НИУ ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по дисциплине
«Системное программное обеспечение»
Вариант 15

Выполнил: студент группы P4116
Крюков Андрей
Преподаватель: Кореньков Юрий Дмитриевич

Санкт-Петербург
2023 г.

Задание:

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 опциональным
    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

Описание архитектуры по варианту

Вариант 15 - статическая типизация, регистровый трёхадресный код, 3 банка памяти: код, константы, данные.

Описание структур данных

Для удобства трансляции я решил преобразовать полученное в рамках первой лабораторной работы AST в более удобную структуру данных. Для этого я создал модуль preprocess_ast, который принимает на вход массив корней AST для каждой из процедур и возвращает массив структур, с которыми работать будет сильно удобнее

Пример:

```
struct preparedLiteral {
   preparedType type;
   char c_value;
   int i_value;
   char *s_value;
   BOOLEAN b_value;
   ASTNode *astNode;
};
struct preparedVar {
   preparedType type;
   char *identifier;
   preparedExpression *initValue;
   int isInitValueExists;
   ASTNode *astNode;

   // for function args
   char *label;
};
```

```
struct expressionsList {
   preparedExpression *expressions;
   int expressionsCount;
   ASTNode *astNode;
};
struct preparedCall {
   char *procedureName;
   expressionsList argumentExpressions;
   ASTNode *astNode;
};
struct preparedIndexer {
   preparedExpression *expression;
   expressionsList indexExpressions;
   ASTNode *astNode;
};
```

Описание разработанных модулей

После того, как отработал модуль preprocess_ast, полученные структуры подаются на вход модуля semantic_analyser. Здесь, путем нисходящего обхода, производятся проверки типов и проверка существования используемых символов с учетом областей видимости. В частности, для проверки существования символов используется модуль symbolic_table, реализующий удобную для этой цели структуру данных.

```
enum symbolCategory {
   SYMBOL_CATEGORY_FUNC,
   SYMBOL_CATEGORY_VAR
};
struct symbol {
   char *identifier;
   preparedType type;
   enum symbolCategory category;
   union ctx ctx;
   char *label;
};
struct symbolicTable {
   symbolicTable *parent;
   symbol *symbols;
   int symbolsCount;
   char *currentFuncId;
   int capacity;
};
```

Если на этом этапе не возникло ошибок, вызывается модуль asm_generator, который путем нисходящего обхода производит трансляцию в ассемблер.

```
int translate_ifs(preparedIf ifs, symbolicTable *table, char *lastGoThroughLabel) {
   put_comment("if")
   put_comment(ifs.statement.condition.astNode->value)
   if (translate_expression(ifs.statement.condition, table) != 0) { // r0 contains bool
expression
       return 1;
   };
   char *goThrough = labelName();
   char *elseLabel;
   pop("r0")
```

```
    if (ifs.elseStatementExists == 1) {
        elseLabel = labelName();
        jumpeq("r0", elseLabel) // if r0 == 0 goto else conditional statement
    } else {
        jumpeq("r0", goThrough) // if r0 == 0 goto else conditional statement
    }
    put_comment("then")
    if (translate_statement(ifs.statement.statement, table, lastGoThroughLabel) != 0) {
        return 1;
    }
    if (ifs.elseStatementExists == 1) {
        jump(goThrough)
        put_label(elseLabel)
        put_comment("else")
        if (translate_statement(ifs.elseStatement, table, lastGoThroughLabel) != 0) {
            return 1;
        }
    }
    put_comment("endif")
    put_label(goThrough)
    return 0;
}
```

После этого в файле с результатом работы программы появляется ассемблерный листинг, который подается на вход RemoteTasks.

Для выполнения сгенерированного кода была написана архитектура для эмулятора в соответствии с вариантом.

Примеры входных данных и результаты их обработки

Калькулятор

В качестве критерия успеха я поставил цель реализовать калькулятор, и на данном этапе развития программный модуль способен сгенерировать корректный и работающий ассемблерный код из такую программу:

```
int read();
void write(int );
#define true 1

void printNumber(int num){
    int revertedNum = 0;
    while (num != 0) {
        revertedNum = (revertedNum * 10) + (num % 10);
        num = num / 10;
    }
    while (revertedNum != 0) {
        write((revertedNum % 10) + 0x30);
        revertedNum = revertedNum / 10;
    }
    write(10);
}
void printError(){
    write(101);
    write(114);
    write(114);
    write(111);
```

```
    write(114);
    write(10);
}

void main() {
    int firstNumber = 0;
    int secondNumber = 0;
    int operation = 0;
    int state = 0;
    while(true){
        int i = read();
        if ((i >= 0x30) && (i <= 0x39)){
            int num = i - 0x30;
            if (state == 0){
                firstNumber = firstNumber * 10 + num;
            } else if (state == 2) {
                secondNumber = secondNumber * 10 + num;
            } else {
                state = -1;
                write(i);
                write(10);
                break;
            }
        } else if (state == 0){
            if (i == 0x2b){ // +
                state = 2;
                operation = 1;
            }else if (i == 0x2d){ // -
                state = 2;
                operation = 2;
            }else if (i == 0x2a){ // *
                state = 2;
                operation = 3;
            }else if (i == 0x2f){ // /
                state = 2;
                operation = 4;
            }
        } else if (state == 2) {
            state = 3;
            break;
        } else {
            write(i);
            write(10);
            state = -1;
            break;
        }
    }
    if (state == 3) {
        if (operation == 1){
            firstNumber = firstNumber + secondNumber;
        } else if (operation == 2) {
            firstNumber = firstNumber - secondNumber;
        } else if (operation == 3) {
            firstNumber = firstNumber * secondNumber;
        } else if (operation == 4) {
            firstNumber = firstNumber / secondNumber;
        }
        printNumber(firstNumber);
    } else {
```

```
        printError();
    }
}
```

Программа принимает на вход строку, соответствующую выражению \d+[\+\-*\/]\d+, и возвращает результат вычисления.

Программа с вызовом процедуры

```
int add(int num1, int num2){
   return num1 + num2;
}
void main() {
   add(1, 2);
}
```

```
[section data_ram]
label_0: dw 0x0
label_1: dw 0x0
[section code_ram]
   jump start
label_2:
   load label_0, r0
   push r0
   load label_1, r0
   push r0
   pop r1
   pop r0
   add r0, r1, r0
   push r0
   pop r0
   ret
   ret
start:
label_3:
   mov 1, r0
   push r0
   pop r0
   store r0, label_0
   mov 2, r0
   push r0
   pop r0
   store r0, label_1
   call label_2
   push r0
   ret
   jump halt
halt:
   hlt
```

Программа с ветвлением

```
void main() {
   bool k;
   if (1 < 3) {
       k = true;
   } else {
       k = false;
   }
}
```

```
[section data_ram]
label_1: dw 0x0
[section code_ram]
   jump start
start:
label_0:
   mov 1, r0
   push r0
   mov 3, r0
   push r0
   pop r1
   pop r0
   sub r0, r1, r0
   jumplt r0, label_2
   mov 0, r0
   jump label_3
label_2:
   mov -1, r0
label_3:
   push r0
   pop r0
   jumpeq r0, label_5
   mov 1, r0
   push r0
   pop r0
   store r0, label_1
   jump label_4
label_5:
   mov 0, r0
   push r0
   pop r0
   store r0, label_1
label_4:
   ret
   jump halt
halt:
   hlt
```

Программа с циклом while и оператором break

```
void main() {
    int i = 0;
    while (i < 5) {
        if ((i % 2) == 0) {
            break;
        }
    }
}
```

```
[section data_ram]
label_1: dw 0x0
[section code_ram]
    jump start
start:
label_0:
    mov 0, r0
    push r0
    pop r0
    store r0, label_1
label_2:
    load label_1, r0
    push r0
    mov 5, r0
    push r0
    pop r1
    pop r0
    sub r0, r1, r0
    jumplt r0, label_3
    mov 0, r0
    jump label_4
label_3:
    mov -1, r0
label_4:
    push r0
    pop r0
    jumpeq r0, label_5
    load label_1, r0
    push r0
    mov 2, r0
    push r0
    pop r1
    pop r0
    rem r0, r1, r0
    push r0
    mov 0, r0
    push r0
    pop r1
    pop r0
    sub r0, r1, r0
    jumpeq r0, label_6
    mov 0, r0
    jump label_7
label_6:
    mov 1, r0
label_7:
    push r0
    pop r0
    jumpeq r0, label_8
    jump label_5
label_8:
    jump label_2
label_5:
    ret
    jump halt
halt:
    hlt
```

Вывод

В ходе выполнения данной лабораторной работы было написано описание архитектуры для эмулятора, а также реализован транслятор в ассемблер для этой архитектуры.