Assignment 3

3190102214 张沛全

01

a.

- 11 instructions indicate 11 cycles at least
- latencies for the instructions: 3+4+10+3+2+2+1+1 = 26
- one-cycle branch delay slot
- basic performance in total: 11+26+1 = 38 cycles

b.

pipeline 1	pipeline 2
fld(Loop)	
<stall>	
<stall>	
<stall>	
fmul(I0)	
<stall>	
<stall>	
<stall>	
<stall>	
fdiv(I1)	fld(I2)
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
<stall>	fadd(I3)
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
fadd(I4)	fsd(I5)
addi(I6)	addi(i7)
sub(I8)	bnz(I9)
	<stall>

- dependency
    - I1 waits for I0 and I0 waits for Loop
    - I3 waits for I2
    - I4 waits for I1 and I0
    - I5 waits for I3
- performance in total: 24 cycles

c.

pipeline 1	pipeline 2
fld(Loop)	fld(I2)
<stall>	<stall>
<stall>	<stall>
<stall>	<stall>
fmul(I0)	fadd(I3)
<stall>	<stall>
<stall>	<stall>
<stall>	fsd(I5)
<stall>	addi(I6)
fdiv(I1)	addi(i7)
<stall>	sub(I8)
<stall>	
<stall>	
<stall>	
<stall>	
<stall>	
<stall>	
<stall>	
<stall>	
<stall>	
fadd(I4)	
bnz(I9)	
<stall>	

- performance in total: 23 cycles

d.

```
fld f2, 0(Rx)
fld f4, 0(Ry)
fmul.d f2, f0, f2
fadd.d f4, f0, f4
fsd f4, 0(Ry)
fadd.d f10, f8, f2
fld f2, 8(Rx)
fld f4, 8(Ry)
fmul.d f2, f0, f2
```

```
fadd.d f4, f0, f4
fsd f4, 8(Ry)
addi Rx, Rx, 16
addi Ry, Ry, 16
fdiv.d f8, f2, f0
sub x20, x4, Rx
bnz x20, Loop
```

02

- Register renaming is a technique that abstracts logical registers from physical registers. Every logical register has a set of physical registers associated with it. Register renaming eliminate false data dependencies arising from the reuse of registers by assigning different logical register names to the register causing false data dependencies
- Example

  ```
  fdiv.d f0,f2,f4
  fadd.d f6,f0,f8
  fsd f6,0(x1)
  fsub.d f8,f10,f14
  fmul.d f6,f10,f8
  ```

  We can see that there are 2 anti-dependencies: one is between `fadd.d` and `fsub.d` caused by `f8` and one is between `fsd` and `fmul.d` caused by `f6`. There is one output dependency between `fadd.d` and `fmul.d` caused by `f6`. By register renaming, we can eliminate these three false data dependencies. We assign `S` to `f6` and `T` to `f8` and we have:

  ```
  fdiv.d f0,f2,f4
  fadd.d S,f0,f8
  fsd S,0(x1)
  fsub.d T,f10,f14
  fmul.d f6,f10,T
  ```

03

a.

There is no penalty and there is even 1 extra cycle being gained. When an unconditional branch instruction is hit in the buffer, the target instruction rather than the branch instruction will be decoded. Hence one cycle for the branch instruction is saved.

b.

- no branch folding: $5\% \times (90\% \times 0 + 10\% \times 2) = 0.01$
  - pipeline should retrieve the target instruction, which saves nothing
- branch folding: $5\% \times (90\% \times (-1) + 10\% \times 2) = -0.035$
  - the target instruction will be decoded if hit, which saves one cycle
  - the negative sign indicates that there is a performance gain

- the improvement is 0.045 CPI
- hit rate
  - if the *performance gain* is evaluated relative to the BTB without branch folding:
    $5\% \times (x \times (-1) + (1-x) \times 2) = 5\% \times (x \times 0 + (1-x) \times 2) \implies x = 0$
  - if the *performance gain* is evaluated relative to the pipeline without a BTB:
    $5\% \times (x \times (-1) + (1-x) \times 2) = 0 \implies x = 66.7\%$

04

a.

- 4 floats to read and 2 floats to write
- 4 multiply operations and 2 add/sub operations: 6 FLOPs in total
- arithmetic intensity = 6/6 = 1

b.

```
1. vmul vld # a_re*b_re & load a_im
2. vmul vld # a_im*b_im & load b_im
3. vsub vst # c_re & store
4. vmul vld # a_re*b_im & load next a_re
5. vmul vld # b_re*a_im & load next b_re
6. vadd vst # c_im & store
```

- 6 chimes are required
- 4 cycles per complex result value
  - total cycles: $6 \times 64 + 15 \times 6 + 8 \times 4 + 5 \times 2 = 516$ cycles
  - complex results (one `c_re` and one `c_im` are treated as separate complex result value): $64 \times 2 = 128$
  - cycles per result: $516/128 \approx 4$ (or 8 cycles per complex result)

c.

```
1. vmul # a_re*b_re
2. vmul # a_im*b_im
3. vsub vst # c_re & store
4. vmul # a_re*b_im
5. vmul vld vld # b_re*a_im & load next a_re &  load next b_re
6. vadd vst vld vld # c_im & store & load next a_im &  load next b_im
```

- the calculation is the same as *b*
- 4 cycles per complex result value

05

- 32/4 = 8 results every cycle on average
- FLOP/s is used to measure throughput
- Base throughput: $1.5G \times 0.80 \times 0.85 \times 0.70 \times 8 \times 10 = 57.12G$FLOP/s

a.

- output 16 results on average
- Throughput: $1.5G \times 0.80 \times 0.85 \times 0.70 \times 16 \times 10 = 114.24G$FLOP/s
- Speedup: 114.24/57.12 = 2

b.

- Throughput: $1.5G \times 0.80 \times 0.85 \times 0.70 \times 8 \times 15 = 85.68G$FLOP/s
- Speedup: 85.68/57.12 = 1.5

c.

- Throughput: $1.5G \times 0.80 \times 0.95 \times 0.70 \times 8 \times 15 = 63.84G$FLOP/s
- Speedup: 63.84/57.12 = 1.1