hw8

孙启翔 241220098

June 2025

3.

(1) 8 位

(2) PC 存放当前指令地址，将 Bgt 指令译码后取出低 8 位 Imm 并进行符号拓展，加法器 A_1 计算 $PC = PC+2$，A_2 计算 $PC = PC+2+Imm8$，新的 PC 值由一个多路选择器控制，控制信号的判断设为标志寄存器 Flag 中的 $ZF + (SF \oplus OF) == 0$

4.

1. RegWr 为 0 时，无法向寄存器堆里写入任何数据，所以 R-型指令如 add,sub，lw，addi，jal 等需要写回寄存器的指令都无法正常工作
2. ALUASrc 为 0 时，ALU 的第一个操作数始终为 busA，不能执行需要 PC 作为 ALU 第一个操作数的指令，比如 beq 的跳转
3. Branch 为 0 时，同样不能正确执行 beq 指令，因为即使 Zero 信号为 1，PC 仍然加 4
4. Jump 为 0 时，无法正确执行 J-型指令，就像上一种情况，PC 的更新逻辑是错误的，无法正确跳转到目标地址
5. MemWr 为 0 时，则无法向数据寄存器中写入任何数据，此时不能执行 sw 指令
6. MemtoReg 为 0 时，写回寄存器的数据总是 ALU 的运算结果，那么 lw 指令不能执行

5.

1. RegWr 为 1 时，sw,beq,jal 等不需要向寄存器堆写入数据的指令无法工作
2. ALUASrc 为 1 时，R-型指令如 add,sub 等，lw,sw,addi,beq 等需要 busA 作为 ALU 第一个操作数的指令无法工作
3. Branch 为 1 时，R-type, lw, sw, addi, jal 指令无法工作
4. Jump 为 1 时，R-type, lw, sw, addi, beq 指令无法工作
5. MemWr 为 1 时，除 sw 外均无法工作
6. MemtoReg 为 1 时，除 lw 外均无法正确工作

6.

(1)
```
xor rs, rs, rt
xor rt, rs, rt
xor rs, rs, rt
```

(2) 设 P_{swap} 为 'swap' 操作在程序中的原始比例，T_{orig} 为原始单条指令平均执行时间。

伪指令实现 (软件): 每个 'swap' 操作由 3 条指令替代。总执行时间

$$T_{SW} = ((1 - P_{swap}) + 3P_{swap})NT_{orig} = (1 + 2P_{swap})NT_{orig}$$

硬件实现: 总执行时间

$$T_{HW} = 1.1NT_{orig}$$

为使硬件实现更优越，需满足：

$$\begin{aligned} 1.1 &< 1 + 2P_{swap} \\ 0.1 &< 2P_{swap} \\ P_{swap} &> 0.05 \end{aligned}$$

当 'swap' 指令在程序中的比例 P_{swap} 大于 5% 时，硬件实现更值得。

(3) 单周期 CPU 的寄存器堆只有一个写端口，所以无法完成 swap 指令

11.

(1) 不能，流水线 CPU 的时钟周期由最长的决定，即 200ps

(2) 没有影响，原因同上

(3) 时钟周期下降为 210ps

13.

(1) 总延迟时间为 $80+30+60+50+70+10=300ps$，若添加一个流水段寄存器，则应选择在 C,D 之间添加，这样时钟周期为 $170+20=190ps$，指令吞吐率为 $\frac{1}{190ps}$，指令执行时间为 $2\times 190ps=380ps$

(2) 方法同上，在 B,C 与 D,E 之间插入流水段寄存器，时钟周期为 130ps，指令吞吐率为 $\frac{1}{130ps}$，指令执行时间为 $3\times 130ps=390ps$

(3) 方法同上，在 AB,CD,EF 之间插入流水段寄存器，时钟周期为 140ps，指令吞吐率为 $\frac{1}{140ps}$，指令执行时间为 $4\times 130ps=560ps$

(4) 方法同上，时钟周期为 100ps，指令吞吐率为 $\frac{1}{100ps}$，指令执行时间为 $6\times 100ps=600ps$

14.

- I1 → I2: I1 写 s3 (WB)，I2 读 s3 (ID/EX)。
- I2 → I3: I2 写 t2 (WB)，I3 读 t2 (EX)。
- I3 → I4: I3 写 t1 (WB)，I4 读 t1 (ID/EX)。

无转发技术下插入 NOP

结果在 WB 阶段（第 5 周期）写回寄存器。后续指令在 ID（第 2 周期）读取或 EX（第 3 周期）使用。需要等待 3 个时钟周期。

- I1 → I2 (s3): 插入 3 条 `nop`。

- I2 → I3 (t2): 插入 3 条 `nop`。
- I3 → I4 (t1): 插入 3 条 `nop`。

总计需插入 $3+3+3=\mathbf{9}$ 条 `nop` 指令。

采用转发技术

转发允许在 EX 阶段末尾或 M 阶段末尾将结果直接传递给后续指令的 ALU 输入。

- I1 → I2 (s3): I1 在 EX 产生 s3，I2 在 EX 使用。可通过转发解决。
- I2 → I3 (t2): I2 在 EX 产生 t2，I3 在 EX 使用。可通过转发解决。
- I3 → I4 (t1): I3 在 M 才从内存获取 t1，I4 在 EX 使用。数据来不及转发。I4 需要等待 1 个时钟周期。添加 1 条 `nop`。

17.

```
lw s2, 100(s6)
lw s3, 200(s7)
add s6, s4, s7
add s2, s2, s3
sub s3, s4, s6
lw s2, 300(s8)
nop
beq s2, s8, Loop
```

18.

分支判断在 EX 阶段。

1. **分支延迟损失时间片： 2 个时钟周期**（冲刷掉 2 条指令）。
2. **流水线阻塞：**
   - `bne t0, s5, Exit`：因与前一条 `lw t0, 0(t1)` 的 Load-Use 冒险，阻塞 **1 个时钟周期**。

- 紧随 `bne` 的指令（如 ‘add s3, s3, s4‘ 和 ‘j Loop‘）：若分支跳转，因控制冒险（分支延迟），造成 **2 个时钟周期**的损失。