Dylan Faulhaber Homework 3

I pledge my honor that I have abided by the Stevens Honor System

Question 1

1.1) If the Br wire was stuck at 0, the branching instructions besides the RET instruction would not work correctly. This would include the instructions: B, BL, CBZ, CBNZ. The CPU would always have the next instruction set to PC+4 unless the RET instruction is called instead, then the branching would work.

1.2) If the ALUsrc wire was stuck at 0, then the instructions that have immediate numbers as one of their inputs would fail to work correctly. This is because the ALUsrc signal tells the processor if it should read inputB as the data from Reg 2 or the 11-bit immediate. If this wire is always stuck at 0, then it will always read the data from Reg 2. Therefore, these instructions would include: ADD, ADDS, SUB, SUBS, LDR, STR, AND, ANDS, ORR. Only when the instruction takes in an immediate then it will be affected.

1.3) If the RegWrite wire were to be stuck at 0, then any instruction that writes back to a register would fail to work. This is essentially every instruction, even branching ones. The reason for this is because it would be impossible to save any data into any registers if the RegWrite wire is set to 0, since it must be 1 for the sequential circuit to save any data. Therefore, these instructions would include: BL, RET, ADD, ADDS, SUB, SUBS, LDR, AND, ANDS, ORR.

Question 2

In order to have a SWAP instruction that swaps the data between the two registers, then the main two changes would have to be another WriteReg port and another RegDataW port. The reason for this is because during the WB stage, this would allow the data loaded into RegData1 and RegData2 to be loaded into opposite registers that they came from. This can happen by having each WriteReg port be a copy of the original registers the data came from. Also, each RegDataW port would take in the data from the two registers. Lastly from here, the data can be swapped and saved into opposite registers.

Question 3

3.1) Old time = 200ps + 250ps + 150ps + 300ps + 200ps = 1,100 ps

New time = 200ps + 250ps + 350ps + 300ps + 200ps = 1,300 ps

3.2) speedup = 1 – newTime/oldTime = 1 – (1300 * .8)/1100 = 1 – .945 = .0545 = 5.45%

The speedup of this new improvement would be by 5.45%. The reason for this is because even though the overall time of the clock cycle is longer (1,300 > 1,100), when we decrease it by 20% it becomes shorter by 5.45% compared to the original time.

3.3) To find the slowest the ALU can be we must find the time where there is no speedup.

1 – (x * .8)/1,100 = 0

1 = (x * .8)/1,100

1,100 = x * .8

1375 = x

Therefore, the total time of the newTime must be less than 1375 picoseconds. To isolate the ALU we add the time of all other stages together and subtract that from the total.

200ps + 250ps + 300ps + 200ps = 950

1375 – 950 = 425 picoseconds.

Therefore we can say the time of the ALU (and the entire EX stage in general) must be less than 425 picoseconds for this improvement to work.

Question 4

4.1) In a non-pipelined processor, the length of the clock cycle would be 1,100 picoseconds because this is all the run times of each stage added together. In a pipelined processor, the clock cycle time would be 300 ps. This is because the longest of the clock cycles is 300 picoseconds, which means that every cycle must be this length to avoid overwriting data.

4.2) The total latency of an LDR instruction in a non-pipelined processor would be 1,100 picoseconds. This is the addition of each stage's run time. In a pipelined processor, the time would be 1,500 picoseconds since this is 5 * 300 = 1,500. This is the clock cycle of 300 picoseconds multiplied by 5 for each stage of the instruction.

4.3) If I were able to split one stage of the pipelined processor into two separate ones half the time, it would be best to split the ME stage into two stages, each 150 picoseconds long. This is because it is the longest stage and by splitting it into two, the new clock cycle of the pipelined processor would be reduced to 250 picoseconds (the ID stage). This is slightly quicker than the previous 300 picosecond clock cycle.

Question 5

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
LDR X20, [X19, 0]	IF	ID	EX	ME	WB					
LDR X21, [X19, 8]		IF	ID	EX	ME	WB				
Bubble for X20					EX	ME	WB			
ADD X22, X21, X20			IF	ID	ID	EX	ME	WB		
SUB X23, X23, X22				IF	IF	ID	EX	ME	WB	

There are three data hazards that occur in this example and one can be fixed with a bubble while the other two are fixed with forwarding.

For the first one, the ADD instruction has a hazard with both the X21 and X22 registers. Therefore, to fix this we have to create a bubble for the ID stage for the ADD instruction. This allows us to stall the stage and acquire the correct value for X20. Then for the second we can forward the data from the WB stage of the second LDR instruction to get the correct value of X21. From here we can continue the program execution.

Lastly, for the final hazard in the final SUB instruction, there is a hazard with the X22 register. For this we forward it from the ME stage in the previous ADD instruction in order to get the correct value.

Question 6

6.1)

LDR X1, [X6, 8]

NOP

NOP

ADD X0, X1, X0

NOP

NOP

STR X0, [X10, 4]

LDR X2, [X6, 12]

NOP

NOP

SUB X3, X0, X2

```
**NOP**
**NOP**
STR X3, [X8, 24]
CBZ X2, 40
```

6.2)

```
LDR X1, [X6, 8]
LDR X2, [X6, 12]
NOP
ADD X0, X1, X0
NOP
NOP
SUB X3, X0, X2
NOP
STR X0, [X10, 4]
STR X3, [X8, 24]
CBZ X2, 40
```