МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики ”

(НИУ ИТМО)

_____

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.04.04 Программная инженерия

Лабораторная работа № 3

Вариант 21

По дисциплине “Системное программное обеспечение”

Студент группы P4114

Трофимова Полина Владимировна

Преподаватель

Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023 г.

Задание на лабораторную

Реализовать формирование линейного кода в терминах некоторого набора инструкций посредством анализа графа потока управления для набора подпрограмм. Полученный линейный код вывести в мнемонической форме в выходной текстовый файл.

Подготовка к выполнению по одному из двух сценариев:

1. Составить описание виртуальной машины с набором инструкций и моделью памяти по варианту

a. Изучить нотацию для записи определений целевых архитектур

b. Составить описание ВМ в соответствии с вариантом

i. Описание набор регистров и банков памяти

ii. Описать набор инструкций: для каждой инструкции задать структуру операционного кода, содержащего описание операндов и набор операций, изменяющих состояние ВМ

1. Описать инструкции перемещения данных и загрузки констант

2. Описать инструкции арифметических и логических операций

3. Описать инструкции условной и безусловной передачи управления

4. Описать инструкции ввода-вывода с использованием скрытого регистра в качестве порта ввода-вывода

iii. Описать набор мнемоник, соответствующих инструкциям ВМ

c. Подготовить скрипт для запуска ассемблированного листинга с использованием описания ВМ:

i. Написать тестовый листинг с использованием подготовленных мнемоник инструкций

ii. Задействовать транслятор листинга в бинарный модуль по описанию ВМ

iii. Запустить полученный бинарный модуль на исполнение и получить результат работы

iv. Убедиться в корректности функционирования всех инструкций ВМ

2. Выбрать и изучить прикладную архитектуру системы команд существующей ВМ

a. Для выбранной ВМ:

i. Должен существовать готовый эмулятор (например qemu)

ii. Должен существовать готовый тулчейн (набор инструментов разработчика): компилятор Си, ассемблер и дизассемблер, линковщик, желательно отладчик

b. Согласовать выбор ВМ с преподавателем

c. Изучить модель памяти и набор инструкций ВМ

d. Научиться использовать тулчейн (собирать и запускать программы из листинга)

e. Подготовить скрипт для запуска ассемблированного листинга с использованием эмулятора

i. Написать тестовый листинг с использованием инструкций ВМ

ii. Задействовать ассемблер и компоновщик из тулчейна

iii. Запустить бинарный модуль на исполнение и получить результат его работы

Порядок выполнения:

1. Описать структуры данных, необходимые для представления информации об элементах образа программы (последовательностях инструкций и данных), расположенных в памяти

a. Для каждой инструкции – имя мнемоники и набор операндов в терминах данной ВМ

b. Для элемента данных – соответствующее литеральное значение или размер экземпляра типа данных в байтах

2. Реализовать модуль, формирующий образ программы в линейном коде для данного набора подпрограмм

a. Программный интерфейс модуля принимает на вход структуру данных, содержащую графы потока управления и информацию о локальных переменных и сигнатурах для набора подпрограмм, разработанную в задании 2 (п. 1.a, п. 2.b)

b. В результате работы порождается структура данных, разработанная в п. 1, содержащая описание образа программы в памяти: набор именованных элементов данных и набор именованных фрагментов линейного кода, представляющих собой алгоритмы подпрограмм

c. Для каждой подпрограммы посредством обхода узлов графа потока управления в порядке топологической сортировки (начиная с узла, являющегося первым базовым блоком алгоритма подпрограммы), сформировать набор именованных групп инструкций, включая пролог и эпилог подпрограммы (формирующие и разрушающие локальное состояние подпрограммы)

d. Для каждого базового блока в составе графа потока управления сформировать группу инструкций, соответствующих операциям в составе дерева операций

e. Использовать имена групп инструкций для формирования инструкций перехода между блоками инструкций, соответствующих узлам графа потока управления в соответствии с дугами в нём

3. Доработать тестовую программу, разработанную в задании 2 для демонстрации работоспособности созданного модуля

a. Добавить поддержку аргумента командной строки для имени выходного файла, вывод информации о графах потока управления сделать опциональных

b. Использовать модуль, разработанный в п. 2 для формирования образа программы на основе информации, собранной в результате работы модуля, созданного в задании 2 (п. 2.b)

c. Для сформированного образа программы в линейном коде вывести в выходной файл ассемблерный листинг, содержащий мнемоническое представление инструкций и

данных, как они описаны в структурах данных (п. 1), построенных разработанным модулем (пп. 2.c-e)

d. Проверить корректность решения посредством сборки сгенерированного листинга и запуска

полученного бинарного модуля на эмуляторе ВМ (см. подготовка п. 1.c или п. 2.e)

4. Результаты тестирования представить в виде отчета, в который включить:

a. В части 3 привести описание разработанных структур данных

b. В части 4 описать программный интерфейс и особенности реализации разработанного модуля

c. В части 5 привести примеры исходных текстов, соответствующие ассемблерные листинги и примера вывода запущенных тестовых программ

Архитектура по варианту: Динамическая типизация, код стековый, банки памяти (код, константы данные).

Разработанная архитектура:

```
architecture c21 {

/*
    case
    data word length    2
    code model          Стековый
    spaces              код, константы, данные
*/

registers:

    storage sp [32];
    storage bp [32];
    storage ip [32];

    storage inp [8];
    storage outp [8];

memory:

    range constant_ram [0x0000 .. 0xffffffff] {
        cell = 8;
        endianess = big-endian;
        granularity = 2;
    }

    range code_ram [0x0000 .. 0xffffffff] {
        cell = 8;
        endianess = little-endian;
        granularity = 2;
    }

    range data_ram [0x0000 .. 0xffffffff] {
        cell = 8;
        endianess = little-endian;
        granularity = 2;
    }
```

```
instructions:

    encode imm16 field = immediate [16];
    encode imm32 field = immediate [32];
    encode imm64 field = immediate [64];

    instruction sum = { 1001 0001 } {
        sp = sp - 8; let a = data_ram:8[sp];
        sp = sp - 8; let b = data_ram:8[sp];
        data_ram:8[sp] = a + b; sp = sp + 8;
        ip = ip + 1;
    };

    instruction sub = { 1001 0010 } {
        sp = sp - 8; let a = data_ram:8[sp];
        sp = sp - 8; let b = data_ram:8[sp];
        data_ram:8[sp] = a - b; sp = sp + 8;
        ip = ip + 1;
    };

    instruction mul = { 1001 0011 } {
        sp = sp - 8; let a = data_ram:8[sp];
        sp = sp - 8; let b = data_ram:8[sp];
        data_ram:8[sp] = a * b; sp = sp + 8;
        ip = ip + 1;
    };

    instruction div = { 1001 0100 } {
        sp = sp - 8; let a = data_ram:8[sp];
        sp = sp - 8; let b = data_ram:8[sp];
        data_ram:8[sp] = a / b; sp = sp + 8;
        ip = ip + 1;
    };

    instruction not = { 1001 0101 } {
        sp = sp - 8; let a = data_ram:8[sp];
        data_ram:8[sp] = !a; sp = sp + 8;
        ip = ip + 1;
    };

    instruction neg = { 1001 0110 } {
        sp = sp - 8; let a = data_ram:8[sp];
        data_ram:8[sp] = -a; sp = sp + 8;
        ip = ip + 1;
    };

    instruction or = { 1001 0111 } {
        sp = sp - 8; let a = data_ram:8[sp];
        sp = sp - 8; let b = data_ram:8[sp];
        data_ram:8[sp] = a | b; sp = sp + 8;
        ip = ip + 1;
    };
```

```
instruction and = { 1001 1000 } {
    sp = sp - 8; let a = data_ram:8[sp];
    sp = sp - 8; let b = data_ram:8[sp];
    data_ram:8[sp] = a & b; sp = sp + 8;
    ip = ip + 1;
};

instruction greater = { 1001 1001} {
    sp = sp - 8; let a = data_ram:8[sp];
    sp = sp - 8; let b = data_ram:8[sp]
    if a>b then
        data_ram:8[sp] = 1 ; sp = sp + 8;
    else
        data_ram:8[sp] = 0 ; sp = sp + 8;
    ip = ip + 1;
};

instruction greaterOrEqual = { 1001 1001} {
    sp = sp - 8; let a = data_ram:8[sp];
    sp = sp - 8; let b = data_ram:8[sp]
    if a>=b then
        data_ram:8[sp] = 1 ; sp = sp + 8;
    else
        data_ram:8[sp] = 0 ; sp = sp + 8;
    ip = ip + 1;
};


instruction lessOrEqual = { 1001 1001} {
    sp = sp - 8; let a = data_ram:8[sp];
    sp = sp - 8; let b = data_ram:8[sp]
    if a<=b then
        data_ram:8[sp] = 1 ; sp = sp + 8;
    else
        data_ram:8[sp] = 0 ; sp = sp + 8;
    ip = ip + 1;
};


instruction pushImm64 = { 0001 0011, imm64 as value } {
    data_ram:8[sp] = value; sp = sp + 8;
    ip = ip + 9;
};

instruction hlt = { 0111 0000 } {
};

//end of frame
instruction dup = { 0000 0001 } {
    sp = sp - 8; let x = data_ram:8[sp];
    data_ram:8[sp] = x; sp = sp + 8;
    data_ram:8[sp] = x; sp = sp + 8;
    ip = ip + 1;
};
```

```
instruction loadIndirectData8 = { 0000 0010 } {
    sp = sp - 8; let ptr = data_ram:8[sp];
    data_ram:8[sp] = data_ram:8[ptr]; sp = sp + 8;
    ip = ip + 1;
};

instruction loadIndirectConst8 = { 0000 0100 } {
    sp = sp - 8; let ptr = constant_ram:8[sp];
    data_ram:8[sp] = data_ram:8[ptr]; sp = sp + 8;
    ip = ip + 1;
};

instruction storeIndirectData8 = { 0000 1000 } {
    sp = sp - 8; let value = data_ram:8[sp];
    sp = sp - 8; let ptr = data_ram:8[sp];
    data_ram:8[ptr] = value;
    ip = ip + 1;
};

instruction storeIndirectData4 = { 0000 1001 } {
    sp = sp - 8; let value = data_ram:8[sp];
    sp = sp - 8; let ptr = data_ram:8[sp];
    data_ram:4[ptr] = value;
    ip = ip + 1;
};

instruction storeIndirectData2 = { 0000 1010 } {
    sp = sp - 8; let value = data_ram:8[sp];
    sp = sp - 8; let ptr = data_ram:8[sp];
    data_ram:2[ptr] = value;
    ip = ip + 1;
};

instruction storeIndirectData1 = { 0000 1011 } {
    sp = sp - 8; let value = data_ram:8[sp];
    sp = sp - 8; let ptr = data_ram:8[sp];
    data_ram:1[ptr] = value;
    ip = ip + 1;
};

instruction brt = { 0001 0100, imm32 as target } {
    sp = sp - 8; let value = data_ram:8[sp];
    if value then
        ip = target;
    else
        ip = ip + 5;
};

instruction br = { 0001 0101, imm32 as target } {
    ip = target;
};

instruction bri = { 0001 0110 } {
    sp = sp - 8; let target = data_ram:8[sp];
    ip = target;
};
```

```
instruction bri = { 0001 0110 } {
    sp = sp - 8; let target = data_ram:8[sp];
    ip = target;
};

instruction allocSlots = { 0001 0111, imm32 as amount } {
    sp = sp + 8 * amount;
    ip = target + 5;
};

instruction call = { 0001 1000, imm32 as target } {
    data_ram:8[sp] = ip; sp = sp + 8;
    data_ram:8[sp] = bp; sp = sp + 8;
    bp = sp;
    ip = target + 5;
};

instruction calli = { 0001 1001 } {
    sp = sp - 8; let target = data_ram:8[sp];
    data_ram:8[sp] = ip; sp = sp + 8;
    data_ram:8[sp] = bp; sp = sp + 8;
    bp = sp;
    ip = target;
};

instruction ret = { 0001 1010 } {
    sp = sp - 8; let retVal = data_ram:8[sp];
    sp = bp;
    sp = sp - 8; bp = data_ram:8[sp];
    sp = sp - 8; let target = data_ram:8[sp];
    data_ram:8[sp] = retVal; sp = sp + 8;
    ip = target;
};
```

```
mnemonics:

    mnemonic hlt();

    format plain1 is "{1}";
    format plain2 is "{1}, {2}";

    mnemonic sum();
    mnemonic sub();
    mnemonic mul();
    mnemonic div();
    mnemonic not();
    mnemonic neg();
    mnemonic or();
    mnemonic and();
    mnemonic greater();
    mnemonic less();
    mnemonic greaterOrEqual();
    mnemonic lessOrEqual();

    mnemonic pushq for pushImm64(value) plain1;
    mnemonic brt(target) plain1;
    mnemonic br(target) plain1;
    mnemonic bri();
    mnemonic dup();
    mnemonic alsl(amount) plain1;
    mnemonic call(target) plain1;
    mnemonic calli(target) plain1;
    mnemonic ret;

    mnemonic ldd8 for loadIndirectData8();
    mnemonic ldc8 for loadIndirectConst8();
    mnemonic std8 for storeIndirectData8();
    mnemonic std4 for storeIndirectData4();
    mnemonic std2 for storeIndirectData2();
    mnemonic std1 for storeIndirectData1();
}
```

Вывод:

В ходе выполнения лабораторной работы, мною была разработана архитектура для последующего преобразования графа потока управления в ассемблерный код.