流水线CPU 答案

流水线原理

1. Branch prediction in a RISC-V pipelined processor improves performance by guessing the outcome of branch instructions and pre-fetching the next instructions, but it introduces a risk of performance degradation due to branch mispredictions. (T) 很对嘛，预测+取指令，预测错了变慢
2. The execution time of a single instruction in a pipelined RISC-V processor is faster than in a single-cycle processor because it moves through multiple stages more quickly. (F)单挑指令是不会变快，甚至更慢了
3. What is the minimum number of cycles needed to completely execute n instructions on a CPU with a k stage pipeline? Justify your formula.

$$\text{第一条指令：}k$$
$$\text{后}n-1\text{条}$$
$$\text{共}k+n-1$$

4. Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

```
addi x11, x12, 5
nop
nop
add  x13, x11, x12
addi x14, x11, 15
nop
add  x15, x13, x12
```

结构冒险

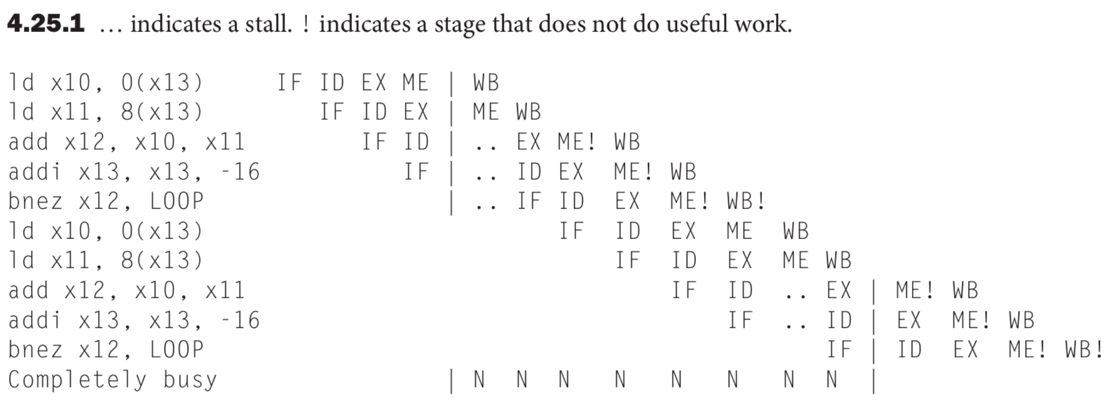
1. D 会有结构冒险，你可能无法同时取指和读取

数据冒险

1. Consider the following loop.

```
LOOP: ld x10, 0(x13)
      ld x11, 8(x13)
      add x12, x10, x11
      subi x13, x13, 16
      bnez x12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage. Show a pipeline execution diagram for the first two iterations of this loop.



4.25.1 … indicates a stall. ! indicates a stage that does not do useful work.

```
ld x10, 0(x13)     IF ID EX ME | WB
ld x11, 8(x13)        IF ID EX | ME WB
add x12, x10, x11        IF ID | .. EX ME! WB
addi x13, x13, -16          IF | .. ID EX  ME! WB
bnez x12, LOOP                 | .. IF ID  EX  ME! WB!
ld x10, 0(x13)                         IF  ID  EX  ME  WB
ld x11, 8(x13)                             IF  ID  EX  ME WB
add x12, x10, x11                              IF  ID  .. EX | ME! WB
addi x13, x13, -16                                 IF  .. ID | EX  ME! WB
bnez x12, LOOP                                            IF | ID  EX  ME! WB!
Completely busy                | N  N  N   N   N   N   N  N  |
```

2. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
ld  x13, 4(x15)
ld  x12, 0(x2)
or  x13, x15, x13
sd  x13, 0(x15)
```

   1. If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

   ```
   add           x15, x12, x11
   nop
   nop
   ld           x13, 4(x15)
   ld           x12, 0(x2)
   nop
   or           x13, x15, x13
   nop
   nop
   sd           x13, 0(x15)
   ```

   2. Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register x17 can be used to hold temporary values in your modified code.

   not possible

   12 指令间有 RAW

   13之间有WAR

   23更换改变不了结果

   24间有WAW和RAW

   34替换不会更优

45间有WAR和WAW

15间有RAW但是不影响

3. If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

   不会有问题，因为没有load-use hazard

4. If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

```
add x15, x12, x11
ld  x13, 4(x15)
ld  x12, 0(x2)
or  x13, x15, x13
sd  x13, 0(x15)
```

- ForwardA = X; ForwardB = X (no instruction in EX stage yet)
- ForwardA = X; ForwardB = X (no instruction in EX stage yet)
- ForwardA = 0; ForwardB = 0 (no forwarding; values taken from registers（此时指令 add x15, x12, x11 在 EX））
- ForwardA = 2; ForwardB = 0 (base register taken from result of previous instruction，指令 ld x13, 4(x15) 的 x15 即 src1 需要从 EX/MEM 中取出)
- ForwardA = 0; ForwardB = 0
- ForwardA = 0; ForwardB = 1 (rs1 = x15 taken from register; rs2 = x13 taken from result of 1st ld—two instructions ago)
- ForwardA = 0; ForwardB = 2 (base register taken from register fi le. Data to be written taken from previous instruction，即 x13 作为 rs2需要从or x13, x15, x13中获得)

控制冒险

1. Consider a five-stage pipeline CPU. It does not consider structural hazard and allows the register file to be read and written in the same clock cycle. Branches are resolved in the ID Stage. And consider the following code segment:

```
Loop: lw x1, 0(x4)          Instr. 1/7
lw x2, 400(x4)              Instr. 2
add x3, x1, x2              Instr. 3
sw x3, 0(x4)                Instr. 4
addi x4, x4, -4             Instr. 5
bne x0, x4, loop            Instr. 6
```

a) No forwarding mechanism is used. Show a pipeline execution diagram for the first loop iteration.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	F	D	E	M	W											
2		F	D	E	M	W										
3			F	D	D	E(load-use harzard)	M	W								
4				F	F(第三条卡住)	D	E	M	W							
5							F	F	F(第四条卡住)	D	E	M	W			
6										F	D	D	D	E(等第五条)	M	W

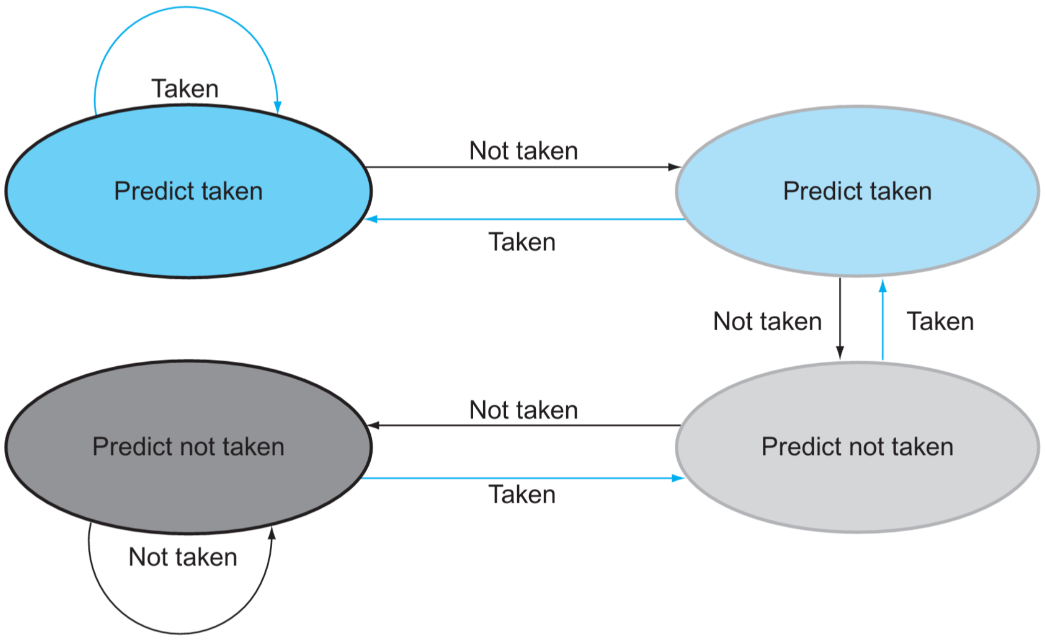
b) Forwarding mechanism is used. Show a pipeline execution diagram for the first loop iteration.
不考虑对 ID 阶段支持 forwarding 这点原题没有提到，请务必在考场问监考老师

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	F	D	E	M	W											
2		F	D	E	M	W										
3			F	D	D	E(load-use)	M	W								
4				F	F(第三条卡住)	D	E	M	W							
5					F	D	E	M	W							
6						F	D	D	D(为了计算branch)	E	M	W				
7										F	D	E	M	W		

Pipeline execution table sample:

Clock Cycle	**1**	**2**	**3**	**4**	**5**	**...**
1	F	D	E	M	W	
2		F				

2. This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT.
    1. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?
        - always taken 60%
        - always not-taken 40%
    2. What is the accuracy of the 2-bit predictor for the first four branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.61 (predict not taken)?



左下--右下--左下--右下--右上，仅第二个正确 25%

冒险

1.