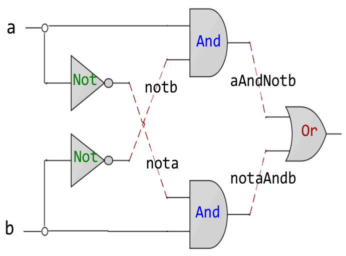
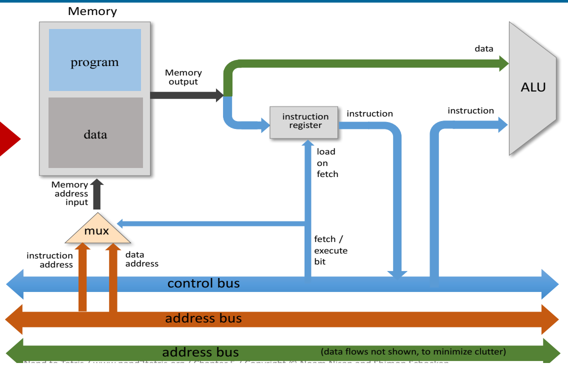
Computer Organization & Assembly Language Programming



```
CHIP Xor {
   IN a, b;
   OUT out;
   PARTS:
   Not(in=a, out=nota);
   Not(in=b, out=notb);
   And(a=nota, b=b, out=w1);
   And(a=a, b=notb, out=w2);
   Or(a=w1, b=w2, out=out);
}
```



```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 27

On Improving Processors Performance

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

For resources visit my personal website:
https://www.arifbutt.me
and course bitbucket repository:
https://bitbucket.org/arifpucit/coal-repo

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- CPU Performance Equation
- Single Cycle vs Multi Cycle CPU Architecture
- Pipelined CPU Architecture
  - Pipeline Stages
  - Pipelined Hazards
  - Solutions of Pipeline Hazards
- CISC vs RISC Architecture

CPU Performance Equation

CPU Performance Equation

$$\frac{time}{program} = \frac{instructions}{program} \times \frac{clock\ cycles}{instructions} \times \frac{time}{clock\ cycles}$$

CPU performance equation analyzes execution time as a product of three factors that are relatively independent of each other, namely Instruction Count (IC), Clock Cycles Per instruction (CPI) and Clock Cycle Time (CCT)

- **Instruction Count**, means number of assembly instructions in the program to perform a specific task
- **Cycles Per Instruction (CPI),** means the number of cycles required to execute one assembly instruction
- **Clock Time or Frequency of CPU Clock,** means the number of times the CPU clock ticks per second, e.g., a clock rate of 1 MHz means, the CPU clock ticks one million times per second. Since one instruction takes a specific number of clock cycles to execute, therefore, the clock time has a direct impact on CPU performance

CPU Performance Equation

$$\frac{time}{program} = \frac{instructions}{program} \times \frac{clock\ cycles}{instructions} \times \frac{time}{clock\ cycles}$$

<u>**Example:**</u>

Suppose a program takes 1 billion instructions to execute on a processor running at 2 GHz. Suppose also that 50% of the instructions execute in 3 clock cycles, 30% execute in 4 clock cycles, and 20% execute in 5 clock cycles. What is the execution time for the program?

<u>**Solution:**</u>

IC = 1.0×10^9

CPI = $0.5 * 3 + 0.3 * 4 + 0.2 * 5 = 3.7$

Clock Time = $1 /(2 \times 10^9) = 0.5 \times 10^{-9}$ seconds

Execution Time = IC x CPI x Clock Time = $10^9 \times 3.7 \times 0.5 \times 10^{-9} = 1.85$ seconds

Single Cycle vs Multi Cycle CPU Architecture

Single Cycle CPU

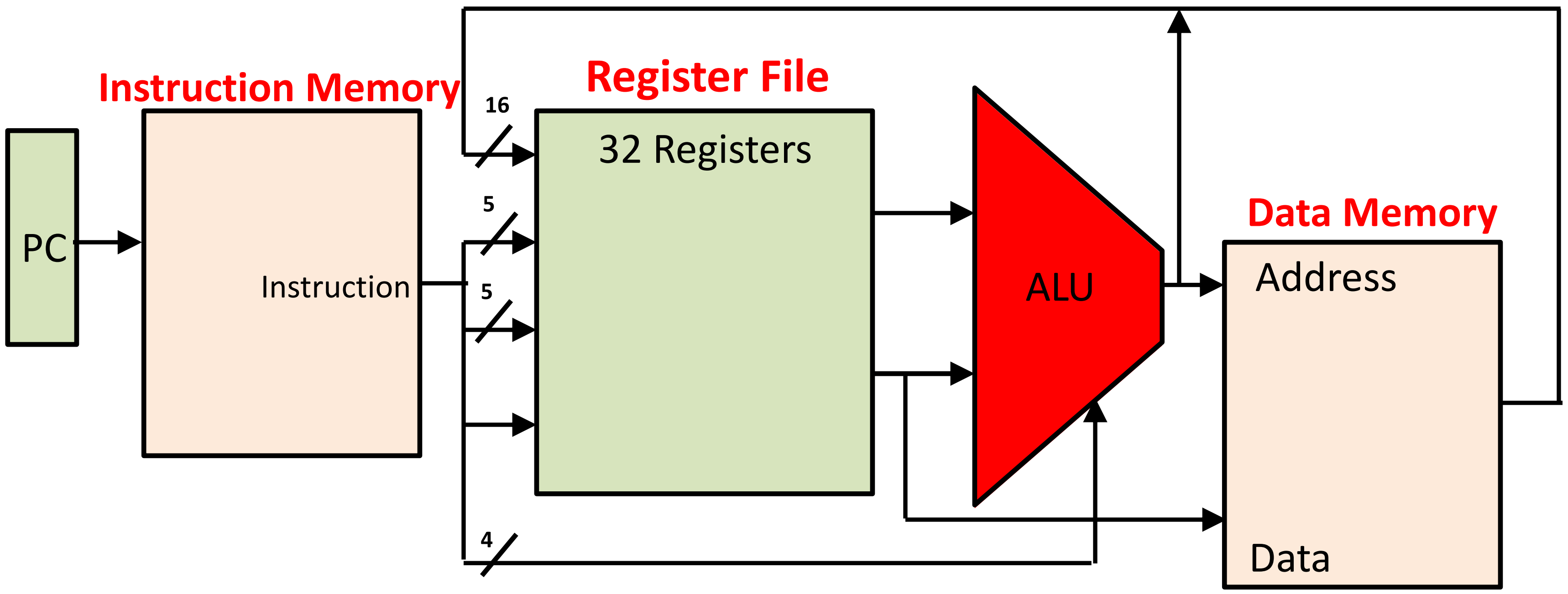
- A **single cycle CPU** executes each instruction in one clock cycle, i.e.,

$$CPI = 1$$

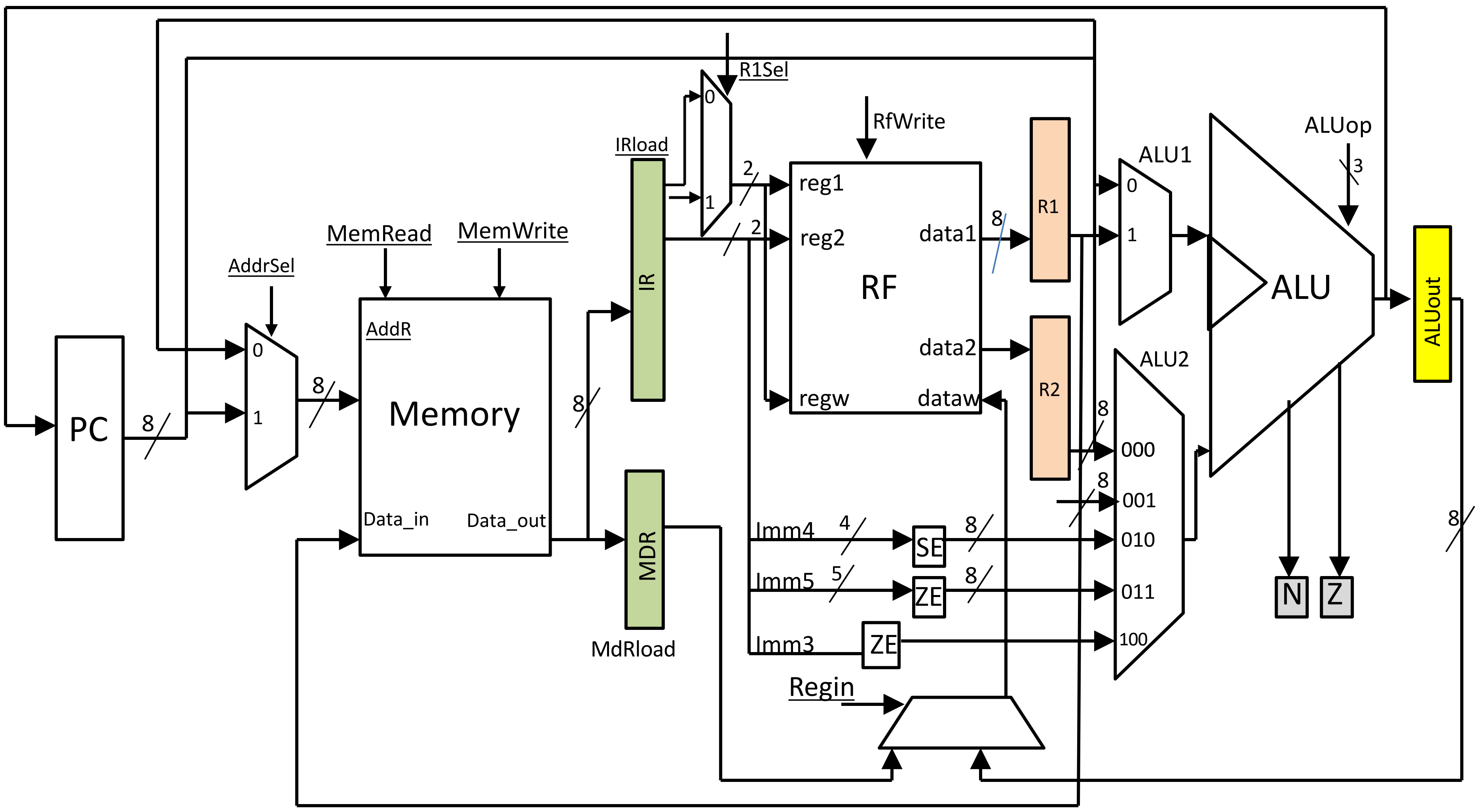
- Irrespective how simple or how complex an instruction is, it will take one clock cycle to execute. So the clock cycle of the CPU has to be large enough to completely execute the most complex or the slowest instruction in the ISA
- The disadvantage of single cycle CPU is that; it must operate at the speed of the slowest instruction (clock cycle is large)
- The advantage of single cycle CPU is that; it is easy to implement

Micro Architecture of a Single Cycle CPU

- Suppose we want to design a CPU whose instruction length is 16 bits. Instruction can have two register operands/addresses or single memory operand/address. 4 bits are reserved for opcode, 5 bits for register address
  - ***Question#1:*** How many CPU registers can be accommodated in this architecture and what can be their size?
  - ***Question#2:*** How many main memory locations can be addressed?
- Below is a diagram showing the data path for single cycle CPU architecture

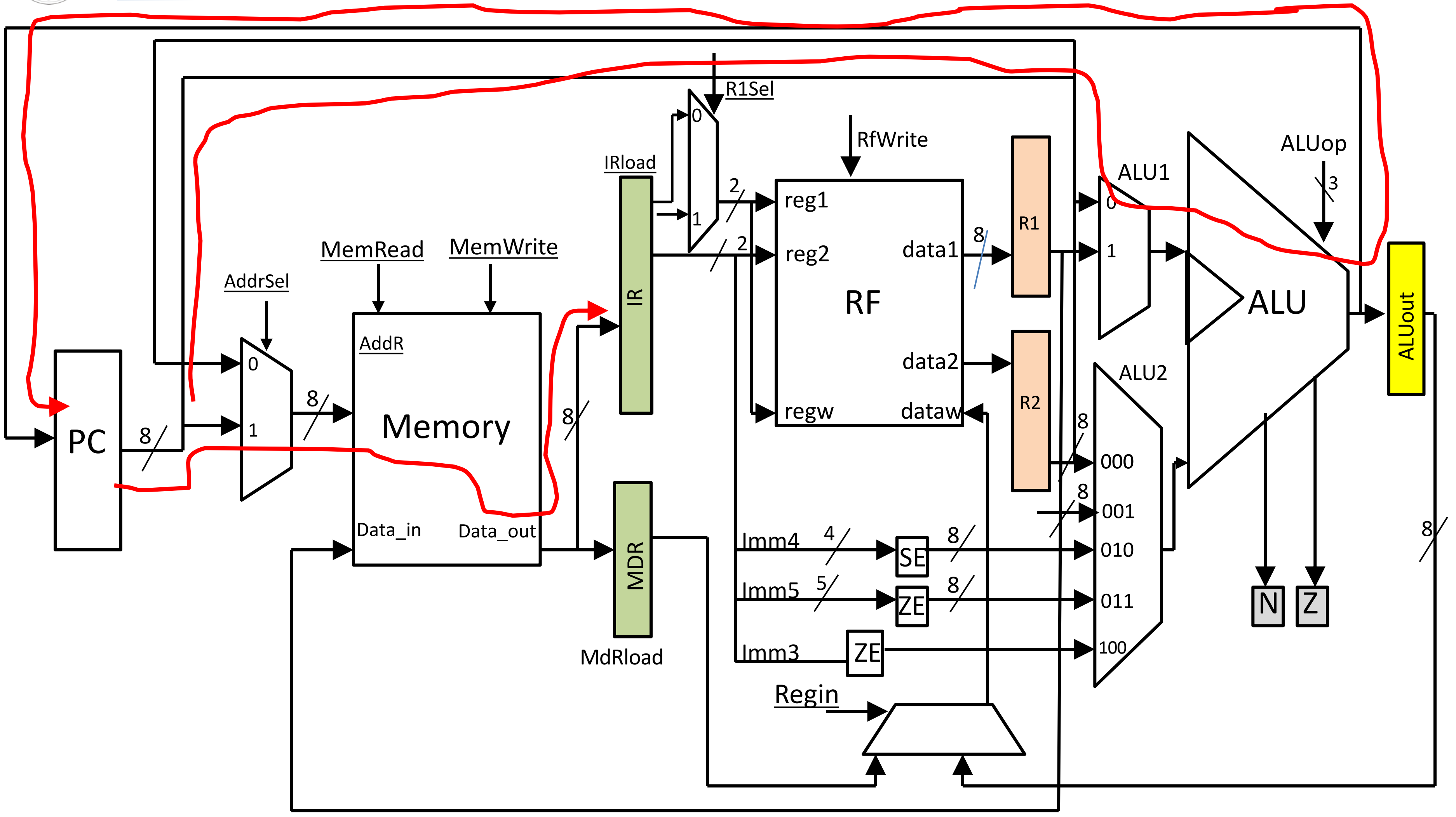


Micro Architecture of Multi Cycle CPU



ADD R5, R1, R2

Cycle 1: Fetching Instruction and Increment PC

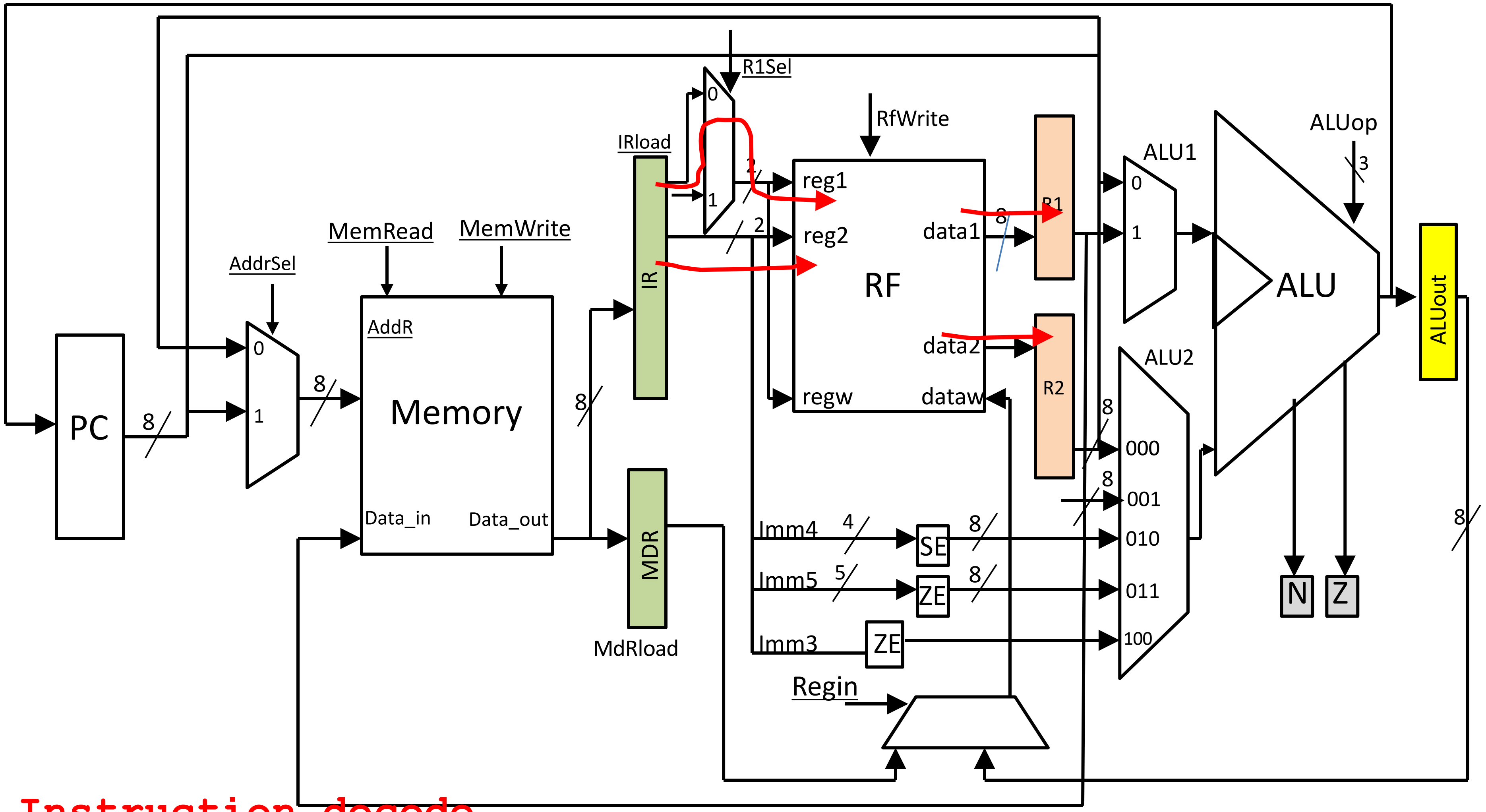


IR = Mem[PC]

[PC] = [PC] +1

ADD R5, R1, R2

Cycle 2: Decoding Instruction and Reading Reg



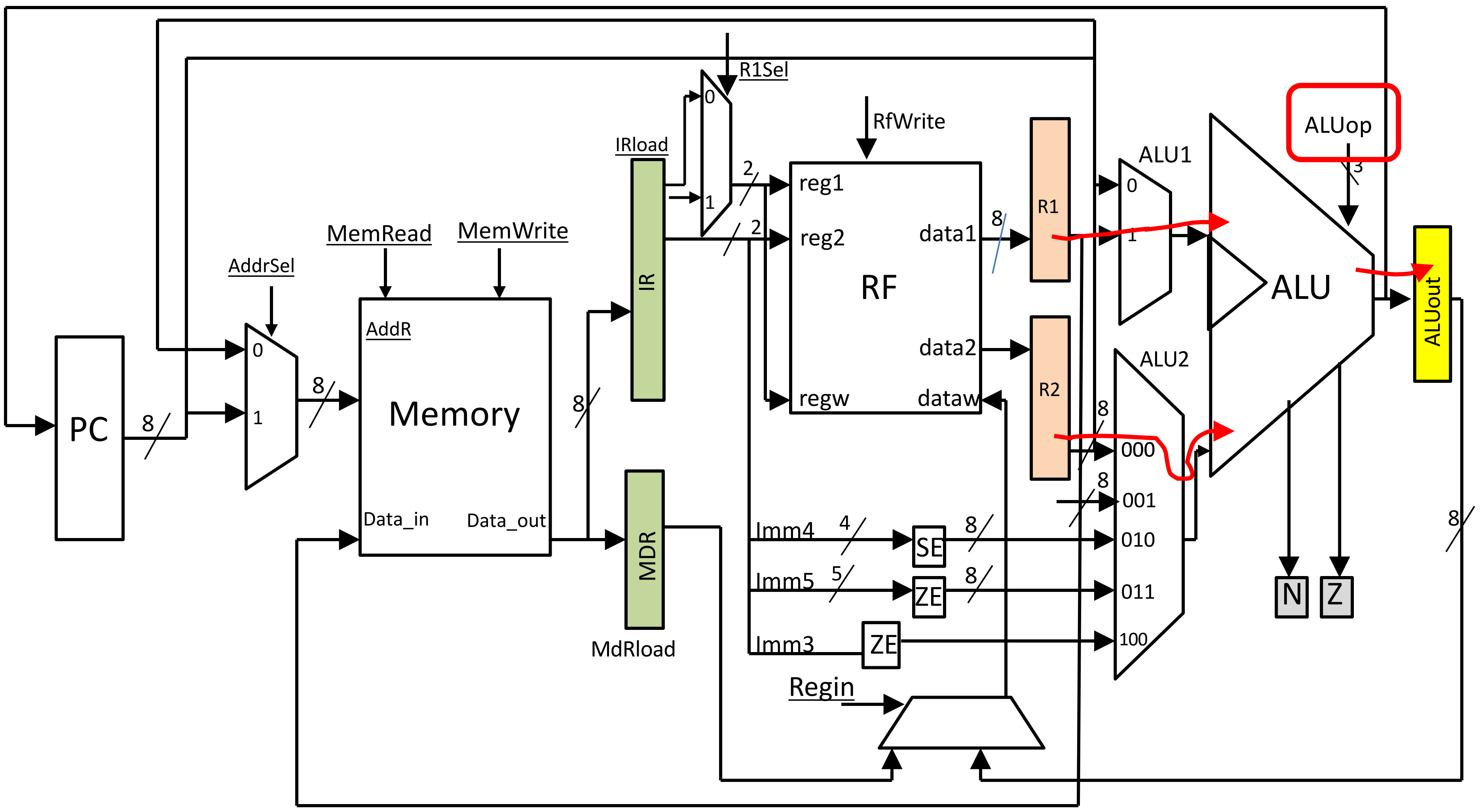
Instruction decode

R1 = RF[7..6]

R2 = RF[5..4]

ADD R5, R1, R2

Cycle 3: Execute Instruction

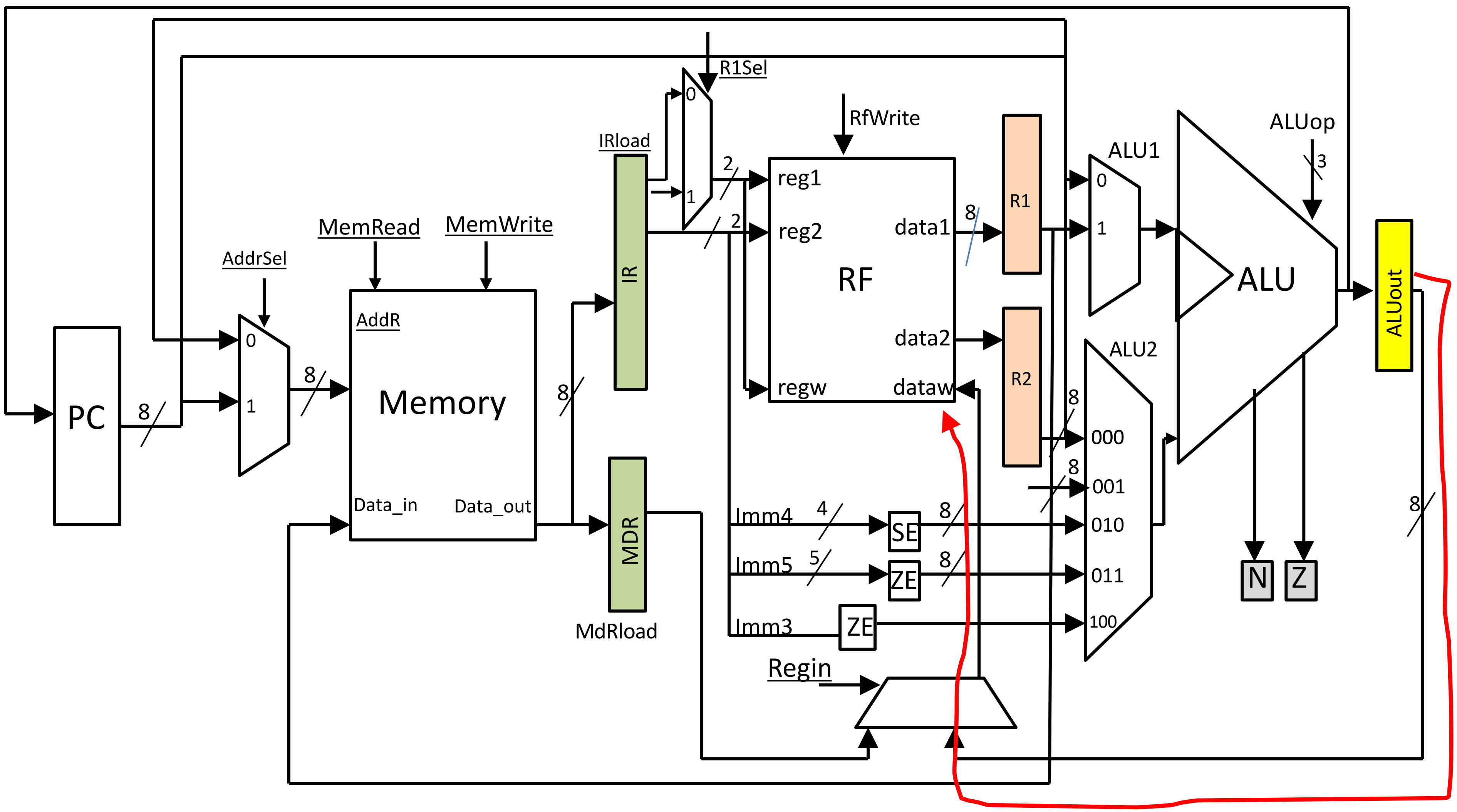


ALUout = R1 + R2

ADD R5, R1, R2

Instructor: Muhammad Arif Butt, Ph.D.

Cycle 4: Write Result in Register



R5 = ALUout

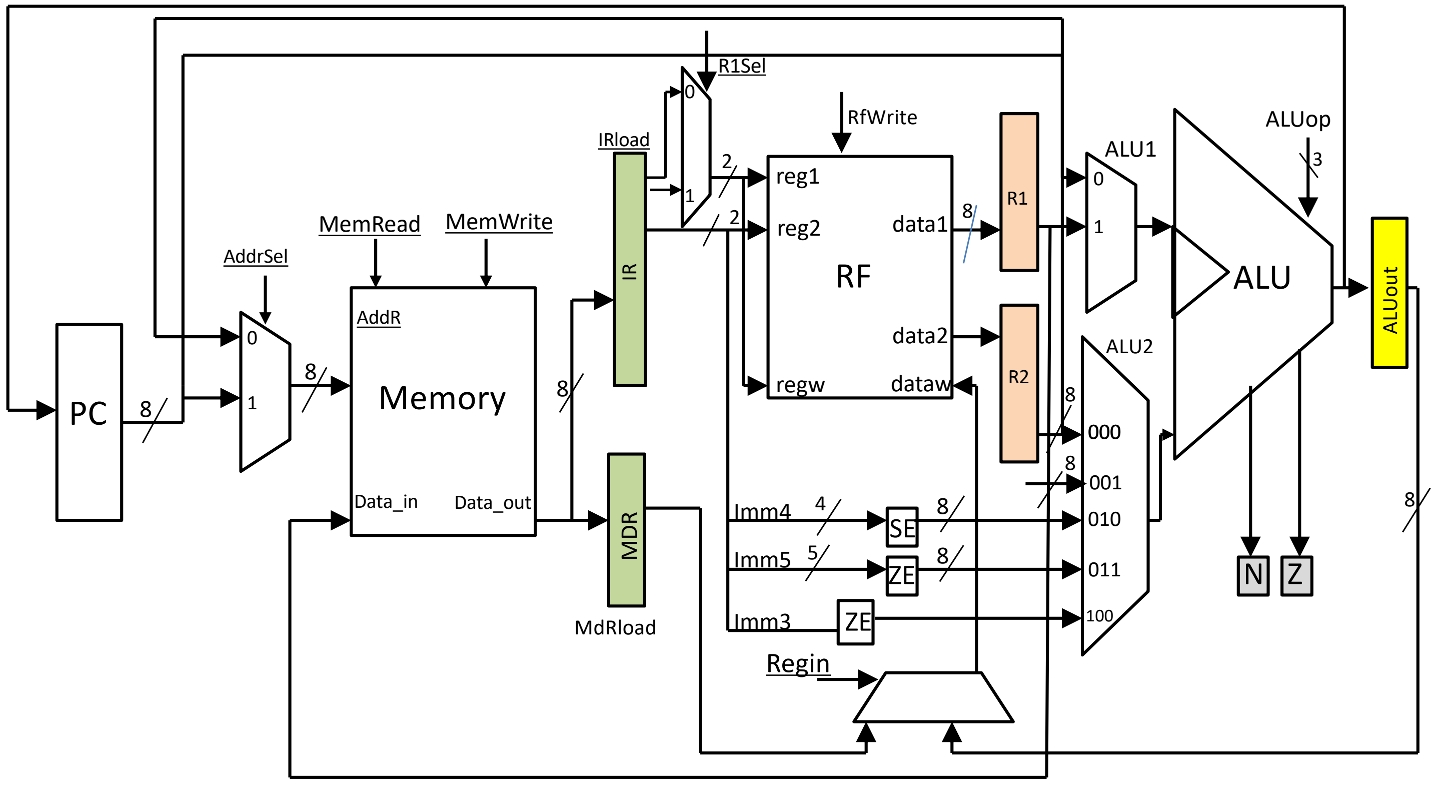
ADD R5, R1, R2

Instructor: Muhammad Arif Butt, Ph.D.

Micro Architecture of Multi Cycle CPU

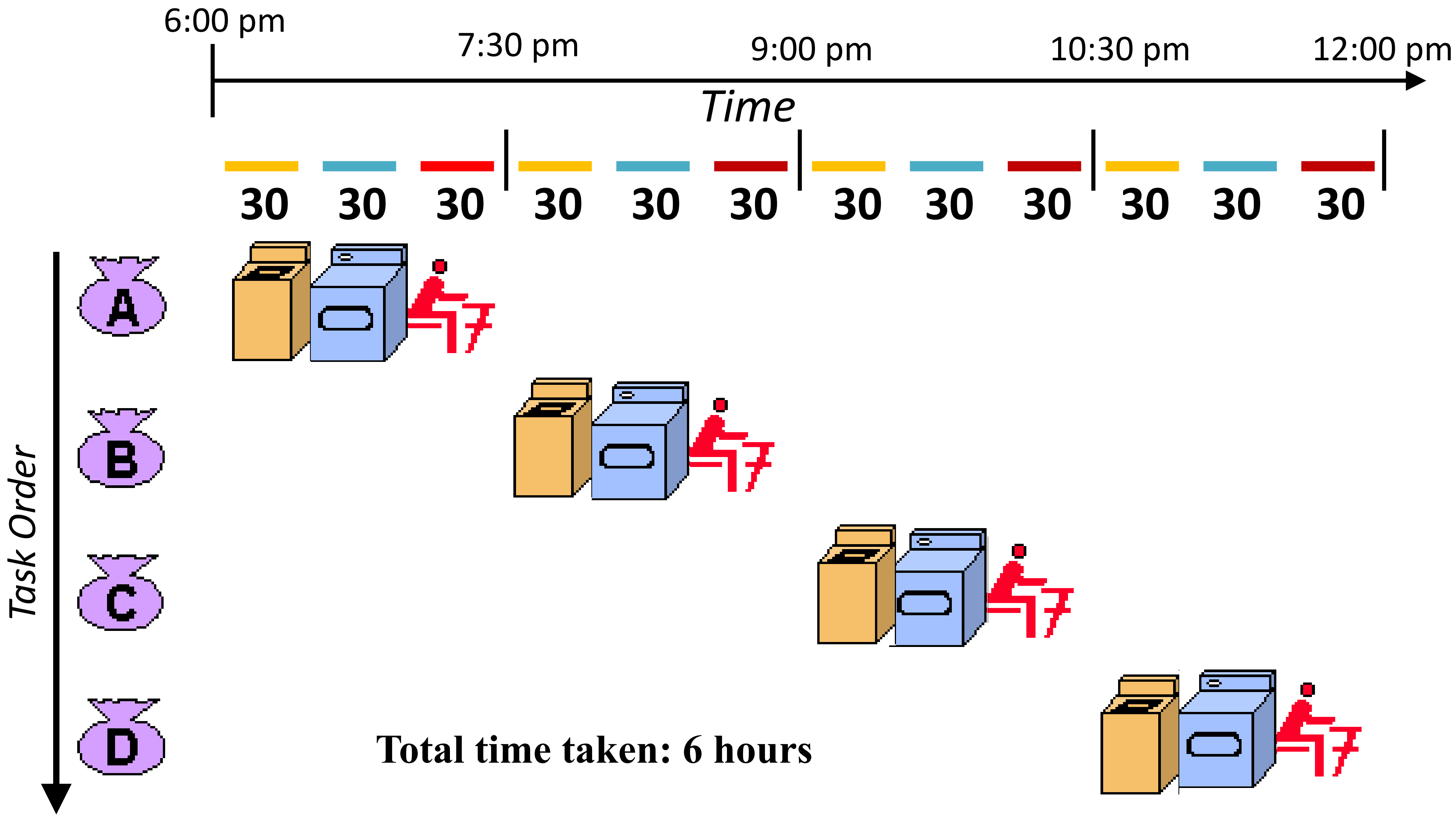
$ 100 Question

Why CPU designers prefer multi-cycle CPU?



Pipelined CPU Architecture

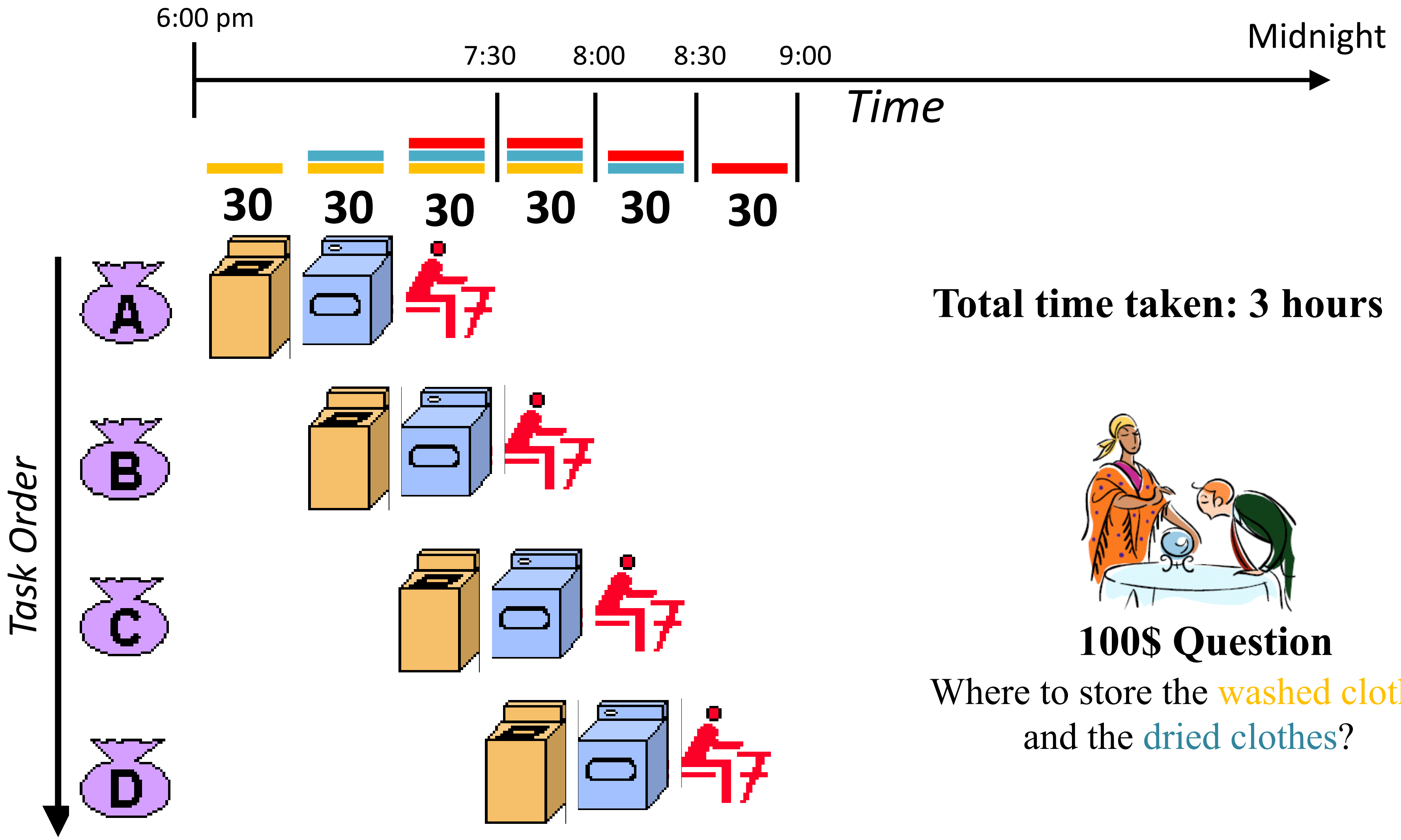
Laundry Analogy



Total time taken: 6 hours

Source: http://www.ece.arizona.edu/~ece462/

Pipeline using Laundry Analogy



Total time taken: 3 hours

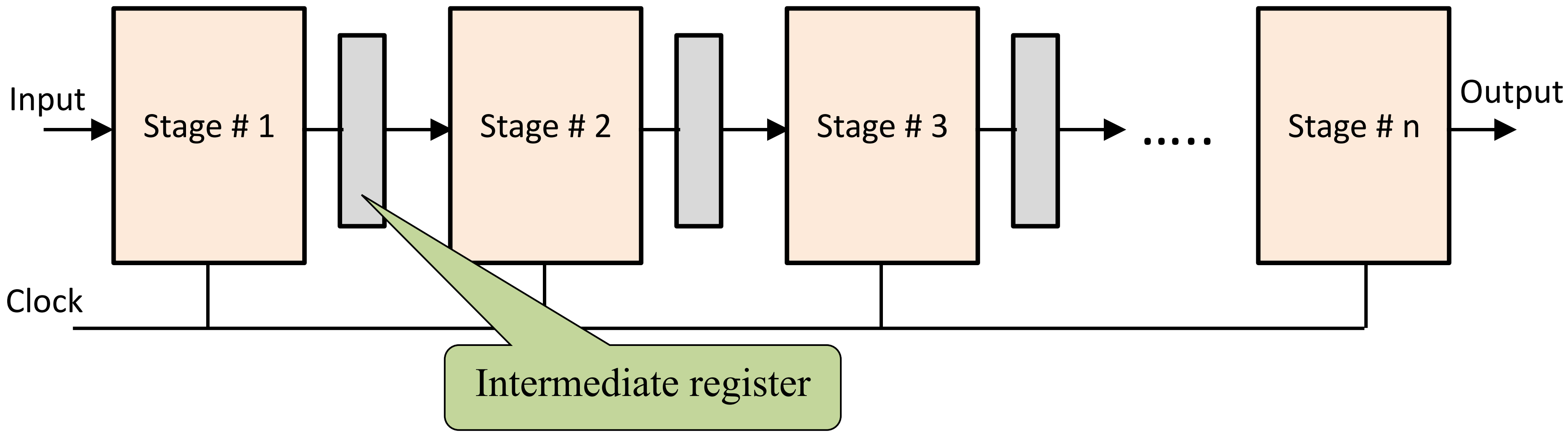
100$ Question

Where to store the washed clothes and the dried clothes?

Source: http://www.ece.arizona.edu/~ece462/

Stages of Pipeline and Intermediate Registers

- Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure
- Input is given from one side and the output of each stage becomes the input of next stage. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput
- A n-stage pipeline with intermediate registers is shown below:



A Simple 5 Stage Pipelined Processor

- In Computer Science, Pipelining is the process of arrangement of hardware elements (functional units) of CPU such that its overall performance is increased. A pipelined processor allow multiple instructions to execute concurrently while each instruction uses a different functional unit in the data path
- The micro-architecture of a 5-stage pipelined processor (**MIPS R3000**) is shown below:

- **IF:** Fetch instruction from memory
- **ID:** Instruction decode and fetch operand from register
- **EX Stage:** Execute the instruction or calculate operand address
- **MEM Stage:** Access an operand from data memory
- **WB Stage:** Write the result into a register

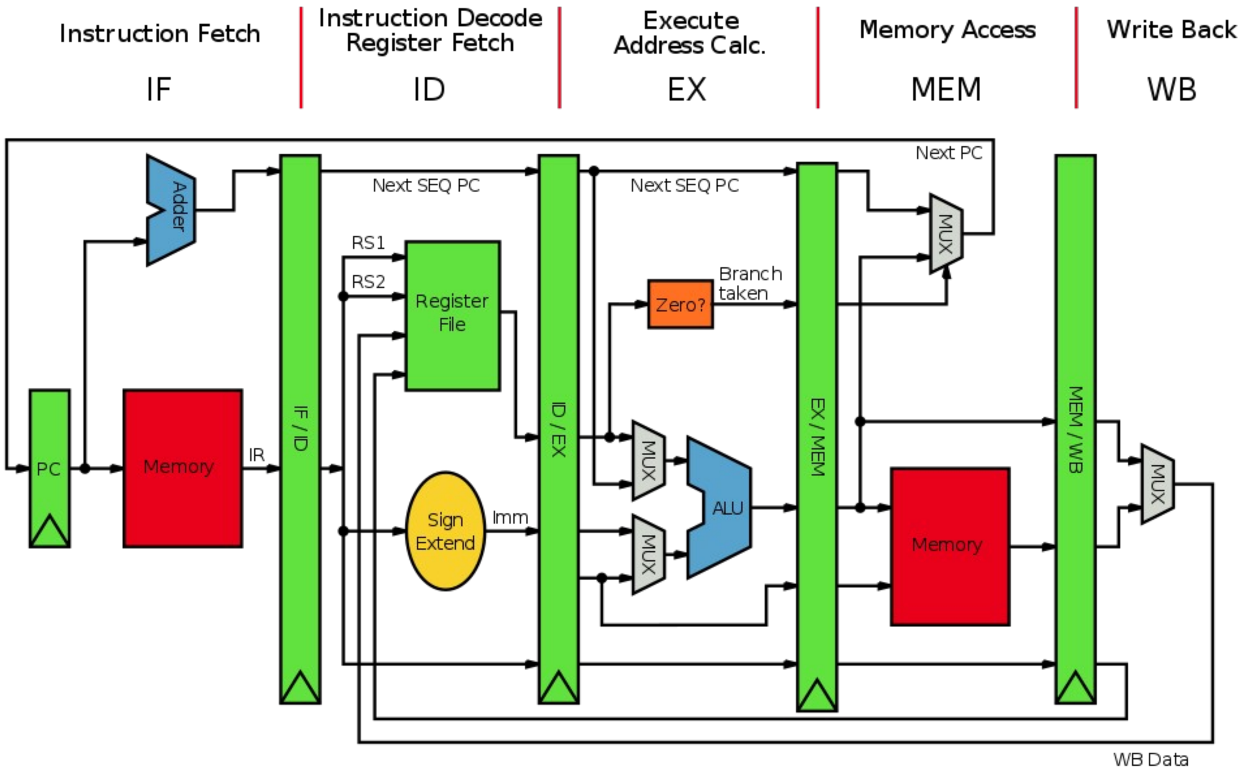


Illustration of Instruction Pipeline

Clock Cycle →	1	2	3	4	5	6	7	8	9	10
Instruction # 1	IF	ID	EX	MEM	WB ✓					
Instruction # 2		IF	ID	EX	MEM	WB ✓				
Instruction # 3			IF	ID	EX	MEM	WB ✓			
Instruction # 4				IF	ID	EX	MEM	WB ✓		
Instruction # 5					IF	ID	EX	MEM	WB ✓	
Instruction # 6						IF	ID	EX	MEM	WB ✓

Time taken to execute 6 instructions by a non-pipelined 5 multi-clock cycle processor **= 6 x 5 = 30 clock cycles**

For `k` stages pipeline: `n`th instruction will complete in `k+(n-1)` clock cycles

For 5 stage pipeline: 1000 instructions will complete in `5+(1000-1) = 1004 cc`

Speed up $= \dfrac{\textit{cc taken on non pipelined processor}}{\textit{cc taken on pipelined processor}} = \dfrac{30}{10}$ **= 3 times faster**

Cycle Time of a Pipelined Processor

Cycle time of a pipelined processor is dependent on four factors:

- Cycle time of non-pipelined version of the processor
- Number of pipeline stages
- Latch latency
- How evenly the data path logic is divided among stages

$$\textbf{Cycle Time of Even Pipelined Processor} = \frac{\text{Cycle Time of non pipelined processor}}{\text{Number of Stages}} + \text{Latch Latency}$$

$$\textbf{Cycle Time of Un-Even Pipelined Processor} = \text{Cycle Time of longest pipeline stage} + \text{Latch Latency}$$

Pipelined Hazards

- In a pipelined processor, ideally we expect a CPI value of 1 and a speedup equal to the number of stages in the pipeline. But, there are a number of factors that limit this. The problems that occur in the pipeline are called hazards
- Pipeline hazards are situations, that prevent the next instruction in the instruction stream from being executing during its designated clock cycle
- There are three classes or types of pipeline hazards
  - **Structural Hazards**
  - **Data Hazards**
  - **Control Hazards**

Structural Hazards

- Structural Hazards occur when multiple instructions are trying to access same resource (caches, memory, I/O devices, data bus, …) in the same clock cycle
- Structural hazards arise because there is not enough duplication of resources

Clock Cycle →	1	2	3	4	5	6	7	8	9	10
Instruction # 1	IF	ID	EX	MEM	WB					
Instruction # 2		IF	ID	EX	MEM	WB				
Instruction # 3			IF	ID	EX	MEM	WB			
Instruction # 4				IF	ID	EX	MEM	WB		
Instruction # 5					IF	ID	EX	MEM	WB	
Instruction # 6						IF	ID	EX	MEM	WB

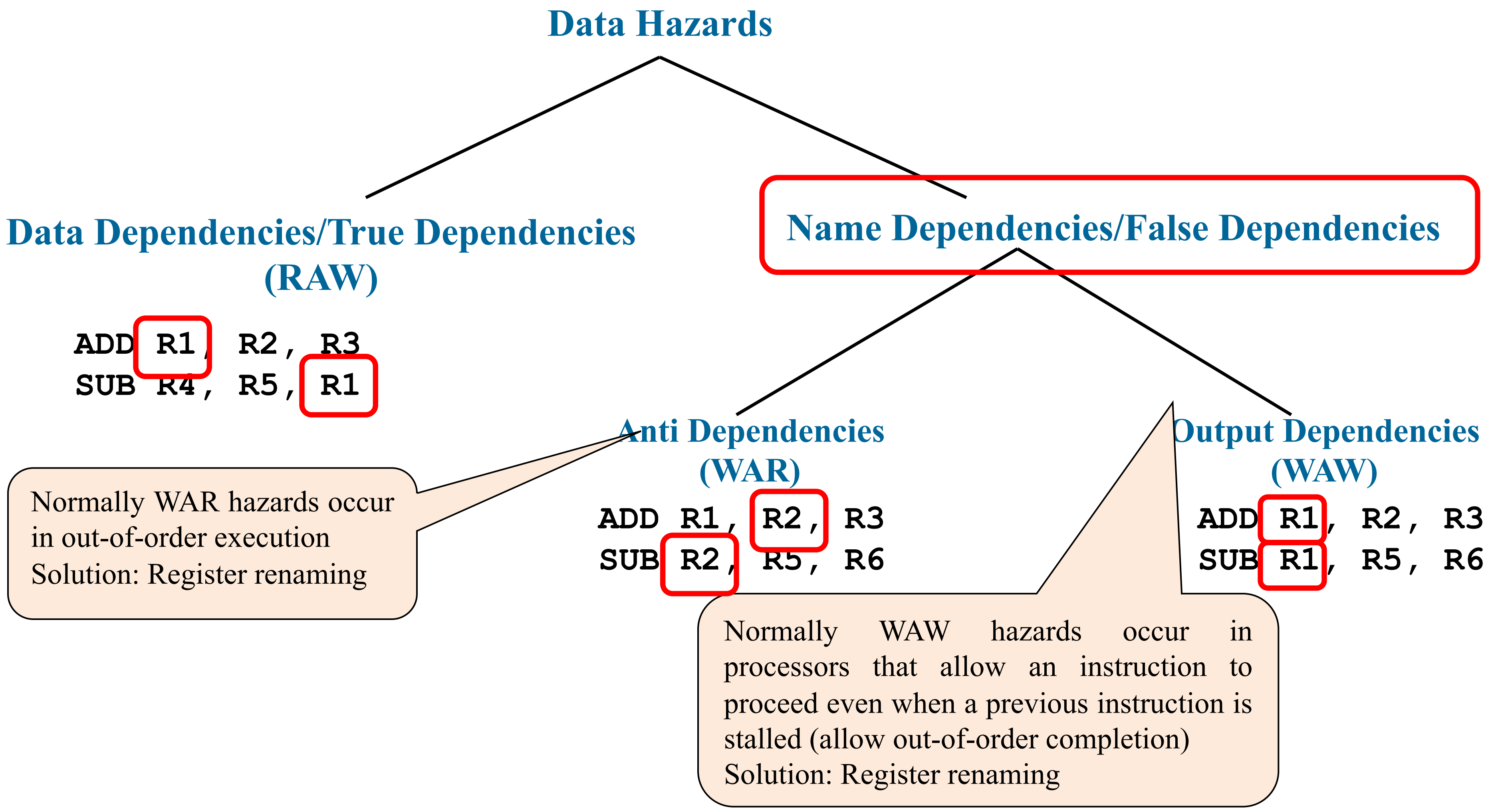
Structural Hazards: Solution

- One of the solution to structural hazards is to add more hardware which is of course expensive solution
- Another solution is wait or stall the pipeline. For this to work, there must be a mechanism to detect the hazard and then to stall. It is simple and less expensive solution, however, increases the overall CPI

Clock Cycle →	1	2	3	4	5	6	7	8	9	10	11
Instruction # 1	IF	ID	EX	MEM	WB						
Instruction # 2		IF	ID	EX	MEM	WB					
Instruction # 3			IF	ID	EX	MEM	WB				
Instruction # 4				**NOP**	IF	ID	EX	MEM	WB		
Instruction # 5						IF	ID	EX	MEM	WB	
Instruction # 6							IF	ID	EX	MEM	WB

Data Hazards

Data Hazards occur when an instruction depends on result of a prior instruction still in the pipeline


Data Hazards
Data Dependencies/True Dependencies (RAW)
Name Dependencies/False Dependencies
Anti Dependencies (WAR)
Output Dependencies (WAW)

Data Dependencies/True Dependencies (RAW)

```
ADD R1, R2, R3
SUB R4, R5, R1
```

Anti Dependencies (WAR)

```
ADD R1, R2, R3
SUB R2, R5, R6
```

Normally WAR hazards occur in out-of-order execution
Solution: Register renaming

Output Dependencies (WAW)

```
ADD R1, R2, R3
SUB R1, R5, R6
```

Normally WAW hazards occur in processors that allow an instruction to proceed even when a previous instruction is stalled (allow out-of-order completion)
Solution: Register renaming

RAW Hazard

Instruction#1: `ADD R1, R2, R3`
Instruction#2: `SUB R4, R5, R1`

Clock Cycle →	1	2	3	4	5	6				
Instruction # 1	IF	ID	EX	MEM	WB					
Instruction # 2		IF	ID	EX	MEM	WB				

RAW Hazard: Solution

```
Instruction#1:    ADD R1, R2, R3
Instruction#2:    SUB R4, R5, R1
```

Clock Cycle →	1	2	3	4	5	6				
Instruction # 1	IF	ID	EX	MEM	WB					
Instruction # 2		IF	ID	EX	MEM	WB				

Clock Cycle →	1	2	3	4	5	6	7	8	9	
Instruction # 1	IF	ID	EX	MEM	WB					
Instruction # 2		IF	NOP	NOP	NOP	ID	EX	MEM	WB	

Solutions to Data Hazards:

- Stall the pipeline
- Result/operand Forwarding
- Out of order or Dynamic Execution
  - Speculative execution
  - Branch predictions
  - Data flow analysis

Control Hazards

- Instructions that change the program counter leads to control hazards. For example, all branch instructions change the PC register from executing the next instruction in sequence to some other location

Instruction#1: `CMP R1, R2`
Instruction#2: `JEQ loop/1048`

Address	Clock Cycle →	1	2	3	4	5	6				
104	CMP R1, R2	IF	ID	EX	MEM	WB					
108	JEQ 1048		IF	ID	EX	MEM	WB				
112	Instruction # 3			IF	ID	EX	MEM				
116	Instruction # 4				IF	ID	EX				
120	Instruction # 5					IF	ID				

Control Hazards: Solutions

- Flush the pipeline
- Delayed Branch (from before, from target, from follow-up)
- Dynamic branch prediction
  - 1 or 2-bit predictor
  - Correlating predictor
  - Tournament predictor

Address	Clock Cycle →	1	2	3	4	5	6				
104	`CMP R1, R2`	IF	ID	EX	MEM	WB					
108	`JEQ 1048`		IF	ID	EX	MEM	$\boxed{\text{WB}}$				
112	Instruction # 3			IF	ID	EX	MEM				
116	Instruction # 4				IF	ID	EX				
120	Instruction # 5					IF	ID				

Control Hazards: Flushing

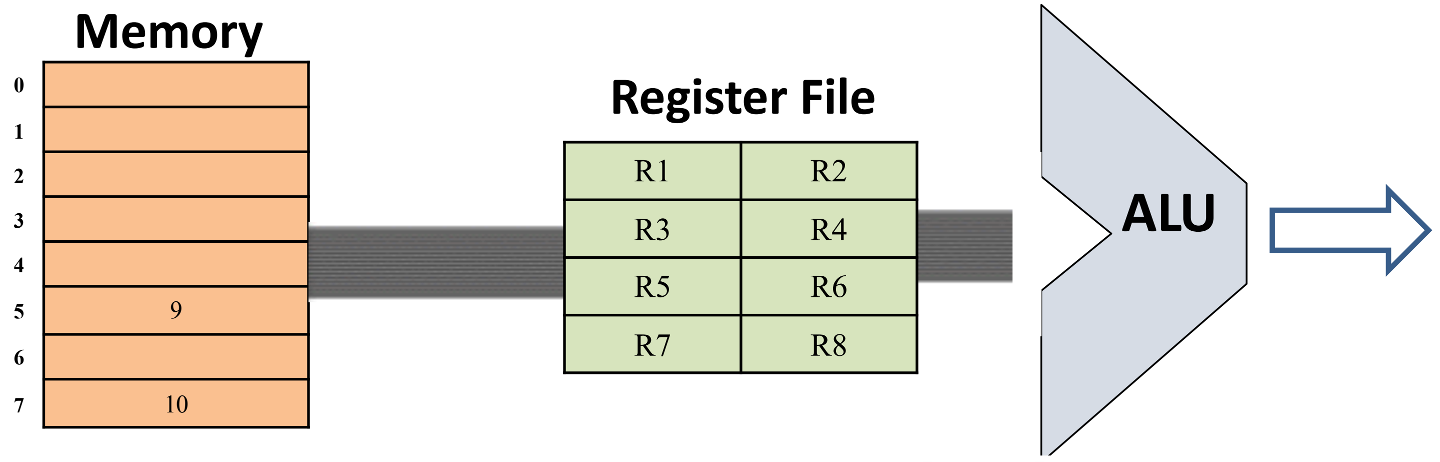
Address	Clock Cycle	1	2	3	4	5	6				
104	CMP R1, R2	IF	ID	EX	MEM	WB					
108	JEQ 1048		IF	ID	EX	MEM	$\boxed{\text{WB}}$				
112	Instruction # 3			IF	ID	EX	MEM				
116	Instruction # 4				IF	ID	EX				
120	Instruction # 5					IF	ID				

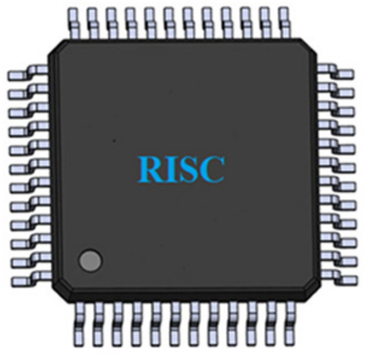
Control Hazards: Flushing

Address	Clock Cycle	1	2	3	4	5	6	7	8	9	
104	`CMP R1, R2`	IF	ID	EX	MEM	WB					
108	`JEQ 1048`		IF	ID	EX	MEM	WB				
1048	Instruction # n							IF	ID	EX	
1052	Instruction # n+1								IF	ID	
1056	Instruction # n+2									IF	

RISC vs CISC Architecture

RISC vs CISC





```
LOAD R1, Mem[5]
LOAD R2, Mem[7]
PROD R1, R2
STORE Mem[5], R1
```

```
MUL Mem[5], Mem[7]
```

RISC vs CISC

Reduced Instruction Computing

1. Simple and less number (around 100) of instructions which are easy to decode and implement
2. Instruction size is fixed and small usually size of a single word (32 bits)
3. Instructions take only one clock cycle to complete their execution (CPI<1.5)
4. CPU control is hard wired without control memory
5. Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions
6. More general purpose registers (32-192)
7. Fewer and simple addressing modes (3-5)
8. Pipelining is easy to implement
9. Examples: RISC-V, MIPS, ARM, Power PC, SPARC, Alpha, Blackfin, Atmel's AVR, Motorola 88000, Intel i860, Intel i960

Complex Instruction Computing

1. Complex and more number (120 – 350) of instructions which are difficult to decode and implement
2. Instructions are of variable sizes depending on their complexity (1-15 bytes)
3. Instructions take varying amount of clock cycles to complete their execution (2>CPI<15)
4. CPU control is micro-coded using control memory (ROM)
5. Arithmetic and logical operations can be applied to both memory and register operands
6. Less general purpose registers (8-24)
7. More and complex addressing modes (12-24)
8. Pipelining is difficult to implement
9. Examples: PDP-11, IBM 370/168, VAX 11/780, Motorola 68000 and Intel x86

Things To Do

Coming to office hours does NOT mean you are academically week!