Oct 27th, 2023

SYS2-Fall23: Computer Systems II
Homework 2: Data and Control Hazards
Due Time: Nov 11th, 2023

Note:
1. In this homework, We all assume that the ***register write*** *is done in* ***the first half of the clock cycle*** *and that* ***register reads*** *are done in* ***the second half of the cycle*** *in the five-stage pipeline design.*
2. The type of ***RAW data dependence*** *is identified by the stage that produces the result (EX or MEM) and the next instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both).*
3. ***Bimodal Predictor****: The simplest dynamic branch direction predictor is an array of* 2^n *two-bit saturating counters. Each counter includes one of four values:* ***strongly taken (T), weakly taken (t), weakly not taken (n), and strongly not taken (N)****. The state diagram of the counter is shown in Figure 1.*
Prediction. To make a prediction, the predictor selects a counter from the table using using the lower-order n bits of the instruction's address (its program counter value). The direction prediction is made based on the value of the counter.
Training. After each branch (correctly predicted or not), the hardware increments or decrements the corresponding counter to bias the counter toward the actual branch outcome (the outcome given in the trace file). As these are two bit saturating counters, decrementing a minimum counter or incrementing a maxed out counter should have no impact.

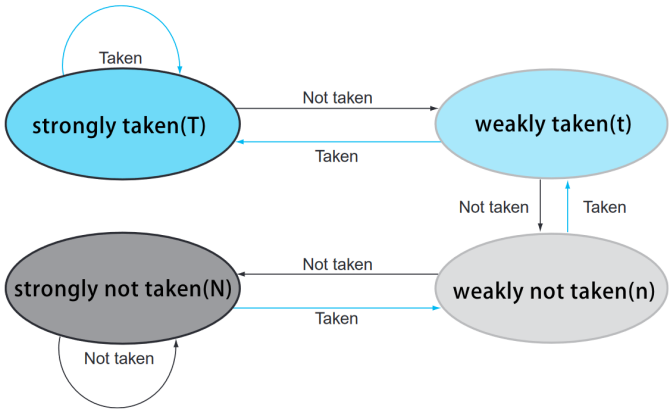

Figure 1: The states in a 2-bit prediction scheme.

Question 1: Pipeline Design Concept (35 points)

The following figures show the execution diagram of the single-cycle and pipeline.

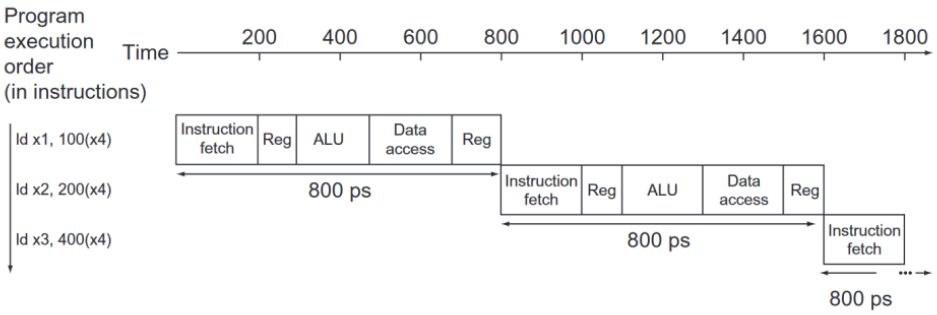

Figure 2: nonpipelined exectuion progress.

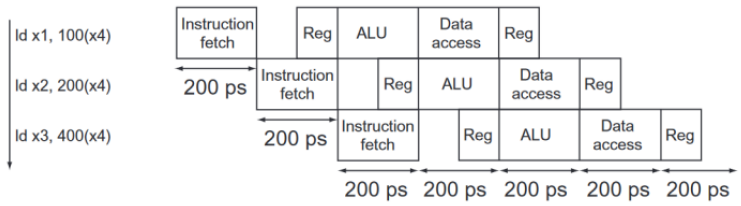

Figure 3: pipelined exectuion progress.

(a) Answer the following choice questions about pipelined processor design. **(2 points each)**

- [ ]. Pipelining improves the _____ of instructions.
  A.throughput B.inherent execution time C.instruction latency
- [ ]. A single-cycle implementation violates the great idea of **making the common case fast**.
  A.True B.False
- [ ]. A longer instruction pipeline is usually associated with higher clock speeds.
  A.True B.False
- [ ]. Deeper pipelines with more stages can degrade execution performance.
  A.True B.False
- [ ]. If the instruction **add x4, x1, x0** is inserted **after ld x1, 100(x4)**, then the last instruction in pipeline starts execution at time _____ . (all forwarding is implemented)
  A.600ps B.800ps C.1000ps

(b) Assume that the critical path in *Instruction fetch* takes 100ps, *Reading* ***or*** *Writing Registers* takes 100ps, *ALU Computing* takes 30ps, *Data Access* taks 500ps. Compute the minimum clock period achievable for the single-cycle and the pipelined design. How much speedup do you achieve over the single-cycle design?**(10 points)**

(c) Based on the critical path in (b), if we want to turn the five-stage into four-stage, which two stages will you merget together to achieve the best performance ? Explain it and compute the speedup again.(you can use stage name IF/ID/EX/MEM/WB) **(15 points)**

Question 2: Data Hazard and Forwarding (35 points)

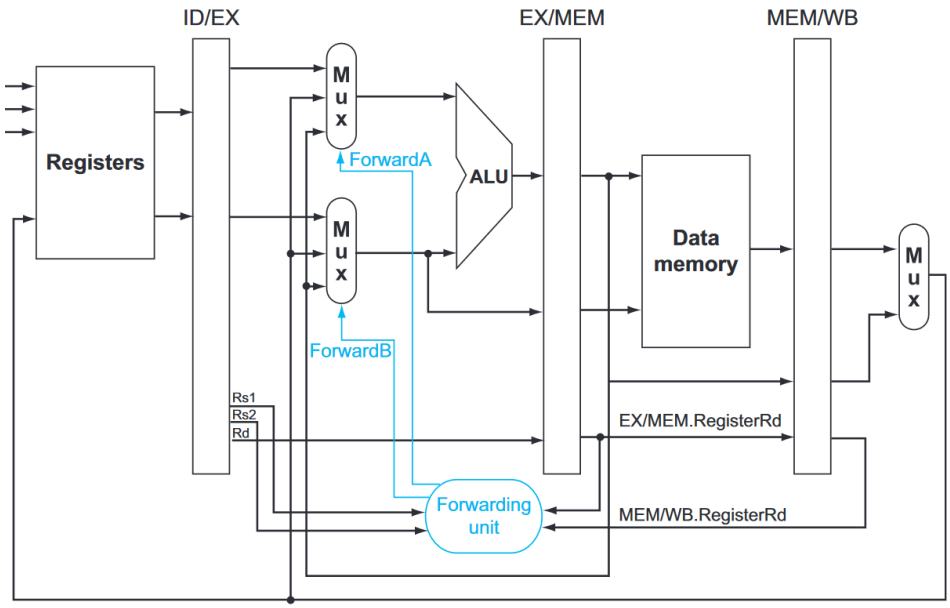

Figure 4: 5-stage pipeline datapath with forwarding logic implemented.

(a) For the following sequence of instructions, insert NOPs to ensure correct execution if there is no forwarding and hazard detection. Change and rearrange the origin code to minimize the number of NOPs needed and give its space time diagram .(Assume register x17 can be used as temporary register)**(15 points)**

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

(b) Only considering the **RAW(read after write)**(see more in Note 2) data hazard, we have the count of forwarding instructions as Table 1. Under different situations, the execution time of different stages are shown as Table 2. What percent of cycles are stalls when running on a CPU with full forwarding? Explain why Why "MEM to 1st and MEM to 2nd" is not an entry in Table 1.(assume there are no any control transfer instructions)**(10points)**

EX to 1st only	MEM to 1st only	EX to 2nd only	MEM to 2nd only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

Table 1: fowarding instructions partition.

IF	ID	EX(no FW)	EX(full FW)	EX(FW from EX/MEM)	EX(FW from MEM/WB)	MEM	WB
200ps	100ps	110ps	130ps	120ps	120ps	200ps	100ps

Table 2: Execution time of different stage.

(c) Based on the two tables above, note the two **three-input** forwarding Mux in EX stage before the ALU and assume we only have the **two-input** multiplexors. Whether it's better to forward only from the EX/MEM pipeline register or only from the MEM/WB pipeline register? Compute their CPI resectively.**(10 points)**

Question 3: Control Hazard and Branch Prediction (30 points)

In this question, we explore the static and dynamic branch prediction in the pipeline design.

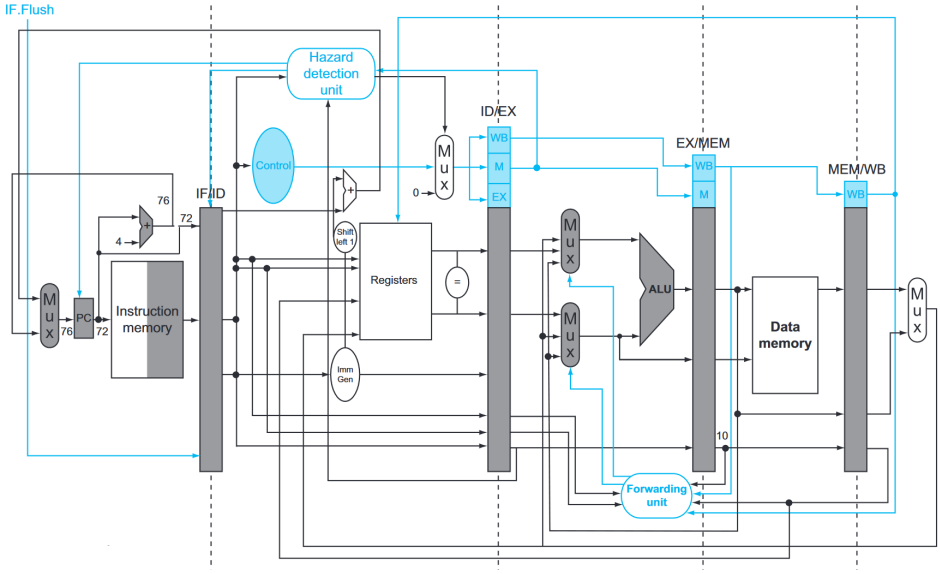

Figure 5: 5-stage pipeline datapath with forwarding logic implemented.

(a) For all conditional branch instructions, we use static prediction strategies of **always-taken and always-not-taken** respectively. Which of these two policies is more accurate for the following code in C (**translate the code into RV32I** first and assume the **array A** has already been written in memory from **address 0x800000** and the **variable** a is saved in register **t0**)? **(10 points)**

```
int A[6] = {3, 7, 4, 9, 2, 1};
int a = 0;
for (i=0; i<6; i++){
    if (A[i]<5)
        a = a+1;
    else
        a = a+2;
}
```

(b) The most classical implementation of the dynamic prediction is the Bimodal Predictor(see Note3 for more). It uses a branch history table to record previous branch taken states(counter) for different entries(usually mapped from lower n bit of the instruction's address). So assume we have a branch history table with **8 entries** and the starting address of your code is **0x0**, **fill in the blank of the history table(Table 3)** initialized by 0. **Compare the performance with static branch prediction in (a)**. If better, explain it. If worse, suggest some method of improvement under the dynamic prediction. **(10 points)**

Note: Iteration is indicated by variable i. In each iteration the counter may change for many times, just write it like 00→01→10.

(c) In Figure 5, we move the branch test to ID stage with two main changes in the logic circuit: an adder in ID stage as one choice of PC, an equal test after the registers. However, if we move the equal test ahead in ID stage, it will raise extra data hazards. Assume we try to deal this by forwarding, please design the circuit again(add a forwarding unit to ID stage and its corresponding lines). **(4 points)**

Note: 3 type of circumstances should be considered, please fill the Table 4. If the source operand is no need to be forwarded, leave the cell blank, otherwise, write the stage it is forwarded from. **(6 points)**

counter entry \ iteration(i)	0	1	2	3	4	5	6
0	00						
1	00						
2	00						
3	00						
4	00						
5	00						
6	00						
7	00						

Table 3: Change of branch history table for dynamic prediction.

Instructions	stall cycles	rs1 forward stage	rs2 forward stage

addi x1, x0, 1
addi x2, x0, 2
beq x1, x2, 0x10			

addi x1, x0, 1
ld x2, 8(x1)
addi x3, x2, 1
beq x2, x3, 0x1c			

ld x2, 8(x0)
ld x3, 24(x2)
beq x2, x3, 0x20			

addi x1, x0, 1
addi x2, x0, 1
beq x0, x1, 0x10			

Table 4: execution cycles of instruction sequences under forwarding implementation with branch test in ID stage