

# 计算机系统结构实验二-Tomasulo算法模拟器 Report

计62 胡致远 2016011260

## 设计思路

模拟器的设计思路是维护一个station,一个board, 一个寄存器组用来记录算法当前运行的状态, 然后根据tomasulo算法中描述四个阶段反复运行, 直到station中没有处于busy的组件且所有下一条需要issue的指令id为总指令数+1

tomasulo算法中四个过程的执行顺序为:

```
WriteBack
Issue
Update/Exec
Check ready for exec
```

## 编译与执行方法

执行make执行可以进行编译, 执行 ./tomasulo InputFileName可以运行程序

## 简易的交互界面

为了方便用户运行程序, 设计了类似GDB的简易交互界面

算法启动后, 将读取文件并显示指令, 然后进入交互界面, 如下:

```
binah@SeventhHeavendeMacBook-Pro:~/study/SA/experiment2/Tomasulo$ ./tomasulo test0.nel
=====INSTRUCTION=====
InstID  OP      Dest   Vj      Vk
0       LD      F1      2      EMPTY
1       LD      F2      1      EMPTY
2       LD      F3     -1      EMPTY
3       SUB      F1      F1      F2
4       DIV      F4      F3      F1
5       JUMP     0       F1      2
6       JUMP    -1      F3     -3
7       MUL      F3      F1      F4

(Tomasulo)
```

执行h可以查看交互界面帮助:

```

(Tomasulo)h
Tomasulo Usage:
n                -- take one step
c                -- continue until program end
s [type]         -- display
s a              -- display All
s i              -- display Instructions
s b              -- display Board status
s s              -- display Station status
s r              -- display Register status
q                -- quit
h                -- show this help
(Tomasulo)

```

执行n可以运行一步(前进一个时钟):

```

(Tomasulo)n
cycle 1

Issue Instruction 0
Station 9 is ready now
Station 9 Exec
(Tomasulo)n
cycle 2

Issue Instruction 1
Station 10 is ready now
Station 10 Exec
(Tomasulo)n
cycle 3

Issue Instruction 2
Station 11 is ready now
(Tomasulo)

```

执行s可以显示station,board和register的信息(为了显示方便寄存器个数设置为了16个):

```

(Tomasulo)s
=====BOARD=====
InstID  IssueTime      ExecCompTime  WriteResultTime
0        1             -1             -1
1        2             -1             -1
2        3             -1             -1

=====STATION=====
Comp  Busy  Op   Vj   Vk   Qj   Qk   Addr  status  remaincycle
Loader0 yes  LD   2    0           2    RUNNING  1
Loader1 yes  LD   1    0           1    RUNNING  2
Loader2 yes  LD  -1    0          -1    READY   0

=====REGISTER=====
RegID  0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15
Addr   0    Loader0 Loader1 Loader2 0    0    0    0    0    0    0    0    0    0    0    0
(Tomasulo)

```

执行c可以连续执行直到程序结束:

```

(Tomasulo)n
cycle 28

(Tomasulo)n
cycle 29

(Tomasulo)c
cycle 30

cycle 31

Station 8 is over
cycle 32

Station 8 Write back
(Tomasulo)

```

执行q可以退出程序：

```

Station 8 Write back
(Tomasulo)q
binah@SeventhHeavendeMacBook-Pro:~/study/SA/experiment2/Tomasulo$

```

## 分支预测

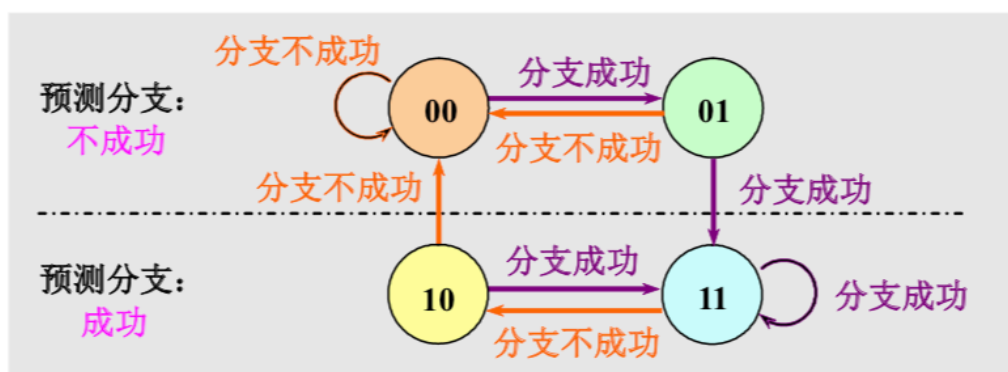
在本程序中实现了基于历史的分支预测，具体原理参照了课件：

### 1-采用分支历史表 BHT

#### 3. 采用两位二进制位来记录历史

- 提高预测的准确度
- **研究表明：**两位分支预测的性能与n位 ( $n > 2$ ) 分支预测的性能差不多。

➤ 两位分支预测的状态转换如下所示：



即使用两位二进制位来记录每条指令对应的分支历史，然后根据指令的历史决定预测的结果

对于预测失败时状态的复原，采用如下方法：

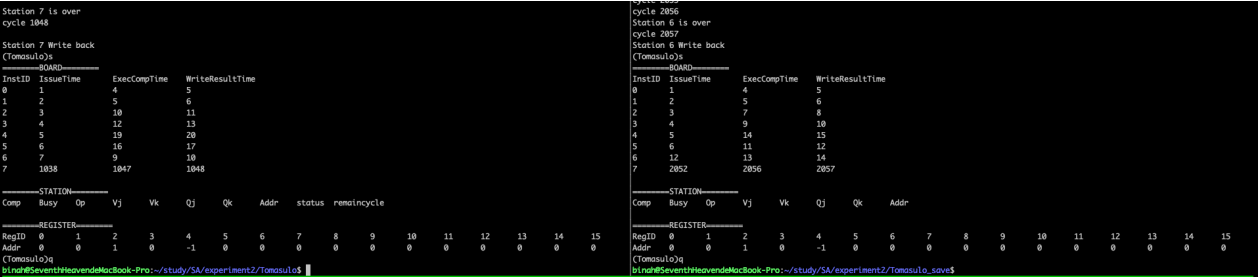
在每次Issue Jump指令时，将全局的状态(status)保存下来，由于可能同时有多个Jump指令正在运行，需要将这些status组织成一个链表，当一条Jump指令写回时，如果发现预测失败，则将对应的status中全局的状态恢复即可。

## 分支预测的实验结果

为了检测上述分支预测算法的合理性，在test0.nel的基础上编写了一个简单的程序，主要完成的任务是进行1600次循环，该nel程序的代码如下：

```
LD,F1,0x100 LD,F2,0x1 LD,F3,0xFFFFFFFF SUB,F1,F1,F2 DIV,F4,F3,F1 JUMP,0x0,F1,0x2
JUMP,0xFFFFFFFF,F3,0xFFFFFFFF MUL,F3,F1,F4
```

分别使用带分支预测的版本(左侧程序)和不带分支预测的版本(右侧程序)执行，结果如下(为了显示方便，寄存器个数设置为16个)：



可以看出，不带分支预测时，整个程序运行了2057个周期才结束，而加上分支预测后，程序只运行了1048个周期，性能提升了96.27%，可见我们设计的分支预测算法是正确的。