

INFB6074 - INFRAESTRUCTURA PARA CS. DE DATOS  
INGENIERÍA CIVIL EN CIENCIA DE DATOS

Actividad N° 1

*“ Boot de Micro S.O. en QEMU. ”*

Dr. Ing. Michael Miranda Sandoval

Mayo de 2025

## ACTIVIDAD N° 1

### INTEGRANTES:

**Nombre:** Felipe Martínez González

**Email:** fmartinezgo@utem.cl

**Nombre:** Christian Pérez Flores

**Email:** cperezfl@utem.cl

**Nombre:** Benjamín Zamorano Soto

**Email:** bzamoranos@utem.cl

## 1. Introducción

En este informe se presenta el desarrollo de un micro sistema operativo (micro S.O.) implementado en lenguaje ensamblador, diseñado para iniciar y ejecutarse en el emulador QEMU. Esta actividad permite comprender y experimentar de manera práctica el ciclo de inicio (boot) de una arquitectura computacional básica. El objetivo principal es implementar un sistema que, al iniciar correctamente, mostrará un mensaje en pantalla, demostrando el entendimiento del proceso de arranque y de los componentes esenciales que lo conforman. Para ello, se utiliza NASM (Netwide Assembler) como herramienta de ensamblado y QEMU como entorno de emulación. En este documento se detalla cada etapa del desarrollo, desde la escritura del código fuente en ensamblador, su compilación y pruebas en QEMU, hasta el análisis del ciclo de inicio del sistema.

## 2. Detalle de las Actividades

### 2.1 Explicación Previa a Cada Parte de la Secuencia Lógica

- 1. Investigación de Herramientas y Conceptos:** Se identificaron dos herramientas clave para este propósito: NASM (Netwide Assembler) y QEMU (Quick Emulator). NASM es un ensamblador ampliamente utilizado que permite escribir código de bajo nivel directamente en lenguaje ensamblador, esencial para crear un bootloader o núcleo del sistema operativo. QEMU, por su parte, es un emulador de hardware que permite simular una arquitectura x86, lo cual resulta invaluable para probar y ejecutar código del sistema sin necesidad de hardware físico adicional. Comprender la estructura del sector de arranque (boot sector), el funcionamiento de la BIOS, y cómo interactúa un sistema operativo con el hardware fueron conceptos fundamentales para establecer una base sólida.
- 2. Preparación del Entorno de Desarrollo:** Una vez identificadas las herramientas, se procedió a preparar el entorno de desarrollo en un sistema Windows. Para ello, se instaló Notepad++ por su agilidad para crear scripts y archivos planos. Se organizó el espacio de trabajo en una carpeta específica que contiene todo el código fuente, scripts y binarios generados.
- 3. Instalación de Herramientas Específicas:** Una vez configurado el entorno, se procedió a la instalación de las herramientas. NASM fue descargado desde su sitio oficial, así como QEMU que también fue descargado desde su sitio oficial. Ambas herramientas fueron instaladas manualmente y no contaban con un instalador automatizado, por lo que se descomprimieron en directorios personalizados. Durante la instalación surgió un inconveniente común en entornos de Windows, donde el sistema no reconocía los comandos nasm ni qemu-system-i386. Esto fue solucionado añadiendo las rutas de los ejecutables manualmente en la variable de entorno PATH del sistema (Figura 1). Este paso adicional fue clave, ya que permitió ejecutar comandos de ambas herramientas desde la terminal sin errores.

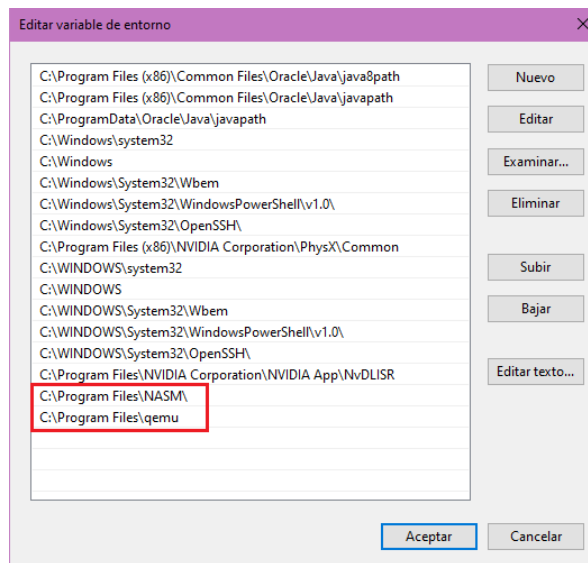


Figure 1: Selección de rutas ejecutables en la variable de entorno PATH del sistema.

4. **Configuración de Herramientas y Entorno:** Después de confirmar que las herramientas estaban correctamente instaladas y accesibles desde la terminal, se procedió a configurar el entorno de trabajo. Se creó el archivo fuente *boot.asm* utilizando Notepad++, asegurándose de que se guardara con la extensión .asm y no como archivo de texto (.txt) (Figura 2).

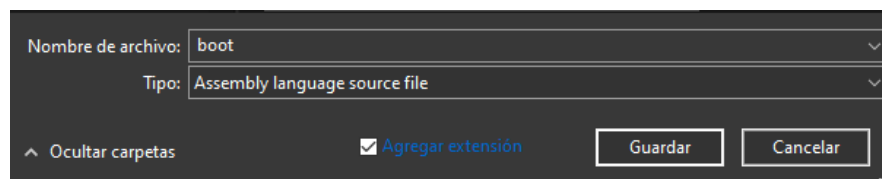


Figure 2: Guardado del archivo *boot.asm* desde Notepad++.

En este archivo se escribió el código de un bootloader simple que imprimía caracteres en pantalla utilizando interrupciones de la BIOS. Este código fue diseñado para ejecutarse directamente al iniciar la máquina virtual, por lo que se emplearon instrucciones en modo real de 16 bits y una dirección de origen (ORG) en 0x7C00, correspondiente al punto de carga de la BIOS. La versión final del bootloader incluyó una serie de características que enriquecieron significativamente la presentación visual y técnica del arranque. En lugar de limitarse a imprimir un solo mensaje, el código genera un recuadro decorativo centrado en la pantalla que muestra el texto "Estoy dentro!" y, tras una línea vacía, despliega un corazón ASCII simétrico compuesto por caracteres como asteriscos y espacios cuidadosamente posicionados (Figura 3). Cada línea del diseño se centra horizontalmente utilizando una función personalizada en ensamblador que calcula la longitud de la cadena y ajusta dinámicamente la posición del cursor. El cursor, además, fue ocultado mediante una interrupción de la BIOS para proporcionar una apariencia más limpia.

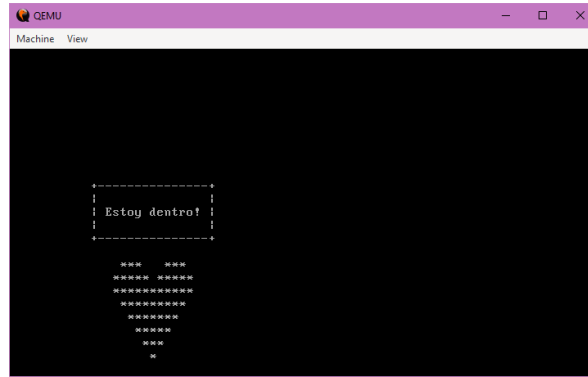


Figure 3: Producto final en QEMU del código a imprimir en pantalla.

Una vez creado el archivo *boot.asm*, se procedió a su compilación manual utilizando NASM desde la terminal con el comando `"nasm -f bin boot.asm -o boot.img"` (Figura 4). Este paso fue esencial para validar que el ensamblador estuviera correctamente instalado y que el archivo fuese sintácticamente correcto, la compilación generó una imagen binaria válida llamada *boot.img*, lista para ser utilizada en un entorno de virtualización.

Luego de generar correctamente el archivo *boot.img*, se procedió a ejecutar QEMU de manera manual, para comprobar visualmente que la imagen compilada funcionara como se esperaba. Para ello se utilizó el comando `"qemu-system-i386 -drive format=raw,file=boot.img"` (Figura 4). Este paso de verificación previa resultó crucial, ya que permitió observar directamente el resultado del bootloader, validar el centrado de texto, el funcionamiento del cursor y la estética del diseño (Figura 3), antes de complicar el proceso con automatización innecesaria.

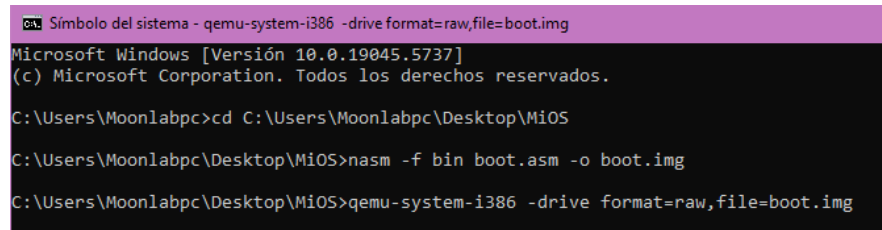
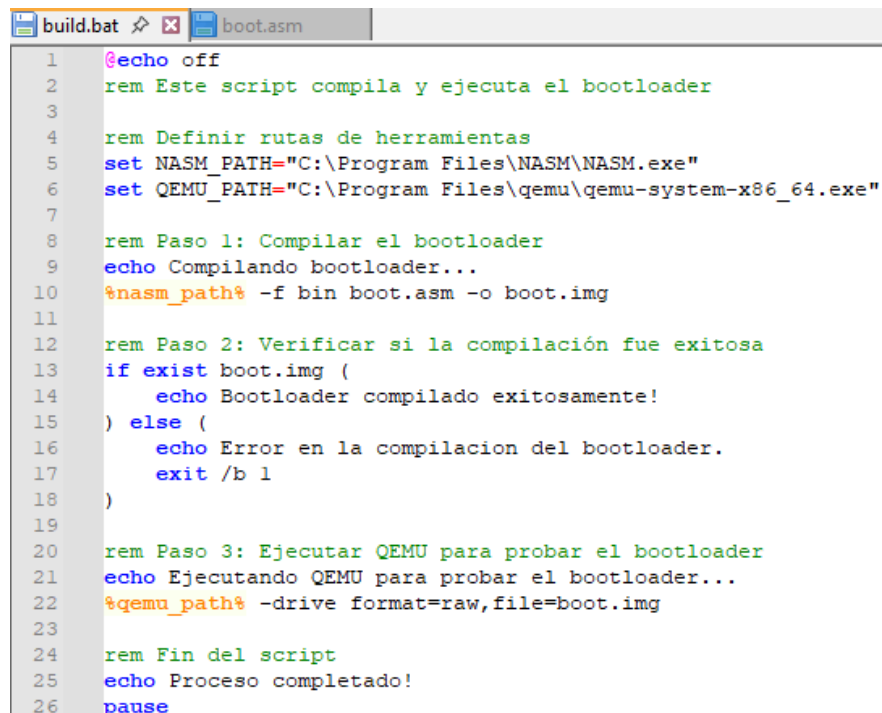


Figure 4: Comandos ejecutados desde la terminal.

Una vez comprobado que la imagen arrancaba correctamente y mostraba la salida prevista, se decidió encapsular todo el flujo en un script por lotes. Se creó entonces un archivo llamado *build.bat*, también con Notepad++, que automatiza todo el proceso (Figura 5: la compilación, la verificación del archivo de salida y la ejecución de QEMU con parámetros adecuados). El script también incluye mensajes informativos, pausa al final y rutas explícitas a los ejecutables, en caso de que el entorno PATH no esté configurado.



```
1 @echo off
2 rem Este script compila y ejecuta el bootloader
3
4 rem Definir rutas de herramientas
5 set NASM_PATH="C:\Program Files\NASM\NASM.exe"
6 set QEMU_PATH="C:\Program Files\qemu\qemu-system-x86_64.exe"
7
8 rem Paso 1: Compilar el bootloader
9 echo Compilando bootloader...
10 %nasm_path% -f bin boot.asm -o boot.img
11
12 rem Paso 2: Verificar si la compilación fue exitosa
13 if exist boot.img (
14     echo Bootloader compilado exitosamente!
15 ) else (
16     echo Error en la compilacion del bootloader.
17     exit /b 1
18 )
19
20 rem Paso 3: Ejecutar QEMU para probar el bootloader
21 echo Ejecutando QEMU para probar el bootloader...
22 %qemu_path% -drive format=raw,file=boot.img
23
24 rem Fin del script
25 echo Proceso completado!
26 pause
```

Figure 5: Código ingresado en el script por lotes en Notepad++.

### 3. Conclusión

En esta actividad se logró implementar un sistema operativo mínimo capaz de iniciar y mostrar un mensaje en pantalla utilizando QEMU. Durante el proceso, se adquirieron conocimientos prácticos sobre el ciclo de arranque de una computadora, la estructura del sector de arranque y el uso de interrupciones BIOS en modo real. Además, se desarrollaron habilidades en el manejo de herramientas como NASM y QEMU, tanto manualmente como mediante scripts automatizados. El diseño final, que incluyó texto centrado y gráficos ASCII, combinó aspectos técnicos y creatividad visual, mejorando la comprensión del funcionamiento de bajo nivel de un sistema operativo.