Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування
і спеціалізованих комп'ютерних системи

Лабораторна робота №3

з дисципліни
«Архітектура комп'ютерів 2. Програмне забезпечення»

Виконав:
Студент групи КВ-82
Іваненко Олександр Андрійович

Перевірив:
Молчанов О. А.

Київ – 2021

Загальне завдання

Реалізувати програму мовою С, що виконує зчитування послідовності команд (програми) з файлу і заміняє віртуальні адреси на фізичні в командах, що визначаються варіантом. Тип організації пам’яті також визначається варіантом. Заміна адреси відбувається у випадку, якщо сторінка та/або сегмент знаходиться в оперативній пам’яті (ОП). Якщо потрібна віртуальна сторінки та/або сегмент відсутній в ОП, тоді має бути виведене повідомлення про помилку відсутності сторінки/сегменту, й аналіз команд має бути продовжено. Таблиця сторінок/сегментів задається у файлі формату CSV.

Завдання за варіантом 10

10 сторінкова

PC: 8 Кбайт

```
1  MOV   <reg1>, <reg2>           1A /reg1 /reg2
2  MOV   <reg>, <addr>            1B 0 /reg /addr
6  ADD   <reg1>, <reg2>, <reg3>   03 0 /reg1 /reg2 /reg3
7  ADD   <reg1>, <reg2>, <addr>   04 /reg1 /reg2 /addr
13 MUL   <reg1>, <reg2>, <reg3>   21 0 /reg1 /reg2 /reg3
15 MUL   <reg1>, <reg2>, <addr>   23 /reg1 /reg2 /addr
20 JMP   <shift>                  90 /shift
21 JMP   <addr>                   91 /addr
27 MOV   <reg>, <lit8>            1C 0 /reg /lit8
```

Лістинг програми

Модуль парсингу файлу пам’яті

```
class Page{
public :
    Page(vector<int> page) {
        this->number = page[0];
        this->byte_presence = page[1];
        this->frame = page[2];
    }
    int number;
    int byte_presence;
    int frame;
};

vector<Page> all_pages;

void parse_memory(){
    ifstream file("memory.csv");
    std::string line;
    while(std::getline(file, line)){
        std::stringstream ss(line);
        std::string token;
        vector<int> page;
        while (std::getline(ss, token, ',')) {
            page.push_back(stoi(token));
        }
        all_pages.emplace_back(page);
    }
}
```

Модуль обробки операндів типу віртуальна пам'ять

```
string create_addr(int number){
    string addr = int_to_hex(number);
    if(addr.size() != 8){
        addr.insert(0, 8 - addr.size(), '0');
    }
    return "[0x" + addr + "]";
}

int create_phys_addr(int frame, int shift){
    int start = 13;  //last power of shift
    while(frame){
        shift += frame % 10 * pow(2, start++);
        frame /= 10;
    }
    return shift;
}


pair<string, string> get_addr(vector<int> addr){
    int full_addr = vector_to_byte(addr);
    string virtual_addr = create_addr(full_addr);

    int number_page = (full_addr & 0xFFFFE000) >> 13; //first 21 bites on
number page
    int shift = full_addr & 0x00001FFF; //last 13 bites on shift

    //search in memory
    for(Page page : all_pages){
        if(number_page == page.number){
            if(page.byte_presence){
                int phys_addr = create_phys_addr(page.frame, shift);
                return make_pair(create_addr(phys_addr), "");
            }
            else{
                return make_pair(virtual_addr, "ERROR: failed to convert
virtual address" +
                virtual addr + "because byte of presence is 0");
            }
        }
    }
    return make_pair(virtual_addr, "ERROR: failed to convert virtual address"
+ virtual_addr + "because page " + to_string(number_page) + " does not
exist");

}
```

Модуль обробки системи команд

```
Command mov_reg_reg(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 2);
    pop_front(bytes);

    int registers = pop_front(bytes);

    vector<string> operands;
    operands.push_back(create_reg((registers & 0xF0) >> 4));
    operands.push_back(create_reg(registers & 0x0F));

    return Command("mov", create_bytes_from_vector(this_command_bytes),
operands);

}

Command mov_reg_addr(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 6);
    vector<string> operands;
    pop_front(bytes);

    int reg = pop_front(bytes);
    operands.push_back(create_reg(reg & 0x0F));

    pair<string, string> addr = get_addr(pop_front(bytes, 4));
    string real_addr = addr.first;
    string error = addr.second;
    operands.push_back(real_addr);

    Command result = Command("mov",
create_bytes_from_vector(this_command_bytes), operands);
    if(!error.empty()){
        result.setError(error);
    }
    return result;
}


Command mov_reg_lit(vector<int> &bytes) {
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 3);
    pop_front(bytes);

    int regist = pop_front(bytes);
    int literal = pop_front(bytes);

    vector<string> operands;
    operands.push_back(create_reg(regist & 0x0F));
    operands.push_back(to_string(literal));

    return Command("mov", create_bytes_from_vector(this_command_bytes),
operands);
}
```

```
Command add_reg_reg_reg(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 3);
    vector<string> operands;
    pop_front(bytes);

    int reg1 = pop_front(bytes);
    int reg23 = pop_front(bytes);

    operands.push_back(create_reg(reg1 & 0x0F));
    operands.push_back(create_reg((reg23 & 0xF0) >> 4));
    operands.push_back(create_reg(reg23 & 0x0F));

    return Command("add", create_bytes_from_vector(this_command_bytes),
operands);

}

Command add_reg_reg_addr(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 6);
    vector<string> operands;
    pop_front(bytes);

    int regs = pop_front(bytes);
    operands.push_back(create_reg((regs & 0xF0) >> 4));
    operands.push_back(create_reg(regs & 0x0F));

    pair<string, string> addr = get_addr(pop_front(bytes, 4));
    string real_addr = addr.first;
    string error = addr.second;
    operands.push_back(real_addr);

    Command result = Command("add",
create_bytes_from_vector(this_command_bytes), operands);
    if(!error.empty()){
        result.setError(error);
    }
    return result;

}

Command mul_reg_reg_reg(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 3);
    pop_front(bytes);

    int reg1 = pop_front(bytes);
    int reg23 = pop_front(bytes);

    vector<string> operands;
    operands.push_back(create_reg(reg1 & 0x0F));
    operands.push_back(create_reg((reg23 & 0xF0) >> 4));
    operands.push_back(create_reg(reg23 & 0x0F));

    return Command("mul", create_bytes_from_vector(this_command_bytes),
operands);
```

```
}

Command mul_reg_reg_addr(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 6);
    vector<string> operands;
    pop_front(bytes);

    int regs = pop_front(bytes);
    operands.push_back(create_reg((regs & 0xF0) >> 4));
    operands.push_back(create_reg(regs & 0x0F));

    pair<string, string> addr = get_addr(pop_front(bytes, 4));
    string real_addr = addr.first;
    string error = addr.second;
    operands.push_back(real_addr);

    Command result = Command("mul",
create_bytes_from_vector(this_command_bytes), operands);
    if(!error.empty()){
        result.setError(error);
    }
    return result;

}

Command jmp_addr(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 5);
    vector<string> operands;
    pop_front(bytes);


    pair<string, string> addr = get_addr(pop_front(bytes, 4));
    string real_addr = addr.first;
    string error = addr.second;
    operands.push_back(real_addr);

    Command result = Command("jmp",
create_bytes_from_vector(this_command_bytes), operands);
    if(!error.empty()){
        result.setError(error);
    }
    return result;

}

Command jmp_shift(vector<int> &bytes){
    vector<int> this_command_bytes = vector<int>(bytes.begin(), bytes.begin()
+ 2);
    vector<string> operands;
    pop_front(bytes);

    int shift = pop_front(bytes);
    operands.push_back(int_to_hex(shift));
```

```
    return Command("jmp", create_bytes_from_vector(this_command_bytes),
operands);
}
```

Модуль обробки вибору команд та зчитування файлу байтів

```
Collector::Collector(const string& name_of_file) {
    int byte;
    std::ifstream file("inputs/" + name_of_file + ".txt");
    if (file.is_open()) {
        while(file >> hex >> byte >> setbase(0)){
            bytes.push_back(byte);
        }
    }
}

Command Collector::define_command(int byte){
    switch(byte){
        case 0x1A: return mov_reg_reg(bytes);
        case 0x1C: return mov_reg_lit(bytes);
        case 0x03: return add_reg_reg_reg(bytes);
        case 0x04: return add_reg_reg_addr(bytes);
        case 0x21: return mul_reg_reg_reg(bytes);
        case 0x23: return mul_reg_reg_addr(bytes);
        case 0x1B: return mov_reg_addr(bytes);
        case 0x90: return jmp_shift(bytes);
        case 0x91: return jmp_addr(bytes);
        default:
            error_indicator = true;
            return Command("Error command with code [" + int_to_hex(byte) +
"]");
    }
}

void Collector::collect_commands() {
    while(!bytes.empty() && !error_indicator){
        commands.push_back(define_command(bytes[0]));
    }
}
```

Тестування програми

Вхідний файл:

```
1a 04
1c 01 04
1a 12
03 03 02
21 03 02
90 04

91 00 00 b0 fb
23 32 00 00 b0 fb
1B 02 00 00 b0 fb
04 12 00 00 b0 fb

91 00 00 90 fb
23 32 00 00 90 fb
1B 02 00 00 90 fb
04 12 00 00 90 fb

91 00 03 f0 fb
23 32 00 03 f0 fb
1B 02 00 03 f0 fb
04 12 00 03 f0 fb
```

Файл пам’яті:

```
0,0,
1,1,001
2,0,
3,0,
4,0,
5,1,010
6,1,011
7,0,
8,1,100
9,0,
10,1,101
11,0,
12,1,110
13,0,
14,1,111
15,0,
16,1,000
```

Результат:

```
1a 04 :
mov r0, r4

1c 01 04 :
mov r1, 4

1a 12 :
mov r1, r2

03 03 02 :
add r3, r0, r2

21 03 02 :
mul r3, r0, r2

90 04 :
jmp 04

91 00 00 b0 fb :
jmp [0x000050fb]

23 32 00 00 b0 fb :
mul r3, r2, [0x000050fb]

1b 02 00 00 b0 fb :
mov r2, [0x000050fb]

04 12 00 00 b0 fb :
add r1, r2, [0x000050fb]

91 00 00 90 fb :
ERROR: failed to convert virtual address[0x000090fb]because byte of presence is 0
jmp [0x000090fb]

23 32 00 00 90 fb :
ERROR: failed to convert virtual address[0x000090fb]because byte of presence is 0
mul r3, r2, [0x000090fb]

1b 02 00 00 90 fb :
ERROR: failed to convert virtual address[0x000090fb]because byte of presence is 0
mov r2, [0x000090fb]

04 12 00 00 90 fb :
ERROR: failed to convert virtual address[0x000090fb]because byte of presence is 0
add r1, r2, [0x000090fb]

91 00 03 f0 fb :
ERROR: failed to convert virtual address[0x0003f0fb]because page 31 does not exist
jmp [0x0003f0fb]

23 32 00 03 f0 fb :
ERROR: failed to convert virtual address[0x0003f0fb]because page 31 does not exist
mul r3, r2, [0x0003f0fb]

1b 02 00 03 f0 fb :
ERROR: failed to convert virtual address[0x0003f0fb]because page 31 does not exist
mov r2, [0x0003f0fb]

04 12 00 03 f0 fb :
ERROR: failed to convert virtual address[0x0003f0fb]because page 31 does not exist
add r1, r2, [0x0003f0fb]
```