Week 2 Problem Set

Course Introduction and Instruction Set Architecture

Contents

Reference: Algorithms

Min value

Consider the high-level code below that finds the minimum value in an array arr of length n.

```
int min = arr[0]

for(int i = 1; i < n; i++) {

      if(arr[i] < min) {

            min = arr[i];

      }

}
```

Assume that the base address of `arr` is stored in r5 and that the address of the last element in the array is stored in r4 (you can also assume the array is non-empty).

Compiling the code for a load-store architecture, where r1 contains min, is shown below:

```
LW R1, 0(R5)
ADDI R5, R5, 4
SEARCH:
      LW R2, 0(R5)
      BGT R2, R1, CONTINUE
      ADD R1, R0, R2
CONTINUE:
      ADD R5, R5, 4
      BNE R5, R4, SEARCH
```

Problems: Pipelined Architectures

This problem is adapted from a question in: *Dubois, Michel, Murali Annavaram, and Per Stenström. Parallel computer organization and design. cambridge university press, 2012.*

In this question, we compare the compiled code and performance of a small program across different pipeline processors that use a load-store architecture.

Every pipelined processor in this question uses a classic 5-stage pipeline design with static branch prediction. The pipeline assumes that branches are not taken and checks whether that is the case in the execute stage of a branch instruction. If the prediction was incorrect, the pipeline is flushed.

Compilation: With and without hazard detection

Your friend, Mario, would like to run the code for the min value algorithm on a processor he has just designed. However, he has not yet included a hazard detection unit. Modify the assembly code so that it runs on Mario's pipelined processor.

Stall cycles

Mario has finally added a hazard detection unit that stalls instructions in the decode stage to avoid data hazards. In this question, we compare the number of cycles needed to execute to different kinds of iterations: iterations where min is re-assigned, and iterations where min is not re-assigned.

Complete the table below by indicating how many cycles it takes to execute an instruction, including cycles spent stalling, on an iteration where min is re-assigned (the if-statement is **taken**). Assume that there are elements of the array to search.

Instruction	Latency including stalls (cycles)
`LW R1, 0(R5)`	1
`ADDI R5, R5, 4`	
`SEARCH:`	N/A
`    LW R2, 0(R5)`	
`    BGT R2, R1, CONTINUE`	
`    ADD R1, R0, R2`	
`CONTINUE:`	N/A
`    ADD R5, R5, 4`	
`    BNE R5, R4, SEARCH`	

What is the total number of cycles to execute one iteration?

Complete the table below by indicating how many cycles it takes to execute an instruction, including cycles spent stalling, on an iteration where min is re-assigned (the if-statement is **not taken**). Assume that there are elements of the array to search.

Instruction	Latency including stalls (cycles)
`LW R1, 0(R5)`	1
`ADDI R5, R5, 4`	
`SEARCH:`	N/A
`    LW R2, 0(R5)`	
`    BGT R2, R1, CONTINUE`	
`    ADD R1, R0, R2`	
`CONTINUE:`	N/A
`    ADD R5, R5, 4`	
`    BNE R5, R4, SEARCH`	

What is the total number of cycles to execute one iteration?

Some bypassing

Mario has implemented some bypassing so that values from the M/W pipeline registers can be used as inputs to the ALU. How would the stall cycles from the question "Stall cycles" be impacted for both types of iterations?

Full bypassing

Mario has implemented full bypassing, so that the values from M/W and E/M pipeline registers can be used as inputs to the ALU. How would the stall cycles from the question "Stall cycles" be impacted for both types of iterations?

Variations of this problem

Consider other algorithms that are common in everyday programming, like checking whether a value exists in an array. We also highly recommend sketching pipeline diagrams for each of the problems below (see template below).

Appendix: Pipeline diagram

Use a pipeline diagram to show how a sequence of instructions moves through each stage of the pipeline. Each row indicates an instruction in the sequence of instructions being executed (e.g., instruction 1 is labelled I1). Each column corresponds to a specific cycle. In the classic 5-stage pipeline, no two instructions can be in the same stage in the same cycle.

Remember to indicate stalls with an asterisk suffix (e.g., a stall in decode appears as D*). When register forwarding (also called bypassing) is enabled, remember to show when those are used by drawing an "L-shaped" arrow from one stage to another (see lecture).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1																					
I2																					
I3																					
I4																					
I5																					
I6																					
I7																					
I8																					
I9																					

For example, for a sequence of two independent instructions:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1	F	D	E	M	W																
I2		F	D	E	M	W															