2025 计算机系统实现 作业检查
50 指令流水线 CPU

ceerRep

I) 简介

本次作业检查分为 **两个部分**。

1. 你需要 **尽可能的** 对你的 MIPS CPU 按照要求回答问题，根据结果撰写报告并提交到 OBE 上。**提交期限为 2025 年 1 月 5 日 23:55 UTC+8。逾期 d 天扣除 $10\% \times 2^{d-1}$ 分数，d 向上取整。**
2. 你需要给你的 50 指令 MIPS CPU 代码编写配置文件，方便进行批量测试。将增加了配置文件的代码提交到 OBE 上 50 指令 MIPS CPU 实验中。提交时限为 2025 年 1 月 6 日 23:55 UTC+8。逾期只算迟交作业，扣除固定分数（不超过 30%）。**开始评测作业后不接受任何自行补交，这个时间不早于 2025 年 1 月 7 日 11:55 UTC+8。**

你的最终期末实验分数由 CPU 实现代码分数 score_code、CPU 实现报告分数 score_report、本次检查第一部分分数 score_check 和检查系数映射函数 f 构成。

具体公式为 $(\mathbf{score\_code} + \mathbf{score\_report}) \times \boldsymbol{f}(\mathbf{score\_check})$。保证映射函数 f 单调不减。附加题可以让 f 的函数值超过 1.0

II) 简答题

本节测试你对你代码的理解程度。你需要 **结合你写的代码 尽可能** 回答下面的问题。将答案提交到 OBE 上。

请注意：评分并不是按照对题目的完成程度评分，而是对你代码的理解程度进行评分。

作答时允许使用 AIGC 工具，但注意工具只能用来辅助理解。

如果报告中展示过多 AI 对你代码的理解程度而不是你对你代码的理解程度，会酌情扣分。

II.A) 流水线旁路机制分析

ChatGPT:

旁路机制 (Forwarding/Bypassing) 是一种用来解决流水线数据相关冒险的数据通路优化技术。它的作用是在指令还没有把结果写回寄存器文件之前，就把运算结果直接从执行阶段或访存阶段“旁路”传送给后续指令需要使用该结果的执行阶段输入，从而避免因为等待写回导致的流水线停顿。通过旁路机制，很多原本需要插入气泡（stall）的数据相关情况可以在硬件层面被消除，使得相关指令能够连续执行，从而提升流水线吞吐率与总体性能，同时减少性能损失和延迟。

1. **结合你的代码**，讲一讲下面的机制是怎么体现的
   - 怎么识别后执行的指令是否用了前执行的指令还没写回寄存器的内容?
   - 指令 stall 在你的代码里什么地方涉及? 你的代码通过什么方法判断需要 stall 几个周期的?
   - 旁路机制是怎么跨流水线转发的? 你的代码通过什么方法判断需要使用旁路数据而不是真的寄存器数据?

II.B) 乘法单元机制分析

我们提供的乘法器 (MDU) 运算需要多个周期。因此，将乘法器和旁路机制结合有独特的挑战。结合你的代码回答以下问题（Lx 代表下面代码的第 x 行）。

```mips
mult $1, $2
addu $10, $11, $12 # 一个无关指令
mflo $3
mfhi $4

addu  $3, $3, $4   # 一个用了乘法输出的指令

mult $1, $2        # 一个乘法
mflo $3
mfhi $4
mult $5, $6        # 另一个乘法
mflo $7
mfhi $8
addu $3, $3, $7    # 处理结果
```

1. 你的乘法器被占用时，不使用乘法器的指令（L2）可以正常执行吗？如果能正常执行，你是怎么做的？如果不能，进行什么修改能让你的 CPU 在进行乘法运算时可以执行其他不使用乘法器的指令？请结合你代码的 stall 生成机制作答。
2. 实验手册要求乘法器被占用时，任何使用乘法器的指令都会被卡在 `EX` 级。请结合你的代码，构思一种方法使得上面的代码在执行时，不卡在 L3 L4，而卡在 L6。
   注意：L6 从 EX 级恢复执行前，往 `$3` `$4` 写入应当重新插入回流水线的 WB 级中。流水线开始正常流动前要保证一级流水线上只有一条指令。
   提示：标记 `mflo` `mfhi` 的目标寄存器为未完成，插入气泡。
3. （附加题）如果你有两个乘法器，在什么文件作出什么修改能让你的 CPU 同时进行两个乘法，让上面代码在 L8-L13 不阻塞，在 L14 阻塞？

III) OBE 大作业提交

III.A) 实验 1-实验 6 提交

实验 1-实验 6 的代码和报告（如果实验手册不要求，以实验手册为准）都应提交到 OBE 上，DDL 为 **2025 年 1 月 6 日 23:55，超出后不接受自行补交**。

实验 5 单周期 CPU 还有同学没有检查。今晚 **2025 年 1 月 5 日 23:55** 前 OBE 上这个实验的分数将会更新，1 分说明已经检查了，0 分说明还没被检查，请单独找我检查。

实验 6 检查和期末实验检查合在一起（因为代码基本一致），但实验报告仍然要有。

OBE 崩溃时没提交的作业和实验都需要在 OBE 上提交。

如果之前的实验有过大没能提交到 OBE 上的，参考 小节 III.B.iii 进行清理之后提交。提交日期我会按照之前发邮箱里的填写，但代码需要在 OBE 上。

III.B) 期末实验 实验 7 提交

为了统一测试，需要大家整理自己的代码并编写配置文件。

III.B.i) 编写配置文件

配置文件是一个放在你的提交文件夹下的 json 文件，形如

```json
```

```
{
    "compiler": "vivado",
    "test_file": "./PipelineCPU-50inst/PipelineCPU-50inst.xpr",
    "data_path": "./PipelineCPU-50inst/PipelineCPU-50inst.srcs/sources_1/imports/src/
DataMemory.sv",
    "code_path": "./PipelineCPU-50inst/PipelineCPU-50inst.srcs/sources_1/imports/src/
InstructionMemory.sv"
}
```

1. Compiler

compiler 指的是编译器，可选项为 `vivado` 或者 `iverilog`。

2. test_file
   `test_file` 是传递给编译器的文件，如果是 `vivado`，这里要填你的 `xpr` 工程文件的 **相对路径**。

   如果是 `iverilog`，这里填你的 mips_tb.v 文件的 **相对路径**。注意， `iverilog` 模式下默认编译子文件夹下的所有 `.v` `.sv` 文件。如果你 `` `include `` 了一个文件且这个文件内定义了一个模块（例如 `MDU.sv`），你可以在文件名前加下划线 `MDU.sv -> _MDU.sv` 来将其排除出编译参数，防止模块重定义。

3. data_path
   你的 DataMemory 定义文件路径。

4. code_path
   你的 InstructionMemory 定义文件路径，里面应当有且仅有一个 `$readmemh` 读取初始化行。

注意，所有路径都应该是相对于提交文件夹（json 所在文件夹）的相对路径，路径分隔符统一使用斜杠 / 而不是反斜杠 \，写 “./some_folder_name/test.v” 而不是 “.\\some_folder_name\\test.v”。

5. 源代码文件
   为了加速仿真，你的所有源代码文件开头应有 `` `timescale 1us/1us ``。

最终提交时，你的压缩包内目录结构应当类似

```
20XXXXXXXX 你的学号-根文件夹
- meta.json 上述配置文件
- XXX_vivado_project 一个工程文件夹，名字只是例子
  - XXX_vivado_project.xpr Vivado 工程文件
  - src
    - … / … / … 一些子文件夹
      - dmem.v
      - imem.v
- XXX_iverilog_project 另一个工程文件夹，名字只是例子
  - cpu.v
  - mips_tb.v
  - dmem.v
  - imem.v
  - _MDU.sv
```

你的 `meta.json` 内容应当类似

json

```
{
  "compiler": "vivado",
  "test_file": "./XXX_vivado_project/XXX_vivado_project.xpr",
```

```
  "data_path": "./XXX_vivado_project/src/..../dmem.v",
  "code_path": "./XXX_vivado_project/src/..../imem.v"
}
```

或者

```json
{
  "compiler": "iverilog",
  "test_file": "./XXX_iverilog_project/mips_tb.v",
  "data_path": "./XXX_iverilog_project/dmem.v",
  "code_path": "./XXX_iverilog_project/imem.v"
}
```

上面两段代码分别是示例提交目录结构里的 Vivado 部分和 iverilog 部分。

III.B.ii) 检验配置文件

压缩包内有配置文件检验脚本 `check.py`。使用方法如下

打开命令行,

- 使用 uv

```bash
uv run check.py 你的配置json路径
```

- 使用 pip

```bash
python3 -m pip install termcolor
python3 check.py 你的配置json路径
```

输出有三种颜色，绿色不用管，黄色需要人去检查，红色说明有问题。 如果报了异常说明问题更大。

III.B.iii) Vivado 清理

由于 Vivado 的临时文件非常大，提交 OBE 前可以先删除一部分。

首先备份你的工程目录 。

1. 关闭 Vivado
2. 删除 `.runs` `.cache` `.Xil` `sim` 文件夹
3. 重新打开工程，查看能否正常仿真
4. 如果可以，关闭 Vivado，重新执行 2，然后打包代码。
5. 如果不行，还原你的备份，一个一个尝试，直到既可以仿真也可以提交 OBE。
   永远不要在你的备份工程里删除文件。