01/08/24

Computing systems

Servers : single / dual socket,

32-64 cores per processor

200 gb + RAM

Mobile phones : SOC (system on chip)

Embedded systems : Automotive

IoT systems : Low compute power & memory

Local Clusters : 100+ nodes

Data centers : 10,000+ nodes

Edge → on device

What are the considerations you need to have to move computation from cloud to edge.

Performance Metrics

Which metrics do we focus on when designing systems for diff. computing systems

1. Performance
   - → Latency
   - → Throughput
   - → No. of floating pt. ops per second
   - → TOPS

2] Power

3] Area

4] Performance per Watt

In this course

1] Domain specific h/w accelerators

- ML
- Bioinformatics

} Need good performance per Watt

2] Systolic arrays / spatial arch.

3] Neuromorphic accelerators

ISA : Hardware - Software interface

Some approaches, when implementing in software, you need to consider h/w changes

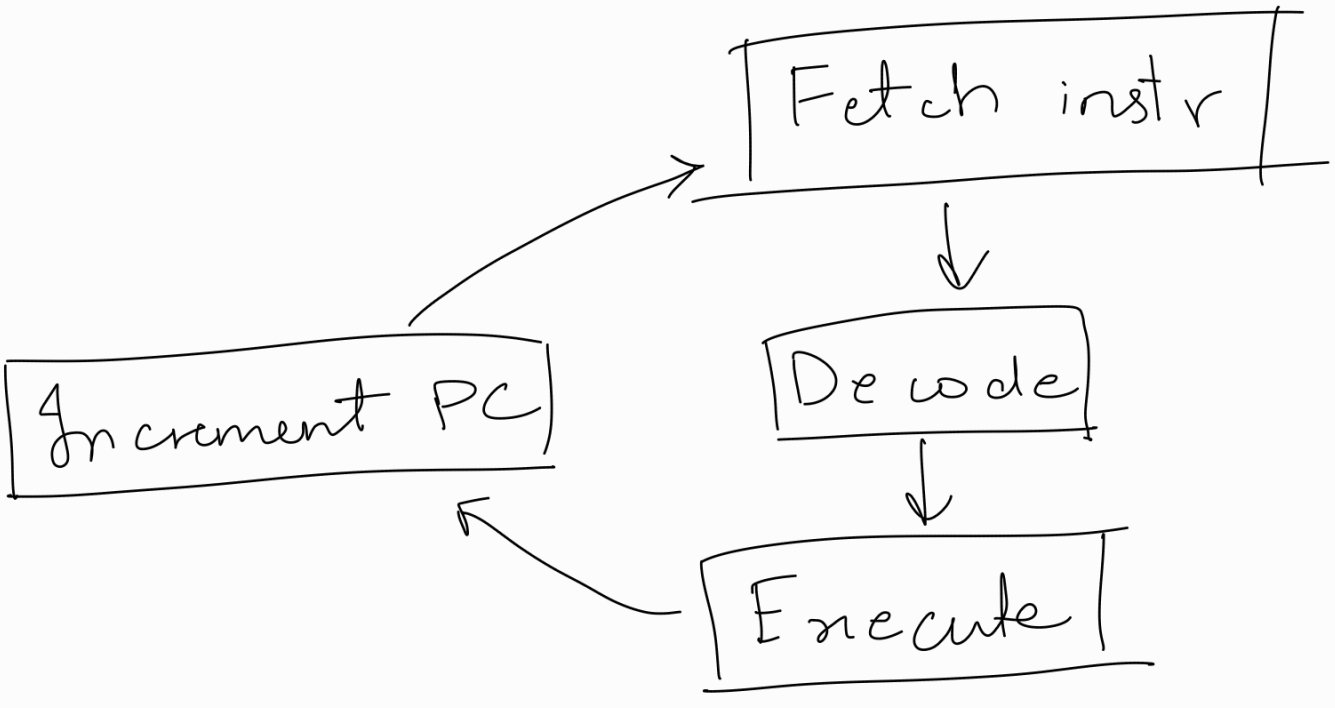
1] Locking primitives

2] Privilege levels

3] Virtual memory

4] Context switching

Von-Neuman Model



Micro arch

Cache size

Power consumption

No. of cores

RISC-V is an open-source ISA.

Project idea: Introduce extra func. in RISCV

03/08/24

Scribe notes

CISC [X86-64]

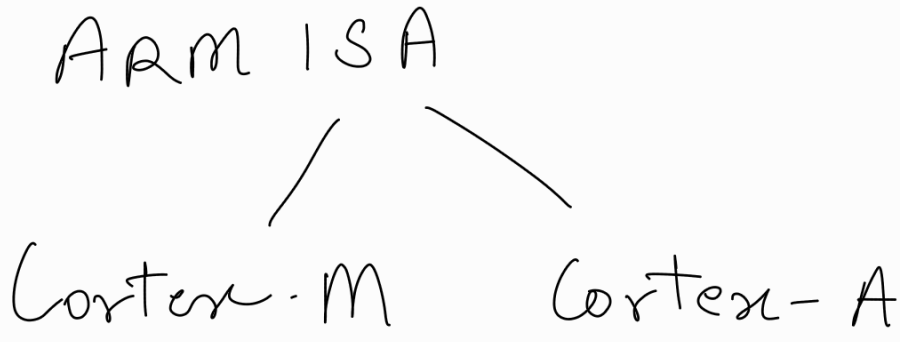
→ This made sense earlier when people wrote assembly code, but is a challenge for compilers

RISC

→ RISC 1 designed in Berkeley. Easier for compilers.

Computer Architecture: A Quantitative Approach

→ Multiple micro-architectures present for single ISA



→ Alder Lake → 2 P-cores (Hyperthreading enabled)
↑ → 6 E-cores (single core)

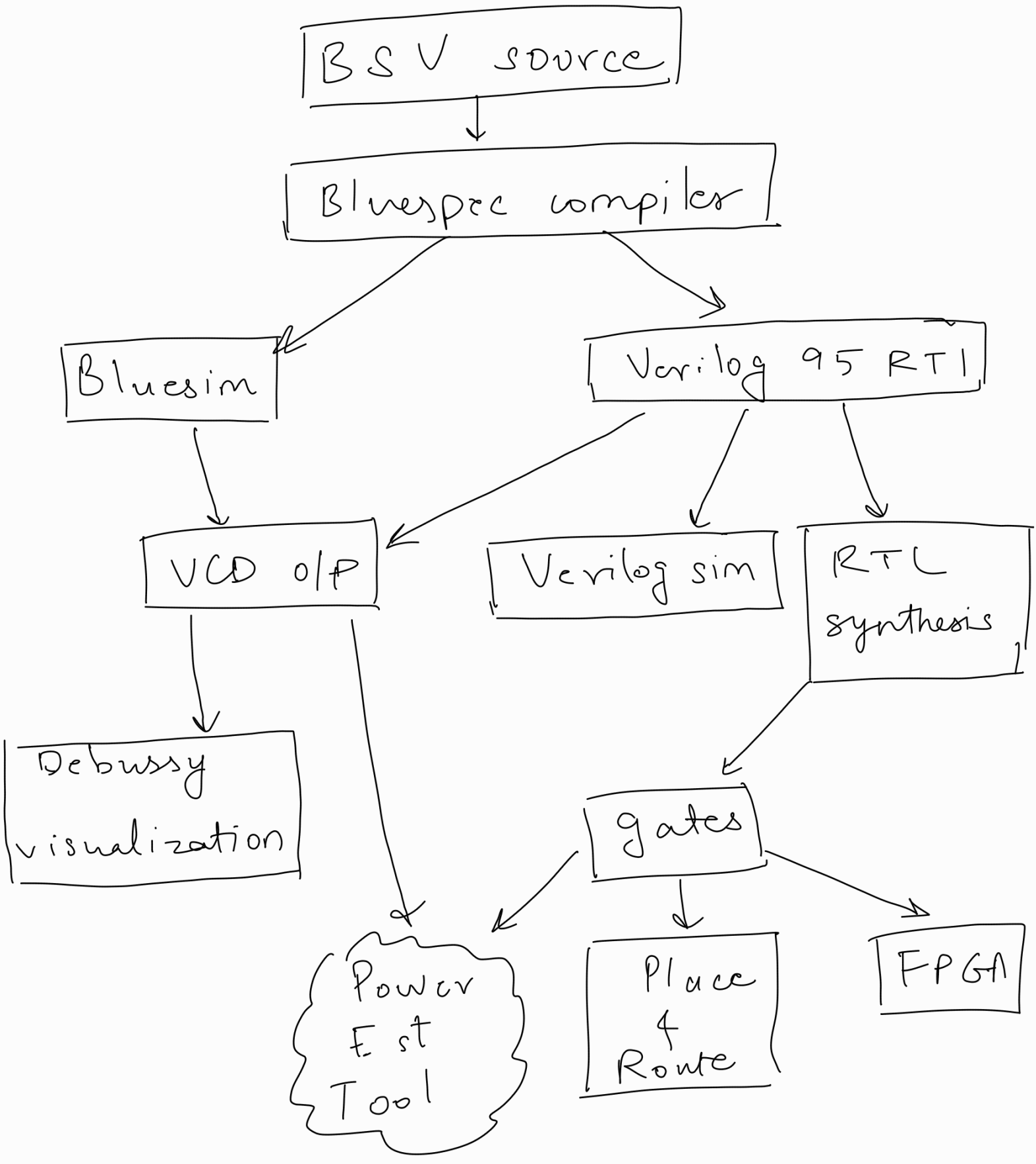
Big Little architecture

→ Design custom hardware accelerator to make working for CV systems faster ⟹ Project

→ HLS converts C to Verilog/VHDL

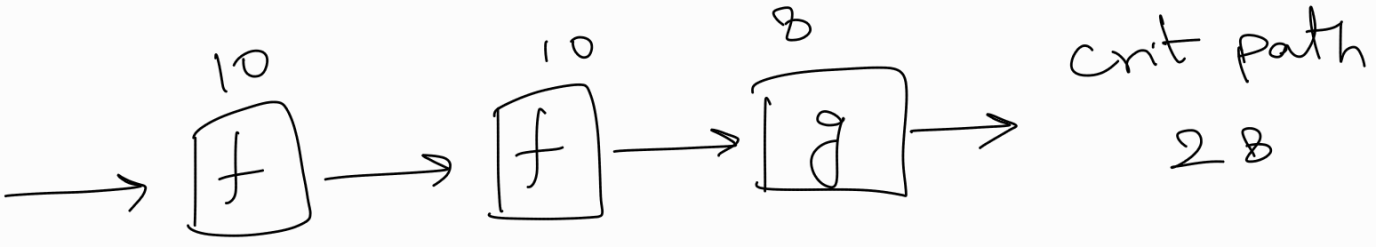
This process not advisable as C is sequential

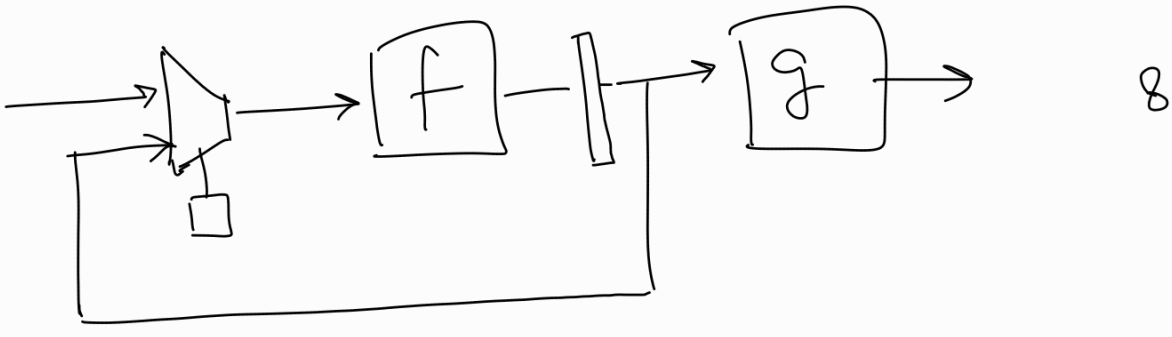
→ MIT comes up with Bluespec System Verilog (BSV).



→ Combinational vs sequential circuits

→ Repeat the permute-butterfly block in the combinational IFFT circuit.





Introduce state elements to hold intermediate values

→ As you decrease area, clock cycles increase. This is the tradeoff

→ CPI and IPC are 2 metrics

Ideal e-CPI = 1

→ Superscalar processors try to get CPI > 1

→ in & when empty, stall pipeline out &, when full, apply back-pressure

→ Clock

comb best folded = normal

Area

folded best comb pipeline

Throughput

Pipeline folded = comb

best

→ Pipelined processors are inorder

Superscalar " " out of order

→ The ability of 2 instructions to be executed in parallel (if they are independent) is called Instruction Level Parallelism (ILP)

→ More transistors reqd. for more ILP

Challenges in early 2000s

1] ILP wall: Increasing transistors now gives diminishing returns

2] Power Wall

3] Memory wall

Multi-core

→ Exploit task-level parallelism instead of ILP

→ Power wall is contained.

3 types of instructions:

ADD R1, R2, R3

ADDI R1, R2, 100

ADD R1, R2, [0XFF]

This makes designing pipeline hard as different instructions require diff. number of clock cycles.

→ In instruction fetch unit, the address bus is 30 bits. The address of an instruction has to be divisble by 4. Hence, the 2 LSBs will always be 0. So no point in storing them.

Calculate critical path length for add.

Show control signals for ADDI, ORI

	ADD	ADDI	ORI	LW	SW	BEQ	Jmp
Br	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	X	1	0	1	1	X	
ALUSrc	0	1	1	1	1	0	
ALUCtr	ADD	ADD	OR	ADD	ADD	SUB	
MemWr	0	0	0	0	1	0	
MemtoReg	0	0	0	1	X	X	
RegDst	1	0	0	0	X	X	
RegWr	1	1	1	1	0	0	

Outputs of ALU

N - Set to one when o/p is -ve

Z - Set to " " " " 0

C - " " " " overflow

V - Signed int overflow

Critical Path

t_{cq} ← Clock to queue time + inst. mem delay + max (control logic delay, RF delay) + ALU delay + t_{su}

t_{su} ← setup time

5/8/24

RISC V

→ 2 types of instructions:

1] User Level

2] Privileged Level

→ 3 modes

1] User 2] Supervisor 3] Machine

→ 3 architectures

32, 64, 128-bit

What does the 32-bit size indicate?

→ Register width

→ Address space = 2^{32} bytes $[0, 2^{32}-1]$

RV32 I → Integer instructions

RV32 M → Multiplication, division

RV32 A → Atomics (semaphores, etc)

RV32 F/D → Float, double

RV32 IMAFD

G ← General

One of the main issues of RISC ISA is the large no. of instructions needed to perform any operation due to the simple instructions. Compressed instructions will play a big role in

freeing up some memory

Core Integer Instructions

1] Data transfer (Load/store)

2] Arithmetic & logic (ADD, AND)

3] Control transfer (JMP)

Program Exec. Time

= Dynamic instr. cnt x CPI

x clock cycle time

26/8/24

Consider raw compute power of my processor is 200 GFlops/sec

Requirement is 1000 images/ms

Current is 90 images/ms

What to do?

→ Compute GFlops/s for processing 1000 img/ms. If that is > 200GF/s it is not feasible. Get new processor.

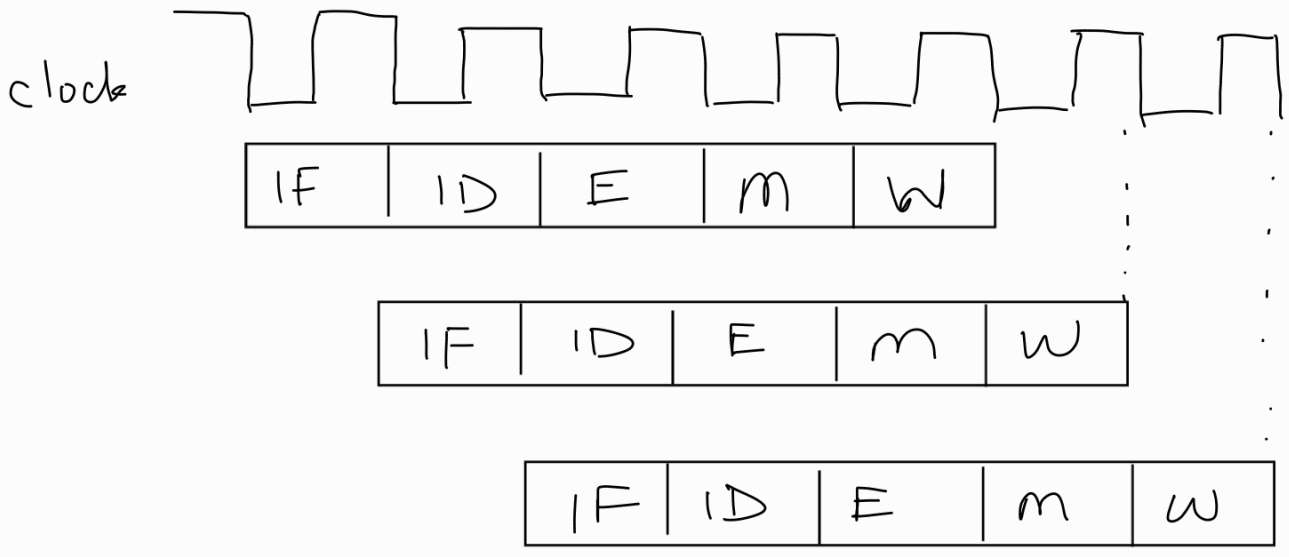
→ Another option is reduce req. Suppose you can process 600 img/ms in 200 GF/s.

→ But you can only do this if the memory can supply 600 imgs/ms. We need to make sure that the memory is not a bottleneck. This is called root cause analysis. Just adding more compute doesn't always help.

→ The clock cycle time depends on the instruction with longest critical path.

→ In sequential execution, performance takes a hit. This is why we use pipelining

Consider the Load instruction:



→ One instruction enters pipeline every cycle

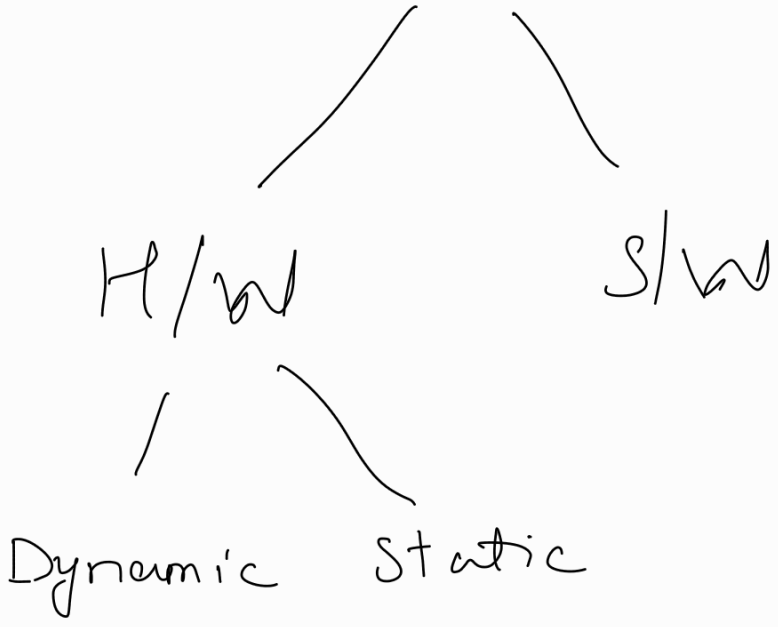
→ One instruction exits every cycle

→ Ideal CPI is 1

Structural Hazard

→ When 2 instructions try to access the same stage simultaneously.

Possible solutions



→ A situation where structural hazard may happen is R-type instr. comes after Load. They will access b/write at same time, as R-type is 4 stage.

→ Possible S/W solution is adding a dummy stage to R-type so they become 5-stage. Mem stage is NOOP stage.

Detail the ADD and BEQ instructions

ADD R_d, R_s, R_t

BEQ R_s, R_T, label

29/8/24

Pipelined Processors

1] Stages

2] Inorder execution (preserves semantics)

Investigate the Shakti Processor by IIT Madras → Github

Data Hazards

1] Read After Write (RAW)

i_1 ADD R1, R2, R3 → R1

i_2 ADD R4, R1, R4

} True/Flow Dependency.

i_2 reads R1 befor i_1 updates it.

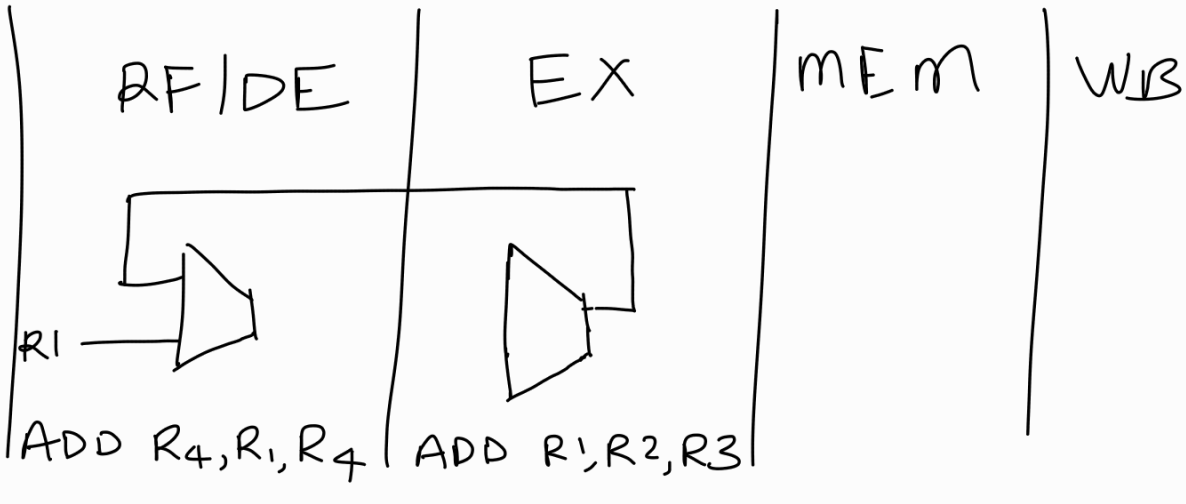
Possible Solutions

1] i_1 sets a flag on R1 and removes it after updating. i_2 cannot read R1 until the flag is removed. This works at hardware level.

2] At software level, the compiler adds NOOP instructions after i_1.

i_1
NOP
NOP
NOP
i_2

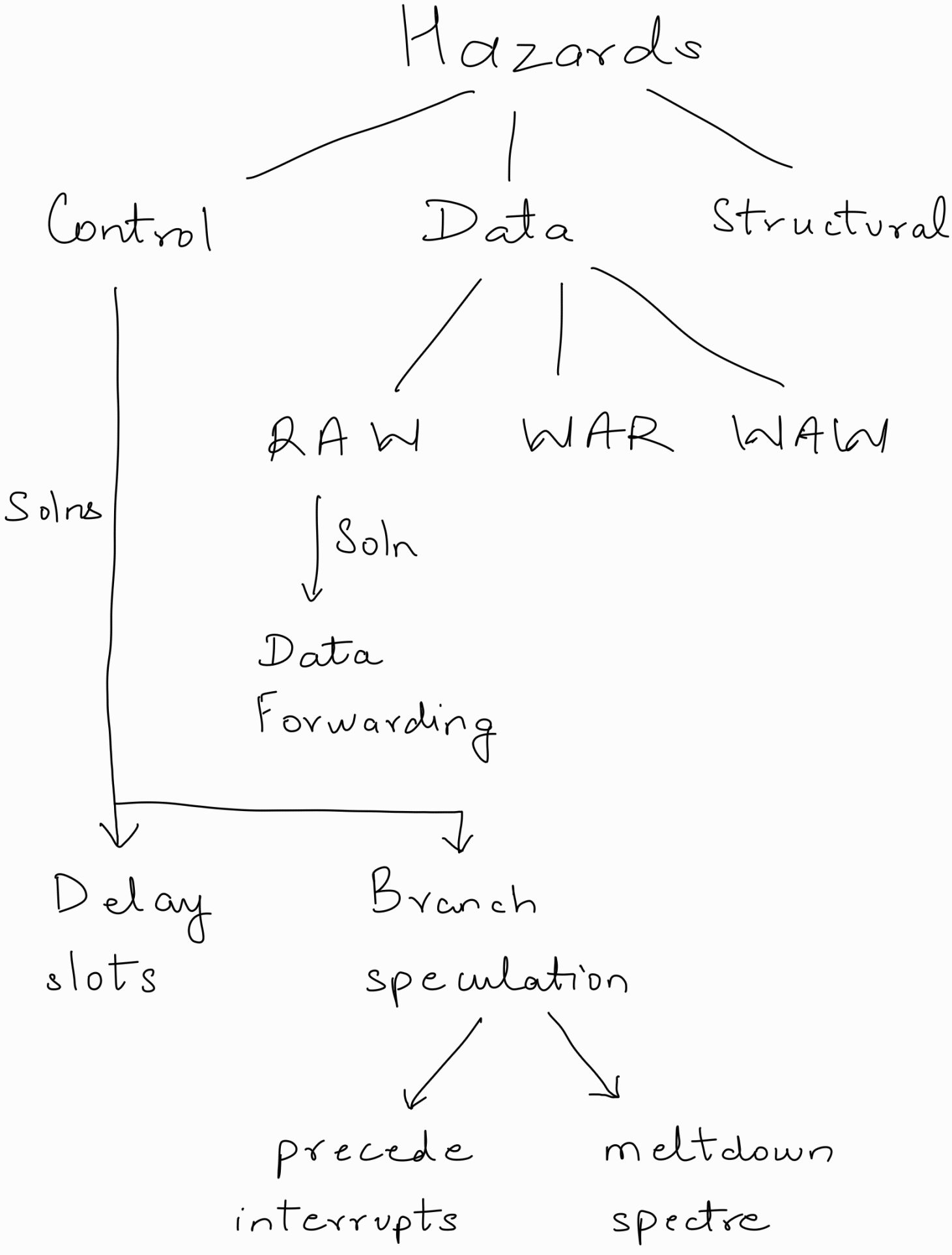
3]



In the i2 RF stage, you take the input from the ALU directly instead of waiting for i1 to write the o/p to R1 and then fetching.

For branching, the 2 instructions after the BEQ instr. are always executed. These are called branch delay instructions.

5/9/24


Hazards
Control
Data
Structural
RAW
WAR
WAW
Solns
Soln
Data Forwarding
Delay slots
Branch speculation
precede interrupts
meltdown spectre

Superscalar Processors

→ A key idea: Allow instructions behind stall to proceed

DIVD F0, F2, F4 } RAW dependency
ADDD F10, F0, F8 }

SUBD F12, F8, F14 ← There is no reason for this instr. to wait for ADDD to complete

→ Do we execute instructions out of order. This is the main feature of superscalar processors

→ WAR and WAW are introduced in OOO processors

Scoreboarding Technique

→ OOO execution divides ID stage

1] Issue - decode, check structural hazards

2] Read operands - wait until no data hazards, read operands

→ Scoreboarding divides the program into basic blocks and executes them

→ Construct the CFG and follow it.

→ WAW eg:

DIVD F0, F1, F2 } RAW dep

ADDD F3, F0, F4 } WAW

SUBD F3, F6, F7

As SUBD and DIVD have no dependency, you execute SUBD before ADDD. This will cause the final o/p in F3 to be ADD D o/p instead of the expected SUB D o/p.

Solutions for WAR

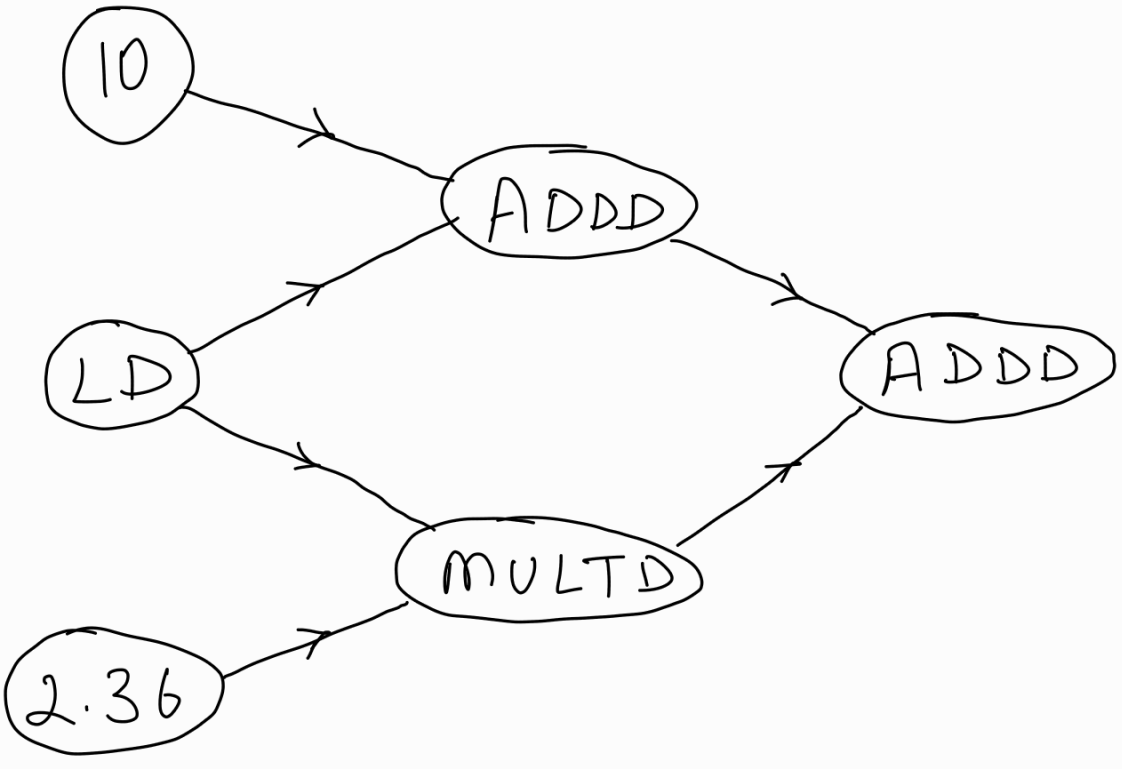
→ Stall WB stage until registers have been read

→ Read registers only during read operands stage.

9/9/24

Dataflow Architectures

→ Construct the data flow graph and perform ops.



→ Execution time of a system is equal to the length of its critical path

→ Barring RAW, we can resolve the other dependencies if we have a sufficient number of registers using register renaming

Stages in Scoreboarding

1] IF

2] Instruction Issue

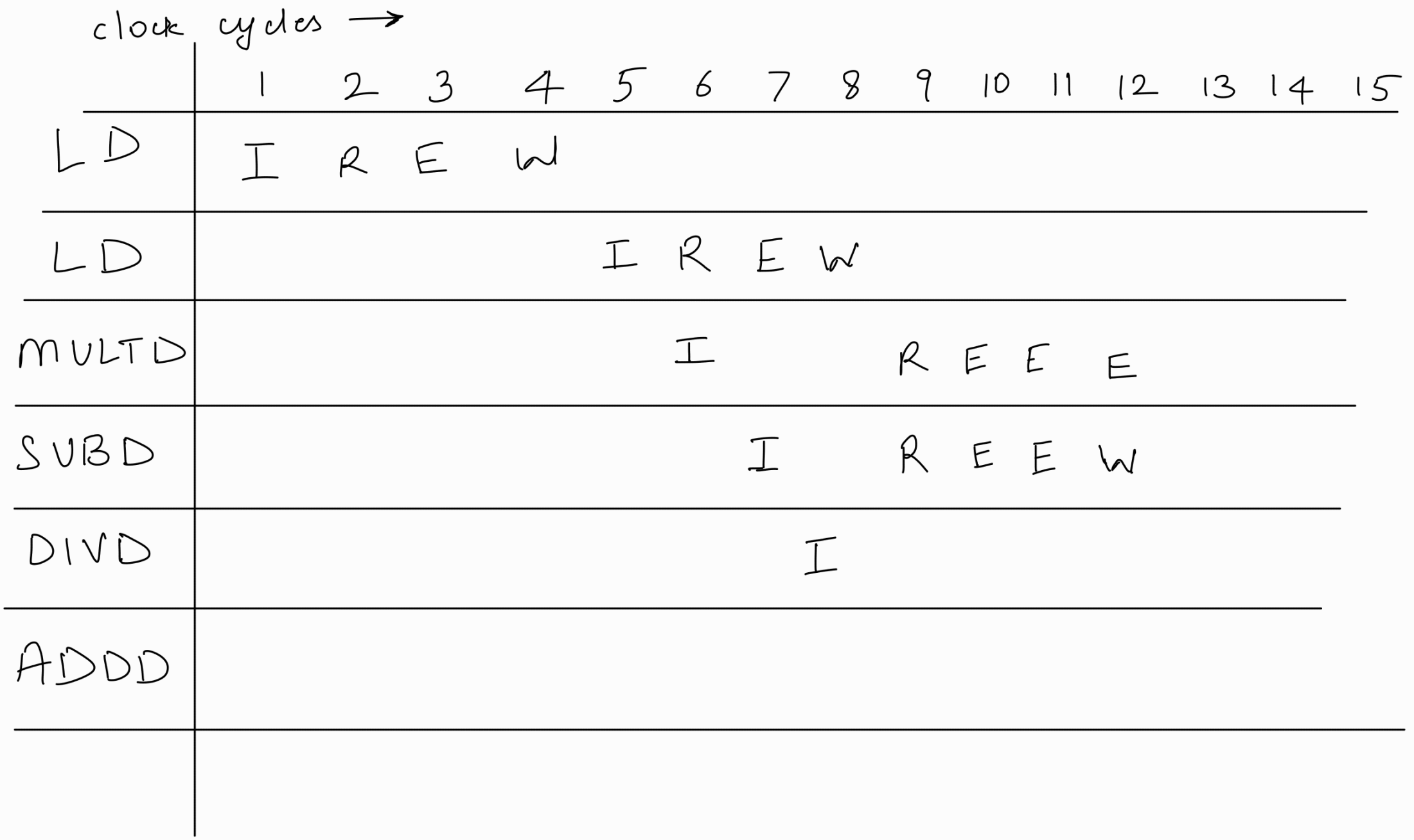
↳ ① Check for structural hazards
Introduce stalls if yes

② Check for WAW

3] Read operands – RAW

4] Execute

5] Write Back – WAR



SUBD - Execute takes 2 clock cycles

MULTD - Execute takes 10 clock "

mid + quiz 2 + final = 60

$2\frac{1}{5}$

2 paper presentations = 10 + 10

1 classical + 1 SOTA

Group of 5 for presentation

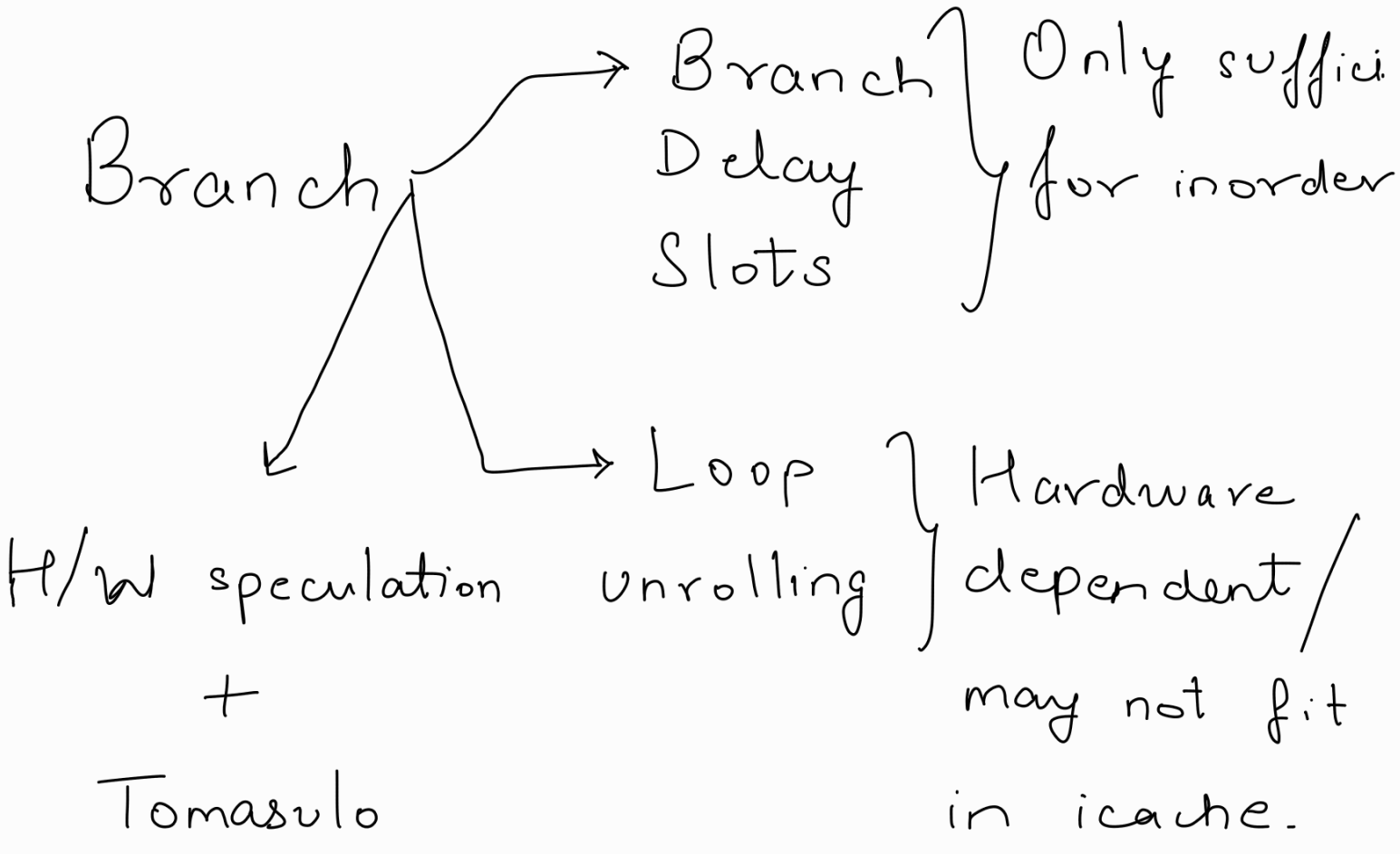
Assignment - 10

Research exploration - 10

Imp Conf

ISCA, MICRO, HPCA

Problem stmt | survey | POC



Branch Prediction Strategies

1] Always not taken
2] Always taken
3] BTFN
4] Profile based
5] Programmer based

} static

→ (2) works better than (1) as most branching instructions are loops.

→ SPEC INT & SPEC FP benchmarks