Тагир Ильясов, 2 курс ФРКТ

23 февраля 2025 г.

Эмулятор процессора на языке C++

При разработке любого процессора возникает потребность в верификации логического дизайна. Один из самых базовых методов отладки заключается в сравнении его работы (RTL-описание) с некой «идеальной» машиной (Обычно, является программой). «Идеальная» машина работает в точном соответствии с поведением полностью отлаженного процессора (в полном соответствии с документацией к процессору и т.п.). Реализовать программный эмулятор процессора намного проще, а, следовательно, сильно проще и *отладить*, чем RTL-дизайн процессора, поэтому, за идеальную машину мы примем именно его.

Для разработки эмулятора потребуется реализовать 2 программы:

1. **Ассемблер** - переводит программу, написанную для процессора на языке ассеблера, в машинный (бинарный) код.

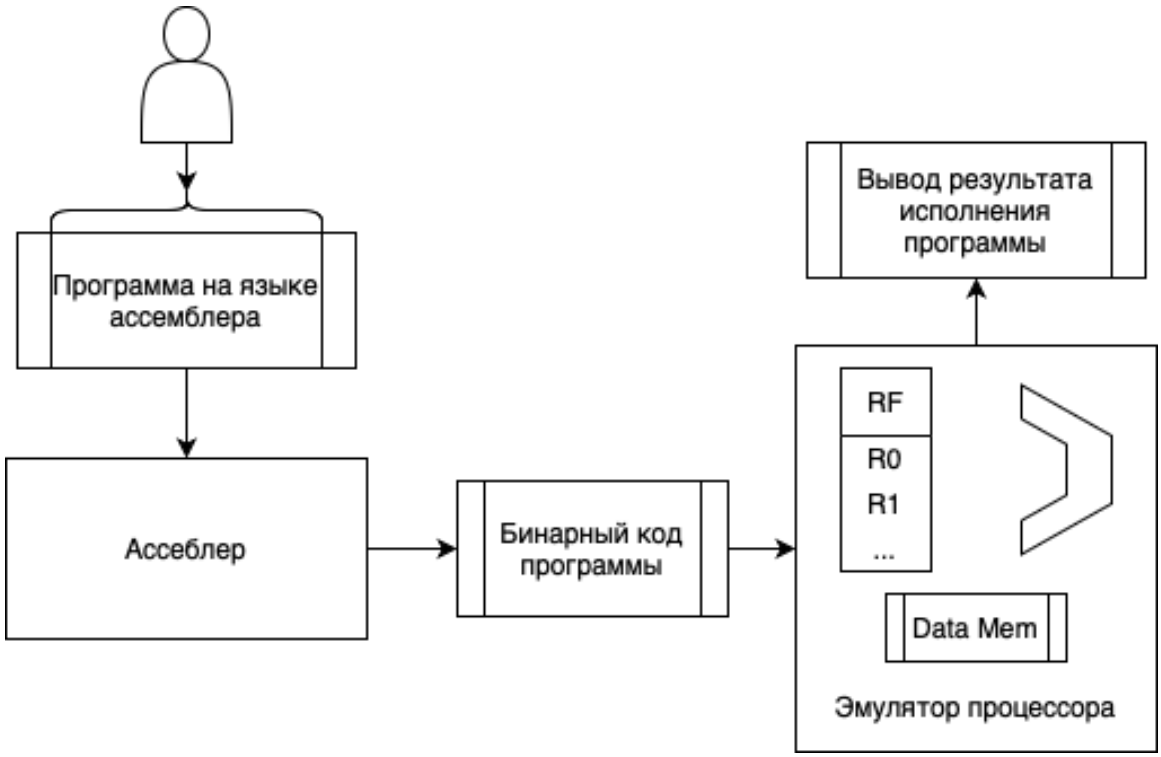
   (Вообще говоря, не является обязательной частью проекта, но так как процессор у нас свой - выдуманный, требуется для него написать и ассемблер)

2. **Эмулятор** - исполняет программу, записанную в виде машинного кода, так, как будто бы она исполнялась на реальной машине (теоретически).

Эмулятор будет *упрощённым* - процессор эмулируется с точностью исполнения машинного кода. Результат исполнения программы на эмуляторе должен быть неотличим от результата, полученного на реальной машине. По-тактово процессор *не эмулируется.* Программа на эмуляторе исполняется *«в иллюзии программиста»* - каждая инструкция выполняется в порядке написания программы (*последовательность*), следующая инструкция не обрабатывается, пока не завершится исполнение предыдущей (*атомарность*). Последовательность команд может быть изменена (*условно* из предыдущих вычислений или *безусловно*) спе-

циальными инструкциями. Память будем считать пронумерованным неограниченным массивом байтов, машина имеет доступ к любой ячейке в любой момент времени, для доступа к ним процессор использует номера соответствующих ячеек - их *адреса*. Память для простоты реализации разделена на *память инструкций* и *память данных*. То есть эмулятор будет представлять собой *программную реализацию машины на Гарвардской архитектуре* с определяемой далее в этом тексте архитектурой системы команд. Отличие Гарвардской архитектуры от архитектуры машины Фон-Неймана заключается в том, что в машине Ф-Н память инструкций и память данных составляют собой единое целое. В ходе исполнения программы машина выводит результаты исполнения, например, в командную строку.

Схема проекта



Архитектура системы команд (вымышленного) процессора

РЕГИСТРЫ.

- IP[15:0] — Instruction Pointer, указатель команды (при инициализации равен 0x00)
- RF[15:0][15:0] — Register File, регистровый файл, 16 регистров по 2Б каждый (при инициализации все регистры равны 0x00)

R0[15:0]
R1[15:0]
:
:
R14[15:0]
R15[15:0]

ФОРМАТЫ КОМАНДЫ.

Размер команд (instr_size) фиксированный - 4Б;

Ins	[31:24]	[23:16]	[15:8]	[7:0]
F1	opc[7:0]	src_0[7:0]	src_1[7:0]	dst[7:0]
F2	opc[7:0]	const[15:8]	const[7:0]	dst[7:0]
F3	opc[7:0]	src_0[7:0]	src_1[7:0]	src_2[7:0]
F4	opc[7:0]	src_0[7:0]	target[15:8]	target[7:0]

- opc[7:0] — код операции (см. далее);
- src_0,1,2[7:0] — номер входного операнда (один из регистров R0-R15)
- dst[7:0] — номер выходного операнда (один из регистров R0-R15)
- const[15:0] — операнд-литерал (константа), закодированный в команде;
- target[7:0] — IP целевой команды, используется в команде bnz;

ПАМЯТЬ КОМАНД/ДАННЫХ

Память неограниченна, доступен весь диапазон значений указателей на инструкции и данные в памяти. Пусть, все незаполненные ячейки памяти будут равными 0x00.

КОМАНДЫ.

- F1:
    - (opc=0x00) **nop**

        - No operation
    - (opc=0x01) **add** src0, src1, dst
        - RF[dst] <- RF[src_0] + RF[src_1]
    - (opc=0x02) **sub** src0, src1, dst
        - RF[dst] <-RF[src_0] — RF[src_1]
    - (opc=0x03) **mul** src0, src1, dst
        - {RF[dst+1], RF[dst]} <-RF[src_0] * RF[src_1]
          (если dst=15, dst+1=0)
    - (opc=0x04) **div** src0, src1, dst
        - RF[dst] <-RF[src_0] / RF[src_1]
    - (opc=0x05) **cmpge** src0, src1, dst
        - RF[dst] <- RF[src_0] >= RF[src_1]
    - (opc=0x06) **rshft** src0, src1, dst
        - RF[dst] <- RF[src_0] >> src_1
    - (opc=0x07) **lshft** src0, src1, dst
        - RF[dst] <- RF[src_0] << src_1
    - (opc=0x08) **and** src0, src1, dst
        - RF[dst] <- RF[src_0] & RF[src_1]
    - (opc=0x09) **or** src0, src1, dst
        - RF[dst] <- RF[src_0] | RF[src_1]
    - (opc=0x0a) **xor** src0, src1, dst
        - RF[dst] <- RF[src_0] ^ RF[src_1]
    - (opc=0x0b) **ld** [src0, src1], dst
        - RF[dst] <- MEM[ADDR][15:0]],
          где ADDR[15:0] = RF[src_0][15:0] + RF[src_1][15:0]
- F2:
    - (opc=0x0c) **set_const** const[7:0], dst
      RF[dst] <- {const_[15:8], const[7:0]}
- F3:
    - (opc=0x0d) **st** [src0, src1], src2
        - MEM[ADDR[15:0]] <- RF[src_2],
          где ADDR[15:0] = RF[src_0][15:0] + RF[src_1][15:0]
- F4:
    - (opc=0x0e) **bnz** target, src0
        - if(src0 != 0)
          IP ← target[15:0]
          else
          IP ← IP + instr_size
    - (opc=0x0f) **ready**
        - IP<-0; конец работы

Существует два сценария завершения исполнения кода задачи:

1. Исполнение команды *ready*
2. Исполнение команды с *IP=0xFFFC*, не являющейся переходом (окончание кода задачи в памяти команд).