

Pipelining (Hazards , Exceptions)

CIT 595
Spring 2008

Pipeline Hazards

- There are three kinds of pipeline hazards:
 1. Structural Hazard
 2. Data Hazard
 3. Control Hazard

CIT 595

2

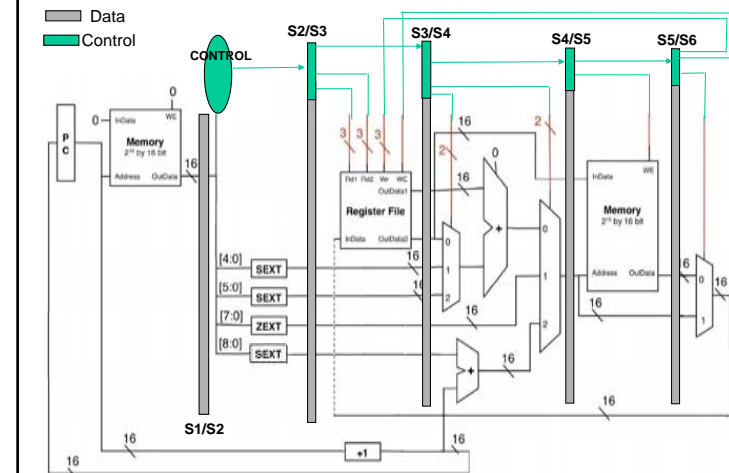
Structural Hazard

- Occurs when hardware cannot support a combination of instructions that we want to execute in parallel
 - In Laundry example: the machine has combined washer or dryer
 - In instruction pipelining: shared memory resources
 - Example: One memory for instruction & data
- Usually overcome by duplicating hardware
 - Memory is separated into instruction and data memory
 - Or memory/register is multi-ported i.e. memory that provides more than one access path to its contents

CIT 595

3

LC3 Pipelined Implementation



CIT 595

4

Data Hazard

- Occurs when an instruction depends on the results of a instruction still in the pipeline

- Example 1:

i1: ADD R1, R2, R3
i2: AND R5, R1, R4

- Example 2:

i1: ADD R1, R2, R3
i2: ST R1, A

- Example 3:

i1: LD R1, A
i2: ADD R2, R1, R2

CIT 595

5

Data Hazard: Example 1

i1: ADD R1, R2, R3

i2: AND R5, R1, R4

S1: Instruction Fetch

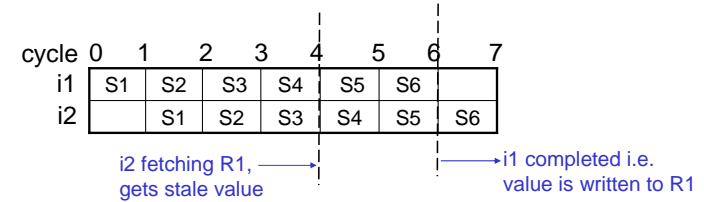
S2: Decode

S3: Register Fetch

S4: Execute/EA

S5: Memory Access (LD/ST)

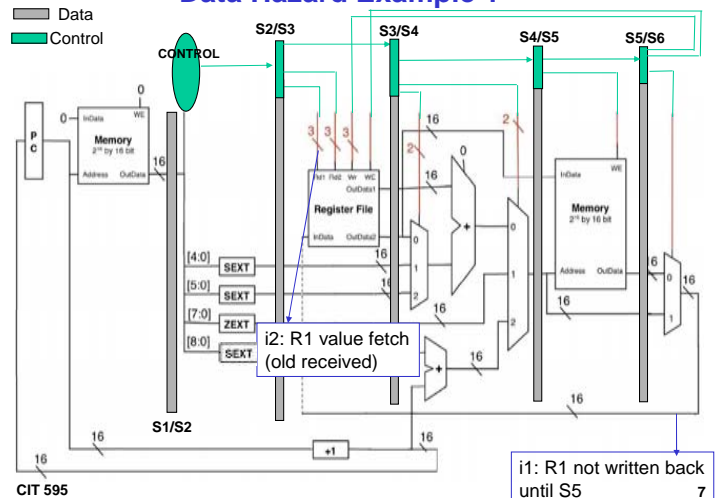
S6: Write Back (register write)



CIT 595

6

Data Hazard Example 1

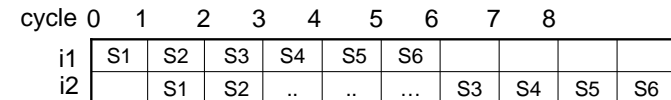


CIT 595

7

Solution to Example 1

- Naive approach, introduce **delay** in the pipeline till instruction i1 finishes
- Also stop any new instructions from being fetched
- Also known as Pipeline **Stall** or **Bubble**



S1: Instruction Fetch

S2: Decode

S3: Register Fetch

S4: Execute/Evaluate Addr

S5: Memory Access (LD/ST)

S6: Write Back (register write)

CIT 595

8

Stall requires extra circuitry

- Extra logic determines whether a hazard could/will occur
 - Hazard logic unit
- If hazard arises then
 - Generates control such that next instruction will not be fetched
 - E.g. Disable write to register S1/S2
 - Suspends the instruction that will cause the hazard
 - Based on pipeline stages the delay can be determined
 - Instead of instruction moving forward, a No-operation (NOP) is carried out such that nothing will happen in stages

CIT 595

9

Solution to Example 1 (contd..)

- Better Solution: **Data Forwarding**
 - Realize that data value from i1 (to be put in R1) is actually available at end of cycle 4
 - Don't need to wait till S7 to fetch the register file, instead forward a copy of the data from S4 of i1 to i2's stage S4

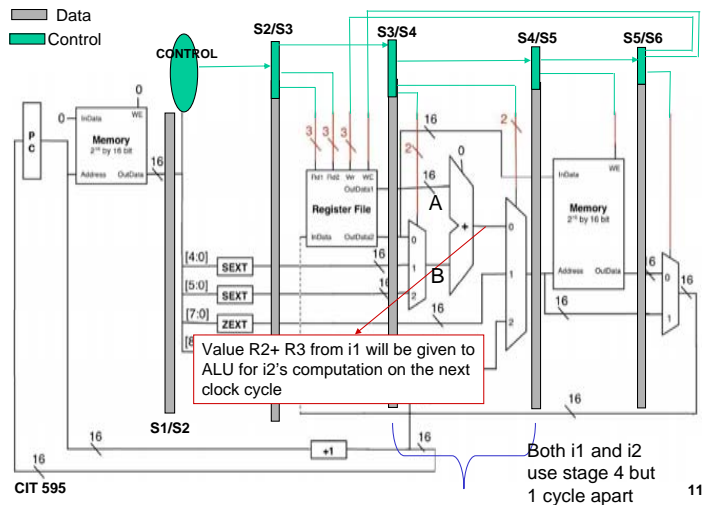
cycle	0	1	2	3	4	5	6	7
i1	S1	S2	S3	S4	S5	S6		
i2		S1	S2	S3	S4	S5	S6	

- S1: Instruction Fetch
- S2: Decode
- S3: Register Fetch
- S4: Execute/EA
- S5: Memory Access (LD/ST)
- S6: Write Back (register write)

CIT 595

10

Data Hazard Example 1: Forwarding



CIT 595

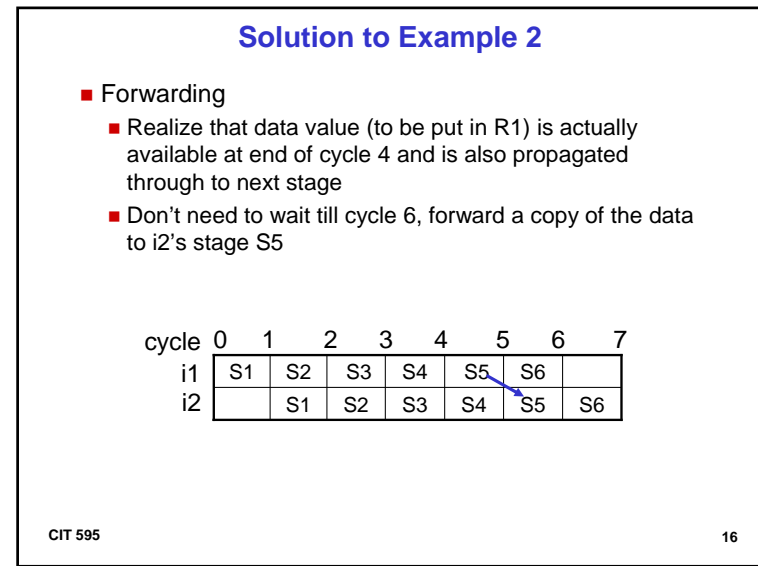
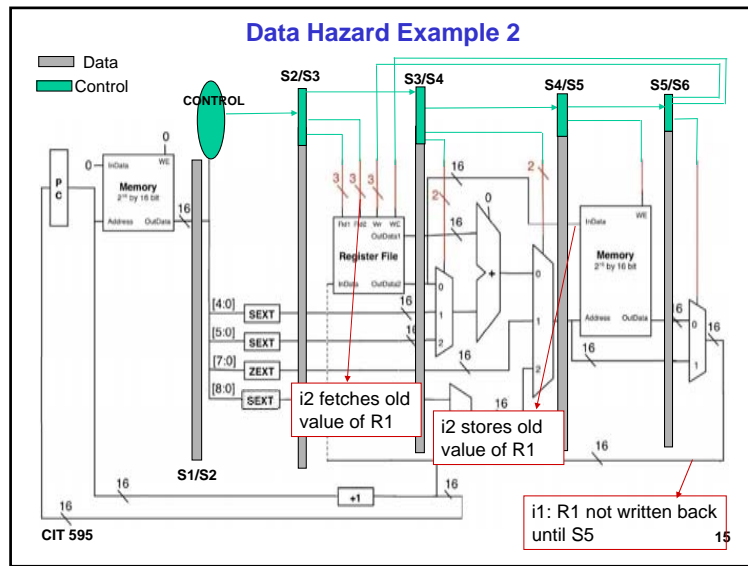
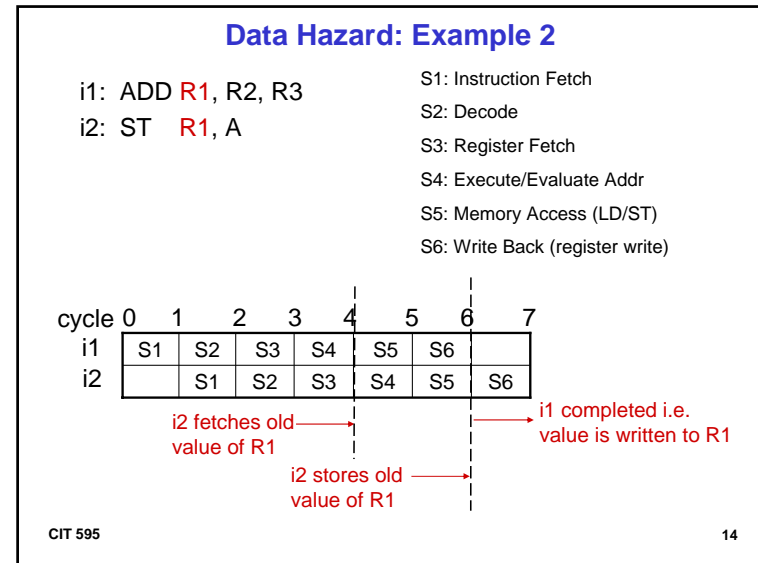
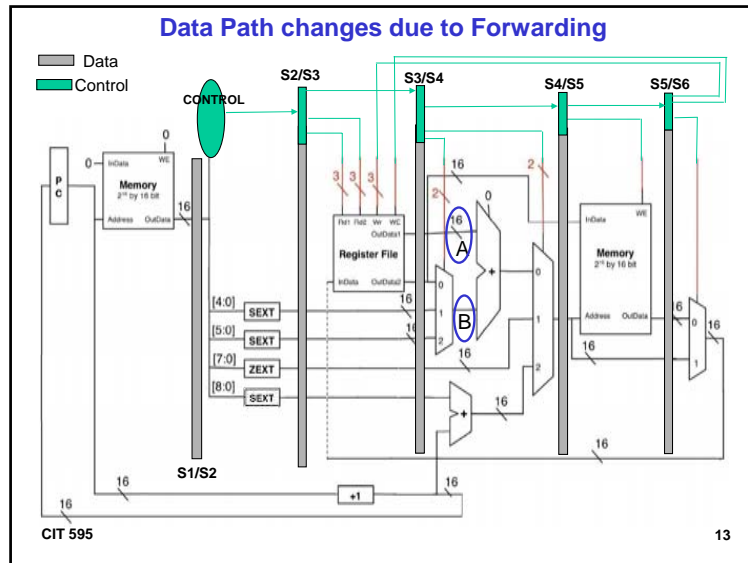
11

Handling Data Forwarding

- Requires additional logic to data path
 - Forward Logic Unit**
- Additional MUX needs to be placed before inputs to ALU
- Forward Logic Unit will set the appropriate MUX control based inputs
 - Rd1 and Rd2 from Decode stage (dependent instruction just decoded)
 - Control info (Rd1, Rd2, Dst) for each instruction ahead that is moved along the pipe

CIT 595

12



Assumptions about load use Hazards

- Assuming that memory will read out data in single cycle
 - This only possible introduction of caches
 - Cache store immediate content needed
- If data is not found in cache then, we need to access main memory
 - If not found in main memory then access the disk
- More on this in chapter 6...
- Bottom line the delay can be more than 1 cycle

CIT 595

21

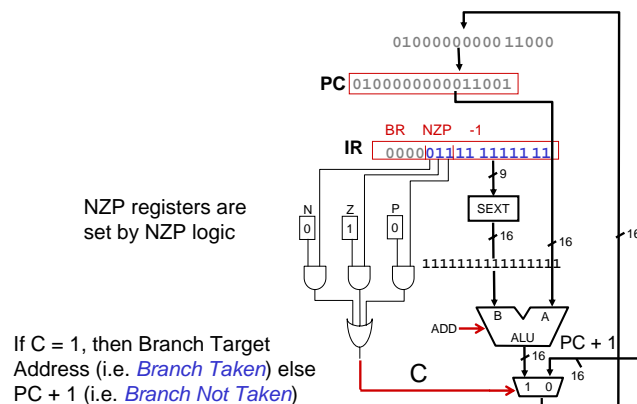
Control Hazards

- Occurs when we need to make a decision based on the result of instruction while others are executing
- Branch Instructions are instructions that make decision
 - Alter the flow of execution in a program and give rise to control hazard
- Note: LC3 Pipelined will have to be modified for BR/JMP

CIT 595

22

Recap: Branch Instruction



CIT 595

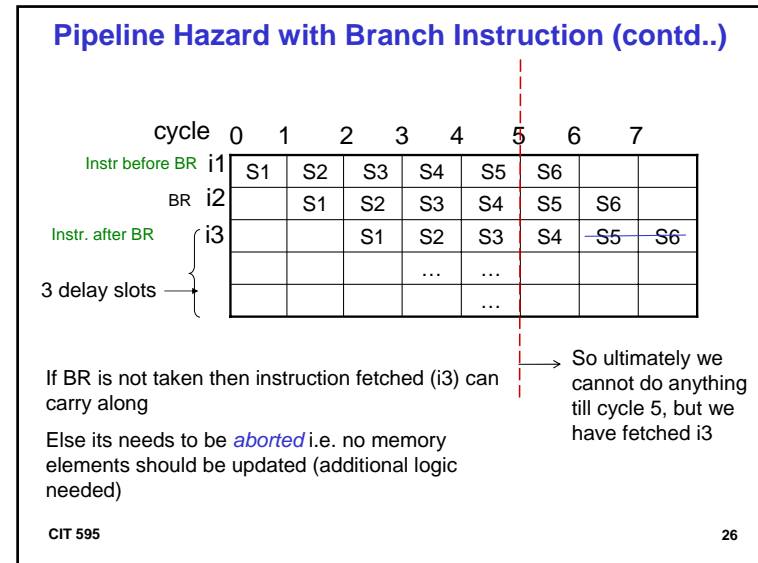
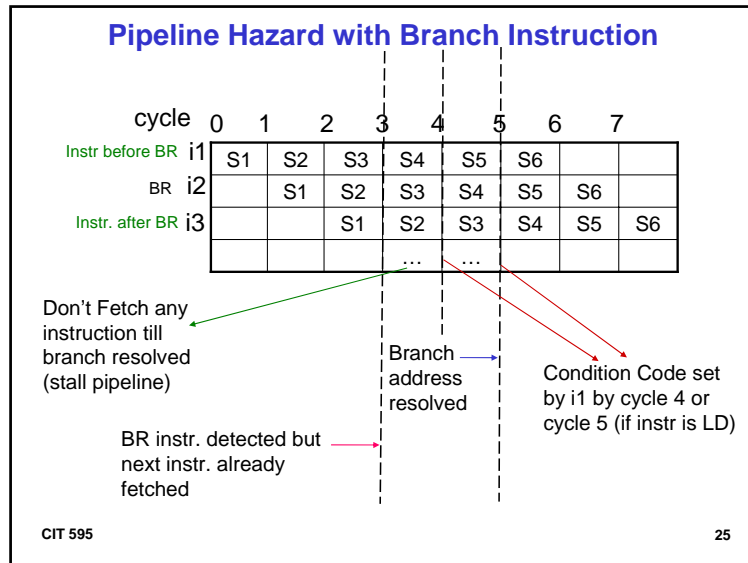
23

Control Hazard

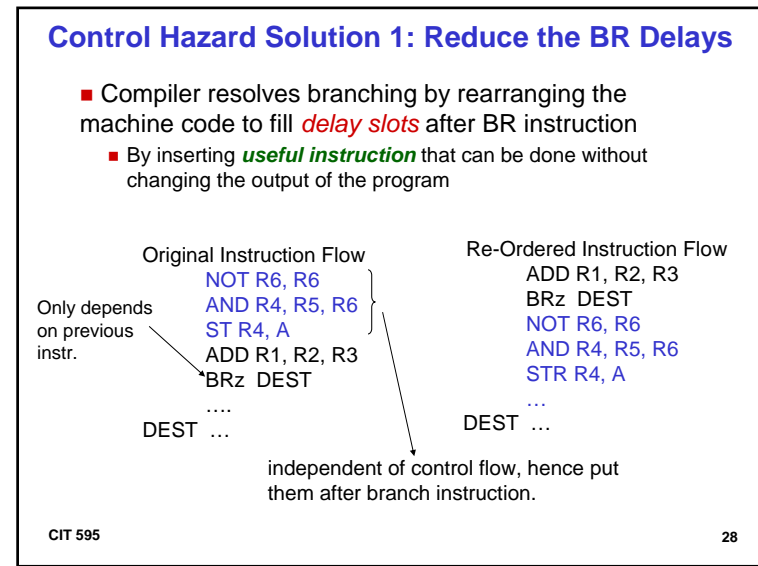
1. Branch Instr only known after *Decode* Stage (S2)
 - By then we have already fetched the next sequential instr.
2. Branch address is resolved only in the *Evaluate Address* phase (S4)
 - Stall the pipeline till we know which address to fetch from i.e. PC + 1 or Branch Target address
3. The instr. before the branch instr. will set *Condition Code* (NZP) one cycle before or the same cycle the branch address is resolved

CIT 595

24



- ### Branch Instruction Impact CPI
- Hazards increase the # cycles/instr. in pipelined implementation
 - Structural and Data Hazard effects can be minimized
 - However, branch hazards cannot be minimized because we have to wait for following information
 - Branch address (tells us where to branch)
 - Condition Code from the previous instruction (tells us whether to branch or not)
 - Most ISAs have some sort of pipelined implementation
 - Many techniques have been studied to *reduce branch delays*
- CIT 595 27



Control Hazard Solution 1: Reduce the BR Delays

- Can only fill delay slots as long there are independent instructions
- Otherwise, compiler will insert **NOP** instruction to keep the pipeline full
 - ISAs provide special NOP instructions to insert delay
 - NOP instructions have all zeros in their bit fields
 - E.g. NOP opcode in LC3 would 0000 be (bits [15:12])

```

ADD R1, R2, R3
BRz DEST
NOP
NOP
NOP
...
DEST ...
    
```

CIT 595

29

Control Hazard Solution 2: Prediction

- Many ISAs also often **predict** the outcome of the branch
 - In these ISAs address calculation of branch is coupled in the same phase as the branch is discovered (i.e. moved from S4 to S2)
 - There is **prediction unit** that records history of branch pattern
 - Once the branch instruction is discovered, the prediction unit guides processor which instructions to fetch next
 - The prediction unit is also updated with the actual outcome

CIT 595

30

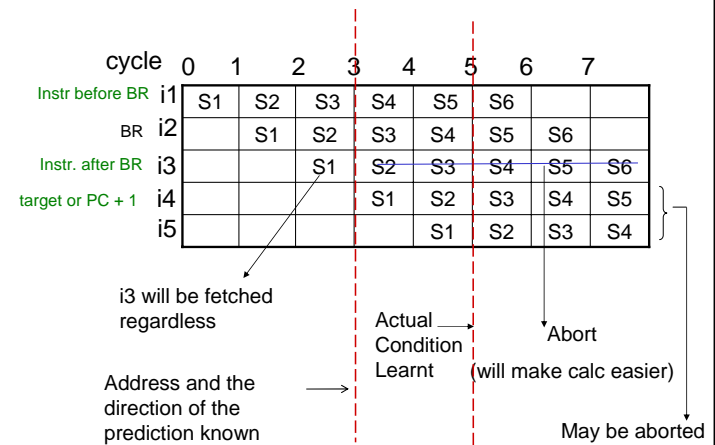
Using Prediction Unit

- If prediction is **Taken**
 - Instruction that are fetched for the delay slot are from the **target address path** (also calculated in the same stage as BR discovered)
- If prediction is **Not Taken**
 - Instructions that are fetched for the delay slot are from **PC + 1** path
- If **actual result = prediction**, don't do anything i.e. continue the processing
 - Else need to abort the fetched instruction and restart
- Also update the prediction unit with the actual result
 - i.e. out come the current branch instruction (for future branches)

CIT 595

31

Pipelining with Branch Prediction



CIT 595

32

Types of Prediction

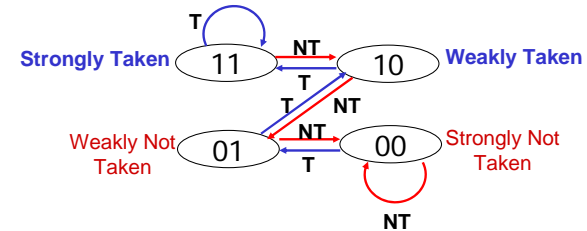
- **Static (wired/fixed)**
 - Always guess “taken” or “not taken”
 - Effective only with loop control structure
- **Dynamic**
 - Another hardware in the datapath keeps tracks of the branch history as the instructions are executing
 - E.g. 2-bit branch predictor using saturating counters

CIT 595

33

Example: Two-bit Branch Predictor

- Keep 2-bit history value for each “recent” BR instruction
- Use 2-bit saturating counter
 - > If branch is actually Taken (T), increment the history value
 - > If Not Taken (NT), decrement the history value
 - > 00 (Strongly NT), 01 (Weakly NT), 10 (Weakly T), 11 (Strongly T)



CIT 595

Typically > 90 % correct predictions

34

Prediction helps reduce impact on CPI

- Each branch instruction takes 1 cycle to complete + additional cycles (due to branch delays caused)
- Lets say 20% instructions are branches
- Assume that branch predictor is 90% accurate

■ Pipeline without Prediction

$$\begin{aligned} \text{CPI}_{\text{branch}} &= \text{Fraction Branches} * (1 + \text{Additional Cycles}) \\ &= (0.2) * (1 + 3) = 0.8 \end{aligned}$$

■ Pipeline with Prediction

$$\begin{aligned} \text{CPI}_{\text{branch}} &= \text{Fraction Branches} * (1 + [1 + (\text{Misprediction Rate} * \text{Additional Cycles})]) \\ &= (0.2) * (2 + (1 - .90) * 2) \\ &= 0.44 \end{aligned}$$

CIT 595

35

Compiler + Prediction helps reduce impact on CPI

- Lets say 20% instructions are branches
- Assume the compiler only finds one useful instruction on average
 - Hence additional cycles needed is 2 due to NOPs

■ Pipeline with Compiler

$$\begin{aligned} \text{CPI}_{\text{branch}} &= \text{Fraction Branches} * (1 + \text{Additional Cycles}) \\ &= (0.2) * (1 + 2) = 0.6 \end{aligned}$$

- Now, if we kick in the predictor (90% accurate) for 2 NOPs

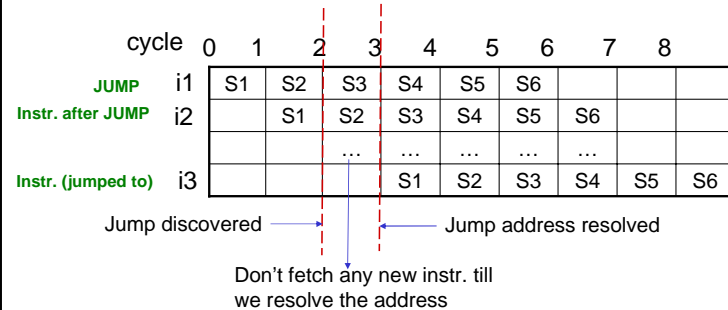
$$\begin{aligned} \text{CPI}_{\text{branch}} &= \text{Fraction Branches} * (1 + [\text{Misprediction Rate} * \text{Additional Cycles}]) \\ &= (0.2) * [1 + [(1 - .90) * 2]] \\ &= 0.24 \end{aligned}$$

CIT 595

36

JUMP Instruction is also Control Hazard

- Jumps don't have conditions i.e. jumps are unconditional branches (we will definitely go to the target address)
- Have wait till we evaluate the address i.e. read address from register file (e.g. JMP R3)



CIT 595

37

Delays due to JUMP instruction

- Standard delay of 2 cycles will be incurred
 - One instruction that will eventually be aborted and
 - One stall for not fetching next instruction after discovering Jump instruction
- Branch Prediction cannot help in this case as we are **not waiting on a condition**
- Any penalty if to be reduced will fall on compiler i.e. find useful instructions to do in the 2 delay slots

CIT 595

38

Deeper Pipelines and Misprediction Penalty

- Dividing the pipeline into even smaller stages increases frequency (i.e. lowers time/cycle)
- But deeper the pipeline, will cause branch resolution to later stages, in turn increasing the CPI due misprediction penalty
 - If we filled pipeline with instructions from the wrong path then we wasted cycles for those instructions
- Hence the performance does not scale well with deeper pipelines

CIT 595

39

Exceptions

- Exceptions are used for signaling a certain condition
- You already know
 1. I/O request: device requests attention from CPU
 2. System call or Supervisor call from software (TRAP in LC3, I/O functions in C)
 3. Arithmetic: Integer or FP, overflow, underflow, division by zero
 4. Invalid instruction
 5. Memory protection: read/write/execute forbidden on requested address
- Yet to learn (or may be heard)
 1. Page fault: requested virtual address was not present in main memory
 2. Misaligned address: bus error
 3. Hardware malfunction: component failure

CIT 595

40

Handling All other Exceptions

- Let the instruction(s) *before* exception condition instruction complete
- Abort the instructions *after* **excepting instruction**
- Save the state of the machine
- Start fetching instructions in memory where the **exception handling routine** instructions are kept
- This is known as implementing **precise exceptions**
 - i.e. undo all instructions after the excepting instructions and restart from the excepting instruction

CIT 595

41

Pipelining Complications

- Due to overlapping of instruction execution, multiple exceptions can occur in the same clock cycle

Stage	Problem Exceptions Occurring
Fetch	Memory-protection violation, Page fault on instruction fetch, misaligned memory access
Decode	Undefined Instruction
Execute	Arithmetic exception
Memory Access	Memory-protection violation, Page fault on instruction fetch, misaligned memory access

CIT 595

42

Pipeline Complication: Example



- The second instruction (ADD) produces an exception first, and then the first (LDR) instruction is restarted, then second instruction is executed twice!!!

CIT 595

43

Solution to Multiple Exceptions in a Pipeline

- Maintain exception vector for each instruction
 - Some hardware logic handles (similar to how NZP registers are maintained)
 - Vector is nothing bit a n-bit register and each bit position indicating a stage of the pipeline
 - To indicate exception set the appropriate bit position
- When instruction enters the last stage of the pipeline, check the exception vector, and handle the exceptions in instruction order
 - If the bit is set means that the instruction had faulted
 - Abort all instructions following this instruction
 - Save the state of the machine
 - Branch to appropriate service routine

CIT 595

44

Summary of Pipelining

- Improves performance
 - Improves runtime of program
 - Reducing the clock cycle time
 - Increase Frequency (faster processing)
 - CPI = 1 (ideal)
 - Speedup = #number of pipe stages (ideal)

- However comes at price of greater CPI penalties
 - Data Hazard (Load-use delay)
 - Control Hazard (Branch/Jump delays)
 - Exceptions