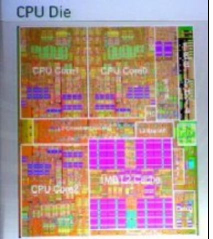


COMP 273
Introduction to Computer Systems

COMP 273

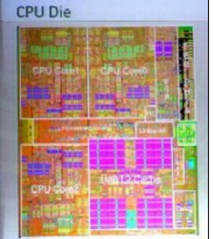
The Control Unit

(the sequencer)

Micro Architecture

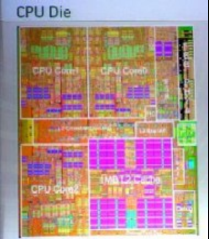
Part 3

Prof. Joseph Vybihal



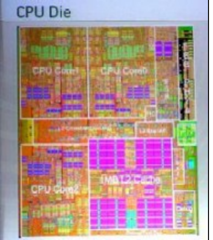
COMP 273
Introduction to Computer Systems

Announcements



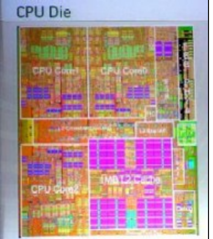
Readings

- Read
  - Edition 4 - Ch 4 sections 1 to 9
- Soul Of A New Machine
- Web Resources:
  - http://www-static.cc.gatech.edu/classes/cs3760_99_spring/lectures/singlecycle.html



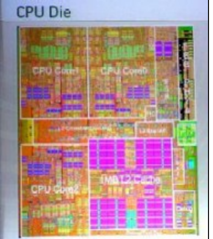
Outline

- The Sequencer (control unit)
- Hazards, faults, stalls and dumps
- Calculating CPU performance



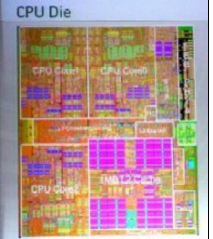
Part 1

The Control Unit

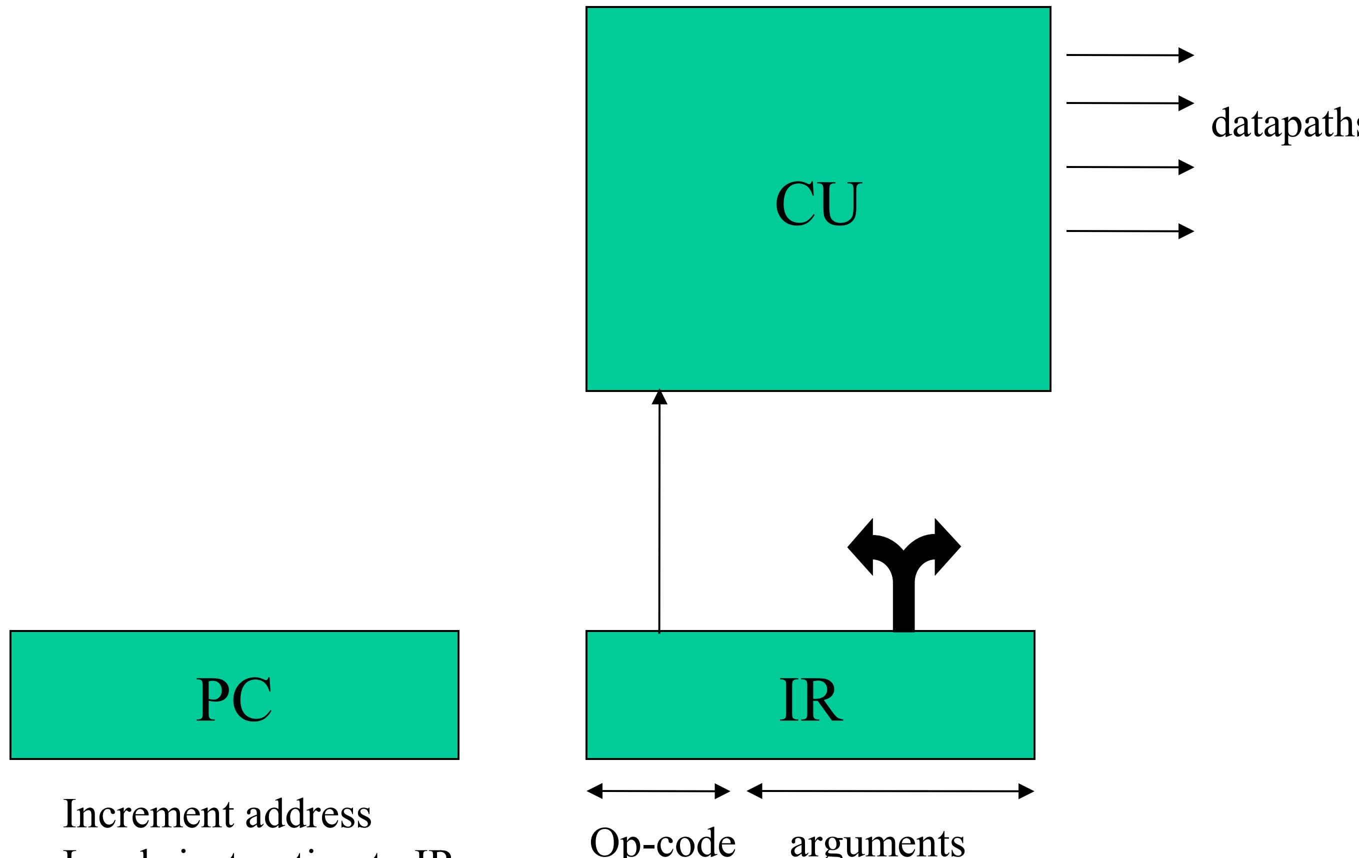


Definitions

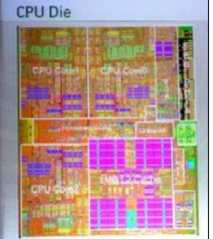
- Datapaths
  - The "wires" of a CPU that need to be engaged at a particular order & tick of the clock to implement an instruction.
  - This includes what the path is connected to: registers, gates, ALU, …
- Control
  - That portion of the CPU, often called the control unit or sequencer, that is responsible for the timing and triggering of a datapath
- Instruction Format
  - The organization of bits in the IR (definition of an instruction)
- Micro-programming (the datapaths that implement the instruction)
  - Flat: One instruction executes at a time
  - Pipeline: Assembly execution of more than 1 instruction
  - Cores: Parallel execution



Flat Control Unit

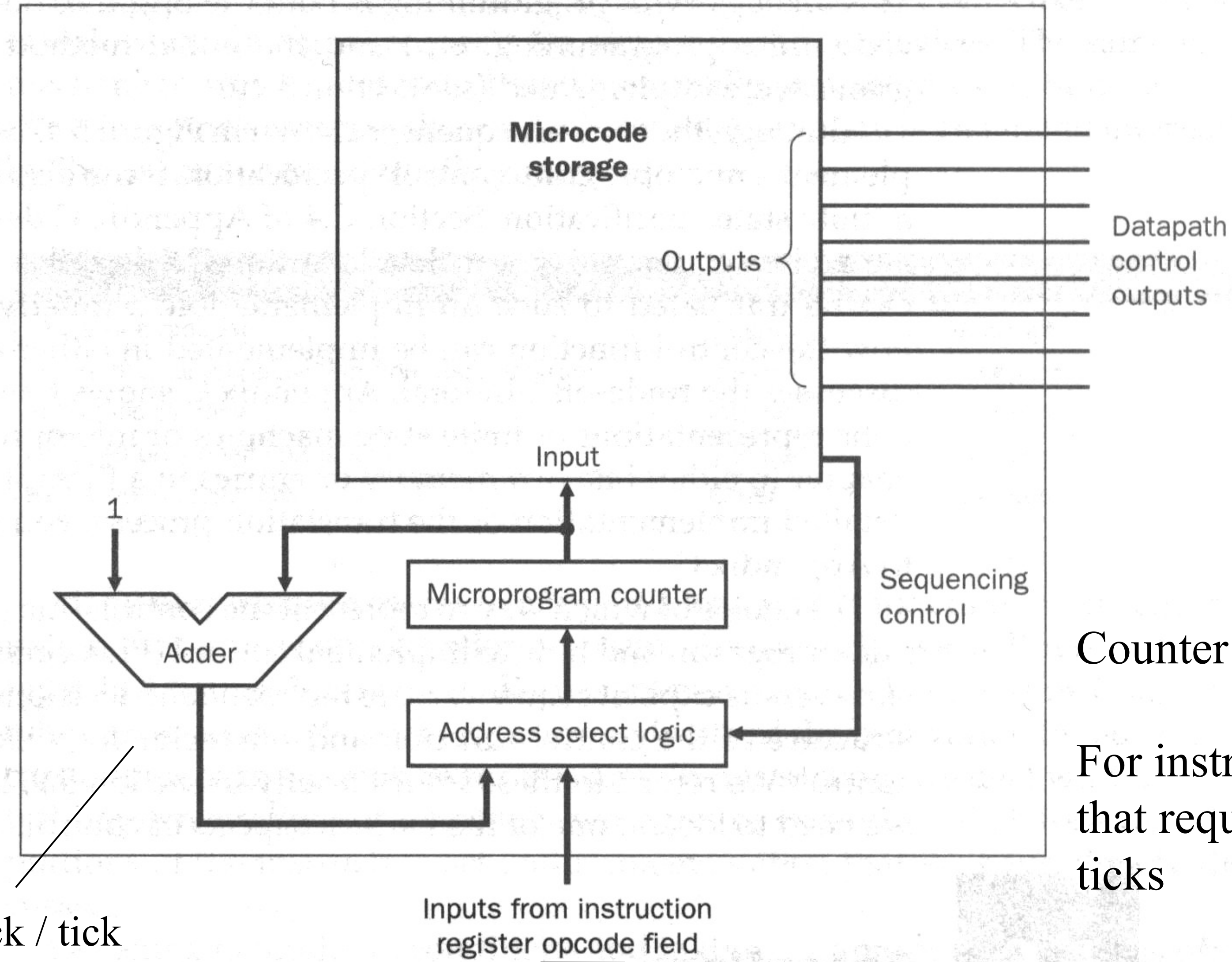


1. Increment address
2. Loads instruction to IR



COMP 273
Introduction to Computer Systems

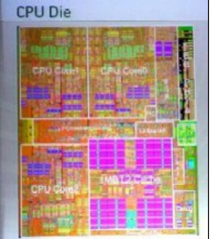
Flat Control Unit



Counter logic

For instructions that require n ticks

Tick / tick



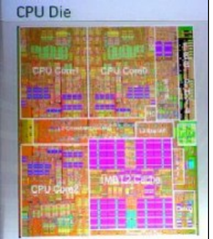
Op-code and Count

OP-CODE	COUNT
00100	00100
	00101
	00110
01000	01000
	01001
	01010
	01011

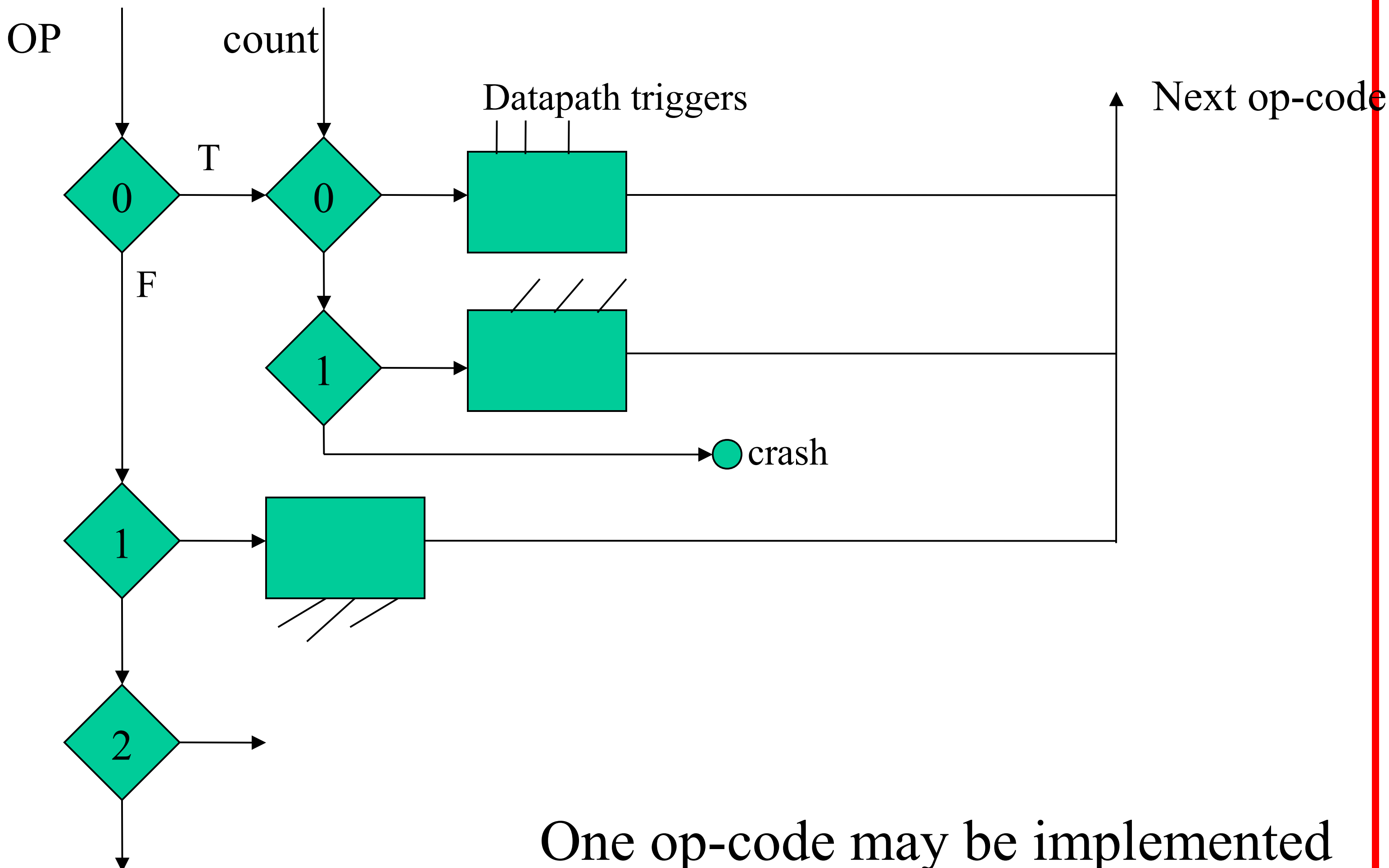
01100

Actual op

The count (built into op-code or sign extended in register)



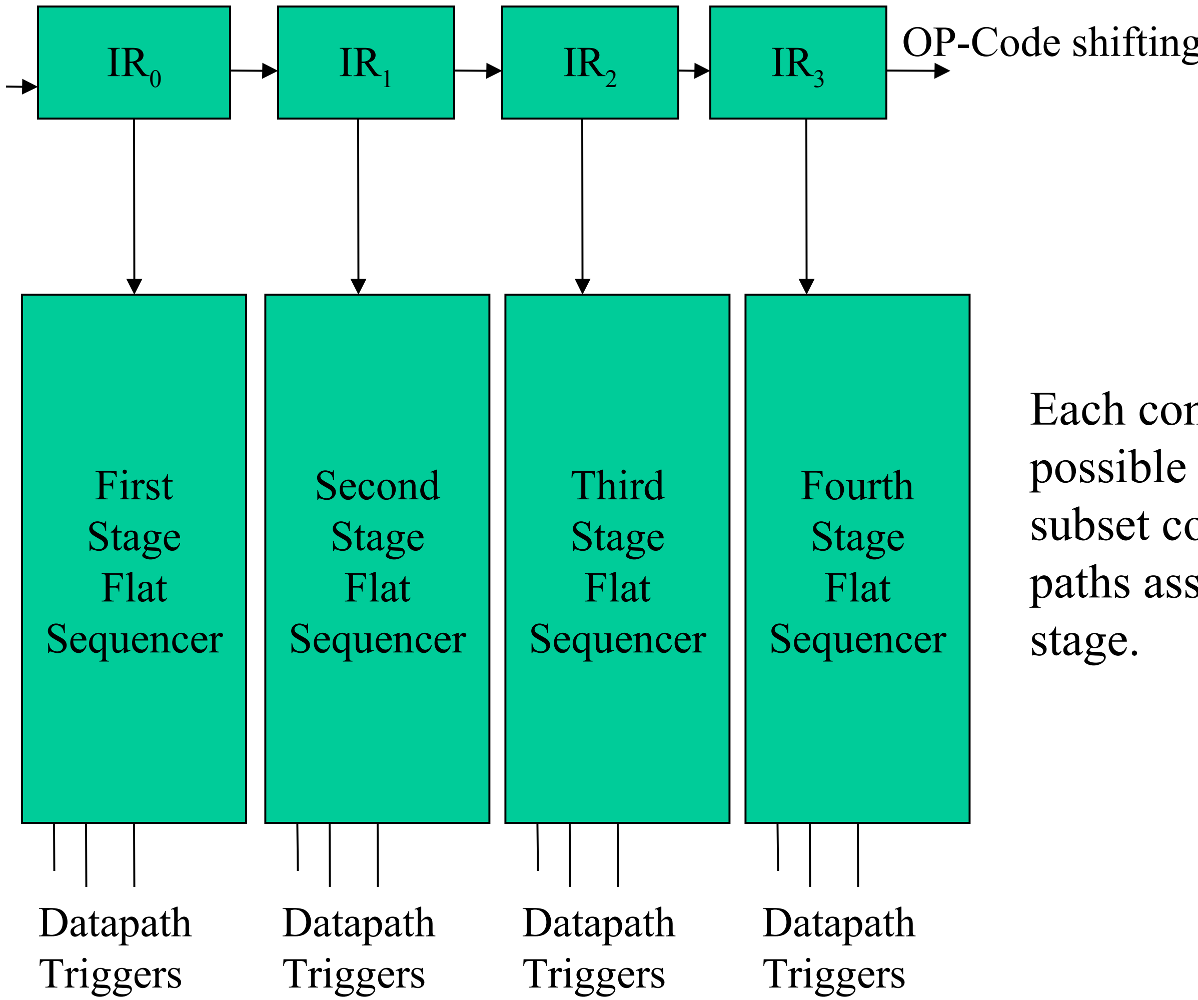
Flat Sequence



One op-code may be implemented via multiple sub-steps (datapaths)

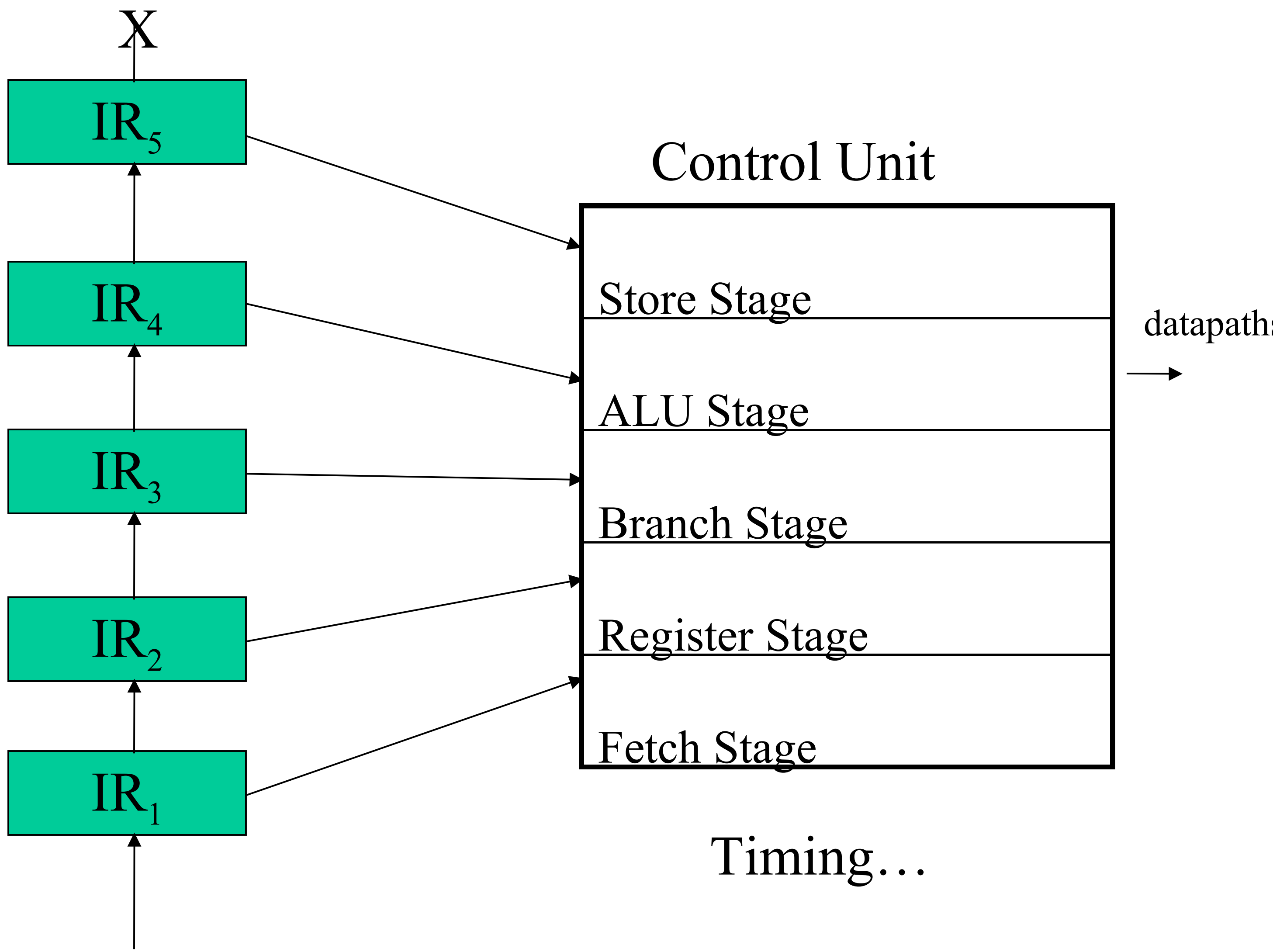
4 IR buffers

Pipeline Sequence



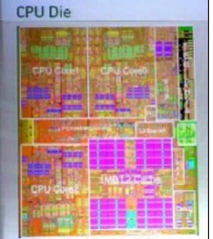
Each contains a subset of all possible datapaths. This subset consists of only those paths associated with the stage.

Pipeline Sequencer (basic)

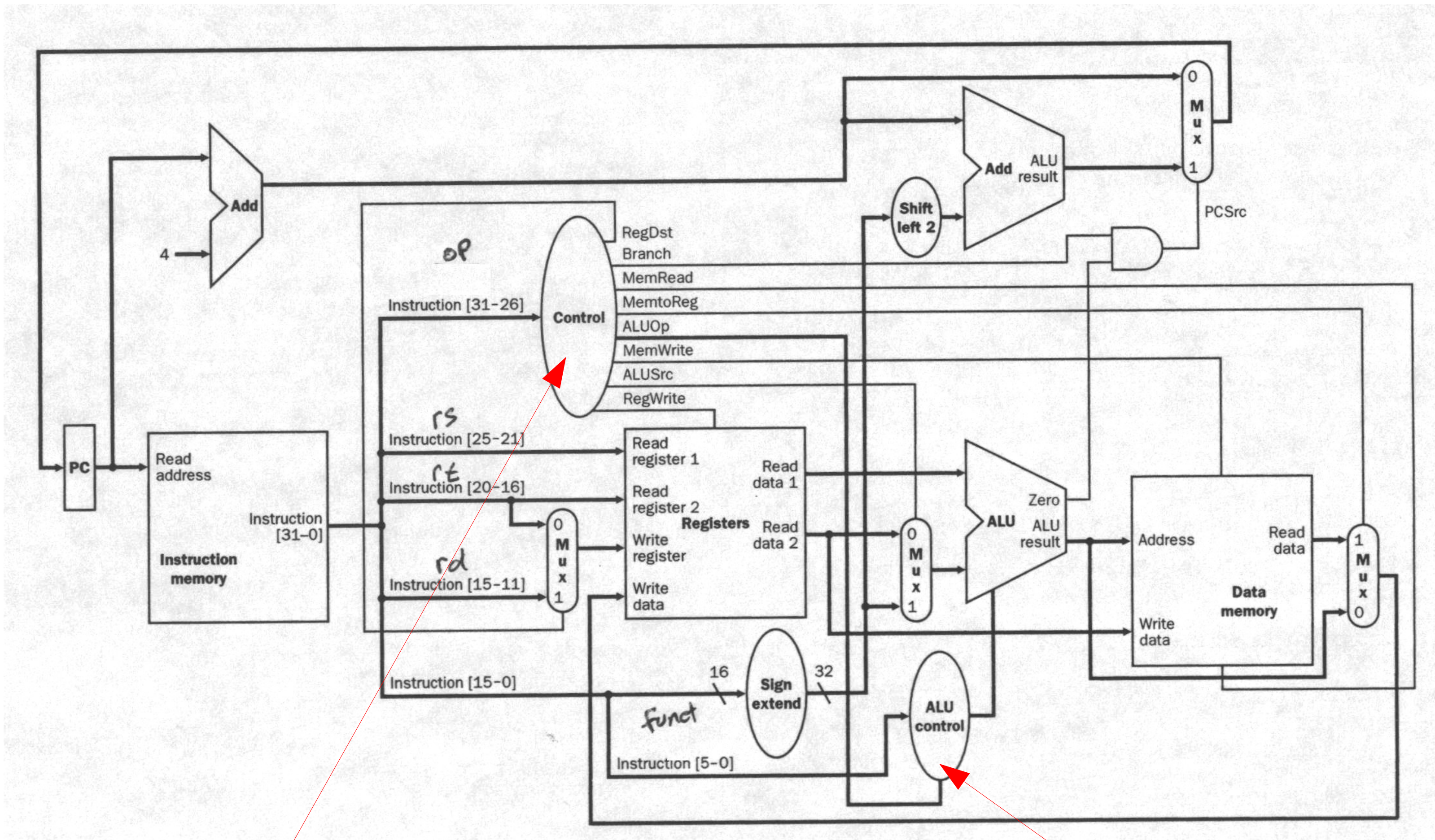


Timing…

COMP 273
Introduction to Computer Systems



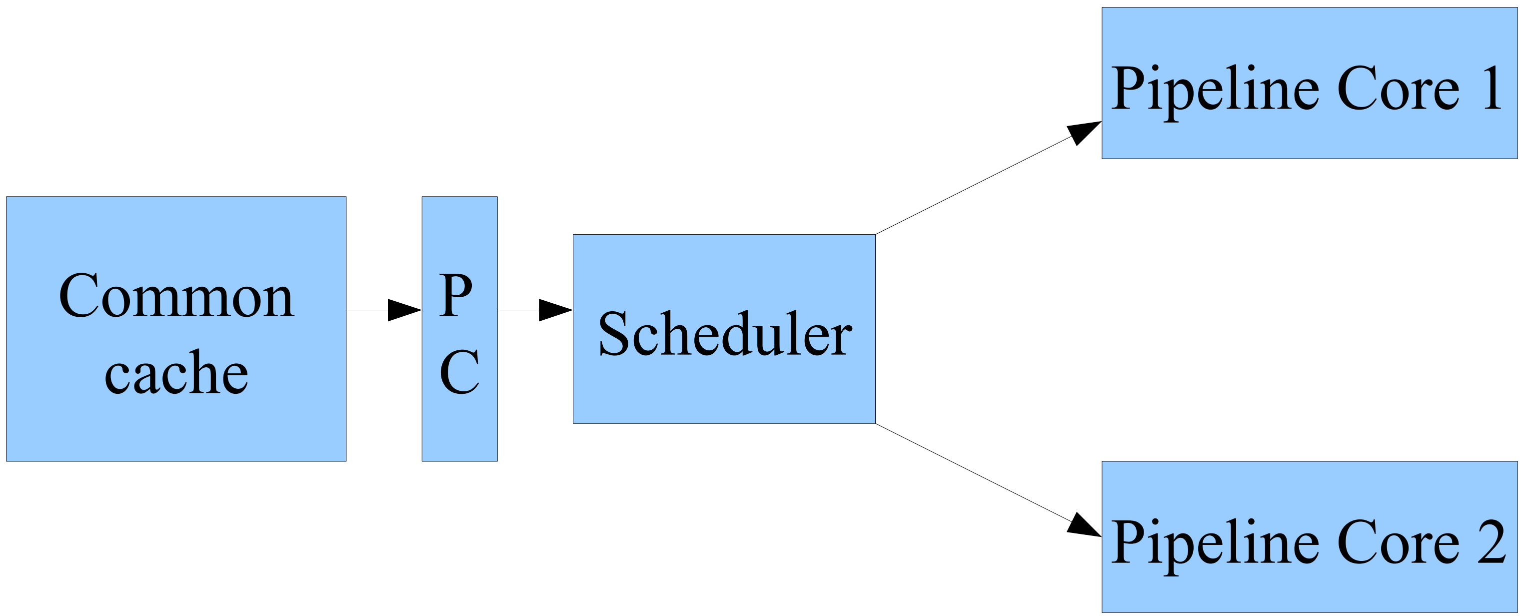
MIPS Sequencer

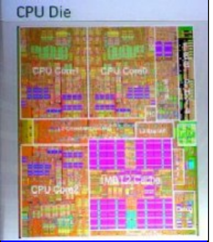


Two sequencer units

Cores Sequencer

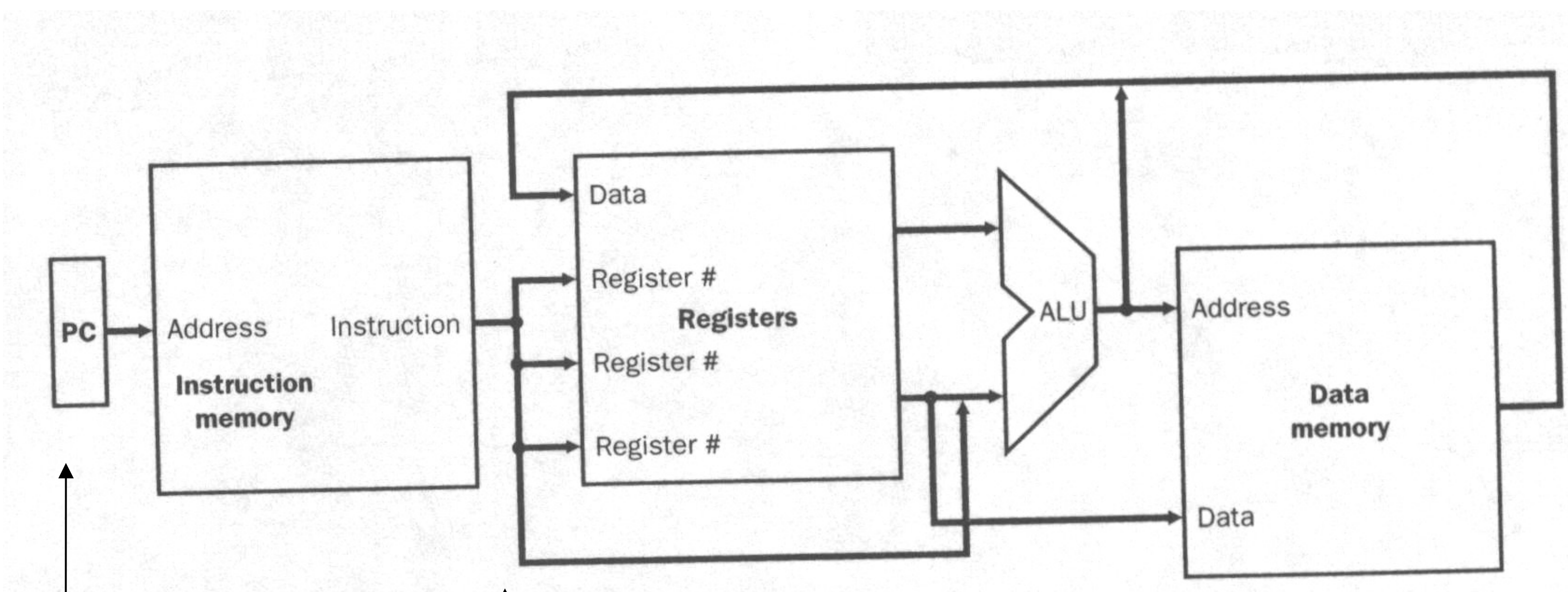
(simplified)





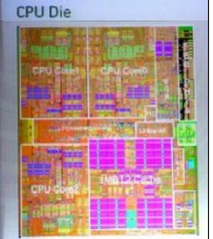
Buffers

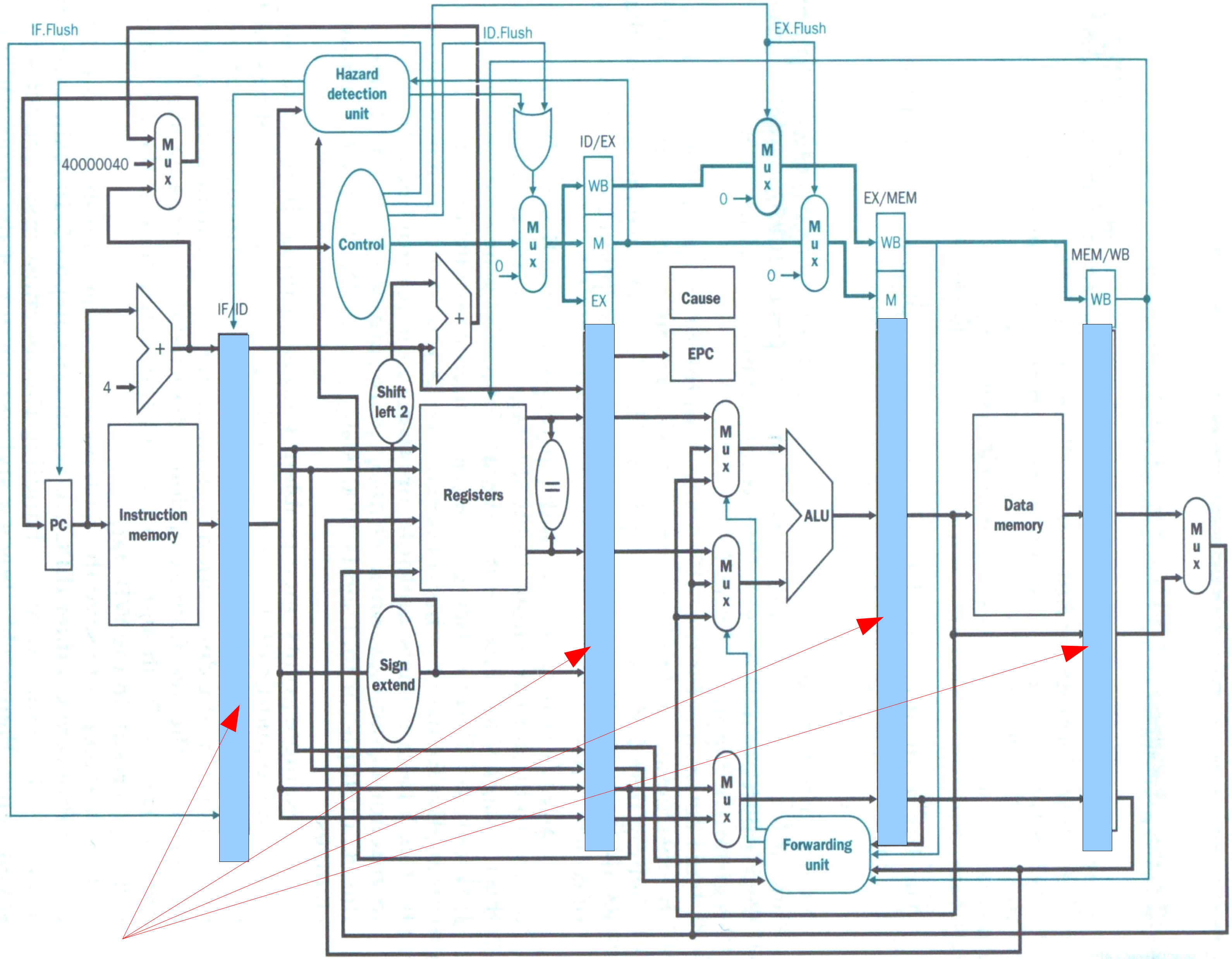
Buffers and Multiple Pathways



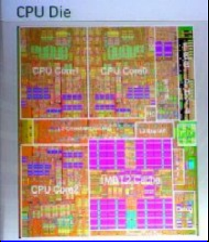
One PC

Multiple IR registers (buffers) for each instruction in pipeline, each passing through one stage of the control unit (sequencer)

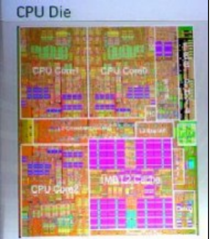

CPU Die


IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
40000040
M u x
Control
ID/EX
WB
M
EX
EX/MEM
MEM/WB
0
Cause
EPC
IF/ID
4
+
Shift left 2
Registers
=
ALU
Data memory
PC
Instruction memory
Sign extend
Forwarding unit

IR (+ buffers)



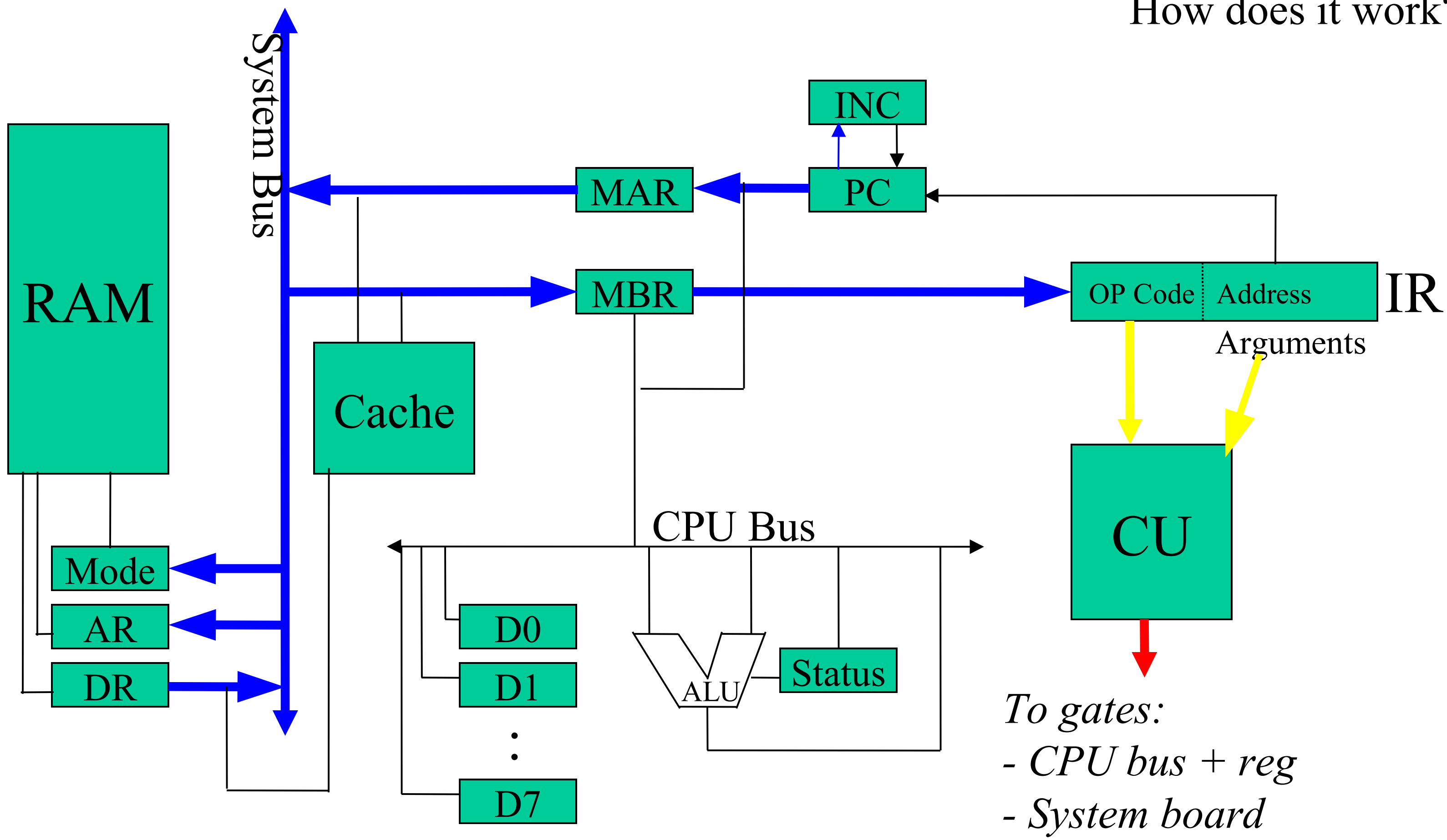
CPU Execution Overview/Review



COMP 273
Introduction to Computer Systems

Classical CPU Design

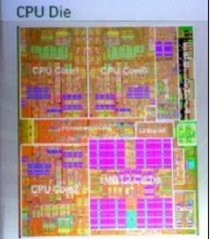
How does it work?



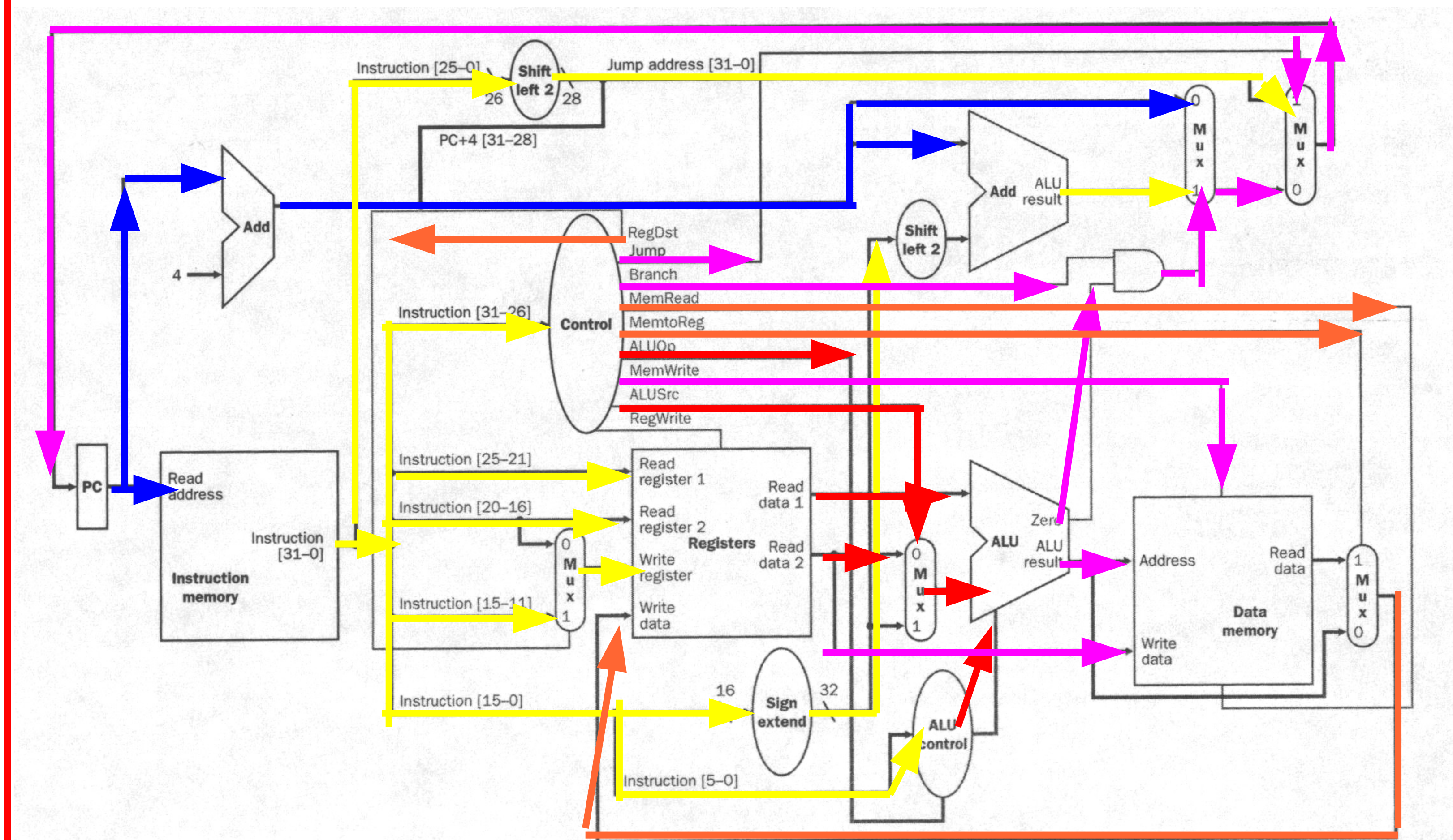
1. Fetch (5 cs)

2. Decode (1 cs)

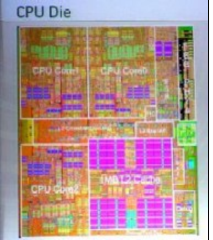
3. Execute (1 cs per step = 3 – 5 steps)



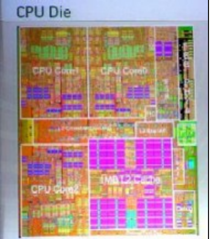
Pipeline CPU Design



1. Fetch (1 cs) 2. Decode (1 cs) 3. Execute (1 cs per step) 4. Store

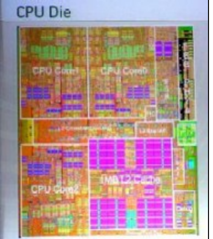


Hazards & Faults



Terminology

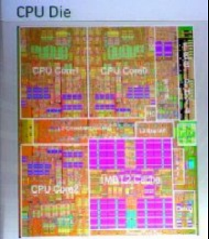
- A “hazard” may lead to a “fault”
  - Hazard: a danger to keep watch for
  - Fault: an error that has occurred
- A “fault” leads to a CPU execution “stall”
  - Stall: Normal CPU execution cycle lengthens
- A “stall” can lead to a “dump” or a “no-op”
  - Dump: All instruction after the fault are dumped from the pipeline and re-loaded after fault is handled (major slow down)
  - No-Op: Additional instruction(s), that do no action, inserted between the fault and the next instruction to allow the CPU to execute the problematic instruction without side-effects (minor slow down)



Types of Hazards

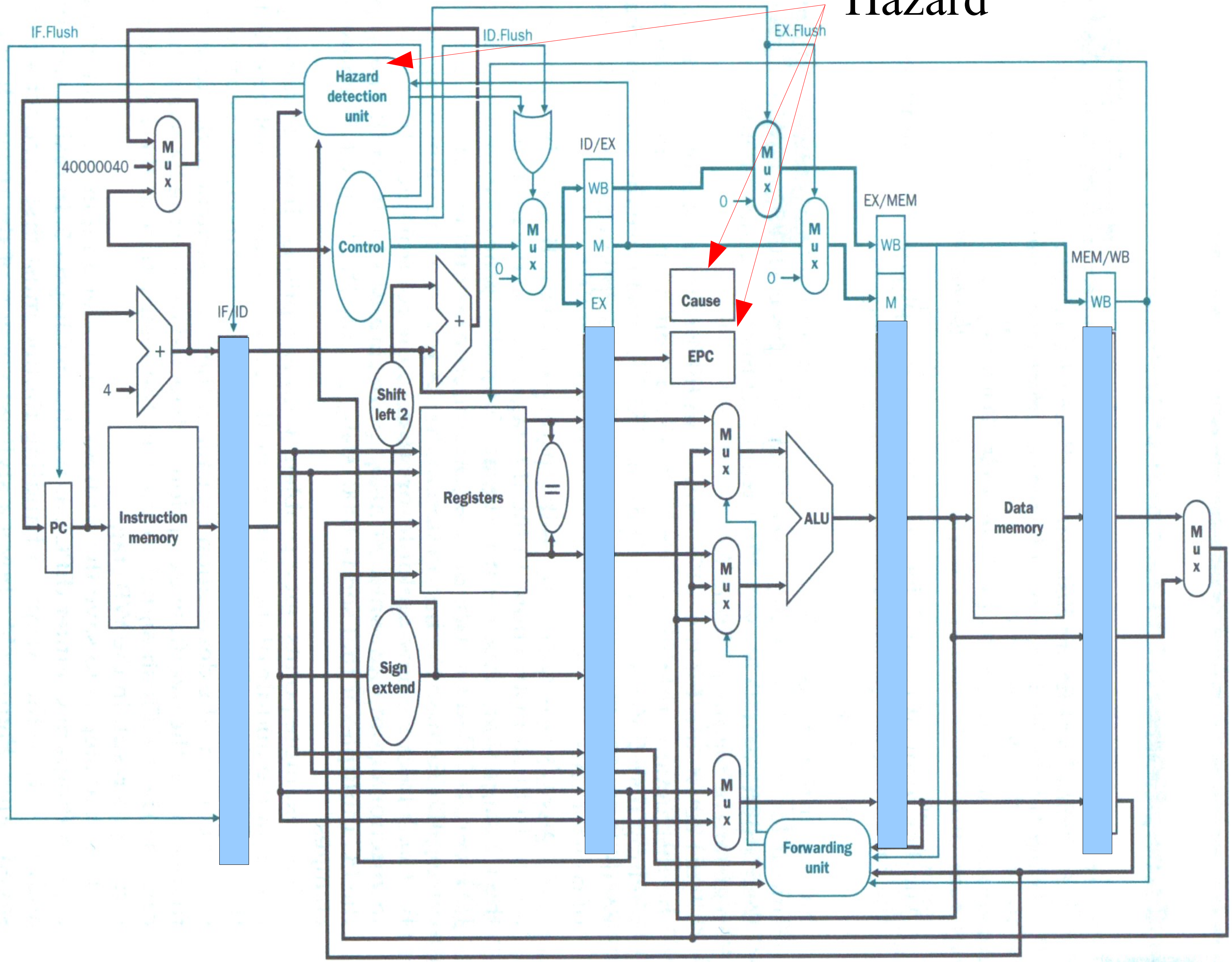
- Hazards
  - Structural Hazards
    - CPU cannot support the combination of instructions in the pipeline (eg. Single instruction store/load crash)
    - Illegal instruction or illegal result (like divide by zero)
  - Control Hazards
    - Branch request causing the semi executed pipeline instructions to be unnecessary
  - Data Hazards
    - An instruction depends on the results of a previous instruction in the pipeline
      - Add $s0, $t0, $t1
      - Sub $t2, $s0, $t3

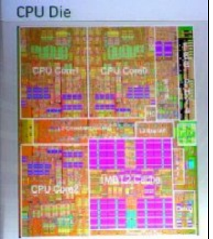
      Identifiable by bit patterns
    - NOP or wiring for Forwarding or Backup buffer



COMP 273
Introduction to Computer Systems

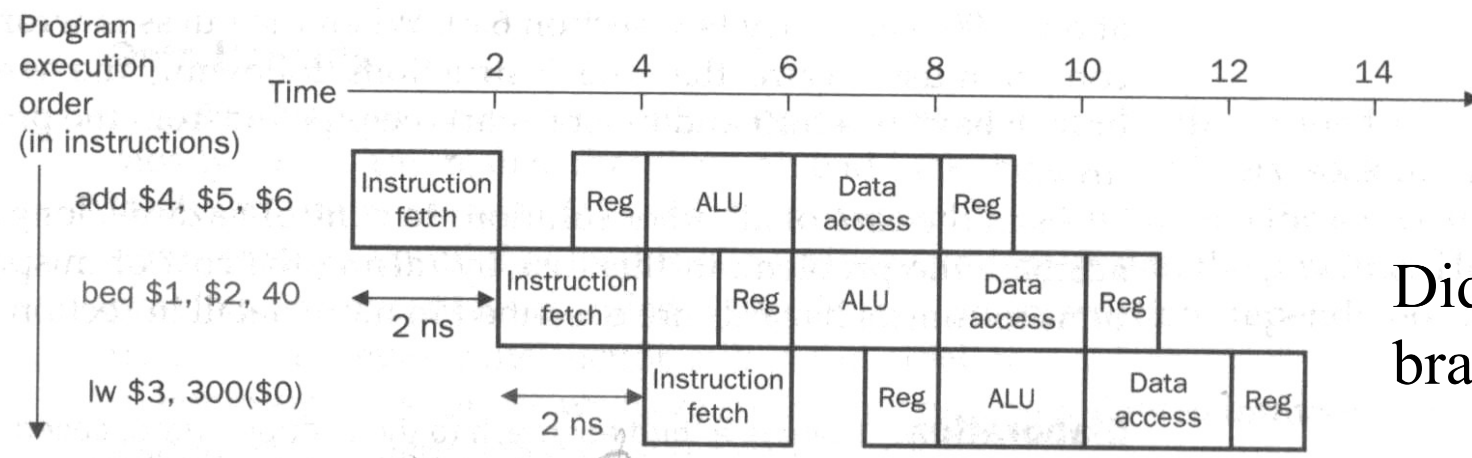
Hazard



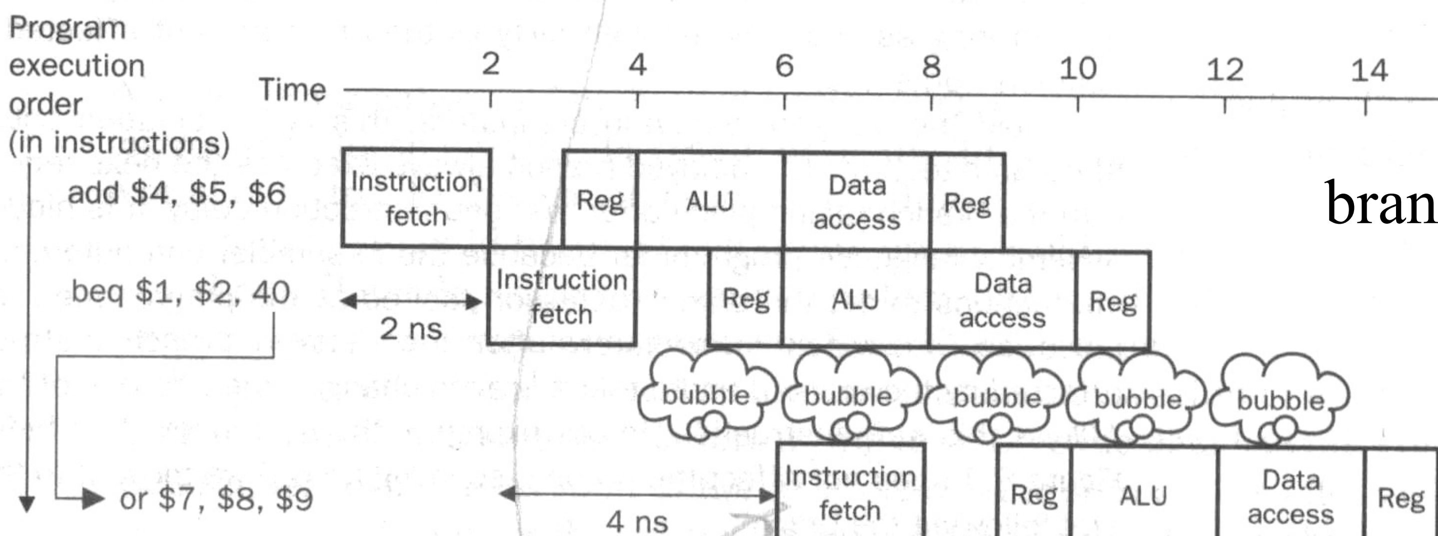


COMP 273
Introduction to Computer Systems

Effects…

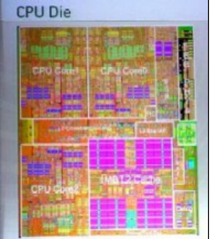


Did not branch



branched

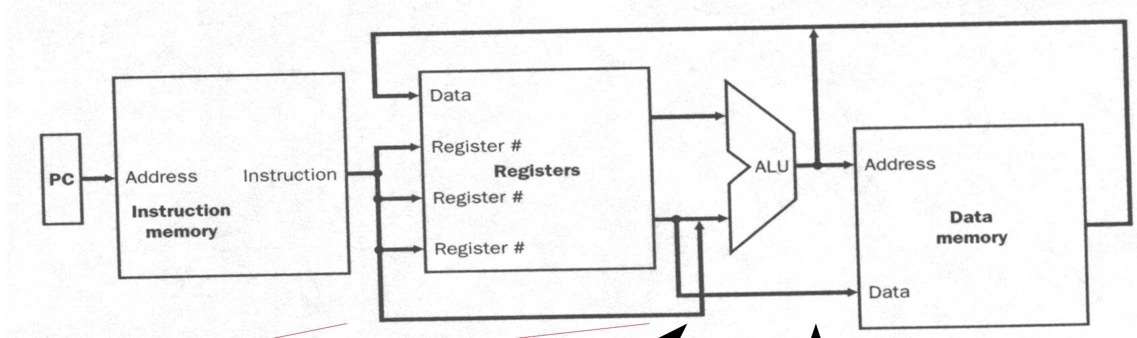
Instruction at address 40



Calculating Loss

- At which stage did the fault occur?
- If no-op, how many were inserted?
- If dump, how many instructions need to be reloaded?

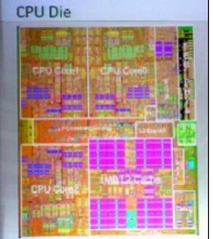
IR1 IR2 IR3 IR4



Dump Points

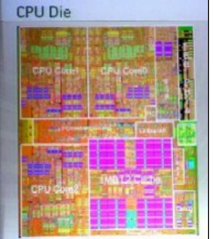
Register Prob

BEQ decision



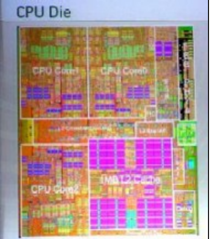
Calculating Loss

- How many instructions execute at full speed (no faults)?
- What do we loose when we do a stage 2 (IR1), stage 3 (IR2) dump or stage 4 (IR3) branch?



Calculating Loss

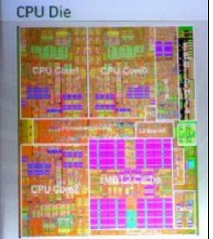
- What performance loss do we experience?
  - Expected(Loss) = [P(Loss) * Cost] * LossCount
- How often can we expect a stall?
  - 17% on average
    - Actually depends on the program,



Solutions?

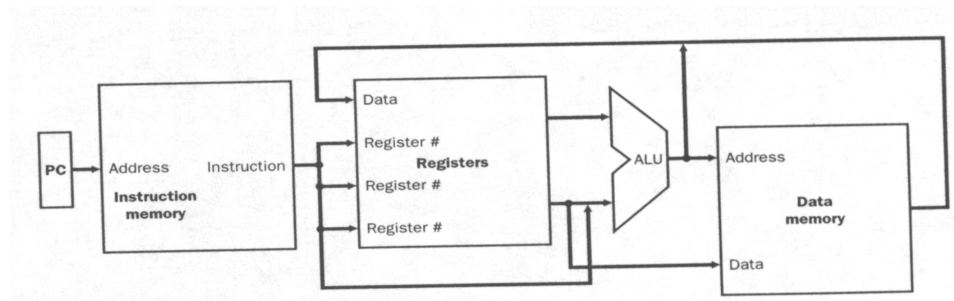
Inserting no-op(s) or reordering

- Prediction
  - Assume branch will always fail (not used)
    - Happens too often...
  - Assume branch success based on type
    - Function calls (yes)
    - Backward jumps (probably loops, yes)
    - If statements (no, fail)
  - Reorder instructions (compiler solution)
    - We need a delay to figure out if branch will happen
    - Instead of using a NOP, reorder branch & preceding instruction



Reorder Example

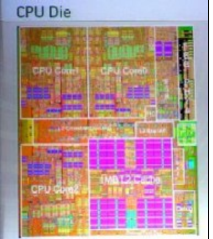
- Original code:
  - Add $4, $5, $6
  - Beq $1, $2, 40
  - Lw $3, 300($0)



- Reorder:
  - Beq $1, $2, 40
  - Add $4, $5, $6
  - Lw $3, 300($0)

Finishes at the same time

This is not loaded



COMP 273
Introduction to Computer Systems

Part 2

CPU Performance Issues

Cycles vs Clock Ticks

- Clock Ticks
  - A count of the number of actual clock ticks required to perform an activity in the CPU
- Cycles (confused definitions)
  - A count of the number of micro instruction required to perform an activity in the CPU
- Note: often equations with cycles can be used with clock ticks

CPU Die

COMP 273
Introduction to Computer Systems

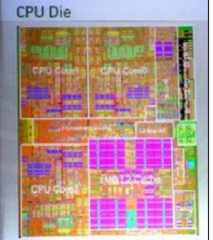
The Role of Performance

Keep in mind that the only complete and reliable measure of CPU performance is **clock ticks**. *CPU Execution Time* is the product of three basic measurements:

- Instruction count (total number of instructions in program)
- Clock cycles per instruction
- Clock cycle time

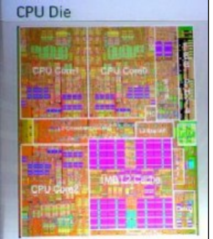
CPU execution time in seconds =

X instructions in program *
Y ticks per instruction *
Z seconds per tick



Example 1

- A 2 Ghz computer ticks 2 billion times/sec.
- If we have CPU that executes a single instruction in a total of 20 ticks
- How long will is take to run a program that has 1000 lines of code (linear)?
  - T = 1000 * 20 / 2billion = 0.00001 sec



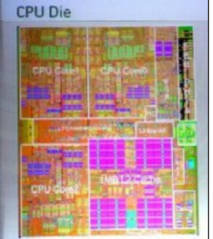
Example 2

Assume that the operation times for the major functional units in a CPU are:

- Memory access: 2 ns
- ALU and adders: 2 ns
- Register file (i/o): 1 ns

Assuming that the multiplexers, control unit, etc. have no delay, which of the following implementations would be faster and by how much?

1. An implementation in which every instruction operates in 1 clock cycle of a fixed length.
2. An implementation where every clock cycle is of variable length equivalent to the instruction step to execute

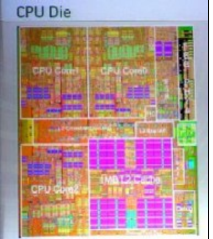


Assuming the program instruction mix is:

- 24% loads
- 12% stores
- 44% ALU
- 18% branches
- 2% jumps

Recall:

CPU Time = X instructions * Y ticks/instr * Z seconds/ticks

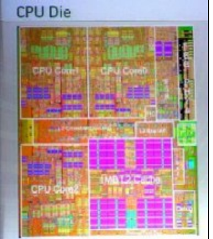


Instruction class	Functional units used by the instruction class				
ALU type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				
	2	1	2	1 or 2	1

Memory access: 2 ns
ALU and adders: 2 ns
Register file (i/o): 1 ns

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
ALU type	2	1	2	0	1	6 ns
Load word	2	1	2	2	1	8 ns
Store word	2	1	2	2		7 ns
Branch	2	1	2			5 ns
Jump	2					2 ns

Variable vs. fixed cycle lengths?



Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
ALU type	2	1	2	0	1	6 ns
Load word	2	1	2	2	1	8 ns
Store word	2	1	2	2		7 ns
Branch	2	1	2			5 ns
Jump	2					2 ns

CPU Time fixed = longest instruction = load = 8 ns

CPU time variable = $8 * 24\% + 7 * 12\% + 6 * 44\% + 5 * 18\% + 2 * 2\% = 6.3$ ns

Performance improvement = $8 / 6.3 = 1.27$

Variable clock cycles are not always used:

- Harder to build
- Delay in logic gates to implement > 1.27