RYERSON UNIVERSITY

Faculty of Engineering, Architecture, and Science

Department of Electrical and Computer Engineering

Course Number	EE8207
Course Title	High Performance Computer System Design
Semester/Year	Winter/2016

Instructor	Dr. Nagi N. Mekhiel

Lab No.	**4**

Assignment Title	Control Hazards and ILP

Submission Date	1st April 2016
Due Date	1st April 2016

Student Name	Muhammad Obaidullah
Student ID.	500671408
Signature*	[illegible]

1 Objectives of the lab

1. Extract ILP with loop unrolling.
2. Use scheduling and register renaming to reduce hazards.
3. Performance improvements of ILP and scheduling.
4. Use of different branch predictors to reduce control hazards.

2 Converting C code to assembly language

2.1 Complete C Code

```c
int main() {
  // Filling x[] up with random data
  int x[1000];
  int a = 9;
  for (int i = 0; i < 1000; i++)
  {
    x[i] = rand() % 100 + 1;
  }
  // Filling y[] up with random data
  int y[1000];
  for (int i = 0; i < 1000; i++)
  {
    y[i] = rand() % 100 + 1;
  }
  // Original Code
  for(int i = 0; i <= 1000; i++)
  {
    x[i] = a * x[i] + y[i];
  }
  return 0;
}
```

3 Without Optimization

3.1 Assembly Code

```
      .data
A:          .word 9
B:          .word 4000
C:          .word 40
D:          .word 6040

      .text
main:
      ld r1,A(r0)
      ld r2,B(r0)
      ld r3,C(r2)
      ld r4,D(r2)
      dmul r5,r1,r3
      dadd r6,r5,r4
      sd r6,C(r2)
      daddi r2,r2,-4
      bnez r2, 8
      halt
```

3.2 Performance

This code runs in 15 cycles with 6 stalls due to RAW and 1 stall due to branch control hazard.

$$Cycles\ Per\ Iteration\ (CPI) = 15 \tag{3.1}$$

If the clock of the computer is 1 GHz:

$$Performance = \frac{15 \times 1000}{1 \times 10^9} = 15\mu s \tag{3.2}$$

3.3 Simulation in WinMIPS

After writing the code is was verified using the asm tool provided with WinMIPS. The following are the results from WinMIPS.

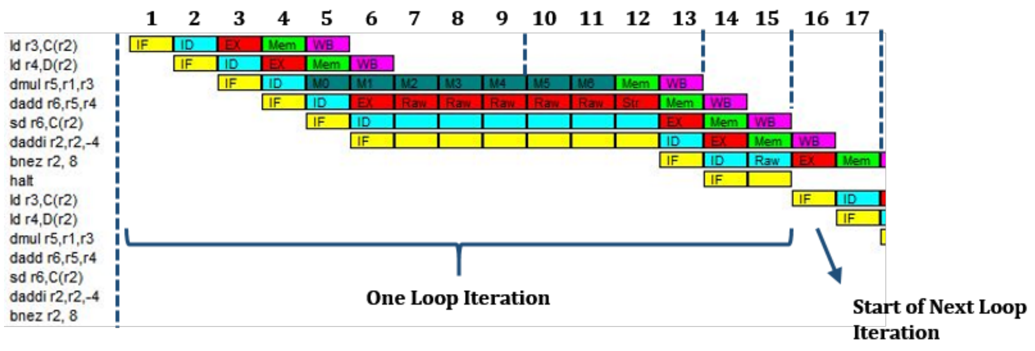


Figure 3.1: First two instructions are just loading the variables so the cycles wasted on it are not counted. Only the iteration cycle time is counted as shown in the figure.

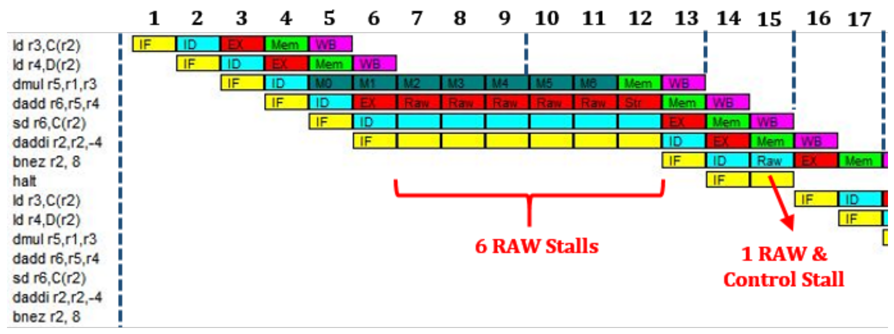


Figure 3.2: The figure shows that there are 6 stalls due to RAW hazard and 1 stall which occurs due to RAW and Control hazard at the same time.

4 With Scheduling Only

4.1 Assembly Code

```
    .data
A:      .word 9
B:      .word 4000
C:      .word 40
D:      .word 6040

    .text
main:
    ld r1,A(r0)
    ld r2,B(r0)
    ld r3,C(r2)
    dmul r5,r1,r3
    ld r4,D(r2)
    daddi r2,r2,-4
    dadd r6,r5,r4
    sd r6,C(r2)
    bnez r2, 8
    halt
```

4.2 Performance

This code runs in 13 cycles with 5 stalls due to RAW and 1 stall due to branch control hazard.

$$Cycles\ Per\ Iteration\ (CPI) = 13 \tag{4.1}$$

If the clock of the computer is 1 GHz:

$$Performance = \frac{13 \times 1000}{1 \times 10^9} = 13\mu s \tag{4.2}$$

Improvement is given by:

$$Improvement = \frac{Old\ CPI}{New\ CPI} = \frac{15}{13} \approx 1.15\ times \tag{4.3}$$

4.3 Simulation in WinMIPS

After writing the code is was verified using the asm tool provided with WinMIPS. The following are the results from WinMIPS.

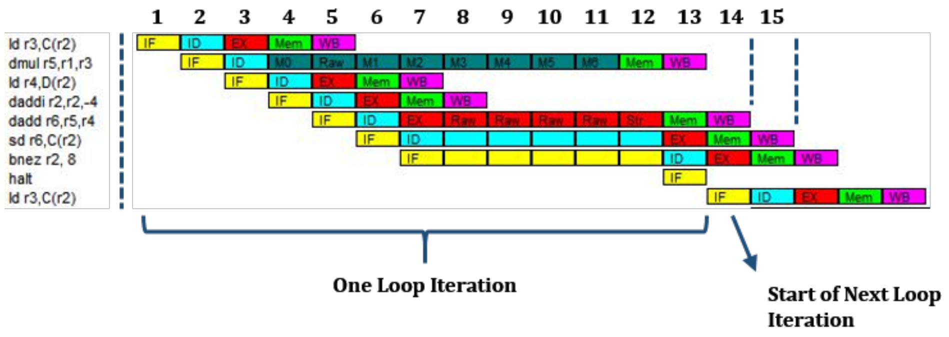


Figure 4.1: First two instructions are just loading the variables so the cycles wasted on it are not counted. Only the iteration cycle time is counted as shown in the figure.

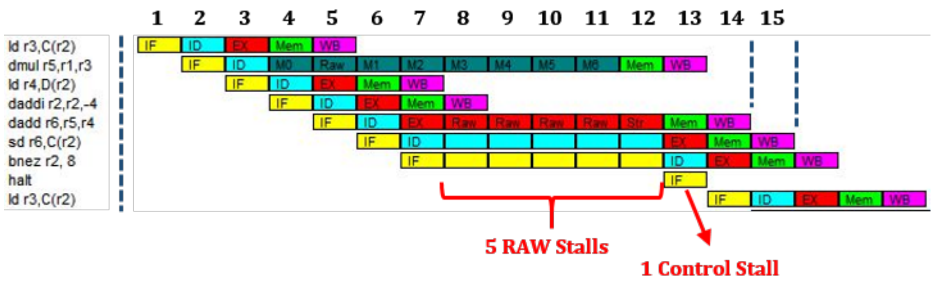


Figure 4.2: The figure shows that there are 5 stalls due to RAW hazard and 1 stall which occurs due Control hazard at the same time.

5 Loop Unrolling without Optimization

5.1 Assembly Code

```
    .data
A:          .word  9
B:          .word  4000
C:          .word  40
D:          .word  8000

    .text
main:
    ld  r1,A(r0)    # Loading a
    ld  r2,B(r0)    # Loading i1
    ld  r3,D(r0)    # Loading i2
    ld  r4,0(r2)    # Loading x[i1]
    ld  r5,-4(r2)   # Loading x[i1 + 1]
    ld  r6,-8(r2)   # Loading x[i1 + 2]
    ld  r7,-12(r2)    # Loading x[i1 + 3]
    ld  r8,-16(r2) # Loading x[i1 + 4]
    dmul  r4,r1,r4 # Multiplying x[i1] with a
    dmul  r5,r1,r5 # Multiplying x[i1 + 1] with a
    dmul  r6,r1,r6 # Multiplying x[i1 + 2] with a
    dmul  r7,r1,r7 # Multiplying x[i1 + 3] with a
    dmul  r8,r1,r8 # Multiplying x[i1 + 4] with a
    ld  r9,0(r3)    # Loading y[i2]
    ld  r10,-4(r3) # Loading y[i2 + 1]
    ld  r11,-8(r3) # Loading y[i2 + 2]
    ld  r12,-12(r3)  # Loading y[i2 + 3]
    ld  r13,-16(r3)  # Loading y[i2 + 4]
    daddi  r2,r2,-20 # i1 = i1 - 20 (5*4 = 20)
    daddi  r3,r3,-20 # i2 = i2 - 20 (5*4 = 20)
    dadd  r4,r9,r4 # Doing x[i1] = a*x[i1] + y[i2]
    dadd  r5,r10,r5  # Doing x[i1 + 1] = a*x[i1 + 1] + y[i2 + 1]
    dadd  r6,r11,r6  # Doing x[i1 + 2] = a*x[i1 + 2] + y[i2 + 2]
    dadd  r7,r12,r7  # Doing x[i1 + 3] = a*x[i1 + 3] + y[i2 + 3]
    dadd  r8,r13,r8  # Doing x[i1 + 4] = a*x[i1 + 4] + y[i2 + 4]
    sd  r4,20(r2)  # Storing x[i1]
    sd  r4,16(r2)  # Storing x[i1 + 1]
    sd  r4,12(r2)  # Storing x[i1 + 2]
    sd  r4,8(r2)   # Storing x[i1 + 3]
    sd  r4,4(r2)   # Storing x[i1 + 4]
    bnez  r2, 12
    halt
```

5.2 Performance

This code runs in 30 cycles with 5 stalls due to Structural hazards and 1 stall due to branch control hazard.

$$Cycles\ Per\ Iteration\ (CPI) = \frac{30\ Clocks\ Per\ Iteration}{5\ Loops\ Per\ Iteration} = 6 \tag{5.1}$$

If the clock of the computer is 1 GHz:

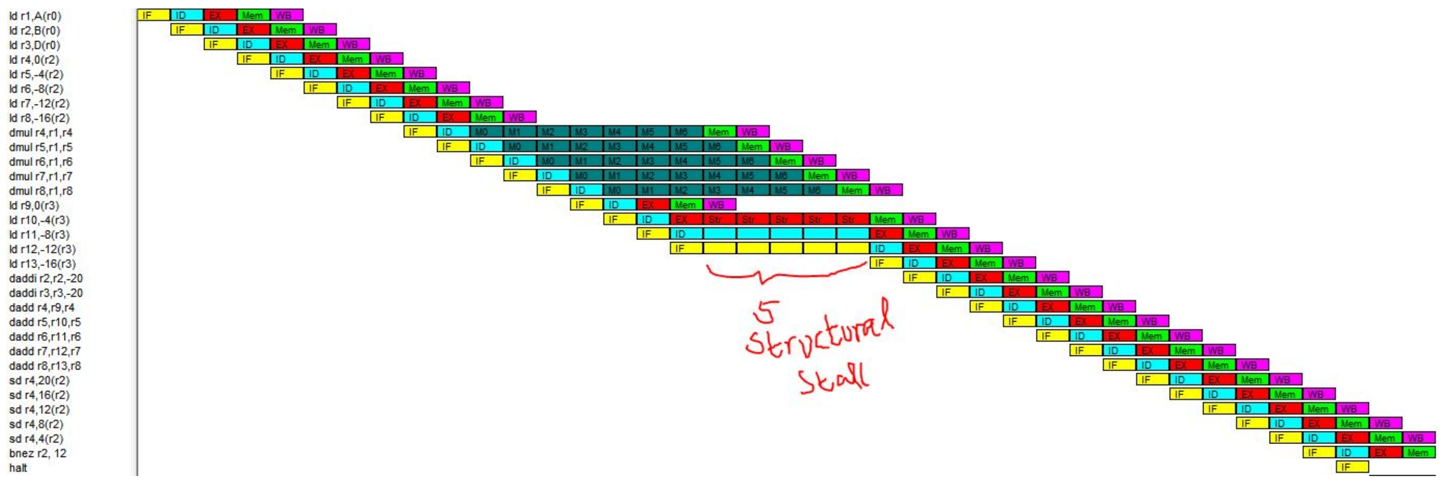
$$Performance = \frac{6 \times 1000}{1 \times 10^9} = 6\mu s \tag{5.2}$$

Improvement is given by:

$$Improvement = \frac{Old\ CPI}{New\ CPI} = \frac{15}{6} \approx 2.5\ times \tag{5.3}$$

5.3 Simulation in WinMIPS

After writing the code is was verified using the asm tool provided with WinMIPS. The following are the results from WinMIPS.



```
ld r1,A(r0)
ld r2,B(r0)
ld r3,D(r0)
ld r4,0(r2)
ld r5,-4(r2)
ld r6,-8(r2)
ld r7,-12(r2)
ld r8,-16(r2)
dmul r4,r1,r4
dmul r5,r1,r5
dmul r6,r1,r6
dmul r7,r1,r7
dmul r8,r1,r8
ld r9,0(r3)
ld r10,-4(r3)
ld r11,-8(r3)
ld r12,-12(r3)
ld r13,-16(r3)
daddi r2,r2,-20
daddi r3,r3,-20
dadd r4,r9,r4
dadd r5,r10,r5
dadd r6,r11,r6
dadd r7,r12,r7
dadd r8,r13,r8
sd r4,20(r2)
sd r4,16(r2)
sd r4,12(r2)
sd r4,8(r2)
sd r4,4(r2)
bnez r2, 12
halt
```

Figure 5.1: This is the cycles window after un-rolling the loop 5 times. However, there are several structural hazards because the multiplier needs to write the values back to memory while the current instruction also needs the Write Back Block.

6 Loop Unrolling With Optimization

6.1 Assembly Code

```
        .data
A:              .word  9
B:              .word  4000
C:              .word  40
D:              .word  8000

        .text
main:
        ld  r1,A(r0)    # Loading a
        ld  r2,B(r0)    # Loading i1
        ld  r3,D(r0)    # Loading i2
        ld  r4,0(r2)    # Loading x[i1]
        ld  r5,-4(r2)   # Loading x[i1 + 1]
        ld  r6,-8(r2)   # Loading x[i1 + 2]
        ld  r7,-12(r2)    # Loading x[i1 + 3]
        ld  r8,-16(r2)  # Loading x[i1 + 4]
```

```
    dmul r4,r1,r4 # Multiplying x[i1] with a
    ld r9,0(r3)   # Loading y[i2]
    daddi r2,r2,-20 # i1 = i1 - 20 (5*4 = 20)
    dmul r5,r1,r5 # Multiplying x[i1 + 1] with a
    ld r10,-4(r3) # Loading y[i2 + 1]
    dmul r6,r1,r6 # Multiplying x[i1 + 2] with a
    ld r11,-8(r3) # Loading y[i2 + 2]
    dmul r7,r1,r7 # Multiplying x[i1 + 3] with a
    ld r12,-12(r3)  # Loading y[i2 + 3]
    dmul r8,r1,r8 # Multiplying x[i1 + 4] with a
    ld r13,-16(r3)  # Loading y[i2 + 4]
    daddi r3,r3,-20 # i2 = i2 - 20 (5*4 = 20)
    dadd r4,r9,r4 # Doing x[i1] = a*x[i1] + y[i2]
    dadd r5,r10,r5  # Doing x[i1 + 1] = a*x[i1 + 1] + y[i2 + 1]
    dadd r6,r11,r6  # Doing x[i1 + 2] = a*x[i1 + 2] + y[i2 + 2]
    dadd r7,r12,r7  # Doing x[i1 + 3] = a*x[i1 + 3] + y[i2 + 3]
    dadd r8,r13,r8  # Doing x[i1 + 4] = a*x[i1 + 4] + y[i2 + 4]
    sd r4,20(r2)  # Storing x[i1]
    sd r4,16(r2)  # Storing x[i1 + 1]
    sd r4,12(r2)  # Storing x[i1 + 2]
    sd r4,8(r2)   # Storing x[i1 + 3]
    sd r4,4(r2)   # Storing x[i1 + 4]
    bnez r2, 12
    halt
```

6.2 Performance

This code runs in 28 cycles with 3 stalls due to Structural hazards and 1 stall due to branch control hazard.

$$Cycles\ Per\ Iteration\ (CPI) = \frac{28\ Clocks\ Per\ Iteration}{5\ Loops\ Per\ Iteration} = 5.6 \tag{6.1}$$

If the clock of the computer is 1 GHz:

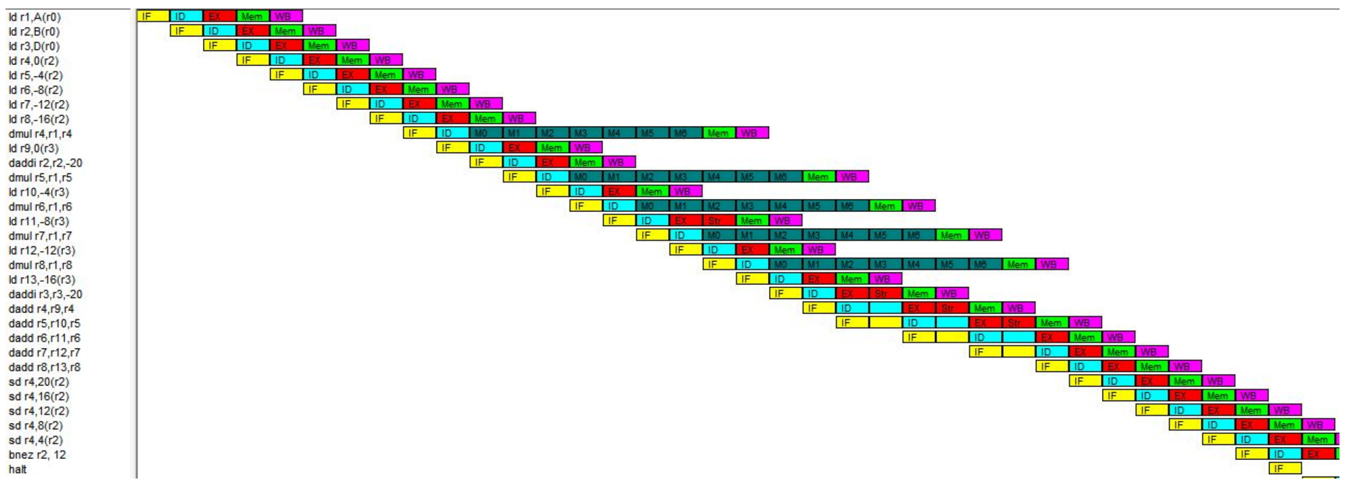
$$Performance = \frac{5.6 \times 1000}{1 \times 10^9} = 5.6\mu s \tag{6.2}$$

Improvement is given by:

$$Improvement = \frac{Old\ CPI}{New\ CPI} = \frac{15}{5.6} \approx 2.68\ times \tag{6.3}$$

6.3 Simulation in WinMIPS

After writing the code is was verified using the asm tool provided with WinMIPS. The following are the results from WinMIPS.



```
ld r1,A(r0)
ld r2,B(r0)
ld r3,D(r0)
ld r4,0(r2)
ld r5,-4(r2)
ld r6,-8(r2)
ld r7,-12(r2)
ld r8,-16(r2)
dmul r4,r1,r4
ld r9,0(r3)
daddi r2,r2,-20
dmul r5,r1,r5
ld r10,-4(r3)
dmul r6,r1,r6
ld r11,-8(r3)
dmul r7,r1,r7
ld r12,-12(r3)
dmul r8,r1,r8
ld r13,-16(r3)
daddi r3,r3,-20
dadd r4,r9,r4
dadd r5,r10,r5
dadd r6,r11,r6
dadd r7,r12,r7
dadd r8,r13,r8
sd r4,20(r2)
sd r4,16(r2)
sd r4,12(r2)
sd r4,8(r2)
sd r4,4(r2)
bnez r2, 12
halt
```

Figure 6.1: This is the cycles window after un-rolling the loop 5 times. Structural hazards previously present are removed up to some point.

7 BENCHMARKING USING SIMPLESCALAR

Following is the result from SimpleScalar simulator:

```
sim: ** starting functional simulation **

sim: ** simulation statistics **
sim_num_insn                    4058 # total number of instructions executed
sim_num_refs                    1337 # total number of loads and stores executed
sim_elapsed_time                   1 # total simulation time in seconds
sim_inst_rate              4058.0000 # simulation speed (in insts/sec)
ld_text_base              0x00400000 # program text (code) segment base
ld_text_size                    2318 # program text (code) size in bytes
ld_data_base              0x10000000 # program initialized data segment base
ld_data_size                    4096 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base             0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                  16384 # program initial stack size
ld_prog_entry             0x00400140 # program entry point (initial PC)
ld_environ_base           0x7fff8000 # program environment base address address
ld_target_big_endian               0 # target executable endian-ness, non-zero if big endian
mem.page_count                    14 # total number of pages allocated
mem.page_mem                     56k # total size of memory pages allocated
mem.ptab_misses                   14 # total first level page table misses
mem.ptab_accesses             334404 # total page table accesses
mem.ptab_miss_rate            0.0000 # first level page table miss rate
```

7.1 AT 1GHZ, HOW MUCH TIME TO PROCESS ?

$$Speed = \frac{4058}{1 \times 10^9} \approx 4\mu s \tag{7.1}$$

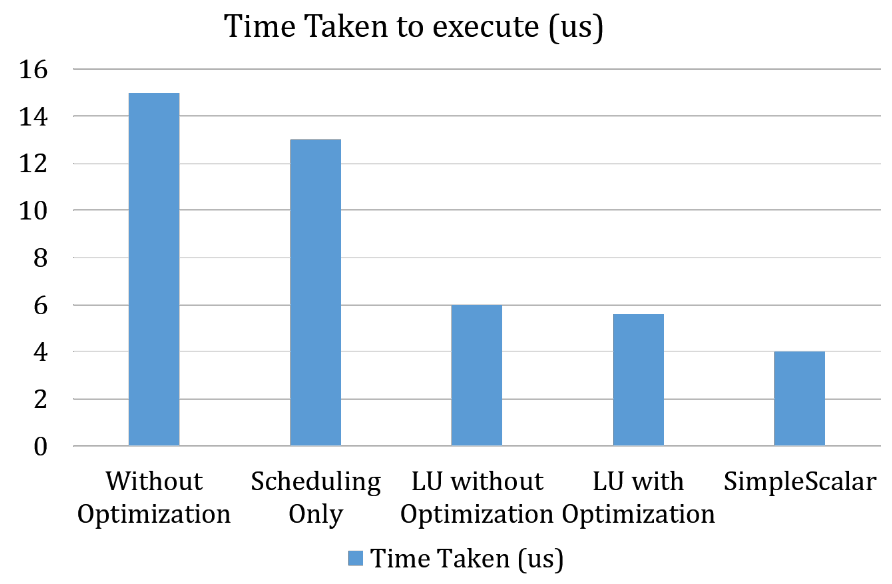


Figure 7.1: Comparison between different optimization types and time taken to execute code. LU means Loop Un-rolling.

8 Conclusion

Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC benchmark suites are used to evaluate several performance properties of a processor, compiler, and memory. SPEC benchmark can also be used to find out which addressing modes are most frequently used by the processor and are worth implementing in hardware. Only Most Frequently Used (MFU) addressing modes are implemented in order to save processor implementation cost (implementing less hardware will save die area and reduce complexity of design).
Immediate, register, direct, memory indirect, and displacement are usually selected as worth implementing in a processor. Size of immediate and size of displacement is kept at 16 bits.

Branching hazards can be dealt with using:

1. **Predict branch is not taken** Using extra adder in decode, and evaluate condition in decode stage that reduces the cost of control hazard to only 1 cycle instead of 3 cycles.We noticed that the instruction next to branch is already being fetched. If we predict or assume that the branch is not taken. If the prediction is correct, we can continue executing the instruction and there will be no cost to branch.
   For example, if 60% of branches are taken and 40% are not taken. Then the cost will be:
   $$Cost = 1 + 0.2 \times 0.6\, times 1 = 1.12$$
   If we mis-predict, convert the instruction to NOOP (No-Operation).

2. **Predict Branch is taken** For this kind of prediction, MIPS ISA does not benefit because we anyways have to calculate the address.

3. **Delayed Branches** Assumes every instruction after branch is always executed no matter if branch is taken or not. Processor has a delayed slot that the compiler will fill with a useful instruction.
   Miss Prediction: Branch Cancellation If miss predict, will use canceling branches. Instruction in delay slot (wrong instruction) is canceled. The advantage of this is that this allows the compiler to become more aggressive.
   Assume: Branch frequency = 20%. We have 50% taken and compiler could only fill 80% of delayed slot and 90% of time correct prediction.
   a) Prediction: That the branch is not taken.
   $$Performance = 1 + 0.2 \times 1 \times 0.5 = 1.10$$
   10% Reduction in performance.

b) Prediction: That the branch is taken.

$$Performance = 1 + 0.2 \times 1 \times 1 = 1.2$$

20% Reduction in performance.

c) Delayed branches. Delay slot mis-prediction.

$$Performance = 1 + 0.2\Big(0.2 \times 1 + 0.8 \times 0.1 \times 1\Big)$$

$$Performance = 1 + 0.2\Big(0.2 + 0.08\Big)$$

$$= 1 + 0.2\Big(0.28\Big) = 1.056$$

Only 5.6% Reduction in performance.

8.1 Exception Handling in Pipeline

Why we need exceptions?

- I/O devices using interrupts to communicate with CPU
- Operating System
- Memory Faults
- Hardware failure/malfunction.

Behavior of exception types:-

1. Synchronous/Asynchronous from software occur at specific location of code. Asynchronous caused by device.
2. Maskable/Non-maskable
3. Resume Execution/Terminate
4. Occur between instructions or within the instruction

It is difficult to support precise exceptions (resume) and if exception is within the instruction. Need to be restartable even within the instruction.

Exceptions in 1st stage (Instruction Fetch)

1. Exception page fault

Exceptions in 2nd stage (Instruction Decode)

1. Wrong Code

Exceptions in 3rd stage (Execute)

1. Arithmetic overflow

Exceptions in 4th stage (Memory)

1. Page Fault

How processor serves an exception?

1. Collect processor state
2. Go to the offending exception and service it
3. Restore the processor state

4. Return back to execute the next instruction. Resume as if nothing occurred.

Why pipelining complicates exception handling?

1. Two instructions generate exceptions at the same time.
2. An earlier instruction causes and exception later after the following instruction.
3. Exceptions must be handled as if it is not pipelined (in order).

How to handle exceptions in pipeline?

1. All instructions before the offending instructions must complete.
2. Must wait for all following instruction until the offending instruction is handled.
3. Turn off all write units of the following instructions.
4. Status vector register collects the interrupts an is examined by processor. Posts any interrupt request.

REFERENCES

[1] M. Scott. (2012, April) Winmips64. [Online]. Available: http://indigo.ie/~mscott/

[2] D. Tullsen. Pipeline hazards. Pdf. Jacobs School of Engineering. [Online]. Available: http://cseweb.ucsd.edu/classes/wi05/cse240a/pipe2.pdf