Mini Disassembler (Partial) – Documentation

1. Project Overview

Project Name: Mini Disassembler (Partial) Language: C++ Platform: Windows/Linux Purpose: The Mini Disassembler reads a binary file and decodes a subset of x86 machine instructions into human-readable assembly mnemonics. This project is educational and demonstrates binary parsing, opcode decoding, and reverse engineering fundamentals.

1. Features

2. Reads a binary file (.bin) into memory.

3. Scans bytes sequentially and decodes instructions.
4. Supports a partial set of x86 instructions:
5. MOV EAX, imm32
6. MOV EBX, imm32
7. ADD r/m32, r32
8. SUB r/m32, r32
9. JMP rel32
10. Prints instruction offset and mnemonic to console.

11. Labels unknown bytes as UNKNOWN.

12. Project Structure

```
MiniDisassembler/
├── disassembler.hpp   # Header: structs, enum, class declaration
├── disassembler.cpp   # Implementation: parsing, decoding, opcode table
└── demo.cpp           # Example usage: loads binary and prints instructions
```

1. Key Concepts Used

2. RAII / Resource Management – Efficient memory handling using std::vector.

3. Enums and Structs – InstrType and Instruction hold instruction metadata.
4. Vectors – Store file bytes and decoded instructions dynamically.
5. Modular Design – Header and implementation separation.

6. Opcode Table – Partial mapping of binary opcodes to mnemonics.

7. File Descriptions

5.1 disassembler.hpp - Defines: - InstrType enum - Instruction struct - Disassembler class declaration

5.2 disassembler.cpp - Implements: - Constructor: reads binary into memory - decodeInstruction(): decodes instruction from opcode - parse(): iterates through all bytes - printInstructions(): prints instructions to console - Contains partial opcode table

5.3 demo.cpp - Example main() program: - Loads test.bin - Calls parse() - Prints decoded instructions

1. Compilation Instructions

Using G++ / MinGW:

1. Open terminal/command prompt in project folder.
2. Compile:

```
g++ -std=c++17 disassembler.cpp demo.cpp -o disassembler
```

1. Run:

```
disassembler.exe   # Windows CMD
.\disassembler.exe # Windows PowerShell
```

Make sure test.bin exists in the same folder.

1. Sample Output

Assuming test.bin contains:

```
B8 BB 01 29 E9
```

Output:

```
0 : MOV EAX, imm32
1 : MOV EBX, imm32
2 : ADD r/m32, r32
3 : SUB r/m32, r32
4 : JMP rel32
```

- Offsets are in hexadecimal
- Only bytes matching opcode table are decoded; others appear as UNKNOWN
- Limitations
- Supports only a small subset of x86 instructions.

• Does not decode operands, ModRM bytes, or 64-bit instructions.
• Cannot interpret instruction prefixes or complex instructions.

• Primarily educational.

• Future Extensions

• Add full x86/x64 instruction support

• Decode operands (registers, immediate values, memory addresses)
• Support ModRM/SIB bytes
• Add GUI interface for visualization

• Export disassembly to text or CSV

• Short Description (For README)

The Mini Disassembler is a partial x86 disassembler written in C++. It reads a binary file and converts recognized machine code bytes into human-readable assembly instructions. This project demonstrates binary parsing, opcode mapping, and C++ concepts such as enums, structs, vectors, and RAII.