CSE 520
Computer Architecture II

CPU Performance Evaluation

Prof. Michel A. Kinsy

Performance Measurement

- Processor performance:
  - Execution time
  - Area
    - Logic complexity
  - Power

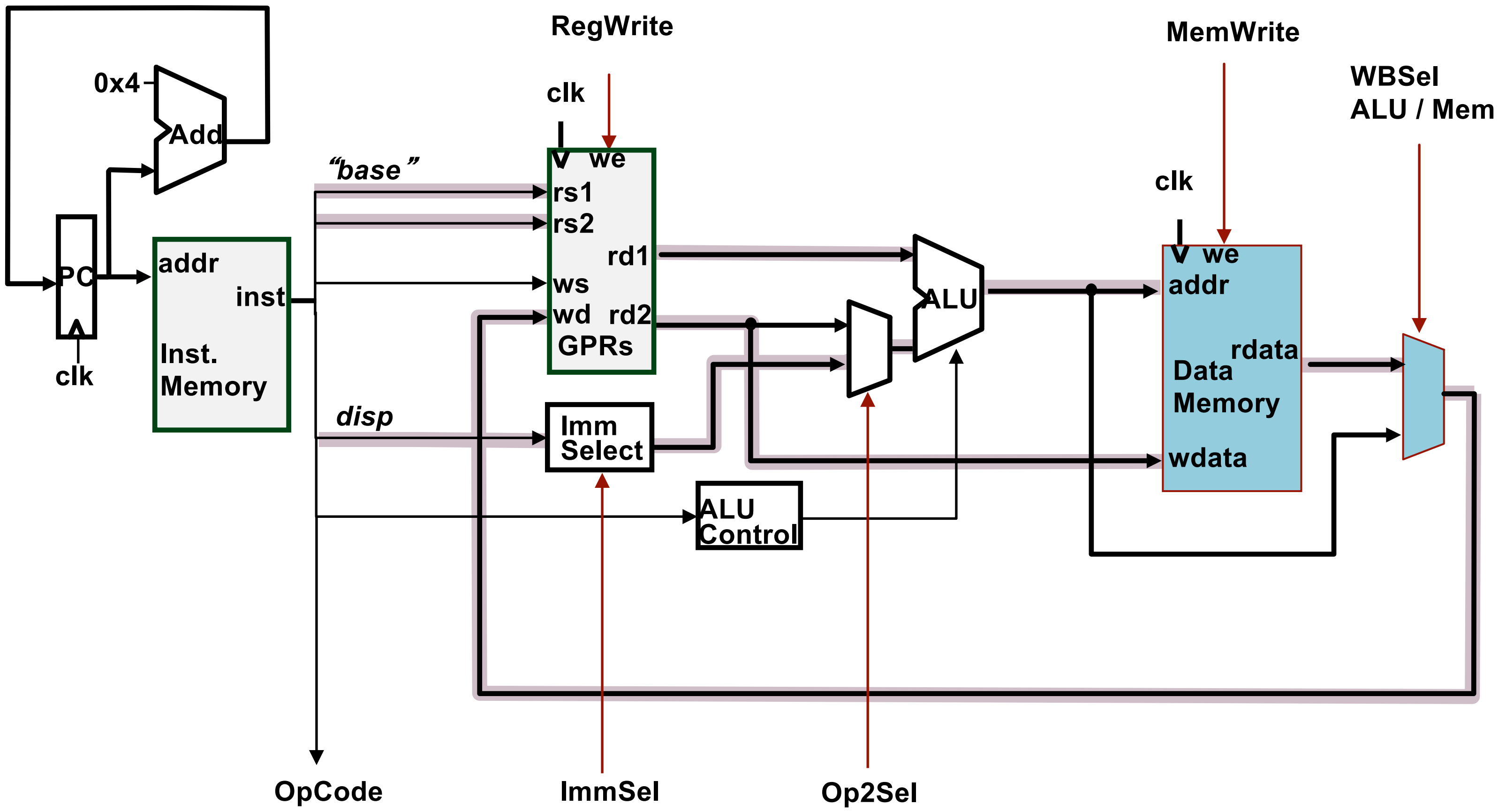
$$\boxed{\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}}$$

- In this class we will focus on Execution time

Datapath for Memory Instructions

- Should program and data memory be separate?
  - Harvard style: separate (Aiken and Mark 1 influence)
    - read-only program memory
    - read/write data memory
- Princeton style: the same (von Neumann's influence)
  - single read/write memory for program and data
    - Executing a Load or Store instruction requires accessing the memory more than once

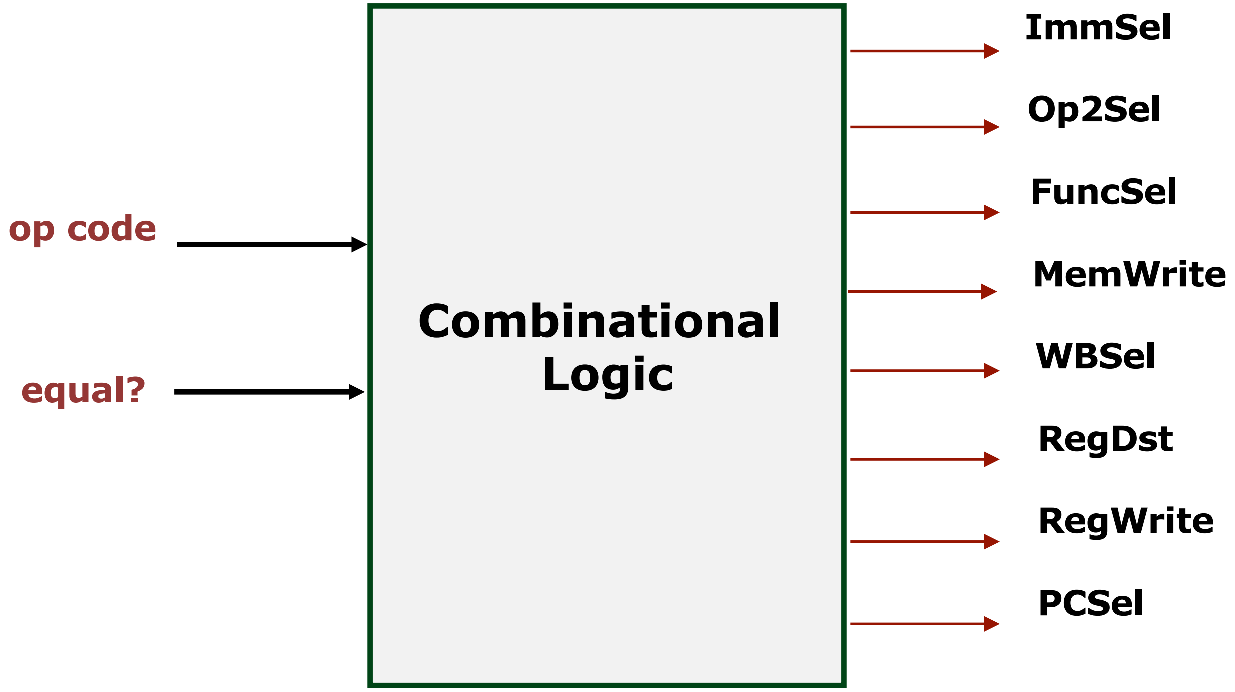
Harvard Architecture



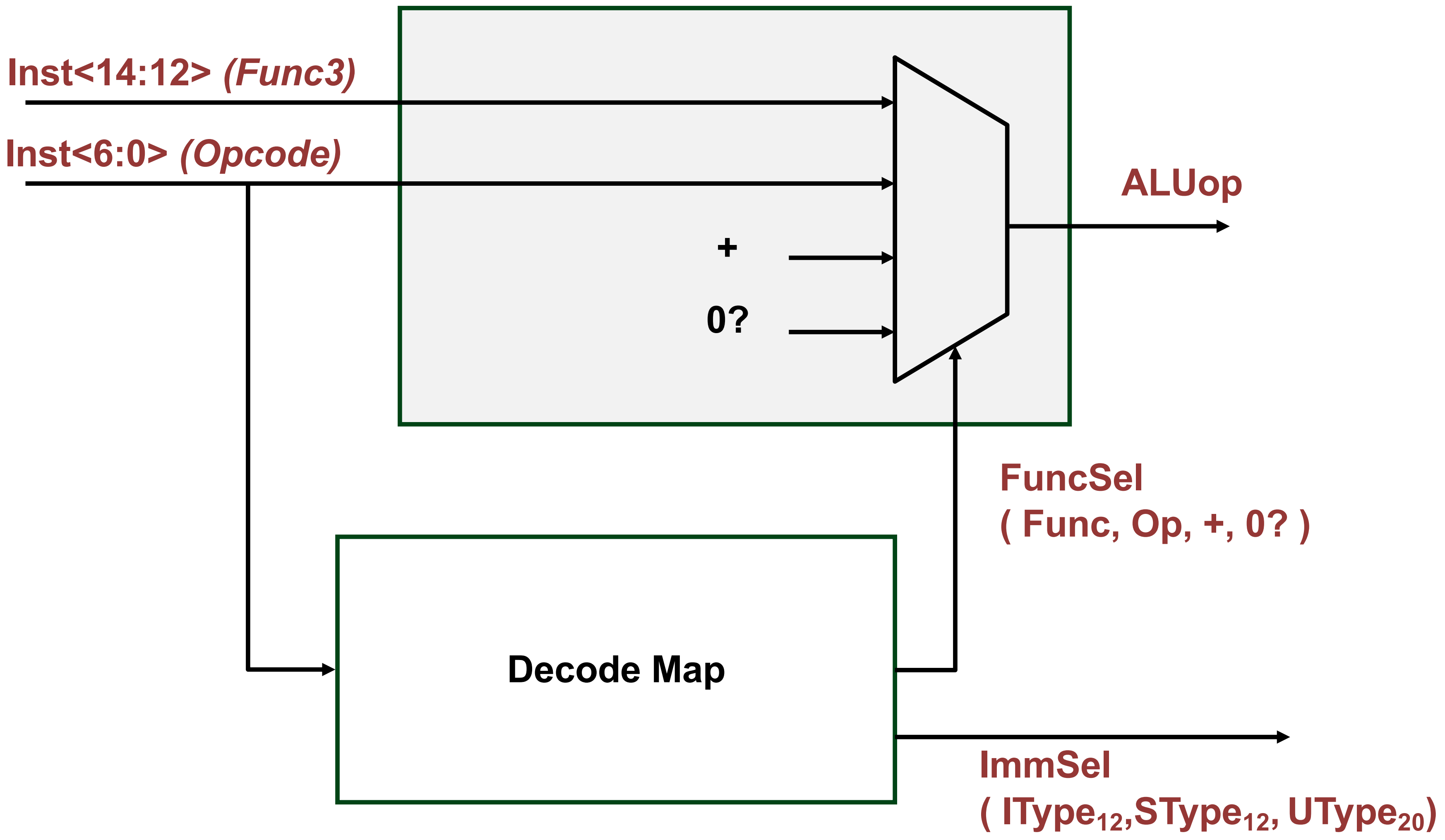
7	5	5	3	5	7	
imm	rs2	rs1	funct3	imm	opcode	Store (rs) + displacement
imm[11:0]		rs1	funct3	rd	opcode	Load

Hardwired Control

- Hardwired Control is pure Combinational Logic



ALU Control & Immediate Extension



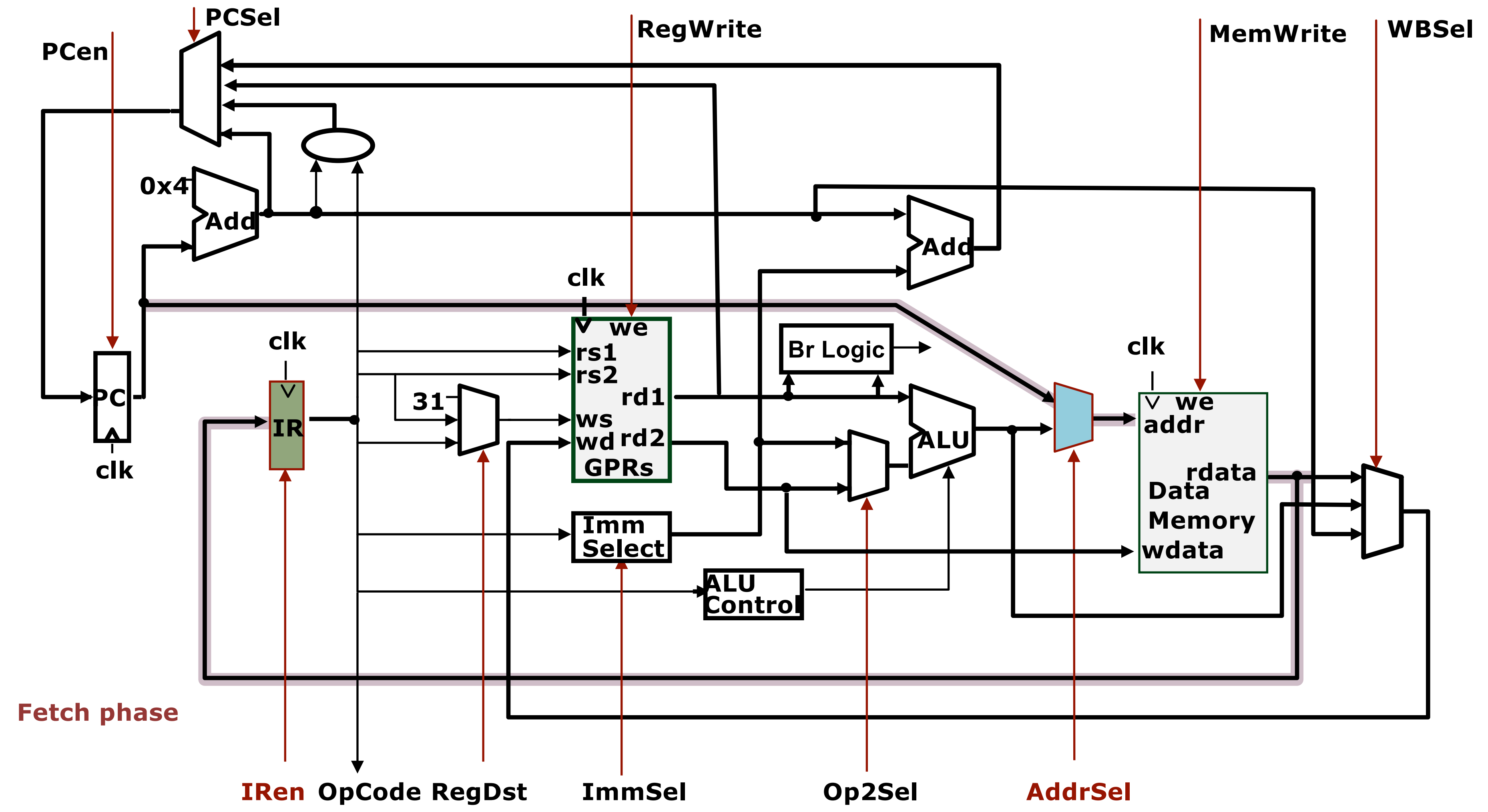
Hardwired Control Table

Opcode	ImmSel	Op2Sel	FuncSel	MemWr	RFWen	WBSel	WASel	PCSel
ALU	*	Reg	Func	no	yes	ALU	rd	pc+4
ALUi	$IType_{12}$	Imm	Op	no	yes	ALU	rd	pc+4
LW	$IType_{12}$	Imm	+	no	yes	Mem	rd	pc+4
SW	$SType_{12}$	Imm	+	yes	no	*	*	pc+4
BEQ_{true}	$SBType_{12}$	*	*	no	no	*	*	br
BEQ_{false}	$SBType_{12}$	*	*	no	no	*	*	pc+4
J	*	*	*	no	no	*	*	jabs
JAL	*	*	*	no	yes	PC	X1	jabs
JALR	*	*	*	no	yes	PC	rd	rind

Single-Cycle Hardwired Control

- Harvard architecture: we will assume that
  - clock period is sufficiently long for all of and the following steps to be "completed":
    - 1. instruction fetch
    - 2. decode and register fetch
    - 3. ALU operation
    - 4. data fetch if required
    - 5. register write-back setup time
  - $t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$

Princeton Microarchitecture

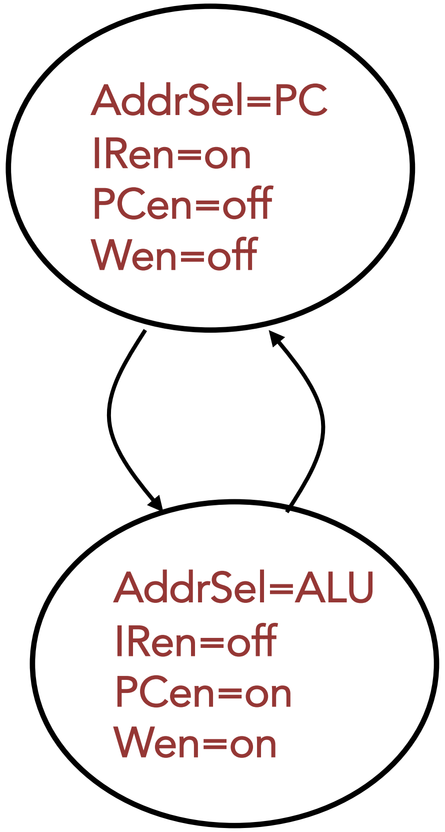


Two-State Controller

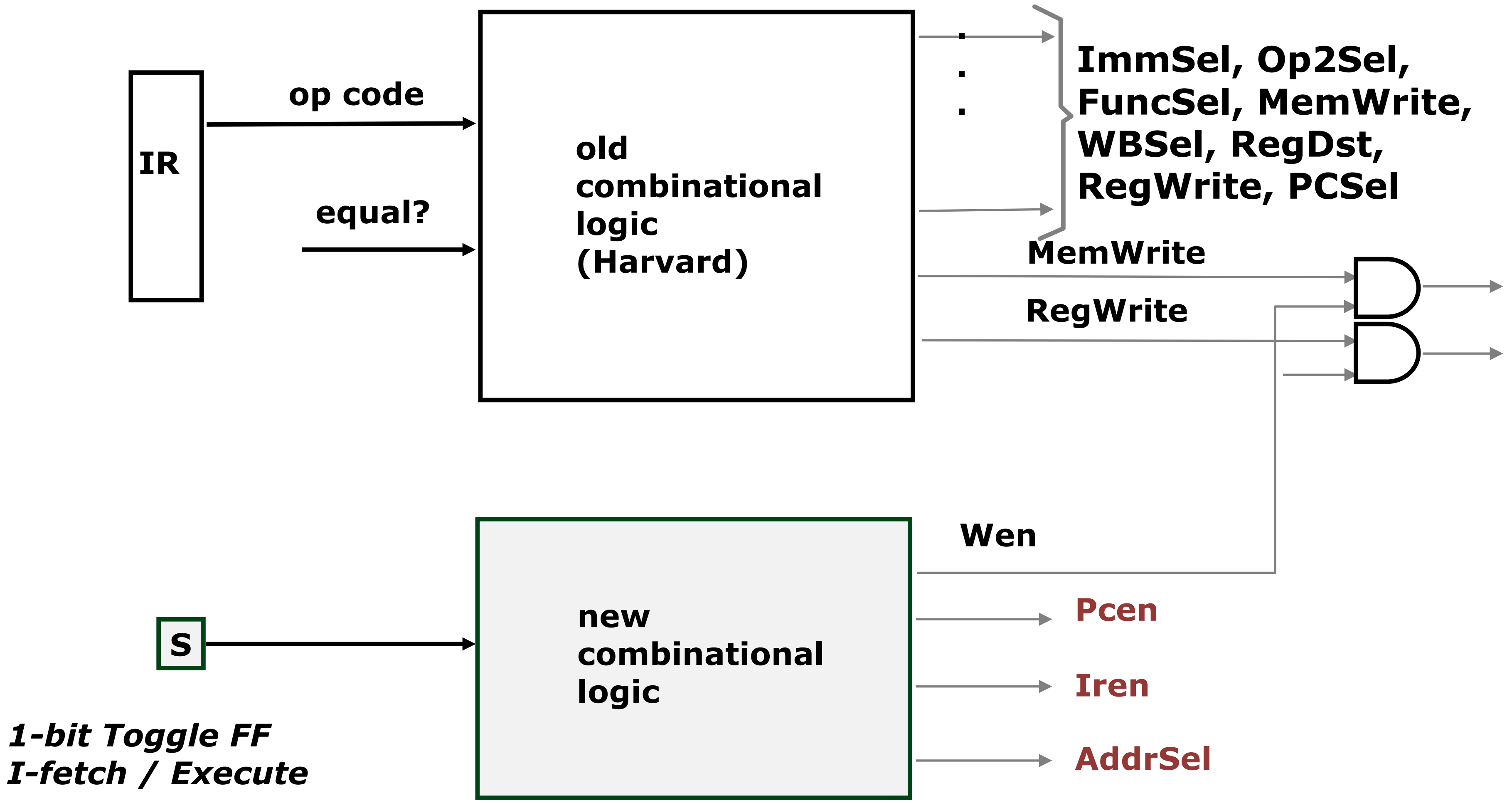
- In the Princeton Microarchitecture, a flipflop can be used to remember the phase

fetch phase

execute phase



Hardwired Controller



Clock Period

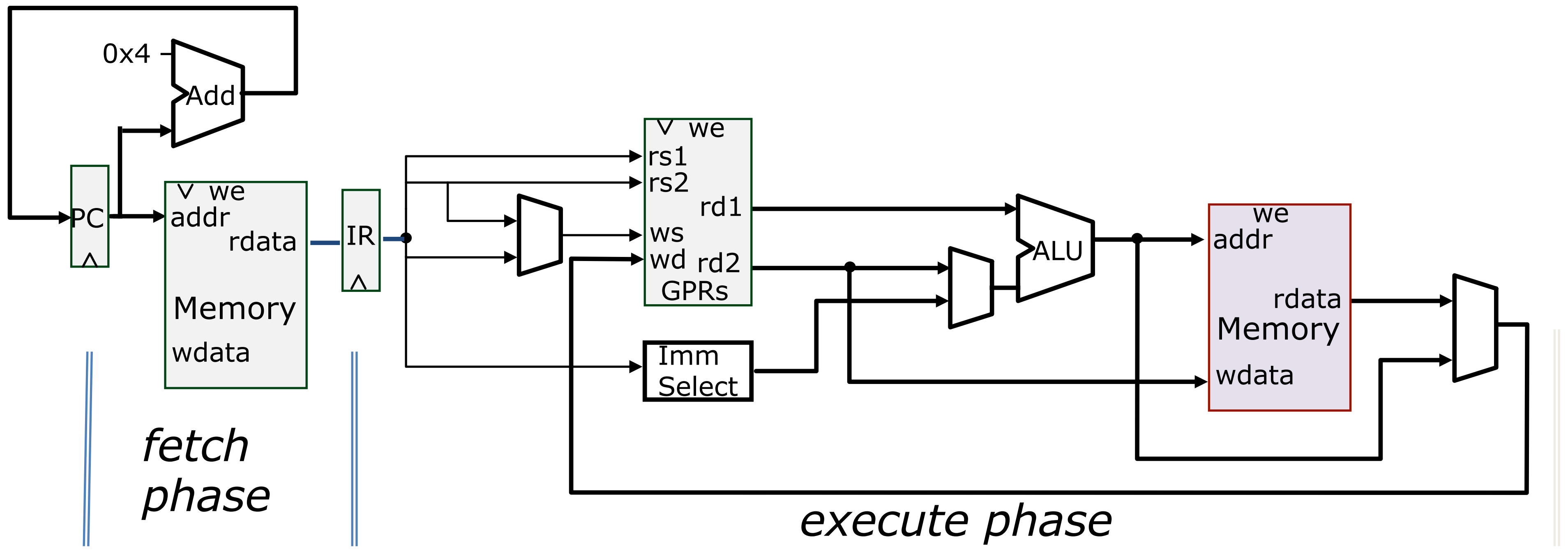
- Princeton architecture
  - $t_{C\text{-}Princeton} > \max\{t_M\ ,\ t_{RF} + t_{ALU} + t_M + t_{WB}\}$
  - $t_{C\text{-}Princeton} > t_{RF} + t_{ALU} + t_M + t_{WB}$
- while in the hardwired Harvard architecture
  - $t_{C\text{-}Harvard} > t_M + t_{RF} + t_{ALU} + t_M + t_{WB}$
- *which will execute instructions faster?*

Clock Rate vs CPI

- Suppose $t_{M} >> t_{RF} + t_{ALU} + t_{WB}$
  - $t_{C\text{-}Princeton} = 0.5 * t_{C\text{-}Harvard}$
    - $CPI_{Princeton} = 2$
    - $CPI_{Harvard} = 1$
- *No difference in performance!*

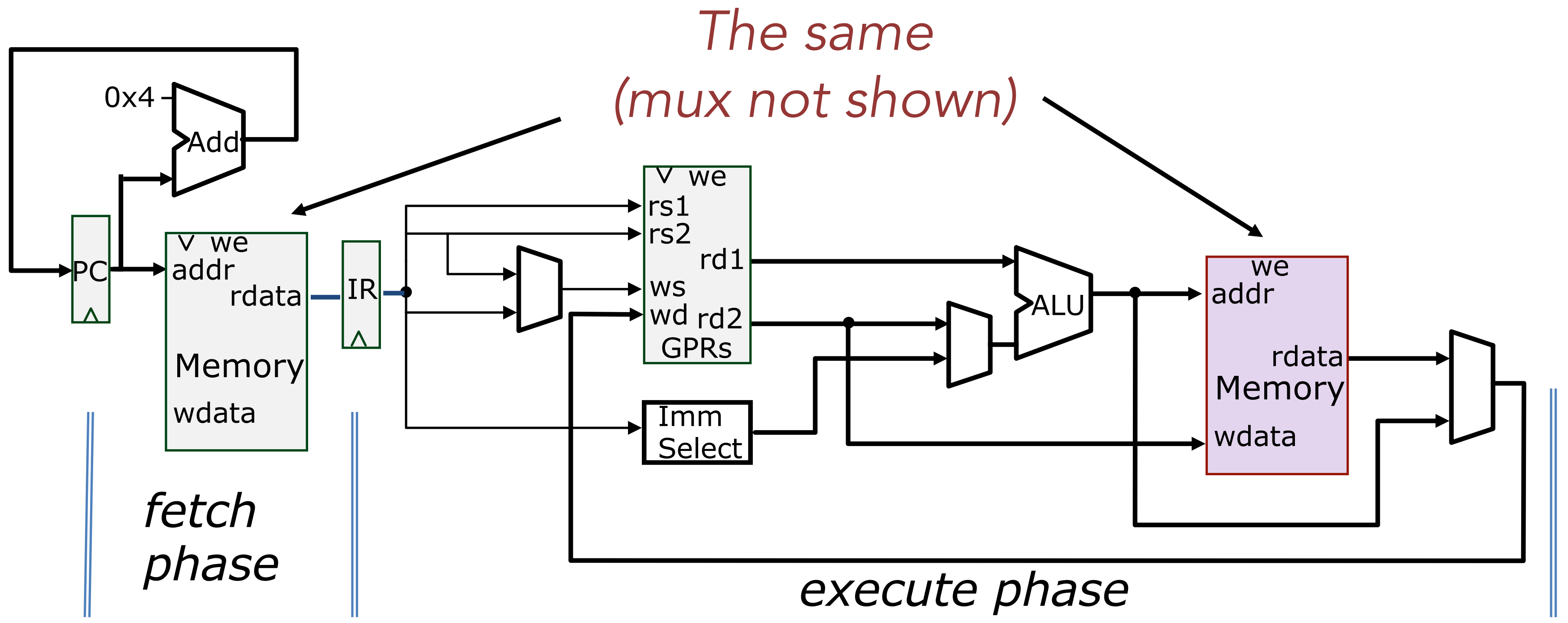
Princeton Microarchitecture

- *Can we overlap instruction fetch and execute?*

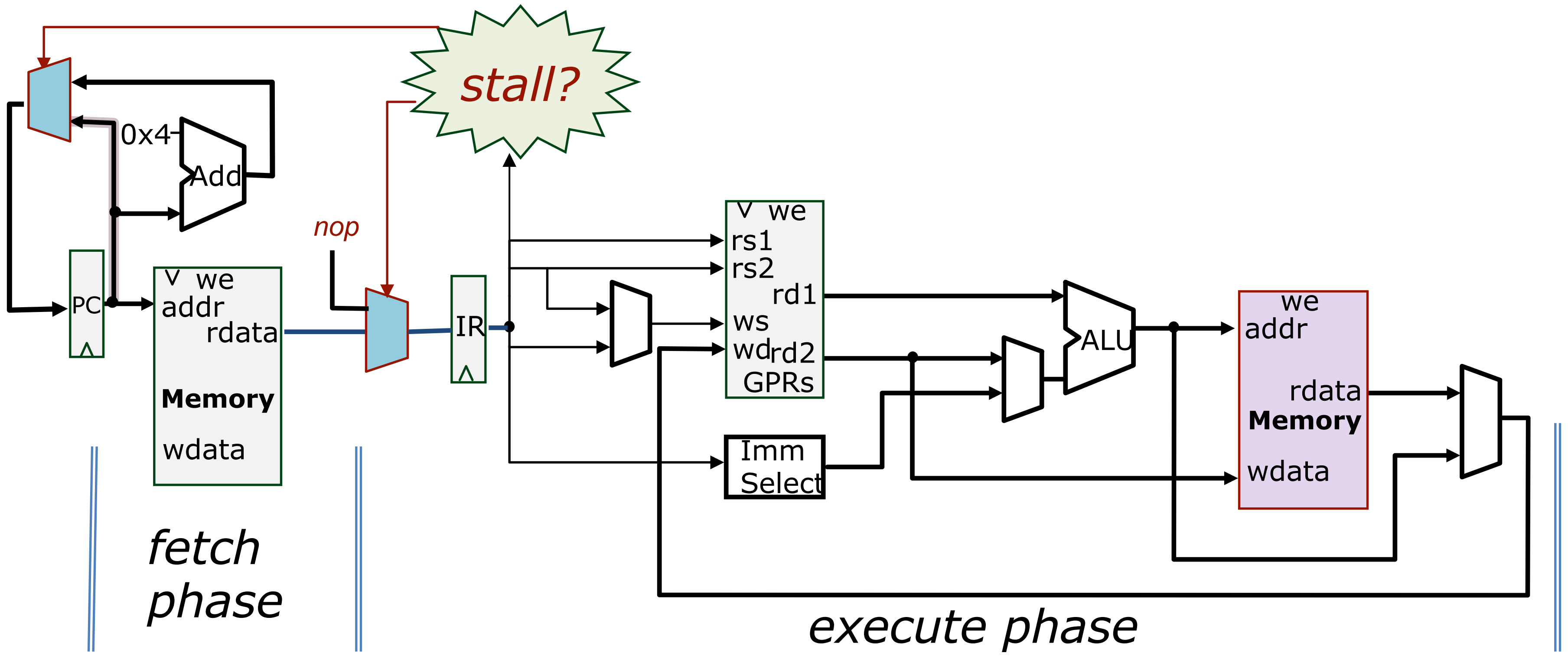


Princeton Microarchitecture

- Only one of the phases is active in any cycle
  - A lot of datapath is not in use at any given time



Stalling the instruction fetch



- When stall condition is indicated
  - Do not fetch a new instruction and do not change the PC
  - Insert a nop in the IR
  - Set the Memory Address mux to ALU (not shown)

Pipelined Princeton Architecture

- *Clock:* $t_{C\text{-}Princeton} > t_{RF} + t_{ALU} + t_{M}$
- *CPI:* $(1 - f) + 2f$ cycles per instruction where f is the fraction of instructions that cause a stall

Compiler Effects on Performance

- CPU time = Instruction count x CPI / Clock rate
  - A machine running at 100 MHz has these instruction classes

Instruction class	CPI
A	1
B	2
C	3

  - For a given program, two compilers produced the following instruction counts

	Instruction counts (in millions) for each instruction class		
Code from:	A	B	C
Compiler 1	50	10	10
Compiler 2	100	10	10

Compiler Effects on Performance

- CPU time = Instruction count x CPI / Clock rate
- For compiler 1:
  - CPI_1 = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.43
  - CPU $time_1$ = ((50 + 10 + 10) x 10^6 x 1.43) / (100 x 10^6) = 1 second
- For compiler 2:
  - CPI_2 = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25
  - CPU $time_2$ = ((100 + 10 + 10) x 10^6 x 1.25) / (100 x 10^6) = 1.5 seconds

Processor Performance

- Speed Up Equations for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{Average Stall cycle per Instruction}$$

$$\text{Speedup} = \frac{\text{Ideal CPI X Pipeline Depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \text{ X } \frac{\text{Clock Cycle}_{Unpipelined}}{\text{Clock Cycle}_{Pipelined}}$$

- If Ideal CPI = 1
  - Speed Up <= Pipeline Depth

$$\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \text{ X } \frac{\text{Clock Cycle}_{Unpipelined}}{\text{Clock Cycle}_{Pipelined}}$$

Illustrative Example

- We want to compare the performance of two machines. Which machine is faster?
  - Machine A: Dual ported memory - so there are no memory stalls
  - Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Assumptions
  - Ideal CPI = 1 for both
  - Loads are 40% of instructions executed

Illustrative Example

- We want to compare the performance of two machines. Which machine is faster?
  - Machine A: Dual ported memory - so there are no memory stalls
  - Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Assumptions
  - Ideal CPI = 1 for both
  - Loads are 40% of instructions executed

Machine A speed $= \text{Pipeline Depth}/(1 + 0) \times (\text{clock}_{\text{unpipeline}}/\text{clock}_{\text{pipeline}})$
$= \text{Pipeline Depth}$

Machine B speed $= \text{Pipeline Depth}/(1 + 0.4 \times 1) \times (\text{clock}_{\text{unpipeline}}/\text{clock}_{\text{pipeline}})$
$= (\text{Pipeline Depth}/1.4) \times (\text{clock}_{\text{unpipeline}}/(1.05 * \text{clock}_{\text{unpipeline}}))$
$= 0.68 \times \text{Pipeline Depth}$

A Speed/ B Speed = Pipeline Depth / (0.68 x Pipeline Depth) = 1.47

Amdahl's Law

- By Gene Amdahl
- This law answers the critical question:
  - How much of a speedup one can get for a given architectural improvement/enhancement?
    - The performance enhancement possible due to a given design improvement is limited by the amount that the improved feature is used
  - Performance improvement or speedup due to enhancement E

$$\text{Speedup(E)} = \frac{\text{Execution Time without E}}{\text{Execution Time with E}} = \frac{\text{Performance with E}}{\text{Performance without E}}$$

Amdahl's Law

- By Gene Amdahl
- This law answers the critical question:
  - How much of a speedup one can get for a given architectural improvement/enhancement?
  - Suppose that enhancement E accelerates a fraction F of the execution time by a factor S and the remainder of the time is unaffected then:
    - Execution Time with E = ((1-F) + F/S) x Execution Time without E
    - Hence speedup is given by:

$$\text{Speedup(E)} = \frac{\text{Execution Time without E}}{((1 - F) + F/S) \times \text{Execution Time without E}} = \frac{1}{(1 - F) + F/S}$$

Amdahl's Law

- For the RISC machine with the following instruction composition:

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement

Amdahl's Law

- For the RISC machine with the following instruction composition:

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement

Fraction enhanced = F = 45% or .45
Unaffected fraction = 100% - 45% = 55% or .55
Factor of enhancement = 5/2 = 2.5

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = \frac{1}{.55 + .45/2.5} = 1.37$$

Amdahl's Law

- For the RISC machine with the following instruction composition:

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement

Amdahl's Law

- For the RISC machine with the following instruction composition:

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement

Old CPI = 2.2

New CPI = .5 x 1 + .2 x 2 + .1 x 3 + .2 x 2 = 1.6

$$\text{Speedup(E)} = \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\text{Instruction count} \times \text{old CPI} \times \text{clock cycle}}{\text{Instruction count} \times \text{new CPI} \times \text{clock cycle}}$$

$$= \frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37$$

Amdahl's Law

- A program takes 100 seconds to execute on a machine with load operations responsible for 80 seconds of this time. By how much must the load operation be improved to make the program four times faster?

Amdahl's Law

- A program takes 100 seconds to execute on a machine with load operations responsible for 80 seconds of this time. By how much must the load operation be improved to make the program four times faster?

$$\text{Desired speedup} = 4 = \frac{100}{\text{Execution Time with enhancement}}$$

Execution time with enhancement = 100 * (1/4) = 25 seconds

$\Rightarrow$ 25 seconds = (100 - 80 seconds) + 80 seconds / n

$\Rightarrow$ 25 seconds = 20 seconds + 80 seconds / n

$\Rightarrow$ 5 = 80 seconds / n

$\Rightarrow$ n = 80/5 = 16

Load operation should be 16 times faster to get a speedup of 4!

Amdahl's Law

- A program takes 100 seconds to execute on a machine with load operations responsible for 80 seconds of this time. By how much must the load operation be improved to make the program five times faster?

Amdahl's Law

- A program takes 100 seconds to execute on a machine with load operations responsible for 80 seconds of this time. By how much must the load operation be improved to make the program five times faster?

$$\text{Desired speedup} = 5 = \frac{100}{\text{Execution Time with enhancement}}$$

Execution time with enhancement = 100 * (1/5) = 20 seconds

$\Rightarrow$ 20 seconds = (100 - 80 seconds) + 80 seconds / n

$\Rightarrow$ 20 seconds = 20 seconds + 80 seconds / n

$\Rightarrow$ 0 = 80 seconds / n

- No amount of load operation improvement will be able achieve this speed

Multiple Enhancements

- Suppose that enhancement E_i accelerates a fraction F_i of the execution time by a factor S_i and the remainder of the time is unaffected then:

$$\textit{Speedup} = \frac{\textit{Original Execution Time}}{\left(\left(1 - \sum_i F_i\right) + \sum_i \frac{F_i}{S_i}\right) \times \textit{Original Execution Time}}$$

$$\textit{Speedup} = \frac{1}{\left(\left(1 - \sum_i F_i\right) + \sum_i \frac{F_i}{S_i}\right)}$$

Multiple Enhancements

- Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

$\text{Speedup}_1 = S_1 = 10$	$\text{Percentage}_1 = F_1 = 20\%$
$\text{Speedup}_2 = S_2 = 15$	$\text{Percentage}_1 = F_2 = 15\%$
$\text{Speedup}_3 = S_3 = 30$	$\text{Percentage}_1 = F_3 = 10\%$

- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.
- What is the resulting overall speedup?

$$\boldsymbol{Speedup} = \frac{1}{\left(\left(1-\sum_i \boldsymbol{F}_i\right)+\sum_i \frac{\boldsymbol{F}_i}{\boldsymbol{S}_i}\right)}$$

Multiple Enhancements

- Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

  $\text{Speedup}_1 = S_1 = 10 \quad \text{Percentage}_1 = F_1 = 20\%$
  $\text{Speedup}_2 = S_2 = 15 \quad \text{Percentage}_1 = F_2 = 15\%$
  $\text{Speedup}_3 = S_3 = 30 \quad \text{Percentage}_1 = F_3 = 10\%$

- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.
- What is the resulting overall speedup?

$$\boldsymbol{Speedup} = \frac{1}{\left(\left(1-\sum_i \boldsymbol{F}_i\right)+\sum_i \frac{\boldsymbol{F}_i}{\boldsymbol{S}_i}\right)}$$

- Speedup = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30)]
  = 1 / [.55 + .0333]
  = 1 / .5833 = 1.71

Amdahl's Law

- Key Insights
  - The performance of any system is constrained by the speed or capacity of the slowest point
  - The impact of an effort to improve the performance of a program is primarily constrained by the amount of time that the program spends in parts of the program NOT TARGETED by the effort
  - Amdahl's Law is a statement of the maximum theoretical speed-up you can ever hope to achieve
  - The actual speed-ups are always less than the speed-up predicted by Amdahl's Law

Amdahl's Law

- For software and hardware engineers MUST have a very deep understanding of Amdahl's Law if they are to avoid having unrealistic performance expectations
  1. For systems folks: this law allows you to estimate the net performance benefit a new hardware feature will add to program executions
  2. For software folks: this law allows you to estimate the amount of parallelism your program/algorithm can achieve before you start writing your parallel code

CPU Performance

- CPU performance factors
  - Instruction count
  - Determined by ISA and compiler
  - CPI and Cycle time
  - Determined by CPU hardware
  - Longest delay determines clock period
    - Critical path: load instruction

CPU Performance

- Longest delay determines clock period
  - Critical path: load instruction
    1. Instruction memory
    2. Register file read
    3. ALU operation
    4. Data memory access
    5. Register file writeback
- Performance can be improved by pipelining

Next Learning Module

- Branch Prediction