



哈爾濱工業大學 (深圳)
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期: 2022 春季

课程名称: 计算机组成原理 (实验)

实验名称: 从 C 语言到机器码

实验性质: 综合设计型

实验学时: 2 地点:

学生班级: 计算机类 4 班

学生学号: 200110428

学生姓名: 杨杰睿

作业成绩:

实验与创新实践教育中心制

2022 年 3 月

1、实验结果截图

注：执行 `make` 会导致 `out/main.s` 原含有注释的代码被删除，请勿在检查前直接执行。除此之外，本人默认选择 `riscv64-linux-gnu-gcc` 作为编译器，如需使用范例所给的 `riscv64-unknown-elf-gcc`，请执行 `make unknown` 命令

```
efjerryyang@LAPTOP-LMAMBQ2N:/mnt/d/Projects/GitHub/computer-system/comp-organ/lab1/lab-submit$ make
Checking..
mkdir -p out
riscv64-linux-gnu-gcc -static -E src/main.c -o out/main.i
riscv64-linux-gnu-gcc -static -S out/main.i -o out/main.s -Og
riscv64-linux-gnu-gcc -static -c out/main.s -o out/main.o -march=rv64g
riscv64-linux-gnu-objdump -D out/main.o > out/main.objdump.s
riscv64-linux-gnu-gcc -static src/main.c -o out/main -Og
command: spike $(which pk) ./out/main
command: cat ./out/main.objdump.s
efjerryyang@LAPTOP-LMAMBQ2N:/mnt/d/Projects/GitHub/computer-system/comp-organ/lab1/lab-submit$ spike $(which pk) ./out/main
bbl loader
21952
efjerryyang@LAPTOP-LMAMBQ2N:/mnt/d/Projects/GitHub/computer-system/comp-organ/lab1/lab-submit$ tree
.
├── 200110428-杨杰睿-实验1-实验报告.docx
├── 200110428-杨杰睿-实验1-实验报告.pdf
├── Makefile
├── out
│   ├── main
│   ├── main.i
│   ├── main.o
│   ├── main.objdump.s
│   └── main.s
├── src
│   ├── main.c
│   └── multiply.h
└── 2 directories, 10 files
efjerryyang@LAPTOP-LMAMBQ2N:/mnt/d/Projects/GitHub/computer-system/comp-organ/lab1/lab-submit$
```

2、汇编代码注释（只需写主程序和子程序即可）

注：更为规整的格式请参见 `out/main.s` 代码行末注释。

```
.file "main.c"
.option pic
.text
.align 1
.globl multiply
.type multiply, @function
multiply:
    mv    a5,a0
    # 将参数寄存器 a0 的值复制到参数寄存器 a5，即 x 保存到 a5
    li    a0,0
    # 加载立即数 0 到寄存器 a0，即 result 初始化为 0
    j     .L2
    # 无条件跳转到.L2 标签
.L3:
    addw   a0,a0,a4
    # 将参数寄存器 a4 与参数寄存器 a0 的值相加，结果复制到 a0 中，即计算 result 加上 a4 寄存器中的值，a4 寄存器的值取决于(y&1)的结果，详见.L2 标签
    slli   a0,a0,48
    # 将参数寄存器 a0 的值逻辑左移 48 位，结果复制到 a0 中，即丢弃寄存器 a0 从 17
```

位到 64 位的值，即保证下一步计算后 result 的结果是 uint16_t

```
srli    a0,a0,48
```

将参数寄存器 a0 的值逻辑右移 48 位，结果复制到 a0 中，即将寄存器 a0 的高 48 位写为 0，结果保存在低 16 位中，即保证 result 的结果是 uint16_t

```
srli    a1,a1,1
```

将参数寄存器 a1 的值逻辑右移 1 位，结果复制到 a1 中，即将 a1 寄存器的值整除 2，即源代码中语句 `y >>= 1`

```
slliw   a5,a5,1
```

将参数寄存器 a5 的值逻辑左移 1 位，结果复制到 a5 中，即将 a5 寄存器的值乘以 2，即源代码中语句 `x <<= 1`

```
slli    a5,a5,48
```

将参数寄存器 a5 的值逻辑左移 48 位，结果复制到 a5 中，即丢弃寄存器 a5 从 17 位到 64 位的值，即保证下一步计算后 x 的类型是 uint16_t

```
srli    a5,a5,48
```

将参数寄存器 a5 的值逻辑右移 48 位，结果复制到 a5 中，即将寄存器 a5 的高 48 位写为 0，结果保存在低 16 位中，即保证 x 的类型是 uint16_t

.L2:

```
beqz    a1,.L6
```

若参数寄存器 a1 的值等于 0，就跳转到 .L6 标签，即 y 等于 0 的时候跳转到 .L6

```
andi    a4,a1,1
```

将 a1 的值和立即数 1 进行“按位与”运算，将结果复制到 a4 寄存器中，即计算 $(y \& 1)$ 的值

```
beqz    a4,.L3
```

若寄存器 a4 的内容值等于 0，就直接跳转到 .L3 标签，即 $(y \& 1)$ 等于 0 的时候跳转到 .L3

```
mv      a4,a5
```

将寄存器 a5 的值复制到寄存器 a4 中，即将 x 的值保存到寄存器 a4

```
j       .L3
```

无条件跳转到 .L3 标签

.L6:

```
ret # 函数返回,跳转到上层调用者处,返回值在参数寄存器 a0 中,即返回值为 result
```

```
.size multiply,.-multiply
```

```
.section .rodata.str1.8,"aMS",@progbits,1
```

```
.align 3 # 对齐为 8 bytes
```

.LC0:

```
.string "%u\n" # 字符串 "%u\n"
```

```
.text
```

```
.align 1 # 对齐为 1 byte
```

```
.globl main
```

```
.type main, @function
```

main:

```
addi    sp,sp,-16
```

将堆栈指针寄存器 sp 与立即数 (-16) 相加，再存入堆栈指针寄存器 sp，即 $sp = sp + (-16)$

```

sd ra,8(sp)
# 将 ra 寄存器的内容写入 sp 所指向地址加 8 的偏移量,即将上级调用者的返回地址
写入方才分配的栈空间最高的双字中(栈空间总共分配了 2 个双字,当前最高位的空间写入
了 ra)
li a1,28
# 将立即数 28 加载到参数寄存器 a1 中,即本例源码中的调用 multiply(x,x)的第 2
个 x 参数
li a0,28
# 将立即数 28 加载到参数寄存器 a0 中,即本例源码中的调用 multiply(x,x)的第 1
个 x 参数
call multiply
# 将 main 部分下一条需要执行的指令地址写入 ra 寄存器,调用函数 multiply,返
回值 y 保存在 a0 参数寄存器中
li a1,28
# 将立即数 28 加载到参数寄存器 a1 中,即本例源码中的调用 multiply(y,x)的第 2
个 x 参数,y 的值保存在 a0 中
call multiply
# 将 main 部分下一条需要执行的指令地址写入 ra 寄存器,调用函数 multiply,返
回值 result 保存在 a0 参数寄存器中
sext.w a1,a0
# 将 a0 参数寄存器的值复制到 a1 中,进行 32 位符号扩展(sign extend word),此
处应该是打印时的格式化指示符%u 所致
lla a0,.LC0
# 将.LC0 标签地址加载到参数寄存器 a0 中,即将字符串"%u\n"地址复制到 a0 中
call printf@plt
# 将 main 部分下一条需要执行的指令地址写入 ra 寄存器,调用函数 printf@plt,
返回值读取的参数个数保存在 a0 参数寄存器中
li a0,0
# 将立即数 0 加载到参数寄存器 a0 中,即 main 返回值为 0,代表正常退出。
ld ra,8(sp)
# 将堆栈指针寄存器 sp 所指向高 8 字节地址的值写入 ra 寄存器中,即将保存的上级
调用者的返回地址从栈空间的最高双字取出复制到 ra 中
addi sp,sp,16
# 将堆栈指针寄存器 sp 与立即数 16 相加,再存入堆栈指针寄存器 sp,即 sp = sp +
16
jr ra
# 返回上级调用者,返回地址为 ra,返回值为 a0,即 main 函数返回上级并返回 0
.size main,.-main
.ident "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04) 9.4.0"
.section .note.GNU-stack,"",@progbits

```

3、机器码注释（只需写主程序和子程序即可）

示例:

注: 更便于查看的文本请参见 `out/main.decoder.s` 的内容.

```
Disassembly of section .text:

0000000000000000 <multiply>:
    0: 00050793          mv  a5,a0
bit-level representation:
    00000000000001010000011110010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1  | funct3 | rd   | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 01010 | 000   | 01111 | 0010011 |
+-----+-----+-----+-----+-----+

    4: 00000513          li  a0,0
bit-level representation:
    00000000000000000000010100010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1  | funct3 | rd   | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 00000 | 000   | 01010 | 0010011 |
+-----+-----+-----+-----+-----+

    8: 0200006f          j  28 <.L2>
bit-level representation:
    0000001000000000000000001101111
+-----+-----+-----+-----+-----+
| imm[20:10:1|11|19:12] | rd   | opcode |
+-----+-----+-----+-----+-----+
| 0000001000000000000000 | 00000 | 1101111 |
+-----+-----+-----+-----+-----+

0000000000000000c <.L3>:
    c: 00e5053b          addw a0,a0,a4
bit-level representation:
    00000000111001010000010100111011
+-----+-----+-----+-----+-----+
| funct7 | rs2  | rs1  | funct3 | rd   | opcode |
+-----+-----+-----+-----+-----+
| 0000000 | 01110 | 01010 | 000   | 01010 | 0111011 |
+-----+-----+-----+-----+-----+
```

```

10: 03051513          slli  a0,a0,0x30
bit-level representation:
00000011000001010001010100010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000110000 | 01010 | 001  | 01010 | 0010011 |
+-----+-----+-----+-----+-----+

14: 03055513          srli  a0,a0,0x30
bit-level representation:
00000011000001010101010100010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000110000 | 01010 | 101  | 01010 | 0010011 |
+-----+-----+-----+-----+-----+

18: 0015d593          srli  a1,a1,0x1
bit-level representation:
000000000000101011101010110010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 0000000000001 | 01011 | 101  | 01011 | 0010011 |
+-----+-----+-----+-----+-----+

1c: 0017979b          slliw a5,a5,0x1
bit-level representation:
000000000000101111001011110011011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 0000000000001 | 01111 | 001  | 01111 | 0011011 |
+-----+-----+-----+-----+-----+

20: 03079793          slli  a5,a5,0x30
bit-level representation:
00000011000001111001011110010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000110000 | 01111 | 001  | 01111 | 0010011 |
+-----+-----+-----+-----+-----+

```

```

+-----+-----+-----+-----+-----+
24: 0307d793          srli  a5,a5,0x30
bit-level representation:
00000011000001111101011110010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000110000 | 01111 | 101  | 01111 | 0010011 |
+-----+-----+-----+-----+-----+

000000000000000028 <.L2>:
28: 00058a63          beqz  a1,3c <.L6>
bit-level representation:
00000000000001011000101001100011
+-----+-----+-----+-----+-----+
| imm[12|10:5] | rs2 | rs1 | funct3 | imm[4:1|11] | opcode |
+-----+-----+-----+-----+-----+
| 00000000 | 00000 | 01011 | 000  | 10100  | 1100011 |
+-----+-----+-----+-----+-----+

2c: 0015f713          andi  a4,a1,1
bit-level representation:
000000000000101011111011100010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 0000000000001 | 01011 | 111  | 01110 | 0010011 |
+-----+-----+-----+-----+-----+

30: fc070ee3          beqz  a4,c <.L3>
bit-level representation:
11111100000001110000111011100011
+-----+-----+-----+-----+-----+
| imm[12|10:5] | rs2 | rs1 | funct3 | imm[4:1|11] | opcode |
+-----+-----+-----+-----+-----+
| 1111110  | 00000 | 01110 | 000  | 11101  | 1100011 |
+-----+-----+-----+-----+-----+

34: 00078713          mv    a4,a5
bit-level representation:
00000000000001111000011100010011
+-----+-----+-----+-----+-----+

```

```

| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000000000 | 01111 | 000 | 01110 | 0010011 |
+-----+-----+-----+-----+

```

```

38: fd5ff06f          j c <.L3>

```

bit-level representation:

```

11111101010111111111000001101111
+-----+-----+-----+
| imm[20|10:1|11|19:12] | rd | opcode |
+-----+-----+-----+
| 11111101010111111111 | 00000 | 1101111 |
+-----+-----+-----+

```

```

00000000000000003c <.L6>:

```

```

3c: 00008067          ret

```

bit-level representation:

```

00000000000000001000000001100111
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000000000 | 00001 | 000 | 00000 | 1100111 |
+-----+-----+-----+-----+

```

```

000000000000000040 <main>:

```

```

40: ff010113          addi sp,sp,-16

```

bit-level representation:

```

11111111000000010000000100010011
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 111111110000 | 00010 | 000 | 00010 | 0010011 |
+-----+-----+-----+-----+

```

```

44: 00113423          sd ra,8(sp)

```

bit-level representation:

```

000000000000100010011010000100011
+-----+-----+-----+-----+-----+
| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
+-----+-----+-----+-----+-----+
| 00000000 | 00001 | 00010 | 011 | 01000 | 0100011 |
+-----+-----+-----+-----+-----+

```



```

48: 01c00593          li a1,28
bit-level representation:
0000000111000000000010110010011
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000011100 | 00000 | 000 | 01011 | 0010011 |
+-----+-----+-----+-----+

4c: 01c00513          li a0,28
bit-level representation:
0000000111000000000010100010011
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000011100 | 00000 | 000 | 01010 | 0010011 |
+-----+-----+-----+-----+

50: 00000097          auipc ra,0x0
bit-level representation:
00000000000000000000000010010111
+-----+-----+-----+
|      imm[31:12]      | rd | opcode |
+-----+-----+-----+
| 00000000000000000000 | 00001 | 0010111 |
+-----+-----+-----+

54: 000080e7          jalr ra # 50 <main+0x10>
bit-level representation:
00000000000000001000000011100111
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000000000 | 00001 | 000 | 00001 | 1100111 |
+-----+-----+-----+-----+

58: 01c00593          li a1,28
bit-level representation:
0000000111000000000010110010011
+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+
| 000000011100 | 00000 | 000 | 01011 | 0010011 |
+-----+-----+-----+-----+

```

```

5c: 00000097          auipc ra,0x0
bit-level representation:
00000000000000000000000010010111
+-----+-----+-----+
| imm[31:12] | rd | opcode |
+-----+-----+-----+
| 00000000000000000000 | 00001 | 0010111 |
+-----+-----+-----+

60: 000080e7          jalr ra # 5c <main+0x1c>
bit-level representation:
000000000000000001000000011100111
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 00001 | 000 | 00001 | 1100111 |
+-----+-----+-----+-----+-----+

64: 0005059b          sext.w a1,a0
bit-level representation:
00000000000001010000010110011011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 01010 | 000 | 01011 | 0011011 |
+-----+-----+-----+-----+-----+

68: 00000517          auipc a0,0x0
bit-level representation:
0000000000000000000000010100010111
+-----+-----+-----+-----+-----+
| imm[31:12] | rd | opcode |
+-----+-----+-----+-----+-----+
| 00000000000000000000 | 01010 | 0010111 |
+-----+-----+-----+-----+-----+

6c: 00050513          mv a0,a0
bit-level representation:
00000000000001010000010100010011
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 01010 | 000 | 01010 | 0010011 |

```

```

+-----+-----+-----+-----+-----+
70: 00000097          auipc ra,0x0
bit-level representation:
00000000000000000000000010010111
+-----+-----+-----+-----+
|      imm[31:12]      |   rd   | opcode |
+-----+-----+-----+-----+
| 00000000000000000000 | 00001 | 0010111 |
+-----+-----+-----+-----+

74: 000080e7          jalr ra # 70 <main+0x30>
bit-level representation:
000000000000000000001000000011100111
+-----+-----+-----+-----+-----+
| imm[11:0] |   rs1  | funct3 |   rd   | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 00001 |   000   | 00001 | 1100111 |
+-----+-----+-----+-----+-----+

78: 00000513          li a0,0
bit-level representation:
00000000000000000000000010100010011
+-----+-----+-----+-----+-----+
| imm[11:0] |   rs1  | funct3 |   rd   | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 00000 |   000   | 01010 | 0010011 |
+-----+-----+-----+-----+-----+

7c: 00813083          ld ra,8(sp)
bit-level representation:
000000001000000010011000010000011
+-----+-----+-----+-----+-----+
| imm[11:0] |   rs1  | funct3 |   rd   | opcode |
+-----+-----+-----+-----+-----+
| 000000001000 | 00010 |   011   | 00001 | 0000011 |
+-----+-----+-----+-----+-----+

80: 01010113          addi sp,sp,16
bit-level representation:
0000000010000000010000000100010011
+-----+-----+-----+-----+-----+
| imm[11:0] |   rs1  | funct3 |   rd   | opcode |
+-----+-----+-----+-----+-----+

```

```
| 000000010000 | 00010 | 000 | 00010 | 0010011 |
+-----+-----+-----+-----+-----+

84: 00008067          ret
bit-level representation:
00000000000000001000000001100111
+-----+-----+-----+-----+-----+
| imm[11:0] | rs1 | funct3 | rd | opcode |
+-----+-----+-----+-----+-----+
| 000000000000 | 00001 | 000 | 00000 | 1100111 |
+-----+-----+-----+-----+-----+
```