**Assignment -3 on Pipelining (Due October 21 by midnight)**

**(65 points)**

Q1) [10 points] Forwarding and Bypassing

Consider the following assembly language code:

I0: ADD R4 = R1 + R0;

I1: SUB R9 = R3 - R4;

I2: ADD R4 = R5 + R6;

I3: LDW R2 = MEM[R3 + 100];

I4: LDW R2 = MEM[R2 + 0];

I5: STW MEM[R4 + 100] = R2;

I6: AND R2 = R2 & R1;

I7: BEQ R9 == R1, Target;

I8: AND R9 = R9 & R1;

Consider a pipeline with forwarding, hazard detection, and 1 delay slot for branches. The  pipeline is the typical 5-stage IF, ID, EX, MEM, WB MIPS design. For the above code, complete the pipeline diagram below (instructions on the left, cycles on top) for the code. Insert the characters IF, ID, EX, MEM, WB for each instruction in the boxes. Assume that there two levels of bypassing, that the second half of the decode stage performs a read of source registers, and that the first half of the write-back stage writes to the register file.

Label all data stalls (write an “s” in the box). Label all data forwards that the forwarding unit detects (arrow between the stages handing off the data and the stages receiving the data). What is the final execution time of the code?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C18 |
| I0 | IF | ID | EX | MEM | WB |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I1 |  | IF | ID | EX | MEM | WB |  |  |  |  |  |  |  |  |  |  |  |  |
| I2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| I8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Q2.** [8 points] Dependence detection

This question covers your understanding of dependences between instructions. Using the code below, list all of the dependence types (RAW, WAR, WAW). List the dependences in the respective table (example INST-X to INST-Y on variable\_name) by writing in the instruction numbers involved with the dependence.

I0: A = B + C;

I1: C = A - B;

I2: D = A + C;

I3: A = B \* C \* D;

I4: C = F / D;

I5: F = A ˆ G;

I6: G = F + D;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RAW Dependence** | | **WAR Dependence** | | **WAW Dependence** | |
| **From Instr** | **To Instr** | **From Instr** | **To Instr** | **From Instr** | **To Instr** |
| I0 | I1 on A | I0 | I1 on C | I0 | I3 on A |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Q3.** [12 points] Pipelining and Bypass. In this question we will explore how bypassing affects program execution performance. To begin consider the standard MIPS 5 stage pipeline. For your reference, refer to the figure below. For this question, we will use the following code to evaluate the pipeline’s performance:

1 add $t2, $s1, $sp

2 lw $t1, $t1, 0

3 addi $t2, $t1, 7

4 add $t1, $s2, $sp

5 lw $t1, $t1, 0

6 addi $t1, $t1, 9

7 sub $t1, $t1, $t2

(a) How many stall cycles are required between the load instruction and its dependent instruction for the standard MIPS 5-stage pipeline?

(b) Once again, using the standard MIPS pipeline, identify whether the value for each register operand is coming from the bypass, immediate field or from the register file. For clarity, please write REG or BYPASS or IMM in each box.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Src Operand One** | **Src operand two** |
| 1 |  |  |
| 2 |  |  |
| 3 | BYPASS | IMM |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |

(c) How many cycles will the program take to execute on the standard MIPS pipeline?

Q4) (10 points) Q 4.8 of HSI book

Parts 4.8.1 to 4.8.5 from the HSI book,. Fill the following tables for parts 4.8.1 to 4.8.5

4.8.1)

|  |  |  |
| --- | --- | --- |
|  | **Pipelined** | **Single Cycle** |
| Clock cycle time |  |  |

4.8.2)

|  |  |  |
| --- | --- | --- |
|  | **Pipelined** | **Single Cycle** |
| LW instruction latency |  |  |

4.8.3)

|  |  |
| --- | --- |
| **Stage to split** | **New clock cycle time** |
|  |  |

Please solve this modified version of 4.8.6,.. ignore 4.8.6 from book

4.8.6) Instead of the single cycle organization, consider the non pipelined multicycle organization where any instruction only goes through the stages it really needs. Fill up the following table with cycle count for this instruction mix. **Compute the average CPI** of multicycle architecture if 25% instructions are load, 5% are jumps, 10% are branch, 20% are stores and 40% are r-type instructions.

|  |  |
| --- | --- |
|  | **Number of cycles** |
| LW instruction |  |
| SW instruction |  |
| Branch instruction |  |
| R-type instruction |  |
| Jump instruction |  |

**Q 5) (15 points) 4.10 from the HSI book.**

4.10.1) (5 points) Please fill the following pipeline diagram for the instruction sequence in this problem with pipeline stages and clock cycle numbers and stalls by “s”,.. please add columns on the right if your diagram does not fit in these. Assume stall on branch delays and branch outcome computation in EXE stage

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | C1 | C2 | C3 | C4 | C5 | … |  |  |  |  |  |  |
| SW R16,12(R60) | IF | ID | EX | MEM | WB |  |  |  |  |  |  |  |
| LW R16,8(R6) |  |  |  |  |  |  |  |  |  |  |  |  |
| BEQ R5,R4,Lbl |  |  |  |  |  |  |  |  |  |  |  |  |
| ADD R5,R1,R4 |  |  |  |  |  |  |  |  |  |  |  |  |
| SLT R5,R15,R4 |  |  |  |  |  |  |  |  |  |  |  |  |

4.10.2) (2 points) What is the consequence of this change?

Fill the following table

|  |  |  |  |
| --- | --- | --- | --- |
| Total No of instructions in the sequence | Total no. of cycles for program execution with 5 stages | Total no. of cycles for program execution with 5 stages | Speed up |
|  |  |  |  |

4.10.3) (2 points) What is the consequence of this change?

Fill the following table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total instructions executed | Branches executed | Total no. of cycles with branch computation in EXE | Total no. of cycles with branch computation in ID stage | Speed up |
|  |  |  |  |  |

4.10.4) (2 points) use the total number of cycles required for execution from 4.10.2

|  |  |  |
| --- | --- | --- |
| Clk cycle time with 5 stages | Clk cycle time with 4 stages | Speedup (ratio of execution times) |
|  |  |  |

4.10.5) (2 points) what is the consequence of this change?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| New ID stage latency | New EX stage latency | New cycle time | Old cycle time | speedup |
|  |  |  |  |  |

4.10.6) (2 points) what is the consequence of this change on clock cycle time and execution time?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cycles with branch in EXE | Execution time with branch in EXE | cycles with branch in MEM | Execution time with branch in MEM | speedup |
|  |  |  |  |  |

Q6) How will the compiler optimize the following code to fill up the branch delay slot (assume branch computation in decode stage therefore single delay slot and data forwarding in the following problem)

Part a) (2 points) Consider the following MIPS code that operates on a linked list. Is it possible to reorder this code to fill up the delay slot without affecting program correctness? If yes reorder the code and show the pipeline table. Add rows or columns is required

Loop: lw r1, 4(r2) #R1 = r2->value

add r3, r3, r1 #sum = sum +r2->value

lw r2, 8(r2) #r2 = r2->next

bne r2, r0, Loop #while (r2!=null) keep looping

sw r3,12(r5) #r5->value=sum

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Part b) (2 points) Assume that the above loop runs for 10 times then what is the total number of cycles required for the optimized solution in part a)?

Part c) (2 points) Assume that instead of delayed branching, the BTFN (backward taken, forward not taken prediction) is used, then how many times will an instruction be misfetched. So what will be the total number of clock cycles required for code execution?

Part d) (2 points) Assume that instead of delayed branching, the *always not taken* branch prediction is applied, then how many times will an instruction be misfetched. What will be the total number of clock cycles required for code execution?

Part e) (2 points) In the following code which instruction is a good candidate for inserting in the delay slot due to the 1st branch instruction? Assume single delay slot and mention whether the inserted instruction comes from code before the branch instruction/ code in the branch target location / or code in the fall-through section

ADD R6, R0, R0

Again: BEQ R5, R6, End

ADD R10, R5, R1

LW R11, 0(R11)

LW R10, 4(R10)

End: SUB R10, R11, R10

ADD R10, R5, R2

SW R10, 0(R11)

ADDI R5, R5, 2

BEQ R0, R5, Again