Assembly Solitaire

x86 Assembly Language · Visual Studio IDE · Team Project

By Logan Pride

Course Project | x86 Assembly | Card Encoding | Stack Operations | Team Collab | Irvine32

PROJECT OVERVIEW

Overview

This is a team-built Solitaire-style card game written entirely in x86 Assembly. The game runs in a Visual Studio console and implements the full card game lifecycle: shuffling, drawing, moving cards between piles, validating moves, and detecting a win condition with zero high-level language abstractions.

Every card is represented as a 32-bit hex value: the high byte encodes the suit (spades, clubs, hearts, diamonds) and the low byte encodes the rank (Ace through King). The game manages 7 pile stacks, 4 suit stacks, 1 draw stack, and the deck — all controlled through manual stack and memory operations. ANSI color codes render suits in the console, with hearts and diamonds in red and spades and clubs in dark.

MY CONTRIBUTION

My Contribution

I was a contributing author for functions that dealt with encoding scheme, data structures, and fundamental operations the entire team built on top of. Key contributions I built and included:

- Designed the card encoding scheme: all 52 cards defined as 32-bit hex values (example: 0x0301 is a 3 of spades). The high byte encodes the suit, and the low byte encodes the rank.
- Implemented all suits and ranks to be displayed as strings, including ANSI color escape sequences to render hearts/diamonds in red and spades/clubs in dark in the Visual Studio console.
- Contributed to the function of the put and take procedures, the push/pop stack operations that move cards between all 11 stack slots. These are the fundamental data structure operations the game runs on.
- Wrote the input validation logic inside the await procedure; the .if eax < 0 || eax > 9 guard that ensures valid source/destination selection across all stack options.
- Contributed to checkWinCondition that will detect when all 4 suit stacks are complete (top card at rank index 12), triggering the win screen.
- Contributed encoding helper procedures used by teammates throughout the project for card lookup, display, and move validation.
- Debugged low-level memory issues using Visual Studio's assembly debugger. Thus, the debugger is tracing register values and catching off-by-one addressing errors.

TECHNICAL HIGHLIGHTS

Technical Highlights

Language	x86 Assembly (MASM syntax) with Irvine32 library
IDE	Microsoft Visual Studio

Encoding	52 cards as 32-bit hex: high byte = suit, low byte = rank
Game Structure	7 pile stacks · 4 suit stacks · 1 draw stack · 1 deck
Key Concepts	Bitwise ops, stack frames, subroutines, memory addressing, ANSI output
Move Validation	checkRankPile (alternating color) · checkRankSuit (same suit ascending)
Win Condition	All 4 suit stacks at rank index 12 (King)
Testing	Step-through debugging with VS register/memory watch windows

CODE SNIPPET

Code Sample — Card Encoding Subroutine

The full project source spans over 800 lines of x86 Assembly. The snippet below shows a sample of the card data definitions I authored: the encoding scheme, display strings, and ANSI color codes that the rest of the game is built on. Full source available upon request.

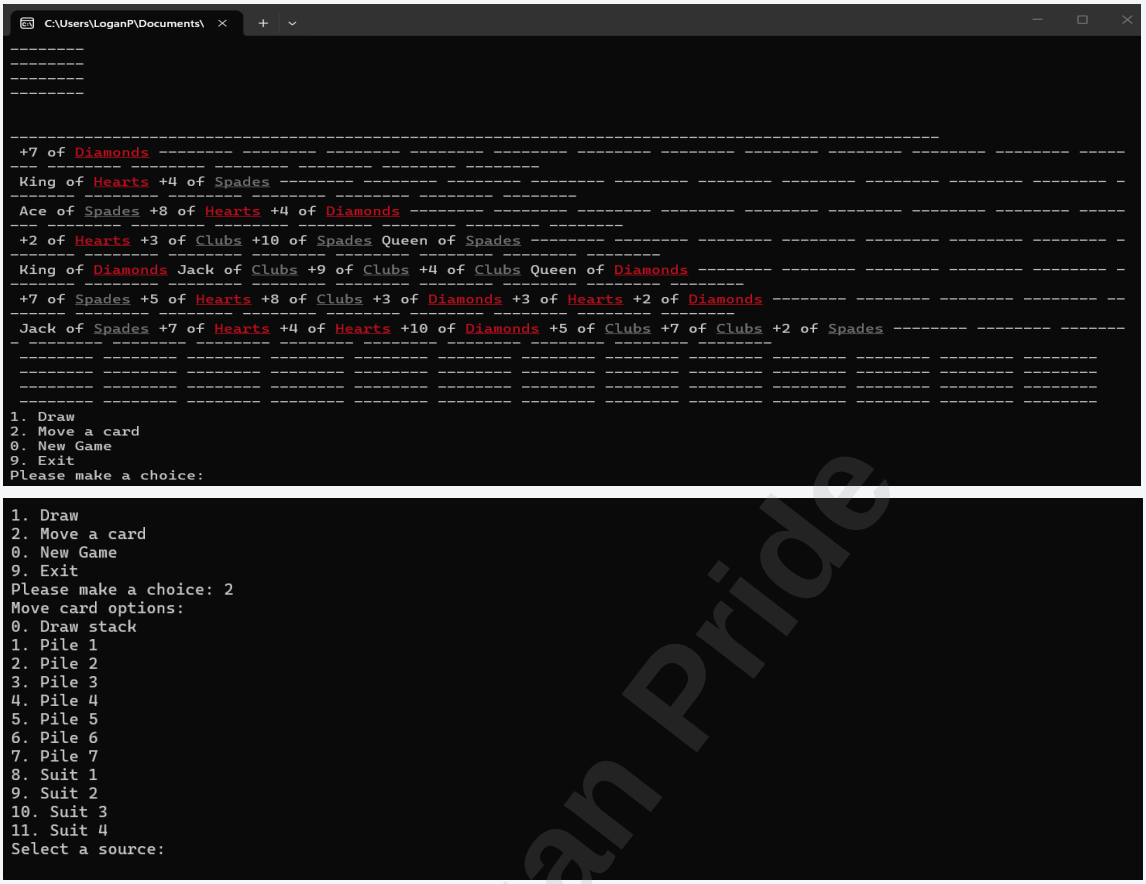
```
;-----Card variables-----
cards   DD 0301h,0302h,0303h,0304h,0305h,0306h,0307h,0308h,0309h,030Ah,030Bh,030Ch,030Dh
        DD 0201h,0202h,0203h,0204h,0205h,0206h,0207h,0208h,0209h,020Ah,020Bh,020Ch,020Dh
        DD 0101h,0102h,0103h,0104h,0105h,0106h,0107h,0108h,0109h,010Ah,010Bh,010Ch,010Dh
        DD 0001h,0002h,0003h,0004h,0005h,0006h,0007h,0008h,0009h,000Ah,000Bh,000Ch,000Dh
noCard BYTE "--------", 0
spadeString BYTE " of ",1Bh, "[90;4mSpades", 1Bh,"[0m",0
heartString BYTE " of ",1Bh, "[31;4mHearts", 1Bh,"[0m",0
clubString BYTE " of ",1Bh, "[90;4mClubs", 1Bh,"[0m",0
diamondString BYTE " of ",1Bh, "[31;4mDiamonds", 1Bh,"[0m",0
aceString BYTE "Ace",0
jackString BYTE "Jack",0
queenString BYTE "Queen",0
kingString BYTE "King",0
;... 800+ lines of procedure follow
```

SCREENSHOT

Game in Action

Screenshot 1 (top): *Game board displaying the 7 card piles with ANSI color-coded suits — Hearts and Diamonds in red, Spades and Clubs in grey.*

Screenshot 2 (bottom): *Move selection menu showing all valid source options: draw stack, piles 1–7, and suit stacks 1–4.*



Visual Studio console output — Solitaire game

TAKEAWAYS

What I Learned

- Designing a binary encoding scheme from scratch; the card hex format taught me how compactly data can be represented at the register level
- How to architect shared subroutines that teammates can depend on; writing put, take, and the card encoding procedures meant owning the interfaces the entire team built against.
- Deep stack discipline: managing register state across a complex call chain of subroutines without corrupting the game loop required precise push/pop bookkeeping.
- Effective use of Visual Studio's low-level debugger to trace register values, inspect memory addresses, and catch off-by-one errors in array indexing.
- Collaborating on a shared codebase without version control, coordinating via code conventions, clear naming, and documentation across a team.

Full source code available upon request — loganpride525@email.com