

EEC 485 High Performance Arch. (Fall 2006)

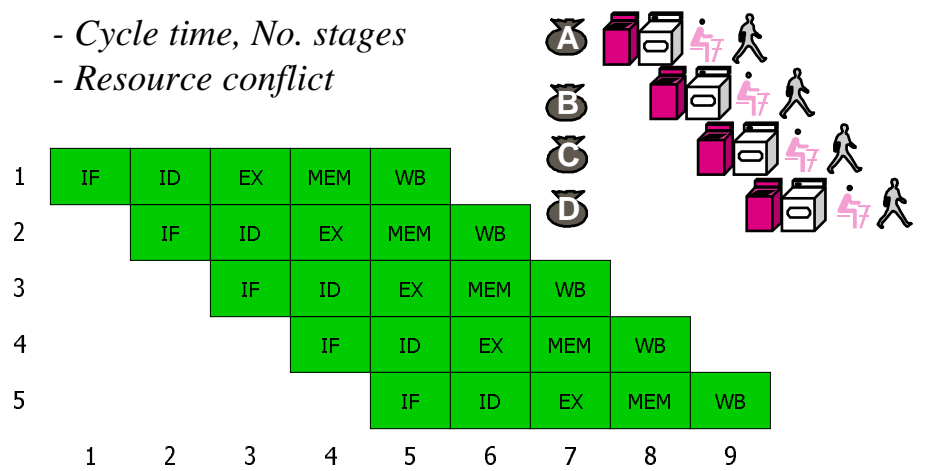
Chapter 6.2 A Pipelined Datapath

Chansu Yu

Cleveland State University

Pipelined Approach

- Cycle time, No. stages
- Resource conflict



Cleveland State
University

2

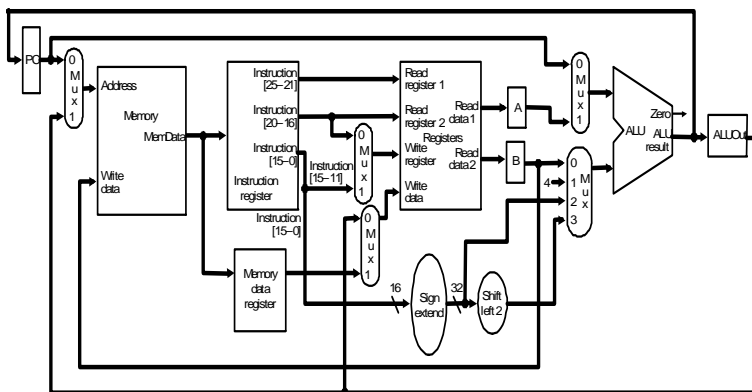
c.yu91@csuohio.edu

Structural Hazards

- ❑ Multicycle implementation has a number of structural hazards.

Multicycle MIPS => Pipelined MIPS

5 stages (IF, ID, EX, MEM, WB) are conflict-free with each other (no structural hazards). Is it true?



Multicycle MIPS => Pipelined MIPS

5 stages (IF, ID, EX, MEM, WB) are conflict-free with each other (no structural hazards). Is it true?

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A == B) then PC = ALUOut	PC = PC [31-28] (IR[25-0] << 2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Resources used in 5 Stages (revisit)

Stages	Register File	ALU	Memory
➤ Instruction Fetch (IF)			
➤ Instruction Decode/Register Fetch (ID)			
➤ Execute, Address computation, Branch/Jump completion (EX)			
➤ Memory access or R-type completion (MEM)			
➤ Memory read completion (WB)			

Now, what are the problems ?
And what are the solutions ?

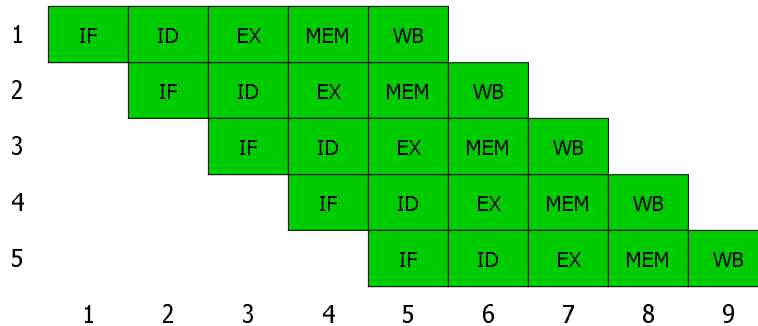
Resource Conflicts (revisit)

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch		IR = Memory[PC]		
Instruction decode/register fetch		PC = PC + 4 A = Reg[IR[25:21]] B = Reg[IR[20:16]]		
Execution, address computation, branch/jump completion	ALUOut = A op B	ALUOut = A + sign-extend(IR[15:0])	if (A == B) then PC = ALUOut	PC = PC[31:29] (IR[25:0] << 2)
Memory access or R-type completion	Reg[IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory[ALUOut] = B		Memory conflict
Memory read completion		Load: Reg[IR[20:16]] = MDR		

Register file conflict (read or write)

Resource Conflicts

- ALU : used in IF, ID, and EX stages (at time 5, instructions ??? collide)
- Memory : used in IF and MEM stages (at time 5, instructions ??? collide)
- Register : used in ID, MEM and WB stages (at time 5, instructions ??? collide)

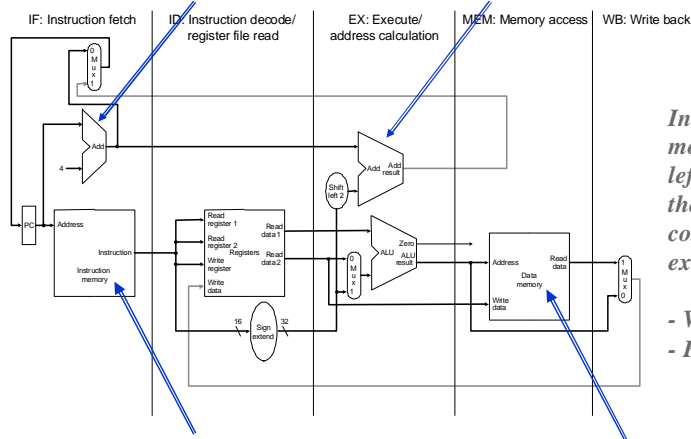


Solutions to Resource Conflicts

- ❑ ALU : used in IF, ID, and EX stages
 - 2 additional adders as in single cycle implementation
- ❑ Memory : used in IF and MEM stages
 - Two independent memories : instruction memory and data memory as in single cycle implementation
- ❑ Register : used in ID, MEM and WB stages
 - Reading register(s) in ID stage
 - Writing into a register in MEM stage
 - Writing into a register in WB stage

conflict ???
conflict ???

Basic Pipeline



Instructions and data move generally from left to right through the five stages as they complete execution except two cases.

- WB stage
- PC selection

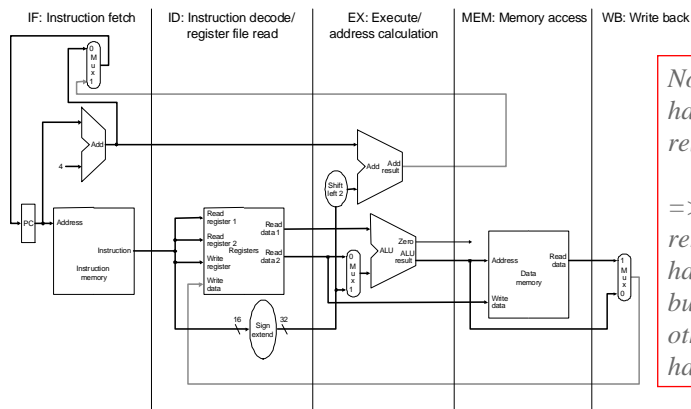
Basic Pipeline

Why move ??
ZF is available during EX stage, anyway.

Why do we still need 2 ALUs at EX stage?
(one for A-B and the other for PC+IR)

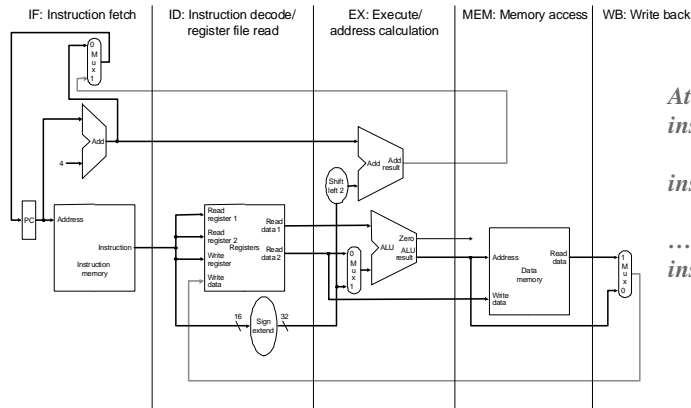
Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch		IR = Memory[PC] PC = PC + 4		
Instruction decode/register fetch		A = Reg[IR[25-21]] B = Reg[IR[20-16]] $ALUOut = PC + (\text{sign-extend}(\text{IR}[15-0]) \ll 2)$		
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A==B) then PC = ALUOut	PC = PC[31-28] (IR[25-0] << 2)
Memory access or R-type completion	Reg[IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory[ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Basic Pipeline



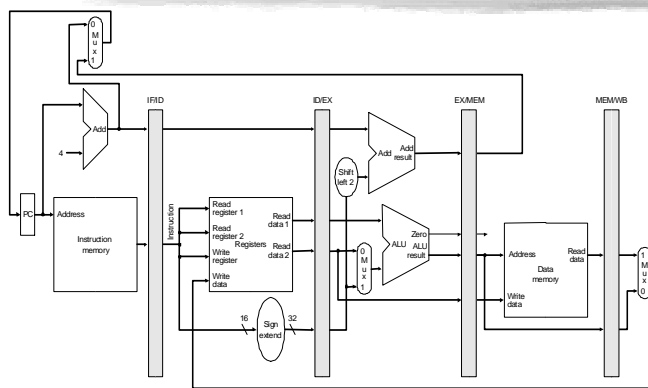
Now, don't we have any other resource conflicts ?
=> We actually remove structural hazards only, but there still are other types of hazards.

Basic Pipeline



At time 5,
inst. 1 uses resources
in WB stage,
inst. 2 uses resources
in MEM stage,
.....
inst. 5 uses resource
in IF stage.

Pipelined Datapath



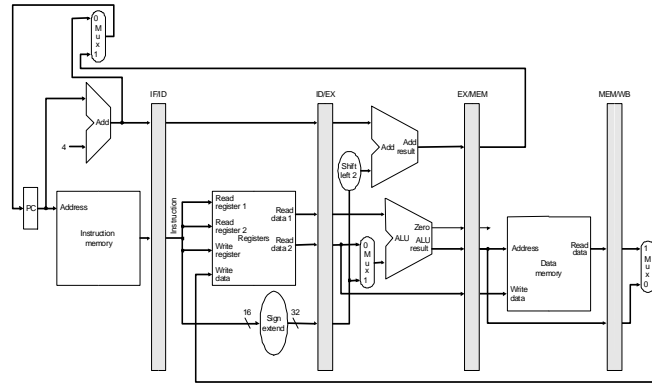
Add to the basic pipeline
in order to actually split
the datapath into stages.

The info. must be placed
in a pipeline register;
otherwise, it is lost when
the next instruction
enters that pipeline
stage.

For store instruction,

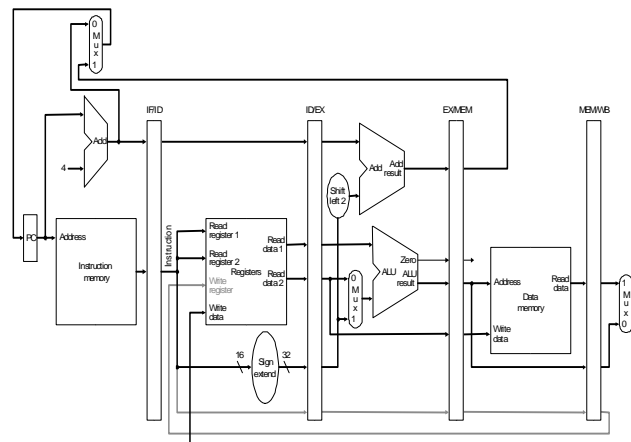
(?) => ID/EX pipeline register => EX/MEM pipeline register => (?)

Pipelined Datapath



*Can you find a problem even if there are no dependencies?
 What instructions can we execute to manifest the problem?*

Corrected Datapath

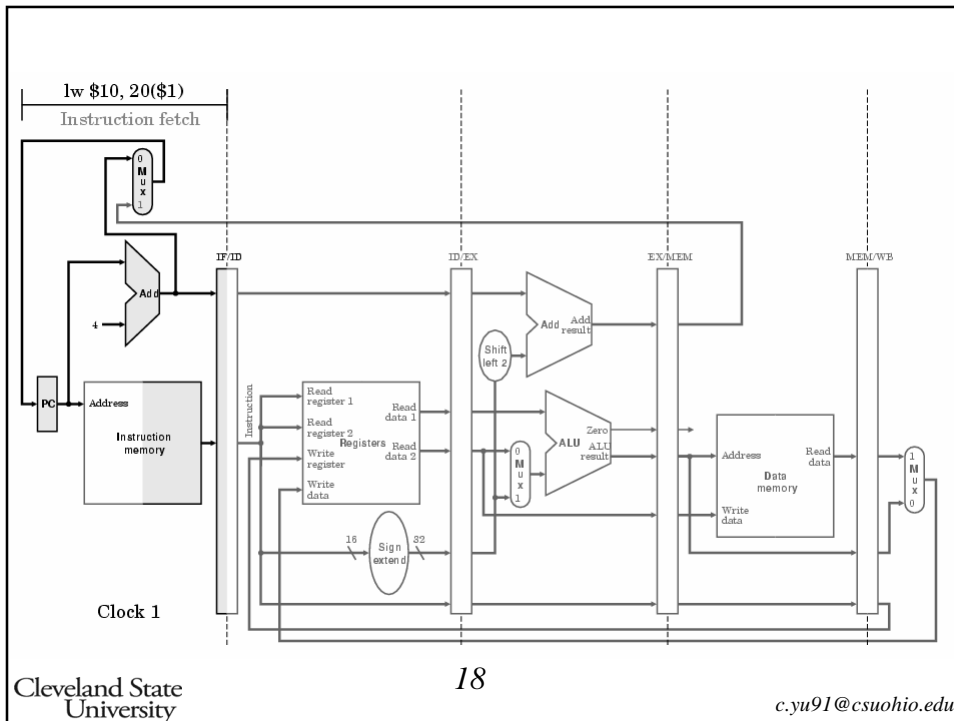


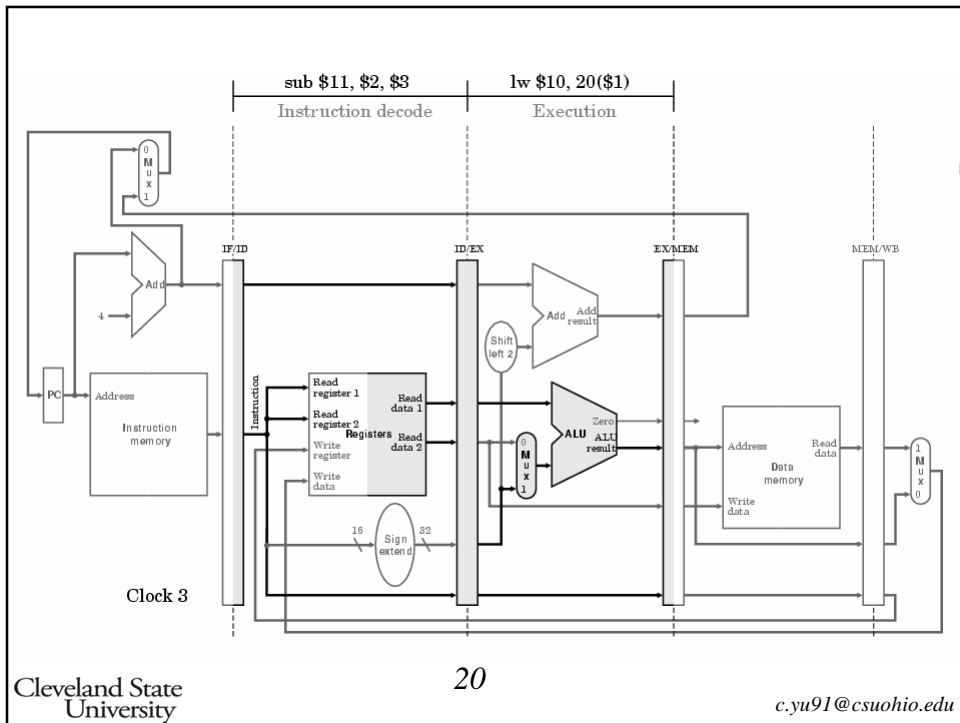
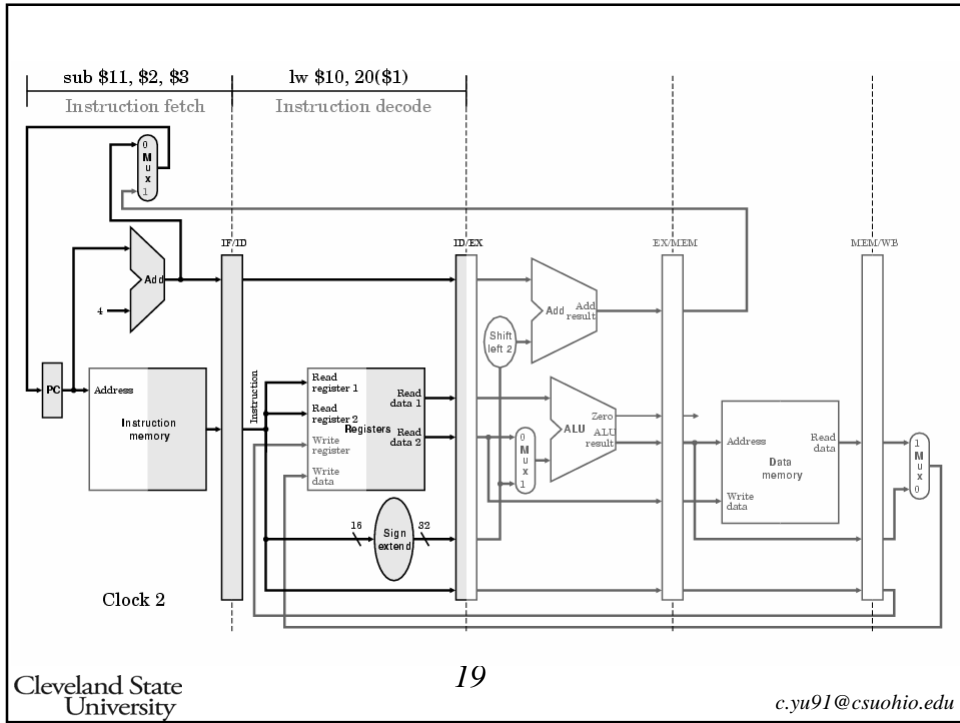
Example

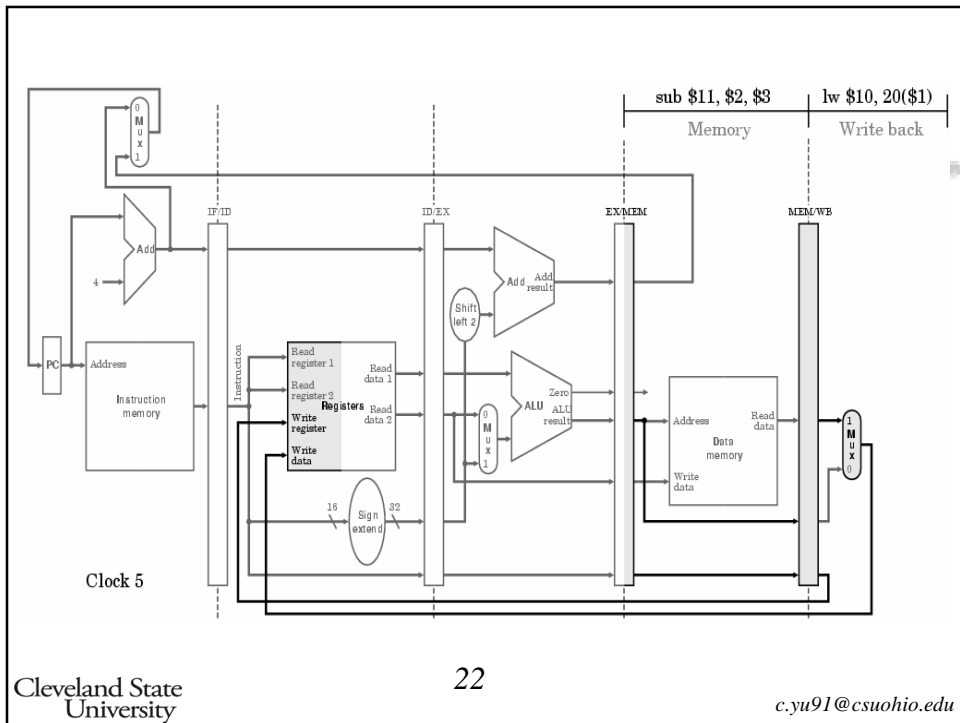
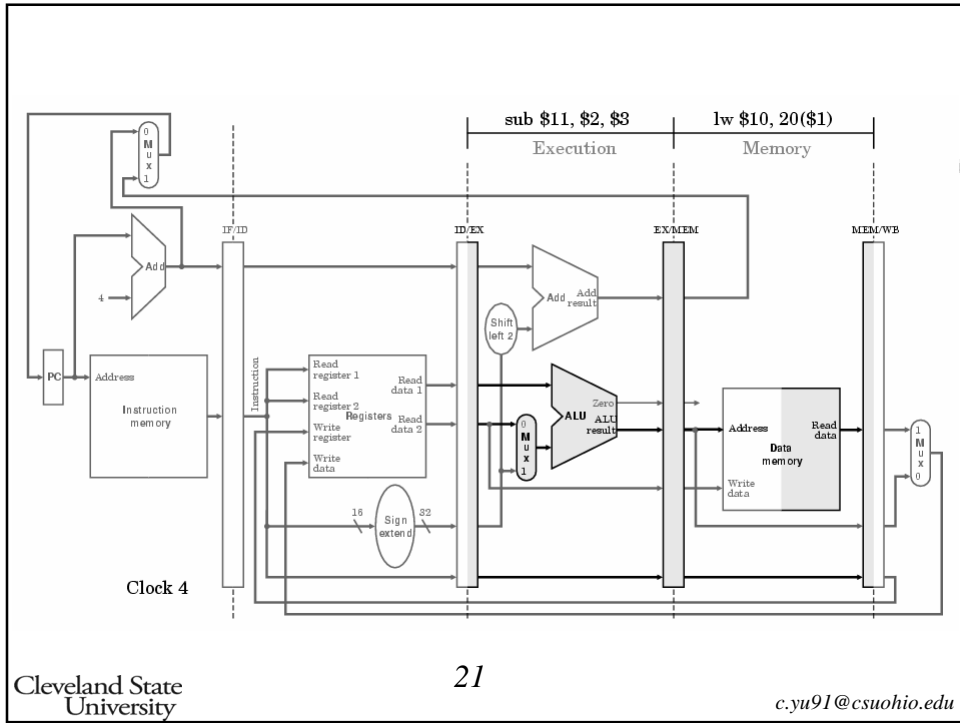
□ Five instructions go through the MIPS pipeline:

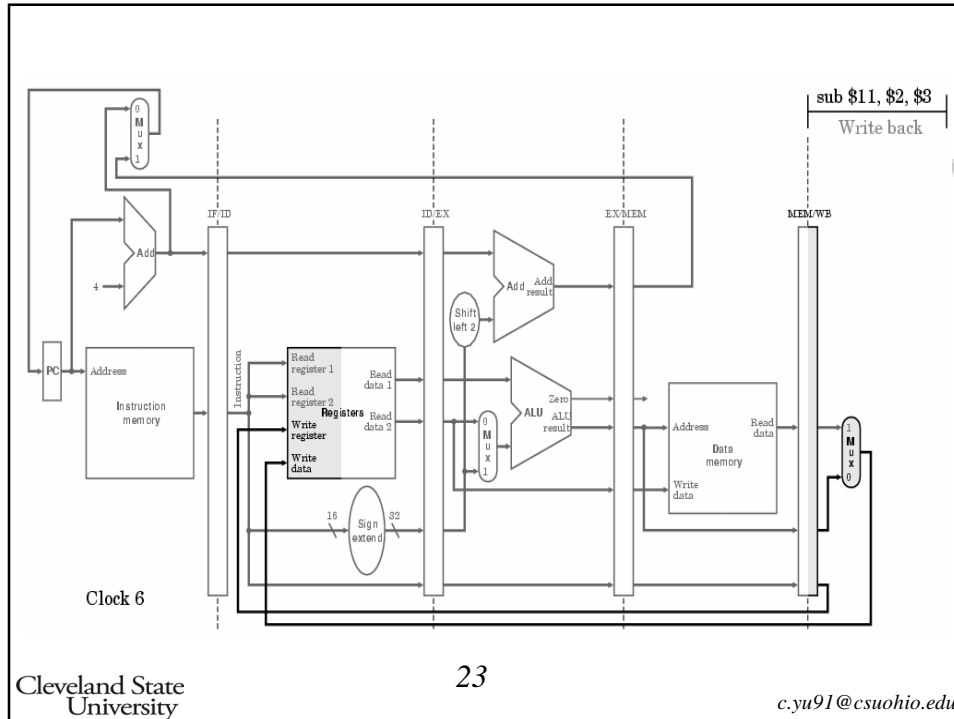
```
lw $10, 20($1)    10011 0001 0101 0000 0000 0001 0100 (8c2a 0014)
sub$11, $2, $3    01000 0010 0011 0101 0000 100100 (4043 5824)
and$12, $4, $5    01000 00100 00101 01100 00000 100110 (4085 6026)
or $13, $6, $7    01000 00110 00111 01101 00000 100111 (40c7 6827)
add$14, $8, $9    01000 01000 01001 01110 00000 100000 (4109 7020)
```

```
$pc = 0000 0000 5000 0000    [0000 0000 0000 1000] = 0000 1000 0000 0000
$1 = 0000 0000 0000 1000    [0000 0000 0000 1004] = 0000 1004 0000 0000
...
$9 = 0000 0000 0000 9000    .....
```





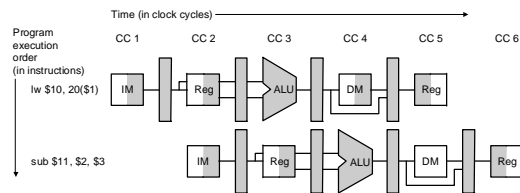




Content of Pipeline Registers

- ❑ Which data should be passed through stages? I.e., what are the contents of pipeline registers?
 - In IF/ID pipeline register
 - PC (32), Inst. (32)
 - In ID/EX pipeline register
 - PC (32), Reg. data 1 (32), Reg. data 2 (32), Offset (32), Reg. no. 2 and 3 (10)
 - In EX/MEM pipeline register
 - PC (32), ZF (1), ALUOut (32), Reg. data 2 (32), Reg. no. (5)
 - In MEM/WB pipeline register
 - Memory data (32), ALUOut (32), Reg. no. (5)

Graphically Representing Pipelines



- ❑ Can help with answering questions like:
 - how many cycles does it take to execute this code?
 - what is the ALU doing during cycle 4?
 - use this representation to help understand datapaths