University of Houston

College of Technology

Computer Engineering Technology

ELET 3405

Microprocessor
Architecture

Final Project:
PACMAN NASM

Report written by:

1. Enlai Yii - 2064210

Introductions

The idea to recreate the classic Pac-Man game in assembly language arose from a desire to push the boundaries of technical expertise. Inspired by a YouTube video showcasing a MASM-based implementation, this project aimed to adapt the game for macOS using NASM. The motivation stemmed from the challenge of working in NASM on macOS, a platform without native support for DOS-based assembly programs. By embracing these challenges, the project became an opportunity to learn assembly programming from the ground up and better understand low-level interactions between hardware and software.

Overview of the Game Logic and Mechanics

The core mechanics of the NASM Pac-Man game remain faithful to the original. Players control Pac-Man, navigating a maze to consume dots while avoiding ghosts. Key game elements include collision detection, animations, and power-up effects. Movement logic ensures that user inputs translate into valid directional shifts, considering maze walls and boundaries. Ghosts roam the maze with distinct behavioral patterns, alternating between chase and scatter modes to increase difficulty. The game implements a scoring system where consuming dots, power pellets, and ghosts contribute to the player's total score, and losing all lives ends the game. These mechanics are presented through pixel-based visuals rendered on the macOS platform, maintaining the charm of classic arcade graphics.

Approach to the Implementation in NASM

One of the first steps in the transition was adapting MASM's Intel-style syntax to NASM's stricter, more explicit format. NASM requires meticulous handling of labels, registers, and memory references. MASM's PROC/ENDP directives were replaced by NASM's modular functions, such as PAC_MOVE, to improve clarity and maintainability. Each NASM function incorporated logic from MASM, including procedures for movement, dot consumption, and collision checks.

The most significant challenge was replacing MASM's DOS-based graphics routines with macOS-compatible calls. While MASM relied on interrupts and BIOS calls, NASM leveraged macOS graphics libraries. This included rewriting image rendering, animation, and screen refresh operations to utilize macOS APIs. These changes were critical for maintaining the visual fidelity of the game while ensuring it ran natively on macOS.

Memory management required restructuring due to differences in how MASM and NASM handle data segments. Variables like Pac-Man's coordinates and game state were explicitly defined in NASM's .data section. Adjustments were also made for the macOS memory model, requiring precise stack and heap allocation to avoid conflicts and ensure smooth gameplay.

One of the most daunting aspects was learning NASM from scratch while transitioning from the MASM implementation. Understanding NASM's syntax and macOS-specific constraints required extensive research and practice. Additionally, macOS lacked direct support for low-level DOS-based assembly instructions, necessitating creative solutions for graphics and

sound. This was overcome by integrating macOS APIs and building custom functions to replicate the original game's features.
Hardware-specific limitations also played a role. Understanding the architecture and behavior of the target machine was vital to ensuring efficient memory usage and avoiding performance bottlenecks. Debugging tools like GDB and NASM's built-in diagnostics proved invaluable in identifying and resolving these issues.

The NASM implementation introduced several enhancements over the MASM version. These included improved animations, optimized collision detection, and a modernized graphical interface using macOS libraries. Additionally, the game incorporated smoother transitions between game states, such as pausing and resuming gameplay. The implementation also opened possibilities for future expansions, such as adding new ghost behaviors or multiplayer functionality.

Conclusion

This project was a journey of technical growth, combining classic game design principles with modern assembly programming challenges. By adapting Pac-Man from MASM to NASM for macOS, the project highlighted the intricacies of low-level programming and cross-platform development. The experience provided deeper insights into assembly language, hardware-software interactions, and the creativity required to overcome platform constraints. For anyone interested in exploring this project further, the full source code is available on GitHub.