Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по дисциплине

«Системное программное обеспечение»

Выполнила

: Ильинская Ольга

Вадимовна

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт-Петербург

2023

Задание

Подготовка к выполнению по одному из двух сценариев:

1. Составить описание виртуальной машины с набором инструкций и моделью памяти по варианту
    - Изучить нотацию для записи определений целевых архитектур
    - 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. Описатьнабормнемоник,соответствующихинструкциямВМ
    - 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. Запуститьполученныйбинарныймодульнаисполнениеи получить результат работы
        iv. Убедиться в корректности функционирования всех инструкций ВМ
2. Выбрать и изучить прикладную архитектуру системы команд существующей ВМ
    - 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
    - Согласовать выбор ВМ с преподавателем
    - Изучить модель памяти и набор инструкций ВМ
    - Научиться использовать тулчейн (собирать и запускать программы из листинга)
    - 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. Запуститьбинарныймодульнаисполнениеиполучить

результат его работы Порядок выполнения:

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

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

- 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)
- 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)

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

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

- 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)
- 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)
- Проверить корректность решения посредством сборки сгенерированного листинга и запуска полученного бинарного модуля на эмуляторе ВМ (см. подготовка п. 1.c или п. 2.e)

Ход работы

За тип переменной отвечает маска. Этот enum может принимать такие значения, как:

```
INT,
LONG,
BYTE,
BOOL,
CHAR
```

Размерность типов (в битах):

`INT`	12
`LONG`	16
`BYTE`	8
`BOOL`	1
`CHAR`	8

Типизация связывается с переменной в момент присваивания значения.

Был реализован модуль, который производит трансляцию в ассемблерный код.

Разработанная архитектура выглядит так:

```
architecture system_software {
   memory:
      range ddram [0x0000..0xffff] {
         cell = 16;
         endianess = big-endian;
         granularity = 0;
      }

      range cdram [0x0000..0xffff] {
         cell = 32;
         endianess = big-endian;
         granularity = 0;
      }

   registers:
      storage r0st [16];   /* регистр для вычислений */
      storage r1st [16];   /* регистр для вычислений */
      storage ip [16];     /* указатель команды */
      storage inp [8];     /* инпут */
      storage outp [8];    /* аутпут */
      storage spst [16];   /* указатель стека */
      storage sp_bst [16]; /* указатель на основание стека */
      storage ps_r[32];    /* регистр состояния */

      view r0 = r0st;
      view r1 = r1st;
      view sp = spst;
```

```
      view sp_b = sp_bst;
      view ZF = ps_r[0];   /* флаг нуля */
      view NF = ps_r[1];   /* флаг отрицательный */


   instructions:
      encode imm16 field = immediate[16] data;
      encode imm8 field = immediate[8] data;

      encode reg field = register {
         r0 = 00,
         r1 = 01,
         sp = 10,
         sp_b = 11
      };

      encode dtt_vl sequence = alternatives {
         dtt16 = {imm16 as value},
         dttsbr = {sp_bst as value},
         dttsr = {spst as value}
      };


      instruction ldc = { 1000 0000 0000 00, reg as to, imm16 as
value} {
         to = value;
         ip = ip + 4;
      };

      instruction ld = { 1100 0000 0000 0000 0000 0000 0000, reg as
to, reg as ptr} {
         to = ddram:2[ptr];
         ip = ip + 4;
      };

      instruction st = { 1101 0001 0000 0000 0000 0000 0000, reg as
from, reg as ptr} {
         ddram:2[ptr] = from;
         ip = ip + 4;
      };

      instruction add-r2r2r = {0001 0001 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 + from2;
         ip = ip + 4;
      };
      instruction add-r2r2val = {0001 0010 0000, reg as to, reg as
from, imm16 as value} {
         to = from + value;
         ip = ip + 4;
      };
      instruction add-r2val2val = {0001 0011 0000, reg as to, 00,
imm8 as value1, imm8 as value2} {
         to = value1 + value2;
         ip = ip + 4;
      };

      instruction sub-r2r2r = {0010 0001 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 - from2;
```

```
         ip = ip + 4;
      };
      instruction sub-r2r2val = {0010 0010 0000, reg as to, reg as
from, imm16 as value} {
         to = from - value;
         ip = ip + 4;
      };
      instruction sub-r2val2val = {0010 0011 0000, reg as to, 00,
imm8 as value1, imm8 as value2} {
         to = value1 - value2;
         ip = ip + 4;
      };

      instruction div-r2r2r = {0011 0001 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 / from2;
         ip = ip + 4;
      };
      instruction div-r2r2val = {0011 0010 0000, reg as to, reg as
from, imm16 as value} {
         to = from / value;
         ip = ip + 4;
      };
      instruction div-r2val2val = {0011 0011 0000, reg as to, 00,
imm8 as value1, imm8 as value2} {
         to = value1 / value2;
         ip = ip + 4;
      };

      instruction mul-r2r2r = {0100 0001 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 * from2;
         ip = ip + 4;
      };
      instruction mul-r2r2val = {0100 0010 0000, reg as to, reg as
from, imm16 as value} {
         to = from * value;
         ip = ip + 4;
      };
      instruction mul-r2val2val = {0100 0011 0000, reg as to, 00,
imm8 as value1, imm8 as value2} {
         to = value1 * value2;
         ip = ip + 4;
      };

      instruction and-r2r2r = {0101 0000 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 & from2;
         ip = ip + 4;
      };
      instruction or-r2r2r = {0110 0000 0000 0000 0000 0000, reg as
to, reg as from1, reg as from2, 00} {
         to = from1 | from2;
         ip = ip + 4;
      };

      encode jmpKind sequence = alternatives {
         simple = { 0 },
         complex = { 1 }
      };
```

```
      instruction jmp = {1111 000, sequence jmpKind, 0000 0000,
imm16 as target } {
         when simple then
            ip = target;
         else
            ip = ip + target;
      };

      instruction jmpIZ = {1111 1000 0000 0000, imm16 as target} {
         if ZF == 1 then
            ip = target;
         else
            ip = ip + 4;
      };
      instruction jmpGE = {1111 1100 0000, reg as v1, reg as v2,
imm16 as target} {
         if v1 > v2 then
            ip = target;
         else
            ip = ip + 4;
      };
      instruction jmpEQ = {1111 0100 0000, reg as v1, reg as v2,
imm16 as target} {
         if v1 == v2 then
            ip = target;
         else
            ip = ip + 4;
      };

      instruction push-r = {0000 0111 0000 0000 0000 0000 0000, reg
as vr, 00} {
         sp = sp - 2;

         ddram:2[sp] = vr;

         ip = ip + 4;
      };

      instruction push-v = {0000 0111 0000 0000, imm16 as value} {
         sp = sp - 2;

         ddram:2[sp] = value;

         ip = ip + 4;
      };


      instruction call = {1011 0000 0000 0000, imm16 as target} {
         sp = sp - 2;
         ddram:2[sp] = ip;
         sp = sp - 2;
         ddram:2[sp] = sp;
         sp = sp - 2;
         ddram:2[sp] = sp_b;
         sp_b = sp;

         ip = target;
      };
```

```
      instruction ret = {0000 1000 0000 0000 0000 0000 0000 0000} {
         ip = ddram:2[sp_b - 4];
         sp = ddram:2[sp_b - 2];
         sp_b = ddram:2[sp_b];
      };

      instruction init = {0000 0001 0000 0000 0000 0000 0000 0000} {
         sp = 0xFFFF;
         sp_b = sp;
         sp = sp - 2;

         ip = ip + 4;
      };

      instruction hlt = {0000 0010 0000 0000 0000 0000 0000 0000}
{};


   mnemonics:
      mnemonic hlt();
      mnemonic init();
      mnemonic ret();

      format plain1 is "{1}";
      format plain2 is "{1}, {2}";
      format plain3 is "{1}, {2}, {3}";

      mnemonic add for add-r2r2r (to, from1, from2) plain3,
                    for add-r2r2val (to, from, value) plain3,
                    for add-r2val2val (to, value1, value2) plain3;
      mnemonic sub for sub-r2r2r (to, from1, from2) plain3,
                    for sub-r2r2val (to, from, value) plain3,
                    for sub-r2val2val (to, value1, value2) plain3;
      mnemonic div for div-r2r2r (to, from1, from2) plain3,
                    for div-r2r2val (to, from, value) plain3,
                    for div-r2val2val (to, value1, value2) plain3;
      mnemonic mul for mul-r2r2r (to, from1, from2) plain3,
                    for mul-r2r2val (to, from, value) plain3,
                    for mul-r2val2val (to, value1, value2) plain3;
      mnemonic and-r2r2r(to, from1, from2) plain3;
      mnemonic or-r2r2r(to, from1, from2) plain3;

      mnemonic call(target) plain1;

      mnemonic push for push-r(vr) plain1,
                  for push-v(value) plain1;

      mnemonic sjmp for jmp (target) plain1 when simple;
      mnemonic cjmp for jmp (target) plain1 when complex;

      mnemonic jmpeq for jmpEQ(v1, v2, target) plain3;
      mnemonic jmpge for jmpGE(v1, v2, target) plain3;
      mnemonic jmpiz for jmpIZ(target) plain1;

      mnemonic store for st(from, ptr) plain2;
      mnemonic ldc(to, value) plain2;
      mnemonic load for ld(to, ptr) plain2;
   }
```

Вывод

Я написала архитектуру и модуль преобразования графа потока управления в ассемблерный код.