System Programming
Assignment #2

411021246
資工四 林威佑

作業問題描述

Write a simple SIC simulator that reads a standard SIC object program into memory.

1. 基本功能
   模擬器接受以下兩個指令：
   (a) load ObjectFileName：將標準 SIC 目標程式載入至指定的記憶體位置。
   (b) show：顯示載入目標程式的記憶體位置內容。
2. 進階功能
   除了基本功能，模擬器還接受以下兩個指令：
   (c) unload：釋放載入目標程式的資料結構。
   (d) exit：成功退出模擬器。
   此外，如果記憶體中已載入一個目標程式，執行 load 指令時需顯示錯誤訊息。
3. 完整功能
   除了進階功能，模擬器還需接受以下指令：
   (e) run：模擬執行已載入的目標程式，並在執行結束後顯示所有暫存器的內容。

程式結構與主要功能概述

s_load Function：

```
void s_load()
{
    // 確認檔案是否重新load
    if (loaded)
    {
        printf("Error: already loaded in memory!\n");
        return;
    }

    f = fopen(fname, "r");
    if (!f)
    {
        printf("Error: Cannot open file %s!\n", fname);
        return;
    }

    while (fgets(o_line, 80, f))
    {
        switch (o_line[0])
        {
        case 'H': // Header record
            rd_header();
            break;
        case 'T': // Text record
            rd_text();
            break;
        case 'E': // End record
            rd_end();
            break;
        }
    }

    fclose(f);
    printf("%s is loaded successfully.(Starts at %x, Length = %X.)\n", fname, start_add, prog_len);
}
```

1. **檢查是否已載入**：
    - 函數首先檢查是否已經有程式載入到記憶體中。如果 loaded 變數為真，則表示已經有程式載入，函數會輸出錯誤訊息並返回。
2. **開啟檔案**：
    - 使用 fopen 函數嘗試開啟指定的檔案 fname。如果檔案無法開啟，則輸出錯誤訊息並返回。
3. **讀取檔案內容**：
    - 使用 fgets 函數逐行讀取檔案內容，每次讀取最多 80 個字元到 o_line 緩衝區中。
4. **解析記錄**：

- 根據每行的第一個字元來判斷記錄的類型：
  - H：標頭記錄（Header record），調用 rd_header 函數處理。
  - T：文字記錄（Text record），調用 rd_text 函數處理。
  - E：結束記錄（End record），調用 rd_end 函數處理。

5. **關閉檔案**：
   - 使用 fclose 函數關閉檔案。
6. **輸出成功訊息**：
   - 輸出載入成功的訊息，顯示檔案名稱、起始位址和程式長度。

s_show Function：

```c
void s_show()
{
    int i;

    if (!loaded)
    {
        printf("Error: No program is loaded!\n");
        return;
    }

    // 內容
    // printf("Memory Contents:\n");
    for (i = 0; i < mem_size - 1; i += 2)
    {
        if (i % 32 == 0)
        {
            if (i > 0)
                printf("\n");
            printf("%06X", start_add + (i / 2));
        }
        if (i % 8 == 0)
            printf("  ");
        printf("%c%c", memory[i], memory[i + 1]);
    }
    printf("\n");
}
```

1. 檢查是否有程式載入：
    - 函數首先檢查是否有程式載入到記憶體中。如果 loaded 變數為假，則表示沒有程式載入，函數會輸出錯誤訊息並返回。
2. 顯示記憶體內容：
    - 使用一個迴圈從記憶體的起始位址開始，遍歷記憶體內容，每次處理 2 個位元組。
    - 每 32 個位元組（16 個記憶體單元）換行一次，並顯示當前的記憶體位址。
    - 每 8 個位元組（4 個記憶體單元）插入兩個空格以便於閱讀。
    - 使用 printf 函數以十六進位格式顯示記憶體內容。

s_unload Function：

```
void s_unload()
{

    if (!loaded)
    {
        printf("Error: No program is loaded!\n");
        return;
    }

    free(memory);
    memory = NULL;
    loaded = 0;
    mem_size = 0;
    prog_len = 0;
    start_add = 0;
    first_add = 0;
    printf("%s is unloaded successfully.\n", fname);
}
```

1. **檢查是否有程式載入**：
    - 函數首先檢查是否有程式載入到記憶體中。如果 loaded 變數為假，則表示沒有程式載入，函數會輸出錯誤訊息並返回。
2. **釋放記憶體**：
    - 使用 free 函數釋放先前分配給 memory 的記憶體空間。
    - 將 memory 指標設為 NULL，以避免懸空指標。
3. **重置狀態變數**：
    - 將 loaded 設為 0，表示沒有程式載入。
    - 將 mem_size 設為 0，表示記憶體大小重置。
    - 將 prog_len 設為 0，表示程式長度重置。
    - 將 start_add 設為 0，表示起始位址重置。
    - 將 first_add 設為 0，表示第一個位址重置。
4. **輸出成功訊息**：
    - 輸出卸載成功的訊息，顯示檔案名稱。

s_run Function：

```
void s_run()
{
    char input;
    if (!loaded)
    {
        printf("Error: No program is loaded!\n");
        return;
    }

    printf("Start running the program.\n");
    init_run();

    while (running)
    {

        get_op();

        switch (op)
        {
        case oADD: // ADD done
            reg_A += get_value(operand, indexed);
            break;
        case oAND: // AND done
            reg_A &= get_value(operand, indexed);
            break;
        case oDIV: // DIV done
            if (get_value(operand, indexed) != 0)
                reg_A /= get_value(operand, indexed);
            break;
        case oLDA: // LDA done
            reg_A = get_value(operand, indexed);
            break;
        case oLDCH: // LDCH done
            reg_A = (reg_A & 0xFFFF00) | get_byte(operand, indexed);
            break;
        case oLDL: // LDL done
            reg_L = get_value(operand, indexed);
            break;
        case oLDX: // LDX done
            reg_X = get_value(operand, indexed);
```

```
        break;
    case oMUL: // MUL done
        reg_A *= get_value(operand, indexed);
        break;
    case oOR: // OR done
        reg_A |= get_value(operand, indexed);
        break;
    case oRSUB: // RSUB done
        if (reg_L == 0)
        {
            // 如果reg_L = 0直接結束
            running = 0;
        }
        else
        {
            reg_PC = reg_L;
            curr_add = (reg_L - start_add) * 2;
        }
        break;
    case oSTA: // STA done
        put_value(reg_A, operand, indexed);
        break;
    case oSTCH: // STCH done
        put_byte(reg_A & 0xFF, operand, indexed);
        break;
    case oSTL: // STL done
        put_value(reg_L, operand, indexed);
        break;
    case oSTX: // STX done
        put_value(reg_X, operand, indexed);
        break;
    case oSUB: // SUB done
        reg_A -= get_value(operand, indexed);
        break;
```

```
case oCOMP: // COMP done
{
    int tmp = get_value(operand, indexed);
    if (reg_A < tmp)
        reg_SW = -1;
    else if (reg_A > tmp)
        reg_SW = 1;
    else
        reg_SW = 0;
}
break;
case oJ: // J done
    reg_PC = operand;
    curr_add = (operand - start_add) * 2;
    break;
case oJEQ: // JEQ done
    if (reg_SW == 0)
    {
        reg_PC = operand;
        curr_add = (operand - start_add) * 2;
    }
    break;
case oJGT: // JGT done
    if (reg_SW > 0)
    {
        reg_PC = operand;
        curr_add = (operand - start_add) * 2;
    }
    break;
case oJLT: // JLT done
    if (reg_SW < 0)
    {
        reg_PC = operand;
        curr_add = (operand - start_add) * 2;
    }
    break:
```

```
        case oJSUB: // JSUB done
            reg_L = reg_PC;
            reg_PC = operand;
            curr_add = (operand - start_add) * 2;
            break;
        case oRD: // RD
            printf("Please input a character: ");
            scanf(" %c", &input);
            reg_A = (reg_A & 0xFFFF00) | (input & 0xFF);
            break;
        case oTD: // TD
            reg_SW = 1;
            break;
        case oTIX: // TIX
            reg_X++;
            {
                int tmp = get_value(operand, indexed);
                if (reg_X < tmp)
                    reg_SW = -1;
                else if (reg_X > tmp)
                    reg_SW = 1;
                else
                    reg_SW = 0;
            }
            break;
        case oWD: // WD
            if (reg_A >= 32 && reg_A <= 126)
                printf("Output a character: [%c]\n", reg_A);
            break;
        default:
            printf("Error: Invalid operation code!\n");
            running = 0;
            break;
        }

        if (reg_PC >= start_add + prog_len)
            running = 0;
    }
```

```
    show_reg();
    printf("Program execution ended!\n");
}
```

1. **檢查是否有程式載入**：
    - 函數首先檢查是否有程式載入到記憶體中。如果 loaded 變數為假，則表示沒有程式載入，函數會輸出錯誤訊息並返回。
2. **初始化執行環境**：
    - 輸出 "Start running the program." 訊息。
    - 調用 init_run 函數初始化執行環境。

3. **執行指令**：
    - 使用 while (running) 迴圈執行程式，直到 running 變數設為 0。
    - 在每次迴圈中，調用 get_op 函數獲取當前指令。
    - 使用 switch 語句根據指令碼 (op) 執行相應的操作。
4. **指令處理**：
    - oADD：將操作數加到暫存器 A。
    - oAND：將操作數與暫存器 A 進行 AND 操作。
    - oDIV：將暫存器 A 除以操作數（如果操作數不為 0）。
    - oLDA：將操作數載入暫存器 A。
    - oLDCH：將操作數的低位元組載入暫存器 A。
    - oLDL：將操作數載入暫存器 L。
    - oLDX：將操作數載入暫存器 X。
    - oMUL：將暫存器 A 乘以操作數。
    - oOR：將操作數與暫存器 A 進行 OR 操作。
    - oRSUB：返回子程式，將暫存器 L 的值載入程式計數器（PC），如果 L 為 0 則結束執行。
    - oSTA：將暫存器 A 的值存儲到操作數位置。
    - oSTCH：將暫存器 A 的低位元組存儲到操作數位置。
    - oSTL：將暫存器 L 的值存儲到操作數位置。
    - oSTX：將暫存器 X 的值存儲到操作數位置。
    - oSUB：從暫存器 A 中減去操作數。
    - oCOMP：比較暫存器 A 和操作數，更新狀態暫存器（SW）。
    - oJ：無條件跳轉到操作數位置。
    - oJEQ：如果狀態暫存器為 0，則跳轉到操作數位置。

- oJGT：如果狀態暫存器大於 0，則跳轉到操作數位置。
- oJLT：如果狀態暫存器小於 0，則跳轉到操作數位置。
- oJSUB：跳轉到子程式，將當前程式計數器存儲到暫存器 L。
- oRD：讀取一個字元輸入到暫存器 A。
- oTD：設置狀態暫存器為 1。
- oTIX：將暫存器 X 增加 1，並與操作數比較，更新狀態暫存器。
- oWD：輸出暫存器 A 中的字元。

5. **檢查程式結束條件**：
   - 如果程式計數器（PC）超過程式的結束位址，則設置 running 為 0，結束執行。
6. **顯示暫存器狀態**：
   - 調用 show_reg 函數顯示暫存器的當前狀態。
   - 輸出 "Program execution ended!" 訊息。

Object Program & Run Result

Test1.obj

```
test1.obj
HCOPY  00100000107A
T0010001E14103348203900103628103030101548206 13C100300102A0C103900102D
T00101E150C10364820610810334C0000454F46000003000040
T0020391E041030001030E0205D30203FD8205D28103030205754903 92C205E38203F
T0020571C1010364C0000F1001000041030E02079302064509039DC20792C1036
T002073073820644C000005
E001000
```

Show the memory of loading Test1.obj

```
SIC Simulator> load test1.obj
test1.obj is loaded successfully.(Starts at 1000, Length = 107A.)
SIC Simulator> show
001000  14103348  20390010  36281030  30101548
001010  20613C10  0300102A  0C103900  102D0C10
001020  36482061  0810334C  0000454F  46000003
001030  000040XX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001040  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001050  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001060  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001070  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001080  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001090  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010A0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010B0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010C0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010D0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010E0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010F0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001100  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001110  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001120  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001130  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001140  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001150  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001160  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001170  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001180  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001190  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0011A0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0011B0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0011C0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0011D0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0011E0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
```

(省略擷取 memory 中間的空值)

```
002030  XXXXXXXX  XXXXXXXX  XX041030  001030E0
002040  205D3020  3FD8205D  28103030  20575490
002050  392C205E  38203F10  10364C00  00F10010
002060  00041030  E0207930  20645090  39DC2079
002070  2C103638  20644C00  0005
```

Result After running Test1.obj

```
SIC Simulator> run
Start running the program.
Please input a character: b
Please input a character: c
Please input a character: d
Please input a character: @
Output a character: [b]
Output a character: [c]
Output a character: [d]
Please input a character: @
Output a character: [b]
Register A  = [000062];
Register X  = [000041];
Register L  = [000000];
Register SW = [000001];
Register PC = [00102A];
Program execution ended!
```

```
SIC Simulator> show
001000  14103348  20390010  36281030  30101548
001010  20613C10  0300102A  0C103900  102D0C10
001020  36482061  0810334C  0000454F  46000003
001030  00004000  00000000  03454F46  XXXXXXXX
001040  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001050  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001060  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001070  XXXXXXXX  XXXXXXXX  XX626364  XXXXXXXX
001080  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
001090  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
0010A0  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
```

```
002030  XXXXXXXX  XXXXXXXX  XX041030  001030E0
002040  205D3020  3FD8205D  28103030  20575490
002050  392C205E  38203F10  10364C00  00F10010
002060  00041030  E0207930  20645090  39DC2079
002070  2C103638  20644C00  0005
```

Test2.obj

```
test2.obj
HMATH  00200000002D
T0020001E00201E1820241C201B0C2027002021182024 1C201B0C202A4C0000000001
T00201E09000005000007000003
E002000
```

Show the memory of loading Test2.obj

```
SIC Simulator> load test2.obj
test2.obj is loaded successfully.(Starts at 2000, Length = 2D.)
SIC Simulator> show
002000  00201E18  20241C20  1B0C2027  00202118
002010  20241C20  1B0C202A  4C000000  00010000
002020  05000007  000003XX  XXXXXXXX  XX
```

Result After running Test2.obj

```
SIC Simulator> run
Start running the program.
Register A  = [000009];
Register X  = [000000];
Register L  = [000000];
Register SW = [000000];
Register PC = [00201B];
Program execution ended!
SIC Simulator> show
002000  00201E18  20241C20  1B0C2027  00202118
002010  20241C20  1B0C202A  4C000000  00010000
002020  05000007  00000300  00070000  09
```

Test3.obj

```
test3.obj
HTEST3 003000000015
T003000120030C18300F0C30124C0000000005000003
E003000
```

Show the memory of loading Test3.obj

```
test3.obj is loaded successfully.(Starts at 3000, Length = 15.)
SIC Simulator> show
003000  00300C18  300F0C30  124C0000  00000500
003010  0003XXXX  XX
```

Result After running Test3.obj

```
SIC Simulator> run
Start running the program.
Register A  = [000008];
Register X  = [000000];
Register L  = [000000];
Register SW = [000000];
Register PC = [00300C];
Program execution ended!
SIC Simulator> show
003000  00300C18  300F0C30  124C0000  00000500
003010  00030000  08
```

Discussion

這份作業設計與實作一個簡單的 SIC 模擬器，幫助深入了解組合語言與計算機架構的基本原理。藉由實現功能如 load、show 和 run，實際體驗目標程式如何被載入記憶體並執行，進一步掌握機器層級指令的運作方式。