Computer Architecture - HW 6

Connor Finley

November 4, 2017

## 1

Describe the effect that a single stuck-at-0 fault (i.e., the signal is always 0 regardless of what it should be) would have for the signals shown below, in the single-cycle Datapath. Which instructions, if any, will not work correctly? Explain why. Consider each of the following faults separately:

### 1.a

RegWrite = 0

Nothing can be written to registers if RegWrite=0. This will render R-type instructions that handle arithmetic or loading useless. Storing or branching can still function, though.

### 1.b

RegDst = 0

This means that R-type instructions will not have the right destination register to write to and won’t work.

### 1.c

ALUSrc = 0

This means that ALU won’t be able to receive the correct sign-extended bits which means I-type instructions won’t work. Branching still works, though.

### 1.d

MemtoReg = 0

Loading from memory to registers function correctly.

### 1.e

Branch = 0

Branching will flip in terms of logic. Branches will happen if the conditions are false, but not true. This essentially flips the expected result.

## 2

Repeat question 1 but this time consider stuck-at-1 faults (the signal is always 1)

### 2.a

RegWrite = 1

Registers can and will be written to for all instructions. Contrary to RegWrite=0, storing and branching will not work, but other R-type instructions will.

### 2.b

RegDst = 1

Loading will have an incorrect destination.

### 2.c

ALUSrc = 1

The ALU won’t receive the register read for the second register which means R-type instructions won’t function correctly.

### 2.d

MemtoReg = 1

R-type instructions will input bad info into registers.

### 2.e

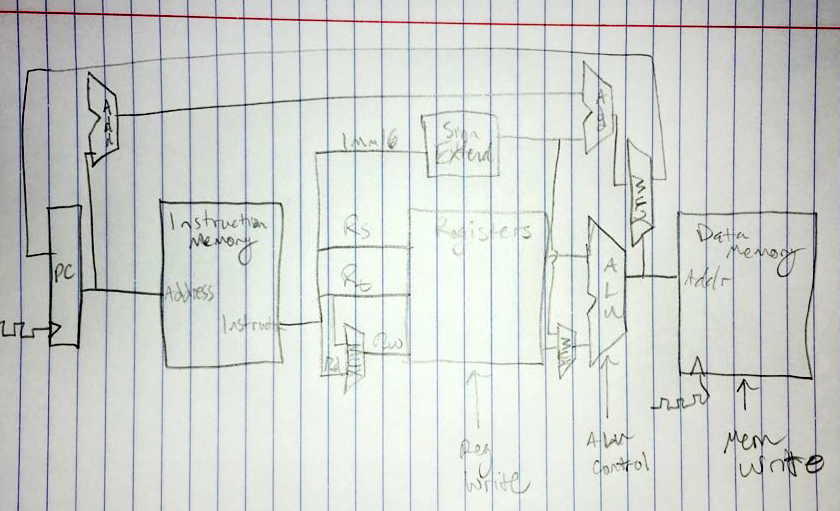
Branch = 1

If the ALU outputs 0, the PC will get an incorrect address. Contrary to Branch=0, branching will function expectedly here.

## 3

We wish to add the instruction jalr (jump and link register) to the single-cycle datapath. Add any necessary datapath and control signals and draw the result datapath. Show the values of the control signals to control the execution of the jalr instruction. The jump and link register instruction is described below: jalr rd, rs # rd = pc + 4 , pc = rs





This is like an R-type instruction, where Rd = 1, RegWrite = 1, Rt = 0, and there is no memory reading or writing.

## 4

Suppose we add the multiply and divide instructions. The operation times are as follows:

* Instruction memory access time = 190 ps,
* Data memory access time = 190 ps,
* Register file read access time = 150 ps,
* Register file write access = 150 ps
* ALU delay for basic instructions = 190 ps,
* ALU delay for multiply or divide = 550 ps

Ignore the other delays in the multiplexers, control unit, sign-extension, etc.

Assume the following instruction mix:

* 30% ALU
* 15% multiply & divide
* 20% load
* 10% store
* 15% branch
* 10% jump.

### 4.a

What is the total delay for each instruction class and the clock cycle for the single-cycle CPU design?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Class | Instruction Memory | Data Memory | Register read | Register write | ALU (basic) | ALU (Mult./Div.) | Total |
| ALU | 190 ps |  | 150 ps | 150 ps | 190 ps |  | 680 ps |
| Mult. & Div. | 190 ps |  | 150 ps | 150 ps |  | 550 ps | 1040 ps |
| Load | 190 ps | 190 ps | 150 ps | 150 ps | 190 ps |  | 870 ps |
| Store | 190 ps | 190 ps | 150 ps |  | 190 ps |  | 720 ps |
| Branch | 190 ps |  | 150 ps |  | 190 ps |  | 530 ps |
| Jump | 190 ps |  | 150 ps |  |  |  | 340 ps |

“the clock cycle is equal to the worst-case delay for all instructions” - chapter 4, page 272

**Worst case: 1040 ps = clock cycle**

### 4.b

Assume we fix the clock cycle to 200 ps for a multi-cycle CPU, what is the CPI for each instruction class and the speedup over a fixed-length clock cycle?

* ALU: 680 ps / 200 ps = 3.4 => 4 cycles per instruction (rounded up)
* Mult. & Div.: 1040 / 200 = 5.2 => 6 cycles per instruction
* Load: 870 ps / 200 ps = 4.35 => 5 cycles per instruction
* Store: 720 ps / 200 ps = 3.6 => 4 cycles per instruction
* Branch: 530 ps / 200 ps = 2.65 => 3 cycles per instruction
* Jump: 340 ps / 200 ps = 1.7 => 2 cycles per instruction

CPI (avg.): (0.3 \* 4) + (0.15 \* 6) + (0.2 \* 5) + (0.1 \* 4) + (0.15 \* 3) + (0.1 \* 2) = 4.15 CPI

Speedup:

* Speedup = CPU time A / CPU time B
* CPU time = Instruction count \* CPI \* Clock cycle time (chapter 1, page 36)
* Instruction count can be left out since they are the same in both parts

Part A:

* CPI = 1
* Part A was a single-cycle CPU, meaning one clock cycle per instruction
* Clock cycle time = 1040 ps
* CPU time = 1 \* 1040 ps = 1040 ps

Part B:

* CPI = 4.15 (average)
* Clock cycle time: 200 ps
* CPU time = 4.15 \* 200 ps = 830 ps

**Speedup: 1040 ps / 830 ps = 1.25**