MIPS 模拟机

一、 问题描述

以程序模拟 MIPS 运行，功能包括：
汇编器：将汇编程序转换成机器码。能有较灵活的格式，可以处理格式指令、表达式、有出错信息。
汇编反汇编：MIPS 汇编指令与机器码的相互转换。
模拟器：根据机器码模拟执行可以运行简单 MIPS 程序。

1、模拟器运行界面设计：可以命令行或窗口界面。可以执行指令的汇编、反汇编，可以单步执行指令观察寄存器、内存的变化。（命令行版可参考 DEBUG）
2、指令伪指令的汇编反汇编：将 MIPS 指令转换成二进制机器码，能够处理标号、变量。
3、MMU 存储器管理单元：存储器存取模拟。大头小头，对齐不对齐，Cache，虚拟存储。
4、格式指令表达式处理：对于汇编程序中的格式指令、表达式的处理。参考网页格式指令。

个人版汇编器应实现：
- >指令：R、LW、SW、BEQ、J 五条；
- >命令：
- →R-看寄存器
- →D-数据方式看内存
- →U-指令方式看内存
- →A-写汇编指令到内存
- →T-单步执行内存中的指令

二、 实现原理

2.1 总体设计

在实际的计算机设计中，内存中的数据和指令是分开存放的，进程通过不同的时间段来区分指令和数据，即在取指令阶段取出的为指令，在执行指令阶段取出的即为数据。同时，进程对指令区和数据区的访问权限不同，对指令区为只读，而对数据区为读写，以防止指令被非法改写。

因此，为方便起见，本程序使用一个数组 `Memory` 来表示内存，其大小为 `16384(B)`（即 `16KB`），它的前半部分（`0-8192B`）用来存放数据，后半部分（`8192-16384B`）用来存放指令。同时，使用变量 `PC` 来模拟计算机中的程序指针，其初始值为 `8192`，即第一条指令的位置。

本程序使用大头（`Big-endian`）方式来存放数据和指令，即将高位字节存放在低地址。例如，`0x12345678` 在内存中按地址递增方向的存放顺序为：`12,34,56,78`。另外，所有数据存放的起始地址都为 4 的倍数（即 4 个地址位存

放一个 32 位机器码）。例如，0x12345678 按照地址为 1296,1297,1298,1299 的空间进行存放， 而不是 1297,1298,1299,1300。这样一来，不同的数据和指令在内存中存放的实际地址应依次相差 4，即 0,4,8,…,16380。同理，在执行 lw,sw 等与地址有关的 MIPS 指令时，基址寄存器中存放的地址必须为 4 的倍数。

这样设计虽与实际计算机的运行存在一些差别，但原理类似，且实现较为方便，能够达到模拟的目的。

程序目前只支持 add，sub，lw，sw 与 j 这五条指令，后续会进行完善。

2.2 汇编反汇编

2.2.1 汇编

使用指针数组 sector 保存输入的汇编指令，如输入“add $s0,$s1,$s2”，则 sector[0] = “add”，sector[1] = “$s0”，sector[2] = “$s1”，sector[3] = “$s2”。然后使用 getRegName 函数得到寄存器的序数，最后根据每种指令类型的格式计算得出十六进制机器码。

1. R 类型指令（实现 add，sub）：


```
result = (getRegName(sector[2]) << 21) +
         (getRegName(sector[3]) << 16) +
           (getRegName(sector[1]) << 11) + Func;
```

其中 Func(add) = 100000，Func(sub) = 100010。

2. I 类型指令（实现 lw，sw，beq，bne）：


```
result = (op << 26) +
         (getRegName(sector[3]) << 21) +
         (getRegName(sector[1]) << 16) +
         (atol(sector[2]) & 0xFFFF);
```

其中 op(lw) = 100011，op(sw) = 101011，op(beq) = 000100，op(bne) = 000101。

3. J 类型指令（实现 j，jal）：


```
result = (op << 26) + (atol(sector[1]) & 0x3FFFFFF);
```

其中 op(j) = 000010，op(jal) = 000011。

2.2.2 反汇编

反汇编是汇编的逆过程，只需通过对机器码进行右移位分别得到各部分的值，然后由此查找寄存器数组得到寄存器名称输出即可。

```
1. add / sub
   rs = (code >> 21) & 31;
   rt = (code >> 16) & 31;
   rd = (code >> 11) & 31;
2. lw / sw
   rs = (code >> 21) & 31;
   rt = (code >> 16) & 31;
   im = (short)(code & 0xFFFF);
3. beq / bne
   rs = (code >> 21) & 31;
   rt = (code >> 16) & 31;
   im = (short)(code & 0xFFFF);
4. j / jal
   tar = (short)(code & 0x3FFFFFF);
```

2.3 R-看寄存器

本程序使用 Register 数组来保存各个寄存器中的值，并通过checkRegister()函数来进行查看。寄存器中的值既可能是数据，也可能是地址，因此考虑使用十六进制输出较为符合实际。具体显示效果如下：

register	data
$zero	0X00000000
$at	0X00000000
$a0	0X00000000
$a1	0X00000000
$a2	0X00000000
$a3	0X00000000
$v0	0X00000000
$v1	0X00000000
$t0	0X00000008
$t1	0X00000009
$t2	0X0000000A
$t3	0X0000000B
$t4	0X0000000C
$t5	0X0000000D

2.4 D-数据方式看内存

数据方式看内存指的是以 32 位机器码的形式查看内存中的数据或指令。本程序通过 checkMemoryData()函数实现这一功能。

由于查看整个内存的数据显示较为冗长，不够直观，因此程序首先要求用户输入查看的起始和终止地址（必须为4的倍数，否则程序分别按离它们最近的4的倍数来显示。具体原因见2.1）。然后，程序按照大头方式，依次取出4个子地址中的数据部分，经过计算得到实际的机器码，予以显示。计算方法为（i为首地址）：

```
result = (Memory[i].content << 24) + (Memory[i + 1].content << 16) + (Memory[i + 2].content << 8) + Memory[i + 3].content
```

具体显示效果如下：

```
Choose the range of memory address (multiple of 4 from 0-8192) you want to show.
Begin address: 6400
End address: 6404
+-------------+---------------------+--------------+
|   address   |       content       |     type     |
+-------------+---------------------+--------------+
| 0X00001900  |      0X00000000     |     data     |
+-------------+---------------------+--------------+
```

2.5 U-指令方式看内存

指令方式看内存指的是以实际指令的形式（如 add $s1, $s2, $s3）查看内存中的指令，若为数据则依然显示数据。本程序通过 checkMemoryInst()函数实现这一功能。

该功能的实现原理与数据方式看内存基本一样，起始地址与终止地址的输入要求也类似，因此此处不再赘述。不同之处在于，程序计算得到机器码后，若为数据则直接显示；若为指令则先根据其 op 与 func 的值来判断指令类型，再行转换。具体显示效果如下：

```
Choose the range of memory address (multiple of 4 from 8192-16384) you want to show.
Begin address: 8192
End address: 8200
+-------------+---------------------+--------------+
|   address   |       content       |     type     |
+-------------+---------------------+--------------+
| 0X00002000  | add   $s1, $s2, $s3 | instruction  |
+-------------+---------------------+--------------+
| 0X00002004  |         null        | instruction  |
+-------------+---------------------+--------------+
```

2.6 A-写汇编指令到内存

程序通过 insertInst()函数实现将汇编指令写入内存的功能。

首先根据用户的选择来确定要输入指令的类型，然后调用相应的汇编转换函数

来获得机器码，最后按顺序写入内存，并遵循大头原则。具体显示效果如下：

```
Choose the type of instruction your want to insert.
1. add
2. sub
3. lw
4. sw
5. beq
6. j
1
Input the instruction.
add     $s1, $s2, $s3
Inserting successful!
```

2.7 T-单步执行内存中的指令

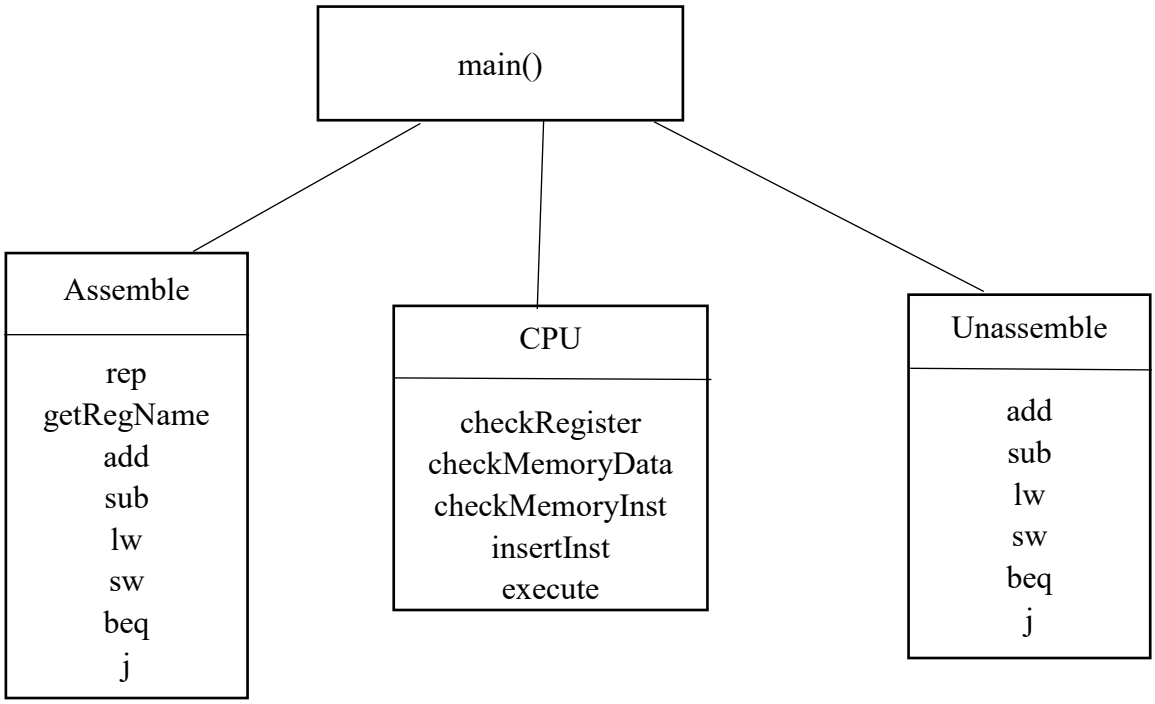
程序通过 execute()函数实现单步执行内存中指令的功能。

首先根据 PC 的值访问内存，即 Memory 数组，判断当前位置是否存有指令，若有，则取出机器码，根据其 op 与 func 的值来确定指令的类型，最后执行这条指令，并改变相应寄存器或内存中的值（即更改 Register 数组与 Memory 数组）。若当前位置没有指令，则不执行，显示出错。具体显示效果如下：

```
Please input the command.
T
Execute: add    $s1, $s2, $s3
```

三、 程序框架

本程序使用 C++语言编写，共有三个类：用于将汇编指令转换成机器码的 Assemble 类，用于将机器码转换回汇编指令的 Unassemble 类，以及用于执行与内存有关操作的 CPU 类。结构图如下：

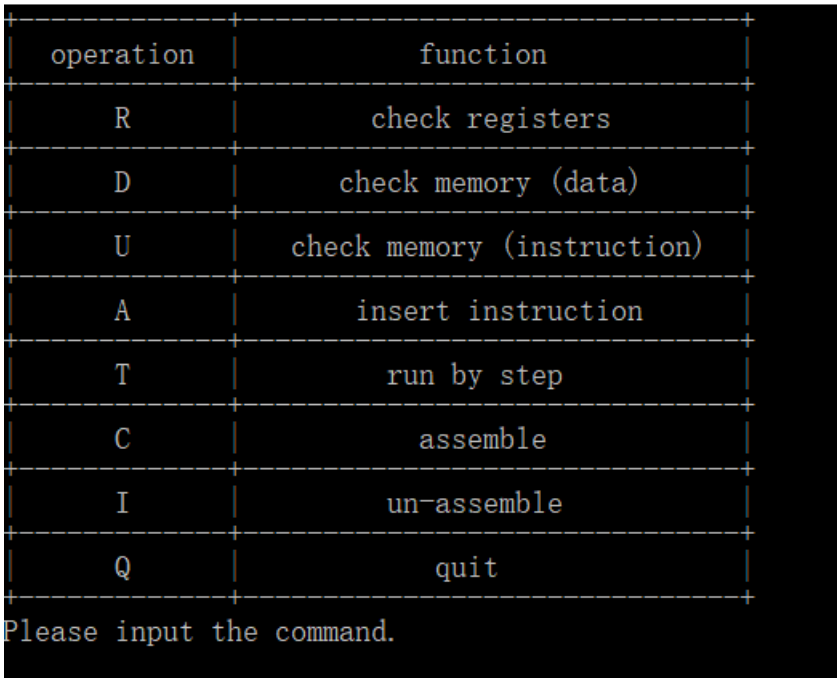


四、 运行环境

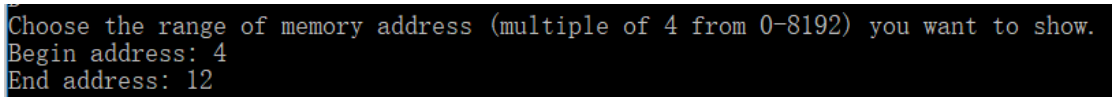
1. 操作系统：Windows
2. 编译环境：Microsoft Visual Studio 2017

五、 使用方法

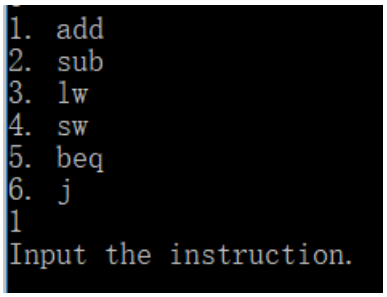
1. 运行程序，进入主界面。



2. 输入指令（需大写）来选择相应的操作。
3. 输入 D 和 U 时，后续还需输入起始地址与终止地址（4 的倍数）。



4. 输入 A、C 和 I 时，后续还需输入指令类型。



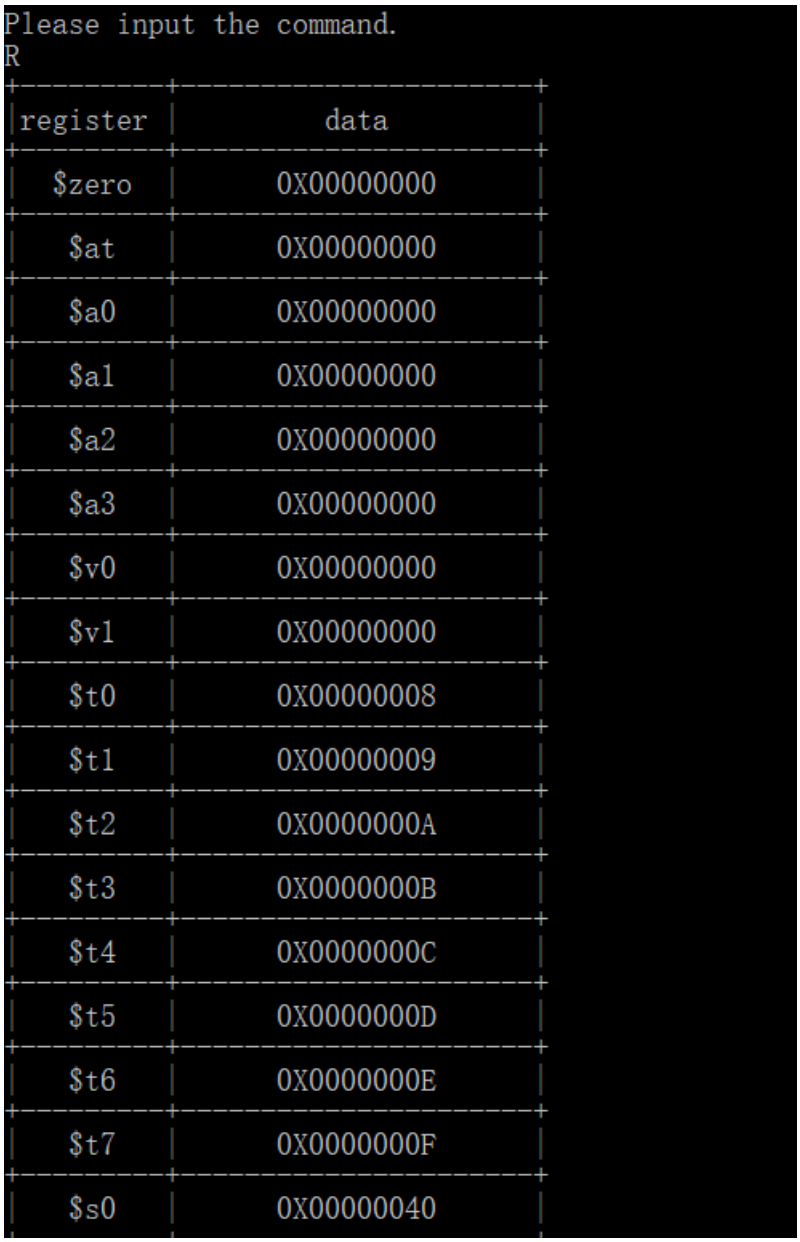
5. 输入 T 进行单步执行内存中的指令。
6. 输入 Q 退出。

六、 测试结果

为方便调试，程序初始化时将寄存器号为 8-15 的寄存器值置为 Register[i] = i，这里面存放的是数据；将寄存器号为 16-23 的寄存器值置为 Register[i] = i * 4，这里面存放的是地址；其他寄存器值都置为 0。以后涉及到数据操作的寄存器只使用$t0-$t7，涉及到地址操作的寄存器只使用$s0,$s7。

将内存 Memory 中的所有值初始化为 0。

6.1 R-看寄存器



6.2 A-写汇编指令到内存

将以下指令依次写入内存：

```
add   $t3, $t1, $t2    //$t3 = $t1 + $t2 = 19
sw    $t3, 100($s1)    //Memory[$s1 + 400 = 468] = $t3 = 19
lw    $t4, 100($s1)    //$t4 = Memory[468] = 19
j     2052   //PC = 2052*4 = 8208
beq   $v0, $zero, 300     //if($v0==$zero)   PC = PC+4+300*4
```

```
Please input the command.
A
Choose the type of instruction your want to insert.
1. add
2. sub
3. lw
4. sw
5. beq
6. j
1
Input the instruction.
add   $t3, $t1, $t2
Inserting successful!

Please input the command.
A
Choose the type of instruction your want to insert.
1. add
2. sub
3. lw
4. sw
5. beq
6. j
4
Input the instruction.
sw    $t3, 100($s1)
Inserting successful!
```

其他同理，故不再截图。

6.3 T-单步执行内存中的指令

以下依次执行上一步写入的指令。

6.3.1 add/sub

```
Please input the command.
T
Execute: add    $t3, $t1, $t2
```

观察此时的寄存器，发现$t3 的值已发生变为$t1 与$t2 的和：

$t1	0X00000009
$t2	0X0000000A
$t3	0X00000013

6.3.2 lw/sw

```
Please input the command.
T
Execute: sw     $t3, 100($s1)
```

观察此时的内存，发现地址为 468 的空间已存储了$t3 的值：

```
Choose the range of memory address (multiple of 4 from 0-8192) you want to show.
Begin address: 468
End address: 476
+-------------+---------------------+--------------+
|   address   |       content       |     type     |
+-------------+---------------------+--------------+
| 0X000001D4  |      0X00000013     |     data     |
+-------------+---------------------+--------------+
| 0X000001D8  |      0X00000000     |     data     |
+-------------+---------------------+--------------+
```

```
Please input the command.
T
Execute: lw     $t4, 100($s1)
```

观察此时的寄存器，发现$t4 的值已变为 Memory[468]中的值：

```
+---------+----------------------+
|   $t3   |      0X00000013      |
+---------+----------------------+
|   $t4   |      0X00000013      |
+---------+----------------------+
```

6.3.3 j

```
Please input the command.
T
Execute: j      2052
```

由于执行该指令后，PC 变为 2052*4=8208，该地址正好是 beq 指令所在的地址，故再单步执行一次，之前输入的 beq 指令得到执行：

```
Please input the command.
T
Execute: beq    $v0, $zero, 300
```

6.3.4 beq

由于$v0 与$zero 中的值都为 0，两者相等，PC 变为原 PC 加 4 后再加 1200，即 9412，此地址没有存放任何指令：

```
Please input the command.
T
There is no instruction at PC = 9412.
```

6.4 D-数据方式看内存

以机器码形式查看之前输入的 5 条指令在内存中的情况：

```
Please input the command.
D
Choose the range of memory address (multiple of 4 from 0-8192) you want to show.
Begin address: 8192
End address: 8224
+-------------+---------------------+---------------+
|   address   |       content       |     type      |
+-------------+---------------------+---------------+
| 0X00002000  |      0X012A5820     | instruction   |
+-------------+---------------------+---------------+
| 0X00002004  |      0XAE2B0064     | instruction   |
+-------------+---------------------+---------------+
| 0X00002008  |      0X8E2C0064     | instruction   |
+-------------+---------------------+---------------+
| 0X0000200C  |      0X08000804     | instruction   |
+-------------+---------------------+---------------+
| 0X00002010  |      0X1006012C     | instruction   |
+-------------+---------------------+---------------+
| 0X00002014  |      0X00000000     | instruction   |
+-------------+---------------------+---------------+
| 0X00002018  |      0X00000000     | instruction   |
+-------------+---------------------+---------------+
| 0X0000201C  |      0X00000000     | instruction   |
+-------------+---------------------+---------------+
```

6.5 U-指令方式看内存

以实际指令形式查看之前输入的 5 条指令在内存中的情况：

```
Please input the command.
U
Choose the range of memory address (multiple of 4 from 8192-16384) you want to show.
Begin address: 8192
End address: 8224
+-------------+----------------------+---------------+
|   address   |       content        |     type      |
+-------------+----------------------+---------------+
| 0X00002000  | add   $t3, $t1, $t2  | instruction   |
+-------------+----------------------+---------------+
| 0X00002004  | sw    $t3, 100($s1)  | instruction   |
+-------------+----------------------+---------------+
| 0X00002008  | lw    $t4, 100($s1)  | instruction   |
+-------------+----------------------+---------------+
| 0X0000200C  | j     2052           | instruction   |
+-------------+----------------------+---------------+
| 0X00002010  | beq   $v0, $zero, 300| instruction   |
+-------------+----------------------+---------------+
| 0X00002014  |         null         | instruction   |
+-------------+----------------------+---------------+
| 0X00002018  |         null         | instruction   |
+-------------+----------------------+---------------+
| 0X0000201C  |         null         | instruction   |
+-------------+----------------------+---------------+
```

6.6 C-汇编

6.6.1 add/sub

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
1
Input the instruction.
add     $s1, $s2, $s3
The machine code is: 0X02538820
```

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
2
Input the instruction.
sub     $t0, $t1, $t2
The machine code is: 0X012A4022
```

6.6.2 lw/sw

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
3
Input the instruction.
lw      $t0, 125($s1)
The machine code is: 0X8E28007D
```

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
4
Input the instruction.
sw      $s1, 23($t0)
The machine code is: 0XAD110017
```

6.6.3 beq

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
5
Input the instruction.
beq     $t0, $t1, 100
The machine code is: 0X11280064
```

6.6.4 j

```
Please input the command.
C
1. add
2. sub
3. lw
4. sw
5. beq
6. j
6
Input the instruction.
j       123
The machine code is: 0X0800007B
```

6.7 I-反汇编

6.7.1 add/sub

```
Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
1
Input the machine code.
0X02538820
The instruction is: add    $s1, $s2, $s3
```

```
Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
2
Input the machine code.
0X012A4022
The instruction is: sub    $t0, $t1, $t2
```

6.7.2 lw/sw

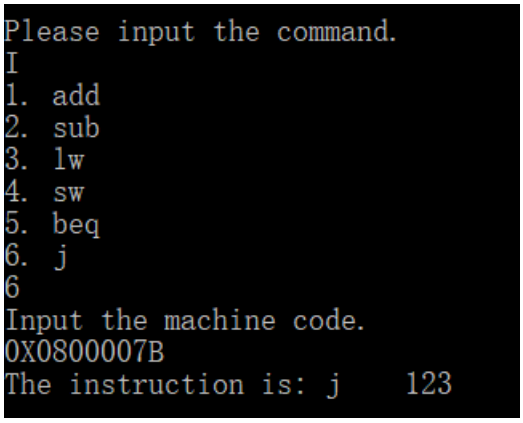
```
Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
3
Input the machine code.
0X8E28007D
The instruction is: lw    $t0, 125($s1)
```

```
Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
4
Input the machine code.
0XAD110017
The instruction is: sw    $s1, 23($t0)
```

6.7.3 beq

```
Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
5
Input the machine code.
0X11280064
The instruction is: beq    $t0, $t1, 100
```

6.7.4 j


Please input the command.
I
1. add
2. sub
3. lw
4. sw
5. beq
6. j
6
Input the machine code.
0X0800007B
The instruction is: j 123