

# **Project for Assembler Programming Languages**

Subject: MASM Mini Disassembler

Academic year: 2025/2026

Group: 1

Team: 4

Team members:

Piotr Krupiński – pk312207@student.polsl.pl

Jeremi Szczotka – [js312226@student.polsl.pl](mailto:js312226@student.polsl.pl)

Szymon Mańka – sm312212@student.polsl.p

# MASM Mini Disassembler (C + MASM Hybrid)

- A 32-bit x86 Disassembler Project for Windows 11
- Hybrid architecture using MASM (assembly) for decoding logic and C for program control, file I/O, and CLI.
- GitHub:  
<https://github.com/piotrek1459/masm-minidis>

# Introduction

---

The MASM Mini Disassembler is a low-level systems project focused on developing a minimal 32-bit x86 disassembler.

It uses:

- MASM for decoding algorithms (opcode tables, ModR/M, SIB handling)

- C for program logic, CLI argument parsing, and I/O

The goal is to produce readable assembly instructions from raw binary data.

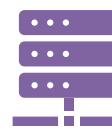
# Why This Project



- Strengthen understanding of x86 architecture and assembly language.



- Combine MASM with C to build a hybrid systems-level tool.



- Practice bitwise operations, data parsing, and control flow logic.



- Create a practical, educational reverse-engineering utility.



- Keep the project compact, maintainable, and extensible.



# Scope and Limitations

- Scope:
  - Reads raw .bin files.
  - Decodes selected 32-bit x86 instructions (MOV, ADD, SUB, PUSH, POP, CALL, JMP, RET, etc.).
  - Displays addresses, bytes, and mnemonics with formatted output.
- Limitations:
  - No PE file parsing.
  - Limited opcode coverage in v1.
  - Decoding only — no instruction execution.

# Example Output

- Input bytes: 55 8B EC 83 EC 10
- Output:
  - 00401000 55 push ebp
  - 00401001 8B EC mov ebp, esp
  - 00401003 83 EC 10 sub esp, 0x10
- Another test:
  - B8 78 56 34 12
  - 00402000 B8 78 56 34 12 mov eax, 0x12345678

# Project Architecture

- The hybrid architecture separates logic and decoding:
  - C handles CLI, file loading, printing, and test harnesses.
  - MASM implements decode\_one(), opcode tables, and ModR/M parsing.
- Structure:
  - masm-minidis/
  - |   └ minidism/ (Visual Studio project)
  - |   |   └ main.c, format.c
  - |   |   └ decoder.asm, tables.asm, util.asm
  - |   |   └ decoder.h, format.h
  - |   └ minidism.sln
  - |   └ samples/ (binary inputs)
  - └ tests/ (validation)

# Implementation Plan



1. Setup hybrid C + MASM build in Visual Studio 2022 (x86).



2. Implement CLI and file I/O in C.



3. Create MASM decode\_one() for opcode recognition.



4. Add ModR/M and SIB support.



5. Implement control flow and relative addressing.



6. Add testing samples and comparison against ndisasm/objdump.



7. Polish output, bounds checks, and --hex option.

# Testing & Validation

- Compare results with ndisasm or objdump.
- Maintain golden binary samples in /samples.
- Store expected disassembly in /tests.
- Fuzz test random inputs to ensure no crashes.
- Validate correctness by comparing instruction-by-instruction output.

## Future Extensions

- x64 support (ml64 + Win64 calling conventions).
- Prefix decoding (0x66, 0x67, 0xF3).
- PE .text section extraction and disassembly.
- Colorized / symbolized console output.
- Two-byte opcode coverage (0x0F prefix group).
- Modular plugin for adding new instruction sets.

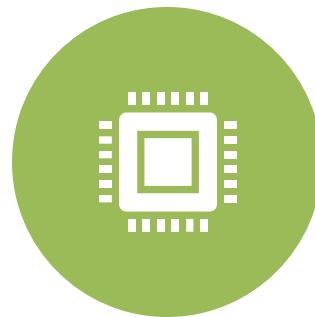
# How to Clone & Run

- 1. Clone the repository:
- git clone  
<https://github.com/piotrek1459/masm-minidis.git>
- cd masm-minidis
- 2. Open minidism/minidism.sln in Visual Studio 2022.
- 3. Install "Desktop development with C++" workload.
- 4. Enable MASM: Project → Build Customizations → check masm.
- 5. Set Platform to x86 (Win32).
- 6. Build → Rebuild Solution.
- 7. Create test.bin with bytes 90 C3 and run:
- minidism.exe -i test.bin -a 0x401000 --hex

# Conclusion



THE MASM MINI DISASSEMBLER  
DEMONSTRATES PRACTICAL  
INTEGRATION BETWEEN ASSEMBLY AND  
C.



IT SOLIDIFIES LOW-LEVEL  
UNDERSTANDING OF CPU ARCHITECTURE  
AND BINARY CODE INTERPRETATION,  
PRODUCING A WORKING TOOL FOR  
ANALYZING MACHINE CODE.



THIS PROJECT PROVIDES AN  
EDUCATIONAL, REALISTIC FOUNDATION  
FOR REVERSE-ENGINEERING AND  
SYSTEMS DEVELOPMENT.