NWEN242 Lab 2

David Barnett (300313764)

Preparatory Questions

Q1

- Structural hazard: Using the same resource, such as memory, at the same time. An example would be reading & writing to registry file at the same time.
- Data hazard: Accessing data that has been modified by another instruction.
- Control hazard: The hazard introduced through branching instructions with a chance of having the wrong instruction in the pipeline. This can be caused by assuming the branch will not be taken but in running it is taken.

Q2

- Splitting reading & writing to the registry file to the start & end of the cycle to solve a structural hazard and help towards data hazard.
- Forwarding data computed but not written to the registry to instructions in earlier parts of the pipeline.
- The hazard control unit inserts bubbles (NOP) when the instruction depends on the result of the next instruction but cannot be forwarded the result, such as loading from memory.

Q3

Number of stages in the pipeline minus one, so the instruction causing the hazard will be completed as the next instruction can safely run after it.

Q4

The special registry file allows for writes at the start of the cycle and reads at the end of the cycle. This allows for a write back and instruction decode to happen at the same time with the register as the target of the write back and a source for the decoding instruction.

Q5

The forwarding unit would forward the value from the MEM/WB stage back to the EX stage. The forwarding unit itself does not handle any special case for `lw` unlike the Hazard Detection unit.

Q6

No bubbles are needed when two hazardous R-Type instructions are run in sequence as the forwarding from EX/MEM will overwrite the now out-of-date values gotten from registry file during the EX stage

Q7

No stalls are required when using a memory instruction after a R-type value as the forwarding unit will update the value if the memory instruction is dependent on the previous R-type instruction.

Q8

With a forwarding unit one nop is needed when a R-Type instruction follows and is dependent on the previous memory load instruction due to the value not being available until MEM/WB stage.

Q9

Given that the forwarding unit can forward to the value to the ID stage the number of NOPs would be 1 due to forwarding from the EX/MEM stage to the ID stage.

Q10

There needs to be no NOPs between the branch and R-type instruction as the result can be forwarded in the EX/MEM stage to the ID stage.

Q11

There needs to be two NOPs between the lw and a following branch instruction because the lw's result is only available for forwarding after MEM/WB.

Exercise 4.1

A

The results of experimenting with the different modes of MIPS using the original, unmodified, program.

Basic

At cycle 10 of the program the register `$2` is updated to `6`

Bubbles

At cycle 10 of the program the register `$2` is updated to `6`

Forwarding

At cycle 10 of the program the register `$2` is updated to `6`

Unmodified

At cycle 10 of the program the register `$2` is updated to `6`

B

Basic

The value of register `$2` is needed at cycle 11 of the program

Bubbles

The value of register `$2` is needed at cycle 10 of the program

Forwarding

The value of register `$2` is needed at cycle 8 of the program but then updated to the new value at cycle 9 via forwarding

C

The problem that occurred is that the second `add` is dependent on the result of the previous `add` instruction. In the case of the unmodified version the wrong value of `$2` is used because the result has not been written back to the registry file yet. This is an example of a data hazard.

Exercise 4.2

A

The results of experimenting with the different modes of MIPS using the original, unmodified, program.

Basic

The branch instruction was ready to jump after `6` cycles of the program

Forwarding

The branch instruction was ready to jump after `4` cycles of the program

Forwarding & Branch

The branch instruction was ready to jump after `4` cycles of the program

B

The following `addi` was loaded into the pipeline following before the branch and not flushed so it completed a full execution and updated `$2` to `1` preventing the, seemingly, infinite loop.

C

The problem occurring is the instruction that was after the branch statement was loaded into the pipeline and executed in the normal linear fashion, but the branch was taken so the operation after the branch was meant to be the instruction at the target location. Since the `addi` instruction was already in the pipeline and not flushed out after the unsuccessful implicit branch prediction. This is called a control hazard.

Exercise 4.3

A

The results of experimenting with the different modes of MIPS using the original, unmodified, program.

Basic

Register `$3` has received result from the `lw` the value after `10` cycles. However the value of `$3` is incorrect from what is expected of the program.

Bubbles

Register `$3` has received result from the `lw` the value after `12` cycles. The value of `$3` was the expected value.

Forwarding

Register `$3` has received result from the `lw` the value after `10` cycles. The value of `$3` was the expected value.

B

The results of experimenting with the different modes of MIPS using the original, unmodified, program.

Basic

The number of clock cycles till the value of `$3` is needed was `8` cycles.

Bubbles

The number of clock cycles till the value of `$3` is needed was `12` cycles.

Forwarding

The number of clock cycles till the value of `$3` is needed was `8` cycles, but then later updated to the result of the `lw` in cycle `9`.

C

The problem is the program requires the results of the previous result, in both cases for the `sw` requires the result of the previous `addi` and also the second `addi` requires the result of the previous `lw`. This is an example of a data hazard.

Exercise 5.1

Mode Option	Instructions	Cycles	CPI (3.d.p)
Basic	208	214	1.028
Bubbles	106	212	2.0
Bubbles with Branch Assumptions	78	156	2.0
Forwarding	132	138	1.045
Forwarding with Branch Assumptions	97	109	1.124

Exercise 5.2

A

The number of cycles is used to calculate the performance of a pipeline because the cycle count gives the total time taken by multiplying it the clock frequency. CPI or total instructions are not appropriate measures as the instruction count only counts the number of instructions loaded into pipeline.The CPI is also not used because it is derived from the instruction count. The table above also shows that having the best instruction count or CPI does not mean the program finished in the fastest time.

B

The different version of the same program using different data paths exhibit differing performance because of the assumptions held. For example with forwarding enabled the data path can gain performance from assuming the correct path will be taken saving it from having to use `nop`s to ensure a correct branch and correct instructions after it. Another example of this is also having write back applied in the first half of the cycle instead of the second saves a need for another `nop` in between two dependent instructions.

Exercise 5.3

Mode Option	Instructions	Cycles	CPI (3.d.p)
Forwarding Optimized	125	131	1.049
Forwarding with Branch Assumptions Optimized	83	96	1.156