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 №2

з дисципліни
«Архітектура для програмістів»

Виконала:
Студентка групи КВ-11
Нестерук Анастасія Олександрівна

Перевірив:
Молчанов О. А.

Київ – 2024

Загальне завдання

Завдання лабораторної роботи наступне: реалізувати програму мовою С або 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.

Програма має містити наступні компоненти:

1. Модуль зчитування таблиці сторінок/сегментів з файлу CSV і створення внутрішнього представлення відповідної таблиці (або таблиць);
2. Модуль з реалізацією функцій зчитування, аналізу і зміни команд з текстового файлу, що виконує заміну віртуальних адрес в зчитаних командах на фізичні;
3. Модуль тестування, що містить тестові утиліти і тести реалізованої програми

Завдання за варіантом 16

Варіант	Тип організації пам'яті	Параметри віртуальної адреси	Список команд
1	сторінкова	PC: 4 Кбайт	1, 2, 12, 14, 24, 25, 26, 27, 29

Команди за варіантом:

№	Команда	Код команди (0x)	Опис
1	MOV <reg1>, <reg2>	1A /reg1 /reg2	перемістити значення з регістру <reg1> у регістр <reg2>
2	MOV <reg>, <addr>	1B 0 /reg /addr	перемістити значення з ОП за адресою <addr> у регістр <reg>
12	MUL <reg1>, <reg2>	20 /reg1 /reg2	множення значення з регістру <reg1> на значення з регістру <reg2> і збереження результату в регістрі <reg1>. Множення відбувається без розширення, тобто без збереження старших розрядів, якщо результат має розрядність більшу за 32 біти

14	MUL <reg>, <addr>	22 0 /reg /addr	множення значення з регістру <reg> на 4-байтове значення з ОП з адресою <addr> і збереження результату в регістрі <reg>. Множення відбувається без розширення, тобто без збереження старших розрядів, якщо результат має розрядність більшу за 32 біти
24	JG <shift>	94 /shift	перехід за 1-байтовим відносним зміщенням <shift> у випадку, якщо ZF = 0 і SF = OF
25	JG <addr>	95 /addr	перехід за 4-байтовою адресою <addr> у випадку, якщо ZF = 0 і SF = OF
26	CMP <reg1>, <reg2>	80 /reg1 /reg2	порівняння двох значень і встановлення відповідних прапорців
27	MOV <reg>, <lit8>	1C 0 /reg /lit8	переміщення 1-байтового числа у регістра <reg>
29	MOV <reg>, <lit32>	1C 2 /reg /lit32	переміщення 4-байтового числа у регістра <reg>

Лістинг програми мовою C++

Main.cpp

```cpp
#include <iostream>

using namespace std;

#include "Convertor.h"
#include "Testing.h"


int main() {

    Testing("test.txt"  , "pages.csv" , "expect.txt");

    return 0;
}
```

Convertor.h

```cpp
#pragma once
#include <sstream>
#include <ostream>
#include <istream>
#include <string>
#include <bitset>
```

```cpp
#include <iomanip>
#include <vector>
#include <fstream>
#include <iostream>
#include <cctype>
#include <algorithm>
#include <functional>

using namespace std;

struct Page
{
    int Number_of_page;
    uint8_t frame_number;
    bool check_bit;
};

class Convertor
{
private:
    vector<unsigned char> Binary_Value;
    vector<string> Hex_Value;
    vector<Page> pages;
    fstream Listing;
    int position;

private:
    uint32_t Convert_Virtual_address_to_Physical(uint32_t
virtual_adress);
    uint8_t Search_frame_page(uint32_t number_of_page, bool&
search_result);
    void MOV_reg_reg();
    void MOV_reg_addr();
    void MUL_reg_reg();
    void MUL_reg_addr();
    void JG_shift();
    void JG_addr();
    void CMP_reg_reg();
    void MOV_reg_lit();


public:
    int Read_Data_From_File(ifstream& input_file);
    int Command_Converting();
    void Get_Pages_info(ifstream& csv_file);
    void Check_Expect(ifstream& expect_file);
    Convertor();
};
```

Convertor.cpp

```cpp
#include "Convertor.h"

Convertor::Convertor():position(0)
{
    Listing.open("output.txt", ios::out);
    if (!Listing.is_open()) {
        cerr << "Error opening output file" << endl;
        exit(-1);
    }
}

int Convertor::Read_Data_From_File(ifstream& input_file) {

    string line;

    if (!input_file.is_open()) {
        cerr << "Error opening input file\n";
        return -1;
    }



    while (getline(input_file, line)) {
        istringstream iss(line);
        string hexString;
        while (iss >> hexString) {
            unsigned long hexValue = stoul(hexString, nullptr, 16);
           // bitset<32> binary(hexValue);
           // output_file << binary << " ";
            Binary_Value.push_back(static_cast<unsigned
char>(hexValue));
            Hex_Value.push_back(hexString);
        }
       // output_file << endl;
    }

    input_file.close();
```

```
/*
    ifstream input_output_file("test_output.txt");
    if (!input_output_file.is_open()) {
        cerr << "Error opening output file for reading\n";
        return -1;
    }

    cout << "Contents of the output file:\n";
    while (getline(input_output_file, line)) {
        cout << line << endl;
    }

    input_output_file.close();
*/
    cout << "File copied successfully." << endl << endl;

    return 1;
}

int Convertor::Command_Converting() {


    while (position < Hex_Value.size()) {

        if (Hex_Value[position] == "1A") {
            MOV_reg_reg();
        }
        else if (Hex_Value[position] == "1B") {
            MOV_reg_addr();
        }
        else if (Hex_Value[position] == "20") {
            MUL_reg_reg();
        }
        else if (Hex_Value[position] == "22") {
            MUL_reg_addr();
        }
        else if (Hex_Value[position] == "94") {
            JG_shift();
        }
        else if (Hex_Value[position] == "95") {
            JG_addr();
        }
        else if (Hex_Value[position] == "80") {
            CMP_reg_reg();
        }
        else if (Hex_Value[position] == "1C") {
```

```
            MOV_reg_lit();
        }
        else {
            cerr << "ERR0R: Incorrect command" << endl;
            exit(-1);
        }

    }
    Listing.close();
        return 0;

}

void Convertor::MOV_reg_reg() {
    uint8_t regs = 0, reg1 = 0, reg2 = 0;
    position++;
    if (Hex_Value[position].size() != 2) {
        cerr << "ERR0R: Incorrect byte assignment for '1A'." << endl;
        exit(-1);
    }
    else {
        regs = Binary_Value[position];
        reg1 = regs >> 4;
        reg2 = regs & 0b00011111;
        cout << "1A " << uppercase << hex << setfill('0') << setw(2) <<
(int)regs << ":" << endl;
        cout << "MOV R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl << endl;
        Listing << "MOV R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl;
        position++;
    }
}

void Convertor::MOV_reg_addr() {
    uint8_t  reg = 0;
    uint32_t virtual_adress = 0, physical_adress = 0;
    position++;
    uint8_t operands[5];
    for (int i = 0; i < 5; i++) {
        if (Hex_Value[position].size() != 2) {
            cerr << "ERR0R: Incorrect byte assignment for '1B'." <<
endl;
            exit(-1);
        }
        else {
            operands[i] = Binary_Value[position];
            position++;
```

```
        }
    }
    if ((operands[0] >> 4) != 0) {
        cerr << "ERR0R: Incorrect command variation for '1B'." << endl;
        exit(-1);
    }
    else reg = operands[0] & 0b00011111;

    for (int i = 1; i < 5; i++) {
        virtual_adress = (virtual_adress << 8) | operands[i];
    }

    physical_adress =
Convert_Virtual_address_to_Physical(virtual_adress);
    cout << "1B ";
    for (int i = 0; i < 5; i++)
        cout << uppercase << hex << setfill('0') << setw(2) <<
(int)operands[i] << " ";

    cout << ":" << endl << "MOV R" << uppercase << hex << (int)reg <<
", [0x" << physical_adress << "]" << endl << endl;
    Listing << "MOV R" << uppercase << hex << (int)reg << ", [0x" <<
physical_adress << "]" << endl << endl;

}

void Convertor::MUL_reg_reg() {
    uint8_t regs = 0, reg1 = 0, reg2 = 0;
    position++;
    if (Hex_Value[position].size() != 2) {
        cerr << "ERR0R: Incorrect byte assignment for '20'." << endl;
        exit(-1);
    }
    else {
        regs = Binary_Value[position];
        reg1 = regs >> 4;
        reg2 = regs & 0b00011111;
        cout << "20 " << uppercase << hex << setfill('0') << setw(2) <<
(int)regs << ":" << endl;
        cout << "MUL R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl << endl;
        Listing << "MUL R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl;
        position++;
    }
}

void Convertor::MUL_reg_addr() {
```

```
    uint8_t  reg = 0;
    uint32_t virtual_adress = 0, physical_adress = 0;
    position++;
    uint8_t operands[5];
    for (int i = 0; i < 5; i++) {
        if (Hex_Value[position].size() != 2) {
            cerr << "ERR0R: Incorrect byte assignment for '22'." <<
endl;
            exit(-1);
        }
        else {
            operands[i] = Binary_Value[position];
            position++;
        }
    }
    if ((operands[0] >> 4) != 0) {
        cerr << "ERR0R: Incorrect command variation for '22'." << endl;
        exit(-1);
    }
    else reg = operands[0] & 0b00011111;
    for (int i = 1; i < 5; i++) {
        virtual_adress = (virtual_adress << 8) | operands[i];
    }

    physical_adress =
Convert_Virtual_address_to_Physical(virtual_adress);
    cout << "22 ";
    for (int i = 0; i < 5; i++)
        cout << uppercase << hex << setfill('0') << setw(2) <<
(int)operands[i] << " ";

    cout << ":" << endl << "MUL R" << uppercase << hex << (int)reg <<
", [0x" << physical_adress << "]" << endl << endl;
    Listing << "MUL R" << uppercase << hex << (int)reg << ", [0x" <<
physical_adress << "]" << endl;
}

void Convertor::JG_shift() {
    uint8_t shift = 0;
    position++;
    if (Hex_Value[position].size() != 2) {
        cerr << "ERR0R: Incorrect byte assignment for '94'." << endl;
        exit(-1);
    }
    else {
        shift = Binary_Value[position];
        cout << "94 " << uppercase << hex << setfill('0') << setw(2) <<
(int)shift << ":" << endl;
```

```
        cout << "JG [0x" << uppercase << hex << (int)shift << "]" <<
endl << endl;
        Listing << "JG [0x" << uppercase << hex << (int)shift << "]" <<
endl;
        position++;
    }
}

void Convertor::JG_addr() {
    uint32_t virtual_adress = 0, physical_adress = 0;
    position++;
    uint8_t operands[4];
    for (int i = 0; i < 4; i++) {
        if (Hex_Value[position].size() != 2) {
            cerr << "ERR0R: Incorrect byte assignment for '95'." <<
endl;
            exit(-1);
        }
        else {
            operands[i] = Binary_Value[position];
            position++;
        }
    }
    for (int i = 0; i < 4; i++) {
        virtual_adress = (virtual_adress << 8) | operands[i];
    }

    physical_adress =
Convert_Virtual_address_to_Physical(virtual_adress);
    cout << "95 ";
    for (int i = 0; i < 4; i++)
        cout << uppercase << hex << setfill('0') << setw(2) <<
(int)operands[i] << " ";

    cout << ":" << endl << "JG " << uppercase << hex << "[0x" <<
physical_adress << "]" << endl << endl;
    Listing << "JG " << uppercase << hex << "[0x" << physical_adress <<
"]" << endl;
}

void Convertor::CMP_reg_reg() {
    uint8_t regs = 0, reg1 = 0, reg2 = 0;
    position++;
    if (Hex_Value[position].size() != 2) {
        cerr << "ERR0R: Incorrect byte assignment for '80'." << endl;
        exit(-1);
    }
    else {
```

```
        regs = Binary_Value[position];
        reg1 = regs >> 4;
        reg2 = regs & 0b00011111;
        cout << "80 " << uppercase << hex << setfill('0') << setw(2) <<
(int)regs << ":" << endl;
        cout << "CMP R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl << endl;
        Listing << "CMP R" << uppercase << hex << (int)reg1 << ", R" <<
(int)reg2 << endl;
        position++;
    }
}

void Convertor::MOV_reg_lit() {
    uint8_t  reg = 0, tmpreg = 0;
    char lit32 = 0;
    uint8_t lit8 = 0;
    position++;
    if (Hex_Value[position].size() != 2) {
        cerr << "ERR0R: Incorrect byte assignment for '1C'." << endl;
        exit(-1);
    }
    else {
        tmpreg = Binary_Value[position];
        if ((tmpreg >> 4) == 0) {
            reg = tmpreg & 0b00011111;
            position++;
            if (Hex_Value[position].size() != 2) {
                cerr << "Incorrect constant length for '1C 0'." <<
endl;
                exit(-1);
            }
            lit8 = Binary_Value[position];
            cout << "1C " << uppercase << hex << setfill('0') <<
setw(2) << (int)tmpreg << " " << (int)lit8 << ":" << endl;
            cout << "MOV R" << uppercase << hex << (int)reg << ", " <<
setfill('0') << setw(2) << (int)lit8 << endl << endl;
            Listing << "MOV R" << uppercase << hex << (int)reg << ", "
<< setfill('0') << setw(2) << (int)lit8 << endl;
            position++;
        }
        else if ((tmpreg >> 4) == 2) {
            reg = tmpreg & 15;
            position++;
            if (Hex_Value[position].size() != 8) {
                cerr << "ERR0R: Incorrect constant length for '1C 2'."
<< endl;
                exit(-1);
```

```
            }
            lit32 = Binary_Value[position];
            cout << "1C " << uppercase << hex << setfill('0') <<
setw(2) << (int)tmpreg << " " << (int)lit32 << ":" << endl;
            cout << "MOV R" << uppercase << hex << (int)reg << ", " <<
setfill('0') << setw(2) << (int)lit32 << endl << endl;
            Listing << "MOV R" << uppercase << hex << (int)reg << ", "
<< setfill('0') << setw(2) << (int)lit32 << endl;
            position++;
        }
        else {
            cerr << "ERR0R: Incorrect command variation for '1C'.";
            exit(-1);
        }

    }

}


void Convertor::Get_Pages_info(ifstream& csv_file)
{
    string line;
    while (getline(csv_file, line))
    {
        stringstream ss(line);
        string token;
        vector<string> tokens;
        while (getline(ss, token, ','))
        {
            tokens.push_back(token);
        }

        Page page;

        page.Number_of_page = stoi(tokens[0] , nullptr , 10);

        if (page.Number_of_page <= 0xFFFFF) {

            page.check_bit = stoi(tokens[1], nullptr, 2);

            if (page.check_bit)
            {
                page.frame_number = stoi(tokens[2], nullptr, 2);
            }
            else {
                page.frame_number = 0;
```

```
            }

            pages.push_back(page);
        }
        else {
            cerr << "ERR0R: Incorrect length of page number" << endl;
            exit(-1);
        }


    }
}


uint32_t Convertor::Convert_Virtual_address_to_Physical(uint32_t
virtual_adress) {
    uint32_t number_of_page = virtual_adress >> 12;
    uint32_t page_offset = (virtual_adress << 20) >> 20;

    bool search_result = false;
    uint8_t frame_number = Search_frame_page(number_of_page,
search_result);

    if (search_result == false) {
        std::cerr << "ERROR: Failed to convert virtual address [0x" <<
uppercase << hex << setfill('0') << virtual_adress << "] because page "
<< dec << number_of_page << " doesn't exist.";
        exit(-1);
    }

    uint32_t physical_address = (frame_number << 12) | page_offset;
    return physical_address;
}

uint8_t Convertor::Search_frame_page(uint32_t number_of_page, bool&
search_result) {
    for (const auto& page : pages) {
        if (page.Number_of_page == number_of_page && page.check_bit) {
            search_result = true;
            return page.frame_number;
        }
    }
    search_result = false;
}


void Convertor::Check_Expect(ifstream& expect_file) {
```

```
        ifstream Listing("output.txt");

        if (!Listing.is_open()) {
            cerr << "Error opening Listing file" << endl;
            return;
        }

        if (!expect_file.is_open()) {
            cerr << "Error opening expect file" << endl;
            Listing.close();
            return;
        }

        string value1, value2;

        while (Listing >> value1 && expect_file >> value2) {


            if (value1 != value2) {
                cout << "Files do not match." << endl;
                cout << "Expectation: " << value2 << endl;
                cout << "Reality: " << value1 << endl;
                Listing.close();
                expect_file.close();
                return;
            }
        }

        if ((Listing >> value1) || (expect_file >> value2)) {
            cout << "Files do not match." << endl;
        }
        else {
            cout << "Files match." << endl;
        }


        Listing.close();
        expect_file.close();
    }
```

Testing.h

```
    #pragma once
    #include <iostream>
    #include <fstream>
    #include "Convertor.h"
```

```
    void Testing(string Input_File, string csv_File, string expect_File);
```

Testing.cpp

```
    #include "Testing.h"


    void Testing(string Input_File,  string csv_File ,string expect_File)
    {
        ifstream input_file(Input_File);
        ifstream csv_file(csv_File);
        ifstream expect_file(expect_File);
        Convertor lab2;
        lab2.Get_Pages_info(csv_file);
        lab2.Read_Data_From_File(input_file);
        lab2.Command_Converting();
        cout << "Conversion is completely successful" << endl;
        lab2.Check_Expect(expect_file);


    }
```

Тестування програми

Expect:

```
MOV R8,


R5

MOV R2,

[0xBF00]

MUL R8,      R6

MUL R6, [0x1100]

JG [0xFF]

JG [0x1100]

CMP R6, R7

MOV R2, FF

MOV R3, FFFFFFAA
```

Тест-1

```
1A 85 1B 02 FF 00 FF 00 20 86
22 06 00 11 11 00 94 ff
95  00 11 11 00
80 67

1C 02 ff 1C 23 4524ffaa
```

Результат

1A 85:

MOV R8, R5

1B 02 FF 00 FF 00 :

MOV R2, [0xBF00]

20 86:

MUL R8, R6

22 06 00 11 11 00 :

MUL R6, [0x1100]

94 FF:

JG [0xFF]

95 00 11 11 00 :

JG [0x1100]

80 67:

CMP R6, R7

1C 02 FF:

MOV R2, FF

```
1C 23 FFFFFFAA:

MOV R3, FFFFFFAA

Files match.
```

Тест-2

```
1A 5
1B 02 FF 00 FF 00
20 86
95  00 11 11 00
80 67
```

Результат

```
ERR0R: Incorrect byte assignment for '1A'.
```

Тест-3

```
1A 05
1B 02 FF FF 00 20 86
95  00 11 11 00
80 67
```

Результат

```
1A 05:

MOV R0, R5

ERROR: Failed to convert virtual address [0xFFFF0020] because page 1048560 doesn't
exist.
```

Тест-4

```
1A 05
1B 12 00 FF FF 00
20 86
```

Результат

```
1A 05:

MOV R0, R5

ERR0R: Incorrect command variation for '1B'.
```

Тест-5

```
1A 85

1C 02 ff5
1C 23 4524ffaa
```

Результат

```
1A 85:

MOV R8, R5

Incorrect constant length for '1C 0'.
```

Тест-6

Expect:

```
MOV R8, R4

MOV R2, FF

MOV R3, FFFFFFAA
```

Тест

```
1A 85

1C 02 ff5
1C 23 4524ffaa
```

Результат

```
1A 85:

MOV R8, R5


1C 02 FF:

MOV R2, FF


1C 23 FFFFFFAA:

MOV R3, FFFFFFAA

Files do not match.

Expectation: R4

Reality: R5
```