Write, Compile And Run An Assembly Program For A Phone Book, Using 8086 Emulator Only, That Can (upon

Write, Compile And Run An Assembly Program For A Phone Book, Using 8086 Emulator Only, That Can (upon in the first paragraph.

If you're venturing into low-level programming and want to understand how to develop practical applications such as a phone book, learning to write, compile, and run an assembly program for a phone book using an 8086 emulator is a fantastic starting point. This approach allows you to grasp the fundamentals of assembly language, understand memory management, and see how data is handled at a hardware level. Whether you're a student, hobbyist, or professional looking to deepen your understanding of computer architecture, this guide will walk you through the process step-by-step, focusing solely on the 8086 emulator environment.

Understanding the Basics of Assembly Programming for 8086

Before diving into coding, it's crucial to understand the foundational concepts of assembly language programming on the 8086 processor. Assembly language provides a human-readable way to write instructions that are directly executed by the CPU, making it ideal for understanding hardware operations and creating efficient programs.

What is the 8086 Emulator?

The 8086 emulator is a software tool that mimics the behavior of the Intel 8086 microprocessor. It allows developers to write, assemble, and execute assembly language programs without needing physical hardware. Popular emulators include DOSBox, EMU8086, and NASM, which provide simulated environments for 8086 assembly development.

Key Features of Assembly Language for 8086

- Registers: General-purpose registers (AX, BX, CX, DX), segment registers (CS, DS, SS, ES), and pointer registers (SP, BP, SI, DI).
- Memory management: Direct access to memory addresses for storing data.
- Instruction set: Operations like MOV, ADD, SUB, INT, etc., to perform computations and I/O.
- Interrupts: Used for input/output operations, display, and other system functions.

Designing a Phone Book in Assembly Language

Creating a phone book in assembly involves managing data storage, user input, searching, and display routines. Since assembly programming is low-level, careful planning is essential.

Data Storage Structure

A simple approach is to store entries in memory as pairs: name and phone number.

- Each entry can be stored sequentially in memory.
- Allocate fixed sizes for names and phone numbers for simplicity (e.g., 20 bytes for name, 15 bytes for phone number).
- Maintain a count of total entries to facilitate searches and additions.

User Interaction and Menu

Implement a simple menu that offers options such as:

1. Add a new contact
2. Search for a contact by name
3. Display all contacts
4. Exit

Each option triggers specific routines in the assembly program.

Writing the Assembly Program for the Phone Book

This section covers the core logic and assembly code structure, tailored for the 8086 emulator.

Setting Up the Data Segment

Define data storage:

```assembly
.data
max_entries dw 100 ; Maximum number of entries
entry_count dw 0 ; Current number of entries
name_size db 20 ; Max length for name
phone_size db 15 ; Max length for phone number
entries times 100 (name_size + phone_size + 2) db 0 ; Storage for entries
menu_text db '1. Add Contact', 13, 10, '2. Search Contact', 13, 10, '3. Display All', 13, 10, '4. Exit', 13, 10, '$'
prompt_choice db 'Enter your choice: $'
```

Code for Main Routine and User Menu

Implement main loop:

```assembly
.code
main:
mov ax, @data
mov ds, ax
main_loop:
; Display menu
lea dx, menu_text
mov ah, 09h
int 21h

; Prompt for user choice
lea dx, prompt_choice
mov ah, 09h
int 21h

; Read user input
mov ah, 01h
int 21h
sub al, '0' ; Convert ASCII to number

cmp al, 1
je add_contact
cmp al, 2
je search_contact
cmp al, 3
je display_contacts
cmp al, 4
je exit_program
jmp main_loop
```

Implement routines for adding, searching, and displaying contacts.

Adding a New Contact

```assembly
add_contact:
; Check if max entries reached
mov bx, [entry_count]
cmp bx, max_entries
ja end_add ; if full, exit

; Prompt for name
lea dx, name_prompt
mov ah, 09h
int 21h
; Read name
lea dx, name_buffer
mov ah, 0Ah
mov dx, name_buffer
int 21h

; Prompt for phone
lea dx, phone_prompt
mov ah, 09h
int 21h
; Read phone number
lea dx, phone_buffer
mov ah, 0Ah
mov dx, phone_buffer
int 21h

; Store name and phone in entries
; Calculate address based on entry count
mov si, bx
inc bx
mov [entry_count], bx

; Calculate offset
lea di, entries
mov cx, si
mov dx, (name_size + phone_size + 2) ; size per entry
mul dx
add di, ax

; Copy name
lea si, name_buffer + 2 ; Skip length byte
mov cx, name_buffer + 1
mov ch, 0
mov cl, [si-1] ; name length
rep movsb

; Copy phone
```

```
lea si, phone_buffer + 2
mov cx, phone_buffer + 1
mov ch, 0
mov cl, [si-1] ; phone length
rep movsb

jmp main_loop
end_add:
; Handle full entries case
jmp main_loop
```

Similarly, implement search and display routines with string comparison, iteration over entries, and output routines.

Assembling and Running the Phone Book Assembly Program

Once the source code is written, follow these steps:

Assembling the Code

- Use an assembler compatible with 8086 assembly, such as NASM or MASM.
- Save the code in a `.asm` file.
- Assemble the code with commands like:

```bash
nasm -f bin phonebook.asm -o phonebook.com
```

or

```bash
masm phonebook.asm;
link phonebook.obj;
```

Ensure the output is a `.com` or `.exe` file compatible with the emulator.

Running in the 8086 Emulator

- Launch your emulator (e.g., DOSBox or EMU8086).
- Mount the directory containing your assembled program.
- Run the program by typing its filename, e.g., `phonebook.com`.

- Interact with your phone book program through the menu options.

Conclusion

Developing a phone book in assembly language using an 8086 emulator, though challenging, provides invaluable insights into low-level programming, data management, and system calls. This process enhances understanding of how high-level features are implemented at the hardware level and sharpens problem-solving skills in constrained environments. Remember, starting with simple routines and gradually adding features will make the development process manageable and educational. With patience and practice, you can master writing, compiling, and running practical applications in assembly language, paving the way for advanced systems programming.

Additional Tips and Resources

- Use debugging tools within your emulator to step through code and verify data management.
- Consult the Intel 8086 processor manual for detailed instruction set information.
- Explore online tutorials and communities focused on assembly language programming.
- Experiment with extending your phone book program, such as adding delete functionality or persistent storage.

Embark on your assembly programming journey by creating a functional phone book, and deepen your understanding of how computers operate at the lowest levels!

Frequently Asked Questions

What are the basic steps to write an assembly program for a phone book using the 8086 emulator?

The basic steps include designing the data structure for storing contacts, writing assembly code to input, display, add, delete, and search contacts, assembling the program using an assembler like MASM or TASM, and then running it on the 8086 emulator to test functionality.

How can I compile an assembly phone book program in an 8086 emulator environment?

You compile the assembly source code by using an assembler such as MASM or TASM within the emulator or on your host system. The assembler converts your .asm file into an executable or object

file that can be run in the emulator.

What are the key considerations for writing a phone book program in 8086 assembly?

Key considerations include managing memory efficiently, handling string inputs and outputs, implementing data structures like arrays or linked lists for contacts, and ensuring proper input validation and error handling within the constraints of assembly language.

Can I run a complete phone book program on an 8086 emulator without external tools?

Yes, by writing the entire program in assembly, assembling it with a compatible assembler, and running it within the 8086 emulator, you can execute a complete phone book application without external tools during runtime.

What are the common challenges faced when writing and running assembly phone book programs on 8086 emulator?

Common challenges include managing complex data structures manually, handling user input and display output in assembly, debugging assembly code, and ensuring the program operates within the limited memory and instruction set of the 8086 architecture.

How do I implement 'search' functionality in an assembly phone book program for the 8086 emulator?

Implement search by iterating through stored contact records in memory, comparing input search keys (like name or number) with stored data using string comparison routines, and displaying the matching contact if found.

What tools are recommended for writing and testing assembly phone book programs for 8086?

Recommended tools include MASM or TASM for assembling code, DOSBox or similar 8086 emulators for running programs, and debugging tools like Debug or Turbo Debugger to troubleshoot assembly code.

Are there sample codes or templates available for writing a phone book in 8086 assembly?

Yes, many educational resources and tutorials provide sample assembly programs for simple data management like phone books. These can serve as a starting point, which you can modify and expand based on your requirements.

What are best practices for testing an assembly phone book program in 8086 emulator?

Best practices include writing modular code for easy debugging, thoroughly testing each functionality (add, delete, search, display), validating user inputs, and using step-by-step debugging tools to trace execution and verify data handling.

Additional Resources

Write, Compile And Run An Assembly Program For A Phone Book, Using 8086 Emulator Only, That Can (upon

In a world increasingly driven by high-level languages and sophisticated development environments, the foundational understanding of assembly language remains a vital cornerstone for computer science enthusiasts and professionals alike. The 8086 microprocessor, introduced by Intel in the late 1970s, continues to serve as an educational platform for understanding low-level programming, system architecture, and the inner workings of computer hardware. Today, we delve into a practical yet intellectually stimulating project: crafting a simple phone book application entirely in assembly language, executing it within an 8086 emulator environment. This task, while seemingly modest, encapsulates core programming concepts such as data storage, user input handling, control flow, and output display—all at the assembly level.

This article aims to guide you through the entire process—from writing the assembly code to compiling and executing it—using only an 8086 emulator. Whether you're a student seeking to strengthen your understanding of assembly language or a seasoned programmer exploring historical computing paradigms, this tutorial offers valuable insights into low-level programming techniques and emulator-based development.

Understanding the 8086 Architecture and Emulator Environment

Before diving into code, it’s essential to comprehend the underlying platform: the 8086 microprocessor and the emulator environment.

The 8086 Microprocessor: A Brief Overview

The Intel 8086 is a 16-bit microprocessor with a segmented memory architecture, capable of addressing up to 1MB of memory. Its key features include:

- Registers: AX, BX, CX, DX, SI, DI, BP, SP, and segment registers like CS, DS, ES, SS.
- Segmented Memory Model: Memory addresses are formed by combining a segment register and an offset.

- Instruction Set: Supports a rich set of instructions, including data movement, arithmetic, control flow, string operations, and I/O.

Understanding these features is crucial because assembly programming involves manipulating these registers and memory segments directly.

Choosing the Right Emulator

To develop and test 8086 assembly programs without physical hardware, you need an emulator that simulates the 8086 environment. Popular options include:

- EMU8086: A beginner-friendly DOS-based emulator with integrated assembler and debugger.
- DOSBox: A DOS emulator suitable for running older programs.
- PCem or QEMU: More complex but versatile options.

For this tutorial, EMU8086 is recommended due to its simplicity and built-in features tailored for assembly language learning.

Designing the Phone Book Program in Assembly

Creating a phone book in assembly entails managing data storage, user interaction, and program control flow, all within the constraints of the 8086 architecture.

Core Features of the Program

The program will support:

- Adding Entries: Inputting a name and phone number.
- Viewing Entries: Listing stored contacts.
- Searching Entries: Looking up a contact by name.
- Exiting: Terminating the program gracefully.

Given the limitations and complexity, the initial version will focus on adding and viewing entries, with search functionality as an extension.

Data Storage Strategy

Since assembly language offers no high-level data structures, you'll need to define fixed-size memory regions:

- Name Storage: Allocate a fixed length (e.g., 20 characters) per name.

- Phone Number Storage: Fixed length (e.g., 15 characters).
- Entries Array: A block of memory to hold multiple entries (e.g., 10 contacts).

For example:

- Each entry: 20 bytes (name) + 15 bytes (number) = 35 bytes.
- Total storage: 10 entries 35 bytes = 350 bytes.

This structured approach simplifies data management.

Step-by-Step: Writing the Assembly Code

Creating the assembly program involves several key components:

1. Program Initialization
2. Display Menu and Handle User Choice
3. Input Handling for Adding Entries
4. Display Stored Entries
5. Program Exit Routine
6. Supporting Subroutines

---

1. Program Initialization

Start by setting up the data segment, initializing stack pointers, and preparing the screen:

```assembly
.data
menuMsg db 'Phone Book Menu:$'
addEntryMsg db 'Add New Entry:$'
viewEntriesMsg db 'View Entries:$'
exitMsg db 'Exit Program:$'
nameBuffer db 20 dup('$') ; Buffer for input name
phoneBuffer db 15 dup('$') ; Buffer for input phone number
entries db 10 dup(0) ; Array to store 10 entries
entrySize equ 35 ; Size of each entry

.code
main:
mov ax, @data
mov ds, ax
; Initialize program
call displayMenu
```

---

2. Display Menu and Handle User Choice

Create a loop that displays options and waits for user input:

```assembly
displayMenu:
; Display menu options
mov dx, offset menuMsg
call printString
; Read user input
call readChar
cmp al, '1'
je addEntry
cmp al, '2'
je viewEntries
cmp al, '3'
je exitProgram
jmp displayMenu
```

Implementing `printString` and `readChar` subroutines is critical for interaction.

---

3. Adding a New Entry

When user selects to add an entry:

```assembly
addEntry:
; Prompt for name
mov dx, offset addEntryMsg
call printString
call readString
mov si, offset nameBuffer
; Store name in entries array
mov di, offset entries
call storeEntry
; Prompt for phone number
mov dx, offset 'Enter Phone Number: $' ; You can create similar prompt
call printString
call readString
mov si, offset phoneBuffer
; Store phone number
call storePhone
```

jmp displayMenu
```

Here, `readString` captures user input into buffers, and `storeEntry` copies data into the entries array.

---

## 4. Viewing Stored Entries

Listing all saved contacts:

```assembly
viewEntries:
mov cx, 10 ; Loop through all entries
mov si, 0
view_loop:
; Calculate address of current entry
lea di, [entries + si entrySize]
; Display Name
mov dx, offset 'Name: '
call printString
; Display stored name
mov dx, di
call printString
; Display Phone Number
mov dx, offset 'Phone: '
call printString
; Display stored phone
lea si, [di + 20] ; Offset for phone in entry
mov dx, si
call printString
; New line
call printNewLine
add si, entrySize
loop view_loop
jmp displayMenu
```

---

5. Exiting the Program

Clean up and terminate:

```assembly
exitProgram:
```
```

```
mov ah, 4ch
int 21h
```

Supporting Subroutines and Details

Implementing robust subroutines is essential:

- printString: Outputs a string pointed to by DX.
- readChar: Reads a single character from keyboard input.
- readString: Reads user input into a buffer until Enter is pressed.
- storeEntry/storePhone: Copies data from buffers into the entries array.
- printNewLine: Outputs a new line.

For example, `printString` might look like:

```assembly
printString:
mov ah, 09h
int 21h
ret
```

Similarly, `readChar`:

```assembly
readChar:
mov ah, 01h
int 21h
ret
```

And `readString` involves reading characters in a loop, handling backspaces, and terminating on Enter.

Compiling and Running the Program in the Emulator

Once the assembly code is written, you need to assemble and run it within the emulator.

Assembly and Linking

- Use an assembler like NASM or MASM compatible with 8086 syntax.
- Compile the `.asm` source file into a `.COM` or `.EXE` executable.

For example, with NASM:

```bash
nasm -f bin phonebook.asm -o phonebook.com
```

Ensure the code is in `.COM` format for simplicity.

Loading in Emulator

- Launch EMU8086.
- Load the compiled program.
- Run the program and interact via the emulator’s console.

Challenges and Best Practices

Writing an assembly program for a phone book involves careful management of data and control flow. Here are some tips:

- Fixed-size buffers: Simplify input handling.
- Use subroutines: Modularize code and improve readability.
- Comment extensively: Assembly language is dense; comments aid understanding.
- Test incrementally: Start with simple display routines before adding input and data storage.
- Backup regularly: Assembly programs can be prone to errors like overwriting memory.
