Project Report

16x16 Color Map Project by

RABIA AMIR

Executive Summary

This project aims to create a 16x16 color map using Assembly Language. The primary objective is to utilize all possible colors in a text mode environment and to display them on the screen in a structured manner. The project demonstrates the use of Assembly Language for controlling video modes and text attributes on a legacy DOS-based system. Key outcomes include a functional program that successfully prints a 16x16 grid of characters, each with a different color.

TABLE OF CONTENTS

1. Introduction

The 16x16 Color Map Project represents a significant exploration into the capabilities of Assembly Language within the context of legacy DOS-based systems. This project aims to leverage the intricate control offered by Assembly Language to create a visually compelling 16x16 grid of characters, each uniquely colored, in a text mode environment. By doing so, it not only demonstrates technical proficiency but also serves as a testament to the creative application of low-level programming languages in graphical representation.

2. Literature Review

The literature review examines previous research and projects related to the use of Assembly Language for graphical displays in text mode environments. Key studies include foundational texts on Assembly Language programming, documentation on the use of BIOS interrupts (specifically INT 10h) for video services, and scholarly articles on the optimization of low-level programming for hardware control. Relevant projects analyzed include early DOS-based graphical programs and color manipulation techniques in legacy systems. This review provides a comprehensive understanding of the challenges and solutions in managing video modes and text attributes, which informed the methodology and implementation of the 16x16 color map project.

3. Methodology

The methodology section outlines the structured approach taken to design, develop, and implement the 16x16 color map project using Assembly Language.

3.1 Project Design:

The project was designed to display a 16x16 grid of characters, each with a unique color, in a text mode environment. The design involved planning the overall structure of the code, including setting the video mode, initializing variables, and implementing loops for row and column traversal.

3.2 Tools and Technologies:

Key tools and technologies used include:

- Assembly Language for low-level programming.
- An emulator such as DOSBox to run and test the Assembly code.
- A text editor for writing the Assembly code.

3.3 Data Collection:

Data collection was not a primary component of this project, as it focused on code implementation and execution.

3.4 Implementation Approach:

The implementation approach involved several steps:

 Initial Setup: Setting the video mode to 80x25 text mode and disabling text blinking using BIOS interrupts.
 Initialization: Initializing variables for cursor position and color attributes.
 Main Loop: Implementing loops to traverse through rows and columns, setting cursor positions, and printing characters with different colors.
 Finalization: Setting the cursor to a specific position and waiting for a key press to end the program.

4. Assembly Language Code Implementation

4.1 Task/Problem Description:

The task involves writing an Assembly Language program to display a 16x16 grid of characters, each with a unique color, on the screen in text mode.

4.2 Algorithm Overview:

The main algorithm for the program involves the following steps:

1. **Set Video Mode**: Use BIOS interrupt INT 10h to set the video mode to 80x25 text mode.
2. **Blinking Text**: Use BIOS interrupt INT 10h to disable text blinking to allow for more color attributes.
3. **Initialize Variables**: Set initial values for cursor position (rows and columns) and color attributes.
4. **Loop Through Rows and Columns**: Implement nested loops to traverse each row and column of the 16x16 grid.
5. **Set Cursor Position and Print Character**: Use BIOS interrupts to set the cursor position and print a character with the current color attribute.
6. **Change Color Attribute**: Increment the color attribute for each character printed.
7. **Wait for Key Press**: After completing the grid, set the cursor to a specific position and wait for a key press to terminate the program.

4.3 Code Structure:

The structure of the code is divided into several main sections:

1. **Program Initialization**:

   . Set Video Mode:

   mov ax, 3

   int 10h

   . Disable Blinking Text:

   mov ax, 1003h

   mov bx, 0

   int 10h

2. **Variable Initialization**:
   . Initialize cursor position and color attributes:
   mov dl, 0
   mov dh, 0
   mov bl, 0
3. **Main Loop**:
   . Loop through rows:
   next_row:
   inc dh
   cmp dh, 16
   je stop_print
   mov dl, 0

. Loop through columns and print characters:

```
next_char:
; Set cursor position at (dl, dh)
mov ah, 02h
int 10h

mov al, 'r'
mov bh, 0
mov cx, 1
mov ah, 09h
int 10h

inc bl
inc dl
cmp dl, 16
je next_row
jmp next_char`
```

4. **Finalization**:

. Set cursor to a specific position and wait for a key press:

```
stop_print:
; Set cursor position at (dl, dh)
mov dl, 10
mov dh, 5
mov ah, 02h
int 10h

; Test of teletype output
mov al, 'x'
```

```
mov ah, 0eh
int 10h

; Wait for any key press
mov ah, 0
int 16h

ret
```

4.4 *Key Functions and Variables:*

- **mov ax, 3**: Set the video mode to 80x25 text mode.
- **mov ax, 1003h**: Disable blinking text.
- **mov dl, 0**: Initialize the current column.
- **mov dh, 0**: Initialize the current row.
- **mov bl, 0**: Initialize the current color attribute.

4.5 *Code Snippets:*

assembly
Copy code

```
name "16x16 color map"
org 100h

; Set video mode:
mov ax, 3
int 10h
```

```
; Disable blinking:
mov ax, 1003h
mov bx, 0
int 10h

; Initialize position and color
mov dl, 0
mov dh, 0
mov bl, 0

jmp next_char

; Main loop to print characters
next_row:
inc dh
cmp dh, 16
je stop_print
mov dl, 0

next_char:
; Set cursor position at (dl, dh)
mov ah, 02h
int 10h

mov al, 'r'
mov bh, 0
mov cx, 1
mov ah, 09h
int 10h
```

```
inc bl
inc dl
cmp dl, 16
je next_row
jmp next_char

stop_print:
; Set cursor position at (dl, dh)
mov dl, 10
mov dh, 5
mov ah, 02h
int 10h

; Test of teletype output
mov al, 'x'
mov ah, 0eh
int 10h

; Wait for any key press
mov ah, 0
int 16h

ret
`
```

5. Results and Discussion

5.1 Code Output:

The program outputs a 16x16 grid of the character 'r', with each character having a unique color. Starting from the top-left corner (0, 0) to the bottom-right corner (15, 15), each cell in the grid displays the character 'r' in a different color. The colors change incrementally, covering the entire range of available text mode colors. After displaying the grid, the program moves the cursor to a specified position and waits for a key press, ensuring the display remains visible until user interaction.

5.2 Performance Metrics:

The program's performance is evaluated based on its speed and efficiency in displaying the 16x16 color grid. Key metrics include:

- **Execution Time**: The program runs quickly, with minimal delay in setting the video mode, initializing variables, and displaying the grid.
- **Memory Usage**: The program has a low memory footprint, typical of Assembly Language programs, efficiently utilizing system resources.
- **Responsiveness**: The program responds promptly to key presses, demonstrating effective handling of user input.

5.3 Challenges Encountered:

- **Understanding Video Modes and Text Attributes**: Grasping the intricacies of BIOS interrupts (INT 10h) for setting video modes and controlling text attributes required thorough study and experimentation.
- **Managing Cursor Position and Color Changes**: Efficiently handling cursor positioning and incrementing color attributes within nested loops presented a challenge, necessitating precise control flow and variable management to ensure correct grid display.

5.4 Solutions implemented:

- Thoroughly studied the INT 10h interrupts and their functions.
- Used loops and conditional jumps to manage the 16x16 grid display.

6. Conclusion

6.1 Summary of Findings:

- The project successfully displayed a 16x16 grid of characters, each with a unique color, using Assembly Language. It demonstrated effective manipulation of video modes and text attributes on a DOS-based system.

6.2 Project Achievements

- Key achievements include the successful implementation of the color grid, efficient use of Assembly Language for low-level programming, and the practical application of BIOS interrupts for video control.

6.3 Lessons Learned

- Important lessons learned include a deeper understanding of Assembly Language and BIOS interrupts, improved skills in managing low-level hardware control, and the importance of meticulous planning and debugging in Assembly programming.

8. Future Work

7.1 Opportunities for Improvement:

- Enhance the user interface with more diverse characters and interactive features.
- Optimize the code for faster execution and reduced memory usage.

7.2 Potential Extensions

- Expand the grid to display more colors or different patterns.
- Add functionality for dynamic color changes and user-controlled input for real-time interaction.

8. Project Management

8.1 Gantt Chart:

TASK	WEEK 1	WEEK 2	WEEK 3
Project Planning	x	x	
Tools Setup		x	
Coding		x	x
Testing			x
Documentation			x

8.2 Timeline Overview:

- **Week 1**: Project planning and initial setup.
- **Week 2**: Tools setup and coding phase.
- **Week 3**: Testing, final documentation, and project completion.

9 Video Presentation:

9.1 Overview of Presentation:

This Assembly Language program sets up a DOS-based system to display a 16x16 grid of characters, each with a unique color. It begins by configuring the video mode to text mode with 80 columns, 25 rows, and 16 available colors using BIOS interrupt 10h. To ensure uninterrupted color display, it disables blinking text through interrupt 10h with function 1003h. The program then initializes the starting position at column 0, row 0, and sets the initial color attribute to 0. Using nested loops, it iterates through each row and column of the grid, setting the cursor position, printing the character 'r' with the current color attribute, and incrementing the color attribute for each subsequent character. After printing the grid, it positions the cursor at column 10, row 5, and tests the teletype output by printing a single 'x' character using the current color attributes. Finally, the program waits for any key press to terminate execution, showcasing basic Assembly Language capabilities in controlling text attributes and displaying a structured grid of characters with varied colors.

9.2 Key Points Covered:

 Objective: Demonstrating the creation of a 16x16 grid of characters with unique colors on a DOS-based system using Assembly Language.

 Video Mode Setup: Configuring the system to text mode with 80x25 resolution and 16-color palette using BIOS interrupt 10h.

 Disabling Blinking Text: Ensuring all 16 colors are used without blinking distractions through BIOS interrupt 10h with function 1003h.

 Initialization: Starting the grid display at column 0, row 0, and setting the initial color attribute.

 Printing the Grid: Using nested loops to iterate through each row and column, setting cursor positions, printing characters ('r') with varying colors, and managing color attribute increments.

 Teletype Output: Testing functionality by printing an 'x' character at a specified position to verify color settings and cursor management.

 Program Termination: Ending the program execution upon receiving any key press using BIOS interrupt 16h.

10. ***References***

NUMERADE:

https://www.numerade.com/ask/question/video/qexplain-this-code-lines-this-sample-prints-16x16-color-map-it-uses-all-possible-colors-name-colors-org-100h-set-video-mode-text-mode-80x25-16-colors-8-pages-mov-ax-3-int-10h-blinking-disabl-41701/

11 Appendices

 Supplementary Code: Additional segments of Assembly Language code relevant to your project, such as different variations of the grid display or alternative color handling techniques.

 Charts and Diagrams: Visual representations like flowcharts illustrating the program's logic, or diagrams showing memory layouts or data structures used in your implementation.

 Screen Captures: Screenshots or photographs demonstrating the output of your program running on the DOS-based system, showcasing the 16x16 grid with different colors.

 Performance Metrics: If applicable, include performance metrics such as execution time measurements or memory usage analysis during the program's operation.

 Technical Documentation: Any detailed technical documentation or notes that further explain specific aspects of your implementation, BIOS interrupts used, or hardware configurations.

 References to External Resources: Links or citations to external resources that were particularly helpful or influential in your project's development, such as online tutorials, forums, or technical articles.