John Gangemi
u68714612
CDA 4205
10/28/2014

HOMEWORK #6

4.8

4.8.1

The clock cycle time for a pipelined processor must be long enough to accommodate the slowest operation or in this case the slowest stage, ID. Therefore the clock cycle time is 350ps.

The clock cycle time for a non-pipelined processor (single-cycle) is given by the total delay accumulated within an operation or in this case the total delay from all five stages. Therefore the clock cycle time is…

250ps + 350ps + 150ps + 300ps + 200ps = 1250ps.

4.8.2

The total latency of a load word instruction in a pipelined processor is a formulation of the slowest stage latency multiplied by the number of stages in the pipeline. For the load word instruction the slowest stage latency is 350ps. Therefore the total latency is…

350ps * 5 stages = 1750ps

The total latency of a load word instruction in a non-pipelined (single-cycle) processor is the summation of stage delays.

250ps + 350ps + 150ps + 300ps + 200ps = 1250ps

4.8.3

If it were possible to split one of the five stages in the pipelined datapath into two new stages, each with half the latency of the original stage, then it would be most prudent to split the slowest stage ID. After splitting stage ID, latency for the two new stages will become 175ps, thus the new slowest stage in the pipeline will be MEM. Therefore the clock cycle time of the new pipeline is 300ps.

4.8.4

If there are no stalls or hazards, the utilization of data memory is directly related to the instructions that access data memory, LW and SW. Combining their respective execution times…

20% (LW) + 15% (SW) = 35% data memory utilization

4.8.5

If there are no stalls or hazards, the utilization of the write-register port of the Registers Unit is directly related to the instructions that use a destination register ($rd) such as ALU and LW. Combining their respective execution times…

45% (ALU) + 20% (LW) = 65% write-register port utilization

4.8.6
The clock cycle time for a single-cycle organization is given by the accumulation of stage delays, the clock cycle time for a pipelined organization is the delay of the slowest stage, the clock cycle time for a multi-cycle organization is found to be the summation of stage delays required for a specific instruction (according to the description of multi-cycle in the question).

4.9

instrA == OR R1, R2, R3
instrB == OR R2, R1, R4
instrC == OR R1, R1, R2

4.9.1
Data dependences and their type:
Reading R1 in instrB and instrC while not yet writing R1 in instrA
Reading R2 in instrC while not yet writing R2 in instrB
Writing R1 in instrC while not yet writing R1 in instrA

4.9.2
If there is no forwarding in this pipelined processor, then incurring hazards stem from reading a register in a current instruction that depended on that register being written too in the preceding instruction.

The result from instrA is not written to register R1 until the fifth stage of the pipeline therefore by the third clock cycle register R1 in instrB is waiting to be read. The same is true for register R2 in instrB and instrC.

In order to eliminate such hazards, the *nop* instruction must be used to stall instructions in the pipeline, approximately by two clock cycles for each hazard. If the register write happens in the first half of the clock cycle and the register read follows in the second half then the reorder code is as follows…

OR R1, R2, R3 *[clk cyc 1-5]*
NOP *[clk cyc 2]*
NOP *[clk cyc 3]*
OR R2, R1, R4 *[clk cyc 4-8]*
NOP *[clk cyc 5]*
NOP *[clk cyc 6]*
OR R1, R1, R2 *[clk cyc 7-11]*

4.9.3
If there is full forwarding in this pipelined processor, then incurring hazards are none existent and modifying the code with *nop* instructions is not necessary. This is possible due to the ALU's ability to 'forward' data to the execution stage of the next instruction.

4.9.4
Number of cycles = k stages + (n instructions - 1)
Without forwarding:

total execution time (Ta) = clock cycle time * number of cycles
= 250ps * 11cycles
= 2750ps

With full forwarding:

total execution time (Tb) = clock cycle time * number of cycles
= 300ps * 7cycles
= 2100ps

Speedup:

Ta / Tb = 2750ps / 2100ps = 1.31, a big improvement with full forwarding

4.9.5
It is not necessary to add *nop* instructions to the code because the hazards that were present with no forwarding (ex 4.9.2) are eliminated. Register R1 in instrA will be forwarded from stage EX to instrB Ex and by the time instrC needs R1, instrA will have finished its last stage (written to register R1). Also, register R2 in instrB will be forwarded from stage EX to instrC EX.

4.9.6
Number of cycles = k stages + (n instructions - 1)
Without forwarding:

total execution time (Ta) = clock cycle time * number of cycles
= 250ps * 11cycles
= 2750ps

With ALU-ALU forwarding (Tc):

total execution time = clock cycle time * number of cycles
= 290ps * 7cycles
= 2030ps

Speedup:

Tc / Ta = 2750ps / 2030ps = 1.35, better than full forwarding

4.10

4.10.1

For the given instruction sequence the structural hazard is encountered on the fourth clock cycle, when the add instruction is to be fetched from memory while the store word instruction is simultaneously using the same singular memory. The add instruction cannot be fetched until the sixth clock cycle since the branch if equal instruction does not need memory access in that clock cycle. From observing the sequence of instructions it is obvious that *nop* instructions are needed to stall the add instruction.

However it is not possible to add *nop* instructions since they too must be fetched from memory and this conflicts with load/store word instructions also using the same memory in the cycles that need to be stalled for the add instruction.

With the instruction sequence defined by the code plus two additional stalls generated due to the add instruction, the total number of clock cycles is [k stages + (n instructions - 1) + 2]
= 11 cycles

Clock cycle time in a pipelined organization is the slowest stage delay, 200ps.

Total execution time = clock cycle time * number of clock cycles
= 200ps * 11cycles
= 2200ps

4.10.2

Assuming clock cycle time is 200ps:

For a 4 stage pipeline the number of cycles needed for program execution is…
number of cycles = k stages + (n instructions - 1) = 8 cycles
total execution time = clock cycle time * number of clock cycles
= 200ps * 8 cycles
= 1600ps

For a 5 stage pipeline the number of cycles needed for program execution is…
number of cycles = k stages + (n instructions - 1) = 9 cycles
total execution time = clock cycle time * number of cycles
= 200ps * 9
= 1800ps

Speedup:
5 stage execution time / 4 stage execution time = 1800ps / 1600ps = 1.125

4.10.3
Assuming stall-on-branch and clock cycle time is 200ps:
If branch outcomes are determined in the Instruction Decode (ID) stage then only one stall is necessary, however branch outcomes determined in the Execution (EX) stage require two stalls.

ID outcomes:

number of cycles = k stages + (n instructions - 1) + (# branches * # stalls)
= 5 stages + (5-1) + (1 branch * 1 stall)
= 10 cycles

total execution time = clock cycle time * number of cycles
= 200ps * 10 cycles
= 2000ps

EX outcomes:

number of cycles = k stages + (n instructions - 1) + (# branches * # stalls)
= 5 stages + (5-1) + (1 branch * 2 stall)
= 11 cycles

total execution time = clock cycle time * number of cycles
= 200ps * 11 cycles
= 2200ps

Speedup:

EX outcomes / ID outcomes = 2200ps / 2000ps = 1.10

4.10.4
Assuming clock cycle time for 4 stage pipeline is 190ps + 20ps = 210ps
and clock cycle time for 5 stage pipeline remains 200ps:

For a 4 stage pipeline the number of cycles needed for program execution is…

number of cycles = k stages + (n instructions - 1) = 8 cycles

total execution time = clock cycle time * number of clock cycles
= 210ps * 8 cycles
= 1680ps

For a 5 stage pipeline the number of cycles needed for program execution is…

number of cycles = k stages + (n instructions - 1) = 9 cycles

total execution time = clock cycle time * number of cycles
= 200ps * 9
= 1800ps

Speedup:

5 stage execution time / 4 stage execution time = 1800ps / 1680ps = 1.07

4.10.5

Assuming stall-on-branch:
If the latency of the ID stage increases by 50% and the latency of the EX stage decreases by 10ps when branch outcomes occur in the ID the clock cycle will still be 200ps (slowest stage is IF). Therefore the speedup doesn't change from exercise 4.10.3.

Speedup:
EX outcomes / ID outcomes = 2200ps / 2000ps = 1.10

4.10.6
None of the modifications made to the pipeline stages affect the clock cycle time it still remains at 200ps (IF). However by moving the branch outcome to the MEM stage an extra cycle is added and thus the execution time must be recomputed.

EX outcomes:
number of cycles = k stages + (n instructions - 1) + (# branches * # stalls)
= 5 stages + (5-1) + (1 branch * 2 stall)
= 11 cycles
total execution time = clock cycle time * number of cycles
= 200ps * 11 cycles
= 2200ps

MEM outcomes:
number of cycles = k stages + (n instructions - 1) + (# branches * # stalls)
= 5 stages + (5-1) + (1 branch * 3 stall)
= 12 cycles
total execution time = clock cycle time * number of cycles
= 200ps * 12 cycles
= 2400ps

Speedup:
EX outcomes / MEM outcomes = 2200ps / 2400ps = 0.92, this results in subpar performance

4.11

4.11.1

Given the code, there will need to be a stall in the AND instruction to allow the write-back stage to update register R1, a stall in the following LW instruction to allow the MEM stage update register R1, a stall in the following LW instruction and a final stall in the BEQ instruction.

```
IF  ID  EX  MEM  WB
    IF   ID   stall    EX  MEM  WB
         IF    stall    ID   EX     MEM   WB
                        IF    ID     stall     EX    MEM  WB
                              IF      stall     ID     EX      MEM  WB
```