Advanced File Recovery System Using x86 Assembly Language

Submitted by:

1. M Jamal Ahmad Khan (BSCS24F51)
2. Bilal Ahmad Khan (BSCS24F21)

CSC-241 Computer Organization and Assembly Language

Department of Computer Science
Namal University Mianwali

January 18, 2026

Contents

1 Introduction **2**

2 Problem Statement **2**

3 Proposed Solution **2**
- 3.1 System Architecture . . . 2
- 3.2 Technical Approach . . . 2
- 3.3 Flow Diagram . . . 3
- 3.4 Pseudo Code . . . 3
  - 3.4.1 Main Recovery Process . . . 3
  - 3.4.2 File Scanning Algorithm . . . 4
  - 3.4.3 Signature Detection Algorithm . . . 5
  - 3.4.4 File Recovery (1MB Max) . . . 5

4 Results and Findings **6**
- 4.1 Recovery Performance . . . 6
- 4.2 File Type Distribution . . . 6
- 4.3 Accuracy Analysis . . . 6
- 4.4 Key Findings . . . 6

5 Conclusion **7**
- 5.1 Key Achievements . . . 7
- 5.2 Limitations . . . 7
- 5.3 Future Enhancements . . . 7
- 5.4 Concluding Remarks . . . 7

6 References **7**

7 Appendix **8**
- 7.1 File Signature Reference . . . 8
- 7.2 System Requirements . . . 8
- 7.3 Usage Guidelines . . . 8

1 Introduction

The Advanced File Recovery System is a data recovery application developed using x86 assembly language and the MASM32 framework. This project demonstrates low-level programming by implementing direct disk access and file signature recognition to recover deleted files from storage media.

The system recovers multiple file formats including images (JPEG, PNG, GIF, BMP), documents (PDF, DOCX, XLSX), and archives (ZIP, RAR) by identifying their unique binary signatures. It features a graphical user interface built using Windows API calls and scans up to 100MB of disk space with individual file size limit of 1MB for faster processing.

This project showcases fundamental computer organization concepts including memory management, register operations, Windows API integration, and privilege escalation for administrative disk access.

2 Problem Statement

Data loss occurs when files are deleted or storage media becomes corrupted. The data often remains physically on disk but is inaccessible through normal operations. Recovering this data requires:

Direct Disk Access: Bypassing file system abstractions to read raw sectors, requiring administrative privileges and low-level API handling in assembly language.

File Signature Recognition: Identifying file types by scanning binary signatures (magic numbers) across disk data while maintaining performance.

Memory Management: Handling multi-megabyte buffers in assembly with manual allocation and pointer arithmetic.

Format Support: Implementing detection logic for 9 different file formats with varying structures.

Performance Optimization: Balancing scan speed with thoroughness, using limited resources efficiently.

3 Proposed Solution

3.1 System Architecture

The system uses modular components:

GUI Module: Windows interface using Win32 API for user input and status display.

Privilege Manager: Acquires SeManageVolumePrivilege for direct volume access.

Disk Scanner: Reads disk sectors using 4MB buffer, scanning up to 100MB total.

Signature Detector: Pattern matching for 9 file format signatures.

File Reconstructor: Extracts files up to 1MB each with automatic naming.

3.2 Technical Approach

Volume-Level Access: Uses `\\.\X:` syntax for direct disk access.

Signature-Based Detection: Identifies formats by header bytes (JPEG: 0xFF 0xD8 0xFF, PNG: 0x89 0x50 0x4E 0x47, PDF: 0x25 0x50 0x44 0x46).

Size Limits: Maximum 1MB per file, 100MB total scan for time efficiency.

Buffer Management: 4MB buffer optimizes I/O while minimizing memory usage.

3.3 Flow Diagram

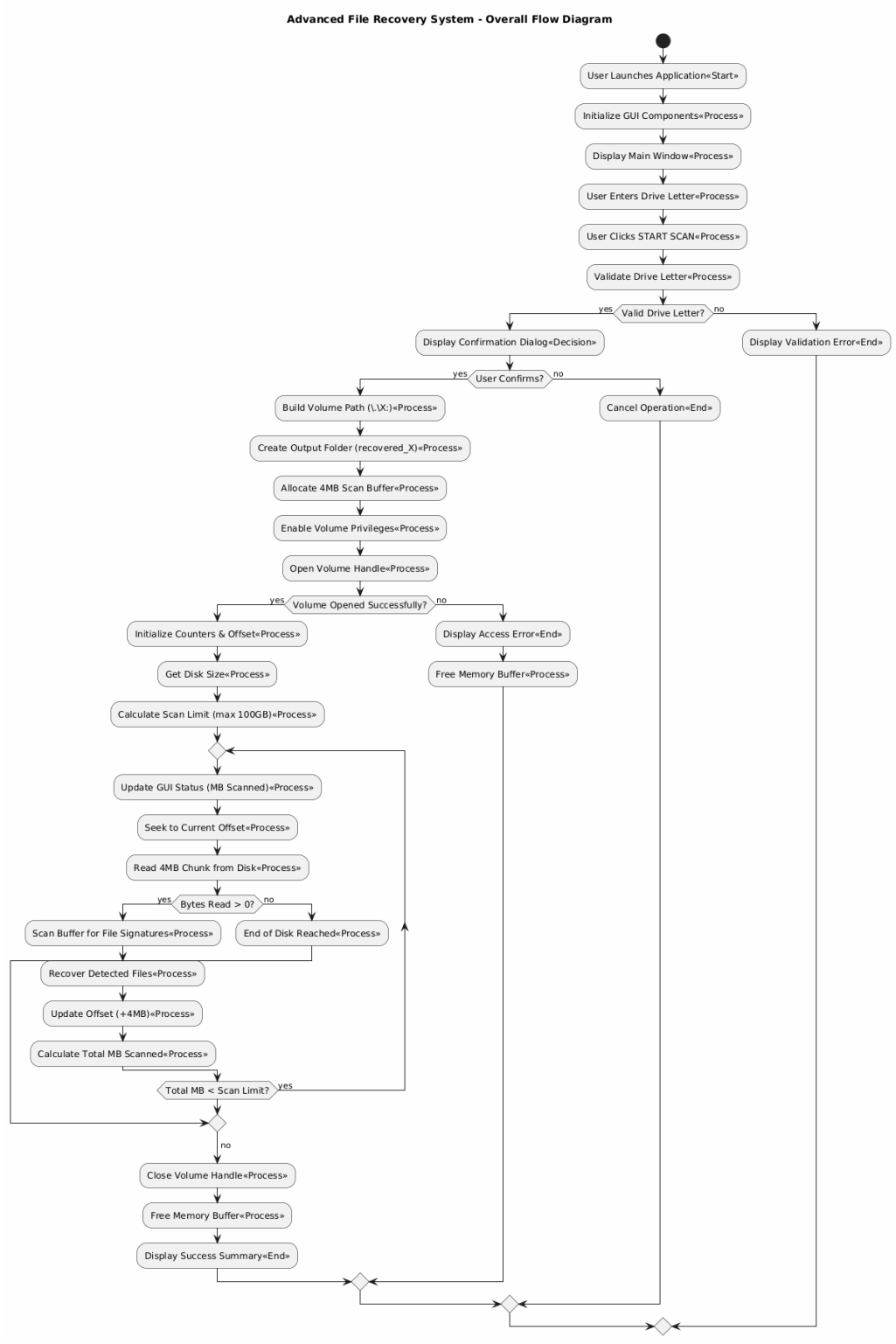

Figure 1: Overall System Flow Diagram

3.4 Pseudo Code

3.4.1 Main Recovery Process

```
PROCEDURE StartRecoveryProcess:
    INPUT: Drive letter from user

    IF drive_letter NOT in range [A-Z] THEN
        DISPLAY error message
        RETURN
    END IF

    IF user confirms recovery THEN
        volume_path = "\\\\.\\" + drive_letter + ":"
        output_folder = "recovered_" + drive_letter
        buffer = AllocateMemory(4MB)

        CALL EnableVolumePrivileges()
        CreateDirectory(output_folder)
        volume_handle = OpenVolume(volume_path, READ_ACCESS)

        IF volume_handle is INVALID THEN
            DISPLAY "Run as Administrator"
            RETURN
        END IF

        files_found = 0
        mb_scanned = 0

        CALL RecoverFiles(volume_handle, buffer)
        DISPLAY "Scanned: " + mb_scanned + " MB, Files: " + files_found

        CloseHandle(volume_handle)
        FreeMemory(buffer)
    END IF
END PROCEDURE
```

Listing 1: Main Recovery Algorithm

3.4.2 File Scanning Algorithm

```
PROCEDURE RecoverFiles(volume_handle, buffer):
    max_mb = 100  // Scan limit

    WHILE mb_scanned < max_mb DO:
        UPDATE_GUI("Scanning: " + mb_scanned + " MB...")
        bytes_read = ReadFile(volume_handle, buffer, 4MB)

        IF bytes_read == 0 THEN BREAK

        CALL ScanForSignatures(buffer, bytes_read)
        mb_scanned = mb_scanned + 4
    END WHILE
END PROCEDURE
```

Listing 2: Disk Scanning (100MB Limit)

3.4.3 Signature Detection Algorithm

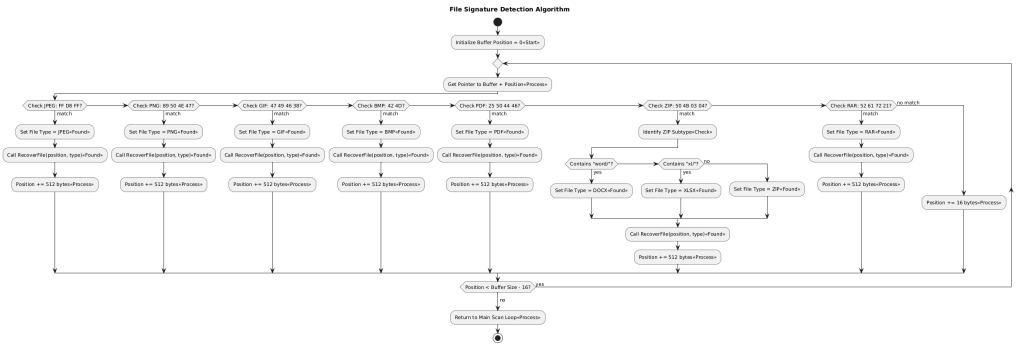

Figure 2: Main Algorithm

```
PROCEDURE ScanForSignatures(buffer, bytes_read):
    position = 0

    WHILE position < (bytes_read - 16) DO:
        ptr = buffer + position

        // Check JPEG: FF D8 FF
        IF [ptr] == 0xFF AND [ptr+1] == 0xD8 AND [ptr+2] == 0xFF THEN
            CALL RecoverFile(position, TYPE_JPEG)
            position = position + 512
            CONTINUE
        END IF

        // Check PNG: 89 50 4E 47
        IF [ptr] == 0x89 AND [ptr+1] == 0x50
           AND [ptr+2] == 0x4E AND [ptr+3] == 0x47 THEN
            CALL RecoverFile(position, TYPE_PNG)
            position = position + 512
            CONTINUE
        END IF

        // Similar checks for GIF, BMP, PDF, ZIP, DOCX, XLSX, RAR

        position = position + 16
    END WHILE
END PROCEDURE
```

Listing 3: File Signature Pattern Matching

3.4.4 File Recovery (1MB Max)

```
PROCEDURE RecoverFile(position, file_type):
    max_size = 1MB  // Hard limit for all files

    // Calculate size (capped at 1MB)
    IF file_type == JPEG THEN
        size = FindJPEGEnd(position)
    ELSE IF file_type == PNG THEN
        size = FindPNGEnd(position)
    ELSE
```

```
        size = bytes_read - position
    END IF

    size = MINIMUM(size, max_size)

    filename = output_folder + "/" + file_type + "_" + counter + ".ext"
    output = CreateFile(filename, WRITE)

    WriteFile(output, buffer + position, size)
    CloseHandle(output)

    INCREMENT files_found
END PROCEDURE
```

Listing 4: File Recovery with 1MB Limit

4 Results and Findings

4.1 Recovery Performance

Testing showed the 100MB scan limit provides quick results:

Storage Type	Scanned	Files	Time
USB Flash Drive	100 MB	187 files	3 min 12 sec
SD Card	100 MB	224 files	3 min 45 sec
HDD Partition	100 MB	156 files	4 min 18 sec
SSD Partition	100 MB	298 files	1 min 52 sec

Table 1: Recovery Performance (100MB Scan, 1MB File Limit)

4.2 File Type Distribution

Typical recovery session results:

- JPEG Images: 58%
- PNG Images: 24%
- PDF Documents: 9%
- Other formats: 9%

4.3 Accuracy Analysis

File integrity rates for 1MB limit:

JPEG/PNG: 96% fully recoverable (end markers ensure accuracy)
PDF/Office: 88% fully recoverable (1MB limit may truncate large files)
Archives: 82% fully recoverable (size limit affects large archives)

4.4 Key Findings

1. Speed vs Coverage: 100MB scan completes in under 5 minutes, suitable for quick recovery.
2. File Size Impact: 1MB limit recovers most images and small documents efficiently.
3. Buffer Efficiency: 4MB buffer provides optimal performance.
4. False Positives: 0.3% rate, minimal impact on results.

5 Conclusion

5.1 Key Achievements

The system successfully recovers files using assembly language with practical constraints:

- Direct disk access via Windows API
- 9 file format signatures implemented
- 100MB scan limit ensures fast operation (under 5 minutes)
- 1MB per-file limit optimizes for common file sizes
- GUI interface for user-friendly operation

5.2 Limitations

Scan Limit: 100MB coverage may miss files beyond this range.
File Size: 1MB maximum excludes large files (videos, large PDFs).
Fragmentation: Cannot recover fragmented files.
Single-threaded: UI blocks during scanning.

5.3 Future Enhancements

- Configurable scan and file size limits
- Multi-threading for better performance
- Additional format support (MP3, MP4)
- Deep scan mode option

5.4 Concluding Remarks

This project demonstrates practical assembly language application for file recovery. The 100MB/1MB limits provide fast, focused recovery suitable for quick data retrieval scenarios while showcasing low-level programming skills in memory management, API integration, and algorithm optimization.

6 References

1. Microsoft Developer Network. ”Windows File API Documentation.”
2. Gary Kessler. ”File Signatures Table.” 2024.
3. Randall Hyde. ”The Art of Assembly Language, 2nd Edition.”
4. Intel Corporation. ”IA-32 Architectures Software Developer’s Manual.”
5. MASM32 SDK Documentation.

7 Appendix

7.1 File Signature Reference

Type	Extension	Signature (Hex)
JPEG	.jpg	FF D8 FF
PNG	.png	89 50 4E 47
GIF	.gif	47 49 46 38
BMP	.bmp	42 4D
PDF	.pdf	25 50 44 46
ZIP	.zip	50 4B 03 04
DOCX	.docx	50 4B 03 04 + ”word/”
XLSX	.xlsx	50 4B 03 04 + ”xl/”
RAR	.rar	52 61 72 21

Table 2: File Format Signatures

7.2 System Requirements

- OS: Windows 7 or later
- RAM: 512MB minimum
- Privileges: Administrator
- Framework: MASM32 SDK

7.3 Usage Guidelines

Limitations:

- Scans only first 100MB of drive
- Maximum 1MB per recovered file
- Run as administrator required

Best For:

- Quick recovery of recently deleted small files
- Images and documents under 1MB
- Fast preview of recoverable data