Tomasulo Algorithm Implementation

1. Introduction

Tomasulo's algorithm is for dynamic scheduling of instructions that allows out-of-order execution and enables more efficient use of multiple execution units. **In this assignment, you will implement Tomasulo's algorithm using your familiar programming language (e.g., c/c++/java/python).**

For a given instruction execution sequence, your program should output instruction status, reservation status, load buffer status and register result status cycle by cycle. The sample output should be offered in clarity and readability just like "Chap03-ILP-Part3-TomasuloExample". **Please refer to the appendix for specific input and output formats.**

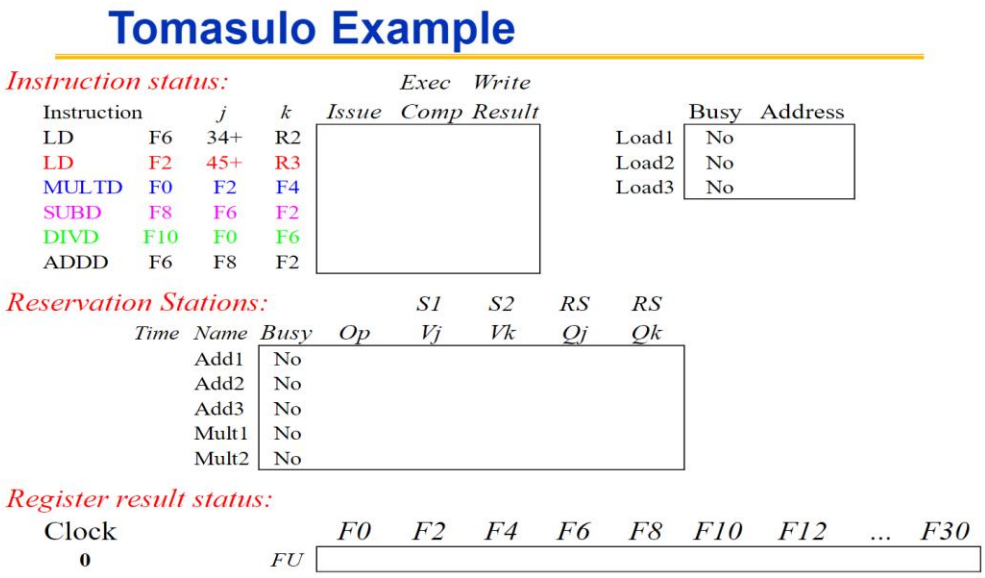


Figure 1 Sample output

2. About your machine

For this problem use the single-issue Tomasulo MIPS pipe-line of Figure 3.6 with the pipeline latencies from the table above.

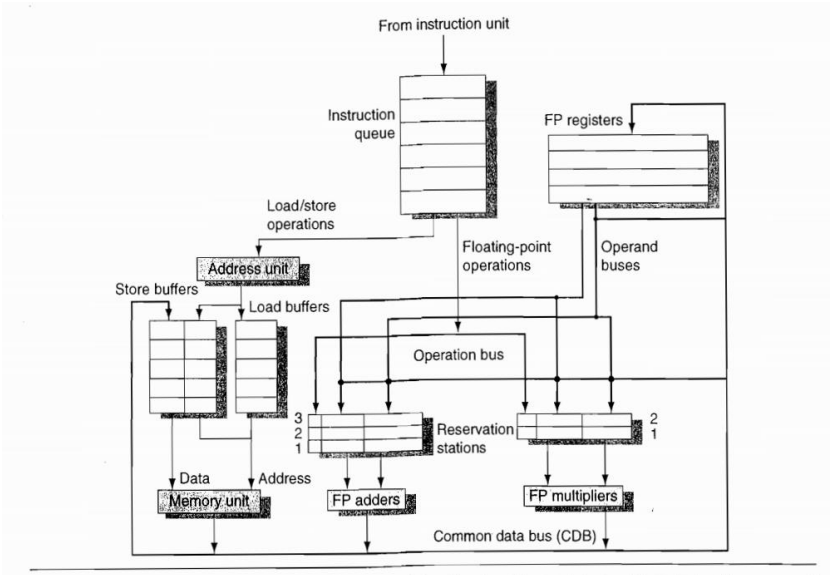


Figure 3.6 The basic structure of a MIPS floating-point unit using Tomasulo's algorithm. Instructions are sent from the instruction unit into the instruction queue from which they are issued in first-in, first-out (FIFO) order. The reservation stations include the operation and the actual operands, as well as information used for detecting and resolving hazards. Load buffers have three functions: (1) hold the components of the effective address until it is computed, (2) track outstanding loads that are waiting on the memory, and (3) hold the results of completed loads that are waiting for the CDB. Similarly, store buffers have three functions: (1) hold the components of the effective address until it is computed, (2) hold the destination memory addresses of outstanding stores that are waiting for the data value to store, and (3) hold the address and value to store until the memory unit is available. All results from either the FP units or the load unit are put on the CDB, which goes to the FP register file as well as to the reservation stations and store buffers. The FP adders implement addition and subtraction, and the FP multipliers do multiplication and division.

Assume the following:

- Functional units are not pipelined.
- There is no forwarding between functional units; results are communicated by the common data bus (CDB).
- The execution stage (EX) does both the effective address calculation and the memory access for loads and stores. Thus, the pipeline is IF/ID/IS/EX/WB. Loads require two clock cycle.
- The issue (IS) and write-back (WB) result stages each require one clock cycle. There are three load buffer slots and three store buffer slots.

- Assume that the Branch on Not Equal to Zero (BNEZ) instruction requires one clock cycle.

Part of execution latency assumptions of FP instruction are described in the table below.

FP instruction	EX Cycles
fadd	2
fsub	2
fmult	10
fdiv	20

Note: some of the assumptions not mentioned above but used in programming should be given in the report for easy correction

3. Assignment submission

Please provide a single **pdf document** which shows that your program can successfully complete dynamic scheduling of instructions illustrated in "input1.txt" and "input2.txt". Meanwhile, your submission should also provide the **original source code** and the **output file** for reviewing. Code annotation is needed to improve code readability.

4. Scoring

The assignment is evaluated from the following aspects:

report (40%), output correctness (30%), code readability (30%)

Note that plagiarism is prohibited!

附输入输出格式规定：

输入格式：

一条指令一行并以 txt 文件格式读入，可能的指令输入格式如下图 2 所示。

```
1  LD F6 34+ R2
2  LD F2 45+ R3
3  MULTD 0 F2 F4
4  SUBD F8 F6 F2
5  DIVD F10 F0 F6
6  ADDD F6 F8 F2
```

图 2 指令输入格式

输出格式：

由每个周期状态和指令最终执行情况表两部分输出组成，最终输出到 ouput1.txt 或者 ouput2.txt 文件中：

1、 每个时钟周期状态输出格式要求：

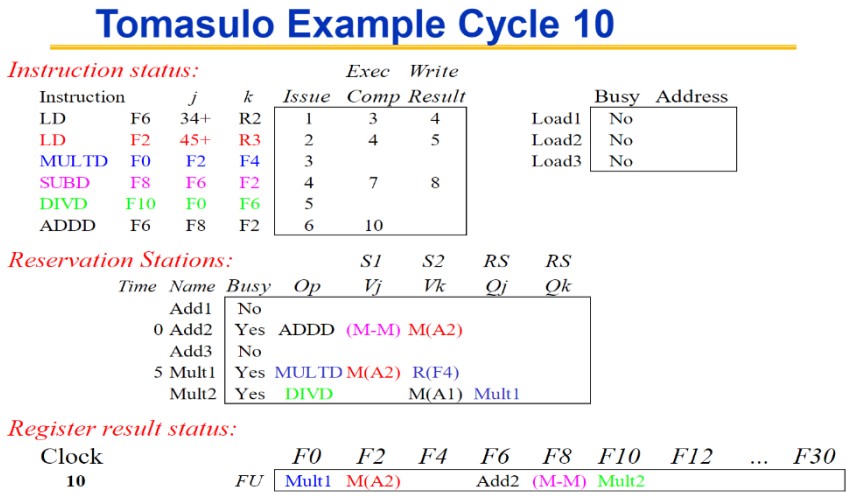
- 每个 cycle 输出 13 行状态信息，其中 cycle 标识 1 行，load buffer 状态 3 行，store buffer 状态 3 行，RS 状态 5 行，FU 状态 1 行。
- 请用“;” 间隔不同实体，用“:” 间隔实体标识符和实体属性，用“,”间隔同一实体不同属性。
- 如属性为空或未知，请留空。
- 多个连续周期状态相同仅输出首个相同周期状态，状态标识修改为“cycle_n-m”（其中 n 为第一个周期相同，m 为最后一个相同周期）。

综上，每个周期输出的状态格式应如图 3 所示：

```
//Cycle_n;
//Load1:(busy),(address);
//Load2:(busy),(address);
//Load3:(busy),(address);
//Store1:(busy),(address),(Fu);
//Store2:(busy),(address),(Fu);
//Store3:(busy),(address),(Fu);
//Add1:(busy),(op),(Vj),(Vk),(Qj),(Qk);
//Add2:(busy),(op),(Vj),(Vk),(Qj),(Qk);
//Add3:(busy),(op),(Vj),(Vk),(Qj),(Qk);
//Mult1:(busy),(op),(Vj),(Vk),(Qj),(Qk);
//Mult2:(busy),(op),(Vj),(Vk),(Qj),(Qk);
//F0:(status);F2:(status);...;F12:(status);
```

图 3 单个周期状态输出格式

以“Chap03-ILP-Part3-TomasuloExample”PPT 中第 1 个例子第 10 周期为例，一个可能的输出如下（仅供格式参考，具体内容应根据实际运行情况获得）：




```
Load1:No,;
Load2:No,;
Load3:No,;
Store1:No,,;
Store2:No,,;
Store3:No,,;
Add1:No,,,,,;
Add2:Yes,ADDD,M(34+R2)-M(45+R3),M(45+R3),,;
Add3:No,,,,,;
Mult1:Yes,MULTD,M(45+R3),R(F4),,;
Mult2:Yes,DIVD,,M(34+R2),Mult1,;
F0:Mult1;F2:M(45+R3);F4:;F6:Add2;F8:M(34+R2)-M(45+R3);F10:Mult2;F12:;
```

图 4 一个可能的周期状态及其对应输出

2、 最终指令最终执行情况表输出要求：仅在所有周期状态输出完后输出一次，一条指令对应一行，具体格式为：“(Instruction):(Issue cycle),(Exec comp cycle),(Write rusult cycle);”，一个可能的输出如下图 5 所示：

```
LD F6 34+ R2 :1,3,4;
LD F2 45+ R3 :2,4,5;
MULTD 0 F2 F4 :3,15,16;
SUBD F8 F6 F2 :4,7,8;
DIVD F10 F0 F6 :5;56,57;
ADDD F6 F8 F2 :6,10,11;
```

图 5 一个可能的最终指令最终执行情况表输出