Midterm Review

February 6th, 2023

Quizzes

- Some of quiz questions were discussed 154B discussion sessions.

  https://jlpteaching.github.io/comparch/img/154B_discussions/154B-discussion-2.pdf#page=15

  https://jlpteaching.github.io/comparch/img/154B_discussions/154B-discussion-3.pdf#page=23

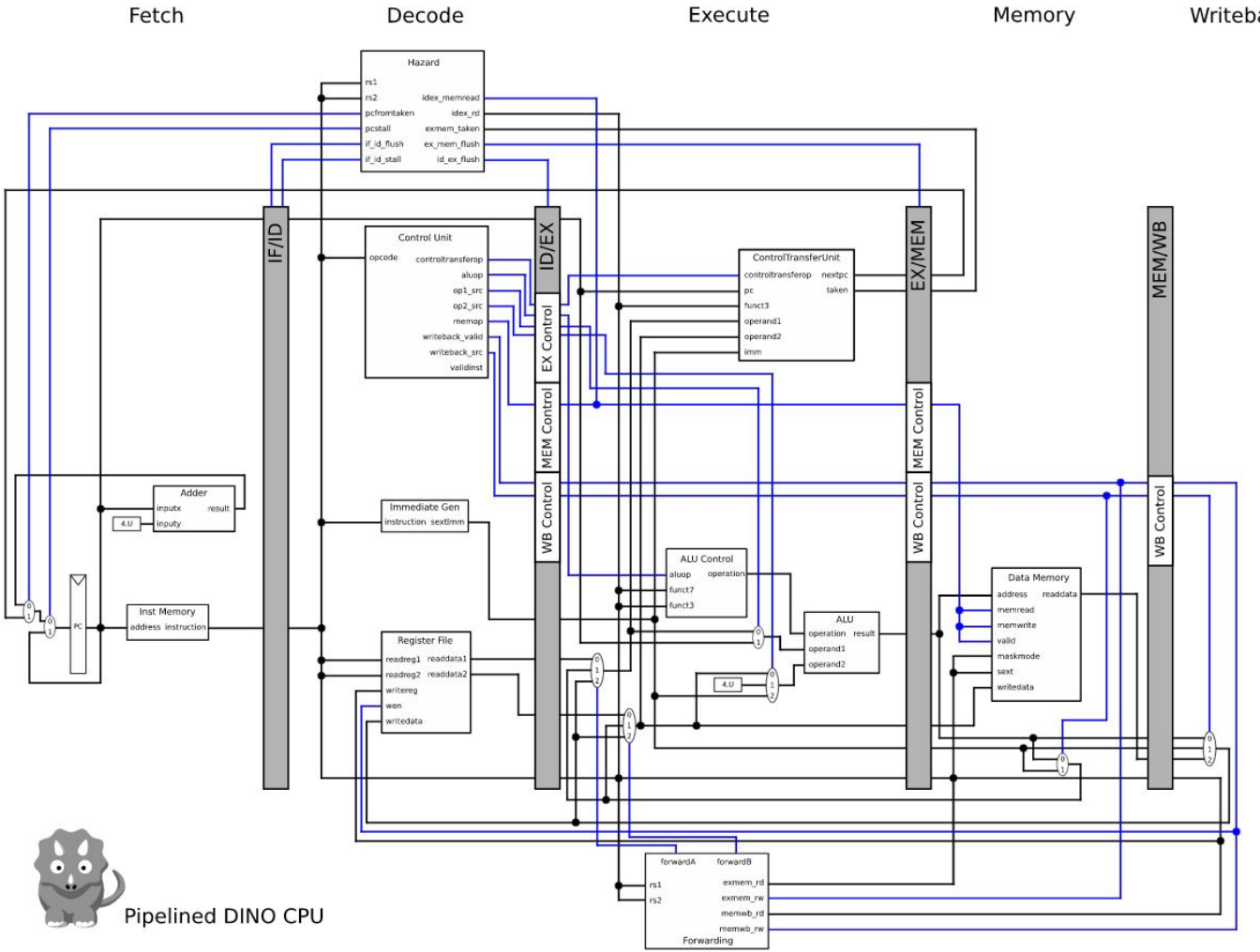
  https://jlpteaching.github.io/comparch/img/154B_discussions/154B-discussion-4.pdf#page=16

Textbook’s / DINO Pipelined CPU

- 5 stages:
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Execute (EX)
  - Memory (MEM)
  - Writeback (WB)
- Each stage only handles one instruction at a time.
- All instructions must go through all stages, unless they are flushed.

Textbook’s / DINO Pipelined CPU

- https://jlpteaching.github.io/comparch/img/dinocpu/pipelined.svg


Fetch
Decode
Execute
Memory
Writeback
Hazard
rs1
rs2
pcfromtaken
pcstall
if_id_flush
if_id_stall
idex_memread
idex_rd
exmem_taken
ex_mem_flush
id_ex_flush
IF/ID
ID/EX
EX/MEM
MEM/WB
Control Unit
opcode
controltransferop
aluop
op1_src
op2_src
memop
writeback_valid
writeback_src
validinst
EX Control
MEM Control
WB Control
ControlTransferUnit
controltransferop
nextpc
pc
taken
funct3
operand1
operand2
imm
Adder
inputx
result
inputy
4_U
Immediate Gen
instruction
sextImm
Inst Memory
address
instruction
PC
Register File
readreg1
readreg2
writereg
wen
writedata
readdata1
readdata2
ALU Control
aluop
operation
funct7
funct3
ALU
operation
result
operand1
operand2
Data Memory
address
readdata
memread
memwrite
valid
maskmode
sext
writedata
forwardA
forwardB
rs1
rs2
exmem_rd
exmem_rw
memwb_rd
memwb_rw
Forwarding
Pipelined DINO CPU

Textbook’s / DINO Pipelined CPU

- What stage does what?
  - IF: fetches instruction according to the PC register.
  - ID: decodes an instruction, fetches values of registers from the register file, generates the immediate value.
  - EX:
    - Memory instructions: calculates the effective address.
    - Branch/Jump: calculates the next pc (the result of nextpc is only available to use in Memory stage).
    - Arithmetic instructions (R-types, I-types): calculates the result of instruction.
  - MEM:
    - Memory instructions: access data memory.
  - WB:
    - Writes the result back to the register file.

Control Transfer Instructions

- Control transfer instructions
  - Branch instructions in RISC-V: BEQ, BNE, BLT, BGE, BLTU, BGEU (B prefix)
  - Jump instructions in RISC-V: JAL, JALR (J prefix)
- Branch instruction outcome is dependent on the value of registers.
- Jump instruction outcome is always taken.

Control Transfer Instructions in Pipelined CPU

- Branch/Jump instruction causes Control hazards in pipelined CPUs because they alternate the next PC to values other than PC+4.
- Without branch predictor, the pipeline has to be stalled until the control transfer instruction reaches the Memory stage.

Branch Predictor in Pipelined CPU

- Predict the branch outcome (taken or not taken) as well as the branch target (the nextpc) of all branch/jump instructions.
- Branch prediction is performed at the control transfer instruction’s Decode stage.
- The actual outcome of the branch is known at the control transfer instruction’s Memory stage.
- If the prediction is correct, no flush happens.
- If the prediction is incorrect, the instructions at Execute, Decode, Fetch stages are not supposed to be in the pipeline. They should be flushed.
- Flush: replacing an instruction with a NOP.
- NOP: an instruction that does nothing.

Example: Pipelined CPU without a Branch Predictor

0x10 + 0x14 = 0x24

Cycle	1	2	3	4	5	6	7	8	9	10	11
PC=0x10 bne t1, t2, 0x14	IF	ID	EX	MEM	WB						
PC=0x14 add t5, t3, t4		IF	IF	IF							
PC=0x18 xor t6, t3, t4											
PC=0x1c sub t1, t2, t3											
PC=0x20 and t2, t3, t3											
PC=0x24 or t4, t1, t6					IF	ID	EX	MEM	WB		
PC=0x28 mul t1, t2, t3						IF	ID	EX	MEM	WB	

Example: Pipelined CPU with a Branch Predictor

Cycle	1	2	3	4	5	6	7	8	9	10	11
PC=0x10 bne t1, t2, 0x14	IF	ID	EX	MEM	WB						
PC=0x14 add t5, t3, t4		IF	ID	EX	Flushed						
PC=0x18 xor t6, t3, t4			IF	ID	Flushed						
PC=0x1c sub t1, t2, t3				IF	Flushed						
PC=0x20 and t2, t3, t3											
PC=0x24 or t4, t1, t6					IF	ID	EX	MEM	WB		
PC=0x28 mul t1, t2, t3						IF	ID	EX	MEM	WB	

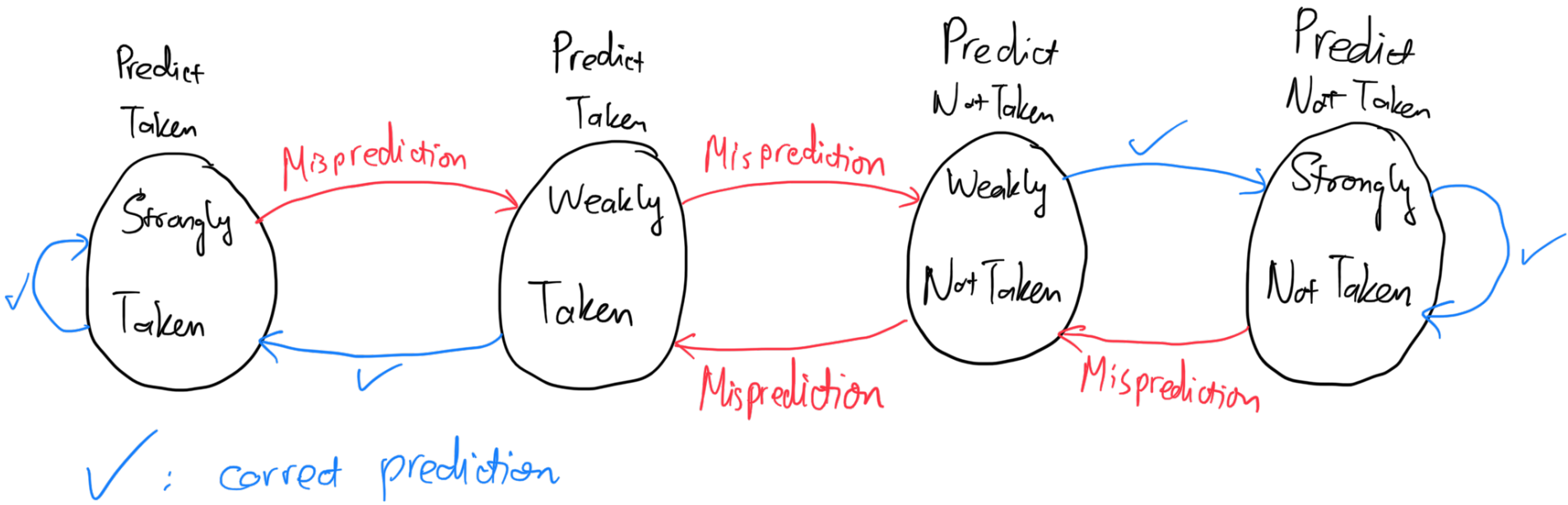
Example: Pipeline-centric view of instructions

bp: predicted not taken
outcome: taken

```
Cycle
PC=0x10 bne t1, t2, 0x14
PC=0x14 add t5, t3, t4
PC=0x18 xor t6, t3, t4
PC=0x1c sub t1, t2, t3
PC=0x20 and t2, t3, t3
PC=0x24 or  t4, t1, t6
PC=0x28 mul t1, t2, t3
```

	IF	ID	EX	MEM	WB
Cycle=1	bne				
Cycle=2	add	bne			
Cycle=3	xor	add	bne		
Cycle=4	sub	xor	add	bne	
Cycle=5	or	nop	nop	nop	bne

2-bit saturating counter

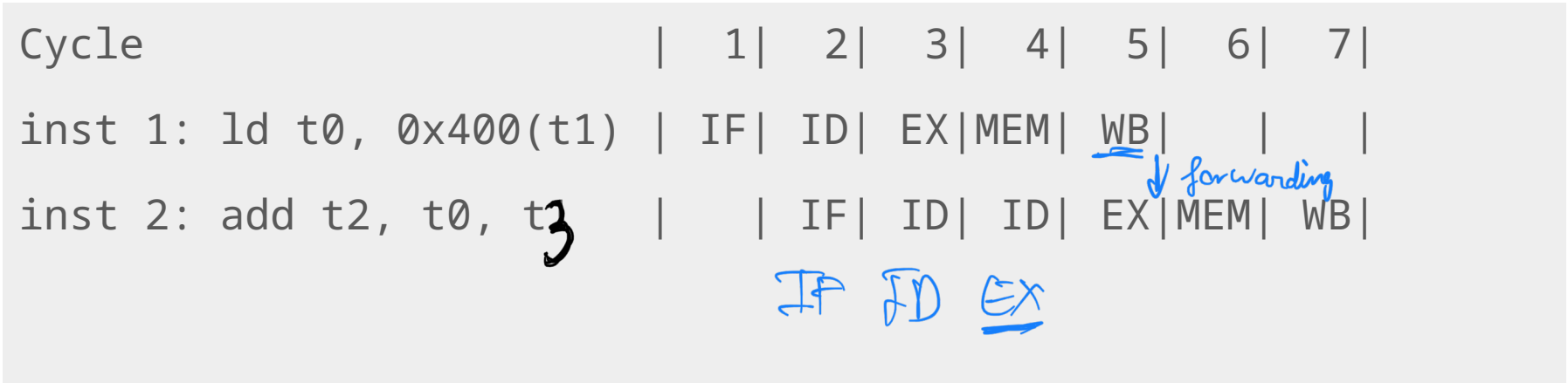


Data Hazards

- Data dependencies *among registers*.
- RAW: read-after-write (true dependency), mitigation: forwarding/stalling
- WAR: write-after-read (anti dependency), mitigation: reg-renaming/stalling
- WAW: write-after-write (anti dependency), mitigation: reg-renaming/stalling

Data Hazards

- In the 5-stage pipeline, we also have load-to-use hazard.
  - The value loaded from memory is not available to use until the write stage.
  - The next instruction reads the loaded value.
  - Mitigation: inserting a bubble after the load instruction.



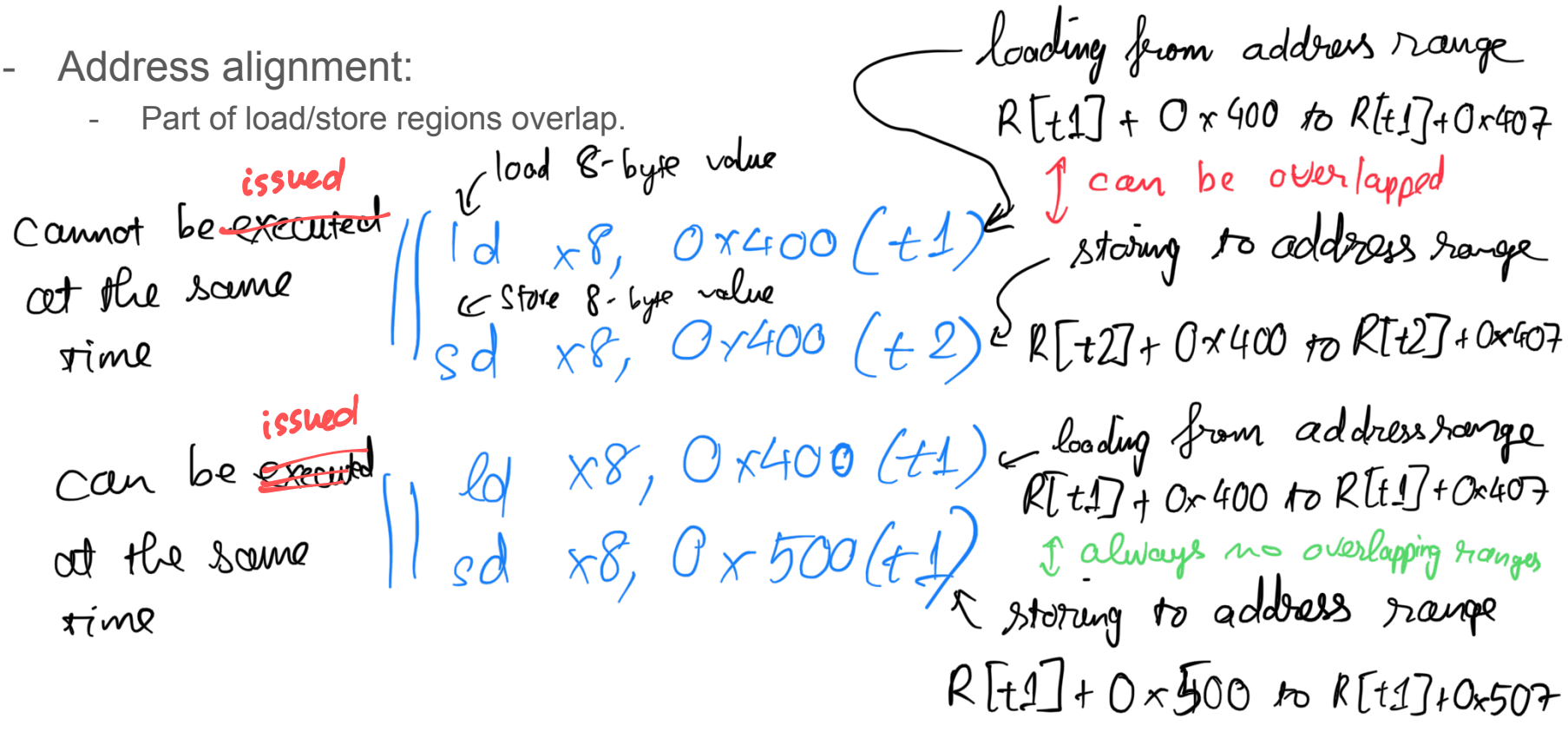
Data Hazards: Dynamic Data Dependency

- Assume a pipeline that performs effective address and memory access all in Execute stage. Assume that the order of execution in the Execute stage is out-of-order.
- Each memory instruction requires calculating effective address before accessing memory.
- We can issue a memory instruction to the Execute stage if we issue the instruction and following conditions do not occur in the Execute stage,
  - There's a pair of a store instruction and another memory instruction (load/store) that we don't know whether their effective addresses align.
  - Essentially, we are avoiding RAW, WAR, WAW hazards when accessing memory.

→ Cannot issue a store instruction with other memory instructions in the same cycle if the memory alignment is unknown. It's okay to have them in the Execute stage at the same time, however.

Data Hazards: Dynamic Data Dependency

- Address alignment:
  - Part of load/store regions overlap.



Forwarding

- In 5-stage pipeline:
  - Forward from MEM to EX stage.
  - Forward from WB to EX stage.
  - Note: There's **no** forwarding from WB stage to MEM stage.
  - Have to stall if there's no forwarding mechanism.

- There's no intra-stage forwarding unless specified otherwise.

Quizzes

RISC-V instruction formats:

rd : destination register
rs1, rs2 : source registers
imm : immediate

Arithmetics: (R-types, I types, LUI, AUIPC):

(R) add / or / ... rd, rs1, rs2 lui / auipc rd, imm

(I) addi / ori / ... rd, rs1, imm

Jumps / Branches: (J-types, B-types) Branches: blt rs1, rs2, imm

Jumps: jal rd, imm

jalr rd, rs1, imm

Memory (Loads / Stores) loads → ld, lw, lh, lb | ld rd, imm(rs1)

stores → sd, sw, sh, sb | sd rs2, imm(rs1)

↳ Store insts do not write to reg

Quizzes

e.g. average speed up

samples	geomean	arth mean
$a, 4a$	$2a$	$2.5a$
$a, \frac{1}{4}a$	$\frac{1}{2}a$	$0.625a$