each scanline is in, and, if it crosses a bank boundary, at which X-coordinate this happens.

The array should have as many elements as there are scan lines in the entire virtual screen. When you are writing to the screen, a scan line at a time, you need two inner loops, one to handle lines which do not cross a boundary, and one which handles lines which do.

The format of this array would be as follows:

```
structure BankArray
    integer BankNum          ; The bank the first pixel on the scanline
    ; resides in
    integer Boundary         ; the last pixel in the scan line which
    ; resides in that bank. If there is no boundary
    ; then set this value to the Logical screen width
end structure
array BankArray(0 to NumScanLines-1)
```

um, I think an example would be in order:

OK, so i'll take the standard example. The screen is in 640x480 256 colour mode. The banks are 65536 bytes, and the window granularity is also 65536 bytes, so each bank occupies 102.4 scanlines.

The first 102 scanlines reside entirely inside bank 0.

BankArray.Banknum(i) = 0

BankArray.Boundary(i) = 0 ;0 meaning that no bank change happens

The 103rd scanline (number 102) crosses a bank boundary. Its 257th pixel (number 256) resides in bank 1:

BankArray.Banknum(102) = 0

BankArray.Boundary(102) = 255 ;last pixel on this line in bank 0

Before the filling algorithm writes to a scanline, it should check to see if either of the two X coordinates are greater than BankArray.Boundary(i).

If they are not, then fill normally.

If they are both larger, then switch to the next bank and fill normally.

If they are on either side of the boundary, then it should fill up to the boundary, switch bank, and continue the rest of the fill.

I think it's time for a little pseudocode to make things clearer. I can't guarantee that this code works. I just wrote it off the top of my head. Please tell me if you spot a mistake.

```

Procedure FillRectangle (x, y, xszie, ysize, colour)
    ScreenPointer = y * LogicalWidth + x
    SwitchBank(BankArray.BankNum(y))
    loop y from y to (y+ysize-1)
        ; is there a bank boundary on this line ?
        if BankArray.BankBoundary(y) > 0 then
            ; does it happen before the rectangle ?
            if BankArray.BankBoundary(y) <= x then
                switch bank to BankArray.BankNum(y)
                ScreenPointer = x - BankArray.BankBoundary(y)
                ; draw this line
                write xszie bytes to the screen
            end if
        ; does it happen in the rectangle ?
        if (BankArray.BankBoundary(y) > x) and (BankArray.BankBoundary(y) < x+xsize-1) then
            ; draw the first part of this line
            write (BankArray.BankBoundary(y)-x) bytes to the screen
            switch bank to BankArray.BankNum(y)
            ScreenPointer = 0                                ; the begining of the new bank
            ; draw the rest of this line
            write xszie-(BankArray.BankBoundary(y)-x) bytes to the screen
            ScreenPointer = ScreenPointer + 640-xszie      ; next line
        end if
    ; does it happen after the rectangle ?
    if BankArray.BankBoundary(y) > x+xsize-1 then
        ; draw this line
        write xszie bytes to the screen
        switch bank to BankArray.BankNum(y)
        ScreenPointer = ScreenPointer + 640-xszie - 65536
    end if
    ; No bank boundary on this line
    else
        ; otherwise, just keep drawing
        write xszie bytes to the screen          ; write the colour
        ScreenPointer = ScreenPointer + 640-xszie ; next line
    end if
end loop
end FillRectangle
```

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_svga.htm](http://freespace.virgin.net/hugo.elias/graphics/x_svga.htm)>

# Page 3: Save The State of the Video Mode

jueves, 10 de julio de 2014

14:52

Right, now this isn't always necessary, and there's no need to bother doing this unless you really need to. For example if you were writing a popup TSR or something. However, most programs will just want to drop back into DOS in mode 3.

What has to be done here is to ask the graphics card how much memory it would require to save its current state. I.E. its resolution, virtual screen width, palette etc. You can actually choose what parts of the state are saved by setting bits in the CX register.

CX bit 0: Hardware state

CX bit 1: BIOS data state

CX bit 2: DAC state

CX bit 3: SuperVGA state

The BIOS replies, telling you how many blocks of 64 bytes it needs.

You must then allocate enough memory to hold the buffer.

And lastly, call the BIOS again to save the state.

All the requesting buffer sizes, saving and restoring are handled by the same function, Function 04h.

It is split up into 3 sub functions:

subfunction 00h: Request buffer size

subfunction 01h: Save state

subfunction 02h: Restore state

Right here's some pseudo code which will save the video state, and then restore it again:

## **Firstly ask it how much memory we need:**

```
AH = 4Fh      ;Super VGA support  
AL = 04h      ;Function 04h (save state)  
DL = 00h      ;Subfunction 00h (request amount of memory needed)  
CX = 00001111b ;save everything please  
call interrupt 10h
```

## **Next allocate enough memory to hold the buffer, and save the state**

```
SaveBuffer = AllocateMemory (BX * 64) bytes  
AH = 4Fh      ;Super VGA support  
AL = 04h      ;Function 04h (save state)  
DL = 01h      ;Subfunction 01h (save the state)  
CX = 00001111b ;save everything please  
ES:[DI] -> SaveBuffer  
call interrupt 10h
```

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_vapp03.htm](http://freespace.virgin.net/hugo.elias/graphics/x_vapp03.htm)>

# Page 5: Change the Virtual Screen Width

jueves, 10 de julio de 2014

14:53

Right, we're going to want this program to do some scrolling around, so we'll make the virtual screen twice the width of the physical screen. The width of the virtual screen is also known as the Logical Scan Line Length.

This is done using Function 06h. You can ask it to set the width to anything you like, but it won't always be able to do it for you. This could be for various reasons, hardware considerations, or there may not be enough memory. If it is unable to set the width you require, it will set it to the next highest it can. Whatever the result, it will tell you what width it has been set to. It will also tell you the maximum number of scanlines possible with the amount of RAM on the graphics card, And it will tell you how many bytes each scanline uses. You can either tell it to set the width, or you can ask it what the current width is.

## Set Logical Scan Line Length

```
AX = 4F06h          ; Function 06h
BL = 00h          ; Subfunction 00h (select Scan Line Length)
CX = DesiredWidth
call interrupt 10h
BytesPerScanLine = BX
LogicalWidth = CX
MaxScans = DX
if LogicalWidth not= DesiredWidth then
    print "could not set virtual screen to desired width."
    print "it has been set to ", LogicalWidth, " instead"
end if
print "the maximum number of scanlines available is ", MaxScans
print "each scanline takes up ", BytesPerScanLine, " bytes"
```

## Request current Logical Scan Line Length

```
AX = 4F06h          ; Function 06h
BL = 01h          ; Subfunction 01h (request scanline length)
BytesPerScanLine = BX
LogicalWidth = CX
MaxScans = DX
```

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_vapp05.htm](http://freespace.virgin.net/hugo.elias/graphics/x_vapp05.htm)>

# Page 6 - Plotting pixels

jueves, 10 de julio de 2014  
14:54

Right, this can be tricky. There's lots of horrid Bank Switching and stuff, and moving these windows around. I'll assume for the moment that we're plotting 256 colour pixels.

If you just want to plot a single pixel, the process requires several steps:

1. Calculate the Memory location of the pixel
2. Calculate the Required Window position
3. Switch to the Bank
4. Calculate the memory offset of the pixel from the start of the bank.
5. write the pixel to the screen.

This is much easier in a flat memory mode.

There are a few Global variables that this procedure will need when calculating things. If you don't like global variables, then you can figure out the cute little message passing stuff yourself.

BytesPerScanline ; the number of bytes used per scanline. see page 5  
WinGranularity ; the accuracy to which a memory window can be positioned see page 0  
CurrentWindowPos ; the current position of the Write Window (in granularity units).  
If it's already in the right position, we won't bother to move it.

OK, i'll explain how to perform each step in turn to plot a pixel at (x, y):

## **Step 1: Calculate the Memory location of the pixel**

If you are writing this in assembler, then you will need to use the extended registers (eax, ebx etc) to calculate the memory position, because it may well be higher than 65535. If you are writing it in a high level language, then the variable type will probably be called LONG or DOUBLE or something.

MemoryPosition = (BytesPerScanLine \* y) + x

That's simple enough.

## **Step 2: Calculate the Required Window position**

WindowPosition = MemoryPosition / WinGranularity

## **Step 3: Move the Window into that position**

SetWindowPosition(WindowPosition)

CurrentWindowPos = WindowPosition

THERE ARE TWO WINDOWS, A (0) AND B(1).

## **Step 4: Calculate address of pixel within window**

PixelOffset = MemoryPosition - (WinGranularity \* WindowPosition)

## **Step 5: Write The pixel**

Write a byte at [PixelOffset]

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_vapp06.htm](http://freespace.virgin.net/hugo.elias/graphics/x_vapp06.htm)>

# Page 7 - Scrolling

jueves, 10 de julio de 2014  
14:54

OK, this is nice and easy. What we're going to do is to scroll the screen in a big circle so that the physical screen just touches each edge of the logical screen.

The formula for moving in a circle, as if you need telling, is:

$$X = \sin(\text{angle}) * \text{xradius} + \text{xcenter}$$

$$Y = \cos(\text{angle}) * \text{yradius} + \text{ycenter}$$

I am assuming that we were able to set the Logical screen width to twice that of the X resolution, and that the variable MaxScans has been set from page 5.

So, first calculate the size of the circle to move in:

$$\text{xcenter} = (\text{Xresoluion} - \text{LogicalWidth}) / 2$$

$$\text{xradius} = (\text{Xresoluion} - \text{LogicalWidth}) / 2$$

$$\text{ycenter} = (\text{Yresoluion} - \text{MaxScans}) / 2$$

$$\text{yradius} = (\text{Yresoluion} - \text{MaxScans}) / 2$$

Now, make the screen scroll in a full circle in steps of 0.1 of a degree.

for angle=0 to 720 in steps of 0.1

begin loop

$$\text{AX} = 4F07h$$

$$\text{BX} = 0$$

$$\text{CX} = \sin(\text{angle}) * \text{xradius} + \text{xcenter}$$

$$\text{DX} = \cos(\text{angle}) * \text{yradius} + \text{ycenter}$$

call int 10h

end loop

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_vapp07.htm](http://freespace.virgin.net/hugo.elias/graphics/x_vapp07.htm)>

## Page 8: Restore previous state

jueves, 10 de julio de 2014  
14:55

Assuming that you saved the video state in page 3, it is now possible to restore it.

```
AH = 4Fh      ;Super VGA support
AL = 04h      ;Function 04h (save state)
DL = 02h      ;Subfunction 02h (restore the state)
CX = 00001111b ;restore everything please
ES:[DI] -> SaveBuffer
call interrupt 10h
```

If you did not, then, if you're going back into DOS, you should set the video mode back to 03h.

Pegado de <[http://freespace.virgin.net/hugo.elias/graphics/x\\_vapp08.htm](http://freespace.virgin.net/hugo.elias/graphics/x_vapp08.htm)>

# Abe's bag of tricks, SVGA Putpixel

jueves, 10 de julio de 2014

14:56

Welcome to this, the very first little trick in my bag. Hopefully there will be lots of tricks here that you can put in your own bag of tricks or perhaps you prefer to have them in your hat :-). I plan to put small tricks here that doesn't really fit into any part of the normal demoschool but is good to know anyway. I also plan to answer often asked questions here.

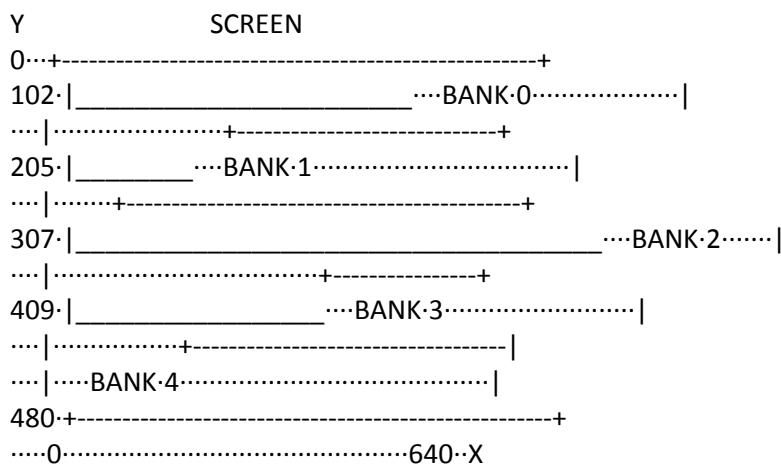
One question that I've received many mails about is

\* How do I put a pixel in svga mode 640\*480 256 colors?  
I only seem to be able to put them in the upper part of the screen.

Answer:

In this mode the screen has 640\*480 pixels and each pixel takes one byte in memory since it's 256 colors. As you perhaps know, one segment in real mode on the PC is 64k.  $640 \times 480 = 307200$  bytes). That means that it would take  $307200 / 64 = 4768$  segments to fill one 640\*480 screen. The PC only uses one 64k segment, A000:0000 - A000:FFFF, for graphic memory. With this you can only reach about 1/5 of the top of the screen. But there has got to be a way to draw an entire 640\*480 screen you may say, and it does. The answer is bank switching. With a couple of out instructions to the graphic card you can tell the card which portion of the screen should be mapped to A000:0000 - A000:FFFF. Each "bank" is 64k big. The first bank (bank 0) reaches down to the middle of row 103 ( $65536 / 640 = 102.5$ ). To reach the rest of row 103 and the next 102 rows you have to switch to bank 1. The 102 rows after that is bank 2 and so on. The last row is somewhere in bank 4.

It looks something like this:



The figure isn't exact but you get the idea I hope.

The address of a pixel is calculated like this:

$$\text{address} = 640 * y + x$$

Out of this address you get the banknumber:  $\text{address} / 0ffffh$   
and the offset into the bank:  $\text{address} \% 0ffffh$

In assembler you can calculate both the banknumber and offset with only one mul like this:

```
mov ax,[y]      ;y to ax
mul 640        ;(640*y)%0ffffh to ax
                ;and (640*y)/0ffffh to dx
add ax,[x]      ;add the x-value
jnc nocarry    ;if carry is set then
inc dx         ;increase dx (just passed a bankborder with that add)
nocarry:
```

;Now dx is the banknumber of the point and ax is the offset into 0a000h

```
.          ;switch bank
.
;just draw the point like we are used to, offset is in ax
mov bx,0a000h
mov es,bx      ;a000h to es
mov di,ax      ;calculated offset to di
mov ax,[color] ;the color of the pixel
mov [es:di],ax ;draw it
```

You may have noticed that I skipped the switch bank part. That was on purpose because there are two ways to do this. Either you can write directly to the ports of the graphic card or you can use the VESA bankswitch interrupt. To write to the ports seems to be slightly different depending on which card you have. That is NO GOOD. You'd have to write many bankswitch routines which works for their special card and detect which card is in the machine at the initialization of the program. This is too much work for our bag of tricks so we use the slow VESA bankswitch interrupt which at least works on all modern cards.

It's straightforward, just put 4f05 in ax, 0 in bx and banknumber\*16 in dx. Then call int 10h.

Here is the bankswitching code:

```
asm mov ax,4f05h /* VESA switch bank */
asm xor bx,bx   /* bx should be 0 */
asm mov dx,[bank] /* dx = bank*16 */
asm shl dx,4
asm int 10h     /* and do it */
```

Remember to save ax if you put it in the pixel code above. I have written a very small C-program that puts pixels on a 640\*480 screen in different psychedelic patterns. Then the palette is rotated, just like in the demoschool part I. You can change pattern by pressing + or -. Stop the paletterotation with space and end the program with Escape. If you keep pressing + a while there will appear some very interesting patterns. I didn't count with this, it was just a coincident but that's cool.

The C-code, Mr Ant on Magic Mushrooms:

```
<-----Cut--Here-----
>

/*****************************************/
/*      Abe's bag of tricks, SVGA Putpixel      */
/*                                              */

```

```

/*
 * In part 4 I said that part 5 would be about filled vector graphics */
/* But this is something completely different, this is Svga.      */
/* So I can't call this part 5. I could call it part 4.5 or I could */
/* Create a new section in the demoschool, and that's what I did.   */
/*                                         */
/*          1996-04-30 Abe Racadabra           */
/*          mail: dat94avi@bilbo.mdh.se       */
/*                                         */
/*****************************************/
#include<conio.h>
#include<stdio.h>

#define PALSIZE 256

int bank;      /* bank and pal is global just because I'm so lazy */
char pal[PALSIZE*3];

void printinfo(void)
{
    clrscr();
    printf("This silly little program's sole purpose is to demonstrate\n");
    printf("how to use a putpixel routine in svga 640*480 256 colors.\n\n");
    printf("Keys: + increase w\n");
    printf(" - decrease w\n");
    printf(" space stop cycling the palette\n");
    printf(" Esc quit\n\n");
    printf("Press a key to begin\n");
    getch();
}

void wtsync(void)
{
    asm mov dx,3DAh
    WaitVR1:
    asm in al,dx
    asm test al,8
    asm jne WaitVR1
    WaitVR2:
    asm in al,dx
    asm test al,8
    asm je WaitVR2
}

void getpal(void)
{
    int i;
    outp(0x3c7,0);
    for(i=0;i<256*3;i++)
        pal[i]=inp(0x3c9);
}

void setpal(void)
{
    int i;
    outp(0x3c8,0);
    for(i=0;i<256*3;i++)

```

```

outp(0x3c9,pal[i]);
}

void cyclepal(void)
{
char r,g,b;
int i;
r=pal[0];
g=pal[1];
b=pal[2];
for(i=0;i<(PALSEIZE-1)*3;i++) pal[i]=pal[i+3];
pal[(PALSEIZE-1)*3+0]=r;
pal[(PALSEIZE-1)*3+1]=g;
pal[(PALSEIZE-1)*3+2]=b;
wtsync();
setpal();
}

/* change these color-runs for different colors (duh . . .) */
void coolpal(void)
{
int i,r=0,g=0,b=0,col;

col=0;

outp(0x3c8,0);

for(i=0;i<64;i++) /* color 0-63 */
{
    pal[col++]=r;
    pal[col++]=g;
    pal[col++]=b;
    if(r<63) r++;
}
for(i=0;i<64;i++) /* color 64-127 */
{
    pal[col++]=r;
    pal[col++]=g;
    pal[col++]=b;
    if(g<63) g++;
}
for(i=0;i<64;i++) /* color 128-191 */
{
    pal[col++]=r;
    pal[col++]=g;
    pal[col++]=b;
    if(b<63) b++;
}
for(i=0;i<64;i++) /* color 192-255 */
{
    pal[col++]=r;
    pal[col++]=g;
    pal[col++]=b;
    if(r>0) r--;
    if(g>0) g--;
    if(b>0) b--;
}

```

```

}

setpal();
}

void setmode(int mode)
{
asm mov ax,mode
asm int 10h
}

/* this is the slow way to switch bank, with the VESA bios interrupt */
/* it is faster to write to the ports of the graphic card directly */
/* but that seems to be different for every card, this works for all */
void setbank(void) /* the bank that shall be set is a global variable */
{
asm mov ax,4f05h /* VESA switch bank */
asm xor bx,bx /* bx should be 0 */
asm mov dx,bank /* dx = bank*16 */
asm shl dx,4
asm int 10h /* and do it */
}

/* The address is calculated with is 640*y + x */
/* a mul works great because it puts the high 16 bits of a mul in dx */
/* and the low 16 bits in ax */
/* after the mul, dx will contain the bank number */
/* and ax will be the offset into that bank */
void putpixel(int x,int y,unsigned char col)
{
int ofs;
asm mov ax,640
asm mov bx,y
asm mul bx /* ax = 640*y, dx = bank */
asm add ax,x /* ax = (640*y + x)%(64k) */
asm jnc noc /* if the add went into a new bank (carry set) */
asm inc dx /* then bank = bank + 1 */
noc:
asm mov ofs,ax /* mov ax (offset into bank) to ofs */
asm cmp dx,bank /* compare this pixel's bank to the current bank */
asm jz same /* if it's the same, don't set the bank */
asm mov bank,dx /* else update bank */
setbank(); /* and set new bank (faster to put the code here directly) */
same:
asm push di
asm mov ax,0a000h /* and put the pixel into a000:offset */
asm mov es,ax
asm mov di,ofs
asm mov al,col
asm mov [es:di],al
asm pop di
}

void main(void)
{
int i,j,w;
char ch;
printinfo();
}

```

```
setmode(0x005f); /* change into 640*480 256 colors (on my cirrus logic) */  
/* I'm not sure if mode 5f is 640*480 for all cards */
```

```
bank=0;  
w=48; /* how the screen will look depends on w */  
getpal();  
coolpal(); /* set a smooth palette, comment this line out for a strange  
effect */  
do  
{  
for(j=0;j<480;j++) /* draw a screen */  
{  
for(i=0;i<640;i++) /* I just experimented until I got this formula */  
putpixel(i,j,i*j+i*i+j*j+i*j*w); /* change this formula for different  
effects, for instance try sin & cos  
combinations . . . */  
}  
  
do  
{  
cyclepal(); /* cycle the palette until a key is pressed */  
}while(!kbhit());  
ch=getch(); /* check what key was pressed */  
  
switch(ch) /* and react to the keypress */  
{  
case ' ': getch(); /* if space, stop cycling */  
break;  
case '+': w++; /* case +, increase w */  
break;  
case '-': w--; /* case -, decrease w */  
break;  
}  
}while(ch!="\x1b"); /* Escape ends */
```

```
setmode(3); /* get back to dos mode */  
printf("\nw = %d\n",w); /* write the last w just incase it was really cool */  
/* and you want know what value w had */  
}
```

```
<-----Cut--Here-----  
>
```

Well that's all folks. \*Click\*

Read more: <http://www.intel-assembler.it/portale/5/programming-vga-card-asm-c/programming-vga-card-asm-c.asp#ixzz374OW25Ar>

Pegado de <<http://www.intel-assembler.it/portale/5/programming-vga-card-asm-c/programming-vga-card-asm-c.asp>>

# Intel 8086 microprocessor architecture

jueves, 10 de julio de 2014

14:58

## Memory

Program, data and stack memories occupy the same memory space. The total addressable memory size is 1MB KB. As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory. To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory (see the "Registers" section below). 16-bit pointers and data are stored as:

address: low-order byte

address+1: high-order byte

32-bit addresses are stored in "segment:offset" format as:

address: low-order byte of segment

address+1: high-order byte of segment

address+2: low-order byte of offset

address+3: high-order byte of offset

Physical memory address pointed by segment:offset pair is calculated as:

address = (<segment> \* 16) + <offset>

**Program memory** - program can be located anywhere in memory. Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory. All conditional jump instructions can be used to jump within approximately +127 - -127 bytes from current instruction.

**Data memory** - the 8086 processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB (if all four segments point to different 64 KB blocks). Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES: (some registers and instructions by default may use the ES or SS segments instead of DS segment).

Word data can be located at odd or even byte boundaries. The processor uses two memory accesses to read 16-bit word located at odd byte boundaries. Reading word data from even byte boundaries requires only one memory access.

**Stack memory** can be placed anywhere in memory. The stack can be located at odd memory addresses, but it is not recommended for performance reasons (see "Data Memory" above).

### Reserved locations:

- 0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment:offset.
- FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.

## Interrupts

The processor has the following interrupts:

**INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction. When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 \* <interrupt type>. Interrupt processing routine should return with the IRET instruction.

**NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.

**Software interrupts** can be caused by:

- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: divide error (type 0), unused opcode (type 6) and escape

opcode (type 7).

Software interrupt processing is the same as for the hardware interrupts.

## I/O ports

65536 8-bit I/O ports. These ports can be also addressed as 32768 16-bit I/O ports.

## Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the 8086 microprocessor uses four segment registers:

**Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

**Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

**Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

**Extra segment** (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

**Accumulator** register consists of 2 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

**Base** register consists of 2 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

**Count** register consists of 2 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

**Data** register consists of 2 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

The following registers are both general and index registers:

**Stack Pointer** (SP) is a 16-bit register pointing to program stack.

**Base Pointer** (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

**Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

**Destination Index** (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

**Instruction Pointer** (IP) is a 16-bit register.

**Flags** is a 16-bit register containing 9 1-bit flags:

- Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- Direction Flag (DF) - if set then string manipulation instructions will auto-decrement

- index registers. If cleared then the index registers will be auto-incremented.
- Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.
  - Single-step Flag (TF) - if set then single-step interrupt will occur after the next instruction.
  - Sign Flag (SF) - set if the most significant bit of the result is set.
  - Zero Flag (ZF) - set if the result is zero.
  - Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
  - Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
  - Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

## Instruction Set

Instruction set of Intel 8086 processor consists of the following instructions:

- Data moving instructions.
- Arithmetic - add, subtract, increment, decrement, convert byte/word and compare.
- Logic - AND, OR, exclusive OR, shift/rotate and test.
- String manipulation - load, store, move, compare and scan for byte/word.
- Control transfer - conditional, unconditional, call subroutine and return from subroutine.
- Input/Output instructions.
- Other - setting/clearing flag bits, stack operations, software interrupts, etc.

## Addressing modes

**Implied** - the data value/data address is implicitly associated with the instruction.

**Register** - references the data in a register or in a register pair.

**Immediate** - the data is provided in the instruction.

**Direct** - the instruction operand specifies the memory address where data is located.

**Register indirect** - instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.

**Based** - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.

**Indexed** - 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

**Based Indexed** - the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

**Based Indexed with displacement** - 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

Pegado de <<http://www.cpu-world.com/Arch/8086.html>>

## FLAGS register

From Wikipedia, the free encyclopedia

The **FLAGS register** is the [status register](#) in [Intel x86 microprocessors](#) that contains the current state of the processor. This register is [16 bits](#) wide. Its successors, the **EFLAGS** and **RFLAGS** registers, are [32 bits](#) and [64 bits](#) wide, respectively. The wider registers retain compatibility with their smaller predecessors.

## Contents

- [1 Flags](#)
- [2 Use](#)
- [3 Determination of processor type](#)
- [4 See also](#)
- [5 References](#)

## Flags

Intel x86 FLAGS  
register<sup>[1]</sup>

Bit #	Abbreviation	Description	Category
<b>FLAGS</b>			
0	CF	<a href="#">Carry flag</a>	Status
1	1	Reserved	
2	PF	<a href="#">Parity flag</a>	Status
3	0	Reserved	
4	AF	<a href="#">Adjust flag</a>	Status
5	0	Reserved	
6	ZF	<a href="#">Zero flag</a>	Status
7	SF	<a href="#">Sign flag</a>	Status
8	TF	<a href="#">Trap flag</a> (single step)	Control
9	IF	<a href="#">Interrupt enable flag</a>	Control
10	DF	<a href="#">Direction flag</a>	Control
11	OF	<a href="#">Overflow flag</a>	Status
12-13	IOPL	<a href="#">I/O privilege level</a> (286+ only), always 1 on 8086 and 186	System
14	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System
15	0	Reserved, always 1 on 8086 and 186, always 0 on later models	
<b>EFLAGS</b>			
16	RF	<a href="#">Resume flag</a> (386+ only)	System
17	VM	<a href="#">Virtual 8086 mode</a> flag (386+ only)	System
18	AC	Alignment check (486SX+ only)	System
19	VIF	Virtual interrupt flag (Pentium+)	System
20	VIP	Virtual interrupt pending (Pentium+)	System
21	ID	Able to use CPUID instruction (Pentium+)	System
22	0	Reserved	
23	0	Reserved	
24	0	Reserved	
25	0	Reserved	
26	0	Reserved	
27	0	Reserved	
28	0	Reserved	
29	0	Reserved	
30	0	Reserved	
31	0	Reserved	
<b>RFLAGS</b>			
32-63	0	Reserved	

## Use

The POPF, POPFD, and POPFQ instructions read from the stack the first 16, 32, and 64 bits of the flags register, respectively. POPFD was introduced with the [i386](#) architecture and POPFQ with the

[x64](#) architecture. In 64-bit mode, PUSHF/POPF and PUSHFQ/POPFQ are available but not PUSHFD/POPFD.<sup>[2]</sup>

The following [assembly code](#) changes the direction flag (DF):

```
pushf ; Pushes the current flags onto the stack  
pop ax ; Pop the flags from the stack into ax register  
push ax ; Push them back onto the stack for storage  
xor ax, 400h ; toggle the DF flag only, keep the rest of the flags  
push ax ; Push again to add the new value to the stack  
popf ; Pop the newly pushed into the FLAGS register  
; ... Code here ...  
popf ; Pop the old FLAGS back into place
```

In practical software, the cld and std instructions are used to clear and set the direction flag, respectively. Some instructions in [assembly language](#) use the FLAGS register. The conditional jump instructions use certain flags to compute. For example, jz uses the zero flag, jc uses the carry flag, jo uses the overflow flag...

## Determination of processor type

Testing if certain bits in the FLAGS register are changeable allows determining what kind of processor is installed. For example, the alignment flag can only be changed on the [486](#) and above, so if it can be changed then the CPU is a 486 or higher. These methods of processor detection were made obsolete by the [CPUID](#) instruction, which was first included in the [Intel Pentium](#).

## See also

- [Status register](#)
- [Flag byte](#)
- [Flag \(computing\)](#)
- [Program status word](#)
- [Control register](#)
- [CPU flag \(x86\)](#)
- [x86](#)
- [x86 assembly language](#)
- [x86 instruction listings](#)

## References

1. [Intel 64 and IA-32 Architectures Software Developer's Manual 1](#). May 2012. pp. 3–21.
2. [Intel 64 and IA-32 Architectures Software Developer's Manual 2B](#). May 2012. pp. 4–349, 4–432.

Pegado de <[http://en.wikipedia.org/wiki/FLAGS\\_register](http://en.wikipedia.org/wiki/FLAGS_register)>

# Selected 8086 Instructions

jueves, 10 de julio de 2014  
15:02

## Quick Reference List

<a href="#">adc</a>	Add with carry flag
<a href="#">add</a>	Add two numbers
<a href="#">and</a>	Bitwise logical AND
<a href="#">call</a>	Call procedure or function
<a href="#">cbw</a>	Convert byte to word (signed)
<a href="#">cli</a>	Clear interrupt flag (disable interrupts)
<a href="#"> cwd</a>	Convert word to doubleword (signed)
<a href="#">cmp</a>	Compare two operands
<a href="#">dec</a>	Decrement by 1
<a href="#">div</a>	Unsigned divide
<a href="#">idiv</a>	Signed divide
<a href="#">imul</a>	Signed multiply
<a href="#">in</a>	Input (read) from port
<a href="#">inc</a>	Increment by 1
<a href="#">int</a>	Call to interrupt procedure
<a href="#">iret</a>	Interrupt return
<a href="#">j??</a>	Jump if ?? condition met
<a href="#">jmp</a>	Unconditional jump
<a href="#">lea</a>	Load effective address offset
<a href="#">mov</a>	Move data
<a href="#">mul</a>	Unsigned multiply
<a href="#">neg</a>	Two's complement negate
<a href="#">nop</a>	No operation
<a href="#">not</a>	One's complement negate
<a href="#">or</a>	Bitwise logical OR
<a href="#">out</a>	Output (write) to port
<a href="#">pop</a>	Pop word from stack
<a href="#">popf</a>	Pop flags from stack
<a href="#">push</a>	Push word onto stack
<a href="#">pushf</a>	Push flags onto stack
<a href="#">ret</a>	Return from procedure or function
<a href="#">sal</a>	Bitwise arithmetic left shift (same as shl)
<a href="#">sar</a>	Bitwise arithmetic right shift (signed)
<a href="#">sbb</a>	Subtract with borrow
<a href="#">shl</a>	Bitwise left shift (same as sal)
<a href="#">shr</a>	Bitwise right shift (unsigned)
<a href="#">sti</a>	Set interrupt flag (enable interrupts)
<a href="#">sub</a>	Subtract two numbers
<a href="#">test</a>	Bitwise logical compare
<a href="#">xor</a>	Bitwise logical XOR

## Detailed Instruction List

A complete listing of all x86 instructions along with usage and encoding information can be found in the [NASM Manual](#) (852 KB). However, when using this manual, be careful to only use instructions compatible with the 8086. The [Am186/Am188 Instruction Set Manual](#) (2,242 KB) contains a more detailed description of instruction behavior for instructions compatible with the 8086. However, these AMD processors also support the following x86 instructions which are not 8086 compatible: bound, enter, ins, leave, outs, popa, and pusha.

### Important Usage Notes:

1. The first operand of an instruction is also the destination if there is a resulting value. Divide and multiply instructions are common exceptions to this rule.
2. There can be *at most* one memory operand per instruction.
3. There can be *at most* one immediate operand per instruction.
4. Operands generally must be of the same size (i.e., byte or word).
5. Using a label is the same as using an immediate or constant value.
6. When BP is used in a memory reference, SS is assumed as the segment. Otherwise DS is assumed.
7. While an instruction is executing, IP refers to the next instruction.
8. Many instructions are smaller if you use the appropriate registers (usually AX or AL).
9. In NASM, all labels are case sensitive but instruction and register names are not.

#### **Terminology Used:**

- **memory** - Refers to an 8 or 16-bit memory location determined by an effective address.
- **register** - AX, BX, CX, DX, SI, DI, BP, or SP as well as the 8-bit derivatives of AX, BX, CX, and DX (other registers or flags are not allowed).
- **immediate** - A numeric constant or label.
- **REG1::REG2** - The concatenation of two registers (e.g., the 32-bit value DX::AX) A single colon is used for memory addresses.
- **XF or XF=b** - A flag's value after an instruction can be 0 or 1 and usually depends on the result of the instruction. A flag being set to '?' by an instruction indicates that the flag is undefined after the operation.

#### **Instructions:**

##### **adc Add with carry flag**

Syntax:       adc     dest, src

dest: memory or register

src: memory, register, or immediate

Action: dest = dest + src + CF

Flags Affected: OF, SF, ZF, AF, PF, CF

Notes: This instruction is used to perform 32-bit addition.

##### **add Add two numbers**

Syntax:       add     dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest + src

Flags Affected: OF, SF, ZF, AF, PF, CF

Notes: Works for both signed and unsigned numbers.

##### **and Bitwise logical AND**

Syntax:       and     dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest & src

Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

##### **call Call procedure or function**

Syntax:       call     addr

addr: register, memory, or immediate

Action: Push IP onto stack, set IP to addr.

Flags Affected: None

##### **cbw Convert byte to word (signed)**

Syntax:       cbw

Action: Sign extend AL to create a word in AX.

Flags Affected: None

Notes: For unsigned numbers use "mov ah, 0".

##### **cli Clear interrupt flag (disable interrupts)**

Syntax:       cli

Action: Clear IF

Flags Affected: IF=0

**cmp Compare two operands**

Syntax: cmp op1, op2

op1: register or memory

op2: register, memory, or immediate

Action: Perform op1-op2, discarding the result but setting the flags.

Flags Affected: OF, SF, ZF, AF, PF, CF

Notes: Usually used before a conditional jump instruction.

**cwd Convert word to doubleword (signed)**

Syntax: cwd

Action: Sign extend AX to fill DX, creating a dword contained in DX::AX.

Flags Affected: None

Notes: For unsigned numbers use "xor dx, dx" to clear DX.

**dec Decrement by 1**

Syntax: dec op

op: register or memory

Action: op = op - 1

Flags Affected: OF, SF, ZF, AF, PF

**div Unsigned divide**

Syntax: div op8

div op16

op8: 8-bit register or memory

op16: 16-bit register or memory

Action: If operand is op8, unsigned AL = AX / op8 and AH = AX % op8

If operand is op16, unsigned AX = DX::AX / op16 and DX = DX::AX % op16

Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?

Notes: Performs both division and modulus operations in one instruction.

**idiv Signed divide**

Syntax: idiv op8

idiv op16

op8: 8-bit register or memory

op16: 16-bit register or memory

Action: If operand is op8, signed AL = AX / op8 and AH = AX % op8

If operand is op16, signed AX = DX::AX / op16 and DX = DX::AX % op16

Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?

Notes: Performs both division and modulus operations in one instruction.

**imul Signed multiply**

Syntax: imul op8

imul op16

op8: 8-bit register or memory

op16: 16-bit register or memory

Action: If operand is op8, signed AX = AL \* op8

If operand is op16, signed DX::AX = AX \* op16

Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF

**in Input (read) from port**

Syntax: in AL, op8

in AX, op8

op8: 8-bit immediate or DX

Action: If destination is AL, read byte from 8-bit port op8.

If destination is AX, read word from 16-bit port op8.

Flags Affected: None

**inc Increment by 1**

Syntax: inc op

op: register or memory

Action: op = op + 1

Flags Affected: OF, SF, ZF, AF, PF

**int Call to interrupt procedure**

Syntax: int imm8

imm8: 8-bit unsigned immediate

Action: Push flags, CS, and IP; clear IF and TF (disabling interrupts); load word at address (imm8\*4) into IP and word at (imm8\*4 + 2) into CS.

Flags Affected: IF=0, TF=0

Notes: This instruction is usually used to call system routines.

#### **iret Interrupt return**

Syntax:      iret

Action: Pop IP, CS, and flags (in that order).

Flags Affected: All

Notes: This instruction is used at the end of ISRs.

#### **j?? Jump if ?? condition met**

Syntax:      j??      rel8

rel8: 8-bit signed immediate

Action: If condition ?? met, IP = IP + rel8 (sign extends rel8)

Flags Affected: None

Notes: Use the cmp instruction to compare two operands then j?? to jump conditionally. The ?? of the instruction name represents the jump condition, allowing for following instructions:

ja      jump if above, unsigned >

jae    jump if above or equal, unsigned >=

jb     jump if below, unsigned <

jbe    jump if below or equal, unsigned <=

je     jump if equal, ==

jne    jump if not equal, !=

jg     jump if greater than, signed >

jge    jump if greater than or equal, signed >=

jle    jump if less than or equal, signed <=

All of the ?? suffixes can also be of the form n?? (e.g., jna for

jump if not above). See 8086 documentation for many more ?? conditions.

An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump.

Note also that rel8 operand greatly limits conditional jump distance

(-127 to +128 bytes from IP). Use the jmp instruction in combination with j?? to overcome this barrier.

#### **jmp Unconditional jump**

Syntax:      jump    rel

        jump op16

        jump seg:off

rel: 8 or 16-bit signed immediate

op16: 16-bit register or memory

seg:off: Immediate 16-bit segment and 16-bit offset

Action: If operand is rel, IP = IP + rel

If operand is op16, IP = op16

If operand is seg:off, CS = seg, IP = off

Flags Affected: None

Notes: An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump.

#### **lea Load effective address offset**

Syntax:      lea      reg16, memref

reg16: 16-bit register

memref: An effective memory address (e.g., [bx+2])

Action: reg16 = address offset of memref

Flags Affected: None

Notes: This instruction is used to easily calculate the address of data in memory. It does not actually access memory.

#### **mov Move data**

Syntax:        mov     dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = src

Flags Affected: None

### **mul    Unsigned multiply**

Syntax:        mul     op8

                mul     op16

op8: 8-bit register or memory

op16: 16-bit register or memory

Action: If operand is op8, unsigned AX = AL \* op8

If operand is op16, unsigned DX::AX = AX \* op16

Flags Affected: OF, SF=? , ZF=? , AF=? , PF=? , CF

### **neg    Two's complement negate**

Syntax:        neg     op

op: register or memory

Action: op = 0 - op

Flags Affected: OF, SF, ZF, AF, PF, CF

### **nop    No operation**

Syntax:        nop

Action: None

Flags Affected: None

### **not    One's complement negate**

Syntax:        not     op

op: register or memory

Action: op = ~op

Flags Affected: None

### **or    Bitwise logical OR**

Syntax:        or     dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest | src

Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

### **out    Output (write) to port**

Syntax:        out     op, AL

                out     op, AX

op: 8-bit immediate or DX

Action: If source is AL, write byte in AL to 8-bit port op.

If source is AX, write word in AX to 16-bit port op.

Flags Affected: None

### **pop    Pop word from stack**

Syntax:        pop     op16

reg16: 16-bit register or memory

Action: Pop word off the stack and place it in op16 (i.e., op16 = [SS:SP]  
then SP = SP + 2).

Flags Affected: None

Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.

### **popf    Pop flags from stack**

Syntax:        popf

Action: Pop word from stack and place it in flags register.

Flags Affected: All

### **push    Push word onto stack**

Syntax:        push     op16

op16: 16-bit register or memory

Action: Push op16 onto the stack (i.e., SP = SP - 2 then [SS:SP] = op16).

Flags Affected: None

Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.

**pushf Push flags onto stack**

Syntax: pushf

Action: Push flags onto stack as a word.

Flags Affected: None

**ret Return from procedure or function**

Syntax: ret

Action: Pop word from stack and place it in IP.

Flags Affected: None

**sal Bitwise arithmetic left shift (same as shl)**Syntax: sal op, 1  
          sal op, CL

op: register or memory

Action: If operand is 1, op = op &lt;&lt; 1

If operand is CL, op = op &lt;&lt; CL

Flags Affected: OF, SF, ZF, AF=?, PF, CF

**sar Bitwise arithmetic right shift (signed)**Syntax: sar op, 1  
          sar op, CL

op: register or memory

Action: If operand is 1, signed op = op &gt;&gt; 1 (sign extends op)

If operand is CL, signed op = op &gt;&gt; CL (sign extends op)

Flags Affected: OF, SF, ZF, AF=?, PF, CF

**sbb Subtract with borrow**

Syntax: sbb dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest - (src + CF)

Flags Affected: OF, SF, ZF, AF, PF, CF

Notes: This instruction is used to perform 32-bit subtraction.

**shl Bitwise left shift (same as sal)**Syntax: shl op, 1  
          shl op, CL

op: register or memory

Action: If operand is 1, op = op &lt;&lt; 1

If operand is CL, op = op &lt;&lt; CL

Flags Affected: OF, SF, ZF, AF=?, PF, CF

**shr Bitwise right shift (unsigned)**Syntax: shr op, 1  
          shr op, CL

op: register or memory

Action: If operand is 1, op = (unsigned)op &gt;&gt; 1

If operand is CL, op = (unsigned)op &gt;&gt; CL

Flags Affected: OF, SF, ZF, AF=?, PF, CF

**sti Set interrupt flag (enable interrupts)**

Syntax: sti

Action: Set IF

Flags Affected: IF=1

**sub Subtract two numbers**

Syntax: sub dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest - src

Flags Affected: OF, SF, ZF, AF, PF, CF

Notes: Works for both signed and unsigned numbers.

**test Bitwise logical compare**

Syntax: test op1, op2

op1: register, memory, or immediate

op2: register, memory, or immediate

Action: Perform op1 & op2, discarding the result but setting the flags.

Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

Notes: This instruction is used to test if bits of a value are set.

### **xor      Bitwise logical XOR**

Syntax:      xor      dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = dest ^ src

Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

Pegado de <<http://ece425web.groups.et.byu.net/stable/labs/8086InstructionSet.html>>

# Intel x86 JUMP quick reference

jueves, 10 de julio de 2014

15:11

## [Home](#)

- [Contact](#)
- [About](#)
- [TechTips](#)
- [Tools&Source](#)
- [Evo Payroll](#)
- [Research](#)
- [AT&T 3B2](#)
- [Advisories](#)
- [News/Pubs](#)
- [Literacy](#)
- [Calif.Voting](#)
- [Personal](#)
- [Tech Blog](#)
- [SmokeBlog](#)

Getting the sense for jumps and flags has long been a troublesome area for me, especially since the Intel assembler book shows 32 of these, all with similar-sounding names. Looking more closely I found that many of the instructions were synonyms for each other, and in practice the whole gamut is not needed, and in the process found that my copy of Intel's 80386 Programmer's Reference Manual gave an incorrect description for one of the instructions.

So I have grouped these functionally, with all instruction synonyms in the same row.

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JO	Jump if overflow		OF = 1	70	0F 80
JNO	Jump if not overflow		OF = 0	71	0F 81
JS	Jump if sign		SF = 1	78	0F 88
JNS	Jump if not sign		SF = 0	79	0F 89
JE	Jump if equal		ZF = 1	74	0F 84
JZ	Jump if zero				
JNE	Jump if not equal		ZF = 0	75	0F 85
JNZ	Jump if not zero				
JB	Jump if below	unsigned	CF = 1	72	0F 82
JNAE	Jump if not above or equal				
JC	Jump if carry				
JNB	Jump if not below	unsigned	CF = 0	73	0F 83
JAE	Jump if above or equal				
JNC	Jump if not carry				
JBE	Jump if below or equal	unsigned	CF = 1 or ZF = 1	76	0F 86
JNA	Jump if not above				
JA	Jump if above	unsigned	CF = 0 and ZF = 0	77	0F 87
JNBE	Jump if not below or equal				
JL	Jump if less	signed	SF <> OF	7C	0F 8C
JNGE	Jump if not greater or equal				
JGE	Jump if greater or equal	signed	SF = OF	7D	0F 8D
JNL	Jump if not less				

JLE	Jump if less or equal	signed	ZF = 1 or SF <> OF	7E	OF 8E
JNG	Jump if not greater				
JG	Jump if greater	signed	ZF = 0 and SF = OF	7F	OF 8F
JNLE	Jump if not less or equal				
JP	Jump if parity		PF = 1	7A	OF 8A
JPE	Jump if parity even				
JNP	Jump if not parity		PF = 0	7B	OF 8B
JPO	Jump if parity odd				
JCXZ	Jump if %CX register is 0		%CX = 0		E3
JECXZ	Jump if %ECX register is 0		%ECX = 0		

## Processor Flags

The x86 processors have a large set of flags that represent the state of the processor, and the conditional jump instructions can key off of them in combination.

CF - carry flag

Set on high-order bit carry or borrow; cleared otherwise

PF - parity flag

Set if low-order eight bits of result contain an even number of "1" bits; cleared otherwise

ZF - zero flags

Set if result is zero; cleared otherwise

SF - sign flag

Set equal to high-order bit of result (0 if positive 1 if negative)

OF - overflow flag

Set if result is too large a positive number or too small a negative number (excluding sign bit) to fit in destination operand; cleared otherwise

Pegado de <<http://www.unixwiz.net/techtips/x86-jumps.html>>

## Far Jump

A far jump is an intersegment jump, which means that the destination address is in a different code segment. This will be a 5-byte instruction, the first byte being the opcode, the second and the third, the new value for the IP, and the fourth and fifth the new values of CS.

To specify that a jump is to a different code segment, a far pointer can be used in the format:

**JMP FAR PTR**

### Far Jump example:

JMP EBX means to instructs the CPU to jump to address pointed by EBX register value. If EBX is 12345678h, it'll jump to address 12345678h. So the instruction is interpreted as JMP 12345678h.

JMP DWORD PTR DS:[EBX] means to instructs the CPU to jump to address pointed by a DWORD value read from a memory address pointed by EBX register value and the DS selector.

If EBX is 11111111h, the CPU will read the DWORD value from memory at address DS:11111111h, then jump to the address pointed by that DWORD value (the value from the memory). If the DWORD value from the memory is 22222222h, it'll jump to address 22222222h. So the instruction is interpreted as JMP DWORD PTR DS:[11111111h]. After the memory is read, it would be interpreted as JMP 22222222h.

Pegado de <<http://stackoverflow.com/questions/12246581/jmp-instruction>>

# BT Instruction (Bit Test)

jueves, 10 de julio de 2014

15:04

From Wikipedia, the free encyclopedia

The **BT** [x86 assembly language](#) instruction stands for **Bit Test** and was added to the [x86 instruction set](#) with the [80386](#) processor. BT copies a [bit](#) from a given [register](#) to the [carry flag](#).<sup>[1]</sup>

Example: copy the third least significant [bit](#) from EAX to the [carry flag](#)

BT EAX, 2

BTS (Bit Test and Set) operates the same, but also sets the bit in the register,<sup>[2]</sup> while BTR (Bit Test and Reset) resets it,<sup>[3]</sup> and BTC (Bit Test and Complement) flips it.<sup>[4]</sup>

## References

1. "[BT - Bit Test](#)". Retrieved 2011-08-21.
2. "[BTS - Bit Test and Set](#)". Retrieved 2011-08-21.
3. "[BTR - Bit Test and Reset](#)". Retrieved 2011-08-21.
4. "[BTC -- Bit Test and Complement](#)". Retrieved 2012-11-04.

Pegado de <[http://en.wikipedia.org/wiki/Bit\\_Test](http://en.wikipedia.org/wiki/Bit_Test)>

# Full list of Instructions

jueves, 10 de julio de 2014  
15:10

## x86 integer instructions

This is the full 8086/8088 instruction set, most, if not all of these instructions are available in 32-bit mode, they just operate on 32-bit registers (eax, ebx, etc.) and values instead of their 16-bit (ax, bx, etc.) counterparts. See also [x86 assembly language](#) for a quick tutorial for this processor family. The updated instruction set is also grouped according to architecture (i386, i486, i686) and more generally is referred to as x86\_32 and x86\_64 (also known as AMD64).

### Original 8086/8088 instructions

Instructi on	Meaning	Notes	Opcode
<a href="#">AAA</a>	ASCII adjust AL after addition	used with unpacked <a href="#">binary coded decimal</a>	0x37
AAD	ASCII adjust AX before division	8086/8088 datasheet documents only base 10 version of the AAD instruction ( <a href="#">opcode</a> 0xD5 0x0A), but any other base will work. Later Intel's documentation has the generic form too. NEC V20 and V30 (and possibly other NEC V-series CPUs) always use base 10, and ignore the argument, causing a number of incompatibilities	0xD5
AAM	ASCII adjust AX after multiplication	Only base 10 version (Operand is 0xA) is documented, see notes for AAD	0xD4
AAS	ASCII adjust AL after subtraction		0x3f
ADC	Add with carry	destination := destination + source + <a href="#">carry flag</a>	
ADD	Add	(1) r/m += r/imm; (2) r += m/imm;	
AND	<a href="#">Logical AND</a>	(1) r/m &= r/imm; (2) r &= m/imm;	
CALL	Call procedure	push eip; eip points to the instruction directly after the call	
CBW	Convert byte to word		0x98
CLC	Clear <a href="#">carry</a> <a href="#">flag</a>	CF = 0;	0xF8
CLD	Clear <a href="#">direction flag</a>	DF = 0;	0xFC
<a href="#">CLI</a>	Clear <a href="#">interrupt flag</a>	IF = 0;	0xFA
CMC	Complement carry flag		0xF5
CMP	Compare operands		
CMPSB	Compare bytes in memory		0xA6

CMPSW	Compare words	0xA7
CWD	Convert word to doubleword	0x99
<a href="#">DAA</a>	Decimal adjust AL after addition (used with packed <a href="#">binary coded decimal</a> )	0x27
<a href="#">DAS</a>	Decimal adjust AL after subtraction	0x2F
DEC	Decrement by 1	
DIV	Unsigned divide DX:AX = DX:AX / r/m; resulting DX = remainder	
ESC	Used with <a href="#">floating-point unit</a>	
<a href="#">HLT</a>	Enter halt state	0xF4
IDIV	Signed divide DX:AX = DX:AX / r/m; resulting DX = remainder	
IMUL	Signed multiply (1) DX:AX = AX * r/m; (2) AX = AL * r/m	
IN	Input from port (1) AL = port[imm]; (2) AL = port[DX]; (3) AX = port[DX];	
INC	Increment by 1	
<a href="#">INT</a>	Call to <a href="#">interrupt</a>	
INTO	Call to interrupt if overflow	0xCE
IRET	Return from interrupt	
Jcc	<a href="#">Jump if condition</a> (JA, JAE, JB, JBE, JC, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ)	
JCXZ	Jump if CX is zero	0xE3
<a href="#">JMP</a>	Jump	
LAHF	Load FLAGS into AH register	0x9F
LDS	Load pointer using DS	0xC5
LEA	<a href="#">Load Effective Address</a>	0x8D
LES	Load ES with	0xC4

	pointer		
LOCK	Assert BUS LOCK# signal	(for multiprocessing)	0xF0
LODSB	Load string byte	if (DF==0) AL = *SI++; else AL = *SI--;	0xAC
LODSW	Load string word	if (DF==0) AX = *SI++; else AX = *SI--;	0xAD
LOOP/L OOPx	Loop control	( <i>LOOPE</i> , <i>LOOPNE</i> , <i>LOOPNZ</i> , <i>LOOPZ</i> ) if (x && --CX) goto lbl;	
<u>MOV</u>	Move	copies data from one location to another, (1) r/m = r; (2) r = r/m;	
MOVSB	Move byte from string to string	if (DF==0) *(byte*)DI++ = *(byte*)SI++; else *(byte*)DI-- = *(byte*)SI--;	0xA4
MOVSW	Move word from string to string	if (DF==0) *(word*)DI++ = *(word*)SI++; else *(word*)DI-- = *(word*)SI--;	0xA5
MUL	Unsigned multiply	(1) DX:AX = AX * r/m; (2) AX = AL * r/m;	
NEG	Two's complement negation	r/m *= -1;	
<u>NOP</u>	No operation	opcode equivalent to <i>XCHG EAX, EAX</i> , but it is never optimized to "nothing happens", always using a fixed number of processor ticks	0x90
NOT	Negate the operand, <u>logical NOT</u>	r/m ^= -1;	
OR	<u>Logical OR</u>	(1) r/m  = r/imm; (2) r  = m/imm;	
OUT	Output to port	(1) port[imm] = AL; (2) port[DX] = AL; (3) port[DX] = AX;	
POP	Pop data from <u>stack</u>	r/m = *SP++; POP CS (opcode 0x0F) works only on 8086/8088. Later CPUs use 0x0F as a prefix for newer instructions.	
POPF	Pop data from <u>flags register</u>	FLAGS = *SP++;	0x9D
PUSH	Push data onto stack	*--SP = r/m;	
PUSHF	Push FLAGS onto stack	*--SP = FLAGS;	0x9C
RCL	Rotate left (with carry)		
RCR	Rotate right (with carry)		
REPxx	Repeat MOVS/STOS/ CMPS/LODS/S CAS	( <i>REP</i> , <i>REPE</i> , <i>REPNE</i> , <i>REPNZ</i> , <i>REPZ</i> )	
RET	Return from procedure	Not a real instruction. The assembler will translate these to a RETN or a RETF depending on the memory model of the target	

system.

RETN	Return from near procedure		
RETF	Return from far procedure		
ROL	Rotate left		
ROR	Rotate right		
SAHF	Store AH into FLAGS		0x9E
SAL	<u>Shift</u> (1) r/m <= 1; (2) r/m <= CL; <u>Arithmetically</u> left (signed shift left)		
SAR	Shift            (1) (signed)r/m >= 1; (2) (signed)r/m >= CL; Arithmetically right (signed shift right)		
SBB	Subtraction with borrow	alternative 1-byte encoding of <i>SBB AL, AL</i> is available via <u>undocumented</u> SALC instruction	
SCASB	Compare byte string		0xAE
SCASW	Compare word string		0xAF
SHL	<u>Shift</u> left (unsigned shift left)		
SHR	Shift right (unsigned shift right)		
STC	Set carry flag	CF = 1;	0xF9
STD	Set direction flag	DF = 1;	0xFD
<u>STI</u>	Set interrupt	IF = 1; flag	0xFB
STOSB	Store byte in string	if (DF==0) *ES:DI++ = AL; else *ES:DI-- = AL;	0xAA
STOSW	Store word in string	if (DF==0) *ES:DI++ = AX; else *ES:DI-- = AX;	0xAB
SUB	Subtraction	(1) r/m -= r/imm; (2) r -= m/imm;	
<u>TEST</u>	Logical compare (AND)	(1) r/m & r/imm; (2) r & m/imm;	
WAIT	Wait until not busy	Waits until BUSY# pin is inactive (used with <u>floating-point unit</u> )	0x9B
XCHG	Exchange data	r := r/m; A <u>spinlock</u> typically uses xchg as an <u>atomic operation</u> . <u>(coma bug)</u> .	
XLAT	Table look-up translation	behaves like MOV AL, [BX+AL]	0xD7

XOR      [Exclusive OR](#)    (1) r/m ^= r/imm; (2) r ^= m/imm;

Original 8086/8088 instruction set

## **Added in specific processors**

**Added with [80186/80188](#)**

Instruction	Meaning	Notes
BOUND	Check array index against bounds	raises software interrupt 5 if test fails
ENTER	Enter stack frame	Modifies stack for entry to procedure for high level language. Takes two operands: the amount of storage to be allocated on the stack and the nesting level of the procedure.
INS	Input from port to string	equivalent to IN (E)AX, DX MOV ES:[(E)DI], (E)AX ; adjust (E)DI according to operand size and DF
LEAVE	Leave stack frame	Releases the local stack storage created by the previous ENTER instruction.
OUTS	Output string to port	equivalent to MOV (E)AX, DS:[(E)SI] OUT DX, (E)AX ; adjust (E)SI according to operand size and DF
POPA	Pop all general purpose registers from stack	equivalent to POP DI POP SI POP BP POP AX ;no POP SP here, only ADD SP,2 POP BX POP DX POP CX POP AX
PUSHA	Push all general purpose registers onto stack	equivalent to PUSH AX PUSH CX PUSH DX PUSH BX PUSH SP PUSH BP PUSH SI PUSH DI
PUSH immediate	Push an immediate byte/word value onto the stack	equivalent to PUSH 12h PUSH 1200h
POP immediate	Pop an immediate byte/word value off the stack	equivalent to POP 12h POP 1200h
IMUL immediate	Unsigned multiplication of immediate byte/word value	equivalent to IMUL 12h IMUL 1200h
SHL/SHR/ROL/RO R/RCL/RCR	Rotate/shift bits with an immediate value	equivalent to ROL AX,3

immediate greater than 1 SHR BL,3

### **Added with [80286](#)**

Instruction	Meaning	Notes
ARPL	Adjust RPL field of selector	
CLTS	Clear task-switched flag in register CR0	
LAR	Load access rights byte	
LGDT	Load global descriptor table	
LIDT	Load interrupt descriptor table	
LLDT	Load local descriptor table	
LMSW	Load machine status word	
<a href="#">LOADALL</a>	Load all CPU registers, including internal ones such as GDT	Undocumented, 80286 and 80386 only
LSL	Load segment limit	
<a href="#">LTR</a>	Load task register	
SGDT	Store global descriptor table	
SIDT	Store interrupt descriptor table	
SLDT	Store local descriptor table	
SMSW	Store machine status word	
STR	Store task register	
VERR	Verify a segment for reading	
VERW	Verify a segment for writing	

### **Added with [80386](#)**

Instruction	Meaning	Notes
BSF	Bit scan forward	
BSR	Bit scan reverse	
<a href="#">BT</a>	Bit test	
<a href="#">BTC</a>	Bit test and complement	
<a href="#">BTR</a>	Bit test and reset	
<a href="#">BTS</a>	Bit test and set	
CDQ	Convert double-word to quad-word	Sign-extends EAX into EDX, forming the quad-word EDX:EAX. Since (I)DIV uses EDX:EAX as its input, CDQ must be called after setting EAX if EDX is not manually initialized (as in 64/32 division) before (I) DIV.
CMPSD	Compare string double-word	Compares ES:[(E)DI] with DS:[SI]
CWDE	Convert word to double-word	Unlike CWD, CWDE sign-extends AX to EAX instead of AX to DX:AX
INSD	Input from port to string double-word	
IRET <sub>x</sub>	Interrupt return; D suffix means 32-bit return, F suffix means do not generate epilogue code (i.e. LEAVE instruction)	Use IRETD rather than IRET in 32-bit situations

LFS, LGS	Load far pointer	
LSS	Load stack segment	
LODSD	Load string double-word	can be prefixed with REP
LOOPW, LOOPccW	Loop, conditional loop	Same as LOOP, LOOPcc for earlier processors
LOOPD, LOOPccD	Loop while equal	if (cc && --ECX) goto lbl; cc = Z(ero), E(qual), NonZero, N(on)E(qual)
MOV to/from CR/DR/TR	Move to/from special registers	CR=control registers, DR=debug registers, TR=test registers (up to 80486)
MOVSD	Move string double-word	$*(\text{dword}^*)\text{ES:EDI}\pm\pm = (\text{dword}^*)\text{ESI}\pm\pm$ ; ( $\pm\pm$ depends on DF)
MOVSX	Move with sign-extension	(long)r = (signed char) r/m; and similar
MOVZX	Move with zero-extension	(long)r = (unsigned char) r/m; and similar
OUTSD	Output to port from string double-word	port[DX] = *(long*)ESI $\pm\pm$ ; ( $\pm\pm$ depends on DF)
POPAD	Pop all double-word (32-bit) registers from stack	Does not pop register ESP off of stack
POPFD	Pop data into EFLAGS register	
PUSHAD	Push all double-word (32-bit) registers onto stack	
PUSHFD	Push EFLAGS register onto stack	
SCASD	Scan string data double-word	
SETcc	Set byte to one on condition, zero otherwise	(SETA, SETAE, SETB, SETBE, SETC, SETE, SETG, SETGE, SETL, SETLE, SETNA, SETNAE, SETNB, SETNBE, SETNC, SETNE, SETNG, SETNGE, SETNL, SETNLE, SETNO, SETNP, SETNS, SETNZ, SETO, SETP, SETPE, SETPO, SETS, SETZ)
SHLD	Shift left double-word	
SHRD	Shift right double-word	r1 = r1>>CL   r2<<(32-CL); Instead of CL, immediate 1 can be used
STOSD	Store string double-word	$*\text{ES:EDI}\pm\pm = \text{EAX}$ ; ( $\pm\pm$ depends on DF, ES cannot be overridden)

### Added with [80486](#)

Instruction	Meaning	Notes
BSWAP	Byte Swap	$r = r<<24   r<<8\&0x00FF0000   r>>8\&0x0000FF00   r>>24$ ; Only works for 32 bit registers
CMPXCH G	atomic CoMPare and eXChAnGe	See <a href="#">Compare-and-swap</a>
INVD	Invalidate Internal Caches	Flush internal caches
INVLPG	Invalidate <a href="#">TLB</a> Entry	Invalidate TLB Entry for page that contains data specified
WBINVD	Write Back and Invalidate Cache	Writes back all modified cache lines in the processor's internal cache to main memory and invalidates the internal caches.
XADD	eXchange and ADD	Exchanges the first operand with the second operand, then loads

XADD eXchange and ADD Exchanges the first operand with the second operand, then loads the sum of the two values into the destination operand.

### **Added with Pentium**

Instruction	Meaning	Notes
<a href="#">CPUID</a>	CPU IDentification	Returns data regarding processor identification and features, and returns data to the EAX, EBX, ECX, and EDX registers. Instruction functions specified by the EAX register. <a href="#">[1]</a> This was also added to later <a href="#">80486</a> processors
CMPXCH G8B	CoMPare and eXCHanGe 8 bytes	Compare EDX:EAX with m64. If equal, set ZF and load ECX:EBX into m64. Else, clear ZF and load m64 into EDX:EAX.
<a href="#">RDMSR</a>	ReaD from Model-specific register	Load <a href="#">MSR</a> specified by ECX into EDX:EAX
<a href="#">RDTSC</a>	ReaD Time Stamp Counter	Returns the number of processor ticks since the processor being "ONLINE" (since the last power on of system)
<a href="#">WRMSR</a>	WRite to Model-Specific Register	Write the value in EDX:EAX to <a href="#">MSR</a> specified by ECX
RSM <a href="#">[1]</a>	Resume from System Management Mode	This was introduced by the <a href="#">i386SL</a> and later and is also in the <a href="#">i486SL</a> and later. Resumes from <a href="#">System Management Mode</a> (SMM)

### **Added with Pentium MMX**

Instruction	Meaning	Notes
<a href="#">RDPMC</a>	Read the PMC [Performance Monitoring Counter]	Specified in the ECX register into registers EDX:EAX

Also MMX registers and MMX support instructions were added. They are usable for both integer and floating point operations, see below.

### **Added with AMD K6**

#### **Instruction Meaning Notes**

SYSCALL functionally equivalent to SYSENTER

SYSRET functionally equivalent to SYSEXIT

AMD changed the CPUID detection bit for this feature from the K6-II on.

### **Added with Pentium Pro**

#### **Instruction Meaning Notes**

CMOVcc Conditiona l move (*CMOVA, CMOVAE, CMOVB, CMOVBE, CMOVC, CMOVE, CMOVG, CMOVGE, CMOVL, CMOVLE, CMOVNA, CMOVNAE, CMOVNB, CMOVNBE, CMOVNC, CMOVNE, CMOVNG, CMOVNGE, CMOVNL, CMOVNL, CMOVNO, CMOVNP, CMOVNS, CMOVNZ, CMOVO, CMOVP, CMOVPE, CMOVPO, CMOVS, CMOVZ*)

SYSENTE SYStem call

R ENTER

SYSEXIT SYStem call

EXIT

UD2 Undefined Instruction Generates an invalid opcode. This instruction is provided for software testing to explicitly generate an invalid opcode. The opcode for this instruction is reserved for this purpose.

... ... ... ...

<b>Instruction</b>	<b>Meaning</b>	<b>Notes</b>
MASKMOV Q	Masked Move of Quadword	Selectively write bytes from mm1 to memory location using the byte mask in mm2
MOVNTPS	Move Aligned Four Packed Single-FP Non Temporal	Move packed single-precision floating-point values from xmm to m128, minimizing pollution in the cache hierarchy.
MOVNTQ	Move Quadword Non-Temporal	
PREFETCH 0	Prefetch Data from Address	Prefetch into all cache levels
PREFETCH 1	Prefetch Data from Address	Prefetch into all cache levels EXCEPT L1
PREFETCH 2	Prefetch Data from Address	Prefetch into all cache levels EXCEPT L1 and L2
PREFETCH NTA	Prefetch Data from Address	Prefetch into all cache levels to non-temporal cache structure
SFENCE	Store Fence	Processor hint to make sure all store operations that took place prior to the SFENCE call are globally visible

#### **Added with SSE2**

<b>Instruction</b>	<b>Meaning</b>	<b>Notes</b>
CLFLUSH	Cache Line Flush	Invalidates the cache line that contains the linear address specified with the source operand from all levels of the processor cache hierarchy
LFENCE	Load Fence	Serializes load operations.
MASKMOV DQU	Masked Move of Double Quadword Unaligned	Stores selected bytes from the source operand (first operand) into a 128-bit memory location
MFENCE	Memory Fence	Performs a serializing operation on all load and store instructions that were issued prior the MFENCE instruction.
MOVNTD Q	Move Double Quadword Non-Temporal	Move double quadword from xmm to m128, minimizing pollution in the cache hierarchy.
MOVNTI	Move Doubleword Non-Temporal	Move doubleword from r32 to m32, minimizing pollution in the cache hierarchy.
MOVNTPD	Move Packed Double-Precision Floating-Point Values Non-Temporal	Move packed double-precision floating-point values from xmm to m128, minimizing pollution in the cache hierarchy.
PAUSE	Provides a hint to the processor that the following code is a spin loop	for cacheability

#### **Added with SSE3**

<b>Instruction</b>	<b>Meaning</b>	<b>Notes</b>
LDDQU	Load Unaligned Integer 128 bits	Instructionally equivalent to MOVDQU. Used for video encoding.
MONITOR ECX, EDX	Setup Monitor Address	Sets up a linear address range to be monitored by hardware and activates the monitor.
MWAIT ECX	Monitor Wait	Processor hint to stop instruction execution and enter an implementation-dependent optimized state until occurrence of

ECX implementation-dependent optimized state until occurrence of a class of events.

#### **Added with [x86-64](#)**

Instruction	Meaning	Notes
CDQE	Sign extend EAX into RAX	
CQO	Sign extend RAX into RDX:RAX	
CMPSQ	CoMPare String Quadword	
CMPXCHG16B	CoMPare and eXChAnGe 16 Bytes	
IRETQ	64-bit Return from Interrupt	
JRCXZ	Jump if RCX is zero	
LODSQ	LOaD String Quadword	
MOVSXD	MOV with Sign Extend 32-bit to 64-bit	
POPFQ	POP RFLAGS Register	
PUSHFQ	PUSH RFLAGS Register	
RDTSCP	ReaD Time Stamp Counter and Processor ID	
SCASQ	SCAn String Quadword	
STOSQ	STOre String Quadword	
SWAPGS	Exchange GS base with KernelGSBase MSR	

#### **Added with [AMD-V](#)**

Instruction	Meaning	Notes
CLGI	Clear Global Interrupt Flag	Clears the GIF
INVLPGA	Invalidate TLB entry in a specified ASID	Invalidates the TLB mapping for the virtual page specified in RAX and the ASID specified in ECX.
MOV(CRn)	Move to or from control registers	Moves 32- or 64-bit contents to control register and vice versa.
SKINIT	Secure Init and Jump with Attestation	Verifiable startup of trusted software based on secure hash comparison
STGI	Set Global Interrupt Flag	Sets the GIF.
VMLOAD	Load state From VMCB	Loads a subset of processor state from the VMCB specified by the physical address in the RAX register.
VMMCALL	Call VMM	Used exclusively to communicate with VMM
VMRUN	Run virtual machine	Performs a switch to the guest OS.
VMSAVE	Save state To VMCB	Saves additional guest state to VMCB.

#### **Added with [Intel VT-x](#)**

VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON

#### **Added with [SSE4a](#)**

LZCNT, POPCNT (POPulation CouNT) - advanced bit manipulation

## **x87 floating-point instructions**

### **Original [8087](#) instructions**

Instruction	Meaning	Notes
F2XM1	$(2^x) - 1$	more precise than

for  
*x*  
close to zero

FABS	Absolute value	
FADD	Add	
FADDP	Add and pop	
FBLD	Load BCD	
FBSTP	Store BCD and pop	
FCHS	Change sign	
FCLEX	Clear exceptions	
FCOM	Compare	
FCOMP	Compare and pop	
FCOMPP	Compare and pop twice	
FDECSTP	Decrement floating point stack pointer	
FDISI	Disable interrupts	8087 only, otherwise FNOP
FDIV	Divide	<a href="#">Pentium FDIV bug</a>
FDIVP	Divide and pop	
FDIVR	Divide reversed	
FDIVRP	Divide reversed and pop	
FENI	Enable interrupts	8087 only, otherwise FNOP
FFREE	Free register	
FIADD	Integer add	
FICOM	Integer compare	
FICOMP	Integer compare and pop	
FIDIV	Integer divide	
FIDIVR	Integer divide reversed	
FILD	Load integer	
FIMUL	Integer multiply	
FINCSTP	Increment floating point stack pointer	
FINIT	Initialize floating point processor	
FIST	Store integer	
FISTP	Store integer and pop	
FISUB	Integer subtract	
FISUBR	Integer subtract reversed	
FLD	Floating point load	
FLD1	Load 1.0 onto stack	
FLDCW	Load control word	
FLDENV	Load environment state	
FLDENVV	Load environment state, 16-bit	
FLDL2E	Load $\log_2(e)$ onto stack	
FLDL2T	Load $\log_2(10)$ onto stack	
FLDLG2	Load $\log_{10}(2)$ onto stack	

FLDLG2	Load $\log_{10}(2)$ onto stack	
FLDLN2	Load $\ln(2)$ onto stack	
FLDPI	Load $\pi$ onto stack	
FLDZ	Load 0.0 onto stack	
FMUL	Multiply	
FMULP	Multiply and pop	
FNCLEX	Clear exceptions, no wait	
FNDISI	Disable interrupts, no wait	8087 only, otherwise FNOP
FNENI	Enable interrupts, no wait	8087 only, otherwise FNOP
FNINIT	Initialize floating point processor, no wait	
FNOP	No operation	
FNSAVE	Save FPU state, no wait, 8-bit	
FNSAVEW	Save FPU state, no wait, 16-bit	
FNSTCW	Store control word, no wait	
FNSTENV	Store FPU environment, no wait	
FNSTENV	Store FPU environment, no wait, 16-bit	
FNSTSW	Store status word, no wait	
FPATAN	Partial arctangent	
FPREM	Partial remainder	
FPTAN	Partial tangent	
FRNDINT	Round to integer	
FRSTOR	Restore saved state	
FRSTORW	Restore saved state	Perhaps not actually available in 8087
FSAVE	Save FPU state	
FSAVEW	Save FPU state, 16-bit	
FSCALE	Scale by factor of 2	
FSQRT	Square root	
FST	Floating point store	
FSTCW	Store control word	
FSTENV	Store FPU environment	
FSTENVW	Store FPU environment, 16-bit	
FSTP	Store and pop	
FSTSW	Store status word	
FSUB	Subtract	
FSUBP	Subtract and pop	
FSUBR	Reverse subtract	
FSUBRP	Reverse subtract and pop	
FTST	Test for zero	
FWAIT	Wait while FPU is executing	
FXAM	Examine condition flags	
FXCH	Exchange registers	

FXTRACT	Extract exponent and significand	
FYL2X	$y * \log_2 x$	if $y = \log_b 2$ , then the base- $b$ logarithm is computed
FYL2XP1	$y * \log_2 (x+1)$	more precise than $\log_2 z$ if $x$ is close to zero

## Added in specific processors

Added with [80287](#)

Instruction	Meaning	Notes
FSETPM	Set protected mode	80287 only, otherwise FNOP

Added with [80387](#)

Instruction	Meaning	Notes
FCOS	Cosine	
FLDENVD	Load environment state, 32-bit	
FSAVED	Save FPU state, 32-bit	
FSTENV	Store FPU environment, 32-bit	
FPREM1	Partial remainder	Computes IEEE remainder
FRSTORD	Restore saved state, 32-bit	
FSIN	Sine	
FSINCOS	Sine and cosine	
FSTENV	Store FPU environment, 32-bit	
FUCOM	Unordered compare	
FUCOMP	Unordered compare and pop	
FUCOMPP	Unordered compare and pop twice	

Added with [Pentium Pro](#)

- [FCMOV](#) variants: FCMOVNB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU
- [FCOMI](#) variants: FCOMI, FCOMIP, FUCOMI, FUCOMIP

Added with [SSE](#)

FXRSTOR, FXSAVE

These are also supported on later Pentium IIs which do not contain SSE support

Added with [SSE3](#)

FISTTP (x87 to integer conversion with truncation regardless of status word)

## Undocumented x87 instructions

FFREEP performs FFREE ST(i) and pop stack

## SIMD instructions

### MMX instructions

Added with [Pentium MMX](#)

Instruction	Meaning	Notes
EMMS	Empty MMX Technology State	Marks all x87 FPU registers for use by FPU
MOVD mm, r/m32	Move doubleword	
MOVQ mm, r/m64	Move quadword	
PACKSSDW mm1, mm2/m64	Pack doubleword to word (signed with saturation)	
PACKSSWB mm1, mm2/m64	Pack word to byte (signed with saturation)	

mm2/m64	
PACKUSWB mm1, mm2/m64	Pack word to byte (signed with unsaturation)
PADDB mm, mm/m64	Add packed byte integers
PADDW mm, mm/m64	Add packed word integers
PADDD mm, mm/m64	Add packed doubleword integers
PADDSB mm, mm/m64	Add packed signed byte integers and saturate
PADDSW mm, mm/m64	Add packed signed word integers and saturate
PADDUSB mm, mm/m64	Add packed unsigned byte integers and saturate
PADDUSW mm, mm/m64	Add packed unsigned word integers and saturate
PAND mm, mm/m64	Bitwise AND
PANDN mm, mm/m64	Bitwise AND NOT
POR mm, mm/m64	Bitwise OR
PXOR mm, mm/m64	Bitwise XOR
PCMPEQB mm, mm/m64	Compare packed byte integers for equality
PCMPEQW mm, mm/m64	Compare packed word integers for equality
PCMPEQD mm, mm/m64	Compare packed doubleword integers for equality
PCMPGTB mm, mm/m64	Compare packed signed byte integers for greater than
PCMPGTW mm, mm/m64	Compare packed signed word integers for greater than
PCMPGTD mm, mm/m64	Compare packed signed doubleword integers for greater than
PMADDWD mm, mm/m64	Multiply packed word integers, add adjacent doubleword results
PMULHW mm, mm/m64	Multiply packed signed word integers, store high 16 bit results
PMULLW mm, mm/m64	Multiply packed signed word integers, store low 16 bit results
PSLLW mm, mm/m64	Shift left word, shift in zeros
PSLLD mm, mm/m64	Shift left doubleword, shift in zeros
PSLLQ mm, mm/m64	Shift left quadword, shift in zeros
PSRAD mm, mm/m64	Shift right doubleword, shift in sign bits
PSRAW mm, mm/m64	Shift right word, shift in sign bits
PSRLW mm, mm/m64	Shift right word, shift in zeros
PSRID mm, mm/m64	Shift right doubleword, shift in zeros

PSRLQ mm, mm/m64	Shift right quadword, shift in zeros
PSUBB mm, mm/m64	Subtract packed byte integers
PSUBW mm, mm/m64	Subtract packed word integers
PSUBD mm, mm/m64	Subtract packed doubleword integers
PSUBSB mm, mm/m64	Subtract packed signed byte integers with saturation
PSUBSW mm, mm/m64	Subtract packed signed word integers with saturation
PSUBUSB mm, mm/m64	Subtract packed unsigned byte integers with saturation
PSUBUSW mm, mm/m64	Subtract packed unsigned word integers with saturation
PUNPCKHBW mm, mm/m64	Unpack and interleave high-order bytes
PUNPCKHWD mm, mm/m64	Unpack and interleave high-order words
PUNPCKHDQ mm, mm/m64	Unpack and interleave high-order doublewords
PUNPCKLBW mm, mm/m64	Unpack and interleave low-order bytes
PUNPCKLDQ mm, mm/m64	Unpack and interleave low-order words
PUNPCKLWD mm, mm/m64	Unpack and interleave low-order doublewords

## MMX+ instructions

Added with [Athlon](#)

Same as the SSE [SIMD](#) integer instructions which operated on [MMX](#) registers.

## EMMX instructions

### EMMI instructions

(added with [6x86MX](#) from [Cyrix](#), deprecated now)

PAVEB, PADDIWI, PMAGW, PDISTIB, PSUBSIW, PMVZB, PMULHRW, PMVNZB, PMVLZB, PMVGEZB, PMULHRIW, PMACHRIW

## 3DNow! instructions

Added with [K6-2](#)

FEMMS, PAVGUSB, PF2ID, PFACC, PFADD, PFCMPEQ, PFCMPGE, PFCMPGT, PFMAX, PFMIN, PFMUL, PFRCP, PFRCPIT1, PFRCPIT2, PFRSQIT1, PFRSQRT, PFSUB, PFSUBR, PI2FD, PMULHRW, PREFETCH, PREFETCHW

## 3DNow!+ instructions

Added with [Athlon](#) and [K6-2+](#)

PF2IW, PFNACC, PFPNACC, PI2FW, PSWAPD

Added with [Geode GX](#)

PFRSQRTV, PFRCPV

## SSE instructions

Added with [Pentium III](#)

### SSE SIMD floating-point instructions

ADDPS, ADDSS, CMPPS, CMPSS, COMISS, CVTPI2PS, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTPS2PI, CVTSS2SI, DIVPS, DIVSS, LDMXCSR, MAXPS, MAXSS, MINPS, MINSS, MOVAPS, MOVLHPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVNTPS, MOVSS, MOVUPS, MULPS, MULSS, RCPPS, RCPSS,

MOVLHPS, MOVLPS, MOVMSKPS, MOVNTPS, MOVSS, MOVUPS, MULPS, MULSS, RCPPS, RCPSS, RSQRTPS, RSQRTSS, SHUFP, SQRTPS, SQRTSS, STMCSR, SUBPS, SUBSS, UCOMISS, UNPCKHPS, UNPCKLPS

### SSE SIMD integer instructions

ANDNPS, ANDPS, ORPS, PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSPB, PMULHUW, PSADBW, PSHUFW, XORPS

Instruction	Opcode	Meaning
MOVUPS xmm1, xmm2/m128	OF 10 /r	Move Unaligned Packed Single-Precision Floating-Point Values
MOVSS xmm1, xmm2/m32	F3 OF 10 /r	Move Scalar Single-Precision Floating-Point Values
MOVUPS xmm2/m128, xmm1	OF 11 /r	Move Unaligned Packed Single-Precision Floating-Point Values
MOVSS xmm2/m32, xmm1	F3 OF 11 /r	Move Scalar Single-Precision Floating-Point Values
MOVLPS xmm, m64	OF 12 /r	Move Low Packed Single-Precision Floating-Point Values
MOVHLPS xmm1, xmm2	OF 12 /r	Move Packed Single-Precision Floating-Point Values High to Low
MOVLPS m64, xmm	OF 13 /r	Move Low Packed Single-Precision Floating-Point Values
UNPCKLPS xmm1, xmm2/m128	OF 14 /r	Unpack and Interleave Low Packed Single-Precision Floating-Point Values
UNPCKHPS xmm1, xmm2/m128	OF 15 /r	Unpack and Interleave High Packed Single-Precision Floating-Point Values
MOVHPS xmm, m64	OF 16 /r	Move High Packed Single-Precision Floating-Point Values
MOVLHPS xmm1, xmm2	OF 16 /r	Move Packed Single-Precision Floating-Point Values Low to High
MOVHPS m64, xmm	OF 17 /r	Move High Packed Single-Precision Floating-Point Values
PREFETCHNTA	OF 18 /0	Prefetch Data Into Caches (non-temporal data with respect to all cache levels)
PREFETCH0	OF 18 /1	Prefetch Data Into Caches (temporal data)
PREFETCH1	OF 18 /2	Prefetch Data Into Caches (temporal data with respect to first level cache)
PREFETCH2	OF 18 /3	Prefetch Data Into Caches (temporal data with respect to second level cache)
NOP	OF 1F /0	No Operation
MOVAPS xmm1, xmm2/m128	OF 28 /r	Move Aligned Packed Single-Precision Floating-Point Values
MOVAPS xmm2/m128, xmm1	OF 29 /r	Move Aligned Packed Single-Precision Floating-Point Values
CVTPI2PS xmm, mm/m64	OF 2A /r	Convert Packed Dword Integers to Packed Single-Precision FP Values
CVTSI2SS xmm, r/m32	F3 OF 2A /r	Convert Dword Integer to Scalar Single-Precision FP Value
MOVNTPS m128, xmm	OF 2B /r	Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint
CVTTPS2PI mm, xmm/m64	OF 2C /r	Convert with Truncation Packed Single-Precision FP Values to Packed Dword Integers
CVTTSS2SI r32, xmm/m32	F3 OF 2C /r	Convert with Truncation Scalar Single-Precision FP Value to Dword Integer

CVTPS2PI mm, xmm/m64	OF 2D /r	Convert Packed Single-Precision FP Values to Packed Dword Integers
CVTSS2SI r32, xmm/m32	F3 OF 2D /r	Convert Scalar Single-Precision FP Value to Dword Integer
UCOMISS xmm1, xmm2/m32	OF 2E /r	Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS
COMISS xmm1, xmm2/m32	OF 2F /r	Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS
SQRTPS xmm1, xmm2/m128	OF 51 /r	Compute Square Roots of Packed Single-Precision Floating-Point Values
SQRTSS xmm1, xmm2/m32	F3 OF 51 /r	Compute Square Root of Scalar Single-Precision Floating-Point Value
RSQRTPS xmm1, xmm2/m128	OF 52 /r	Compute Reciprocal of Square Root of Packed Single-Precision Floating-Point Value
RSQRTSS xmm1, xmm2/m32	F3 OF 52 /r	Compute Reciprocal of Square Root of Scalar Single-Precision Floating-Point Value
RCPPS xmm1, xmm2/m128	OF 53 /r	Compute Reciprocal of Packed Single-Precision Floating-Point Values
RCPSS xmm1, xmm2/m32	F3 OF 53 /r	Compute Reciprocal of Scalar Single-Precision Floating-Point Values
ANDPS xmm1, xmm2/m128	OF 54 /r	Bitwise Logical AND of Packed Single-Precision Floating-Point Values
ANDNPS xmm1, xmm2/m128	OF 55 /r	Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values
ORPS xmm1, xmm2/m128	OF 56 /r	Bitwise Logical OR of Single-Precision Floating-Point Values
XORPS xmm1, xmm2/m128	OF 57 /r	Bitwise Logical XOR for Single-Precision Floating-Point Values
ADDPS xmm1, xmm2/m128	OF 58 /r	Add Packed Single-Precision Floating-Point Values
ADDSS xmm1, xmm2/m32	F3 OF 58 /r	Add Scalar Single-Precision Floating-Point Values
MULPS xmm1, xmm2/m128	OF 59 /r	Multiply Packed Single-Precision Floating-Point Values
MULSS xmm1, xmm2/m32	F3 OF 59 /r	Multiply Scalar Single-Precision Floating-Point Values
SUBPS xmm1, xmm2/m128	OF 5C /r	Subtract Packed Single-Precision Floating-Point Values
SUBSS xmm1, xmm2/m32	F3 OF 5C /r	Subtract Scalar Single-Precision Floating-Point Values
MINPS xmm1, xmm2/m128	OF 5D /r	Return Minimum Packed Single-Precision Floating-Point Values
MINSS xmm1, xmm2/m32	F3 OF 5D /r	Return Minimum Scalar Single-Precision Floating-Point Values
DIVPS xmm1, xmm2/m128	OF 5E /r	Divide Packed Single-Precision Floating-Point Values
DIVSS xmm1, xmm2/m32	F3 OF 5E /r	Divide Scalar Single-Precision Floating-Point Values
MAXPS xmm1, xmm2/m128	OF 5F /r	Return Maximum Packed Single-Precision Floating-Point Values
MAXSS xmm1, xmm2/m32	F3 OF 5F /r	Return Maximum Scalar Single-Precision Floating-Point Values

xmm2/m32		values
PSHUFW mm1, mm2/m64, imm8	OF 70 /r ib	Shuffle Packed Words
LDMXCSR m32	OF AE /2	Load MXCSR Register State
STMXCSR m32	OF AE /3	Store MXCSR Register State
SFENCE	OF AE /7	Store Fence
CMPPS xmm1, xmm2/m128, imm8	OF C2 /r ib	Compare Packed Single-Precision Floating-Point Values
CMPSS xmm1, xmm2/m32, imm8	F3 OF C2 /r ib	Compare Scalar Single-Precision Floating-Point Values
PINSRW mm, r32/m16, imm8	OF C4 /r	Insert Word
PEXTRW r32, mm, imm8	OF C5 /r	Extract Word
SHUFPS xmm1, xmm2/m128, imm8	OF C6 /r ib	Shuffle Packed Single-Precision Floating-Point Values
PMOVMSKB r32, mm	OF D7 /r	Move Byte Mask
PMINUB mm1, mm2/m64	OF DA /r	Minimum of Packed Unsigned Byte Integers
PMAXUB mm1, mm2/m64	OF DE /r	Maximum of Packed Unsigned Byte Integers
PAVGB mm1, mm2/m64	OF EO /r	Average Packed Integers
PAVGW mm1, mm2/m64	OF E3 /r	Average Packed Integers
PMULHUW mm1, mm2/m64	OF E4 /r	Multiply Packed Unsigned Integers and Store High Result
MOVNTQ m64, mm	OF E7 /r	Store of Quadword Using Non-Temporal Hint
PMINSW mm1, mm2/m64	OF EA /r	Minimum of Packed Signed Word Integers
PMAXSW mm1, mm2/m64	OF EE /r	Maximum of Packed Signed Word Integers
PSADBW mm1, mm2/m64	OF F6 /r	Compute Sum of Absolute Differences
MASKMOVQ mm1, mm2	OF F7 /r	Store Selected Bytes of Quadword

## SSE2 instructions

Added with [Pentium 4](#) Also see [integer instructions added with Pentium 4](#)

### SSE2 SIMD floating-point instructions

Instruction	Opcode	Meaning
ADDPD xmm1, xmm2/m128	66 OF 58 /r	Add Packed Double-Precision Floating-Point Values
ADDSD xmm1, xmm2/m64	F2 OF 58 /r	Add Low Double-Precision Floating-Point Value
ANDNPD xmm1, xmm2/m128	66 OF 55 /r	Bitwise Logical AND NOT
CMPPD xmm1, xmm2/m128, imm8	66 OF C2 /r ib	Compare Packed Double-Precision Floating-Point Values
CMPSD xmm1, xmm2/m64, imm8	F2 OF C2 /r ib	Compare Low Double-Precision Floating-Point Values

ADDPD, ADDSD, ANDNPD, ANDPD, CMPPD, CMPSD\*, COMISD, CVTDQ2PD, CVTDQ2PS, CVTPD2DQ,  
CVTPD2PI, CVTPD2PS, CVTPI2PD, CVTPS2DQ, CVTPS2PD, CVTSD2SI, CVTSD2SS, CVTSI2SD, CVTSS2SD,  
CVTTPD2DQ, CVTTPD2PI, CVTTPS2DQ, CVTTS2SI, DIVPD, DIVSD, MAXPD, MAXSD, MINPD, MINSD,  
[MOVAPD](#), [MOVHPD](#), MOVLPD, MOVMSKPD, MOVSD\*, MOVUPD, MULPD, MULSD, ORPD, SHUFPD,  
SQRTPD, SQRTSD, SUBPD, SUBSD, UCOMISD, UNPCKHPD, UNPCKLPD, XORPD

- CMPSD and MOVSD have the same name as the [string instruction mnemonics](#) CMPSD (CMPS) and MOVSD (MOVS); however, the former refer to scalar [double-precision floating-points](#)

## SSE2 SIMD integer instructions

MOVDQ2Q, MOVDQA, MOVDQU, MOVQ2DQ, PADDQ, PSUBQ, PMULUDQ, PSHUFHW, PSHUFLW, PSHUFD, PSLLDQ, PSRLDQ, PUNPCKHQDQ, PUNPCKLQDQ

## SSE3 instructions

Added with Pentium 4 supporting SSE3 Also see integer and floating-point instructions added with Pentium 4 SSE3

### SSE3 SIMD floating-point instructions

- ADDSUBPD, ADDSUBPS (for Complex Arithmetic)
- HADDPD, HADDP, HSUBPD, HSUBPS (for Graphics)
- [MOVDDUP](#), MOVSHDUP, MOVSLDUP (for Complex Arithmetic)

## SSSE3 instructions

Added with [Xeon](#) 5100 series and initial [Core 2](#)

- PSIGNW, PSIGND, PSIGNB
- PSHUFB
- PMULHRSW, PMADDUBSW
- PHSUBW, PHSUBSW, PHSUBD
- PHADDW, PHADDSW, PHADD
- PALIGNR
- PABSW, PABSD, PABSB

## SSE4 instructions

### SSE4.1

Added with [Core 2](#) manufactured in [45nm](#)

- MPSADBW
- PHMINPOSUW
- PMULLD, PMULDQ
- DPPS, DPPD
- BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW
- PMINSB, PMAXSB, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINSD, PMAXSD
- ROUNDPS, ROUNDSS, ROUNDPD, ROUNDSD
- INSERTPS, PINSRB, PINSRD/PINSRQ, EXTRACTPS, PEXTRB, PEXTRW, PEXTRD/PEXTRQ
- PMOVSBW, PMOVZBW, PMOVSB, PMOVZBD, PMOVSBQ, PMOVZBQ, PMOVSXWD, PMOVZXWD, PMOVSXWQ, PMOVZXWQ, PMOVSDQ, PMOVZDQ
- PTEST
- PCMPEQQ
- PACKUSDW
- MOVNTDQA

### SSE4a

Added with [Phenom](#) processors

- LZCNT, POPCNT (POPulation CouNT) - advanced bit manipulation
- EXTRQ/INSERTQ
- MOVNTSD/MOVNTSS

### SSE4.2

Added with [Nehalem](#) processors

- CRC32
- PCMPESTRI
- PCMPESTRM
- PCMPISTRI
- PCMPISTRM
- PCMPGTQ

## FMA instructions

Instruction	Opcode	Meaning	Notes
VFMADDPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 69 /r /is4	Fused Multiply-Add of Packed Double-Precision Floating-Point Values	
VFMADDPS xmm0,	C4E3 WvvvvL01	Fused Multiply-Add of Packed Single-	

VFMADDPS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 68 /r /is4	Fused Multiply-Add of Packed Single-Precision Floating-Point Values
VFMADDSD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6B /r /is4	Fused Multiply-Add of Scalar Double-Precision Floating-Point Values
VFMADDSS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6A /r /is4	Fused Multiply-Add of Scalar Single-Precision Floating-Point Values
VFMADDSUBPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 5D /r /is4	Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values
VFMADDSUBPS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 5C /r /is4	Fused Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values
VFMSUBADDPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 5F /r /is4	Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values
VFMSUBADDP S xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 5E /r /is4	Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values
VFMSUBPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6D /r /is4	Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFMSUBPS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6C /r /is4	Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values
VFMSUBSD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6F /r /is4	Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFMSUBSS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 6E /r /is4	Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values
VFNMADDPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 79 /r /is4	Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values
VFNMADDPS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 78 /r /is4	Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values
VFNMADDSD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7B /r /is4	Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values
VFNMADDSS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7A /r /is4	Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values
VFNMSUBPD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7D /r /is4	Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFNMSUBPS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7C /r /is4	Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values
VFNMSUBSD xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7F /r /is4	Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFNMSUBSS xmm0, xmm1, xmm2, xmm3	C4E3 WvvvvL01 7E /r /is4	Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

## Intel AES instructions

Main article: [AES instruction set](#)

6 new instructions.

Instruction	Description
AESENC	Perform one round of an <a href="#">AES</a> encryption flow
AESENCLAST	Perform the last round of an AES encryption flow
AESDEC	Perform one round of an AES decryption flow
AESDECLAST	Perform the last round of an AES decryption flow...

AESKEYGENASSIST Assist in AES round key generation

AESIMC Assist in AES Inverse Mix Columns

## Undocumented instructions

The x86 CPUs contain [undocumented instructions](#) which are implemented on the chips but not listed in some official documents. They can be found in various sources across the Internet, such as [Ralf Brown's Interrupt List](#) and at <http://sandpile.org>.

Mnemonic	Opcode	Description	Status
AAM imm8	D4 imm8	Divide AL by imm8, put the quotient in AH, and the remainder in AL	Available beginning with 8086, documented since Pentium (earlier documentation lists no arguments)
AAD imm8	D5 imm8	Multiplication counterpart of AAM	Available beginning with 8086, documented since Pentium (earlier documentation lists no arguments)
SALC	D6	Set AL depending on the value of the Carry Flag (a 1-byte alternative of SBB AL, AL)	Available beginning with 8086, but only documented since Pentium Pro.
UD1	OF B9	Intentionally undefined instruction, but unlike UD2 this was not published	
ICEBP	F1	Single byte single-step exception / Invoke <a href="#">ICE</a>	Available beginning with 80386, documented (as INT1) since Pentium Pro
<a href="#">LOADALL</a>	OF 05	Loads All Registers from Memory Address 0x000800H	Only available on 80286
Unknown mnemonic	OF 04	Exact purpose unknown, causes CPU hang. (the only way out is CPU reset) <a href="#">[2]</a> In some implementations, emulated through <a href="#">BIOS</a> as a <a href="#">halting</a> sequence. <a href="#">[3]</a>	Only available on 80286
LOADALLD	OF 07	Loads All Registers from Memory Address ES:EDI	Only available on 80386
POP CS	OF	Pop top of the stack into CS segment register (causing a far jump)	Only available on earliest models of 8086. Beginning with 80286 this opcode is used as a prefix for 2-Byte-Instructions
MOV CS,r/m	8E/1	Moves a value from register/memory into CS segment register (causing a far jump)	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception
MOV ES,r/m	8E/4	Moves a value from register/memory into ES segment register	Only available on earliest models of 8086. On 80286 this opcode causes an invalid opcode exception. Beginning with 80386 the value is moved into the FS segment register.
MOV CS,r/m	8E/5	Moves a values from register/memory into CS segment register	Only available on earliest models of 8086. On 80286 this opcode causes an invalid opcode exception. Beginning with 80386 the value is moved into the GS segment register.

MOV SS,r/m	8E/6	Moves a value from register/memory into SS segment register	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception
MOV DS,r/m	8E/7	Moves a value from register/memory into DS segment register	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception

## See also

- [CLMUL](#)
- [XOP](#)
- [F16C](#)
- [FMA](#)
- [RdRand](#)
- [Larrabee extensions](#)
- [Advanced Vector Extensions 2](#)
- [Bit Manipulation Instruction Sets](#)
- [CPUID](#)

## References

1. "[Re: Intel® Processor Identification and the CPUID Instruction](#)". Retrieved 2013-04-21.
2. "[Re: Undocumented opcodes \(HINT\\_NOP\)](#)". Retrieved 2010-11-07.
3. "[Re: Also some undocumented 0Fh opcodes](#)". Retrieved 2010-11-07.
- [Intel Software Developer's Manuals](#)

## External links

	The Wikibook <a href="#">x86 Assembly</a> has a page on the topic of: <a href="#">x86 Instructions</a>
---	--

- [The 8086 / 80286 / 80386 / 80486 Instruction Set](#)
- [Free IA-32 and x86-64 documentation](#), provided by Intel
- [Netwide Assembler Instruction List](#) (from [Netwide Assembler](#))
- [x86 Opcode and Instruction Reference](#)

<a href="#">[hide]</a> • <a href="#">v</a> • <a href="#">t</a> • <a href="#">e</a>	
<b>x86 assembly topics</b>	
<b>Topics</b>	<ul style="list-style-type: none"> <li>• <a href="#">Assembly language</a></li> <li>• <a href="#">Comparison of assemblers</a></li> <li>• <a href="#">Disassembler</a></li> <li>• <a href="#">Instruction set</a></li> <li>• <a href="#">Low-level programming language</a></li> <li>• <a href="#">Machine code</a></li> <li>• <a href="#">Microassembler</a></li> <li>• <a href="#">x86 assembly language</a></li> </ul>
<a href="#">x86 Assemblers</a>	<ul style="list-style-type: none"> <li>• <a href="#">A86/A386</a></li> <li>• <a href="#">FASM</a></li> <li>• <a href="#">GAS</a></li> <li>• <a href="#">HLA</a></li> <li>• <a href="#">MASM</a></li> <li>• <a href="#">NASM</a></li> <li>• <a href="#">TASM</a></li> <li>• <a href="#">WASM</a></li> <li>• <a href="#">YASM</a></li> </ul>
<b>Programming issues</b>	<ul style="list-style-type: none"> <li>• <a href="#">Call stack</a></li> </ul>

- [Carry flag](#)
- [Direction flag](#)
- [Interrupt flag](#)
- [Overflow flag](#)
- [Zero flag](#)
- [Opcode](#)
- [Program counter](#)
- [Processor register](#)
- [x86 calling conventions](#)
- [x86 instruction listings](#)
- [x86 registers](#)

Pegado de <[http://en.wikipedia.org/wiki/X86\\_instruction\\_listings](http://en.wikipedia.org/wiki/X86_instruction_listings)>

# Real Mode Memory

jueves, 10 de julio de 2014

15:12

## Real mode

From Wikipedia, the free encyclopedia



This article **needs additional citations for verification**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be challenged and removed. (November 2012)

### [Microprocessor modes for the x86 architecture](#)

- [Real mode \(Intel 8086\)](#)
- [8080 emulation mode \(NEC V20/V30 only\)](#)
- [Protected mode \(Intel 80286\)](#)
- [Virtual 8086 mode \(Intel 80386\)](#)
- [Unreal mode \(Intel 80386\)](#)
- [System Management Mode \(Intel 386SL\)](#)
- [Long mode \(AMD Opteron\)](#)
- [x86 virtualization \(Intel Pentium 4, models 6x2\)](#)

*First supported platform shown in parentheses*

- V  
t  
e
- V  
t  
e
- V  
t  
e

**Real mode**, also called **real address mode**, is an operating mode of all [x86](#)-compatible [CPUs](#). Real mode is characterized by a 20-bit segmented [memory address](#) space (giving exactly 1 [MiB](#) of addressable memory) and unlimited direct software access to all memory, I/O addresses and peripheral hardware. Real mode provides no support for memory protection, multitasking, or code privilege levels. Before the release of the [80286](#), which introduced [Protected mode](#), real mode was the only available mode for x86 CPUs.<sup>[1]</sup> In the interests of [backwards compatibility](#), all x86 CPUs start in real mode when reset.

## Contents

- [1 History](#)
- [2 Addressing capacity](#)
- [3 A20 line](#)
- [4 Switching to real mode](#)
- [5 See also](#)
- [6 References](#)
- [7 External links](#)

## History

The 286 architecture introduced [protected mode](#), allowing for (among other things) hardware-level memory protection. Using these new features, however, required a new [operating system](#) that was specifically designed for protected mode. Since a primary [design specification](#) of x86 microprocessors is that they are fully backwards compatible with software written for all x86 chips before them, the 286 chip was made to start in 'real mode' – that is, in a mode which turned off the new memory protection features, so that it could run [operating systems](#) written for the [8086](#) and the [80186](#). As of 2014, even the newest x86 CPUs (including [x86-64](#) CPUs) start in real mode at power-on and can run software written for almost any previous x86 chip.

The PC BIOS which IBM introduced operates in real mode, as do the [DOS](#) operating systems ([MS-DOS](#), [DR-DOS](#), etc.). Early versions of [Microsoft Windows](#) ran in real mode, until [Windows 386](#), which ran in protected mode, and the more fully realized [Windows 3.0](#), which could run in either real or protected mode. Windows 3.0 could actually run in two "flavours" of protected mode: "standard mode", which ran using protected mode, and "386-enhanced mode", which is a virtualized version of

standard mode and thus would not run on a 286. Windows 3.1 removed support for real mode, and it was the first mainstream operating environment which required at least an 80286 processor. Almost all modern x86 operating systems ([Unix](#), [Linux](#), [OS/2](#), [Windows 95](#) and later, etc.) switch the CPU into protected mode at startup, but 64-bit operating systems will use this only as another stepping stone to get to [long mode](#). It is worth noting that the protected mode of the 80286 is considerably more primitive than the improved protected mode introduced with the 80386; the latter is sometimes called 386 protected mode, and is the mode most modern 32-bit x86 operating systems run in.

## Addressing capacity

The 8086, 8088, and 80186 have a 20-bit address bus, but the unusual segmented addressing scheme Intel chose for these processors actually produces effective addresses which can have 21 significant bits. This scheme shifts a 16-bit segment number left four bits (making a 20-bit number with four least-significant zeros) before adding to it a 16-bit address offset; the maximum sum occurs when both the segment and offset are 0xFFFF, yielding  $0xFFFF0 + 0xFFFF = 0x10FFEF$ . On the 8086, 8088, and 80186, the result of an effective address that overflows 20 bits is that the address "wraps around" to the zero end of the address range, i.e. it is taken modulo  $2^{20}$  ( $2^{20} = 1048576 = 0x100000$ ). However, the 80286 has 24 address bits and computes effective addresses to 24 bits even in real mode. Therefore, for the segment 0xFFFF and offset greater than 0x000F, the 80286 would actually make an access into the beginning of the second [mebibyte](#) of memory, whereas the 80186 and earlier would access an address equal to [offset]-0x10, which is at the beginning of the first mebibyte. (Note that on the 80186 and earlier, the first [kibibyte](#) of the address space, starting at address 0, is the permanent, immovable location of the interrupt vector table.) So, the actual amount of memory addressable by the 80286 and later x86 CPUs in real mode is  $1 \text{ MiB} + 64 \text{ KiB} - 16 \text{ B} = 1114096 \text{ B}$ .

## A20 line

Main article: [A20 line](#)

Some programs predating the 80286 were designed to take advantage of the wrap-around (modulo) memory addressing behavior, so the 80286 presented a problem for backward compatibility. Forcing the 21st address line (the actual logic signal wire coming out of the chip) to a logic low, representing a zero, results in a modulo- $2^{20}$  effect to match the earlier processors' address arithmetic, but the 80286 has no internal capability to perform this function. When IBM used the 80286 in their [IBM Personal Computer AT](#), they solved this problem by including a software-settable gate to enable or disable (force to zero) the A20 address line, between the A20 pin on the 80286 and the system bus; this is known as Gate-A20 (the A20 gate), and it is still implemented in PC chipsets to this day. Most versions of the HIMEM.SYS extended memory driver for IBM-/MS-DOS famously displayed upon loading a message that they had installed an "A20 handler", a piece of software to control Gate-A20 and coordinate it to the needs of programs. Obviously, in protected mode the A20 line needs to be enabled, or else physical addressing errors will occur, likely leading to a system crash.

## Switching to real mode

Intel introduced protected mode into the x86 family with the intention that operating systems which used it would run entirely in the new mode and that all programs running under a protected mode operating system would run in protected mode as well. Because of the substantial differences between real mode and even the rather limited 286 protected mode, programs written for real mode cannot run in protected mode without being rewritten. Therefore, with a wide base of existing real mode applications which users depended on, abandoning real mode posed problems for the industry, and programmers sought a way to switch between the modes at will. However, Intel, consistent with their intentions for the processor's usage, provided an easy way to switch into protected mode on the 80286 but no easy way to switch back to real mode. Before the 386 the only way to switch from protected mode back to real mode was to reset the processor; after a reset it always starts up in real mode to be compatible with earlier x86 CPUs back to the 8086. Resetting the processor does not clear the system's RAM, so this, while awkward and inefficient, is actually feasible. From protected mode, the processor's state is saved in memory, then the processor is reset, restarts in real mode, and executes some real mode code to restore the saved state from memory. It can then run other real mode code until the program is ready to switch back to protected mode. The switch to real mode is costly in terms of time, but this technique allows protected mode

programs to use services such as BIOS, which runs entirely in real mode (having been designed originally for the [8088](#)-based [IBM Personal Computer](#) model (*machine type*) 5150). This mode-switching technique is also the one used by [DPMI](#) (under real, not emulated, DOS) and [DOS extenders](#) like [DOS/4GW](#) to allow protected mode programs to run under DOS; the DPMI system or DOS extender switches to real mode to invoke DOS or BIOS calls, then switches back to return to the application program which runs in protected mode. This is probably the reason why until Windows ME it was possible to restart the computer to MS DOS mode from within the operating system. [[citation needed](#)] The changing towards the NT kernel resulted in the operating system not needing DOS to boot the computer as well as unable to use it. The need to restart the computer in Real Mode MS DOS declined after Windows 3.1x until it was no longer needed. The only way of currently running DOS applications in Real Mode from within newer versions of Windows is by using emulators such as [DOSBox](#) or [x86 virtualization](#) products.

## See also

- [Unreal mode](#)
- [80386](#)
- [IA-32](#)
- [x86 assembly language](#)
- [Conventional memory](#)

## References

1. [\[1\]](#) A brief x86 history

## External links

- [Code Project Real Mode Tutorial](#)

Pegado de <[http://en.wikipedia.org/wiki/Real\\_mode](http://en.wikipedia.org/wiki/Real_mode)>

# Protected Mode Memory

jueves, 10 de julio de 2014

15:13

## Protected mode

From Wikipedia, the free encyclopedia

This article is about the x86 processor mode. For Internet Explorer Protected Mode, see [Mandatory Integrity Control](#).

### [Micropocessor modes for the x86 architecture](#)

- [Real mode \(Intel 8086\)](#)
- [8080 emulation mode \(NEC V20/V30 only\)](#)
- [Protected mode \(Intel 80286\)](#)
- [Virtual 8086 mode \(Intel 80386\)](#)
- [Unreal mode \(Intel 80386\)](#)
- [System Management Mode \(Intel 386SL\)](#)
- [Long mode \(AMD Opteron\)](#)
- [x86 virtualization \(Intel Pentium 4, models 6x2\)](#)

*First supported platform shown in parentheses*

In computing, **protected mode**, also called **protected virtual address mode**,<sup>[1]</sup> is an operational mode of x86-compatible [central processing units](#) (CPU). It allows [system software](#) to use features such as [virtual memory](#), [paging](#) and safe [multi-tasking](#) designed to increase an operating system's control over [application software](#).<sup>[2][3]</sup>

When a processor that supports x86 protected mode is powered on, it begins executing instructions in [real mode](#), in order to maintain [backwards compatibility](#) with earlier x86 processors.<sup>[4]</sup> Protected mode may only be entered after the system software sets up several descriptor tables and enables the Protection Enable (PE) [bit](#) in the [control register](#) 0 (CR0).<sup>[5]</sup>

Protected mode was first added to the [x86 architecture](#) in 1982,<sup>[6]</sup> with the release of [Intel's 80286](#) (286) processor, and later extended with the release of the [80386](#) (386) in 1985.<sup>[7]</sup> Due to the enhancements added by protected mode, it has become widely adopted and has become the foundation for all subsequent enhancements to the x86 architecture.<sup>[8]</sup>

## Contents

- [1 History](#)
  - [1.1 The 286](#)
  - [1.2 The 386](#)
- [2 386 additions to protected mode](#)
- [3 Entering and exiting protected mode](#)
- [4 Features](#)
  - [4.1 Privilege levels](#)
  - [4.2 Real mode application compatibility](#)
  - [4.3 Virtual 8086 mode](#)
  - [4.4 Segment addressing](#)
    - [4.4.1 Protected mode](#)
    - [4.4.2 286](#)
    - [4.4.3 386](#)
    - [4.4.4 Structure of segment descriptor entry](#)
  - [4.5 Paging](#)
  - [4.6 Multitasking](#)
- [5 Operating systems](#)
- [6 See also](#)
- [7 References](#)
- [8 External links](#)

## History

The [Intel 8086](#), the predecessor to the 286, was originally designed with a 20-bit address bus for its memory.<sup>[9]</sup> This allowed the processor to access  $2^{20}$  bytes of memory, equivalent to 1 megabyte.<sup>[9]</sup> At the time, 1 megabyte was considered a relatively large amount of memory,<sup>[10]</sup> so the designers of the [IBM Personal Computer](#) reserved the first 640 kilobytes for use by applications and the operating system and the remaining 384 kilobytes for the [BIOS](#) (Basic Input/Output System) and memory for add-on devices.<sup>[11]</sup>

As the cost of memory decreased and memory use increased, the 1 MB limitation became a significant problem. [Intel](#) intended to solve this limitation along with others with the release of the 286.<sup>[11]</sup>

## The 286

For more details on this topic, see [Intel 80286](#).

The initial protected mode, released with the 286, was not widely used.<sup>[11]</sup> It was used for example by Microsoft [Xenix](#) (around 1984),<sup>[12]</sup> by [Coherent](#),<sup>[13]</sup> and by [Minix](#).<sup>[14]</sup> Several shortcomings such as the inability to access the BIOS or DOS calls due to inability to switch back to real mode without resetting the processor prevented widespread usage.<sup>[15]</sup> Acceptance was additionally hampered by the fact that the 286 only allowed memory access in 16 bit segments via each of four segment registers, meaning only  $4 * 2^{16}$  bytes, equivalent to 256 kilobytes, could be accessed at a time.<sup>[11]</sup> Because changing a segment register in protected mode caused a 6-byte segment descriptor to be loaded into the CPU from memory, the segment register load instruction took many tens of processor cycles, making it much slower than on the 8086; therefore, the strategy of computing segment addresses on-the-fly in order to access data structures larger than 128 kilobytes (the combined size of the two data segments) became impractical, even for those few programmers who had mastered it on the 8086/8088.

The 286 maintained backwards compatibility with its precursor the 8086 by initially entering [real mode](#) on power up.<sup>[14]</sup> Real mode functioned virtually identically to the 8086, allowing the vast majority of existing 8086 [software](#) to run unmodified on the newer 286. Real mode also served as a more basic mode in which protected mode could be set up, solving a sort of chicken-and-egg problem. To access the extended functionality of the 286, the operating system would set up some tables in memory that controlled memory access in protected mode, set the addresses of those tables into some special registers of the processor, and then set the processor into protected mode. This enabled 24 bit addressing which allowed the processor to access  $2^{24}$  bytes of memory, equivalent to 16 megabytes.<sup>[9]</sup>

## The 386



An Intel 80386 microprocessor

For more details on this topic, see [Intel 80386](#).

With the release of the 386 in 1985,<sup>[7]</sup> many of the issues preventing widespread adoption of the previous protected mode were addressed.<sup>[11]</sup> The 386 was released with an address bus size of 32 bits, which allows for  $2^{32}$  bytes of memory accessing, equivalent to 4 gigabytes.<sup>[16]</sup> The segment sizes were also increased to 32 bits, meaning that the full address space of 4 gigabytes could be accessed without the need to switch between multiple segments.<sup>[16]</sup> In addition to the increased size of the address bus and segment registers, many other new features were added with the intention of increasing operational security and stability.<sup>[17]</sup> Protected mode is now used in virtually all modern

[operating systems](#) which run on the x86 architecture, such as [Microsoft Windows](#), [Linux](#), and many others.<sup>[18]</sup>

Furthermore, learning from the failures of the 286 protected mode to satisfy the needs for [multiuser DOS](#), Intel added a separate [virtual 8086 mode](#)<sup>[19]</sup> which allowed multiple [virtualized](#) 8086 processors to be emulated on the 386. Hardware support for virtualizing the protected mode itself would however have to wait another 20 years.<sup>[20]</sup>

## 386 additions to protected mode

With the release of the 386, the following additional features were added to protected mode:<sup>[21]</sup>

- [Paging](#)
- [32-bit physical and virtual address space](#) (The 32-bit physical address space is not present on the [80386SX](#), and other 386 processor variants which use the older 286 bus.<sup>[21]</sup>)
- 32-bit [segment](#) offsets
- Ability to switch back to real mode without resetting
- [Virtual 8086 mode](#)

## Entering and exiting protected mode

Until the release of the 386, protected mode did not offer a direct method to switch back into real mode once protected mode was entered. [IBM](#) devised a workaround (implemented in the [IBM AT](#)) which involved resetting the CPU via the keyboard controller and saving the system registers, [stack pointer](#) and often the interrupt mask in the real-time clock chip's RAM. This allowed the BIOS to restore the CPU to a similar state and begin executing code before the reset.<sup>[clarification needed]</sup> Later, a [triple fault](#) was used to reset the 286 CPU, which was a lot faster and cleaner than the keyboard controller method (and does not depend on IBM AT-compatible hardware, but will work on any 80286 CPU in any system).

To enter protected mode, the [Global Descriptor Table](#) (GDT) must first be created with a minimum of three entries: a null descriptor, a code segment descriptor and data segment descriptor. In an IBM-compatible machine, the [A20 line](#) (21st address line) also must be enabled to allow the use of all the address lines so that the CPU can access beyond 1 megabyte of memory (Only the first 20 are allowed to be used after power-up, to guarantee compatibility with older software written for the Intel 8088-based [IBM PC](#) and [PC/XT](#) models). After performing those two steps, the PE bit must be set in the CRO register and a far jump must be made to clear the [prefetch input queue](#).

```
; set PE bit  
mov eax, cr0  
or eax, 1  
mov cr0, eax  
  
; far jump (cs = selector of code segment)  
jmp cs:@pm  
  
@pm:  
; Now we are in PM.
```

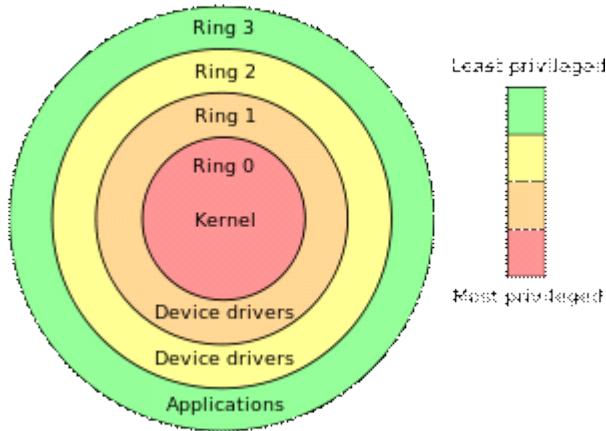
With the release of the 386, protected mode could be exited by loading the segment registers with real mode values, disabling the A20 line and clearing the PE bit in the CRO register, without the need to perform the initial setup steps required with the 286.

## Features

Protected mode has a number of features designed to enhance an operating system's control over application software, in order to increase security and system stability.<sup>[3]</sup> These additions allow the operating system to function in a way that would be significantly more difficult or even impossible without proper hardware support.<sup>[22]</sup>

## Privilege levels

For more details on this topic, see [Ring \(computer security\)](#).



Example of privilege ring usage in an operating system using all rings

In protected mode, there are four privilege levels or [rings](#), numbered from 0 to 3, with ring 0 being the most privileged and 3 being the least. The use of rings allows for system software to restrict tasks from accessing data, [call gates](#) or executing privileged instructions.<sup>[23]</sup> In most environments, the operating system and some [device drivers](#) run in ring 0 and applications run in ring 3.<sup>[23]</sup>

## Real mode application compatibility

According to the *Intel 80286 Programmer's Reference Manual*,<sup>[24]</sup>

“...the 80286 remains upwardly compatible with most 8086 and 80186 application programs. Most 8086 applications programs can be re-compiled or re-assembled and executed on the 80286 in Protected Mode.”

For the most part, the binary compatibility with real-mode code, the ability to access up to 16 MB of physical memory, and 1 GB of [virtual memory](#), were the most apparent changes to application programmers.<sup>[24]</sup> This was not without its limitations, if an application utilized or relied on any of the techniques below it wouldn't run.<sup>[25]</sup>

- Segment arithmetic
- Privileged instructions
- Direct hardware access
- [Writing to a code segment](#)
- Executing data
- Overlapping segments
- Use of BIOS functions, due to the BIOS interrupts being reserved by Intel<sup>[26]</sup>

In reality, almost all [DOS](#) application programs violated these rules.<sup>[27]</sup> Due to these limitations, [virtual 8086 mode](#) was introduced with the 386. Despite such potential setbacks, [Windows 3.0](#) and its successors can take advantage of the binary compatibility with real mode to run many Windows 2.x ([Windows 2.0](#) and [Windows 2.1x](#)) applications, which run in real mode in Windows 2.x, in protected mode.<sup>[28]</sup>

## Virtual 8086 mode

Main article: [Virtual 8086 mode](#)

With the release of the 386, protected mode offers what the Intel manuals call *virtual 8086 mode*. Virtual 8086 mode is designed to allow code previously written for the 8086 to run unmodified and concurrently with other tasks, without compromising security or system stability.<sup>[29]</sup>

Virtual 8086 mode, however, is not completely backwards compatible with all programs. Programs that require segment manipulation, privileged instructions, direct hardware access, or use [self-modifying code](#) will generate an [exception](#) that must be served by the operating system.<sup>[30]</sup> In addition, applications running in virtual 8086 mode generate a [trap](#) with the use of instructions that involve [input/output](#) (I/O), which can negatively impact performance.<sup>[31]</sup>

Due to these limitations, some programs originally designed to run on the 8086 cannot be run in virtual 8086 mode. As a result, system software is forced to either compromise system security or backwards compatibility when dealing with [legacy software](#). An example of such a compromise can be seen with the release of [Windows NT](#), which dropped backwards compatibility for "ill-behaved" DOS applications.<sup>[32]</sup>

## Segment addressing



### Virtual segments of 80286

For more details on this topic, see [X86 memory segmentation](#).

In real mode each logical address points directly into physical memory location, every logical address consists of two 16 bit parts: The segment part of the logical address contains the base address of a segment with a granularity of 16 bytes, i.e. a segments may start at physical address 0, 16, 32, ...,  $2^{20}-16$ . The offset part of the logical address contains an offset inside the segment, i.e. the physical address can be calculated as  $\text{physical\_address} := \text{segment\_part} \times 16 + \text{offset}$  (if the address [line A20](#) is enabled), respectively  $(\text{segment\_part} \times 16 + \text{offset}) \bmod 2^{20}$  (if A20 is off) [clarification needed](#). Every segment has a size of  $2^{16}$  bytes.

### Protected mode

In protected mode the segment\_part is replaced by a 16 bit *selector*, the 13 upper bits (bit 3 to bit 15) of the selector contains the index of an *entry* inside a *descriptor table*. The next bit (bit 2) specifies if the operation is used with the GDT or the LDT. The lowest two bits (bit 1 and bit 0) of the selector are combined to define the privilege of the request; where a value of 0 has the highest priority and value of 3 is the lowest.

The descriptor table entry defines:

- the real *linear* address of the segment
- a limit value for the segment size
- some attribute bits (flags)

### 286

The segment address inside the descriptor table entry has a length of 24 bits so every byte of the physical memory can be defined as bound of the segment. The limit value inside the descriptor table entry has a length of 16 bits so segment length can be between 1 byte and  $2^{16}$  byte. The calculated linear address equals the physical memory address.

### 386

The segment address inside the descriptor table entry is expanded to 32 bits so every byte of the physical memory can be defined as bound of the segment. The limit value inside the descriptor table entry is expanded to 20 bits and completed with a granularity flag (G-bit, for short):

- If G-bit is zero limit has a granularity of 1 byte, i.e. segment size may be 1, 2, ...,  $2^{20}$  bytes.
- If G-bit is one limit has a granularity of  $2^{12}$  bytes, i.e. segment size may be  $1 \times 2^{12}, 2 \times 2^{12}, \dots, 2^{20} \times 2^{12}$  bytes. If paging is off, the calculated linear address equals the physical memory address. If paging is on, the calculated linear address is used as input of paging.

The 386 processor also uses 32 bit values for the address offset.

For maintaining compatibility with 286 protected mode a new default flag (D-bit, for short) was added. If the D-bit of a code segment is off (0) all commands inside this segment will be interpreted as 16-bit commands by default; if it is on (1), they will be interpreted as 32-bit commands.

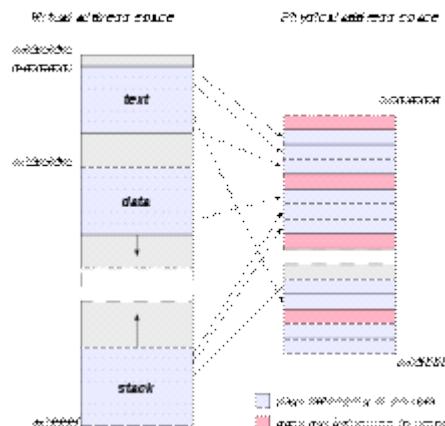
### Structure of segment descriptor entry

B <sup>[a]</sup>	Bits <sup>[b]</sup>	80286	80386	B <sup>[a]</sup>	52	4	unused, available for operating system
0	00..07, 0..7	limit	bits 0..15 of limit	0			
1	08..15, 0..7			1	53	5	reserved, should be zero
2	16..23, 0..7	base address	bits 0..23 of base address	2	54	6	default flag / D-bit
3	24..31,			3	55	7	granularity flag / G-bit

	0..7		Attribute flags #2
4	32..39, 0..7	4	
5	40..47, 0..7	attribute flags #1	5
6	48..51, 0..3	unused	bits 16..19 of 6 limit
	52..55, 4..7		attribute flags #2
7	56..63, 0..7		bits 24..31 of 7 base address

- a. Byte offset inside entry.
- b. First range is the bit offset inside entry; second range is the bit offset inside byte.

## Paging



Common method of using paging to create a virtual address space



paging (intel 80386) with page size of 4K  
For more details on this topic, see [Paging](#).

In addition to adding virtual 8086 mode, the 386 also added **paging** to protected mode.<sup>[33]</sup> Through paging, system software can restrict and control a task's access to pages, which are sections of memory. In many operating systems, paging is used to create an independent virtual address space for each task. This prevents one task from manipulating the memory of another. Paging also allows for pages to be moved out of [primary storage](#) and onto a slower and larger [secondary storage](#), such as a [hard disk](#).<sup>[34]</sup> This allows for more memory to be used than physically available in primary storage.<sup>[34]</sup> The x86 architecture allows control of pages through two [arrays](#): page directories and [page tables](#). Originally, a page directory was the size of one page, 4 kilobytes, and contained 1,024 page directory entries (PDE), although subsequent enhancements to the x86 architecture have added the ability to use larger page sizes. Each PDE contained a [pointer](#) to a page table. A page table was also originally 4 kilobytes in size and contained 1,024 page table entries (PTE). Each PTE contained a pointer to the actual page's physical address and are only used when 4 kilobyte pages are used. At any given time, only one page directory may be in active use.<sup>[35]</sup>

## Multitasking

For more details on this topic, see [Computer multitasking](#).

Through the use of the rings, privileged [call gates](#), and the [Task State Segment](#) (TSS), introduced with the 286, [preemptive multitasking](#) was made possible on the x86 architecture. The TSS allows

general-purpose registers, segment selector fields, and stacks to all be modified without affecting those of another task. The TSS also allows a task's privilege level, and I/O port permissions to be independent of another task's.

In many operating systems, the full features of the TSS are not used.<sup>[36]</sup> This is commonly due to portability concerns or due to the performance issues created with hardware task switches.<sup>[36]</sup> As a result many operating systems use both hardware and software to create a multitasking system.<sup>[37]</sup>

## Operating systems

Operating systems like [OS/2](#) 1.x try to switch the processor between protected and real modes. This is both slow and unsafe, because a real mode program can easily [crash](#) a computer. OS/2 1.x defines restrictive programming rules allowing a *Family API* or *bound* program to run in either real or protected mode. Some early [Unix](#) operating systems, [OS/2](#) 1.x, and Windows used this mode.

[Windows 3.0](#) was able to run real mode programs in 16-bit protected mode. [Windows 3.0](#), when switching to protected mode, decided to preserve the single privilege level model that was used in real mode, which is why Windows applications and DLLs can hook interrupts and do direct hardware access. That lasted through the [Windows 9x](#) series. If a Windows 1.x or 2.x program is written properly and avoids segment arithmetic, it will run the same way in both real and protected modes. Windows programs generally avoid segment arithmetic because Windows implements a software virtual memory scheme, moving program code and data in memory when programs are not running, so manipulating absolute addresses is dangerous; programs should only keep [handles](#) to memory blocks when not running. Starting an old program while Windows 3.0 is running in protected mode triggers a warning dialog, suggesting to either run Windows in real mode or to obtain an updated version of the application. Updating well-behaved programs using the MARK utility with the MEMORY parameter avoids this dialog. It is not possible to have some GUI programs running in 16-bit protected mode and other GUI programs running in real mode. In [Windows 3.1](#) real mode disappeared.

Today, 16-bit protected mode is still used for running applications, e.g. [DPMI](#) compatible [DOS extender](#) programs (through [virtual DOS machines](#)) or Windows 3.x applications (through the [Windows on Windows](#) subsystem) and certain classes of [device drivers](#) (e.g. for changing the screen-resolution using BIOS functionality) in OS/2 2.0 and later, all under control of a 32-bit kernel.

## See also

- [Assembly language](#)
- [Intel](#)
- [Ring \(computer security\)](#)
- [x86 assembly language](#)

## References

1. "[Memory access control method and system for realizing the same](#)" (Patent). *US Patent 5483646*. May 23, 1995. Retrieved 2007-07-14. "The memory access control system according to claim 4, wherein said first address mode is a real address mode, and said second address mode is a protected virtual address mode."
2. "[2.1.3 The Intel 386 Processor \(1985\)](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. [Denver, Colorado: Intel](#). May 2007. p. 35.
3. "[Guide: What does protected mode mean?](#)" (Guide). Delorie software. July 14, 2007. Retrieved 2007-07-14. "The purpose of protected mode is not to protect your program. The purpose is to protect everyone else (including the operating system) from your program."
4. "[3.2 Modes of Operation](#)". *Intel 65 and IA-32 Architectures Software Developer's Manual*. [Denver, Colorado: Intel](#). May 2005. p. 59.
5. Collins, Robert (2007). "[Protected Mode Basics](#)" (PDF). [ftp.utcluj.ro](#). Retrieved 2009-07-31.
6. "[2.1.2 The Intel 286 Processor \(1982\)](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. [Denver, Colorado: Intel](#). May 2007. p. 34.
7. "[Intel Global Citizenship Report 2003](#)" (Timeline). Archived from [the original](#) on 2008-03-22. Retrieved 2007-07-14. "1985 Intel launches Intel386 processor"
8. "[2.1 Brief History of the IA-32 Architecture](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. [Denver, Colorado: Intel](#). May 2007. p. 31.
9. "[A+ - Hardware](#)" (Tutorial/Guide). *PC Microprocessor Developments and Features Tutorials*. BrainBell.com. Retrieved 2007-07-24.

10. Risley, David (March 23, 2001). "[A CPU History](#)" (Article). PCMechanic. Archived from [the original](#) on 2008-01-15. Retrieved 2007-07-24. "What is interesting is that the designers of the time never suspected anyone would ever need more than 1 MB of RAM."
11. Kaplan, Yariv (1997). "[Introduction to Protected-Mode](#)" (Article). Internals.com. Retrieved 2007-07-24.
12. [http://www.tenox.net/docs/microsoft\\_xenix\\_30\\_286\\_press\\_release.pdf](http://www.tenox.net/docs/microsoft_xenix_30_286_press_release.pdf)
13. <http://textfiles.com/internet/FAQ/coherent.faq>
14. <http://minix.net/minix/minix.html>
15. Mueller, Scott (March 24, 2006). "[P2 \(286\) Second-Generation Processors](#)". *Upgrading and Repairing PCs, 17th Edition* (Book) (17 ed.). Que. [ISBN 0-7897-3404-4](#). Retrieved July 2007.
16. "2.1 Memory Organization and Segmentation". *Intel 80386 Programmer's Reference Manual 1986* (Manual). Santa Clara, CA: Intel. 1986.
17. "[3.1 Modes of Operation](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver, Colorado: Intel. May 2007. p. 55.
18. Hyde, Randall (November 2004). "[12.10. Protected Mode Operation and Device Drivers](#)". *Write Great Code*. O'Reilly. [ISBN 1-59327-003-8](#).
19. [Charles Petzold](#), Intel's 32-bit Wonder: The 80386 Microprocessor, *PC Magazine*, November 25, 1986, pp. 150-152
20. <http://images.infoworld.com/d/hardware/sending-software-do-hardwares-job-425>
21. Shvets, Gennadiy (June 3, 2007). "[Intel 80386 processor family](#)" (Article). Retrieved 2007-07-24. "80386SX — low cost version of the 80386. This processor had 16 bit external data bus and 24-bit external address bus."
22. "7 Multitasking". *Intel 80386 Programmer's Reference Manual 1986* (Manual). Santa Clara, CA: Intel. 1986.
23. "[6.3.5 Calls to Other Privilege Levels](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver, Colorado: Intel. May 2007. p. 162.
24. "1.2 Modes of Operation". *Intel 80286 Programmer's Reference Manual 1987* (Manual). Santa Clara, CA: Intel. 1987.
25. "Appendix C 8086/8088 Compatibility Considerations". *Intel 80286 Programmer's Reference Manual 1987* (Manual). Santa Clara, CA: Intel. 1987.
26. "[Memory access control method and system for realizing the same](#)" (Patent). US Patent 5483646. May 6, 1998. Retrieved 2007-07-25. "This has been impossible to-date and has forced BIOS development teams to add support into the BIOS for 32 bit function calls from 32 bit applications."
27. Robinson, Tim (August 26, 2002). "[Virtual 8086 Mode](#)" (Guide). berliOS. Retrieved 2007-07-25. "...secondly, protected mode was also incompatible with the vast amount of real-mode code around at the time."
28. Robinson, Tim (August 26, 2002). "[Virtual 8086 Mode](#)" (Guide). berliOS. Retrieved 2007-07-25.
29. "[15.2 Virtual 8086 Mode](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver, Colorado: Intel. May 2007. p. 560.
30. "[15.2.7 Sensitive Instructions](#)". *Intel 64 and IA-32 Architectures Software Developer's Manual*. Denver, Colorado: Intel. May 2007. p. 568.
31. Robinson, Tim (August 26, 2002). "[Virtual 8086 Mode](#)" (Guide). berliOS. Retrieved 2007-07-25. "A downside to using V86 mode is speed: every IOPL-sensitive instruction will cause the CPU to trap to kernel mode, as will I/O to ports which are masked out in the TSS."
32. Dabak, Prasad; Millind Borate (October 1999). *Undocumented Windows NT* (Book). Hungry Minds. [ISBN 0-7645-4569-8](#).
33. "[ProtectedMode overview \[deinmeister.de\]](#)" (Website). Retrieved 2007-07-29.
34. "[What Is PAE X86?](#)" (Article). Microsoft TechNet. May 28, 2003. Retrieved 2007-07-29. "The paging process allows the operating system to overcome the real physical memory limits. However, it also has a direct impact on performance because of the time necessary to write or retrieve data from disk."
35. Gareau, Jean. "[Advanced Embedded x86 Programming: Paging](#)" (Guide). Embedded.com. Retrieved 2007-07-29. "Only one page directory may be active at a time, indicated by the CR3 register."
36. "[news: Multitasking for x86 explained #1](#)" (Article). NewOrder. NewOrder. May 2, 2004. Archived from [the original](#) on 2007-02-12. Retrieved 2007-07-29. "The reason why software

task switching is so popular is that it can be faster than hardware task switching. Intel never actually developed the hardware task switching, they implemented it, saw that it worked, and just left it there. Advances in multitasking using software have made this form of task switching faster (some say up to 3 times faster) than the hardware method. Another reason is that the Intel way of switching tasks isn't portable at all"

37. "[news: Multitasking for x86 explained #1](#)" (Article). *NewOrder*. NewOrder. May 2, 2004. Archived from [= 10562 the original](#) on 2007-02-12. Retrieved 2007-07-29. "...both rely on the Intel processors ability to switch tasks, they rely on it in different ways."

## External links

- [Protected Mode Basics](#)
- [Introduction to Protected-Mode](#)
- [Overview of the Protected Mode Operations of the Intel Architecture](#)
- [Intel 64 and IA-32 Architectures Software Developer's Manuals](#)
- [TurboIRC.COM tutorial to enter protected mode from DOS](#)
- [Protected Mode Overview and Tutorial](#)
- [Code Project Protected Mode Tutorial](#)
- [Akernelloader switching from real mode to protected mode](#)

<a href="#">[hide]</a> • <a href="#">v</a> • <a href="#">t</a> • <a href="#">e</a>						
<a href="#">Operating system</a>						
<a href="#">General</a>	<ul style="list-style-type: none"> <li>• <a href="#">Advocacy</a></li> <li>• <a href="#">Comparison</a></li> <li>• <a href="#">History</a></li> <li>• <a href="#">Hobbyist development</a></li> <li>• <a href="#">List</a></li> <li>• <a href="#">Timeline</a></li> <li>• <a href="#">Usage share</a></li> </ul>					
	<a href="#">Kernel</a>	<table> <tr> <td><a href="#">Architectures</a></td><td> <ul style="list-style-type: none"> <li>• <a href="#">Exokernel</a></li> <li>• <a href="#">Hybrid</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">Monolithic</a></li> </ul> </td></tr> <tr> <td><a href="#">Components</a></td><td> <ul style="list-style-type: none"> <li>• <a href="#">Device driver</a></li> <li>• <a href="#">Loadable kernel module</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">User space</a></li> </ul> </td></tr> </table>	<a href="#">Architectures</a>	<ul style="list-style-type: none"> <li>• <a href="#">Exokernel</a></li> <li>• <a href="#">Hybrid</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">Monolithic</a></li> </ul>	<a href="#">Components</a>	<ul style="list-style-type: none"> <li>• <a href="#">Device driver</a></li> <li>• <a href="#">Loadable kernel module</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">User space</a></li> </ul>
<a href="#">Architectures</a>	<ul style="list-style-type: none"> <li>• <a href="#">Exokernel</a></li> <li>• <a href="#">Hybrid</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">Monolithic</a></li> </ul>					
<a href="#">Components</a>	<ul style="list-style-type: none"> <li>• <a href="#">Device driver</a></li> <li>• <a href="#">Loadable kernel module</a></li> <li>• <a href="#">Microkernel</a></li> <li>• <a href="#">User space</a></li> </ul>					
<a href="#">Process management</a>	<a href="#">Concepts</a>	<ul style="list-style-type: none"> <li>• <a href="#">Context switch</a></li> <li>• <a href="#">Interrupt</a></li> <li>• <a href="#">IPC</a></li> <li>• <a href="#">Process</a></li> <li>• <a href="#">Process control block</a></li> <li>• <a href="#">Thread</a></li> <li>• <a href="#">Time-sharing</a></li> </ul>				
	<a href="#">Scheduling algorithms</a>	<ul style="list-style-type: none"> <li>• <a href="#">Computer multitasking</a></li> <li>• <a href="#">Fixed-priority preemptive</a></li> <li>• <a href="#">Multilevel feedback queue</a></li> <li>• <a href="#">Preemptive</a></li> <li>• <a href="#">Round-robin</a></li> <li>• <a href="#">Shortest job next</a></li> </ul>				
<a href="#">Memory management and resource protection</a>		<ul style="list-style-type: none"> <li>• <a href="#">Bus error</a></li> <li>• <a href="#">General protection fault</a></li> </ul>				

	<ul style="list-style-type: none"> <li>• <a href="#">Memory protection</a></li> <li>• <a href="#">Paging</a></li> <li>• <a href="#">Security rings</a></li> <li>• <a href="#">Segmentation fault</a></li> <li>• <a href="#">Virtual memory</a></li> </ul>
<a href="#"><u>Storage access and file systems</u></a>	<ul style="list-style-type: none"> <li>• <a href="#">Boot loader</a></li> <li>• <a href="#">Defragmentation</a></li> <li>• <a href="#">Device file</a></li> <li>• <a href="#">File attribute</a></li> <li>• <a href="#">Inode</a></li> <li>• <a href="#">Journal</a></li> <li>• <a href="#">Partition</a></li> <li>• <a href="#">Virtual file system</a></li> <li>• <a href="#">Virtual tape library</a></li> </ul>
<a href="#"><u>List</u></a>	<ul style="list-style-type: none"> <li>• <a href="#">AmigaOS</a></li> <li>• <a href="#">Android</a></li> <li>• <a href="#">BeOS</a></li> <li>• <a href="#">BSD</a></li> <li>• <a href="#">DOS</a></li> <li>• <a href="#">GNU Hurd</a></li> <li>• <a href="#">iOS</a></li> <li>• <a href="#">Linux</a></li> <li>• <a href="#">Mac OS</a></li> <li>• <a href="#">MorphOS</a></li> <li>• <a href="#">OpenVMS</a></li> <li>• <a href="#">OS/2</a></li> <li>• <a href="#">OSv</a></li> <li>• <a href="#">QNX</a></li> <li>• <a href="#">ReactOS</a></li> <li>• <a href="#">RISC OS</a></li> <li>• <a href="#">Solaris</a></li> <li>• <a href="#">TPF</a></li> <li>• <a href="#">Unix</a></li> <li>• <a href="#">VM/CMS</a></li> <li>• <a href="#">Windows</a></li> <li>• <a href="#">z/OS</a></li> </ul>
<a href="#"><u>Miscellaneous concepts</u></a>	<ul style="list-style-type: none"> <li>• <a href="#">API</a></li> <li>• <a href="#">Computer network</a></li> <li>• <a href="#">HAL</a></li> <li>• <a href="#">Live CD</a></li> <li>• <a href="#">Live USB</a></li> <li>• <a href="#">OS shell</a> <ul style="list-style-type: none"> <li>◦ <a href="#">CLI</a></li> <li>◦ <a href="#">GUI</a></li> <li>◦ <a href="#">TUI</a></li> <li>◦ <a href="#">VUI</a></li> </ul> </li> <li>• <a href="#">PXE</a></li> </ul>

Pegado de <[http://en.wikipedia.org/wiki/Protected\\_mode](http://en.wikipedia.org/wiki/Protected_mode)>

## System Instructions

These instructions were added with the Pentium II.

### **sysenter**

This instruction causes the processor to enter protected system mode (supervisor mode or "kernel mode").

**sysexit**

This instruction causes the processor to leave protected system mode, and enter user mode.

Pegado de <[http://en.wikibooks.org/wiki/X86\\_Assembly/Other\\_Instructions#I.2FO\\_Instructions](http://en.wikibooks.org/wiki/X86_Assembly/Other_Instructions#I.2FO_Instructions)>

# C and ASM (Inline assembler)

jueves, 10 de julio de 2014  
21:19

## Calling C functions

You can do a call to any C standard function, and non-standard functions included in your program, by using a normal call in Inline Assembler. Example:

```
Char msg[12] = "Hello world!";

__asm{
    mov ax, msg      ;Mov the address of MSG to AX
    push ax          ;Push it into the stack, to pass it as an argument
    call printf      ;Prints the text into the screen by using C's printf
    Pop bx           ;Clear the stack!
};
```

Any possible return values are stored in AX (or EAX for 32 bit assembler). So, if we make a call to the getchar() function, the keyboard character would be stored in AX after the call.

Also, be aware that **fastcall** functions use the registers to pass the arguments. We want to avoid using fastcalls when using inline assembler.

## \_\_asm Blocks as Macros

C macros offer a convenient way to insert assembly code into your source code, but they demand extra care because a macro expands into a single logical line. To create trouble-free macros, follow these rules:

- Enclose the **\_\_asm** block in braces.
- Put the **\_\_asm** keyword in front of each assembly instruction.
- Use old-style C comments (`/* comment */`) instead of assembly-style comments (`( ; comment)` or single-line C comments (`// comment`)).

To illustrate, the following example defines a simple macro:

```
#define PORTIO __asm      \
/* Port output */          \
{                          \
    __asm mov al, 2        \
    __asm mov dx, 0xD007   \
    __asm out al, dx       \
}
```

At first glance, the last three **\_\_asm** keywords seem superfluous. They are needed, however, because the macro expands into a single line:

```
__asm /* Port output */ { __asm mov al, 2 __asm mov dx, 0xD007 __asm out al, dx }
```

The third and fourth **\_\_asm** keywords are needed as statement separators. The only statement separators recognized in **\_\_asm** blocks are the newline character and **\_\_asm** keyword. Because a block defined as a macro is one logical line, you must separate each instruction with **\_\_asm**.

The braces are essential as well. If you omit them, the compiler can be confused by C or C++ statements on the same line to the right of the macro invocation. Without the closing brace, the compiler cannot tell where assembly code stops, and it sees C or C++ statements after the **\_\_asm** block as assembly instructions.

Assembly-style comments that start with a semicolon (;) continue to the end of the line.

This causes problems in macros because the compiler ignores everything after the comment, all the way to the end of the logical line. The same is true of single-line C or C++ comments ( // comment). To prevent errors, use old-style C comments ( /\* comment \*/ ) in **`__asm`** blocks defined as macros.

An **`__asm`** block written as a C macro can take arguments. Unlike an ordinary C macro, however, an **`__asm`** macro cannot return a value. So you cannot use such macros in C or C++ expressions.

Be careful not to invoke macros of this type indiscriminately. For instance, invoking an assembly-language macro in a function declared with the **`__fastcall`** convention may cause unexpected results. (See [Using and Preserving Registers in Inline Assembly](#).)

## Using and Preserving Registers in Inline Assembly

[Top](#)

In general, you should not assume that a register will have a given value when an **`__asm`** block begins. Register values are not guaranteed to be preserved across separate **`__asm`** blocks. If you end a block of inline code and begin another, you cannot rely on the registers in the second block to retain their values from the first block. An **`__asm`** block inherits whatever register values result from the normal flow of control.

If you use the **`__fastcall`** calling convention, the compiler passes function arguments in registers instead of on the stack. This can create problems in functions with **`__asm`** blocks because a function has no way to tell which parameter is in which register. If the function happens to receive a parameter in EAX and immediately stores something else in EAX, the original parameter is lost. In addition, you must preserve the ECX register in any function declared with **`__fastcall`**.

To avoid such register conflicts, don't use the **`__fastcall`** convention for functions that contain an **`__asm`** block. If you specify the **`__fastcall`** convention globally with the /Gr compiler option, declare every function containing an **`__asm`** block with **`__cdecl`** or **`__stdcall`**. (The **`__cdecl`** attribute tells the compiler to use the C calling convention for that function.) If you are not compiling with /Gr, avoid declaring the function with the **`__fastcall`** attribute.

When using **`__asm`** to write assembly language in C/C++ functions, you don't need to preserve the EAX, EBX, ECX, EDX, ESI, or EDI registers.

For example, in the POWER2.C example in [Writing Functions with Inline Assembly](#), the power2 function doesn't preserve the value in the EAX register. However, using these registers will affect code quality because the register allocator cannot use them to store values across **`__asm`** blocks. In addition, by using EBX, ESI or EDI in inline assembly code, you force the compiler to save and restore those registers in the function prologue and epilogue.

You should preserve other registers you use (such as DS, SS, SP, BP, and flags registers) for the scope of the **`__asm`** block. You should preserve the ESP and EBP registers unless you have some reason to change them (to switch stacks, for example). Also see [Optimizing Inline Assembly](#).

**Note** If your inline assembly code changes the direction flag using the STD or CLD instructions, you must restore the flag to its original value.

Pegado de <[https://homepages.thm.de/~hg13025/vorlesung/msvc\\_asm.html](https://homepages.thm.de/~hg13025/vorlesung/msvc_asm.html)>

## IVT (Interrupt Vector Table)

jueves, 10 de julio de 2014

15:15

300	C5-255	
	ID-255	
82	C5-52	
80	(P-22	Vector 5
78	C5-31	
76	(P-31	Vector 3
10	C5-6	
12	(P-4	Vector 6
14	C5-5	
16	(P-2	Vector 5
18	C5-4	
20	(P-4	Vector 4
22	C5-3	
24	(P-3	Vector 3
26	C5-2	
28	(P-2	Vector 2
30	C5-1	
32	(P-1	Vector 1
34	C5 Value - Vector 0 (C50)	
36	(P Value - Vector 0 (P0)	Vector 0
38	-2.81753	

Pegado de <[http://en.wikipedia.org/wiki/Interrupt\\_vector\\_table](http://en.wikipedia.org/wiki/Interrupt_vector_table)>

An **interrupt vector table**, a concept common across various processor architectures, is a table of interrupt vectors that associates an interrupt handler with an interrupt request in a machine specific way. A dispatch table is one method of implementing an interrupt vector table.

Most processors have an interrupt vector table (IVT), including chips from Infineon, Microchip<sup>[1]</sup>, Atmel,<sup>[2]</sup> Freescale, AMD, Intel, etc.

An interrupt vector table is used in all 3 of the 3 most popular methods of finding the starting address of the interrupt service routine:

The "predefined" method loads the [program counter](#) (PC) directly with the address of some entry inside the interrupt vector table. The [jump table](#) itself contains executable code. While in principle an extremely short interrupt handler could be stored entirely inside the interrupt vector table, in practice the code at each and every entry is "JMP address" where the address is the address of the interrupt service routine (ISR) for that interrupt. The Atmel AVR<sup>[3][4]</sup> and all 8051 and Microchip microcontrollers<sup>[5]</sup> use the predefined approach.

The "fetch" method loads the PC indirectly, using the address of some entry inside the interrupt vector table to pull an address out of that table, and then loading the PC with that address.<sup>[5]</sup> Each and every entry of is the address of an interrupt service routine. All Motorola/Freescale microcontrollers use the fetch method.<sup>[5]</sup>

The "interrupt acknowledge" method, the external device gives the CPU an interrupt handler number. The interrupt acknowledge method is used by the Intel Pentium and many older microprocessors.<sup>[5]</sup>

When the CPU is interrupted by an [interrupt](#), it looks up the [interrupt handler](#) in the interrupt vector table, and transfers control to it.

A commonly used x86 Real Mode interrupt is 0x10, the [VGA BIOS](#) code to handle primitive screen drawing functions.<sup>[\[citation needed\]](#)</sup>

#### **See also**

- **Interrupt Descriptor Table** (x86 Architecture implementation)

## External links

1. ["dsPIC33F Family Reference Manual"](#) section 29.1.1 Interrupt Vector Table
  2. ["AVR Libc User Manual"](#) section: Introduction to avr-libc's interrupt handling
  3. Roger L. Traylor. ["Interrupts: AVR interrupt servicing"](#)
  4. Gary Hill. ["Atmel AVR Interrupt and Timing Subsystems: ATMEGA328P interrupt vector table"](#)

5. Huang, Han-Wat (2005). *Pic Microcontroller: An Introduction to Software and Hardware Interfacing*. Cengage Learning. p. 247. ISBN 978-1-4018-3967-3. Retrieved 22 April 2013.
- [Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide](#)
  - [Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1](#) (see CHAPTER 6, INTERRUPT AND EXCEPTION HANDLING and CHAPTER 10, ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER)]
- [Motorola M68000 Exception and Vector Table](#)

Pegado de <[http://en.wikipedia.org/wiki/Interrupt\\_vector\\_table](http://en.wikipedia.org/wiki/Interrupt_vector_table)>

# Software Interrupt

jueves, 10 de julio de 2014  
22:46

## INT (x86 instruction)

From Wikipedia, the free encyclopedia

INT is an [assembly language](#) instruction for [x86 processors](#) that generates a [software interrupt](#). It takes the interrupt number formatted as a [byte](#) value.<sup>[1]</sup>

When written in assembly language, the instruction is written like this:

INT X

where X is the software interrupt that should be generated (0-255).

Depending on the context, [compiler](#), or [assembler](#), a software interrupt number is often given as a [hexadecimal](#) value, sometimes with a prefix *0x* or the suffix *h*. For example, INT 21H will generate the software interrupt 0x21 (33 in decimal), causing the function pointed to by the 34th vector in the interrupt table to be executed, which is typically an [MS-DOS API](#) call.

## Contents

- [1 Real mode](#)
- [2 INT 3](#)
- [3 See also](#)
- [4 References](#)

## Real mode

When generating a software interrupt, the processor calls one of the 256 functions pointed to by the interrupt address table, which is located in the first 1024 bytes of memory while in [real mode](#) (See [Interrupt vector](#)). It is therefore entirely possible to use a far-call instruction to start the interrupt-function manually after pushing the flag register.

One of the most useful DOS software interrupts was interrupt 0x21. By calling it with different parameters in the registers (mostly ah and al) you could access various IO operations, string output and more.<sup>[2]</sup>

Most [Unix](#) systems and derivatives do not use software interrupts, with the exception of interrupt 0x80, used to make [system calls](#). This is accomplished by entering a 32-bit value corresponding to a kernel function into the EAX register of the processor and then executing INT 0x80.

## INT 3

The INT 3 instruction is defined for use by [debuggers](#) to temporarily replace an instruction in a running program in order to set a [breakpoint](#). Other INT instructions are encoded using two bytes. This makes them unsuitable for use in patching instructions (which can be one byte long); see [SIGTRAP](#).

The opcode for INT 3 is 0xCC, as opposed to the opcode for INT *immediate*', which is 0xCD imm8. Since the dedicated 0xCC opcode has some desired special properties for debugging, which are not shared by the *normal* two-byte opcode for an INT 3, assemblers do not normally generate the generic 0xCD 0x03 opcode from mnemonics.<sup>[3]</sup>

## See also

- [INT 10H](#)
- [INT 13H](#)
- [INT 21H](#)
- [Interrupt](#)
- [Hardware interrupt](#)

- [BIOS interrupt call](#)
- [Ralf Brown's Interrupt List](#)

## References

1. "[Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual](#)". Retrieved 2007-07-13.
2. [Definition of: int 21](#)
3. [Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, and 3C \(PDF\)](#) (PDF). Intel. 2013-06 [1997]. 325462-047US. Retrieved 2013-07-02.

Pegado de <[http://en.wikipedia.org/wiki/INT\\_\(x86\\_instruction\)](http://en.wikipedia.org/wiki/INT_(x86_instruction))>

# Spin Lock

jueves, 10 de julio de 2014  
21:01

In [software engineering](#), a **spinlock** is a [lock](#) which causes a [thread](#) trying to acquire it to simply wait in a loop ("spin") while repeatedly checking if the lock is available. Since the thread remains active but is not performing a useful task, the use of such a lock is a kind of [busy waiting](#). Once acquired, spinlocks will usually be held until they are explicitly released, although in some implementations they may be automatically released if the thread being waited on (that which holds the lock) blocks, or "goes to sleep".

Because they avoid overhead from [operating system process rescheduling](#) or [context switching](#), spinlocks are efficient if [threads](#) are only likely to be blocked for a short period. For this reason, spinlocks are often used inside [operating system kernels](#). However, spinlocks become wasteful if held for longer durations, as they may prevent other threads from running and require rescheduling. The longer a lock is held by a thread, the greater the risk is that the thread will be interrupted by the OS scheduler while holding the lock. If this happens, other threads will be left "spinning" (repeatedly trying to acquire the lock), while the thread holding the lock is not making progress towards releasing it. The result is an indefinite postponement until the thread holding the lock can finish and release it. This is especially true on a single-processor system, where each waiting thread of the same priority is likely to waste its quantum (allocated time where a thread can run) spinning until the thread that holds the lock is finally finished.

Implementing spin locks correctly is difficult because one must take into account the possibility of simultaneous access to the lock, which could cause [race conditions](#). Generally, such implementation is only possible with special [assembly language](#) instructions, such as [atomic test-and-set](#) operations, and cannot be easily implemented in [high-level programming languages](#) or in languages not supporting truly atomic operations.<sup>[1]</sup> On architectures without such operations, or if high-level language implementation is required, a non-atomic locking algorithm may be used, e.g. [Peterson's algorithm](#). But note that such an implementation may require more [memory](#) than a spinlock, be slower to allow progress after unlocking, and may not be implementable in a high-level language if [out-of-order execution](#) is allowed.

## Contents

- [1 Example implementation](#)
- [2 Significant optimizations](#)
- [3 Alternatives](#)
- [4 See also](#)
- [5 References](#)
- [6 External links](#)

## Example implementation

The following example uses x86 assembly language to implement a spinlock. It will work on any [Intel 80386](#) compatible processor.

*; Intel syntax*

```
locked:           ; The lock variable. 1 = locked, 0 = unlocked.  
dd    0
```

```
spin_lock:  
    mov  eax, 1      ; Set the EAX register to 1.
```

```
    xchg eax, [locked] ; Atomically swap the EAX register with  
                      ; the lock variable.
```

```

; This will always store 1 to the lock, leaving
; previous value in the EAX register.

test eax, eax    ; Test EAX with itself. Among other things, this will
; set the processor's Zero Flag if EAX is 0.
; If EAX is 0, then the lock was unlocked and
; we just locked it.
; Otherwise, EAX is 1 and we didn't acquire the lock.

jnz spin_lock   ; Jump back to the MOV instruction if the Zero Flag is
; not set; the lock was previously locked, and so
; we need to spin until it becomes unlocked.

ret             ; The lock has been acquired, return to the calling
; function.

spin_unlock:
    mov eax, 0      ; Set the EAX register to 0.

    xchg eax, [locked] ; Atomically swap the EAX register with
; the lock variable.

ret             ; The lock has been released.

```

## Significant optimizations

The simple implementation above works on all CPUs using the x86 architecture. However, a number of performance optimizations are possible:

On later implementations of the x86 architecture, *spin\_unlock* can safely use an unlocked MOV instead of the slower locked XCHG. This is due to subtle [memory ordering](#) rules which support this, even though MOV is not a full [memory barrier](#). However, some processors (some [Cyrix](#) processors, some revisions of the [Intel Pentium Pro](#) (due to bugs), and earlier [Pentium](#) and [i486 SMP](#) systems) will do the wrong thing and data protected by the lock could be corrupted. On most non-x86 architectures, explicit memory barrier or atomic instructions (as in the example) must be used. On some systems, such as [IA-64](#), there are special "unlock" instructions which provide the needed memory ordering.

To reduce inter-CPU [bus traffic](#), code trying to acquire a lock should loop reading without trying to write anything until it reads a changed value. Because of [MESI](#) caching protocols, this causes the cache line for the lock to become "Shared"; then there is remarkably *no* bus traffic while a CPU waits for the lock. This optimization is effective on all CPU architectures that have a cache per CPU, because MESI is so widespread.

## Alternatives

The primary disadvantage of a spinlock is that, while [waiting](#) to acquire a lock, it wastes time that might be productively spent elsewhere. There are two ways to avoid this:

1. Do not acquire the lock. In many situations it is possible to design data structures that [do not require locking](#), e.g. by using per-thread or per-CPU data and disabling [interrupts](#).
2. [Switch](#) to a different thread while waiting. This typically involves attaching the current thread to a queue of threads waiting for the lock, followed by switching to another thread that is ready to do some useful work. This scheme also has the advantage that it guarantees that [resource starvation](#) does not occur as long as all threads eventually relinquish locks they acquire and scheduling decisions can be made about which thread should progress first. Spinlocks that never entail switching, usable by [real-time operating system](#), are sometimes called *raw spinlocks*.<sup>[2]</sup>

Most operating systems (including [Solaris](#), [Mac OS X](#) and [FreeBSD](#)) use a hybrid approach called "adaptive [mutex](#)". The idea is to use a spinlock when trying to access a resource locked by a currently-running thread, but to sleep if the [thread](#) is not currently running. (The latter is *always* the

case on single-processor systems.)<sup>[3]</sup>

## See also

- [Synchronization](#)
- [Busy spin](#)
- [Deadlock](#)
- [Seqlock](#)
- [Ticket lock](#)

## References

1. Silberschatz, Abraham; Galvin, Peter B. (1994). *Operating System Concepts* (Fourth Edition ed.). Addison-Wesley. pp. 176–179. [ISBN 0-201-59292-4](#).
2. Jonathan Corbet (9 December 2009). "[Spinlock naming resolved](#)". [LWN.net](#). Retrieved 14 May 2013.
3. Silberschatz, Abraham; Galvin, Peter B. (1994). *Operating System Concepts* (Fourth Edition ed.). Addison-Wesley. p. 198. [ISBN 0-201-59292-4](#).

## External links

- [pthread\\_spin\\_lock documentation](#) from The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 Edition
- [Variety of spinlock Implementations](#) from Concurrency Kit
- Article "[User-Level Spin Locks - Threads, Processes & IPC](#)" by [Gert Boddaert](#)
- Paper "[The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors](#)" by [Thomas Anderson](#)
- Paper "[Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors](#)" by [John M. Mellor-Crummey](#) and [Michael L. Scott](#). This paper received the [2006 Dijkstra Prize in Distributed Computing](#).
- [Spin-Wait Lock](#) by [Jeffrey Richter](#)
- [Austria C++ SpinLock Class Reference](#)
- [Interlocked Variable Access\(Windows\)](#)

Pegado de <<http://en.wikipedia.org/wiki/Spinlock>>

# Common BIOS Interrupt List

jueves, 10 de julio de 2014

15:17

## The list of all interrupts that are currently supported by the 8086 assembler emulator.

These interrupts should be compatible with IBM PC and all generations of x86, original Intel 8086 and AMD compatible microprocessors, however Windows XP may overwrite some of the original interrupts.

Quick reference:

<a href="#">INT 10h/00h</a>	<a href="#">INT 10h/1003h</a>	<a href="#">INT 21h</a>	<a href="#">INT 21h/35h</a>	<a href="#">INT 33h/0000h</a>
<a href="#">INT 10h/01h</a>	<a href="#">INT 11h</a>	<a href="#">INT 21h/01h</a>	<a href="#">INT 21h/39h</a>	<a href="#">INT 33h/0001h</a>
<a href="#">INT 10h/02h</a>	<a href="#">INT 12h</a>	<a href="#">INT 21h/02h</a>	<a href="#">INT 21h/3Ah</a>	<a href="#">INT 33h/0002h</a>
<a href="#">INT 10h/03h</a>	<a href="#">INT 13h/00h</a>	<a href="#">INT 21h/05h</a>	<a href="#">INT 21h/3Bh</a>	<a href="#">INT 33h/0003h</a>
<a href="#">INT 10h/05h</a>	<a href="#">INT 13h/02h</a>	<a href="#">INT 21h/06h</a>	<a href="#">INT 21h/3Ch</a>	
<a href="#">INT 10h/06h</a>	<a href="#">INT 13h/03h</a>	<a href="#">INT 21h/07h</a>	<a href="#">INT 21h/3Dh</a>	
<a href="#">INT 10h/07h</a>	<a href="#">INT 15h/86h</a>	<a href="#">INT 21h/09h</a>	<a href="#">INT 21h/3Eh</a>	
<a href="#">INT 10h/08h</a>	<a href="#">INT 16h/00h</a>	<a href="#">INT 21h/0Ah</a>	<a href="#">INT 21h/3Fh</a>	
<a href="#">INT 10h/09h</a>	<a href="#">INT 16h/01h</a>	<a href="#">INT 21h/0Bh</a>	<a href="#">INT 21h/40h</a>	
<a href="#">INT 10h/0Ah</a>	<a href="#">INT 19h</a>	<a href="#">INT 21h/0Ch</a>	<a href="#">INT 21h/41h</a>	
<a href="#">INT 10h/0Ch</a>	<a href="#">INT 1Ah/00h</a>	<a href="#">INT 21h/0Eh</a>	<a href="#">INT 21h/42h</a>	
<a href="#">INT 10h/0Dh</a>	<a href="#">INT 20h</a>	<a href="#">INT 21h/19h</a>	<a href="#">INT 21h/47h</a>	
<a href="#">INT 10h/0Eh</a>		<a href="#">INT 21h/25h</a>	<a href="#">INT 21h/4Ch</a>	
<a href="#">INT 10h/13h</a>		<a href="#">INT 21h/2Ah</a>	<a href="#">INT 21h/56h</a>	
		<a href="#">INT 21h/2Ch</a>		

the short list of supported interrupts with descriptions:

**INT 10h / AH = 0** - set video mode.

*input:*

**AL** = desired video mode.

these video modes are supported:

**00h** - text mode. 40x25. 16 colors. 8 pages.

**03h** - text mode. 80x25. 16 colors. 8 pages.

**13h** - graphical mode. 40x25. 256 colors. 320x200 pixels. 1 page.  
example:

```
mov al, 13h  
mov ah, 0
```

int 10h



**INT 10h / AH = 01h** - set text-mode cursor shape.

*input:*

**CH** = cursor start line (bits 0-4) and options (bits 5-7).

**CL** = bottom cursor line (bits 0-4).

when bit 5 of CH is set to **0**, the cursor is visible. when bit 5 is **1**, the cursor is not visible.

; hide blinking text cursor:

```
    mov ch, 32  
    mov ah, 1  
    int 10h
```

; show standard blinking text cursor:

```
    mov ch, 6  
    mov cl, 7  
    mov ah, 1  
    int 10h
```

; show box-shaped blinking text cursor:

```
    mov ch, 0  
    mov cl, 7  
    mov ah, 1  
    int 10h
```

; note: some bioses required CL to be >=7,  
; otherwise wrong cursor shapes are displayed.



**INT 10h / AH = 2** - set cursor position.

*input:*

**DH** = row.

**DL** = column.

**BH** = page number (0..7).

example:

```
    mov dh, 10  
    mov dl, 20  
    mov bh, 0  
    mov ah, 2  
    int 10h
```



**INT 10h / AH = 03h** - get cursor position and size.

*input:*

**BH** = page number.

*return:*

**DH** = row.

**DL** = column.

**CH** = cursor start line.

**CL** = cursor bottom line.



**INT 10h / AH = 05h** - select active video page.

*input:*

**AL** = new page number (0..7).

the activated page is displayed.



**INT 10h / AH = 06h** - scroll up window.

**INT 10h / AH = 07h** - scroll down window.

*input:*

**AL** = number of lines by which to scroll (00h = clear entire window).

**BH** = [attribute](#) used to write blank lines at bottom of window.

**CH, CL** = row, column of window's upper left corner.

**DH, DL** = row, column of window's lower right corner.



**INT 10h / AH = 08h** - read character and [attribute](#) at cursor position.

*input:*

**BH** = page number.

*return:*

**AH** = [attribute](#).

**AL** = character.



**INT 10h / AH = 09h** - write character and [attribute](#) at cursor position.

*input:*

**AL** = character to display.

**BH** = page number.

**BL** = [attribute](#).

**CX** = number of times to write character.



**INT 10h / AH = 0Ah** - write character only at cursor position.

*input:*

**AL** = character to display.

**BH** = page number.

**CX** = number of times to write character.



**INT 10h / AH = 0Ch** - change color for a single pixel.

*input:*

**AL** = pixel color

**CX** = column.

**DX** = row.

example:

```
mov al, 13h  
mov ah, 0  
int 10h ; set graphics video mode.  
mov al, 1100b  
mov cx, 10  
mov dx, 20  
mov ah, 0ch  
int 10h ; set pixel.
```



**INT 10h / AH = 0Dh** - get color of a single pixel.

*input:*

**CX** = column.

**DX** = row.

*output:*

**AL** = pixel color



**INT 10h / AH = 0Eh** - teletype output.

*input:*

**AL** = character to write.

this function displays a character on the screen, advancing the cursor and scrolling the screen as necessary. the printing is always done to current active page.

example:

```
mov al, 'a'  
mov ah, 0eh  
int 10h  
; note: on specific systems this  
; function may not be supported in graphics mode.
```



**INT 10h / AH = 13h** - write string.

*input:*

**AL** = write mode:

**bit 0**: update cursor after writing;

**bit 1**: string contains attributes.

**BH** = page number.

**BL** = attribute if string contains only characters (bit 1 of AL is zero).

**CX** = number of characters in string (attributes are not counted).

**DL,DH** = column, row at which to start writing.

**ES:BP** points to string to be printed.

example:

```
mov al, 1  
mov bh, 0  
mov bl, 0011_1011b  
mov cx, msg1end - offset msg1 ; calculate message size.  
mov dl, 10  
mov dh, 7  
push cs  
pop es  
mov bp, offset msg1  
mov ah, 13h  
int 10h  
jmp msg1end  
msg1 db " hello, world! "  
msg1end:
```



**INT 10h / AX = 1003h** - toggle intensity/blinking.

*input:*

**BL** = write mode:

**0**: enable intensive colors.

**1**: enable blinking (not supported by the emulator and windows command prompt).

**BH** = 0 (to avoid problems on some adapters).

example:

```
mov ax, 1003h  
mov bx, 0  
int 10h
```

### **bit color table:**

character attribute is 8 bit value, low 4 bits set fore color, high 4 bits set background color.

note: the emulator and windows command line prompt do not support background blinking, however to make colors look the same in dos and in

full screen mode it is required to turn off the background blinking.

HEX	BIN	COLOR
0	0000	black
1	0001	blue
2	0010	green
3	0011	cyan
4	0100	red
5	0101	magenta
6	0110	brown
7	0111	light gray
8	1000	dark gray
9	1001	light blue
A	1010	light green
B	1011	light cyan
C	1100	light red
D	1101	light magenta
E	1110	yellow
F	1111	white

note:

; use this code for compatibility with dos/cmd prompt full screen mode:

```
mov ax, 1003h  
mov bx, 0 ; disable blinking.  
int 10h
```



## **INT 11h - get BIOS equipment list.**

*return:*

**AX** = BIOS equipment list word, actually this call returns the contents of the word at 0040h:0010h.

Currently this function can be used to determine the number of installed number of floppy disk drives.

Bit fields for BIOS-detected installed hardware:

bit(s) Description

15-14 Number of parallel devices.

13 Reserved.

12 Game port installed.

11-9 Number of serial devices.

8 Reserved.

7-6 Number of floppy disk drives (minus 1):

00 single floppy disk;

01 two floppy disks;

10 three floppy disks;

11 four floppy disks.

5-4 Initial video mode:

00 EGA,VGA,PGA, or other with on-board video BIOS;

01 40x25 CGA color.

- 10 80x25 CGA color (emulator default).
- 11 80x25 mono text.
- 3 Reserved.
- 2 PS/2 mouse is installed.
- 1 Math coprocessor installed.
- 0 Set when booted from floppy.



### **INT 12h - get memory size.**

*return:*

**AX** = kilobytes of contiguous memory starting at absolute address 00000h, this call returns the contents of the word at 0040h:0013h.

**Floppy drives are emulated using FLOPPY\_0(..3) files.**

### **INT 13h / AH = 00h - reset disk system.**



### **INT 13h / AH = 02h - read disk sectors into memory.**

### **INT 13h / AH = 03h - write disk sectors.**

*input:*

- AL** = number of sectors to read/write (must be nonzero)
- CH** = cylinder number (0..79).
- CL** = sector number (1..18).
- DH** = head number (0..1).
- DL** = drive number (0..3 , for the emulator it depends on quantity of FLOPPY\_ files).
- ES:BX** points to data buffer.

*return:*

- CF** set on error.
- CF** clear if successful.
- AH** = status (0 - if successful).
- AL** = number of sectors transferred.

Note: each sector has **512** bytes.



### **INT 15h / AH = 86h - BIOS wait function.**

*input:*

- CX:DX** = interval in microseconds

*return:*

- CF** clear if successful (wait interval elapsed),
- CF** set on error or when wait function is already in progress.

*Note:*

the resolution of the wait period is 977 microseconds on many

systems (1 million microseconds - 1 second).  
Windows XP does not support this interrupt (always sets CF=1).



### **INT 16h / AH = 00h** - get keystroke from keyboard (no echo).

*return:*

**AH** = BIOS scan code.

**AL** = ASCII character.

(if a keystroke is present, it is removed from the keyboard buffer).



### **INT 16h / AH = 01h** - check for keystroke in the keyboard buffer.

*return:*

**ZF = 1** if keystroke is not available.

**ZF = 0** if keystroke available.

**AH** = BIOS scan code.

**AL** = ASCII character.

(if a keystroke is present, it is not removed from the keyboard buffer).



### **INT 19h** - system reboot.

Usually, the BIOS will try to read sector 1, head 0, track 0 from drive **A:** to 0000h:7C00h. The emulator just stops the execution, to boot from floppy drive select from the menu: '**virtual drive**' -> '**boot from floppy**'



### **INT 1Ah / AH = 00h** - get system time.

*return:*

**CX:DX** = number of clock ticks since midnight.

**AL** = midnight counter, advanced each time midnight passes.

*notes:*

there are approximately **18.20648** clock ticks per second,  
and **1800B0h** per 24 hours.

**AL** is not set by the emulator.



### **INT 20h** - exit to operating system.

## The short list of emulated MS-DOS interrupts -- INT 21h

DOS file system is emulated in **C:\emu8086\vdrive\x** (x is a drive letter)

If no drive letter is specified and current directory is not set, then **C:\emu8086\MyBuild\** path is used by default. **FLOPPY\_0,1,2,3** files are emulated independently from DOS file system.

For the emulator physical drive **A:** is this file **c:\emu8086\FLOPPY\_0** (for BIOS interrupts: **INT 13h** and boot).

For DOS interrupts (**INT 21h**) drive **A:** is emulated in this subdirectory: **C:\emu8086\vdrive\A\**

Note: DOS file system limits the file and directory names to 8 characters, extension is limited to 3 characters; example of a valid file name: **myfile.txt** (file name = 6 chars, extension - 3 chars). extension is written after the dot, no other dots are allowed.



**INT 21h / AH=1** - read character from standard input, with echo, result is stored in **AL**.

if there is no character in the keyboard buffer, the function waits until any key is pressed.

example:

```
mov ah, 1  
int 21h
```



**INT 21h / AH=2** - write character to standard output.

entry: **DL** = character to write, after execution **AL = DL**.

example:

```
mov ah, 2  
mov dl, 'a'  
int 21h
```



**INT 21h / AH=5** - output character to printer.

entry: **DL** = character to print, after execution **AL = DL**.

example:

```
mov ah, 5  
mov dl, 'a'  
int 21h
```



**INT 21h / AH=6** - direct console input or output.

parameters for output: **DL** = 0..254 (ascii code)

parameters for input: **DL** = 255

for output returns: **AL** = **DL**

for input returns: **ZF** set if no character available and **AL = 00h**, **ZF** clear if character available.

**AL** = character read; buffer is cleared.

example:

```
mov ah, 6  
mov dl, 'a'  
int 21h ; output character.  
  
mov ah, 6  
mov dl, 255  
int 21h ; get character from keyboard buffer (if any) or set ZF=1.
```



**INT 21h / AH=7** - character input without echo to AL.

if there is no character in the keyboard buffer, the function waits until any key is pressed.

example:

```
mov ah, 7  
int 21h
```



**INT 21h / AH=9** - output of a string at **DS:DX**. String must be terminated by '\$'.

example:

```
org 100h  
mov dx, offset msg  
mov ah, 9
```

```
int 21h  
ret  
msg db "hello world $"
```



**INT 21h / AH=0Ah** - input of a string to **DS:DX**, first byte is buffer size, second byte is number of chars actually read. this function does **not** add '\$' in the end of string. to print using **INT 21h / AH=9** you must set dollar character at the end of it and start printing from address **DS:DX + 2**.

example:

```
org 100h  
mov dx, offset buffer  
mov ah, 0ah  
int 21h  
jmp print  
buffer db 10,?, 10 dup(' ')  
print:  
xor bx, bx  
mov bl, buffer[1]  
mov buffer[bx+2], '$'  
mov dx, offset buffer + 2  
mov ah, 9  
int 21h  
ret
```

the function does not allow to enter more characters than the specified buffer size.

see also **int21.asm** in c:\emu8086\examples



**INT 21h / AH=0Bh** - get input status;  
returns: **AL = 00h** if no character available, **AL = 0FFh** if character is available.



**INT 21h / AH=0Ch** - flush keyboard buffer and read standard input.  
entry: **AL** = number of input function to execute after flushing buffer (can be 01h,06h,07h,08h, or 0Ah - for other values the buffer is flushed but no input is attempted); other registers as appropriate for the selected input function.



**INT 21h / AH= 0Eh** - select default drive.

Entry: **DL** = new default drive (0=A:, 1=B:, etc)

Return: **AL** = number of potentially valid drive letters

Notes: the return value is the highest drive present.



**INT 21h / AH= 19h** - get current default drive.

Return: **AL** = drive (0=A:, 1=B:, etc)



**INT 21h / AH=25h** - set interrupt vector;

input: **AL** = interrupt number. **DS:DX** -> new interrupt handler.



**INT 21h / AH=2Ah** - get system date;

return: **CX** = year (1980-2099). **DH** = month. **DL** = day. **AL** = day of week (00h=Sunday)



**INT 21h / AH=2Ch** - get system time;

return: **CH** = hour. **CL** = minute. **DH** = second. **DL** = 1/100 seconds.



**INT 21h / AH=35h** - get interrupt vector;

entry: **AL** = interrupt number;

return: **ES:BX** -> current interrupt handler.



### **INT 21h / AH= 39h** - make directory.

entry: **DS:DX** -> ASCIZ pathname; zero terminated string, for example:

```
org 100h  
mov dx, offset filepath  
mov ah, 39h  
int 21h  
ret  
filepath DB "C:\mydir", 0      ; path to be created.  
end
```

the above code creates **c:\emu8086\vdrive\C\mydir** directory if run by the emulator.

Return: **CF** clear if successful **AX** destroyed. **CF** set on error **AX** = error code.

Note: all directories in the given path must exist except the last one.



### **INT 21h / AH= 3Ah** - remove directory.

Entry: **DS:DX** -> ASCIZ pathname of directory to be removed.

Return:

**CF** is clear if successful, **AX** destroyed **CF** is set on error **AX** = error code.

Notes: directory must be empty (there should be no files inside of it).



### **INT 21h / AH= 3Bh** - set current directory.

Entry: **DS:DX** -> ASCIZ pathname to become current directory (max 64 bytes).

Return:

**Carry Flag** is clear if successful, **AX** destroyed.

**Carry Flag** is set on error **AX** = error code.

Notes: even if new directory name includes a drive letter, the default drive is not changed,  
only the current directory on that drive.



**INT 21h / AH= 3Ch** - create or truncate file.

entry:

**CX** = file attributes:

```
mov cx, 0    ; normal - no attributes.  
mov cx, 1    ; read-only.  
mov cx, 2    ; hidden.  
mov cx, 4    ; system  
mov cx, 7    ; hidden, system and read-only!  
mov cx, 16   ; archive
```

**DS:DX** -> ASCIZ filename.

returns:

**CF** clear if successful, **AX** = file handle.  
**CF** set on error **AX** = error code.

**note: if specified file exists it is deleted without a warning.**

example:

```
org 100h  
mov ah, 3ch  
mov cx, 0  
mov dx, offset filename  
mov ah, 3ch  
int 21h  
jc err  
mov handle, ax  
jmp k  
filename db "myfile.txt", 0  
handle dw ?  
err:  
; ....  
k:  
ret
```



**INT 21h / AH= 3Dh** - open existing file.

Entry:

**AL** = access and sharing modes:

```
mov al, 0 ; read  
mov al, 1 ; write  
mov al, 2 ; read/write  
DS:DX -> ASCIZ filename.
```

Return:

**CF** clear if successful, **AX** = file handle.  
**CF** set on error **AX** = error code.

note 1: file pointer is set to start of file.  
note 2: file must exist.

example:

```
org 100h  
mov al, 2  
mov dx, offset filename  
mov ah, 3dh  
int 21h  
jc err  
mov handle, ax  
jmp k  
filename db "myfile.txt", 0  
handle dw ?  
err:  
; ....  
k:  
ret
```



**INT 21h / AH= 3Eh** - close file.

Entry: **BX** = file handle

Return:

**CF** clear if successful, **AX** destroyed.  
**CF** set on error, **AX** = error code (06h).



**INT 21h / AH= 3Fh** - read from file.

Entry:

**BX** = file handle.  
**CX** = number of bytes to read.

**DS:DX** -> buffer for data.

Return:

**CF** is clear if successful - **AX** = number of bytes actually read; 0 if at EOF (end of file) before call.

**CF** is set on error **AX** = error code.

Note: data is read beginning at current file position, and the file position is updated after a successful read the returned **AX** may be smaller than the request in **CX** if a partial read occurred.



**INT 21h / AH= 40h** - write to file.

entry:

**BX** = file handle.

**CX** = number of bytes to write.

**DS:DX** -> data to write.

return:

**CF** clear if successful; **AX** = number of bytes actually written.

**CF** set on error; **AX** = error code.

note: if **CX** is zero, no data is written, and the file is truncated or extended to the current position data is written beginning at the current file position, and the file position is updated after a successful write the usual cause for **AX < CX** on return is a full disk.



**INT 21h / AH= 41h** - delete file (unlink).

Entry:

**DS:DX** -> ASCIZ filename (no wildcards, but see notes).

return:

**CF** clear if successful, **AX** destroyed. **AL** is the drive of deleted file (undocumented).

**CF** set on error **AX** = error code.

Note: DOS does not erase the file's data; it merely becomes inaccessible because the FAT chain for the file is cleared deleting a file which is

currently open may lead to filesystem corruption.



**INT 21h / AH= 42h** - SEEK - set current file position.

Entry:

**AL** = origin of move: **0** - start of file. **1** - current file position. **2** - end of file.

**BX** = file handle.

**CX:DX** = offset from origin of new file position.

Return:

**CF** clear if successful, **DX:AX** = new file position in bytes from start of file.

**CF** set on error, AX = error code.

Notes:

for origins **1** and **2**, the pointer may be positioned before the start of the file; no error is returned in that case, but subsequent attempts to read or write the file will produce errors. If the new position is beyond the current end of file, the file will be extended by the next write (see [AH=40h](#)).

example:

```
org 100h
mov ah, 3ch
mov cx, 0
mov dx, offset filename
mov ah, 3ch
int 21h ; create file...
mov handle, ax
mov bx, handle
    mov dx, offset data
    mov cx, data_size
    mov ah, 40h
    int 21h ; write to file...
mov al, 0
    mov bx, handle
    mov cx, 0
    mov dx, 7
    mov ah, 42h
    int 21h ; seek...
mov bx, handle
    mov dx, offset buffer
    mov cx, 4
    mov ah, 3fh
```

```
int 21h ; read from file...
mov bx, handle
    mov ah, 3eh
    int 21h ; close file...
    ret
filename db "myfile.txt", 0
    handle dw ?
    data db " hello files! "
    data_size=$-offset data
    buffer db 4 dup(' ')

```



## **INT 21h / AH= 47h** - get current directory.

Entry:

**DL** = drive number (00h = default, 01h = A:, etc)  
**DS:SI** -> 64-byte buffer for ASCIZ pathname.

Return:

**Carry** is clear if successful  
**Carry** is set on error, **AX** = error code (0Fh)

Notes:

the returned path does not include a drive and the initial backslash.



## **INT 21h / AH=4Ch** - return control to the operating system (stop program).



## **INT 21h / AH= 56h** - rename file / move file.

Entry:

**DS:DX** -> ASCIZ filename of existing file.  
**ES:DI** -> ASCIZ new filename.

Return:

**CF** clear if successful.

**CF** set on error, **AX** = error code.

Note: allows move between directories on same logical drive only; open files should not be renamed!



## mouse driver interrupts -- INT 33h

**INT 33h / AX=0000** - mouse initialization. any previous mouse pointer is hidden.

returns:

if successful: **AX**=0FFFFh and **BX**=number of mouse buttons.

if failed: **AX**=0

example:

```
mov ax, 0
```

```
int 33h
```

see also: mouse.asm in examples.



**INT 33h / AX=0001** - show mouse pointer.

example:

```
mov ax, 1
```

```
int 33h
```



**INT 33h / AX=0002** - hide visible mouse pointer.

example:

```
mov ax, 2
```

```
int 33h
```



**INT 33h / AX=0003** - get mouse position and status of its buttons.

returns:

if left button is down: **BX=1**  
if right button is down: **BX=2**  
if both buttons are down: **BX=3**  
**CX = x**  
**DX = y**

example:

```
mov ax, 3  
int 33h  
; note: in graphical 320x200 mode the value of CX is doubled.  
; see mouse2.asm in examples.
```



Click [here](#) to view the list of frequently asked questions.

[IBM](#) ® is a registered trademark of IBM Corporation.  
[Intel](#) ® is a registered trademark of Intel Corporation.  
[AMD](#) ® is a registered trademark of AMD Corporation.  
[Microsoft](#) ® is a registered trademark of Microsoft Corporation.  
All other trademarks mentioned in the emu8086.com web pages  
are the property of their respective holders.

Pegado de <[http://wwwcomputing.dcu.ie/~ray/teaching/CA296/notes/8086\\_bios\\_and\\_dos\\_interrupts.html](http://wwwcomputing.dcu.ie/~ray/teaching/CA296/notes/8086_bios_and_dos_interrupts.html)>

## Common DOS interrupt list

jueves, 10 de julio de 2014

15:17

# DOS INT 21h - DOS Function Codes

The follow abridged list of DOS interrupts has been extracted from a large list compiled by Ralf Brown. These are available on any Simtel mirror (e.g. [sunsite.anu.edu.au](http://sunsite.anu.edu.au)) under the directory ms-dos/info/interNNp.zip

AH	Description	AH	Description
01	<a href="#">Read character from STDIN</a>	02	<a href="#">Write character to STDOUT</a>
05	<a href="#">Write character to printer</a>	06	<a href="#">Console Input/Output</a>
07	<a href="#">Direct char read (STDIN), no echo</a>	08	<a href="#">Char read from STDIN, no echo</a>
09	<a href="#">Write string to STDOUT</a>	0A	<a href="#">Buffered input</a>
0B	<a href="#">Get STDIN status</a>	0C	<a href="#">Flush buffer for STDIN</a>
0D	<a href="#">Disk reset</a>	0E	<a href="#">Select default drive</a>
19	<a href="#">Get current default drive</a>	25	<a href="#">Set interrupt vector</a>
2A	<a href="#">Get system date</a>	2B	<a href="#">Set system date</a>
2C	<a href="#">Get system time</a>	2D	<a href="#">Set system time</a>
2E	<a href="#">Set verify flag</a>	30	<a href="#">Get DOS version</a>
35	<a href="#">Get Interrupt vector</a>		
36	<a href="#">Get free disk space</a>	39	<a href="#">Create subdirectory</a>
3A	<a href="#">Remove subdirectory</a>	3B	<a href="#">Set working directory</a>
3C	<a href="#">Create file</a>	3D	<a href="#">Open file</a>
3E	<a href="#">Close file</a>	3F	<a href="#">Read file</a>
40	<a href="#">Write file</a>	41	<a href="#">Delete file</a>
42	<a href="#">Seek file</a>	43	<a href="#">Get/Set file attributes</a>
47	<a href="#">Get current directory</a>	4C	<a href="#">Exit program</a>
4D	<a href="#">Get return code</a>	54	<a href="#">Get verify flag</a>
56	<a href="#">Rename file</a>	57	<a href="#">Get/Set file date</a>



## AH = 01h - READ CHARACTER FROM STANDARD INPUT, WITH ECHO

Return: AL = character read

Notes:

- ^C/^Break are checked
- ^P toggles the DOS-internal echo-to-printer flag
- ^Z is not interpreted, thus not causing an EOF if input is redirected character is echoed to standard output

SeeAlso: AH=06h,AH=07h,AH=08h,AH=0Ah



## AH = 02h - WRITE CHARACTER TO STANDARD OUTPUT

Entry: DL = character to write

Return: AL = last character output

Notes:

- ^C/^Break are checked

- the last character output will be the character in DL unless DL=09h on entry, in which case AL=20h as tabs are expanded to blanks
- if standard output is redirected to a file, no error checks (write-protected, full media, etc.) are performed

SeeAlso: AH=06h,AH=09h



## AH = 05h - WRITE CHARACTER TO PRINTER

Entry: DL = character to print

Notes:

- keyboard checked for ^C/^Break
- STDPRN is usually the first parallel port, but may be redirected under DOS 2+
- if the printer is busy, this function will wait

SeeAlso: INT 17/AH=00h



## AH = 06h - DIRECT CONSOLE OUTPUT

Entry: DL = character (except FFh)

Return: AL = character output

Notes: does not check ^C/^Break

SeeAlso: AH=02h,AH=09h



## AH = 06h - DIRECT CONSOLE INPUT

Entry: AH = 06h DL = FFh

Return:

- ZF set if no character available and AL = 00h
- ZF clear if character available AL = character read

Notes:

- ^C/^Break are NOT checked
- if the returned character is 00h, the user pressed a key with an extended keycode, which will be returned by the next call of this function
- although the return of AL=00h when no characters are available is not documented, some programs rely on this behavior

SeeAlso: AH=0Bh



## AH=07h - DIRECT CHARACTER INPUT, WITHOUT ECHO

Return: AL = character read from standard input

Notes: does not check ^C/^Break

SeeAlso: AH=01h,AH=06h,AH=08h,AH=0Ah



## AH = 08h - CHARACTER INPUT WITHOUT ECHO

Return: AL = character read from standard input

Notes: ^C/^Break are checked

SeeAlso: AH=01h,AH=06h,AH=07h,AH=0Ah,AH=64h



## AH = 09h - WRITE STRING TO STANDARD OUTPUT

Entry: DS:DX -> '\$'-terminated string

Return: AL = 24h

Notes: ^C/^Break are checked

SeeAlso: AH=02h,AH=06h"OUTPUT"



## AH = 0Ah - BUFFERED INPUT

Entry: DS:DX -> [buffer \(see below\)](#)

Return: buffer filled with user input

Notes:

- ^C/^Break are checked
- reads from standard input

SeeAlso: AH=0Ch

Format of DOS input buffer:

Offset	Size	Description
00	1	maximum characters buffer can hold
01	1	number of chars from last input which may be recalled OR number of characters actually read, excluding CR
02	n	actual characters read, including the final carriage return



## AH=0Bh - GET STDIN STATUS

Return:

- AL = 00h if no character available
- AL = FFh if character is available

Notes: ^C/^Break are checked

SeeAlso: AH=06h"INPUT"



## AH = 0Ch - FLUSH BUFFER AND READ STANDARD INPUT

Entry:

- AL = STDIN input function to execute after flushing buffer
- other registers as appropriate for the input function

Return: as appropriate for the specified input function

Note: if AL is not one of 01h,06h,07h,08h, or 0Ah, the buffer is flushed but no input is attempted

SeeAlso: AH=01h,AH=06h"INPUT",AH=07h,AH=08h,AH=0Ah



## AH = 0Dh - DISK RESET

Notes: This function writes all modified disk buffers to disk, but does not update the directory information

SeeAlso: AX=5D01h



## AH = 0Eh - SELECT DEFAULT DRIVE

Entry: DL = new default drive (0=A:, 1=B:, etc)

Return: AL = number of potentially valid drive letters

Notes: the return value is the highest drive present

SeeAlso: AH=19h,AH=3Bh,AH=DBh



## AH = 19h - GET CURRENT DEFAULT DRIVE

Return: AL = drive (0=A:, 1=B:, etc)

SeeAlso: AH=0Eh,AH=47h,AH=BBh



## AH = 25h - SET INTERRUPT VECTOR

Entry:

- AL = interrupt number
- DS:DX -> new interrupt handler

Notes: this function is preferred over direct modification of the interrupt vector table

SeeAlso: AX=2501h,AH=35h



## AH = 2Ah - GET SYSTEM DATE

Return: CX = year (1980-2099) DH = month DL = day AL = day of week (00h=Sunday)

SeeAlso: AH=2Bh"DOS",AH=2Ch,AH=E7h



## AH = 2Bh - SET SYSTEM DATE

Entry: CX = year (1980-2099) DH = month DL = day

Return:

- AL = 00 successful
- FFh invalid date, system date unchanged

Note: DOS 3.3+ also sets CMOS clock

SeeAlso: AH=2Ah,AH=2Dh



## AH = 2Ch - GET SYSTEM TIME

Return: CH = hour CL = minute DH = second DL = 1/100 seconds

Note: on most systems, the resolution of the system clock is about 5/100sec, so returned times generally do not increment by 1 on some systems, DL may always return 00h

SeeAlso: AH=2Ah,AH=2Dh,AH=E7h



## AH = 2Dh - SET SYSTEM TIME

Entry: CH = hour CL = minute DH = second DL = 1/100 seconds

Return:

- AL = 00h successful
- FFh if invalid time, system time unchanged

Note: DOS 3.3+ also sets CMOS clock

SeeAlso: AH=2Bh"DOS",AH=2Ch



## AH = 2Eh - SET VERIFY FLAG

Entry: AL = new state of verify flag (00 off, 01h on)

Notes:

- default state at system boot is OFF
- when ON, all disk writes are verified provided the device driver supports read-after-write verification

SeeAlso: AH=54h



## AH=30h - GET DOS VERSION

Entry: AL = what to return in BH (00h OEM number, 01h version flag)

Return:

- AL = major version number (00h if DOS 1.x)
- AH = minor version number
- BL:CX = 24-bit user serial number (most versions do not use this) if DOS <5 or AL=00h
- BH = MS-DOS OEM number if DOS 5+ and AL=01h
- BH = version flag bit 3: DOS is in ROM other: reserved (0)

Notes:

- DOS 4.01 and 4.02 identify themselves as version 4.00
- MS-DOS 6.21 reports its version as 6.20; version 6.22 returns the correct value
- Windows95 returns version 7.00 (the underlying MS-DOS)

SeeAlso: AX=3000h/BX=3000h,AX=3306h,AX=4452h



## AH=35h - GET INTERRUPT VECTOR

Entry: AL = interrupt number

Return: ES:BX -> current interrupt handler

SeeAlso: AH=25h,AX=2503h



## AH = 36h - GET FREE DISK SPACE

Entry: DL = drive number (0=default, 1=A:, etc)

Return:

- AX = FFFFh if invalid drive
- AX = sectors per cluster BX = number of free clusters CX = bytes per sector DX = total clusters

on drive

Notes:

- free space on drive in bytes is AX \* BX \* CX
- total space on drive in bytes is AX \* CX \* DX
- "lost clusters" are considered to be in use
- this function does not return proper results on CD-ROMs; use AX=4402h"CD-ROM" instead

SeeAlso: AH=1Bh,AH=1Ch,AX=4402h"CD-ROM"



## AH = 39h - "MKDIR" - CREATE SUBDIRECTORY

Entry: DS:DX -> ASCIZ pathname

Return:

- CF clear if successful AX destroyed
- CF set on error AX = error code (03h,05h)

Notes:

- all directories in the given path except the last must exist
- fails if the parent directory is the root and is full
- DOS 2.x-3.3 allow the creation of a directory sufficiently deep that it is not possible to make that directory the current directory because the path would exceed 64 characters

SeeAlso: AH=3Ah,AH=3Bh,AH=6Dh



## AH = 3Ah - "RMDIR" - REMOVE SUBDIRECTORY

Entry: DS:DX -> ASCIZ pathname of directory to be removed

Return:

- CF clear if successful, AX destroyed
- CF set on error AX = error code (03h,05h,06h,10h)

Notes: directory must be empty (contain only '.' and '..' entries)

SeeAlso: AH=39h,AH=3Bh



## AH = 3Bh - "CHDIR" - SET CURRENT DIRECTORY

Entry: DS:DX -> ASCIZ pathname to become current directory (max 64 bytes)

Return:

- CF clear if successful, AX destroyed
- CF set on error AX = error code (03h)

Notes: if new directory name includes a drive letter, the default drive is not changed, only the current directory on that drive

SeeAlso: AH=47h,AH=71h,INT 2F/AX=1105h



## AH = 3Ch - "CREAT" - CREATE OR TRUNCATE FILE

Entry:

- CX = [file attributes](#)
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX = file handle
- CF set on error AX = error code (03h,04h,05h)

Notes: if a file with the given name exists, it is truncated to zero length

SeeAlso: AH=16h,AH=3Dh,AH=5Ah,AH=5Bh



## AH = 3Dh - "OPEN" - OPEN EXISTING FILE

Entry:

- AL = access and sharing modes
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX = file handle
- CF set on error AX = error code (01h,02h,03h,04h,05h,0Ch,56h)

Notes:

- file pointer is set to start of file
- file handles which are inherited from a parent also inherit sharing and access restrictions
- files may be opened even if given the hidden or system attributes

SeeAlso: AH=0Fh,AH=3Ch,AX=4301h,AX=5D00h



## AH = 3Eh - "CLOSE" - CLOSE FILE

Entry: BX = file handle

Return:

- CF clear if successful, AX destroyed
- CF set on error, AX = error code (06h)

Note: if the file was written to, any pending disk writes are performed, the time and date stamps are set to the current time, and the directory entry is updated

SeeAlso: AH=10h,AH=3Ch,AH=3Dh



## AH = 3Fh - "READ" - READ FROM FILE OR DEVICE

Entry:

- BX = file handle
- CX = number of bytes to read
- DS:DX -> buffer for data

Return:

- CF clear if successful - AX = number of bytes actually read (0 if at EOF before call)
- CF set on error AX = error code (05h,06h)

Notes:

- data is read beginning at current file position, and the file position is updated after a successful read
- the returned AX may be smaller than the request in CX if a partial read occurred
- if reading from CON, read stops at first CR

SeeAlso: AH=27h,AH=40h,AH=93h



## AH=40h - "WRITE" - WRITE TO FILE OR DEVICE

Entry:

- BX = file handle
- CX = number of bytes to write
- DS:DX -> data to write

Return:

- CF clear if successful - AX = number of bytes actually written
- CF set on error - AX = error code (05h,06h)

Notes:

- if CX is zero, no data is written, and the file is truncated or extended to the current position
- data is written beginning at the current file position, and the file position is updated after a successful write
- the usual cause for AX < CX on return is a full disk

SeeAlso: AH=28h,AH=3Fh



## AH = 41H - "UNLINK" - DELETE FILE

Entry:

- DS:DX -> ASCIZ filename (no wildcards, but see notes)
- CL = attribute mask for deletion (server call only, see notes)

Return:

- CF clear if successful, AX destroyed (DOS 3.3) AL seems to be drive of deleted file
- CF set on error AX = error code (02h,03h,05h)

Notes:

- (DOS 3.1+) wildcards are allowed if invoked via AX=5D00h, in which case the filespec must be canonical (as returned by AH=60h), and only files matching the attribute mask in CL are

deleted

- DOS does not erase the file's data; it merely becomes inaccessible because the FAT chain for the file is cleared
- deleting a file which is currently open may lead to filesystem corruption.

SeeAlso: AH=13h,AX=4301h,AX=4380h,AX=5D00h,AH=60h,AH=71h



## AH=42h - "LSEEK" - SET CURRENT FILE POSITION

Entry:

- AL = origin of move 00h start of file 01h current file position 02h end of file
- BX = file handle
- CX:DX = offset from origin of new file position

Return:

- CF clear if successful, DX:AX = new file position in bytes from start of file
- CF set on error, AX = error code (01h,06h)

Notes:

- for origins 01h and 02h, the pointer may be positioned before the start of the file; no error is returned in that case, but subsequent attempts at I/O will produce errors
- if the new position is beyond the current end of file, the file will be extended by the next write (see AH=40h)

SeeAlso: AH=24h



## AH=43 - GET FILE ATTRIBUTES

Entry:

- AL = 00h
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful CX = [file attributes](#)
- CF set on error, AX = error code (01h,02h,03h,05h)

BUG: Windows for Workgroups returns error code 05h (access denied) instead of error code 02h (file not found) when attempting to get the attributes of a nonexistent file.

SeeAlso: AX=4301h,AX=4310h,AX=7143h,AH=B6h



## AH=43 - "CHMOD" - SET FILE ATTRIBUTES

Entry:

- AL = 01h
- CX = new [file attributes](#)
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX destroyed
- CF set on error, AX = error code (01h,02h,03h,05h)

Notes:

- will not change volume label or directory attribute bits, but will change the other attribute bits of a directory
- MS-DOS 4.01 reportedly closes the file if it is currently open

SeeAlso: AX=4300h,AX=4311h,AX=7143h,INT 2F/AX=110Eh

Bitfields for file attributes:

Bits	7	6	5	4	3	2	1	0
Description	shareable	-	archive	directory	vol. label	system	hidden	read-only



## AH = 47h - "CWD" - GET CURRENT DIRECTORY

Entry:

- DL = drive number (00h = default, 01h = A:, etc)
- DS:SI -> 64-byte buffer for ASCIZ pathname

Return:

- CF clear if successful
- CF set on error, AX = error code (0Fh)

Notes:

- the returned path does not include a drive or the initial backslash
- many Microsoft products for Windows rely on AX being 0100h on success

SeeAlso: AH=19h,AH=3Bh,AH=71h



## AH = 4Ch - "EXIT" - TERMINATE WITH RETURN CODE

Entry: AL = return code

Return: never returns

Notes: unless the process is its own parent, all open files are closed and all memory belonging to the process is freed

SeeAlso: AH=00h,AH=26h,AH=4Bh,AH=4Dh



## AH = 4Dh - GET RETURN CODE (ERRORLEVEL)

Return:

- AH = termination type (00=normal, 01h control-C abort, 02h=critical error abort, 03h terminate and stay resident)
- AL = return code

Notes:

- the word in which DOS stores the return code is cleared after being read by this function, so the return code can only be retrieved once
- COMMAND.COM stores the return code of the last external command it executed as ERRORLEVEL

SeeAlso: AH=4Bh,AH=4Ch,AH=8Ah



## AH = 54h - GET VERIFY FLAG

Return: AL = verify flag (00h=off, 01h=on, i.e. all disk writes verified after writing)

SeeAlso: AH=2Eh



## AH = 56h - "RENAME" - RENAME FILE

Entry:

- DS:DX -> ASCIZ filename of existing file (no wildcards, but see below)
- ES:DI -> ASCIZ new filename (no wildcards)
- CL = attribute mask (server call only, see below)

Return:

- CF clear if successful
- CF set on error, AX= error code (02h,03h,05h,11h)

Notes:

- allows move between directories on same logical volume
- this function does not set the archive attribute
- open files should not be renamed
- (DOS 3.0+) allows renaming of directories



## AH = 57h - GET FILE'S LAST-WRITTEN DATE AND TIME

Entry:

- AL = 00h (Get attribute)
- BX = file handle

Return:

- CF clear if successful, CX = file's time DX = file's date
- CF set on error, AX = error code (01h,06h)

SeeAlso: AX=5701h

Bitfields for file time:

Bits	15-11	10-5	4-0
------	-------	------	-----

Description	hours	minutes	seconds
-------------	-------	---------	---------

Bitfields for file date:

Bits	15-9	8-5	4-0
Description	year (1980-)	month	day



## AH = 57h - SET FILE'S LAST-WRITTEN DATE AND TIME

Entry:

- AL =01h (Set attributes)
- BX = file handle
- CX = [new time](#)
- DX = [new date](#)

Return:

- CF clear if successful
- CF set on error AX = error code (01h,06h)

SeeAlso: AX=5700h

*This page is maintained by [Barry Wilks](#).*

Pegado de <<http://spike.scu.edu.au/~barry/interrupts.html>>

# x86 memory segmentation

jueves, 10 de julio de 2014  
20:57

**x86 memory segmentation** refers to the implementation of [memory segmentation](#) in the Intel [x86](#) computer [instruction set architecture](#). Segmentation was introduced on the [Intel 8086](#) in 1978 as a way to allow programs to address more than 64 KB (65,536 [bytes](#)) of memory. The [Intel 80286](#) introduced a second version of segmentation in 1982 that added support for [virtual memory](#) and [memory protection](#). At this point the original model was renamed [real mode](#), and the new version was named [protected mode](#). The [x86-64](#) architecture, introduced in 2003, has largely dropped support for segmentation in 64-bit mode.

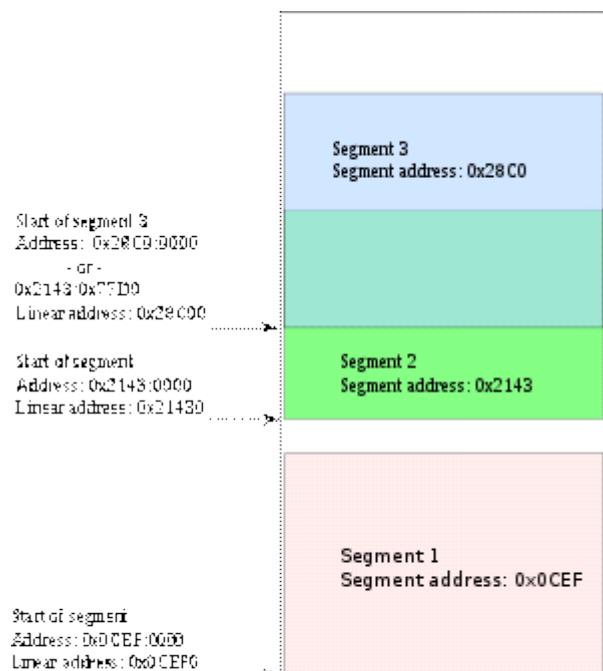
In both real and protected modes the system uses 16-bit *segment registers* to derive the actual memory address. In real mode the registers CS, DS, SS, and ES point to the currently used program [code segment](#) (CS), the current [data segment](#) (DS), the current [stack segment](#) (SS), and one *extra* segment determined by the programmer (ES). The [Intel 80386](#), introduced in 1985, adds two additional segment registers, FS and GS, with no specific uses defined by the hardware. The way in which the segment registers are used differs between the two modes.<sup>[1]</sup>

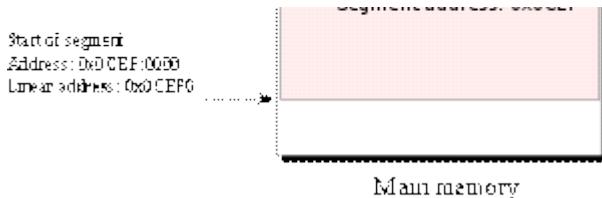
The choice of segment is normally defaulted by the processor according to the function being executed. Instructions are always fetched from the code segment. Any stack push or pop or any data reference referring to the stack uses the stack segment. All other references to data use the data segment. The extra segment is the default destination for string operations (for example MOVS or CMPS). FS and GS have no hardware-assigned uses. The instruction format allows an optional *segment prefix* byte which can be used to override the default segment for selected instructions if desired.<sup>[2]</sup>

## Contents

- [1 Real mode](#)
- [2 Protected mode](#)
  - [2.1 80286 protected mode](#)
  - [2.2 Detailed Segmentation Unit Workflow](#)
  - [2.3 80386 protected mode](#)
- [3 Later developments](#)
- [4 Practices](#)
- [5 Notes and references](#)
- [6 See also](#)
- [7 External links](#)

## Real mode





Three segments in [real mode](#) memory (click on image to enlarge). Note the overlap between segment 2 and segment 3; the bytes in the turquoise area can be used from both segment selectors. In [real mode](#) or [V86 mode](#), a segment is always 65,536 [bytes](#) in size (using 16-bit offsets). The 16-bit segment selector in the segment register is interpreted as the most significant 16 bits of a linear 20-bit address, called a segment address, of which the remaining four least significant bits are all zeros. The segment address is always added to a 16-bit offset in the instruction to yield a *linear address*, which is the same as [physical address](#) in this mode. For instance, the segmented address 06EFh:1234h (here the suffix "h" means [hexadecimal](#)) has a segment selector of 06EFh, representing a segment address of 06EF0h, to which we add the offset, yielding the linear address 06EF0h + 1234h = 08124h ([hexadecimal](#)).

Because of the way the segment address and offset are added, a single linear address can be mapped to up to 4096 distinct segment:offset pairs. For example, the linear address 08124h can have the segmented addresses 06EFh:1234h, 0812h:0004h, 0000h:8124h, etc. This could be confusing to programmers accustomed to unique addressing schemes, but it can also be used to advantage, for example when addressing multiple nested data structures. While real mode segments are always 64 [KiB](#) long, the practical effect is only that no segment can be longer than 64 KiB, rather than that every segment *must* be 64 KiB long. Because there is no protection or privilege limitation in real mode, even if a segment could be defined to be smaller than 64 KiB, it would still be entirely up to the programs to coordinate and keep within the bounds of their segments, as any program can always access any memory (since it can arbitrarily set segment selectors to change segment addresses with absolutely no supervision). Therefore, real mode can just as well be imagined as having a variable length for each segment, in the range 1 to 65536 bytes, that is just not enforced by the CPU.

(Note that the leading zeros of the linear address, segmented addresses, and the segment and offset fields, which are usually neglected, were shown here for clarity.)

The effective 20-bit [address space](#) of real mode limits the [addressable memory](#) to  $2^{20}$  bytes, or 1,048,576 bytes (1 [MiB](#)). This derived directly from the hardware design of the Intel 8086 (and, subsequently, the closely related 8088), which had exactly 20 [address pins](#). (Both were packaged in 40-pin DIP packages; even with only 20 address lines, the address and data buses were multiplexed to fit all the address and data lines within the limited pin count.)

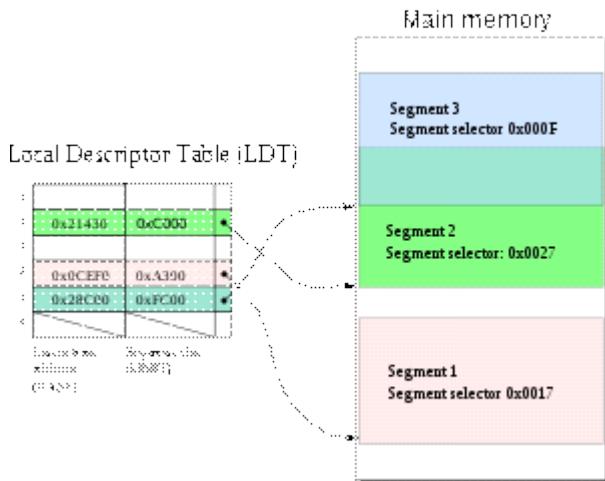
Each segment begins at a multiple of 16 bytes, called a *paragraph*, from the beginning of the linear (flat) address space. That is, at 16 byte intervals. Since all segments are 64 KiB long, this explains how overlap can occur between segments and why any location in the linear memory address space can be accessed with many segment:offset pairs. The actual location of the beginning of a segment in the linear address space can be calculated with segment $\times$ 16. A segment value of 0Ch (12) would give a linear address at C0h (192) in the linear address space. The address offset can then be added to this number. 0Ch:0Fh (12:15) would be C0h+0Fh=CFh (192+15=207), CFh (207) being the linear address. Such address translations are carried out by the segmentation unit of the CPU. The last segment, FFFFh (65535), begins at linear address FFFF0h (1048560), 16 bytes before the end of the 20 bit address space, and thus, can access, with an offset of up to 65,536 bytes, up to 65,520 (65536 - 16) bytes past the end of the 20 bit 8088 address space. On the 8088, these address accesses were wrapped around to the beginning of the address space such that 65535:16 would access address 0 and 65535:1000 would access address 952 of the linear address space. Programmers using this feature led to the [Gate A20](#) compatibility issues in later CPU generations, where the linear address space was expanded past 20 bits.

In 16-bit real mode, enabling applications to make use of multiple memory segments (in order to access more memory than available in any one 64K-segment) is quite complex, but was viewed as a necessary evil for all but the smallest tools (which could do with less memory). The root of the problem is that no appropriate address-arithmetic instructions suitable for flat addressing of the entire memory range are available. [citation needed] Flat addressing is possible by applying multiple

instructions, which however leads to slower programs.

The [memory model](#) concept derives from the setup of the segment registers. for example in the *tiny model* CS=DS=SS, that is the program's code, data, and stack are all contained within a single 64 KB segment. In the *small* memory model DS=SS, so both data and stack reside in the same segment; CS points to a different code segment of up to 64 KB.

## Protected mode



Three segments in [protected mode](#) memory (click on image to enlarge), with the **local descriptor table**.

## 80286 protected mode

The [80286's protected mode](#) extends the processor's address space to  $2^{24}$  bytes (16 mebibytes), but not by adjusting the shift value. Instead, the 16-bit segment registers now contain an index into a table of [segment descriptors](#) containing 24-bit base addresses to which the offset is added. To support old software, the processor starts up in "real mode", a mode in which it uses the segmented addressing model of the 8086. There is a small difference though: the resulting physical address is no longer truncated to 20 bits, so [real mode](#) pointers (but not 8086 pointers) can now refer to addresses between  $100000_h$  and  $10FFEF_h$ . This roughly 64-kilobyte region of memory was known as the [High Memory Area](#) (HMA), and later versions of [MS-DOS](#) could use it to increase the available "conventional" memory (i.e. within the first [MiB](#)). With the addition of the HMA, the total address space is approximately 1.06 MiB. Though the 80286 does not truncate real-mode addresses to 20 bits, a system containing an 80286 can do so with hardware external to the processor, by gating off the 21st address line, the [A20 line](#). The IBM PC AT provided the hardware to do this (for full backward compatibility with software for the original [IBM PC](#) and [PC/XT](#) models), and so all subsequent "[AT-class](#)" PC clones did as well. The first megabyte of memory cannot be accessed. (also true of 32-bit protected mode).

286 protected mode was seldom used as it would have excluded the large body of users with 8086/88 machines. Moreover, it still necessitated dividing memory into 64k segments like was done in real mode. This limitation can be worked around on 32-bit CPUs which permit the use of memory pointers greater than 64k in size, however as the base addresses remain 24-bit, the maximum segment size that can be created is 16MB (although paging can be used to allocate more memory, no individual segment may exceed 16MB). This method was commonly used on Windows 3.x applications to produce a flat memory space, although as the OS itself was still 16-bit, API calls could not be made with 32-bit instructions. Thus, it was still necessary to place all code that performs API calls in 64k segments.

Once 286 protected mode is invoked, it cannot be exited except by performing a hardware reset. Windows 3.x worked around this problem by triggering an intentional segment fault that would cause the CPU to drop back into real mode.

## Detailed Segmentation Unit Workflow

A logical address consists of a 16-bit segment selector (supplying 13+1 address bits) and a 16-bit offset. The segment selector must be located in one of the segment registers. That selector consists of a 2-bit Requested [Privilege Level](#) (RPL), a 1-bit Table Indicator (TI), and a 13-bit index.

When attempting address translation of a given logical address, the processor reads the 64-bit [segment descriptor](#) structure from either the [Global Descriptor Table](#) when TI=0 or the [Local Descriptor Table](#) when TI=1. It then performs the privilege check:

$$\max(\text{CPL}, \text{RPL}) \leq \text{DPL}$$

where CPL is the current privilege level (found in the lower 2 bits of the CS register), RPL is the requested privilege level from the segment selector, and DPL is the descriptor privilege level of the segment (found in the descriptor). All privilege levels are integers in the range 0–3, where the lowest number corresponds to the highest privilege.

If the inequality is false, the processor generates a [general protection \(GP\) fault](#). Otherwise, address translation continues. The processor then takes the 32-bit or 16-bit offset and compares it against the segment limit specified in the segment descriptor. If it is larger, a GP fault is generated.

Otherwise, the processor adds the 24-bit segment base, specified in descriptor, to the offset, creating a linear physical address.

The privilege check is done only when the segment register is loaded, because [segment descriptors](#) are cached in hidden parts of the segment registers. [[citation needed](#)]

## 80386 protected mode

In the [Intel 80386](#) and later, protected mode retains the segmentation mechanism of 80286 protected mode, but a [paging](#) unit has been added as a second layer of address translation between the segmentation unit and the physical bus. Also, importantly, address offsets are 32-bit (instead of 16-bit), and the segment base in each segment descriptor is also 32-bit (instead of 24-bit). The general operation of the segmentation unit is otherwise unchanged. The paging unit may be enabled or disabled; if disabled, operation is the same as on the 80286. If the paging unit is enabled, addresses in a segment are now virtual addresses, rather than physical addresses as they were on the 80286. That is, the segment starting address, the offset, and the final 32-bit address the segmentation unit derived by adding the two are all virtual (or logical) addresses when the paging unit is enabled. When the segmentation unit generates and validates these 32-bit virtual addresses from a program's logical (46-bit<sup>[3]</sup>) addresses, the enabled paging unit finally translates these virtual addresses into physical addresses. The physical addresses are 32-bit on the [386](#), but can be larger on newer processors which support [Physical Address Extension](#).

The 80386 also introduced two new general-purpose data segment registers, FS and GS, to the original set of four segment registers (CS, DS, ES, and SS).

Unlike 286 protected mode, 386 CPUs can be put back into real mode merely by clearing a bit in the CR0 control register.

## Later developments

The [x86-64](#) architecture does not use segmentation in long mode (64-bit mode). Four of the segment registers: CS, SS, DS, and ES are forced to 0, and the limit to  $2^{64}$ . The segment registers FS and GS can still have a nonzero base address. This allows operating systems to use these segments for special purposes.

For instance, [Microsoft Windows](#) on x86-64 uses the GS segment to point to the [Thread Environment Block](#), a small data structure for each [thread](#), which contains information about exception handling, thread-local variables, and other per-thread state. Similarly, the [Linux kernel](#) uses the GS segment to store per-CPU data.

On x64, the CPU powers on into real mode and is indistinguishable from a 32-bit Pentium IV. 64-bit instructions cannot be used unless long mode is set, which first requires entering 32-bit protected mode. When long mode is operating, 16-bit instructions and virtual x86 mode are disabled and protected mode disappears.

## Practices

Logical addresses can be explicitly specified in [x86 assembly language](#), e.g. (AT&T syntax):

`movl $42, %fs:(%eax)` ; Equivalent to  $M[\text{fs:eax}] <- 42$  in [RTL](#)

or in [Intel syntax](#):

`mov dword [fs:eax], 42`

However, segment registers are usually used implicitly.

- All CPU instructions are implicitly fetched from the [code segment](#) specified by the segment selector held in the CS register.
- Most memory references come from the [data segment](#) specified by the segment selector held

in the DS register. These may also come from the extra segment specified by the segment selector held in the ES register, if a segment-override prefix precedes the instruction that makes the memory reference. Most, but not all, instructions that use DS by default will accept an ES override prefix.

- Processor [stack](#) references, either implicitly (e.g. **push** and **pop** instructions) or explicitly ([memory accesses using the \(E\)SP or \(E\)BP registers](#)) use the *stack segment* specified by the segment selector held in the SS register.
- [String instructions](#) (e.g. **stos**, **movs**), along with data segment, also use the *extra segment* specified by the segment selector held in the ES register.

Segmentation cannot be turned off on x86-32 processors (this is true for 64-bit mode as well, but beyond the scope of discussion), so many 32-bit operating systems simulate a [flat memory model](#) by setting all segments' bases to 0 in order to make segmentation neutral to programs. For instance, the [Linux kernel](#) sets up only 4 general purpose segments:

Name	Description	Base	Limit	DPL
_KERNEL_CS	Kernel code segment	0	4 GiB	0
_KERNEL_DS	Kernel data segment	0	4 GiB	0
_USER_CS	User code segment	0	4 GiB	3
_USER_DS	User data segment	0	4 GiB	3

Since the base is set to 0 in all cases and the limit 4 GiB, the segmentation unit does not affect the addresses the program issues before they arrive at the [paging](#) unit. (This, of course, refers to 80386 and later processors, as the earlier x86 processors do not have a paging unit.)

Current Linux also uses GS to point to [thread-local storage](#).

Segments can be defined to be either code, data, or system segments. Additional permission bits are present to make segments read only, read/write, execute, etc.

Note that, in protected mode, code may always modify all segment registers *except CS* (the [code segment](#) selector). This is because the current privilege level (CPL) of the processor is stored in the lower 2 bits of the CS register. The only way to raise the processor privilege level (and reload CS) is through the **Icall** (far call) and **int (interrupt)** instructions. Similarly, the only way to lower the privilege level (and reload CS) is through **Iret** (far return) and **iret** (interrupt return) instructions. In real mode, code may also modify the CS register by making a far jump (or using an undocumented POP CS instruction on the 8086 or 8088)<sup>[4]</sup>. Of course, in real mode, there are no privilege levels; all programs have absolute unchecked access to all of memory and all CPU instructions.

For more information about segmentation, see the [IA-32](#) manuals freely available on the [AMD](#) or [Intel](#) websites.

## Notes and references

1. "Intel 64 and IA-32 Architectures Software Developer's Manual", Volume 3, "System Programming Guide", published in 2011, Page "Vol. 3A 3-11", the book is written: "*Every segment register has a "visible" part and a "hidden" part. (The hidden part is sometimes referred to as a "descriptor cache" or a "shadow register.") When a segment selector is loaded into the visible part of a segment register, the processor also loads the hidden part of the segment register with the base address, segment limit, and access control information from the segment descriptor pointed to by the segment selector. The information cached in the segment register (visible and hidden) allows the processor to translate addresses without taking extra bus cycles to read the base address and limit from the segment descriptor.*"
2. Intel Corporation (2004). [IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture](#).
3. The 46 bits are 14 bits from a 16-bit segment register (the other two bits are the privilege level) plus a 32-bit offset.
4. POP CS must be used with extreme care and has limited usefulness, because it immediately changes the effective address that will be computed from the instruction pointer to fetch the next instruction. Generally, a far jump is much more useful. The existence of POP CS is probably an accident, as it follows a pattern of PUSH and POP instruction opcodes for the four segment registers on the 8086 and 8088.

## See also

- [Intel Memory Model](#)
- [THE multiprogramming system](#)

## External links

- [Home of the IA-32 Intel Architecture Software Developer's Manual](#)
- [The Segment:Offset Addressing Scheme](#)

Pegado de <[http://en.wikipedia.org/wiki/X86\\_memory\\_segmentation](http://en.wikipedia.org/wiki/X86_memory_segmentation)>

# Memory Manipulation

jueves, 10 de julio de 2014  
16:55

When storing a value into a certain offset of the memory, in real mode, we put the address we want to store the information in, between brackets, like this:

**mov [di], BEEFh**

This will store the hex value 0xBEEF, in **DS:DI**.

If we want to use offsets, starting from the given offset, we can just add the value to the given offset, like this:

**mov al, 5**  
**mov [di+al], BEEFh**

This stores the value 0xBEEF into **DS:DI+AL**, which means, the information is stored 5 bytes after DI (because we set AL to 5 in the example).

The same way we store information, we can do the opposite and load any value from any offset.

**mov ax, [si]**

This will load into AX, whatever is in **DS:SI**. You can also apply bite offsets the same way as before.

Addictionaly, we can use the instructions **lodsb** and **lodsw**, to sequentially load bytes, or words, into AL, or AX, respectively, from **[DS:SI]**.

We can also manipulate a memory offset from another segment different from the one in DS. This is done by using **far pointers** (**FAR PTR[SEGMENT:OFFSET]**). Something like this:

**mov FAR PTR[BX:DI], ax**

This code stores the value of AX, into BX:DI.

## B.98 LODSB, LODSW, LODSD: Load from String

LODSB	; AC	[8086]
LODSW	; o16 AD	[8086]
LODSD	; o32 AD	[386]

LODSB loads a byte from [DS:SI] or [DS:ESI] into AL. It then increments or decrements (depending on the direction flag: increments if the flag is clear, decrements if it is set) SI or ESI.

The register used is SI if the address size is 16 bits, and ESI if it is 32 bits. If you need to use an address size not equal to the current BITS setting, you can use an explicit a16 or a32 prefix.

The segment register used to load from [SI] or [ESI] can be overridden by using a segment register name as a prefix (for example, es lodsb).

LODSW and LODSD work in the same way, but they load a word or a doubleword instead of a byte, and increment or decrement the addressing registers by 2 or 4 instead of 1.

Pegado de <<https://courses.engr.illinois.edu/ece390/archive/spr2002/books/labmanual/inst-ref-lodsb.html>>

# Port I/O (For x86 only)

jueves, 10 de julio de 2014  
21:44

## I/O address range Device

00 – 1F	First <a href="#">DMA controller</a> 8237 A-5
20 – 3F	First <a href="#">Programmable Interrupt Controller</a> , <a href="#">8259A</a> , Master
40 – 5F	<a href="#">Programmable Interval Timer</a> (System Timer), <a href="#">8254</a>
60 – 6F	<a href="#">Keyboard</a> , <a href="#">8042</a>
70 – 7F	<a href="#">Real Time Clock</a> , NMI mask
80 – 9F	<a href="#">DMA</a> Page Register, 74LS612
87	<a href="#">DMA</a> Channel 0
83	<a href="#">DMA</a> Channel 1
81	<a href="#">DMA</a> Channel 2
82	<a href="#">DMA</a> Channel 3
8B	<a href="#">DMA</a> Channel 5
89	<a href="#">DMA</a> Channel 6
8A	<a href="#">DMA</a> Channel 7
8F	Refresh
A0 – BF	Second <a href="#">Programmable Interrupt Controller</a> , <a href="#">8259A</a> , Slave
C0 – DF	Second <a href="#">DMA controller</a> 8237 A-5
F0	Clear 80287 Busy
F1	Reset 80287
F8 – FF	<a href="#">Math coprocessor</a> , <a href="#">80287</a>
F0 – F5	<a href="#">PCjr</a> Disk Controller
F8 – FF	Reserved for future microprocessor extensions
100 – 10F	POS Programmable Option Select (PS/2)
110 – 1EF	System I/O channel
140 – 15F	Secondary <a href="#">SCSI host adapter</a>
170 – 177	Secondary <a href="#">Parallel ATA</a> Disk Controller
1F0 – 1F7	Primary <a href="#">Parallel ATA</a> Hard Disk Controller
200 – 20F	<a href="#">Game port</a>
210 – 217	Expansion Unit
220 – 233	<a href="#">Sound Blaster</a> and most other <a href="#">sound cards</a>
278 – 27F	<a href="#">Parallel port</a> 3
280 – 29F	<a href="#">LCD</a> on <a href="#">Wyse</a> 2108 PC SMC Elite default factory setting
2B0 – 2DF	Alternate <a href="#">Enhanced Graphics Adapter</a> (EGA) display control
2E8 – 2EF	<a href="#">Serial port</a> 4
2E1	GPIB/ <a href="#">IEEE-488</a> Adapter 0
2E2 – 2E3	Data acquisition
2F8 – 2FF	<a href="#">Serial port</a> 2

2F8 – 2FF	<a href="#">Serial port</a> 2
300 – 31F	Prototype Card
300 – 31F	Novell <a href="#">NE1000</a> compatible Ethernet network interfaces
300 – 31F	<a href="#">AMD Am7990 Ethernet</a> network interface, IRQ=5.
320 – 323	<a href="#">ST-506</a> and compatible <a href="#">hard disk drive</a> interface
330 – 331	<a href="#">MPU-401 MIDI Processing Unit</a> on most sound cards
340 – 35F	Primary <a href="#">SCSI host adapter</a>
370 – 377	Secondary <a href="#">floppy disk drive</a> controller
378 – 37F	<a href="#">Parallel port</a> 2
380 – 38C	Secondary Binary Synchronous Data Link Control ( <a href="#">SDLC</a> ) adapter
388 – 389	<a href="#">AdLib</a> Music Synthesizer Card
3A0 – 3A9	Primary Binary Synchronous Data Link Control ( <a href="#">SDLC</a> ) adapter
3B0 – 3BB	<a href="#">Monochrome Display Adapter</a> (MDA) display control
3BC – 3BF	<a href="#">Parallel port</a> 1 on MDA card
3C0 – 3CF	<a href="#">Enhanced Graphics Adapter</a> (EGA) display control
3D0 – 3DF	<a href="#">Color Graphics Adapter</a> (CGA)
3E8 – 3EF	<a href="#">Serial port</a> 3
3F0 – 3F7	Primary <a href="#">floppy disk drive</a> controller. Primary IDE controller (slave drive) (3F6–3F7h)
3F8 – 3FF	<a href="#">Serial port</a> 1
CF8 – CFC	<a href="#">PCI configuration space</a>

Pegado de <[http://en.wikipedia.org/wiki/Input/Output\\_Base\\_Address](http://en.wikipedia.org/wiki/Input/Output_Base_Address)>

## I/O Instructions

**in** src, dest [GAS Syntax](#)

**in** dest(AX), src(DX) [Intel syntax](#)

The **IN** instruction almost always has the operands AX and DX (or EAX and EDX) associated with it. DX (src) frequently holds the port address to read, and AX (dest) receives the data from the port. In Protected Mode operating systems, the IN instruction is frequently locked, and normal users can't use it in their programs.

**out** src, dest [GAS Syntax](#)

**out** dest(AX), src(DX) [Intel syntax](#)

The **OUT** instruction is very similar to the IN instruction. OUT outputs data from a given register (src) to a given output port (dest). In protected mode, the OUT instruction is frequently locked so normal users can't use it.

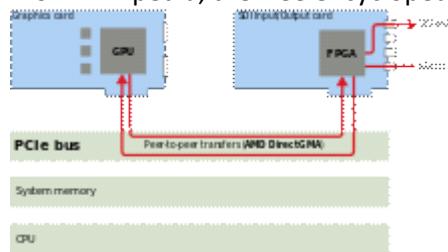
Pegado de <[http://en.wikibooks.org/wiki/X86\\_Assembly/Other\\_Instructions#I.2FO\\_Instructions](http://en.wikibooks.org/wiki/X86_Assembly/Other_Instructions#I.2FO_Instructions)>

# DMA (Direct Memory Access)

jueves, 10 de julio de 2014

22:56

From Wikipedia, the free encyclopedia



AMD DirectGMA is a form of DMA. It enables low-latency peer-to-peer data transfers between devices on the [PCIe bus](#) and [AMD FirePro](#)-branded products. [SDI](#) devices supporting DirectGMA can write directly into the graphics memory of the GPU and vice versa the GPU can directly access the memory of a peer device.

**Direct memory access (DMA)** is a feature of [computers](#) that allows certain hardware subsystems within the computer to access system [memory](#) independently of the [central processing unit](#) (CPU).

Without DMA, when the CPU is using [programmed input/output](#), it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, does other operations while the transfer is in progress, and receives an [interrupt](#) from the DMA controller when the operation is done.

This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including [disk drive](#) controllers, [graphics cards](#), [network cards](#) and [sound cards](#). DMA is also used for intra-chip data transfer in [multi-core processors](#).

Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or [scatter-gather](#) operations, from the CPU to a dedicated DMA engine. An implementation example is the [I/O Acceleration Technology](#).

## Contents

- [1 Principle](#)
- [2 Modes of operation](#)
  - [2.1 Burst mode](#)
  - [2.2 Cycle stealing mode](#)
  - [2.3 Transparent mode](#)
- [3 Cache coherency](#)
- [4 Examples](#)
  - [4.1 ISA](#)
  - [4.2 PCI](#)
  - [4.3 I/OAT](#)
  - [4.4 AHB](#)
  - [4.5 Cell](#)
- [5 See also](#)

- [6 Notes](#)
- [7 References](#)

## Principle

A DMA controller can generate [memory addresses](#) and initiate memory read or write cycles. It contains several [processor registers](#) that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers. The control registers specify the I/O port to use, the direction of the transfer (reading from the I/O device or writing to the I/O device), the transfer unit (byte at a time or word at a time), and the number of bytes to transfer in one burst.<sup>[1]</sup>

To carry out an input, output or memory-to-memory operation, the host processor initializes the DMA controller with a count of the number of [words](#) to transfer, and the memory address to use. The CPU then sends commands to a peripheral device to initiate transfer of data. The DMA controller then provides addresses and read/write control lines to the system memory. Each time a byte of data is ready to be transferred between the peripheral device and memory, the DMA controller increments its internal address register until the full block of data is transferred.

DMA transfers can either occur one byte at a time or all at once in burst mode. If they occur a byte at a time, this can allow the CPU to access memory on alternate bus cycles – this is called [cycle stealing](#) since the DMA controller and CPU contend for memory access. In *burst mode DMA*, the CPU can be put on hold while the DMA transfer occurs and a full block of possibly hundreds or thousands of bytes can be moved.<sup>[2]</sup> When memory cycles are much faster than processor cycles, an *interleaved* DMA cycle is possible, where the DMA controller uses memory while the CPU cannot.

In a [bus mastering](#) system, both the CPU and peripherals can be granted control of the memory bus. Where a peripheral can become bus master, it can directly write to system memory without involvement of the CPU, providing memory address and control signals as required. Some measure must be provided to put the processor into a hold condition so that bus contention does not occur.

## Modes of operation

### Burst mode

An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode".it is also used to stop unnecessary data.

### Cycle stealing mode

The *cycle stealing mode* is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using **BR (Bus Request) and BG (Bus Grant) signals**, which are the two signals controlling the interface between the CPU and the DMA controller.

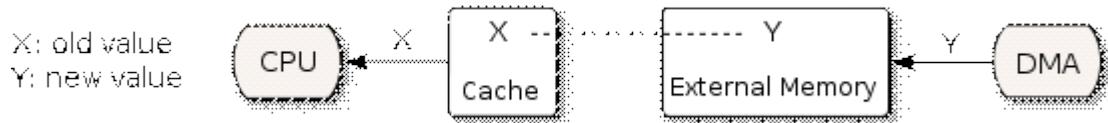
However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers.

The CPU processes an instruction, then the DMA controller transfers one data value, and so on. On the one hand, the data block is not transferred as quickly in cycle stealing mode as in burst mode, but on the other hand the CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

## Transparent mode

The *transparent mode* takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. The DMA controller only transfers data when the CPU is performing operations that do not use the system buses. It is the primary advantage of the transparent mode that the CPU never stops executing its programs and the DMA transfer is free in terms of time. The disadvantage of the transparent mode is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.

## Cache coherency



Cache incoherence due to DMA

DMA can lead to [cache coherency](#) problems. Imagine a CPU equipped with a cache and an external memory that can be accessed directly by devices using DMA. When the CPU accesses location X in the memory, the current value will be stored in the cache. Subsequent operations on X will update the cached copy of X, but not the external memory version of X, assuming a [write-back cache](#). If the cache is not flushed to the memory before the next time a device tries to access X, the device will receive a stale value of X.

Similarly, if the cached copy of X is not invalidated when a device writes a new value to the memory, then the CPU will operate on a stale value of X.

This issue can be addressed in one of two ways in system design: Cache-coherent systems implement a method in hardware whereby external writes are signaled to the cache controller which then performs a [cache invalidation](#) for DMA writes or cache flush for DMA reads.

Non-coherent systems leave this to software, where the OS must then ensure that the cache lines are flushed before an outgoing DMA transfer is started and invalidated before a memory range affected by an incoming DMA transfer is accessed. The OS must make sure that the memory range is not accessed by any running threads in the meantime. The latter approach introduces some overhead to the DMA operation, as most hardware requires a loop to invalidate each cache line individually.

Hybrids also exist, where the secondary L2 cache is coherent while the L1 cache (typically on-CPU) is managed by software.

## Examples

### ISA

In the original [IBM PC](#), there was only one [Intel 8237](#) DMA controller capable of providing four DMA channels (numbered 0–3), as part of the so-called [Industry Standard Architecture](#), or ISA. These DMA channels performed 8-bit transfers and could only address the first megabyte of RAM. With the [IBM PC/AT](#), a second 8237 DMA controller was added (channels 5–7; channel 4 is dedicated as a cascade channel for the first 8237 controller), and the page register was rewired to address the full 16 MB memory address space of the 80286 CPU. This second controller performed 16-bit transfers.

Due to their lagging performance (2.5 Mbit/s<sup>[3]</sup>), these devices have been largely obsolete since the advent of the [80386](#) processor in 1985 and its capacity for 32-bit transfers. They are still supported to the extent they are required to support built-in legacy PC hardware on modern machines. The only pieces of legacy hardware that use ISA DMA and are still fairly common are [Super I/O](#) devices on motherboards that often integrate the a built-in [floppy disk](#) controller, an [IrDA](#) infrared controller when FIR (fast infrared) mode is selected, and a [IEEE 1284](#) parallel port controller when ECP mode is

selected.

Each DMA channel has a 16-bit address register and a 16-bit count register associated with it. To initiate a data transfer the device driver sets up the DMA channel's address and count registers together with the direction of the data transfer, read or write. It then instructs the DMA hardware to begin the transfer. When the transfer is complete, the device [interrupts](#) the CPU.

Scatter-gather or [vectored I/O](#) DMA allows the transfer of data to and from multiple memory areas in a single DMA transaction. It is equivalent to the chaining together of multiple simple DMA requests. The motivation is to off-load multiple [input/output](#) interrupt and data copy tasks from the CPU.

DRQ stands for *Data request*; DACK for *Data acknowledge*. These symbols, seen on hardware [schematics](#) of computer systems with DMA functionality, represent electronic signaling lines between the CPU and DMA controller. Each DMA channel has one Request and one Acknowledge line. A device that uses DMA must be configured to use both lines of the assigned DMA channel. Standard ISA DMA assignments:

1. [DRAM](#) Refresh (obsolete),
2. User hardware,
3. [Floppy disk](#) controller,
4. [Hard disk](#) (obsoleted by [PIO](#) modes, and replaced by [UDMA](#) modes), Parallel Port (ECP capable port),
5. Cascade from XT DMA controller,
6. Hard Disk ([PS/2](#) only), user hardware for all others,
7. User hardware.
8. User hardware.

## PCI

A [PCI](#) architecture has no central DMA controller, unlike ISA. Instead, any PCI component can request control of the bus ("become the [bus master](#)") and request to read from and write to system memory. More precisely, a PCI component requests bus ownership from the PCI bus controller (usually the [southbridge](#) in a modern PC design), which will [arbitrate](#) if several devices request bus ownership simultaneously, since there can only be one bus master at one time. When the component is granted ownership, it will issue normal read and write commands on the PCI bus, which will be claimed by the bus controller and will be forwarded to the memory controller using a scheme which is specific to every chipset.

As an example, on a modern [AMD Socket AM2](#)-based PC, the southbridge will forward the transactions to the [northbridge](#) (which is integrated on the CPU die) using [HyperTransport](#), which will in turn convert them to [DDR2](#) operations and send them out on the DDR2 memory bus. As can be seen, there are quite a number of steps involved in a PCI DMA transfer; however, that poses little problem, since the PCI device or PCI bus itself are an order of magnitude slower than the rest of the components (see [list of device bandwidths](#)).

A modern x86 CPU may use more than 4 GB of memory, utilizing [PAE](#), a 36-bit addressing mode, or the native 64-bit mode of [x86-64](#) CPUs. In such a case, a device using DMA with a 32-bit address bus is unable to address memory above the 4 GB line. The new [Double Address Cycle](#) (DAC) mechanism, if implemented on both the PCI bus and the device itself,<sup>[4]</sup> enables 64-bit DMA addressing. Otherwise, the operating system would need to work around the problem by either using costly [double buffers](#) (DOS/Windows nomenclature) also known as [bounce buffers](#) ([FreeBSD](#)/[Linux](#)), or it could use an [IOMMU](#) to provide address translation services if one is present.

## I/OAT

Main article: [I/O Acceleration Technology](#)

As an example of DMA engine incorporated in a general-purpose CPU, newer Intel [Xeon](#) chipsets

include a DMA engine technology called [I/O Acceleration Technology](#) (I/OAT), meant to improve network performance on high-throughput network interfaces, in particular [gigabit Ethernet](#) and faster.<sup>[5]</sup> However, various benchmarks with this approach by Intel's [Linux kernel](#) developer Andrew Grover indicate no more than 10% improvement in CPU utilization with receiving workloads, and no improvement when transmitting data.<sup>[6]</sup>

## AHB

Main article: [Advanced Microcontroller Bus Architecture](#)

In [systems-on-a-chip](#) and [embedded systems](#), typical system bus infrastructure is a complex on-chip bus such as [AMBA High-performance Bus](#). AMBA defines two kinds of AHB components: master and slave.

A slave interface is similar to programmed I/O through which the software (running on embedded CPU, e.g. [ARM](#)) can write/read I/O registers or (less commonly) local memory blocks inside the device. A master interface can be used by the device to perform DMA transactions to/from system memory without heavily loading the CPU.

Therefore high bandwidth devices such as network controllers that need to transfer huge amounts of data to/from system memory will have two interface adapters to the AHB: a master and a slave interface. This is because on-chip buses like AHB do not support [tri-stating](#) the bus or alternating the direction of any line on the bus. Like PCI, no central DMA controller is required since the DMA is bus-mastering, but an [arbiter](#) is required in case of multiple masters present on the system.

Internally, a multichannel DMA engine is usually present in the device to perform multiple concurrent [scatter-gather](#) operations as programmed by the software.

## Cell

Main article: [Cell \(microprocessor\)](#)

As an example usage of DMA in a [multiprocessor-system-on-chip](#), IBM/Sony/Toshiba's [Cell processor](#) incorporates a DMA engine for each of its 9 processing elements including one Power processor element (PPE) and eight synergistic processor elements (SPEs).

Since the SPE's load/store instructions can read/write only its own local memory, an SPE entirely depends on DMAs to transfer data to and from the main memory and local memories of other SPEs. Thus the DMA acts as a primary means of data transfer among cores inside this [CPU](#) (in contrast to cache-coherent CMP architectures such as Intel's cancelled [general-purpose GPU, Larrabee](#)).

DMA in Cell is fully [cache coherent](#) (note however local stores of SPEs operated upon by DMA do not act as globally coherent cache in the [standard sense](#)). In both read ("get") and write ("put"), a DMA command can transfer either a single block area of size up to 16 KB, or a list of 2 to 2048 such blocks. The DMA command is issued by specifying a pair of a local address and a remote address: for example when a SPE program issues a put DMA command, it specifies an address of its own local memory as the source and a virtual memory address (pointing to either the main memory or the local memory of another SPE) as the target, together with a block size. According to a recent experiment, an effective peak performance of DMA in Cell (3 GHz, under uniform traffic) reaches 200 GB per second.<sup>[7]</sup>

## See also

- [AT Attachment](#)
- [Blitter](#)
- [Channel I/O](#)
- [DMA attack](#)
- [Polling \(computer science\)](#)
- [Remote direct memory access](#)

## Notes

1. Osborne, Adam (1980). *An Introduction to Microcomputers: Volume 1: Basic Concepts* (2nd ed.). Osborne McGraw Hill. pp. 5–64 through 5–93. [ISBN 0931988349](#).
2. Horowitz, Paul; Hill, Winfield (1989). *The Art of Electronics* (Second ed.). Cambridge University Press. p. 702. [ISBN 0521370957](#).
3. Intel publication 03040, Aug 1989
4. "[Physical Address Extension — PAE Memory and Windows](#)". Microsoft Windows Hardware Development Central. 2005. Retrieved 2008-04-07.
5. Corbet, Jonathan (2005-12-06). "[Memory copies in hardware](#)". [LWN.net](#) (December 8, 2005). Retrieved 2006-11-12.
6. Grover, Andrew (2006-06-01). "[I/OAT on LinuxNet wiki](#)". Overview of I/OAT on Linux, with links to several benchmarks. Retrieved 2006-12-12.
7. Kistler, Michael (May 2006). "[Cell Multiprocessor Communication Network](#)". Extensive benchmarks of DMA performance in Cell Broadband Engine.

## References

- [DMA Fundamentals on Various PC Platforms](#), from A. F. Harvey and Data Acquisition Division Staff NATIONAL INSTRUMENTS
- [mmap\(\) and DMA](#), from *Linux Device Drivers, 2nd Edition*, Alessandro Rubini & [Jonathan Corbet](#)
- [Memory Mapping and DMA](#), from *Linux Device Drivers, 3rd Edition*, [Jonathan Corbet](#), Alessandro Rubini, [Greg Kroah-Hartman](#)
- [DMA and Interrupt Handling](#)
- [DMA Modes & Bus Mastering](#)

<a href="#">[hide]</a> <ul style="list-style-type: none"><li>• <a href="#">v</a></li><li>• <a href="#">t</a></li><li>• <a href="#">e</a></li></ul>	
<a href="#">Computer bus — technical and <i>de facto</i> standards (wired)</a>	
<b>General</b>	<ul style="list-style-type: none"><li>• <a href="#">System bus</a></li><li>• <a href="#">Front-side bus</a></li><li>• <a href="#">Back-side bus</a></li><li>• <a href="#">Daisy chain</a></li><li>• <a href="#">Control bus</a></li><li>• <a href="#">Address bus</a></li><li>• <a href="#">Bus contention</a></li><li>• <a href="#">Network on a chip</a></li><li>• <a href="#">Plug and play</a></li><li>• <a href="#">List of bus bandwidths</a></li></ul>
<b>Standards</b>	<ul style="list-style-type: none"><li>• <a href="#">S-100 bus</a></li><li>• <a href="#">Unibus</a></li><li>• <a href="#">VAXBI</a></li><li>• <a href="#">MBus</a></li><li>• <a href="#">STD Bus</a></li><li>• <a href="#">SMBus</a></li><li>• <a href="#">Q-Bus</a></li><li>• <a href="#">Europe Card Bus</a></li><li>• <a href="#">ISA</a></li><li>• <a href="#">STEbus</a></li><li>• <a href="#">Zorro II</a></li><li>• <a href="#">Zorro III</a></li><li>• <a href="#">CAMAC</a></li><li>• <a href="#">FASTBUS</a></li></ul>

	<ul style="list-style-type: none"> <li>• <a href="#">LPC</a></li> <li>• <a href="#">HP Precision Bus</a></li> <li>• <a href="#">EISA</a></li> <li>• <a href="#">VME</a></li> <li>• <a href="#">VXI</a></li> <li>• <a href="#">VXS</a></li> <li>• <a href="#">NuBus</a></li> <li>• <a href="#">TURBOchannel</a></li> <li>• <a href="#">MCA</a></li> <li>• <a href="#">SBus</a></li> <li>• <a href="#">VLB</a></li> <li>• <a href="#">PCI</a></li> <li>• <a href="#">PXI</a></li> <li>• <a href="#">HP GSC bus</a></li> <li>• <a href="#">CoreConnect</a></li> <li>• <a href="#">InfiniBand</a></li> <li>• <a href="#">UPA</a></li> <li>• <a href="#">PCI Extended (PCI-X)</a></li> <li>• <a href="#">AGP</a></li> <li>• <a href="#">PCI Express (PCIe)</a></li> <li>• <a href="#">Direct Media Interface (DMI)</a></li> <li>• <a href="#">RapidIO</a></li> <li>• <a href="#">Intel QuickPath Interconnect</a></li> <li>• <a href="#">HyperTransport</a></li> </ul>
<b>Storage</b>	<ul style="list-style-type: none"> <li>• <a href="#">ST-506</a></li> <li>• <a href="#">ESDI</a></li> <li>• <a href="#">SMD</a></li> <li>• <a href="#">Parallel ATA (PATA)</a></li> <li>• <a href="#">SSA</a></li> <li>• <a href="#">DSSI</a></li> <li>• <a href="#">HIPPI</a></li> <li>• <a href="#">USB MSC</a></li> <li>• <a href="#">IEEE 1394 interface (FireWire)</a></li> <li>• <a href="#">Serial ATA (SATA)</a></li> <li>• <a href="#">eSATA</a></li> <li>• <a href="#">eSATAp</a></li> <li>• <a href="#">mSATA</a></li> <li>• <a href="#">SCSI</a></li> <li>• <a href="#">Parallel SCSI</a></li> <li>• <a href="#">SAS</a></li> <li>• <a href="#">SATA Express</a></li> <li>• <a href="#">M.2</a></li> </ul>
<b>Peripheral</b>	<ul style="list-style-type: none"> <li>• <a href="#">Apple Desktop Bus</a></li> <li>• <a href="#">HIL</a></li> <li>• <a href="#">MIDI</a></li> <li>• <a href="#">Multibus</a></li> <li>• <a href="#">RS-232</a></li> <li>• <a href="#">RS-422</a></li> <li>• <a href="#">RS-423</a></li> <li>• <a href="#">RS-485</a></li> </ul>

	<ul style="list-style-type: none"> <li>• <a href="#">DMX512-A</a></li> <li>• <a href="#">IEEE-488 (GPIB)</a></li> <li>• <a href="#">IEEE-1284 (parallel port)</a></li> <li>• <a href="#">UNI/O</a></li> <li>• <a href="#">ACCESS.bus</a></li> <li>• <a href="#">1-Wire</a></li> <li>• <a href="#">D<sup>2</sup>B</a></li> <li>• <a href="#">I<sup>2</sup>C</a></li> <li>• <a href="#">SPI</a></li> <li>• <a href="#">Parallel SCSI</a></li> <li>• <a href="#">Profibus</a></li> <li>• <a href="#">USB</a></li> <li>• <a href="#">IEEE 1394 (FireWire)</a></li> <li>• <a href="#">Camera Link</a></li> <li>• <a href="#">External PCIe</a></li> <li>• <a href="#">Thunderbolt</a></li> </ul>
<b>Audio</b>	<ul style="list-style-type: none"> <li>• <a href="#">ADAT Lightpipe</a></li> <li>• <a href="#">AES3</a></li> <li>• <a href="#">Intel HD Audio</a></li> <li>• <a href="#">I<sup>2</sup>S</a></li> <li>• <a href="#">MADI</a></li> <li>• <a href="#">McASP</a></li> <li>• <a href="#">S/PDIF</a></li> <li>• <a href="#">TOSLINK</a></li> </ul>
<b>Portable</b>	<ul style="list-style-type: none"> <li>• <a href="#">PC Card</a></li> <li>• <a href="#">ExpressCard</a></li> </ul>
<b>Embedded</b>	<ul style="list-style-type: none"> <li>• <a href="#">Multidrop bus</a></li> <li>• <a href="#">AMBA</a></li> <li>• <a href="#">Wishbone</a></li> </ul>
<p><i>Note: interfaces are listed in speed ascending order (roughly), the interface at the end of each section should be the fastest</i></p>  <a href="#">Category</a>	

Pegado de <[http://en.wikipedia.org/wiki/Direct\\_memory\\_access](http://en.wikipedia.org/wiki/Direct_memory_access)>

## Vectored I/O

From Wikipedia, the free encyclopedia

This article is about the I/O method. For the vector addressing type, see [Gather-scatter \(vector addressing\)](#).

In [computing](#), **vectored I/O**, also known as **scatter/gather I/O**, is a method of input and output by which a single procedure-call sequentially writes data from multiple buffers to a single [data stream](#) or reads data from a data stream to multiple buffers. The buffers are given in a vector of buffers. *Scatter/gather* refers to the process of gathering data from, or scattering data into, the given set of buffers. Vectored I/O can operate synchronously or asynchronously. The main reasons for using vectored I/O are efficiency and convenience.

There are several usages for vectored I/O:

- [Atomicity](#): If the particular vectored I/O implementation supports atomicity, a process can write into or read from a set of buffers to or from a file without risk that another [thread](#) or [process](#) might perform I/O on the same file between the first process' reads or writes, thereby corrupting the file or compromising the integrity of the input

- Concatenating output: An application that wants to write non-sequentially placed data in memory can do so in one vectored I/O operation. For example, writing a fixed-size header and its associated payload data that are placed non-sequentially in memory can be done by a single vectored I/O operation without first concatenating the header and the payload to another buffer
- Efficiency: One vectored I/O read or write can replace many ordinary reads or writes, and thus save on the overhead involved in [syscalls](#)
- Splitting input: When reading data that is in a format that defines a fixed-size header, one can use a vector of buffers in which the first buffer is the size of that header; and the second buffer will contain the data associated with the header

Standards bodies document the applicable functions [readv<sup>\[1\]</sup>](#) and [writev<sup>\[2\]</sup>](#) in [POSIX](#) 1003.1-2001 and the [Single UNIX Specification](#) version 2. The [Windows API](#) has analogous functions [ReadFileScatter](#) and [WriteFileGather](#); however, unlike the POSIX functions, they require the alignment of each buffer on a [memory page](#).<sup>[3]</sup> [Windows Sockets](#) provide separate [WSASend](#) and [WSARecv](#) functions without this requirement.

While working directly with a vector of buffers can be significantly harder than working with a single buffer, there are often higher-level APIs<sup>[4]</sup> for working efficiently that can mitigate the problem.

## References

1. [readv](#) in the Single Unix Specification
2. [writev](#) in the Single Unix Specification
3. [ReadFileScatter](#) in MSDN Library
4. [Vstr](#) the Vectored String API

Pegado de <[http://en.wikipedia.org/wiki/Vectored\\_I/O](http://en.wikipedia.org/wiki/Vectored_I/O)>

# 74LS612 (DMA Page Register)

jueves, 10 de julio de 2014  
23:07

## What is it?

The 612 is an MMU, Memory Management Unit, which enables you (in this specific case) to expand your system with 8 extra address lines. For example, a Commodore 64 or VIC-20 could be expanded up to 16 MB!

Notify the fact that I use the word "system" and NOT processor! The processor itself won't be aware at all of those extra address lines, it is the Operating System that must handle all the extra address space. Seen from the outside, the processor will only see parts of the complete configuration. This is realised by feeding a part of the address lines to some very fast RAM. This RAM outputs the new address lines. In case of the 612 it is 4\*12 bits ie. 16 registers of 12 bits each.

## How does a MMU work?

If you own a C64 then this is easier to understand. It has 64 KB of RAM, 4 KB of I/O and at least 20 KB of ROM. But the 6510 has a 16 bit address bus and therefore is only capable of addressing 64 KB. How do these numbers match? Many programmers know that they can read the contents of the RAM "under" the BASIC- or KERNAL-ROM but cannot access that ROM at the same time: it is either RAM or ROM but not both!

A more experienced user knows that the I/O-registers of the 6510 and the PLA give the C64 the ability to handle more memory than the processor itself is capable of. This ability to switch between various parts of memory or I/O is called 'Bank switching'.

The 74LS612 does about the same but is much more flexible: it does not only give you the capability of bank switching but also allows you to decide where which part of the memory or I/O should show up.

## How do these 8 extra address lines fit in the system?

As said before, the 612 has 16 registers of 12 bits each. The normal procedure is to address these registers using the four most significant address lines; in case of a 6510 the address lines A12..A15. The 12 bits represent the 12 address lines the 612 can output. Four of those lines (can) reflect the state of the four originally attached address lines. The other eight are the extra eight address lines. Every bit can be programmed as the user wishes, meaning that the four bits reflecting the original address lines can completely differ from the original values. Reconfiguring the original address lines gives you the ability to re-arrange the original configuration of the memory map. For example: one cannot use the RAM under the Kernal-ROM to test a new one; the reset-procedure resets the I/O register of the 6510, thereby disabling the RAM under the ROM. Skipping this particular routine is a possibility but then this Kernal would not be really the real thing. With a MMU you could swap the range \$8000/\$9FFF with \$E000/\$FFFF. As the Kernal has no knowledge of the existence of the MMU, it won't reset it and the 6510 will run the new Kernal in \$8000/\$9FFF as if it was in \$E000/\$FFFF. But be aware: this will only work with jumping to the new Kernal under software (also called: soft reset); a hard reset can reset 612 as well and then the 6510 will see the original Kernal as well.

But a remark is on its place here: reconfiguring the original memory map does not mean that you are now able to place the CHAR-ROM at \$2000 and the I/O on \$4000. With the MMU you move address ranges, not specific hardware. In the above example of swapping the Kernal it also means there is no RAM under the new Kernal. So you cannot test a new Kernal with a program which normally uses the RAM under ROM. In this case the program will overwrite the Kernal for sure with all devastating effects!

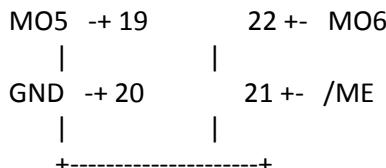
## An idea how to use the MMU.

The extra eight lines can be used to add extra memory (RAM, ROM) or I/O in any way you want. One idea what I have in mind is to connect a complete 1541 to my CBM 3032 by means of a small interface which replaces the 6502 of the 1541. This interface is addressable from \$01000 to \$01FFFF. Now it should be possible to run a program that first loads the new Kernal into memory range \$4000/\$7FFF and then reconfigures the 3032 in such a way that the 6502 sees the range \$0000/\$1FFF of the 1541 as its own \$0000/\$1FFF range and its own range \$4000/\$007FFF as \$C000/\$FFFF. The last action is a jump to the new Kernal and the CBM should now behave like a 1541. This method allows you to test a new Kernal for the 1541 in a very comfortable way. (The originator of this idea is Andre Fachat, Leipzig, Germany)

The moment you re-arrange the internal configuration, you have to take care of one important thing: make always sure you have access to the MMU! The reason is very simple: once you have placed the MMU out of the memory map, you won't be able to change its configuration anymore and you are stuck with the momentary one for "the rest of your life".

## Pin outs

RS2	-+ 1	40	+ -	+5V
MA3	-+ 2	39	+ -	MA2
RS3	-+ 3	38	+ -	RS1
/CS	-+ 4	37	+ -	MA1
/STROBE	-+ 5	36	+ -	RS0
R/W	-+ 6	35	+ -	MA0
D0	-+ 7	34	+ -	D11
D1	-+ 8	33	+ -	D10
D2	-+ 9	32	+ -	D9
D3	-+ 10	31	+ -	D8
	74LS612			
D4	-+ 11	30	+ -	D7
D5	-+ 12	29	+ -	D6
MM	-+ 13	28	+ -	NC / C (see text)
MO0	-+ 14	27	+ -	MO11
MO1	-+ 15	26	+ -	MO10
MO2	-+ 16	25	+ -	MO9
MO3	-+ 17	24	+ -	MO8
MO4	-+ 18	23	+ -	MO7



#### Pin functions:

D0 through D11 = Connection to the data bus when programming the registers  
 RS0 through RS3 = Connection to the address bus when programming the registers,  
 normally A0 through A3

R/W = Read, active (H) and Write, active (L)

STROBE = input to enter data into chosen register, active (L)

CS = chip select, active (L)

MA0 through MA3 = input, to be connected to the address bus of the processor.  
 These inputs choose an internal 12-bits register.

MO0 through MO11 = The generated address lines

MM = Map Mode input. When (H), MO8/MO11 reflect MA0/MA03 and  
 MO0/MO7 are (L). When (L), MO0/MO11 reflect the 12 bits of  
 the chosen register.

ME = When (H), MO0/MO11 are tristate else active.

NC = Not connected.

## 74LS610, 74LS611 and 74LS613.

The 610 is like a 612 but has an extra 12 bits latch at its outputs. So with the 610 you can freeze the output as long as you want. This is archived with an extra input line at pin 28, C (= Clock). A (H) on this input will the 610 cause to behave like a 612, a (L) freezes the configuration.

The 611 and 613 are the Open Collector versions of the 610 and 612.

## The 12 bit registers of the 74LS612 - part 1

The intention is to use the MMU in combination with a 65xx based Commodore. But as you know the 65xx's are all 8-bitters and the 612 is a 12-bitter. This means we either have to disregard 4 bits or use a trick to be able to use all 12 bits. In the first case we are still able to address up to 1 MB of memory or I/O. If you still want have those extra 4 bits then a simple 74LS573, 8-bit latch, can do the trick. Of course you'll need a decoder for this 573. But as you need one for the MMU as well, it won't be that much difficult to combine them into one decoder. (for more details, see part 2)

## Connecting the MMU to a 6502.

Connecting a MMU to a 6502 can be split up in two parts anyway:

- the memory-part
- the I/O-part

### Memory-part

If you only want to reconfigure your system within its own 64 KB, then you only have to "cut" the address lines A12 through A15 and to place the MMU in the created gap. Practically this probably means you replace the processor by a small circuit board with onboard the original processor, the MMU and some logic to perform I/O-operations with the MMU.

If you want to use the extra address lines as well, you have to take care of the fact that the original hardware does not know of the addresses above \$00FFFF. So for any decoder on the original system reading from/writing to \$01C000, \$02C000 or any address above \$00FFFF is equal to reading from/writing to the address \$0C000. There are two solutions for this problem:

- Include the original decoder
- Fool the original system

### Include the original decoder

The C64 uses an 82S100 PLA as decoder. This IC has a CS-line of its own but it is tied to Ground. Tying

this line to a decoder, which decodes the address lines above A15, solves the problem.

### Fooling the original system

Sometimes the above solution is impossible due to the way the decoding is done. In case of the CBM 8032 the range from \$8000 to \$FFFF is decoded by a 74LS154. But the range below \$8000 is decoded by gates. This simply means that you have a lot of soldering to do.

With "fool the system" we force the original system to read a safe address during the time the processor is handling an address above \$0FFFF. AFAIK a safe address for all Commodores is \$FFFF. But reading implies outputting data, which we have to shield from the processor. This can be realized by using a buffer like the 74LS245. The fake address is realized by placing buffers in the address lines between the original system and the processor/MMU. Using \$FFFF has the advantage that we only need some pull-up resistors to generate this address the moment the bus is disabled.

### I/O-part

Another decision to be made is where to place the address decoding of the MMU in the memory map: behind or before the MMU itself. If you place it before the MMU it always will be accessible but the disadvantage is that it can get in the way when there is a need to remap certain areas to the area the MMU occupies. If you place it behind the MMU and you make a mistake with programming the MMU, it is possible you cannot access the MMU anymore meaning you are stuck with the momentary configuration unless you reset your system.

## Enabling the MMU.

Negating the "Map-mode"-input enables the function of the MMU. So we need at least a programmable 1-bit register. But it is a MUST that its output always is (H) after a reset. A [74LS74](#) could do the trick. Connect its Preset-input with the reset line, the data-input with a data line and clock-input with an active (L) chip select-line. If you intend to use all 12 bits, this 74 can be placed parallel to the used 573.

## The 12 bits registers of the 74LS612 - part 2

We already know we have to provide four extra data bits for the MMU. We also need a bit for enabling the MMU-mode. As said a 74LS573 could do the trick but only using a 573 means we cannot check in one or another way what we actually have written to the MMU. We could add a 74LS541 buffer to solve a part of the problem but in that case I propose to use a CIA, PIA, VIA, PIO or other equivalent type of IC. Which type you use partly depends on the processor used in the system. One advantage of using such IC is that you have much more functionality then when using the 573/541 combination. Another advantage is that one I/O-line can be used for steering the "Map-Mode"-input of the MMU. I will use the 6522 VIA as example from now on.

Now we are able to read the data written towards the MMU but we are still not able to read the contents of all bits of the 16 registers of the MMU. Luckily I found a solution where we only need an extra 573 and inverter. The in- and outputs of four bits of the 573 are connected to the four I/O-lines responsible for the four extra bits of the MMU. The data on these lines is latched every time the /CS-input of the MMU is enabled (the 04 inverts this CS-signal to the right level for the clock-input of the 573). An I/O line of the 6522 takes care of dis/enabling the output of the 573.

- The moment we want the 65xx to write towards the MMU, the output of the 573 must be disabled. The I/O lines of the 6522 must have been programmed as outputs and have been filled with valid data.
- The moment we want the 65xx to read data from the MMU, the I/O lines must be programmed as input and the output of the 573 must be disabled.
- The moment we want to output the latched data of the 573 to be read by the 6522, its I/O lines must be programmed as inputs.

As after a RESET of the system all the I/O lines have been switched to input mode, a resistor should take care of pulling the MMU-input (H), disabling the map-mode in this way. Another resistor must take care of disabling the output of the 573.

## **Extra functionality's of the 6522**

I also have another reason to use the 6522. I intend to use the 65816 instead of the 6510 (C64) or 8502 (C128). But these processors have an onboard I/O-port which the 65816 lacks. The 6522 can supply this port.

## **Availability**

[Nick Coplin](#) had some ideas for using the 74LS612 only to find out that they are hard to get. The ones I have I got by recycling old PC motherboards but I cannot expect everyone having a load of old motherboards for scrapping laying around. So I decided to make a [MMU](#) of my own.

Having questions or comment? You want more Info?

You can email me [here](#).

Pegado de <<http://www.baltissen.org/newhtm/74ls612.htm>>

# Programmable Interval Timer (PIT)

jueves, 10 de julio de 2014

23:34

## Programmable interval timer

From Wikipedia, the free encyclopedia

(Redirected from [Programmable Interval Timer](#))

In [computing](#) and in [embedded systems](#), a **programmable interval timer (PIT)** is a counter that generates an output signal when it reaches a programmed count. The output signal is often used to trigger an [interrupt](#).

### Contents

- [1 Common features](#)
- [2 IBM PC compatible](#)
- [3 See also](#)
- [4 References](#)
- [5 External links](#)

### Common features

PITs may be one-shot or periodic. One-shot timers will signal only once and then stop counting. Periodic timers signal every time they reach a specific value and then restart, thus producing a signal at periodic intervals. Periodic timers are typically used to invoke activities that must be performed at regular intervals.

Counters are usually programmed with fixed intervals that determine how long the counter will count before it signals. The interval determines how long the counter will count before it will output a signal

### IBM PC compatible

The [Intel 8253](#) PIT was the original timing device used on [IBM PC compatibles](#). It used a 1.193182 MHz clock signal (one third of the [color burst](#) frequency used by [NTSC](#), one twelfth of the system clock [crystal oscillator](#)<sup>[1]</sup>) and contains three timers. Timer 0 is used by [Microsoft Windows](#) (uniprocessor) and [Linux](#) as a system timer, timer 1 was historically used for [dynamic random access memory](#) refreshes and timer 2 for the [PC speaker](#).<sup>[2]</sup>

The [LAPIC](#) in newer Intel systems offers a higher-resolution (one microsecond) timer.<sup>[3]</sup> This is used in preference to the PIT timer in for [Linux kernels](#) starting with 2.6.18.<sup>[4]</sup>

### See also

- [Intel 8253](#)
- [NE555](#)
- [High Precision Event Timer](#)

### References

1. "[Bran's Kernel Development Tutorial: The Programmable Interval Timer](#)". Osdever.net. Retrieved 2013-10-30.
2. "[Programmable Interval Timer - OSDev Wiki](#)". Wiki.osdev.org. 2012-06-26. Retrieved 2013-10-30.
3. Uwe Walter, Vincent Oberle [u-second precision timer support for the Linux kernel](#)
4. [Determining and changing the rate of timer interrupts a guest operating system requests \(1005802\)](#)

### External links

- <http://www.luxford.com/high-performance-windows-timers>
- <http://stackoverflow.com/questions/10567214/what-are-linux-local-timer-interrupts>



This computer hardware article is a [stub](#). You can help Wikipedia by [expanding it](#).

Categories:

- [IBM PC compatibles](#)
- [Digital electronics](#)
- [Computer hardware stubs](#)

Pegado de <[http://en.wikipedia.org/wiki/Programmable\\_Interval\\_Timer](http://en.wikipedia.org/wiki/Programmable_Interval_Timer)>

# IRQ (Interrupt Request)

jueves, 10 de julio de 2014

23:41

## Interrupt request

From Wikipedia, the free encyclopedia



This article includes a [list of references](#), related reading or [external links](#), but its sources remain unclear because it lacks [inline citations](#). Please [improve](#) this article by introducing more precise citations. (November 2011)

In a [computer](#), an **interrupt request** (or **IRQ**) is a hardware signal sent to the processor that temporarily stops a running program and allows a special program, an [interrupt handler](#), to run instead. Interrupts are used to handle such events as data receipt from a modem or network, or a key press or mouse movement. The [interrupt request level](#) (**IRQL**) is the [priority](#) of an interrupt request.

Interrupt lines are often identified by an index with the format of *IRQ* followed by a number. For example, on the [Intel 8259](#) family of [PICs](#) there are eight interrupt inputs commonly referred to as *IRQ0* through *IRQ7*. In [x86](#) based [computer systems](#) that use two of these [PICs](#), the combined set of lines are referred to as *IRQ0* through *IRQ15*. Technically these lines are named *IRQ* 0 through *IRQ* 7, and the lines on the [ISA](#) bus to which they were historically attached are named *IRQ0* through *IRQ15*. Newer [x86](#) systems integrate an [Advanced Programmable Interrupt Controller](#) (APIC) that conforms to the [Intel APIC Architecture](#). These APICs support a programming interface for up to 255 physical hardware IRQ lines per APIC, with a typical system implementing support for only around 24 total hardware lines.

## Contents

- [1 Overview](#)
- [2 x86 IRQs](#)
  - [2.1 Master PIC](#)
  - [2.2 Slave PIC](#)
- [3 Conflicts](#)
- [4 See also](#)
- [5 References](#)
- [6 External links](#)

## Overview

When working with personal computer hardware, installing and removing devices, the system relies on interrupt requests. There are default settings that are configured in the system [BIOS](#) and recognized by the operating system. These default settings can be altered by advanced users.

Modern [plug and play](#) technology has not only reduced the need for concern for these settings, but has virtually eliminated manual configuration.

## x86 IRQs

See [Intel 8259](#) for a common list and discussion of hardware IRQ lines in [x86](#) systems.

Typically, on systems using the [Intel 8259](#), 16 IRQs are used. IRQs 0 to 7 are managed by one Intel 8259 PIC, and IRQs 8 to 15 by a second Intel 8259 PIC. The first PIC, the master, is the only one that directly signals the CPU. The second PIC, the slave, instead signals to the master on its IRQ 2 line, and the master passes the signal on to the CPU. There are therefore only 15 interrupt request lines available for hardware.

On newer systems using the [Intel APIC Architecture](#), typically there are 24 IRQs available, and the extra 8 IRQs are used to route PCI interrupts, avoiding conflict between dynamically configured PCI interrupts and statically configured ISA interrupts. On early APIC systems with only 16 IRQs or with only [Intel 8259](#) interrupt controllers, PCI interrupt lines were routed to the 16 IRQs using a PIR integrated into the southbridge.

The easiest way of viewing this information on [Microsoft Windows](#) is to use [Device Manager](#) or [System Information](#) (msinfo32.exe). On [Linux](#), IRQ mappings can be viewed by executing cat

/proc/interrupts or procinfo programs.

## Master PIC

- IRQ 0 — system timer (cannot be changed);
- IRQ 1 — [keyboard controller](#) (cannot be changed);
- IRQ 2 — cascaded signals from IRQs 8–15;
  - *any devices configured to use IRQ 2 will actually be using IRQ 9*
- IRQ 3 — [serial port controller](#) for [serial port 2](#) (shared with serial port 4, if present);
- IRQ 4 — serial port controller for serial port 1 (shared with serial port 3, if present);
- IRQ 5 — [parallel port](#) 2 and 3 or [sound card](#);
- IRQ 6 — [floppy disk controller](#);
- IRQ 7 — parallel port 1. It is used for printers or for any parallel port if a printer is not present.

## Slave PIC

- IRQ 8 — [real-time clock](#)
- IRQ 9 — [Advanced Configuration and Power Interface](#) system control interrupt on Intel chipsets.<sup>[1]</sup> Other chipset manufacturers might use another interrupt for this purpose.
  - *any devices configured to use IRQ 2 will actually be using IRQ 9*
- IRQ 10 — The Interrupt is left open for the use of peripherals. *open interrupt / available* or SCSI or [NIC](#);
- IRQ 11 — The Interrupt is left open for the use of peripherals. *open interrupt / available* or SCSI or NIC;
- IRQ 12 — [mouse on PS/2 connector](#);
- IRQ 13 — CPU [co-processor](#) or integrated [floating point unit](#) or [inter-processor interrupt](#) (use depends on OS);
- IRQ 14 — primary [ATA](#) channel;
- IRQ 15 — secondary [ATA](#) channel;  
ATA interface usually serves [hard disks](#) and [CD drives](#).

## Conflicts

In IBM-compatible personal computers, an *IRQ conflict* is a once common hardware error, received when two devices were trying to use the same **interrupt request** (or IRQ) to signal an interrupt to the [Programmable Interrupt Controller](#) (PIC).

The PIC expects interrupt requests from only one device per line, thus more than one device sending IRQ signals along the same line will generally cause an IRQ conflict that can freeze a [computer](#).

In some rare conditions, two devices can share the same IRQ as long as they are not used simultaneously.

If one adds a [modem expansion card](#) to a system, and assign it to IRQ4, which is traditionally assigned to the [serial port](#) 1, it will likely cause an IRQ conflict.

## See also



### [Computer science portal](#)

- [Advanced Programmable Interrupt Controller](#) (APIC)
- [Programmable Interrupt Controller](#) (PIC)
- [Intel 8259](#)
- [Interrupt handler](#)
- [Input/Output Base Address](#)
- [Plug and play](#)
- [Polling](#)
- [Interrupt](#)

## References

1. Oshins, Jake (December 30, 2001). "[RE: ACPI Machines and IRQ 9 \[was: Communicating with the NT developers\]](#)". Retrieved April 17, 2014.
- Gilliwe, Frank van. *The Undocumented PC, Second Edition*, Addison-Wesley Developers Press, 1997. [ISBN 0-201-47950-8](#)
- Shanley, Tom. *ISA System Architecture, Third Edition*, Addison-Wesley Publishing Company, 1995. [ISBN 0-201-40996-8](#)

- Solari, Edward. *PCI & PCI-X Hardware and Software Architecture & Design, Sixth Edition*, Research Tech Inc., 2004. [ISBN 0-9760865-0-6](#)
- [IRQ interrupt request](#)
- [Expansion Bus Configuration](#)

## External links

More information on the Intel 8259 PIC and its IRQ lines can be found in the [IA-32 Intel Architecture Software Developer's Manual, Volume 3A: System Programming Guide, Part 1](#), freely available on the [Intel](#) website.

- [Ralf Brown's Interrupt List](#)

Pegado de <[http://en.wikipedia.org/wiki/Interrupt\\_request](http://en.wikipedia.org/wiki/Interrupt_request)>

## IRQL (Windows)

From Wikipedia, the free encyclopedia

An **Interrupt Request Level (IRQL)** is the priority given to any [interrupt request](#) generated by computer hardware. If for instance, a [hard disk drive](#) generates an interrupt request signal, and another device, such as a [USB](#) printer, generates an interrupt request signal, the two interrupts can not be serviced simultaneously. The IRQL is used to determine which one is serviced first. In this example, a hard disk interrupt requires a higher priority than a printer interrupt.

The criteria used to prioritize interrupt request levels are determined by the [operating system](#), which is programmed to give higher priority to more important devices. For example, if a [gamepad](#) and a hard disk generated simultaneous interrupt requests, the operating system would assign a higher priority to the hard disk, because the hard disk contains information necessary for the operating system or its applications to function. If hard disk requests are delayed, many [applications](#) or even the operating system will be slowed down.

Pegado de <[http://en.wikipedia.org/wiki/IRQL\\_\(Windows\)](http://en.wikipedia.org/wiki/IRQL_(Windows))>

## Interrupt handler

From Wikipedia, the free encyclopedia

In computer system programming, an **interrupt handler**, also known as an **interrupt service routine** or **ISR**, is a [callback](#) function in [microcontroller firmware](#), an [operating system](#) or a [device driver](#), whose execution is triggered by the reception of an [interrupt](#). Interrupt handlers have a multitude of functions, which vary based on the reason the interrupt was generated and the speed at which the interrupt handler completes its task.

An interrupt handler is a low-level counterpart of [event handlers](#). These handlers are initiated by either hardware interrupts or interrupt instructions in software, and are used for servicing hardware devices and transitions between protected modes of operation such as system calls.

## Overview

In several operating systems - [Linux](#), [Unix](#), [Mac OS X](#), [Microsoft Windows](#), and some other operating systems in the past, interrupt handlers are divided into two parts: the **First-Level Interrupt Handler (FLIH)** and the **Second-Level Interrupt Handlers (SLIH)**. FLIHs are also known as *hard interrupt handlers* or *fast interrupt handlers*, and SLIHs are also known as *slow/soft interrupt handlers*, [Deferred Procedure Call](#).

A FLIH implements at minimum platform-specific interrupt handling similar to *interrupt routines*. In response to an interrupt, there is a [context switch](#), and the code for the interrupt is loaded and executed. The job of a FLIH is to quickly service the interrupt, or to record platform-specific critical information which is only available at the time of the interrupt, and [schedule](#) the execution of a SLIH for further long-lived interrupt handling.

FLIHs cause [jitter](#) in process execution. FLIHs also mask interrupts. Reducing the jitter is most important for [real-time operating systems](#), since they must maintain a guarantee that execution of specific code will complete within an agreed amount of time. To reduce jitter and to reduce the potential for losing data from masked interrupts, programmers attempt to minimize the execution time of a FLIH, moving as much as possible to the SLIH. With the speed of modern computers, FLIHs may implement all device and platform-dependent handling, and use a SLIH for further platform-independent long-lived handling.

FLIHs which service hardware typically mask their associated interrupt (or keep it masked as the case may be) until they complete their execution. An (unusual) FLIH which unmasks its associated interrupt before it completes is called a [reentrant interrupt handler](#). Reentrant interrupt handlers might cause a [stack overflow](#) from multiple [preemptions](#) by the same [interrupt vector](#), and so they are usually avoided. In a [priority interrupt](#) system, the FLIH also (briefly) masks other interrupts of equal or lesser priority.

A SLIH completes long interrupt processing tasks similarly to a process. SLIHs either have a dedicated [kernel](#) thread for each handler, or are executed by a pool of kernel worker threads. These threads sit on a [run queue](#) in the operating system until processor time is available for them to perform processing for the interrupt. SLIHs may have a long-lived execution time, and thus are typically scheduled similarly to threads and processes.

In Linux, FLIHs are called *upper half*, and SLIHs are called *lower half* or *bottom half*. This is different from naming used in other Unix-like systems, where both are a part of *bottom half*.

Pegado de <[http://en.wikipedia.org/wiki/Interrupt\\_handler](http://en.wikipedia.org/wiki/Interrupt_handler)>

## 32 bit example

viernes, 01 de agosto de 2014

1:20

The nasm source code is [printf2.asm](#)

The result of the assembly is [printf2.lst](#)

The equivalent "C" program is [printf2.c](#)

Running the program produces output [printf2.out](#)

This program demonstrates basic use of "C" library function printf.

The equivalent "C" code is shown as comments in the assembly language.

; printf2.asm use "C" printf on char, string, int, double

;

; Assemble: nasm -f elf -l printf2.lst printf2.asm

; Link: gcc -o printf2 printf2.o

; Run: printf2

; Output:

;Hello world: a string of length 7 1234567 6789ABCD 5.327000e-30 -1.234568E+302

;

; A similar "C" program

;#include

; int main()

;

; char char1='a'; /\* sample character \*/

; char str1[]="string"; /\* sample string \*/

; int int1=1234567; /\* sample integer \*/

; int hex1=0x6789ABCD; /\* sample hexadecimal \*/

; float flt1=5.327e-30; /\* sample float \*/

; double flt2=-123.4e300; /\* sample double \*/

;

; printf("Hello world: %c %s %d %X %e %E \n", /\* format string for printf \*/

; char1, str1, int1, hex1, flt1, flt2);

; return 0;

;

extern printf ; the C function to be called

SECTION .data ; Data section

msg: db "Hello world: %c %s of length %d %d %X %e %E",10,0

; format string for printf

char1: db 'a' ; a character

str1: db "string",0 ; a C string, "string" needs 0

len: equ \$-str1 ; len has value, not an address

inta1: dd 1234567 ; integer 1234567

hex1: dd 0x6789ABCD ; hex constant

flt1: dd 5.327e-30 ; 32-bit floating point

flt2: dq -123.456789e300 ; 64-bit floating point

SECTION .bss

fltmp: resq 1 ; 64-bit temporary for printing flt1

SECTION .text ; Code section.

global main ; "C" main program

main: ; label, start of main program

fld dword [flt1] ; need to convert 32-bit to 64-bit

fstp qword [fltmp] ; floating load makes 80-bit,

; store as 64-bit

```
; push last argument first
push  dword [flt2+4]      ; 64 bit floating point (bottom)
push  dword [flt2]        ; 64 bit floating point (top)
push  dword [fltmp+4]     ; 64 bit floating point (bottom)
push  dword [fltmp]       ; 64 bit floating point (top)
push  dword [hex1]        ; hex constant
push  dword [inta1]       ; integer data pass by value
push  dword len          ; constant pass by value
push  dword str1         ; "string" pass by reference
push  dword [char1]       ; 'a'
push  dword msg          ; address of format string
call   printf             ; Call C function
add    esp, 40            ; pop stack 10*4 bytes
mov    eax, 0              ; exit code, 0=normal
ret                   ; main returns to operating system
```

Pegado de <<http://www.csee.umbc.edu/portal/help/nasm/sample.shtml#printf2>>

## EXE format

viernes, 01 de agosto de 2014  
1:49

### EXE Format

Note: all multi-byte values are stored LSB first. One block is 512 bytes, one paragraph is 16 bytes.

See also [the entry in Ralf Brown's Interrupt List](#)

Offset (hex)	Meaning
-----------------	---------

- |       |   |
|-------|---|
| 00-01 | 0x4d, 0x5a. This is the "magic number" of an EXE file. The first byte of the file is 0x4d and the second is 0x5a.   |
| 02-03 | The number of bytes in the last block of the program that are actually used. If this value is zero, that means the entire last block is used (i.e. the effective value is 512).   |
| 04-05 | Number of blocks in the file that are part of the EXE file. If [02-03] is non-zero, only that much of the last block is used.   |
| 06-07 | Number of relocation entries stored after the header. May be zero.  |
| 08-09 | Number of paragraphs in the header. The program's data begins just after the header, and this field can be used to calculate the appropriate file offset. The header includes the relocation entries. Note that some OSs and/or programs may fail if the header is not a multiple of 512 bytes. |
| 0A-0B | Number of paragraphs of additional memory that the program will need. This is the equivalent of the BSS size in a Unix program. The program can't be loaded if there isn't at least this much memory available to it.   |
| 0C-0D | Maximum number of paragraphs of additional memory. Normally, the OS reserves <i>all</i> the remaining conventional memory for your program, but you can limit it with this field.   |
| 0E-0F | Relative value of the stack segment. This value is added to the segment the program was loaded at, and the result is used to initialize the SS register.  |
| 10-11 | Initial value of the SP register.   |
| 12-13 | Word checksum. If set properly, the 16-bit sum of all words in the file should be zero. Usually, this isn't filled in.  |
| 14-15 | Initial value of the IP register.   |
| 16-17 | Initial value of the CS register, relative to the segment the program was loaded at.  |
| 18-19 | Offset of the first relocation item in the file.  |
| 1A-1B | Overlay number. Normally zero, meaning that it's the main program.  |

Here is a structure that can be used to represent the EXE header and relocation entries, assuming a 16-bit LSB machine:

```
struct EXE {  
    unsigned short signature; /* == 0x5a4D */  
    unsigned short bytes_in_last_block;  
    unsigned short blocks_in_file;  
    unsigned short num_relocs;  
    unsigned short header_paragraphs;  
    unsigned short min_extra_paragraphs;  
    unsigned short max_extra_paragraphs;  
    unsigned short ss;  
    unsigned short sp;  
    unsigned short checksum;  
    unsigned short ip;  
    unsigned short cs;  
    unsigned short reloc_table_offset;  
    unsigned short overlay_number;  
};  
struct EXE_RELOC {  
    unsigned short offset;  
    unsigned short segment;  
};
```

The offset of the beginning of the EXE data is computed like this:

```
exe_data_start = exe.header_paragraphs * 16L;
```

The offset of the byte just after the EXE data (in DJGPP, the size of the stub and the start of the COFF image) is computed like this:

```
extra_data_start = exe.blocks_in_file * 512L;  
if (exe.bytes_in_last_block)  
    extra_data_start -= (512 - exe.bytes_in_last_block);
```

Pegado de <<http://www.delorie.com/djgpp/doc/exe/>>

When running a native .exe the following happens (grossly simplified):

- A process object is created.
- The exe file is read into that process's memory. Different sections of the .exe (code, data, etc.) are mapped in separately and given different permissions (code is execute, data is read/write, constants are read-only).
- Relocations occur in the .exe (addresses get patched if the .exe was not loaded at its preferred address.)
- The import table is walked and dependent DLL's are loaded.
- DLL's are mapped in a similar method to .exe's, with relocations occurring and their dependent DLL's being loaded. Imported functions from DLL's are resolved.
- The process starts execution at an initial stub in NTDLL.
- The initial loader stub runs the entry points for each DLL, and then jumps to the entry point of the .exe.

Managed executables contain MSIL (Microsoft Intermediate Language) and may be compiled so they can target any CPU that the CLR supports. I am not that familiar with the inner workings of the CLR loader (what native code initially runs to bootstrap the CLR and start interpreting the MSIL) - perhaps someone else can elaborate on that.

Pegado de <<http://stackoverflow.com/questions/1495638/whats-in-a-exe-file>>

## PE File Structure

The "portable executable file format" (PE) is the format of the binary programs (exe, dll, sys, scr) for MS windows NT, windows 95 and win32s. It can also be used for object files ( bpl, dpl, cpl, ocx, acm, ax).

The format was designed by Microsoft and then in 1993 standardized by the **Tool Interface**

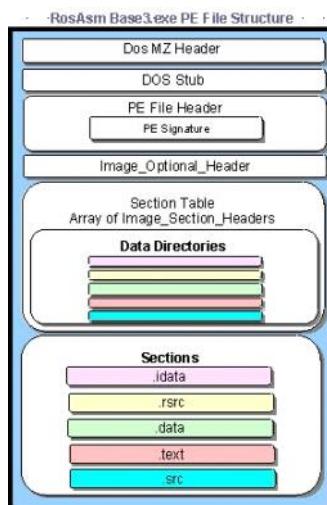
**Standard Committee** (Microsoft, Intel, Borland, Watcom, IBM and others), apparently based on the "common object file format" (COFF), the format used for object files and executables on several UNIX and VMS OSes.

The following 3 paragraphs copied from Mark Pietrek's article on MSDN  
The term "Portable Executable" was chosen because the intent was to have a common file format for all flavors of Windows, on all supported CPUs. To a large extent, this goal has been achieved with the same format used on Windows NT and descendants, Windows 95 and descendants, and Windows CE.

A very handy aspect of PE files is that the Data Structures on disk are the same data structures used in memory. Loading an executable into memory is primarily a matter of mapping certain ranges of a PE file into the address space. Thus, a data structure is identical on disk and in memory. The key point is that if you know how to find something in a PE file, you can almost certainly find the same information after the file is loaded in memory. It's important to note that PE files are not just mapped into memory as a single memory-mapped file. Instead, the Win32 loader looks at the PE file and decides what portions of the file to map in. This mapping is consistent in that higher offsets in the file correspond to higher memory addresses when mapped into memory. The offset of an item in the disk file may differ from its offset once loaded into memory. However, all the information is present to allow you to make the translation from disk offset to memory offset.

A module in memory represents all the code, data, and resources from an executable file that is needed by a process. Other parts of a PE file may be read, but not mapped in (for instance, relocations). Some parts may not be mapped in at all, for example, when debug information is placed at the end of the file. A field in the PE header tells the system how much memory needs to be set aside for mapping the executable into memory. Data that won't be mapped in is placed at the end of the file, past any parts that will be mapped in.

The PE data structures are: **DOS header**, **DOS stub**, **PE File Header**, **Image Optional Header**, **Section Table** - which has a trailing array of **Section Headers** - **Data Directories** - these directories contain pointers to data in the individual sections and, lastly, the individual **Sections** themselves.



Each section header has some flags about what kind of data it contains ("initialized data", "readable data", "writable data" and so on), whether it can be shared etc., and pointers ([RVAs](#)). A PE header component is called the "IMAGE\_DIRECTORY\_HEADER". This header holds information about some PE Sections (Resources, Import, and so on). Each Record being [PointerToSection] [Size].

Some sections also have additional headers (for example .rsrc) called **Section Data Headers**. Some don't, directoryless types of contents are, for example, "executable code" or "initialized data". Essentially, the sections' contents is what you really need to execute a program, and all the header and directory stuff is just there to help you or the win32 loader to find it.

### PE File Layout Rosasm Base3.exe

#### Dos MZ Header

00000000	4D 5A	90 00 03 00 00 00	04 00	00 00 FF FF	00 00 M2	0000000000000000	yy	00
00000010	B8 00	00 00 00 00 00 00	40 00	00 00 00 00 00 00	,	0000000000000000	00	00
00000020	00 00	00 00 00 00 00 00	00 00 00 00 00 00	00 00 00 00 00 00	00	0000000000000000	00	00
00000030	00 00	00 00 00 00 00 00	20 83 0C 00	80 00 00 00	00	0000000000000000	00	00

This makes a PE file an MS-DOS executable. First 2 bytes are always: 4Dh 5Ah = "MZ" is dos exe signature starting at 0h, Last Page Size (2 bytes), Total Pages in File (2 bytes), Relocation Items (2 bytes). The word at offset 8h tells the # of 16 byte paragraphs the DOS header contains > Dos Header Size 4h (16\*4) = 40h bytes in length,

Min Size (2 bytes), Max Size FFFFh @ offset 0Ch, Initial Stack Segment (SP register value at run time 2 bytes), Initial Stack Pointer 2 bytes, 2 byte Checksum for Header @ offset 12h, Initial Instruction Pointer 2 bytes, Initial Code Segment 2 bytes, Relocation Table Offset 40h @ offset 18h 2 bytes, Overlay # default 2 bytes, Betov's CheckSum 4 bytes @ offset 38h (located in the last part of 8 reserved words ) and the PE Header Pointer > 80h 4 bytes @ offset 3Ch.

The above as seen in Quickview. (00h thru 3Fh)

#### Header Information

Signature:	5a4d
Last Page Size:	0090
Total Pages in File:	0003
Relocation Items:	0000
Paragraphs in Header:	0004
Minimum Extra Paragraphs:	0000
Maximum Extra Paragraphs:	fff
Initial Stack Segment:	0000
Initial Stack Pointer:	00b8
Complemented Checksum:	0000
Initial Instruction Pointer:	0000
Initial Code Segment:	0000
Relocation Table Offset:	0040
Overlay Number:	0000
Reserved:	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 8320 000c
Offset to New Header:	00000080
Memory Needed:	2K

In a Win32 system the PE loader just skips the following MS-DOS Stub.

#### MS-DOS executable ("stub")

0E 1F BA 0E 09 CD 21 B8 01 4C CD 21 53 70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
69 6E 64 6F 7A 20 33 32 20 73 70 69 74 20 50 45 inodz 32 spit PE  
66 69 6C 65 20 6D 61 64 65 20 77 69 7A 20 52 6F file made wiz Ro  
73 41 73 6D 20 41 73 73 65 6D 62 6C 65 72 2E 24 sAsm Assembler.  
The DOS stub is actually a valid EXE for PE-files, it is a MS-DOS 2.0 compatible executable that almost always consists of a small number of bytes that output an error message. It can simply display a string like "This program requires Windows" or "Cannot be run in DOS mode" or the "RosAsm message above".

The bytes from 40h to 4Dh above is the actual code: push cs // pop ds // mov dx 0E // mov ah 09 // int 021 // mov ax 4C01 // int 021 to print the message if the program is run in DOS.

#### PE File header (in the COFF-format)

00000080 50 45 00 00 00 4C 01 04 00 00 00 00 00 00 00 00 PE 00 00 00 00 00 00 00 00  
00000090 00 00 00 00 E0 00 0F 01 0B 01 03 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00

First row: The 32-bit-PE signature (first 4 bytes contains the number 4550h > "PE") @ offset 80h , the 2 byte IMAGE\_FILE\_MACHINE @ offset 84h for x86: (14Ch) for Intel 80386 processor or better, (14Dh) for Intel 80486 processor or better,(14Eh) for Intel Pentium processor or better. 2 bytes @offset 86h contains how many sections are in it 04h . Next is a timestamp 'TimeDateStamp' (32 bit value).The members 'PointerToSymbolTable' (32 bit value) and,

Second row:'NumberOfSymbols' 0h @ offset 90h (32 bit value) are used for debugging information. 'SizeOfOptionalHeader' E0h (16 bit) @ offset 94h.This header tells us how the binary should be loaded: The starting address, the amount of stack to reserve, the size of the data segment etc.. 'Characteristics' is 16 bits 10Fh (1\_00001111) @ offset 96h and consists of a collection of flags, most of which are valid for libraries and object files. Optional Header Starts @ offset 98h the 16-bit-word is 'Magic' always contains the value 10Bh.The next 2 bytes 3h @ offset 9Ah are the 'Version' of the linker.'SizeOfCode' is last 4 bytes Size of the executable code.

The above as seen in Quickview. ( 80h thru 9Fh)

#### Image File Header

Signature:	00004550
Machine:	Intel 386
Number of Sections:	0004
Time Date Stamp:	00000000
Symbols Pointer:	00000000
Number of Symbols:	00000000
Size of Optional Header	00e0

#### Image Optional Header

Magic:	010b
Linker Version:	3.00
Size of Code:	00000200

Thusfar in Base3.exe as written by RosAsm, we have encountered the Dos header, Dos stub and the PE 'image file header' - that tells us most importantly, what machine it runs on, how many sections are in it and the size of the 'Image Optional Header', which immediately follows and we are now in....starting at 98h,with magic and is E0h bytes long, adding the two together we get 178h... which is the start of the IMAGE\_DATA\_DIRECTORY array as we will see below after we finish with the 'Image Optional Header' ..

000000A0 00 0E 00  
000000B0 00 30 00 00 00 00 40 00 00 10 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00

First row: The next 2 longwords @ offset A0h (32 bits each) are intended to be the size of the initialized data ('SizeOfInitializedData'), the "data segment" and the size of the uninitialized data ('SizeOfUninitializedData') the so-called "bss segment". This is the size of everything but Code and Virtual data (size of initialized data .data + .rsrc+... + .reloc). Next is a 32-bit-value @ offset A8h that is an RVA (relative virtual address). This RVA is the offset to the code's entry point, program execution starts here ('AppVAEntryPoint').The next 2 32-bit-values are the offsets to the executable code ('AppBaseOfCode') and,

Second row: The initialized data @ offset B0h ('SHAppBaseOfData').The next entry is a 32-bit-value giving the preferred (linear) load address ('ImageBase') of the entire binary, including all headers 400000h = (The image base linker default value). The next 2 32-bit-values @ offset B8h are the alignments of the PE-file's sections in RAM 'Section Alignment' when the image has been loaded and in the 'File Alignment' in the file.

The above as seen in Quickview. ( A0h thru BFh )

Size of Initialized Data:	00000e00
Size of Uninitialized Data:	00000000
Address of Entry Point:	00004000
Base of Code:	00004000
Base of Data:	00003000
Image Base:	00400000
Section Alignment:	00001000
File Alignment:	00000200

000000C0 04 00 00 00 01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
000000D0 00 50 00 00 00 04 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00

First row: The next 4 16-bit-words are the expected operating system version ('MajorOperatingSystemVersion' and 'MinorOperatingSystemVersion', the next 2 16-bit-words are the expected subsystem version and 4 bytes of reserved space.

Second row: 'SizeOfImage' 32 bit value@ offset D0h.It is the sum of all headers' and sections' lengths if aligned to 'SectionAlignment'. It is a hint to the loader how many pages it will need in order to load the image in RAM.

Next 'SizeOfHeaders' @ offset D4h a 32-bit-value giving the total length of all headers including the data directories and the section headers, it is also the offset to the sections. Then we have got a 32-bit-checksum 'Checksum'. The algorithm to compute the checksum is property of Microsoft, and they won't tell you, it is only needed for Driver PEs. The checksum need not be supplied and may be 0. Then there is a 16-bit-word 'Subsystem' @ offset DCh IMAGE\_SUBSYSTEM\_WINDOWS\_GUI (2h). Windows 95 binaries will always use the Win32 subsystem, so the only legal values for these binaries are 2 and 3. The last thing in the second row is a 16-bit-value that tells, if the image is a DLL.

The above as seen in Quickview. ( C0h thru DFh )

Operating System Version:	4.00
Image Version:	1.00
Subsystem Version:	4.00
Reserved1:	00000000
Size of Image:	00005000
Size of Headers:	00000400
Checksum:	00000000
Subsystem:	Image runs in the Windows GUI subsystem.
DLL Characteristics:	0000

000000B0 00 00 10 00 00 10 00 00 00 50 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
000000C0 00 00 00 00 10 00

**First row:** The next 4 32-bit-values starting @ offset E0h are the size of stack reserve ('**SizeOfStackReserve**'), the size of initially committed stack ('**SizeOfStackCommit**' ), the size of the reserved heap ('**SizeOfHeapReserve**' ) and the size of the committed heap ('**SizeOfHeapCommit**' ).

**Second row:** After these stack- and heap-descriptions, we find 32 bits of '**'LoaderFlags'**'. Then the **Size of Data Directory**, number of possible entries in the ***following section table*** (**16 records**) 4 bytes @ offset F4h.called '**'NumberOfRvaAndSizes'**'. The last 8 bytes would be for the Export Directory pointers entry if used.

The above as seen in Quickview. ( E0h thru FFh )

Size of Stack Reserve: 00100000  
Size of Stack Commit: 00001000  
Size of Heap Reserve: 00005000  
Size of Heap Commit: 00001000  
Loader Flags: 00000000  
Size of Data Directory: 00000010

**First row:** The next 4 32-bit-values starting @ offset 100h are Import Directory Virtual Address (**'AppBaseOfImport'**), the Import Directory Size (**'AppImportSize'**), Resource Directory Virtual Address (**'AppBaseOfRsrc'**) and the Resource Directory Size (**'AppRsrcSize'**).

**Second row:** The 32-bit-values starting @ offset 110h and those following to 157h are filled with zeros and would be entries, RVAs and Sizes for: Exception, Security, Relocation Table, Debug, Copyright, Global Mips gp and Global Pointer, Thread Local Storage and Load Configuration, if used. The above as seen in QuickView, (100h thru 10Fh)

The above as seen in Quickview. ( 100h thru 10Fh )  
Import Directory Virtual Address: 1000  
    Import Directory Size: 003c  
    Resource Directory  
        Virtual Address: 2000  
    Resource Directory Size: 0800

**First row:** The 32-bit values starting @ offset 158h are the Second Import Address Table

**First row:** The 32-bit-values starting @ offset 158h are the Second Import Address Table 'AppSecondImport' and its size 'AppSecondImportSize'. This is explained below.

**Second row:** The space from 160h filled with zeros is reserved and the **Image Optional Header** ends at 177h.

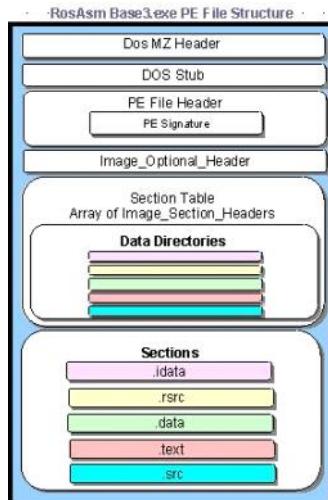
## The entire 'Image Optional Header' as seen in Quickview

## Image Optional Header

```

Magic:          010b
Linker Version:    3.00
Size of Code:      00000200
Size of Initialized Data: 00000e00
Size of Uninitialized Data: 00000000
Address of Entry Point: 00004000
Base of Code:      00004000
Base of Data:      00003000
Image Base:        00400000
Section Alignment: 00001000
File Alignment:    00000200
Operating System Version: 4.00
Image Version:    1.00
Subsystem Version: 4.00
        Prevernt f: 00000000
Size of Image:     00005000
Size of Headers:   00000400
Checksum:          00000000
Subsystem:         Image runs in the Windows GUI subsystem
DLL Characteristics: 0000
Size of Stack Reserve: 00100000
Size of Stack Commit: 00001000
Size of Heap Reserve: 00005000
Size of Heap Commit: 00001000
Loader Flags:       00000000
Size of Data Directory: 00000010
Import Directory Virtual Address: 1000
Import Directory Size: 003c
Resource Directory Virtual Address: 2000
Resource Directory Size: 0800

```



## Section Headers

Between the PE headers and the raw data for the image's Sections lies the **Section Table**.

There is one section header for each section, and each data directory will point to one of the sections. Several data directories may point to the same section, and there may be sections without a data

The sections in the image are sorted by their starting address.

Sections have two alignment values, one within the disk file (**Pointer to Raw Data**) and the other in memory (**Virtual Address**). The PE file header specifies both of these values, which can differ.

file, a typical alignment would be 200h. Thus, every section begins at a **file offset** that's a multiple of 200h. Once mapped into memory, sections always start on at least a **page boundary**. That is, when a PE section is mapped into memory, the first byte of each section corresponds to a memory page. On x86 CPUs pages are 4KB aligned, while on the IA-64 CPUs they are 8KB aligned.

The Section Table is an **array** of IMAGE\_NUMBER\_OF\_DIRECTORY\_ENTRIES (16 spaces reserved for entries) IMAGE\_DATA\_DIRECTORYs. Each of these directories describes the location (32 bits RVA called 'Virtual Address') and size (also 32 bit, called 'Size of Raw Data') of a particular piece of information, which is located in one of the sections that follow the directory entries. Some elements at the end of the array are currently unused.

This array allows the **loader** to quickly find a particular section of the image (for example, the imported function table), **without having to iterate** through each of the images sections, **comparing names** as it goes along. In the section table the first entry of an array element is of the SHORT\_NAME 8 bytes, that make up the name (in ASCII) of the section. If all of the 8 bytes are used there is no 0 terminator for the string... Followed by two sets of Dwords that are the sizes and the second ones that are the RVA addresses for each entry and data characteristics sometimes called flags.

Partial Section Records of Directory entries of Base3.exe each .jpg showing the relevant entry.

**Import Table - Data Section Header starts at 178h**

000000170	00 00 00 00 00 00 00 00 2E 69 64 61 74 61 00 00	00000000000000000000000000000000	.idata	00
000000180	04 02 00 00 00 10 00 00 00 04 00 00 00 04 00 00	00000000000000000000000000000000		
000000190	00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0	00000000000000000000000000000000	00000000000000000000000000000000	A

.idata': @ offset 178h padded with 2 zeros = 8 bytes, **AppImportTrueSize**: (Virtual Size) 204h @ offset 180h 4 bytes, **AppBaseOfImports**: RVA 1000h @ offset 184h 4 bytes, **AppImportAlignedSize**: 400h @ offset 188h 4 bytes, **AppStartOfImport**: 400h @ offset 18Ch 4 bytes, 12 bytes not used. The **flags** describing how the section's memory should be treated **DataCharacteristics DS 0\_C0000040** @ offset 19Ch; readable, writeable, initialized data.

The above as seen in Quickview. (170h thru 19Fh)

Section name:	.idata
Virtual Size:	000000204
Virtual Address:	00001000
Size of raw data:	000000400
Pointer to Raw Data:	000000400
Pointer to Relocations:	000000000
Pointer to Line Numbers:	000000000
Number of Relocations:	0000
Number of Line Numbers:	0000
Characteristics:	Section contains initialized data Section is readable Section is writeable

As we can see the .idata section starts at 400h in the file and is 400h bytes in length...

When you use code or data from a DLL, you're importing it. When any PE file loads, one of the jobs of the **Win32 loader** is to locate all the **imported functions and data and make those addresses available** to the file being loaded. When you link directly against the code and data of another DLL, you're implicitly linking against the DLL. You don't have to do anything to make the addresses of the imported APIs available to your code. Within a PE file, (Not shown in .jpgs) there's an **array of data structures**, one per imported DLL. Each of these structures gives the name of the imported DLL and points to an array of function pointers. The array of function pointers is known as the **Import Address Table (IAT)**. Each imported API has its own reserved spot in the IAT where the address of the imported function is written by the **Win32 loader**. This last point is particularly important: once a module is loaded into RAM, the IAT contains the address that is invoked when calling imported APIs. The beauty of the IAT is that there's **just one place in a PE file when loaded into RAM where an imported API's address is stored**. all the calls go through the same function pointer in the IAT.

Actual table as seen in Quickview.

#### Import Table

KERNEL32.dll	
Ordinal	Function Name
0000	GetModuleHandleA
0000	ExitProcess

USER32.dll	
Ordinal	Function Name
0000	LoadIconA
0000	LoadCursorA
0000	RegisterClassA
0000	LoadMenuA
0000	CreateWindowExA
0000	ShowWindow
0000	UpdateWindow
0000	TranslateMessage
0000	DispatchMessageA
0000	GetMessageA
0000	DestroyWindow
0000	PostQuitMessage
0000	SendMessageA
0000	MessageBoxA
0000	DefWindowProcA

The important parts of an import table are the imported DLL name and the **two arrays** of IMAGE\_IMPORT\_BY\_NAME pointers. In the EXE file, the **two arrays** (pointed to by the Characteristics and FirstThunk fields) run parallel to each other, and are terminated by a NULL pointer entry at the end of each array. The pointers in both arrays point to an IMAGE\_IMPORT\_BY\_NAME structure. Why are there two parallel arrays of pointers to the IMAGE\_IMPORT\_BY\_NAME structures? The array pointed to by the Characteristics field is **left alone, and never modified**. It's sometimes called the hint-name table. The array pointed to by the FirstThunk field is **overwritten by the PE loader**. The loader iterates through each pointer in the array and finds the address of the function that each IMAGE\_IMPORT\_BY\_NAME structure refers to. The loader then **overwrites** in RAM the pointer with the found function's address. Since the array of pointers that are **overwritten** by the loader eventually holds the addresses of all the imported functions, it's called the Import Address Table.

#### Resource - Data Section Header

000001A0	2E 72 73 72 63 00 00 00 80 06 00 00 20 00 00	00000000000000000000000000000000	.rsrc	00000000000000000000000000000000	
000001B0	00 08 00 00 00 08 00 00 00 00 00 00 00 00 00 00	00000000000000000000000000000000			
000001C0	00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00	00000000000000000000000000000000	00000000000000000000000000000000	.data	00

.rsrc': padded with 3 zeros = 8 bytes, **AppRsrcTrueSize**: virtual size 680h 4 bytes, **AppBaseOfRsrcs**: virtual address 200h 4 bytes, RVA **AppRsrcAlignedSize**: size of raw data 800h 4 bytes, **AppStartOfRsrc**: pointer to raw data 800h 4 bytes, 12 bytes not used. The **flags** describing how the section's memory should be treated, **DataCharacteristics:D\$ 0\_40000040**; readable, initialized data.

The resources, such as dialog boxes, menus, icons and so on, are in the data directory pointed to by IMAGE\_DIRECTORY\_ENTRY\_RESOURCE.

It is in a section that has, at least, the bits 'IMAGE\_SCN\_CNT\_INITIALIZED\_DATA' and 'IMAGE\_SCN\_MEM\_READ' set.

The above as seen in Quickview. (1A0h thru 1C7h)



The above IMAGE\_DATA\_DIRECTORIES in **178h** to **267h** are followed by unused space and are padded with zeros to the file boundary **3FFh**.... As we saw above the Import Section begins at **400H** , Resource Section at **800h**, Data Section at **1000h**, Text at **1200h**, RosAsm source at **1400h**. Each of these sections are also padded with zeros to the file boundaries, which in our files is **200h > 512 bytes** and just happens to be the exact size of a single disk sector..... As we have also seen the RVAs and offsets are used by the Win32 loader to map the file into memory differently....

[To see how the file is loaded into memory click here.....](#)

#### The Sections found in RosAsm Base3.exe

The .idata section contains information about functions (and data) that the module imports from other DLLs.

The .rsrc section contains all the resources for the module.

The .data section is where your initialized data goes.

The .text section is where all general-purpose code emitted by the compiler or assembler resides.

The .src section contains all the RosASM source code for the module.

#### Other Sections that you may encounter in other PE files

The .bss section is where any uninitialized static and global variables are stored.

The .crt is another initialized data section utilized by the Microsoft C/C++ run-time libraries (hence the name).

The .edata section is a list of the functions and data that the PE file exports for other modules.

The .reloc section holds a table of base relocations. A base relocation is an adjustment to an instruction or initialized variable value that's needed if the loader couldn't load the file where the linker assumed it would. If the loader is able to load the image at the linker's preferred base address, the loader completely ignores the relocation information in this section.

The .tls section, which refers to "thread local storage," and is related to the TlsAlloc family of Win32 functions. When dealing with a .tls section, the memory manager sets up the page tables so that whenever a process switches threads, a new set of physical memory pages is mapped to the .tls section's address space. This permits per-thread global variables.

The .rdata section is used for at least two things. First, in Microsoft linker-produced EXEs, the .rdata section holds the debug directory, which is only present in EXE files. The other useful portion of an .rdata section is the description string.

**NOTE:** There seems to be a wide latitude in the way PE files are written by the various

Compilers/Linkers and if you look at a number of different PE files you will find that some of them may not adhere to what is found in this document. Nevertheless most of it applies and the Win32 Loader is capable of uploading them into RAM, therefore the loader seems to be quite flexible in its own right. It also seems that while the Linker Version is present in the Optional Header it is useless as it does not identify which linker it is.... Of course we do know that **RosAsm** wrote our Base3.exe....

[Home page](#) <.>[Links Page](#)

Notes about **Relative Virtual Addresses**: The PE format makes heavy use of so-called RVAs. An RVA, aka "relative virtual address", is used to describe a memory address if you don't know the **image base** address. It is the value you need to add to the image base address to get the actual linear address. The base address is the address the PE image is loaded to in RAM, and may vary from one invocation to the next. **Example:** Suppose an executable file is loaded to address **400000h** and program execution starts at RVA **4000h**. The effective execution start will then be at the address **404000h**. If the executable were loaded to **100000h**, the execution start would be **104000h**.

...Defined **directory indexes** are:

IMAGE\_DIRECTORY\_ENTRY\_EXPORT (0) The directory of exported symbols. mostly used for DLLs.

IMAGE\_DIRECTORY\_ENTRY\_IMPORT (1) The directory of imported symbols.

IMAGE\_DIRECTORY\_ENTRY\_RESOURCE (2) Directory of resources.

IMAGE\_DIRECTORY\_ENTRY\_EXCEPTION (3) Exception directory - structure and purpose unknown.

IMAGE\_DIRECTORY\_ENTRY\_SECURITY (4) Security directory - structure and purpose unknown.

IMAGE\_DIRECTORY\_ENTRY\_BASERELOC (5) Base relocation table. Necessary for DLLs.

IMAGE\_DIRECTORY\_ENTRY\_DEBUG (6) Debug directory - contents is compiler dependent. Moreover, many compilers stuff the debug information into the code section and don't create a separate section for it.

IMAGE\_DIRECTORY\_ENTRY\_COPYRIGHT (7) Description string - some arbitrary copyright note or the like.

IMAGE\_DIRECTORY\_ENTRY\_GLOBALPTR (8) Machine Value (MIPS GP) - structure and purpose unknown.

IMAGE\_DIRECTORY\_ENTRY\_TLS (9) Thread local storage directory - structure unknown; contains variables that are declared "\_\_declspec(thread)", i.e. per-thread global variable.

IMAGE\_DIRECTORY\_ENTRY\_LOAD\_CONFIG (10) Load configuration directory - structure and purpose unknown.

IMAGE\_DIRECTORY\_ENTRY\_BOUND\_IMPORT (11) Bound import directory - see description of import directory.

IMAGE\_DIRECTORY\_ENTRY\_IAT (12) Import Address Table - see description of import directory.

As an example, if we find at index 7 the 2 longwords **1200h** and **33h**, and the load address is **1000h**, we know that the copyright data is at address **10000h+1200h** (in whatever section there may be), and the copyright note is **33h** bytes long. If a directory of a particular type is not used in a binary, the Size and VirtualAddresses are both set to **0h**.

#### Flags:

What most programmers call flags, the COFF/PE format calls **characteristics**. This field is a set of flags that indicate the section's attributes (such as code/data, readable, or writeable.).

For a complete list of all possible section attributes, see the IMAGE\_SCN\_XXX\_XXX

#defines in WINNT.H. Some of the more important flags are shown below:

0x00000020 This section **contains code**. Usually set in conjunction with the executable flag (0x80000000).

0x00000040 This section **contains initialized data**. Almost all sections except executable and the .bss section have this flag set.

0x00000080 This section **contains uninitialized data** (for example, the .bss section).

0x00000200 This section **contains comments** or some other type of information. A typical use of this section is the .directive section emitted by the compiler, which contains commands for the linker.

0x00000800 This section's contents shouldn't be put in the final EXE file. These sections are used by the compiler/assembler to pass information to the linker.

0x02000000 This section **can be discarded**, since it's not needed by the process once it's been loaded. The most common discardable section is the base relocations (.reloc).

0x10000000 This section is **shareable**. When used with a DLL, the data in this section will be shared among all processes using the DLL. The default is for data sections to be nonshared, meaning that each process using a DLL gets its own copy of this section's data. In more technical terms, a shared section tells the memory manager to set the page mappings for this section such that all processes using the DLL refer to the same physical page in memory. To make a section shareable, use the SHARED attribute at link time. For example

0x20000000 This section is **executable**. This flag is usually set whenever the "contains

code" flag (0x000000020) is set.  
0x40000000 This section is **readable**. This flag is almost always set for sections in EXE files.  
0x80000000 The section is **writable**. If this flag isn't set in an EXE's section, the loader should mark the memory mapped pages as read-only or execute-only. Typical sections with this attribute are .data and .bss. Interestingly, the .idata section also has this attribute set.  
/ More Section characteristics.  
0x00000000 // Reserved.  
0x00000001 // Reserved.  
0x00000002 // Reserved.  
0x00000004 // Reserved.  
0x00000008 // Reserved.  
0x00000010 // Reserved.  
0x00000020 // Section contains code.  
0x00000040 // Section contains initialized data.  
0x00000080 // Section contains uninitialized data.  
0x00000100 // Reserved.  
0x00000200 // Section contains comments or some other type of information.  
0x00000400 // Reserved.  
0x00000800 // Section contents will not become part of image.  
0x00001000 // Section contents comdat.  
0x00002000 // Reserved.  
Obsolete 0x00004000  
0x00008000 // Section content can be accessed relative to GP.

#### Acknowledgements:

To complete this tutorial I primarily used information that I learned from:  
Rosasm source code, Matt Pietrek's article on MSDN & from B.Luevlsmeier at iplan.heitec.net and other information, such as winnt.h file with Borland Bcc55, I had in my files.

Words of Matt Pietrek:

"A good understanding of the Portable Executable (PE) file format leads to a good understanding of the operating system. If you know what's in your DLLs and EXEs, you'll be a more knowledgeable programmer.".

"You might be wondering why you should care about the executable file format. The answer is the same now as it was then: an operating system's executable format and data structures reveal quite a bit about the underlying operating system. By understanding what's in your EXEs and DLLs, you'll find that you've become a better programmer all around."

[Home page](#) < [Links Page](#)

Pegado de <<http://www.thehackademy.net/madchat/vxdev/papers/winsys/pefile/pefile.htm>>

## Finding program's entry point

```

00401689 | > 73 ED    L_INB SHORT 00401679
0040168C | > 5E          POP ESI
0040168D | > 68 18504000 PUSH OFFSET 00405018
00401692 | > 68 14504000 PUSH OFFSET 00405014
00401697 | > 68 24000000 PUSH OFFSET 004016C6
0040169C | > 59          POP ECX
0040169D | > 59          POP ECX
0040169E | > 68 20504000 PUSH OFFSET 00405020
0040169F | > 68 15504000 PUSH OFFSET 0040501C
004016A0 | > 68 19000000 CALL 004016C6
004016A4 | > 5F          POP EDI
004016A5 | > 5F          POP ECX
004016A6 | > 5F          TEST EBX, EBX
004016A7 | > 5B          POP EBX
004016B1 | > 75 18    JNE SHORT 004016C4
004016B4 | > FF7424 08 PUSH DWORD PTR SS:[ARG_1]
004016B5 | > 8B3D 00540000 MOV DWORD PTR DS:[4095B0], EDI
004016B6 | > 8F15 3C484000 CALL DWORD PTR DS:[&KERNEL32.ExitProcess]
004016C4 | > 5F          POP EDI
004016C5 | C3          RETN
004016C6 | > 5E          POP ESI
004016C7 | > 8B7424 08 MOU ESI,DWORD PTR SS:[ARG_1]
004016CB | > 8B7424 0C FCMPI ESI,DWORD PTR SS:[ARG_2]
004016CE | > 73 00    INC ESI,DWORD PTR SS:[ARG_2]
004016CF | > FF7424 0C FCMPI ESI,DWORD PTR SS:[ARG_2]

```

Corte de pantalla realizado: 01/08/2014 2:32

Usually, just above the ExitProcess function, there is a jump to the main function.

```

27B4
400
40140E -> 4016B0
MSVCRT._cexit
KERNEL32.ExitProcess
MOV EBX,EAX (89C3)
CALL <JMP.&msvcrt._cexit> (E8 26220000)
CALL <JMP.&kernel32.ExitProcess> (E8 F6220000)

CALL DWORD PTR DS:[KERNEL32.ExitProcess]

```

```

0040141D . E8 F6220000 CALL <JMP.&kernel32.ExitProcess> ; \ExitProcess

```

## Predefined Sections

An application for Windows NT typically has the nine predefined sections named .text, .bss, .rdata, .data, .rsrc, .edata, .idata, .pdata, and .debug.

Some applications do not need all of these sections, while others may define still more sections to suit their specific needs. This behavior is similar to code and data segments in MS-DOS and Windows version 3.1. In fact, the way an application defines a unique section is by using the standard compiler directives for naming code and data segments or by using the name segment compiler option **-NT**--exactly the same way in which applications defined unique code and data segments in Windows version 3.1.

The following is a discussion of some of the more interesting sections common to typical Windows NT PE files.

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

### Executable code section, .text

One difference between Windows version 3.1 and Windows NT is that the default behavior combines all code segments (as they are referred to in Windows version 3.1) into a single section called ".text" in Windows NT. Since Windows NT uses a page-based virtual memory management system, there is no advantage to separating code into distinct code segments. Consequently, having one large code section is easier to manage for both the operating system and the application developer.

The .text section also contains the entry point mentioned earlier. The IAT also lives in the .text section immediately before the module entry point. (The IAT's presence in the .text section makes sense because the table is really a series of jump instructions, for which the specific location to jump to is the fixed-up address.) When Windows NT executable images are loaded into a process's address space, the IAT is fixed up with the location of each imported function's physical address. In order to find the IAT in the .text section, the loader simply locates the module entry point and relies on the fact that the IAT occurs immediately before the entry point. And since each entry is the same size, it is easy to walk backward in the table to find its beginning.

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

#### Data sections, .bss, .rdata, .data

The .bss section represents uninitialized data for the application, including all variables declared as static within a function or source module.

The .rdata section represents read-only data, such as literal strings, constants, and debug directory information.

All other variables (except automatic variables, which appear on the stack) are stored in the .data section. Basically, these are application or module global variables.

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

#### Resources section, .rsrc

The .rsrc section contains resource information for a module. It begins with a resource directory structure like most other sections, but this section's data is further structured into a resource tree. The **IMAGE\_RESOURCE\_DIRECTORY**, shown below, forms the root and nodes of the tree.

WINNT.H

```
typedef struct _IMAGE_RESOURCE_DIRECTORY {
    ULONG    Characteristics;
    ULONG    TimeDateStamp;
    USHORT   MajorVersion;
    USHORT   MinorVersion;
    USHORT   NumberOfNamedEntries;
    USHORT   NumberOfIdleEntries;
} IMAGE_RESOURCE_DIRECTORY, *PIMAGE_RESOURCE_DIRECTORY;
```

Recorte de pantalla realizado: 01/08/2014 13:04

A directory entry consists of two fields, as described in the following **IMAGE\_RESOURCE\_DIRECTORY\_ENTRY** structure:

```
WINNT.H
typedef struct _IMAGE_RESOURCE_DIRECTORY_ENTRY {
    ULONG    Name;
    ULONG    OffsetToData;
} IMAGE_RESOURCE_DIRECTORY_ENTRY, *PIMAGE_RESOURCE_DIRECTORY_ENTRY;
```

Recorte de pantalla realizado: 01/08/2014 13:04

Leaf nodes are the lowest node in the resource tree. They define the size and location of the actual resource data. Each leaf node is represented using the following **IMAGE\_RESOURCE\_DATA\_ENTRY** structure:

```
WINNT.H
typedef struct _IMAGE_RESOURCE_DATA_ENTRY {
    ULONG    OffsetToData;
    ULONG    Size;
    ULONG    CodePage;
    ULONG    Reserved;
} IMAGE_RESOURCE_DATA_ENTRY, *PIMAGE_RESOURCE_DATA_ENTRY;
```

Recorte de pantalla realizado: 01/08/2014 13:05

To make all of this a little clearer, consider Figure 2.

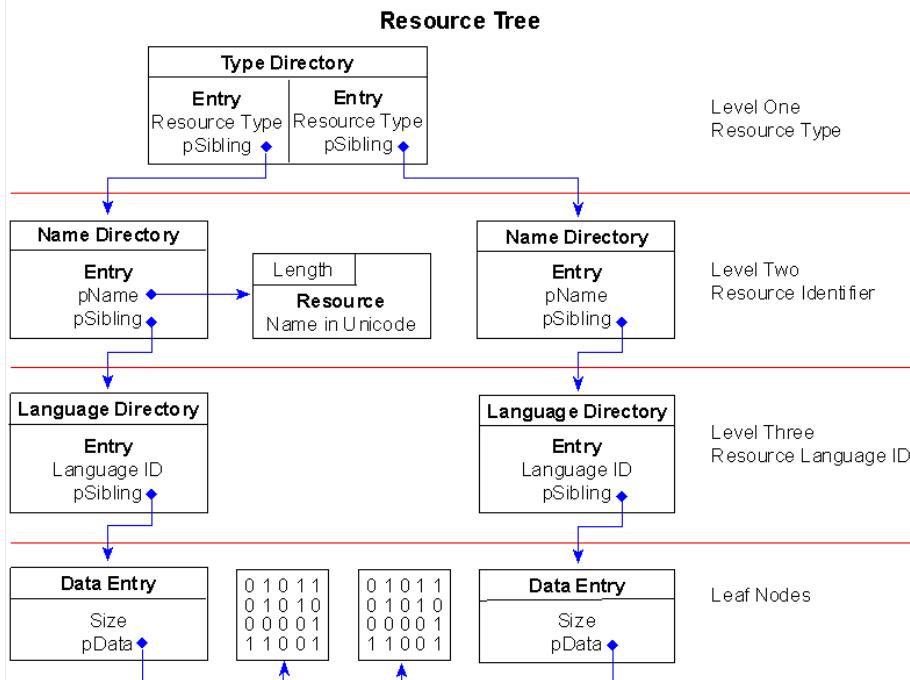


Figure 2. A simple resource tree structure

Recorte de pantalla realizado: 01/08/2014 13:05

Figure 2. A simple resource tree structure

Figure 2 depicts a very simple resource tree containing only two resource objects, a menu, and a string table. Further, the menu and string table have only one item each. Yet, you can see how complicated the resource tree becomes--even with as few resources as this.

At the root of the tree, the first directory has one entry for each type of resource the file contains, no matter how many of each type there are. In Figure 2, there are two entries identified by the root, one for the menu and one for the string table. If there had been one or more dialog resources included in the file, the root node would have had one more entry and, consequently, another branch for the dialog resources.

The basic resource types are identified in the file WINUSER.H and are listed below:

#### WINUSER.H

```
/*
 * Predefined Resource Types
 */
#define RT_CURSOR      MAKEINTRESOURCE(1)
#define RT_BITMAP      MAKEINTRESOURCE(2)
#define RT_ICON        MAKEINTRESOURCE(3)
#define RT_MENU        MAKEINTRESOURCE(4)
#define RT_DIALOG      MAKEINTRESOURCE(5)
#define RT_STRING      MAKEINTRESOURCE(6)
#define RT_FONTDIR    MAKEINTRESOURCE(7)
#define RT_FONT        MAKEINTRESOURCE(8)
#define RT_ACCELERATOR MAKEINTRESOURCE(9)
#define RT_RCDATA      MAKEINTRESOURCE(10)
#define RT_MESSAGETABLE MAKEINTRESOURCE(11)
```

Recorte de pantalla realizado: 01/08/2014 13:05

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

#### Export data section, .edata

The .edata section contains export data for an application or DLL. When present, this section contains an export directory for getting to the export information.

#### WINNT.H

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    ULONG    Characteristics;
    ULONG    TimeDateStamp;
    USHORT   MajorVersion;
    USHORT   MinorVersion;
    ULONG    Name;
    ULONG    Base;
    ULONG    NumberOfFunctions;
    ULONG    NumberOfNames;
    PULONG   *AddressOfFunctions;
    PULONG   *AddressOfNames;
    PUSHORT  *AddressOfNameOrdinals;
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

Recorte de pantalla realizado: 01/08/2014 13:06

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

#### Import data section, .idata

The .idata section is import data, including the import directory and import address

name table.

Although an IMAGE\_DIRECTORY\_ENTRY\_IMPORT directory is defined, no corresponding import directory structure is included in the file WINNT.H. Instead, there are several other structures called IMAGE\_IMPORT\_BY\_NAME, IMAGE\_THUNK\_DATA, and IMAGE\_IMPORT\_DESCRIPTOR.

```
PEFILE.H
typedef struct tagImportDirectory
{
    DWORD     dwRVAFunctionNameList;
    DWORD     dwUseless1;
    DWORD     dwUseless2;
    DWORD     dwRVAModuleName;
    DWORD     dwRVAFunctionAddressList;
} IMAGE_IMPORT_MODULE_DIRECTORY,
* PIMAGE_IMPORT_MODULE_DIRECTORY;
```

Recorte de pantalla realizado: 01/08/2014 13:06

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

### Debug information section, .debug

Debug information is initially placed in the .debug section. The PE file format also supports separate debug files (normally identified with a .DBG extension) as a means of collecting debug information in a central location. The debug section contains the debug information, but the debug directories live in the .rdata section mentioned earlier. Each of those directories references debug information in the .debug section.

```
WINNT.H
typedef struct _IMAGE_DEBUG_DIRECTORY {
    ULONG    Characteristics;
    ULONG    TimeDateStamp;
    USHORT   MajorVersion;
    USHORT   MinorVersion;
    ULONG    Type;
    ULONG    SizeOfData;
    ULONG    AddressOfRawData;
    ULONG    PointerToRawData;
} IMAGE_DEBUG_DIRECTORY, *PIMAGE_DEBUG_DIRECTORY;
```

The section is divided into separate portions of data representing debug information are listed below:

```
WINNT.H
#define IMAGE_DEBUG_TYPE_UNKNOWN      0
#define IMAGE_DEBUG_TYPE_COFF         1
#define IMAGE_DEBUG_TYPE_CODEVIEW     2
#define IMAGE_DEBUG_TYPE_FPO          3
#define IMAGE_DEBUG_TYPE_MISC         4
```

Recorte de pantalla realizado: 01/08/2014 13:07

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

## Summary of the PE File Format

The PE file format for Windows NT introduces a completely new structure to developers familiar with the Windows and MS-DOS environments. Yet developers familiar with the UNIX environment will find that the PE file format is similar to, if not based on, the COFF specification.

The entire format consists of an MS-DOS MZ header, followed by a real-mode stub program, the PE file signature, the PE file header, the PE optional header, all of the section headers, and finally, all of the section bodies.

The optional header ends with an array of data directory entries that are relative virtual addresses to data directories contained within section bodies. Each data directory indicates how a specific section body's data is structured.

The PE file format has eleven predefined sections, as is common to applications for Windows NT, but each application can define its own unique sections for code and data.

The .debug predefined section also has the capability of being stripped from the file into a separate debug file. If so, a special debug header is used to parse the debug file, and a flag is specified in the PE file header to indicate that the debug data has been stripped.

Pegado de <<http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html>>

Table 2 - Section names

".text"	Code Section
"CODE"	Code Section of file linked by Borland Delphi or Borland Pascal
".data"	Data Section
"DATA"	Data Section of file linked by Borland Delphi or Borland Pascal
".rdata"	Section for Constant Data
".idata"	Import Table
".edata"	Export Table
".tls"	TLS Table
".reloc"	Relocation Information
".rsrc"	Resource Information

Recorte de pantalla realizado: 01/08/2014 13:09

# Microsoft Windows library files

viernes, 01 de agosto de 2014

13:19

From Wikipedia, the free encyclopedia



[\[hide\]](#) This article has multiple issues. Please help [improve it](#) or discuss these issues on the [talk page](#).

This article's [lead section](#) may not adequately [summarize](#) key points of its contents. (February 2013)

This article [needs additional citations for verification](#). (February 2013)

The [Microsoft Windows operating system](#) supports a form of [shared libraries](#) known as [dynamic-link libraries](#), which are code libraries that can be used by multiple processes while only one copy is loaded into [memory](#). This article provides an overview of the core libraries that are included with every modern Windows installation, on top of which most Windows applications are built.

## Contents

- [1 Internal components](#)
  - [1.1 Hal.dll](#)
  - [1.2 NTDLL.DLL](#)
- [2 Win32 API](#)
  - [2.1 KERNEL32.DLL](#)
  - [2.2 GDI32.DLL](#)
  - [2.3 USER32.DLL](#)
  - [2.4 COMCTL32.DLL](#)
- [3 Other APIs](#)
  - [3.1 SHSCRAP.DLL](#)
- [4 Runtime libraries](#)
  - [4.1 MSVCRT.DLL and MSVCP.DLL](#)
  - [4.2 Other runtime libraries](#)
  - [4.3 .NET Framework libraries](#)
- [5 See also](#)
- [6 References](#)
- [7 External links](#)

## Internal components

The library files in this section are not used directly by most programs; however, they are a dependency of other libraries that are used by programs. [[citation needed](#)]

### Hal.dll

The Windows [Hardware Abstraction Layer](#) (HAL) is implemented in **Hal.dll**.<sup>[1]</sup> The HAL implements a number of functions that are implemented in different ways by different hardware platforms. Other components in the operating system can then call these functions in the same way on all platforms, without regard for the actual implementation.

For example, responding to an interrupt is quite different on a machine with an [Advanced Programmable Interrupt Controller \(APIC\)](#) than on one without. The HAL creates one, use-all function that works with all kinds of interrupts by various chipsets.

The HAL is loaded into kernel address space and runs in kernel mode, so routines in the HAL cannot be called directly by applications, and no user mode APIs correspond directly to HAL routines.

Instead, the HAL provides services primarily to the Windows executive and kernel and to kernel mode device drivers. Although drivers for most hardware are contained in other files, commonly of file type .sys, a few core drivers are compiled into Hal.dll.

Kernel mode device drivers for devices on buses such as [PCI](#) and [PCI Express](#) directly call routines in the HAL to access [I/O ports](#) and registers of their devices. The drivers use HAL routines because different platforms may require different implementations of these operations. The HAL implements the operations appropriately for each platform, so the same driver executable file can be used on all

platforms using the same CPU architecture, and the driver source file can be portable across all architectures.

On [x86](#) systems, there are several different HAL files on the installation media. The Windows installation procedure determines which ones are appropriate for the current platform and copies it to the hard drive, renaming it to Hal.dll if necessary. Among the criteria for this selection are: the presence of an [ACPI](#)-compatible BIOS, the presence of an [APIC](#), and whether or not multiple processors are present and enabled. (The multiple cores of a [multi-core CPU](#), and even the "logical processors" implemented by a [hyperthreading](#) CPU, all count as "processors" for this purpose.) On x86-64 and Itanium platforms there is just one possible Hal.dll for each CPU architecture.

If a PC running Windows 7 is turned on while missing Hal.dll (as would be the case if the system were restarted after deleting C:\Windows\System32), instead of starting normally, the screen will display white text on a black background(terminal style) stating that the system failed to start because of this file being nonexistent. Since it displays this file in the error message even if the entire System32 folder is deleted, this implies that it is the first file loaded during the boot process.

## NTDLL.DLL

NTDLL.DLL exports the Windows [Native API](#). The Native API is the interface used by user-mode components of the operating system that must run without support from Win32 or other API subsystems. Most of this API is implemented in **NTDLL.DLL** and at the upper edge of [ntoskrnl.exe](#) (and its variants), and the majority of exported symbols within these libraries are prefixed **Nt**, for example **NtDisplayString**. Native APIs are also used to implement many of the "kernel APIs" or "base APIs" exported by KERNEL32.DLL.<sup>[2][3][4]</sup> The large majority of Windows applications do not call NTDLL.DLL directly.<sup>[5]</sup>

Applications that are [linked](#) directly against this library are known as **native applications**; the primary reason for their existence is to perform tasks that must run early in the system startup sequence before the Win32 subsystem is available. An obvious but important example is the creation of the Win32 subsystem process, [csrss.exe](#). Before the csrss.exe process exists, no Win32 processes may be created, therefore the process that creates it (Smss.exe, the "session manager") must be a native application. [csrss.exe](#) itself is a native application.

Despite having an ".exe" file extension, native applications cannot be executed by the user (or any program in the Win32 or other subsystems). An example is the **autochk.exe** binary that runs [chkdsk](#) during the system initialization "Blue Screen". Other prominent examples are the services that implement the various subsystems, such as [csrss.exe](#).

Unlike [Win32](#) applications, native applications instantiate within the Kernel runtime code ([ntoskrnl.exe](#)) and so they must have a different entry point (**NtProcessStartup**, rather than **(w)** **(Win)MainCRTStartup** as is found in a Win32 application),<sup>[3]</sup> obtain their command-line arguments via a pointer to an in-memory structure, manage their own memory using the **Rtl** heap API,(which the Win32 heap APIs are just wrappers around -- no real difference there) and return execution with a call to [NtTerminateProcess](#) (as opposed to [ExitProcess](#)). A common library linked with Native applications is nt.lib, which contains startup code for Native applications, similar to how the C runtime provides startup code for Win32 apps.<sup>[6]</sup>

Though most of the API is undocumented, Native Applications can be built using the [Windows Driver Development Kit](#); many [AntiVirus](#) and other utility software vendors incorporate Native Applications within their products, usually to perform some boot-time task that cannot be carried out in [userspace](#).<sup>[citation needed]</sup>

## Win32 API

For more details on this topic, see [Windows API](#).

The libraries in this section each implement various subsets of the Win32 API.

## KERNEL32.DLL

KERNEL32.DLL exposes to applications most of the Win32 base APIs, such as [memory management](#), [input/output \(I/O\)](#) operations, [process](#) and [thread](#) creation, and synchronization functions. Many of these are implemented within KERNEL32.DLL by calling corresponding functions in the [native API](#), exposed by NTDLL.DLL.<sup>[7]</sup>

## GDI32.DLL

GDI32.DLL exports [Graphics Device Interface \(GDI\)](#) functions that perform primitive drawing functions for output to video displays and printers. Applications call GDI functions directly to

perform low-level drawing, text output, font management, and similar functions.<sup>[7][8]</sup> Initially, GDI supported 16 and 256 color [EGA/VGA display cards](#) and [monochrome](#) printers. The functionality has expanded over the years, and now includes support for things like [TrueType fonts](#), [alpha channels](#), and [multiple monitors](#).<sup>[9]</sup>

## USER32.DLL

Further information: [Windows USER](#)

**USER32.DLL** implements the Windows USER component that creates and manipulates the standard elements of the Windows user interface, such as the desktop, windows, and menus. It thus enables programs to implement a [graphical user interface \(GUI\)](#) that matches the Windows look and feel. Programs call functions from Windows USER to perform operations such as creating and managing windows, receiving window messages (which are mostly user input such as mouse and keyboard events, but also notifications from the operating system), displaying text in a window, and displaying message boxes.

Many of the functions in USER32.DLL call upon GDI functions exported by GDI32.DLL to do the actual rendering of the various elements of the user interface. Some types of programs will also call GDI functions directly to perform lower-level drawing operations within a window previously created via USER32 functions.

## COMCTL32.DLL

**COMCTL32.DLL** implements a wide variety of standard Windows controls, such as File Open, Save, and Save As dialogs, progress bars, and list views. It calls functions from both USER32.DLL and GDI32.DLL to create and manage the windows for these UI elements, place various graphic elements within them, and collect user input.

## Other APIs

### SHSCRAP.DLL

**SHSCRAP.DLL** is part of the [Object Linking and Embedding \(OLE\)](#) mechanism. It implements support for shell scrap files, which are automatically created when you drag selected content from an OLE-capable application into an Explorer window or desktop,<sup>[10]</sup> but you can also use the [Object Packager](#) to create them. They can then be dragged into another OLE-capable application.

This functionality was removed from Windows Vista (and therefore later versions) to improve security and rid the operating system of generally unused functionality.<sup>[11]</sup> Scrap (.shs) files have been used by viruses because they can contain a wide variety of files (including executable code), and the file extension is not shown even when "Hide file extensions from known file types" is disabled.<sup>[12]</sup> The functionality can be restored by copying registry entries and the DLL from a Windows XP system.<sup>[13]</sup>

## Runtime libraries

### MSVCRT.DLL and MSVCPP.DLL

**MSVCRT.DLL** is the Microsoft Visual [C Run-Time Library](#) (**MSVCPP.DLL** being the C++ library) for Visual C++ version 4.2 to 6.0. It provides programs compiled with these versions of [Visual C++](#) as well as a typical set of library functions required by C and C++ programs. These include string manipulation, memory allocation, C-style input/output calls, among others.

It has shipped with Windows versions since Windows 2000 for use by other Windows components. In older versions of Windows, programs which linked against MSVCRT.DLL were expected to install a compatible copy in the System32 folder, but this contributed to [DLL Hell](#).

Versions of Visual C++ before 4.0 and since 7.0 have used differently named DLLs for each version (MSVCR20.DLL, MSVCR70.DLL, MSVCR71.DLL, MSVCP110.DLL etc.). Applications are required to install the appropriate version.<sup>[14]</sup>

Microsoft Visual C++ Run-Time is included in Windows. Runtimes newer than installed OS are available in packages called **Visual C++ Redistributables**.

Source code for runtime libraries is included in Visual C++<sup>[15]</sup> for reference and debugging (e.g. in C:\Program Files\Microsoft Visual Studio 11.0\VC\crt\src).

This runtime library is used by programs written in Visual C++ and a few other compilers (e.g. [MinGW](#)). Some compilers have their own runtime libraries.

## Other runtime libraries

- **ATL\*.DLL** — [Active Template Library](#)
- **MFC\*.DLL** — [Microsoft Foundation Classes](#)
- **MSVBVM60.DLL** — [Visual Basic](#) 6.0 Virtual Machine ([Visual Basic.NET](#) programs require [.NET Framework](#) instead)

## .NET Framework libraries

Programs written in [C#](#), [Visual Basic.NET](#), [C++/CLI](#) and other .NET languages require the [.NET Framework](#). It has many libraries (one of them is **mscorlib.dll** — Multilanguage Standard Common Object Runtime Library, formerly Microsoft Common Object Runtime Library[\[16\]](#)) and so-called assemblies (e.g. **System.Windows.Forms.dll**).

Pegado de <[http://en.wikipedia.org/wiki/Microsoft\\_Windows\\_library\\_files#Hal.dll](http://en.wikipedia.org/wiki/Microsoft_Windows_library_files#Hal.dll)>

# Written by Ethan Brodsky

## Version 3.5 - 6/26/97

Note: 97/6/11

I am experimenting with using transparency in the images to allow irregularly-shaped diagrams without making assumptions about your browser's background color. If the DSP command or mode diagrams are blue, please contact me with information on your configuration so I can attempt to fix it.

[Download SB16DOC.ZIP](#) (6,509 bytes)

### Jump to section:

- [Disclaimer](#)
- [Introduction](#)
- [Sound Blaster 16 DSP I/O Ports](#)
- [Resetting the DSP](#)
- [Writing to the DSP](#)
- [Reading from the the DSP](#)
- [Programming the DMA Controller](#)
- [Setting the sampling rate](#)
- [Digitized sound I/O](#)
- [DSP Commands](#)
- [Auto-initialized DMA](#)
- [References](#)

### Disclaimer

This information is distributed AS IS. The author specifically disclaims responsibility for any loss of profit or any consequential, incidental, or other damages resulting from the use or misuse of this information. This information may be freely distributed in any form as long as this disclaimer remains intact.

### Introduction

The Sound Blaster 16 is capable of both FM and digitized sounds. Digitized sound capabilities range from 8-bit mono 5000 HZ sound all the way up to 16-bit stereo sound at 44khz. This FAQ documents programming the SB16 DSP CT1341 chip for recording and playback of digitized audio. Prior knowledge on programming earlier Sound Blaster sound cards is necessary.

### The Sound Blaster 16 DSP I/O Ports

The SB16's DSP chip is programming using several I/O ports at a base I/O address determined by jumper settings. On the SB16, there are 16 I/O ports which are used for FM synthesized music, mixer settings, DSP programming and CD-ROM access. Five of these ports are used in programming the DSP. They are listed below.

- 2x6h - DSP Reset
- 2xAh - DSP Read
- 2xCh - DSP Write (Command/Data), DSP write-buffer status (Bit 7)
- 2xEh - DSP Read-buffer status (Bit 7), DSP interrupt acknowledge
- 2xFh - DSP 16-bit interrupt acknowledge

### Resetting the DSP

You have to reset the DSP before you can program it. The DSP can be reset using the following procedure:

1. Write a 1 to the reset port (2x6)
2. Wait for 3 microseconds

3. Write a 0 to the reset port (2x6)
4. Poll the read-buffer status port (2xE) until bit 7 is set
5. Poll the read data port (2xA) until you receive an AA

The DSP usually takes about 100 microseconds to initialize itself. After this period of time, if the return value is not AA or there is no data at all, then the SB card may not be installed or an incorrect I/O address is being used.

## Writing to the DSP

To write a byte to the SB16, the following procedure should be used:

1. Read the write-buffer status port (2xC) until bit 7 is cleared
2. Write the value to the write port (2xC)

## Reading from the DSP

To read a byte from the SB16, the following procedure should be used:

1. Read the read-buffer status port (2xE) until bit 7 is set
2. Read the value from the read port (2xA)

## Programming the DMA Controller

The DMA (Direct Memory Access) controller controls data transfers between I/O devices and memory without using the CPU. An Intel 8237 DMAC integrated circuit is used to control this. An IBM compatible computer has two DMA controllers, one for 8-bit transfers and one for 16-bit transfers. The DMA controller, coupled with an external page register, is capable of transferring blocks of up to 64k to the SB16. Here is information on I/O ports and register settings necessary for sound card I/O:

### I/O port addresses for the DMA Address and Count Registers

Controller	I/O Address	Function
DMA 1	00	Channel 0 address
8-bit	01	Channel 0 count
Slave	02	Channel 1 address
	03	Channel 1 count
	04	Channel 2 address
	05	Channel 2 count
	06	Channel 3 address
	07	Channel 3 count
DMA 2	C0	Channel 4 address
16-bit	C2	Channel 4 count
Master	C4	Channel 5 address
	C6	Channel 5 count
	C8	Channel 6 address
	CA	Channel 6 count
	CC	Channel 7 address
	CE	Channel 7 count

### I/O port addresses for the control registers

Address		Function
DMAC1	DMAC2	
0A	D4	Write single mask register
0B	D6	Write mode register
0C	D8	Clear byte pointer flip-flop

### I/O port addresses for lower page registers

I/O Address	Function
87	8-bit DMA channel 0 page
83	8-bit DMA channel 1 page
81	8-bit DMA channel 2 page
82	8-bit DMA channel 3 page
8B	16-bit DMA channel 5 page
89	16-bit DMA channel 6 page
8A	16-bit DMA channel 7 page

#### Mode register bit assignments

Bits/value	Function
Bits 7:6	<b>Mode selection bits</b>
00	<b>Demand mode selected</b>
01	<b>Single mode selected</b>
10	<b>Block mode selected</b>
11	<b>Cascade mode selected</b>
Bit 5	<b>Address increment/decrement</b>
0	<b>Address increment selected</b>
1	<b>Address decrement selected</b>
Bit 4	<b>Auto-initialization enable</b>
0	<b>Single-cycle DMA</b>
1	<b>Auto-initialized DMA</b>
Bits 3:2	<b>Transfer bits</b>
00	<b>Verify transfer</b>
01	<b>Write transfer (record)</b>
10	<b>Read transfer (playback)</b>
11	<b>Illegal</b>
XX	<b>Ignored if bits 7:6 = 11</b>
Bits 1:0	<b>Channel selection</b>
00	<b>Select channel 0 (4)</b>
01	<b>Select channel 1 (5)</b>
10	<b>Select channel 2 (6)</b>
11	<b>Select channel 3 (7)</b>

#### Write single mask bit assignments

Bits/value	Function
Bits 7:3	<b>Unused (Set to 0)</b>
Bit 2	<b>Set/clear mask bits</b>
1	<b>Set mask bit (Disable channel)</b>
0	<b>Clear mask bit (Enable channel)</b>
Bits 1:0	<b>Channel selection</b>
00	<b>Select channel 0 (4)</b>
01	<b>Select channel 1 (5)</b>
10	<b>Select channel 2 (6)</b>
11	<b>Select channel 3 (7)</b>

DMAC2 is used for 16-bit I/O and DMAC1 is used for 8-bit I/O. The procedure for starting a transfer is complicated, so I'll list the steps for starting the type of DMA transfers used for sound I/O:

1. Calculate the absolute linear address of your buffer  
 $\text{LinearAddr} := \text{Seg}(\text{Ptr}^*) * 16 + \text{Ofs}(\text{Ptr}^*);$
2. Disable the sound card DMA channel by setting the appropriate mask bit  
 $\text{Port}[\text{MaskPort}] := 4 + (\text{Channel mod } 4);$
3. Clear the byte pointer flip-flop  
 $\text{Port}[\text{ClrBytePtr}] := \text{AnyValue};$

4. Write the DMA mode for the transfer

The mode selection bits should be set to 01 for single-mode. The address inc/dec bit should be set to 0 for address increment. The auto-initialization bit should be set appropriately. I will discuss auto-initialized DMA later. The transfer bits should be set to 10 for playback and 01 for recording. The channel select should be set to the sound card DMA channel. Be aware that "read" means a read from memory (Write to sound card) and that "write" means a write to system memory.

Port[ModePort] := Mode + (Channel mod 4);

Some often used modes are:

- **48h+Channel** - Single-cycle playback
- **58h+Channel** - Auto-initialized playback
- **44h+Channel** - Single-cycle record
- **54h+Channel** - Auto-initialized recording

5. Write the offset of the buffer, low byte followed by high byte. For sixteen bit data, the offset should be in words from the start of a 128kbyte page. The easiest method for computing 16-bit parameters is to divide the linear address by two before calculating offset.

if SixteenBit

then

begin

BufOffset := (LinearAddr div 2) mod 65536;

Port[BaseAddrPort] := Lo(BufOffset);

Port[BaseAddrPort] := Hi(BufOffset);

end

else

begin

BufOffset := LinearAddr mod 65536;

Port[BaseAddrPort] := Lo(BufOffset);

Port[BaseAddrPort] := Hi(BufOffset);

end;

6. Write the transfer length, low byte followed by high byte. For an 8-bit transfer, write the number of bytes-1. For a 16-bit transfer, write the number of words-1.

Port[CountPort] := Lo(TransferLength-1);

Port[CountPort] := Hi(TransferLength-1);

7. Write the buffer page to the DMA page register.

Port[PagePort] := LinearAddr div 65536;

8. Enable the sound card DMA channel by clearing the appropriate mask bit

Port[MaskPort] := DMAChannel mod 4;

## Setting the sampling rate

Unlike earlier Sound Blasters, the SB16 is programmed with actual sampling rates instead of time constants. On the SB16, the sampling rate is set using DSP commands 41h and 42h. Command 41h is used for output and 42h is used for input. I have heard that on the SB16, both these command currently do the same thing, but I would recommend using the individual commands to guarantee compatibility with future sound cards. The procedure for setting the sampling rate is:

1. Write the command (41h for output rate, 42h for input rate)
2. Write the high byte of the sampling rate (56h for 22050 hz)
3. Write the low byte of the sampling rate (22h for 22050 hz)

## Digitized sound I/O

To record or play back sound, you should use the following sequence:

1. Allocate a buffer that does not cross a 64k physical page boundary
2. Install an interrupt service routine
3. Program the DMA controller for background transfer
4. Set the sampling rate
5. Write the I/O command to the DSP
6. Write the I/O transfer mode to the DSP
7. Write the block size to the DSP (Low byte/high byte)

Upon interrupt when using single-cycle DMA:

1. Program DMA controller for next block
2. Program DSP for next block

3. Copy next block if double-buffering
4. Acknowledge the interrupt with the SB by reading from port 2xE for 8-bit sound or port 2xF for 16-bit sound.
5. Acknowledge the end of interrupt with the PIC by writing 20h to port 20h. If the sound card is on IRQ8-15, you must also write 20h to A0h.

## DSP commands

**D0**

Pause 8-bit DMA mode digitized sound I/O initiated by command Cxh.  
Applicable to both single-cycle and auto-initialized DMA I/O.

**D4**

Continue 8-bit DMA mode digitized sound I/O paused using command D0.  
Applicable to both single-cycle and auto-initialized DMA I/O.

**D5**

Pause 16-bit DMA mode digitized sound I/O initiated by command Bxh.  
Applicable to both single-cycle and auto-initialized DMA I/O.

**D6**

Continue 16-bit DMA mode digitized sound I/O paused using command D5.  
Applicable to both single-cycle and auto-initialized DMA I/O.

**D9**

Exit 16-bit auto-initialized DMA mode digitized sound I/O after the end of the current block.

**DA**

Exit 8-bit auto-initialized DMA mode digitized sound I/O after the end of the current block.

**E1**

Get DSP version number. After sending this command, read back two bytes from the DSP. The first byte is the major version number and the second byte is the minor version number. A SB16 should have a DSP version of 4.00 or greater. Check this before using an SB16 specific commands.

**Bx**

Program 16-bit DMA mode digitized sound I/O

Command sequence: Command, Mode, Lo(Length-1), Hi(Length-1)

**Command:**

7	6	5	4	3	2	1	0
1	0	1	1	A/D	A/I	FIFO	0
				0=D→A	0=SC	0=off	
				1=A→D	1=A/I	1=on	

**Common commands:**

- **B8** - 16-bit single-cycle input
- **B0** - 16-bit single-cycle output
- **BE** - 16-bit auto-initialized input
- **B6** - 16-bit auto-initialized output

**Mode:**

7	6	5	4	3	2	1	0
0	0	Stereo	Signed	0	0	0	0
		0 = mono	0 = unsigned				
		1 = stereo	1 = signed				

**Cx**

Program 8-bit DMA mode digitized sound I/O

Same procedure as 16-bit sound I/O using command Bx

**Common commands:**

- **C8** - 8-bit single-cycle input
- **C0** - 8-bit single-cycle output
- **CE** - 8-bit auto-initialized input
- **C6** - 8-bit auto-initialized output

The FIFO is used to eliminate inconsistencies in the sample period when the sound card is not able to get DMA when it needs it. With FIFO disabled, the card attempts DMA at precisely the instant that it

needs a sample. If another device with a higher priority is accessing DMA, the sound card waits for the sample and the sampling rate may be decreased. The FIFO allows the DMA sample period to be more erratic without affecting the audio quality. The FIFO is cleared whenever a command is sent to the DSP. In Single-cycle mode, the DSP is constantly being reprogrammed. The FIFO may still contain data which has not been output when it cleared. To avoid this problem, the FIFO should be turned off for single-cycle mode. When in auto-initialized mode, the DSP is never reprogrammed. The FIFO can be left on and sound quality will be improved.

## Auto-initialized DMA

When single-cycle DMA is used, sound output stops at the end of each block. The interrupt handler can start another transfer, but there will be a break in output. This causes a click between each block, reducing sound quality. When auto-initialized DMA is used, sound output loops around at the end of the buffer. The DMA controller keeps transferring the same block of memory that the DMA transfer was initiated with. When the end of the buffer is reached, it will start sending the buffer again by auto-initializing the current offset and count registers with the values stored in the base offset and count registers. The usual method for achieving click-less output is to allocate a buffer and divide it into two blocks. Program the DMA controller with the length of the whole buffer, but program the SB16 with the length of a block. (Half of the buffer) An interrupt occurs for each sound card block, so two interrupts will occur each time the buffer is played, once at the midpoint (Start of the second block) and once at the end (In effect, the start of the first block) The interrupt handler should copy data into the block that was just finished so that the data is ready when it is needed for output. The programming procedure for an auto-initialized DMA transfer is identical to the procedure for a single-cycle DMA transfer, except that bit 4 of the DMA mode register and bit 3 of the DSP command are set.

### Upon interrupt when using auto-initialized DMA:

1. Copy next chunk into output buffer block that just finished
2. Acknowledge the interrupt with the SB by reading from port 2xE for 8-bit sound or port 2xF for 16-bit sound.
3. Acknowledge the end of interrupt with the PIC by writing 20h to port 20h. If the sound card is on IRQ8-15, you must also write 20h to A0h.

### To stop sound immediately:

- **8-bit** - Write DSP command D0h (Pause 8-bit DMA mode digitized sound I/O)
- **16-bit** - Write DSP command D5h (Pause 16-bit DMA mode digitized sound I/O)

Both commands stop sound immediately, without an interrupt.

### To stop the sound at the end of the currently block:

- **8-bit** - Write DSP command DAh (Stop 8-bit auto-init DMA sound I/O)
- **16-bit** - Write DSP command D9h (Stop 16-bit auto-init DMA sound I/O)

These two commands will stop the sound at the end of the current block. If your program is not prepared for an interrupt after output is finished, it may cause problems.

You can also end auto-initialized mode by reprogramming the DSP for single-cycle mode. The card then switches from A/I mode to S/C mode after the next interrupt. It will then continue to play or record for the length specified, generate an interrupt and stop. This will allow you to stop output exactly at the end of the data, without requiring the remainder of the DMA buffer to be filled with silence. This technique may or may not be useful to you. I would recommend using the pause commands documented in the immediate stop section unless another method is more suited to your purpose.

## References

- Title - [Developer Kit for the Sound Blaster Series, Second Edition.](#)
- Publisher - [Creative Technology Ltd.](#)
- Title 486 Microcomputer Model 401 Board Technical Reference Manual
- Publisher - [Intel Corporation](#)
- Order-number - 504366-002
- Title - 8237A High Performance Programmable DMA Controller specs
- Publisher - [Intel Corporation](#)
- Order-number - 231466-005
- Title - 8259A Programmable Interrupt Controller specs
- Publisher - [Intel Corporation](#)
- Order-number - 231468-003
- Title - [SMIX Sound System](#)

- **Author** - Ethan Brodsky
- **Title** - [SB16SND Sound Code](#)
- **Author** - Ethan Brodsky

Thanks to **Douglas Kaden** at [Creative Labs](#) for information on 16-bit DMA, FIFO mode, and other topics.

[Back to Home Page](#)

[View SMIX information](#)

[View SB16SND information](#)

*Ethan Brodsky <brodskye@cae.wisc.edu>*

Pegado de <<http://homepages.cae.wisc.edu/~brodskye/sb16doc/sb16doc.html>>

23:24

# Pokemon Red

sábado, 17 de mayo de 2014  
20:23

Offset 0x01AD -> Called when an attack move is finish. Instruction RET.

# Pokemon Emerald

domingo, 27 de julio de 2014  
14:44

Routine offsets

0x8272083 - Used for healing (the player's mom use this)  
Cmdc3 0x10 - Something related to healing too (The player's mom use this)

FLAGS

0x85 - Badge

# Third gen scripts examples

domingo, 27 de julio de 2014

15:09

This is a tutorial thread for changing Scripts in Pokemon GBA games. If you would like anything added or if you need help please post it in this thread or pm me.

Current mods of this forum, if I ever become inactive you may edit this thread, but only edit it if you are sure the information is correct.

Please don't post in this thread. Post all of your questions [here](#).

Table of Contents: To jump to a section press ctr-f and type in where you want to go to.

- A. Script Editors [AS]
- B. Script Editor and Advance Map [BS]
- C. Editing Scripts [CE]
- D. Inserting Scripts [DI]
- E. Trainer Battles [EA]
- F. Flags [FF]
- G. Map Scripts [GS]
- H. Special Scripts [HS]

Before I begin scripts are everything that happens in the game. Cut Scenes, trainer battles, and talking are all scripts. You will probably need to know about scripts if you want to make a good hack.

## A. Script Editors [AS]

You will need advance map 1.92 NOT 1.95. If you didn't download it already you can download it here:

<http://filetrip.net/gba-downloads/tools-utilities/download-advance-map-192-f7463.html>

We will use advance map to insert the scripts into the game.

Now you have a choice between PKSV and XSE. I suggest using pksv. I have experience with it so I know it's easy to use, it has a script generator to make scripts for you (saves hours of work), and I'll be using it for this tutorial.

I'm letting you know of xse just in case pksv doesn't work for you you'll have another option. Xse doesn't have a script generator, but I heard it worked very well. None of the scripts in this thread will work for xse so you're on your own if you want to use this. (so you can only use sections B, C and D if you want to use xse)

Both of these programs have the same capabilities. It's just what you prefer.

PKSV: <http://www.pokecommunity.com/showthread.php?t=116370>

XSE: <http://www.mediafire.com/?mss33go3j3p032h>

## B. Script Editor and Advance Map [BS]

You'll need advance map and pksv to do this. (you can use xse if you want, but I'm using pksv for the tutorial)

You'll also need winRAR to extract these files. (link not provided)

Once you have everything downloaded go ahead and make a new folder on your desktop. Then extract all of the tiles from the pksv download into this folder. (make sure every single file is together in this folder. pksv will not work if they aren't) If you want you can also extract advance map to make things easier. Once everything is in the right folder open up advance map. Then go to the settings tab and click "Choose Script Editor". Then choose pksv and a box will pop up. Select "No" in the box. Now Advance map will use pksv open your rom in advance map. Now you're ready to edit scripts



## C. Editing Scripts [CE]?

To edit a script that is already in the game you have to go to the map you want to edit then go to the events tab in advance map. Then you click on what you want to edit (press ctr z and it will show the sprites. this makes it easier to tell what does what) and press "open script". Then pksv should pop up. Now you edit the script to whatever you want. (you can go to tools then script editor and it will make scripts for you). After you made the script go to the rom tab in pksv then press compile. Now open up your emulator and test it out!

If your changes don't work them post your question in the thread linked above or pm me.

Also if you are making a talking script you will need to know this.

\n Goes to a new line when the player presses A. If you don't add this then the text will overlap the box.

\p Makes a red arrow in the corner and then when the player presses A it will skip two lines.

\l This is the same thing as \p, but it just skips two lines and doesn't make an arrow.

\e Makes the e in Pokemon.

// Makes it so the rest of the line doesn't show up in the game. So you can make it say "Hi!\\This will make the person say hi." This will make it easy to keep notes when you are making big scripts.

\h01 Makes it say the player's name.  
\h06 Makes it say your rival's name.

#### D. Inserting Scripts [DI]

This is the same as editing scripts, but with a few extra steps. So open up advance map, go to the map you want to edit, and then you have to add an event. You can add events in the box in the bottom right corner. Now you need to know what type of script it will be. Do you want a person talking, a signpost, a warp or an event script. (a warp takes you to another map, like a door) Once you've chosen what you want to add click on the up arrow next to one of them then click change events. The new event will be in the top left corner. Click and drag it to wherever you want it to go.

Now you have to open up pksv. Then make the script you want and compile it. Now two boxes will pop up. Go to the one that says dynamic offsets. Now on the top row copy the offset. (the numbers after where it says "0x"  
<http://s1.postimg.org/90mpic37j/Captureoffset.png>

Now paste the offset in the box that says "Script" Offset" Feel free to change the picture number and movement type. You'll want to experiment with this to get the right one.

Now I'm going to explain what a basic talking script looks like and the components of a script. If this is the first script you've ever seen don't worry. PKSV and XSE make the scripts look a lot nicer.

```
#dyn 0x740000 //This is how you start every script. This finds free space for the script to go.  
#org @start //You'll always need this. This will tell the game where to start the script. You can change "Start" with any word. Just don't use the same word twice.  
lock //This locks the player so they can't move.  
faceplayer //Makes the person face the player.  
msgbox @text ' Hi //This makes a text box and tells the script to look for "@text". @text will be replaced by something else when it is compiled into the rom.  
callstd MSG_NORMAL //This tells the game that this is a normal talking script.  
release //releases the player  
end //ends the script
```

```
#org @text  
= Hi
```

#### E. Trainer Battles [EA]

This part will show you how to make trainer battle scripts. If you want to edit what pokemon the trainer has use Advance Trainer or Trainer Editor.

```
#dyn 0x740000  
#org @begin  
trainerbattle 0x0 0x001 0x0 @intro @defeat //This is the trainer battle command. The second 0x is the trainer number. You can find the trainer number in Advance Trainer, Trainer Editor or PET.  
msgbox @afterwards  
callstd msg_normal  
end  
#org @intro  
= This is what the trainer says\nbefore the battle.  
#org @defeat  
= This is what the trainer says\nwhen he loses.  
#org @afterwards  
= This is what the trainer says\nafter the battle.
```

When you put this in advance map you need to check the trainer battle box and change the trainer's view radius. If you want to make a more complicated trainer battle then you should use a script tile and not a person event. To do this make the rest of the script and put the trainer battle command somewhere.

#### F. Flags [FF]

Flags make it so things happen once or something happens after something else. It's best to use flag 200- 2FF. Some of them are already used in the game so I recommend you test them out. If you want to add a flag then you'll need to add "checkflag 0x" and "setflag 0x" to your script.

```
#dyn 0x740000  
#org @main  
lock  
faceplayer  
checkflag 0x200 //This checks to see if the flag has been set already.  
if == jump @haveitem //This tells the game where to go if the flag was set.
```

```

message @talk
callstd MSG_YESNO
compare LASTRESULT YES
if == jump @pushedyes
jump @pushedno

#org @haveitem
message @howsit
callstd MSG_NORMAL
release
end

#org @pushedyes
setflag 0x200
additem MASTERBALL 1
message @give
callstd MSG_NORMAL
release
end

#org @pushedno
message @dont
callstd MSG_NORMAL
release
end

#org @talk
= I have a Masterball. Do you want it?
#org @howsit
= Catch legendary pokemon!
#org @give
= I'm only giving you one.
#org @dont
= You don't?
If you want something to happen after something else happens then you'll need to use compare var. More info on
that in the special scripts section.

```

#### G. Map Scripts [GM]

Map scripts are scripts that happen when you enter the map. It's hard for me to explain so just watch these.

<http://www.youtube.com/watch?v=iE42FptSCJM>  
[http://www.youtube.com/watch?v=Q0a\\_J7beki4](http://www.youtube.com/watch?v=Q0a_J7beki4)

#### H. Special Scripts [HS]

This section will show you what some scripts look like. These scripts are not available in pksv's script generator or are made wrong in the generator.

No I will not put every type of script here. If you want a script just go into pksv and use the script generator or go into advance map and open the script you want. If you can't find a script then post it in the thread linked above and I will make it for you. Now this isn't an excuse to request me to make a whole game for you. These will just be templates.

#### Yes or No questions.

This is a Yes or No question script. You can edit it however you want. This example is from the Pewter City Bug Kid:

```

#dyn 0x740000
#org @main
lock
faceplayer
message @First
callstd MSG_YESNO //This makes it a yes or no question.
compare LASTRESULT YES //The script generator in pksv doesn't add this so be sure to add it.
if == jump @pushedyes //The jump command makes it go to @pushedyes then come back.
jump @pushedno
#org @pushedyes
message @Yes
callstd MSG_NORMAL
release
end
#org @pushedno
message @No

```

```

callstd MSG_NORMAL
release
end
#org @First
= Do you want an item?
#org @Yes
= Great!
#org @No
= Sorry.

```

#### Movement

Movement scripts tell a sprite to move. Like when Professor Oak takes you into his lab. This is a basic movement script. You can edit it to what you want.

```

#dyn 0x740000
#org @main
lock
faceplayer
message @First
callstd MSG_NORMAL
compare playerfacing up //These lines will compare the way the player is facing. So if you are facing a certain direction then it will go to where you tell it to go in the next line.
if == call @up //The call command makes it go to that spot then come back. So if the player is facing up it will go to @up then it will come back and finish the rest of the script.
compare playerfacing down
if == call @down
compare playerfacing left
if == call @left
compare playerfacing right
if == call @right
message @Last
callstd MSG_NOCLOSE //This will make it so the message wont close until the close message command is activated.
pause 0xA
closemsg
applymovement 0x1 @walkaway
applymovement PLAYER @seeaway
pauseevent 0x0
fadedefault //This will make it go back to how it was before.
disappear 0x1 //This will remove the sprite so you can't talk to it again. Change the number to whatever the sprite number is in advance map.
setflag 0x200 //This will make the event not happen again.
release
end

#org @up
applymovement 0x1 @leadup //This will make sprite number one move.
applymovement PLAYER @followup //This will make the player move.
pauseevent 0x0 //This will pause the game.
return //This will make the script return to where you jumped.
#org @down
applymovement 0x1 @leaddown
applymovement PLAYER @followdown
pauseevent 0x0
return
#org @left
applymovement 0x1 @leadleft
applymovement PLAYER @followleft
pauseevent 0x0
return
#org @right
applymovement 0x1 @leadright
applymovement PLAYER @followright
pauseevent 0x0
return
#org @First
= Follow me!
#org @Last
= Bye!

#org @walkaway
m walk_down end //Make sure you use "M"s and not "="s. Be sure to use the movement helper in pksv. It will make this faster.

```

```

#org @seeaway
m look_down_delayed
#org @leadup
m walk_right end
#org @followup
m walk_up end
#org @leaddown
m walk_right end
#org @followdown
m walk_down end
#org @leadleft
m walk_down end
#org @followleft
m look_down end

```

#### Camera Movement

This moves the camera around so you can see the map. I don't think there is an example in the game, but you can try this out.

```

#dyn 0x740000
#org @Main
lock
faceplayer
message @First
callstd MSG_YESNO
compare LASTRESULT YES
if == jump @pushedyes
jump @pushedno
#org @pushedyes
message @Yes
callstd MSG_NORMAL
release
end
#org @pushedno
message @No
callstd MSG_NORMAI
special CAMERA_START //this will activate the camera movement
applymovement CAMERA @Show
pauseevent 0x0 //This will pause for a little.
message @Done
callstd MSG_NOCLOSE //This will make it so you can't close the window.
pause 0xA
closemsg
applymovement CAMERA @return
special CAMERA_END
release
end
#org @First
= Do you want me to move the camera \nto the right?
#org @Yes
= My powers aren't working.
#org @No
= I'll do it anyway!!!
#org @Done
= Yay it worked!
#org @Show
M walk_right end //You can change this to whatever you want. You can also add more of these. pksv comes with
a "Movement Helper" and it will make all of the movements for you. Don't make it go out of the location because
it will glitch the game.
#org @return
M walk_left end

```

#### Camera Movement (2)

This second script will make the camera move only once and it will be a tile that you step on. It will also make the camera move after you stepped on a different tile. Your's doesn't have to have text, but this is just an example/. You'll also need advance map with this. Var 4011- 40FF are safe to use. Some of them are already used in the game so I recommend you test it out first.

```

#dyn 0x740000
#org @main
lock
faceplayer
message @First
callstd MSG_NORMAL
setvar 0x4011 0x1 //The first number is the var value and the second one is the var number. These are similar to

```

flags. You can't use the same var number unless one var triggers another. If one var triggers another they will have different var values.

```
release
end
#org @First
= Keep walking and i'll show you \n your house.
```

Now here is a second script. This will happen after you step on the first tile. You can just use one tile that lets you see, but this gives you more options.

```
#dyn 0X740000
#org @main
compare 0x6001 0x1 //This will make it so if you didn't step on the first square the second one wont work.
if == jump @First
release
end
#org @First
message @talk
callstd MSG_NORMAL
special CAMERA_START
applymovement CAMERA @show
pauseevent 0x0
message @done
callstd MSG_NOCLOSE
pause 0xA
closemsg
applymovement CAMERA @return
pauseevent 0x0
special CAMERA_END
setvar 0x4011 0x2 //This is 0x2 because it's the second event. You can add more if necessary.
release
end
#org @talk
= Wanna see your house?
#org @done
= There it is.
#org @show
M walk_left walk_left walk_left walk_left walk_left walk_left walk_left end
#org @return
M walk_right walk_right walk_right walk_right walk_right walk_right walk_right end
```

Now once you've compiled these into advance map you'll have to add a green script box. Go to the events tab then add two "Script"s. Now paste in the script offsets in both of the scripts. Here is where it starts getting tricky. For all of the green script boxes set the unknown to "0003" (make sure it's the first unknown box. keep the second one 00 00) and set the var number to the number you used before. So for this example it will be "6001". And finally for the var value put the one you want the player to step on first as "0000", second as "0001" and so on if you have more boxes.

### Compare Var

This is like flags, but it will use green script boxes instead. You might want to watch this video.

<http://www.youtube.com/watch?v=cRvkOcSm0Ts>

This is really hard for me to explain in words. Also you should look through the camera movements first because it says things about vars.

### Battle Cry

This script will play a Pokemon's battle cry. Like when you talk to pokemon or when you battle them.

```
#dyn 0x740000
#org @main
lock
faceplayer
cry PIKACHU 0x2 //You need to keep the 0x2, but you can change the pokemon.
waitcry //This will pause the game so the cry will play. This is optional.
release //You can also add a 'waitcry' command to delay the cry sound.
end
```

### Change Music

This will change the music of the map. The music will keep playing until you go to another area or enter a house. You can find a complete list of sounds in advance map's header tab.

```
#dyn 0x740000
```

```

#org @main
lock
faceplayer
playsound 0x12C 0x0 //You can change the 0x12C to whatever you want. This will play Pallet Town's
music.
pause 0xA //This will pause so you can hear the music.
fadedefault //This will turn the music back to the default music.
message @talk
callstd MSG_NOCLOSE //No matter what, message will not close until there is a 'closemsg'
playsound 0x123 0x0 //This will play Route 1's Music.
closemsg //When 'A' is pressed, the message will close.
release
end

```

So this script will play Pallet Town for a little then it will change it to Route 1 until you enter another map.

### Sounds

```

This will play a sound.
#dyn 0x740000
#org @main
lock
faceplayer
sound 0x1 //You can change this to whatever you want.
pause 0x1 //You can change this to whatever you want
message @talk
callstd MSG_NORMAL
release
end
#org @talk
= I play sound a lot.

```

### Random Events

```

This is a really cool feature. This lets the game generate a random number then the game will do something
depending on the number.
#dyn 0x740000
#org @Main
lock
faceplayer
random 0x5 //This makes it so the game will pick a number between 0 and 5. The possible numbers will be 0, 1,
2, 3, or 4.
compare LASTRESULT 3 //This will make the game look at then number picked then determine what to do.
if == jump @Good //This means if it is 3 then you get the item.
jump @Bad //This means if it isn't 3 you don't get an item.
#org @Good
message @GoodMSG
callstd MSG_LOCK
additem MASTERBALL 1
release
end
#org @Bad
message @BadMSG
callstd MSG_LOCK
additem POTION 1
release
end
#org @GoodMSG
= Here, take this materball.
#org @BadMSG
= Here, take this potion.

```

### Fadescreen

```

Fadescreens are useful when you want to replace tile sets or remove sprites. This script is a basic fadescreen
script.
#dyn 0x740000
#org 0x8740001
lock
faceplayer
message @First

```

```

callstd MSG_LOCK
fadescreen FADEOUT_BLACK //This will turn the screen black. Remember to put this and the next line in the
right order.
fadescreen FADEIN_BLACK //This will turn the screen back to normal.
release
end
#org @First
= Cool Right?

```

### Replacing Tiles

This is easy, but learning may be a little difficult. This is useful if you want to change map tiles.

```

#dyn 0x740000
#org @main
lock
faceplayer
message @talk
callstd MSG_YESNO
compare LASTRESULT YES
if == jump @yes
jump @no

#org @yes
message @speak
callstd MSG_NORMAL
fadescreen FADEOUT_BLACK //This will turn the screen black.
setmaptile 0x1 0x1 0x1 0x0 //The first one is the X coordinate, the second is the Y coordinate, the third is the tile,
and the fourth is the movement permission. Feel free to change these.
special 0x8E //I'm not sure what this does, but you need to keep this in.
fadescreen FADEIN_BLACK
message @done
callstd MSG_LOCK
fadescreen FADEOUT_BLACK
disappear 0x1 //This makes the sprite disappear so the event doesn't happen twice. Change the number to
whatever sprite you want to disappear.
setflag 0x250
fadescreen FADEIN_BLACK
release
end

#org @no
message @dontspeak
callstd MSG_NOCLOSE
closemsg
release
end

#org @first
= I'll destroy the block for you!
#org @speak
= HRRRR
#org @done
= Your welcome.
#org @dontspeak
= Sob.

#org @First
= Do you want me to remove\nthe annoying block?
#org @speak
= Okay!\nWatch the amazing power\lof setmaptile scripts!
#org @done
= See how cool it is?\nOkay, I'm done.\nSee ya later!
#org @dontspeak
= All right, your loss.\nThe result is amazing!

```

### Money

Also make people give you money and pay other people money. Here is an example.

```

#dyn 0x740000
#org @Main
lock
faceplayer

```

```
showmoney 0x0 0x0 // The first value is the X coordinate and the second is the Y coordinate. This will show where the money box is located on the screen. You can replace this with "Showpokepic" and it will show a pokemon. "Hidepokepic" will make it go away.  
message @First  
callstd MSG_Normal  
givemoney 1000 0 //You can replace give money with paymoney. Paymoney will make the player lose money.  
This givemoney command makes the script give the player money.  
pause 0xF //This will make a pause  
updateMoney 0 0 0 //This updates the box so it shows the right amount of money. The first two values are the coordinates and the third one is nothing.  
message @Second  
callstd MSG_Normal  
hideMoney 0 0  
release  
end  
#org @First  
= Here.  
#org @Second  
= Don't spend it all in one place!
```

Pegado de <<http://www.vizzed.com/boards/thread.php?id=60655>>

# Emerald Ninjihaku Hack

domingo, 27 de julio de 2014  
21:38

## FLAGS

0x  
200 - HELIOS BATTLE  
201 -  
202 -  
203 -  
204 -  
205 -

## OFFSETS

eXtreme Script Editor outputs:

eXtreme Script Editor v1.1.1

-----  
07-28-2014 2:16:33

HELIOS SCRIPT

-----  
Opening output: C:\Emulators\VBA\GBA Roms\Pokemon Emerald NinjihakuHack.gba...

Processing input script...

1 - #DYNAMIC  
> lDynamicStart = 0xC3500  
3 - #ORG  
> lNewOffset = 0x223D22  
4 - (2B) - CHECKFLAG [+3]  
> iWord = 0x200  
5 - (??) - IF (native) [+6]  
> bCondition = 0x1  
This is a jumping IF, 0x06.  
> pTarget = 0x82E9837  
6 - (??) - IF (native) [+6]  
> bCondition = 0x0  
This is a jumping IF, 0x06.  
> pTarget = 0x82E9803  
8 - #ORG  
> lNewOffset = 0x2E9803  
9 - (6A) - LOCK [+1]  
10 - (5A) - FACEPLAYER [+1]  
11 - (0F) - MSGBOX (native) [+8]  
> pText = 0x82E9856  
> bType = 0x6  
12 - (0F) - MSGBOX (native) [+8]  
> pText = 0x82E9889  
> bType = 0x6  
13 - (0F) - MSGBOX (native) [+8]

```
> pText = 0x82E98C7
> bType = 0x6
14 - (0F) - MSGBOX (native) [+8]
> pText = 0x82E98FB
> bType = 0x5
15 - (21) - COMPARE [+5]
> iWord = 0x800D
> iWord = 0x1
16 - (??) - IF (native) [+6]
> bCondition = 0x1
This is a jumping IF, 0x06.
> pTarget = 0x82E992A
17 - (??) - IF (native) [+6]
> bCondition = 0x0
This is a jumping IF, 0x06.
> pTarget = 0x82E9953
19 - #ORG
> lNewOffset = 0x2E9837
20 - (6A) - LOCK [+1]
21 - (5A) - FACEPLAYER [+1]
22 - (0F) - MSGBOX (native) [+8]
> pText = 0x82E99F7
> bType = 0x6
23 - (6C) - RELEASE [+1]
24 - (02) - END [+1]
26 - #ORG
> lNewOffset = 0x2E9844
27 - RAW TEXT [+16]
> sText = "Hello there \v\h01!"
29 - #ORG
> lNewOffset = 0x2E9856
30 - RAW TEXT [+49]
> sText = "My name is Helios.\nI'm here to test your skills."
32 - #ORG
> lNewOffset = 0x2E9889
33 - RAW TEXT [+60]
> sText = "If you win me in a battle, I shall\ngrant you a MASTER BALL!"
35 - #ORG
> lNewOffset = 0x2E98C7
36 - RAW TEXT [+50]
> sText = "But I have to warn you, I'm\nnot a common trainer!"
38 - #ORG
> lNewOffset = 0x2E98FB
39 - RAW TEXT [+45]
> sText = "So tell me, are you willing\nnot to challenge me?"
41 - #ORG
> lNewOffset = 0x2E992A
42 - (5C) - TRAINERBATTLE [+14]
> bKind = 0x0
> iIndex = 0x357
> iReserved = 0x0
> pChallenge = 0x82E998B
> pDefeat = 0x82E99EA
43 - (0F) - MSGBOX (native) [+8]
> pText = 0x82E99B4
> bType = 0x6
44 - (44) - ADDITEM [+5]
```

```
> iWord = 0x1
> iWord = 0x1
45 - (29) - SETFLAG [+3]
> iWord = 0x200
46 - (0F) - MSGBOX (native) [+8]
> pText = 0x82E99F7
> bType = 0x6
47 - (6C) - RELEASE [+1]
48 - (02) - END [+1]
50 - #ORG
> lNewOffset = 0x2E9953
51 - (0F) - MSGBOX (native) [+8]
> pText = 0x82E995E
> bType = 0x6
52 - (6C) - RELEASE [+1]
53 - (02) - END [+1]
55 - #ORG
> lNewOffset = 0x2E995E
56 - RAW TEXT [+43]
> sText = "Fine. Come see me\nwhenever you feel ready."
58 - #ORG
> lNewOffset = 0x2E998B
59 - RAW TEXT [+39]
> sText = "Very well then.\nLet the battle beggin!"
61 - #ORG
> lNewOffset = 0x2E99B4
62 - RAW TEXT [+52]
> sText = "You won me. Now I shall grant\nyou this MASTER BALL."
64 - #ORG
> lNewOffset = 0x2E99EA
65 - RAW TEXT [+11]
> sText = "Well done!"
67 - #ORG
> lNewOffset = 0x2E99F7
68 - RAW TEXT [+66]
> sText = "There are more challengers\nwaiting for you.\nGo and look for them!"
```

---

DYNAMIC\_OFFSET 1

```
> sLabel = @main
> lOffset = 0x223D22
```

DYNAMIC\_OFFSET 2

```
> sLabel = @battle
> lOffset = 0x2E9803
```

DYNAMIC\_OFFSET 3

```
> sLabel = @done
> lOffset = 0x2E9837
```

DYNAMIC\_OFFSET 4

```
> sLabel = @present
> lOffset = 0x2E9844
```

DYNAMIC\_OFFSET 5

```
> sLabel = @dialog1
> lOffset = 0x2E9856
```

DYNAMIC\_OFFSET 6

```
> sLabel = @dialog2
> lOffset = 0x2E9889
```

DYNAMIC\_OFFSET 7

```
> sLabel = @dialog3
```

```
> lOffset = 0x2E98C7
DYNAMIC_OFFSET 8
> sLabel = @dialog4
> lOffset = 0x2E98FB
DYNAMIC_OFFSET 9
> sLabel = @yes
> lOffset = 0x2E992A
DYNAMIC_OFFSET 10
> sLabel = @no
> lOffset = 0x2E9953
DYNAMIC_OFFSET 11
> sLabel = @nomsg
> lOffset = 0x2E995E
DYNAMIC_OFFSET 12
> sLabel = @before
> lOffset = 0x2E998B
DYNAMIC_OFFSET 13
> sLabel = @win
> lOffset = 0x2E99B4
DYNAMIC_OFFSET 14
> sLabel = @defeat
> lOffset = 0x2E99EA
DYNAMIC_OFFSET 15
> sLabel = @over
> lOffset = 0x2E99F7
```

---

Cleaning up...

Closing output...

Finished processing input in 0.027 seconds.

# XSE Special commands

domingo, 27 de julio de 2014  
22:00

special 0x0 - Heal Pokemon  
special 0x3C - Access Bill's PC (FR/LG)  
special 0x98 - Going up to Mountain (R/S)  
special 0x9C - Wally Catch (R/S)  
special 0x9F - choose A Pokemon (R/S)  
special 0xE0 - PokeBlock Case (R/S)  
special 0x10F - Restart Game  
special 0x110 - Hall of Fame and Credits  
special 0x111 - Elevator Animation  
special 0x119 - Groudon's Orb effect (R/S)  
special 0x131 - Earthquake (R/S)  
special 0x132 - Show Floors  
special 0x136 - Earthquake (FR/LG)  
special 0x137 - Lava Battle  
special 0x156 - Battle with Ghost (FR/LG)  
special 0x157 - Get on Bike (FR/LG)  
special 0x161 - Start Surfing (FR/LG)  
special 0x166 - Nickname  
special 0x16F - Activate National Dex (FR/LG)  
special 0x17B - Seagallop Animation  
special 0x191 - SS. Anne Leaving  
special 0x1F3 - Activate National Dex (Emerald)

Pegado de <<http://www.shaychu.com/t1054-cirrus-ridiculously-long-xse-scripting-tutorial-not-finished-atm-oo>>

## Ninji's Chan Front Page

sábado, 12 de julio de 2014  
3:30

### /main/ - Ninjihaku's Imageboard

Chill place

Name	Dogoo
Email	noko
Subject	<input type="text"/> New Topic <input checked="" type="checkbox"/> Spoiler Image
Comment	    
File	<input type="button" value="Examinar..."/> No se ha seleccionado ningún archivo.
Password	***** (For file deletion.)

Before anything, [click on this link and read it very carefully](#). This is just an experimental imageboard. Troll content or anything that is against the law or the internal rules will be immediately deleted. We also record the IP addresses of every participant, and store cookies to identify you. If you don't agree with this, please stay away from this place. Otherwise, just chill and take it easy, and everything will be cool :)

File: [1404687739572.png](#) (463.63 KB, 665x374, pic.png)



■ **Noire## ModIIZA5WYXOLRI** 07/06/14 (Sun) 23:02:19 No. 5  [Reply]

H-Hello everyone.

I am the CPU Noire, a-a-and I hate every single one of you! All of you are fat, retarded, no-lifes who spend every second of their day looking at stupid ass manga drawings. Y-You are everything bad in the world! Honestly... have any of you ever gotten any shares? I mean, I guess it's fun making fun of people because of your own insecurities, but you all take to a whole new level. This is even worse than jerking off to drawings on Pixiv!

D-Don't make the wrong ideal I-I am not telling this for your own good or anything! Just hit me with your best shot. I-I'm pretty much perfect! I was a top Idol, and was awarded with many cosplay awards. What hobbies do you practice, other than "jack off to naked drawn Japanese people"? I also have my own nation, Lastation, and have a little sister (She just transform into a CPU for her first time; Shit was SO cash). Y-Y-You are all faggots who should just kill yourselves! Thanks for listening.

Pic Related: It's me.

7 posts and 5 image replies omitted. Click reply to view.

■ **Van Darkholme** 07/07/14 (Mon) 00:15:39 No.15

File: [1404692139617.jpg](#) (21.82 KB, 480x360, hqdefault.jpg)



Hello guys.

We're looking for xlescu. We heard he's here. If you find him, please tell him he has to come back and work with us again. We loved his performance in our last movie.

See ya ♥

File: [1404692687635.jpg](#) (165.3 KB, 540x392, grass.jpg)



■ **Anonymous** 07/07/14 (Mon) 00:24:47 No. 16 [Reply]

<https://www.youtube.com/watch?v=nbzoJ98LeiA>

■ **Anonymous** 07/10/14 (Thu) 03:23:29 No.24

what happen  
did we go too fast?

File: [1404852803332.jpg](#) (28.3 KB, 402x304, 1404773044846.jpg)



post vehicles and discuss them here!

■ 07.08.14 (Tue) 21:34:46 No. 23

I wanna post my airplane papercrafts

File: 1404852489585.jpg (265.46 KB, 828x1384, 1372705713137.jpg)



■ Oh Look, Lolis Honk 07/08/14 (Tue) 20:48:09 No. 18 [Reply]

## **It begins. The Great Thread of our Time**

■ Honk 07/08/14 (Tue) 20:49:50 No.19

File: [1404852590553.png](#) (12.6 KB, 400x400, Francesca-Lucchini-strike-wit...)



>>18

■ **Honk** 07/08/14 (Tue) 20:51:58 No.20

File: [1404852718820.jpg](#) (554.75 KB, 1348x892, 1373743803656.jpg)



>>19

TWENTY GET

■ **Ninjihaku** 07/08/14 (Tue) 20:52:07 No.21

File: [1404852727743.jpg](#) (339.72 KB, 674x760, 43692434\_big\_p3.jpg)



'kay

File: [1404756691564.jpg](#) (634.58 KB, 805x1094, samoyedterasu.jpg)

■ **Anonymous** 07/07/14 (Mon) 18:11:31 No. 17 [Reply]

ohayou



File: 1404683472239.jpg (23.59 KB, 460x309, a0qMXG\_L\_460s\_v2.jpg)

## IF YOU REMEMBER THIS...



YOUR CHILDHOOD WAS AWESOME :)

■ **Anonymous** 07/06/14 (Sun) 21:51:12 No. 1 [Reply]

ITT Childhood things

1 post omitted. Click reply to view.

■ **Anonymous** 07/06/14 (Sun) 22:03:25 No. 4

File: 1404684197979.jpg (9.04 KB, 236x338, handheld.jpg)



This was my first handheld console.

20+ years later and i still never beat it

■ **Anonymous** 07/06/14 (Sun) 23:13:01 No. 9

ah, CD players  
I remember them

■ **Ninjihaku!2ggIpGcHvk** 07/06/14 (Sun) 23:33:28 No. 11

File: 1404689608462.jpg (30.5 KB, 518x583, 2013-new-puzzle-game-tetris-ga...)

I remember playing with those things.



File: 1404683728616.gif (66.51 KB, 450x326, pedobear.gif.gif)



■ **Pedo Stuff Xlessian** 07/06/14 (Sun) 21:55:28 No. 2 [Reply]

Ninji PEDO!

USER WAS BAN-HAMMERED IN THE HEAD FOR THIS POST. ALSO, XLESCU PEDO

Delete Post [  File] Password  Delete

Reason  Report

Previous [1] Next

Powered by [Tinyboard](#) v0.9.6-dev-22 | [Tinyboard](#) Copyright © 2010-2014 Tinyboard Development Group  
All trademarks, copyrights, comments, and images on this page are owned by and are the responsibility of their respective parties.

[Yotsuba B] [Yotsuba]

Recorte de pantalla realizado: 12/07/2014 3:31

# Logic Test

sábado, 12 de julio de 2014  
2:18

```
// By Ninjihaku
//

#include "stdafx.h" //Standard Visual Studio Header. Includes stdio.h and tchar.h.
#include <math.h> //Math library required for POW function.

//Program entry point (use main if you're not in VS).
//argv can be replaced by a char pointer (char*), if tchar.h is not included.
//Not required, though.

int _tmain(int argc, _TCHAR* argv[])
{
    //Inputs: a, b, c
    //Outputs: s1
    //LOGIC FUNCTION: c*b + a * (b*c)
    //When at least two of these 3 booleans are TRUE, s1 is also TRUE.

    bool a = false, b = false, c = false; //Define inputs
    bool s1 = false; //Define outputs
    int inputs = 3; //Number of inputs

    //Get number of possible cases (2^inputs)
    double cases = pow((double)2,inputs);
    //OR you can also do:
    //double cases = inputs*inputs;
    //So you don't require math.h, which might be an optimization.

    printf("LOGIC FUNCTION: C*B + A*(B+C)\n\nINPUTS | OUTPUTS\nC B A | S1 \n");

    //Compute all cases
    for(double x = 0; x < cases; x++)
    {
        //Calculate the state of s1 for the current inputs
        s1 = (c && b) || (a && (b || c));

        //Print the current state. Replace ON and OFF with TRUE or FALSE if you want.
        if(s1)
            printf("%u %u %u -> ON\n",c,b,a);
        else
            printf("%u %u %u -> OFF\n",c,b,a);

        //Switch the inputs.
        c = !c;
        if(!c)
            b = !b;
        if(!b && !c)
            a = !a;
    }
}
```

```
//Pause the console so we can read the output.  
getchar();  
return 0;  
}
```

//This program has the following output:

```
/*  
LOGIC FUNCTION: C*B + A*(B+C)
```

INPUTS | OUTPUTS

C B A | S1

0 0 0 -> OFF

1 0 0 -> OFF

0 1 0 -> OFF

1 1 0 -> ON

0 0 1 -> OFF

1 0 1 -> ON

0 1 1 -> ON

1 1 1 -> ON

\*/

Pegado de <<http://nинихаку.ку9.ко/блог/>>

## First Post

sábado, 12 de julio de 2014

2:19

[July 6, 2014](#) [Leave a comment](#) [Edit](#)

Howdy!

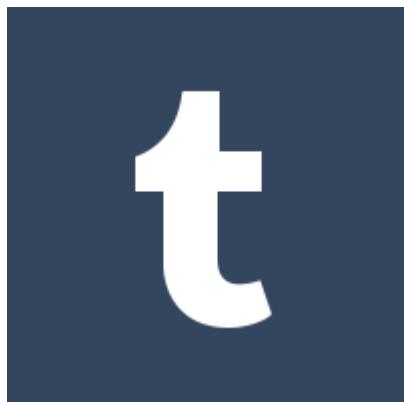
I've always wanted to have some short of "personal" blog where I could write things and share my hobbies with others. It's not the first time I make a blog or anything, I've tried things like this many times before, but in another way.

I once had a blog, in Spanish, that was only about technology. I want to write about that and, in fact, I'd like to translate some of the articles I wrote there to English, since there is some interesting stuff there. The reason why I deleted that blog, and started this one... well, there are two reasons. First of all, I wanted a blog that was more "me", by adding all of my hobbies. And also, I wanted to start an english blog, so I could reach more people.

I'm glad I can speak the two most spoken languages in the world, but only writing in spanish is not enough. I want to be able to reach more people.

If you want to know what will I be uploading here, it's simple. Drawing stuff, technology in general and programming. Also some personal stuff. Not fully personal, of course, but things that are related to me.

Oh, and also videogames.



My goal is to use the lesser number of networks possible, although I can't make it less than 2. I also have a tumblr, which is [ninjihaku.tumblr.com](http://ninjihaku.tumblr.com), that I dedicate exclusively to drawings and videogame stuff. If you want to check it out, feel free to do so.

There will also be a download section where I will upload my programs, games and stuff. I won't upload viruses or anything like that, but if you don't trust them, I will understand. However, I'd suggest you at least check it out. You might find something interesting, when I upload it.

Also, I have in mind to add a very tiny image board. Just one board, though. Only as an experiment. I don't want anyone to vandalize or abuse of it, be aware of that.

Pegado de <<http://ninjihaku.cu9.co/blog/>>

# Choose your destiny!

sábado, 12 de julio de 2014

2:20

## Choose your Destiny! (Or how to choose your programming language wisely)

[July 6, 2014](#) [Leave a comment](#) [Edit](#)



One of the most concerning topics I've ever talked about in my past blog, was the election of the adecuated programming language, for starters. Like, if you are new or you want to become a programmer, you might find yourself overwhelm by the amount of languages in the industry. C languages, BASIC languages, Script languages, Perl, Python, Pascal, Java... Wow! So many languages, yet you can achieve the exact same thing with most of them. Depending on your knowledge and the tools you use, of course.

Don't panic! The language is just an interface, and once you've learn about the main languages (which are 3), everything else is piece of cake, as they follow the same logic, but with slightly different keywords or syntax. Like you could make a program using C++, and port it to Java, making a few modifications.

There are 3 main programming languages, and the rest (or most of them) are just "branches". Those languages are ASM (Assembler, I will talk about Intel's 8086 and derivated), C, and BASIC.

The ASM language is the most low level programming language, as it lets you interact with the hardware. Well, unless you are in [protected mode](#) (in other words, inside a 32-bit operating system, in a computer). With this, you can make anything. Mostly. However, it's extremely complex. This language is for experts only, or people that is into robotics, or engineers.

ASM should be your last step into your learning process, as understanding it requires to understand first how a computer works, with a huge level of detail. Also, depending on the processor or controller you're programming, this language may vary on it's opcodes. Meaning that an [8086](#) program is not valid for a [16 bit PIC](#) microcontroller, or a [Motorola 68000](#).

This example shows a 16 bit ASM program for an 8086 processor, and derivated, that allocate a portion of memory using MS-DOS calls, and store an user input text on it, then prints it.

```
1      ; 16 BIT ASM MEMORY ALLOCATION EXAMPLE Revision 2
2      ; COMPILES WITH NASM
3      ; 06-10-2013
4      ; By NinjaHaku
5
6      ;CHANGELOG:
7      ;- Changed AX for CX, to be used as a counter for the
8      ; print routine.
9      ; CX is a base register meant to be used to track loops.
10     ; Any base register can be used for that purpose, though. But
11     ; this way, the code is more organized and clean.
12
```

```

13      ;
14      ; This program allocate 32 bytes in memory using
15      ; DOS INT 21H (AH=48H), then store user input characters
16      ; into the allocated memory.
17      ; When the user press ENTER, or reach the character limit,
18      ; the program prints the text stored in the memory into the
19      ; screen.
20
21      ;DOS Allocate memory using "paragraphs". One paragraph is
22      ;16 bytes long.
23
24      [BITS 16]
25      ORG 100h
26
27      section .text
28
29      MAIN:
30          mov ah, 0x0  ;Set video mode
31          mov al, 0x12 ;Mode 12, VGA 16 colors
32          int 0x10    ;Start VGA. Unnecesary, but it looks better
33          .MALLOC      ;Memory allocation
34          xor ax, ax  ;Initialize ax
35          mov bx, 0x02 ;16*2 = 32 bytes
36          mov ah, 0x48 ;Function 48 (malloc)
37          int 0x21    ;Do it
38
39          mov di, ax  ;Set destination pointer to the allocated memory
40          xor bx, bx  ;Initialize BX
41
42          .GETCHAR:   ;Read a character from keyboard, store it in memory
43          push bx    ;Just in case, save bx in the stack
44          xor al, al  ;Initialize al
45          mov ah, 0x01 ;Mode 1, get character with echo
46          int 0x21    ;Do it
47          pop bx    ;Restore bx
48          cmp al, 0x0d ;Has the user pressed ENTER?
49          je .PRINT  ;If so, print the text
50          mov [bx+di], al ;Store al in memory
51          inc bx    ;Increment in 1 the value of bx
52          cmp bx, 32  ;Check for memory overflow
53          je .PRINT  ;If so, scape
54          jmp .GETCHAR ;Keep reading the keyboard
55
56          .PRINT      ;Print the text on screen (ploop is the loop)
57          cmp bx, 0  ;Check if the user input something
58          je .end    ;If not, exit
59          mov si, di  ;Prepare to read the memory
60          xor ax, ax  ;Initialize AL
61          push bx    ;We'll need bx's value it later
62
63          mov dh, 0x02 ;We'll move the cursor to the second line
64          mov dl, 0x0  ;Column 0
65          mov bh, 0x0  ;First page
66          mov ah, 0x02 ;Set cursor position mode
67          int 0x10    ;Do it!
68
69          pop bx    ;Recover bx
70          xor cx, cx  ;We'll use CX as a counter
71          .ploop
72          push bx    ;We need to push again inside the loop
73          lodsb     ;Get next character
74          mov ah, 0x0e ;Text mode
75          mov bh, 0x0  ;First Page
76          mov bl, 0x07 ;Text attribute
77          int 0x10    ;Do it
78          pop bx    ;Recover bx
79          inc cx    ;Increment AL
80          cmp cx, bx  ;Check if we have readed every byte
81          jne .ploop ;Keep printing characters
82

```

```

83     .end          ;Wait for a keypress, then end
84         mov ah, 0x01 ;We are going to wait for the user to press a key
85         int 0x16    ;Do it
86         je .end    ;Keep waiting until it happen
87
88     .dealloc      ;Free the used memory
89         mov ah, 0x49 ;Deallocate memory
90         mov es, si   ;Set the segment parameter
91         int 0x21    ;Do it!
92
93     .exitprogram
94         mov al, 0x03 ;Return to text mode
95         mov ah, 0x00 ;Set video mode
96         int 0x10    ;Do it
97         int 0x20    ;Exit program with no parameters
98
99     section .data
100    Signature db "By NinjaHaku Software" ;Unused, ignore it
102
103    section .bss

```

The C language is the most common and standard language. It's also low level, sometimes you can also interact directly with the hardware (again, if you're not into protected mode) and sometimes using a mixture with ASM, through "inline assembler", making it the perfect general-purpose programming language. As well as ASM, it's very complex, but it's still way better than using ASM when it comes to complex programs such as videogames, emulators/interpreters, and robotics.

Unlike ASM, you can compile a C program for different devices, making it more portable. However, it depends on the compiler and the toolkits you use, meaning that you might still have to adapt the code a bit.

However, C has a disadvantage against it's successor. It lacks of [Object-oriented programming syntax](#). This means, you can't create instances of "objects" in C, and thus, the management of each "field" in your program must be performed manually. Like, for example, if you want to make a program that controls 3 windows, you have to build a structure that defines what a window is, and its properties and pointers to each one of their functions, and then build an array that contains the 3 windows. While in C++, you just create a class that defines a window, its functions, and then you create instances of it (You still need to build the array, though).

An example of C, a Window station made using BGI (A Borland C library used for 16-bit graphics)

```

1      /*
2      * bgitest.cpp
3      *
4      * Author: Aaron C d C
5      *
6      * Date: 05/08/2013
7      *
8      * Description: Program that creates and manages a
9      * DOS window station using BGI
10
11     * WINMANAG.H : http://pastebin.com/LMrjmsmf
12     */
13
14     #include <dos.h>
15     #include <math.h>
16     #include <conio.h>
17     #include <stdio.h>
18     #include <stdlib.h>
19     #include <stdarg.h>
20     #include <graphics.h>
21     #include <bios.h>
22     #include <math.h>

```

```

23 #include "winmanag.h"
24
25 int GraphDriver;      /* The Graphics device driver      */
26 int GraphMode;        /* The Graphics mode value       */
27 int ErrorCode;        /* Reports any graphics errors   */
28
29 /* Define a command button */
30 struct LcmdButton{
31     int orX, orY, wh, hh;
32     char isSelected; //Selected flag
33     char* txt; //Caption text
34     char (*comm)(int, char*[]); //Pointer to a function
35 };
36
37 /* Define a window */
38 struct LWindow{
39     int orX, orY, wh, hh; //Origin X and Y, width, height, and ID
40     char isActive; //Active window flag
41     char nRef; //Need Refresh flag
42     char* title; // Title string
43     char* txt; // Text inside
44     LcmdButton* cmdns; //Pointer to an array of buttons
45     int buttons; //Quantity of buttons
46 };
47
48 /* Initialize graphics. Taken from bgidemo.c */
49 void Initialize()
50 {
51     GraphDriver = DETECT; /* Request auto-detection */
52     initgraph( &GraphDriver, &GraphMode, "C:\\BC45\\BGI" );
53     ErrorCode = graphresult(); /* Read result of initialization*/
54     if( ErrorCode != grOk ){ /* Error occurred during init */
55         printf(" Graphics System Error: %s\n", grapherrmsg( ErrorCode ) );
56         exit( 1 );
57     }
58 }
59
60 /* Prints a string with format. Taken from bgidemo.c */
61 int gprintf( int *xloc, int *yloc, char *fmt, ... )
62 {
63     va_list argptr; /* Argument list pointer */
64     char str[140]; /* Buffer to build string into */
65     int cnt; /* Result of SPRINTF for return */
66
67     va_start( argptr, fmt ); /* Initialize va_ functions */
68
69     cnt = vsprintf( str, fmt, argptr ); /* prints string to buffer */
70     outtextxy( *xloc, *yloc, str ); /* Send string in graphics mode */
71     *yloc += textheight( "H" ) + 2; /* Advance to next line */
72
73     va_end( argptr ); /* Close va_ functions */
74
75     return( cnt ); /* Return the conversion count */
76 }
77
78 /* Draw a button in a window */
79 int DrawButton(LcmdButton cmdbx)
80 {
81     setfillstyle(9,8); //Shader
82     bar(cmdbx.orX-1,cmdbx.orY-1,cmdbx.wh-1,cmdbx.hh-1);
83
84     if(cmdbx.isSelected == 0x0)
85         setfillstyle(1,8); //Button inactive
86     else
87         setfillstyle(1,7); //Button active
88
89     bar(cmdbx.orX+1,cmdbx.orY+1,cmdbx.wh,cmdbx.hh);
90     outtextxy(cmdbx.orX+(cmdbx.wh/3 - cmdbx.hh/2),cmdbx.orY+(cmdbx.wh/3 - cmdbx.hh/2), cmdbx.txt);
91
92     return 0;

```

```

93     }
94
95
96     /* Draw a window in the screen */
97     int DrawWindow(lWindow windw)
98     {
99         setviewport(windw.orX, windw.orY,windw.wh+windw.orX, windw.hh+windw.orX, 0);
100        setcolor(15);
101        setfillstyle(9,8);
102        bar(1,1,windw.wh-1,windw.hh-1); //Shader
103        setfillstyle(1,3);
104        bar(7,1,windw.wh-1,windw.hh-7); //Window bg
105        setcolor(7);
106        setlinestyle(1,1,3);
107        rectangle(8,2,windw.wh-2,windw.hh-8); //Border
108        setfillstyle(1,7);
109        bar(7,1,windw.wh-1,12); //Title bar
110        setcolor(0);
111        outtextxy(9,4,"-");
112        outtextxy((windw.wh/2)-(12*sizeof(windw.title)),4,windw.title);
113        outtextxy(windw.wh-20,4,"-");
114        setcolor(1);
115        outtextxy(16,16, windw.txt);
116
117        /* If our window has some buttons, draw them */
118        if(windw.buttons > 0)
119            for(int x = 0; x < windw.buttons; x++)
120                DrawButton(windw.cmdns[x]);
121        return 0;
122    }
123
124    /* Create a new window */
125    lWindow InitWindow(int x, int y, int Width, int Height, char* title, char* text, lcmbButton* lcmds, int bts)
126    {
127        lWindow win_;
128
129        win_.orX = x;
130        win_.orY = y;
131        win_.wh = Width;
132        win_.hh = Height;
133        win_.title = title;
134        win_.txt = text;
135        win_.isActive = 0x0;
136        win_.nRef = 0x1;
137        if(win_.cmdns != NULL)
138            win_.cmdns = lcmds;
139        win_.buttons = bts;
140        return win_;
141    }
142
143    /* Create a new button */
144    lcmbButton CreateButton(int x, int y, int Width, int Height, char* title, char isSel, int (*func)(int, char*[]))
145    {
146        lcmbButton cmdb_;
147
148        cmdb_.orX = x; //Origin X
149        cmdb_.orY = y; //Origin Y
150        cmdb_.wh = Width; //Width
151        cmdb_.hh = Height; //Height
152        cmdb_.txt = title; //The caption
153        cmdb_.isSelected = isSel; //Selected flag
154        return cmdb_;
155    }
156
157    /* Beep trough the speaker */
158    void Beep(int ms, int Hz)
159    {
160        sound(Hz);
161        delay(ms);
162        nosound();

```

```

163     }
164
165     /* Used to draw the screen */
166     void DrawScreen(IWindow *wStatl_)
167     {
168         cleardevice();
169         clearviewport();
170
171         setbkcolor(1);
172
173         for(int xi = 0; xi < _MAX_WINDOWS_; xi++)
174         {
175             if(wStatl_[xi].isActive == 0x1 && wStatl_[xi].nRef == 0x1)
176             {
177                 DrawWindow(wStatl_[xi]);
178                 //wStatl_[xi].nRef = 0x0;
179             }
180         }
181     }
182
183
184     /* A function to be called when a command button is triggered */
185     int ButtonOk(int args, char* argv[])
186     {
187         return 0; //Does nothing, though
188     }
189
190     /* Program's entry point */
191     int main()
192     {
193         char key = 0x0;
194         char* tstStrK = "This is a text K window!";
195         char* tstStrQ = "This is a text Q window!";
196         int WinCount = 0;
197         Initialize(); //Initialize the screen
198
199         //Create a window buffer
200         IWindow *wStatList = (IWindow *) malloc(sizeof(IWindow) * _MAX_WINDOWS_);
201         //Draw the current screen
202         DrawScreen(wStatList);
203
204         //Main loop (scape with the scape key)
205         while(key!=0x1b){
206             key = getch();
207
208             if(key == 'k') //K key creates a K window
209             {
210                 wStatList[WinCount] = InitWindow(20,65,350,300,"K Window",tstStrK, NULL, 0);
211                 wStatList[WinCount].isActive = 0x1;
212                 if(WinCount < _MAX_WINDOWS_)
213                     WinCount = WinCount + 1; //Adds 1 window to the count
214             }
215
216             if(key == 'q') //Q key creates a Q window
217             {
218                 //Create a button array. Only one button will be used
219                 lcmdButton* butts = (lcmdButton*) malloc(sizeof(lcmdButton)*1);
220                 butts[0] = CreateButton(85, 50, 125, 70, "OK", 0x1, ButtonOk);
221                 wStatList[WinCount] = InitWindow(40,95,250,100,"Q Window",tstStrQ, butts, 1);
222                 wStatList[WinCount].isActive = 0x1;
223                 if(WinCount < _MAX_WINDOWS_)
224                     WinCount = WinCount + 1; //Adds 1 window to the count
225             }
226
227             if(key == 'c') //C closes the active window
228             {
229                 if(WinCount > 0)
230                     WinCount = WinCount - 1; //One window less to care about
231                 wStatList[WinCount].isActive = 0x0;
232                 wStatList[WinCount].nRef = 0x0;

```

```

233     }
234
235     //Enable the refresh flag for all active windows only
236
237     for(int x = 0; x < _MAX_WINDOWS_; x++)
238         wStatList[x].nRef = (wStatList[x].isActive == 0x1)?0x1:0x0;
239
240     //Redraw the entire screen
241     DrawScreen(wStatList);
242 }
243
244     cleardevice();
245     closegraph(); /* VERY IMPORTANT! */
246     return 0;
247 }
```

Because of this, and because C++ has some kind of “automatized” memory management (thanks to the new C++ standards) through it’s standard libraries, C is almost in desuse when it comes to computer programming. Instead, they use C++.

C++ is also split into two branches. The standard C++, and Microsoft’s Visual C++, that is nothing but an extra API for C++ with additional functions. It still maintains the inline assembler feature, and also combine it with OOP features, but still maintains certain aspects from a low level programming language, meaning that the memory management is, mostly, up to you.

Here’s a small C++ example, that shows the inline assembler feature (using Visual C++’s compiler, which uses the standard ASM syntax, unlike GCC, which uses AT&T which, in my opinion, it’s very confusing).

```

1      /*
2       * VISUAL C++ INLINE ASSEMBLY DEMO
3       * By Ninjihaku
4       * 28-08-2013
5       *
6      */
7
8      #include <cstdio>
9
10     using namespace std;
11
12     /*Main function. argc and argv are unnecesary, you can
13      * delete them if you want */
14     int main(int argc, char* argv[])
15     {
16         //Declare a variable called "Something"
17         unsigned int something;
18         //Store the address of 'something' in sAddr.
19         unsigned int* sAddr = &something;
20         //Operation result
21         unsigned int res = 0;
22         //Inline assembly:
23         //We do: res = 0x1234 + 0x1234; something = 0xBEEF;
24         //In 32 bit assembler
25         __asm {
26             mov eax, 0x1234 //EAX = 0x1234
27             mov ebx, 0x1234 //EBX = 0x1234
28             add eax, ebx //EAX += EBX (EAX = 0x2468)
29             mov [res], eax //res = EAX; res = 0x2468
30             mov edi, [sAddr] //Point to our variable 'something'
31             push eax //Store EAX in the stack (0x2468)
32             mov eax, 0xBEEF //EAX = 0xBEEF
33             mov [edi], eax //something = EAX = 0xBEEF
34             pop eax //Restore EAX (EAX = 0x2468)
35         }
36         //Prints the result
```

```

37     //2468 - BEEF
38     printf("%x - %x",res,something);
39     //Wait for user input
40     getchar();
41     //DONE!
42     return 0;
43 }
```

As for BASIC, it's an ancient programming language meant for newbies to learn something about programming. The first apple computers in the 60's and 70's used it, and most people remember it because MS-DOS included a BASIC (Or rather [QBASIC](#), that stands for Quick BASIC) interpreter that could let you make games and stuff.

It does not have OOP, and it only works in 16 bit machines, though there are some 32-bit compilers on the internet that let you create 32 bit applications using BASIC.

BASIC is not my thing, but this small code demonstrate how to draw a square with 4 lines, using VGA in BASIC.

```
SCREEN 13 LINE (10, 10)-(100, 100), 192, B ' Notice the B WHILE INKEY$ = "": WEND SCREEN 0
```

```

1      SCREEN 13
2      LINE (10, 10)-(100, 100), 192, B
3      ' Notice the B
4      WHILE INKEY$ = "": WEND
5      SCREEN 0
```

I don't really recommend using this language for anything that is not for fun, as it's extremely outdated.

Anyway, Microsoft likes BASIC a lot, and thus, he has its own BASIC branch (more like a successor) called Visual Basic. Which also has two versions. One being the plain, old Visual Basic (32-bit only, no OOP, and works on Windows 9x), and the other is the .NET version. The .NET version also includes OOP, and 32-64 bit support. But it runs in some kind of "virtual machine", as every .NET language (something alike Java. Not the same, but alike). Also note that .NET programs won't work in 9x operating systems, though that should not be a problem nowadays.

This is an example you can find in the Wikipedia's article of VB.NET. It does some maths to solve Floyd's pyramid.

```

1      Imports System.Console
2      Module Program
3          Sub Main()
4              Dim rows As Integer
5              ' Input validation.
6              Do Until Integer.TryParse(ReadLine("Enter a value for how many rows to be displayed: "),
7              rows) AndAlso rows >= 1
8                  WriteLine("Allowed range is 1 and {0}", Integer.MaxValue)
9              Loop
10             ' Output of Floyd's Triangle
11             Dim current = 1
12             For row = 1 To rows
13                 For column = 1 To row
14                     Write("{0,-2}", current)
15                     current += 1
16                 Next
17                 WriteLine()
18             Next
19         End Sub
20         "" <summary>
21         "" Shadows Console.ReadLine with a version which takes a prompt string.
22         "" </summary>
23         Function ReadLine(Optional prompt As String = Nothing) As String
24             If prompt IsNot Nothing Then
```

```

25     Write(prompt)
26     End If
27     Return Console.ReadLine()
28   End Function
29 End Module
30

```

And now, it is the time to decide which language to learn first.

I always give two options to whoever ask me about this. One option, the one I don't recomend, is to start strong. Learn C++. I don't recomend this at first, because it may be very confusing.

The second option, the one I recomend, is to learn C#, another .NET language. Why? It's simple. C# is very similar to C++ in its syntax, so if you learn C#, learning C-Based languages is way easier, although there are several differences between C/C++ and C# (Java is extremely similar btw, so learning C# almost feels like learning Java). It includes OOP as any .NET language, and also, the memory management is automatized. So you don't have to deal with pointers, etc. Making it ridiculously easy to learn and masterize.

Here's one example, a console application in C# that mix two lines of text:

```

1  static void Main(string[] args)
2  {
3      //Entrada means Input, Salida means Output. Textos means Texts.
4      string Entrada = "", Salida;
5      string[] Textos = new String[2];
6      char[] t1, t2 = new char[1024];
7
8      Console.WriteLine("  TEXT MIXER  \n\nWrite a text of no more than 1024 characters: \n");
9      Entrada = Console.ReadLine();
10     t1 = Entrada.ToCharArray();
11     Console.WriteLine("\nWrite another text of no more than 1024 characters: \n");
12     Entrada = Console.ReadLine();
13     t2 = Entrada.ToCharArray();
14
15     Console.WriteLine("Mixing...\n\n");
16
17     for (int x = 0; x < t1.Length-1; x += 2)
18     {
19         if (x > (t2.Length-1) || x > (t1.Length-1)) { break; }
20         t2[x] = t1[x];
21     }
22
23     Salida = new String(t2);
24     Console.WriteLine(Salida);
25     Console.ReadKey();
26 }

```

In conclusion, my recommended learning course is:

C# -> C++ & C -> ASSEMBLER

If you dominate these, you can program on nearly anything easily.

Pegado de <<http://nинихаку.ку9.ко/blog/>>

# Technical Difficulties

sábado, 12 de julio de 2014  
2:22

## Technical difficulties

[July 9, 2014](#) [Leave a comment](#) [Edit](#)

Seems like the site is having some issues, typical from free webhostings. I'm glad it's the host having difficulties, and not the fact that last night, we were spamming while doing some testings in my chan. I deleted everything, though, just in case.

Pegado de <<http://nинихаку.сu9.co/blog/>>

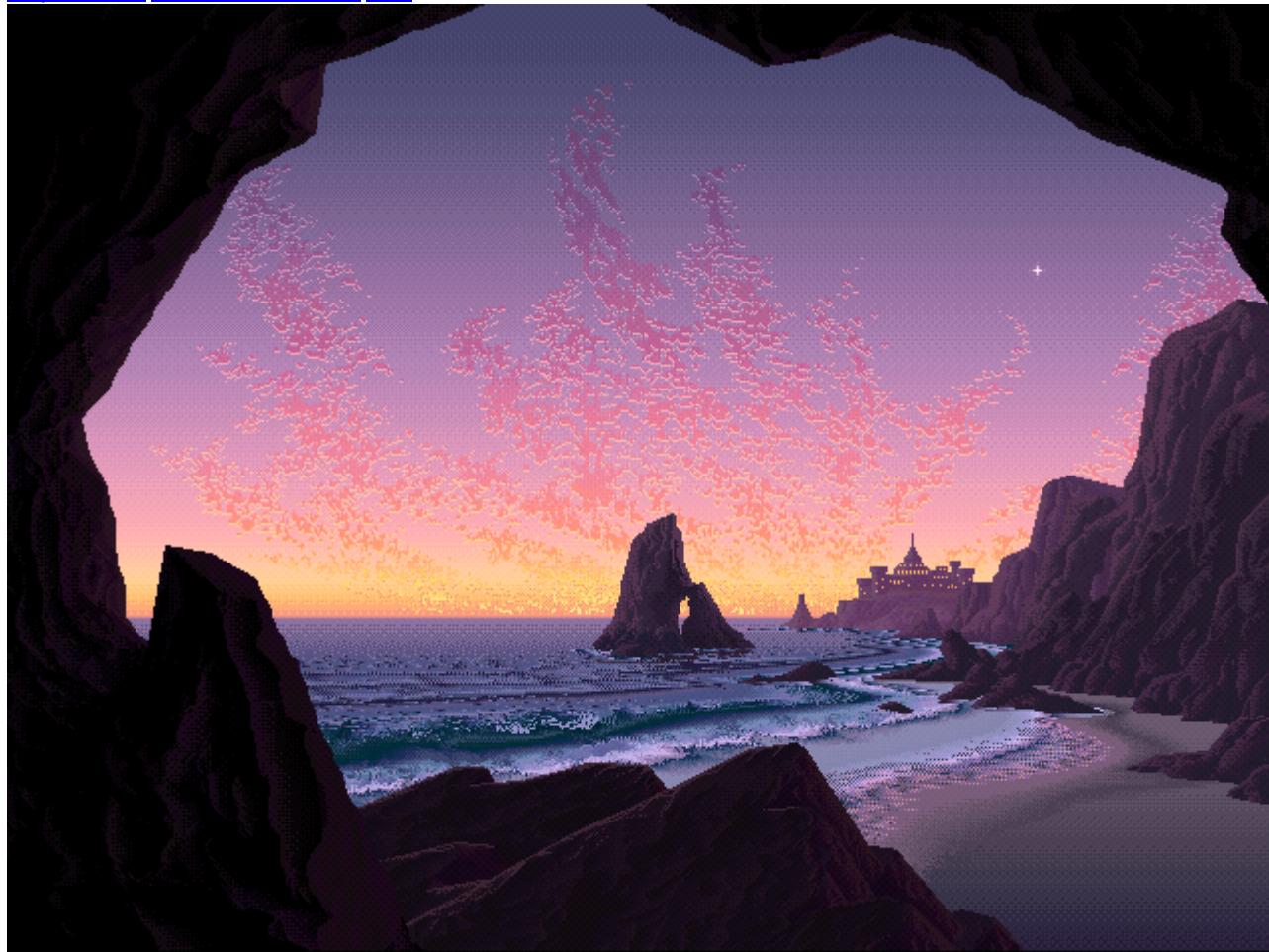
# Working on SVGA DRIVERS

sábado, 12 de julio de 2014

2:22

## Working on SVGA “drivers”

[July 9, 2014](#) [Leave a comment](#) [Edit](#)



If I could achieve something like this...

Last night I was working in some kind of SVGA driver, to be able to use screen sizes larger than 320×210, with at least 256 colors, in my DOS programs. It took me a few very long hours of research until I finally got in the right track. Or I think I did.

You might not know, but VGA uses a range of memory, from 0xA0000 to 0xBFFF (Or 0xB0000 to 0xBFFFF in text mode). The problem is, you can't use screen sizes larger than 320×210, using 256 colors. You can use, at most, 640×480, but only 16 colors.

That means that if you want to use screen sizes larger than that, with 256 colors, you need SVGA drivers.

The VESA SVGA modes has a peculiarity. As the whole screen does not fit in the portion of the memory reserved for VGA, it divides the screen in portions, and let you switch the portion of the screen that is mapped to the VGA address.

I'm currently working on this, but I think I got this working after a lot of research. The first step, is to get the VESA extension information, to check if it's available (should be), and it's characteristics. In C, I use a “struct” to store and easily retrieve this information. There is also a pointer to a 16 bit integer array that list all the video modes available (like 0x100, 0x101, etc), being the last one 0xffff (-1). This can be useful for the next step.

The next step, is to check either if the video mode we are going to use is supported, or each one of the video modes that VESA says are currently supported. Like, in most cases, you should be able to use up to 1280×1024, but maybe your computer has 1Mb of video memory, and it can't be used because you don't have enough video memory. We can retrieve each mode's information into another structure, and test the very first bit of the ModeAttributes field. If this bit is 0, the mode is unsupported.

The next step, is set the video mode (in other words, start the graphic mode). It can be easily done through BIOS interrupt 10h, being AX 0x4f02, and BX the video mode. I'm using the 0x101 video mode, which is 640×480 @ 256 colors. Video mode 3 (0x3) is DOS's text mode, in case you want the program to return to text mode after it's execution (and you want).

The next step, set the DAC Color Palette. I did this very late in the night and I didn't pay enough attention to it, but it seems like we tell the DAC that we want to use 256 color, and store the palette information into an array of 256\*3 bytes. Or the number of colors we are going to use, multiply by 3. This is also done by the 0x10 Interrupt, being AX 0x4f09, BL and DX 0 (not sure why right now), CX the number of colors (in my case 256), and ES:DI pointing to the segment:offset of the array of colors.

The two last steps would be, writing a function to draw (I think you can also use BIOS interrupts to draw pixels, but it's really slow) by calculating the portion of the screen where every pixel goes (and using another 0x10 interrupt to switch to that portion), and of course, the page switch function (to change the portion of the screen we want to draw to). This is what I'm currently working on, shouldn't be too hard (just maths).

This C code is the main function of the program. It's still WIP, but it seems like it's working (At least it seems to set the video mode). Not sure though, I must do the drawing functions first to test it properly. It compiles in Open Watcom, in a DOS, 16 bit EXE project.

```
1      /*
2      * Ninjihaku's VGA
3      * main.c : Main program
4      * 09/07/14
5      */
6
7      #include <conio.h>
8      #include <stdio.h>
9      #include <dos.h>
10     #include <i86.h>
11     #include <stdlib.h>
12     #include <string.h>
13     #include "nga.h"
14
15     VESA_INFO vinfo;
16     MODE_INFO minfo;
17     int CursorOffset = 0;
18     int CursorPosition = 0;
19     char gotVesaInfo = 0x0;
20     char _paletteData[768];
21
22     void __detectVESA()
23     {
24         unsigned int offset_ = FP_OFF(&vinfo);
25         unsigned int segment_ = FP_SEG(&vinfo);
26         union REGS r;
27         struct SREGS s;
28
29         r.x.ax = 0x4f00;
30         r.x.di = offset_;
31         s.es = segment_;
32         int86x(0x10,&r,&r,&s);
33
34         if(r.x.ax == 0x004f && strcmp(vinfo.VESASignature, "VESA") == 0)
```

```

35         gotVesaInfo = 0x1;
36     else
37     {
38         printf("\nERROR 01\n");
39         return;
40     }
41
42     printf("\n%s\n", vinfo.VESASignature);
43
44     r.x.ax = 0x4f01;
45     r.x.cx = 0x101; //The mode to test
46     r.x.di = FP_OFF(&minfo);
47     s.es = FP_SEG(&minfo);
48     int86(0x10,&r,&r,&s);
49     if(!CHECK_BIT(minfo.ModeAttributes, 0))
50         printf("\nNot supported in this computer\n");
51     return;
52 }
53
54 void __initVesaMode()
55 {
56     union REGS r;
57     r.x.ax = 0x4f02;
58     r.x.bx = 0x101;
59     int86(0x10,&r,&r);
60     return;
61 }
62
63 void __initTextMode()
64 {
65     union REGS r;
66     r.x.ax = 0x4f02;
67     r.x.bx = 0x3;
68     int86(0x10,&r,&r);
69     return;
70 }
71
72 int __calcPosition(int x, int y)
73 {
74     CursorOffset = (x+y*minfo.XResolution) % minfo.WinGranularity;
75     CursorPosition = (x+y*minfo.XResolution) / minfo.WinGranularity;
76     return CursorPosition;
77 }
78
79 void __switchPage(char Window, int Position)
80 {
81     union REGS r;
82     r.x.ax=0x4f05;
83     r.h.bh = 0;
84     r.h.bl = Window;
85     r.x.dx = Position;
86     int86(0x10,&r,&r);
87     return;
88 }
89
90 void __WInFuncPtr(char Window, int Position)
91 {
92     unsigned long wfunc = minfo.WinFuncPtr;
93     __asm{
94         sub bh,bh
95         mov bl,Window
96         mov dx, Position
97         call wfunc
98     };
99     return;
100 }
101
102 void __setDACPalette()
103 {
104     union REGS r;

```

```

105     struct SREGS s;
106     r.x.ax = 0x4f09;
107     r.h.bl = 0;
108     r.x.cx = 256;
109     r.x.dx = 0;
110     r.x.di=FP_OFF(&_paletteData);
111     s.es=FP_SEG(&_paletteData);
112     int86x(0x10,&r,&r,&s);
113     return;
114 }
115
116 //TODO: Drawing Functions
117 //Check browser's favourites for references
118
119 int main()
120 {
121     __detectVESA();
122     __initVesaMode();
123     __setDACPalette();
124     getch();
125     return 0;
126 }
127
128 /*
129 *09-07-14 - 3:37 beta 001
130 */

```

The following code, is the `nga.h` header file. It contains the VESA data structures, and some macros and definitions.

```

1  /*
2  * Ninjihaku's VGA
3  * nga.h : VGA/SVGA definitions
4  * 09/07/14
5  */
6
7 #ifndef __NGH_
8 #define __NGH_
9
10 #ifndef NULL
11 #define NULL (void*)0
12 #endif
13
14 //SVGA MODES
15 #define SVGA_640x400_256 0x100
16 #define SVGA_640x480_256 0x101
17 #define SVGA_800x600_256 0x103
18 #define SVGA_1024x768_256 0x105
19 #define SVGA_1280_1024_256 0x107
20
21 #define CHECK_BIT(var,pos)((var)&(1<<(pos)))
22
23 typedef struct VESA_INFO
24 {
25     unsigned char VESASignature[4];
26     unsigned short VESAVersion;
27     unsigned long OEMStringPtr;
28     unsigned char Capabilities[4];
29     unsigned long VideoModePtr;
30     unsigned short TotalMemory;
31     unsigned short OemSoftwareRev;
32     unsigned long OemVendorNamePtr;
33     unsigned long OemProductNamePtr;
34     unsigned long OemProductRevPtr;
35     unsigned char Reserved[222];
36     unsigned char OemData[256];
37 }VESA_INFO;
38

```

```

39
40     typedef struct MODE_INFO
41     {
42         unsigned short ModeAttributes;
43         unsigned char WinAAttributes;
44         unsigned char WinBAttributes;
45         unsigned short WinGranularity;
46         unsigned short WinSize;
47         unsigned short WinASegment;
48         unsigned short WinBSegment;
49         unsigned long WinFuncPtr;
50         unsigned short BytesPerScanLine;
51         unsigned short XResolution;
52         unsigned short YResolution;
53         unsigned char XCharSize;
54         unsigned char YCharSize;
55         unsigned char NumberOfPlanes;
56         unsigned char BitsPerPixel;
57         unsigned char NumberOfBanks;
58         unsigned char MemoryModel;
59         unsigned char BankSize;
60         unsigned char NumberOfImagePages;
61         unsigned char Reserved_page;
62         unsigned char RedMaskSize;
63         unsigned char RedMaskPos;
64         unsigned char GreenMaskSize;
65         unsigned char GreenMaskPos;
66         unsigned char BlueMaskSize;
67         unsigned char BlueMaskPos;
68         unsigned char ReservedMaskSize;
69         unsigned char ReservedMaskPos;
70         unsigned char DirectColorModeInfo;
71         unsigned long PhysBasePtr;
72         unsigned long OffScreenMemOffset;
73         unsigned short OffScreenMemSize;
74         unsigned char Reserved[206];
75     }MODE_INFO;
76
77 #endif

```

I'll work a little bit more on this later. Seems like it's going well. I hope it works fine. If it does, I will be able to do a lot of interesting things...

Pegado de <<http://nинихаку.сu9.co/blog/>>

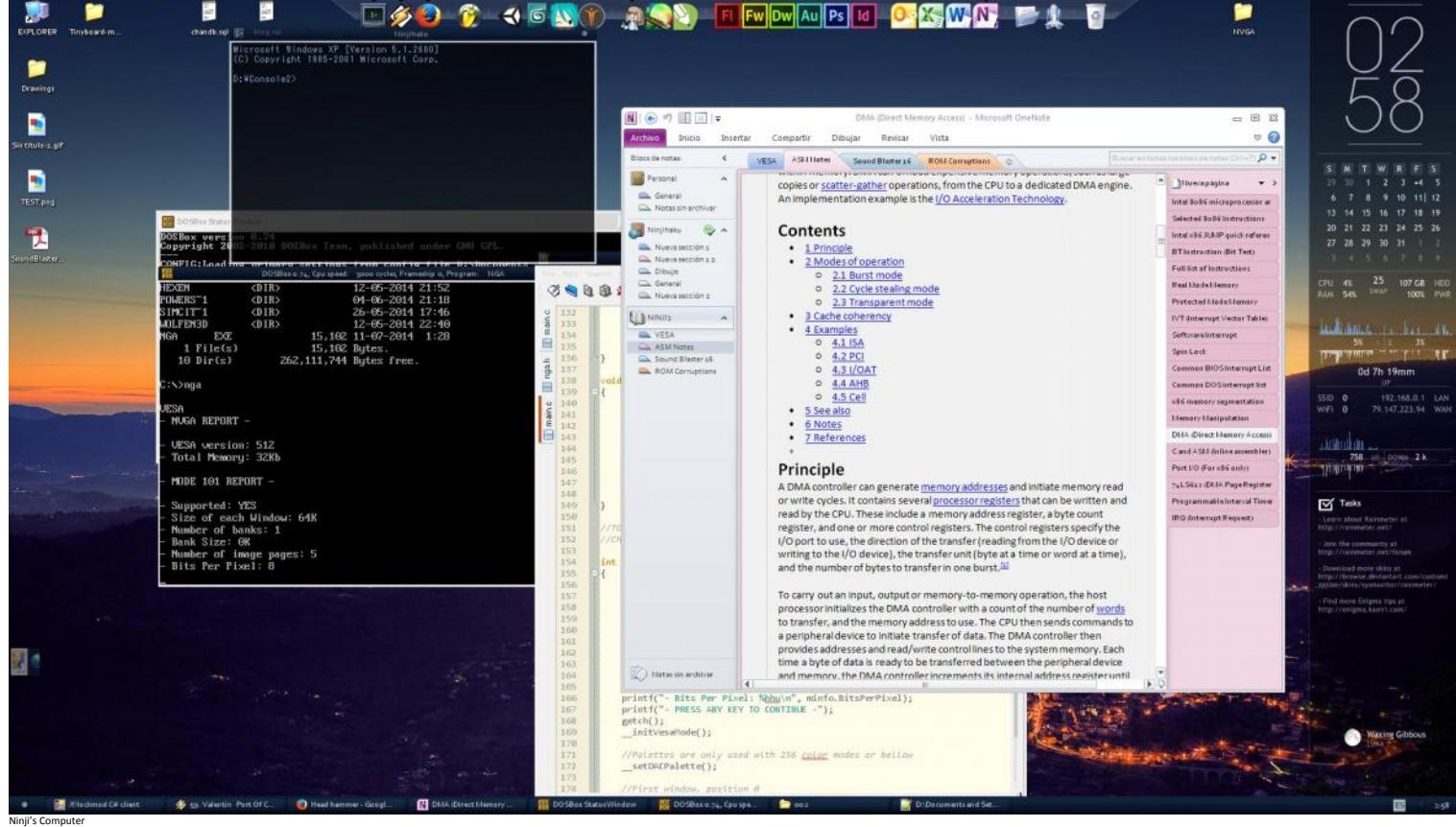
## So Much Bullshit

sábado, 12 de julio de 2014

2:23

### So much bullshit

July 11, 2014 Leave a comment Edit



After spending hours of researching about VESA extensions, ASM and computer architecture in general (including Port I/O, Direct Memory Access and Interrupt Request, among other things), I decided it was about time to keep working in my SVGA drawing application.

I have to say, that things are not going as expected. And not because it doesn't work (which it should), but because of lack of information.

You see, the internet can be a very nice place to share information. But when certain people, that do not know exactly how to explain things, try to explain something as complex as direct video manipulation in an x86 machine using VESA BIOS extension, you know that person is prone to commit mistakes, or leave things unexplained. Or worst of all, both.

There are two ways to explain what happen today. The easy to understand for everyone, and the detailed one that not everyone will understand. Let's start with the easiest one.

#### About how NOT to explain things...

Let's say I want to explain how to write an entry on a blog. I give you these steps:

- Step one: Open up your blog.
- Step two: Enter your control panel.
- Step three: Go to the Posts sections
- Step four: Click on "Add new"
- Step five: Write a title, and your post
- Step six: Publish it!

And now, in this part of the tutorial, I say "This is not all. I've left a lot uncovered".

Okay, hold on... What did I miss here? Did I said something about how to access your control panel? Did I explain how to format your text? Insert images, maybe? Did I, at least, state that you have a Publish button? That you can also save drafts of your blog entry? That you can program it to be published for a certain date and hour? No!

This kind of tutorials, are complete GARBAGE. And unfortunately, most of the things I find on the internet about these topics, are explained like this example. Yes, I'm not exaggerating.

#### About how NOT to explain VESA

This is going to be loooooong and boring, so maybe you want to skip this.

Okay, so as I stated in my previous entry, I can only draw in a portion of the memory, which is 64K only. From A000:0000 to A000:FFFF Also I stated this wrong in my previous entry). And if you want to manage a video mode that is larger than that, well... VESA segments the video memory into "Banks", that you can map to that portion of the memory.

I stated that you can "switch" the banks to map them to that portion. What I didn't say, is how VESA makes that Switch. In fact, it's not a switch. Here, let me show you this:



THIS is the complete memory range for a VESA mode 0x101, which is 640x480 x 256 colors. What did I miss in the last post? The Windows! How could you blame me? I was using a shitty explanation from a blog I found, that I considered legit.

So this is how the thing works. When you want to draw in a portion of the screen, you move one of the Windows through the memory, so that portion of the memory is mapped into A000:0000. That way we can make a far pointer to that address and manipulate it.

However, Which window should I move? A? B? Both? And how? Because this is what the other documentation page I found state about this:

The BIOS may provide you with up to 2 windows, called A and B. They may be either readable or writable or both. Often, though, one will be readable, the other writable. You can move

these windows around the memory. The picture shows them in arbitrary places in memory, but often, you will only be able to move them in 4Kb or 64Kb steps.

Ok, now I have to say... when did people start naming things with A or B in programming? I'll tell you their real name, those windows are called 0x0, and 0x1. And the writable one is 0x0. Also, you move them through the use of the interrupt 0x10.

And now, here comes the real deal. Pixel plotting. Or, in other words, drawing. You give the program 2 coordinates, then calculate the portion of the screen you draw in, move the screen 0x0 there, and draw into the memory region A000+0000+offset.

As I said, you have to do it this way, calculating the portion of the memory you have to draw your pixel in. Otherwise, it won't work. [Like this guy](#), that has a page that, if you look at the title, it says "SVGA Plotting that WORKS!". I knew it wasn't going to work anyway, but I had to try his solution.

Basically what he does is to use the interrupt's 0x10 pixel plotting function, that is meant for standard VGA. Here's how it works:

```
1 <b>INT 10h, 0Ch (12) Write Pixel</b>
2   Writes a pixel dot of a specified color at a specified screen
3   coordinate.
4
5   <b>On entry:</b> AH 0Ch
6   AL  Pixel color
7   CX  Horizontal position of pixel
8   DX  Vertical position of pixel
9   BH  Display page number [graphics modes with more
10    than 1 page]
11  <b>Returns:</b> None
12  <b>Registers destroyed:</b> AX, SP, BP, SI, DI
13
14
```

Okay. If you try it, it does what it's supposed to do. It draws in the video memory. The same as if you do this in C:

```
1 char* vga = (char*)0xA0000000L;
2 //Or also
3 #including <86.h>
4 const _far char* vga = MK_FP(0xA000,0x0000);
5
6 VGA[(y<8)+(y<6)*x]pixel; //pixel = char (obviously)
```

However, disregard the fact that we are in SVGA and not VGA, and thus, it ignores the memory banks, assuming that it will work by drawing in the 64K portion of the memory. So yes, you can see a pixel somewhere on the screen. Somewhere. And no, moving the window won't work. I've tried it already (that, or maybe the function I use to move the window doesn't work, which wouldn't surprise me). This is because the offset in SVGA may vary, so technically, it's not A000:0000.

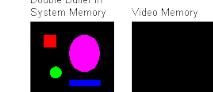
When I finally found a code that I thought would work, I found a problem. Here's an example code that, supposedly, draws in the screen.

```
1 void VbeWrite(int x,int y,int bytes,const char *buffer)
2 /* copies the contents of the buffer (<64k) by
3   starting at pixel (x,y). */
4
5 {
6   long absolute_x=(long)y*_Width; /* absolute offset */
7   long position=absolute/_Window; /* of window */
8   long offset=absolute%_Window; /* of window */
9   char far *vram=MK_FP(Segment0); /* to window */
10
11  VbeSetWindow(0,int)position);
12
13  if(offset>bytes)_Window /* data overruns window */
14  {
15    int n=(int)_Window-offset; /* bytes left */
16
17    _fmemcpy(vram+offset,buffer,n); /* display 1st part */
18    VbeSetWindow(0,int)+position); /* move window */
19    _fmemcpy(vram,buffer+n,bytes-n); /* display rest */
20
21  }
22  else _fmemcpy(vram+offset,buffer,bytes);/* no overrun */
23
24 }/* End of File */
```

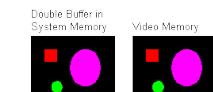
It might work. I hope so. However, it doesn't explain how this work. You don't draw here. What this does, is copying a buffer with the data you've drawn, into the video memory. In other words, **double buffering**. Which I am aware of because I read about it in a page about VGA:

### Double buffering

Double Buffering is a fairly simple concept to grasp. Instead of drawing directly to video memory, the program draws everything to a double buffer (Figure 20a). When finished, the program copies the double buffer to video memory all at once (Figure 20b). At that point the program clears the double buffer (if necessary) and the process starts over.



(a). Instead of drawing to video memory, a double buffer is used.



(b). When finished drawing, the double buffer is copied to video memory.

But, what does this guy tell about this? I'll quote him, literally:

**Is That All?**  
 Heck, no. I've ignored a bunch of stuff in order to nail down the basics without writing a book.  
 There's plenty of territory left to explore: linear frame buffers and DPMI; other memory  
 models and color models; multiple window segments, scrolling displays, and page flipping...you  
 get the idea.

No! I don't get the idea! And I don't get the idea because you've left something basic unexplained! Also, I'm not making anything up. [Here's this guy's website](#). There are 5 more pages, including the one with the code I posted. None of them explain anything with enough level of detail. So now I'm stuck with creating a frame buffer, which I assume SVGA must have it somewhere, since this guy is using a far pointer, and I can't allocate memory randomly by using far pointers while being in an OS (I might break something!). Where is that portion of the memory? I am still looking for it.

On the positive side of things...

I've learned about many interesting stuff today. Basic things like far pointers, that I had a notion but not full knowledge about them, a few things about C, and many about computers like memory models, ports, DMA, IRQs, etc. Also I learned more about VESA and, even if I still didn't get the thing working (I just need the right drawing function, which will someday come, somehow), I feel more comfortable with it. Also, if I manage to understand IRQ and DMA properly, I might as well do some sound drivers. At least for a SB16n which is more than enough since every emulator I know emulate this device.

On the bad side, if people keep explaining things like these guys over the internet, I might as well try to find a proper book about these topics. People think that everything can be found on the internet, but I'm starting to doubt it. Also, when I learn how to use this thing properly, I will explain it properly. And detailed.

Pegado de <http://nininjaku.cur9.co/blog/>

## About today...

sábado, 12 de julio de 2014  
3:26

## About today...

[July 12, 2014](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



mfw...

Well, today I spent most day programming. Or trying.

I got tired of spending my hours searching on the internet, so I made two things. The first one, is a One Note document, that shorts all kind of stuff I gather from the internet.

If you don't know what One Note is, it's one of those Microsoft products that come included with the newer Office packages, that most people ignore and don't give a fuck about, and that I just discovered and found extremely usefull to short all my programming shit and most information I search on the internet and other media. So I won't need to have 1,000 tabs in my browser, and also I have offline access to that stuff.

The Super VGA Graphics Card - Microsoft OneNote

**Archivo** Inicio Insertar Compartir Dibujar Revisar Vista

Insertar espacio Tabla Imagen Recorte de pantalla Vínculo Adjuntar archivo Copia impresa de archivo Copia impresa de escáner Grabar audio Grabar video Fecha Hora Fecha y hora Marca de tiempo Ecuación Símbolo Símbolos

Blocs de notas: VGA and VESA ASM Notes Sound Blaster 16 Old blog stuff ...

Buscar en todos los blocs de notas (Ctrl+T)

Nueva página ▾

The VGA Standard

VESA Programming notes

PCX C Functions

Programming With VESA BI

VESA - INT10 LIST

Vesa C structures & Drawing

The Super VGA Graphics Card

Page 3: Save The State of the

Page 5: Change the Virtual S

Page 6 - Plotting pixels

Page 7 - Scrolling

Page 8: Restore previous stat

Abe's bag of tricks, SVGA Pu

Personal

- General
- Notas sin archivar

Ninjihaku

- Nueva sección 1
- Nueva sección 2
- Dibujo
- General
- Nueva sección 2

NINJI2

- VGA and VESA
- ASM Notes
- Sound Blaster 16
- ROM Corruptions
- Old blog stuff
- New Blog Stuff

Notas sin archivar

address, then the inner loop will have to contain the test for crossing the end or beginning of the CPU address space. This is because if the length of the CPU address space (which is the granularity in this case) is not evenly divisible by the length of a scan line, then the scan line at the end of the CPU address will be in two different video memory which cannot be mapped into the CPU address space simultaneously.

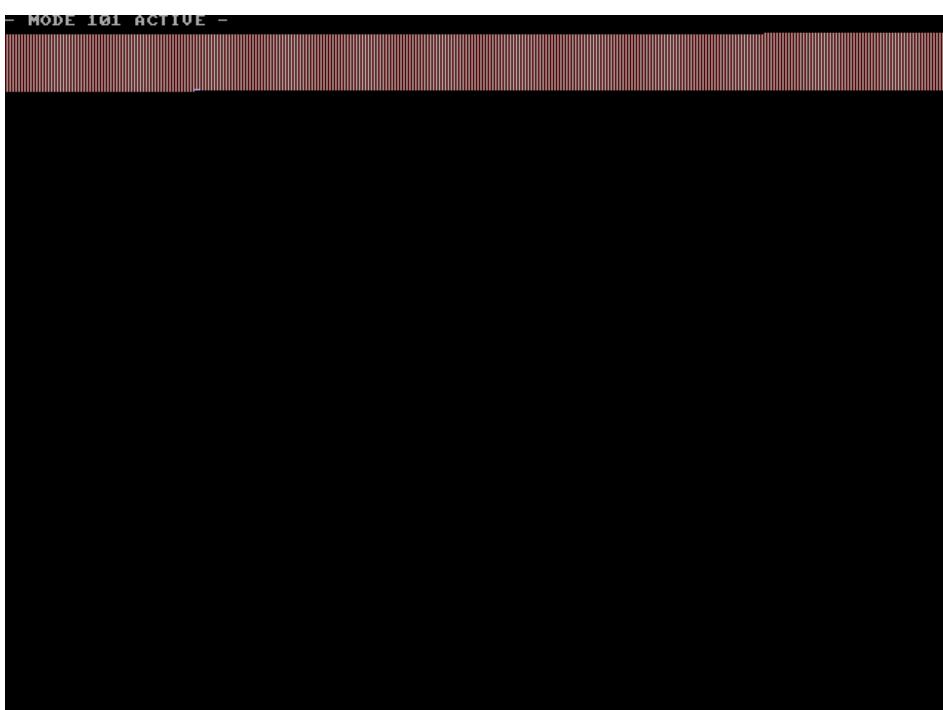
OK, so that's a bit of a mouthfull. However what they say is only partly true. Indeed, if the end of a bank occurs in the middle of a scan line, it can make things a little bit inconvenient, but it should never be necessary to check if the end of the bank has been reached after every pixel write.

It may be a little hard to visualise exactly what is happening here, so I think a picture is in order.

Right, so the big stripy thing is meant to be a representation of the screen, showing the banks. If you look closely, you can see that the end of the banks fall somewhere in the middle of the scanlines.

Now, lets say that we wanted to fill in that rectangle which falls right

And finally, Microsoft did something that is not a steaming pile of shit. Mostly. About my VESA SVGA program... well... I discovered where the linear frame buffer thing is stored, I didn't test double buffering yet. I tried using the information I found on a book to draw in the screen but... not much luck. It does work but, for some reason, it does weird things like this when I try to draw a few pixels on the screen:



## MLG SVGA DRAWING STRATS

Like, it fills a memory bank with garbage. I can't say it doesn't work, it somehow does, but... not as it should. You can notice a few blue pixels after the red garbage, which is what I drew.

About the book I used as reference, I really, really recommend it to everyone interested in programming something outside of any common, easy to use SDK. It's called "The PC Graphics Handbook", by Julio Sanchez and Maria P. Canton (Two hispanic bros! yay!). This book has more than 1100 pages, and it talks about everything related to computers, and graphics, with nearly superb detailed information. From explaining Cathode ray tube screens, to Direct X. Even the Windows API, to make interfaces for your C++ Windows programs. You can read a small preview [here](#).

If you want to put your hands in one of these, be ready to pay about 160€... or maybe you can get it for free, on the internet. Like I have a full copy of a previous version, with 1008 pages, in PDF format. I don't remember where I got it, though, so you'll have to find it by yourself.

I've been browsing through my old blog entries, and I've found some interesting stuff, that I might translate (and maybe expand) into this blog. Among my blog entries, you can even find a complete C# tutorial. However, I'd like to leave this one aside, as it's too long, and it's not detailed enough. Most of the blog entries there, are about programming in general. From when I started using virtual computers and doing all short of C/C++ and Assembler. The most advanced articles include a boot loader, and another one that explains how to boot from a disc. I'd really like to talk about that stuff here, and also improve it with all the things I've been learning those past days, which is a lot of stuff.

Also, I found my first, and most old, "operating system development tutorial". That one is a complete joke I did to gain visitors and subscriptions, without really having any knowledge about what was I doing. Like using C standard libraries for "debugging instructions for your system", and saying thins like "The interruption vector list is used for debugging". Things I just made up back then to pretend I was understanding what I was doing.

I also have some tutorials about Virtual Machines, DOSBox, I even have an article that talks about a Game Boy development toolkit. Well, one day I will talk about these stuff. Oh, and don't worry if you don't understand any of these shit, but you want to do the "cool stuff". I always write my articles for everyone. So if you just want to do the cool stuff without understanding (which is bad, but hey... not everyone is as interested in the complex stuff as I am), you just have to follow the steps as I specify them.

Oh, and btw. My old blog is gone forever. I deleted it, as I stated in previous posts. I got tired of it, and I decided to start a new one in english, so everyone could understand it. Also, in a more personal way.

Pegado de <<http://ninjihaku.cu9.co/blog/2014/07/about-today/>>

# Creepypasta #1 – Superman 64

jueves, 24 de julio de 2014

11:42

## Creepypasta #1 – Superman 64

July 21, 2014 · by [Ninjihaku](#) · in [Uncategorized](#) · [Edit](#)

My name is Timmy, and I'm a 22 years old computer engineer student. I know most people is skeptic about this topic, but I can't hide it anymore. I need to talk about what happen to me the last week. Guys, you have to believe me on this, I swear I'm not bullshitting or anything! What you are about to read is what truly happen to me.

As I said, it all started about a week ago. While doing some clean up at my room in my apartment. I was cleaning in my closet, and I have to tell, it's a huge closet, so I had to enter inside when, suddenly, I found my old Nintendo 64! It also had a controller plugged in. Unfortunately, I didn't find any of my old games, so I didn't have anything to play on it.

I got out of the closet with my console in my arm, and put it over my desk. I really wanted to give it a try. You know, nostalgia and stuff. So, as I didn't have any games, I started looking on ebay an other second-hand pages. There was a lot of games and, unfortunately, I was low on cash. I decided to go on a walk while thinking about what game to buy, and take some fresh air after all the dust I had to clean.

I put some clothes on, and went outside. I was thinking so deeply in what games to buy that I started walking randomly. I soon realized I was in a street I've never walked before. And, guess what I found? A huge second-hand market was right in front of my nose, with tons of old hardware and stuff. Without thinking about it, I step in like an arrow stabbing a knee.

Inside I found a lot of amazing old stuff like really old UNIX machines, a Virtual Boy, a Sega Neptune and a Pluto, and a Nintendo 64 with the 64DD attached. And, of course, a whole self replete of games! Damn! They even had an old Video capturer for the N64! I remember playing with this in Mario Paint64, at a friend's house.

While I was looking at all these stuff, an old man approached to me and asked me, with a raspy voice:

- "Excuse me youngster, are you looking for something?"
- "Oh, sure! I was wondering if you had a Nintendo 64 game like... uhh..."

I thought for a moment what game could I ask for. The first game that came to my mind was my favorite of all.

- "By any chance, would you happen to have a copy of Superman 64?" – I asked to the old man.

The old man started thinking for a moment, then entered inside a door for a few seconds. A lot of noises came from the inside, until he came back with a huge box.

The box was a bit deteriorated, and had a sticker on it that said "SCP PROPERTY – CLASSIFIED REDACTED STUFF" on it. The old man noticed me looking at the sticker, and suddenly rotated the box. He opened the box, that was full of old games, also in a very deteriorated state. Some of them had the label broken, or didn't even have a label. I also noticed a black game boy cartridge with a sticker that said "Pokemon" on it.

I thought it had to be some pretty rare stuff like bootlegs or beta versions of the game. Maybe I was going to be a very lucky guy!

The old man, then, took out of a box a cartridge with a sticker that said "Superman" on it. It had no sticker, it was a normal n64 cartridge with a text written with an edding marker. The old man placed the cartridge over the self.

- "Excuse me, are you sure this is a Superman 64 cartridge?" – I asked to the old man.  
- "Sure, boy. I was waiting for someone to ask me for it. How could I forget about it?"  
- "Excuse me? What was what you said?"  
- "Oh! Nothing! It's nothing. Forget my mumbling, I am becoming old..."  
- "Uh... okay. How much for it?"  
- "Oh! Nothing! You can take it for free!"

I couldn't believe it! I was about to get a free copy of Superman 64! My favorite! But no, wait, how could that be possible?

- "F-For free!? Wow! Why? Is it defective?"  
- "Absolutely not! Nothing in my shop is defective."  
- "Then?"  
- "Are you taking it or not?"

I doubt for a while, but finally, I decided to take it and leave. As soon as I got outside, I saw through the glass door a strange man peeking outside of the door where the old man took the box out. Then, I saw the two of them enter the door again. I wanted to go inside again to take a look and see what was going on, but the door was closed all of a sudden.

- "Oh well, maybe I'm imagining things!" – I thought to myself.

Whatever. I got my favorite game and for free! So I went all my way back home. But the nightmare was only about to begin.

As soon as I arrived, I put the cartridge in, and turn the console power on. However, after placing the switch in the ON position, the screen was completely black. And there was no audio.

I tried taking the cartridge out and in, blowing it, cleaning it with a cue tip and a bit of glass cleaner... but nothing. The game refused to boot, until I realized something was going wrong.

The power cord was not plugged in the power socket.  
- "Oh crap!" I said to myself, "I'm an Idiot!".

I plugged the power cord into the power socket, then made sure the video connectors were also plugged in my TV. Then, I powered the console ON, and this time... it worked. The game booted, and the Titus logo appeared on the screen. Later, the main menu.

I started a new game. I played it so many times, that I still know most of it from memory. I went through all the rings, one by one, without missing a single one. Then, I had to prevent a cars from hitting a peasant.

The goal of this level was to lift the car, and place it somewhere else where the peasant could not get hit. However, for some reason, as much as I tried to grab it, I couldn't. I pressed all buttons, but nothing happen. Damn! Did I forget how to play this game?

I failed for the first time. Then, I had to go all the way back through the rings, and start all over. Again, I failed to grab the car. So I tried for the third time... with no luck.

After failing a mission usually there is the sound of Lex Luthor laughing, plus a game over screen telling you "Lex wins". However, after the third fail, this wasn't the case. Instead of Lex laughing at my fail, there was a very choppy voice that said "Fuck You!". And where it used to say "Lex Wins", now it says "You didn't save him. Here comes your punishment.".

Maybe the text could be from a beta version, but a voice saying "Fuck you"? I couldn't believe what I was seeing. I did not remember anything like this happening in the original game. And I have to admit, that got me more interested in the game.

I started the level for the fourth time, and of course, I had to fly through more rings. Usually when you start the ring stage, a text appears, saying “You have to solve my maze if you want to save your friends”. But this time, it says “You have to escape from my dungeon if you want to live”. Also, while flying through the rings, I noticed the path of the rings changed. Maybe this was a beta cartridge, and I glitched and started in another level?

I kept flying through rings... and then, more rings... and more rings... and more... and more... It didn't have an end! And I wasn't flying in circles, it looked like a random path. After half a hour of flying through rings, my precision was not accurate enough, and then I started missing rings. First one ring... then another... and another... Until, eventually, I ended up loosing. The game over screen appeared and, once again, the voice saying “Fuck you!”. The text, however, were different. It said “HEY BUDDY, I THINK YOU GOT THE WRONG PATH. THE LEATHER CLUB IS TWO RINGS DOWN”.

Okay, I have to admit that when I read that, a bit of crap came out of my anus and I couldn't hold it. I don't know what that means, but that couldn't be normal. I though this was a ROM hack or something. even if it were a beta, this was too much of a joke to be in a beta version of a professional game. It had to be a hack.

However, of course, I was too curious to stop playing now. I kept playing, wanting to see more. I pressed A to restart the level, but the screen went black for a few seconds. I swear I could hear a man screaming in the background, and then another voice that said something. I am pretty sure it said “You like that!?” . I couldn't hear it properly because of the compression, but I am pretty sure it was that.

The game booted again, and the Titus logo appeared again. Then the main menu, as usual. However, I have the feeling that instead of “NEW GAME”, it said “NEW COME”. I pressed the A button so quickly, that I didn't had time enough to read it. The intro started, but this time, all the characters were completely naked, and were male. And instead of Lex Luthor's lab, it was in a room that looks like some short of lockers or something.

The text in the intro also changed. It said “SHOULD YOU AND I SETTLE THIS HERE ON THE RING, TOUGH BOY?”. I pressed A, and the game started, this time, inside of the lockers room. I was facing Lex Luthor, and both of us were completely naked. I tried approaching Lex and using my super punch on him, but he dodged and grab me. He made a lock, and pin me down in the floor. The game over screen appeared, with a text that said: “LEX LUTHOR WINS. And the winner fucks the looser!”. This time, there was no retry button, so I could only press START to exit.

I pressed start, and a cut scene started. Superman was in a dark room. A light was coming from his head. He was fully naked, and tied with green chains (I guess it was crytonite). Then, suddenly, Lex Luthor appeared from the darkness, also naked. He grabs Superman's face with a very gently touch, and with the other hand, he squeezes his crotch. Then, he whispers to his ear...

“I'm going to cum all over you...”

“Okay, that's it! That's it! I'm done!” – I said out loud.

I jumped from my chair, shut down the console, and unplugged it from the socket. That last scene was frightening and too scary for me, I had to stop playing. My body was shivering and my hands were shaking, and for the very first in my life, I can say I was in true fear. But that's not all, the worst was about to come.

After all I went through, I went outside. I though that, maybe, it was a good idea to go back to the store, and ask for an explanation. But the streets, were not the same. As soon as I put my feet out, I noticed that everyone was a male. And not only that. Also, everyone... was naked.

I tried to run away as fast as I could, and find the street where I got this game. And I found the street, but where the second hand store was, is now a Gym. I entered the gym and approached the

reception. There was also a naked guy in the desk. I tried ignoring the fact that he was naked, and asked him about the old store that was here before.

-“What are you talking about? This has been a gym for 5 years, and as far as I know, this place was used as a storage before. Sir... are you all right?”

I nodded, then went outside again. I stopped at a park nearby and sat in a bench. I couldn't believe anything of what was happening. I was still trying to process everything when, suddenly, someone put his hands on my shoulder and whispered to my ear...

“Yaranaika?”

I jumped out of the bench, and started running away screaming in fear. I made it to my house, and hid in my closet. I've been here since then, using my laptop to write those lines, and coming out only when I need to grab food. I know you guys won't believe me, but you have to! It's the truth, I swear! I'm very scared, and worst of all, I know I can't hide here for much longer. I will have to get out soon!

Pegado de <<http://nинjihaku.cu9.co/blog/2014/07/creepypasta-1/>>

# BACKUP

domingo, 27 de julio de 2014  
23:21

[http://youtu.be/ITY\\_xfeDkxA](http://youtu.be/ITY_xfeDkxA)

This has been a trend for quite a long time already. Someone recording himself, playing a game. Why? Because people like let's play videos, or used to. You know, like most of these "Youtubers" that earn money just by recording themselves playing videos. Markiplier, Pewdipie, etc.

So they gain quite a lot of money, and fame, for doing something that everyone can do, playing videogames. And because playing videogames, is something that everyone likes. It's like... Easy money! right? But it's not.

Let me show you the real face of doing let's play, about why is these people so famous, and why you will never ever be successful at this, and why you should NOT do this under any circumstances, unless you know how to correctly do it.

<hr />

**1. What is a "Youtuber"?**

It's like a "despective" name for someone that upload videos to this online service called Youtube, with the goal of making money and become famous, spending little to no effort making their videos.

Many youtubers use this service as some kind of VLOG (Video Blog), or to upload recordings of their gameplays in different videogames.

The most famous Youtubers also have some kind of partnership with certain websites, for advertisement and also product rating (usually altered, confessed by some of these people).

<hr />

**2. What is a "Let's play" or a "Gameplay"?**

A "Let's play" is a series of videos with recorded gameplay of a certain game. This game does not have to be necessarily completed, and there is no need to accomplish a certain goal, other than show random footage of the game.

Many let's players focus on the Online feature of modern videogames such as Call of Duty series, Minecraft, League of Legends, and other modern trending video games.

<hr />

**3. Famous youtubers**

There is a ~~hall of shame~~ [Top 100 chart](http://socialblade.com/youtube/top/100) showing the most famous youtubers in Youtube. In this chart you can see an estimation on how much do they earn, in euros.

(Now please put that gun away from your head, and withdraw it).

<hr />

**4. Are "Youtubers" bad?**

No. It's quite natural for a person, to have fun with something, and wanting to share that moment with everyone.

Making so much money for something that is not worth it, is. Just so you realize the magnitude of this, the people that worked on building all the hardware they use to record their videos, have earned only a 1-2% of what PewDiePie earns in a month, just by playing a videogame and recording it on Youtube.

A standard salary in my country is not even a 1% of what this guy earns in a month.

<hr />

<strong>5. Why shouldn't I make let's play videos?</strong>

The reason is very simple. When a person joins the Youtube scene, is not to make money, nor to become famous. If you are willing to become a let's player for any of these two reasons, you will not succeed.

No money, nor fame. The reason why you want to join the Youtube scene is to <strong>entertain people</strong>.

Now, before you join Youtube and start uploading your crappy let's play videos, you should ask first. "How am I going to entertain people, just by uploading videos of me playing videogames?". Think about it, is it really that hard to play a video game, that everyone will go to your channel just so they can watch videogames instead of playing them? No!

People will only go to your channel if they think you are <strong>fun</strong>. Because videogames, are for <strong>fun</strong>. And people in the internet is constantly looking for <strong>fun</strong>. Let me say it one more time: <strong>fun</strong>. That's a keyword. No fun, no honey.

So let me ask you again, do you want to make people bored by watching crappy let's play about your COD killstreaks, when they could play them instead? NO.

So then, what can you do to entertain people? Well, it depends. I've found quite a few let's players that are not so well known as Please die pie, but are rather fun and enjoyable watching.

One of the examples is Kr1tical (aka penguinz0). He's a standard let's player, also trying to voice act. But he focus on showing funny stuff on his gameplays, rather than just playing the game. Also the way he talks, and how naturally things come from him, is really funny. This guy never fakes a laugh, and the very few times you hear him laughing, it sounds natural.

<http://youtu.be/4Ky46TxmBxs>

<http://youtu.be/g49u3C6ym5I>

Lowtax and Shmorky are two let's players that dedicate to find the worst possible "homemade" games, to make fun of them. They are also quite good at voice acting, and this can lead to some really funny moments.

[http://youtu.be/XTWu\\_sDekCI](http://youtu.be/XTWu_sDekCI)

<http://youtu.be/qyfvSs2Om20>

Vinesauce is not exactly a "Let's play" channel, but rather a community of streamers that also upload their videos to Youtube. Some of them upload videos of corruptions, gameplays and also funny

programs.

<http://youtu.be/s50m1gh7bms>

<http://youtu.be/ZF238BCdMts>

<hr />

<strong>6. So what's the formula for success?</strong>

1. Try to innovate. Do you want to make a Call of Duty let's play? Too bad, you are going to compete with thousands of people, and some of them are already more famous than you are.

2. Again, try to focus on being fun. You are here to entertain people.

3. Learn how to act like a professional. Or better yet, don't act. Just let every emotion come naturally. If you are emotionless, forget about this.

4. Learn to speak properly. Also if your voice does not qualify, then also forget about it.

5. Get proper hardware, and software. Avoid this:

<a href="http://i.imgur.com/rDEP4A4.jpg"></a>

<a href="http://i.imgur.com/YSLPv6Q.jpg"></a>

<http://youtu.be/NVmVr-4Y6M4>

<http://youtu.be/Zpcq6jrrYE>

Remember that if you have to play something on an emulator, <strong>there are many emulators that include an internal video recorder</strong>. Configure it properly, and then, <strong>-&gt; USE IT &lt;-</strong>.

6. Don't expect money, nor fame. Do it for fun.

# Más sobre el boot loader: Crear un CD de arranque

sábado, 12 de julio de 2014  
0:59

[May 22, 2014](#) [Leave a comment](#)

Ésta entrada es bastante interesante para todos aquellos que aún siguen interesados en el tema del sector de arranque y la creación de sistemas operativos, porque vamos a aprender ni más ni menos que a arrancar un sistema operativo desde un CD.

Lo primero de todo, éste proceso no sólo sirve para nuestros sistemas operativos, sino que podemos crear un live CD para cargar cualquier sistema operativo, siempre y cuando cumplamos con una serie de requisitos, por supuesto. Yo lo usé por ejemplo para hacer una imagen de CD de Windows XP a partir de una copia del mismo que tenía en una memoria USB.

Y lo segundo, y más interesante, nos vamos a quitar de un plumazo la limitación de los 512 dichosos bytes (Yay!). Y es que hacerlo de éste modo va a ser todo ventajas, porque gracias a ello también podremos cargar nuestro sistema en un ordenador moderno cualquiera (Adios Virtual Machines).

## ¿Qué necesitamos?

Para empezar, un boot loader compilado. Podeis ir a mi sección de tutoriales, o empezar por [aquí](#) directamente. Lo más importante es que ya no hay límite de 512 bytes, pero la imagen del boot loader debe ser igualmente de un tamaño un múltiplo de 512 (512 bytes, 1kB, 2kB, etc). Podeis modificar si queréis la línea “times” para ajustar el tamaño del cargador a lo que requirais.

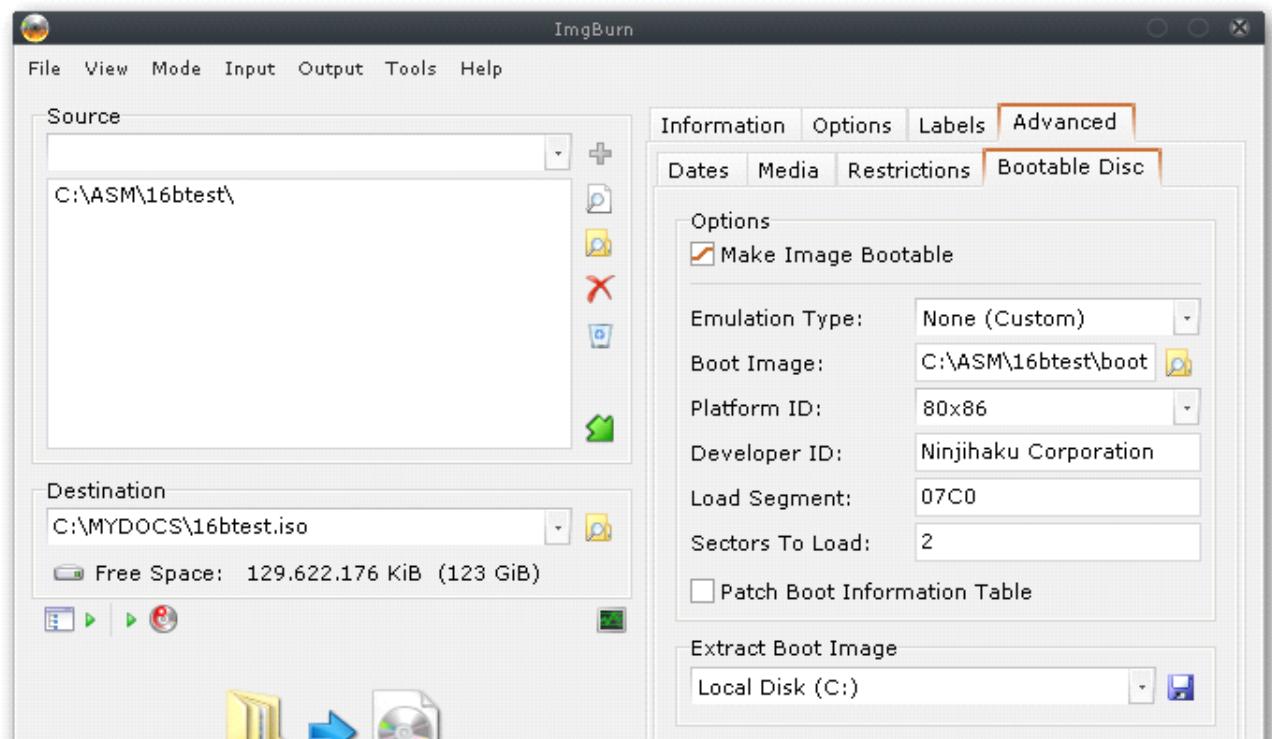
Por último, necesitamos el programa [ImgBurn](#). Éste programa lo parte a la hora de crear imágenes de CD, así que si no lo tenéis ya estais tardando en descargarlo. Ah, y es gratis. Así que podeis cerrar el utorrent.



## ¿Como cargo mi loader en una ISO?

Compila todos los archivos de tu SO, o el boot loader, y guardalo todo en una misma carpeta (puede haber subdirectorios dentro, por supuesto).

Ahora inicia ImgBurn, y pulsa el botón “Create image from files and folders”. La siguiente ventana aparece:





En Source, añadimos la carpeta de nuestro sistema operativo. Y en Destination, especificamos el directorio y nombre de la imagen ISO creada. Si no tenemos archivos en el sistema operativo, pon algún archivo dentro de la carpeta para no dejar la imagen vacía (aunque sea el bin del cargador).

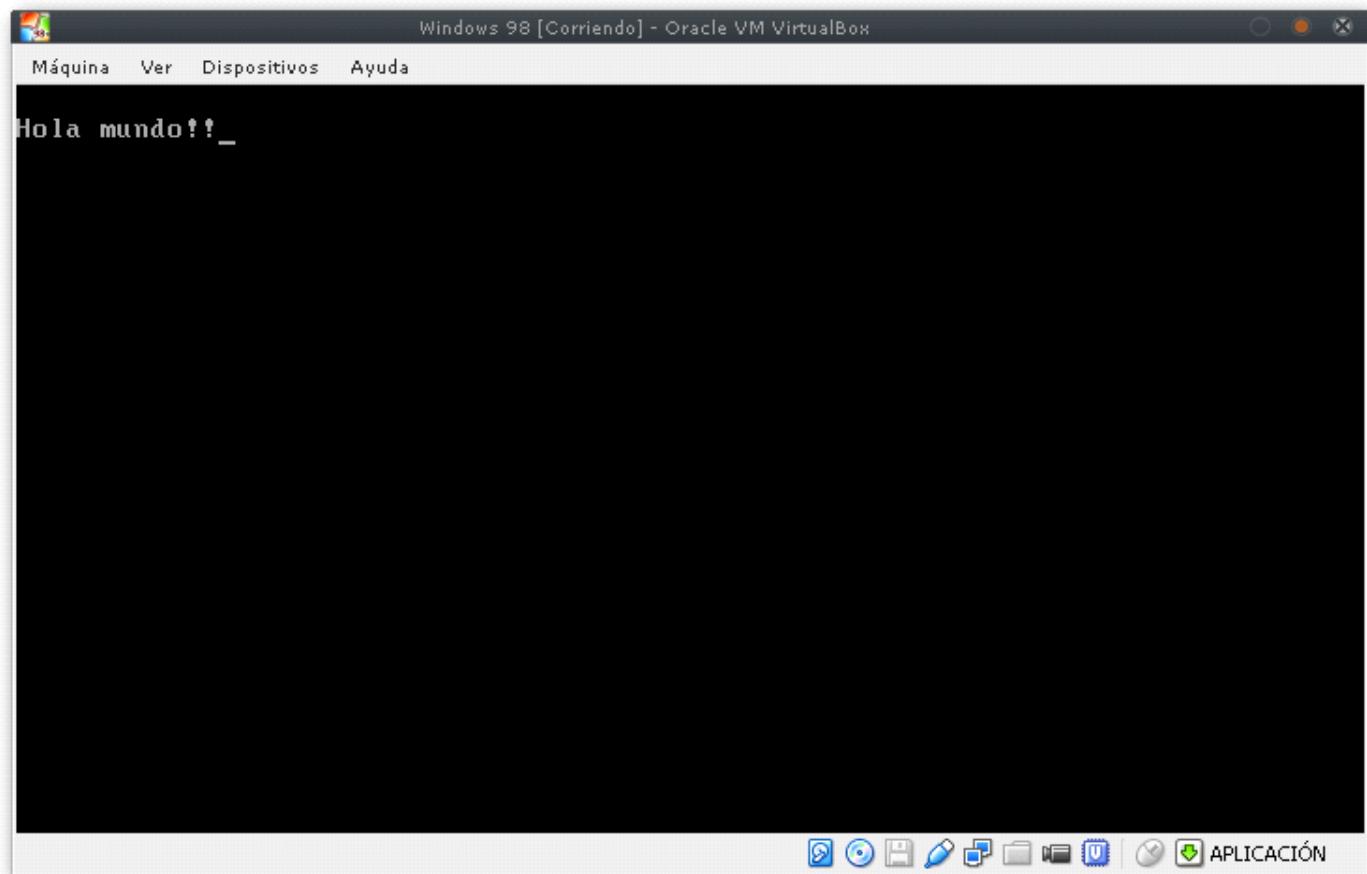
Ahora vamos al lío (Tranquilo, que tampoco es mucho). Primero hacemos click en la pestaña Options. En Data Type, seleccionamos MODE1/2048 (si no lo está), y en File System especificamos ISO9660 + Joliet.

Ahora vamos a la pestaña Advanced, y dentro, a la pestaña Bootable disc. Marcamos la opción "Make Image Bootable". Emulation type en None. En Boot Image cargamos el archivo .bin que hemos compilado con nuestro cargador. En Platform ID, 80x86 (para PC estandard). En Developer ID podeis poner vuestro nombre, si quereis. Y en Load Segment, 07C0 (el segmento de la memoria donde cargamos el boot loader, ya lo conoceis).

Ahora viene el número de sectores a cargar (Sectors To Load). Como ya he dicho, podemos superar los 512 bytes, pero siempre usando múltiplos de 512. Cada 512 bytes, es un sector. Por ejemplo, si nuestro cargador pesa 512 bytes, en Sectors To Load ponemos sólo uno (1), mientras que si pesa 1Kb (1024 bytes), son dos (2), etc.

Por último, pero no por ello menos importante, nos vamos a la pestaña Restrictions, dentro de Advanced, y dentro, a la pestaña ISO9660. Character Set debe estar en Standard, folder/file name length lo dejamos en principio en Level 1 (si ya has desarrollado el sistema, deberías saber lo que poner aquí, si no, dejalo en Level 1), y abajo marcamos las opciones: Allow more than 8 directory levels, allow more than 255 characters in path, y allow files without extensions.

Cuando ya lo tengamos todo, pulsamos el botón Build. Le decimos que sí a todos los mensajes de confirmación (Nos pedirá un nombre para el disco), y listo. Ya tenemos nuestra imagen preparada para grabarla, o para cargarla en un ordenador virtual, o lo que queramos.



Aquí tenéis una [ISO de ejemplo](#). Los archivos del cargador están dentro de la ISO. Si no queréis grabarla (y lógicamente, no querreis), podeis usar WinRAR para extraer los ficheros que hay dentro.

Pegado de <<http://192.168.75.128/wordpress/>>

# Boot Loader

sábado, 12 de julio de 2014  
1:03

## Crear un Boot loader para comenzar tu propio sistema operativo

[August 31, 2010](#) [Ninjihaku](#) [22 Comments](#) [Edit](#)

¿Recordais el [tutorial sobre creación de sistemas operativos](#) que creé hace mucho tiempo? Este tutorial tiene varios inconvenientes que le he encontrado tras analizarlo detenidamente:

1. No explica adecuadamente el proceso.
2. Debe de haber algun error, ya que no a todo el mundo le funcionaba.

Recientemente he encontrado un “nuevo” método (no es nuevo, más bien, uno más adecuado) para crear el boot loader (tambien llamado bootstraper o bootstrap loader) para nuestro sistema operativo. Y no necesito 3 partes para explicarlo, aunque si sería conveniente que fueseis leyendo las otras 3 partes del antiguo tutorial para entrar en materia con los conceptos básicos.

El boot loader es un pequeño programa, de 512 bytes (justos, ni uno más ni uno menos) que se graba en lo que viene a ser el sector de arranque del disco. La BIOS busca éste cargador en los dispositivos de arranque al iniciar el ordenador para así arrancar el sistema operativo del ordenador. Esto es algo que, bien porque en aquél entonces no lo tenía muy claro o bien porque se me pasó por alto, no lo expliqué en ese manual. El programa tiene un formato binario “plano”, se carga en la dirección 7Ch de la memoria que es la dirección de memoria física correspondiente al cargador, y tiene que acabar con la palabra “AA55h” para que la BIOS lo reconozca como un cargador.

El proceso de creación de éste cargador es mucho más sencillo de lo que parece, pero siempre y cuando sepamos y tengamos claro lo que vamos a hacer y estamos haciendo. Programarlo es muy sencillo, necesitamos un lenguaje del más bajo nivel posible, en éste caso en ensamblador.

Seguramente también se pueda utilizar C, aunque seguramente sea un poco más complicado ya que hay que tener en cuenta otras cosas a la hora de compilarlo. El proceso se puede realizar desde cualquier sistema operativo, yo por ejemplo lo he hecho en Windows utilizando el ensamblador NASM.

También (aunque con otra sintaxis de código) se podrá realizar con el propio DEBUG.EXE de windows, que viene incluido en el SO desde tiempos inmemoriables. En Linux se puede también, incluso quizá un poco más sencillo que en Windows.

La idea es crear un bootloader que cargue un texto, tal y como se describe en el tutorial que escribí hace mucho tiempo. Bien, vamos a comenzar por obtener las herramientas necesarias:

### PARA WINDOWS

NASM para Win32 ([Descargar](#)) (NOTA: Solo hay versión de 32 bits).

PartCopy ([Descargar](#))

QEmu para Win32 ([Descargar](#)) O la versión de 64 bits ([Descargar](#))

Notepad++ ([Descargar](#))

### PARA LINUX

NASM (prueba este comando: sudo apt-get install nasm)

QEMU (prueba este comando: sudo apt-get install qemu)

NOTA: No es necesaria la interfaz gráfica del QEmu.

**ADVERTENCIA:** Nunca hagas pruebas en el ordenador personal, podrías dejar el sistema inoperativo y tener que formatear de nuevo.

Bien, tras prepararlo todo, empezaremos por crearemos una carpeta en “Mis Documentos”, o en donde resulte apropiado, donde guardaremos todos los documentos del código fuente y una copia del ejecutable “PartCopy”, por comodidad. Despues abrimos el Notepad++. Yo he puesto el Notepad++ porque es el que uso y el que más apropiado me parece, pero cualquier otro editor de textos con resaltado de código es válido, y si no, con el bloc de notas mismo. Vamos a escribir el siguiente código (RECOMIENDO escribirlo en lugar de copiarlo):

```
[BITS 16]      ; Directiva. Indica al compilador NASM que nuestro programa es un binario de 16 bits.
ORG 0          ; Directiva que indica que queremos comenzar el programa al principio de la memoria.
; Podemos iniciarla directamente en la direccion 7Ch y saltarnos lo siguiente, aunque mejor no hacerlo asi.
main:
; Ajustamos los registros para que vayan a la direccion 7Ch
cli
mov  ax, 0x07C0
mov  ds, ax
mov  es, ax
mov  fs, ax
mov  gs, ax
; Creamos una pila de llamada
mov  ax, 0x0000
mov  ss, ax
mov  sp, 0xFFFF
sti
mov si, msgTexto           ; Cargamos en el registro “si” el texto
call DisplayMessage         ; Llamamos a la subrutina para mostrar el texto
DisplayMessage:             ; Subrutina para mostrar un texto cargado en el
;registro “si”
lodsb                      ; cargamos el siguiente caracter de la cadena de caract.
or   al, al                ; comprobamos que no hay un caracter nulo, en cuyo caso...
jz   .DONE                 ; ... lo interpretamos como final de linea, por lo que termina la
subrutina.
mov  ah, 0x0E               ; Le pedimos a la BIOS que muestre el texto
mov  bh, 0x00               ; Página 0
mov  bl, 0x07               ; Atributo de texto
int  0x10                  ; Interrupcion 10: mostrar el texto en pantalla
jmp  DisplayMessage         ; Si no hemos terminado, repetimos
.DONE:
jmp Loop                   ; Si hemos terminado, creamos un bucle infinito para bloquear el
ordenador
Loop:
jmp Loop
msgTexto db 0x0D, 0x0A, "Hola mundo!!", 0x00 ; Nuestro mensaje
times 510-($-$) db 0
dw 0xAA55
```

El mensaje que está entre comillas puede ser reemplazado por nuestro propio mensaje. Tras ello, lo guardamos con formato .asm

Para compilarlo, creamos un archivo con extensión .bat llamado “compilar” (por ejemplo), y escribimos lo siguiente:

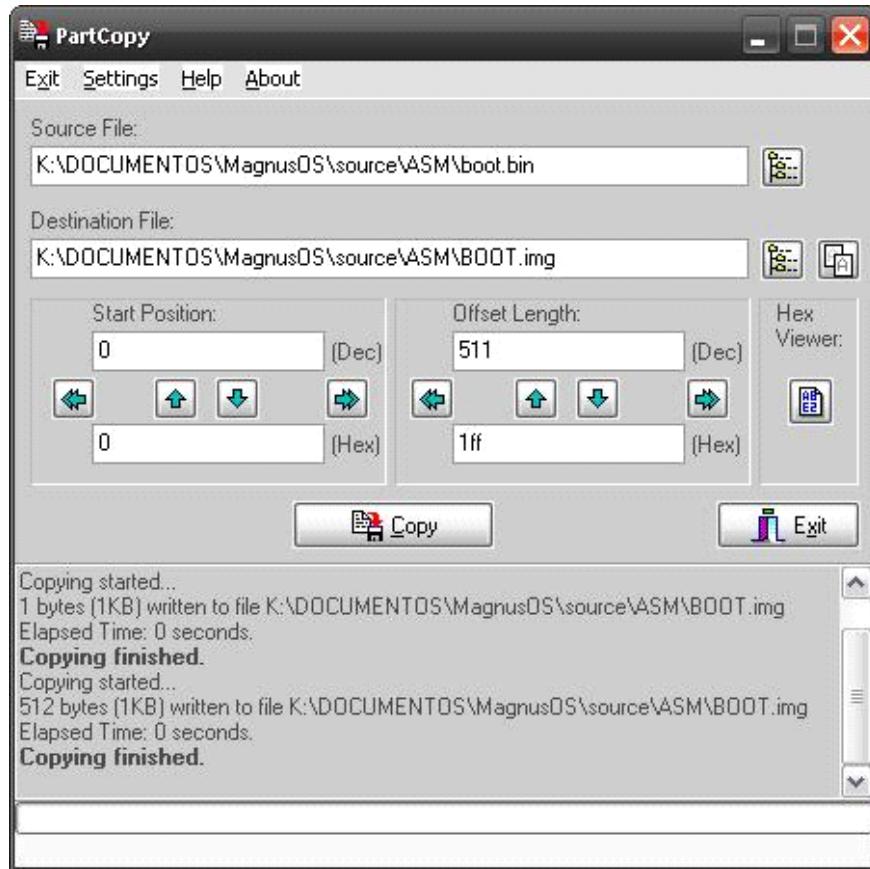
```
nasm bootstrap.asm -f bin -o boot.bin & pause
```

Igual para linux.

Tras ejecutar el archivo, y compilar, pausamos la salida para poder ver, en caso de que haya algún error o alguna advertencia, el mensaje. Si todo va bien, que debería ir bien, solo queda crear un

disco duro con el contenido del bootloader para QEmu. De esto se encargará el PartCopy, ya que copiaremos todos los bytes del archivo ensamblado (el boot.bin) a una imagen de disco duro para el QEmu.

Si abrimos el PartCopy, en la interfaz nos aparecen varias opciones.



Source file es el archivo de origen, en este caso el ensamblado que acabamos de crear. De él se copiarán los 512 bytes al “Destination file” o archivo de destino, que en este caso es la imagen del disco duro del QEmu (Boot.img). No es necesario copiarlo a un disco ya existente, lo podemos crear nuevo directamente desde el PartCopy. Ahora solo falta seleccionar los bytes que queremos copiar, y como lo queremos copiar entero, solamente pondremos en el Offset length el valor 511 (de 0 a 511 son 512 bytes en total). Una vez copiado, movemos la imagen a la carpeta del QEmu.

En la carpeta del QEmu tenemos un archivo llamado qemu-win.bat. Si lo abrimos para modificarlo, la última linea de todas aparecerá una linea parecida a ésta:

```
qemu.exe -L . -m 128 -hda algo.img -soundhw all -localtime -M isapc
```

Solamente cambiamos ese “algo.img” por el nombre de la imagen que acabamos de crear.

```
qemu.exe -L . -m 128 -hda BOOT.img -soundhw all -localtime -M isapc
```

Lo ejecutamos, y observamos los resultados.

```
■ QEMU
Plex86/Bochs VGABios current-cvs 14 Jun 2006
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

cirrus-compatible VGA is detected

QEMU BIOS - build: 11/01/06
$Revision: 1.174 $ $Date: 2006/10/17 16:48:05 $ 
Options: apmbios pcibios eltorito rombios32

ata0 master: QEMU HARDDISK ATA-7 Hard-Disk (0 MBytes)
ata1 master: QEMU CD-ROM ATAPI-4 CD-Rom/DVD-Rom

Booting from Hard Disk...
Hola mundo!!_
```

Si lo queremos hacer desde linux, utilizamos el siguiente comando desde la consola:

```
dd if=boot.bin bs=512 of=boot.img
```

Y ejecutamos el QEmu con la imagen del disco.

Funcionar, funciona claramente. [Aquí os dejo los archivos](#) asm, el ensamblado, la imagen de disco con el ensamblado, el partcopy que utilizo, y ademas otro archivo ASM del cual hablaré en otra entrada, ya que aún tengo que decir más cosas que no me da tiempo ahora a decir. Así que, nos vemos pronto.

Pegado de <<http://nijiserver/WORDPRESS/?p=279>>

# Emulador Nintendo DS

sábado, 12 de julio de 2014

1:08

[Electronica](#), [Informática](#), [Otras](#)

## Emulador Nintendo DS (DS, NO 3DS)

[January 29, 2014](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



**NINTENDO** DS™

Ya sabemos que la gente le está dando a la 3DS desde hace bastante tiempo. Pero hay mucha gente que está esperando un emulador para ésta consola. Y tengo buenas y malas noticias al mismo tiempo con respecto a ese tema. Aún no hay emulador, pero parece que en un par de años más o menos podría aparecer el primero. Quizá incluso algo menos, con suerte.

Así que no podemos jugar a ésta consola todavía desde nuestro ordenador. ¡Lástima!

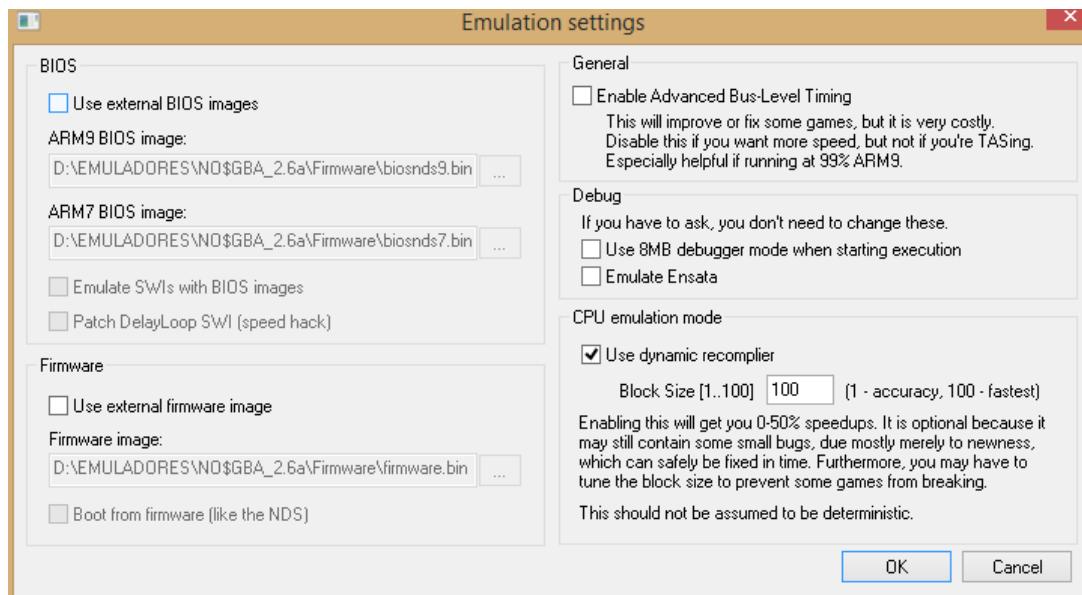
Tal vez otro año podamos jugar a la 3DS en nuestro ordenador, pero de momento podemos conformarnos con jugar a la DS. Eh, ¡Es algo!

Supongo que mucha gente se preguntará sobre las ventajas de jugar en un emulador frente a una consola física. Y más sabiendo que la DS se puede piratear muy fácilmente (tema el cual no voy a cubrir en mi blog, por supuesto). Voy a ser sincero, yo prefiero jugar a un juego siempre en la consola para la cual se ha diseñado. No es lo mismo jugar a cualquier juego en su consola, que en un ordenador que simula el funcionamiento de dicha consola (y en algunos casos, de forma un poco pobre). Pero los emuladores tienen algunos aspectos que añaden un poco de "libertad" a nuestro juego. Podemos jugar a juegos modificados, y no sólo eso, sino también desarrollar nuestros propios juegos, o explorar y modificar en la memoria (se puede físicamente con un dispositivo tipo GameShark o Game Genie, pero no es tan potente como editar la memoria en bruto). O incluso sacar una paleta con los sprites en tiempo real.

¡Bueno! Vayamos al grano. El emulador predilecto para la DS es el [DeSmuME](#). Actualmente en su versión 0.9.10, yo estoy en la 0.9.9 de 32 bits (si tu puedes con 64, adelante).

Tambien nos vendrá bien [ésto](#) si tenemos un mando SIXAXIS y estamos en Windows 7/8, o [ésto](#) si estamos en XP con el mismo mando. Si tienes otro mando, el de la XBOX o uno de ordenador genérico, configuralo como sea adecuado siguiendo el manual del fabricante o un tutorial adecuado. Cuando lo tengamos todo instalado, abrimos el DeSmuME [con permisos siempre de administración](#), y veremos una pantalla vertical en blanco. ¡Todo va bien! Vamos con la configuración, que es lo más importante.

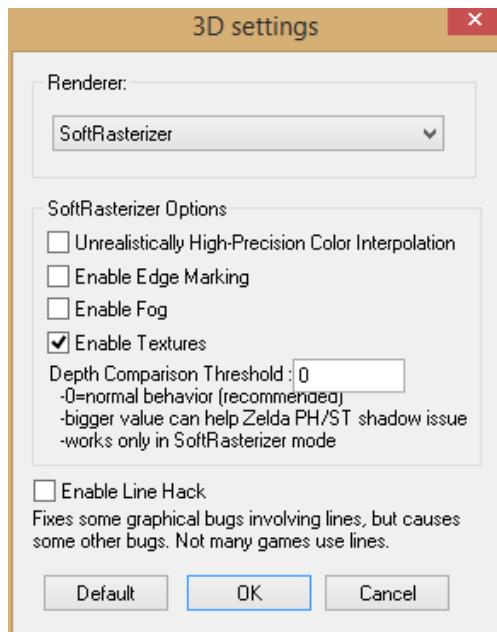
Vamos a Config > Emulation settings.



La configuración optima es ésta. A menos que queramos una emulación lo más cercana posible a la consola real, no usaremos una BIOS externa. Lo primero, no nos hace falta, y además hace las cargas más lentas.

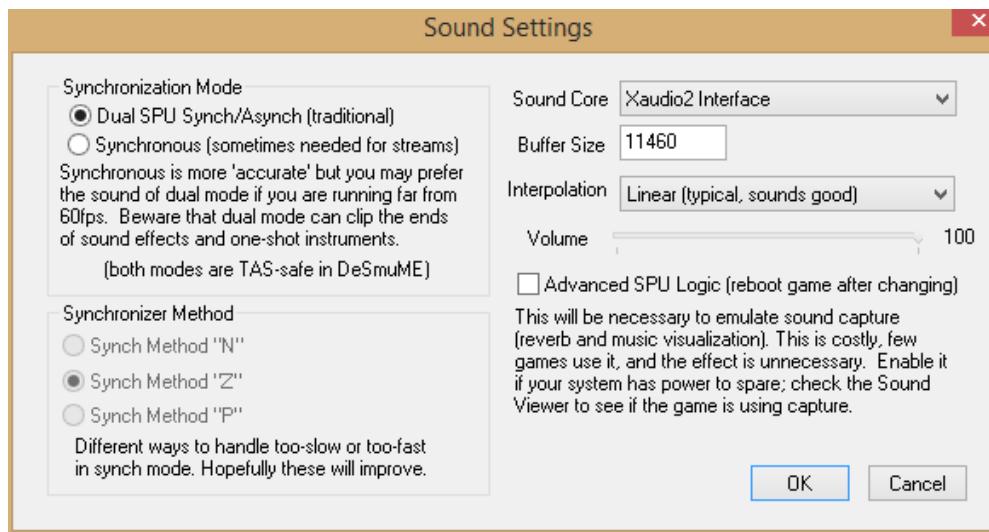
El bus avanzado lo dejamos deshabilitado, a menos que tengamos un ordenador lo suficientemente potente, ya que en caso contrario nos va a ralentizar mucho el juego.

Le damos a OK, y vamos a Config > 3D Settings.



El Renderizador lo dejaremos siempre en SoftRasterizer. Sé que muchos pensareis que usar Open GL o Direct X es mejor, pero la realidad es que no lo es, porque el emulador se aprovecha de las características del Soft Rasterizer. El resto de opciones, salvo las Texturas que debemos dejar siempre activado, podemos activarlas o desactivarlas en función del juego y de nuestro ordenador.

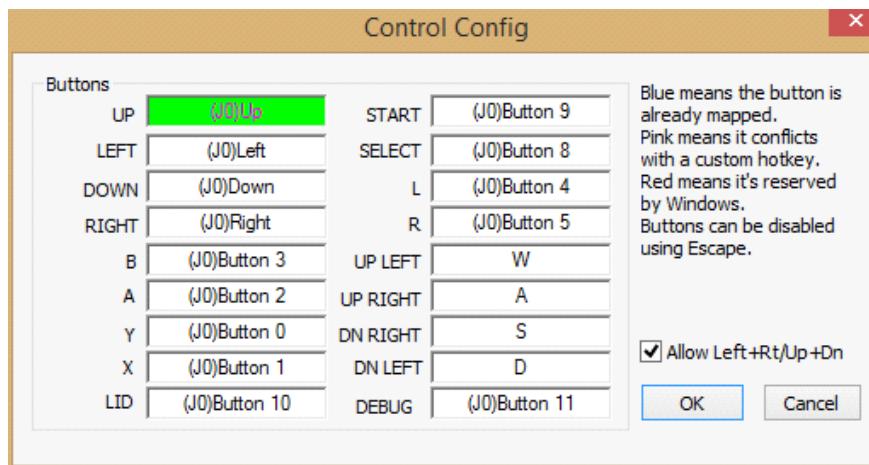
Le damos a OK, vamos a Config > Sound Settings



La configuración de sonido en éste emulador es muy importante. Si no lo configuramos adecuadamente, por un lado puede que se nos desincronize el audio (y habrá veces que igualmente lo hará), y por otro podemos perder “frames” durante la ejecución de un juego. Si vemos que se nos desincroniza el audio, podemos probar el modo Synchronous, y uno de los tres métodos de sincronización de audio. Le damos a OK.

Ahora hay unas cuantas cosejas más de las que ocuparse. Lo primero, en Config > Frame Skip, activamos la opción Limit Framerate, y colocamos el frame skip a 1. Y en Config > Display Method, recomiendo dejarlo en DirectDraw HW y activar la sincronización vertical (VSync), aunque ésta la podemos dejar desactivada perfectamente si no tenemos un ordenador muy en condiciones.

Los controles, se configuran en Config > Control config.



Hacéis click en UP, y vais pulsando los botones de vuestro mando secuencialmente, siguiendo el marcador verde, para asignar cada control a vuestro mando. Si no sabéis la disposición de los botones de la DS, aquí os dejo una referencia:



Cuando lo tengamos, le damos a OK, ¡Y ya estamos listos para empezar a jugar! Aunque, ¡Nos olvidamos de algo! Y es que, como todo buen emulador, nos hacen falta los ROM de los juegos.

Lamentablemente descargar ROMs de juegos se puede considerar piratería, e incluso hay juegos como Pokémon B/W y B2/W2 que tienen sistema de protección anticopia y hay que crackearlos con un editor hexadecimal. No obstante, internet es un lugar muy grande, y existe una página llamada Google que seguro que os puede ayudar a encontrar lo que necesiteis o querais buscar. Solo os diré que los roms van en la carpeta ROMS de vuestro emulador, obviamente.

Descargados los ROM, solo tenemos que ir a File > Open ROM y cargar el que nos apetezca jugar.

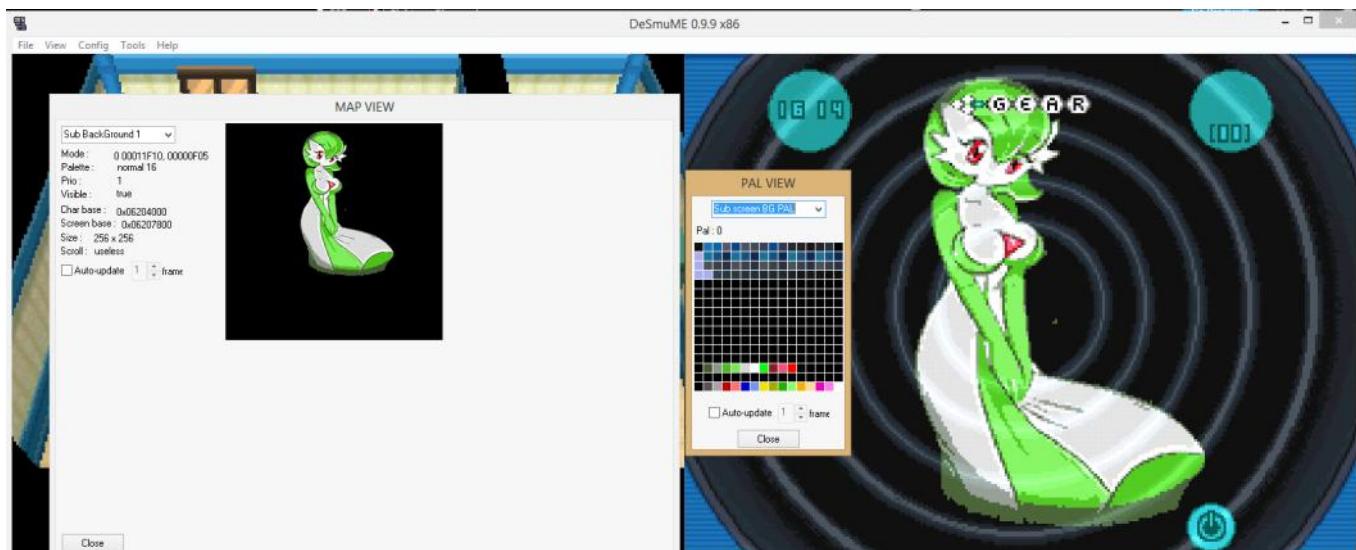


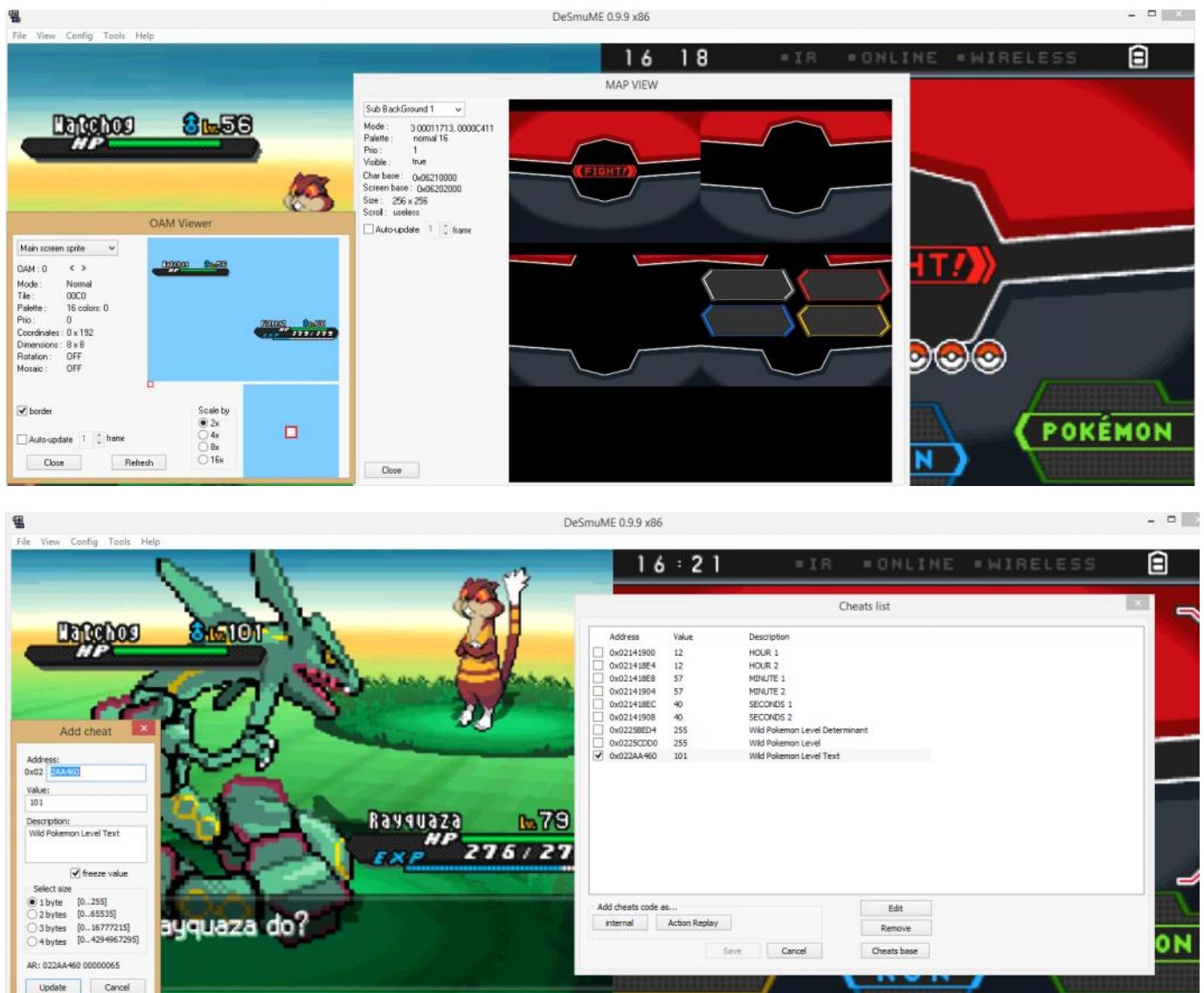


¡Todo listo y funcionando! Y además disponemos de todas las funciones de la consola, incluida pantalla táctil (usando el ratón), micrófono, etc. Los juegos no van 100% perfectos y a veces muestran errores leves, sobre todo en los gráficos, pero funciona bien y con una calidad bastante sorprendente incluso en un Core 2 Duo con 2 Gb de RAM de la época de los picapiedra. Incluso con juegos en 3D. Sólo nos falta una característica que éste emulador no posee, y es el online. Aunque hablaré de eso después, porque hay gente que ha logrado superar esa limitación y jugar online (¡Yay!). Aunque no os hagais muchas ilusiones.

Supongo que al ver las capturas, habréis notado que yo tengo la disposición de las pantallas en horizontal, mientras que vosotros apuesto lo que queráis, a que las tenéis en vertical. ¿He ganado la apuesta? Entonces debéis saber que todo eso se configura desde el menú View > LCDs Layout. También podeis ajustar el tamaño de las ventanas en View > Windows Size. En View > Magnification Filter, podeis añadir un filtro para suavizar un poco los pixels después de aumentar la pantalla. Scanline da un efecto de pantalla LCD bastante interesante.

¿Recordáis que dije que con los emuladores se pueden hacer cosas como modificar la memoria en bruto, y sacar los sprites en tiempo real? En el menú Tools tenéis muchas de esas herramientas. View Memory te permite ver la memoria, pero para modificarla se hace desde Cheats. Si le dais a Search, podeis buscar valores de memoria, y si sabéis manejar programas tipo Cheat Engine, el funcionamiento es muy similar.





Una última cosa que debeis conocer sobre éste emulador, y que cualquier emulador que se precie lo tiene, son los save states.

Los save states nos permiten hacer una copia de la memoria en un instante dado, para despues poder volver a ese estado. Es decir, podemos guardar la partida en cualquier momento pulsando Shift + F1, y si luego nos equivocamos, podemos volver a dicho momento anterior pulsando F1. Podemos guardar hasta 10 estados distintos, desde F1 a F10. Esto no quiere decir que sólo podamos guardar diez estados (podemos sobreescribir cualquiera), pero que podemos tener simultaneamente hasta 10 estados guardados. Ésto también se considera hacer trampa y normalmente sólo se usa en TAS, pero es muy interesante conocerlo.

Una última característica que es muy importante conocer, es la capturadora interna del emulador. Es decir, podemos grabar video y audio internamente desde el emulador, sin necesidad de usar programas como Camtasia o FRAPS. Aunque igualmente vamos a tener una caida de FPS, al menos no es tan intrusivo, y los archivos que genera son bastante livianos. Podemos activarlo desde File > Record AVI. El programa capturará las dos pantallas al mismo tiempo, y las renderizará en disposición vertical.

[youtube <http://www.youtube.com/watch?v=mLr4iErHgFg>]

[youtube <http://www.youtube.com/watch?v=y-W-zxQNAeg>]

Tambien tenemos la opción de grabar nuestros comandos usando la función Record Movie, y por igual la opción de reproducirlos usando Play Movie. Ésta característica se usa sobre todo en TAS.

Ya por último, me queda hablar del tema del online.

Por defecto, DeSmuME no incluye capacidad online. Y los propios desarrolladores han dicho, que no la van a incorporar. Dicen que si lo hacen, Nintendo puede tomar acciones contra los desarrolladores de emuladores. En cualquier caso, existe una versión no oficial modificada llamada DeSmuME WiFi. No voy a explicar cómo se usa ni nada por el estilo, porque el tema tiene bastante tela (teneis un tutorial muy extenso [aquí](#)), y porque aparte, no a todo el mundo le funciona. Supuestamente hay algunos router que no soportan la forma en que Winpcap administra los paquetes de datos, o algo

así. Y yo, desafortunadamente, soy una de esas personas a las que no le funciona (o al menos en su día, no me funcionó).

En cualquier caso, salvo que tengais algún colega que quiera jugar con vosotros, no merece mucho la pena.

Pegado de <<http://nинjiserver/WORDPRESS/?p=979>>

## Image Board

sábado, 12 de julio de 2014

1:11

## **Creación de un imageboard (Tablón de imágenes)**

[January 13, 2014](#) [Ninjihaku](#) [2 Comments](#) [Edit](#)



Los tablones de imágenes son una categoría de foros, comúnmente conocidos por [4chan](#). No obstante, éste tipo de foros se crearon hace muchos años en Japón, siendo [2chan](#) uno de los más conocidos (Y de donde surgió 4chan).

Normalmente la temática central de éstos foros suele ser la cultura asiática (el manga y la animación japonesa (vulgarmente denominado “anime”), la cultura pop de los países asiáticos, etc.). De ahí la denominación “[chan](#)” de éstas páginas (que en Japones, es un diminutivo que hace referencia a una niña pequeña). No obstante, un tablón de imágenes se puede usar para cualquier cosa que se pueda decir con imágenes (fotografía, CG, DIY, etc), e incluso éstas páginas incorporan tableros para temas que no requieren de imágenes, como matemáticas o programación. Incluso no todos los tableros son de imágenes, también los hay de sólo texto.

Los tablones de imágenes categorizan sus secciones por “tablones” o “tableros” (boards). Cada tablón se denomina con unas siglas entre dos barras inclinadas, un nombre, y una descripción. Por ejemplo, uno de los tablones más famosos y polémicos de 4chan sería /b/, denominado “Random”, y cuya descripción actual es “The stories and information posted here are artistic works of fiction and falsehood. Only a fool would take anything posted here as fact.”.

Aunque pongo de ejemplo 4chan al ser el más extendido, tampoco conviene tomar 4chan como un ejemplo a seguir. Su comunidad está nadando en una piscina de polémica y controversia por contenido inadecuado, y por ello no lo considero un buen ejemplo a seguir. Es mejor que impongas tú tus propios criterios a la hora de desarrollar un foro de imágenes, y que lo hagas siempre ateniéndote a las leyes internacionales, y nacionales. No querras ver a los “hombres de negro” pegandote una visita.



## ¿QUÉ VOY A NECESITAR?

Lo primero de todo, y si no lo has hecho ya, hecha un vistazo primero a [ésta entrada](#). Es conveniente crear un entorno de desarrollo web privado primero, antes de hacer nada de cara al público, para poder experimentar con seguridad.

Lo segundo, cuando vayamos a trabajar de cara al público, necesitaremos un buen web host. Y cuando digo uno bueno, me refiero a uno buenísimo, que no limite el ancho de banda, y que tenga muchísima capacidad de almacenamiento. 4chan por ejemplo, alberga alrededor de 200GB diarios de contenido, en imágenes. Multiplica eso por los millones de usuarios que accederán a dicha página a menudo y que estará constantemente refreshando el navegador en busca de contenido nuevo. El tráfico de datos es bestial. No obstante, si es una comunidad muy pequeña, no tendrás que preocuparte tanto por eso.

Lo tercero, el software del tablero de imágenes. Actualmente no existe mucho software “público”, pero sí existen algunas opciones como [Tinyboard](#). Tendréis que bajaros todo el código (El source tarball).

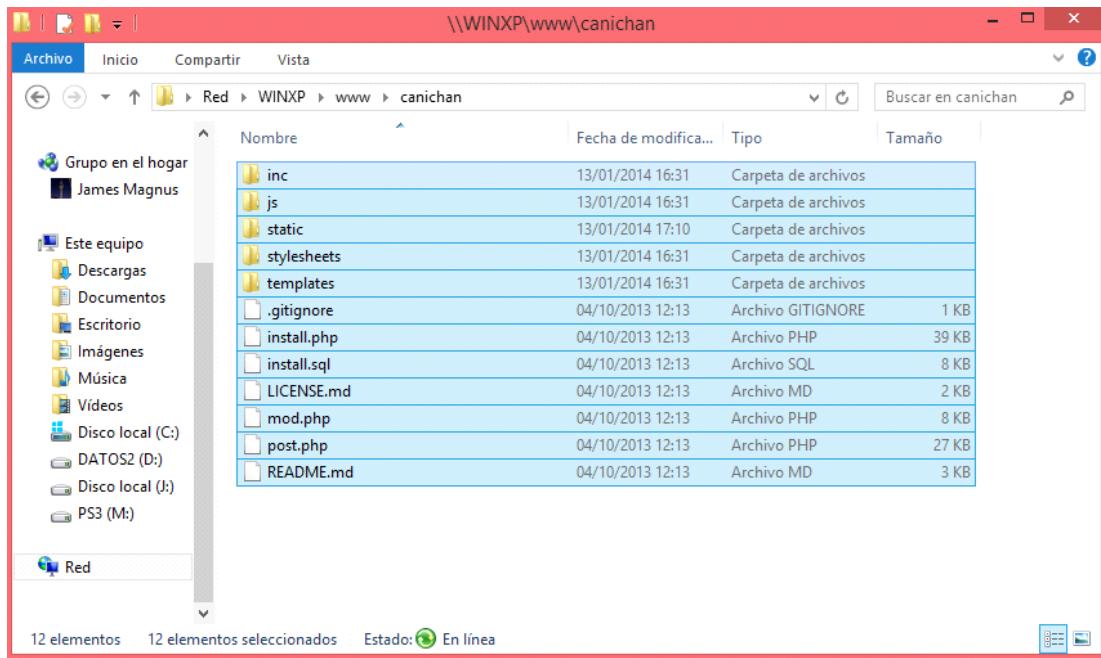
Por último, necesitaremos un gestor de FTP para subir todo al servidor web remoto (salvo en el entorno de desarrollo web, en donde usamos carpetas compartidas en red). Prueba [FireFTP](#).

Una recomendación es tener tambien a mano un editor de texto con sintaxis tipo [Notepad++](#). Al ser de código abierto (el tablero), podemos modificar e incluso añadir características, si tenemos los conocimientos adecuados.

## PASO 1 – INSTALAR EL SOFTWARE

Si usas un servidor externo, lo mejor es que sigas el paso 3 de [éste tutorial](#) que ya escribí. Hay que crear una base de datos MySQL, y subir los archivos por FTP. En el caso de un entorno de desarrollo privado, sólo crea la base de datos MySQL usando el panel PhpMyAdmin con el usuario “root” (sin contraseña), y sube el código del foro a la carpeta www. Conviene crear una carpeta aparte, llámala por el nombre de tu foro (Yo lo voy a llamar “canichan”).

Sigas los pasos que sigas, al final tendrás una estructura como ésta:



Vamos al navegador, y abrimos del directorio donde hemos subido el software del foro, el archivo “install.php”. En mi caso estoy usando una red privada, sería <http://ipdelwebhost/nombre/install.php>. En el caso de un host externo, <http://dominiodelhost/nombre/install.php>.



Lee el acuerdo de usuario, y aceptalo si estás conforme clickando en “I have read and understood the agreeemen. Proceed to installation” (sé que nadie lo hace, pero es lo que se debería de hacer). En la siguiente página, comprobará nuestro entorno. Si no hay errores (0 errors), podremos continuar con la instalación.

## Checking environment

### Pre-installation tests

Category	Test	Result
PHP	PHP ≥ 5.2.5	✓
PHP	PHP ≥ 5.3	✓
PHP	mbstring extension installed	✓
Database	PDO extension installed	✓
Database	MySQL PDO driver installed	✓
Image processing	GD extension installed	✓
Image processing	GD: JPEG	✓
Image processing	GD: PNG	✓
Image processing	GD: GIF	✓
Image processing	Imagick extension installed	⚠
Image processing	'convert' (command-line ImageMagick)	⚠
Image processing	'identify' (command-line ImageMagick)	⚠
Image processing	'gm' (command-line GraphicsMagick)	⚠
Image processing	' gifsicle' (command-line animated GIF thumbnailing)	⚠
File permissions	C:\wamp\www\canichan	✓
File permissions	C:\wamp\www\canichan\templates\cache	✓
File permissions	C:\wamp\www\canichan\inc\instance-config.php	✓
Misc	Caching available (APC, XCache, Memcached or Redis)	⚠
Misc	Tinyboard installed using git	⚠

There were 0 error(s) and 7 warning(s).

Las alertas son porque nos faltan algunas de las extensiones opcionales. Podemos instalarlas aparte antes de instalar el foro, pero son opcionales y no nos hacen falta. Con tener PHP, MySQL, y GD, basta. Además de los permisos de escritura en el directorio de instalación.

El siguiente paso es indicar los datos de la base de datos MySQL donde vamos a instalar el foro, y la configuración global del mismo.

## Configuration

### Database (MySQL)

Server:

192.168.64.131

Database:

test

Table prefix (optional):

cchan\_

Username:

root

Password:

The following is all later configurable. For more options, edit your configuration file.

Lo más importante es la base de datos, el resto lo podemos dejar por defecto. En servidor, si no sabes que poner, dejalo mejor como 'localhost' (Yo he puesto en la imagen la IP del webhost, pero no es correcto). Si localhost no funciona, mira a ver cual es el host donde están alojadas las bases de datos en tu servidor externo. Todos los datos te los administra tu host, salvo el nombre de la base de datos que lo creas tú.

Si todo es correcto, recibiremos un mensaje esperanzador, indicandonos que todo ha ido bien.

## Installation complete

Thank you for using Tinyboard. Please remember to report any bugs you discover. [How do I edit the config files?](#)

Por seguridad, se borrará el archivo "install.php" de nuestro directorio. En caso de querer

reinstalarlo, hay que borrar el archivo “.installed”, y añadir de nuevo el archivo “install.php” en el directorio del foro.

## PASO 2 – ADMINISTRACIÓN DEL FORO

Vamos al archivo “mod.php”. Nos pedirá unas credenciales de usuario. En nombre de usuario introducimos, sin comillas, “admin”, y en la contraseña, “password”. Hacemos click en LOGIN, y ya estamos dentro del panel de administración.

The screenshot shows the Tinyboard dashboard with the title "Dashboard" at the top. Below it, there are several sections:

- Boards**: Includes a link to "/b/" - Random [edit] and a link to Create new board.
- Messages**: Includes links to View all noticeboard entries, News, and PM inbox (0 unread).
- Administration**: Includes links to Report queue (0), Ban list, Manage users, Manage themes, Moderation log, Rebuild, and Configuration.
- Search**: A search bar with a placeholder "Phrase:" and dropdown options for Posts and Search.
- User account**: Includes a link to Logout.

Lo primero y más importante, es cambiar la contraseña del usuario “admin”, por algo más seguro. Ahora mismo cualquiera puede entrar en el panel de administración de nuestro tablero usando las credenciales por defecto. Hacemos click en “Manage users”. En el usuario “admin”, hacemos click en “[EDIT]”. Introducimos una nueva contraseña con cuidado de no confundirnos, y le damos a “Save Changes”.

The screenshot shows the "Manage users (1)" page. It includes a "Return to dashboard" link and a table with the following data:

ID	Username	Type	Boards	Last action	...
1	admin	Admin	all boards	0 seconds	▼ [PM] [edit] [log]

Below the table is a link to Create a new user. At the bottom of the page, there is a footer with the text "Powered by [Tinyboard](#) v0.9.6-dev-22 | [Tinyboard](#) Copyright © 2010-2013 Tinyboard Development Group".

Podemos también cambiar el nombre de usuario, si lo deseamos.

Hacemos click en “Return to dashboard”. Por defecto, el programa nos crea un tablero llamado /b/ – Random. Lo podemos borrar, o modificar si lo deseamos. Vamos a crear nosotros nuestro primer tablero, haciendo click en “Create new board”.

## New board

[Return to dashboard](#)

URI	/C /
Title	Cani
Subtitle	ER fOrO der prEmOH SurManooOh

[Create board](#)

Le damos a “Create board”. Nos mandará directamente a la página de nuestro nuevo tablero.

## /C/ - Cani

ER fOrO der prEmOH SurManooOh

[Return to dashboard](#)

Name	<input type="text"/>
Email	<input type="text"/>
Subject	<input type="text"/> <a href="#">New Topic</a>
Comment	<input type="text"/>
File	<a href="#">Examinar...</a> No se ha seleccionado ningún archivo.
Flags	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Password	<input type="password"/> (For file deletion.)

Has de saber, que todos los tableros que crees, tienen su propia carpeta dentro del directorio del foro, ademas de su propia tabla en la base de datos.

Red > WINXP > www > canichan					<a href="#">Buscar en canichan</a>
	Nombre	Fecha de modifica...	Tipo	Tamaño	
logar	b	13/01/2014 17:24	Carpeta de archivos		
jnus	C	13/01/2014 17:39	Carpeta de archivos		
os	inc	13/01/2014 17:10	Carpeta de archivos		
	js	13/01/2014 17:10	Carpeta de archivos		
	static	13/01/2014 17:10	Carpeta de archivos		
	stylesheets	13/01/2014 17:10	Carpeta de archivos		
	templates	13/01/2014 17:12	Carpeta de archivos		
(C:)	.gitignore	04/10/2013 12:13	Archivo GITIGNORE	1 KB	
I:)	.installed	13/01/2014 17:24	Archivo INSTALLED	1 KB	
(J:)	install.sql	04/10/2013 12:13	Archivo SQL	8 KB	
	LICENSE.md	04/10/2013 12:13	Archivo MD	2 KB	
	main.js	13/01/2014 17:24	Archivo JS	7 KB	
	mod.php	04/10/2013 12:13	Archivo PHP	8 KB	
	post.php	04/10/2013 12:13	Archivo PHP	27 KB	
	README.md	04/10/2013 12:13	Archivo MD	3 KB	

Tú no tienes por qué preocuparte por nada de ésto ya que el foro lo genera automático, pero sí saber que existe.

Todavía hay un par de cosas más que hay que saber, así que hacemos click en “Return to dashboard”.

Si hacemos click en “Configuration”, vamos a la configuración global del foro. Cabe destacar que aparte de la global, cada tablero tiene su propia configuración individual.

Any changes you make here will simply be appended to inc/instance-config.php. If you wish to make the most of Tinyboard's customizability, you can instead edit the file directly. This page is intended for making quick changes and for those who don't have a basic understanding of PHP code.			
Name	Value	Type	Description
global_message		string	Global announcement – the very simple version. This used to be wrongly named \$config['blotter'] (still exists as an alias).
check_updates	<input checked="" type="checkbox"/> Default: true	boolean	Automatically check if a newer version of Tinyboard is available when an administrator logs in.
check_updates_time	43200	integer	How often to check for updates
debug	<input type="checkbox"/> Default: false	boolean	Shows some extra information at the bottom of pages. Good for development/debugging.
verbose_errors	<input checked="" type="checkbox"/> Default: true	boolean	For development purposes. Displays (and "dies" on) all errors and warnings. Turn on with the above.
debug_explain	<input type="checkbox"/> Default: false	boolean	EXPLAIN all SQL queries (when in debug mode).
tmp	C:\WINDOWS\TEMP	string	Directory where temporary files will be created.
redirect_http	303	integer	The HTTP status code to use when redirecting. <a href="http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html">http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html</a> Can be either 303 "See Other" or 302 "Found". (303 is more correct but both should work.) There is really no reason for you to ever need to change this.
has_installed	installed	string	A tiny text file in the main directory indicating that the script has been ran and the board(s) have been generated. This keeps the script from querying the database and causing strain when not needed.
syslog	<input type="checkbox"/> Default: false	boolean	Use syslog() for logging all error messages and unauthorized login attempts.
dns_system	<input type="checkbox"/> Default: false	boolean	Use 'host' via shell_exec() to lookup hostnames, avoiding query timeouts. May not work on your system. Requires safe_mode to be disabled.
shell_path	/usr/local/bin	string	When executing most command-line tools (such as 'convert' for ImageMagick image processing), add this to the environment path (separated by :).
db → type	mysql	string	Database driver ( <a href="http://www.php.net/manual/en/pdo.drivers.php">http://www.php.net/manual/en/pdo.drivers.php</a> ) Only MySQL is supported by Tinyboard at the moment, sorry.

Son muchas configuraciones, y la mayoría conviene dejarlas por defecto. Las más importantes son:

- **global\_message**: Nos permite establecer un anuncio global, visible en todos los tableros.
- **cookies → mod**: Nos permite cambiar el nombre de las cookies para los moderadores. Es mejor poner algo que no sea lo de por defecto, por seguridad.
- **recaptcha**: Nos permite activar el “re-Captcha”, para evitar casos de spam masivos como el de [“Cornelia”](#).
- **flood\_time**: Nos permite ajustar el numero de segundos que un usuario debe esperar, antes de poder postear de nuevo.
- **max\_body**: Numero de caracteres máximo por mensaje.
- **reply\_limit**: El numero de respuestas máximo por tema. Al llegar a éste número, el tema no vuelve a la primera posición tras responder al mismo, hasta que acaba borrado.
- **spoiler\_images**: Permite a los usuarios ocultar sus imágenes en la página del tablero. Las imágenes son sólo accesibles si clickas sobre ellas.
- **country\_flags**: Permite ver la bandera de la nacionalidad del usuario al postear.
- **ban\_show\_post**: Muestra un mensaje en los posts, indicando que el usuario ha sido baneado por dicho post, si dicho usuario es baneado.
- **ban\_page\_extra**: Permite establecer una página personalizada para informar de los baneos a los usuarios baneados.
- **ban\_appeals**: Permite mandar un mensaje a la moderación, apelando a un ban. Solo una vez por usuario baneado.
- **ban\_appeals\_max**: Permite ajustar el número de apelaciones máximas que un usuario baneado puede realizar. Mejor dejarlo en 1.
- **thumb\_width**: Permite ajustar el ancho de las imágenes en los mensajes del tablero.
- **thumb\_height**: Lo mismo, pero con el alto.
- **thumb\_op\_width**: Lo mismo, pero para el primer mensaje de un tema.
- **max\_filesize**: Podemos establecer un tamaño máximo, en bytes, para las imágenes subidas. Si el archivo tiene un tamaño mayor, el software lo rechaza. Para calcularlo, recuerda que 1 Mb son 1024 Kb, y que 1Kb, son 1024 Bytes.
- **max\_width**: Ancho máximo de las imágenes subidas, en píxeles.

- **max\_height**: Alto máximo de las imágenes subidas.
- **threads\_per\_page**: Número máximo de temas por página. Al sobrepasarlo, se genera una nueva página.
- **max\_pages**: El número máximo de páginas. Al sobrepasarlo, se borra el último tema de la última página, junto al contenido asociado.

Conviene saber, que hay tres tipos de usuarios, que podemos crear para nuestro foro. Uno es el usuario "admin", que tiene todos los permisos, incluido el de crear tableros y borrarlos. El otro es el de "Mod", que tiene permisos completos de moderación sobre todos los tableros que se le asigne (incluido banear usuarios). El último es "Janitor", el cual puede borrar temas de los tableros que se le asignen. Todo ésto se puede modificar en el panel de control de usuarios.

## New user

[Return to dashboard](#)

Username	<input type="text"/>
Password	<input type="password"/>
Group	<input type="radio"/> Janitor <input type="radio"/> Mod <input type="radio"/> Admin
Boards	<input type="checkbox"/> "*" - All boards <input type="checkbox"/> /b/ - Random <input type="checkbox"/> /C/ - Cani
<input type="button" value="Create user"/>	

### PASO 3 – USO Y MODERACIÓN

Para usar el tablero de imágenes como un usuario cualquiera, vamos a cualquier tablero. En él, encontraremos el formulario para postear.

## /C/ - Cani

ER fOrO der prEmOH SurManooOh

[Return to dashboard](#)

Name	<input type="text"/>
Email	<input type="text"/>
Subject	<input type="text"/> <input type="button" value="New Topic"/>
Comment	<input type="text"/>
File	<input type="button" value="Examinar..."/> No se ha seleccionado ningún archivo.
Flags	<input type="checkbox"/> Sticky <input type="checkbox"/> Lock <input type="checkbox"/> Raw HTML
Password	<input type="password"/> (For file deletion.)

Por defecto, podemos dejar tanto el Nombre y el eMail en blanco. En “Subject”, podemos escribir una descripción para un nuevo tema, y en “Comment”, lo que es el cuerpo del mensaje. Por defecto, el primer post de un tema debe contener una imagen que cargaremos haciendo click en “Examinar”.

Los “Flags” son exclusivos de la moderación. “Sticky” hace que el tema quede siempre en la primera posición. “Lock” hace que nadie pueda responder a él. Y “Raw HTML” nos permite introducir HTML en el cuerpo del mensaje.

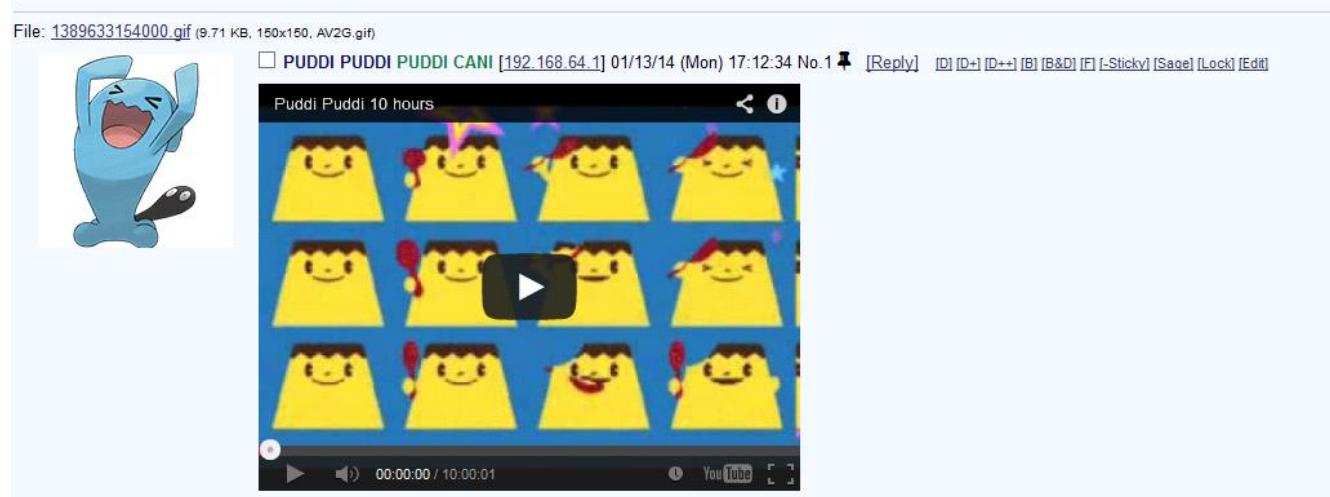
El campo Password nos permite añadir una contraseña, para borrarlo o modificarlo posteriormente. El campo Password está a disposición de todos los usuarios.

Al hacer click en “New topic”, crearemos el nuevo tema en el tablero.

**/C/ - Cani**  
ER fOrO der prEmOH SurManooOh

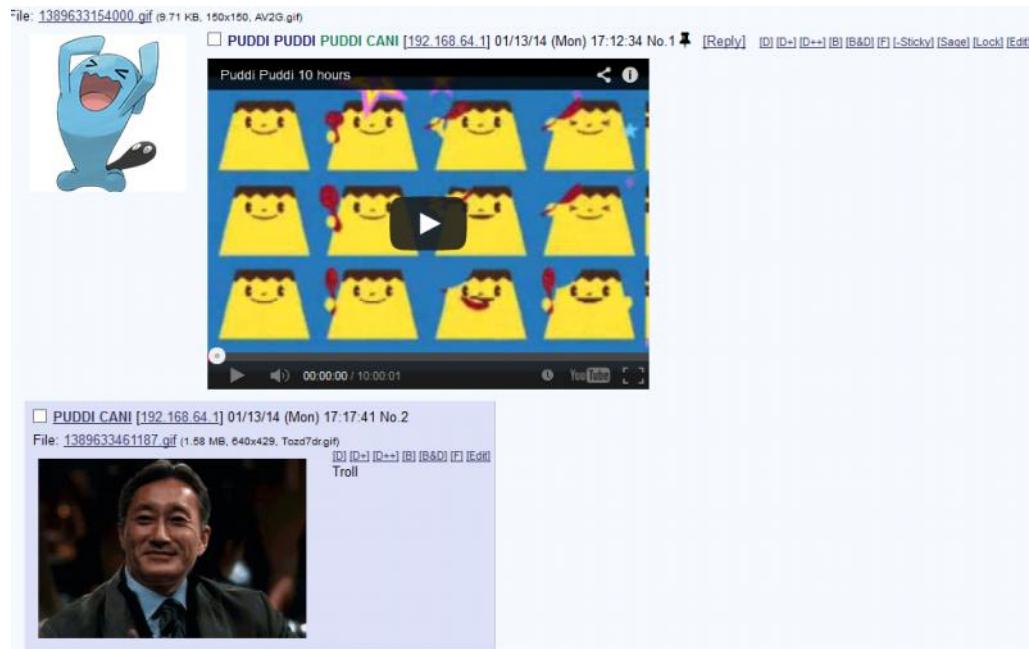
[Return to dashboard](#)

Name	PUDDI CANI
Email	
Subject	PUDDI PUDDI
Comment	<pre>&lt;iframe width="420" height="315" src="//www.youtube.com/embed/7LKHpM1UeDA" frameborder="0" allowfullscreen&gt;&lt;/iframe&gt;</pre>
File	Examinar... AV2G.gif
Flags	<input checked="" type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>
Password	•••••••• (For file deletion.)



Los botones [D] [D+] [D++] [B] [B&D] [F] [-Sticky] [Sage] [Lock] [Edit] solo se muestran a los moderadores y administradores. Nos permiten borrar los mensajes de un determinado usuario, banearle, cerrar el tema, editarlos, etc. Si colocas el cursor encima de cada botón, te mostrará una descripción de lo que hace.

Podemos responder a un tema haciendo click en el botón [Reply] del mismo. A diferencia del primer post de un tema, no es obligatorio poner una imagen. También debemos hacer click en Reply, para ver un árbol con todos los mensajes posteados en el tema.  
Los mensajes dentro de cada tema, se mostrarán en una estructura anidada.



Podemos introducir una serie de comandos especiales en el campo “email”. Podeis ver más detalles [aquí](#). Los usuarios pueden también usar “tripcodes”, que sirven como identificador. Más información, [aquí](#).

¡Y ya está! Esto es todo lo básico que hay que saber.

Pegado de <<http://nинjiserver/WORDPRESS/?p=971>>

# Entorno de desarrollo Web

sábado, 12 de julio de 2014

1:13

## Como montar un entorno de desarrollo web

[January 13, 2014](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



Desarrollar una página web, puede ser una tarea extremadamente sencilla, o extremadamente compleja, en función de lo que queramos hacer. Por ejemplo, podemos crear una simple página en HTML básico, o podemos montarnos un portal completo usando PHP y SQL, además de HTML 5, CSS, etc. Por supuesto, siempre podemos descargarnos todo ya hecho, pero aun así querremos tener un sitio donde poder experimentar con ese software sin riesgos ni para los datos de la web almacenados en el servidor, ni para la seguridad de la misma.

En ésta entrada, os voy a enseñar a montaros un entorno relativamente “seguro” de desarrollo web. No es nada complejo, es sencillo, y además podréis experimentar con total seguridad, sin necesidad tampoco de exponer vuestro ordenador a riesgos innecesarios, ni cargarlo más de lo necesario. También nos ahorrará los inconvenientes y limitaciones que pueda tener hacer ésto mismo en un servicio de alojamiento web remoto, especialmente en los gratuitos.

### ¿QUÉ VAMOS A NECESITAR?

Como no vamos a usar nuestro ordenador “tal cual”, hay dos alternativas. La primera, es tener un ordenador “de sobra” con al menos Windows XP, o un sistema Linux adecuado, y una tarjeta de red Ethernet. Solo lo vamos a usar como un servidor en red local para realizar nuestras pruebas, así que tampoco nos hace falta un “monstruo”.

La segunda alternativa, es tener un ordenador virtual (Virtual Machine). Y para ésto, os presento dos alternativas (de las varias que hay). Una posibilidad es con [VMWare Workstation](#). Es de pago, y cuesta algo caro, pero es muy bueno. No obstante, siempre tenemos una alternativa gratuita (y bastante similar), llamada [Virtual Box](#). Es también bastante bueno, y tiene las mismas características, e incluso posee algunos ajustes extra (Excepto en cuanto a redes). No obstante, y como lo que queremos es crear una red, considero mejor opción VMWare.

También necesitaremos un sistema operativo Windows (XP o superior), o Linux. En caso de

Windows, recomiendo usar XP, o una versión “Server”. En caso de Linux, Ubuntu, o una distribución basada en Debian que nos permita instalar y usar LAMP, y configurar una red local.

Necesitaremos también el software del servidor, el cual vamos a instalar en el otro ordenador (o la máquina virtual). Para no rompernos la cabeza, instalaremos [WAMP](#) en Windows, o [LAMP](#) en Linux. Ambos son exactamente lo mismo, un paquete con Apache (servidor web), y las extensiones PHP, MySQL, más algunos plugin extra, mas además el panel de control PhpMyAdmin (Aunque creo que en Linux el PhpMyAdmin no viene incluido en LAMP, si ese es el caso, se instala por separado).

Si te preguntas por qué no usar nuestro ordenador tal cual, la respuesta es muy sencilla. No queremos ni cargar nuestro ordenador más de lo necesario con procesos “extra”, ni exponerlo a posibles riesgos como puertas abiertas, etc. Por supuesto, la alternativa más sencilla a todo ésto, es instalar WAMP en nuestro ordenador, y usarlo tal cual. Pero recordad que WAMP es un paquete de software para montar un servidor web.

En cuanto al software de desarrollo, eso ya cada uno sabrá lo que necesita, y para qué lo necesita. En principio, para desarrollo básico (HTML (cualquier versión) – PHP – SQL – CSS), con un editor con sintaxis tipo [Notepad++](#) es más que suficiente. Tampoco hay que olvidar, el navegador. Lo ideal es tener todos los navegadores posibles. [Internet Explorer](#), [Mozilla Firefox](#), [Google Chrome](#), y [Opera](#). Si tu página se ve bien en al menos estos navegadores, es prácticamente seguro asumir que en el resto de navegadores (Safari, y otros por ahí muy poco conocidos o exclusivos de Linux como Konqueror) se verá bien. Tampoco está de más tener un editor de imágenes, tipo [GIMP](#) (gratis) o [Photoshop](#).

No cubriré en ésta entrada, el uso de ninguna de éstas herramientas de desarrollo web o edición de imagen.

## ¿POR DONDE EMPIEZO?

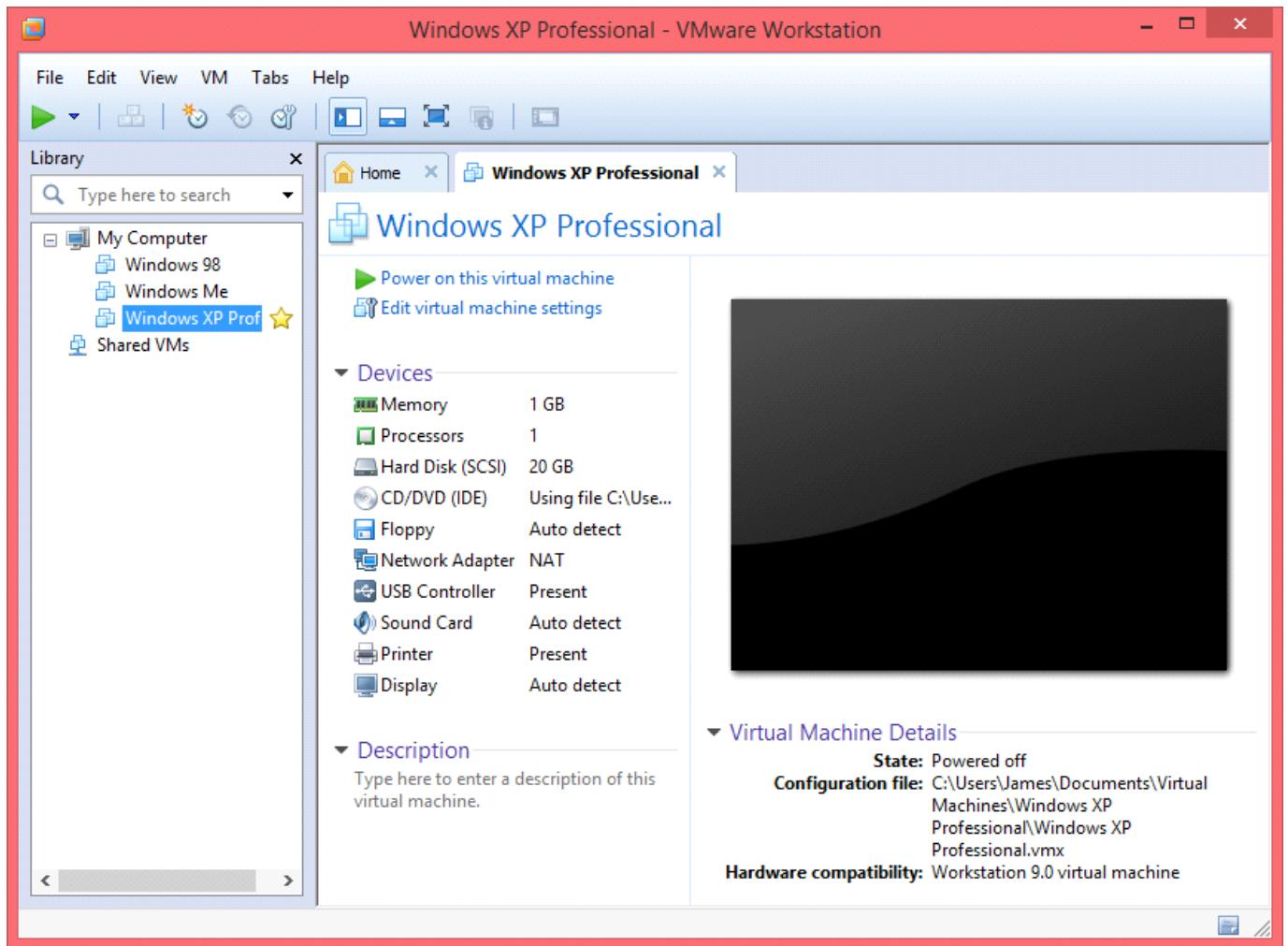
En mi caso, todos los pasos los realizo desde un ordenador con sistema Windows 8.1. Lo que haré, será crear una máquina virtual con Windows XP en VMWare workstation 10, conectarla a una red, e instalar el software del servidor. Además, compartiré la carpeta www del servidor para poder desarrollar directamente desde el ordenador host, y pasar todos los archivos al servidor de forma inmediata.

## PASO 1 – CREAR EL ORDENADOR VIRTUAL

Si vas a usar un ordenador físico, omite éste paso.

Iniciamos el programa VMWare Workstation o Virtual Box. Creamos una nueva máquina virtual (File > New Virtual Machine), y le introducimos el disco físico o la imagen del CD de Windows XP que vamos a instalar, o de Linux.

VMWare incluye una especie de autoinstalador, que nos instalará el sistema de forma automática para los sistemas Windows. Sólo necesitamos introducir la clave del CD, y un nombre para el ordenador, y VMWare se encargará él solo del resto. Si no es así o estamos en Virtual Box, tendremos que hacerlo de la forma tradicional. Introducir la imagen o el CD, arrancar desde el CD, formatear el disco, e instalar el sistema operativo.



Una cosa importante. Tanto VMWare como Virtual Box contienen un paquete de “adicciones”, llamados “VMWare Tools” y “Virtual Box Additions” respectivamente. Hay que instalarlos. Inicia el ordenador virtual con el sistema operativo ya instalado, y haz click en “VM > Install VMWare Tools”, o “Dispositivos > Insertar imagen de las <<Guest additions>>” en Virtual Box.

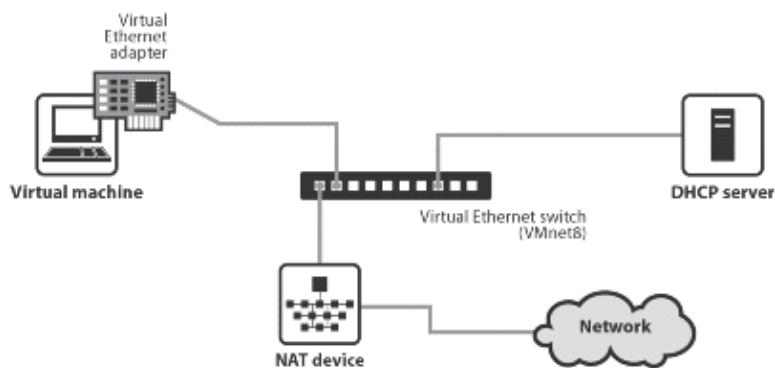
## PASO 2 – CONECTAR TODO A LA RED

En el caso de un ordenador físico, es muy sencillo. Si estás leyendo ésto, es porque dispones de una línea ADSL. Si éste es el caso, tu ISP está obligado a suministrarte un router ADSL. Y ese router ADSL, a su vez, contiene cuatro entradas para conectar hasta cuatro cables de red. Bueno, no siempre es así, pero la mayoría de los ISP si. Si ese es el caso, basta con conectar ambos ordenadores, al mismo router, a través de un cable de red que va desde la tarjeta de red ethernet de cada ordenador, hasta el router.

Si no dispones de un router con un switch (los 4 puertos de los que hablo), puedes o bien comprar un switch, o bien comprar un router de mejor calidad y cambiarlo.

Otra opción sería conectar dos tarjetas de red en el ordenador que está conectado a internet a través del router, y luego conectar mediante un cable de red el otro ordenador que vamos a usar como servidor, al nuestro.

En el caso de un ordenador virtual, disponemos de una serie de adaptadores de red virtuales. Nosotros lo que queremos es crear una NAT, de forma que el ordenador virtual se conecta a nuestro ordenador físico, como si estuviera conectado directamente a la tarjeta de red de nuestro ordenador. Virtual Box nos crea un adaptador de red virtual llamado “VirtualBox Host-Only network”, mientras que VMWare nos crea dos. “VMNet 1” y “VMNet 8”. En cualquier caso, en VMWare, vamos a las propiedades de nuestro ordenador virtual (con él apagado), y vamos a Adaptador de Red (Network Adapter). En él, seleccionamos la opción NAT.

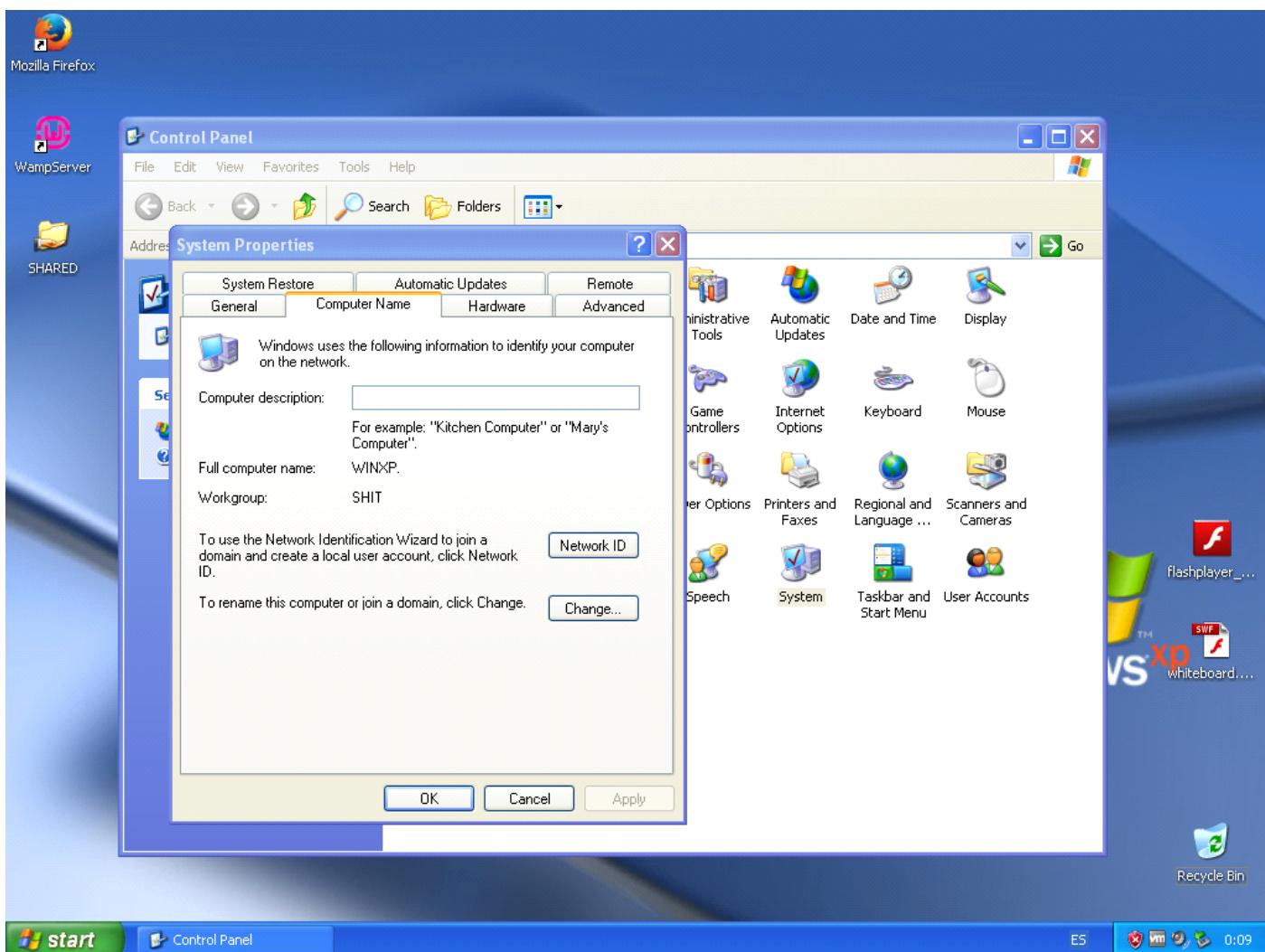


En el caso de VirtualBox, seleccionamos el ordenador virtual (con él apagado), y le damos a Configuración. Le damos a Red, y donde dice “Conectado a:”, seleccionamos NAT.

### PASO 3 – CONFIGURAR LA RED

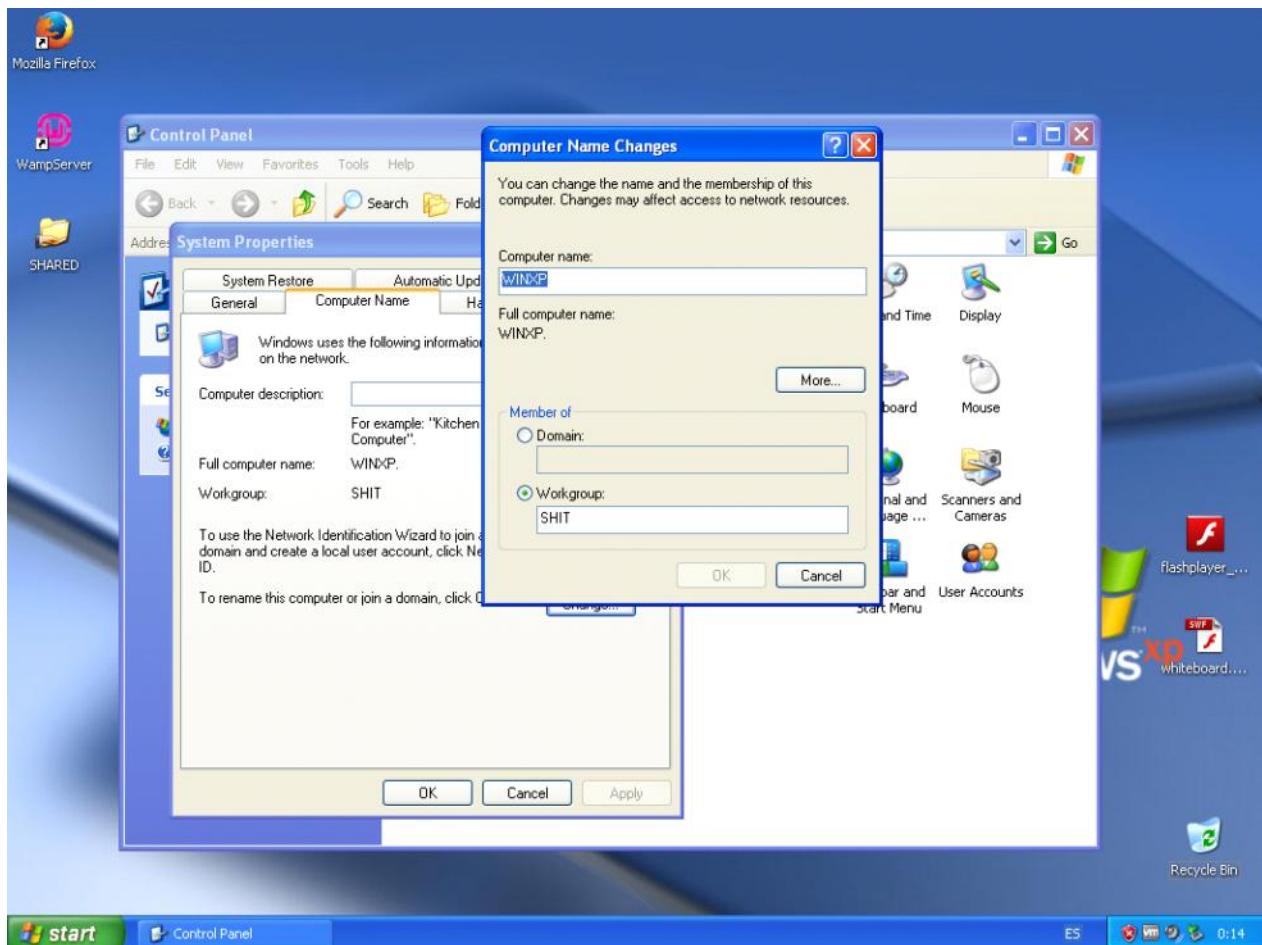
Solo sé configurar la red en Windows XP Professional, y en sistemas Windows en general. Así que para seguir éste paso en otros sistemas (Linux), me temo que tendréis que buscar una guía más completa. La idea, en cualquier caso, es la de conectarse a un grupo de trabajo, y habilitar la configuración de carpetas.

En el ordenador que vamos a usar como servidor, vamos a Inicio > Panel de control, doble click en “Propiedades del sistema”, y vamos a la pestaña “Nombre del Ordenador”.



Hacemos click en el botón “Cambiar”. En el nombre del ordenador, ponemos un nombre con el que podamos identificarlo, y despues seleccionamos “Grupo de trabajo”, y escribimos el nombre del grupo de trabajo de nuestro ordenador host.

Si no lo sabemos o no lo hemos establecido, podemos mirarlo del mismo modo en el ordenador host. En Windows 8 se encuentra en el Panel de control > Sistema (Con todos los elementos visibles) > Configuración avanzada del sistema > Nombre de equipo.



Al cambiar, nos pedirá reiniciar el sistema. Reiniciamos todo. Vamos al ordenador “central”, pulsamos la tecla Windows + R, escribimos CMD, y a continuación introducimos el comando “net view” (sin comillas). Si todo ha ido correcto, deberíamos ver dos ordenadores (el nuestro, y el remoto).

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 6.3.9600]
© 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\James>net view
Servidor           Descripción

\\NINJIHOST
\\WINXP
Se ha completado el comando correctamente.

C:\Users\James>
```

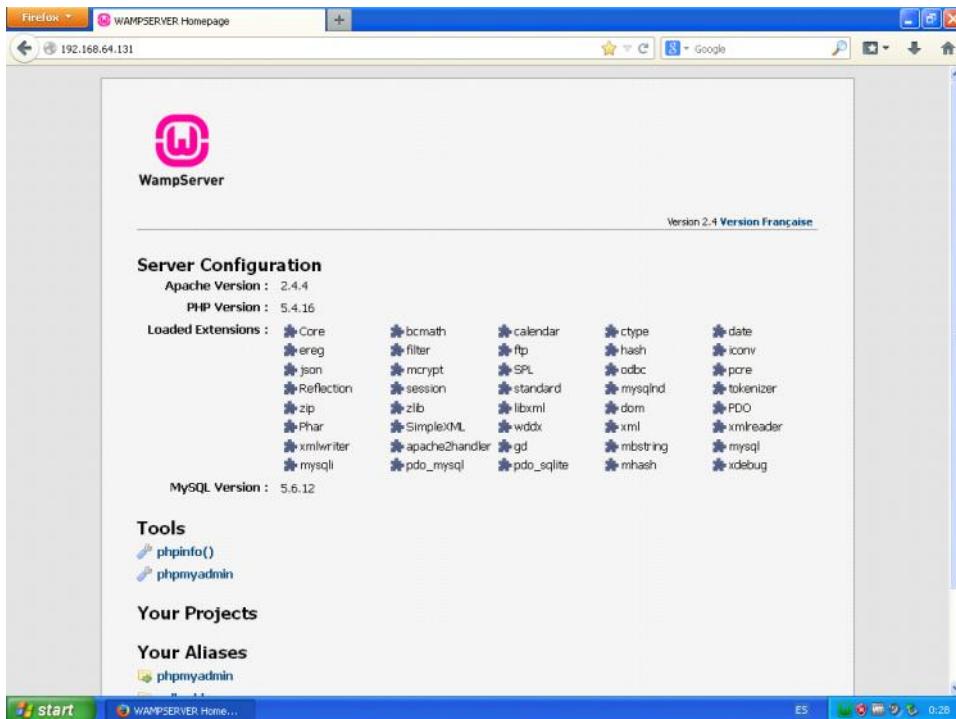
Y lo mismo, si hacemos lo mismo en el otro ordenador. Aunque aún no podemos hacer mucho con ésto. De momento, vamos al siguiente paso.

## PASO 4 – INSTALAR EL SERVIDOR WEB

Descargamos WAMP o LAMP, y lo instalamos en el sistema que vamos a usar como webhost, igual que cualquier otro programa para nuestro sistema. Cuando lo hagamos, reiniciamos el ordenador e iniciamos el servidor WAMP o LAMP. En el caso de WAMP, solo basta con hacer doble click en el ícono del programa. De momento, sólo nos aseguramos de que nos salga en verde, en la barra de tareas.

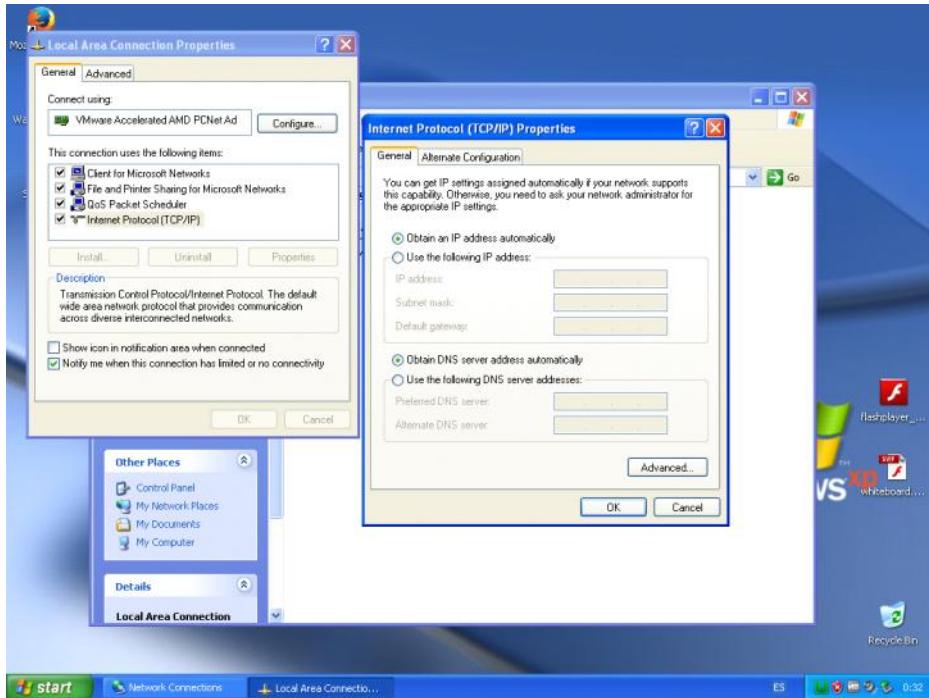


Si es así, vamos al navegador en nuestro web host. Escribimos “localhost”, y miramos si nos carga una página. Si no es así, podemos probar a bajar otra versión y reinstalar dicha versión. Por defecto, debe aparecer en verde, y debe mostrar una página en el navegador al acceder a nuestro ordenador.



Si todo va bien, vamos a configurar los permisos para poder acceder a nuestro web host desde el ordenador central. Primero, y antes de nada, no hemos configurado una IP. En el caso de un ordenador virtual, no es necesario. Lo dejamos por defecto.

En el caso de un ordenador físico, vamos al panel de control > conexiones de red, hacemos click derecho en nuestro adaptador de red, y le damos a propiedades. Hacemos click en “Protocolo de Internet (TCP/IP), y le damos a Propiedades.



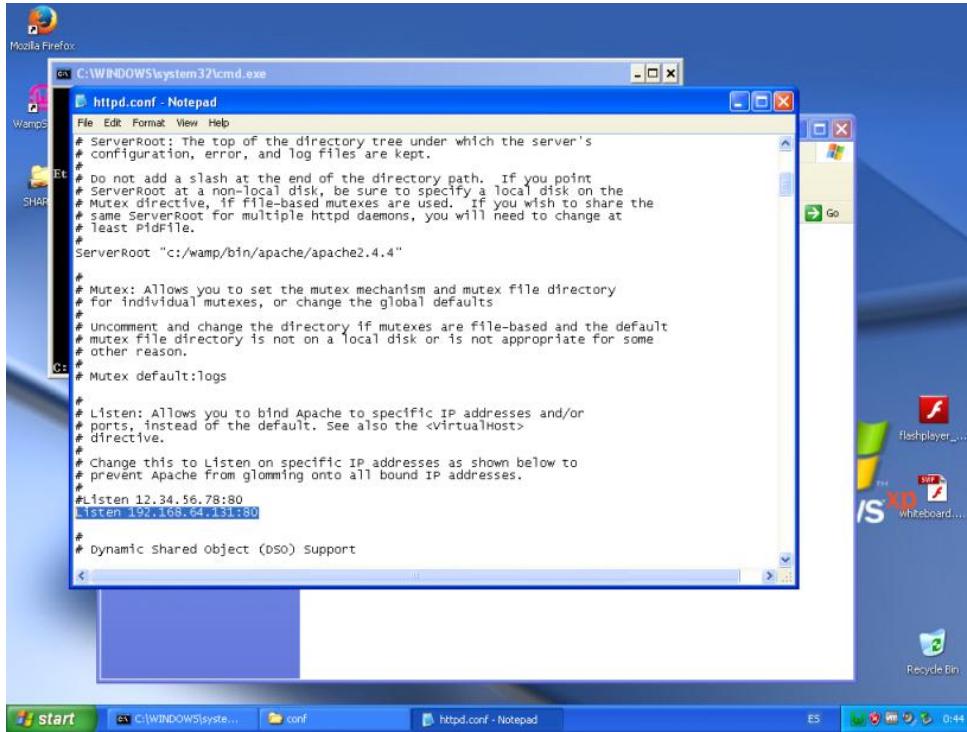
La IP a asignar, varía en función de nuestra red. En principio, una cualquiera en el rango desde 192.168.1.2, hasta 192.168.1.255. Si en el ordenador central vamos a la consola (Windows+R, y escribimos CMD), y luego usamos el comando IPCONFIG -ALL, podemos ver todas las IP que hay en uso en nuestra red. La máscara de red comunmente es de tipo C (255.255.255.0), y la puerta de enlace predeterminada, la IP de nuestro ordenador central (si lo tenemos conectado a él), o la de nuestro router (normalmente 192.168.1.1) si está conectado al router.

Vamos a nuestro webhost, donde tenemos el WAMP instalado. Pulsamos Windows + R, abrimos CMD, y escribimos IPCONFIG -ALL. Y anotamos la IP de nuestro adaptador de red predeterminado.

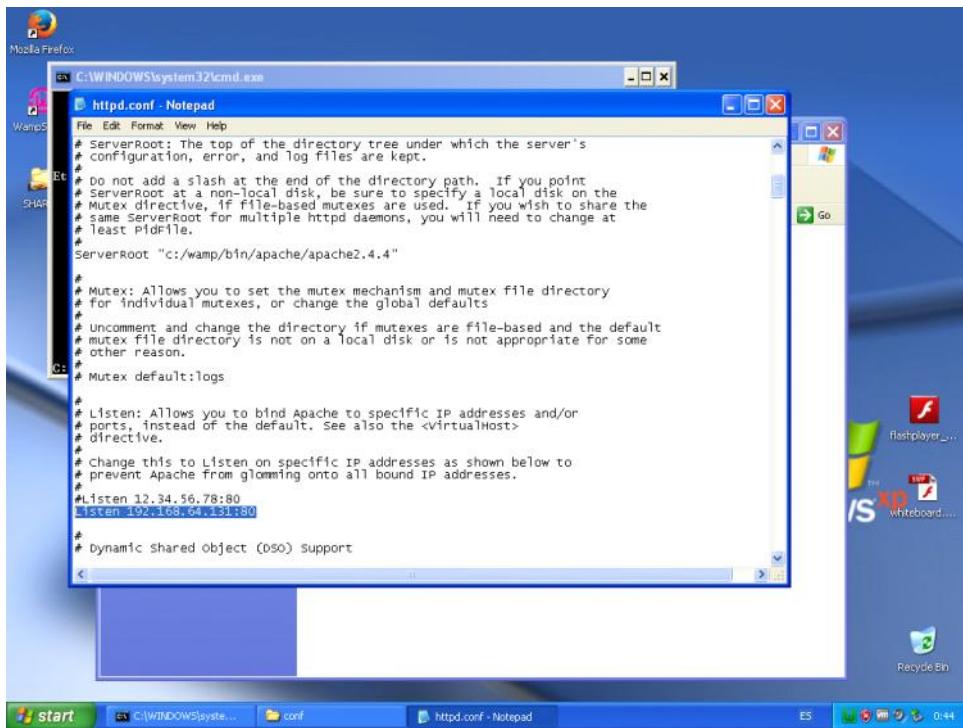


¡Ya queda poco! Vamos a la carpeta del WAMP, que por defecto es C:WAMP, o del LAMP, y vamos a binapacheApache2.x.xconf. Dentro, encontraremos un archivo llamado "httpd.conf" que abriremos con el bloc de notas.

Lo primero es buscar una línea que dirá algo así como "Listen :80". La cambiamos por "Listen nuestraip:80". Cambiando "nuestraip" por nuestra IP.



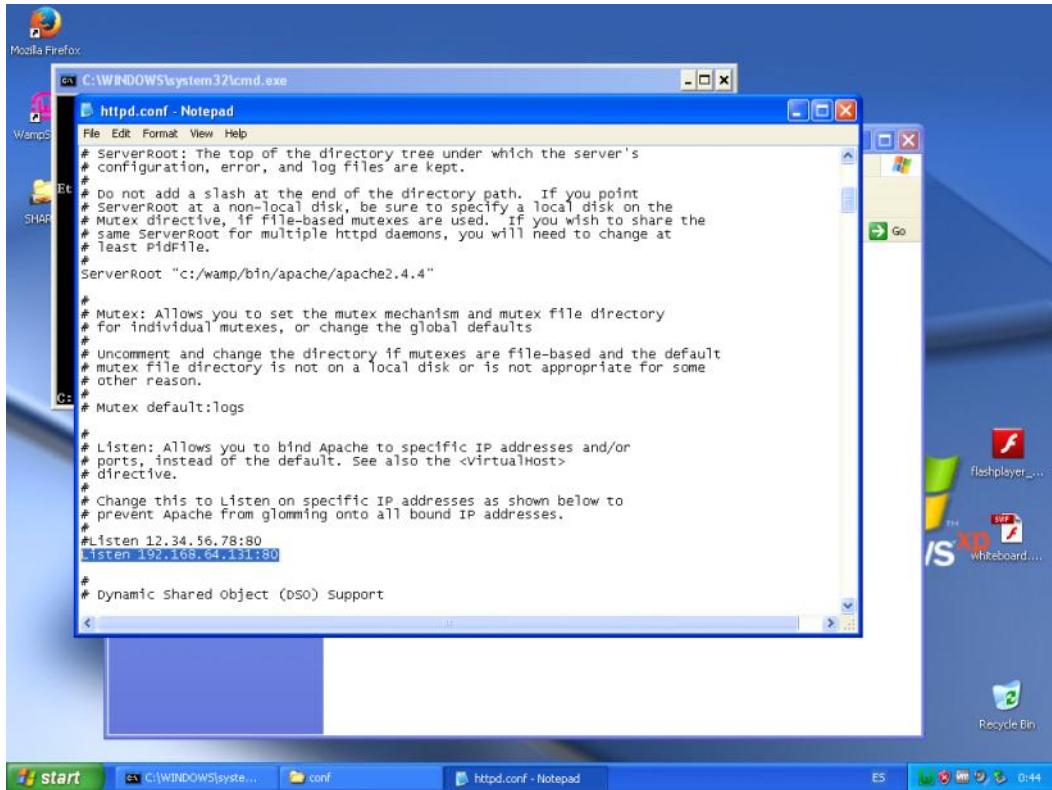
Mas abajo, hay una línea que dice “<Directory “c:/wamp/www”>”. Aquí es donde configuramos los permisos de la carpeta www de Apache, donde guardamos nuestra web. Un poco más abajo, hay una línea que dice “Deny from all”. Lo cambiamos por “Allow from all”. Ya sabes que al hacer ésto, cualquier ordenador que se conecte a nuestro webhost, tiene total acceso a la carpeta raíz del mismo. De ahí el usarlo sólo como servidor local, y no un servidor web convencional.



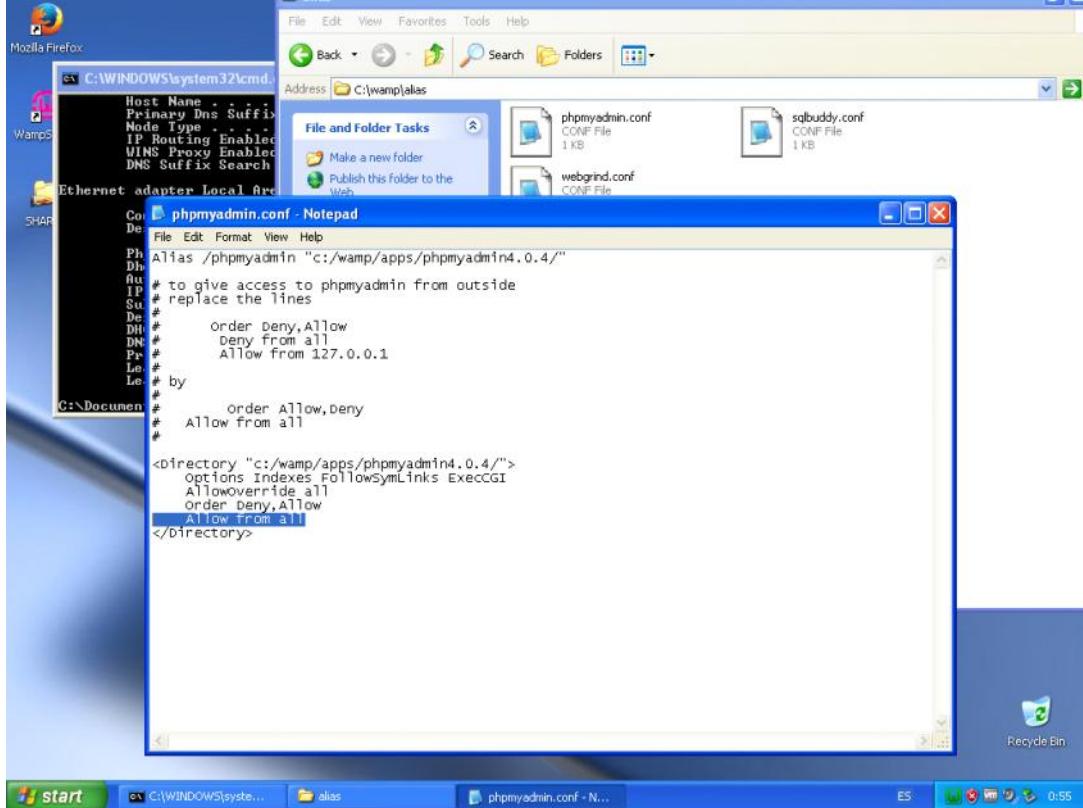
Guardamos, y dentro de la carpeta donde nos encontramos, vamos a la carpeta “extra”. Ahí encontraremos un archivo llamado “httpd-vhosts.conf”. Lo abrimos con el bloc de notas. Al final de éste archivo, añadimos lo siguiente:

```
<VirtualHost nuestraip:80>
    DocumentRoot "C:/wamp/www"
</VirtualHost>
```

Guardamos, y salimos.

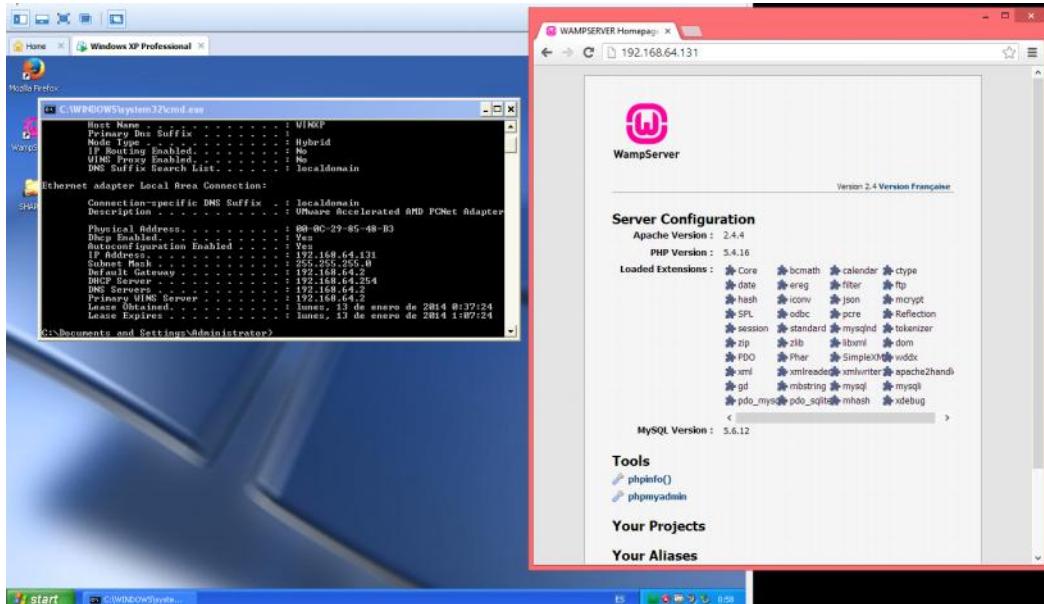


Queda un último archivo por modificar, para poder tener acceso al panel de PhpMyAdmin desde el ordenador central. Vamos a “C:wampalias”, y abrimos el archivo “phpmyadmin.conf” con el bloc de notas. Al igual que con la carpeta www en el paso anterior, donde dice “Deny from all”, lo cambiamos por “Allow from all”. De nuevo, no hagais ésto en un servidor web que vayais a poner al público.

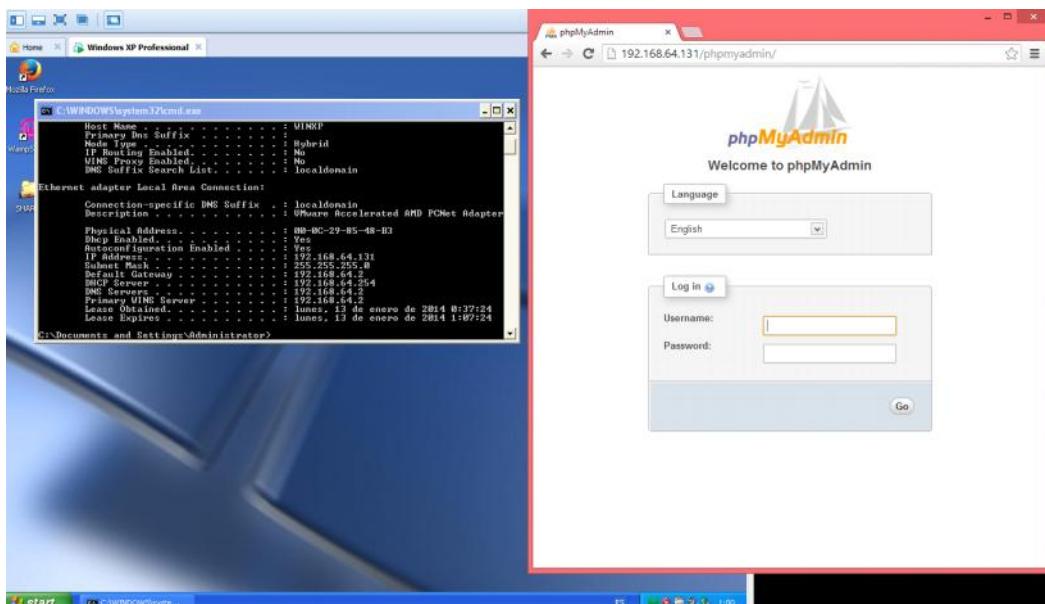


Guardamos, y salimos. Ahora, hacemos click sobre el icono de WAMP de la barra de tareas, seleccionamos “Restart all services”, y esperamos a que se ponga verde. Si lo hace, es que todo está bien.

Vamos al navegador del ordenador central, y escribimos la IP de nuestro webhost. Al hacerlo, deberíamos ver la página web.



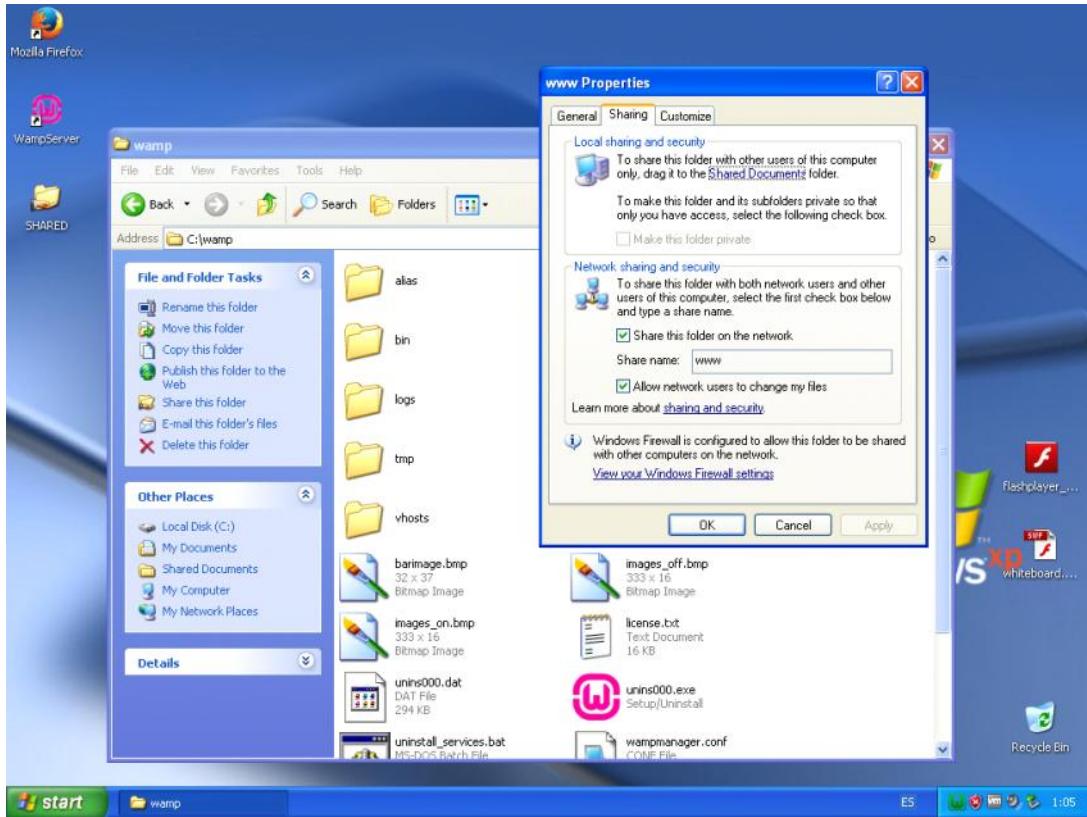
Deberíamos también tener acceso al panel de phpmyadmin.



Asegurate de que el firewall da acceso al puerto 80 del webhost. De lo contrario, no habrá conexión. Para entrar en el panel de phpmyadmin, usa como nombre de usuario “anonymous”, o “root” si necesitas privilegios de administración, sin contraseña, y haz click en Go.

## PASO 5 – COMPARTIR LA CARPETA WWW

Vamos a la carpeta del Wamp, que por defecto es C:wamp. Hacemos click derecho sobre la carpeta www, y seleccionamos “Propiedades”. Hacemos click en la pestaña “Compartir”. Seleccionamos las opciones “Compartir ésta carpeta en la red”, y “Permitir a los usuarios de red cambiar mis archivos”. Como nombre de carpeta, ponemos “www” sin comillas, o lo que sea.

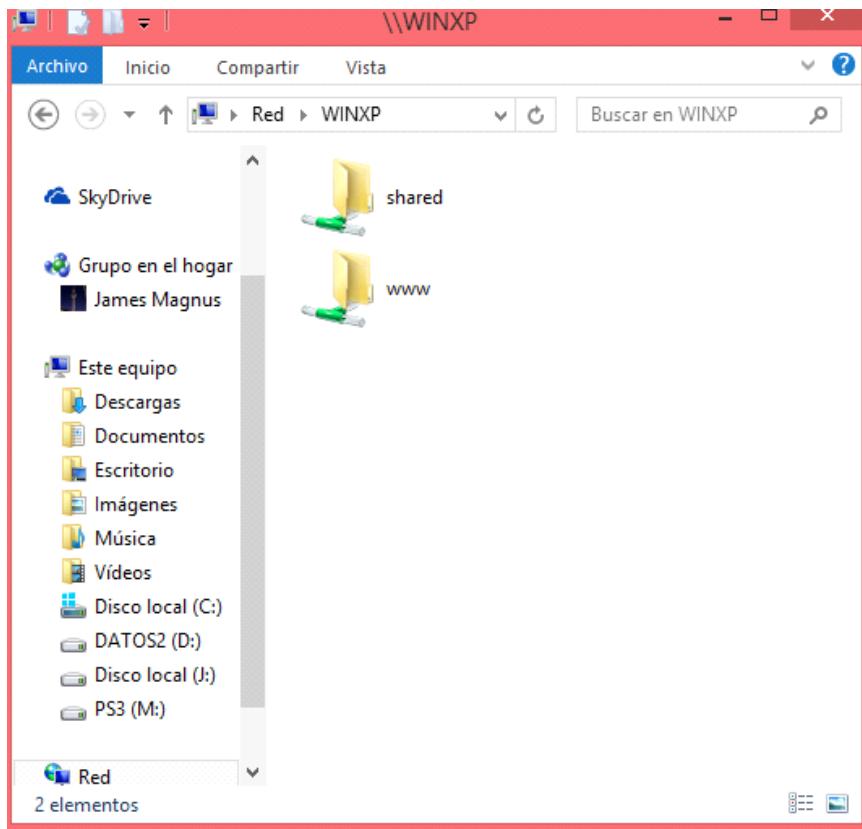


Vamos al ordenador central, y en el explorador de Windows, hacemos click en Red, o en Mis sitios de red. Debería aparecernos dos ordenadores, el nuestro, y nuestro webhost.

#### PC (2)



Hacemos doble click sobre el webhost. Deberíamos tener acceso al menos a la carpeta www, y a sus archivos desde aquí.



Si ésto es así, ¡Enhorabuena! Has completado el tutorial satisfactoriamente. Ahora puedes empezar a desarrollar tu página web, y subir los archivos a la carpeta www del webhost, para despues visualizar los cambios en el navegador de tu ordenador central.

Pegado de <<http://nинjiserver/WORDPRESS/?p=960>>

## Foros

sábado, 12 de julio de 2014

1:16

# Foros. ¿Cómo creo un foro? ¿Cómo lo gestiono y administro?

[December 14, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



Desde tiempos inmemorables, la gente se reunía en foros para socializarse y comerciar. A veces recorrian decenas o cientos de kilómetros sólo con éste propósito. Pero hoy en día, con internet, eso ya ha pasado a mejor vida. Y desde hace algunos años, hacemos lo mismo, pero desde nuestra propia casa. Y es que la sociedad moderna, está llena de vagos y perezosos. Como alguien que está escribiendo éstas líneas (y como él, ninguno ^\_^").

Aunque ya no se usan tanto como hace unos pocos años, en internet existen lo que denominamos "foros". Son exactamente lo mismo que eran en la antigüedad. Puntos de reunión donde la gente se socializa, pero desde internet. Éstos foros, aunque en ocasiones tienen un propósito general, suelen formarse en torno a un tipo de comunidad específica. Hay foros de videojuegos, de comunidades "gamer" (clanes y demás), de programación, de hobistas, de coches, etc. Todos estos foros tienen un propósito y un tipo de audiencia en específico.

Yo os voy a comentar cómo se puede crear un foro, como se puede administrar, y también os daré algunos consejos sobre todo ésto. Porque es bueno estar siempre bien informado.

### 1. Criterios a tener en cuenta, antes de crear un foro.

Lo primero a tener en cuenta es, por supuesto, la temática del foro. A qué gente va destinado el foro, o qué finalidad tiene. Como ya he dicho, un foro es una comunidad donde la gente se reúne para socializarse, y un buen punto es pensar qué gente queremos en nuestro foro. ¿Gamers? ¿Músicos? ¿Programadores?

Por supuesto, nosotros (los creadores), debemos ser parte de ese sector que escogamos. ¿Tiene sentido que yo, que no juego al League of Legends, abra un foro en el que se hable de dicho juego? No.

Existen foros que no tienen un sector específico, sino que son de ámbito general. Éstos foros, o

tienen muchísimo éxito, o normalmente no se comen un mojón porque la gente pasa de ellos.

## 2. ¿Qué necesito para comenzar?

Lo primero de todo, no es para nada una buena idea empezar sólo. Necesitas tener gente desde el principio. Busca amigos, o hazlos por ahí, y acordad crear un foro. Si no lo haces, tu foro quedará totalmente vacío, y eso es lo peor que puede pasar. Creeme, la gente lo que busca es actividad en los foros.

Lo segundo, es ideal contratar un hosting decente, con al menos PHP y bases de datos SQL. Los hay gratuitos, pero no los recomiendo si tu foro va a ser un foro grande. Porque cuando empieze a venir el tráfico fuerte de visitas, te limitará el tráfico.

Lo tercero, es el software del foro. Los hay de pago, como [vBulletin](#), o gratuitos y open source como [phpBB](#). Lo ideal es empezar con algo barato, porque desde luego que vBulletin, no es lo más barato. Si estas pensando en un foro tipo [2chan](#) o 4chan, que son en realidad tablones de imágenes (imageboards), exísten por ejemplo [Tinyboard](#), [Mitsuba BBS](#), o Excelis Imageboard Software.

En cualquier caso, es siempre ideal tener a mano una herramienta FTP como [FileZilla](#). No todos los servicios de alojamiento tienen una herramienta de subida de archivos decente, así que en la mayoría de los casos necesitarás mover los archivos a través de FTP.

## 3. Instalación del software

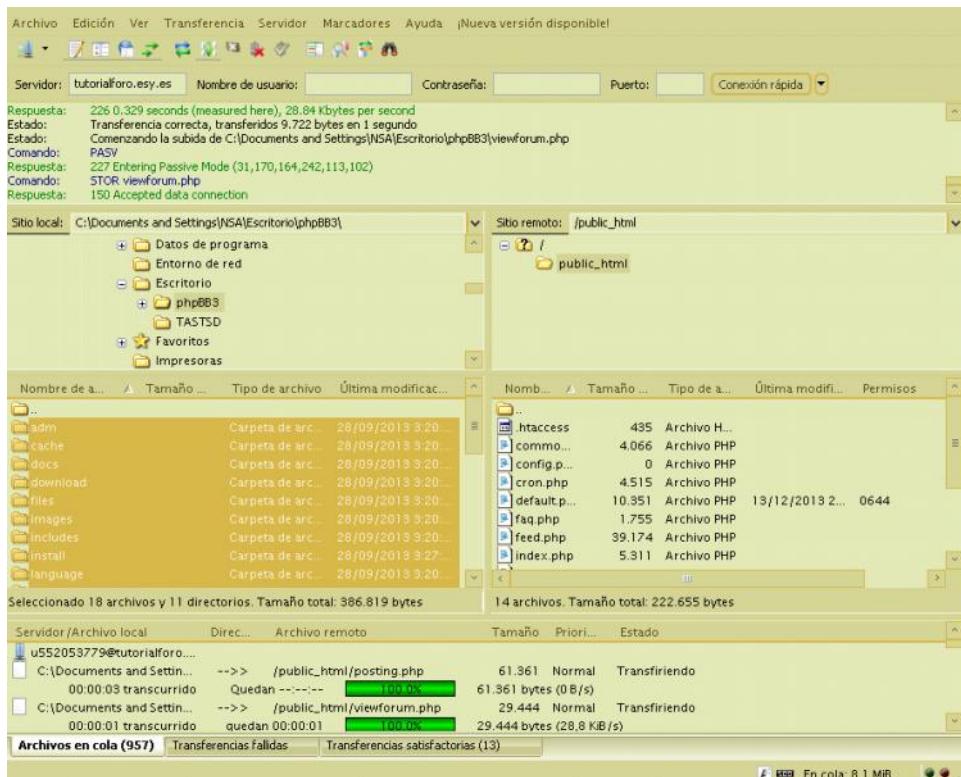
La instalación del software varía un poco en función del software usado, pero los conceptos son siempre los mismos.

Lo primero es siempre crear una base de datos SQL, protegido por contraseña. Ésta base de datos irá dedicada exclusivamente a tu foro. Muchos servicios gratuitos te limitan el número de bases de datos disponibles, otro factor por el cual es muy recomendable alquilar el alojamiento. Para hacer ésto, no creo que debas comerte mucho la cabeza. Todos los alojamientos, incluso los gratuitos, incluyen un panel de control desde el cual puedes realizar ésta, y muchas otras tareas.

Base de Datos MySQL	Usuario MySQL	Host MySQL	Uso de Disco, MB	Acciones
No tienes ninguna base de datos				

Es muy importante, ir anotando todos los nombres de usuario, contraseñas, y nombres de bases de datos, que vayamos creando, en un papel aparte o en algún sitio al cual nadie pueda tener acceso, ni siquiera colándose en tu ordenador.

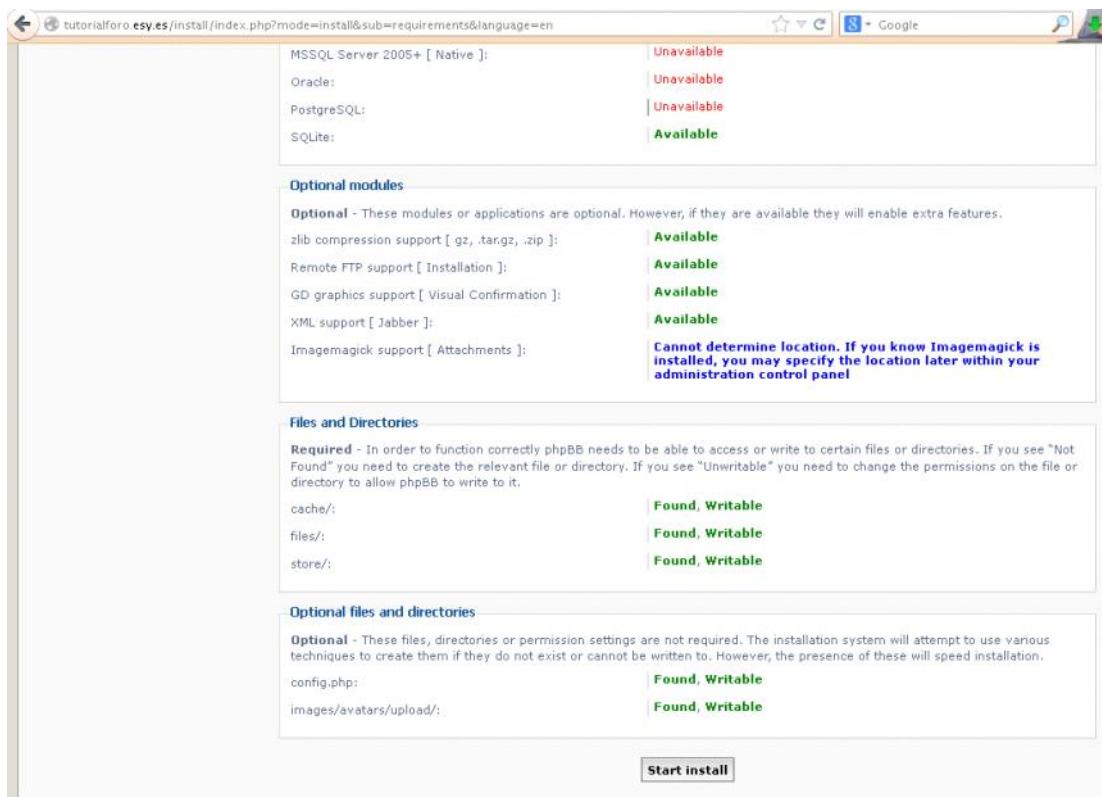
Lo siguiente es la creación de una cuenta FTP. También desde el panel de control de tu hosting. Algunos ya te la dan creada por defecto. Despues de eso, hay que descargar el software del foro deseado, y subirlo por FTP usando FileZilla, o un cliente FTP cualquiera. Yo voy a usar phpBB, por ser gratis, sencillo, y porque ya tengo experiencia con él.



En algunos casos, ni siquiera es necesario descargar nada. Hay muchos hostings que incluyen un autoinstalador de aplicaciones, con las aplicaciones más comunes (WordPress, phpBB, Joomla...). Están simple como ir al menú, seleccionar lo que quieras instalar, e instalarlo.

En caso de subirlo mediante FTP, el siguiente paso es acceder desde tu navegador, a la página de tu foro. Normalmente suele haber un archivo llamado "install.php", o una carpeta llamada "install" con un archivo llamado "index.php", al cual debemos acceder desde nuestro navegador.

Continuamos el proceso de instalación, haciendo click en "INSTALL", y dandole todos los datos que nos solicite, cuando nos lo solicite. En el caso de phpBB también realizará un chequeo del entorno, para ver si somos capaces de poder cargar el software del foro sin problemas. Muy raro es que un host no lo soporte, así que lo más normal es que puedas proceder sin más complicaciones.



Tampoco ocurre nada porque alguna de las bases de datos no sea compatible. Con que al menos soporte MySQL, basta. Y creo que cualquier hosting tiene MySQL hoy en día. Simplemente continuamos la instalación. Nos pedirá los datos de nuestra base de datos, incluido la DNS (el nombre de dominio) donde se aloja la base de datos. En algunos hostings gratuitos como Hostinger, la base de datos se aloja en un dominio distinto al usado por la página (mysql.hostinger.es). Si no sabes qué poner aquí, pon “127.0.0.1” o “localhost”. El puerto lo podemos dejar en blanco, y el resto los datos de nuestra base de datos.

Despues de ésto, empezaremos con los datos del foro. Primero, nuestros datos de administrador y usuario.

### Administrator configuration

Default board language:	<input type="button" value="British English"/>
Administrator username:	<input type="text" value="Ninjihaku"/>
Please enter a username between 3 and 20 characters in length.	
Administrator password:	<input type="password"/>
Please enter a password between 6 and 30 characters in length.	
Confirm administrator password:	<input type="password"/>
Contact e-mail address:	<input type="text"/>
Confirm contact e-mail:	<input type="text"/>

**Proceed to next step**

Despues, la configuración avanzada. De aquí, salvo en casos muy específicos, no debemos tocar nada, en principio.

The settings on this page are only necessary to set if you know that you require something different from the default. If you are unsure, just proceed to the next page, as these settings can be altered from the Administration Control Panel later.

#### E-mail settings

##### Enable board-wide e-mails:

If this is set to disabled no e-mails will be sent by the board at all. Note the user and admin account activation settings require this setting to be enabled. If currently using "user" or "admin" activation in the activation settings, disabling this setting will require no activation of new accounts.

Enabled  Disabled

Yes  No

PLAIN

##### Use SMTP server for e-mail:

Only used if a username/password is set; ask your provider if you are unsure which method to use.

##### SMTP username:

Only enter a username if your SMTP server requires it.

##### SMTP password:

Only enter a password if your SMTP server requires it.

**Warning:** This password will be stored as plain text in the database, visible to everybody who can access your database or who can view this configuration page.

#### Server URL settings

##### Cookie secure:

If your server is running via SSL set this to enabled else leave as disabled. Having this enabled and not running via SSL will result in server errors during redirects.

Enabled  Disabled

Yes  No

http://

##### Force server URL settings:

If set to yes the server settings defined here will be used in favour of the automatically determined values.

##### Server protocol:

This defines the server protocol if these settings are forced. If empty or not forced the protocol is determined by the cookie secure settings (http:// or https://).

##### Domain name:

The domain name this board runs from (for example: www.example.com).

##### Server port:

The port your server is running on, usually 80, only change if different.

##### Script path:

The path where phpBB is located relative to the domain name, e.g. /phpBB.

[Proceed to next step](#)

Despues de éste paso, nos pide por fin que nos logeemos en el foro (¡Por fin!). Antes de nada, debemos borrar el directorio “install” que hemos estado usando para instalar el foro, por seguridad. Si no lo hacemos, alguien nos podría liar una gorda.

#### Congratulations!

You have successfully installed phpBB 3.0.12. Please proceed by choosing one of the following options:

#### Convert an existing board to phpBB3

The phpBB Unified Convertor Framework supports the conversion of phpBB 2.0.x and other board systems to phpBB3. If you have an existing board that you wish to convert, please [proceed to the converter](#).

#### Go live with your phpBB3!

Clicking the button below will take you to a form for submitting statistical data to phpBB in your Administration Control Panel (ACP). We would appreciate it if you could help us by sending that information. Afterwards you should take some time to examine the options available to you. Remember that help is available online via the [Documentation](#), [README](#) and the [Support Forums](#).

Please delete, move or rename the install directory before using your board. While this directory exists, only the Administration Control Panel (ACP) will be accessible.

[Login](#)

Al pulsar sobre Login, el programa nos logeará automáticamente y nos mandará a nuestro panel de administración.

## 4. Configuración del foro

Aquí es donde empieza lo “fuerte”. Éstos foros son un poco complejos, así que sólo cubriré lo básico para sobrevivir en el mundo de los foros. Lo primero es cambiar el nombre, y la descripción. Porque es lo primero que los usuarios identificarán. Supongo que ya tendrás un nombre pensado para tu comunidad, ¿No?

Tenemos una barra lateral con distintas secciones de configuración

Empezamos por entrar a “Board Settings”, que son las opciones globales del foro. Dentro podemos cambiar el nombre y la descripción del foro.

**Board settings**  
Here you can determine the basic operation of your board, give it a fitting name and description, and among other settings adjust the default values for timezone and language.

**Site name:** Foro de prueba

**Site description:** Es un tutorial en elektronblog.wordpress.org

**Disable board:** This will make the board unavailable to users. You can also enter a short (255 character) message to display if you wish.

**Default language:** British English

**Date format:** Fri Dec 13, 2013 10:01 pm | D M d, Y g:i a

**Guest timezone:** [UTC] Western European Time, Greenwich Mean Time

**Enable Summer Time/DST:** No

**Default style:** prosilver

**Override user style:** No

**Warnings:**  
Warning duration: Number of days that will elapse before a warning will automatically expire from a user's record. Set this value to 0 to make warnings permanent.  
90 Days

**Submit changes**

Podemos ver en cualquier momento los cambios que vayamos realizando en nuestro foro, desde su página.

Screenshot of the phpBB® forum control panel showing the "Board index" page.

**phpBB® Foro de prueba**  
Un foro para un tutorial en elektronblog.wordpress.org

User Control Panel (0 new messages) • View your posts

It is currently Fri Dec 13, 2013 10:04 pm  
[ Moderator Control Panel ]

Last visit was: Fri Dec 13, 2013 9:53 pm

View unanswered posts • View unread posts • View new posts • View active topics

Mark forums read

**YOUR FIRST CATEGORY**

	TOPICS	POSTS	LAST POST
Your first forum Description of your first forum.	1	1	by <b>Ninjihaku</b> Fri Dec 13, 2013 9:53 pm

**WHO IS ONLINE**  
In total there is 1 user online :: 1 registered, 0 hidden and 0 guests (based on users active over the past 5 minutes)  
Most users ever online was 1 on Fri Dec 13, 2013 10:03 pm

Registered users: **Ninjihaku**  
Legend: Administrators, Global moderators

**STATISTICS**  
Total posts 1 • Total topics 1 • Total members 1 • Our newest member **Ninjihaku**

**Board features**  
Here you can enable/disable several board features.

**Board features**

**Private messaging:** Enable private messaging for all users.  
 Yes  No

**Allow subscribing to topics:**  Yes  No

**Allow subscribing to forums:**  Yes  No

**Allow username changes:**  Yes  No

**Allow attachments:**  Yes  No

**Allow attachments in private messages:**  Yes  No

**Allow users to report private messages:** If this setting is enabled, users have the option of reporting a private message they have received or sent to the board's moderators. These private messages will then be visible in the Moderator Control Panel.  
 Yes  No

**Allow BBCode:**  Yes  No

**Allow smiles:**  Yes  No

**Allow signatures:**  Yes  No

**Allow disabling of word censoring:** Users can choose to disable the automatic word censoring of posts and private messages.  
 Yes  No

**Allow bookmarking topics:** User is able to store personal bookmarks.  
 Yes  No

**Allow birthdays:** Allow birthdays to be entered and age being displayed in profiles. Please note the birthday list within the board index is controlled by a separate load setting.  
 Yes  No

**Allow quick reply:** This switch allows for the quick reply to be disabled board-wide. When enabled, forum specific settings will be used to determine whether the quick reply is displayed in individual forums.  
 Yes  No

**Load settings**

**Enable birthday listing:** If disabled the birthday listing is no longer displayed. To let this setting take effect the birthday feature needs to be enabled too.  
 Yes  No

**Enable display of moderators:**  Yes  No

**Enable display of jumpbox:**  Yes  No

**Allow styles to display custom profile fields in memberlist:**  Yes  No

**Display custom profile fields in user profiles:**  Yes  No

**Display custom profile fields on topic pages:**  Yes  No

**Submit changes**

Es hora de empezar a configurar las secciones del foro. En "Manage forums". Al principio tenemos por defecto una categoría llamada "Your first category", con un post dentro. Vamos a borrar ésta categoría, dandole al botón rojo que aparece junto a la categoría. Nos preguntará si queremos borrarlo todo, o si queremos mover su contenido. Lo dejamos todo como está, y le damos a borrar.

## Forum administration

In phpBB3 everything is forum based. A category is just a special type of forum. Each forum can have an unlimited number of sub-forums and you can determine whether each may be posted to or not (i.e. whether it acts like an old category). Here you can add, edit, delete, lock, unlock individual forums as well as set certain additional controls. If your posts and topics have got out of sync you can also resynchronise a forum. **You need to copy or set appropriate permissions for newly created forums to have them displayed.**

Board index

Ahora es cuando hacemos click en “Create new forum”. Los foros, al menos los estandar, tienen tres categorías de foros. Las categorías, que pueden contener foros. Los foros, que pueden contener mensajes y subforos, y los enlaces (Links), que no son más que eso.

Así que empezamos por seleccionar en “Forum type” la opción “Category”. Añadimos un nombre y una descripción, y le damos a Submit.

### Edit forum :: General

The form below will allow you to customise this forum. Please note that moderation and post count controls are set via forum permissions for each user or usergroup.

**Forum settings**

**Forum type:** Category

**Parent forum:** No parent

**Copy permissions from:** Do not copy permissions

If you select to copy permissions, the forum will have the same permissions as the one you select here. This will overwrite any permissions you have previously set for this forum with the permissions of the forum you select here. If no forum is selected the current permissions will be kept.

**Forum name:** General

**Description:** Foro general

Any HTML markup entered here will be displayed as is.

**Parse BBCode**  **Parse smilies**  **Parse links**

**Forum image:** Location, relative to the phpBB root directory, of an additional image to associate with this forum.

**Forum password:** Defines a password for this forum, use the permission system in preference.

**Confirm forum password:** Only needs to be set if a forum password is entered.

**Forum style:** Default style

**General forum settings**

**Enable active topics:** If set to yes active topics in selected subforums will be displayed under this category.

Yes  No

A continuación, nos permitirá ajustar los permisos para los distintos grupos de nuestro foro. Los grupos por defecto son: Bots (Los crawler de los buscadores), Guest (Usuarios no registrados), Newly registered users (Nuevos usuarios), Registered users (Usuarios normales y registrados), Global moderators (Moderadores) y Administrators (Administradores, incluyendote tú).

Seleccionamos primero todos los grupos, y hacemos click en “Add permissions”.

**Forum permissions**

Here you can alter which users and groups can access which forums. To assign moderators or define administrators please use the appropriate page.

**Forums:** General

**Users**

**Manage users**

Select all users

**Groups**

**Manage groups**

Select all groups

**Add users**

Place each username on a separate line.

Select anonymous user [ Find a member ]

**Add groups**

**Administrators**  
**Bots**  
**Global moderators**  
**Guests**  
**Newly registered users**  
**Registered users**

**Add permissions**

**Add permissions**

A continuación, nos permite seleccionar el tipo de acceso de cada usuario. Es decir, si queremos darle a un grupo un acceso total (Full access), acceso normal (Standard Access), o uno limitado (Limited access), o simplemente no darle acceso (No access). También podemos darles acceso a herramientas de creación de encuestas (Poll). Recomiendo dejarlo más o menos así:

**General [Forum permissions]**

**Administrators**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Bots**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Global moderators**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Guests**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Newly registered users**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Registered users**

Role:  [All yes · All no · All never](#) [Advanced Permissions](#)

**Apply all permissions** **Reset**

En las secciones a las que los usuarios puedan participar. En cualquier caso, en Administrators, debe establecerse permiso total (Full access).

Ahora que ya tenemos una categoría, podemos comenzar a crear foros dentro de ella. Vamos a “Manage Forums”, y volvemos a hacer click en “Create New Forum”. Esta vez, en “Forum type”, seleccionamos “Forum”, en “Parent Forum”, seleccionamos la categoría que acabamos de crear, y en “Copy permissions from”, la categoría que acabamos de crear.

**Edit forum :: Foro General**

The form below will allow you to customise this forum. Please note that moderation and post count controls are set via forum permissions for each user or usergroup.

**Forum settings**

<b>Forum type:</b>	Forum
<b>Parent forum:</b>	General
<b>Copy permissions from:</b>	General
<b>Forum name:</b>	Foro General
<b>Description:</b>	El foro general
	<input checked="" type="checkbox"/> Parse BBCode <input checked="" type="checkbox"/> Parse smilies <input checked="" type="checkbox"/> Parse links
<b>Forum image:</b>	
<b>Forum password:</b>	
<b>Confirm forum password:</b>	
<b>Forum style:</b>	Default style

**General forum settings**

<b>Forum status:</b>	Unlocked
<b>List subforums in legend:</b>	<input checked="" type="radio"/> Yes <input type="radio"/> No

Le damos a Submit, y ¡Lísto! Los permisos los copia de la categoría que le hemos asignado en "Copy permissions from", así que a menos que éste foro tenga unos permisos distintos del resto, no debemos especificárselos.

Si vamos al índice, podemos comprobar que nuestro foro va cogiendo forma.

phpBB® creating communities

**Foro de prueba**  
Un foro para un tutorial en elektronblog.wordpress.org

Search... Advanced search

Board index

User Control Panel (0 new messages) • View your posts

FAQ Members Logout [ Ninjihaku ]

It is currently Fri Dec 13, 2013 10:35 pm Last visit was: Fri Dec 13, 2013 9:53 pm  
[ Moderator Control Panel ]

View unanswered posts • View unread posts • View new posts • View active topics

GENERAL TOPICS POSTS LAST POST

GENERAL	TOPICS	POSTS	LAST POST
Foro General El foro general	0	0	No posts

Mark forums read

#### WHO IS ONLINE

In total there is 1 user online :: 1 registered, 0 hidden and 0 guests (based on users active over the past 5 minutes)  
Most users ever online was 1 on Fri Dec 13, 2013 10:03 pm

Registered users: **Ninjihaku**  
Legend: *Administrators*, *Global moderators*

#### STATISTICS

Sigue repitiendo los pasos anteriormente descritos para crear tantas categorías y foros, como necesites.

## 5. Añadir moderadores/administradores

Si me has hecho caso, habrás empezado un foro con más gente. Y claro, habréis acordado administrarlo entre todos. Creeme, es lo mejor, si hay confianza. En ese caso, tendrás que darle a tus compañeros un rango adecuado.

En Permissions, vamos a “User Permissions”. Escribimos el nombre del usuario a buscar, y hacemos click en Submit. En la siguiente ventana, seleccionamos el rol del usuario (Usuario normal, Moderador o Administrador), y su tipo de acceso (Similar a la configuración de permisos del foro), y pulsamos en “Apply all permissions”.

You are logged in as: **Ninjihaku** [ Logout ] [ ACP Logout ]

**GLOBAL PERMISSIONS**

- > **User permissions**
  - Group permissions
  - Administrators
  - Global moderators

**FORUM BASED PERMISSIONS**

- Forum permissions
- Copy forum permissions
- Forum moderators
- User forum permissions
- Group forum permissions

**PERMISSION ROLES**

- Admin permissions

**Setting permissions**

Permissions are based on a simple yes/no system. Setting an option to NEVER for a user or usergroup overrides any other value assigned to it. If you do not wish to assign a value for an option for this user or group select no. If values are assigned for this option elsewhere they will be used in preference, else NEVER is assumed. All objects marked (with the checkbox in front of them) will copy the permission set you defined.

Select type: Admin permissions Go > User permissions

User permissions

Usuario

Role: All Features All yes · All no · All never Advanced Permissions

Apply all permissions Reset

## 6. Modificar usuarios

En Users and Groups, en Manage Users, podemos buscar un usuario por su nombre (de forma similar al apartado anterior), y modificar algunos de sus datos, banearle, o incluso borrarle. En Basic tools, además de banear, podemos borrar o mover sus mensajes posteados (Todos ellos), o incluso borrarle la firma (Algo útil cuando se ponen firmas muy grandes).

**Overview**

**Username:** Length must be between 3 and 20 characters. **Usuario** [ Test out user's permissions ]

**Registered:** Fri Dec 13, 2013 9:53 pm

**Registered from IP:** 83.40.80.239 [ Whois ]

**Last active:** -

**Posts:** 0

**Warnings:** 0

**Founder:** Founders have all administrative permissions and can never be banned, deleted or altered by non-founder members.

**E-mail:** usuarioconmail@gmail.com

**Confirm e-mail address:** You only need to specify this if you are changing the users e-mail address.

**New password:** Must be between 6 and 100 characters.

**Confirm password:** You only need to confirm your password if you changed it above.

**Basic tools**

**Quick tools:** Ban by IP

**Reason for ban:** Por gilipollas

**Reason shown to the banned:** Por incumplimiento de las normas del foro.

**Delete user**

**Delete user:** Please note that deleting a user is final, they cannot be recovered. Unread private messages sent by this user will be deleted and will not be available to their recipients. The user has no posts to retain or delete.

## 7. Uso y moderación del foro

Para crear un tema nuevo en un foro, solo tienes que navegar dentro de él, y pulsar sobre “New topic”. Despues, y si tenemos permiso para postear (si eres admin y has seguido el tutorial, deberías tenerlo), nos aparecerá el editor de mensajes.

## Foro General

**POST A NEW TOPIC**

**Subject:** HOLAKASE

**B** **i** **u** **Quote** **Code** **List** **List=** **[\*]** **Img** **URL** **Flash** **Normal** **Font colour**

HOLA K ASE!  
POST DE PRUEBA  
:lol: :lol: :lol: :lol: :lol: :lol: :lol:

**Save draft** **Preview** **Submit**

Síplemente escribimos nuestro mensaje, hacemos click en Submit, y listo.

### HOLAKASE

**POSTREPLY**

Search this topic...

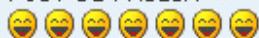
Search

1 post • Page

#### HOLAKASE

By **Ninjihaku** » Fri Dec 13, 2013 11:06 pm

HOLA K ASE!  
POST DE PRUEBA



\* **EDIT**

X

!

?

“ **QUOTE** ”

**Ninjihaku**

Site Admin

Posts: 1  
Joined: Fri Dec 13  
2013 9:53 pm



**POSTREPLY**

1 post • Page

< Return to Foro General

Jump to:

**Foro General**

Quick-mod tools: **Lock topic**

#### WHO IS ONLINE

Users browsing this forum: **Ninjihaku** and 0 guests

**Board index**  **Subscribe topic** **Bookmark topic**

The team • Delete all board cookies • All times are

Powered by phpBB® Forum Software © phpBB Group  
[Administration Control Panel](#)

Si nos equivocamos en algo, siendo usuarios normales o no, podemos hacer click en el botón de Edit, para editar nuestro post.

Si alguien escribe algo, y queremos responderle de forma rápida, podemos hacer click en Quote. Y si lo que queremos es responder a un tema ya creado, hacemos click en Post Reply.

Como moderadores, tenemos algunas opciones extra. Como el botón con la X encima del post, que nos permite eliminarlo. Siendo moderadores, debería aparecer encima de todos los mensajes. El botón “!” permite reportar a la moderación un mensaje. Los mensajes que reporten nuestros usuarios nos aparecerán en el panel de moderación, el cual es accesible desde un enlace en la

esquina superior izquierda del foro, bajo la cabecera.

## Moderator Control Panel

Main

**Front page**

**LATEST 5 POSTS AWAITING APPROVAL**  
There are no posts waiting for approval.

**LATEST 5 REPORTS**  
In total there is 1 report to review.

VIEW DETAILS	REPORTER & FORUM
HOLAKASE Posted by <b>Ninjihaku</b> » Fri Dec 13, 2013 11:06 pm	Reported by <b>Ninjihaku</b> on Fri Dec 13, 2013 11:12 pm Forum: Foro General

**LATEST 5 PM REPORTS**  
There are no PM reports to review.

Si en algún momento queremos cerrar un post (que significa, que nadie puede postear en él, salvo los administradores), en el editor de posts, en la parte de abajo, podemos seleccionar la opción “Lock topic”. Podemos también hacer que el tema quede siempre arriba de la lista de temas, haciendo click en “Sticky”, o convertirlo en un anuncio global con “Global”.

**Options** **Upload attachment** **Poll creation**

Disable BBCode  
 Disable smileys  
 Do not automatically parse URLs  
 Attach a signature (signatures can be altered via the UCP)  
 Notify me when a reply is posted  
 Lock topic  
 Lock post [Prevent editing]

Change topic type to:  Normal  Sticky  Announce  Global

Stick topic for:  Days  
Enter 0 or leave blank for a never ending Sticky/Announcement. Please note that this number is relative to the date of the post.

Reason for editing this post:

[Board index](#) The team • Delete all board cookies • All times are UTC

## 8. Recomendaciones

Mas o menos, éstas son las instrucciones más básicas a la hora de instalar, configurar y administrar un foro. El resto de cosas las podeis encontrar en el manual (si, manual) del software que useis para vuestro foro. En el caso de phpBB, podeis encontrar un manual [aquí](#).

Ahora queda por ver, ¿Qué política imponemos en nuestro foro? Porque ahora somos líderes o jefes de nuestra propia comunidad.

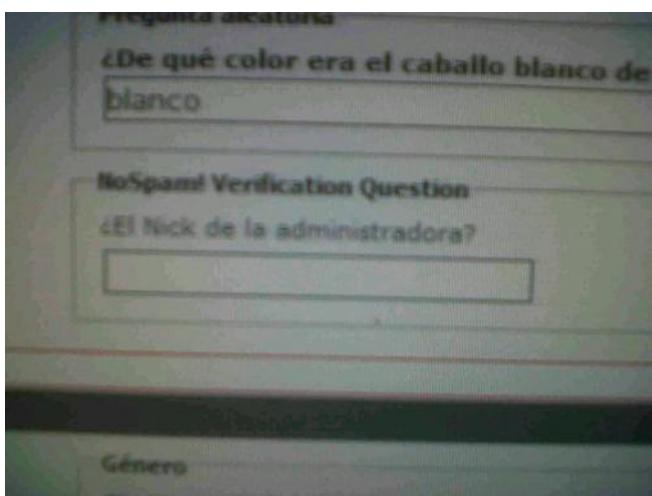
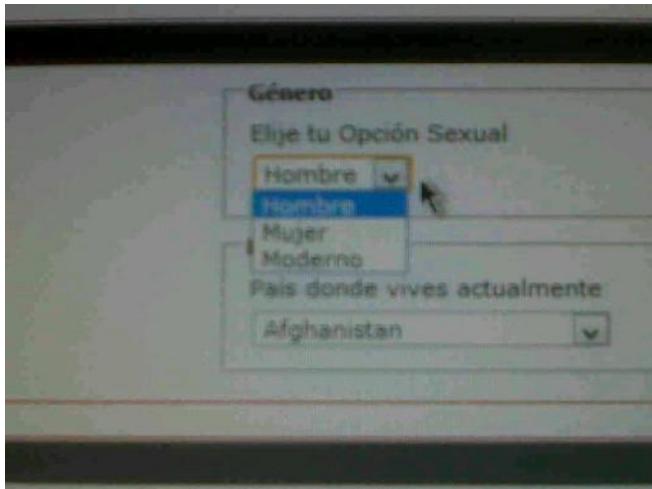
Es obvio que la gente va, sobre todo, a pasarlo bien, o a intercambiar información. Por ello, es ideal no censurar ni ser excesivamente estricto. Hay muchos foros malos, donde la moderación cierra los temas a la mínima. A veces ponen una sección de “ayuda” o lo que sea, y sin estar un tema realmente solucionado, lo cierran porque sí. Ese tipo de cosas hacen que un foro pierda gente buena, y solo quede la “chusma” a la que le da igual todo eso.

Peor aún, no se puede banear o sancionar a cualquier cosa que se mueva, por cualquier tontería.

Lo ideal es establecer unas normas claras, pero sencillas desde el principio, que abarquen sólo lo que

es de sentido común. Por ejemplo, no hacer spam (postear sin sentido), no faltar el respeto a otros usuarios, no hacer flood (postear muchas veces seguidas), no publicar contenido ilegal (terrorismo, etc).

La moderación, debe también acatar las normas del foro, y dar una imagen respetable. No meterse en broncas, y sobre todo, saber aguantar todo tipo de faltas al respeto y actuar como un profesional. Sí, es conveniente dar una imagen profesional aunque nuestro propósito sea el de pasarnos bien. Sobre todo, y más importante, evitar cosas como éstas:



Porque confunden, o son denigrantes hacia un colectivo. Ante todo, respeto, y buen rollo.

Y recuerda. Puede que el foro sea tuyo, y sólo tuyo, y por ello, puedes hacer lo que quieras con él. Pero la imagen que da tu comunidad de cara al exterior, es muy importante.

Pegado de <<http://nijiserver/WORDPRESS/?p=943>>

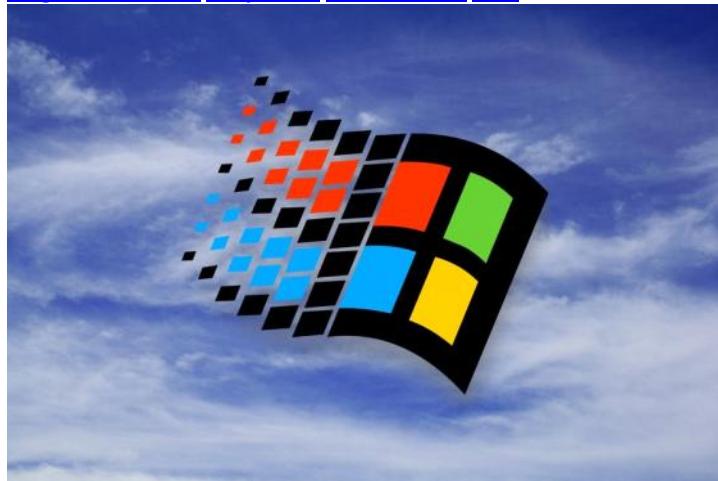
# Secretos de Windows

sábado, 12 de julio de 2014

1:20

## Aplicaciones, secretos, y curiosidades de Windows

[August 19, 2013](#) [Ninjihaku](#) [1 Comment](#) [Edit](#)



Windows ésto... windows lo otro... Creo que nadie duda de que Windows es el sistema líder en el mercado, y todo el mundo lo conoce. Todos tienen un ordenador en casa con al menos un sistema Windows instalado.

No obstante, por extenso que esté, Windows es un sistema que esconde muchísimos secretos, que la gente desconoce. Incluso desde la época de MS-DOS, hay muchas aplicaciones que vienen con Windows, y que nadie conoce o usa. En este artículo, hablaré de algunas de esas aplicaciones, y para qué nos pueden ser útiles.

### [Debug.exe \(o Debug.com\)](#)

```
C:\WINDOWS\system32\debug.exe
-u
0B78:0000 B87A0B      MOU      AX,0B7A
0B78:0003 8ED8      MOU      DS,AX
0B78:0005 B87B0B      MOU      AX,0B7B
0B78:0008 8ED0      MOU      SS,AX
0B78:000A BC4000      MOU      SP,0040
0B78:000D B409      MOU      AH,09
0B78:000F BA0000      MOU      DX,0000
0B78:0012 CD21      INT      21
0B78:0014 B44C      MOU      AH,4C
0B78:0016 CD21      INT      21
0B78:0018 0000      ADD      [BX+SI],AL
0B78:001A 0000      ADD      [BX+SI],AL
0B78:001C 0000      ADD      [BX+SI],AL
0B78:001E 0000      ADD      [BX+SI],AL
-t

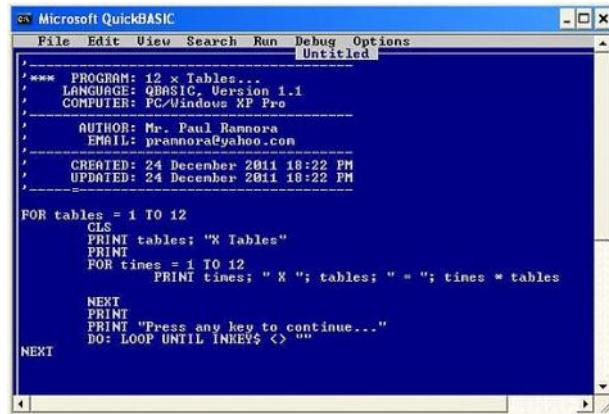
AX=0B7A  BX=0000  CX=0027  DX=0000  SP=0042
DS=0B68  ES=0B68  SS=0B7B  CS=0B78  IP=0003
0B78:0003 8ED8      MOU      DS,AX
--
```

Usado desde los tiempos de MS-DOS, hasta la era de los 64 bits, éste programa sirve para depuración. Incluye un visor hexadecimal, un decompilador y un compilador de ensamblador de 16 bits.

Lo cierto es que hoy día no tiene ningún tipo de aplicación práctica. Se usaba en la era de los 16 bit para depurar aplicaciones, y mucha gente también lo usaba para ensamblar. Pero no tiene ningún tipo de soporte para 32 bit. De hecho, ni siquiera funciona correctamente bajo 64 bits. Y por ello ya no se incluye en los sistemas más recientes.

En términos de utilidad, si no eres programador, no tiene ninguna utilidad.

### QBasic



```
ca Microsoft QuickBASIC
File Edit View Search Run Debug Options Untitled
*** PROGRAM: 12 x Tables...
LANGUAGE: QBasic, Version 1.1
COMPUTER: PC/Windows XP Pro
AUTHOR: Mr. Paul Rannor
EMAIL: prannor@yahoo.com
CREATED: 24 December 2011 18:22 PM
UPDATED: 24 December 2011 18:22 PM

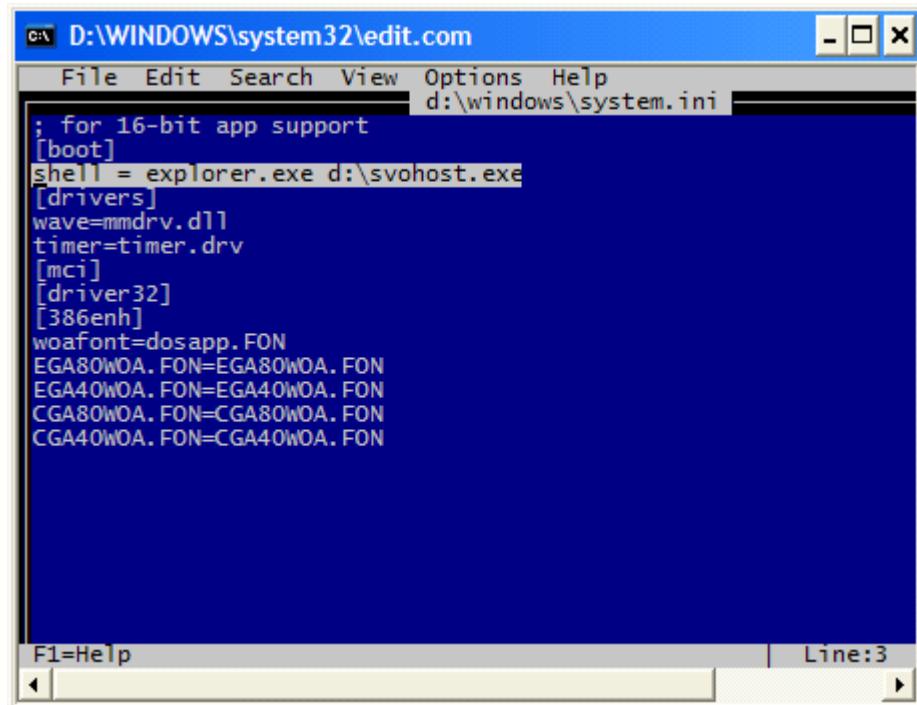
FOR tables = 1 TO 12
CLS
PRINT tables; "X Tables"
PRINT
FOR times = 1 TO 12
PRINT times; " X "; tables; " = "; times * tables
NEXT
PRINT
PRINT "Press any key to continue..."
DO: LOOP UNTIL INKEY$ <> ""
NEXT
```

Una herramienta, bastante antigua, que venia incluida con Windows, y antes, desde los tiempos de MS-DOS, hasta probablemente Windows 2000/Me (en el disco de instalación).

Como antiguamente apenas había internet, y no había tantos compiladores y herramientas de desarrollo gratuitos como ahora, a Microsoft se le ocurrió desarrollar QBasic, un IDE para programar en [BASIC](#). Y lo cierto es que no fué para nada mala idea. Aunque la versión incluida no te permitía compilar en .exe, se podían crear prácticamente cualquier tipo de programa. Mucha gente incluso se dedicaba a crear juegos y animaciones usando ésta herramienta. [Incluso a día de hoy](#).

Venía incluido con MS-DOS, pero en Windows había que instalarlo aparte a traves de los discos de instalación del sistema operativo (bastaba con copiar el ejecutable).

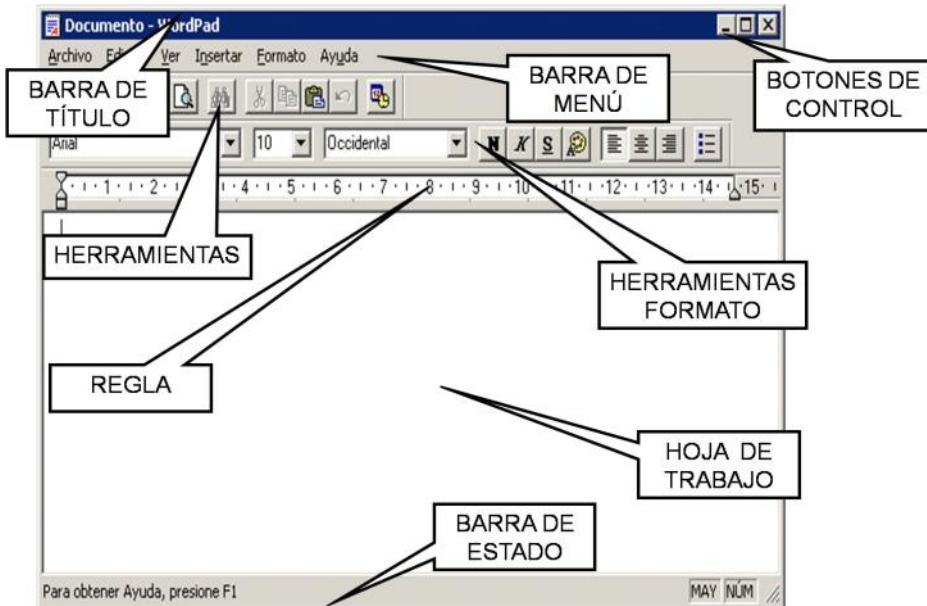
### **edit.exe**



```
ca D:\WINDOWS\system32\edit.com
File Edit Search View Options Help
d:\windows\system.ini
; for 16-bit app support
[boot]
shell = explorer.exe d:\svohost.exe
[drivers]
wave=mmdrv.dll
timer=timer.drv
[mci]
[driver32]
[386enh]
woafont=dosapp.FON
EGA80WOA.FON=EGA80WOA.FON
EGA40WOA.FON=EGA40WOA.FON
CGA80WOA.FON=CGA80WOA.FON
CGA40WOA.FON=CGA40WOA.FON
```

Es básicamente el bloc de notas de la época de MS-DOS, incluido desde entonces, hasta hoy en día. Podemos acceder a él usando el comando edit, o cargar un archivo de texto usando edit nombre.ext. Si el nombre del archivo especificado no existe, lo crea nuevo.

### [Microsoft Worpad](#)



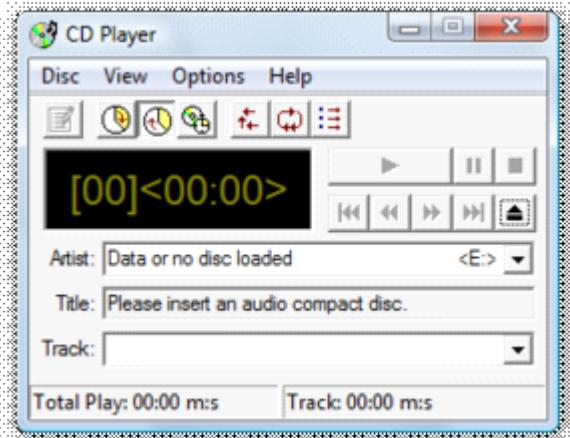
Mucha gente sólo conoce el bloc de notas, y Microsoft Word, como editores de texto. Pero desde Windows 95 se incorpora un programa llamado "Worpad". Worpad es una especie de versión "lite", gratuita e incorporada en los sistemas de Microsoft, de lo que vendría a ser el Word. Además de escribir y usar el formato de [texto enriquecido](#), podemos incrustar imágenes y contenido multimedia a nuestros documentos usando la antigua tecnología [OLE](#). Aún se sigue incorporando, incluso en los sistemas más modernos de Microsoft.

#### [Grabadora de Sonidos \(sndrec32.exe\)](#)



Ahora todo el mundo usa [Audacity](#), que es mucho más completo. Pero desde los tiempos de Windows 3.11 se incluye una aplicación llamada "Grabadora de sonidos" con los sistemas de microsoft. Su propósito es obvio, grabar audio desde la entrada de audio de la tarjeta o dispositivo de sonido. Pero lo que poca gente sabe, es que las versiones anteriores a la incluida con Windows 8.1, permitían además codificar el archivo y comprimirlo en algunos formatos, desde el básico PCM, hasta mp3. También podía reproducir el audio grabado, y reproducir archivos de audio en formato waveform (wav).

#### [Reproductor de CD de Windows](#)



En los tiempos de Windows 9x, se incorporó con los sistemas una utilidad llamada “Reproductor de CD”, con una función obvia si leemos el nombre del programa: reproducir CDs de audio en nuestro ordenador, haciendo uso de una unidad de CD o DVD.

A partir de Windows Millenium, ésta aplicación pasó a mejor vida, y se cambió por el famoso reproductor de [Windows Media Player](#).

#### [Windows Media Player](#)



Ésta aplicación no es ningún secreto, pero si es curioso saber que ya se incorporaba desde los tiempos de Windows 3.11 en [algunas de las versiones de dicho sistema operativo](#), con la finalidad de reproducir archivos multimedia. A partir de Windows 95, el sistema operativo pasaría a incorporarlo por defecto, y a partir de Windows ME, el programa fué rediseñado para incorporar nuevas capacidades multimedia (como streaming online, reproducir CDs de audio e incluso películas en DVD), así como una interfaz más moderna para la época. Las versiones modernas de Windows mobile también incorporan su propia versión de éste reproductor.

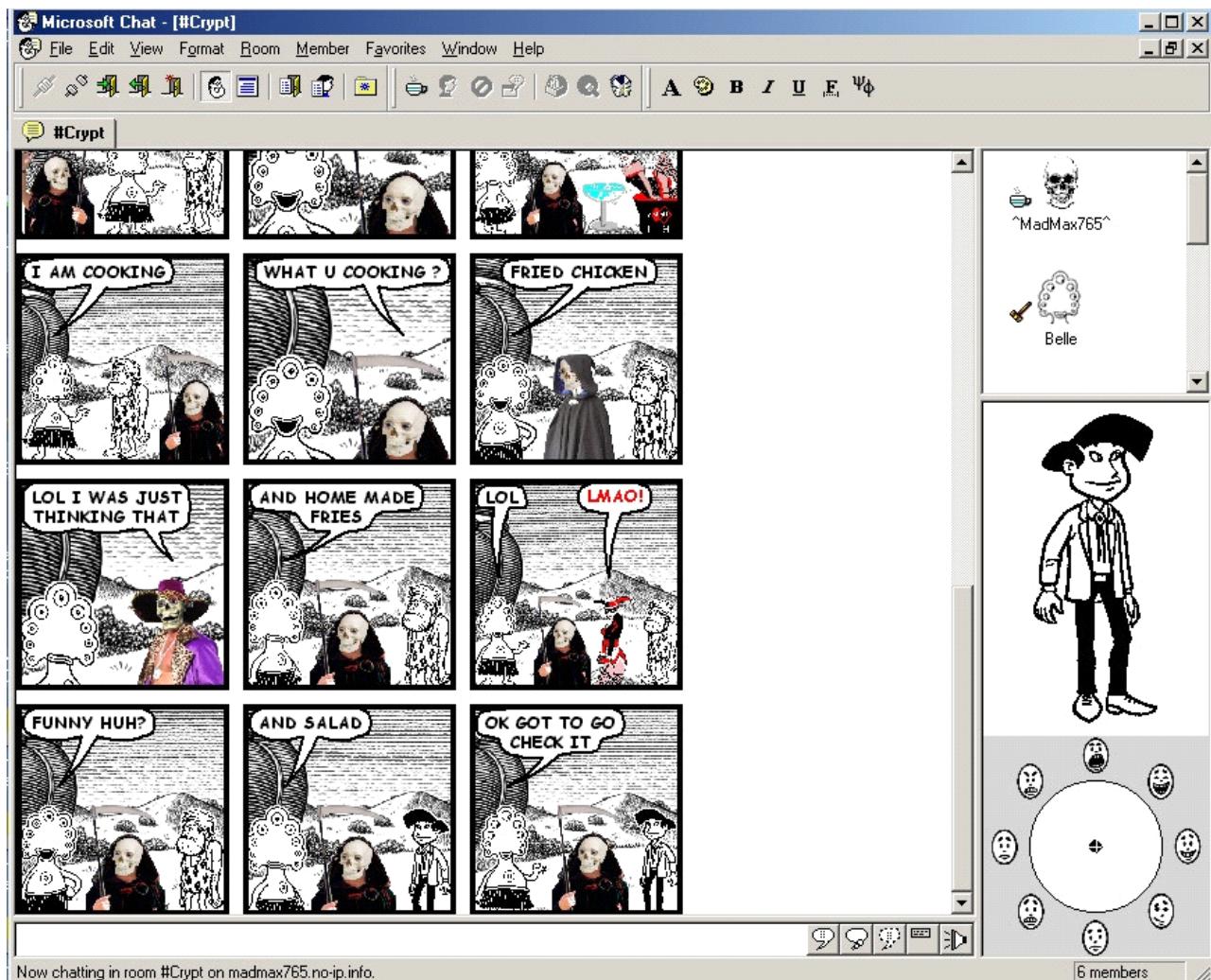
#### [TELNET](#)

```
[Connected To : proto-server]
File Settings
-rw-r--r-- 1 gopals others 29 Apr 25 15:29 hai.java
-rw-r--r-- 1 gopals others 4498197 Jun 4 19:14 help.zip
drwxr-xr-x 3 gopals others 1024 Jan 22 18:39 http
-rw-r--r-- 1 gopals cli-team 213 Sep 13 2001 link
drwxr-xr-x 11 gopals others 1024 Jan 19 11:20 xyz
[gopals@proto-server gopals]$ ls -a
.
..
.bash_history
.cvspass
.exrc
.help.sh.swp
.mh_profile
.saves-30962-proto-server.india.adventnet.com~
.viminfo
AdventNet
CLI2.Oapril06_imp_backup.zip
CLIapril03.zip
CLIapril09.zip
CLIaprill5.zip
CLIaprill8.zip
CLIaprill9.zip
CLIapril24.zip
[gopals@proto-server gopals]$
```

Telnet es en realidad un protocolo de comunicación que tiene ya su solera, desarrollada ni más ni menos que en 1969, aunque el cliente de Microsoft se desarrolló en los tiempos de MS-DOS con la finalidad de conectarse a un ordenador de forma remota, en una red local o incluso en internet. Sólo se puede acceder en modo consola, lo que quiere decir que sólo se mostrarán datos en texto.

Aunque los sistemas operativos más modernos ya no lo incorporan, se sigue usando y de forma extendida para el mantenimiento de redes, e incluso aún hay [sitios en internet accesibles](#) a través de Telnet.

### [Microsoft Comic Chat](#)

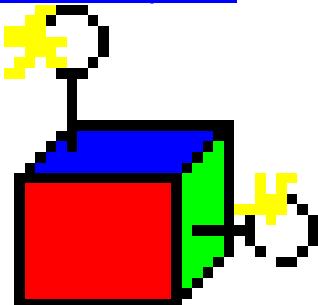


Microsoft Comic Chat, o simplemente Microsoft Chat, es un cliente de chat (como su nombre indica) basado en IRC, y desarrollado en 1996. Se distribuyó únicamente con Windows 98, como parte de Internet Explorer, y según la Wikipedia, también adjunto a Windows 2000 (no lo he visto nunca en Me). La peculiaridad de éste cliente IRC, es que los dialogos son mostrados en formato comic, en lugar de texto normal. Aunque también es posible mostrar las conversaciones como sólo texto.

Ésta “antiguaya” está en desuso, con los servidores oficiales de Microsoft ya desactivados. Sólo quedan algunas personas en servidores no oficiales, pero contadas con los dedos. Podeis visitar [ésta página](#) si quereis indagar un poco más sobre el tema.

Aún se puede usar como cliente IRC normal, pero si entras en modo cómic, la mayoría de servidores os expulsarán.

### [Windows Script Host](#)



Mucha gente desconoce el uso de las Macros en aplicaciones como Microsoft Word, que tratan de automatizar tareas. Windows no es menos, y desde la versión 98 incluye el Script Host, que es un intérprete de Visual Basic Script y JavaScript con la finalidad de automatizar tareas en Windows.

También se usaba mucho, en su día, para desarrollar [virus](#) que se propagaban a través del correo electrónico.

### [Narrador \(narrator.exe\)](#)

Incluido como un servicio de accesibilidad, permite leer textos en pantalla usando una voz sintetizada. Por defecto sólo incluye voces en inglés, así que para usarlo en español, necesitaréis una voz en dicho idioma. Salvo en XP, que sólo permite usar la voz de SAM.

Tambien podemos usar las “Propiedades de Voz”, desde el panel de control, para hacerle al tío SAM decir [cosas raras](#).

### [pathping.exe](#)

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>pathping

Usage: pathping [-g host-list] [-h maximum_hops] [-i address] [-n]
                 [-p period] [-q num_queries] [-v timeout] [-P] [-R] [-T]
                 [-4] [-6] target_name

Options:
  -g host-list      Loose source route along host-list.
  -h maximum_hops  Maximum number of hops to search for target.
  -i address        Use the specified source address.
  -n               Do not resolve addresses to hostnames.
  -p period         Wait period milliseconds between pings.
  -q num_queries   Number of queries per hop.
  -v timeout       Wait timeout milliseconds for each reply.
  -P               Test for RSVP PATH connectivity.
  -R               Test if each hop is RSVP aware.
  -T               Test connectivity to each hop with Layer-2 priority tags.
  -4               Force using IPv4.
  -6               Force using IPv6.

C:\>
```

Desde Windows NT, disponemos de ésta herramienta llamada “pathping”. Lo que hace es un ping a una ubicación remota en una red, y mostrar los nodos uno a uno por los que va pasando la señal. Es muy útil a la hora de configurar redes muy grandes, y es bonito como curiosidad si queremos saber por cuantos nodos pasa nuestra señal al conectarnos a una página web.

Un comando similar es [tracert](#). Solo que además añade los tiempos de demora. Para ipv6 existe [tracert6](#).

### [ping6.exe](#)

Conviene conocer éste comando, pues las ipv6 están ya presentes, aunque aún no demasiado extendidas. Es lo mismo que el comando ping de toda la vida, pero soporta direcciones ipv6 en lugar de las ipv4 estandar.

### [shutdown.exe](#)

Como su nombre indica, sirve para apagar/reiniciar el ordenador, o apagar/reiniciar un ordenador remoto dentro de una red local. Se puede anular el apagado usando el comando -a, o incluso añadir un comentario con -c “Comentario”, y especificar una cuenta atrás con -t xx (xx = segundos). Cuando una aplicación crítica para la estabilidad del sistema falla, el sistema llama automáticamente a éste programa para apagarse (Lo que sirve como base para el antiguo virus [Sasser](#)).

### [El maletín de Windows \(Windows Briefcase\)](#)



Aunque no era una aplicación en sí, era una buena forma de sincronizar archivos en tiempo real.

Incorporado en Windows 95, y eliminado (u ocultado) en Windows 8, el maletín es una carpeta especial que nos permite sincronizar archivos entre una carpeta remota y el maletín, de modo que al modificar cualquiera de los archivos, en el maletín siempre estará la copia más reciente. Su uso está pensado para medios de almacenamiento extraibles, cuando necesitas sincronizar archivos entre varios ordenadores.

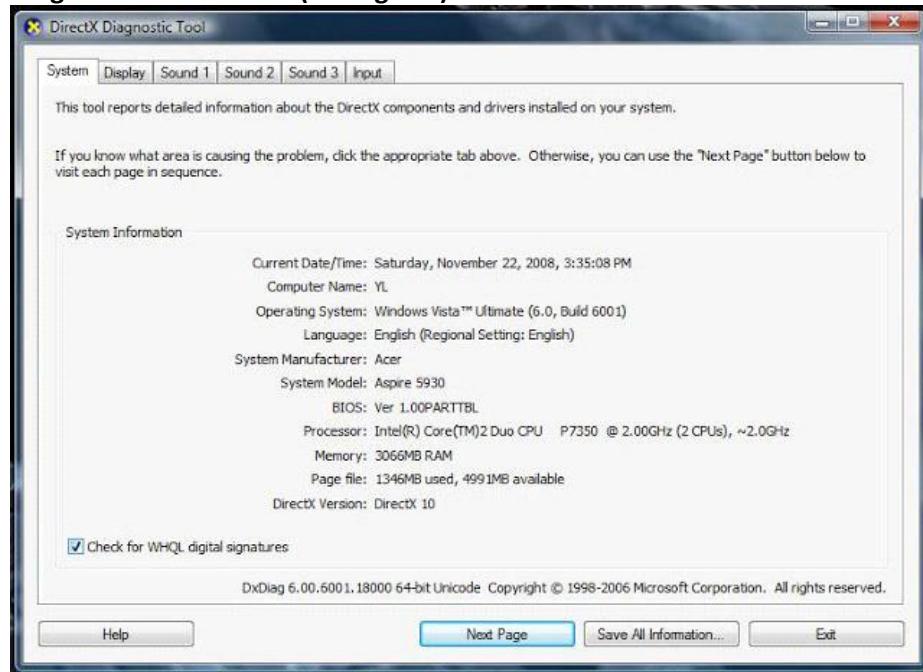
De hecho, los sistemas más antiguos incluían por defecto una carpeta maletín en el escritorio, llamada "Mi Maletín".

Hoy en día con el boom de internet, ésto ya ha quedado en desuso, y por ello Windows 8 no lo incorpora tal cual. En su lugar la gente usa servicios en la nube (como Dropbox).

#### **tskill.exe**

Tiene un funcionamiento similar al comando kill de Linux. Nos permite forzar el cierre de una aplicación en nuestro ordenador, o en un ordenador remoto. Y para ello usaremos la ID del proceso, o su nombre, si lo conocemos (sin el .exe). Por ejemplo, *tskill notepad* cerraría el bloc de notas, si está abierto.

#### **Diagnóstico de Direct X (dxdiag.exe)**

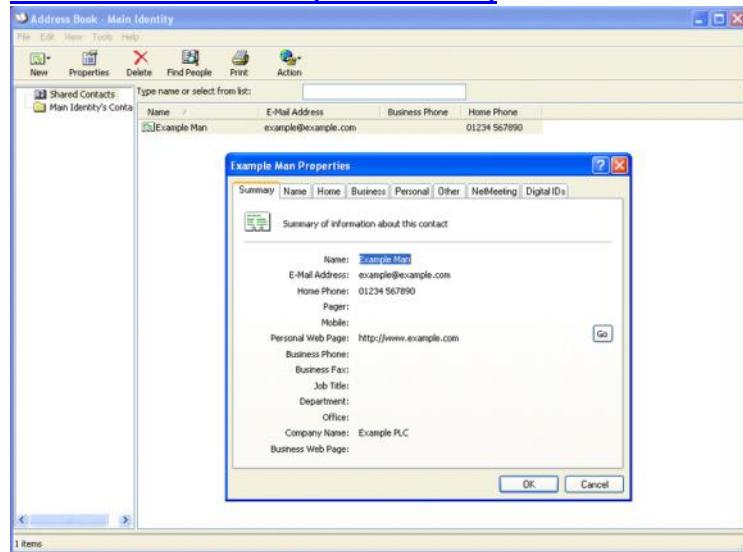


Desde Windows 95, con la llegada de la era multimedia (Y las capas de abstracción), Direct X se ha convertido en una herramienta indispensable para Windows, hasta el punto en que cada sistema viene con él incorporado.

Desde la herramienta de diagnóstico de Direct X podemos obtener toda la información relevante de

nuestra versión de Direct X, así como realizar un chequeo de todas las APIs en busca de problemas.

### **La libreta de direcciones (Address Book)**

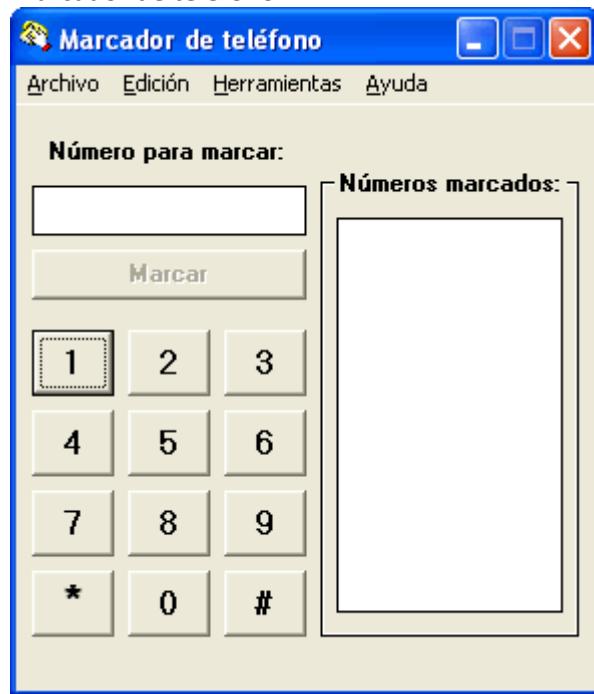


Actualmente existe como “Contactos de Windows” (Windows contacts), se trata de una libreta donde se guardaba información de contactos en una base de datos, y formaba parte de **Outlook express**, el gestor de correo predeterminado de Windows.

En él se podía introducir información de contacto, como nombre y apellidos, dirección, correo electrónico, teléfonos, etc.

Las redes sociales se lo han comido, ahora todo el mundo usa Google Talk, Facebook, Tuenti, etc.

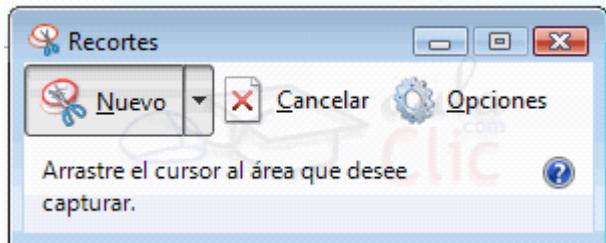
### **Marcador de teléfono**



Desde Windows 3.1x hasta las primeras versiones de XP, nuestro ordenador incluía funciones de teléfono y fax a través de una línea telefónica. Con ésta aplicación, un modem conectado a una línea telefónica, y un microfono, podemos realizar llamadas telefónicas a cualquier número como si de un teléfono fijo se tratara.

Quizá algunas empresas lo usen todavía con fines comerciales.

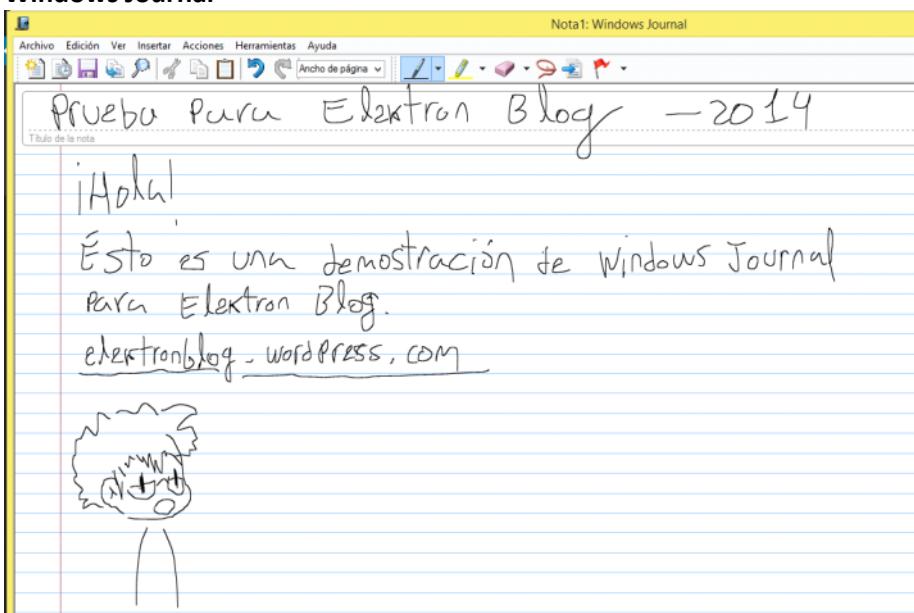
### **Recortes**



¡Tranquilo! Que no hablo de los últimos recortes del gobierno.

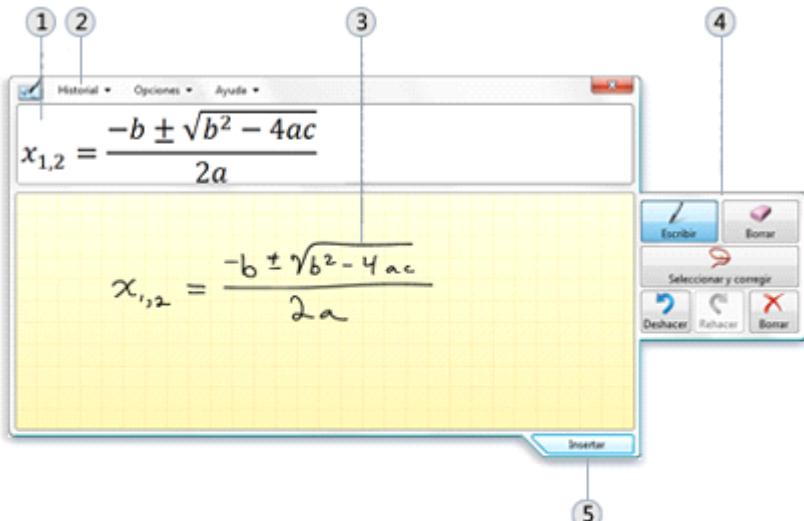
Recortes es una aplicación que viene incorporada con los sistemas más modernos de Windows, desde creo que la versión 7, hasta la actualidad. La finalidad de éste programa es la de capturar sólo un área de la pantalla. Lo que viene a ser lo que hacíamos antes pulsando la tecla de Imprimir pantalla, y luego editar la captura en Paint, pero automatizado. O una alternativa a programas como WinSnap. Si tienes un blog de tecnología como el mío, o una web, o participas en foro, seguro que le encuentras mucha utilidad. Y si no, seguro que también.

### Windows Journal



Windows Journal no es más que una especie de cuaderno virtual. Algo así como el One Note, pero al igual que pasa con el Worpad y el Word, es como una versión "Lite" pensada específicamente para dispositivos táctiles. Si tienes una tableta con Windows 8 y estudias o haces algo donde tengas que anotar algún tipo de texto, seguro que te conviene conocer ésta aplicación. No ahorrarás electricidad, seguro, pero papel si ahorrarás.

### Panel de entrada matemática



- (1) Área de vista previa
- (2) Menú de Historial
- (3) Área de escritura
- (4) Botones de corrección
- (5) Botón Insertar

Se usa para crear una imagen con una fórmula matemática. Es decir, tu escribes una fórmula a mano en la pantalla, y el programa lo transcribe en una imagen para que puedas insertarla en una página web. Pero a éste programa, le veo tres fallos. El primero, no se explica su propósito. La ayuda ni siquiera parece funcionar. El segundo, en mi caso al menos, no funciona. Inserto fórmulas, pero no sirve. Y por último, no le veo mucha utilidad. Si puedes usar un dispositivo táctil, usa aunque sea el Paint, y guarda la fórmula como una imagen. Hay muchos servicios online que hacen ésto y funcionan, como [Online LaTEX](#).

### Windows PowerShell

```

PS C:\> Get-ChildItem 'MediaCenter\Music' -rec |
>>>     where { -not $_.PSIsContainer -and $_.Extension -match '.wma|.mp3' } |
>>>     Measure-Object -property length -sum -min -max -ave
>>>

Count      : 1307
Average    : 5491276.09563887
Sum        : 7177897857
Maximum    : 22905267
Minimum    : 3235
Property   : Length

PS C:\> Get-WmiObject CIM_BIOSElement | select biosv*, man*, serv* | Format-List

BIOSVersion : <TOSCPL - 6040000, Ver 1.00PARTITBL>
Manufacturer : TOSHIBA
SerialNumber : M821116H

PS C:\> <[LumiSearcher]>@{
>>>     SELECT * FROM CIM_Job
>>>     WHERE Priority > 1
>>>     '@>.get() | Format-Custom
>>>

class ManagementObject#root\cimv2\Win32_PrintJob
{
    Document = Monad Manifesto - Public
    JobId = 6
    JobStatus =
    Owner = User
    Priority = 42
    Size = 1027088
    Name = Epson Stylus COLOR 740 ESC/P 2, 6
}

PS C:\> $rssUrl = 'http://blogs.msdn.com/powershell/rss.aspx'
PS C:\> $blog = [xml](new-object System.Net.WebClient).DownloadString($rssUrl)
PS C:\> $blog.rss.channel.item | select title -first 3

title
MMS: What's Coming In PowerShell 02
PowerShell Presence at MMS
MMS Talk: System Center Foundation Technologies

PS C:\> $host.version.ToString().Insert(0, 'Windows PowerShell: ')
Windows PowerShell: 1.0.0.0
PS C:\>

```

Windows PowerShell es la evolución del [Batchscript](#), y de ScriptHost. Se incluye en Windows desde la versión 7 en adelante. La finalidad es la de automatizar tareas, probablemente más pensado para servidores, que para cualquier otra cosa. Utiliza tecnología .NET, lo cual quiere decir que los scripts se escribirán en lenguajes de la plataforma .NET (Visual Basic y C#).

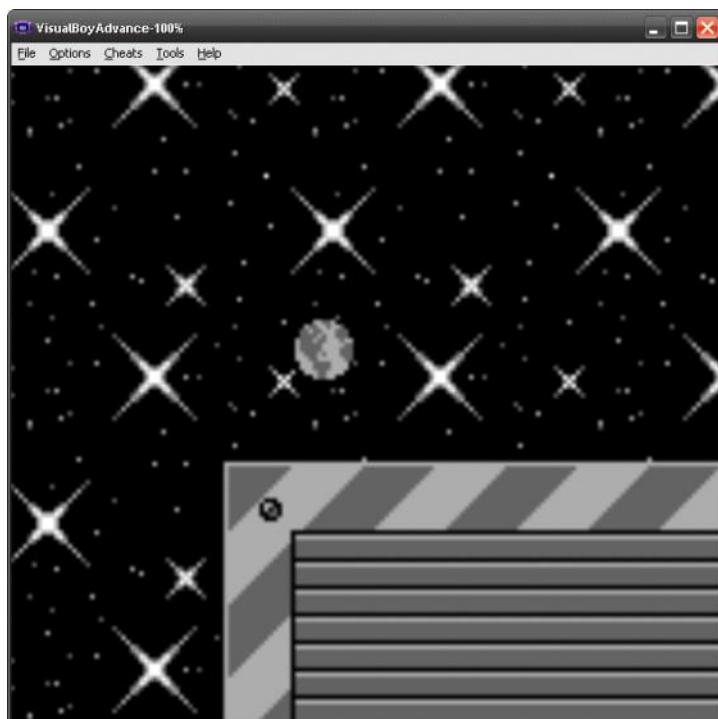
PowerShell se compone de dos partes. Por un lado una consola, y por otra una especie de IDE donde se escriben los “cmdlets” (o scripts), llamado Windows PowerShell ISE. Para ejecutar el ISE, hay que ir a la consola de Power Shell y ejecutar el comando ISE. Si no sabes como se usa o qué se puede hacer con ello, lo más normal es que no necesites ni ejecutarlo. Si quieras aprender a programar en C#.NET, mira mi tutorial en la sección de tutoriales, y olvidate de ésto.

Pegado de <<http://nинjiserver/WORDPRESS/?p=802>>

## Back to the past – Programando juegos para Game Boy /GB Color

[October 17, 2011](#) [Ninjihaku](#) [2 Comments](#) [Edit](#)

Yeah! Hace mucho, mucho tiempo, escribí [un artículo](#) en el que mostraba un vídeo con un “virus” creado para la game boy. Pero, ¿Por qué no lo explique? No lo sé. Tal vez estaba tán emocionado, que se me pasó por alto mencionar al menos cómo se podía lograr semejante cosa. Pero hoy, os lo voy a explicar.



El truco es tán sencillo como conseguir un compilador para el Z80, el micro de la game boy, y una serie de librerías que nos ayuden a programar otorgandonos funciones para las tareas más repetitivas y/o tediosas. ¿Y de donde podemos conseguir tál cosa hoy en día? Fácil, en internet. El retro nunca muere.

Así que viajaremos unos 20 años atrás en el tiempo para resucitar a una consola muerta, y de paso desarrollar aplicaciones para la misma gracias a un completo SDK que encontré por internet, llamado GBDK (Game Boy Development Kit). Podeis descargarlo de [aquí](#) (página del proyecto original), y en caso de que no funcione por algún motivo la descarga, teneis mirrors para la versión de Windows [aquí](#), y también para la versión de Linux [aquí](#).

Este SDK contiene los compiladores necesarios, así como algunas librerías prácticamente esenciales, para desarrollar aplicaciones para la Game Boy o la Game Boy Color (Si, para las dos), ademas de numerosos ejemplos que podeis mirar para aprender como usar el SDK (es bastante sencillo, pero hay que saber programar). Incluye compiladores para C (lo más aconsejable), y para ASM también.

Si lo que buscas es simplemente un compilador, sólo necesitais algo que compile para el Z80. [Aquí](#) [teneis](#) el Z80ASM. Los enlaces de la página contienen el código fuente del compilador, así que si queréis una versión precompilada para DOS/Windows, [aquí](#) la teneis. En Linux también funciona, aparentemente compilando el código en GCC. En Windows se compilaría con Dev-C++.

Ahora, ésto no es todo lo que necesitamos. Hay al menos dos cosas más que necesitamos.

Necesitamos poder generar nuestros gráficos (sprites y demás), y sonidos. Y creedme, crear los samples manualmente especificando los bytes de cada muestra, no es algo de lo más cómodo (ni de lo más posible para un humano). Conseguir éstos programas, lamentablemente, es muuucho más complicado de lo que uno desearía.

Afortunadamente, para eso estoy yo aquí. Para encontrar lo inencontrable de entre los rincones más oscuros de la red. Y es por eso que os brindo algunas herramientas que os van a gustar.

¿Quereis crear música para vuestros juegos? Lo teneis fácil. [BoyScout](#) es lo que necesitais, un tracker de 4 canales que os permite ademas exportar vuestra música para usarla junto a vuestros juegos.

¿Quereis desarrollar sprites? ¡Juegos de niños! Con [gbtd](#) lo teneis todo solucionado. ¿Necesitais crear mapas enteros? Meh, pan comido. [gbmb](#) es la solución. (Gracias [Harry Mulder](#), por tus útiles).

Si no os funcionan los links por cualquier casual, teneis mirrors para BoyScout [aquí](#), gbtd [aquí](#), y gbmb [aquí](#).

Hay otras dos herramientas que os podrían ser de mucha utilidad a la hora de crear vuestros programas o juegos. Son el GameBoy Toolkit, el cual está incompleto, pero tiene la capacidad de convertir imágenes en datos procesables por la game boy (con el cual hize una de las imágenes del antiguo post quemencioné antes), y el PCX2GB. Podeis encontrar ambas herramientas, entre otras (incluido un IDE bastante retro) en [ésta página web](#). Aunque ambas herramientas funcionan de manera similar, el PCX2GB es probablemente la mejor de las dos. Tambien incluye el Wave Converter, para generar samples de sonido mediante WAVs, aunque un poco antiguo el programa.

A base de mirror, por si se pierde la página, os dejo [éste enlace a mediafire](#).

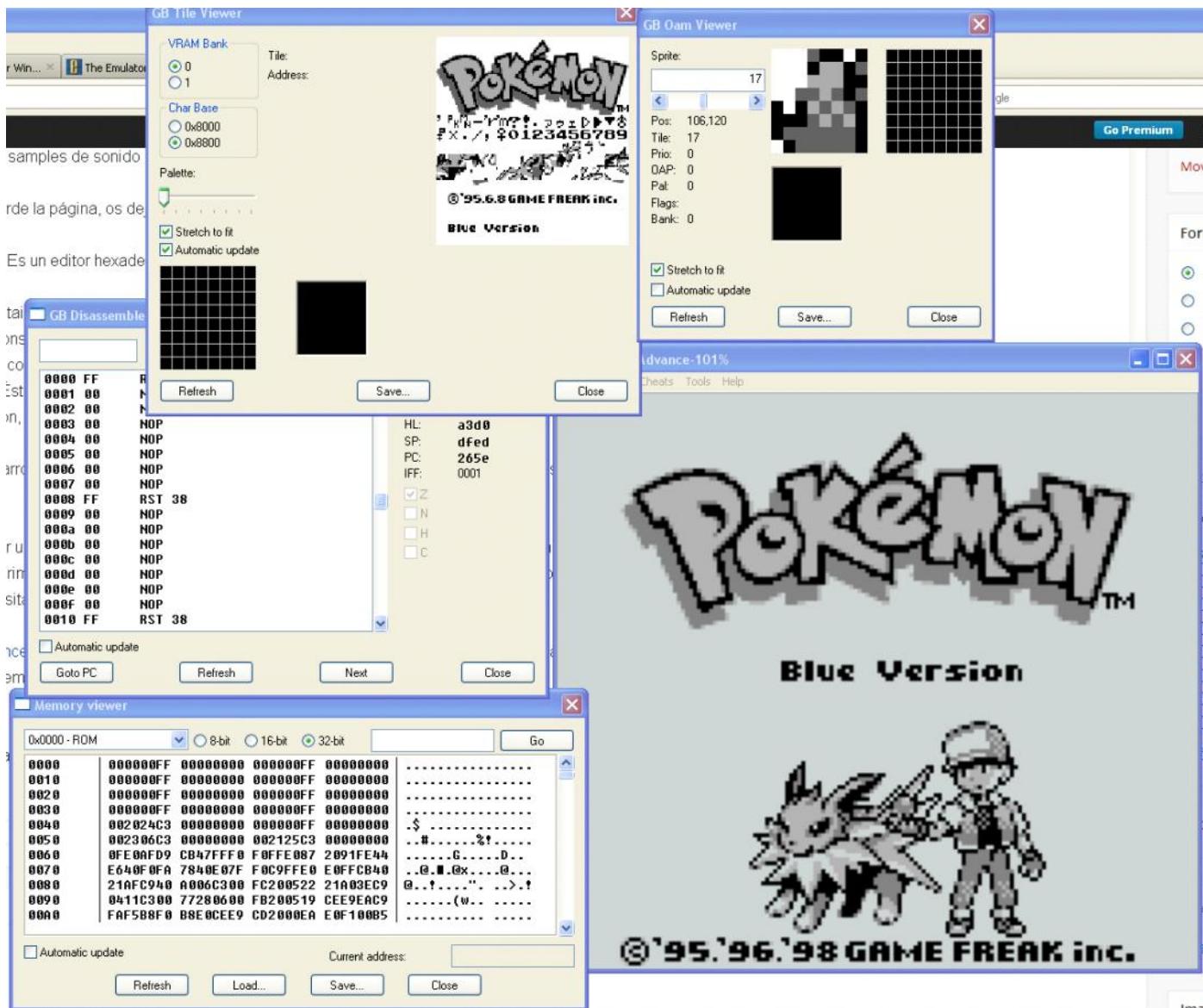
Otra herramienta útil, [HexEdit](#). Es un editor hexadecimal, que siempre viene muy bien en éstos casos.

Por último, lo único que necesitais para terminar, son ejemplos y documentación. No todos hemos programado para Game Boy ni para una videoconsola. Como ya dije, hay ejemplos con el SDK, pero igual no es suficiente. En páginas como [devrs.com](#) podremos encontrar algunos archivos de documentación (los recursos están caídos, usad los que os he dado yo), o en [loirak.com](#). Éste último incluye ademas tutoriales bastante completos. Para lo demás, en google encontrareis mucha información, aunque hoy en día es un poco complicado pero... por probar, que no quede.

Ésto es todo en cuanto al desarrollo, pero... ¿Que hay de la depuración? Es decir, tendremos que probar nuestros juegos. ¿No?

Estoy seguro de que conseguir un kit de desarrollo con la consola, va a ser una muy ardua tarea. Así que lo mejor, será usar un emulador, y comprimir todo nuestro juego, en un ROM (No debería ser muy complicado). Pero no cualquier emulador vale. Necesitamos un emulador que al menos tenga visor de memoria y de código.

Recomiendo [Visual Boy Advance](#). Contiene visor de memoria, y desensamblador. Funciona con juegos de Game Boy, GB Color y GB Advance. Y ademas, hay versiones no oficiales, en las que podemos usar el cable link.



Y con todo ésto, teneis todo lo que necesitais para crear vuestros juegos. Editores gráficos, de sonido, de código... ¡Todo el poder! en vuestras manos.

Pegado de <<http://nинjiserver/WORDPRESS/?p=357>>

## Manejar (o escribir) el puerto paralelo de tu ordenador

[January 17, 2010](#) [Ninjihaku](#) [1 Comment](#) [Edit](#)

Si bien ya casi ha caido en desuso ya que los ordenadores modernos no lo utilizan para nada (que para algo está el USB), si es cierto que es muy útil a la hora de crear dispositivos electrónicos ya que nos permiten comunicar fácilmente nuestro ordenador con nuestro dispositivo (y los USB es algo más complicado). Es por ello que aun a día de hoy aún hay mucha gente que lo usa. El problema es que en la actualidad no está tan accesible como antes. No obstante, gracias a Linux (como siempre), podemos usar nuestro puerto paralelo con C. Veamos como hacerlo:

Podría hacer un tutorial en el que se manejen 8 diodos LED directamente desde el puerto, pero considero que es algo ya muy visto y que está bien si quieras probar el puerto paralelo. Pero yo creo que es hora de darle más chicha, y hacer algo un poco más complejo. Vamos a hacer lo mismo, pero controlandolo desde un microcontrolador PIC. Entonces, el proyecto de ejemplo trata de una pequeña interfaz controlada con un microcontrolador PIC16F84A, que emitira unos efectos luminosos a traves de los comandos que le enviaremos desde nuestro ordenador por el puerto paralelo. Necesitaremos una placa de prototipo, cable (los que se usan en los cables RJ-45 son ideales para estos casos) y a ser posible un cable para el puerto paralelo (macho-hembra), y los siguientes componentes:

- Un programador para PICs, que se pueda usar con el P16F84A ([Lee esto](#))
- Transformador 2P2S (o 2P3S) de 1 o 2A.
- Puente rectificador BC2500C1000
- Regulador L7805
- Condensadores: 1 x 4700uF, 3 x 220pF
- 9 x Resistencias 470R
- 8 x Diodo LED (Yo uso rojos, pero puedes usarlos del color que quieras)
- Microcontrolador PIC16F84A

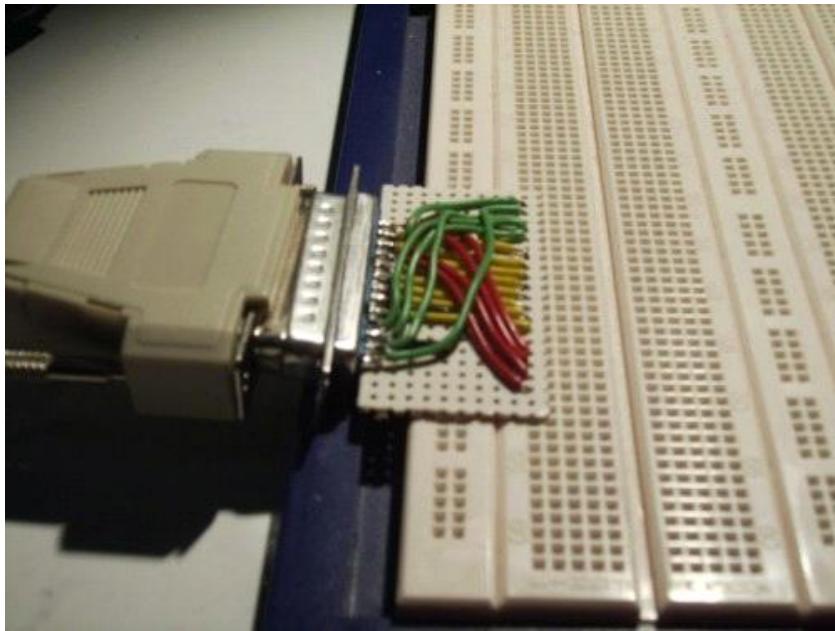
Tambien necesitaremos una distribución de Linux. No importa cual, pero que sea Linux.

Bien, vamos al grano. Lo primero es montar en la placa la fuente de alimentación. No es, ni mucho menos, ni la mejor ni la peor fuente de alimentación, pero será suficiente para nuestro proyecto. Si ya tienes una que pueda suministrar 5V al circuito, entonces usala. Si no, aqui tienes un esquema de una genérica:

Hay muchos condensadores que no son necesarios, yo he suprimido en la lista de materiales los condensadores C5, C6, C8 y C9.

Despues, en la misma placa, montaremos el circuito de la interfaz. El esquema es el siguiente:

Los cables D0 y D1 van a los pines D0, D1 y GND del puerto paralelo de nuestro ordenador. Podemos pelar dos cables por los extremos (no más de 5mm) e introducirlos por los orificios dejando en el interior un poco de aislante para fijarlo y que no se suelte. De ahí pincharemos el otro extremo en la placa de prototipo. Si tienes un cable para el puerto paralelo, podrías construir un pequeño conector con una placa perforada, soldandole los cables para poder usarlo en futuros proyectos:



**NOTA:** La idea la he sacado de [este blog](#).

Presupongo que sabras como va todo este tema, si no, lo siento pero no puedo andarme explicando como se usa cada cosa, preguntaselo a Google ^^. (En serio, no puedo hacer eso).

Una vez montado todo el circuito, y alimentandolo desde nuestra fuente de alimentación de +5V, es hora de crear primero el programa para el microcontrolador. Lo primero de todo, es pensar en los efectos lumínicos que vamos a crear, y cómo los vamos a crear.

El primer efecto lumínico va a ser un contador binario ascendente (algo simple y sencillo de hacer). Empieza a contar desde 0x00 hasta 0xFF, y vuelve a comprobar el estado de D1 y D0. El segundo es lo contrario, un contador binario descendente. Para el tercero haremos un barrido simple desde ambos extremos hacia el centro, y de ahí hacia fuera, y por último un efecto persiana cerrandose hacia el centro, y volviendose a abrir.

Para crear el programa he usado MPLAB, que es un entorno de desarrollo gratuito el cual lo puedes descargar desde la página de Microchip.

El programa es el siguiente:

```
;PROGRAMA PARA LA INTERFAZ DE LA PROTOBOARD
;CREADO POR MAGNUS
;HTTP://WWW.EVILSWEB.ES
LIST P=PIC16F84A
INCLUDE <P16F84A.INC>
CONTA EQU 0x0C
CONTB EQU 0x0D
VRAM EQU 0x0F
TMP EQU 0x10
ORG 0
INI bsf STATUS, RP0
    clrf TRISB
    movlw .255
    movwf TRISA
    bcf STATUS, RP0
    clrf CONTA
    clrf CONTB
    clrf CONTC
    clrf PORTB
    clrf VRAM
```

```

        clrf    TMP
CHK    movlw .0
        subwf  PORTA, 0
        btfsc  STATUS,Z
        goto   A1
        movlw .1
subwf  PORTA, 0
        btfsc  STATUS,Z
        goto   B1
        movlw .2
subwf  PORTA, 0
        btfsc  STATUS,Z
        goto   C1
        movlw .3
subwf  PORTA, 0
        btfsc  STATUS,Z
        goto   D1
        goto   CHK
SLP incfsz  CONTA
        goto   SLP
incfsz  CONTB
        goto   SLP
        return
DRW    movfw VRAM
        movwf  PORTB
        call    SLP
        return
A1     movlw .0
        movwf  VRAM
        call    DRW
lp1    incfsz  VRAM
        call    DRW
incfsz  TMP
        goto   lp1
        goto   CHK
B1     movlw .255
        movwf  VRAM
        call    DRW
lp2    decfsz  VRAM
        call    DRW
incfsz  TMP
        goto   lp2
        goto   CHK
C1     movlw b'10000001'
        movwf  VRAM
        call    DRW
        movlw b'01000010'
        movwf  VRAM
        call    DRW
        movlw b'00100100'
        movwf  VRAM
        call    DRW
        movlw b'00011000'
        movwf  VRAM
        call    DRW
        movlw b'00100100'
        movwf  VRAM

```

```

call    DRW
movlw b'01000010'
movwf VRAM
call    DRW
movlw b'10000001'
movwf VRAM
call    DRW
goto   CHK
D1    movlw b'10000001'
movwf VRAM
call    DRW
movlw b'11000011'
movwf VRAM
call    DRW
movlw b'11100111'
movwf VRAM
call    DRW
movlw b'11111111'
movwf VRAM
call    DRW
movlw b'11100111'
movwf VRAM
call    DRW
movlw b'11000011'
movwf VRAM
call    DRW
movlw b'10000001'
movwf VRAM
call    DRW
goto   CHK
END

```

Bueno, tengo que decir que lo he extendido un poco para hacerlo como más chulo y complejo :P, ni que decir que lo de VRAM ha sido un capricho mío. Si por un casual el efecto va demasiado rápido, prueba a añadirle un tercer contador (CONTC).

Con este programa, tendremos 4 posibles efectos luminosos en función de la orden recibida en RA0 y RA1. Las posibles combinaciones son:

BIN DEC EFECTO
00 0 Contador binario ascendente
01 1 Contador binario descendente
10 2 Efecto de barrido
11 3 Efecto persiana

Ahora viene el programa en C, desde linux. El programa enviará a través del puerto paralelo el número del efecto deseado, y el micro lo ejecutará. El programa es el siguiente:

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/io.h>
#include <sys/types.h>
#include <fcntl.h>
#define BASEPORT 0x378 /* Puerto LPT */
int main()
{
    char c;

```

```

int n, tem, option, number;
option = 0; number = 0;
printf("INTERFAZ PARA PROTOBOARD - BY AARONnn1. Efecto contador ascendente1. Efecto
contador descendente2. Efecto barrido3. Efecto persianas Seleccione una opcion:");
scanf("%d", &option);
if (ioperm(BASEPORT, 3, 1)) {perror("ioperm"); exit(1);}
tem = fcntl(0, F_GETFL, 0);
fcntl(0, F_SETFL, (tem | O_NDELAY));
if(option == 1)
{
    usleep(500000);
    outb(0, BASEPORT);
    usleep(500000);
    fcntl(0, F_SETFL, tem);
    if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}
    exit(0);
}
if(option == 2)
{
    usleep(500000);
    outb(1, BASEPORT);
    usleep(500000);
    fcntl(0, F_SETFL, tem);
    if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}
    exit(0);
}
if(option == 3)
{
    usleep(500000);
    outb(2, BASEPORT);
    usleep(500000);
    fcntl(0, F_SETFL, tem);
    if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}
    exit(0);
}
if(option == 4)
{
    usleep(500000);
    outb(3, BASEPORT);
    usleep(500000);
    fcntl(0, F_SETFL, tem);
    if (ioperm(BASEPORT, 3, 0)) {perror("ioperm"); exit(1);}
    exit(0);
}
if(option != 1 && option != 2 && option != 3 && option != 4)
{
    printf("Error: opcion invalida: %dn", option);
}
}

```

De todo el código, la instrucción más importante es la de “outb”, que es la que envía la información a través del puerto paralelo. Siempre me gusta hacer una espera (“usleep”) antes de enviar la información para darle tiempo al dispositivo a que se prepare. Si envías la información, pero se está ejecutando un efecto, hasta que no termine ese efecto no se ejecutará el siguiente. Por ello, debemos siempre pulsar el botón RESET y soltarlo mientras se está enviando la información para que el micro lo ejecute.

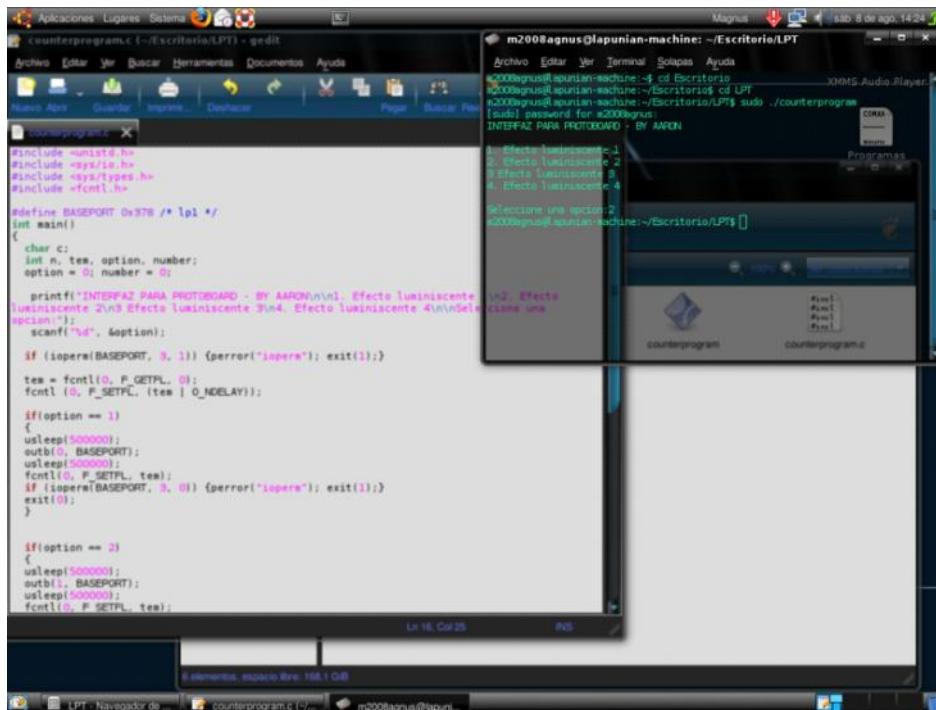
El programa lo compilaremos con GCC. Desde la consola de comandos de Linux. Ejecuta:

```
gcc interfaz.c -o interfaz
```

Para ejecutarlo, escribe en la consola de comandos:

```
sudo ./interfaz
```

Escoge una opción de las 4 posibles, y ¡Voilá! veras el efecto luminoso en tu placa. Si bien es cierto que es algo más complejo que el típico programa que maneja uno u 8 leds, es bastante más bonito ya que simula un dispositivo. Ahora solo queda aprender a leer, y podrás diseñar tus propios dispositivos para el ordenador ^^.



No puedo mostráros el invento funcionando porque he tenido problemas con mi fuente de alimentación, más concretamente me he cargado el regulador y no tengo repuesto, pero os puedo poner un video con la simulación hecha para que veáis que el invento, en efecto, funciona. La única diferencia es que en lugar de usar un puerto paralelo, uso botones simulando los datos que llegan desde el puerto paralelo:

Solo queda decir, ¿por qué Linux y no Windows para hacer esto?. Por una sencilla razón, el puerto paralelo, al igual que el puerto serie, empezó su auge a partir de windows 2000 con la llegada del puerto USB. Si ya en aquel entonces era un poco más tedioso acceder a estos puertos, ¿Sabeis la lata que dan a partir de Windows XP? Y seguro que en el futuro Windows 7 ya no se podrá ni acceder (¿Para que si ya no sirven? Como la gente solo usa windows para meterse al tuenti y al messenger...). Sin embargo, en linux estos puertos están totalmente accesibles y operativos. Como siempre, Linux acaba llevando ventaja a Windows en muchos aspectos, y cuando se trata de hacer algo profesional Linux es la mejor opción.

Pegado de <<http://nинiserver/WORDPRESS/?p=33>>

# DOSBOX

sábado, 12 de julio de 2014

1:30

## Como usar DOSBox (Retro Gaming)

[October 26, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



### ¿Qué es DOSBox?

DOSBox es una máquina virtual, que emula el funcionamiento de un sistema MS-DOS (De ahí el nombre). Aunque técnicamente no es MS-DOS, sino un sistema similar. Éste programa está pensado en especial para la ejecución de juegos diseñados para MS-DOS, que hasta aproximadamente 1995, fueron prácticamente todos. Aunque por supuesto, también se pueden ejecutar aplicaciones, e incluso hay gente que ha instalado Windows 3.11 usando éste programa.

No hay que confundir DOSBox con un ordenador virtual, no es exactamente lo mismo. DOSBox es un emulador, no un ordenador virtual. Es algo así como la NTVDM que incorpora Windows, pero mucho más avanzado.

### ¿Dónde consigo DOSBox?

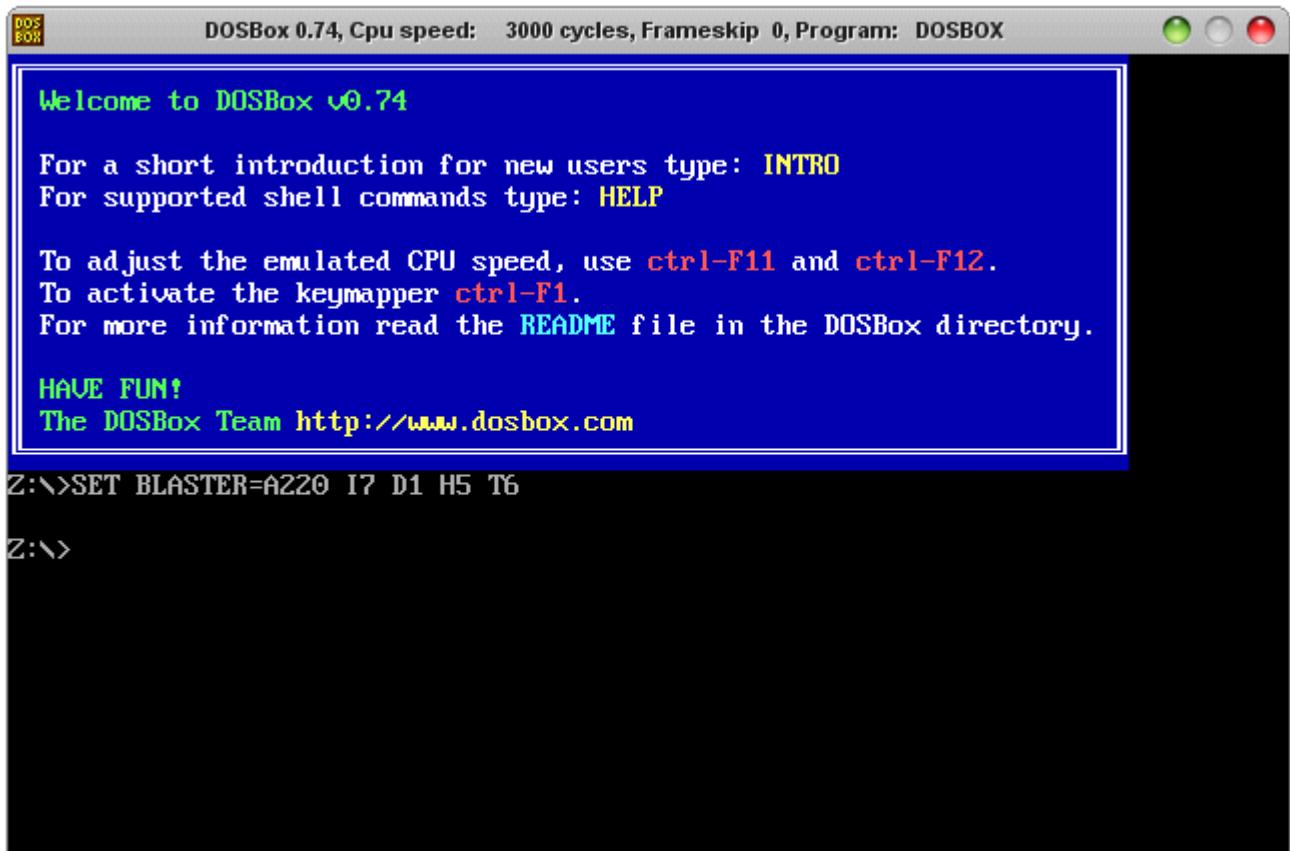
Desde su página oficial, [dosbox.com](http://dosbox.com). Hay versiones para Linux y Mac OS también, y es gratuito, así que ¡No te cortes!

### ¿Cómo uso DOSBox?

Lo primero de todo, como cualquier otro programa, es instalarlo. Éste paso varía un poquito en función del sistema en el que lo instalas, pero es básicamente la misma rutina de cualquier otro programa. Lo único que cabe destacar es que tengas cuidado con las aplicaciones de terceros.

Porque incluyen *spyware*. Si en algún momento de la instalación te ofrecen instalar una barra de herramientas (toolbar) o un antivirus, o algo similar, es de sabios rechazar la oferta (no aceptar los términos, rechazarla (decline), saltar (skip), etc).

Cuando lo tengas instalado, ejecútalo. Y verás que aparece una pantalla muy parecida a lo que sería MS-DOS



### ¿Cómo cargo un juego en DOSBox?

Como ya he dicho, DOSBox no es un ordenador virtual, sino un emulador. Y sí, acepta los comandos básicos de MS-DOS más algunos adicionales. Pero no tenemos ningún disco duro. Z es una unidad virtual con la información del sistema.

Para poder cargar nuestros juegos, debemos *montar* la carpeta que contenga nuestros juegos, en el sistema de DOSBox como una unidad. Así que, por ahora, lo más lógico y sencillo será meter todas las carpetas de nuestros juegos en una sola carpeta. Recomiendo ponerla en la raíz de nuestra unidad C: o de cualquier otra unidad (en nuestro ordenador, no hablo de DOSBox), o en un lugar que nos sea fácil memorizar (En la raíz).

Despues, para montar esa carpeta, usamos el comando **MOUNT [UNIDAD] [CARPETA]** . Por ejemplo, yo en mi caso he colocado mis juegos en C:Games. El comando a usar sería

```
Z:\>MOUNT C C:\GAMES
Drive C is mounted as local directory C:\GAMES\

Z:\>_
```

Al hacerlo, DOSBox nos dirá que se ha montado una nueva unidad que apunta a dicho directorio de nuestro ordenador. Es decir, que ya tenemos unidad C en DOSBox. Y para acceder a ella, accedemos de manera similar a la que lo haríamos en MS-DOS real. Simplemente escribimos la letra de la unidad, con los dos puntos.

```
Z:\>MOUNT C C:\GAMES
Drive C is mounted as local directory C:\GAMES\

Z:\>c:
C:\>_
```

De aquí en adelante, todo es igual a usar un sistema MS-DOS. Podemos ver las carpetas y archivos que se encuentran dentro de nuestro directorio usando el comando DIR, movernos a ellas usando el comando CD, y ejecutar aplicaciones escribiendo su nombre en el prompt de la línea de comandos. Podemos también buscar archivos por nombre, y/o extensión, usando el comando [DIR](#).

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
DESENT      <DIR>        26-10-2013  1:14
DOOM         <DIR>        11-02-2011  11:08
DOOM2        <DIR>        26-10-2013  2:01
DOOMII       <DIR>        26-10-2013  1:57
OUTRUN       <DIR>        25-10-2013  16:49
PAPERBOY    <DIR>        25-10-2013  16:49
PRINCEOF    <DIR>        25-10-2013  16:49
RISEN3D     <DIR>        26-10-2013  1:25
THHOS        <DIR>        08-08-2013  10:56
ZOOM         <DIR>        25-10-2013  16:49
  0 File(s)          0 Bytes.
  15 Dir(s)        262,111,744 Bytes free.

C:\>CD DOOM2

C:\DOOM2>DIR *.EXE
Directory of C:\DOOM2\.
DOOM2   EXE        686,921 26-10-2013  1:57
IPXSETUP EXE        13,787 26-10-2013  1:57
SERSETUP EXE        15,506 26-10-2013  1:57
SETUP   EXE        92,726 26-10-2013  1:57
  4 File(s)        808,940 Bytes.
  0 Dir(s)        262,111,744 Bytes free.

C:\DOOM2>
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: DOOM2
DOOMII      <DIR>        26-10-2013  1:57
OUTRUN      <DIR>        25-10-2013  16:49
PAPERBOY    <DIR>        25-10-2013  16:49
PRINCEOF    <DIR>        25-10-2013  16:49
RISEN3D     <DIR>        26-10-2013  1:25
THHOS        <DIR>        08-08-2013  10:56
ZOOM         <DIR>        25-10-2013  16:49
  0 File(s)          0 Bytes.
  15 Dir(s)        262,111,744 Bytes free.

C:\>CD DOOM2

C:\DOOM2>DIR *.EXE
Directory of C:\DOOM2\.
DOOM2   EXE        686,921 26-10-2013  1:57
IPXSETUP EXE        13,787 26-10-2013  1:57
SERSETUP EXE        15,506 26-10-2013  1:57
SETUP   EXE        92,726 26-10-2013  1:57
  4 File(s)        808,940 Bytes.
  0 Dir(s)        262,111,744 Bytes free.

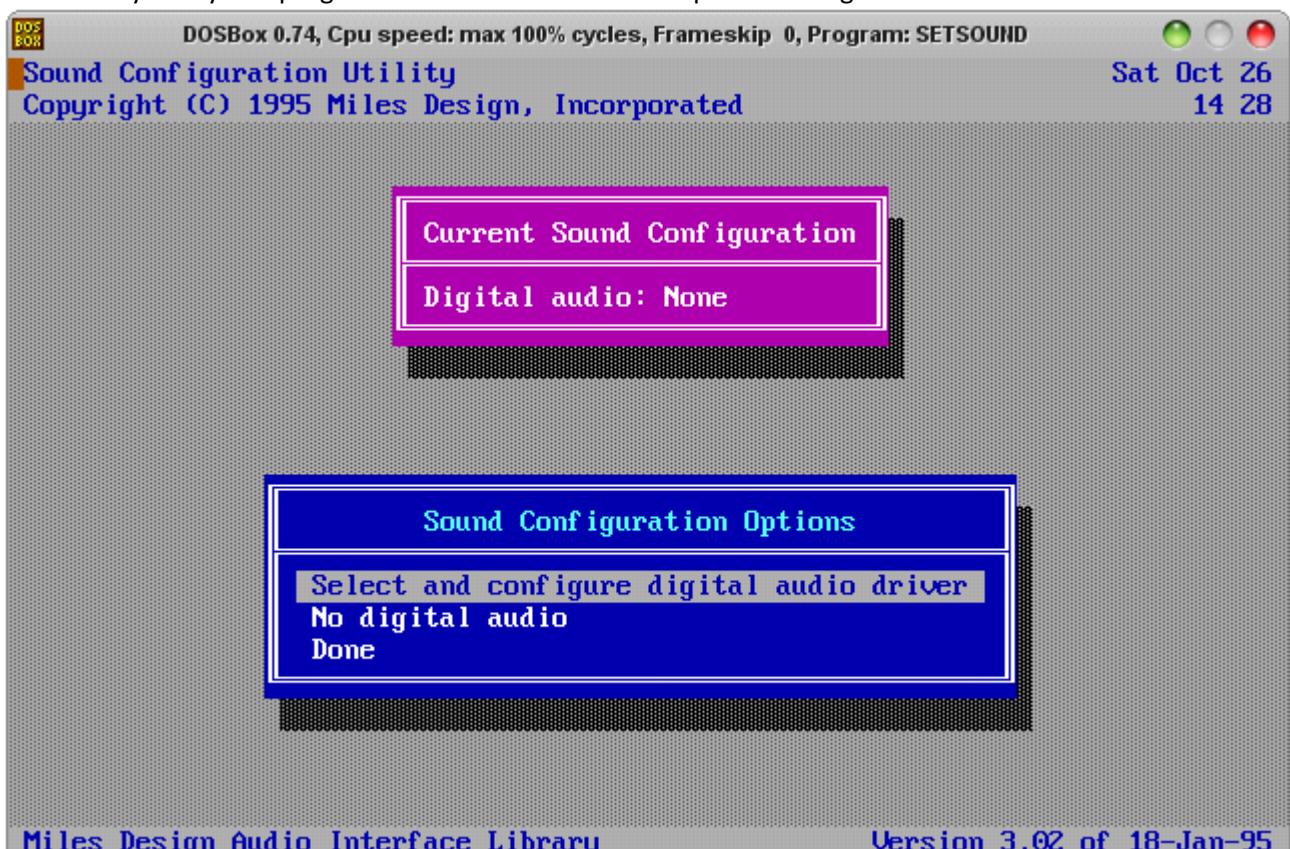
C:\DOOM2>DOOM2.EXE
DOS/4GW Professional Protected Mode Run-time Version 1.95
Copyright (c) Rational Systems, Inc. 1990-1993
```



Si debes realizar una instalación del juego o programa en cuestión, no dudes en hacerlo. Lo instalará siempre dentro de la carpeta asignada a la unidad, así que no temas.

#### ¿Cómo configuro el sonido en algunos juegos?

Algunos juegos requieren configurar primero el sonido, ya que en caso contrario no habrá sonido. Normalmente éstos juegos ya vienen con un programa específico para configurarlo. Por ejemplo, Ascendancy incluye un programa llamado SETSOUND.EXE para la configuración de sonido.

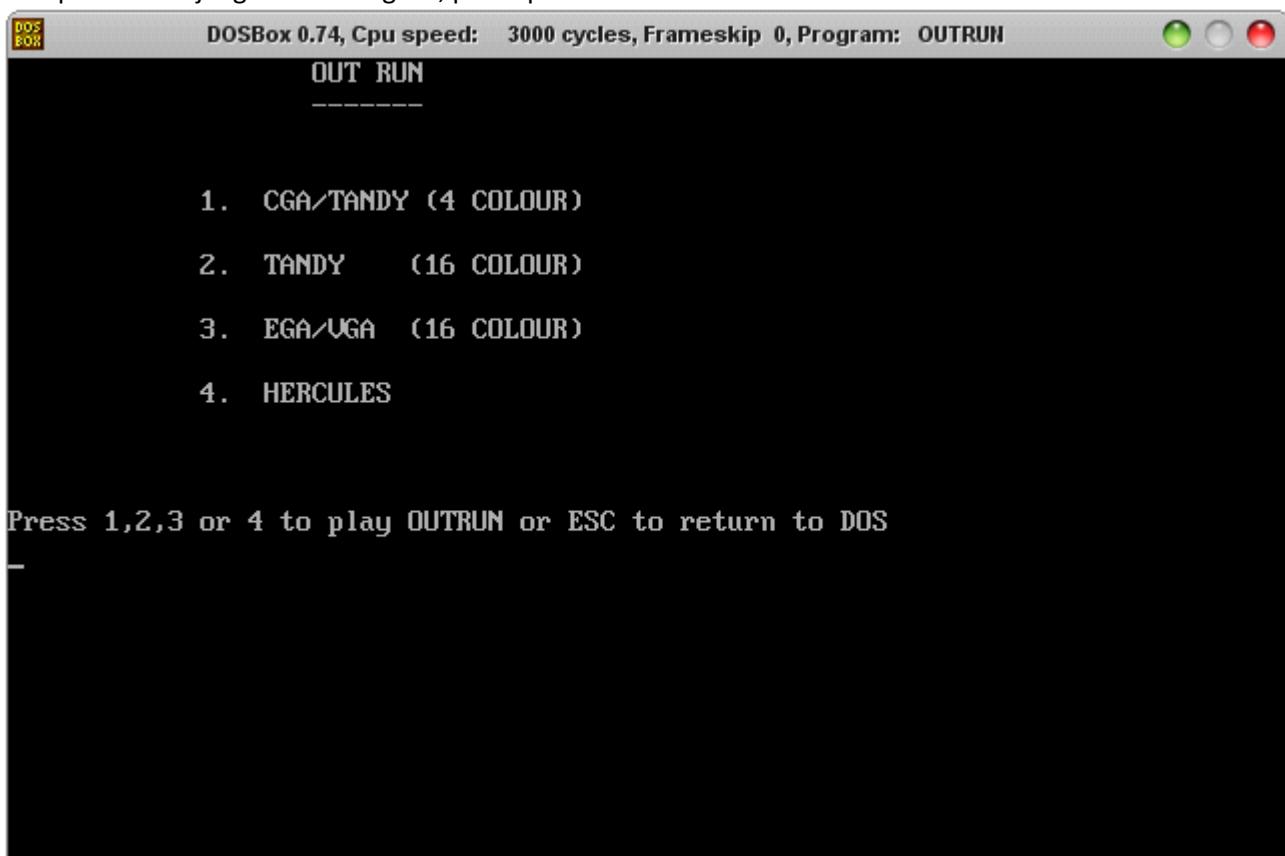


El proceso varía en función del juego y del programa, en algunos casos incluso detecta automáticamente el dispositivo de audio. Pero si no es así, lo configuramos manualmente. El dispositivo es siempre (POR DEFECTO, salvo que lo cambiemos manualmente) una Sound Blaster 16

(o 100% compatible), la IRQ es 220, interrupción (IRQ) 7, 8 bit DMA 1 y 16 bit DMA 5. Al iniciar DOSBox, el primer comando donde dice SET BLASTER te da todos éstos parámetros.

### ¿VGA, CGA, EGA...?

Algunos juegos, al ejecutarlos, te piden que selecciones el adaptador gráfico que deseas usar. Ésto solo pasa en los juegos más antiguos, por supuesto.



En caso de duda, selecciona siempre EGA, o VGA.

### Configuración avanzada

La configuración por defecto nos permite jugar a prácticamente cualquier juego, y ejecutar casi cualquier programa de 16-bits. Pero si queremos apretarle un poco más las tuercas al emulador, siempre podemos modificar a mano casi cualquier parámetro del emulador.

Para ello, vamos a la carpeta donde tengamos instalado DOSBox. Por defecto, suele ser *C:Archivos de programaDOSBox-0.xx*

En dicha carpeta encontraremos un archivo llamado DOSBox 0.74 Options.bat que debemos ejecutar. Dicho archivo nos abre un archivo de texto con la configuración del emulador. En él podemos configurar los siguientes apartados:

- **SDL**: Es el renderizador de la pantalla. Nos permite poner el programa a pantalla completa por defecto, ajustar la resolución, etc. También nos permite asignar un renderizador como DirectDraw u OpenGL.
- **DOSBOX**: La configuración de la interfaz, y el tipo de máquina a emular. También podemos ajustar la RAM, aunque recomiendo no poner valores demasiado altos. Cualquier valor entre 16 y 64 Mb, siempre en potencias de 2, es bueno.
- **RENDER**: Nos permite ajustar algunos parámetros del renderizador. No hay mucho que tocar aquí.
- **CPU**: Podemos establecer el tipo de procesador y los ciclos a emular. Conviene no tocar mucho por aquí.
- **MIXER**: Nos permite modificar la salida de audio. Ojo, la salida. No el dispositivo de audio emulado. Podemos también desactivar el audio.
- **MIDI**: Configuración relacionada con el dispositivo MIDI emulado. No hay mucho que tocar.
- **SBLASTER**: Nos permite configurar la tarjeta de sonido [Sound Blaster](#) emulada. Pudiendo seleccionar su modelo, puerto base, IRQ, DMA y 16 bit DMA (HDMA).
- **GUS**: Permite activar o desactivar la emulación [Gravis Ultrasound](#) y configurar su puerto base, IRQ y DMA.
- **SPEAKER**: La configuración del altavoz interno a emular. Ojo, no se refiere al altavoz donde se emiten los pitidos, sino a un altavoz que solían traer los ordenadores antiguos en la misma torre

como salida de audio. Tambien permite activar o desactivar la emulación del [Covox Sound Source de Disney](#).

Tanto si usas SB, como GUS, o incluso COVOX, es indiferente a la hora de renderizar audio en el emulador. Pues practicamente es como si usaramos un COVOX, ya que nuestra tarjeta o chip de audio (físico) integra un DAC. Lo único que hay algunos juegos que son compatibles con el COVOX de Disney, y solo emitiran determinado audio en dicho dispositivo. También encontraremos juegos diseñados para x dispositivo de video o audio.

- **JOYSTICK**: Nos permite configurar la emulación un dispositivo de joystick.

- **SERIAL**: Nos permite emular (ojo, emular) un dispositivo conectado a un puerto serie. Tambien podemos conectarlo con un puerto serie físico de nuestro ordenador, aunque en Windows (en las versiones más actuales) es un poco más complicado.

- **DOS**: Nos permite ajustar algunos soportes y la distribución del teclado del sistema DOS emulado.

- **IPX**: Permite emular el protocolo IPX.

- **AUTOEXEC**: Probablemente una de las opciones más interesantes. Podemos añadir líneas de comandos que se ejecutaran nada más iniciar el emulador. Podemos, por ejemplo, añadir una línea con la carpeta que vamos a montar, y cambiar el prompt automaticamente a dicha unidad. Por ejemplo:

```
[autoexec]
# Lines in this section will be run at startup.
# You can put your MOUNT lines here.
mount c c:games
c:
```

## Fuentes de juegos

[Abandonia](#) – Tienen un buen catálogo de juegos abandonware, aunque han retirado bastantes recientemente.

[MyAbandonware](#) – Recomiendo hechar un buen vistazo a ésta página, tiene un catálogo también interesante.

[GameGraveyard](#) – Una tercera página con otro catálogo. Por si aún no has encontrado el juego que buscabas.

[Google](#) – La última opción es buscar en Google el juego que quieras, si no se encuentra en ninguna de éstas otras tres páginas.

## Recomendaciones personales

[Ascendancy](#) – Un juego de estrategia espacial muy adictivo, especialmente si os gustan los juegos de navegador tipo OGame o (ya inexistente) XWars. Teneis un pequeño manual no oficial [aquí](#).

[Doom](#) y [Doom II](#) – Totalmente IMPRESCINDIBLES. Uno de los primeros FPS que, junto con Duke Nukem 3D, hicieron historia. Teneis también los [niveles maestros](#) de Doom II.

[Duke Nukem 3D](#) – No puedo ofrecer un enlace porque no es abandonware, pero si teneis la posibilidad de hacerlos con una copia, ¡Hacedlo!

[Theme Hospital](#) - Un simulador de la inSeguridad Social española.



Es muy adictivo, recomiendo probarlo.

[Rayman](#) – Otro juego del que debeis haceros con una copia como sea. Es mi plataformas 2D preferido. No es abandonware ni freeware, así que tendreis que buscarlo por vuestra cuenta.

## Y para acabar...

[Inertia Player](#) – No es un juego, pero es un interesante programa gratuito, que permite reproducir archivos de audio en formato tracker (MOD y S3M). Conviene configurar primero el dispositivo de audio usando el programa ISETUP.EXE incorporado.

Pegado de <<http://ninjiserver/WORDPRESS/?p=925>>

# Mi ordenador no arranca!

sábado, 12 de julio de 2014

1:31

## ¡Ayuda! ¡Mi ordenador no arranca!

[December 22, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



Tu ordenador ha llegado a un punto en el que, al encenderlo, no arranca. No suena ningún pitido de la BIOS, y la pantalla se queda en negro. O sí arranca, pero se reinicia al poco tiempo. ¡Que no cunda el pánico! Con ésta pequeña guía, te será un poco más fácil encontrar el problema.

### Possible causa 1: Problemas de temperatura

Si hay un problema grave de hardware, siempre digo que el primer causante podría ser la temperatura. Cuando la BIOS detecta que tu ordenador se está calentando más de lo que debería, se reinicia o se apaga, dependiendo de la placa base. He visto placas base que incluso te avisan de que la temperatura está muy alta, con una voz en inglés por el altavoz interno del ordenador, o incluso que emiten una melodía.

El primer paso es hacer una buena limpieza interna, cambiar la pasta térmica, comprobar que los disipadores están haciendo contacto correctamente, etc. De éste modo, nos aseguramos de que la temperatura no es tanto problema. Aunque parece mentira, un disipador obstruido por el polvo puede aumentar la temperatura en unos 10 grados celsius.

El segundo paso, es asegurarse de que los ventiladores funcionan adecuadamente. Hay BIOS que si detectan que un ventilador está desconectado, incluyendo el de la propia GPU, se reinicia automáticamente para proteger el componente. En otros casos, te avisa con un mensaje de error. El ventilador es otro componente muy importante a la hora de disipar el calor, y si no hay ventilador, el componente acaba por quemarse aunque tenga el disipador encima.

### Possible causa 2: Mal funcionamiento del hardware

Si la BIOS detecta que un componente esencial falta o falla, detiene la ejecución. Los casos más comunes suelen ser por un fallo en la memoria RAM, o en la GPU, que son los componentes críticos del sistema. Aunque si el fallo está en la propia placa base, el resultado podría ser el mismo.

Si dispones de dos memorias, empieza por cambiarlas de lugar. El ordenador tiene prioridad a la hora de escoger en qué memoria va a alojar los datos, así que si no encuentra una memoria RAM en condiciones en la primera ranura, es como si no tuvieras RAM, a pesar de que tengas otra memoria al lado.

Reconecta y asegurate de que todos los dispositivos están bien conectados. Y si tienes algún repuesto, prueba a conectar el repuesto para ver si funciona.

Si todo ésto falla, solo nos quedan 2 opciones.

### Possible causa 3: Mal funcionamiento de la fuente de alimentación

Los componentes de la placa base deben estar bien alimentados con un voltaje adecuado. Si se les suministra un valor inadecuado, se causa un mal funcionamiento de los componentes.

Si tienes un voltmetro o polímetro, puedes medir los voltajes de los cables que van a la placa base. Enchufa (con cuidado de no darte un calambrazo) la fuente, y mide la tensión colocando el borne positivo en uno de los cables activos (rojo, amarillo, naranja o marrón), y el negativo en uno de los negros. Cada color indica un valor.

Color	Signal
Orange	Power Good or +3.3 V newer
Red	+5 V
Yellow	+12 V
Blue	-12 V
Black	Ground
White	-5 V
Green	Power on

Si se puentea el verde con el negro más cercano, se activa la fuente de alimentación (necesario para poder medir).

Si alguno de los voltajes falla, podemos intentar abrirla (después de desenchufarla, por supuesto) y buscar, si sabemos, el causante. O directamente cambiarla por otra, que tampoco son muy caras.

### Possible causa 4: Mal funcionamiento interno de la placa base

Si hemos descartado todas las posibilidades anteriores, entonces mucho me temo que el fallo está en la propia placa base. Eso quiere decir, que lo más rentable en la mayoría de casos, será tirar el ordenador, y comprar uno nuevo. O podemos intentar cambiar la placa por otra, ya no son tan caras como antes.

Pegado de <<http://ninjiserver/WORDPRESS/?p=947>>

# Diferencias entre RAM y Paganación

sábado, 12 de julio de 2014

1:32

## Diferencias entre la RAM, y el archivo de paginación

[October 24, 2013](#) [Ninjihaku](#) [3 Comments](#) [Edit](#)



La gente conoce lo que es la memoria RAM. Aunque no conoce con exactitud como funciona. Pero sí confunde de una manera muy estrepitosa el funcionamiento del archivo de paginación de Windows. Mucha gente ofrece tutoriales sobre cómo alojar éste archivo en otro disco (la mayor aberración que he visto, ha sido meterlo en una memoria USB), o incluso desactivarlo (cosa que no recomiendo, salvo que querais que vuestro sistema tenga más errores que un windows 9x).

El sistema usa el archivo de paginación con el fin de **ahorrar memoria ram**. El primer uso que le da, es cuando hay procesos en segundo plano inactivos. Cuando se da éste caso, el sistema los mueve al archivo de paginación para que no malgasten espacio en la memoria RAM, y lo devuelve a ésta sólo cuando se continua con su ejecución.

El segundo caso en el que se usa, es cuando el sistema se queda sin espacio en la memoria ram. En éste caso, el sistema se ve forzado a mover lo que no quepa en la memoria, al archivo de paginación.

¿Es más rápido el archivo de paginación que la memoria RAM? – Evidentemente que **NO**. El archivo de paginación es un archivo más en tu disco duro, y leer del disco duro es siempre más lento que leer en una memoria.

¿Sería más eficiente moverlo a una memoria USB? - **NO**. Mucha gente se cree que por ser una memoria FLASH, va a ser más rápido. Pero no tienen en cuenta que la memoria se conecta a través de un puerto USB, que son más lentos que el disco duro. Especialmente si el bus de datos está ocupado. Por no mencionar que la estabilidad de tu sistema va a depender de que esa memoria esté conectada al puerto USB, dejando un puerto más sin poder usar (y ralentizando el resto de puertos USB, al estar constantemente usando el bus de datos).

¿Sería mejor desactivar el archivo de paginación? – **NO**. Hacerlo no hará más que darle problemas a tu sistema, sin suponer mejora alguna en el rendimiento (Más bien al contrario). Imagina que te quedas sin memoria física y dejas al sistema sin memoria virtual.

¿Es bueno aumentar el tamaño del archivo de paginación? – Sólo si te quedas sin memoria demasiado a menudo, pero es mejor (si tu placa y bolsillo lo permiten) ampliar la memoria RAM. Y ésto solo se puede conseguir, cambiando tu RAM física por “pastillas” (como las llama vulgarmente la gente) de mayor tamaño.

Hay que verlo de ésta manera. Windows incluye de forma predeterminada un archivo llamado “archivo de paginación”. Ese archivo te dicen que es para ahorrar memoria. Si el sistema lo trae de forma predeterminada porque lo necesita, ¿Por qué vamos a tener que tocarlo? Lo mejor es dejar éste archivo tal y como está, donde está. Porque si el sistema funciona bien como está, no hay que hacer cambios. A menos que sepas muy bien lo que haces, y la mayoría de la gente no lo sabe.

Me hace gracia, tanta gente diciendo que “Windows es una mierda”. Alegando que sus sistemas dan errores. Es lógico, si la gente se cree más lista que Microsoft, y hace tonterías como éstas, o instalar tropecientasmil aplicaciones que traen spyware e incluso malware, el sistema tendrá problemas. Si no sabemos lo que estamos haciendo, conviene no tocar nada.

Otro mito es el de aumentar la memoria RAM, manipulando éste archivo. Creo que ya he citado que **éste archivo no tiene nada que ver con la memoria física**. Es simplemente una “memoria virtual”, un archivo en el disco duro que actúa como una “memoria de emergencia”. Por mucho que toques ésta memoria, no vas a tener más memoria. Y verás que cuando se te llene la RAM, el ordenador se va a ralentizar excesivamente, aumentes o no éste archivo. Y por supuesto, como la quites y se te llene la memoria física, adios a la estabilidad del sistema.

Pegado de <<http://nинjiserver/WORDPRESS/?p=918>>

# Debug.exe

sábado, 12 de julio de 2014

1:33

## Como usar debug.exe para programar

[October 9, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)

```
C:\>debug win.com
-u
BBP0:0000 0E      PUSH    CS
BBP0:0001 1P      POP     DS
BBP0:0002 BA0EE00 MOU     DX,000E
BBP0:0005 B409    MOU     AH,09
BBP0:0007 CD21    INT     21
BBP0:0009 B8014C  MOU     AX,4C01
BBP0:000C CD21    INT     21
BBP0:000E 54      PUSH    SP
BBP0:000F 68      DB      68
BBP0:0010 69      DB      69
BBP0:0011 7320    JNB    0033
BBP0:0013 7872    JO     0087
BBP0:0015 6P      DB      6F
BBP0:0016 67      DB      67
BBP0:0017 7261    JBB    007A
BBP0:0019 6D      DB      6D
BBP0:001A 206361  AND    [BP+DI+61],AH
BBP0:001D 6E      DB      6E
BBP0:001E 6E      DB      6E
BBP0:001F 6P      DB      6F
```

Ya hablé un poco por encima de éste programa en [ésta entrada](#). Se trata de un pequeño programa de 16 bits que se lleva incluyendo en los sistemas de Microsoft desde la época de MS-DOS.

Por pequeño que parezca, en su día era muy útil y potente, pues incluía un ensamblador, un desensamblador, e incluso un editor hexadecimal. Puede que no fuera una aplicación muy intuitiva, pues Microsoft la incluía para depurar su sistema operativo, pero sí muy potente.

En los sistemas de 64 bits parece que no funciona muy bien, y de hecho creo recordar que ya no se encuentra disponible en Windows 8. Lo cuál es una lástima, pues es bastante divertido, si sabes programar. No obstante, yo recomiendo usarlo en MS-DOS, bien en un [ordenador virtual](#), en [DOSBOX](#), etc. Por tonto que parezca, programar en 16 bits es muy divertido. Y a veces también es doloroso.

### 1. ¿Cómo se usa debug.exe?

Se puede ejecutar desde la línea de comandos, usando el comando **debug**, siempre y cuando dicha aplicación esté presente en nuestro ordenador. Adicionalmente, podemos cargar un archivo o aplicación escribiendo su nombre al lado. Ejemplo: **debug.exe command.com**. El programa siempre se carga en el offset 100.

Dentro del programa, los comandos mas usados son:

- a [offset] -> Nos permite ensamblar instrucciones a partir del offset especificado.
- u [offset] ([final]) -> Desensambla el programa, desde offset hasta final (si se especifica), y nos muestra el resultado en pantalla.
- d [offset] ([final]) -> Muestra el contenido de la memoria, desde offset, hasta final (si se especifica), como un editor hexadecimal.
- h [final] [principio] -> Calcula, en hexadecimal y en bytes, el tamaño total del programa.
- r [registro] -> Te muestra y te permite cambiar el valor de un registro de 16 bits.
- n [nombre] -> Especifica un nombre de archivo, para cargarlo en memoria o escribir en el.
- l -> Carga un programa, cuyo nombre se especifica con -n.
- w -> Escribe un programa, cuyo nombre se especifica con -n. El número de bytes a escribir se

especifica en el registro cx (usando -r cx).

## 2. ¿Cómo ensamblamos un programa?

Para ensamblar un programa de 16 bits, hay que conocer [ensamblador](#) (o por lo menos las instrucciones más básicas, incluyendo los tipos de saltos), hay que conocer las [interrupciones](#) de MS-DOS y la BIOS, y hay que conocer como funciona la memoria en [modo real \(segmento:offset\)](#). Un entendimiento de la estructura del [8086](#) también nos pondrá las cosas mucho más fáciles (Conocer los registros y las banderas (FLAGS), y para qué se usa cada uno).

O podeis limitaros a copiar lo que yo exponga aquí. Experimentar también es divertido, aunque un poco peligroso. Hay instrucciones que pueden hacer que escribas por error en un disco, o algo peor. Ten cuidado, y documentate bien antes de dar cualquier paso.

Empezaremos por abrir debug.exe usando el comando *debug*.



```
C:\Console2>debug
```

Ahora nos encontramos dentro del programa *debug.exe*. Vamos a ensamblar un programa, lo llamaremos “hiworld.com” (recordad que en MS-DOS, el límite de caracteres para un nombre son 8 caracteres). Así que usaremos primero el comando:

-n hiworld.com

Una vez hecho ésto, podemos comenzar a ensamblar usando la instrucción

-a 100

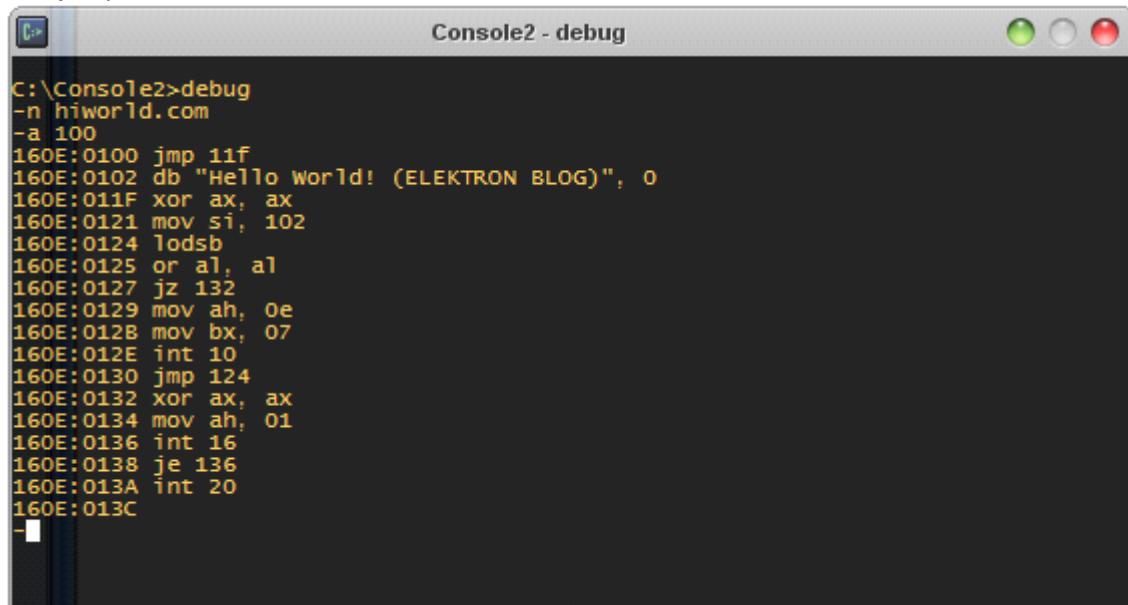
100 representa el offset en el que vamos a empezar a escribir instrucciones. Deberíamos siempre empezar por ésta dirección (0x100). Nos aparecerá lo siguiente:



```
C:\Console2>debug
-n hiworld.com
-a 100
160E:0100 ■
```

160E representa el segmento en el que se ha alojado nuestro programa. Y 0100, el offset. Ahora podemos empezar a insertar instrucciones, teniendo en cuenta que sólo podemos ensamblar en 16

bit, y que todos los valores numéricos deben ser representados en hexadecimal.  
Un ejemplo de un 'hello world':

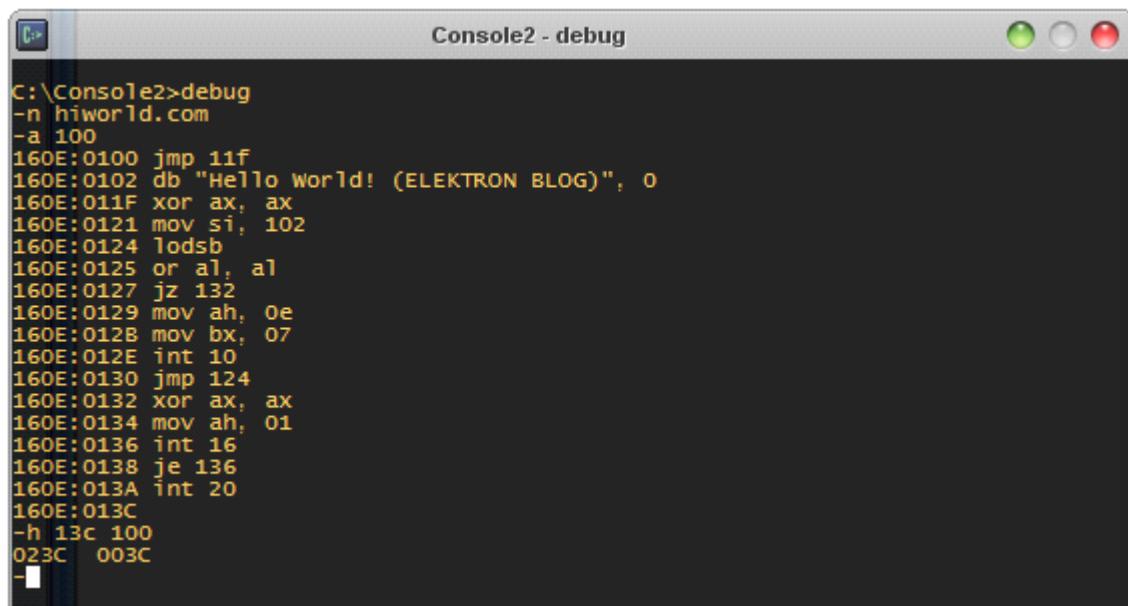


The screenshot shows a terminal window titled "Console2 - debug". The command entered is:

```
C:\Console2>debug  
-n hiworld.com  
-a 100  
160E:0100 jmp 11f  
160E:0102 db "Hello World! (ELEKTRON BLOG)", 0  
160E:011F xor ax, ax  
160E:0121 mov si, 102  
160E:0124 lodsb  
160E:0125 or al, al  
160E:0127 jz 132  
160E:0129 mov ah, 0e  
160E:012B mov bx, 07  
160E:012E int 10  
160E:0130 jmp 124  
160E:0132 xor ax, ax  
160E:0134 mov ah, 01  
160E:0136 int 16  
160E:0138 je 136  
160E:013A int 20  
160E:013C
```

Escribimos hasta la última linea (13C) incluida, y pulsamos ENTER para salir. No os preocupeis por el programa, luego lo explico.

El tamaño del programa es fácil de calcular. Es el offset final, menos 100, en hexadecimal todo. En nuestro caso, son 3C bytes. Si tenemos problemas con el cálculo, podemos usar la calculadora, o el comando -h



The screenshot shows a terminal window titled "Console2 - debug". The command entered is:

```
C:\Console2>debug  
-n hiworld.com  
-a 100  
160E:0100 jmp 11f  
160E:0102 db "Hello World! (ELEKTRON BLOG)", 0  
160E:011F xor ax, ax  
160E:0121 mov si, 102  
160E:0124 lodsb  
160E:0125 or al, al  
160E:0127 jz 132  
160E:0129 mov ah, 0e  
160E:012B mov bx, 07  
160E:012E int 10  
160E:0130 jmp 124  
160E:0132 xor ax, ax  
160E:0134 mov ah, 01  
160E:0136 int 16  
160E:0138 je 136  
160E:013A int 20  
160E:013C  
-h 13c 100  
023C 003C
```

El segundo valor dado por el comando, se corresponde con el tamaño de nuestro programa. Ahora sólo queda indicarle al programa, que queremos escribir 3C bytes. El programa carga en el registro cx, el contador de bytes. Y por ello, le especificaremos ahí que queremos escribir esa cantidad de bytes, usando el comando -r.

```
Console2 - debug
-n hiworld.com
-a 100
160E:0100 jmp 11f
160E:0102 db "Hello World! (ELEKTRON BLOG)", 0
160E:011F xor ax, ax
160E:0121 mov si, 102
160E:0124 lodsb
160E:0125 or al, al
160E:0127 jz 132
160E:0129 mov ah, 0e
160E:012B mov bx, 07
160E:012E int 10
160E:0130 jmp 124
160E:0132 xor ax, ax
160E:0134 mov ah, 01
160E:0136 int 16
160E:0138 je 136
160E:013A int 20
160E:013C
-h 13c 100
023C 003C
-r cx
CX 0000
:3C
-q
```

Ya está todo listo. Ejecutamos el comando `-w` para escribir el programa en el archivo especificado. Como no hemos especificado ninguna ruta, se guardará en el directorio donde se encuentra la consola (normalmente C:\WINDOWS\SYSTEM32). Al finalizar, usamos `-q` para salir.

```
Console2 - debug
160E:0100 jmp 11f
160E:0102 db "Hello World! (ELEKTRON BLOG)", 0
160E:011F xor ax, ax
160E:0121 mov si, 102
160E:0124 lodsb
160E:0125 or al, al
160E:0127 jz 132
160E:0129 mov ah, 0e
160E:012B mov bx, 07
160E:012E int 10
160E:0130 jmp 124
160E:0132 xor ax, ax
160E:0134 mov ah, 01
160E:0136 int 16
160E:0138 je 136
160E:013A int 20
160E:013C
-h 13c 100
023C 003C
-r cx
CX 0000
:3C
-w
Escribiendo 0003C bytes
-q
```

Para ejecutar el programa, simplemente iniciamos `hiworld.com`. Veremos nuestro programa en acción:

```
Console2
160E:0124 lodsb
160E:0125 or al, al
160E:0127 jz 132
160E:0129 mov ah, 0e
160E:012B mov bx, 07
160E:012E int 10
160E:0130 jmp 124
160E:0132 xor ax, ax
160E:0134 mov ah, 01
160E:0136 int 16
160E:0138 je 136
160E:013A int 20
160E:013C
-h 13c 100
023C 003C
Console.exe  console.xml  ConsoleHo...  FreeImage.dll  FreeImageP...  HIWORLD.C...
-r cx
CX 0000
:3C
-w
Escribiendo 0003C bytes
-q
C:\Console2>hiworld.com
Hello World! (ELEKTRON BLOG)
C:\Console2>
```

Es MUY IMPORTANTE que escribais el programa tal cual se muestra en la imagen, o no funcionará.

Ahora explicaré por qué.

### 3. Explicación del programa hiworld.com

```
100 jmp 11f
102 db "Hello World! (ELEKTRON BLOG)", 0
11F xor ax, ax
121 mov si, 102
124 lodsb
125 or al, al
127 jz 132
129 mov ah, 0e
12B mov bx, 07
12E int 10
130 jmp 124
132 xor ax, ax
134 mov ah, 01
136 int 16
138 je 136
13C int 20
```

El programa empieza en la línea 100h, con un salto a la 11f (jmp 11f). De éste modo, nos saltamos los datos binarios de la línea 102.

Para calcular el salto de la primera línea, se hace de la siguiente forma. El texto de la siguiente línea son 28 bytes, más 1 byte por el terminador 0 que le hemos incluido. A los 29 bytes del texto, le añadimos otros 2 extra por la instrucción JMP que hemos usado. En total son 31 bytes de código, que en hexadecimal son 1F. Dado que el salto comienza en la línea 100, calculamos: 100h+1Fh = 11Fh. Por eso saltamos a la línea 0x11F (Y de ahí que dijera que el programa no funcionaría, a menos que lo copieis exactamente igual).

La línea 102 contiene el texto que vamos a mostrar. Como lo que queremos es almacenar un texto, tenemos que almacenarlo carácter por carácter. Es decir, byte por byte. Por ello, usamos db, que es una directiva que le indica que lo que hay ahí, son datos binarios, en bytes.

El programa continua en la línea 11F, 29 bytes después. En ésta línea, ponemos AX a 0 realizando una operación OR Exclusiva consigo mismo. Es más rápido que hacer MOV AX, 0 (consume menos ciclos).

En la línea 121, lo que hacemos es apuntar el Source Index a la línea que contiene nuestro texto, para leerlo después. Si es un registro que se usa para almacenar un puntero a una dirección sobre la que vamos a realizar operaciones de lectura, mientras que DI (destination Index) es para operaciones de escritura. Algunas interrupciones usan ambos (DI:SI) para especificar una dirección real entera (segment:offset).

LODSB es una instrucción que obtiene el siguiente byte, desde la dirección almacenada en SI, y lo almacena en AL. Recuerda que los registros de 16 bits, como por ejemplo AX, se dividen en dos de 8 bits: AL para el byte de menor peso (Accumulator Low), y AH para el de mayor peso (Accumulator High). De modo que AX = AH:AL.

Las líneas 125 y 127, comprueban si AL es 0, en cuyo caso indica que el texto se acaba ahí, y por tanto dejaremos de escribir texto en la pantalla.

Desde 129 a 130 (h), el programa escribe un texto haciendo uso de la interrupción INT 10H. El parámetro AH=0E permite realizar ésta tarea haciendo uso de ésta interrupción. Los otros dos parámetros son BH, que indica la página en la que escribimos (normalmente lo ponemos a 0), y BL el color de texto (lo establecemos a 0x07). Para resumirlo, simplemente establecemos BX a 0x0007, y ejecutamos el interruptor.

En la línea 130, hacemos un bucle, de modo que leemos uno a uno cada byte desde la posición

indicada en SI.

Por último, hacemos que el programa espere a que el usuario introduzca una pulsación del teclado, haciendo uso de la interrupción [16H \(AH=0X01\)](#). Y salimos, usando la interrupción 20H.

### 3. Corregir un programa

Hemos creado un programa, y se supone que al finalizar el mismo, debería esperar una pulsación de teclado y salir. No obstante, en lugar de eso, se queda bloqueado. Cometí un ligero error al escribir el programa, pero ¡No hay problema! Siempre podemos rectificar sin tener que reensamblar el programa entero.

Volvemos a la consola, y escribimos **debug hiworld.com**. Esto iniciará debug.exe y cargará en memoria nuestro programa. También podemos especificar el nombre con -n, y cargarlo usando -l. Ahora usaremos el comando -u para ver nuestro programa. Escribimos -u 11f (Así nos saltamos los datos binarios de la línea 102, que al desensamblarlos da cosas extrañas).

```
mov    fs, ax          Console2 - debug hiworld.com
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Console2>debug hiworld.com
-u 11f
16C0:011F 31C0 0xFFFF XOR    AX, AX
16C0:0121 BE0201 MOV    SI, 0102
16C0:0124 AC LODSB
16C0:0125 08C0 Texto OR     AL, AL      ; Cargamos en el
16C0:0127 7409 i el texto JZ    0132
16C0:0129 B40E MOV    AH, 0E
16C0:012B BB0700 Message MOV    BX, 0007      ; Llamamos a la
16C0:012E CD10 para mostrar el texto INT   10
16C0:0130 EBF2 JMP    0124
16C0:0132 31C0 XOR    AX, AX      ; Subrutina para
16C0:0134 B401 , cargado en el registro "si"
16C0:0136 CD16 INT    16      ; cargamos el
16C0:0138 74FC carácter de cada carácter
16C0:013A CD20 INT    20      ; comprobamos que
16C0:013C 0000 al ADD    [BX+SI], AL      ; el carácter no esté en [BX+SI], AL
16C0:013E 0000 character no en [BX+SI], AL
-| jz    .DONE           ; ... Lo
  | interpretamos como final de linea, por lo que termina la
  | subrutina
```

El error es sencillo, pero puede resultar muy difícil de ver para un novato. Al ejecutar una interrupción, ésta modifica internamente algunos registros para poder llevar a cabo su tarea. En nuestro caso, modifica AH, que es el registro que contiene el parámetro.

Nuestro programa realiza un bucle directamente hacia la línea 136 (INT 16). Cuando se ejecuta, AH se modifica internamente. Con lo que, en el siguiente ciclo, no se ejecuta con el parámetro AH = 0x01, sino que se ejecuta con cualquier otro parámetro. Ésto hace que, en ocasiones puede que funcione, y en la mayoría de ocasiones no funcionará o dará error.

Para solucionarlo, haremos que el programa salte una línea antes, a la 134, para así asegurarnos de que la interrupción se ejecuta siempre con el parámetro AH = 0x01.

Usamos el comando -a para ensamblar en la línea 138, que es la que contiene el error. Y la corregimos para que salte a la 134.

Console2 - debug hiworld.com

```

Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
0x00 ; Pagina 0
C:\Console2>debug hiworld.com ; Atributo de texto
-u 11f
16C0:011F 31C0 XOR AX,AX
16C0:0121 BE0201 ntalia MOV SI,0102
16C0:0124 AC LODSB ; Interrupcion
16C0:0125 08C0 OR AL,AL
16C0:0127 7409 JZ 0132
16C0:0129 B40E MOV AH,OE
16C0:012B BB0700 MOV BX,0007
16C0:012E CD10 INT 10 ; Si no hemos
16C0:0130 EBF2 jmp para 0124 cerrar el ordenador
16C0:0132 31C0 XOR AX,AX
16C0:0134 B401 MOV AH,01
16C0:0136 CD16 INT 16
16C0:0138 74FC JZ 0136
16C0:013A CD20 INT 20
16C0:013C 0000 ADD [BX+SI],AL
16C0:013E 0000 ADD [BX+SI],AL
-a 138
16C0:0138 jz 134
16C0:013A
C:\Console2> "REST IN PEACE", 0x07, 0x00 ; Nuestro

```

Una vez finalizada la modificación, lo demás es todo igual. Pulsamos enter sin escribir nada en 13A, colocamos el número de bytes a escribir en CX, y escribimos igual que antes.

Console2

```

16C0:0124 AC LODSB
16C0:0125 08C0 OR AL,AL
16C0:0127 7409 JZ 0132 Pagina 0
16C0:0129 B40E MOV AH,OE ; Atributo de texto
16C0:012B BB0700 MOV BX,0007
16C0:012E CD10 INT 10 ; Interrupcion
16C0:0130 EBF2 jmp para 0124
16C0:0132 31C0 XOR AX,AX
16C0:0134 B401 MOV AH,01
16C0:0136 CD16 INT 16
16C0:0138 74FC JZ 0136
16C0:013A CD20 INT 20
16C0:013C 0000 ADD [BX+SI],AL
16C0:013E 0000 ADD [BX+SI],AL
-a 138
16C0:0138 jz 134
16C0:013A
-r CX
CX 003C
:3C
-w
Escribiendo 0003C bytes
-q
C:\Console2> "REST IN PEACE", 0x07, 0x00 ; Nuestro

```

Ahora ejecutamos el programa, y observamos que ésta vez, funciona. Al pulsar enter, volveremos a la línea de comandos.

Console2

```

16C0:0129 B40E MOV AH,OE
16C0:012B BB0700 MOV BX,0007
16C0:012E CD10 INT 10
16C0:0130 EBF2 JMP 0124
16C0:0132 31C0 XOR AX,AX
16C0:0134 B401 MOV AH,01
16C0:0136 CD16 INT 16
16C0:0138 74FC JZ 0136
16C0:013A CD20 INT 20
16C0:013C 0000 ADD [BX+SI],AL
16C0:013E 0000 ADD [BX+SI],AL
-a 138
16C0:0138 jz 134
16C0:013A
-r CX
CX 003C
:3C
-w
Escribiendo 0003C bytes
-q
C:\Console2>hiworld.com
Hello World! (ELEKTRON BLOG)
C:\Console2>
C:\Console2>

```

Borrador guardado a las 13:41:53.

Y por hoy, ésto es todo. Seguiré hablando un poco de éste tema en futuras entradas, pero por el momento no hay mucho más que decir. Un ejemplo:

Pegado de <<http://nинjiserver/WORDPRESS/?p=879>>

# Virus Informáticos. Verdades y mentiras.

sábado, 12 de julio de 2014

1:34

## Virus Informáticos. Verdades y mentiras.

[October 5, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



La entrada de hoy la voy a dedicar a un mito. Un mito, no porque no sea real, pero sí porque es algo ya bastante antiguo. Los **virus informáticos**. Y sí, también hablaré de algunos mitos sobre los virus.

### 1. ¿Qué es un virus informático?

A cualquier programa que afecte negativamente a la estabilidad del sistema, se le llama virus. No obstante, "virus" tiene una definición propia. Para que un programa se pueda considerar Virus, debe replicarse al ser ejecutado. Normalmente injectando su código en otros programas y archivos, e incluso infectando otros ordenadores a través de la red y el correo electrónico.

Así pues, un virus informático tiene un comportamiento similar al de un virus biológico, pero aplicado a la informática.

### 2. El primer virus



Un primer “virus” podría ser [Creeper](#), en 1971. No era capaz de dañar un ordenador, pero sí de replicarse a través de archivos, medios extraíbles y redes. Infectaba sistemas [TOPS-20](#). Con el fin de neutralizar su expansión, se desarrolló *Reaper*.

La primera epidemia no sería hasta 1986, con el denominado [Brain](#). Se desarrolló con el fin de infectar el sector de arranque (ya sabéis, los primeros 512 bytes) de los medios extraíbles (disquettes). En la información del disquete se encontraba también los nombres de los autores, y un teléfono para solicitar una desinfección. En la actualidad, los desarrolladores tienen un ISP, Brain NET, en Pakistán.

### 3. ¿Cómo funciona un virus común?

Los virus comunes, los que todos conocemos porque destruyen ordenadores o molestan de algún modo, se replican al ser ejecutados. Normalmente buscan otros archivos del sistema, para copiar su código (inyectarse) en ellos. Muchos virus de tipo DOS y W9x se inyectan en los ejecutables que vamos abriendo.

En algunos casos, antes de que ésto ocurra, se replica también a través de las redes, y envía correos con el código a todos tus contactos (si tienes contactos).

No es hasta que reiniciamos, que empezamos a notar los efectos. Éstos varían en función del virus. Algunos llenan la memoria del sistema hasta que revienta (flooder), otros destruyen archivos, o incluso la FAT de la partición primaria.

En raras ocasiones, son capaces incluso de inutilizar por completo el ordenador, borrando por completo la BIOS, o incluso en algún caso se ha llegado a destruir físicamente algún componente de la placa base, en ordenadores muy antiguos, causando un mal funcionamiento.

### 4. Los virus, hoy en día

Con el tiempo, los fabricantes de componentes de PC y los desarrolladores de Software y de sistemas operativos, se hicieron más conscientes del daño que éstos programas producían en los ordenadores, y del impacto económico que éstos tenían. Y por ello cada vez se añadían más y más sistemas de seguridad en los ordenadores, poniéndose más difícil cada vez a los que desarrollaban éste tipo de malware.

El último virus de éste tipo del que tengo constancia, es el famoso [Sasser](#), que afecta a algunas versiones antiguas de Windows XP. Éste virus desbordaba la memoria de un proceso de seguridad

de Windows XP, lsass.exe. Al hacerlo, causaba un error en éste proceso, obligandole a cerrarse. A consecuencia de ésto, el sistema iniciaba un proceso de apagado de emergencia haciendo uso del comando shutdown.exe.

A partir de entonces, los virus dejarían a un lado su capacidad destructiva para dedicarse al *espionaje*. Tanto comercial, como industrial.

Así pues, muchos “virus” nos vienen ahora en forma de barras del navegador, con el fin de recolectar nuestras estadísticas y datos personales (spyware), para venderselos a terceras partes. Así pueden hacer un estudio de mercado, y de paso mandarte publicidad. También tenemos lo que se denomina “ad-ware”, que nos muestra pop-ups cada cierto tiempo con publicidad, aunque tengamos el navegador cerrado.

Otros casos de virus espía, son los *keylogger*. Son muy peligrosos, pues detectan cualquier pulsación en el teclado, y la registran en un archivo que después manda por correo al autor del programa. De éste modo, nos pueden quitar las contraseñas e incluso en casos más serios, pueden obtener nuestra información de crédito.

Los casos de virus más serios en la actualidad exísten para espionaje gubernamental, como por ejemplo [Flame](#).

## 5. Los virus, y Linux

Es una creencia popular que Linux es un sistema inexpugnable. Y es una creencia totalmente falsa. El hecho de que Linux sea Open Source, para nada implica que no se pueda desarrollar malware para él. Todo lo contrario, facilita que un cracker pueda encontrar una vulnerabilidad en el sistema que se pueda explotar.

[Aquí](#) podeis encontrar una lista de Virus para Linux. Uno de los más notables probablemente sea [Hand of Thief](#).

La razón por la que poca gente se centra en éste sistema, o en Mac, es porque no está lo suficientemente extendido. Siendo siempre Windows el sistema preferido por los crackers.

## 6. Antivirus

En un vago intento por proteger a los usuarios de las amenazas de éstos virus, muchas empresas han desarrollado programas antivirus, que permiten su detección y, en ocasiones, eliminación.

No obstante, son bastante inútiles, tál y como ya expliqué en [ésta entrada](#).

Supongo que, como siempre, la seguridad depende sólo y exclusivamente del usuario. E incluso es sabido que éstas compañías que desarrollan antivirus, también desarrollan virus para alentar a la gente a pagar por su protección.

## 7. Producción de virus

Los virus se producen de igual forma que cualquier otro programa. En la antiguedad se usaba mucho C y ASM. Durante la época de W9x, se usaba también scripts en Visual Basic Script.

Pegado de <<http://nininiserver/WORDPRESS/?p=875>>

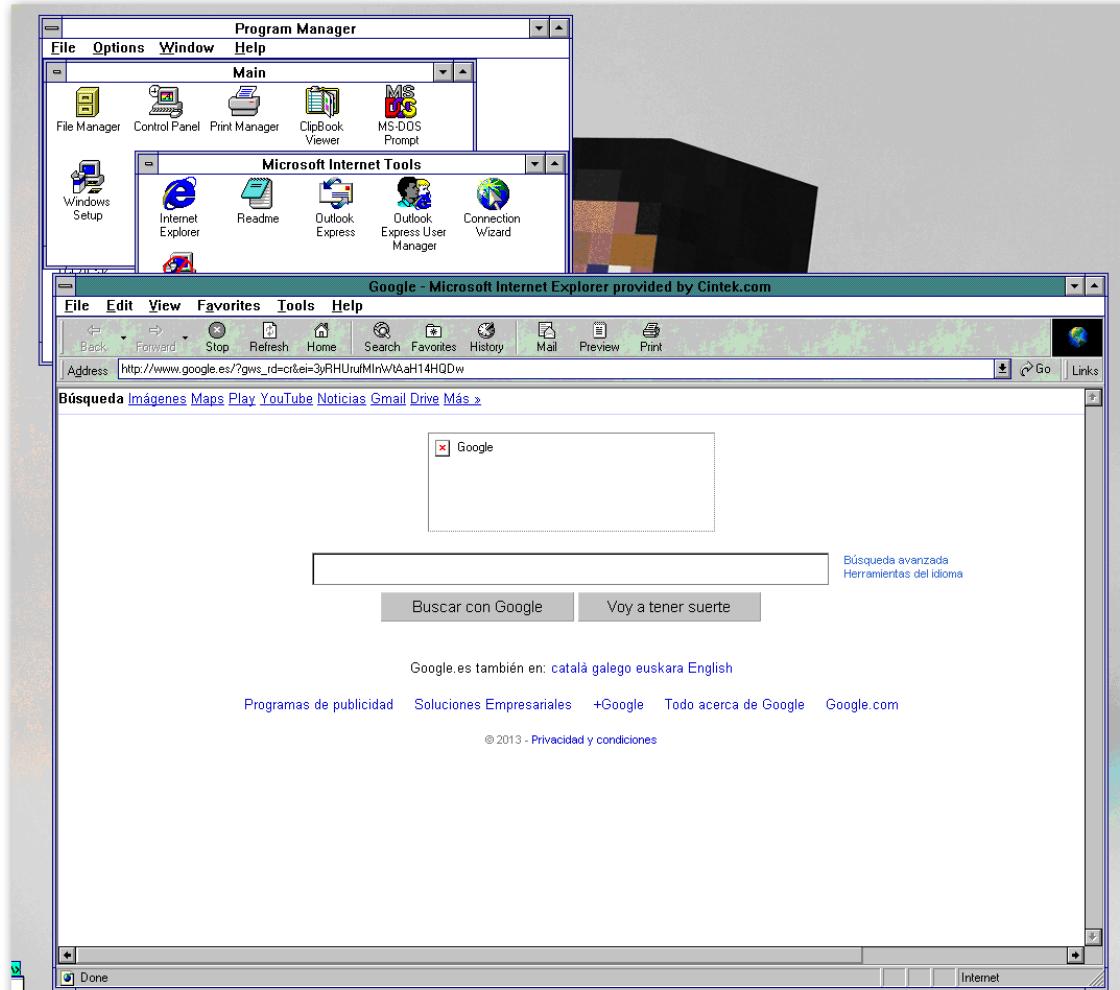
# Internet en Windows 3.11 for Workgroups

sábado, 12 de julio de 2014

1:37

## Reto superado – Internet en Windows 3.11 for Workgroups

[September 28, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



¡Lo conseguí! Despues de mucho tiempo, he conseguido conectar una máquina virtual con Windows 3.11 for Workgroups, a internet. El problema es que apenas hay páginas que ver, pues la mayoría harán que tu navegador colapse, pero... ¡Eh! Es algo.

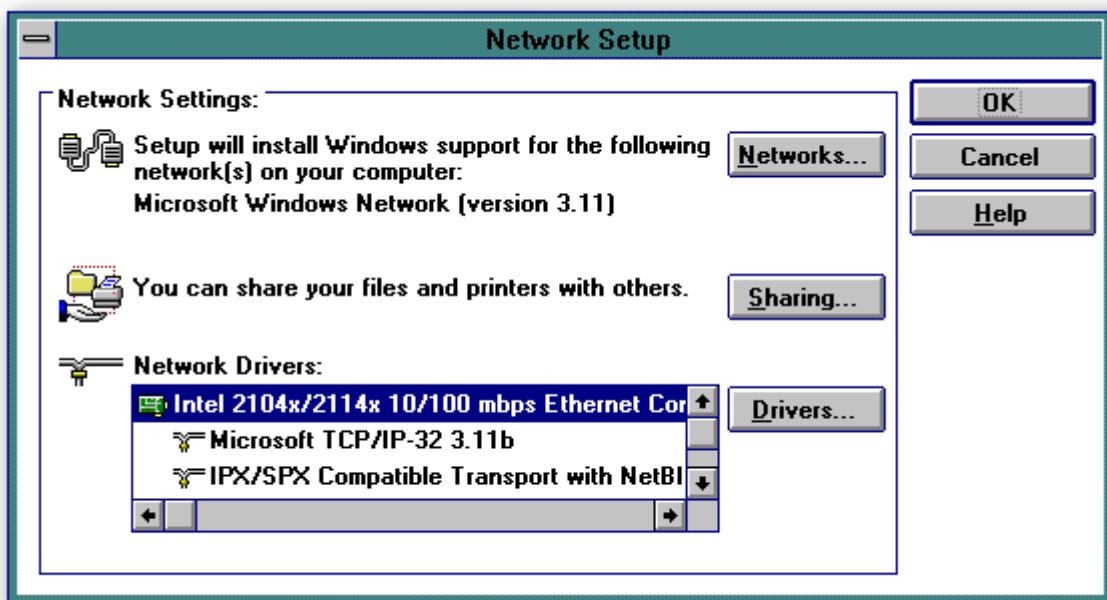
En realidad es todo un pelín más complejo de lo que aparenta, pero tampoco es tán complicado. Hacen falta los drivers de la tarjeta de red del [kit del otro artículo](#), una pila TCP/IP que funcione, y un navegador (Internet Explorer 5 funciona, Mosaic no). Me he molestado en subir [una ISO](#) con algunas de éstas utilidades, entre otras cosas.

La instalación es un poco compleja. Para empezar, hay que cambiar el adaptador de red del ordenador virtual. En configuración de red, el Adaptador 1 lo ponemos en Red compartida (NAT), ya que no disponemos de modem, así que obtendremos la conexión mediante area local (desde nuestro ordenador host).



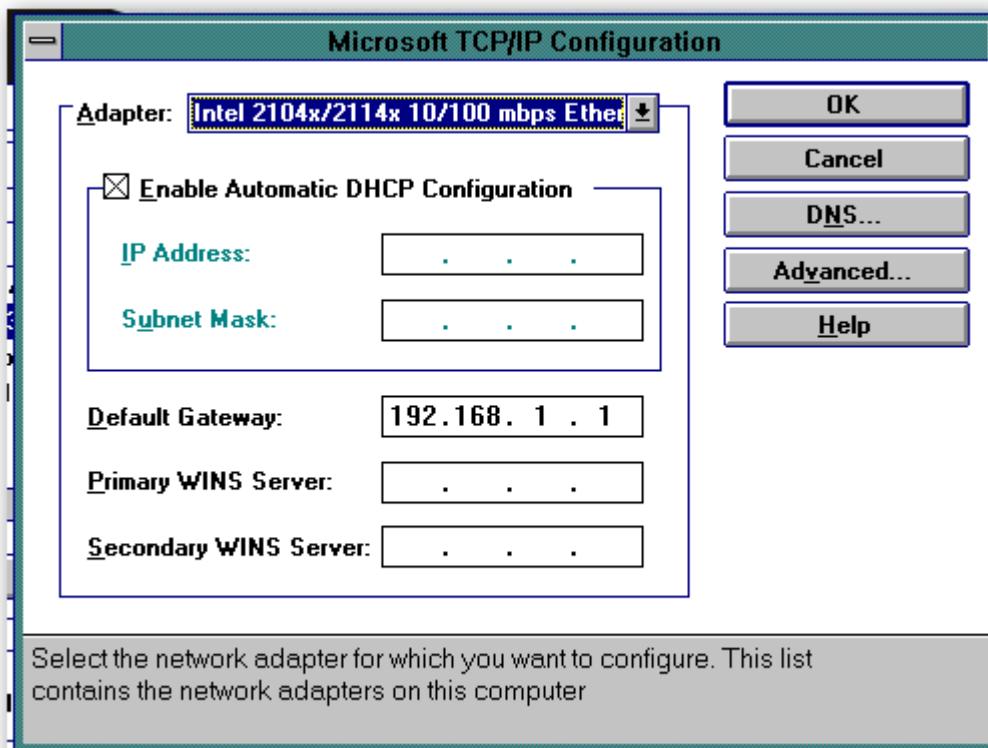
Bien, vamos a lo complejo. Primero instalamos los drivers de la tarjeta de red, suponiendo que no lo hayais hecho ya. Le dais a Windows Setup (en Main) > Options > Change Network Settings. Le dais a Drivers..., y despues a Add Adapter. En la siguiente ventana, a Unlisted or Updated Netwoerk Adapter, y a OK.

Os pedirá un disquette. Insertais el nicd.vfd de mi Kit, y le dais a Browse. Entre todas las carpetas, hay una que pone algo así como WFW311. La seleccionais, y le dais a OK. A continuación os irán pidiendo disquettes. Habrá algunas veces que os pida uno de la instalación de Windows 3.11, y otras os pedirá el nicd.vfd. Es un poco caótico, pero unos cambios de disquette despues, estará instalada. Reiniciamos despues.



Ahora hay que instalar el protocolo TCP/IP. Es fácil, copiamos el archivo tcp32b.exe que se encuentra en nuestra iso, a C:WINDOWStcp32b.exe, y lo ejecutamos. Se descomprimiran e instalarán algunos archivos. Reiniciamos.

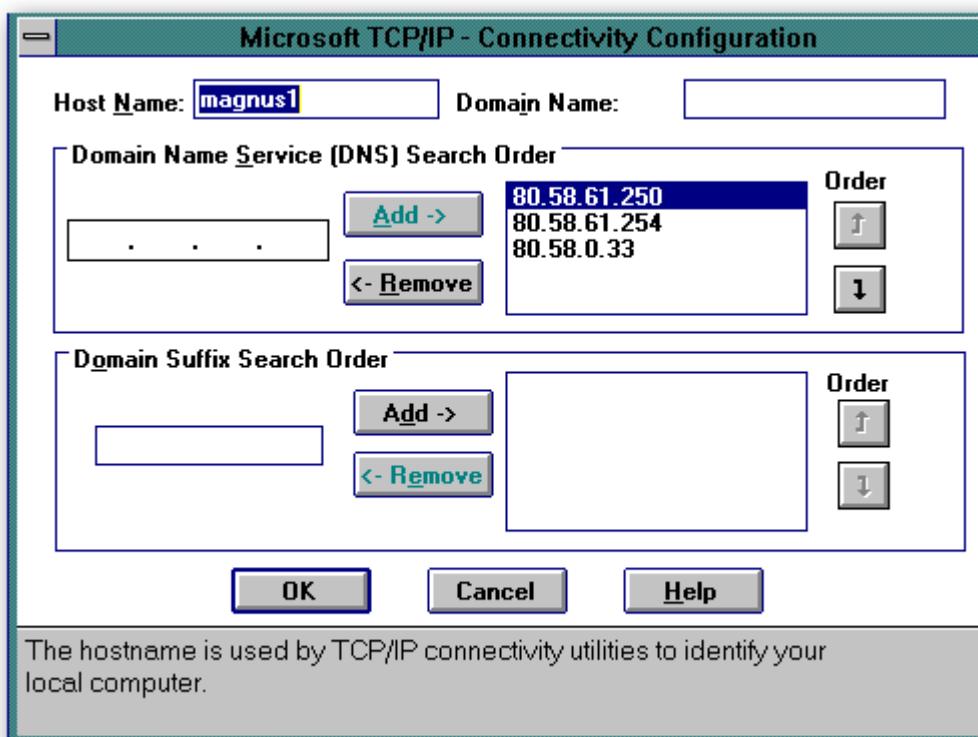
Ahora hay que añadir el protocolo a nuestra red. Fácil, vamos otra vez a la pantalla de configuración de Red donde hemos estado antes (Network Setup), hacemos click en drivers y en la pantalla que nos aparece, seleccionamos nuestra tarjeta de red, y le damos a Add Protocol. Seleccionamos Microsoft TCP/IP-32 3.11b, y le damos a OK. Si nos pide algún disquete, en su lugar le especificamos la dirección donde hemos guardado la pila (C:WINDOWS). Despues de instalar, nos aparecerá la configuración TCP/IP



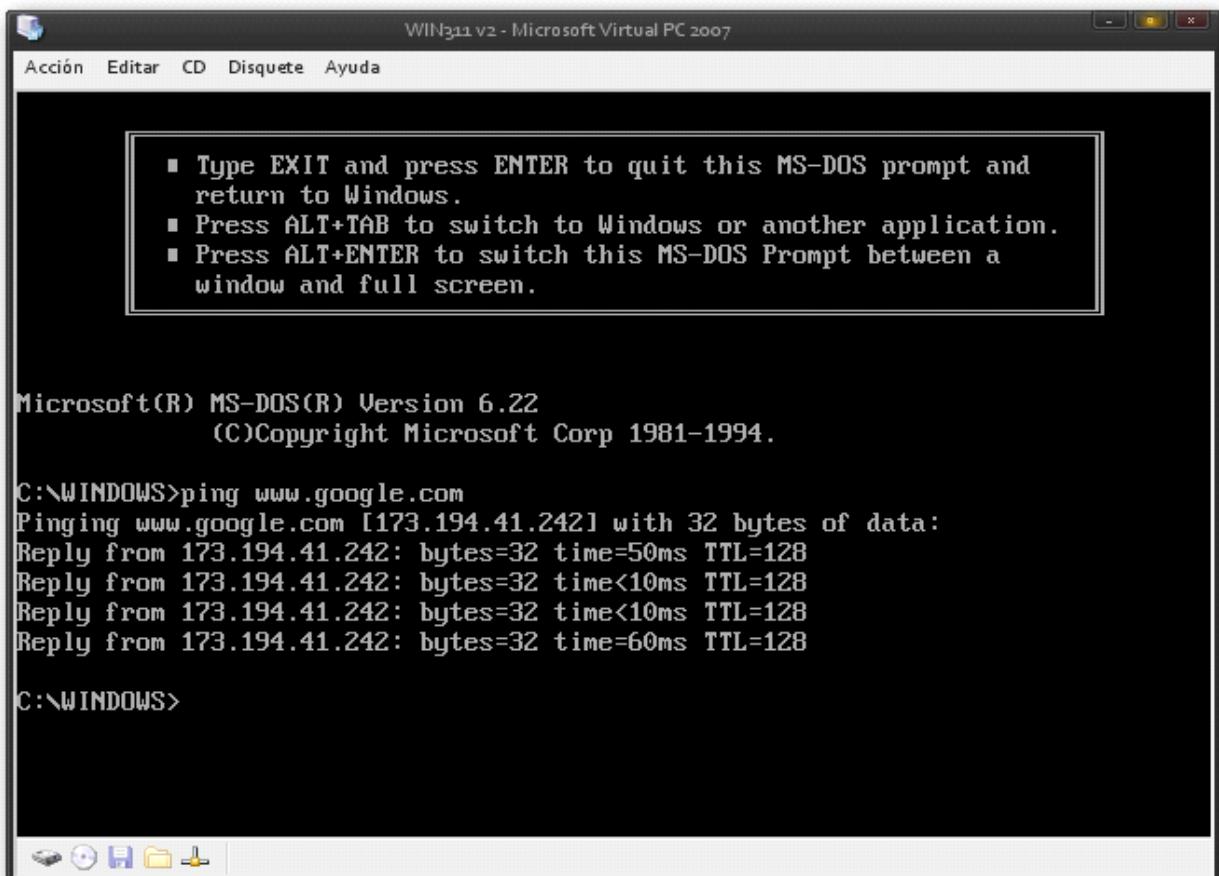
Seleccionamos “Enable Automatic DHCP Configuration”, y en Default Gateway, 192.168.1.1. Hacemos click en Advanced, y en Windows Networking Parameters, hacemos click en Enable DNS for Windows Name Resolution. Le damos a Ok, y ahora vamos a DNS. En Host Name ponemos lo que queramos, y en Domain Name Service (DNS) Search Order, añadimos estas 3 IP (Si vivís en España):

- 80.58.61.250
- 80.58.61.254
- 80.58.0.33

O si vivís en otra parte del globo, buscad DNS cerca de vuestra ubicación (Una búsqueda en Google bastará).



Y con ésto, la TCP/IP está configurada. Hacemos click en Ok unas cuantas veces, y reiniciamos. Volvemos a Windows 3.11, e iniciamos una ventana MS-DOS (MS-DOS Prompt). Ahora hacemos un ping a cualquier dirección, y miramos si hay respuesta. En caso afirmativo, estamos en linea.



Si no la hay, probad antes de nada a hacer un ping usando una IP en lugar de un nombre de dominio (por ejemplo, 173.194.41.242). Si hay respuesta, reconfigurad las DNS, pues no funcionan. Si no, comprobad primero si teneis conexión a internet desde el ordenador host, y si la hay, comprobad los pasos descritos de nuevo.

Una última cosa, en la configuración de Red, en los dispositivos de red (Network Drivers), estableced TCP/IP como protocolo primario haciendo click en Set as Default Protocol.

Ahora queda una cosa. Instalar el navegador (Internet Explorer 5). Basta con ejecutar el archivo ie5win31.exe contenida en la ISO que os he dejado, pero ¡Ojo!

Si os pide instalar un modem, no lo hagais, haced click en Cancel y omitíd ese paso. Así mismo, no ejecutéis el navegador al instalarlo. Nada más instalarlo, reiniciad, e id al Panel de control (Control panel), y a Internet. En Home Page, haced click en Use Blank (hacedme caso, la mayoría de páginas bloquearan el navegador). Ahora haced click en la pestaña Connections, y en Connection, haced click en Your local area network or another dialer. Haced click en Ok, y reiniciad una vez más.

Ahora sí, iniciad Internet explorer, si os sale un asistente de conexión lo rechazais (Cancel), y os conectais a alguna página (como google). ¡Y listo! Con eso basta. Ahora podemos conectarnos a internet, usar protocolos como el FTP, e incluso crear una red local. ¡Por fin!

Recordad, sólo funciona en la versión **For Workgroups**. La versión "casera", por alguna razón, no nos lo permite, o no de una forma tan sencilla.

Pegado de <<http://nинjiserver/WORDPRESS/?p=849>>

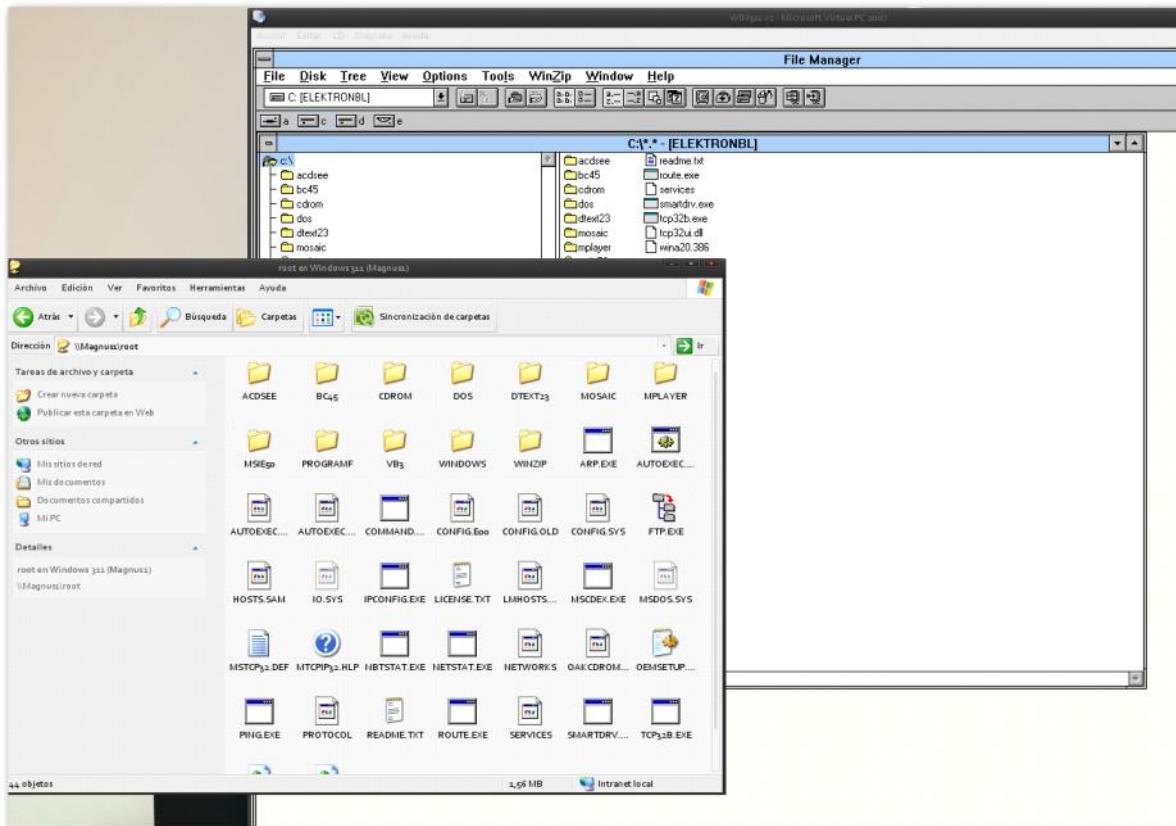
# Redes en Windows 3.11

sábado, 12 de julio de 2014

1:37

## Crear una red en Windows 3.11

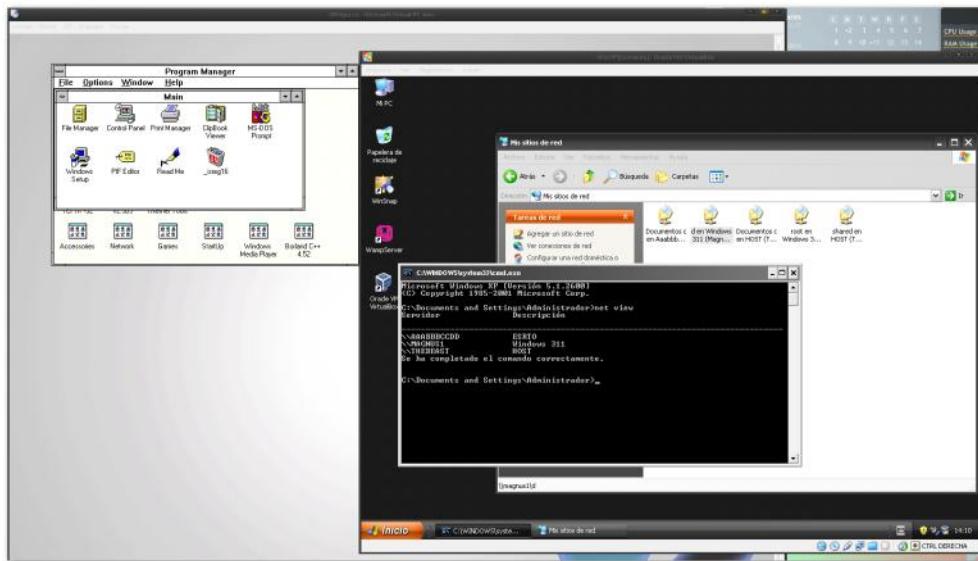
[September 29, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



Y llegamos al último capítulo de “cosas que hacer en Windows 3.11”. En los otros dos capítulos he mencionado en algunas ocasiones la posibilidad de crear una red con Windows 3.11. Por “Crear una red” no sólo me refiero a conectar ordenadores virtuales, sino también el host (donde se ejecutan éstos ordenadores virtuales). Ésto es muy útil a la hora de intercambiar archivos de forma rápida y sencilla, ya que las extensiones de los ordenadores virtuales no se pueden habilitar para Windows 3.11. Me refiero a lo de las carpetas compartidas de Virtual PC, o el drag & drop de VirtualBox.

También es útil tener otro ordenador virtual en red, si eres desarrollador. MS-DOS y Windows 3.11 no tienen protección de memoria como los sistemas más modernos, y cualquier fallo en un puntero puede hacer que incluso te cargues la configuración de la CMOS (me ha pasado).

Siguiendo éstos pasos también podemos conectar más ordenadores con otros sistemas operativos. No sólo Windows 3.11. Incluso desde diferentes ordenadores virtuales con diferentes sistemas operativos.



### Configurar una red local en el ordenador host

Antes de nada, confieso que aún soy un poco novato en el tema de redes, pues nunca he dispuesto de una para experimentar con ella. Es por ello que puede que alguna de la información aquí expuesta, no simbolize los métodos más adecuados para conseguir ésta tarea. Lo aquí expuesto es sólo el método que, a mi personalmente, me ha dado resultados positivos. Siguelos por tu cuenta y riesgo.

Configurar una red puede ser un tema un poco complejo, si no sabemos como. La cuestión es que debemos crear un Puente de red, ya que en Windows 3.11 no podemos obtener internet directamente desde nuestra tarjeta de red (recordad, lo configuramos para conectarnos desde otro dispositivo). Ésto implica que necesitaremos también otro dispositivo de red que podamos puentejar.

Virtual Box nos instala una serie de dispositivos de red virtuales que le sirven a la máquina para conectarse a internet. Así que, aunque sé que suena chapucero, una buena idea sería instalar Virtual Box (Si no lo teneis), sólo por el adaptador. Podeis descargarlo de [aquí](#).

Una vez instalado, nos instalará un dispositivo de red llamado “VirtualBox Host-Only Network” en nuestras conexiones de red.



Ahora hay que preparar el ordenador para la red. Basta con crear un puente de red, y configurar el ordenador para compartir carpetas. Una manera rápida y sencilla de hacer esto, es mediante el *Asistente para configuración de red*, del panel de control. Lo tiene el XP, y deberían tenerlo también los más modernos (De hecho, Windows 8 te configura una red domestica de forma automatica al instalarlo, si se lo permites).

Síplemente, sigue con especial atención a todo lo que te diga el asistente. Cuando te pregunte por el método de conexión, selecciona “Este equipo se conecta directamente a Internet”. De no hacerse así, le estaremos diciendo al asistente que lo que queremos es conectarnos a internet desde otro ordenador, lo cual no es bueno. A menos que sea cierto que queremos conectarnos desde otro ordenador.

## Asistente para configuración de red

### Seleccione un método de conexión.



Seleccione la opción que mejor describa el comportamiento de su equipo:

- Este equipo se conecta directamente a Internet. Los otros equipos de mi red se conectan a Internet a través de este equipo.

[Ver un ejemplo.](#)

- Este equipo se conecta a Internet a través de una puerta de enlace residencial o de otro equipo de mi red.

[Ver un ejemplo.](#)

- Otros

Obtenga más información acerca de [configuraciones de red doméstica o de oficina pequeña](#).

< Atrás

Siguiente >

Cancelar

Después te preguntará por una descripción y un nombre para el equipo. Pon lo que quieras. Lo más importante viene después, que es el nombre del grupo de trabajo.

El nombre del grupo de trabajo debe siempre coincidir con el del resto de ordenadores de la red. Así que si el nombre del grupo de trabajo que has establecido en tu 3.11 es “PAQUITO”, así será en el resto de ordenadores.

## Asistente para configuración de red

### Dé nombre a su red.



Especifique un nombre de grupo de trabajo para su red. Todos los equipos en su red deben tener el mismo nombre de grupo de trabajo.

Nombre del grupo de trabajo:

Ejemplos: CASA u OFICINA

< Atrás

Siguiente >

Cancelar

Luego te pedirá si quieres activar el uso de archivos e impresoras compartidos. Dale a Activar. Revisa los parámetros de la configuración por última vez, dale a Siguiente y espera un poco. Si te pregunta si quieres añadir algunos dispositivos de red a tu red, añade todos, **menos el de área local**.



Por último, te dirá si quieres crear un disco de configuración. Selecciona “Finalizar el asistente.” Reinicia, y ya tienes tu red configurada en el ordenador host.

Todos ésto se puede hacer también de forma manual, sólo que por mi falta de experiencia en temas de redes, no sé como. Una vez más, realiza ésta operación por tu cuenta y riesgo. Y recuerda hecharle un ojo a la seguridad de tu sistema. No te vendría mal instalar un Firewall.

### Configurar una red en Windows 3.11

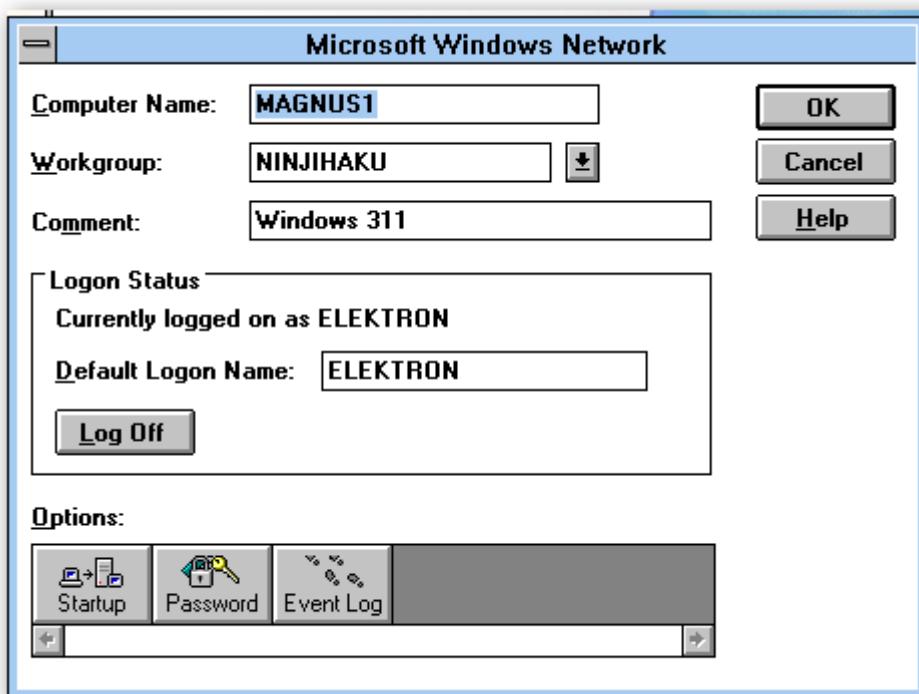
Lo primero de todo, y si aún no lo habeis hecho, es seguir [éstos pasos](#). Pero con una diferencia. En la configuración de red del ordenador virtual, en los adaptadores de red, hay que usar el Puente minipuerto MAC, con el dispositivo de red de VirtualBox Bridged Network. Ésto es común para todos tus ordenadores virtuales.



Si no dispones de él, es porque no tienes un puente en el dispositivo de VirtualBox. Lo cual quiere

decir que algo ha fallado en los pasos anteriores.

Ya sólo queda asegurarnos de que el nombre del grupo sea el correcto. En windows 3.11, vamos al panel de control (Control Panel) en Main, le damos a Network, y en Workgroup establecemos Workgroup con el nombre correcto. También podemos modificar el nombre, y un comentario por si queremos reconocer nuestro equipo en la red.



Reiniciamos todo, e iniciamos sesión en todos los ordenadores. Ahora, en el ordenador host, vamos a la consola. Escribimos sin comillas “net view”, y le damos a enter. Deberíamos ver nuestro ordenador virtual en una lista.

A screenshot of a Windows XP command prompt window titled "Console2". The window shows the output of the "net view" command. The output is as follows:

```
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Console2>net view
Servidor           Descripción

----- 
\\AAABBBCCDD      ESRTO
\\MAGNUS1          Windows 311
\\THEBEAST         HOST
Se ha completado el comando correctamente.

C:\Console2>
```

The window has a dark background with yellow text. The title bar is "Console2". The command prompt line is "C:\Console2>". The output starts with the copyright notice and then shows the results of the "net view" command, listing network resources by their share names and descriptions. The window has standard Windows XP window controls (minimize, maximize, close) at the top right.

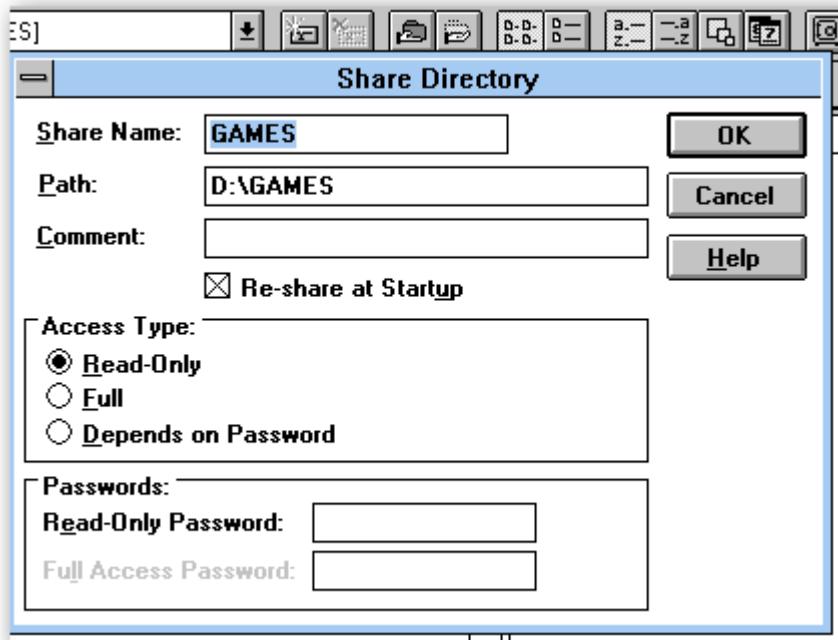
Si ésto es así, habremos configurado por completo nuestra red.

### Crear carpetas compartidas

Volvemos a Windows 3.11, y vamos al administrador de ficheros (File Manager). Vamos a la carpeta que queremos compartir y la seleccionamos. En la barra de herramientas, al lado de la lista desplegable con el nombre de la unidad, hay una lista de botones. Uno de ellos tiene una mano que sostiene una carpeta en gris. Ese es el botón de compartir carpeta. Basta con darle click.

En la siguiente ventana, podemos establecer el nombre con el que aparecerá la carpeta para los demás usuarios, la ubicación (automática), y un comentario. Lo más importante es la opción “Reshare at Startup”, que nos permite automontar la carpeta en la red al iniciar el ordenador, y el tipo de acceso.

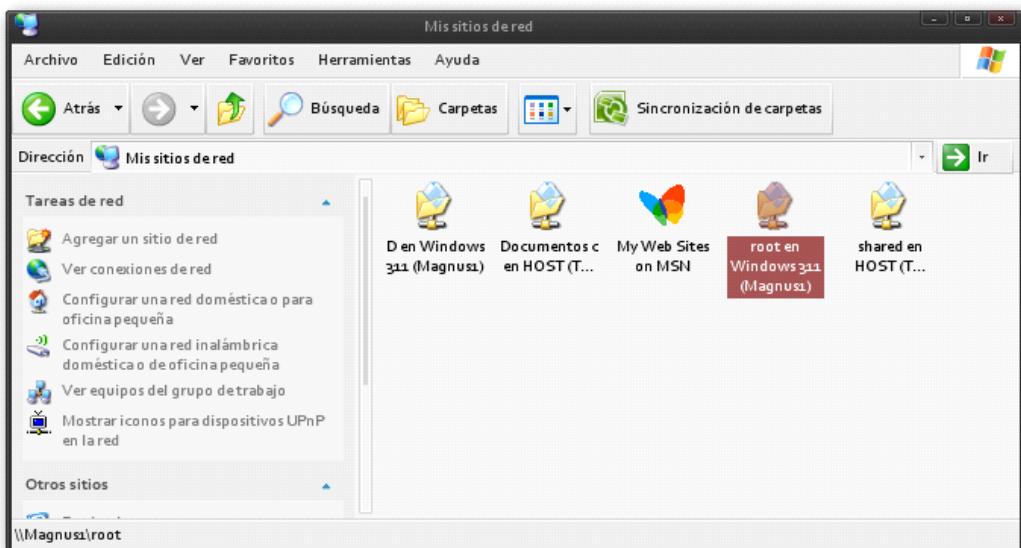
Read-Only solo nos permite leer los archivos que hay dentro, pero no modificarlos. Full, es acceso total (Lectura y escritura). Y “Depends on Password” nos permite establecer dos contraseñas, dependiendo de cual introduzcamos nos dará unos permisos, u otros.



Nunca viene mal ponerle una contraseña, pero como no vamos a meter datos sensibles, ni nos importa una posible pérdida de datos (pues podemos hacer back ups del disco virtual), podemos darle a Full sin contraseña. Aunque en caso contrario, conviene establecer una contraseña.

Le damos a Ok, y con eso la carpeta estará compartida.

Ahora, para acceder desde otro ordenador a nuestra carpeta, vamos a ese ordenador (por ejemplo el host), abrimos una carpeta cualquiera, y seleccionamos “Mis sitios de red”. Dentro aparecerán una a una las carpetas que tenemos compartidas.



Simplemente accedemos a ellas como si fuesen cualquier otra carpeta.

Pegado de <<http://nininiserver/WORDPRESS/?p=854>>

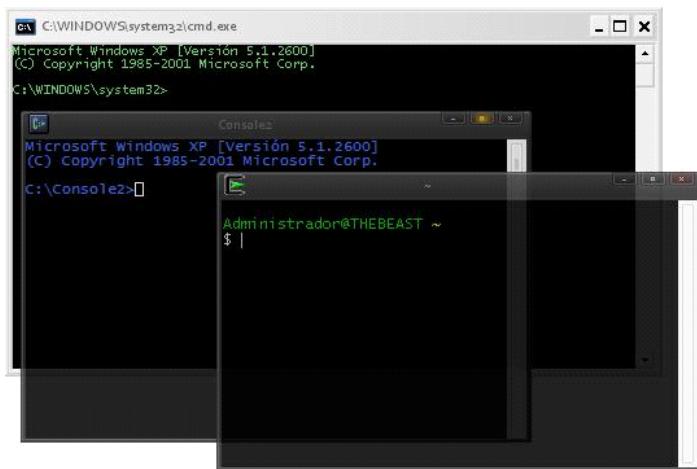
# ¿Se pueden crear juegos para la consola de comandos?

sábado, 12 de julio de 2014

1:39

## La consola de comandos de Windows. ¿Se puede crear juegos para la consola de comandos?

[August 18, 2013](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



De lo que voy a hablar hoy es de la famosa, pero incomprendida consola de Windows. A veces llamada también “Símbolo del sistema”, o en ocasiones y de forma muy vulgar, “eme-ese-dos”. Precisamente, la idea principal de éste artículo, es hacer comprender a la gente cuál es la diferencia entre MS-DOS, y la consola de comandos. Pero antes de eso, un poquito de historia.

### ¿Qué es MS-DOS?



[MS-DOS](#) fué el primer sistema operativo desarrollado por Microsoft en los años 80, con 8 versiones desde la 1, hasta la 8 y final. No obstante, no todas las 8 versiones fueron sistemas operativos como tál.

MS-DOS fué un sistema operativo independiente hasta su versión 6. A partir de entonces, los sistemas Windows tomarían el liderazgo, empezando por [Windows 3.11](#) (1.0 y 2.0 no cuentan para mi).

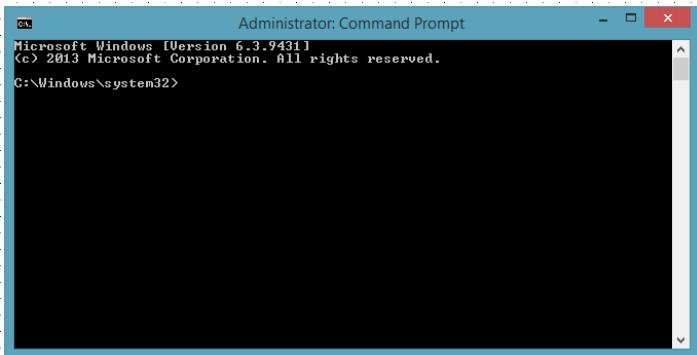
Windows 3.1x no es en realidad un sistema operativo en sí, más bien es como un “shell”, una aplicación que se solapa a lo que es el nucleo del sistema. Podría ser algo así comparable con lo que es desde Windows 95, el [explorer.exe](#).

Al arrancar Windows 3.11, MS-DOS pasaba a ser algo así como un “subsistema”. Un sistema operativo, que exíste dentro de otro. Y por ello, en Windows 95, ya no era un shell, sino que Windows ya pasaba a ser un sistema operativo integro, en conjunto con MS-DOS (COMMAND.COM). La versión 7 de MS-DOS coexistiría con los sistemas [Windows 95](#) y [Windows 98](#), la versión SE incluida. Mientras que la 8 coexistiría con [Windows 2000](#) y [Me](#).

Ninguno de éstos sistemas operativos incluía una consola de comandos, sino que usaban ese subsistema MS-DOS para ejecutar aplicaciones de 16 bits. Microsoft pensó en eliminar el subsistema MS-DOS para Windows Millenium (Me), pero no fué hasta más tarde, en [Windows XP](#), que se eliminó por completo. Siendo reemplazado por la famosa [consola de comandos](#).

#### ¿Qué es la consola de comandos de Windows?





La consola de comandos es un interprete independiente de líneas de comandos. Es algo así como una aplicación, a traves de la cual se pueden mandar ordenes al sistema operativo.

Ésta aplicación no tiene nada que ver con el antiguo MS-DOS. Es una aplicación totalmente independiente, aunque conserva la mayoría de los comandos y aplicaciones que se usaban en la era de MS-DOS.

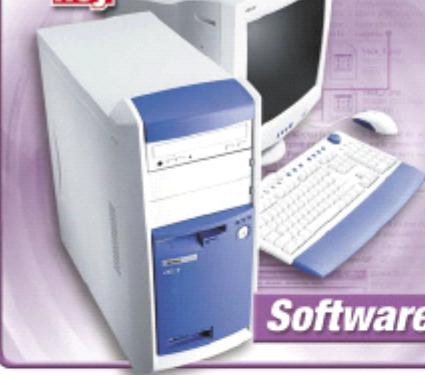
La consola de comandos está pensada sólo para aplicaciones en modo texto. Ésto quiere decir, que no puede ejecutar, por sí sola, aplicaciones que usen una interfáz gráfica de cuando la era de MS-DOS, o cosas similares. En realidad, ni siquiera ejecuta aplicaciones.

La consola actúa como un intermediario entre el usuario, y el sistema. Cuando el usuario manda una orden a la consola, lo que hace la consola es pasarsela al sistema operativo. Y el sistema operativo, muestra el resultado de la operación, a traves de la consola.

En otras palabras. MS-DOS es un sistema operativo. La consola de comandos, una aplicación para un sistema operativo. Esa es la principal y más fundamental diferencia.

No son pocos los que han caido en el error de llamar a la consola del sistema de Windows por el nombre de su hermana mayor. Incluso algunas revistas comerciales en España, no han podido evitar caer en el dichoso error de denominar a la consola como MS-DOS.

# Computer Trucos Hoy



**Software**

## Windows XP



### 1 MS-DOS a todo color

¿Cansado ya del aburrido y soporífero color blanco y negro de las **ventanas DOS** [01 \(Pág. 34\)](#)? No lo dudes ya más, y dales una nota de color siguiendo este truco. Gracias a él, podrás determinar cuál será el color de fondo de estas ventanas y el del texto que escribas en ellas.

**1** Para hacer este truco debes recurrir al **Registro de Windows** [02 \(Pág. 34\)](#), así que haz click en y , teclea

Es justo el único que te interesa de todos los que estarás visualizando.

**3** Así que haz doble click sobre él, activa la casilla  Hexadecimal y teclea por ejemplo esta entrada,



### Sumario

Windows XP	32
Windows 98/Me	33
OpenOffice Writer 1.1	34
Access 97/2000/2002	35
Paint Shop Pro 7/8	36
Outlook 2000/2002	38

y pincha ya sobre y en . Esta vez, la ventana DOS que se muestra tiene un aspecto muy diferente al habitual:

`Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

**7** A continuación, teclea el comando

**Ejecutar**  
 Escribe el nombre del recurso de Internet que Abrir: `cmd /t:1`

seguido del código que representa al color de fondo, `0x10`, y el del color del texto, `0x1E`. Tras pulsar una vez sobre el botón **Aceptar**, comprobarás el resultado:

`Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

`Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

**8** Cierra la ventana DOS anterior, y prueba ahora a teclear El efecto será este otro:

**Ejecutar**  
 Escribe el nombre del recurso de Internet Abrir: `cmd /t:FS`

`Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

`Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

No quiero hechar tierra encima de ésta revista, no es una revista que me disguste, y de hecho, es con la que me inicié en el mundillo. Pero cometer errores como éste cuando estás vendiendo algo, no es permisible. Tema aparte, el numero en el que cometieron éste error es de hace bastantes años, y probablemente hayan cambiado ya varias veces de redactor.

Entonces, ¿Cómo es que puedo ejecutar juegos de MS-DOS a través de la consola?



Símplo. Por compatibilidad, los sistemas más modernos de Windows incluyen una máquina virtual que simula el funcionamiento de MS-DOS ([VDM](#)). Cuando iniciamos una aplicación de 16 bit, de la era de MS-DOS, la consola no ejecuta el programa, sino que inicia ésta máquina virtual, la cual intenta ejecutar la aplicación, y la muestra en la ventana de nuestra consola.

Sin embargo, si pinchas de la manera habitual sobre , la ventana mostrará el aspecto de siempre, es decir, éste:

`F:\WINDOWS\System32\Microsoft Windows XP [Versión 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
F:\Documents and Settings\Fuencisla>`

## **¿Quiere eso decir que no puedo desarrollar juegos usando la consola de comandos?**

Y aquí es donde quería llegar yo.

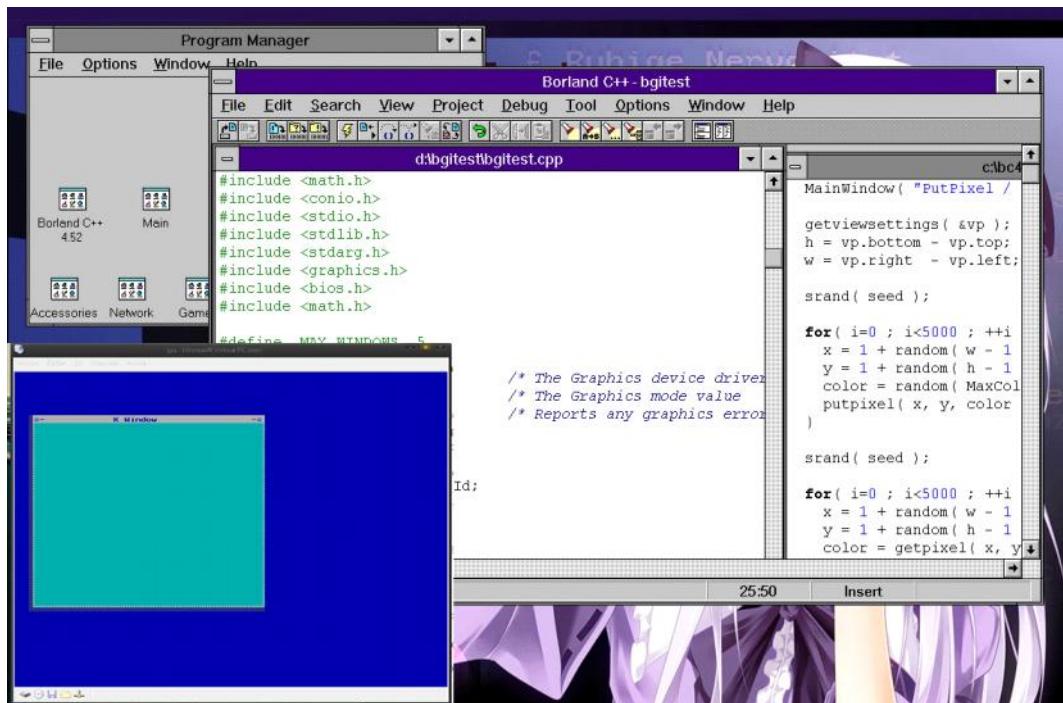
La respuesta, es NO. O al menos, no juegos como por ejemplo un “Doom”. La consola de comandos, no está pensada para mostrar gráficos, y por ello no lo hará. Sólo podemos mostrar texto. Si con ello nos resulta suficiente para hacer un juego (como una aventura conversacional), entonces perfecto. Pero si necesitamos mostrar gráficos, entonces es un no definitivo.

Si lo que queremos es desarrollar juegos para MS-DOS usando gráficos, hoy día no merece la pena complicarse la vida. Los sistemas más modernos ya no ejecutan correctamente juegos de la era de los 16-bit, principalmente porque el hardware ha evolucionado mucho, y también porque los sistemas operativos modernos añaden [capas de abstracción](#) que sirven como capa de seguridad para evitar que un programa pueda interactuar abiertamente con el hardware del ordenador (Motivo por el cual se creó [Direct X](#)). Incluso el propio microprocesador incluye varios niveles de protección, llamados [anillos](#), que determinan qué instrucciones puede ejecutar una aplicación, y cuales no. No obstante, además de la VDM implementada en Windows, tenemos también [software de virtualización](#) a nuestro alcance. La mayoría de los gamer “retro” utilizan [DOSBOX](#) para emular un subsistema MS-DOS y poder así jugar a juegos antiguos de 16 bit. Y por cierto, si eres uno de esos retro gamer, seguro que te encontrarás [un paraíso en esta página](#).

Volviendo al tema que nos concierne, si quieres programar juegos retro para jugarlos en sistemas Windows modernos, tu única opción es usar Direct X, o algún SDK (o una librería como Alegro, o incluso Open GL), y darle a tu juego un estilo retro. Intentarlo por la fuerza a base de usar instrucciones [VGA en ensamblador](#), es masoquismo.

## **¿Y si soy tán cabezota que quiero hacerlo exclusivamente de 16 bit?**

Si eres tán cabezota como yo, que te mola lo retro y no quieres usar otra cosa, tu única posibilidad es la de usar software de virtualización para simular un sistema con MS-DOS y Windows 3.11, y programar desde ahí. O un ordenador viejo con dichos sistemas, si lo tienes.



Puedes luego pasarlo a traves de un disquette a tu sistema operativo moderno. Si no tienes disquettera, y muy probablemente ya no tengas, puedes usar una disquettera virtual. Puedes conseguir el VFD desde [aquí](#). Es un pack donde vienen incluidos programas y utilidades antiguas mencionadas en [éste artículo](#). Y por cierto, aún sigo sin poder conectarme a internet. Aunque conseguí [conectar dos ordenadores virtuales por red](#) (Usando la versión NT, eso sí).

**Pero, ¡Yo he visto a gente usar la consola para mostrar gráficos!**



No miento. He dicho que la consola no puede mostrar imágenes. Y no puede. Sólo imprime texto.

No obstante, podemos superponer un panel (una ventana, para que se entienda) dentro de la consola. De esta forma, logramos el efecto de haber colocado una imagen en la consola. En C/C++ se puede hacer mediante la librería windows.h, entre otras. Y en C#, supongo que se podrá usar GDI+. [Un ejemplo un poco chapucero.](#)

**En resumen...**

MS-DOS es un sistema operativo. La consola de comandos, un programa diseñado para trabajar con comandos.

Puedes usar la consola para cualquier cosa que sólo requiera texto. NO puedes usar la consola para mostrar gráficos o manipular el hardware de tu ordenador. Ésto quiere decir, que nada de juegos para la consola de Windows, al menos con gráficos.

Pegado de <<http://nинjiserver/WORDPRESS/?p=795>>

- // put a bitmap image on a Windows Console display  
// BCX basic original by Joe Caverly and Kevin Diggins  
// BCX generated C code modified for PellesC/Dev-C++  
#include <stdio.h>  
#include <string.h>  
#include <windows.h> // Win32Api Header File  
#include <iostream>  
#include <ctime>  
  
static HWND hConWnd;  
  
HWND BCX\_Bitmap(char\*,HWND=0,int=0,int=0,int=0,int=0,int=0,int=0,int=0,int=0);  
HWND GetConsoleWndHandle(void);  
HWND hWnd = ::GetForegroundWindow();  
using namespace std;  
  
void clearconsole() {  
COORD topLeft = { 0, 0 };  
HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);  
CONSOLE\_SCREEN\_BUFFER\_INFO screen;  
DWORD written;  
  
GetConsoleScreenBufferInfo(console, &screen);  
FillConsoleOutputCharacterA(  
 console, ' ', screen.dwSize.X \* screen.dwSize.Y, topLeft, &written  
>);  
FillConsoleOutputAttribute(  
 console, FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE,  
 screen.dwSize.X \* screen.dwSize.Y, topLeft, &written  
>);  
SetConsoleCursorPosition(console, topLeft);  
}  
  
int main(int argc, char\* argv[]){  
 hConWnd = GetConsoleWndHandle();  
 char delx = 0x127;  
 int chars = 0, lines = 1;  
 char\* filename = "imgs\\Cirno.bmp";  
  
 if(argc > 1 && strcmp(argv[1], "-f") == 0)  
 {  
 filename = argv[2];  
 }

```

clearconsole();

if(argc < 2)
{
    std::cout << "\t\t\t _____" << std::endl;
    std::cout << "\t\t\t/\\" << std::endl;
    std::cout << "\t\t\t| Usage: |" << std::endl;
    std::cout << "\t\t\t| cirmosay [-f(filepath)][-t] message |" << std::endl;
    std::cout << "\t\t\t|-f: A path to a bmp file.MUST be BMP |" << std::endl;
    std::cout << "\t\t\t|-t: Prints the current time and date |" << std::endl;
    std::cout << "\t\t\t| message: The words you want to print |" << std::endl;
    std::cout << "\t\t\t\\/" << std::endl;
    std::cout << "\t\t\t/" << std::endl;
    std::cout << "\t\t\t/" << std::endl;
    lines = 5;
}

else
{
    time_t t = time(0);
    struct tm * now = localtime( & t );
    char* result = asctime(now);
    std::cout << "\t\t\t _____" << std::endl;
    std::cout << "\t\t\t/\\" << std::endl;

    for(int n = 1; n < argc; n++)
    {
        if(strcmp(argv[n],"-t") == 0)
            std::cout << "\t\t\t| " << result;
        if(strcmp(argv[n],"-f") != 0 && strcmp(argv[n],"-t") != 0)
        {
            if(chars == 0)
                std::cout << "\t\t\t| ";
            std::cout << " " << argv[n];
            chars = chars + sizeof(argv[n]) + 1;
            if(chars > 20)
            {
                std::cout << std::endl;
                chars = 0;
                lines++;
            }
        }
    }

    std::cout << std::endl;
    std::cout << "\t\t\t\\/" << std::endl;
    std::cout << "\t\t\t/" << std::endl;
}

if (hConWnd)
{
// select a bitmap file you have or use one of the files in the Windows folder
// filename, handle, ID, ulcX, ulcY, width, height 0,0 auto-adjusts
    BCX_Bitmap(filename,hWnd,123,1,9+(lines*9),0,0);
    getchar(); // wait
}
return 0;
}

```

```

// draw the bitmap
HWND BCX_Bitmap(char* Text,HWND hWnd,int id,int X,int Y,int W,int H,int Res,int Style,int
Exstyle)
{
    HWND A;
    HBITMAP hBitmap;
    // set default style
    if (!Style) Style = WS_CLIPSIBLINGS|WS_CHILD|WS_VISIBLE|SS_BITMAP|WS_TABSTOP;
    // form for the image
    A = CreateWindowEx(Exstyle,"static",NULL,Style,X,Y,0,0,hWnd,(HMENU)
id,GetModuleHandle(0),NULL);
    // Text contains filename
    hBitmap=(HBITMAP)
LoadImage(0,Text,IMAGE_BITMAP,128,128,LR_LOADFROMFILE|LR_CREATEDIBSECTION);
    // auto-adjust width and height
    if (W || H) hBitmap = (HBITMAP)
CopyImage(hBitmap,IMAGE_BITMAP,W,H,LR_COPYRETURNORG);
    SendMessage(A,(UINT)STM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)
hBitmap);
    if (W || H) SetWindowPos(A,HWND_TOP,X,Y,W,H,SWP_DRAWFRAME);
    return A;
}

// tricking Windows just a little ...
HWND GetConsoleWndHandle(void)
{
    HWND hConWnd;
    OSVERSIONINFO os;
    char szTempTitle[64], szClassName[128], szOriginalTitle[1024];
    os.dwOSVersionInfoSize = sizeof( OSVERSIONINFO );
    GetVersionEx( &os );
    // may not work on WIN9x
    if ( os.dwPlatformId == VER_PLATFORM_WIN32s ) return 0;
    GetConsoleTitle( szOriginalTitle, sizeof( szOriginalTitle ) );
    sprintf( szTempTitle,"%u - %u", GetTickCount(), GetCurrentProcessId() );
    SetConsoleTitle( szTempTitle );
    Sleep( 40 );
    // handle for NT and XP
    hConWnd = FindWindow( NULL, szTempTitle );
    SetConsoleTitle( szOriginalTitle );
    // may not work on WIN9x
    if ( os.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS )
    {
        hConWnd = GetWindow( hConWnd, GW_CHILD );
        if ( hConWnd == NULL ) return 0;
        GetClassName( hConWnd, szClassName, sizeof( szClassName ) );
        // while ( _stricmp( szClassName, "ttyGrab" ) != 0 )
        while ( strcmp( szClassName, "ttyGrab" ) != 0 )
        {
            hConWnd = GetNextWindow( hConWnd, GW_HWNDNEXT );
            if ( hConWnd == NULL ) return 0;
            GetClassName( hConWnd, szClassName, sizeof( szClassName ) );
        }
    }
    return hConWnd;
}

```

Pegado de <<http://pastebin.com/hTXeLpvE>>

# Diferencias entre un ordenador antiguo, y uno moderno. Leyendas urbanas, consejos, información...

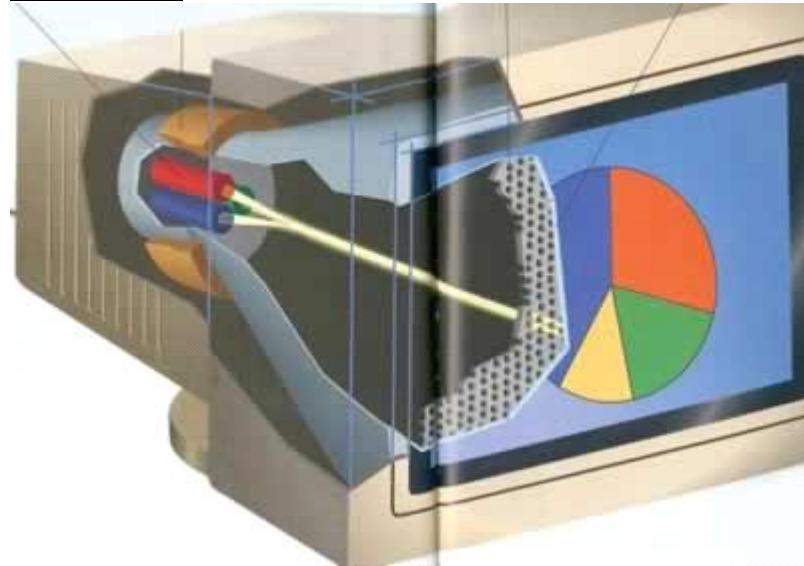
sábado, 12 de julio de 2014

1:46

[June 16, 2012](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)

Los antiguos ordenadores eran mucho más frágiles que los modernos. Es por ello que antiguamente había muchas más cosas que teníamos que tener en consideración, a la hora de usar un ordenador. Cosas que, obviamente, ya no se tienen en cuenta o, al menos, no tanto.

## Los monitores



Los antiguos monitores CRT, que aún se siguen viendo bastante en empresas donde no han invertido en actualizar algo de material, o en casas de personas muy mayores, tenían la peculiaridad de que la imagen la formaban lanzando rayos. Sí, rayos. Concretamente, "estampaban" electrones contra una pantalla. Al hacerlo, la zona del impacto se iluminaba formando un determinado color. En el caso de las pantallas de fósforo verde tan famosas, pues sólo podía ser de un color, pero en las más modernas (de color), se iluminaba siguiendo una mezcla de colores (rojo-verde-azul).

Lo más importante, es que éstas pantallas son muy susceptibles a los campos electromagnéticos, y en nuestras casas hay un sinfín de campos electromagnéticos. Como por ejemplo, en la fuente de alimentación de nuestro ordenador, o en altavoces.

Es por ello que antiguamente, nos decían que no debíamos pegar nuestro monitor a la torre del ordenador (vulgarmente conocida como CPU). Éstos campos electromagnéticos pueden afectar al tubo de rayos, imantandolo. Al imantarlo, la imagen se distorsiona de multiples maneras.



Al igual que una pantalla se puede imantar, los disquettes (unidad de almacenamiento más extendida de la época) también eran susceptibles a los campos magnéticos, ya que la información se almacenaba mediante pequeños imanes que apuntaban a un lado o a otro en función de los estados de cada bit.

Ésto ya no ocurre con la tecnología actual. Nuestras pantallas actuales (LCD, LED y plasma), ya no son susceptibles a los campos magnéticos (o al menos, no de forma tan dramática).

Otra cuestión relacionada con los CRT, son los denominados "protectores de pantalla" (no confundir con los salvapantallas).



Antiguamente, y como ya he dicho, los monitores lanzaban electrones contra la pantalla. Si habeis vivido aquella época, o habeis tenido la oportunidad de experimentarlo, seguramente habreis colocado el brazo cerca de la pantalla, y habreis notado cómo los pelos se os erizaban. Ésto son los electrones impactando contra vuestra piel. Ahora imaginad esos electrones impactando continuamente sobre vuestros ojos. ¿No creéis que puede ser un poco peligroso?

De ahí es de donde surgen éstos protectores de pantalla. Son como filtros, que impiden que los electrones pasen de la pantalla, protegiéndonos a nosotros de un posible cancer.

Evidentemente, los CRT ya no son tan peligrosos, y por tanto actualmente los protectores sólo se limitan a proteger la pantalla, sin importar lo que nos pase a nosotros.

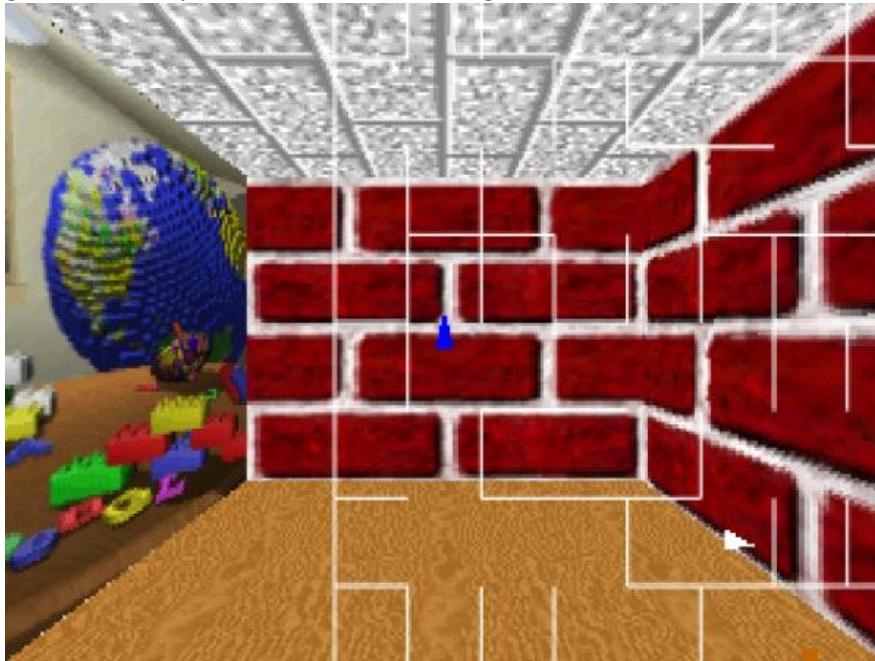
Otra consideración con los CRT, es el espacio entre la pared, y el monitor.  
Vamos a hechar un vistazo a la parte trasera de un monitor CRT:



¿Veis esas rendijas en la parte trasera? Esa es la ventilación. Ahora imaginad que lo pegais a la pared, ¿Qué creéis que pasa?

Efectivamente, para garantizar que el monitor se refrigerara adecuadamente, la ventilación debe estar separada de la pared. Así nos aseguramos de que la ventilación no está obstruida, y seguramente ahorremos unos cuantos años más de vida del monitor. Los monitores actuales, si os fijais, la ventilación no la tienen en la parte trasera. Si os fijais, está toda en la parte superior e inferior, pero nunca en la parte de atrás. Ahora imagináis por qué.

Una última cosa en relación al monitor. Los salvapantallas (ahora sí). Lejos de ser un elemento decorativo, antiguamente “salvaban” a la pantalla de que se quemase, pues a las antiguas CRT no las gustaba nada que las abrasaran con imágenes estáticas durante mucho tiempo.



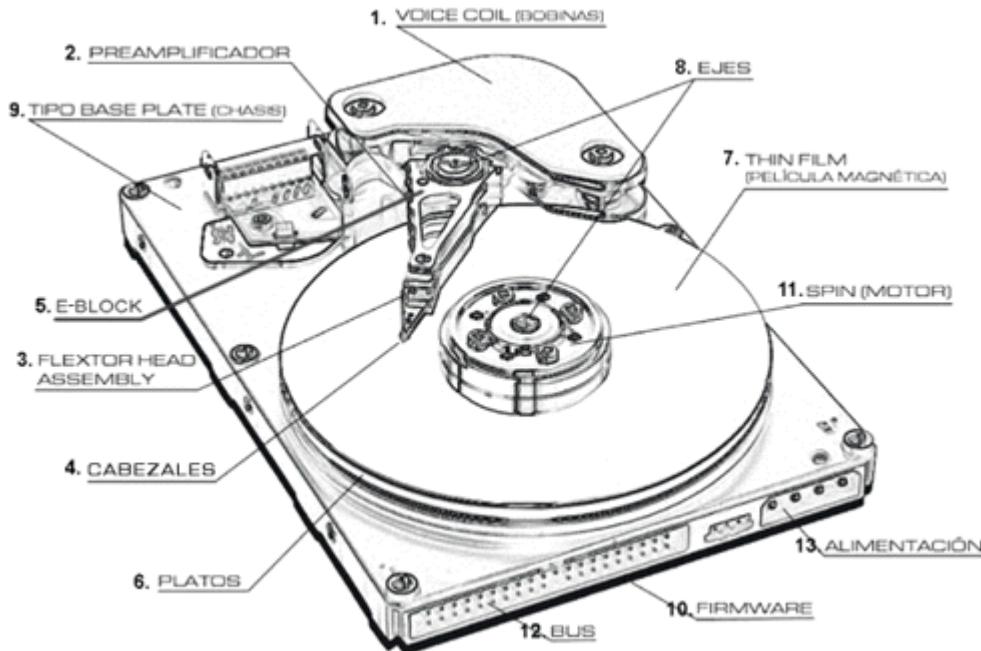
Actualmente las LCD y LED, en principio, no deberían tener ningún problema con que las dejes unas horas con una imagen estática. Hombre, yo creo que como con todo, si las dejas mucho, mucho tiempo, tal vez algo ocurra, pero no debería ser un problema. No puedo decir lo mismo de las pantallas de plasma, que sí parecen tener problemas con las imágenes estáticas. Pero eso sí, esas no

se diseñan para ser usadas con un ordenador. ¡Tenedlo en cuenta!

Recientemente he oído que hay gente que afirma que los salvapantallas sirven para ahorrar energía. El ordenador sigue en funcionamiento mientras se ejecuta, así que no. Más bien al contrario, los salvapantallas se programan como juegos, haciendo uso tanto de la CPU, como de la GPU. Con lo cual, diría que consumen más.

Y ahora que he acabado con el monitor, vamos a desmentir algunos mitos y leyendas urbanas relacionados con la torre.

## Discos Duros



Los discos duros convencionales son muy susceptibles a cualquier tipo de suciedad. Más que los CDs (son extremadamente frágiles), así que en ningún caso, a menos que ya no lo quieras y te de igual dañarlo, deberás abrirlo.

Los datos que borras de un disco duro, no se borran permanentemente. Lo que tu ordenador hace es indicar en una tabla, que un determinado rango de datos en tu disco duro (comunmente llamado "archivo"), está en desuso y, por tanto, se puede escribir encima. Mientras no se haya escrito encima esos datos siguen estando en el disco duro (invisibles para nosotros desde el SO, pero están ahí).

Ésto quiere decir que los datos eliminados pueden ser recuperados, siempre que no se haya escrito encima de ellos. Y existen programas y aplicaciones para recuperarlos. Y hablo de discos duros, como puedo hablar de discos SSD, memorias SD, FLASH ROM (Pen Drive, tarjetas de memoria de Play Station), etc.

Cuando apagas de pronto el ordenador, tu disco duro no corre riesgo de "quemarse". Ni va a explotar ni le van a salir piernas y va a salir corriendo. Pero puede ser objeto de una pérdida de datos (y como siempre, ésto es aplicable a cualquier otro tipo de memoria, excepto la RAM, que está diseñada para perder todos sus datos al apagar, lo hagas bien o no). El motivo es que el ordenador lee y escribe constantemente datos, la mayor parte del tiempo sin tú saberlo (en segundo plano, o a escondidas). Si apagas de golpe y porrazo el ordenador mientras está escribiendo, pues pasa que puede haber una corrupción de datos en esa parte, aunque actualmente los sistemas de escritura de datos tienen algunas protecciones para que eso no ocurra (pero eso no quiere decir que no pueda ocurrir).

Antiguamente a la hora de apagar el ordenador (y cuando digo antiguamente, digo MUY antiguamente), existían comandos que permitían "aparcar" el disco duro, o lo que es lo mismo, "apagarlo" y dejarlo preparado para un futuro uso. Actualmente eso es impensable. Desconozco como opera éste componente al apagar el ordenador, pero diría que más bien lo que hace es recolocar los cabezales nada más encenderlo. De ésta forma, aun si se va la luz y los cabezales se encuentran en una posición "rara", volverán a una posición de origen al iniciar el ordenador.

En temas de software, hay gente que cree que tras un antivirus va a estar protegido. Y gente que cree que tener 50 antivirus le va a dar más protección. Tener 50 antivirus es peor que tener uno

solo. No sólo no otorga ningún tipo de protección (ni siquiera uno), sino que ademas va a ralentizar el ordenador de forma descomunal. Para más información, lo mejor es leer [éste artículo](#).

No conviene tener tropecientasmíl barras de herramientas en tu navegador. No sólo dificultan tu navegación, sino que también te espían, recopilando datos de navegación. Muchas aplicaciones gratuitas incluyen en su instalador la opción de instalar barras de herramientas o chorraditas similares. Si el instalador de un programa te pide instalar una barra de herramientas (y me da igual que sea la de yahoo), o aceptar los términos y condiciones de tal programa que no quieras, es de sabios decirle "no, gracias". El programa principal seguirá instalándose, pero si es legal, lo hará sin dichas adiciones.

Vamos con el correo electrónico. Las famosas cadenitas de correo electrónico. "Manda éste mensaje a tropecientasmíl personas y ocurrirá un milagro". Al hacerlo, estás engrosando una lista de direcciones de correo. Direcciones que, al final, llegarán a manos de una persona muy oscura, que las usará para mandarte publicidad, entre otras cosas peores. Para hacer algo así, mejor usa el campo CCO (Con Copia Original).

En Facebook, ten cuidado con la gente que dice "da un like aquí para ver tal video", "da un like aquí y si llego a tantos likes, pasará ésto o haré lo otro", o "Pulsa L y verás que pasa". Estarás engrosando otra lista de SPAM.

Conviene tener mucho cuidado con cómo usas las redes sociales. Las empresas ven una plataforma excepcional para hacer publicidad en ellas. "Da like y disfruta de una promoción". "Da like, y participa en un sorteo". "Da like, y serás dios...". Etc. Si ves que al intentar permitir el paso a una aplicación, ésta tiene acceso a tu información personal, y crees que tienes información delicada en esos campos, lo mejor es denegarla. O mejor aún, no des información veraz. Inventate todos tus datos. Y recuerda, los "Likes" y "Follows" se deben ganar, no dar.



Hay muchas más cosas y consejos de los que hablar, pero por ahora éste artículo ha dado todo lo de sí que ha podido. ¿Tienes alguna duda? ¿Quieres más consejos? Hazmelo saber a través de los comentarios, y no te olvides de dar "Like" en nuestra [página de facebook](#). Si lo haces, Rajoy nos sacará de la crisis, los cerdos volarán y tú entraráς en el sorteo ficticio ante notario imaginario de 1 billón de petrodólares. O no, pero igual te enteras de algo interesante.

## **Edito: Nuevos "mitos"**

Vamos a desmentir algunas cosillas que he oído recientemente en relación al software.

Primero, **que el software gratuito sea mejor que el de pago. Y vice-versa.**

Que un programa sea mejor que otro no depende de su precio, sino del programa en sí. Habrá programas freeware que sean mejores que sus alternativas de pago, y programas de pago que sean mejores que sus alternativas gratuitas. Personalmente, a mí Linux de siempre me ha dado más problemas que Windows.



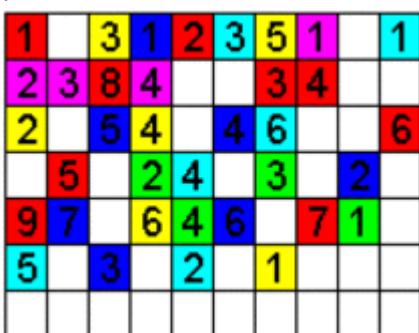
Precisamente de **Linux trata nuestro siguiente mito: Linux es más seguro que Windows.**

Linux es muy seguro, de eso no cabe duda y por ello (y sobre todo, porque es gratuito) lo usan en muchas empresas. Pero eso no lo convierte en una fortaleza inexpugnable, como mucha gente nos quiere hacer creer.

Linux tiene virus. Y cualquiera puede programar un virus para Linux. ¡Para algo es open source! Incluso yo mismo creé una modificación del shell de gnome que me permitía obtener permisos de administrador usando una contraseña incorrecta. El problema es que hoy en día los virus son comerciales. No son realmente virus, son adware y spyware, programas que recopilan tus datos (suelen venir sobre todo en forma de barra de herramientas, e incluso en navegadores completos como Chrome). Es por ello que las empresas que crean éstos programas intentarán acaparar el mayor número de “víctimas” posibles: Windows.

Ahora, ¿Podrían éstas empresas hacer lo mismo con Linux? La respuesta es, **sí**. Definitivamente. Sobre antivirus, ya escribí un artículo en su día. Sobre el típico mito, de que **los antivirus te protegen de todo**. No te protegen, intentan arreglar el daño cuando ya te lo han hecho. Recomiendo leer el artículo:

<http://elektronblog.wordpress.com/2011/10/19/un-antivirus-es-el-mejor-sistema-de-seguridad-que-existe/>



El siguiente mito trata sobre el desfragmentador de Windows. Hay gente que afirma que **desfragmentar no sirve para nada, y que ademas elimina archivos**.

Eso es porque hay gente que no entiende qué es y para qué sirve desfragmentar. Para explicarlo tendría que explicar lo que son las tablas de archivos y como se organizan los datos en el disco duro, pero eso es demasiado así que os recomiendo que lo mireis en la [wikipedia](#). Basicamente, al desfragmentar lo que estamos haciendo es **organizar archivos**. Ni más, ni menos. Si los datos están muy fragmentados, el sistema tiene que andar “saltando” de un lado para otro en busca de esos datos, mientras que si está todo organizado (desfragmentado), el sistema sabe donde encontrarlo. Ésto SIEMPRE va a suponer una mejora de rendimiento, sea o no notable. Lo mismo se aplica al desfragmentar el registro, o los archivos de Steam.

Recientemente he oido una auténtica chorrada. **Actualizar no sirve para nada**.

Queridos amigos, al próximo que le oiga decir ésto, voy a su casa con un trabuco y abro fuego contra él. Los desarrolladores de software, y hardware (por los drivers), ofrecen actualizaciones de su software. Éstas actualizaciones pueden contener: correcciones de errores (en su mayoría), optimizaciones (para que funcionen de forma más eficiente), y nuevas características (para aumentar su capacidad).

¿Es necesario que diga, que actualizar no solo es necesario, sino que ademas es estrictamente recomendable? No, no hace falta.

Una última cosa antes de acabar. Un mito que también estoy oyendo mucho. **Formatear no sirve para nada**. Lo oigo mucho, demasiado.

Primero de todo, ¿Qué es eso de formatear? Una vez más tendría que explicar en detalle como funciona el disco duro. Básicamente, consiste en dar un formato que va a servir para almacenar información. Ejemplos: el clásico FAT (FAT16 y FAT 32) que se usaba en los tiempos de MS-DOS y Windows 9x, luego el NTFS (*New Technology File System*) de Microsoft, y los EXT (ext2, ext3, ext4...) de Linux. Habrá mas formatos, seguramente, pero los principales son éstos.

Cuando le damos formato a un disco, o a una partición del mismo, lo que hacemos es indicarle al disco duro cómo vamos a almacenar los datos en él. Por ejemplo, en la FAT, va por una tabla de archivos, mientras que NTFS utiliza **metadatos**.

Evidentemente, en el proceso perdemos todos los datos que apuntan a nuestros ficheros. Algo parecido a cuando borramos un archivo por medio de la papelera de reciclaje. Por ello, usamos el

formateo también para “eliminar” la información.

No obstante, esa información sigue en el disco (oculta, pero sigue). Es por ello que si queremos borrar definitivamente esa información, requeriríamos de un **formateo de bajo nivel**. En él, llenamos todo el disco duro con un valor, eliminando así todo lo que hubiera antes.

Después de esto, instalamos el sistema operativo sobre la partición activa de nuestro disco duro, la cual debe tener el formato correcto.

¿Qué conseguimos con ésto? Una instalación fresca del sistema operativo.

Lo primero que vas a notar, sobre todo si usas mucho el ordenador y llevas mucho tiempo sin formatear, es que todo va mucho más fluido y más rápido. ¿Por qué? Porque un sistema recién instalado no tiene procesos ni archivos que lo carguen, con lo cual trabaja con toda su eficiencia. Más o menos, así es.

Con lo cual decimos que **formatear si sirve**. Los expertos lo recomiendan, de hecho. Al menos, en los sistemas más antiguos.

Una cosa más, acerca del formateo. He dicho que se elimina la información de la tabla de archivos al formatear. Pero eso sólo ocurre en el caso de realizar lo que se denomina **formato rápido**. Es decir, borra la información del formato, y coloca uno nuevo, dejando los archivos “ocultos” o “perdidos” hasta que se escribe encima. Existe otro proceso de formateo, denominado **formato a bajo nivel**. Éste tipo de formateo, lo que hace primero es poner todos los bits de la unidad a un mismo valor, eliminando toda la información, incluso la que está fuera de la tabla, de modo que esa información queda **irrecuperable**. Y luego escribe los datos del formato seleccionado. Éste proceso es mucho más lento y sólo se usa cuando se quieren borrar datos sensibles y/o cuando vas a vender la unidad en segunda mano, para que no se puedan recuperar.

La gente piensa que instalando programas como CCleaner ya lo tienen todo hecho. CCleaner sólo hace algunas cosas que podemos hacer nosotros mismo a mano, como borrar archivos temporales, historial, archivos sin usar, e incluso puede que claves del registro en desuso. Pero nada más.

Pegado de <<http://nинjiserver/WORDPRESS/?p=513>>

## Que bonita es la informatica

sábado, 12 de julio de 2014  
1:47

[February 7, 2012](#) [Ninjihaku](#) [1 Comment](#) [Edit](#)



Es tán relajante a veces...

Os contare una historia. Hace unos días le dije a un colega mío, que iba a empezar a crear un programa de chat con encriptación. No penseis nada raro, lo hago por placer. En ésto que me pongo a diseñarlo, decidí crearlo en C++. Yo ya sabeis que vengo de C#, así que de punteros y demás, ni papa. Estuve leyendo un libro sobre el tema (C++, por Fco Javier Cerballos, editorial Ra-Ma. Os lo recomiendo), y al poco tiempo me puse a la faena, no sin antes buscar en google información sobre el uso de sockets.

Empiezo el proyecto en Dev-C++. Las librerías no me funcionan por extrañas y sobrenaturales razones (intente linkearlas). Así que me pasé a Code::Blocks que dicen que es mejor (Es mejor, ya os lo digo yo). Total, que empiezo ahí mi proyecto, y funciona de mil maravillas.

Creo un programa servidor, lo pruebo y funciona las mil maravillas. Comienzo con el programa cliente y...

Error crítico. Al parecer por un maldito puntero no válido que producía una violación de acceso en la memoria (código de error 0xC0000005, me lo sé de memoria de tanto verlo). Yo evidentemente, ni papa de punteros. Empecé a cambiar cosas de aquí, cosas de allá... nada. No funcionaba absolutamente nada.

Tuve que preguntar a los gurús de "elhacker.net". Cabe destacar que si bien hay gurús de la programación, también los hay de la política. Porque algunos tienen el arte de hablar sin decir nada.

El tema es

([http://foro.elhacker.net/programacion\\_cc/error\\_de\\_violacion\\_de\\_acceso\\_en\\_mi\\_programa-t352764.0.html](http://foro.elhacker.net/programacion_cc/error_de_violacion_de_acceso_en_mi_programa-t352764.0.html)), aunque ya está solucionado (no gracias a quien haya cerrado el tema, por cierto, aunque muchas gracias a los que SÍ han intentado ayudarme de verdad). Total, que empezamos con lo típico. Que si éste puntero no es válido, que si tutoriales por aquí y por allá... nada.

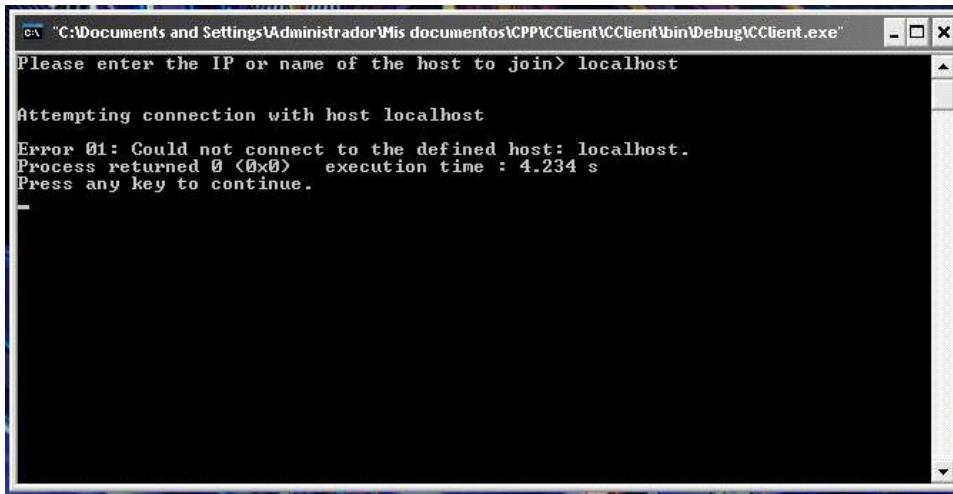
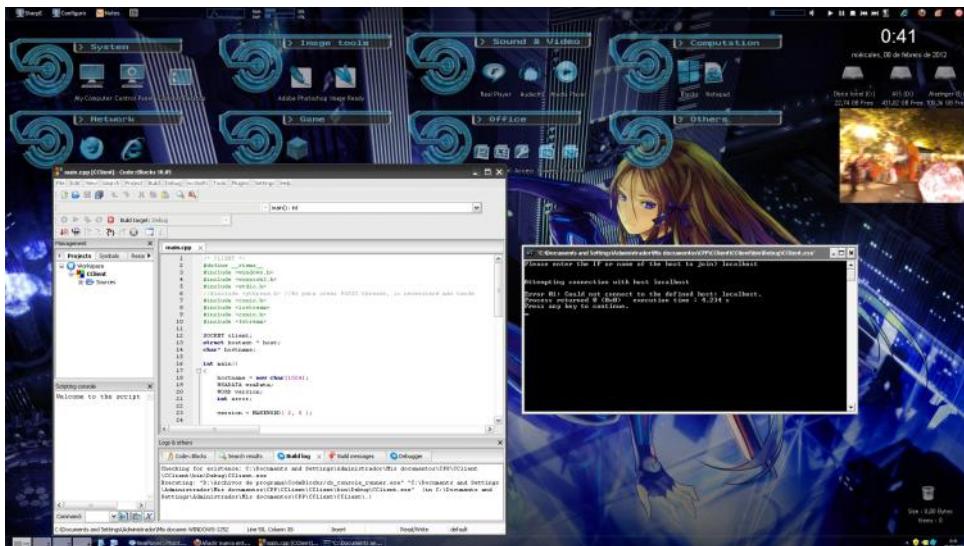
El código compilaba bien, pero por alguna extraña y bizarra razón no funcionaba al ejecutarlo. El dichoso error 0xC0000005 aparecía siempre, modificara lo que modificara.

"Ya estas mezclando códigos de diferentes fuentes y haciendo copy/paste sin entender..." mentira. Empecé de 0 en el proyecto usando los tutoriales para comprobar que ahora el error me lo daba en una función que antes me funcionaba. Nada parecía tener sentido ni lógica, y no parecía que internet fuese a ser de ayuda.

"El ultimo código que habias dejado debería funcionar". ¿En serio? Hice la prueba, instalé Visual C++ 6.0. Configuré un nuevo proyecto, enlazé las librerías de winsock... error. En la misma línea. Algo no iba bien en mi ordenador, definitivamente. Sabía que tenía que funcionar, pero... ¿Por qué no lo hacía, y qué podía hacer para solventarlo? Eso era lo que yo quería saber. Así que, finalmente, tuve una idea. Formatear.

Si, formatear, empezar de cero. Empecé a sospechar que el problema podría ser de Windows y no mío, ya que el programa era correcto, y el error que me mostraba no tenía sentido. ¿Alguna configuración? ¿Algún programa interfiriendo? ¿El propio SO lleno de mierda? Quién sabe.

La cuestión es, que formateé el ordenador (es la forma vulgar de decir que has formateado la partición primaria del disco duro y reinstalado el sistema operativo anterior, lo sé). Instalé lo mínimo necesario, probé el mismo programa que había posteado...



Una imagen vale más que mil palabras. Funcionó. Despues de formatear, lo que fuese que afectara a mi programa desapareció, y el programa retornó 0x0, en lugar del dichoso 0xC0000005.

Evidentemente, sin ejecutar el programa servidor, no se puede estabilizar la conexión, pero lo intenta, que es lo que me interesaba que hiciera.

Y así acaba nuestra historia. Una historia de una batalla que duró días, en donde la perseverancia vale oro, y en donde la paz se obtuvo mediante la destrucción total de miles de bytes. Muchas gracias, de nuevo, a todos los que en verdad intentaron ayudarme. Puede que ninguno de vosotros me diera la solución al problema, pero como para saber lo que había que hacer en éste caso. Y a los demás, deciros que cuando alguien necesita ayuda, si no vas a decir algo que sirva de ayuda cuando alguien os pide ayuda, es mejor no decir nada. Si leéis ésto, no os lo tomeis a mal lo que digo. Es sólo que me ofendió un poco que aun viendo que necesitaba ayuda, me cerrarais el tema sin más. Y al resto de lectores, si estais metidos en ésto de la programación y teneis un problema similar... ya sabeis lo que podeis probar en última instancia. Un último recurso.

PD: ¿Que os parece mi nuevo escritorio? ¿Mola o no?

Pegado de <<http://niniiserver/WORDPRESS/?p=435>>

**Autor** Autor **Tema: Error de violación de acceso en mi programa (Leido 2,302 veces)**

**Chains** **Error de violación de acceso en mi programa**  
« en: 5 Febrero 2012, 19:48 »

Desconectado Mensajes: 9

Buenos días. Estoy tratando de crear un pequeño programa cliente-servidor usando C++. He de admitir que soy un poco nuevo en esto de C++ y sobre todo en el uso de sockets (soy programador de C#).

Bien, la cuestión es que mi programa produce un error de violación de acceso (0xC0000005) al ejecutar una de sus líneas. Éste es el programa en cuestión (Es el cliente):

```
Código:
SOCKET client;
struct hostent * host;
char* hostname;

int main()
{
    hostname = new char[1024];
    client = socket( AF_INET, SOCK_STREAM, 0 );
    printf("Please enter the IP or name of the host to join> ");
    fflush( stdout );
    scanf("%s", hostname);
    printf("\n\nAttempting connection with host %s", hostname);
    host = gethostbyname( hostname );
    struct sockaddr_in sin;

    memset( &sin, 0, sizeof sin );

    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = ((struct in_addr *)(host->h_addr))->s_addr;
    sin.sin_port = htons( 21 );
    if ( connect( client, (PSOCKADDR)&sin, sizeof sin ) == SOCKET_ERROR )
    {
        printf("Error 01: Could not connect to the defined host: %s.", hostname);
        return FALSE;
    }
    closesocket( client );
    return 0;
}
```

El error en cuestión me lo produce al ejecutar la siguiente línea:

```
Código:
sin.sin_addr.s_addr = ((struct in_addr *)(host->h_addr))->s_addr;
```

Recorte de pantalla realizado: 12/07/2014 1:48

Y no sé qué hacer para solucionarlo, pues apenas hay documentación en internet acerca del uso de sockets en C++. Les agradecería que me guiaran un poco en mi tarea. Muchas gracias.

**leon** **Re: Error de violación de acceso en mi programa**  
« Respuesta #1 en: 6 Febrero 2012, 00:02 »

Desconectado Mensajes: 99

No estoy muy metido en los sockets pero me da que el fallo es más bien por la clase/estructura o por el puntero.

El error de violación de acceso se da cuando intentas acceder a una zona de memoria errónea. Si muestras las estructuras quizás se vea más claro pero posiblemente sea o porque estás intentando acceder a una zona de la memoria (de una clase) declarada como privada sin ningún método accessor, o porque estás asignándole a un puntero el valor NULL.

Comprueba a ver y cometas. Un saludo.

**En línea**

**Cambio de configuración del sistema**

Windows ha detectado que ha movido su mouse.  
Para que la nueva configuración tenga efecto, deberá reiniciar su PC.  
¿Desea reiniciar su PC ahora?

**En línea**

Recorte de pantalla realizado: 12/07/2014 1:48

**Chains**

Desconectado

Mensajes: 9

**Re: Error de violación de acceso en mi programa**  
« Respuesta #2 en: 6 Febrero 2012, 12:58 »

Pues poca cosa he podido sacar. He estado comprobando las estructuras, pero no veo nada raro.

La estructura de hostent es:

```
Código:
struct hostent {
    char *h_name;
    char **h_aliases;
    short h_addrtype;
    short h_length;
    char **h_addr_list;
#define h_addr h_addr_list[0]
};

Y la del in_addr:
Código:
struct in_addr {
    union {
        struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
        struct { u_short s_w1,s_w2; } S_un_w;
        u_long S_un;
    } S_un;
```

Ambas pertenecen a la cabecera winsock2.h

Empiezo a sospechar que el error pudiera venir de antes, concretamente de ésta línea:

```
Código:
memset(&sin, 0, sizeof sin );
```

Como aloja sin en la memoria, tal vez por algún motivo el puntero se salga de la región de sin y apunte a otro lado que no debe, y de ahí el error. No estoy muy seguro.

« Última modificación: 6 Febrero 2012, 13:00 por Chains »

En línea

Recorte de pantalla realizado: 12/07/2014 1:48

**Eternal Idol**  
Moderador

Desconectado

Mensajes: 5.016

**Re: Error de violación de acceso en mi programa**  
« Respuesta #3 en: 6 Febrero 2012, 14:44 »

La linea que decis solo pone ceros.

¿Que devuelve gethostbyname? Si devuelve 0 entonces es claro el motivo del fallo.



La mano invisible del mercado me roba la billeteira

La economía nunca ha sido libre: o la controla el Estado en beneficio del Pueblo o lo hacen los grandes consorcios en perjuicio de éste.  
Juan Domingo Perón

**Chains**

Desconectado

Mensajes: 9

**Re: Error de violación de acceso en mi programa**  
« Respuesta #4 en: 6 Febrero 2012, 20:21 »

gethostbyname es una estructura tipo hostent . Pero he descubierto que si intento obtener el valor de h\_addr de la variable host (la que contiene el valor de retorno de gethostbyname()), me da el error de violación de acceso, así que el error o está en h\_addr\_list, o en h\_addr, o peor aún en la estructura host.

Me da error igual tanto si intento obtenerlo mediante h\_addr, como por h\_addr\_list[0]. También he intentado definir h\_addr como una variable independiente en la estructura en lugar de asignarle una región de h\_addr\_list, pero tampoco ha funcionado.

« Última modificación: 6 Febrero 2012, 20:24 por Chains »

En línea

Recorte de pantalla realizado: 12/07/2014 1:49

**Eternal Idol**  
Moderador

 Desconectado

Mensajes: 5.016



 **Re: Error de violación de acceso en mi programa**  
« Respuesta #5 en: 6 Febrero 2012, 20:31 »

gethostbyname es una función que devuelve o un puntero a una estructura del tipo hostent o un puntero nulo, h\_addr es una macro que está definida al primer elemento de h\_addr\_list, así que cambiar uno por otro es no cambiar nada. Te repito la pregunta: ¿Qué devuelve gethostbyname? Dudo que sea un puntero válido ...

 En línea



La mano invisible del mercado me roba la billetera

La economía nunca ha sido libre: o la controla el Estado en beneficio del Pueblo o lo hacen los grandes consorcios en perjuicio de éste.

Juan Domingo Perón

**Chains**

 Desconectado

Mensajes: 9

 **Re: Error de violación de acceso en mi programa**  
« Respuesta #6 en: 6 Febrero 2012, 20:47 »

Ah, de acuerdo, no sabía a lo que te referías con el valor de gethostbyname. Lo he estado comprobando y devuelve 0.

 En línea

Recorte de pantalla realizado: 12/07/2014 1:49

**Eternal Idol**  
Moderador

 Desconectado

Mensajes: 5.016



 **Re: Error de violación de acceso en mi programa**  
« Respuesta #7 en: 6 Febrero 2012, 21:07 »

Ese es el problema, no puedes acceder a host: es un puntero nulo.

 En línea



La mano invisible del mercado me roba la billetera

La economía nunca ha sido libre: o la controla el Estado en beneficio del Pueblo o lo hacen los grandes consorcios en perjuicio de éste.

Juan Domingo Perón

**Chains**

 Desconectado

Mensajes: 9

 **Re: Error de violación de acceso en mi programa**  
« Respuesta #8 en: 6 Febrero 2012, 21:25 »

Entiendo. Ahora mi pregunta es, si no puedo acceder a host porque es un puntero no válido, ¿Qué tengo que hacer entonces para obtener la dirección del servidor a partir del nombre del host especificado, sin usar la estructura hostent?

Siento ser tan persistente, pero como ya he dicho, aunque tengo nociones de lo que estoy haciendo, no estoy muy experimentado en C++, y no estoy demasiado acostumbrado a todos estos temas de punteros y sockets. Intento sacar la mayor información que puedo de internet, pero no encuentro la suficiente. El código de mi programa, es sacado también de una página, y ya he tenido que resolver un sinfín de errores. No lo entiendo...

 En línea

Recorte de pantalla realizado: 12/07/2014 1:49

Eternal Idol  
Moderador

Desconectado

Mensajes: 5.016



Re: Error de violación de acceso en mi programa  
«Respuesta #9 en: 6 Febrero 2012, 22:07»

¿Qué estas escribiendo en consola? ¿Estas trabajando bajo Windows? Si es así tenes que inicializar Winsock.



La mano invisible del mercado me roba la billeteira



La economía nunca ha sido libre: o la controla el Estado en beneficio del Pueblo o lo hacen los grandes consorcios en perjuicio de éste.

Juan Domingo Perón

En línea

Chains

Desconectado

Mensajes: 9

Re: Error de violación de acceso en mi programa  
«Respuesta #10 en: 6 Febrero 2012, 22:57»

Si, estoy realizando un programa de consola en Windows. Uso el entorno de desarrollo Code::Blocks, y de compilador el GNU GCC Compiler, que viene por defecto con este entorno de desarrollo. El código, por cierto, lo saqué de esta página: <http://www.nullterminator.net/winsock.html>, y sólo le realicé algunas modificaciones al código del servidor, porque me daba algunos errores y también para ajustarlo a mis necesidades.



He probado a inicializar winsock en el programa cliente usando el siguiente código:

Código

```
1. WSADATA wsaData;
2. WORD version;
3. int error;
4.
5. version = MAKEWORD( 2, 0 );
6.
7. error = WSAStartup( version, &wsaData );
8.
9. /* check for error */
10. if ( error != 0 )
11. {
12. /* error occurred */
13. return FALSE;
14. }
15.
16. /* check for correct version */
17. if ( LOBYTE( wsaData.wVersion ) != 2 || 
18. HIBYTE( wsaData.wVersion ) != 0 )
19. {
20. /* incorrect WinSock version */
21. printf("\n\nError 01: Incorrect WinSock Version. LOBYTE: %n . HIBYTE: %n .", (INT*)LOBYTE( wsaD
22. WSACleanup();
23. return FALSE;
24. }
25.
26. client = socket( AF_INET, SOCK_STREAM, 0 );
27.
28. /* Resto del programa */
```

También he añadido las correspondientes líneas al final del código para cerrar el socket, pero sigo recibiendo el mismo error al llegar a la misma línea.

A continuación dejo el código entero del cliente con las modificaciones realizadas:

Recorte de pantalla realizado: 12/07/2014 1:50

Código

```
1. /* CLIENT */
2. #define _RTEMS_
3. #include <windows.h>
4. #include <winsock2.h>
5. #include <stdio.h>
6. // #include <pthread.h> // Es para crear POSIX threads, lo necesitaré más tarde
7. #include <conio.h>
8. #include <iostream>
9. #include <conio.h>
10. #include <fstream>
11.
12. SOCKET client;
13. struct hostent * host;
14. char* hostname;
15.
16. int main()
17. {
18.     hostname = new char[1024];
19.     WSADATA wsaData;
20.     WORD version;
21.     int error;
22.
23.     version = MAKEWORD( 2, 0 );
24.
25.     error = WSAStartup( version, &wsaData );
26.
27.     /* check for error */
28.     if ( error != 0 )
29.     {
30.         /* error occurred */
31.         return FALSE;
32.     }
33.
34.     /* check for correct version */
35.     if ( LOBYTE( wsaData.wVersion ) != 2 || 
36.         HIBYTE( wsaData.wVersion ) != 0 )
37.     {
38.         /* incorrect WinSock version */
39.         printf("\n\nError 01: Incorrect WinSock Version. LOBYTE: %d . HIBYTE: %d .", (INT*)LOBYTE( wsaD
40.         WSACleanup();
41.         return FALSE;
42.     }
43.
44.     client = socket( AF_INET, SOCK_STREAM, 0 );
45.
46.     printf("Please enter the IP or name of the host to join> ");
47.     fflush( stdout );
48.     ---->>>
```

Recorte de pantalla realizado: 12/07/2014 1:50

```
47.     fflush( stdout );
48.     scanf("%s", hostname);
49.     printf("\n\nAttempting connection with host %s", hostname);
50.     host = gethostbyname( hostname );
51.     //printf("\nsp", host); //DEBUG
52.     struct sockaddr_in sin;
53.
54.     memset( &sin, 0, sizeof sin );
55.     //printf("\nSuccess0"); //DEBUG
56.     sin.sin_family = AF_INET;
57.     //printf("\nSuccess1"); //DEBUG
58.     sin.sin_addr.s_addr = ((struct in_addr *) (host->h_addr))->s_addr;
59.     //printf("\nSuccess2"); //DEBUG, el programa se traba aqui
60.     sin.sin_port = htons( 21 );
61.
62.     if ( connect( client, (PSOCKADDR)&sin, sizeof sin ) == SOCKET_ERROR )
63.     {
64.         printf("\n\nError 01: Could not connect to the defined host: %s.", hostname);
65.         return FALSE;
66.     }
67.
68.     WSACleanup();
69.     closesocket( client );
70.     return 0;
71. }
72.
```

En línea

Recorte de pantalla realizado: 12/07/2014 1:51

Desconectado

Mensajes: 3.182

I'Love...!!.

Te dejo un trozo de mi clase CSocketClient, Posiblemente te sirva para solventar ese error (lineas en color resaltado):  
Ademas te dejo dos PDF para saber sobre WinSock (Tuto por MazarD) y la creación de Hilos (estandares POSIX) en este enlace (<http://infrangelux.systes.net/filex/?dir=/BlackZeroX/Programacion/papers> ).

Código

```
1. bool CSockClient::connect(const char* szIP, unsigned short int iPort)
2. {
3.     hostent* lpHosten = NULL;
4.
5.     if (this->iState != SCKCLOSED )
6.     (
7.         this->setError();
8.         return false;
9.     )
10.
11.    if (szIP == NULL || iPort == 0)
12.        return false;
13.
14.    this->iRemotePort = iPort;
15.    this->sRemoteHost = szIP;
16.
17.    memset(&this->udtSockAddrIn, 0, sizeof(sockaddr_in));
18.
19.    this->udtSockAddrIn.sin_family = AF_INET;
20.    this->udtSockAddrIn.sin_port = htons(iPort);
21.    this->iState = SCKOPEN;
22.
23.    if (this->udtSockAddrIn.sin_port == INVALID_SOCKET)
24.    (
25.        this->setError();
26.        return false;
27.    )
28.
29.    lpHosten = ::gethostbyname(this->sRemoteHost.c_str());
30.
31.    if (lpHosten == NULL)
32.    (
33.        this->setError();
34.        return false;
35.    )
36.
37.
```

Recorte de pantalla realizado: 12/07/2014 1:51

```
37.    this->udtSockAddrIn.sin_addr.s_addr = *((unsigned long*)lpHosten->h_addr_list[0]);
38.    // memcpy(&iuiRet, hHost->h_addr_list[0], hHost->h_length);
39.
40.    this->sLocalHostIP.assign(inet_ntoa(this->udtSockAddrIn.sin_addr));
41.
42.    if (this->udtSockAddrIn.sin_addr.s_addr == INADDR_NONE)
43.    (
44.        this->setError();
45.        return false;
46.    )
47.
48.    this->mySock = ::socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
49.
50.    if (this->mySock == INVALID_SOCKET)
51.    (
52.        this->setError();
53.        return false;
54.    )
55.
56.    if (!this->getSocketOpt())
57.        return false;
58.
59.    //Creamos el hilo para resivir los datos de este Socket.
60.    if (pthread_create(&this->threadConnect, NULL, myCallConnect, this))
61.    (
62.        if (this->pEventError != NULL) //Evento
63.            this->pEventError(this);
64.
65.        if (this->mySock != INVALID_SOCKET )
66.            closesocket(this->mySock);
67.
68.        this->mySock = INVALID_SOCKET;
69.
70.        this->setError();
71.
72.        return false;
73.    }
74.    pthread_detach(this->threadConnect); // No guardar Returns del hilo...
75.
76.
77. }
```

Recorte de pantalla realizado: 12/07/2014 1:51

**CScript (Actualizado 26/06/2013).**

FileX <-- Re-modelando...  
**Web Principal-->[ Blog(VB6/C/C++) | Host File | Scan Port ]**

The Dark Shadow is my passion.  
 El infierno es mi Hogar, mi novia es Lilit y el metal mi relig

Recorte de pantalla realizado: 12/07/2014 1:51

**Chains** Desconectado

Mensajes: 9



**Re: Error de violación de acceso en mi programa**  
 « Respuesta #12 en: 7 Febrero 2012, 01:31 »

Muchísimas gracias por tu aporte, BlackZeroX. Sinceramente, me es de gran ayuda. Pero hay una cosa que no me explico. He logrado pasar de la linea erronea, pero ahora el error me lo produce en la función "connect".

## Código

```
1. connect(sock, (SOCKADDR *)&remoto, sizeof (remoto));
```

Pero lo mejor, es que he descubierto, que la línea que antes no me funcionaba, en éste programa me deja de funcionar si declaro "hostent \* direc" fuera de la función main.

Es decir, que si hago ésto

## Código

```
1. int main()
2. {
3.     hostname = new char[1024];
4.     hostent * direc = NULL;
5. /* ... */
6.     remoto.sin_addr = *((struct in_addr *)direc->h_addr);
7. /* ... */
8. }
```

El programa no me devuelve error en esa línea. Pero si hago ésto

## Código

```
1. hostent * direc = NULL;
2.
3. int main()
4. {
5.     hostname = new char[1024];
6. /* ... */
7.     remoto.sin_addr = *((struct in_addr *)direc->h_addr);
8. /* ... */
9. }
```

Al llegar a esa línea, da error de violación de acceso. Ésto es algo que no logro comprender, y que tiene algo que ver seguro con la gestión de la memoria, pero no sé si es error mío, del compilador, del ordenador... no logro entender nada y mi cabeza hecha chispas.

He probado a meter el programa en Dev-C++, pero me da error en todas las funciones de la librería winsock (aunque no me da error al importarla en la cabecera). No sé si es que tal vez debería añadir algún parametro especial en las opciones del enlazador. Lo mismo pasa si en Code::Blocks cambio el compilador al LCC, que me lo bajé para probar a ver si funcionaba bien así, sólo que en éste caso me da error en todas las líneas. Es un verdadero caos.

Recorte de pantalla realizado: 12/07/2014 1:51

**BlackZeroX (Astaroth)**  **Re: Error de violación de acceso en mi programa**  
Wiki  
« Respuesta #13 en: 7 Febrero 2012, 02:12 »

 Desconectado realiza un lindeo a esta libreria (CodeBlock):  
Mensajes: 3.182 MinGW\lib\libws2\_32.a  
I'Love....i.  
 

 En línea

**CScript (Actualizado 26/06/2013).**  
FileX <-- Re-modelando...  
**Web Principal-->** [ Blog(VB6/C/C++) | Host File | Scan Port ]  
The Dark Shadow is my passion.  
El infierno es mi Hogar, mi novia es Lilit y el metal mi relig

Páginas: [1] 2   

Ir a: => Programación C/C++ 

Recorte de pantalla realizado: 12/07/2014 1:52

 **Foro de elhacker.net**  
 **Programación**  
 **Programación C/C++** (Moderadores: Eternal Idol, Littlehorse)  
 **Error de violación de acceso en mi programa**

 8+1  122 O Usuarios y 1 Visitante están viendo este tema.

Páginas: 1 [2]   

**Autor** **Tema: Error de violación de acceso en mi programa (Leído 2,302 veces)**

**Eternal Idol**  **Re: Error de violación de acceso en mi programa**  
Moderador  
« Respuesta #14 en: 7 Febrero 2012, 06:33 »

 Desconectado Ya estas mezclando codigos de diferentes fuentes y haciendo copy/paste sin entender, es el problema de dejar codigo compulsivamente en el foro ante cada duda. El ultimo codigo que habias dejado deberia funcionar (comentando la inclusion de windows.h va perfecto con VC++), dale las vueltas necesarias (no te olvides de depurar) hasta lograrlo.  
Mensajes: 5.016

  
La mano invisible del mercado me robo la billetera

  
CRIS PASIÓN

 En línea

Ya estas mezclando codigos de diferentes fuentes y haciendo copy/paste sin entender, es el problema de dejar codigo compulsivamente en el foro ante cada duda. El ultimo codigo que habias dejado deberia funcionar (comentando la inclusion de windows.h va perfecto con VC++), dale las vueltas necesarias (no te olvides de depurar) hasta lograrlo.

La economía nunca ha sido libre: o la controla el Estado en beneficio del Pueblo o lo hacen los grandes consorcios en perjuicio de éste.  
Juan Domingo Perón

Páginas: 1 [2]   

Ir a: => Programación C/C++ 

Recorte de pantalla realizado: 12/07/2014 1:52

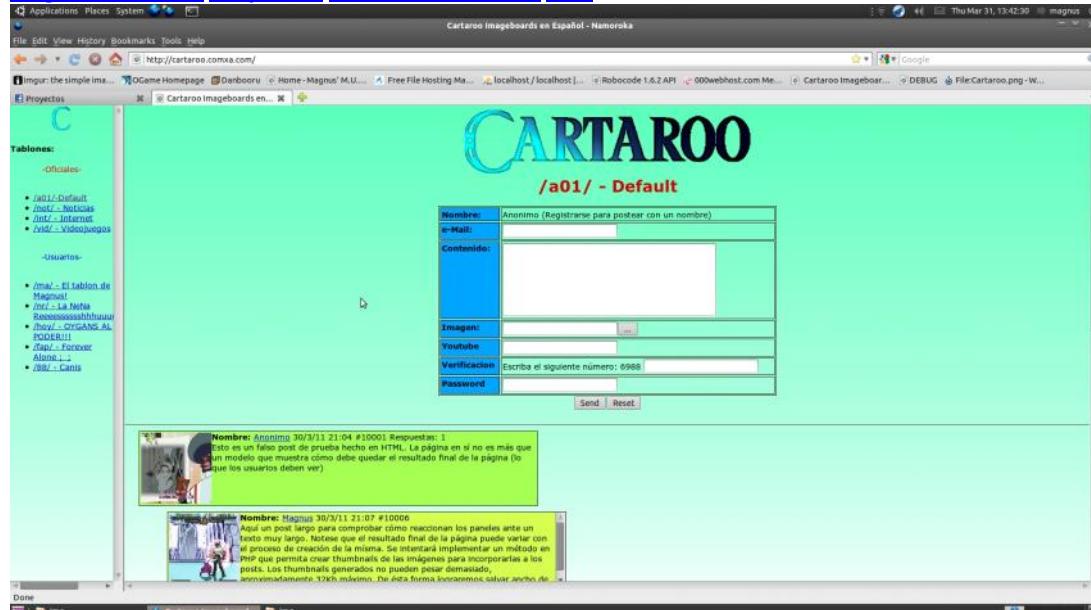
# Cartaroo

sábado, 12 de julio de 2014

1:57

## Proyecto Cartaroo

[August 19, 2011](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)



Seguro que conocéis una buena cantidad de foros de imágenes en la red. Desde el ya conocido 4chan, hasta cualquier otro que podáis conocer. Sin embargo, el software que usan todas estas páginas es de código cerrado, cuestan dinero, y no existen en la actualidad demasiados servicios que te ofrezcan crear tu propio imageboard sin limitartelo hasta un extremo. Existió uno una vez, que cerró porque abusaban de sus servidores. Ademas, estas páginas tienen numerosos problemas con la gestión y moderación, ya que la única forma de identificar a un usuario, es mediante su IP.

A raíz de estos problemas surge el proyecto [Cartaroo](#).

El proyecto Cartaroo surge para desbancar a todo el software de imageboards existentes en el mercado. Es gratuito, open source, está escrito íntegramente en PHP usando algunos plug-ins esenciales (que la inmensa mayoría de los servers modernos, incluidos los gratuitos, tienen), fácil de usar, seguro, es enormemente flexible y ademas le da la opción a sus usuarios no sólo de registrarse para postear con una identidad si así lo desean, sino ademas de crear, gestionar y administrar a sus usuarios registrados un tablero. Esto dá un amplio abanico de posibilidades, ya que cada uno puede tener su espacio en el foro, y permite a los moderadores y dueños de un tablero moderar con mayor facilidad, al requerir un registro para los usuarios y poder, por ejemplo, restringir el uso de un tablero a usuarios registrados.

El proyecto Cartaroo está de momento en un estado de desarrollo avanzado, y podeis verlo desde [aquí](#). La página está de prueba, y aunque se puede usar, es probable que la experiencia no sea idéntica a la de la versión final.

De momento el proyecto está parado, seguirá en desarrollo al termino del verano.

Pegado de <<http://nijiserver/WORDPRESS/?p=335>>

# Minecraft

sábado, 12 de julio de 2014

1:59

## Minecraft – Un juego para explorar, construir y sobrevivir

[October 11, 2010](#) [Ninjihaku](#) [1 Comment](#) [Edit](#)

Ser minero no debe ser nada divertido. Y si no me creéis, podeis preguntarselo a los pobres mineros chilenos que aún están atrapados a la espera de que los rescaten (Ánimo, ya falta poco. Podría haber sido peor). Pero si lo practicamos de forma virtual, puede ser muy divertido.

Basicamente, el autor de éste juego llamado Minecraft, define su juego de forma muy básica como “un juego de colocar bloques mientras huyes de esqueletos”. En realidad el concepto de éste juego es muchísimo más amplio. Apareceremos en un mundo vacío, generado de forma totalmente aleatoria, habitado por la vegetación y animales salvajes como cerdos, vacas y gallinas. No obstante, al caer la noche, hordas de no-muertos (zoombies, esqueletos, arañas, criaturas explosivas...) tomarán nuestras tierras e intentaran atacarnos. Y nuestro objetivo es sobrevivir a base de construir estructuras donde poder resguardarnos de dichas criaturas. Al ser no-muertos, no tendremos mucho problema ya que no son muy inteligentes (aunque son muy peligrosos si te pillan al aire libre).

Eso sí, si queremos construir estructuras, vamos a necesitar un porrón de recursos y herramientas. Es por ello que tendremos que recoger los recursos que nos ofrece la naturaleza para crear con ello, mediante diferentes procesos, bloques con los que construir estructuras al más puro estilo “Lego” (o mejor dicho, mediante voxels), nuevas herramientas, armas, armaduras, o incluso cocinar para restaurar nuestra salud.

Podemos construir granjas, sembrar, ir a pescar (aún no, pero es seguro que se va a poder dentro de poco), o incluso construir enormes dispositivos electrónicos sobre las rocas como, por ejemplo, un procesador (si, ví una vez un video de una persona que montó un procesador de 16 bits en Minecraft). Ciertos recursos los encontraremos en la superficie, pero muchos de ellos como la piedra, hierro, oro, las gemas, etc, se encuentran a mucha profundidad del suelo. Es por ello que debemos crear minas. Bien al aire libre como la de mi imagen, o bien por medio de galerías.

Y aquí es donde viene lo mejor. Mientras cavamos tuneles enormes, podemos encontrarnos cuevas llenas de recursos (o incluso tesoros) preciados, y también hordas de criaturas peligrosas a la espera de truncar nuestra partida. Y no solo eso, sino que cuanto más cavemos, mayor será la posibilidad de que nos encontremos lava por nuestro camino, muy peligrosa también pero que también podemos usar a nuestro favor para defender nuestras creaciones, o simplemente barrer un grupo amplio de enemigos, si tenemos un cubo donde transportarla. En definitiva, todo una aventura.

No temas por los recursos. ¿Crees que ya has agotado los recursos por tu zona? Puedes ir a cualquier otro sitio a por más. El mapa no tiene límites. Es decir, segun vas viajando por el mundo, se generará nuevo mapa para añadirlo a la zona donde te encuentres, de modo que nos encontramos ante un mundo absolutamente infinito y lleno de recursos. No tengas miedo en explorar.

El juego se encuentra en su versión “Alfa”, es decir, la anterior a la beta. Es por ello que también incluye todavía unos cuantos errores, e items que aún no tienen uso como, por ejemplo, la caña de pescar o los huevos que ponen de vez en cuando las gallinas, pero que algún día lo tendrán. También hay que pagar por jugarlo: 10€. Sin embargo, si pagamos por el juego, obtendremos acceso a la versión definitiva sin tener que pagar más (que el tío quiere ponerlo a 20€, y 10€ me parece más razonable). Ademas la versión final incluirá un modo multijugador, muy interesante, con posibilidad incluso de crear clanes. En otras palabras, tribus de pobladores construirán sus ciudades y lucharán por controlar el máximo territorio posible (o al menos espero que sea algo así porque estaría muy bien, sería Rol, estrategia y aventura en un juego indie, todo mezclado en uno).

Si no queremos pagar aún, podemos jugar de forma gratuita al modo creativo (o clásico). En este modo tenemos un inventario lleno de bloques de todo, ilimitados, tipo para construir lo que nos dé la gana sin límites, junto con otras personas a través de internet, o en solitario. Creatividad sin límites.

La página web del juego es [www.minecraft.net](http://www.minecraft.net). A continuación, os pongo algunos recursos visuales para que veáis por vosotros mismos como es el juego.



[youtube=http://www.youtube.com/watch?v=uar4H6RyTYQ]

[youtube=http://www.youtube.com/watch?v=LGkkyKZVzug]

[youtube=http://www.youtube.com/watch?v=s8nGJR556Gg]

[youtube=http://www.youtube.com/watch?v=2bxjQx\_2agE]

Pegado de <<http://ninjiserver/WORDPRESS/?p=311>>

# Reglas de internet

sábado, 12 de julio de 2014

2:09

[January 17, 2010](#) [Ninjihaku](#) [Leave a comment](#) [Edit](#)

Estas son las reglas básicas de cualquier internauta, usadas en cualquier chat de cualquier web:

– CHATS EN GENERAL –

- 1) Saluda siempre con el CAPS LOCK activado y diciendo insultos del tipo “joputas”, “chupapollas” y “noobs”.
- 2) Abusa siempre que sea posible de emoticonos, sonidos, colorines y demás. Cuanto más llamativo, más llamarás la atención, y más te odiaran.
- 3) Si alguna supuesta chica te pregunta si tienes cam, dila que si, pero antes que te enseñe las tetas y la cara (por si acaso)
- 4) Si alguien te insulta, ponlo a caldo. No digas nada a los administradores, tu solo insulta.
- 5) Procura usar siempre que sea posible 13375p34k y frases del argot 1337 como “n00b”, “ch3473r”, “suxx0r”, “stfu”, “gtfo”, etc...
- 6) Si alguien pregunta, pasa de el/ella
- 7) El 60% de las chicas en internet son tios, un 15% bots, y el 25% son de moralidad difusa. Está demostrado científicamente.
- 8) Si alguien pretende que te metas en una pagina o descargues algo, no lo hagas. Podría ser un hacker.
- 9) Si alguien hace SPAM, estás obligado a hacer SPAM, independientemente de lo que sea. Si no hay nadie haciendo SPAM, deberás TÚ de comenzar a hacer SPAM.
- 10) Si alguien te dice que te va a demandar durante el cumplimiento de la regla 9, dile que se valla a la mierda y que es un “n00b” llorica.
- 11) No seas cortés nunca, se rudo y haz que todo el mundo te odie.
- 12) Si un emo entra en un canal de chat, hablale de lo triste que es la vida, e incitale para que se suicide.
- 13) Pon siempre amenazas del tipo “dime donde vives si tienes cojones, que voy allí y te parto la cara”. No es que imponga mucho, pero da mucha rabia a los que están allí...

– MSN –

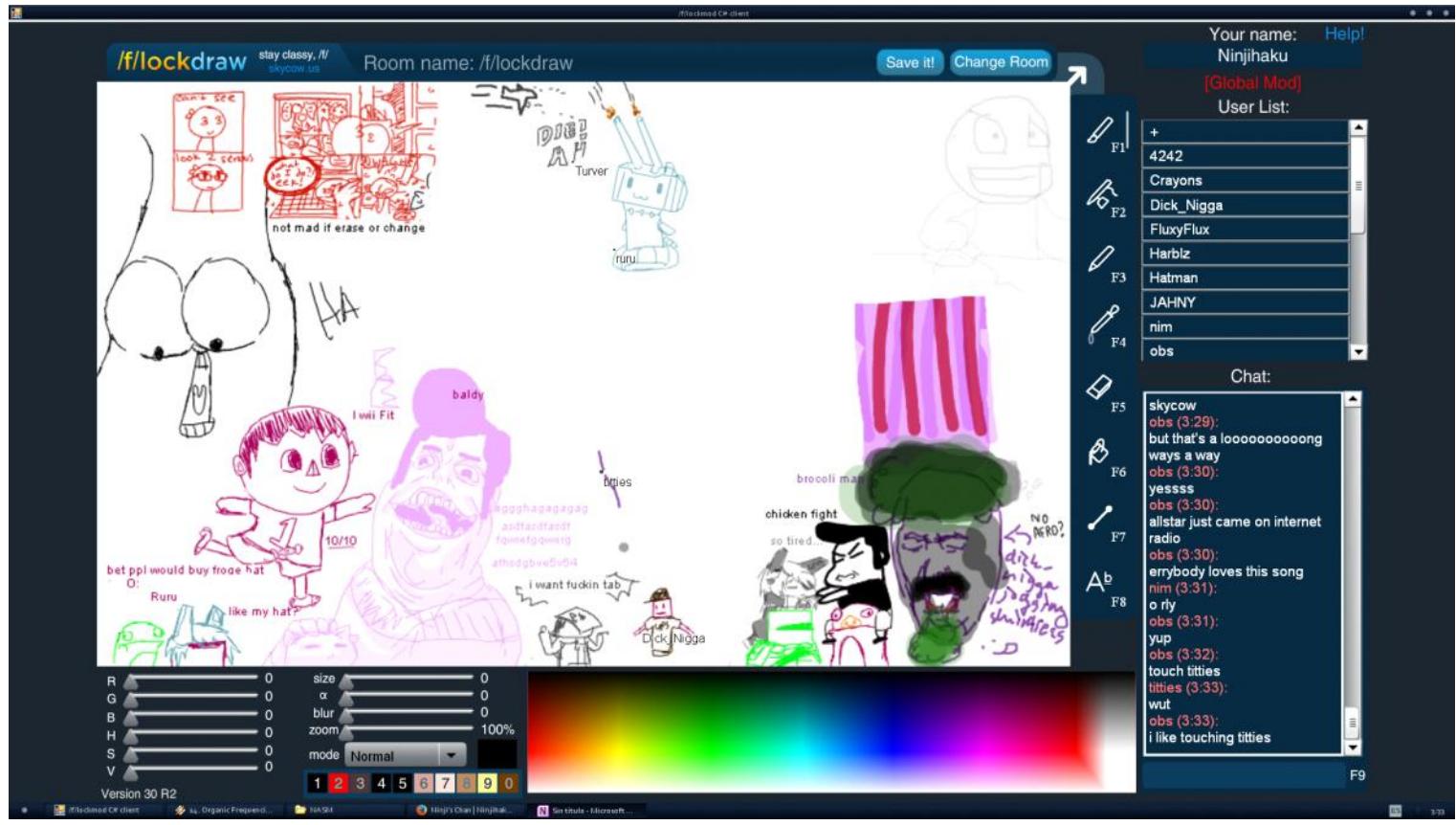
- 14) En el MSN, siempre inicia una conversación con un emoticono de “hola” y zumbale hasta que te conteste. Si no te contesta, inundale la bandeja de correo con SPAM hasta que se digne a hacerte caso (que se deje de pajas y conteste, hostias)
- 15) El 90% de los contactos en estado “No Disponible” están disponibles, el 10% restante lo están solo para la churri. A ese 10% deberás zumbarle hasta que se harte y te deniege.
- 16) No aceptes archivos de nadie, el 50% están infectados.
- 17) Llena siempre tus mensajes de emoticones hasta que estos queden ilegibles.
- 18) Si alguien te manda una cadena por correo, deberás seguirla, aunque no seas supersticioso.
- 19) Deberás ponerte en el nick el nombre de la churri y lo mucho que la quieres. No olvides que la churri es un verbo, por lo que debes poner “ers lo mejor k m ha pasado”.
- 20) Si no tienes churri, haz como que la tienes (inventate un nombre y sigue la regla 19).
- 21) Si alguien te insulta por el MSN, insultale, no lo denieges.

Pegado de <<http://nijiserver/WORDPRESS/?p=51>>

# A normal day on /f/lockdraw

sábado, 12 de julio de 2014

3:33

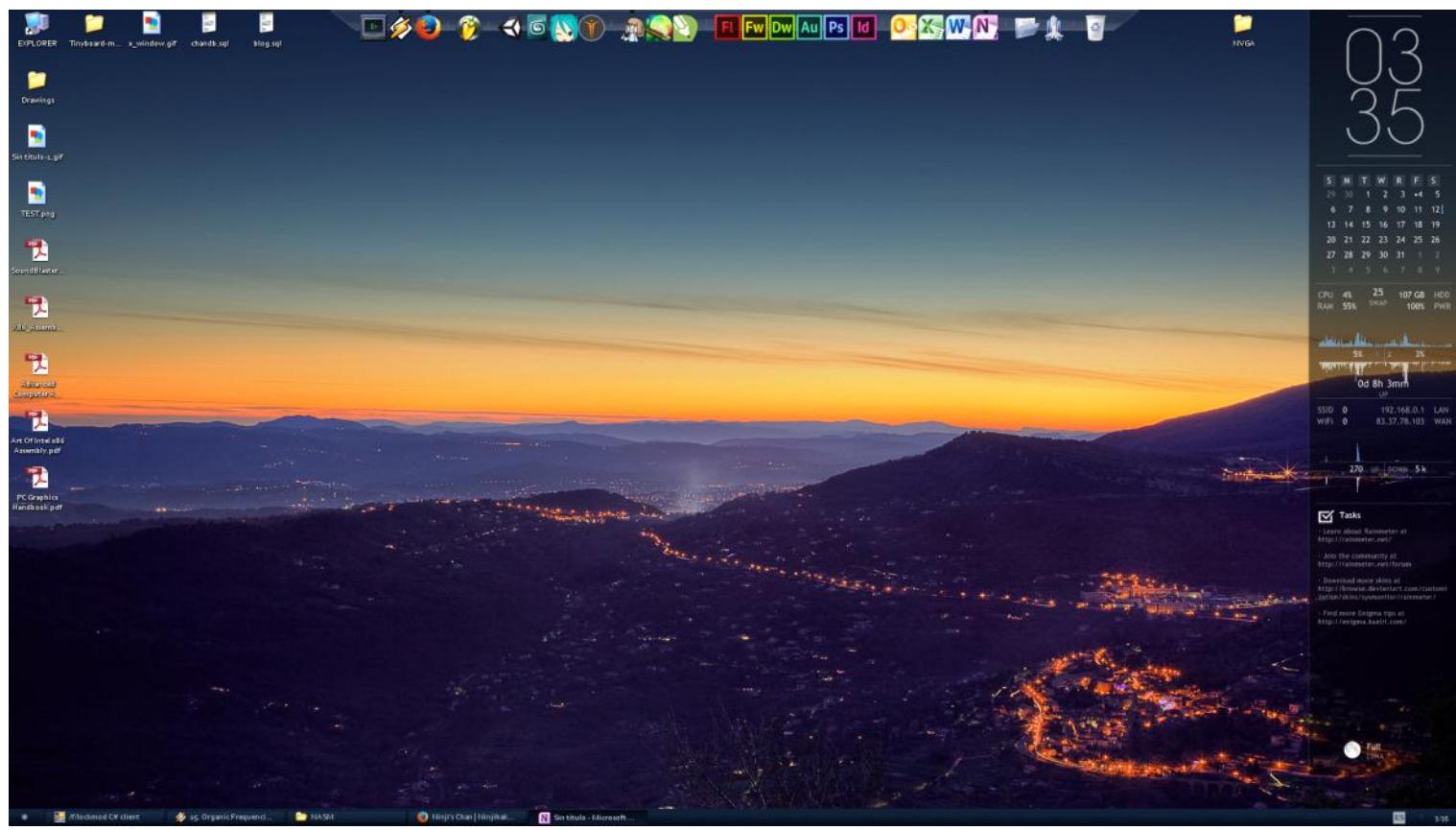


Recorte de pantalla realizado: 12/07/2014 3:33

## A normal day on Ninja's Computer

sábado, 12 de julio de 2014

3:34



Recorte de pantalla realizado: 12/07/2014 3:35



Recorte de pantalla realizado: 12/07/2014 3:36

# BASH COMMANDS

sábado, 12 de julio de 2014  
23:59

## An A-Z Index of the Bash command line for Linux.

[alias](#) Create an alias •

[apropos](#) Search Help manual pages (man -k)

[apt-get](#) Search for and install software packages (Debian/Ubuntu)

[aptitude](#) Search for and install software packages (Debian/Ubuntu)

[aspell](#) Spell Checker

[awk](#) Find and Replace text, database sort/validate/index

b

[basename](#) Strip directory and suffix from filenames

[bash](#) GNU Bourne-Again SHell

[bc](#) Arbitrary precision calculator language

[bg](#) Send to background

[break](#) Exit from a loop •

[builtin](#) Run a shell builtin

[bzip2](#) Compress or decompress named file(s)

c

[cal](#) Display a calendar

[case](#) Conditionally perform a command

[cat](#) Concatenate and print (display) the content of files

[cd](#) Change Directory

[cfdisk](#) Partition table manipulator for Linux

[chgrp](#) Change group ownership

[chmod](#) Change access permissions

[chown](#) Change file owner and group

[chroot](#) Run a command with a different root directory

[chkconfig](#) System services (runlevel)

[cksum](#) Print CRC checksum and byte counts

[clear](#) Clear terminal screen

[cmp](#) Compare two files

[comm](#) Compare two sorted files line by line

[command](#) Run a command - ignoring shell functions •

[continue](#) Resume the next iteration of a loop •

[cp](#) Copy one or more files to another location

[cron](#) Daemon to execute scheduled commands

[crontab](#) Schedule a command to run at a later time

[csplit](#) Split a file into context-determined pieces

[cut](#) Divide a file into several parts

d

[date](#) Display or change the date & time

[dc](#) Desk Calculator

[dd](#) Convert and copy a file, write disk headers, boot records

[ddrescue](#) Data recovery tool

[declare](#) Declare variables and give them attributes •

[df](#) Display free disk space

[diff](#) Display the differences between two files

[diff3](#) Show differences among three files

[dig](#) DNS lookup

[dir](#) Briefly list directory contents

[dircolors](#) Colour setup for `ls'

dirname Convert a full pathname to just a path  
dirs Display list of remembered directories  
dmesg Print kernel & driver messages  
du Estimate file space usage

e

echo Display message on screen •  
egrep Search file(s) for lines that match an extended expression  
eject Eject removable media  
enable Enable and disable builtin shell commands •  
env Environment variables  
ethtool Ethernet card settings  
eval Evaluate several commands/arguments  
exec Execute a command  
exit Exit the shell  
expect Automate arbitrary applications accessed over a terminal  
expand Convert tabs to spaces  
export Set an environment variable  
expr Evaluate expressions

f

false Do nothing, unsuccessfully  
fdformat Low-level format a floppy disk  
fdisk Partition table manipulator for Linux  
fg Send job to foreground  
grep Search file(s) for lines that match a fixed string  
file Determine file type  
find Search for files that meet a desired criteria  
fmt Reformat paragraph text  
fold Wrap text to fit a specified width.  
for Expand words, and execute commands  
format Format disks or tapes  
free Display memory usage  
fsck File system consistency check and repair  
ftp File Transfer Protocol  
function Define Function Macros  
fuser Identify/kill the process that is accessing a file

g

gawk Find and Replace text within file(s)  
getopts Parse positional parameters  
grep Search file(s) for lines that match a given pattern  
groupadd Add a user security group  
groupdel Delete a group  
groupmod Modify a group  
groups Print group names a user is in  
gzip Compress or decompress named file(s)

h

hash Remember the full pathname of a name argument  
head Output the first part of file(s)  
help Display help for a built-in command •  
history Command History  
hostname Print or set system name

i

iconv Convert the character set of a file  
id Print user and group id's  
if Conditionally perform a command  
ifconfig Configure a network interface  
ifdown Stop a network interface  
ifup Start a network interface up

[import](#) Capture an X server screen and save the image to file

[install](#) Copy files and set attributes

j

[jobs](#) List active jobs •

[join](#) Join lines on a common field

k

[kill](#) Stop a process from running

[killall](#) Kill processes by name

l

[less](#) Display output one screen at a time

[let](#) Perform arithmetic on shell variables •

[link](#) Create a link to a file

[ln](#) Create a symbolic link to a file

[local](#) Create variables •

[locate](#) Find files

[logname](#) Print current login name

[logout](#) Exit a login shell •

[look](#) Display lines beginning with a given string

[lpc](#) Line printer control program

[lpr](#) Off line print

[lprint](#) Print a file

[lprintd](#) Abort a print job

[lprintq](#) List the print queue

[lprm](#) Remove jobs from the print queue

[ls](#) List information about file(s)

[lsof](#) List open files

m

[make](#) Recompile a group of programs

[man](#) Help manual

[mkdir](#) Create new folder(s)

[mkfifo](#) Make FIFOs (named pipes)

[mkisofs](#) Create an hybrid ISO9660/JOLIET/HFS filesystem

[mknod](#) Make block or character special files

[more](#) Display output one screen at a time

[mount](#) Mount a file system

[mtools](#) Manipulate MS-DOS files

[mtr](#) Network diagnostics (traceroute/ping)

[mv](#) Move or rename files or directories

[mmv](#) Mass Move and rename (files)

n

[netstat](#) Networking information

[nice](#) Set the priority of a command or job

[nl](#) Number lines and write files

[nohup](#) Run a command immune to hangups

[notify-send](#) Send desktop notifications

[nslookup](#) Query Internet name servers interactively

o

[open](#) Open a file in its default application

[op](#) Operator access

p

[passwd](#) Modify a user password

[paste](#) Merge lines of files

[pathchk](#) Check file name portability

[ping](#) Test a network connection

[pkill](#) Stop processes from running

[popd](#) Restore the previous value of the current directory

[pr](#) Prepare files for printing

printcap Printer capability database  
printenv Print environment variables  
printf Format and print data •  
ps Process status  
pushd Save and then change the current directory  
pv Monitor the progress of data through a pipe  
pwd Print Working Directory

q  
quota Display disk usage and limits  
quotacheck Scan a file system for disk usage  
quotactl Set disk quotas

r  
ram ram disk device  
rcp Copy files between two machines  
read Read a line from standard input •  
readarray Read from stdin into an array variable •  
readonly Mark variables/functions as readonly  
reboot Reboot the system  
rename Rename files  
renice Alter priority of running processes  
remsync Synchronize remote files via email  
return Exit a shell function  
rev Reverse lines of a file  
rm Remove files  
rmdir Remove folder(s)  
rsync Remote file copy (Synchronize file trees)

s  
screen Multiplex terminal, run remote shells via ssh  
scp Secure copy (remote file copy)  
sdiff Merge two files interactively  
sed Stream Editor  
select Accept keyboard input  
seq Print numeric sequences  
set Manipulate shell variables and functions  
sftp Secure File Transfer Program  
shift Shift positional parameters  
shopt Shell Options  
shutdown Shutdown or restart linux  
sleep Delay for a specified time  
slocate Find files  
sort Sort text files  
source Run commands from a file ''  
split Split a file into fixed-size pieces  
ssh Secure Shell client (remote login program)  
strace Trace system calls and signals  
su Substitute user identity  
sudo Execute a command as another user  
sum Print a checksum for a file  
suspend Suspend execution of this shell •  
sync Synchronize data on disk with memory

t  
tail Output the last part of file  
tar Store, list or extract files in an archive  
tee Redirect output to multiple files  
test Evaluate a conditional expression  
time Measure Program running time  
timeout Run a command with a time limit

[times](#) User and system times  
[touch](#) Change file timestamps  
[top](#) List processes running on the system  
[traceroute](#) Trace Route to Host  
[trap](#) Run a command when a signal is set(bourne)  
[tr](#) Translate, squeeze, and/or delete characters  
[true](#) Do nothing, successfully  
[tsort](#) Topological sort  
[tty](#) Print filename of terminal on stdin  
[type](#) Describe a command •

u

[ulimit](#) Limit user resources •  
[umask](#) Users file creation mask  
[umount](#) Unmount a device  
[unalias](#) Remove an alias •  
[uname](#) Print system information  
[unexpand](#) Convert spaces to tabs  
[uniq](#) Uniquify files  
[units](#) Convert units from one scale to another  
[unset](#) Remove variable or function names  
[unshar](#) Unpack shell archive scripts  
[until](#) Execute commands (until error)  
[uptime](#) Show uptime  
[useradd](#) Create new user account  
[userdel](#) Delete a user account  
[usermod](#) Modify user account  
[users](#) List users currently logged in  
[uuencode](#) Encode a binary file  
[uudecode](#) Decode a file created by uuencode

v

[v](#) Verbosely list directory contents ('ls -l -b')  
[vdir](#) Verbosely list directory contents ('ls -l -b')  
[vi](#) Text Editor  
[vmstat](#) Report virtual memory statistics

w

[wait](#) Wait for a process to complete •  
[watch](#) Execute/display a program periodically  
[wc](#) Print byte, word, and line counts  
[whereis](#) Search the user's \$path, man pages and source files for a program  
[which](#) Search the user's \$path for a program file  
[while](#) Execute commands  
[who](#) Print all usernames currently logged in  
[whoami](#) Print the current user id and name ('id -un')  
[wget](#) Retrieve web pages or files via HTTP, HTTPS or FTP  
[write](#) Send a message to another user

x

[xargs](#) Execute utility, passing constructed argument list(s)  
[xdg-open](#) Open a file or URL in the user's preferred application.  
[yes](#) Print a string until interrupted  
[zip](#) Package and compress (archive) files.  
[\\_](#) Run a command script in the current shell  
[!!](#) Run the last command again  
[###](#) Comment / Remark

Commands marked • are bash *built-ins*

Many commands particularly the Core Utils are also available under alternate [shells](#) (C shell, Korn shell etc).

More bash commands: [Linux Command Directory](#) from O'Reilly, GNU [CoreUtils](#).

[SS64 bash discussion forum](#)  
[Links to other Sites, books etc](#)

Pegado de <<http://ss64.com/bash/>>

# Argument list

miércoles, 23 de julio de 2014  
13:45

```
int main(int args, char* argv[])
{
    if(args > 0)
        for(int x = 0; x < args; x++)
            printf("%i: %s\n", x, argv[x]);
    getchar();
    return 0;
}
```

**Args:** Argument count

**Argv:** Arguments (Also called Argument Vector)

Index 0 in the argument vector is reserved. It sometimes contain the path of the executable, and other times the executable name. The rest are the arguments.

*DEMO.EXE helloworld hello world "hello world"*

```
C:\MYDOCS\Visual Studio 2010\Projects\NRFPACKAGER\Debug>NRFPACKAGER.EXE helloworld hello world "hello world"
0: NRFPACKAGER.EXE
1: helloworld
2: hello
3: world
4: hello world
```

Recorte de pantalla realizado: 23/07/2014 13:49

# Printf: Print text with format

miércoles, 23 de julio de 2014

13:49

Specified in:

<cstdio> (C)  
<stdio.h> (C++)

## Print formatted data to stdout

Writes the C string pointed by *format* to the standard output ([stdout](#)). If *format* includes *format specifiers* (subsequences beginning with %), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

Pegado de <<http://www.cplusplus.com/reference/cstdio/printf/>>

## PARAMETERS

### format

C string that contains the text to be written to [stdout](#).

It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype: [[see compatibility note below](#)]

%[flags][width][.precision][length]specifier

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

Pegado de <<http://www.cplusplus.com/reference/cstdio/printf/>>

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a <code>signed int</code> . The number of characters written so far is stored in the pointed location.	
*	A % followed by another % character will write a single % to the stream.	%

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

<b>flags</b>	<b>description</b>
-	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceded with 0, 0x or 0X respectively for values different than zero.
*	Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> sub-specifier).

<b>width</b>	<b>description</b>
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The <i>width</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

<b>precision</b>	<b>description</b>
.	For integer specifiers (d, i, o, u, x, X): <i>precision</i> specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A <i>precision</i> of 0 means that no character is written for the value 0.
.number	For a, A, e, E, f and F specifiers: this is the number of digits to be printed <b>after</b> the decimal point (by default, this is 6). For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for <i>precision</i> , 0 is assumed.
.*	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

The *length* sub-specifier modifies the length of the data type. This is a chart showing the types used to interpret the corresponding arguments with and without *length* specifier (if a different type is used, the proper type promotion or conversion is performed, if allowed):

<b>length</b>	<b>specifiers</b>							
	d i	u o x X	f F e E g G a A	c	s	p		n
(none)	int	unsigned int	double	int	char*	void*	int*	
hh	signed char	unsigned char						signed char*
h	short int	unsigned short int						short int*
l	long int	unsigned long int		wint_t	wchar_t*			long int*
ll	long long int	unsigned long long int						long long int*
j	intmax_t	uintmax_t						intmax_t*
z	size_t	size_t						size_t*
t	ptrdiff_t	ptrdiff_t						ptrdiff_t*
L			long double					

Note that the c specifier takes an int (or wint\_t) as argument, but performs the proper conversion to a char value (or a wchar\_t) before formatting it for output.

**Note:** Yellow rows indicate specifiers and sub-specifiers introduced by C99. See [<cinttypes>](#) for the specifiers for extended types.

### ... (additional arguments)

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for *n*).

There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.

Pegado de <<http://www.cplusplus.com/reference/cstdio/printf/>>

## Return Value

On success, the total number of characters written is returned.

If a writing error occurs, the *error indicator* ([ferror](#)) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, [errno](#) is set to EILSEQ and a negative number is returned.

Pegado de <<http://www.cplusplus.com/reference/cstdio/printf/>>

## Example

```
1 /* printf example */
2 #include <stdio.h>
3
4 int main()
5 {
6     printf ("Characters: %c %c \n", 'a', 65);
7     printf ("Decimals: %d %ld\n", 1977, 650000L);
8     printf ("Preceding with blanks: %10d \n", 1977);
9     printf ("Preceding with zeros: %010d \n", 1977);
10    printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
11    printf ("floats: %4.2f %+0.e %E \n", 3.1416, 3.1416, 3.1416);
12    printf ("Width trick: %*d \n", 5, 10);
13    printf ("%s \n", "A string");
14    return 0;
15 }
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:      1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Recorte de pantalla realizado: 23/07/2014 13:53

## Compatibility

Particular library implementations may support additional *specifiers* and *sub-specifiers*.

Those listed here are supported by the latest C and C++ standards (both published in 2011), but those in yellow were introduced in C99 (only required for C++ implementations since C++11), and may not be supported by libraries that comply with older standards.

Pegado de <<http://www.cplusplus.com/reference/cstdio/printf/>>

# GCC Intel Inline Assembly

viernes, 01 de agosto de 2014  
0:34

You have to pass an argument to GCC assembler.

```
gcc.exe -masm=intel -c Main.c
gcc.exe Main.o -oMain.exe
And you have C code like this:
#include <conio.h>
#include <stdio.h>
int myVar = 0;
int main(int argc, char *argv[])
{
    asm("mov eax, dword ptr fs:[0x18]");
    asm("mov eax, dword ptr ds:[eax+0x30]");
    asm("movzx eax, byte ptr ds:[eax+0x2]");
    asm("mov _myVar, eax");
if(myVar == 1) printf("This program has been debugged.\r\n");
    printf("Welcome.\r\n");
    getch();
return 0;
}
```

Don't forget to add prefix underscore (\_) for every variables in asm() keyword, or it won't recognize it.

And keyword asm() use prefix '0x' for every hexadecimal integer, not suffix 'h'.

Pegado de <<http://stackoverflow.com/questions/5397677/gcc-intel-syntax-inline-assembly>>

```
#include <stdio.h>
```

```
char* text = "Hello world!";
```

```
int main()
{
    asm("mov eax, _text");
    asm("push eax");
    asm("call _printf");
    asm("pop eax");

    getchar();
    return 0;
}
```

```
del test.exe
gcc.exe -masm=intel -c test.c
gcc.exe test.o -o test.exe
pause
```

# Matriz Pentadimensional

jueves, 24 de julio de 2014  
11:34

```
TEXT BELOW IS SELECTED. PLEASE PRESS CTRL+C TO COPY TO YOUR CLIPBOARD. (⌘+C ON MAC)
1. static void Main(string[] args)
2. {
3.     int[,,,,] Pentadimensional = new int[8,8,8,8,8];
4.
5.     for (int x = 0; x < 8; x++)
6.     {
7.         for (int y = 0; y < 8; y++)
8.         {
9.             for (int z = 0; z < 8; z++)
10.            {
11.                for (int a = 0; a < 8; a++)
12.                {
13.                    for (int b = 0; b < 8; b++)
14.                    {
15.                        Pentadimensional[x, y, z, a, b] = new Random().Next(0, 1000);
16.                    }
17.                }
18.            }
19.        }
20.    }
21.    while (true)
22.    {
23.        Random rnd = new Random();
24.        int xi = rnd.Next(0, 7);
25.        int yi = rnd.Next(0, 7);
26.        int zi = rnd.Next(0, 7);
27.        int ai = rnd.Next(0, 7);
28.        int bi = rnd.Next(0, 7);
29.        Console.WriteLine("Pentadimensional[" + xi.ToString() + "," + yi.ToString() + "," + zi.ToString() + "," +
ai.ToString() + "," + bi.ToString() + "] = " + Pentadimensional[xi, yi, zi, ai, bi].ToString());
30.        Console.ReadLine();
31.    }
32. }
```

Recorte de pantalla realizado: 24/07/2014 11:35

## Cirnosay.exe

jueves, 24 de julio de 2014  
11:35



The screenshot shows a code editor window with a light gray background and a dark gray header bar. The header bar contains three icons: a file icon, a copy/paste icon, and a save icon. The main area of the editor displays a block of C++ code. The code is color-coded, with different colors used for various syntax elements. The code itself is a function named BCX\_Bitmap, which takes a character pointer and several integer parameters. It includes includes for stdio.h, string.h, windows.h, iostream, and ctime. It also uses the ::GetForegroundWindow() function and the std namespace. The function implementation involves getting a handle to the console, filling the screen buffer with a specific character and color, and then setting the cursor position.

```
1.
2.
3. // put a bitmap image on a Windows Console display
4. // BCX basic original by Joe Caverly and Kevin Diggins
5. // BCX generated C code modified for PellesC/Dev-C++
6. #include <stdio.h>
7. #include <string.h>
8. #include <windows.h> // Win32Api Header File
9. #include <iostream>
10. #include <ctime>
11.
12. static HWND hConWnd;
13.
14. HWND BCX_Bitmap(char*,HWND=0,int=0,int=0,int=0,int=0,int=0,int=0,int=0);
15. HWND GetConsoleWndHandle(void);
16. HWND hWnd = ::GetForegroundWindow();
17. using namespace std;
18.
19. void clearconsole() {
20.     COORD topLeft = { 0, 0 };
21.     HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
22.     CONSOLE_SCREEN_BUFFER_INFO screen;
23.     DWORD written;
24.
25.     GetConsoleScreenBufferInfo(console, &screen);
26.     FillConsoleOutputCharacterA(
27.         console, ' ', screen.dwSize.X * screen.dwSize.Y, topLeft, &written
28.     );
29.     FillConsoleOutputAttribute(
30.         console, FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE,
31.         screen.dwSize.X * screen.dwSize.Y, topLeft, &written
32.     );
33.     SetConsoleCursorPosition(console, topLeft);
34. }
35.
```

```
36. int main(int argc, char* argv[])
37. {
38.     hConsole = GetConsoleWindow();
39.     char delX = 0x127;
40.     int chars = 0, lines = 1;
41.     char* filename = "imgs\\Cirno.bmp";
42.
43.     if(argc > 1 && strcmp(argv[1], "-f") == 0)
44.     {
45.         filename = argv[2];
46.     }
47.
48.     clearconsole();
49.
50.     if(argc < 2)
51.     {
52.         std::cout<< "\t\t\t _____" << std::endl;
53.         std::cout<< "\t\t\t / \\" << std::endl;
54.         std::cout<< "\t\t\t | Usage: |" << std::endl;
55.         std::cout<< "\t\t\t | cirnosay [-f(filepath)][-t] message |" << std::endl;
56.         std::cout<< "\t\t\t | -f: A path to a bmp file.MUST be BMP |" << std::endl;
57.         std::cout<< "\t\t\t | -t: Prints the current time and date |" << std::endl;
58.         std::cout<< "\t\t\t | message: The words you want to print |" << std::endl;
59.         std::cout<< "\t\t\t \\_____/\" << std::endl;
60.         std::cout<< "\t\t\t /" << std::endl;
61.         std::cout<< "\t\t\t/" << std::endl;
62.         lines = 5;
63.     }else{
64.
65.     time_t t = time(0);
66.     struct tm * now = localtime( & t );
67.     char* result = asctime(now);
68.     std::cout<< "\t\t\t _____" << std::endl;
69.     std::cout<< "\t\t\t / \\" << std::endl;
70. }
```

```
70.
71.     for(int n = 1; n < argc; n++)
72.     {
73.         if(strcmp(argv[n],"-t") == 0)
74.             std::cout<< "\t\t\t | " << result;
75.         if(strcmp(argv[n],"-F") != 0 && strcmp(argv[n],"-t") != 0)
76.             {
77.                 if(chars == 0)
78.                     std::cout<< "\t\t\t | ";
79.                 std::cout<< " " << argv[n];
80.                 chars = chars + sizeof(argv[n]) + 1;
81.                 if(chars > 20)
82.                     {
83.                         std::cout<< std::endl;
84.                         chars = 0;
85.                         lines++;
86.                     }
87.             }
88.
89.     }
90.     std::cout << std::endl;
91.     std::cout<< "\t\t\t \\_________________________________/" << std::endl;
92.     std::cout<< "\t\t\t /" << std::endl;
93.     std::cout<< "\t\t\t/" << std::endl;
94. }
95. if (hConWnd)
96. {
97. // select a bitmap file you have or use one of the files in the Windows folder
98. // filename, handle, ID, ulcX, ulcY, width, height 0,0 auto-adjusts
99. BCX_Bitmap(filename,hWnd,123,1,9+(lines*9),0,0);
100. getchar(); // wait
101. }
102. return 0;
103. }
104.
```

```

105. // draw the bitmap
106. HWND BCX_Bitmap(char* Text,HWND hWnd,int id,int X,int Y,int W,int H,int Res,int Style,int Exstyle)
107. {
108.     HWND A;
109.     HBITMAP hBitmap;
110.     // set default style
111.     if (!Style) Style = WS_CLIPSIBLINGS|WS_CHILD|WS_VISIBLE|SS_BITMAP|WS_TABSTOP;
112.     // form for the image
113.     A = CreateWindowEx(Exstyle,"static",NULL,Style,X,Y,0,0,hWnd,(HMENU)id,GetModuleHandle(0),NULL);
114.     // Text contains filename
115.     hBitmap=(HBITMAP)LoadImage(0,Text,IMAGE_BITMAP,128,128,LR_LOADFROMFILE|LR_CREATEDIBSECTION);
116.     // auto-adjust width and height
117.     if (W || H) hBitmap = (HBITMAP)CopyImage(hBitmap,IMAGE_BITMAP,W,H,LR_COPYRETURNORG);
118.     SendMessage(A,(UINT)STM_SETIMAGE,(WPARAM)IMAGE_BITMAP,(LPARAM)hBitmap);
119.     if (W || H) SetWindowPos(A,HWND_TOP,X,Y,W,H,SWP_DRAWFRAME);
120.     return A;
121. }
122.
123. // tricking Windows just a little ...
124. HWND GetConsoleWndHandle(void)
125. {
126.     HWND hConWnd;
127.     OSVERSIONINFO os;
128.     char szTempTitle[64], szClassName[128], szOriginalTitle[1024];
129.     os.dwOSVersionInfoSize = sizeof( OSVERSIONINFO );
130.     GetVersionEx( &os );
131.     // may not work on WIN9x
132.     if ( os.dwPlatformId == VER_PLATFORM_WIN32s ) return 0;
133.     GetConsoleTitle( szOriginalTitle, sizeof( szOriginalTitle ) );
134.     sprintf( szTempTitle,"%u - %u", GetTickCount(), GetCurrentProcessId() );
135.     SetConsoleTitle( szTempTitle );
136.     Sleep( 40 );
137.     // handle for NT and XP
138.     hConWnd = FindWindow( NULL, szTempTitle );
139.     SetConsoleTitle( szOriginalTitle );
140.     // may not work on WIN9x
141.     if ( os.dwPlatformId == VER_PLATFORM_WIN32s )
142.     {
143.         hConWnd = GetWindow( hConWnd, GW_CHILD );
144.         if ( hConWnd == NULL ) return 0;
145.         GetClassName( hConWnd, szClassName, sizeof( szClassName ) );
146.         // while ( _stricmp( szClassName, "ttyGrab" ) != 0 )
147.         while ( strcmp( szClassName, "ttyGrab" ) != 0 )
148.         {
149.             hConWnd = GetNextWindow( hConWnd, GW_HWNDNEXT );
150.             if ( hConWnd == NULL ) return 0;
151.             GetClassName( hConWnd, szClassName, sizeof( szClassName ) );
152.         }
153.     }
154.     return hConWnd;
155. }

```

Recorte de pantalla realizado: 24/07/2014 11:37

## Pointers in C/C++

jueves, 24 de julio de 2014  
11:39

```
1. /*
2. We use pointers when we use dynamic memory. For example, when we use arrays of 'n' elements.
3.
4. Since we don't know exactly how many elements do we have in an array, we pass a pointer to the
5. first element, and keep reading from there. The type of the pointer define the size of each element.
6. For example, a char* is 1 byte per element, while an int* is 4 bytes per element (in 32+ bits programming).
7.
8. If we are not sure how many bytes a type or structure has, we can always use the 'sizeof()' function.
9. */
10.
11. #include <cstdlib>
12. #include <cstdio>
13. #include <cstring>
14. #include <iostream>
15. #include <Windows.h>
16.
17. using namespace std;
18.
19. //Function that reads an integer from a given memory address.
20. int ReadMemory(int *addr)
21. {
22.     //Define x
23.     int x = 0;
24.     // '*' is a pointer to an address (dPtr points now to addr)
25.     int *dPtr = addr;
26.     //Get first value from address stored in dPtr, store value in x
27.     x = dPtr[0];
28.     //Return value of x
29.     return x;
30. }
31.
32. //Main function. Program starts here.
33. int main(int argc, char* argv[])
34. {
35.     //Define 'a' (a = 25)
36.     int a = 25;
37.     //Define 'b'. Stroe in 'b' whatever is in the address of 'a'.
38.     // '&' means 'address of'.
39.     int b = ReadMemory(&a);
40.
41.     //Print value of 'b' (25)
42.     cout << b << endl;
```

```
44. /*CONCLUSION:  
45. *  
46. * int x defines a variable.  
47. * int *x defines a pointer, that points to a memory address  
48. * x is the value of the variable x  
49. * &x is the memory address where x is stored.  
50. *  
51. * x = &y Makes 'x' contains the address of 'y' as a value  
52. * *x = &y Makes 'x' point to the address of 'y'  
53. * A pointer must be defined as a pointer. You can't do the following:  
54. * int x = 0; *x = &y;  
55. * (And even if you could, it would give you a runtime error).  
56. * *x = (int*)0x12345678 would make 'x' point to address 0x12345678  
57. * int y = x[0] being *x = 0x12345678 would give you an access violation error  
58. * The reason is because you can't access a memory address outside of your region.  
59. * x[0] Refer to the first value stored in the address of 'x'  
60. *  
61. */  
62.  
63. getchar();  
64. return 0;  
65. }
```

Recorte de pantalla realizado: 24/07/2014 11:39

# Memory allocation 16 bit NASM

jueves, 24 de julio de 2014

11:40

```
; 16 BIT ASM MEMORY ALLOCATION EXAMPLE Revision 2
; COMPILES WITH NASM
; 06-10-2013
; By NinjiHaku

;CHANGELOG:
;- Changed AX for CX, to be used as a counter for the
; print routine.
; CX is a base register meant to be used to track loops.
; Any base register can be used for that purpose, though. But
; this way, the code is more organized and clean.

;
; This program allocate 32 bytes in memory using
; DOS INT 21H (AH=48H), then store user input characters
; into the allocated memory.
; When the user press ENTER, or reach the character limit,
; the program prints the text stored in the memory into the
; screen.

;DOS Allocate memory using "paragraphs". One paragraph is
;16 bytes long.

[BITS 16]
ORG 100h

section .text

MAIN:
    mov ah, 0x0    ;Set video mode
    mov al, 0x12   ;Mode 12, VGA 16 colors
    int 0x10      ;Start VGA. Unnecesary, but it looks better
.MALLOC          ;Memory allocation
    xor ax, ax    ;Initialize ax
    mov bx, 0x02   ;16*2 = 32 bytes
    mov ah, 0x48   ;Function 48 (malloc)
    int 0x21      ;Do it

    mov di, ax    ;Set destination pointer to the allocated memory
    xor bx, bx    ;Initialize BX

.GETCHAR:       ;Read a character from keyboard, store it in memory
    push bx      ;Just in case, save bx in the stack
    xor al, al    ;Initialize al
    mov ah, 0x01  ;Mode 1, get character with echo
    int 0x21      ;Do it
    pop bx       ;Restore bx
    cmp al, 0x0d  ;Has the user pressed ENTER?
    je .PRINT    ;If so, print the text
    mov [bx+di], al ;Store al in memory
    inc bx       ;Increment in 1 the value of bx
```

```

        cmp bx, 32    ;Check for memory overflow
        je .PRINT    ;If so, scape
        jmp .GETCHAR ;Keep reading the keyboard

.PRINT           ;Print the text on screen (ploop is the loop)
        cmp bx, 0    ;Check if the user input something
        je .end      ;If not, exit
        mov si, di   ;Prepare to read the memory
        xor ax, ax   ;Initialize AL
        push bx     ;We'll need bx's value it later

        mov dh, 0x02 ;We'll move the cursor to the second line
        mov dl, 0x0  ;Column 0
        mov bh, 0x0  ;First page
        mov ah, 0x02 ;Set cursor position mode
        int 0x10    ;Do it!

        pop bx      ;Recover bx
        xor cx, cx   ;We'll use CX as a counter
.ploop
        push bx      ;We need to push again inside the loop
        lodsb       ;Get next character
        mov ah, 0x0e ;Text mode
        mov bh, 0x0  ;First Page
        mov bl, 0x07 ;Text attribute
        int 0x10    ;Do it
        pop bx      ;Recover bx
        inc cx      ;Increment AL
        cmp cx, bx   ;Check if we have readed every byte
        jne .ploop  ;Keep printing characters

.end           ;Wait for a keypress, then end
        mov ah, 0x01 ;We are going to wait for the user to press a key
        int 0x16    ;Do it
        je .end      ;Keep waiting until it happen

.dealloc        ;Free the used memory
        mov ah, 0x49 ;Deallocate memory
        mov es, si   ;Set the segment parameter
        int 0x21    ;Do it!

.exitprogram
        mov al, 0x03 ;Return to text mode
        mov ah, 0x0  ;Set video mode
        int 0x10    ;Do it
        int 0x20    ;Exit program with no parameters

```

section .data

Signature db "By NinjiHaku Software" ;Unused, ignore it

section .bss

Pegado de <<http://pastebin.com/9zSTnMqJ>>

# Logic Test

jueves, 24 de julio de 2014  
11:41

```
// LogicTest.cpp : Defines the entry point for the console application.
// By Ninjihaku
//

#include "stdafx.h" //Standard Visual Studio Header. Includes stdio.h and tchar.h.
#include <math.h> //Math library required for POW function.

//Program entry point (use main if you're not in VS).
//argv can be replaced by a char pointer (char*), if tchar.h is not included.
//Not required, though.

int _tmain(int argc, _TCHAR* argv[])
{
    //Inputs: a, b, c
    //Outputs: s1
    //LOGIC FUNCTION: c*b + a * (b*c)
    //When at least two of these 3 booleans are TRUE, s1 is also TRUE.

    bool a = false, b = false, c = false; //Define inputs
    bool s1 = false; //Define outputs
    int inputs = 3; //Number of inputs

    //Get number of possible cases (2^inputs)
    double cases = pow((double)2,inputs);

    printf("LOGIC FUNCTION: C*B + A*(B+C)\n\nINPUTS | OUTPUTS\nC B A | S1 \n");

    //Compute all cases
    for(double x = 0; x < cases; x++)
    {
        //Calculate the state of s1 for the current inputs
        s1 = (c && b) || (a && (b || c));

        //Print the current state. Replace ON and OFF with TRUE or FALSE if you want.
        if(s1)
            printf("%u %u %u -> ON\n",c,b,a);
        else
            printf("%u %u %u -> OFF\n",c,b,a);

        //Switch the inputs.
        c = !c;
        if(!c)
            b = !b;
        if(!b && !c)
            a = !a;

    }

    //Pause the console so we can read the output.
    getchar();
    return 0;
}
```

```
}
```

```
//This program has the following output:
```

```
/*
LOGIC FUNCTION: C*B + A*(B+C)
```

```
INPUTS | OUTPUTS
```

```
C B A | S1
```

```
0 0 0 -> OFF
```

```
1 0 0 -> OFF
```

```
0 1 0 -> OFF
```

```
1 1 0 -> ON
```

```
0 0 1 -> OFF
```

```
1 0 1 -> ON
```

```
0 1 1 -> ON
```

```
1 1 1 -> ON
```

```
*/
```

Pegado de <<http://pastebin.com/KMr2NK1Q>>