中国大学生计算机系统与程序设计竞赛

CCF-CCSP-2016

时间：2016 年 11 月 26 日 09:00 ~ 22:00

题目名称	选座	虚拟机设计（第一部分）	虚拟机设计（第二部分）
题目类型	传统型	传统型	提交答案型
输入	标准输入	标准输入	`*.in`
输出	标准输出	标准输出	`*.out`
每个测试点时限	1.0 秒	1.0 秒	N/A
内存限制	2 GB	2 GB	N/A
测试点数目	10	15	3
每个测试点分值	10	非等分测试点	非等分测试点

选座（ticket_chooser）

【题目描述】

小 B 是一个电影迷，只要有时间，她就要去观摩最新的大片。但她不喜欢自己在电脑或其他电子设备上观看，而是喜欢去电影院，因为她觉得那里更有气氛。

由于工作关系，小 B 这次被派到 A 地一段时间，她发现这里的电影院和她熟悉的模式完全不同。

电影院内部都是正方形的，一共有 k 排，从前到后按照 1 到 k 编号，每排内有 k 个座位，从左到右按照 1 到 k 编号，其中 k 为奇数。

考虑到安全因素，座位不允许购票者自行挑选，而是由售票人员通过电脑程序确定。由于大家都希望有更好的观影效果，因此一般都倾向于选择更靠近影院中心的座位。

电脑程序选择座位的过程为：

如果有人需要购买 m 张电影票，程序首先会确定一个排号 x，并从中选择一段连续且尚未售出的座位号 $[l,r]$ ，其中 $r-l+1=m$ 。

如果没有任何一排中有 m 个连续的空座位，则电脑程序会报错，在这个购票请求中将不会卖出任何票。

在保证选出的座位在同一排且座位号连续的前提下，程序会选择最 ‘‘接近中心’’ 的座位。

具体来说，令 $x_c=y_c=\frac{k+1}{2}$ ，表示影院中最中心的座位。定义选出的这些座位到影院中心的距离函数为：

$$\sum_{i=l}^{r}|x-x_c|+|i-y_c|$$

最 ‘‘接近中心’’ 的座位为能最小化上述函数的座位。若有多个可选座位均满足离影院中心距离最小的条件，则选座程序优先选择靠前的座位（即排号 x 最小的座位）。若仍有多个座位符合要求，则选座程序优先选择靠左的座位（即座位号 l 值最小的座位）。

假设电影院最开始没有售出任何座位，小 B 希望知道对于给出的 n 个购票请求，每次售出的票都能买到哪些座位？

【输入描述】

输入包含多组数据，你需要用判断是否读到文件末尾的形式判断输入是否结束。

每组数据的第一行包含两个正整数 n 和 k，表示购票请求的数量和影院大小。保证 $1\le n\le 3\times 10^5, 1\le k\le 300,001$，且 k 为奇数。

第二行为空格分隔的 n 个正整数，其中第 i（$1\le i\le n$）个数为 m_i，表示每次要求购买的票数，保证 $1\le m_i\le k$。

【输出描述】

每组数据输出包含 n 行，每个购买请求的结果为一行。

如果无法在一排中买到 m_i 个连续的座位，则在对应的行中输出 -1。否则输出三个空格分隔的整数 x, l, r，为所买电影票的排号和起止座位号。

【样例 1 输入】

```
2 1
1 1
4 3
1 2 3 1
```

【样例 1 输出】

```
1 1 1
-1
2 2 2
1 1 2
3 1 3
2 1 1
```

【样例 2】

见题目目录下的 ***2.in*** 与 ***2.ans***。

【子任务】

对于 20% 的测试数据，$n \le 50, k \le 25$;

对于 40% 的测试数据，$n \le 100, k \le 101$;

对于 50% 的测试数据，$n \le 1000, k \le 501$;

对于 60% 的测试数据，$n \le 1000, k \le 1001$;

对于 70% 的测试数据，$n \le 50000, k \le 50001$;

对于 80% 的测试数据，$n \le 100000, k \le 100001$;

对于 90% 的测试数据，$n \le 200000, k \le 200001$;

对于 100% 的测试数据，$n \le 300000, k \le 300001$，保证每个测试点的数据组数不超过 5 组。

虚拟机设计（第一部分）（trick）

此部分分值：105.0 分

【问题描述】

菜菜在上过汇编语言这门课后，自己设计了一套“精简指令集”。为了测试以及进一步地调整，菜菜需要一个虚拟机来将这些指令执行起来。你能帮帮他么？

菜菜需要一个 16 位的虚拟机，内存地址从 `0000` 到 `FFFF`，总共有 2^{16} 字节。规定其中 $[3000, B000)$ 是数据段（区间左闭右开），即指令可以进行读写操作的内存段只有 2^{15} 字节。CPU 内有四个 16 位寄存器可供使用：`AX`、`BX`、`CX`、`DX`。CPU 处理的所有数据皆为 16 位（二进制位）的无符号整数。

指令中的操作数总共有三种形式：立即数、寄存器和内存。

- 立即数：一个四位的 16 进制常数，不会省略前导零，字母使用大写，如 `02C0`；
- 寄存器：`AX`、`BX`、`CX` 或 `DX`，字母皆为大写。
- 内存：采用“立即数直接寻址”和“寄存器间接寻址”两种方式，给出内存地址，根据该地址去内存中存取数据。具体的形式为 `T+ 立即数` 和 `T+ 寄存器`，如 `T02C0` 和 `TAX`，其中立即数或相应寄存器中的值表示内存地址。

每条指令对操作数中的值进行相应地处理。对于三种不同类型的操作数，“操作数中的值”这一概念的具体含义略有不同：立即数本身即是值；寄存器中存放的整数是值；内存则是根据内存地址取出的数据是它的值。

数据在内存中以**小端模式**存储，即数据的高字节保存在内存的高地址中，而数据的低字节保存在内存的低地址中。菜菜的虚拟机中所有的数据均为 16 位（二进制位）无符号整数，在内存中将占据相邻的两个字节，其中高 8 位将存于高地址处，低 8 位则存于低地址处。

下图显示了一段内存的存储情况（`4000` 到 `4004`）。

‘4000‘	‘4001‘	‘4002‘	‘4003‘	‘4004‘
‘00‘	‘00‘	‘0C‘	‘FF‘	‘00‘

对于上图中的情况，如果想要根据地址 `4002` 从内存中读取数据，则将会读取 `4002` 和 `4003` 中的数据。其中高地址 `4003` 的 `FF` 作为高位，低地址 `4002` 的 `0C` 作为低位，读取的数据则为 `FF0C`。

如果想要向地址 `4001` 处写入数据 `54AC`，则将会把数据写入 `4001` 和 `4002` 中。其中高位的 `54` 写入高地址 `4002` 处，低位的 `AC` 则写入低地址 `4001` 处。执行写入操作后，内存情况变为下图：

‘4000‘	‘4001‘	‘4002‘	‘4003‘	‘4004‘
‘00‘	‘AC‘	‘54‘	‘FF‘	‘00‘

菜菜目前需要你来实现下面几种基本的指令：

1. `RUN`：标识着程序的开始。如无特殊说明，内存和寄存器均已初始化为 0。
2. `STOP`：标识着程序的正常结束。
3. `ECHO A`：将操作数 A 中的值输出。
4. `ADD A B`：将操作数 A 中的值与 B 中的值相加，结果存回 A。相加产生溢出时，直接将溢出部分丢弃即可（截断）—---无需向更高位进位，存回 A 的同样是一个 16 位（二进制位）无符号整数。A 不能为立即数。
5. `INC A`：将操作数 A 中的值加 1，结果存回 A。同样忽略溢出，A 不能是立即数。
6. `MOV A B`：将操作数 B 中的值写入 A，A 不能是立即数。
7. `CMP A B`：比较操作数 A 和 B 中的值的大小，结果将作为条件跳转指令的依据。
8. 跳转指令。

这里详细说明一下跳转指令：在没有跳转指令的情况下，虚拟机会按照程序编写的顺序执行每一条指令，而跳转指令则能够指定虚拟机接下来执行哪一条指令。具体地来说，假设总共有 n 条指令，从 1 到 n 对其进行标号。当第 i 条指令执行完后，如果不是跳转指令，虚拟机将自动执行下一条即第 $i+1$ 条指令；但如果第 i 条指令是跳转指令，接下来将可能执行它指定的某一条指令。

跳转指令分为两种：条件跳转和无条件跳转。顾名思义，无条件跳转指令执行完后，一定会执行它指定的某一条指令；而条件跳转指令执行完后，只有在满足某些条件时，才会执行它指定的指令，如果不满足则仍然按照默认顺序执行下一条指令。

无条件跳转指令只有一种：`JMP X`

不妨假设操作数 X 中的值是 i，则该指令执行完后，将去执行第 i 条指令。

条件跳转指令则是以**上一次 `CMP A B` 指令执行时**的比较结果作为条件，根据操作数 A 和 B 中的值的大小关系，共有以下 6 种形式。这里同样不妨假设 A 中的值是 a，B 中的值是 b。

1. `JG X`：a 大于 b 时跳转
2. `JL X`：a 小于 b 时跳转
3. `JE X`：a 等于 b 时跳转
4. `JNG X`：a 不大于 b 时跳转
5. `JNL X`：a 不小于 b 时跳转
6. `JNE X`：a 不等于 b 时跳转

不妨假设操作数 X 中的值是 i，则该指令执行完后，如果满足相应的条件，将去执行第 i 条指令，否则按默认顺序继续执行下一条指令。需要注意的是，`CMP` 指令与条件跳转指令不是一一对应的关系。一条 `CMP` 指令的结果将作为所有条件跳转指令的依据，直到执行下一条 `CMP` 指令为止。

菜菜的虚拟机对程序的格式也有着严格的要求：

1. 每行一条指令，指令内部不同部分之间仅用一个空格进行分隔，不允许有多余空格。

2. 第一行指令是 RUN，最后一行指令是 STOP，程序中不允许有其它的 RUN 和 STOP。
3. 指令中所有的字母均为大写。
4. 出于对时钟周期的考量，一条指令中不能同时存在两个内存操作数。

菜菜已经用上述 8 种指令写好了一段程序。虽然每一条指令都严格符合上述约定，但由于数据的不可预知性，程序在运行过程中仍然可能会遇到以下几种问题：

1. 存取非法。程序只能对寄存器和数据段 $[3000, B000)$ 进行操作，如果当前指令试图读写数据段以外的内存，则虚拟机应当立即报错，不会执行当前以及后面的指令。
2. 耗时过长。程序本身算法复杂度较高，或者陷入死循环死递归，将无法在短时间内得出结果。所以规定，一段程序最多执行一百万条次的指令。因为**跳转指令**的存在，每条指令可能执行多次，这里按照执行指令的次数总和计算。需要注意的是，不满足条件的**条件跳转指令**同样算做一条次指令计入总和。如果第一百万条次的指令顺利执行完毕，并且本身不是 STOP 指令，则虚拟机应当强制终止该程序，不再执行后面的指令。
3. 跳转错误。假设总共有 n 条指令，因为第一条指令一定是 RUN，所以规定**跳转指令**只能跳转到第 2 到第 n 条指令。当**无条件跳转指令**或满足条件的**条件跳转指令**试图跳转到第 i 条指令时，如果 $i < 2$ 或 $i > n$ 虚拟机应当立即报错并终止该程序。
4. CMP 缺失。**条件跳转指令**是以 CMP 指令的结果为依据的，所以在执行**条件跳转指令**时，如果之前从未执行过 CMP 指令，虚拟机应当立即报错并终止该程序。

考虑到内在的逻辑关系，除了耗时过长是在指令执行结束后判断，其余三种错误的优先级从高到低依次为 CMP 缺失 > 存取非法 > 跳转错误。如果某条指令执行时同时涉及多个错误，则把其中优先级最高的视为程序异常退出的原因。

希望你能按照上述要求将虚拟机实现，来执行菜菜的这段程序。

【输入格式】

给出用上述 8 种指令编写的一段程序，每行一条指令，保证没有格式错误。

【输出格式】

每个顺利执行的 ECHO 指令输出一行，一个四位十六进制整数（字母大写、不足四位用前导零补齐），表示该操作数中的值。

如果程序因**存取非法**而异常退出，则再输出一行 ACCESS_VIOLATION。

如果程序因**耗时过长**而异常退出，则再输出一行 TLE。

如果程序因**跳转错误**而异常退出，则再输出一行 RUNTIME_ERROR。

如果程序因 **CMP 缺失**而异常退出，则再输出一行 CMP_MISSING。

【样例 1 输入】

```
RUN
MOV T4001 54AC
MOV T4003 00FF
ECHO T4002
MOV BX T4002
ECHO BX
STOP
```

【样例 1 输出】

```
FF54
FF54
```

【样例 2 输入】

```
RUN
MOV T4001 54AC
MOV T4003 00FF
MOV BX T4002
ADD BX T4001
ECHO BX
ADD BX T4000
ECHO TBX
ECHO BX
STOP
```

【样例 2 输出】

```
5400
ACCESS_VIOLATION
```

【样例 3 输入】

```
RUN
JMP 0002
ECHO T0000
STOP
```

【样例 3 输出】

```
TLE
```

【样例 4 输入】

```
RUN
MOV BX 0003
CMP AX BX
JNL 0010
ECHO AX
INC AX
JMP AX
STOP
```

【样例 4 输出】

```
0000
RUNTIME_ERROR
```

【子任务】

每个测试用例中的程序均小于等于 100 行。
对于前三分之一的测试用例，程序中不涉及内存操作，即操作数中不会出现 T；
对于前三分之二的测试用例，程序中没有跳转指令和 CMP 指令。

虚拟机设计（第二部分）（trick_a）

这是一道提交答案题。

此部分分值：45.0 分

【题目描述】

最终，菜菜自己也将虚拟机编写了出来。菜菜的虚拟机不仅实现了前文所述的所有要求，还增加了检查程序格式错误的功能。只有格式无误的程序才会被执行。出于测试的目的，菜菜希望你能用这套指令编写程序，在虚拟机上完成下述几个任务。每个任务的程序长度均不得多于 40 行。

【样例任务】

已知 *A* 和 *B*，输出 *A* 和 *B* 中的较小值。

A 和 *B* 已经存放于内存当中，*A* 存放在 3000 处，*B* 存放在 3002 处。

任务解释

包括后面的任务，在没有特殊说明的情况下，所有的已知数据均默认为四位十六进制无符号整数。这里称 “*A* 存放在 3000 处”，意味着 3000 处存放着 *A* 的低 8 位（二进制位），3001 处存放着 *A* 的高 8 位（二进制位）。其余没有存放已知数据的内存和寄存器，均已被初始化为 0。

在求得答案后，只需要使用 ECHO 指令将其输出即可。

如果提交的程序长度小于等于 40 行、没有格式错误、能通过 STOP 指令正常结束，并且输出了正确的答案，你将会得到该任务相应的分数。

内存情况示意图：

3000	3001	3002	3003	3004
0B	02	15	00	00

上图即为 A 等于 43（十进制），B 等于 21（十进制）时内存的初始情况。

样例程序

```
RUN
MOV AX T3000
MOV BX T3002
CMP AX BX
JNL 0008
ECHO AX
```

```
JMP 0009
ECHO BX
STOP
```

【任务 1（10 分）】

已知 N（$N > 0$）个数 $A_1, A_2, \ldots, A_N$，试输出它们的最大值。

$A_1, A_2, \ldots, A_N$ 均已存放在内存的数据段中，依次位于 $3000, 3002, 3004, \ldots$

寄存器 AX 存储了 A_N 所在地址。

【任务 2（15 分）】

已知 N（$N > 0$）个数 $A_1, A_2, \ldots, A_N$，试输出其中不同的值的个数。

$A_1, A_2, \ldots, A_N$ 均已存放在内存的数据段中，依次位于 $3000, 3002, 3004, \ldots$

寄存器 AX 存储了 A_N 所在地址。

保证 N 和 $A_1, A_2, \ldots, A_N$ 均小于等于 5×10^3。

【任务 3（20 分）】

已知 N（$N > 0$）和 M，试输出组合数 C_N^M 对 65536 取模的结果。

寄存器 AX 存储了 N 的值，寄存器 BX 存储了 M 的值。

$$C_N^M = \frac{N!}{M!\,(N-M)!}$$

这里 $N!$ 表示 N 的阶乘，$0! = 1$。

保证 $M \le N \le 2 \times 10^2$。