Арифметические основы ЭВМ «RISC» — цикл лабораторных работ

М. М. Шихов

25 марта 2016 г.

Оглавление

Введение

Информатика — это дисциплина, посвящённая автоматизации процесса обработки[1] информации. Т.е. *информатика* рождается в тот момент, когда для обработки *информации* применяется *автоматика*.

Лабораторные работы по информатике посвящены арифметическим основам ЭВМ, т.е. методам обработки символьных представлений чисел.

[1] хранения, передачи

1 RISC

RISC (reduced instruction set of commands) processor — процессор с минималистичным набором команд.

Задание на лабораторную работу.

- Реализовать программу, выполняющую команды на языке ассемблера специальной RISC-машины. Система команд машины задается индивидуально.
- На языке ассемблера RISC-машины реализовать заданный в индивидуальном порядке алгоритм.

1.1 Архитектура специальной RISC-машины

Модель программиста специальной ВМ довольно проста:

- В машине имеется 8 байтовых (8-и битных) регистров `r0`,`r1`,...,`r7`.
- В машине доступна память данных из 256 8-битных ячеек с адресами от 0 до 255.
- Доступен единственный порт ввода 8-и битного числа, см. описание команды `in`.
- Доступен единственный порт вывода 8-и битного числа, см. описание команды `out`.
- Обращение к памяти возможно двумя способами:
  - Непосредственная адресация. Например, «`[0]`» — обращение к нулевой ячейке памяти.
  - Регистровая адресация. Например, «`[r1]`» — обращение к ячейке памяти с адресом, значение которого берется из регистра `r1`. Использовать для обращения к памяти можно любой из 8-и регистров.

Система команд машины минималистична. Конкретный набор команд задается для каждого варианта индивидуального задания. Расширять заданный набор инструкций нельзя.

Далее приводится описание всех возможных команд.

- «**in** `приемник`». Команда ввода байта из порта ввода в `приемник`. При выполнении этой команды пользователю предлагается ввести число (байт).
- «**out** `источник`». Команда вывода байта в порт вывода. При выполнении этой команды значение байта `источник` выводится на экран.
- «**ror** `сдвигаемое`, `количество-разрядов`, `результат`». Циклический сдвиг вправо на заданное количество разрядов.
- «**rol** `сдвигаемое`, `количество-разрядов`, `результат`». Циклический сдвиг влево на заданное количество разрядов.
- «**not** `операнд`, `результат`». Поразрядное логическое НЕ.
- «**or** `операнд1`, `операнд2`, `результат`». Поразрядное логическое ИЛИ.
- «**and** `операнд1`, `операнд2`, `результат`». Поразрядное логическое И.
- «**nor** `операнд1`, `операнд2`, `результат`». Поразрядное логическое ИЛИ-НЕ.
- «**nand** `операнд1`, `операнд2`, `результат`». Поразрядное логическое И-НЕ.
- «**xor** `операнд1`, `операнд2`, `результат`». Поразрядное логическое XOR.
- «**add** `операнд1`, `операнд2`, `результат`». Арифметическое сложение (с потерей единицы переноса).
- «**sub** `операнд1`, `операнд2`, `результат`». Арифметическое вычитание (с потерей единицы переноса).

$$\texttt{результат} = \texttt{операнд1} + \overline{\texttt{операнд2}} + 1.$$

- «**jz** `операнд`, `имя-метки`». Переход на метку, если все разряды `операнд` равны нулю.
- «**jo** `операнд`, `имя-метки`». Переход на метку, если все разряды `операнд` равны единице.

В общем случае:

- `результат`, `приемник` — это регистр или ячейка памяти;
- `операнд`, `источник`, `сдвигаемое`, `количество-разрядов` — это константа, регистр или ячейка памяти;

1.2 Требования к программе на ассемблере

- Программа оформляется в виде текстового файла, содержащего команды RISC-машины, комментарии, литералы целочисленных констант и метки.
- Литералы чисел могут представляться в десятичной, шестнадцатеричной (префикс «`0x`») и двоичной (префикс «`0b`») системах счисления. Например: `10`, `0xA`, `0b1010`.
- Разделителем команд является перевод строки.
- Количество пробелов-разделителей не имеет значения.
- Пустые строки игнорируются.
- В непустой строке текстового файла может быть команда, метка или комментарий.
- Метка задается как «`имя-метки:`»
- Имена меток (без двоеточия) используются в командах перехода.
- В программе не может двух и более меток с одинаковыми именами.
- Комментарий может следовать после команды или метки, признаком начала комментария всегда является символ «`;`».
- Признаком конца комментария является перевод строки.

1.3 Общие требования к выполнению программы на ассемблере

Необходимо реализовать выполнение команд RISC-машины из текстового файла. Выполнение начинается с начала файла. Комментарии и мнемоника выполняемой команды выводятся на экран. После каждой команды выполняется ожидание вода пользвателя. Если пользователь вводит:

- «`<Enter>`»

  Выполняется следующая команда.

- «`<Esc>`»

  Выполнение программы прерывается.

- «`r <Enter>`»

  Выводится значения всех 8-и регистров машины.

- «`m <базовый-адрес>:<количество-байт> <Enter>`»

  В шестнадцатеричном виде на экран выводятся значения соответствующих ячеек памяти.

- «`s <число-команд> <Enter>`»

  Выполняется заданное число команд без ожидания ввода после каждой команды.

- «`t <Enter>`»

  Выводится время выполнения последней команды и общее время выполнения программы в тактах. Правила подсчета времени определены в разделе 1.4.

После того, как будет достигнут конец файла, программа в бесконечном цикле считывает и интерпретирует ввод пользвателя, как описано выше.

1.4 Подсчет времени выполнения

Любая операция выполняется за один такт. Доступ к регистру осуществляется за один такт. Доступ к ячейке памяти осуществляется за 8 тактов. На обработку константы (и метки в командах перехода) времени не требуется. Например, общее время выполнения (19 тактов) команды

```
add [r1], r2, [5]
```

складывается как

1. доступ к регистру $t(\texttt{r1}) = 1$;
2. доступ к памяти $t(\texttt{[r1]}) = 8$;
3. доступ к регистру $t(\texttt{r2}) = 1$;
4. выполнение операции $t(\texttt{add}) = 1$;
5. доступ к памяти $t(\texttt{[5]}) = 8$.

На выборку команды процессор затрачивает 3 такта.

1.5 Варианты индивидуальных заданий

1.5.1 Варианты наборов команд RISC-машины

Реализовать программу, выполняющую программу на ассемблере RISC-машины, состоящую *только* из перечисленных для Вашего варианта команд.

Варианты:

1. **in**, **out**, **ror**, **or**, **not**, **add**, **jz**;
2. **in**, **out**, **ror**, **or**, **not**, **add**, **jo**;
3. **in**, **out**, **ror**, **or**, **not**, **sub**, **jz**;
4. **in**, **out**, **ror**, **or**, **not**, **sub**, **jo**;
5. **in**, **out**, **ror**, **nor**, **add**, **jz**;
6. **in**, **out**, **ror**, **nor**, **add**, **jo**;
7. **in**, **out**, **ror**, **nor**, **sub**, **jz**;
8. **in**, **out**, **ror**, **nor**, **sub**, **jo**;
9. **in**, **out**, **ror**, **nand**, **add**, **jz**;
10. **in**, **out**, **ror**, **nand**, **add**, **jo**;
11. **in**, **out**, **ror**, **nand**, **sub**, **jz**;
12. **in**, **out**, **ror**, **nand**, **sub**, **jo**;
13. **in**, **out**, **ror**, **xor**, **or**, **add**, **jz**;
14. **in**, **out**, **ror**, **xor**, **or**, **add**, **jo**;
15. **in**, **out**, **ror**, **xor**, **or**, **sub**, **jz**;
16. **in**, **out**, **ror**, **xor**, **or**, **sub**, **jo**;
17. **in**, **out**, **rol**, **or**, **not**, **add**, **jz**;
18. **in**, **out**, **rol**, **or**, **not**, **add**, **jo**;
19. **in**, **out**, **rol**, **or**, **not**, **sub**, **jz**;
20. **in**, **out**, **rol**, **or**, **not**, **sub**, **jo**;
21. **in**, **out**, **rol**, **nor**, **add**, **jz**;

22. **in**, **out**, **rol**, **nor**, **add**, **jo**;
23. **in**, **out**, **rol**, **nor**, **sub**, **jz**;
24. **in**, **out**, **rol**, **nor**, **sub**, **jo**;
25. **in**, **out**, **rol**, **nand**, **add**, **jz**;
26. **in**, **out**, **rol**, **nand**, **add**, **jo**;
27. **in**, **out**, **rol**, **nand**, **sub**, **jz**;
28. **in**, **out**, **rol**, **nand**, **sub**, **jo**;
29. **in**, **out**, **rol**, **xor**, **or**, **add**, **jz**;
30. **in**, **out**, **rol**, **xor**, **or**, **add**, **jo**;
31. **in**, **out**, **rol**, **xor**, **or**, **sub**, **jz**;
32. **in**, **out**, **rol**, **xor**, **or**, **sub**, **jo**;

1.5.2 Варианты алгоритмов для реализации на ассемблере

На ассемблере своей RISC-машины реализовать задание для Вашего варианта. Для каждого варианта определяется разрядность исходных данных, порядок байт и алгоритм, который требуется реализовать. Операнды должны быть считаны в заданном порядке из порта ввода. Результат должен быть выведен в порт вывода, также в заданном порядке.

Варианты задания:

1. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией I-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.
2. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией II-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.
3. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией III-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

4. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией IV-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

5. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка I-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

6. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка II-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

7. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка III-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

8. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка IV-м способом. Разядность операндов: 16 бит. Порядок байт: big-endian.

9. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией I-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

10. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией II-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

11. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией III-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

12. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией IV-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

13. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка I-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

14. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка II-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

15. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка III-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

16. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка IV-м способом. Разядность операндов: 24 бит. Порядок байт: big-endian.

17. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией I-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

18. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией II-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

19. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией III-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

20. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией IV-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

21. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка I-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

22. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка II-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

23. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка III-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

24. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка IV-м способом. Разядность операндов: 16 бит. Порядок байт: little-endian.

25. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией I-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

26. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией II-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

27. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией III-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

28. Алгоритмы: подсчета бит в числе и алгоритм умножения в дополнительном коде с автоматической коррекцией IV-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

29. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка I-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

30. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка II-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

31. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка III-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.

32. Алгоритмы: подсчета бит в числе и алгоритм умножения беззнаковых чисел с ускорением второго порядка IV-м способом. Разядность операндов: 24 бит. Порядок байт: little-endian.