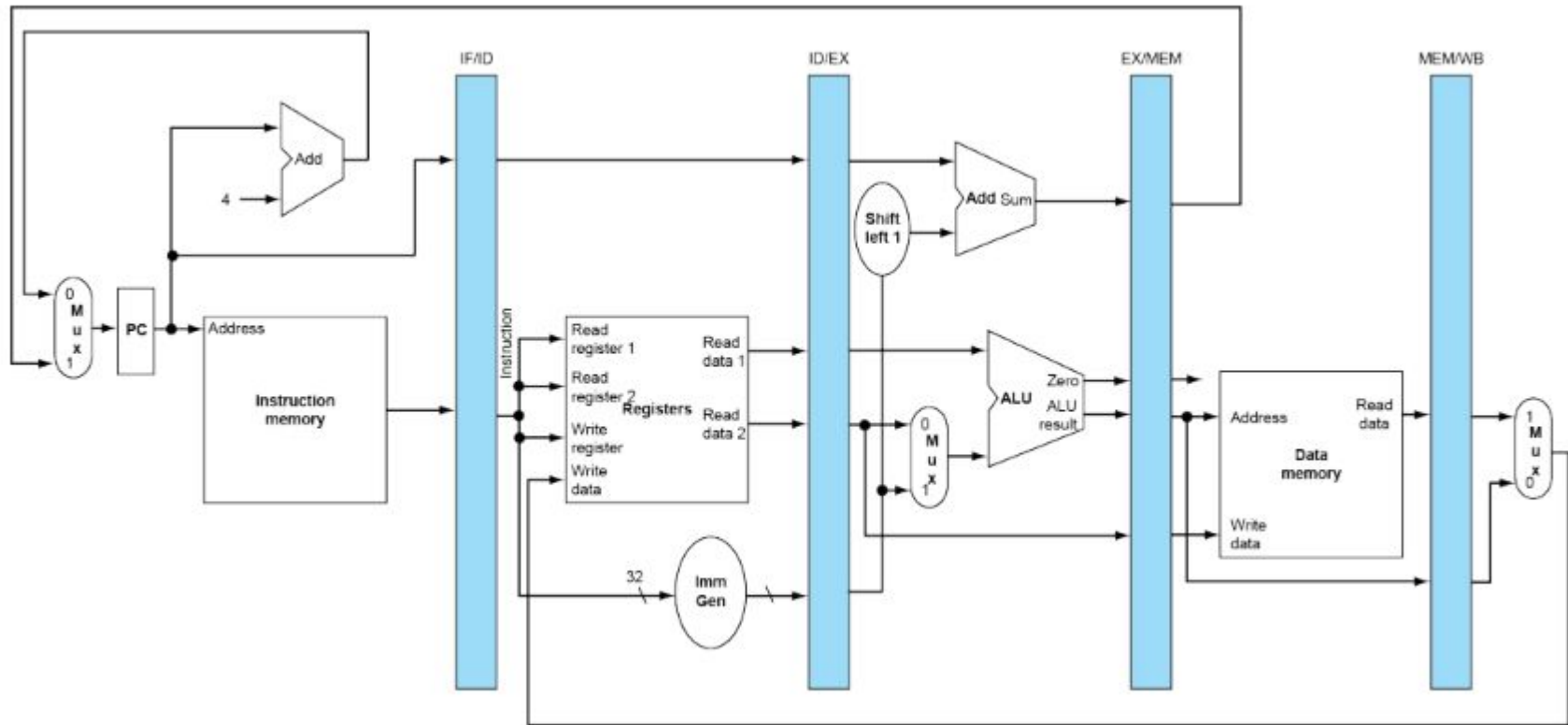

ECE M116C / CS 151B: Week 4 Section

Justin Feng



Hazards

Data Hazards

Read After Write (RAW):

When you read from a register you just wrote to

Ex) add x5, x6, x7

sub x4, x5, x8

-This is an issue (fix: stall, or forwarding)

Write After Read (WAR):

When you write to a register you just read from

Ex) add x5, x6, x7

sub x6, x4, x8

-Not an issue for in order execution

Data Hazards

Write After Write (WAW):

When you write to a register you just wrote to

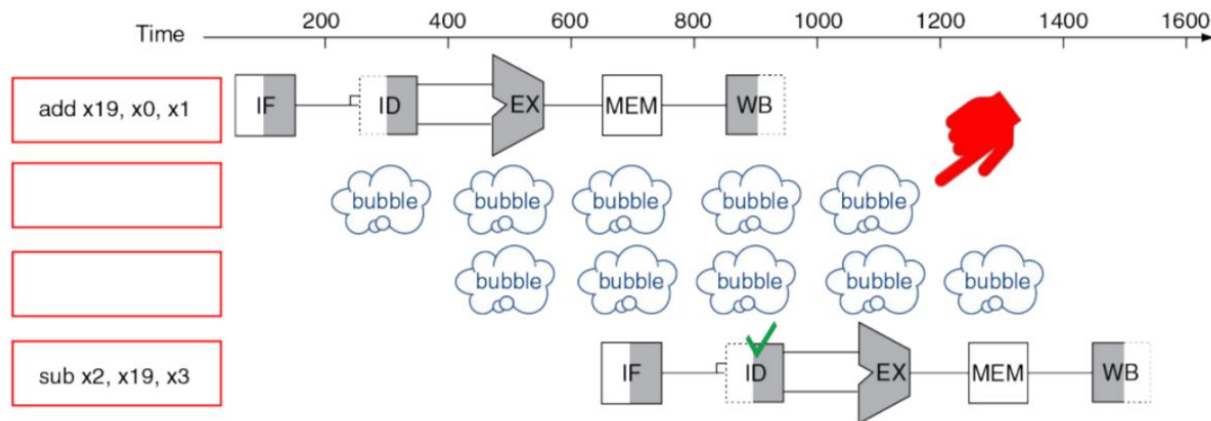
Ex) add x5, x6, x7

sub x5, x6, x7

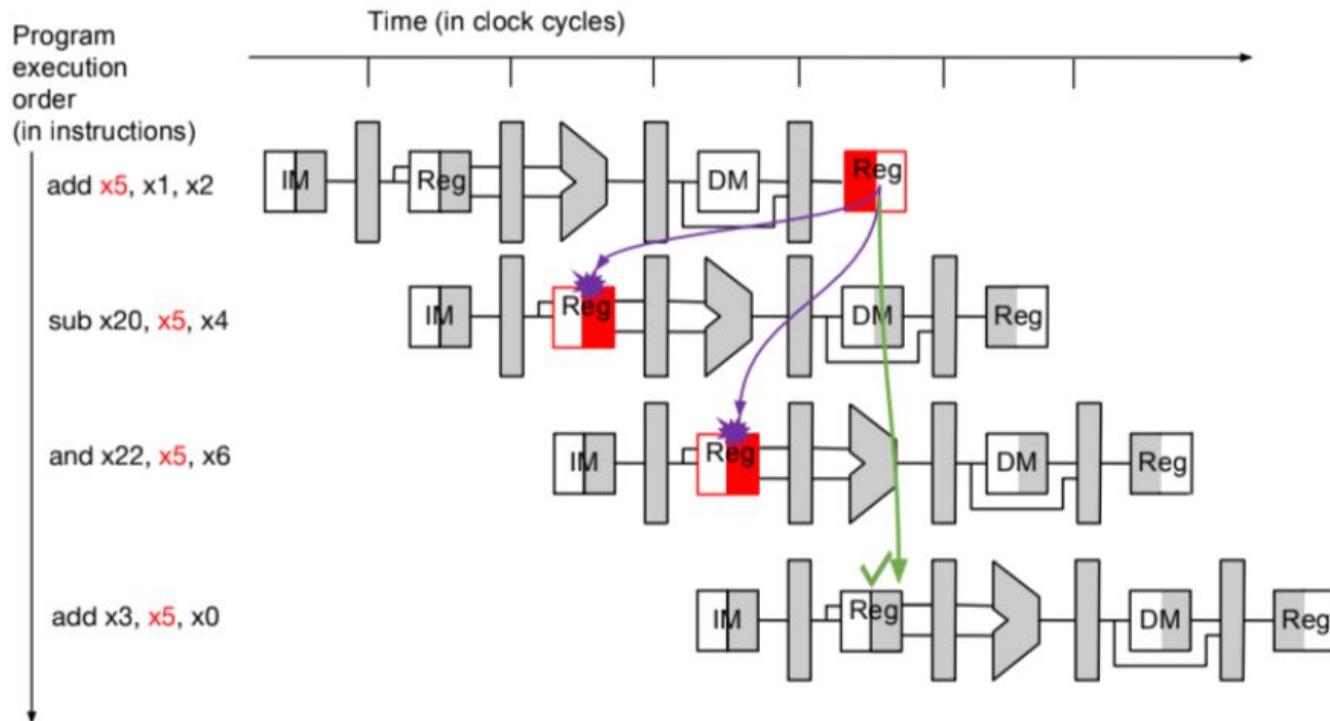
-Not an issue for in order execution

Data Hazard Solution: Stall

- Can be used to make use of half cycle
- Downside: decreased throughput!



Data Hazard Solution: Forwarding



-If want to avoid stalls to increase throughput, the next 2 instructions (next 2 cycles) might not have correct values!

Forwarding

-Used when data has not been written back into register file in time for a following instruction

-What about half cycle? If instruction that computes value in writeback, and instruction that needs value in decode, value will be available in time, w/o forwarding

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--------|---|---|---|---|---|---|---|---|
| Instr1 | I | D | E | M | W | | | |
| Instr2 | | I | D | E | M | W | | |
| Instr3 | | | I | D | E | M | W | |
| instr4 | | | | I | D | E | M | W |

Forwarding


-Case 1: dependence on result one instruction previous

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------|---|---|---|---|---|---|---|---|
| add x5, x6, x7 | I | D | E | M | W | | | |
| sub x8, x5, x9 | | I | D | E | M | W | | |
| Instr3 | | | I | D | E | M | W | |
| instr4 | | | | I | D | E | M | W |

Forwarding

-Case 2: dependence on result two instructions previous

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------|---|---|---|---|---|---|---|---|
| add x5, x6, x7 | I | D | E | M | W | | | |
| and x6, x7, x8 | | I | D | E | M | W | | |
| sub x8, x5, x9 | | | I | D | E | M | W | |
| instr4 | | | | I | D | E | M | W |



Forwarding

-Case 3: dependence on result for both previous instruction and two instructions previous

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------|---|---|---|---|---|---|---|---|
| add x5, x6, x7 | I | D | E | M | W | | | |
| and x5, x7, x8 | | I | D | E | M | W | | |
| sub x8, x5, x9 | | | I | D | E | M | W | |
| instr4 | | | | I | D | E | M | W |

When to Forward

*If ((MEM stage RD = EX stage RS1 OR MEM stage RD = EX stage RS2)
AND MEM stage RegWrite = 1)*

-> Forward data from MEM stage to EX stage

*Else If ((WB stage RD = EX stage RS1 OR WB stage RD = EX stage RS2)
AND WB stage RegWrite = 1)*

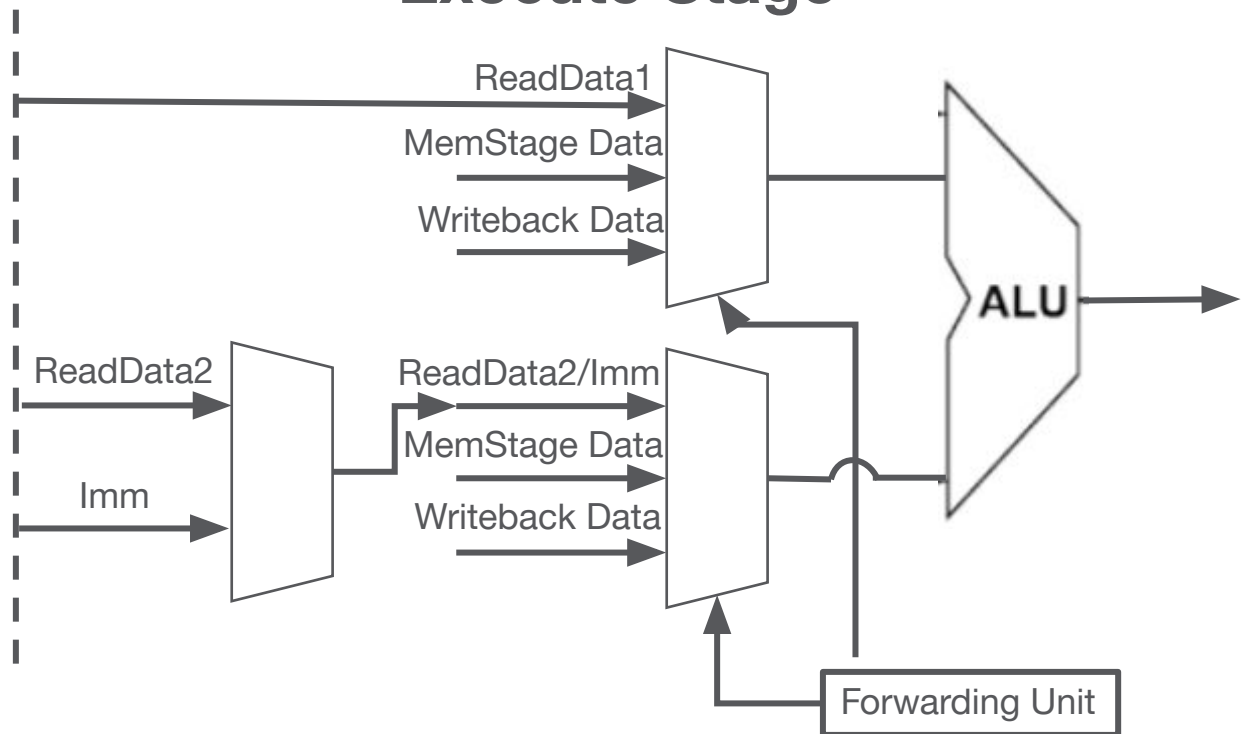
-> Forward data from WB stage to EX stage

Else

-> Don't forward data (use original register and/or immediate values)

Forwarding

Execute Stage



Load Use Hazard Solution: Stall

- Need to wait for data to be read from memory
 - Memory read takes too long to be ready in time for the next instruction's execute stage

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--------|---|---|---|---|---|---|---|---|
| Instr1 | I | D | E | M | W | | | |
| load | | I | D | E | M | W | | |
| nop | | | - | - | - | - | - | |
| instr4 | | | | I | D | E | M | W |



When to Use NOP for Load Use Hazard

*If ((EX stage RD = DEC stage RS1 OR EX stage RD = DEC stage RS2)
AND EX stage MemRead = 1)
-> Stall and insert bubble*

Note: then can use forwarding from load's writeback stage

Pipelining Example 1

addi t1, zero, 5

lw t1, 3(t1)

addi a0, t1, 5

Value at mem[8] = 10, value at mem[16] = 20

t1 originally = 13

What is value of a0, **1)** single cycle, **2)** pipelining w/o forwarding or stalling?

3) Where do we need to forward or insert stall?

Pipelining Example 1

addi t1, zero, 5

lw t1, 3(t1)

addi a0, t1, 5

Value at mem[8] = 10, value at mem[16] = 20

t1 originally = 13

What is value of a0, **1)** single cycle, **2)** pipelining w/o forwarding or stalling?

3) Where do we need to forward or insert stall?

Answer:

1) Single cycle: a0 = 15

2) Pipelining w/o forwarding or stalling: 18

October 20, 2022

Pipelining Example 1

addi t1, zero, 5


lw t1, 3(t1)

addi a0, t1, 5

Value at mem[8] = 10, value at mem[16] = 20

t1 originally = 13

New value of t1 is not written back in time for lw to use



| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|
| addi | I | D | E | M | W | | |
| lw | | I | D | E | M | W | |
| addi | | | I | D | E | M | W |

Pipelining Example 1

addi t1, zero, 5

lw t1, 3(t1)

addi a0, t1, 5

Value at mem[8] = 10, value at mem[16] = 20

t1 originally = 13

load instruction has not written back either

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|
| addi | I | D | E | M | W | | |
| lw | | I | D | E | M | W | |
| addi | | | I | D | E | M | W |

Pipelining Example 1

addi t1, zero, 5

lw t1, 3(t1)

addi a0, t1, 5

Value at mem[8] = 10, value at mem[16] = 20

t1 originally = 13

Forward from 1st addi to lw (MEM to EX).

Forward from lw to 2nd addi (WB to EX).

Need nop after
lw as value not
read from
mem in time
for decode
stage

| Instr | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|---|---|
| addi | I | D | E | M | W | | | |
| lw | | I | D | E | M | W | | |
| nop | | | - | - | - | - | - | - |
| addi | | | | I | D | E | M | E |

Pipelining Example 2

`addi t1, zero, 2`

`sw zero, 10(t1)`

`lw a0, 10(t1)`

Value at `mem[12]` is initially 1.

In a pipelined processor (with forwarding and load use hazard nop), does `a0 = 0`?

Pipelining Example 2

```
addi t1, zero, 2
```

```
sw zero, 10(t1)
```

```
lw a0, 10(t1)
```

Value at mem[12] is initially 1.

In a pipelined processor, does $a0 = 0$?

Answer:

Yes! The result of the addi in t1 gets forwarded to both the sw and lw. Even though the lw is right after the sw, since each stage and instruction is in its own stage, the store finishes in the mem stage before the load enters the mem stage

Pipelining Example 3

addi t1, zero, 0

lw t1, 0(t1)

sw t1, 8(t1)

lw t1, 8(t1)

Value at mem[0] = 8, mem[8] = 4, mem[16] = 2, t1 = 16

1) What is happening in single cycle?

What is happening in pipeline **2)** without forwarding **3)** with forwarding **4)** w/ forwarding, and nop after lw inserted?

Pipelining Example 3

addi t1, zero, 0

lw t1, 0(t1)

sw t1, 8(t1)

lw t1, 8(t1)

Value at mem[0] = 8, mem[8] = 4, mem[16] = 2, t1 = 16

1) What is happening in single cycle?

What is happening in pipeline **2)** without forwarding **3)** with forwarding **4)** w/ forwarding, and nop after lw inserted?

Single Cycle: t1 = 8

Pipelining Example 3

addi t1, zero, 0

lw t1, 0(t1)

sw t1, 8(t1)

lw t1, 8(t1)

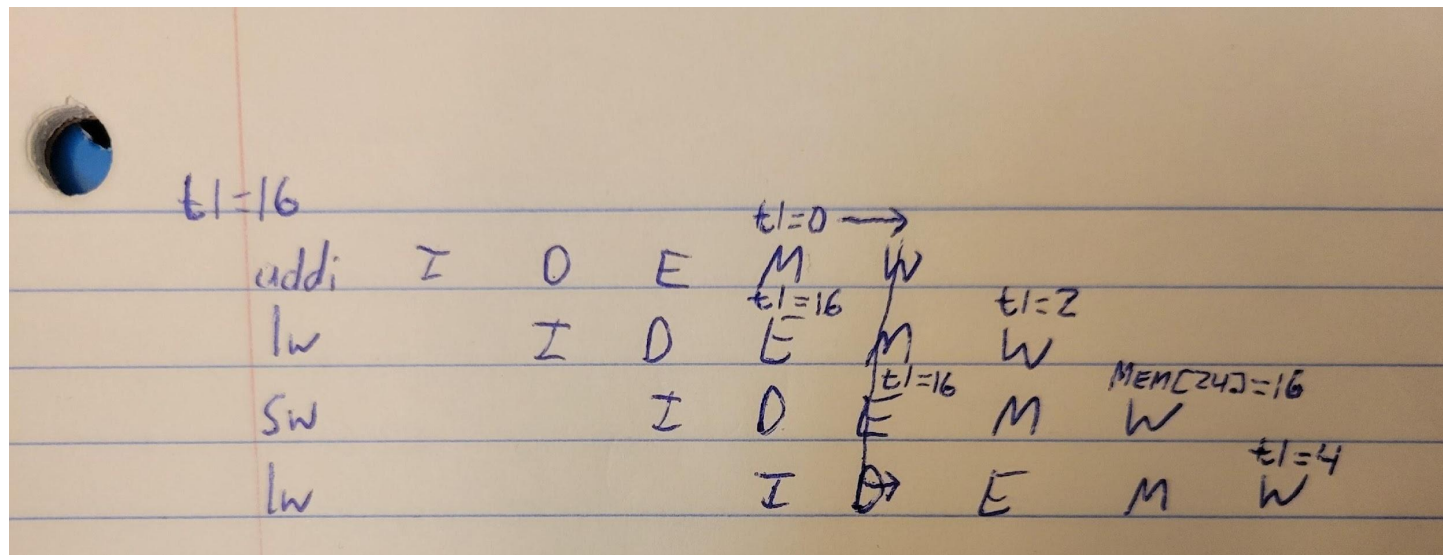
Value at mem[0] = 8, mem[8] = 4, mem[16] = 2, t1 = 16

1) What is happening in single cycle?

What is happening in pipeline **2)** without forwarding **3)** with forwarding **4)** w/ forwarding, and nop after lw inserted?

Pipelining without forwarding: t1 = 4

Pipelining Example 3: Pipelining w/o Forwarding



Pipelining Example 3

addi t1, zero, 0

lw t1, 0(t1)

sw t1, 8(t1)

lw t1, 8(t1)

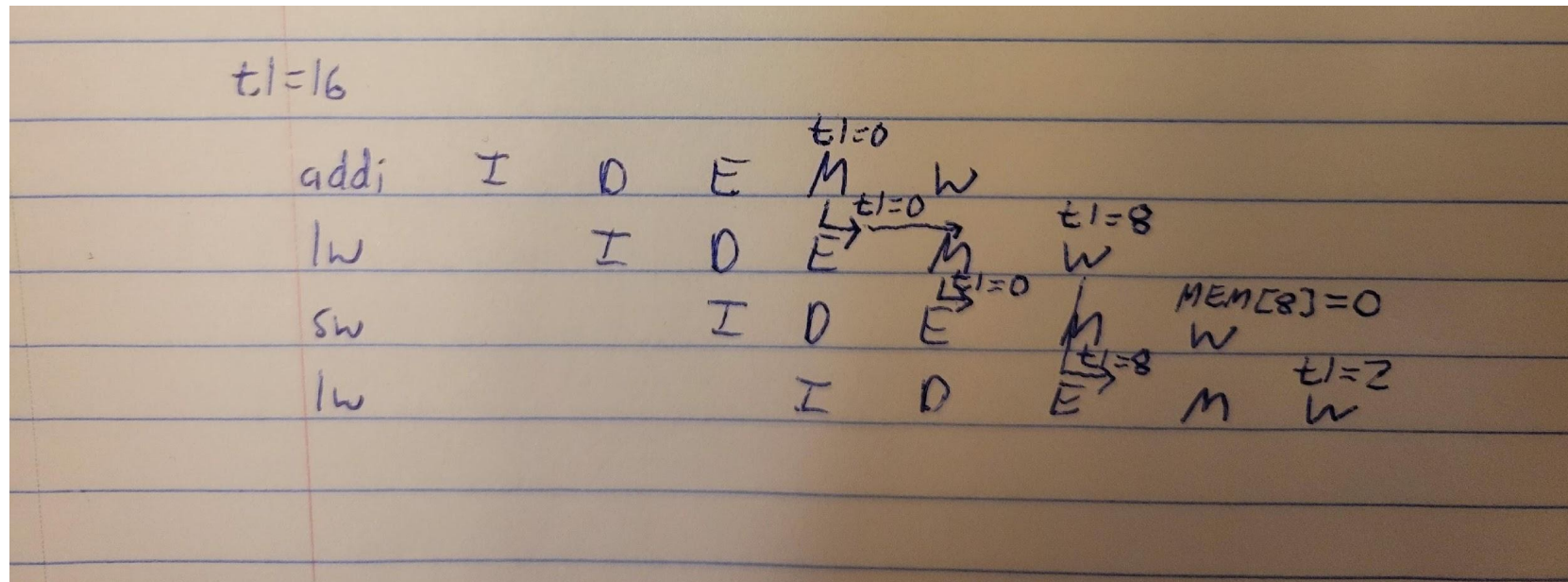
Value at mem[0] = 8, mem[8] = 4, mem[16] = 2, t1 = 16

1) What is happening in single cycle?

What is happening in pipeline **2)** without forwarding **3)** with forwarding **4)** w/ forwarding, and nop after lw inserted?

Pipelining with forwarding: t1 = 2

Pipelining Example 3: Pipelining w/ Forwarding



Pipelining Example 3

addi t1, zero, 0

lw t1, 0(t1)

sw t1, 8(t1)

lw t1, 8(t1)

Value at mem[0] = 8, mem[8] = 4, mem[16] = 2, t1 = 16

1) What is happening in single cycle?

What is happening in pipeline **2)** without forwarding **3)** with forwarding **4)** w/ forwarding, and nop after lw inserted?

Pipelining with nop after lw: t1 = 8

Control Hazards: Branches

When

-To know when the instruction is a branch (end of DEC stage -> the controller)

Where

-To know where to branch to (end of EX, or beg of MEM -> PC + offset happens at end of EX)

Whether

-To know whether to branch or not (end of MEM -> result of AND gate happens at end of MEM)

Control Hazards: Solutions

Waiting/Stalling

- Generally too slow. Also too many branch instructions

Guessing

- Better in most cases. Handle mispredictions by flushes
(when predict branch not taken, but branch is indeed taken)
- Wastes 3 cycles (by time decide branch is taken in MEM stage, have already processed 3 more instructions) -> flush 3 instructions

Control Hazards: Solutions

Resolve branch early

- If resolve branch in decode stage, only waste one cycle (one extra instruction) in the case of a branch misprediction, and flush 1 instruction
- See if rs1 and rs2 are equal in the decode stage
- Add offset to pc in decode stage
- Note: need to forward rs1 and rs2 to decode stage to compensate