ICS 233
COMPUTER ARCHITECTURE

MIPS PROCESSOR
Datapath & Control

Lecture 20

Building ALU Control Unit

- **Implementing ALU Control**
  - **ALU has four control inputs**
  - **Only six of the possible 16 input combinations are used**

ALU Control lines	Functions
0000	AND
0001	OR
0010	add
0110	subtract
0111	Set less than
1100	NOR

- **Depending on the instruction class, the ALU will need to perform one of the five functions**

  - For load word and store word instructions, use the ALU to compute the memory address by addition

  - For the R-type instructions, the ALU needs to perform one of the six actions (AND, OR, subtract, add, set less than, NOR) depending on the value of the 6-bit funct (function) field in the lower order bits of the instruction

  - For branch equal, the ALU must perform a subtraction.

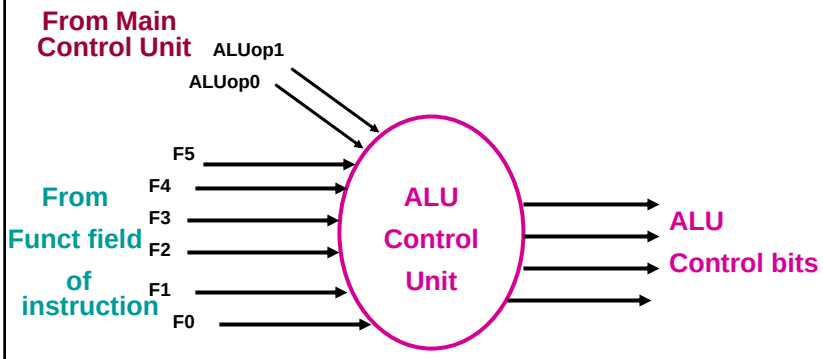
Building ALU Control Unit

➢ **Implementing ALU Control**

- Generate the 4-bit ALU control input using a small control unit that has as inputs the function field of the instruction and a 2-bit control field called ALUOp.
- ALUOp indicates whether operation to be performed should be
  - add (**00**) for loads and stores
  - subtract (**01**) for beq or
  - determined by the operation encoded in the funct field (**10**)
- The output of the ALU Control unit is a 4-bit signal that directly controls the ALU by generating one of the six 4-bit combinations.

Building ALU Control Unit

➢ **Implementing ALU Control**



Building ALU Control Unit

➢ Implementing ALU Control

ALU control bits depend on ALUop control bits and Funct field bits

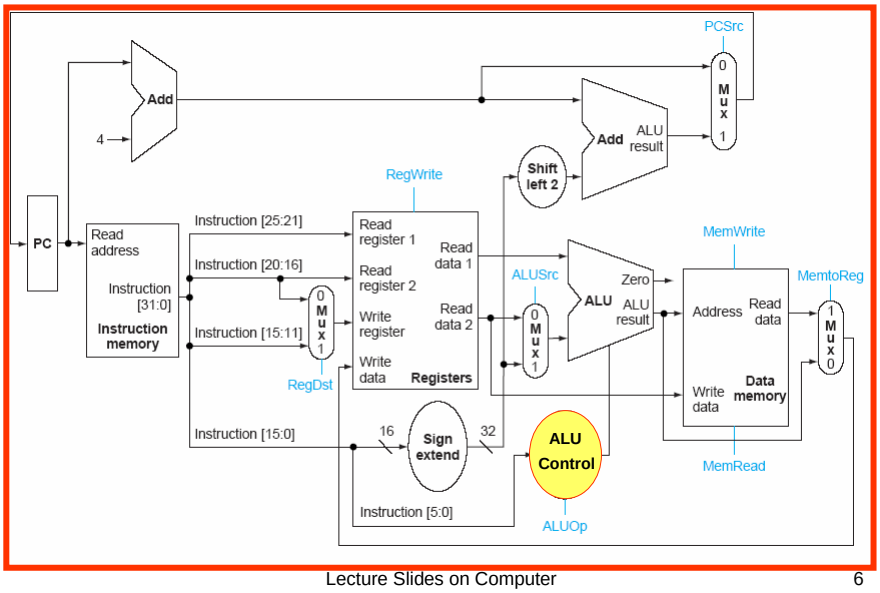
Instruction Opcode	ALUop	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	Load word	xxxxxx	add	0010
SW	00	Store word	xxxxxx	add	0010
Branch equal	01	Branch equal	xxxxxx	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	Set on less than	101010	slt	0111

Truth Table for the four ALU control bits

ALUop		Funct field						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	
0	0	x	x	x	x	x	x	0010
x	1	x	x	x	x	x	x	0110
1	x	x	x	0	0	0	0	0010
1	x	x	x	0	0	1	0	0110
1	x	x	x	0	1	0	0	0000
1	x	x	x	0	1	0	1	0001
1	x	x	x	1	0	1	0	0111

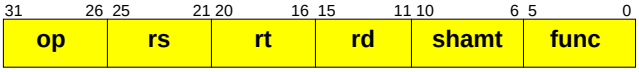
Building ALU Control Unit

➢ Datapath with all necessary multiplexors and all control lines

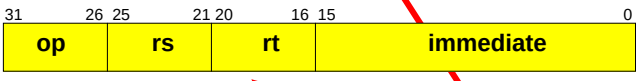


Building ALU Control Unit

R-format



I-format

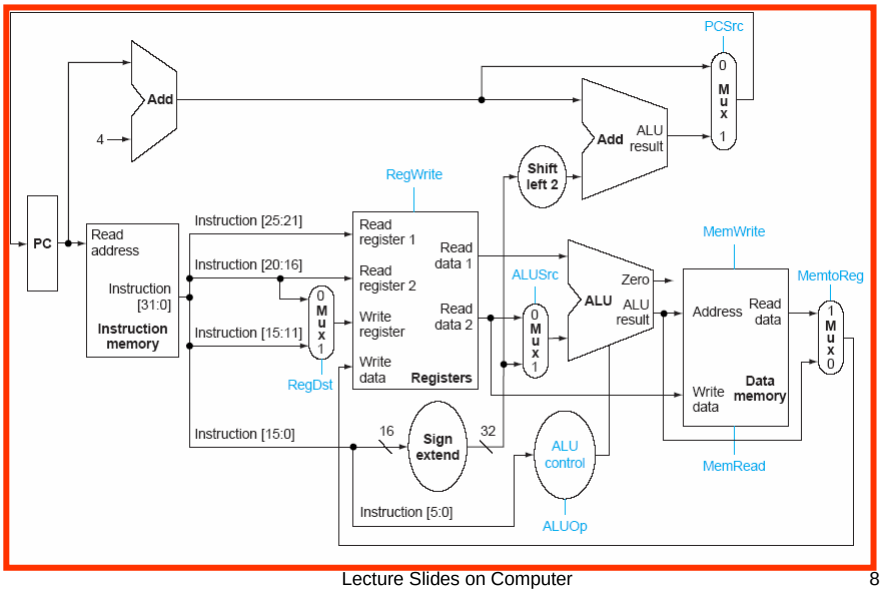


J-format

Destination Register

Building Main Control Unit

- **Datapath with all necessary multiplexors and all control lines**

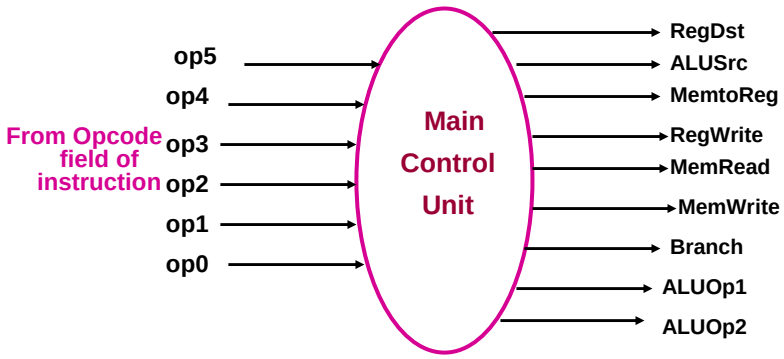


Main Control Signals

- **To specify control for the datapath, appropriate control bit values are needed to be set during each operation**
- **The control lines associated with components active in an operation can be identified as follows :**

RegDst	- To select the Result register
ALUOp	- To select ALU operation
ALUSrc	- To select either Read data 2 or a sign-extended immediate for the ALU
Branch	- Set by branch equal instruction
MemRead	- Set by load instruction
MemWrite	- Set by store instruction
MemtoReg	- decides between sending the ALU result or the memory value to the register file
RegWrite	- writes the chosen value to the Register file

Building Main Control Unit



Building Main Control Unit

Effect of each of the seven control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16)	The register destination number for the Write register comes from the rt field (bits 15:11)
RegWrite	None	The register on the Write register input is written with the value on the Write data input
ALUSrc	The second ALU operand comes from the second register file output (Read data 2)	The second ALU operand is the sign-extended lower 16 bits of the instruction
PCSrc	The PC is replaced by the output of the adder that computes the value of PC +4	The PC is replaced by the output of the adder that computes the branch target
MemRead	None	Data memory contents designated by the address input are put on the Read data output
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input
MemtoReg	The value fed to the register Write data input comes from the ALU	The value fed to the register Write data input comes from the data memory

Building Main Control Unit

Setting of the Control lines Completely determined by the Opcode field bits of the instruction

Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop1	ALUop2
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	x	1	x	0	0	1	0	0	0
beq	x	0	x	0	0	0	1	0	1

Name	Opcode in decimal	Opcode in binary					
		op5	op4	op3	op2	op1	op0
R-format	0	0	0	0	0	0	0
lw	35	1	0	0	0	1	1
sw	43	1	0	1	0	1	1
beq	4	0	0	0	1	0	0

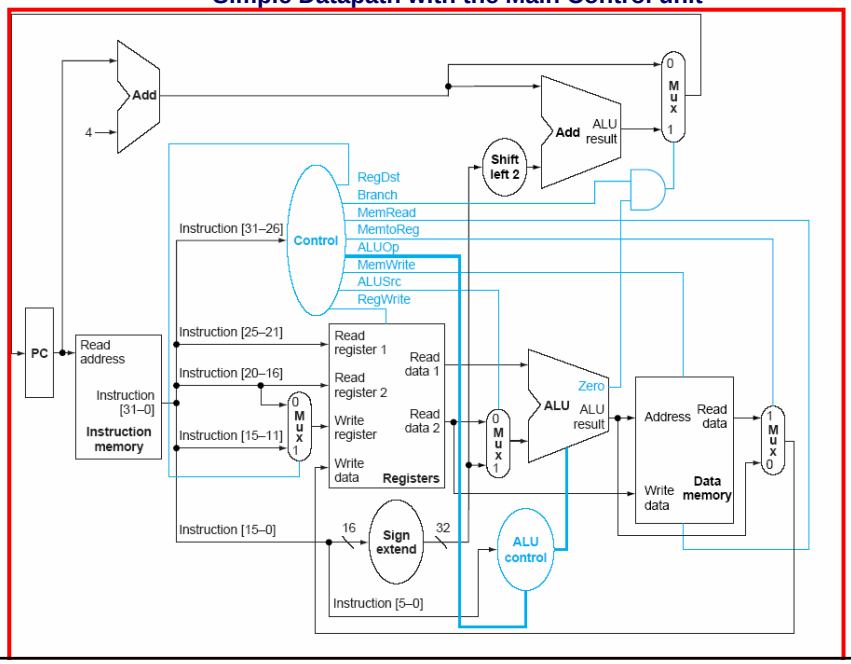
Building Main Control Unit

Truth Table for the Main control lines

Input or Output	Signal name	R-format	lw	sw	beq
Inputs	op5	0	1	1	0
	op4	0	0	0	0
	op3	0	0	1	0
	op2	0	0	0	1
	op1	0	1	1	0
	op0	0	1	1	0
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUop1	1	0	0	0
	ALUop0	0	0	0	1

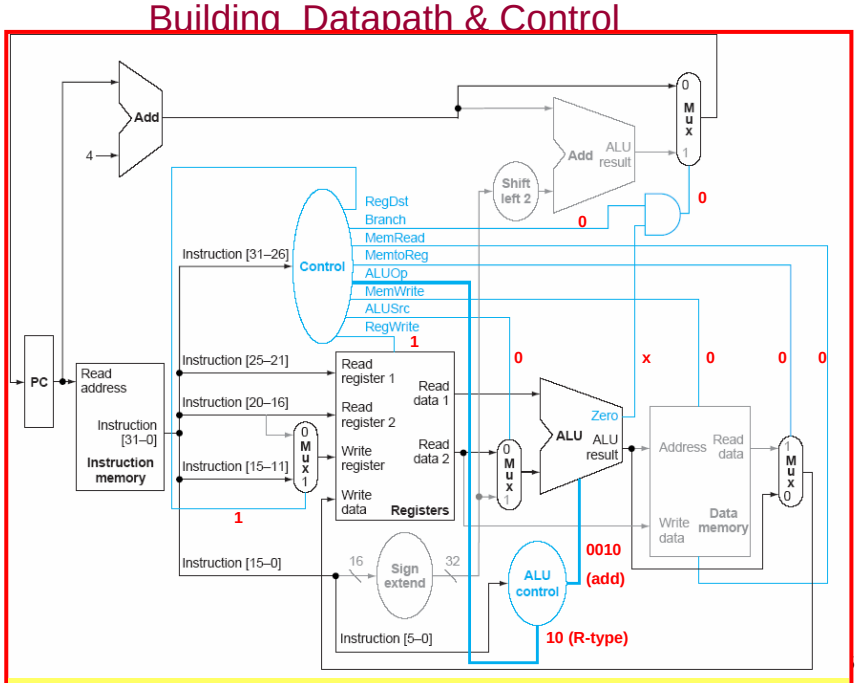
Building Datapath & Control

Simple Datapath with the Main Control unit



Building Datapath & Control

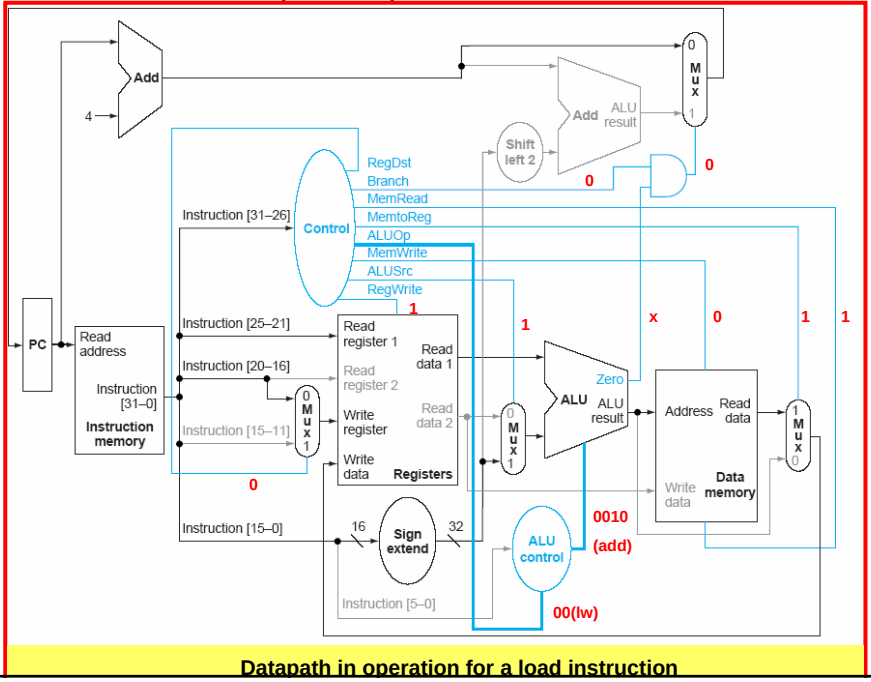
SINGLE CYCLE IMPLEMENTATION



Datapath in operation for an R-type instruction such as add $t1, $t2, $t3

Building Datapath & Control

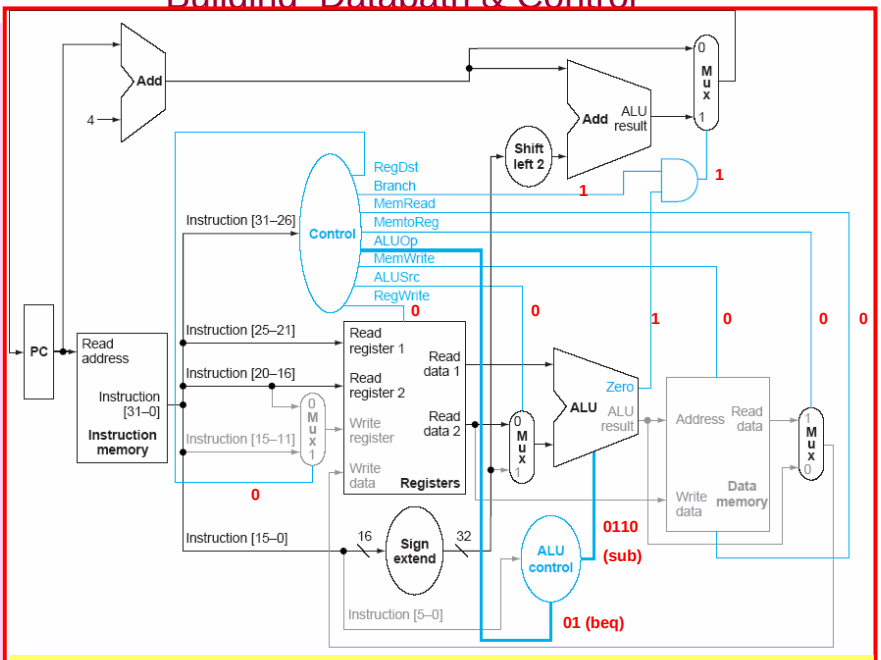
SINGLE CYCLE IMPLEMENTATION



Datapath in operation for a load instruction

Building Datapath & Control

SINGLE CYCLE IMPLEMENTATION



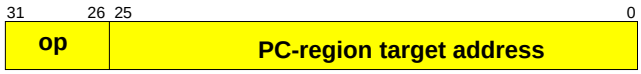
Datapath in operation for a branch equal instruction

Building Datapath & Control

➢ Implementing the Jump Instruction

- **Jump Target address computation is different from branch target address computation**

J-format



Target address = upper 4 bits of (Current PC + 4) ● 4*offset

Concatenation operator

- **New PC value (i.e., target address) is computed by concatenating**

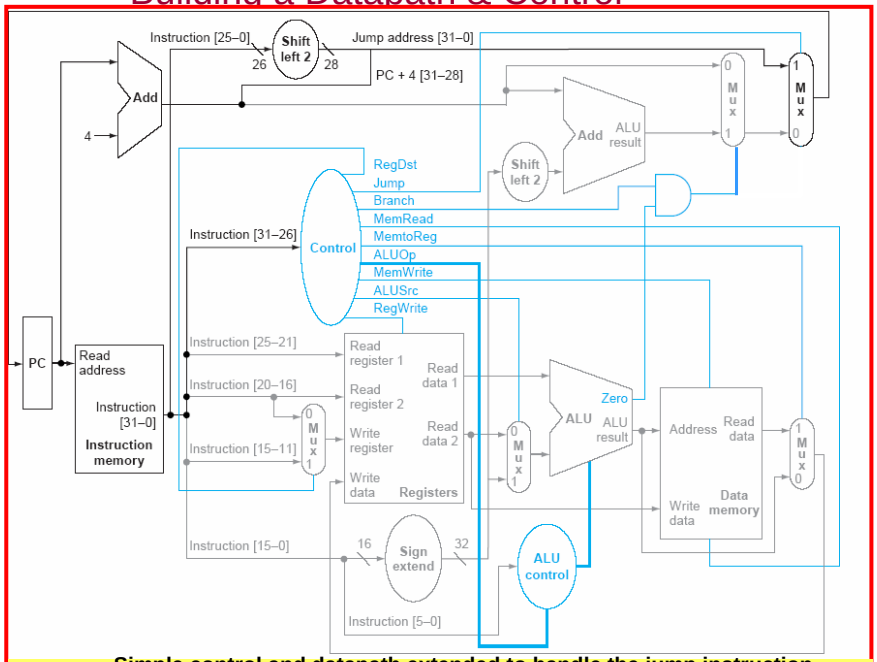
- the upper four bits of the current PC + 4 (form bits 31 to 28 of new PC)

- the 26-bit immediate field bits of the jump instruction (form bits 27 to 2 of new PC)

- the bits 00 (form bits 1 to 0 of new PC)

Building a Datapath & Control

SINGLE CYCLE IMPLEMENTATION



Simple control and datapath extended to handle the jump instruction

Performance of Single-Cycle Machines

- **Question :**

Assume that the operating times for the major functional units in the single-cycle implementation are the following :

- Memory units : 200 picoseconds(ps)
- ALU and adders : 100 ps
- Register file (read or write) : 50 ps

Assume that the multiplexors, control unit, PC access, sign extension unit, and wires 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 instruction executes in 1 clock cycle using a variable-length clock, which for each instruction is only as long as it needs to be.

To compare the performance, assume the following instruction mix : 25% loads, 10% stores, 45% ALU instructions, 15% branches, and 5% jumps.

Performance of Single-Cycle Machines

- **Answer :**

Let us start by comparing the CPU execution times

CPU execution time = Instruction Count x CPI x Clock cycle time

Since single cycle implementation, CPI = 1, then

CPU execution time = Instruction Count x Clock cycle time

We need to find the clock cycle time for the two implementations, since the instruction count and CPI are the same for both the implementations

The critical path for the different instruction classes is as follows :

Instruction class	Functional units used by the instruction class				
R-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				

Performance of Single-Cycle Machines

- **Answer :**

Using these critical paths, we can compute the required length for each instruction class:

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400ps
Load word	200	50	100	200	50	600ps
Store word	200	50	100	200		550ps
Branch	200	50	100	0		350ps
Jump	200					200ps

1. **The clock cycle for a machine with a single clock for all instructions will be determined by the longest instruction, which is 600 ps. (inefficient implementation)**
2. **A machine with a variable clock will have a clock cycle that varies between 200ps and 600ps.**

Now, we can find the average clock cycle length for a machine with a variable-length clock using the information above and the instruction frequency distribution.

Thus, the average time per instruction with a variable clock is :

CPU clock cycle time = 600x25% + 550x10% + 400x45% + 350x15% + 200x5%

= 447.5 ps

Performance of Single-Cycle Machines

❑ **Answer :**

Since the variable clock implementation has a shorter average clock cycle, it is clearly faster.

Performance Ratio :

$$\frac{\text{CPU performance}_{\text{variableclock}}}{\text{CPU performance}_{\text{singleclock}}} = \frac{\text{CPU execution time}_{\text{singleclock}}}{\text{CPU execution time}_{\text{variableclock}}}$$

$$= \frac{\text{IC x CPU clock cycle}_{\text{single clock}}}{\text{IC x CPU clock cycle}_{\text{variable clock}}}$$

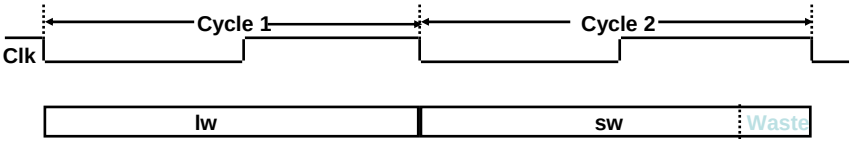
$$= \frac{\text{IC x CPU clock cyclesingle clock}}{\text{IC x CPU clock cyclevariable clock}}$$

= 600/447.5 = 1.34

The variable clock implementation would be 1.34 times faster (extremely difficult to implement)

Single Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instruction
  - especially problematic for more complex instructions like floating point multiply



- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle,
- But implementation Is simple and easy to understand