191220154 张涵之 第 5 章作业

3. 每次按同步方式从主存读取一个字，则不需要使用 WMFC 控制信号。

1) CALL 指令功能：设置 PC 为子程序起始地址（从而跳转执行子程序）并将 CALL 指令后一条指令的首地址保存到栈中（便于子程序执行完毕后返回）。

2) 读取并执行 CALL 指令分为取指令译码、设置 PC 和保存返回地址三个阶段：

```
① PCout, MOVb, MARin        // 通过总线 B 将 PC 的内容送到 MAR 的输入端
② read, PCout, b + 1, PCin  // 取指令操作码，并设置 PC = PC + 1
③ MDRout, MOVb, IRin        // 将 MDR 的内容通过总线 B 送到指令寄存器 IR
④ PCout, MOVb, MARin        // 通过总线 B 将新 PC 的内容送到 MAR 的输入端
⑤ read, PCout, b + 1, Yin   // 取子程序地址，将下条指令地址送到寄存器 Y
⑥ MDRout, MOVb, PCin        // 将 MDR 的内容通过总线 B 送到 PC
⑦ SPout, MOVb, MARin        // 将栈指示器 SP 的内容送到 MAR 的输入端
⑧ Yout, MOVb, MDRin         // 将寄存器 Y 的内容送到 MDR 的输入端
⑨ write, SPout, b + 1, SPin // 保存返回地址到栈顶，并更新 SP = SP + 1
```

以需要时间最长的 read/write 为时间周期设置标准，至少需要 9 个时钟周期。

4. 写出下列指令在指令执行阶段的控制信号序列，并说明需要几个时钟周期。

1) R[R1] ← R[R1] + imm16：不需要访问存储器，共需要 3 个时钟周期

```
MDRout, Yin       // 将 MDR 的内容（即 imm16）送到寄存器 Y
R1out, add, (Zin) // 在 ALU 中 R1 与 Y 的内容相加，输出送到寄存器 Z
Zout, R1in        // 将寄存器 Z 中的内容送到 R1
```

2) R[R1] ← R[R1] + M[imm16]：访问一次存储器，共需要 5 个时钟周期

```
MDRout, MARin       // 将 MDR 的内容（imm16）送到 MAR
read1, R1out, Yin   // 取 M[imm16]到 MDR，R1 的内容送到 Y
read2               // 等待存储器访问操作完成
MDRout, add, (Zin)  // 在 ALU 中 MDR 与 Y 的内容相加，输出送到 Z
Zout, R1in          // 将寄存器 Z 中的内容送到 R1
```

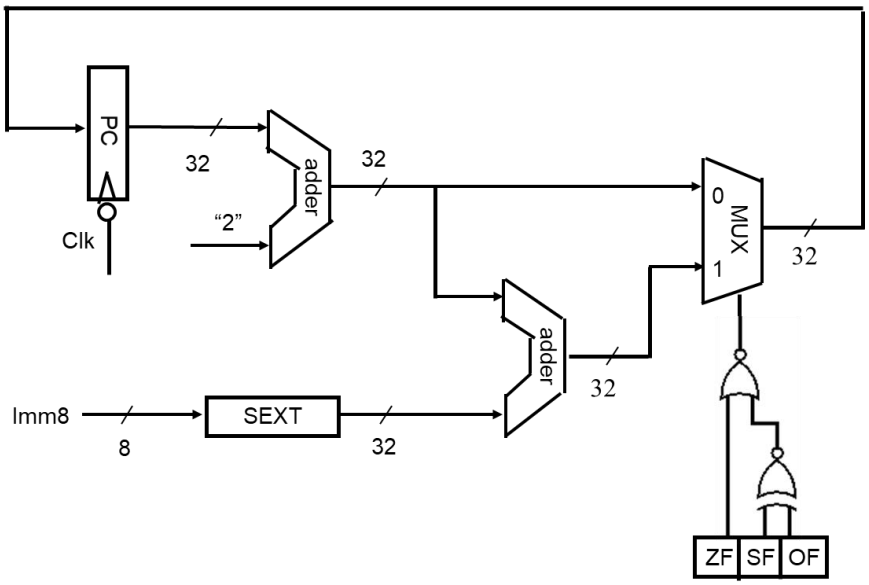
3) R[R1] ← R[R1] + M[M[imm16]]：访问两次存储器，共需要 8 个时钟周期

```
MDRout, MARin       // 将 MDR 的内容（imm16）送到 MAR
read1               // 取 M[imm16]到 MDR
read2               // 等待存储器访问操作完成
MDRout, MARin       // 再将 MDR 的内容（M[imm16]）送到 MAR
read1, R1out, Yin   // 取 M[M[imm16]]到 MDR，R1 的内容送到 Y
read2               // 等待存储器访问操作完成
MDRout, add, (Zin)  // 在 ALU 中 MDR 与 Y 的内容相加，输出送到 Z
Zout, R1in          // 将寄存器 Z 中的内容送到 R1
```

5. 完成如下要求并回答问题：

   1) 该计算机采用双字节定长指令字，当 bgt 指令条件不满足时，通过 PC = PC + 2 实现正常的顺序执行，则下一条指令的相对偏移量为 2，即存储器按字节编址。

   2) 如下图为实现 bgt 指令的数据通路。



6. RegWr = 0：无法写目的寄存器，所有 R 型、I 型、U 型和 J 型指令不能正确执行。
   ALUASrc = 0：总是选 busA 作为 ALU 的 A 口操作数，J 型指令不能正确执行。
   Branch = 0：只有 B 型指令不能正确执行。
   Jump = 0：只有 J 型指令不能正确执行。
   MemWr = 0：总是不写信息到数据存储器，S 型（Store）指令不能正确执行。
   MemtoReg = 0：总是选 ALU 的输出送目的寄存器，Load 指令不能正确执行。

7. RegWr = 1：总是写目的寄存器，S 型（Store）和 B 型指令不能正确执行。
   ALUASrc = 1：总是选 PC 作为 A 口操作数，除了 J 型指令和 lui 都不能正确执行。
   Branch = 1：除了 B 型指令都不能正确执行。
   Jump = 1：除了 J 型指令都不能正确执行。
   MemWr = 1：总是写信息到数据存储器，除了 S 型（Store）指令都不能正确执行。
   MemtoReg = 1：总是选数据存储器的输出送目的寄存器，除了 S 型和 B 型不写入目的寄存器，以及 Load 指令确实取的是数据存储器输出，其他指令都不能正确执行。

8. 若要在 RV32I 指令集中增加一条 swap 指令（功能是交换两个寄存器的内容）。

   1) 要用伪指令方式实现“swap rs, rt”的指令序列，且不使用其他额外寄存器以免破坏这些寄存器的值，则可以用三次异或操作来实现两个寄存器内容的互换。

      ```
      xor     rs, rs, rt
      xor     rt, rs, rt
      xor     rs, rs, rt
      ```

2) 设 swap 指令在程序中所占比例为 x（$0 \le x \le 1$），则采用伪指令方式（软件方式）增加 swap 指令后，程序运行所需时间为原先的 $1 - x + 3x = 1 + 2x$ 倍；若采用硬件方式实现，所需时间为原先的 1.1 倍，令 $1 + 2x > 1.1$，解得 $x > 0.05$，即 swap 指令要在程序中占比超过 5%，值得用硬件方式实现才比软件方式更经济实惠。

3) 在单周期数据通路中，一条指令必须在一个时钟周期内完成，且一个时钟周期内只能写一次寄存器。由于 swap 指令的功能是交换两个寄存器的内容，则至少写两次寄存器，不能在一个时钟周期内完成，也就无法在不对通用寄存器组进行修改（以支持同时写两个以上的寄存器）的情况下，直接采用硬件方式实现。
在多周期数据通路中，一个指令可通过多个时钟周期来完成，因此也可以写多次寄存器，即使不对通用寄存器组进行修改，也可以直接采用硬件方式实现。

10. PcWr = 0：取指令阶段无法将下一条指令地址送 PC，则所有指令都不能正确执行。
MARWr = 0：无法写入 Load/Store 指令的地址，则 Load/Store 指令不能正确执行。
RegWr = 0：无法写入寄存器，则 R 型指令、I 型运算指令、Load 指令不能正确执行。
BMUX = 0：无法满足 RExec 状态的要求，R 型指令不能正确执行。
PCout = 0：无法取指令，则所有指令都不能正确执行。

11. PcWr = 1：每个时钟都不停地更新 PC，则所有的指令执行都会失去控制。
MARWr = 1：每个时钟都更新 MAR，但除了 Load/Store 指令都不会送入总线；而对于 Load/Store 指令，MAR 的内容将在“投机”计算指令地址暂存的下一个周期，即 lwExec 和 swExec 状态下被送入总线，此时 MAR 还没来得及被更新。由于 lwFinish 状态不需要读 MAR，Load 仍能正确执行；swFinish 状态需要 MAR，则 Store 不能正确执行。
RegWr = 1：总是写目的寄存器，S 型（Store）和 B 型指令不能正确执行。
BMUX = 1：无法满足 IExec 状态的要求，I 型指令不能正确执行。
PCout = 1：无法满足 lwExec、SwExec 的要求，Load/Store 指令都不能正确执行。

14. 假定在一个 5 级流水线（图 5.43）处理器中：回答以下问题。

1) 执行阶段 EX 所用的 ALU 操作时间缩短 20%不能加快流水线的执行速度。因为根据流水线设计的原则，流水段的长度以最复杂的操作（此处为存储单元, 200ps）为准，ALU 操作时间缩短 20%，流水段的长度仍为 200ps，执行速度不变。

2) 若 ALU 操作时间增加 20%，对流水线的性能没有影响。因为 ALU 的操作时间原本为 150ps，增加 20%为 180ps，仍小于 200ps，流水段长度不变，执行速度不变。

3) 若 ALU 操作时间增加 40%，流水线的性能会变差，因为 ALU 的操作时间增进 40%为 210ps，大于 200ps，流水段长度也应增加到 210ps，执行耗时增加了 5%。

16. A: 80ps，B: 30ps，C: 60ps，D: 50ps，E: 70ps，F: 10ps，寄存器延迟：20ps。

1) 插入 1 个流水段寄存器，得到一个 2 级流水线：
在 C 和 D 之间插入，时钟周期取 $\max(80 + 30 + 60, 50 + 70 + 10) + 20 = 190$ps，指令吞吐率为 1/190ps = 5.26G 条/秒，指令执行时间为 $2 \times 190 = 380$ps

2) 插入 2 个流水段寄存器，得到一个 3 级流水线：
分别在 B 和 C、D 和 E 之间插入，时钟周期取 max(80 + 30, 60 + 50, 70 + 10) + 20 = 130ps，指令吞吐率 1/130ps = 7.69G 条/秒，指令执行时间为 3 × 130 = 390ps

3) 插入 3 个流水段寄存器，得到一个 4 级流水线：
分别在 A 和 B、C 和 D、D 和 E 之间插入，时钟周期取 max(80, 30 + 60, 50, 70 + 10) + 20 = 110ps，指令吞吐率 1/110ps = 9.09G 条/秒，指令执行时间为 4 × 110 = 440ps

4) 吞吐量最大的流水线：
各部分延迟中最大的为 80ps，则时钟周期最小为 80 + 20 = 100ps，观察相邻的各部分中，E 和 F 合并后仍不超过 80ps，则插入 4 个流水段寄存器，分别在 A 和 B、B 和 C、C 和 D、D 和 E 之间，得到一个 5 级流水线。此时时钟周期取 100ps，指令吞吐率 1/100ps = 10G 条/秒，指令执行时间为 5 × 100 = 500ps

17. 发生数据相关的指令对是 1-2（s3）、2-3（t2）、2-4（t2）以及 3-4（t1）。如果不采用“转发”技术，需要在分别在第 2、第 3、第 4 条指令前各插入 3 条 nop 指令才能避免数据冒险。如果采用“转发技术”，则第 3 和第 4 条指令之间的数据冒险不能完全解决，仍需要在第 4 条指令前加入 1 条 nop 指令，才能使这段 RV32I 程序不发生数据冒险。

18. RV32I 指令序列在图 5.43 所示的流水线数据通路中执行：回答以下问题。

1) 第 1 条指令在 EX 阶段结束时已经得到 s3 的新值，可以直接从 EX/M 流水段寄存器取出送到 ALU 的输入端，这样在第二条指令执行 ALU 时用的就是 s3 的新值；
第 2 条指令在 EX 阶段结束时已经得到 t2 的新值，可以直接从 EX/M 流水段寄存器取出送到 ALU 的输入端，供第 3 条指令执行 ALU 时使用；
同样，第 4 条指令在 ALU 中用到的 t2 可以直接从 M/WB 流水段寄存器获得；
第 3 条指令在 M 段结束时得到 t1（准备 WB 段存入）的新值，从 M/WB 寄存器转发到第 4 条指令的 ALU 输入端（但是来不及到达，发生 Load-use 数据冒险）。

2) 如上，第 3 和第 4 条指令之间存在 Load-use 数据冒险。

3) 假设处理器提供对转发技术和 Load-use 数据冒险的支持，第 5 周期结束时：
第 1 条指令：处于 WB 阶段，寄存器 s3 将被写入数据。
第 2 条指令：处于 M 阶段，对于 sub 指令来说，此时什么也不用做。
第 3 条指令：处于 EX 阶段，正在从 EX/M 流水段寄存器读取被转发的 t2 新值。
第 4 条指令：处于 ID 阶段，从 t1 和 t2 寄存器读取数据。考虑到检测出 Load-use 冒险，需要插入气泡来作阻塞处理，还可能正在将控制信号/指令/PC 写使能清 0。
第 5 条指令：处于 IF 阶段，正在从 PC 取指令并计算 PC + 4，但由于在此之前已经发生了 Load-use 冒险，PC 可能将不会被写入 PC + 4 的值，而是维持不变。

19. 假定有一个程序的指令序列为“lw，add，lw，add，…”。回答以下问题。

1) 每条 lw 和紧跟其后的 add 之间都发生 Load-use 冒险，则每个 lw-add 指令对中间都需要插入一个气泡（阻塞一个时钟周期），则 CPI 为 3/2 = 1.5

2) 寄存器写口和读口分别在一个时钟周期的前、后半个周期内独立工作。每两条相邻指令之间有数据依赖，都需要插入两个气泡（阻塞两个时钟周期），CPI 为 3。

20. 流水线带转发，则只需要解决 1-2、6-7 两个指令对的 Load-use 数据冒险即可。考虑从选择两条执行顺序不影响的指令，分别插入第 1 和 2、6 和 7 条指令之间，如下：

```
1   lw      s2, 100 (s6)
4   add     s6, s4, s7
2   add     s2, s2, s3
3   lw      s3, 200 (s7)
6   lw      s2, 300 (s8)
5   sub     s3, s4, s6
7   beq     s2, s8, Loop
```

将与第 2 和第 3 条指令无关的第 4 条指令插入到第 1 和第 2 条指令之间，与第 6 条指令无关的第 5 条指令插入到第 6 和第 7 条之间，则调整后的指令序列全都可以顺利进行转发，而不会出现 Load-use 数据冒险，执行时的性能达到最好。

21. 检测相减结果是否为 0 的操作在执行阶段进行，则分支延迟损失时间片（分支延迟槽）为 C = 2。考虑数据转发，lw 和 bne 指令之间发生 Load-use 数据冒险，需要阻塞 1 个时钟周期；bne 和 add 指令之间（当转移条件满足时）发生控制冒险，需要阻塞 2 个时钟周期；j 指令的阻塞时间与何时 PC 更新操作有关，若在译码/取数阶段更新 PC，则需要阻塞 1 个时钟周期，否则若在执行阶段更新 PC，需要阻塞 2 个时钟周期。

22. 通过分别计算三种实现方式下每条指令的（平均）执行时间来比较快慢。

1) 单周期方式：时钟周期取最复杂指令 lw 的指令周期，为 PC 锁存延迟 + 取指令时间 + 寄存器取数时间 + ALU 延迟 + 存储器取数时间 + 寄存器建立时间 + 时钟偏移，PC 锁存延迟和时钟偏移忽略不计，得 200 + 50 + 100 + 200 + 50 = 600ps。单周期方式下每条指令在这个时钟周期内完成，则每条指令的执行时间为 600ps

2) 多周期方式：CPI = 7 × 25% + 6 × 10% + 5 × 52% + 4 × 11% + 4 × 2% = 5.47，又时钟周期取存储操作时间的一半即 100ps，每条指令的平均执行时间为 547ps

3) 流水线方式：下面先依次分析每一类指令的 CPI。
对于取数指令，若其后第一条指令与之存在依赖关系需要阻塞两个时钟周期，若其后第一条指令不存在而第二条存在依赖关系，需要阻塞一个时钟周期，其他的情况均不需要阻塞。则 CPI = 1/2 × 3 + (1 – 1/2) × 1/4 × 2 + 3/8 × 1 = 17/8 = 2.125

* 我理解这个“Load 指令与后续各指令之间存在依赖关系的概率分别为 1/2、1/4、1/8……”中的 1/2、1/4……是各自独立的，所以这里 2 乘的是“取数指令后第一条指令不依赖而第二条依赖”的概率，3/8 是后面两条都不依赖的概率。

对于存数指令均不需要阻塞，CPI = 1；

对于 ALU 指令采用“转发”技术处理数据冒险，也不需要阻塞，CPI = 1；
对于分支指令，预测准确率为 75%，分支延迟损失片为 2，即预测失败时需要阻塞两个时钟周期，则 CPI = 75% × 1 + (1 – 75%) × 3 = 1.5；
对于跳转指令最早在译码阶段确定跳转地址，需要阻塞两个时钟周期，CPI = 3；
综上，CPI = 2.125 × 25% + 1 × 10% + 1 × 52% + 1.5 × 11% + 3 × 2% = 1.37625，又时钟周期取存储操作时间的一半即 100ps，每条指令的平均执行时间为 138ps

则三种实现方式流水线最快，多周期次之，单周期最慢，其中多周期其实只比单周期略快（10%不到），而流水线方式执行速度分别是单周期的 4.35 和多周期的 3.96 倍。

23. 分析并给出以下急着预测方案的预测准确率。

1) 静态预测，总是预测转移（taken）。预测准确率为 15 / 25 = 60%
   - 分支指令 1：预测正确 3 次，错误 0 次；
   - 分支指令 2：预测正确 0 次，错误 4 次；
   - 分支指令 3：预测正确 3 次，错误 3 次；
   - 分支指令 4：预测正确 4 次，错误 1 次；
   - 分支指令 5：预测正确 5 次，错误 2 次。

2) 静态预测，总是预测不转移（not taken）。预测准确率为 10 / 25 = 40%
   - 分支指令 1：预测正确 0 次，错误 3 次；
   - 分支指令 2：预测正确 4 次，错误 0 次；
   - 分支指令 3：预测正确 3 次，错误 3 次；
   - 分支指令 4：预测正确 1 次，错误 4 次；
   - 分支指令 5：预测正确 2 次，错误 5 次。

3) 一位动态预测，初始预测转移（taken）。预测准确率为 13 / 25 = 52%
   - 分支指令 1：预测 T – T – T，正确 3 次，错误 0 次；
   - 分支指令 2：预测 T – N – N – N，正确 3 次，错误 1 次；
   - 分支指令 3：预测 T – T – N – T – N – T，正确 1 次，错误 5 次；
   - 分支指令 4：预测 T – T – T – T – N，正确 3 次，错误 2 次；
   - 分支指令 5：预测 T – T – T – N – T – T – N，正确 3 次，错误 4 次。

4) 两位动态预测，初始预测弱转移（taken）。预测准确率为 18 / 25 = 72%
   * 以下 wT/wN (weak)表示弱转移/不转移，sT/sN (strong)表示强转移/不转移。
   - 分支指令 1：预测 wT – sT – sT，正确 3 次，错误 0 次；
   - 分支指令 2：预测 wT – wN – sN – sN，正确 3 次，错误 1 次；
   - 分支指令 3：预测 wT – sT – wT – sT – wT – ST，正确 3 次，错误 3 次；
   - 分支指令 4：预测 wT – sT – sT – sT – wT，正确 4 次，错误 1 次；
   - 分支指令 5：预测 wT – sT – sT – wT – sT – sT – wT，正确 5 次，错误 2 次。