实验报告

PB20061372 朱云沁

实验内容

在 LC-3 机器中，实现乘法，写出对应程序机器码。

实验要求

- 两个运算数分别放置于 `R0` 和 `R1`，结果存储到 `R7`。初始状态时，`R0` 和 `R1` 存放待计算数，其余寄存器全部为0。
- 程序应当通过以下测试样例：
  - 计算 1 * 1；
  - 计算 5 * 4000；
  - 计算 4000 * 5；
  - 计算 -500 * 433 （刻意溢出）；
  - 计算 -114 * -233 。
- 提交的机器码仅含乘法运算过程，不包括初始化及`HALT`指令。
- 编写两个版本：
  - **L版本**尽量编写更少的代码行数（小于14行）；
  - **P版本**尽量让程序执行更少的指令（平均条数小于130）。
- 评估程序的代码行数、完成实验功能所需要执行的指令数。

实验方法

1. 使用汇编语言，编写实现乘法功能的 *.asm* 文件（不含初始化）；
2. 使用 C++ 和 LC3Tools API，编写测试程序，用于汇编 *.asm* 文件，模拟 LC-3 机器，生成随机样例，检验程序正确性，并计算所有通过样例平均所执行的指令条数；
3. 将通过测试的 *.asm* 文件的程序主体部分转写为机器码，以文本形式存于 *.txt* 文件中。

测试程序的完整 C++ 代码如下：

```
#include <algorithm>
#include <chrono>
#include <iostream>
#include <random>

#define API_VER 2
#include "common.h"
#include "console_inputter.h"
#include "console_printer.h"
#include "interface.h"

using namespace std;

const int CASE_NUM = 100;
const string OBJ_SUFFIX = ".obj";
const string BIN_SUFFIX = ".bin";
uint32_t print_level = 4;
bool enable_liberal_asm = false;
bool ignore_privilege = false;
uint32_t inst_limit = 1919810;

int main(int argc, char* argv[]) {
  if (argc != 2) return 0;
  // Initialize
  lc3::ConsolePrinter printer;
  lc3::ConsoleInputter inputter;
  lc3::conv converter(printer, print_level);
  lc3::as assembler(printer, print_level, enable_liberal_asm);
  lc3::sim simulator(printer, inputter, print_level);
  simulator.setIgnorePrivilege(ignore_privilege);
  simulator.setRunInstLimit(inst_limit);
  string filename(argv[1]);
  if (filename.size() < OBJ_SUFFIX.size() ||
      !equal(OBJ_SUFFIX.rbegin(), OBJ_SUFFIX.rend(), filename.rbegin())) {
    filename =
        filename.size() > BIN_SUFFIX.size() &&
                equal(BIN_SUFFIX.rbegin(), BIN_SUFFIX.rend(), filename.rbegin())
            ? *converter.convertBin(filename)
            : assembler.assemble(filename)->first;
  }

  // Test
  uint64_t prev_count, sum = 0;
  int16_t num0, num1, result,
      sample[5][2] = {{1, 1}, {5, 4000}, {4000, 5}, {-500, 433}, {-114, -233}};
  mt19937 gen(unsigned(time(0)));
  uniform_int_distribution<int16_t> dis(0x0000, 0xffff);
  for (int i = 0; i < CASE_NUM; i++) {
    // Generate random numbers
    i < 5 ? (num0 = sample[i][0], num1 = sample[i][1])
```

```
            : (num0 = dis(gen), num1 = dis(gen));
      // Set machine state
      prev_count = simulator.getInstExecCount();
      simulator.zeroState();
      simulator.writeReg(0, num0);
      simulator.writeReg(1, num1);
      if (!simulator.loadObjFile(filename)) {
        cerr << "invalid obj file" << endl;
        return 0;
      }
      // Run and check
      simulator.runUntilHalt();
      result = static_cast<int16_t>(simulator.readReg(7));
      if ((int16_t)(num0 * num1) != result) {
        cerr << "wrong answer" << endl;
        return 0;
      }
      sum += simulator.getInstExecCount() - prev_count;
      if (i < 5) cout << num0 << " * " << num1 << " = " << result << endl;
      if (i == 4) cout << "average instruction count: " << 1.0 * sum / 5 << endl;
    }
    // Print result
    cout << CASE_NUM - 5 << " random cases passed" << endl;
    cout << "instruction count: " << sum << endl;
    cout << "average instruction count: " << 1.0 * sum / CASE_NUM << endl;
    return 0;
}
```

在 Powershell 中运行程序 *lab1_test.exe* 并加载 *lab1.asm*，输入、输出信息应具有如下形式：

```
PS C:\> .\lab1_test lab1.asm
attempting to convert lab1.asm into lab1.obj
conversion successful
1 * 1 = 1
5 * 4000 = 20000
4000 * 5 = 20000
-500 * 433 = -19892
-114 * -233 = 26562
average instruction count: 固定样例平均执行指令数
95 random cases passed
instruction count: 全部样例总执行指令数
average instruction count: 全部样例平均执行指令数
```

当某样例的计算结果不正确时，程序应在中途输出：

```
wrong answer
```

借此，程序的正确性以及执行效率得以评估。

实验成果

L版本(最初)

- **设计思路**

朴素思想：记16位补码形式的有符号整数构成集合$\mathbb{G}=\{g\in\mathbb{Z}|-2^{15}\leq g\leq 2^{15}-1\}$，模 2^{16} 剩余类加法群为 $\langle\mathbb{Z}_{65536},+\rangle$，双射$f:\mathbb{G}\to\mathbb{Z}_{65536},f(g)=[g]_{65536}$。设 $f(a)=\alpha,f(b)=\beta$，那么在 LC-3 机器中，将 a,b 相乘，其结果写成有符号整数应为 $f^{-1}(\alpha\cdot\beta)=f^{-1}(\alpha\cdot(b\ mod\ 2^{16}))=f^{-1}(\underbrace{\alpha+\cdots+\alpha}_{b\ mod\ 2^{16}})$。其中，$b\ mod\ 2^{16}$ 为 b 对应的无符号整数。又在 LC-3 机器中，将 a,b 相加，其结果写成有符号整数为 $f^{-1}(\alpha+\beta)$。故要将 `R0`，`R1` 的值相乘并存储结果到 `R7` 中，只需将 `ADD R7, R7, R0` 执行「 `R1` 的值（无符号整数）」次即可。

该算法的时间复杂度为 $\mathrm{O}((b-1)\ mod\ 2^{16}+1)$。

- **汇编码**

```
.ORIG x3000
    LOOP ADD R7, R7, R0
    ADD R1, R1, #-1
    BRnp LOOP
HALT
.END
```

- **机器码**

```
0001 111 111 000 000
0001 001 001 1 11111
0000 101 111111101
```

- **运行结果**

```
attempting to assemble lab1l.asm into lab1l.obj
assembly successful
1 * 1 = 1
5 * 4000 = 20000
4000 * 5 = 20000
-500 * 433 = -19892
-114 * -233 = 26562
average instruction count: 41846.2
95 random cases passed
instruction count: 8873494
average instruction count: 88734.9
```

- **程序评估**
  - 正确性：✓
  - 代码行数：3
  - 平均执行指令数：88734.9

该程序虽然仅有3行，对100个样例均给出正确结果，但执行效率极低，不具有实用性。显然，这是由于 `R1` 存储的无符号整数较大、为零、为负数造成的。当 `R1 = x0000` 时，最坏执行指令数达到 3×2^{16}。因此，需要在代码行数尚可接受的前提下，从原理上予以改进。

L版本(最终)

- **设计思路**

快速幂：前文提到，在 LC-3 机器中，将有符号整数 a, b 相乘，其结果写成有符号整数应为 $f^{-1}(\alpha \cdot \beta) = f^{-1}(\alpha \cdot (b \ mod \ 2^{16})) = f^{-1}(\alpha^{b \ mod \ 2^{16}})$，只需在群$\langle \mathbb{Z}_{65536}, + \rangle$ 中，求 α 的 $b \ mod \ 2^{16}$ 次幂。考虑 $b \ mod \ 2^{16}$ 的二进制拆分 $\sum_{i=0}^{15} b_i \cdot 2^i$，则 $\alpha^{b \ mod \ 2^{16}} = \sum_{i=0}^{15} \alpha^{b_i \cdot 2^i} = \sum_{i=0}^{15} b_i \cdot 2^i \alpha$。设 α 存储在 `R1` 中，那么 $2^i \alpha$ 可以通过 `ADD R1, R1, R1` 操作递推得到；设 $b \ mod \ 2^{16}$ 存储在 `R0` 中，位向量 2^i （递推得到）存储在 `R2` 中， b_i 存储在 `R3` 中，那么 b_i 可以通过 `AND, R3, R0, R2` 操作得到。显然，该问题在一个循环结构中即可得到解决，跳出循环的条件为位向量 $2^i = \mathrm{0x0000}$ 。

该算法的时间复杂度为 $\mathrm{O}(\log(b \ mod \ 2^{16})) = \mathrm{O}(1)$。

- **汇编码**

```
.ORIG x3000
    ADD R2, R2, #1
    LOOP AND R3, R0, R2
    BRz NEXT
    ADD R7, R7, R1
    NEXT ADD R1, R1, R1
    ADD R2, R2, R2
    BRnp LOOP
HALT
.END
```

- **机器码**

```
0001 010 010 1 00001
0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010
0000 101 111111010
```

- **运行结果**

```
attempting to assemble lab1l.asm into lab1l.obj
assembly successful
1 * 1 = 1
5 * 4000 = 20000
4000 * 5 = 20000
-500 * 433 = -19892
-114 * -233 = 26562
average instruction count: 88
95 random cases passed
instruction count: 8997
average instruction count: 89.97
```

- **程序评估**
  - 正确性：✓
  - 代码行数：7
  - 平均执行指令数：89.97

该程序对100个样例均给出正确结果，代码行数较少，执行效率较好。

因此，将该程序作为L版本，机器码存入 *ver\\lab1l.txt* 中。

P版本(最初)

- **设计思路**

在L版本中，控制循环的指令 `BRnp LOOP` 共累计16次，造成指令数的冗余。由于循环次数确定，可将循环结构改写为顺序结构，以空间效率换取时间效率。

理论上，平均执行指令将减少16次。

- **汇编码**

```
.ORIG x3000
    ADD R2, R2, #1

    AND R3, R0, R2
    BRz NEXT1
    ADD R7, R7, R1
    NEXT1 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT2
    ADD R7, R7, R1
    NEXT2 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT3
    ADD R7, R7, R1
    NEXT3 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT4
    ADD R7, R7, R1
    NEXT4 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT5
    ADD R7, R7, R1
    NEXT5 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT6
    ADD R7, R7, R1
    NEXT6 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT7
    ADD R7, R7, R1
    NEXT7 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT8
    ADD R7, R7, R1
    NEXT8 ADD R1, R1, R1
    ADD R2, R2, R2
```

```
51
52      AND R3, R0, R2
53      BRz NEXT9
54      ADD R7, R7, R1
55      NEXT9 ADD R1, R1, R1
56      ADD R2, R2, R2
57
58      AND R3, R0, R2
59      BRz NEXT10
60      ADD R7, R7, R1
61      NEXT10 ADD R1, R1, R1
62      ADD R2, R2, R2
63
64      AND R3, R0, R2
65      BRz NEXT11
66      ADD R7, R7, R1
67      NEXT11 ADD R1, R1, R1
68      ADD R2, R2, R2
69
70      AND R3, R0, R2
71      BRz NEXT12
72      ADD R7, R7, R1
73      NEXT12 ADD R1, R1, R1
74      ADD R2, R2, R2
75
76      AND R3, R0, R2
77      BRz NEXT13
78      ADD R7, R7, R1
79      NEXT13 ADD R1, R1, R1
80      ADD R2, R2, R2
81
82      AND R3, R0, R2
83      BRz NEXT14
84      ADD R7, R7, R1
85      NEXT14 ADD R1, R1, R1
86      ADD R2, R2, R2
87
88      AND R3, R0, R2
89      BRz NEXT15
90      ADD R7, R7, R1
91      NEXT15 ADD R1, R1, R1
92      ADD R2, R2, R2
93
94      AND R3, R0, R2
95      BRz NEXT16
96      ADD R7, R7, R1
97      NEXT16 ADD R1, R1, R1
98      ADD R2, R2, R2
99  HALT
100 .END
```

- **机器码**

  略

- **运行结果**

```
attempting to assemble lab1p.asm into lab1p.obj
assembly successful
1 * 1 = 1
5 * 4000 = 20000
4000 * 5 = 20000
-500 * 433 = -19892
-114 * -233 = 26562
average instruction count: 72
95 random cases passed
instruction count: 7379
average instruction count: 73.79
```

- **程序评估**
  - 正确性：✓
  - 代码行数：81
  - 平均执行指令数：73.79

该程序对100个样例均给出正确结果，与L版本相比，代码行数增多，但执行效率有一定的改进。平均执行指令减少约16次，与理论分析相符。

P版本(最终)

- **设计思路**

不难发现，在将循环结构改写为顺序结构后，位向量 2^i 的处理存在一定的改进空间：$2^0, 2^1, 2^2, 2^3$ 可以在 `imm5` 中直接得到；2^{15} 用任意负整数来代替，也可以在 `imm5` 中得到。由此，将 `R2` 的更新次数减少了5次。另外，由于不再需要判断循环结束， `R1` 的更新次数也将减少1次。

理论上，平均执行指令将再减少6次。

- **汇编码**

```
.ORIG x3000
    ADD R2, R2, #8

    AND R3, R0, #1
    BRz NEXT1
    ADD R7, R7, R1
    NEXT1 ADD R1, R1, R1

    AND R3, R0, #2
    BRz NEXT2
    ADD R7, R7, R1
    NEXT2 ADD R1, R1, R1

    AND R3, R0, #4
    BRz NEXT3
    ADD R7, R7, R1
    NEXT3 ADD R1, R1, R1

    AND R3, R0, R2
    BRz NEXT4
    ADD R7, R7, R1
    NEXT4 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT5
    ADD R7, R7, R1
    NEXT5 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT6
    ADD R7, R7, R1
    NEXT6 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT7
    ADD R7, R7, R1
    NEXT7 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT8
    ADD R7, R7, R1
    NEXT8 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT9
```

```
    ADD R7, R7, R1
    NEXT9 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT10
    ADD R7, R7, R1
    NEXT10 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT11
    ADD R7, R7, R1
    NEXT11 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT12
    ADD R7, R7, R1
    NEXT12 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT13
    ADD R7, R7, R1
    NEXT13 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT14
    ADD R7, R7, R1
    NEXT14 ADD R1, R1, R1
    ADD R2, R2, R2

    AND R3, R0, R2
    BRz NEXT15
    ADD R7, R7, R1
    NEXT15 ADD R1, R1, R1

    AND R3, R0, #-1
    BRzp NEXT16
    ADD R7, R7, R1
NEXT16 HALT
.END
```

- **机器码**

```
0001 010 010 1 01000

0101 011 000 1 00001
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001

0101 011 000 1 00010
0000 010 000 000 001
0001 111 111 000 001
0001 001 001 000 001

0101 011 000 100 100
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
```

```
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001
0001 010 010 000 010

0101 011 000 000 010
0000 010 000000001
0001 111 111 000 001
0001 001 001 000 001

0101 011 000 1 11111
0000 011 000000001
0001 111 111 000 001
```

- **运行结果**

```
attempting to assemble lab1p.asm into lab1p.obj
assembly successful
1 * 1 = 1
5 * 4000 = 20000
4000 * 5 = 20000
-500 * 433 = -19892
-114 * -233 = 26562
average instruction count: 66
95 random cases passed
instruction count: 6794
average instruction count: 67.94
```

- **程序评估**
  - 正确性：✓
  - 代码行数：75
  - 平均执行指令数：67.94

该程序对100个样例均给出正确结果，并且执行效率较高。与上一版相比，代码减少6行，平均执行指令减少约6次，与理论分析相符。

因此，将该程序作为P版本，机器码存入 *verp\lab1p.txt* 中。