

# 中国科学技术大学

# 实验报告



## 计算机系统详解

## Data Lab

学生姓名： 朱云沁

学生学号： PB20061372

完成时间： 二〇二二年三月二十九日

## 目录

一、 简介	2
1. 实验目的	2
2. 实验要求	2
3. 实验环境	2
二、 实验结果	3
三、 实现思路	3
1. <code>bitXor</code>	3
2. <code>tmin</code>	4
3. <code>isTmax</code>	4
4. <code>allOddBits</code>	5
5. <code>negate</code>	5
6. <code>isAsciiDigit</code>	6
7. <code>conditional</code>	6
8. <code>isLessOrEqual</code>	7
9. <code>logicalNeg</code>	7
10. <code>howManyBits</code>	8
11. <code>floatScale2</code>	9
12. <code>floatFloat2Int</code>	10
13. <code>floatPower2</code>	10
四、 总结	11
附录 A 环境搭建	13
附录 B 代码清单	15
1. <code>bits.c</code>	15
2. <code>Dockerfile</code>	17

## 一、简介

### 1. 实验目的

- 加深对计算机中整数及浮点数的位级表示的理解.
- 掌握 C 语言中位运算符的使用, 学会通过位操作来高效实现某些功能.

### 2. 实验要求

使用规定的运算符, 在一定的操作数内实现以下逻辑与算术函数:

Name	Description	Rating	Max ops
<code>bitXor(x,y)</code>	$x \oplus y$ using only <code>&amp;</code> and <code>~</code> .	1	14
<code>tmin()</code>	Smallest two's complement integer.	1	4
<code>isTmax(x)</code>	True only if $x$ is largest two's comp. integer.	1	10
<code>allOddBits(x)</code>	True only if all odd-numbered bits in $x$ set to 1.	2	12
<code>negate(x)</code>	Return $-x$ with using <code>-</code> operator.	2	5
<code>isAsciiDigit(x)</code>	True if $0x30 \leq x \leq 0x39$ .	3	15
<code>conditional</code>	Same as $x ? y : z$	3	16
<code>isLessOrEqual(x,y)</code>	True if $x \leq y$ , false otherwise.	3	24
<code>logicalNeg(x)</code>	Compute $!x$ without using <code>!</code> operator.	4	12
<code>howManyBits(x)</code>	Min. no. of bits to represent $x$ in two's comp.	4	90
<code>floatScale2(uf)</code>	Return bit-level equiv. of $2 * f$ for f.p. arg. $f$ .	4	30
<code>floatFloat2Int(uf)</code>	Return bit-level equiv. of $(int)f$ for f.p. arg. $f$ .	4	30
<code>floatPower2(x)</code>	Return bit-level equiv. of $2.0^x$ for integer $x$ .	4	30

表 1: 实验所给问题

对于前 10 个整数问题, 代码中不允许出现超过 8 位的常数. 对于后 3 个浮点数问题, 允许出现 `if` 等逻辑控制. 每个问题的具体限制, 参见[实现思路](#)中各小题代码的注释.

程序编写完毕后, 用测试程序 `btest` 检查程序功能的正确性, 并用调整过的 ANSI C 编译器 `d1c` 检查代码的合法性. 也可用 Perl 脚本 `driver.pl` 进行综合评定.

### 3. 实验环境

本实验所有程序均在以下环境编译并通过测试:

Machine	MacBook Pro 13"
SoC	Apple M1, 基于 ARM, 含 8 核 CPU、8 核 GPU 及 16GB RAM
OS	macOS Monterey 12.3.1
IDE	Visual Studio Code 1.65.2
Docker	Docker 20.10.13
Image	Ubuntu 20.04.4, x86-64 平台, 详见 <a href="#">环境搭建</a> 部分
Packages	GCC 9.4.0, Make 4.2.1, Perl 5.30.0

## 二、实验结果

依据要求完成 `bits.c` 文件 (见[代码清单](#)) 并修改 `Makefile` (见[环境搭建](#)), 在 VS Code 远程容器终端中运行 `driver.pl`, 得到结果如下图所示:

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PB20061372 at 1f128b80133f in /workspaces/Repositories/Computer-System/Labs/data/datalab-handout on masterxxx 22-04-13 - 3:22:11
perl driver.pl
1. Running './dlc -z' to identify coding rules violations.

2. Compiling and running './btest -g' to determine correctness score.
gcc -O -w -lm -o btest bits.c btest.c decl.c tests.c

3. Running './dlc -Z' to identify operator count violations.

4. Compiling and running './btest -g -r 2' to determine performance score.
gcc -O -w -lm -o btest bits.c btest.c decl.c tests.c

5. Running './dlc -e' to get operator count of each function.

Correctness Results      Perf Results
Points Rating Errors Points Ops Puzzle
1 1 0 2 7 bitXor
1 1 0 2 1 tmin
1 1 0 2 5 isTmax
2 2 0 2 7 allOddBits
2 2 0 2 2 negate
3 3 0 2 8 isAsciiDigit
3 3 0 2 7 conditional
3 3 0 2 7 isLessOrEqual
4 4 0 2 5 logicalNeg
4 4 0 2 30 howManyBits
4 4 0 2 11 floatScale2
4 4 0 2 11 floatFloat2Int
4 4 0 2 9 floatPower2

Score = 62/62 [36/36 Corr + 26/26 Perf] (110 total operators)

```

图 1: `driver.pl` 运行结果

可见, 程序的评估结果为满分, 正确性及性能均符合要求.

## 三、实现思路

### 1. `bitXor`

本小题要求用位非和位与实现位异或. 对任意逻辑变量  $A$  和  $B$ , 根据异或运算的定义, 有

$$A \oplus B = (A + B)(AB)'$$

应用摩根律, 得

$$A \oplus B = (A'B')'(AB)'$$

使用 C 语言位运算符表达即可. 代码如下:

```

/*
 * bitXor - x^y using only ~ and &
 *
 * Example: bitXor(4, 5) = 1
 *
 * Legal ops: ~ &
 *
 * Max ops: 14
 *
 * Rating: 1
 */

```

```
int bitXor(int x, int y) { return ~(x & y) & ~(~x & ~y); }
```

根据评测结果, 该函数功能正确, 共计使用 7 个运算符.

## 2. tmin

本小题要求返回用补码表示的 32 位有符号整数的最小值. 根据补码的定义, 有符号整数的最小值仅有符号位为 1, 其余位为 0, 答案应为 0x8000. 由于实验对常数大小的限制, 使用左移运算得到该值. 代码如下:

```
/*
 * tmin - return minimum two's complement integer
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 4
 *   Rating: 1
 */
int tmin(void) { return 1 << 31; }
```

根据评测结果, 该函数功能正确, 仅使用 1 个运算符.

## 3. isTmax

本小题要求判断一个数是否为有符号整数的最大值, 返回值相当于  $x == 0x7FFFFFFF$  或  $!(x \wedge 0x7FFFFFFF)$ . 然而, 由于常数和运算符的限制, 应当利用该整数特有的运算性质来间接判断.

经过简单观察, 发现该整数的反码恰等于其原码加一, 即  $\sim x == x+1$ , 或  $!(\sim x \wedge (x+1))$ . 然而, 0xFFFFFFFF 也满足此运算性质, 需要根据  $x+1 != 0$ , 或  $!!(x+1)$  来排除. (此处连续使用 2 个逻辑非, 是为了将任意非零值变换为 0x1, 下文涉及相同策略时不再特别指出.)

继续观察, 发现  $x+1$  在以上操作中反复出现, 因而仍有优化空间——可以考虑将  $x+1$  赋值给某个变量, 以便后续使用. 事实上, 对以上运算性质进一步思考, 发现该等式可以变换为  $0xFFFFFFFF - x == x+1$ , 或  $(x+1) + (x+1) == 0x0$ , 从而充分利用  $x+1$ . 综合上述讨论, 并运用摩根律, 得到如下代码:

```
/*
 * isTmax - returns 1 if x is the maximum, two's complement number,
 *           and 0 otherwise
 *   Legal ops: ! ~ & ^ | +
 *   Max ops: 10
 *   Rating: 1
 */
int isTmax(int x) {
```

```

x = x + 1;
return !(~x | (x + x));
}

```

根据评测结果, 该函数功能正确, 共计使用 5 个运算符.

#### 4. allOddBits

本小题要求判断一个 32 位二进制字的奇数位是否全为 1. 一种直接的思路是借助位掩码来判断, 记 `mask=0x55555555`, 那么 `x&mask==mask`, 或者说 `!(~x&~mask)`. 然而, 由于常数大小的限制, 无法直接赋值上述 `mask` 或 `~mask`.

存在两种解决方案, 一是通过移位和位或运算生成该位掩码, 二是通过移位和位与运算合并 `x` 的奇数位, 从而可用更短的位掩码 `0xAA` 来判断. 笔者采用第二种方案, 代码如下:

```

/*
 * allOddBits - return 1 if all odd-numbered bits in word set to 1
 *   where bits are numbered from 0 (least significant) to 31 (most significant)
 *   Examples allOddBits(0xFFFFFFFF) = 0, allOddBits(0xAAAAAAAA) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 2
 */
int allOddBits(int x) {
    x = x & x >> 16;
    return !(~(x & x >> 8) & 170);
}

```

根据评测结果, 该函数功能正确, 共计使用 7 个运算符.

#### 5. negate

本小题要求对一个整数取反, 即求其补码. 根据定义有 `-x==~x+1`. 代码如下:

```

/*
 * negate - return -x
 *   Example: negate(1) = -1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 5
 *   Rating: 2
 */
int negate(int x) { return ~x + 1; }

```

根据评测结果, 该函数功能正确, 共计使用 2 个运算符.

## 6. isAsciiDigit

本小题要判断一个 ASCII 字符是否为数字 0~9, 即  $0x30 \leq x \leq 0x39$ . 考虑  $x - 0x30$  和  $x - 0x3A$  的符号位, 应有  $x - 0x30 \gg 31 == 0x0$  和  $x - 0x3A \gg 31 == 0x-1$ . 此处, 应当注意右移运算扩展有符号数最高位的方式. 对常数取反进行优化, 有  $(x + \sim 57 \& \sim(x + \sim 47)) \gg 31 == 0x-1$ . 合法的返回值应是上述表达式的最低位, 只需与  $0x1$  做位与运算. 代码如下:

```
/*
 * isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0'
 * to '9') Example: isAsciiDigit(0x35) = 1. isAsciiDigit(0x3a) = 0.
 *               isAsciiDigit(0x05) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 3
 */
int isAsciiDigit(int x) { return (x + ~57 & ~(x + ~47)) >> 31 & 1; }
```

根据评测结果, 该函数功能正确, 共计使用 8 个运算符.

## 7. conditional

本小题要求用位运算实现三目运算符, 可以抽象为 2 选 1 数据选择器. 考虑到记号的简洁性, 不妨设地址信号  $X$ 、输入数据  $Y, Z$  和输出数据  $D$  均为一维逻辑变量, 则有

$$D = XY + X'Z$$

要对位向量  $x, y, z$  的每一位进行上述操作, 应将任意非零值  $x$  统一变换为  $0xFFFFFFFF$ . 可以先通过逻辑非运算将取值限定为 0 或 1, 再减去 1 得到预期的变换结果, 即  $x = !x + \sim 0$ . 代码如下:

```
/*
 * conditional - same as x ? y : z
 *   Example: conditional(2,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int conditional(int x, int y, int z) {
    x = !x + ~0;
    return (x & y) | (~x & z);
}
```

根据评测结果, 该函数功能正确, 共计使用 7 个运算符.

## 8. isLessOrEqual

本小题要求比较两个有符号整数的大小, 返回  $x \leq y$ . 由于补码运算的溢出问题, 或者说由于模  $2^{32}$  整数加群的结构, 不能简单通过  $y-x$  的符号位来判断.

对模  $2^{32}$  剩余类进行分析, 发现可能存在两种错误情况: 一是  $x$  为正整数,  $y$  为负整数, 但  $y-x$  为正整数; 二是  $x$  为负整数,  $y$  为非负整数, 但  $y-x$  为负整数. 应当在  $x$  和  $y$  符号相反时采用其他判断方法——显然, 只需返回  $x$  的符号位.

综上所述, 当  $(x \wedge y) \gg 31 == 0x-1$  (符号相反) 时, 返回  $x$  的符号位; 当  $(x \wedge y) \gg 31 == 0x0$  (符号相同) 时, 返回  $y-x$  的符号位取反, 或者直接返回  $x-y-1$  (即  $x+\sim y$ ) 的符号位.

能否将两种情况合为一个表达式? 答案是肯定的, 只需将  $\sim((x \wedge y) \gg 31)$  作为位掩码, 用于决定是否将  $x$  加上  $\sim y$ . 进一步利用摩根律, 得到最终代码如下:

```
/*
 * isLessOrEqual - if x <= y then return 1, else return 0
 *   Example: isLessOrEqual(4,5) = 1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 24
 *   Rating: 3
 */
int isLessOrEqual(int x, int y) { return (x + ~((x ^ y) >> 31 | y)) >> 31 & 1; }
```

根据评测结果, 该函数功能正确, 共计使用 7 个运算符.

## 9. logicalNeg

本小题要求用位运算实现逻辑非: 将任意非零值映射为 0, 而将 0 映射为 1. 相较于 0, 非零整数的特征在于其自身和相反数必有一个符号位为 1. 遵循这个思路, 再借助右移运算, 可将任意非零值映射为  $0xFFFFFFFF$ , 加上 1 便得到预期返回值. 代码如下:

```
/*
 * logicalNeg - implement the ! operator, using all of
 *               the legal operators except !
 *   Examples: logicalNeg(3) = 0, logicalNeg(0) = 1
 *   Legal ops: ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 4
 */
int logicalNeg(int x) { return ((x | (~x + 1)) >> 31) + 1; }
```

根据评测结果, 该函数功能正确, 共计使用 5 个运算符.



## 10. howManyBits

本小题要求返回用补码表示某个整数需要的最少二进制位数. 由于该整数已由 32 位补码给出, 经分析, 需进行如下处理:

对于符号位为 0 的非负整数, 找到最高位的 1 所在位置; 对于符号位为 1 的负整数, 找到最高位的 0 所在位置. 取该位置之后的所有位并加上符号位, 就是表示该整数的最短补码.

为了统一上述两种情况而节省操作数, 首先将  $x$  赋值为  $x \gg 31 \wedge x$ , 使得负数取反, 非负数保持不变. 然后, 采用分治策略查找最高位的 1:

1. 若  $x$  的高 16 位有 1 (即  $!!(x \gg 16)$ ), 则将  $x$  右移 16 位, 在高 16 位继续查找; 否则, 在低 16 位继续查找.
2. 将  $x$  视作 16 位补码, 若  $x$  的高 8 位有 1 (即  $!!(x \gg 8)$ ), 则将  $x$  右移 8 位, 在高 8 位继续查找; 否则, 在低 8 位继续查找.
3. 将  $x$  视作 8 位补码, 依此类推, 直到找到最高位的 1 或找不到 1.

上述分治过程为计算最高位 1 的位置提供了便利. 记最高位 1 的位置为  $sum$ , 考虑其二进制表示, 每当  $x$  右移  $2^i$  位 ( $i = 4, 3, 2, 1, 0$ ), 将  $sum$  加上  $2^i$ . 若最后  $x$  为 0, 说明找不到为 1 的位, 函数返回  $sum + 1$ ; 若最后  $x$  为 1, 返回  $sum + 2$ .

完整代码如下:

```
/* howManyBits - return the minimum number of bits required to represent x in
 *                two's complement
 * Examples: howManyBits(12) = 5
 *           howManyBits(298) = 10
 *           howManyBits(-5) = 4
 *           howManyBits(0) = 1
 *           howManyBits(-1) = 1
 *           howManyBits(0x80000000) = 32
 * Legal ops: ! ~ & ^ | + << >>
 * Max ops: 90
 * Rating: 4
 */
int howManyBits(int x) {
    int sum, tmp;
    x = x >> 31 ^ x;
    sum = !!(x >> 16) << 4, x = x >> sum;
    tmp = !!(x >> 8) << 3, sum = sum + tmp, x = x >> tmp;
    tmp = !!(x >> 4) << 2, sum = sum + tmp, x = x >> tmp;
    tmp = !!(x >> 2) << 1, sum = sum + tmp, x = x >> tmp;
    tmp = x >> 1, sum = sum + tmp, x = x >> tmp;
    return sum + x + 1;
}
```

```
}
```

根据评测结果, 该函数功能正确, 共计使用 30 个运算符.

## 11. floatScale2

本小题以无符号整数 `uf` 的形式给出一个 IEEE 标准的 32 位浮点数 `f`, 要求返回  $2*f$ . 为使程序逻辑清晰, 首先利用位掩码将该浮点数拆分为符号位 `s`, 阶数部分 `e`, 以及尾数部分 `f`, 进而分类讨论:

1. 若阶数为 0, 为非规格数, 乘 2 相当于将尾数部分左移 1 位. 经检验, 该操作能够兼顾乘 2 后非规格数变为规格数 (阶数加 1) 的情况. 返回 `s|m<<1`.
2. 若阶数为 254, 乘 2 后为无穷, 应将尾数置零并继续下一步.
3. 若阶数为 255, 为 NaN 或无穷, 返回原值; 否则将阶数加 1 后返回 `s|e|m`.

完整代码如下:

```
/*
 * floatScale2 - Return bit-level equivalent of expression 2*f for
 * floating point argument f.
 * Both the argument and result are passed as unsigned int's, but
 * they are to be interpreted as the bit-level representation of
 * single-precision floating point values.
 * When argument is NaN, return argument
 * Legal ops: Any integer/unsigned operations incl. ||, &, also if, while
 * Max ops: 30
 * Rating: 4
 */
unsigned floatScale2(unsigned uf) {
    int s = uf & 0x80000000;
    int e = uf & 0x7F800000;
    int m = uf & 0x007FFFFF;
    if (!e)
        return s | m << 1;
    if (e == 0x7F000000)
        m = 0;
    if (e != 0x7F800000)
        e += 0x00800000;
    return s | e | m;
}
```

根据评测结果, 该函数功能正确, 共计使用 11 个运算符.

## 12. floatFloat2Int

本小题要求实现浮点数向整数的类型转换。同样，为使程序逻辑清晰，首先利用位掩码将该浮点数拆分为符号位  $s$ ，阶数部分  $e$ ，以及尾数部分  $f$ ，进而分类讨论：

1. 若阶数小于 127，偏移后指数小于 0，该浮点数仅有小数部分，返回 0。
2. 若阶数大于 157，该浮点数超过 32 位整数范围，溢出。按注释要求，返回 0x80000000。
3. 其他情况，将尾数并上最高有效位 1，左移与符号位对齐，最后将整体根据指数右移，得到所求整数。

完整代码如下：

```
/*
 * floatFloat2Int - Return bit-level equivalent of expression (int) f
 * for floating point argument f.
 * Argument is passed as unsigned int, but
 * it is to be interpreted as the bit-level representation of a
 * single-precision floating point value.
 * Anything out of range (including NaN and infinity) should return
 * 0x80000000u.
 * Legal ops: Any integer/unsigned operations incl. ||, &&, also if, while
 * Max ops: 30
 * Rating: 4
 */
int floatFloat2Int(unsigned uf) {
    int s = uf & 0x80000000;
    int e = (uf >> 23) & 0xFF;
    int m = uf & 0x007FFFFF;
    if (e < 127)
        return 0;
    if (e > 157)
        return 0x80000000;
    return (s | 0x40000000 | m << 7) >> 157 - e;
}
```

根据评测结果，该函数功能正确，共计使用 11 个运算符。

## 13. floatPower2

本小题要求用浮点数返回 2 的  $x$  次幂。分三种情况讨论：

1. 若  $x > 127$ ，上溢，返回正无穷 (0x7F800000)。
2. 若  $x \leq 127 \&\& x \geq -126$ ，为规格数，偏移得阶数为  $x + 127$ ，返回  $x + 127 < 23$ 。

3. 若  $x < -126$  且  $x \geq -149$ , 为非规格数, 仅尾数部分的第  $149+x$  位为 1, 返回  $1 \ll 149+x$ .
4. 若  $x < -149$ , 下溢, 返回 0.

完整代码如下:

```
/*
 * floatPower2 - Return bit-level equivalent of the expression 2.0^x
 * (2.0 raised to the power x) for any 32-bit integer x.
 *
 * The unsigned value that is returned should have the identical bit
 * representation as the single-precision floating-point number 2.0^x.
 * If the result is too small to be represented as a denorm, return
 * 0. If too large, return +INF.
 *
 * Legal ops: Any integer/unsigned operations incl. ||, &&. Also if, while
 * Max ops: 30
 * Rating: 4
 */
unsigned floatPower2(int x) {
    if (x > 127)
        return 0x7F800000;
    if (x >= -126)
        return x + 127 << 23;
    if (x >= -149)
        return 1 << 149 + x;
    return 0;
}
```

根据评测结果, 该函数功能正确, 共计使用 9 个运算符.

## 四、 总结

完成 Data Lab, 主要有以下收获:

- 加深了对计算机中整数和浮点数格式的理解, 尤其是浮点数运算的实现思路.
- 掌握了 C 语言位运算符的优先级关系和特性, 尤其是右移运算符补全高位的方式.
- 积累了位运算符的使用技巧, 如: 通过两次逻辑非将任意整数转换为布尔类型, 通过位掩码得到指定位的值等.
- 理解了镜像、容器等概念, 学会了 Docker 的使用, 能够编写简易的 Dockerfile.
- 熟悉了 Linux 中常见软件包, 能够在终端中使用 Linux 常见命令.

本实验的所有材料及代码已上传至 GitHub:

<https://github.com/HasiNed/Computer-System>

## 附录 A 环境搭建

本实验涉及的部分二进制文件依赖于 Linux/amd64 环境以及相关软件包。由于笔者的机器基于 ARM 指令集, 无法直接满足环境需求, 因而使用 Docker 仿真 x86-64 平台, 并搭建 Ubuntu 虚拟环境。具体步骤如下:

1. 安装 VS Code、Docker Desktop 等应用, 此处从略。
2. 编写 Dockerfile (完整文件见[代码清单](#)), 大意如下。

- 从 Docker Hub 引入 Ubuntu 官方镜像:

```
FROM --platform=linux/x86_64 ubuntu:latest
```

- 安装 build-essential, perl 等软件包:

```
RUN apt-get update \
    && apt-get -y install sudo git zsh vim perl curl locales build-essential \
    && apt-get clean -y
```

此处, 出于功能性考虑, 额外安装了 sudo, zsh, vim 等包。

- 创建 non-root 用户, 用于存放学号:

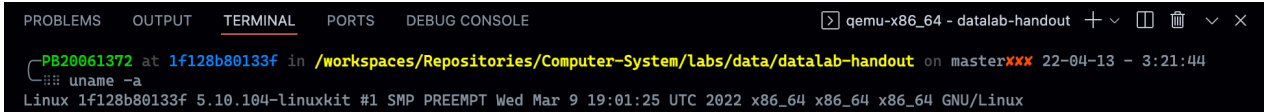
```
ARG USERNAME="PB20061372"
RUN useradd $USERNAME -m \
    && echo "$USERNAME ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/$USERNAME \
    && chmod 0440 /etc/sudoers.d/$USERNAME
USER $USERNAME
```

此处, 应当赋予新用户 sudo 命令权限。

- 安装 oh-my-zsh 并个性化主题, 从而显示必要信息:

```
RUN sh -c "$(curl -fsSL
↪ https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
↪ ""
RUN sed -i "s/robbyrussell/finn-time/g" ~/.zshrc
```

3. 在 VS Code 中, 安装 Remote - Containers 插件。在工作区中配置 Dev Container, 用于自动生成容器以及切换工作环境。
4. 在远程容器中打开工作路径并执行 `uname -a` 命令, 应有如下格式输出:



```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
qemu-x86_64 - datalab-handout + - [] -v x
PB20061372 at 1f128b80133f in /workspaces/Repositories/Computer-System/labs/data/datalab-handout on masterxxx 22-04-13 - 3:21:44
$ uname -a
Linux 1f128b80133f 5.10.104-linuxkit #1 SMP PREEMPT Wed Mar 9 19:01:25 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
```

图 2: 在远程容器中查看机器环境

5. 将所给的Makefile文件中 `CFLAGS = -O -Wall -m32` 修改为 `CFLAGS = -O -w`, 从而编译 64 位程序, 避免安装交叉编译所需的额外软件包, 同时消除因编程风格导致的部分警告.

至此, 实验环境搭建完毕. `bits.c`, `ishow.c`, `fshow.c`, `btest.c`, `dlc`, `driver.pl` 等均可在 Ubuntu 容器中编译或运行.

## 附录 B 代码清单

1. bits.c<sup>1</sup>


---

```

1  /*
2   * CS:APP Data Lab
3   * Yungqin Zhu 2022/03/29
4   */
5  int bitXor(int x, int y) { return ~(x & y) & ~(~x & ~y); }
6
7  int tmin(void) { return 1 << 31; }
8
9  int isTmax(int x) {
10     x = x + 1;
11     return !(~x | (x + x));
12 }
13
14 int allOddBits(int x) {
15     x = x & x >> 16;
16     return !((~(x & x >> 8) & 170));
17 }
18
19 int negate(int x) { return ~x + 1; }
20
21 int isAsciiDigit(int x) { return (x + ~57 & ~(x + ~47)) >> 31 & 1; }
22
23 int conditional(int x, int y, int z) {
24     x = !x + ~0;
25     return (x & y) | (~x & z);
26 }
27
28 int isLessOrEqual(int x, int y) { return (x + ~((x ^ y) >> 31 | y)) >> 31 & 1; }
29
30 int logicalNeg(int x) { return ((x | (~x + 1)) >> 31) + 1; }
31
32 int howManyBits(int x) {
33     int sum, tmp;
34     x = x >> 31 ^ x;
35     sum = !(x >> 16) << 4, x = x >> sum;
36     tmp = !(x >> 8) << 3, sum = sum + tmp, x = x >> tmp;
37     tmp = !(x >> 4) << 2, sum = sum + tmp, x = x >> tmp;
38     tmp = !(x >> 2) << 1, sum = sum + tmp, x = x >> tmp;
39     tmp = x >> 1, sum = sum + tmp, x = x >> tmp;
40     return sum + x + 1;
41 }
42
43 unsigned floatScale2(unsigned uf) {

```

---

<sup>1</sup>出于篇幅考虑，删除了大段注释。



```
44     int s = uf & 0x80000000;
45     int e = uf & 0x7F800000;
46     int m = uf & 0x007FFFFF;
47     if (!e)
48         return s | m << 1;
49     if (e == 0x7F000000)
50         m = 0;
51     if (e != 0x7F800000)
52         e += 0x00800000;
53     return s | e | m;
54 }
55
56 int floatFloat2Int(unsigned uf) {
57     int s = uf & 0x80000000;
58     int e = (uf >> 23) & 0xFF;
59     int m = uf & 0x007FFFFF;
60     if (e < 127)
61         return 0;
62     if (e > 157)
63         return 0x80000000;
64     return (s | 0x40000000 | m << 7) >> 157 - e;
65 }
66
67 unsigned floatPower2(int x) {
68     if (x > 127)
69         return 0x7F800000;
70     if (x >= -126)
71         return x + 127 << 23;
72     if (x >= -149)
73         return 1 << 149 + x;
74     return 0;
75 }
```

---

## 2. Dockerfile

---

```
1  # Emulate x86 architecture
2  FROM --platform=linux/x86_64 ubuntu:latest
3
4  # Switch apt source to mirror
5  RUN sed -i "s/archive.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list
6  RUN sed -i "s/security.ubuntu.com/mirrors.ustc.edu.cn/g" /etc/apt/sources.list
7
8  # Install packages
9  RUN apt-get update \
10     && DEBIAN_FRONTEND=noninteractive apt-get install -y build-essential sudo git locales zsh
11     && vim perl curl \
12     && apt-get clean -y
13
14 # Generate locale
15
16 # Create a user to show my student ID
17 ARG USERNAME="PB20061372"
18 RUN useradd $USERNAME -m \
19     && echo "$USERNAME ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/$USERNAME \
20     && chmod 0440 /etc/sudoers.d/$USERNAME
21 USER $USERNAME
22
23 # Install oh-my-zsh and set theme to fino-time (my favorite)
24 RUN sh -c "$(curl -fsSL
25     ↪ https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)" ""
26 RUN sed -i "s/robbyrussell/fino-time/g" ~/.zshrc
```

---