

哈爾濱工業大學

实验报告

实 验（二）

题 目 DataLab 数据表示

专 业 计算机

学 号 1170300825

班 级 1703008

学 生 李大鑫

指 导 教 师 郑贵滨

实 验 地 点 _____

实 验 日 期 _____

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 4 -
1.1 实验目的	- 4 -
1.2 实验环境与工具	- 4 -
1.2.1 硬件环境	- 4 -
1.2.2 软件环境	- 4 -
1.2.3 开发工具	- 4 -
1.3 实验预习	- 4 -
第 2 章 实验环境建立	- 6 -
2.1 UBUNTU 下 CODEBLOCKS 安装 (5 分)	- 6 -
2.2 64 位 UBUNTU 下 32 位运行环境建立 (5 分)	- 6 -
第 3 章 C 语言的位操作指令	- 8 -
3.1 逻辑操作 (1 分)	- 8 -
3.2 无符号数位操作 (2 分)	- 8 -
3.3 有符号数位操作 (2 分)	- 8 -
第 4 章 汇编语言的位操作指令	- 9 -
4.1 逻辑运算(1 分)	- 9 -
4.2 无符号数左右移 (2 分)	- 9 -
4.3 有符号左右移 (2 分)	- 9 -
4.4 循环移位 (2 分)	- 9 -
4.5 带进位位的循环移位 (2 分)	- 9 -
4.6 测试、位测试 BTX (2 分)	- 9 -
4.7 条件传送 CMOVXX (2 分)	- 10 -
4.8 条件设置 SETCXX (1 分)	- 10 -
4.9 进位位操作 (1 分)	- 10 -
第 5 章 BITS 函数实验与分析	- 10 -
5.1 函数 LSBZERO 的实现及说明	- 11 -
5.2 函数 BYTENOT 的实现及说明函数	- 12 -
5.3 函数 BYTEXOR 的实现及说明函数	- 12 -
5.4 函数 LOGICALAND 的实现及说明函数	- 12 -
5.5 函数 LOGICALOR 的实现及说明函数	- 13 -
5.6 函数 ROTATELEFT 的实现及说明函数	- 13 -
5.7 函数 PARITYCHECK 的实现及说明函数	- 14 -
5.8 函数 MUL2OK 的实现及说明函数	- 14 -
5.9 函数 MULT3DIV2 的实现及说明函数	- 15 -
5.10 函数 SUBOK 的实现及说明函数	- 16 -
5.11 函数 ABSVAL 的实现及说明函数	- 16 -

5.12 函数 FLOAT_ABS 的实现及说明函数.....	- 17 -
5.13 函数 FLOAT_F2I 的实现及说明函数	- 17 -
5.14 函数 XXXX 的实现及说明函数（CMU 多出来的函数-不加分）	- 18 -
第 6 章 总结	- 19 -
10.1 请总结本次实验的收获.....	- 19 -
10.2 请给出对本次实验内容的建议.....	- 19 -
参考文献.....	- 20 -

第 1 章 实验基本信息

1.1 实验目的

- 熟练掌握计算机系统的数据表示与数据运算
- 通过 C 程序深入理解计算机运算器的底层实现与优化
- 掌握 Linux 下 makefile 与 GDB 的使用

1.2 实验环境与工具

1.2.1 硬件环境

- X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

- Windows7 64 位以上; VirtualBox/Vmware 11 以上;
Ubuntu 16.04 LTS 64 位/优麒麟 64 位;

1.2.3 开发工具

- Visual Studio 2010 64 位以上; CodeBlocks ;
vi/vim/gpedit+gcc

1.3 实验预习

- 上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。
- 写出 C 语言下的位操作指令:
 - 逻辑
 - 无符号

- 有符号
- 写出汇编语言下的位操作指令：
 - 逻辑运算
 - 无符号
 - 有符号
 - 测试、位测试 BT_x
 - 条件传送 CMOV_{xx}
 - 条件设置 SET_{xx}
 - 进位位(CF)操作

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 安装 (5 分)

CodeBlocks 运行界面截图：编译、运行 hellolinux.c

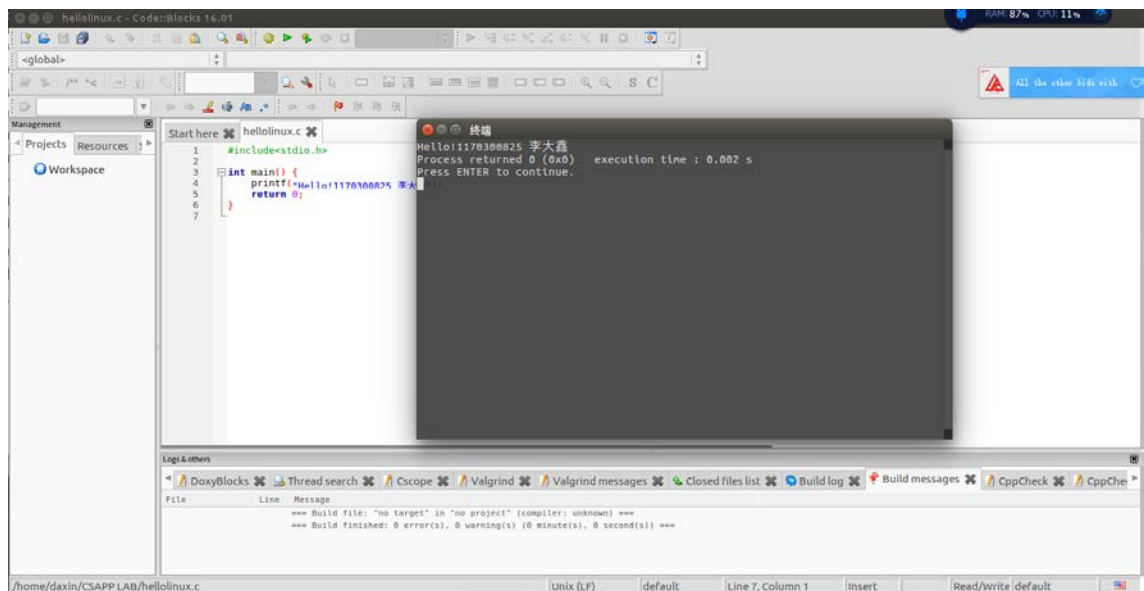


图 2-1 Ubuntu 下 CodeBlocks 截图

2.2 64 位 Ubuntu 下 32 位运行环境建立 (5 分)

在终端下，用 gcc 的 32 位模式编译生成 hellolinux.c。执行此文件。
Linux 及终端的截图。

```
root@vm: /home/daxin/CSAPP LAB
正在设置 lib32cilkrts5 (5.4.0-6ubuntu1~16.04.10) ...
正在设置 libx32cilkrts5 (5.4.0-6ubuntu1~16.04.10) ...
正在设置 lib32mpx0 (5.4.0-6ubuntu1~16.04.10) ...
正在设置 lib32quadmath0 (5.4.0-6ubuntu1~16.04.10) ...
正在设置 libx32quadmath0 (5.4.0-6ubuntu1~16.04.10) ...
正在设置 lib32gcc-5-dev (5.4.0-6ubuntu1~16.04.10) ...
正在设置 libx32gcc-5-dev (5.4.0-6ubuntu1~16.04.10) ...
正在设置 gcc-5-multilib (5.4.0-6ubuntu1~16.04.10) ...
正在设置 lib32stdc++-5-dev (5.4.0-6ubuntu1~16.04.10) ...
正在设置 libx32stdc++-5-dev (5.4.0-6ubuntu1~16.04.10) ...
正在设置 g++-5-multilib (5.4.0-6ubuntu1~16.04.10) ...
正在设置 gcc-multilib (4:5.3.1-1ubuntu1) ...
正在设置 g++-multilib (4:5.3.1-1ubuntu1) ...
正在设置 module-assistant (0.11.8) ...
正在处理用于 libc-bin (2.23-0ubuntu10) 的触发器 ...
root@vm:/home/daxin/CSAPP LAB# gcc -m32 hellolinux.c
root@vm:/home/daxin/CSAPP LAB# ls
a.out  hellolinux.c  hellolinux.o  lab1-handout
root@vm:/home/daxin/CSAPP LAB# gcc -m32 -o hellolinux hellolinux.c
root@vm:/home/daxin/CSAPP LAB# ls
a.out  hellolinux  hellolinux.c  hellolinux.o  lab1-handout
root@vm:/home/daxin/CSAPP LAB# ./hellolinux
Hello!1170300825 李大鑫root@vm:/home/daxin/CSAPP LAB#
root@vm:/home/daxin/CSAPP LAB#
```

图 2-2 32 位运行环境建立

第 3 章 C 语言的位操作指令

写出 C 语言例句

3.1 逻辑操作 (1 分)

A|B
A&B
~A

3.2 无符号数位操作 (2 分)

```
unsigned int x,y;  
y=x<<1;  
y=x>>1;    //逻辑右移，右移空出的位用 0 填充
```

3.3 有符号数位操作 (2 分)

```
int x,y;  
y=x<<1;  
y=x>>1;    //算数右移，右移空出的位用最高位符号位填充
```


第 4 章 汇编语言的位操作指令

写出汇编语言例句

4.1 逻辑运算 (1 分)

```
AND %eax,%ebx  
OR %eax,%ebx  
XOR %eax,%ebx  
NOT %eax
```

4.2 无符号数左右移 (2 分)

```
SHL %eax  
SHR %eax
```

4.3 有符号左右移 (2 分)

```
SAL %eax  
SAR %eax
```

4.4 循环移位 (2 分)

```
ROL %eax  
ROR %eax
```

4.5 带进位位的循环移位 (2 分)

```
RCL %eax  
RCR %eax
```

4.6 测试、位测试 BT_x (2 分)

```
TEST %eax,%ebx  
BTC r16/imm8,r/m16  
BTR r16/imm8,r/m16  
  
BTS r16/imm8,r/m16
```

4.7 条件传送 CMOVxx (2 分)

```
CMOVG %eax,%ebx
CMOVNLE %eax,%ebx
CMOVGE %eax,%ebx
CMOVNGE %eax,%ebx
CMOVL %eax,%ebx
CMOVLE %eax,%ebx
CMOVNG %eax,%ebx
CMOVNO %eax,%ebx
CMOVS %eax,%ebx
CMOVNS %eax,%ebx
```

4.8 条件设置 SETxx (1 分)

```
SETE %eax
SETNE %eax
SETS %eax
SETNS %eax
SETG %eax
SETGE %eax
SETL %eax
SETLE %eax
SETA %eax
SETAE %eax
SETB %eax
SETBE %eax
```

4.9 进位位操作 (1 分)

```
STC
CLC
```

第5章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分

语法检查命令 ./dlc -e bits.c 的结果截图：

```

new@new-virtual-machine:~/hitics/lab1-handout$ ./dlc bits.c
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.

Compilation Successful (1 warning)
new@new-virtual-machine:~/hitics/lab1-handout$ ./dlc -e bits.c
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.
dlc:bits.c:176:lsbZero: 2 operators
dlc:bits.c:187:byteNot: 3 operators
dlc:bits.c:200:byteXor: 9 operators
dlc:bits.c:209:logicalAnd: 4 operators
dlc:bits.c:218:logicalOr: 5 operators
dlc:bits.c:229:rotateLeft: 11 operators
dlc:bits.c:245:parityCheck: 15 operators
dlc:bits.c:257:mul20K: 5 operators
dlc:bits.c:274:mult3div2: 8 operators
dlc:bits.c:289:sub0K: 18 operators
dlc:bits.c:304:absVal: 5 operators
dlc:bits.c:325:float_abs: 8 operators
dlc:bits.c:365:float_f2i: 25 operators

Compilation Successful (1 warning)
new@new-virtual-machine:~/hitics/lab1-handout$

```

说明：这个命令之前截图截错了，后来重装了系统之后又做了一次。

5.1 函数 lsbZero 的实现及说明

程序如下：

```

int lsbZero(int x) {
    return x & (~0x1);
}

```

btest (命令 ./btest -f lsbZero) 的结果截图：

```

daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f lsbZero
Score  Rating  Errors  Function
1      1        0      lsbZero

```

设计思想：

将 x 的最低二进制位设置为 0，只需要将它和 $0xFFFFFFFFE$ 相与， $0xFFFFFFFFE$ 通过 $\sim 0x1$ 获得。

5.2 函数 byteNot 的实现及说明函数

程序如下：

```
int byteNot(int x, int n) {
    return x^(0xFF<<(n<<3));
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f byteNot
Score Rating Errors Function
2      2      0      byteNot
Total points: 2/2
```

设计思想：

当一位二进制数与 1 抑或的时候相当于将这一位二进制取反。所以只要将第 n 个 byte 与第 n 个 byte 全是 1、其他位全是 0 的数抑或就可以将 x 的第 n 个 byte 全部取反。通过 $0xFF \ll (n \ll 3)$ 得到一个第 n 个 byte 全是 1、其他位全是 0 的数。

5.3 函数 byteXor 的实现及说明函数

程序如下：

```
int byteXor(int x, int y, int n) {
    return !((x&(0xFF<<(n<<3)))^(y&(0xFF<<(n<<3)))) ;
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f byteXor
Score Rating Errors Function
2      2      0      byteXor
Total points: 2/2
```

设计思想：

首先取出第 n 个 byte 位，如 x ，将第 n 个 byte 设置为全 1 其他为 0 相与得到： $x \& (0xFF \ll (n \ll 3))$ ，两个数抑或得到 byteXor。

5.4 函数 logicalAnd 的实现及说明函数

程序如下：

```
int logicalAnd(int x, int y) {
    return !((!x)|(!y));
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f logicalAnd
Score Rating Errors Function
3      3      0      logicalAnd
Total points: 3/3
```

设计思想：

如果 ax ay 分别代表 x y 是 1 否，是则为 1 否则为 0。ans=ax&ay。我们知道！运算符可以将非 0 变为 0，将 0 变为 1，所以!x 可以得到 x 为 0 否，是则 1 否则 0，则 ax=!!x。所以 ans=(!!x)&(!!y)=!((!x)|(!y))。

5.5 函数 logicalOr 的实现及说明函数

程序如下：

```
int logicalOr(int x, int y) {
    return (!!x)|(!y);
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f logicalOr
Score Rating Errors Function
3      3      0      logicalOr
Total points: 3/3
```

设计思想：

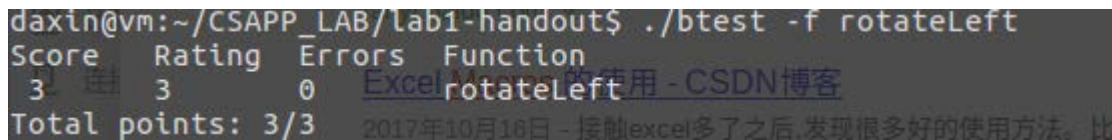
同上个题，使用!!x 得到 x 是否为 1

5.6 函数 rotateLeft 的实现及说明函数

程序如下：

```
int rotateLeft(int x, int n) {
    return (x<<n) | (x>>(32+(-n+1))) & ((1<<n)+(-1+1));
}
```

btest 截图:



```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f rotateLeft
Score  Rating  Errors  Function
3      3      0      rotateLeft
Total points: 3/3
```

设计思想:

不考虑 x 的算数右移, 则答案为 $(x << n) | (x >> (32-n))$, 当考虑算数右移的时候, 需要 $\&((1 << n) - 1)$ 保留低 n 位, 此时: $(x << n) | (x >> (32-n)) \&((1 << n) - 1)$, 因为不能使用减号, 所以使用+补码的方式代替。

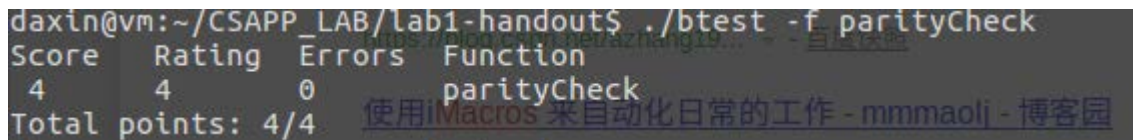
5.7 函数 parityCheck 的实现及说明函数

程序如下:



```
int parityCheck(int x) {
    int y = x;
    y = y ^ (y >> (1 << 4));
    y = y ^ (y >> (1 << 3));
    y = y ^ (y >> (1 << 2));
    y = y ^ (y >> (1 << 1));
    y = y ^ (y >> 1);
    return y & 1;
}
```

btest 截图:



```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f parityCheck
Score  Rating  Errors  Function
4      4      0      parityCheck
Total points: 4/4
```

设计思想:

总的思路是通过位之间的相抑或得到是否奇偶。因为有 32 位所以不能一位位抑或, 所以通过折半相抑或的方法一句句列出。

5.8 函数 mul20K 的实现及说明函数

程序如下:


```
int mul20K(int x) {
    return (~((x>>31))^(x>>30))&1;
}
```

btest 截图:

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f mul20K
Score Rating Errors Function
2      2      0      mul20K
Total points: 2/2
```

设计思想:

当 x 为正数时, 此时第 32 位符号位是 0, 乘 2 相当于左移 1 位, 当第 31 位是 1 的时候产生溢出; 当 x 为负数时, 此时符号位是 1, 从二进制来看, x 实际上表达的大小为 $-2^{32} + (\text{前 31 位单独表示的数的大小})$, 当 $x < -(2^{30})$ 的时候产生负数溢出, 此时 $(\text{前 31 位单独表示的数的大小}) < 2^{30}$, 此时第 31 位总是 0, 变的是前 30 位数。综上只要第 32 位和第 31 位不同就会产生溢出。分别左移 30 位和 31 位相抑或, \sim 之后最低位是结果, $\&1$ 取出最低位。

5.9 函数 mult3div2 的实现及说明函数

程序如下:

```
int mult3div2(int x) {
    int y = (x << 1) + x;
    int z = (y >> 31) & 1;
    int m = y & 1;
    return (y >> 1) + (z & m);
}
```

btest 截图:

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f mult3div2
Score Rating Errors Function
2      2      0      mult3div2
Total points: 2/2
```

设计思想:

$y=3*x$, z 为 y 的符号位, m 为 y 的最低位, 因为当 y 为负数的时候左移直接截取不会考虑进位的问题, 当 y 的符号位和最低位同时为 1 的时候, 此时右移会产生 0.5 的小数部分, 因此需要进行进位+1。

5.10 函数 subOK 的实现及说明函数

程序如下：

```
int subOK(int x, int y) {
    int z = x + (~y + 1);
    int fx = (x >> 31) & 1;
    int fy = (y >> 31) & 1;
    int fz = (z >> 31) & 1;
    return ((~fx & ~fy) | (fy & ~fz) | (fx & fz)) & 1;
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f subOK
Score  Rating  Errors  Function
3      3        0      subOK
Total points: 3/3
```

设计思想：

根据操作数与结果的符号位判断是否发生溢出。令 z 为差，首先罗列所有不会发生溢出的情况：1) x 和 y 为正，符号位均为 0。2) y 为负， z 为正， y 的符号位为 1， z 的符号位为 0。3) x 和 z 为负，两者符号位为 1。根据右移相与操作得到各自的符号位，总表达式如程序。（当 xy 的符号不同且 z 的符号不同于 x 的时候会产生溢出）

5.11 函数 absVal 的实现及说明函数

程序如下：

```
int absVal(int x) {
    int y = x;
    y = y >> 31;
    y = y ^ x;
    return y + ((x >> 31) & 1);
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f absVal
Score  Rating  Errors  Function
4      4        0      absVal
```

设计思想：

当 x 为正数时，不用变；当 x 是负数时，取反+1 之后为绝对值。根据两者符号位的不同进行设计，首先区别是否取反， $x^{(x>>1)}$ ，其中 $x>>1$ 是算数右移，如果是正数相当于全是 0 抑或之后为原值不变，如果是负数相当于全是 1 抑或之后相当于取反；其次区别是否+1，加一个符号位即可。

5.12 函数 float_abs 的实现及说明函数

程序如下：

```
unsigned float_abs(unsigned uf) {  
    //unsigned sign = uf >> 31;  
    unsigned exp = uf >> 23 & 0xFF;  
    unsigned frac = uf & 0x7FFFFFFF;  
    if(exp==0xFF && frac!=0){  
        return uf;  
    }else {  
        return frac|exp<<23;  
    }  
}
```

btest 截图：

```
daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f float_abs  
Score  Rating  Errors  Function  
2      2        0      float_abs  
Total points: 2/2
```

设计思想：

现将浮点数的指数部分 exp 和小数部分 frac 通过位移和与操作取得。当 exp 全是 1 (0xFF) 且 frac 不为 0 的时候此时 uf 是 NAN，返回原值，否则，省去符号位， $\text{frac}|\text{exp}<<23$ 之后返回绝对值。

5.13 函数 float_f2i 的实现及说明函数

程序如下：

```

int float_f2i(unsigned uf) {
    int sign;
    unsigned exp = (uf >> 23) & 0xFF;
    unsigned frac = uf & 0x7FFFFFFF;
    unsigned shlbit = exp - 127;
    if(uf >> 31) sign = -1;
    else sign = 1;
    if(exp == 0xFF) {
        return 0x80000000u;
    }
    if(exp == 0) {
        return 0;
    }
    if(exp < 127) {
        return 0;
    }
    if(shlbit <= 23) {
        unsigned ans = (1 << shlbit) | (frac >> (23 - shlbit));
        if((frac >> (22 - shlbit)) & 1) ans = ans + 1;
        return ans * (sign);
    } else {
        unsigned ans = frac << shlbit | (1 << (shlbit + 1));
        if (shlbit < 32)
            return ans * sign;
        else
            return 0x80000000u;
    }
}

```

btest 截图:

```

daxin@vm:~/CSAPP_LAB/lab1-handout$ ./btest -f float_f2i
Score  Rating  Errors  Function
  4      4      0      float_f2i
Total points: 4/4

```

设计思想:

处理特殊情况: 1) NAN & infinity. 2) 当指数部分 (减去 bias 之后) 小于 127 的时候此时对应 $1 > \text{ans} > 0$, 直接返回 0。

处理之后得到的指数部分非负, 1) 如果指数部分 (shlbit) 小于等于 23, 对 1.frac 左移 shlbit 位, 不能全部左移到小数点的右边, 此时需要考虑进位的问题。2) 指数部分大于 23, 如果指数大于等于 32, 此时 int 无法表达, 设置为指定值, 否则直接将 1.frac 左移指数位。

5.14 函数 XXXX 的实现及说明函数 (CMU 多出来的函数-不加分)

第 6 章 总结

10.1 请总结本次实验的收获

如何用位运算模拟计算机汇编层面实现的功能。

数据的表示。

使用位运算简化操作，节省减少计算量。

10.2 请给出对本次实验内容的建议

可以增加对书籍提供代码的讲解，这样能更深的理解程序的编译运行问题而且还能更加熟悉 linux 下程序的编译运行。

参考文献

为完成本次实验你翻阅的书籍与网站等

- [1] 林来兴. 空间控制技术[M]. 北京: 中国宇航出版社, 1992: 25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集: A 集[C]. 北京: 中国科学出版社, 1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北: 天下文化出版社, 1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm> (Big5) .
- [4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨: 哈尔滨工业大学, 1992: 8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359): 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.