Computer Architecture Homework 2 Report

----------------------------------------------------------------------

StudentID:B11901164 Name:陳秉緯

**Q1. (Using Ripes) Imply simulation on hw1\_2.s with different processors and make discussion.**

* **Only use 32 bits processor**
* **Store your final program into hw2\_1.s**

Q1-1. What is the cycle count when using 5-stage processor and 5-stage processor w/o forwarding unit? (5%)

|  |  |
| --- | --- |
| Processor | Cycle Count |
| 5-stage processor | 729 |
| 5-stage processor w/o forwarding | 1210 |

Q1-2. How does forwarding improve efficiency? Provide a small example from your code.(10%)

The forwarding unit improves efficiency by allowing the processor to use data as soon as it's computed, rather than waiting for it to be written back to the register file.

Example:

```assembly

lw t4, 0(t3) # t4 = nums[i]

bge t2, t4, continue\_inner # if nums[j] >= nums[i], skip

```

Without forwarding, each time this sequence executes, the processor must stall until t4 is written back to the register file before the branch can use it.

Q1-3. Did your program run correctly under 5-stage processor w/o hazard detection? What kind of hazard might occur in this setting? Try to fix the program and compare the cycle count with 5-stage processor.(25%)

No.

load-use hazards:

```assembly

lw t3, 0(t2) # Load value

add t4, t3, t5 # Use loaded value immediately

```

I need to manually insert NOPs or independent instructions between loads and their dependent instructions.

|  |  |
| --- | --- |
| Processor | Cycle Count |
| Original 5-stage processor | 729 |
| Fixed 5-stage processor w/o hazard detection | 800 |

Because of extra NOPs, the cycle counts are more than the original 5-stage processor without adding NOPs.

**Q2. Run your Homework 1 assembly code (hw1\_2.s) on gem5. Optimize the simulation time and list the changes in simulation time (ticks) before and after the modifications. Describe the methods you used to optimize the simulation time. (30%)**

* **Don't use any optimization option (O2,O3…) in toolchain**
* **Don’t modify compiler option and gem5\_config.py**
* **Store your final program into hw2\_2.s**

|  |  |
| --- | --- |
| Version | simTicks |
| Original (hw1\_2.s) | 8143749000 |
| Final (hw2\_2.s) | 8127261000 |

* Precompute nums[i] once before inner loop to avoid duplicate slli and add

Original:

```assembly

outer\_loop:

...

inner\_loop:

...

slli t0, s0, 2 # t0 = i \* 4

add t3, a1, t0 # t3 = &nums[i]

lw t4, 0(t3) # t4 = nums[i]

...

```

Optimized:

```assembly

outer\_loop:

...

slli t6, s0, 2 # t6 = i\*4

add t3, a1, t6 # t3 = &nums[i]

lw t4, 0(t3) # t4 = nums[i]

...

inner\_loop:

...

```

* Precompute dp address for i once

Original:

```assembly

outer\_loop:

...

inner\_loop:

...

add t5, a2, t0 # t5 = &dp[i]

lw a4, 0(t5) # a4 = dp[i]

...

```

Optimized:

```assembly

outer\_loop:

...

add t5, a2, t6 # t5 = &dp[i]

lw a4, 0(t5) # a4 = dp[i]

...

inner\_loop:

...

```

**Q3. Implement the same function using both C++ and assembly.**

Q3-1. Record the instruction count and execution time of two version (10%)

(Read simInsts, simTicks from stats.txt)

|  |  |  |
| --- | --- | --- |
| Version | simInsts | simTicks |
| hw2\_assembly | 122860 | 9029659000 |
| hw2\_ccode | 132688 | 10352589000 |

Q3-2. Discuss your observations. What are the advantages and disadvantages of these two implementations?(20%)

* hw2\_assembly
  + advantages:
    - Fewer instructions and cycles because I can manually optimize loops, avoid redundant computations, and control data access patterns.
    - No compiler-generated boilerplate, such as unnecessary safety checks.
  + disadvantages:
    - Hard to write and maintain.
    - Difficult to debug.
* hw2\_ccode
  + advantages:
    - Easier to read, write, and maintain.
  + disadvantages:
    - Function calls, stack management, and type safety checks introduce additional instructions and execution time.