

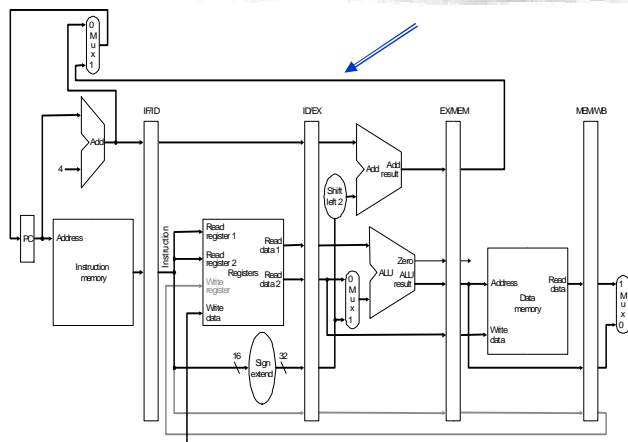
EEC 485 High Performance Arch. (Fall 2007)

Chapter 6.4 Data Hazards and Forwarding

Chansu Yu

Cleveland State University

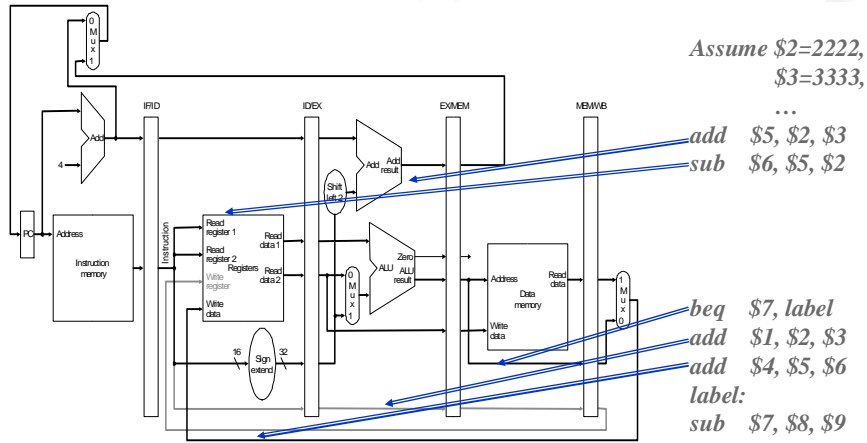
5-Stage Pipeline



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

- WB stage
- PC selection at MEM stage

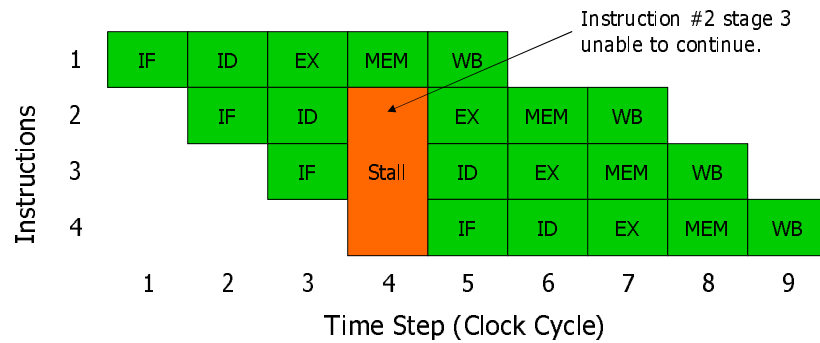
Pipeline Complexities



Hazards

Hazard = when an instruction's stage is unable to execute during the current cycle.

Forced to *stall* (delay) the pipeline:



Hazard Types

- ❑ Structural hazards : Necessary functional unit is busy
 - suppose we had only one memory
 - we already remove this type of hazards

- ❑ Control hazards : Next instruction address unknown
 - need to worry about branch instructions

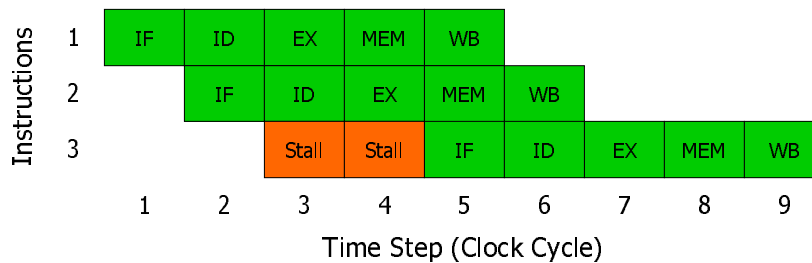
- ❑ Data hazards : Source data not yet available
 - an instruction depends on a previous instruction

Structural Hazards

A needed functional unit is busy executing a previous instruction.

Example:

- Our sample MIPS pipeline has none.
- What if PC+4 computation used main ALU instead of separate adder?

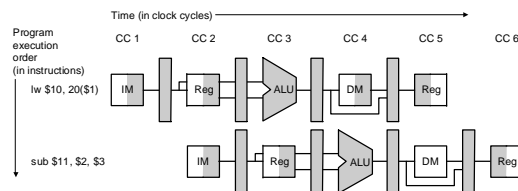


Structural Hazards: Avoiding

- ❑ Add or replicate functional units.
 - Usually easy in modern processors (lots of hardware real estate).
 - Our sample pipeline avoids all structural hazards.

- ❑ Load/store architecture
 - Avoids structural hazards by limiting memory access only to load and store instructions
 - “sub \$1, 200(\$10)”
 - IF – ID – EX (Addr. Comp.) – MEM – “EX (subtraction)” – WB
 - One more stage (EX) between MEM/WB with additional ALU is not a big problem
 - But all other instructions should be wasting a clock since they need to execute the same number of stages

*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

Data Hazards

- ❑ Needed data still being computed by previous instruction

```

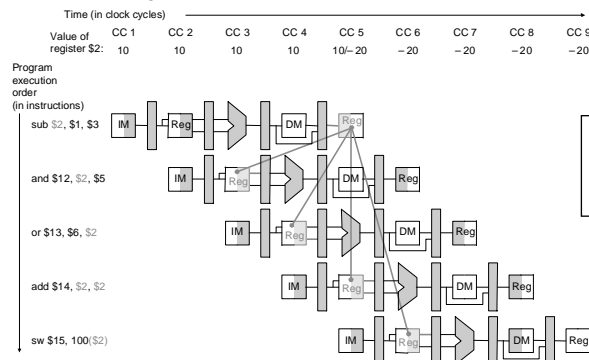
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
add  $14, $2, $2
sw   $15, 100($2)
    
```

Assume \$1=10,
\$2=10, \$3=30

9

Data Hazards: Dependencies

- ❑ Problem with starting next instruction before first is finished
 - dependencies that “go backward in time” are data hazards



“and” has a problem
“or” has a problem
“add” ???
“sw” is OK

10

Data Hazards: Software Solution

- ❑ Have compiler guarantee no hazards
- ❑ Where do we insert the “nops” ?

```

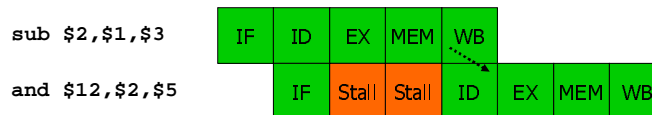
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
add  $14, $2, $2
sw   $15, 100($2)
    
```

*Is compiler dependent on machine (CPU)?
 ⇒ A LOT !!!
 (because it is supposed to generate machine code)
 ⇒ CPU designer & compiler designer should work together.*

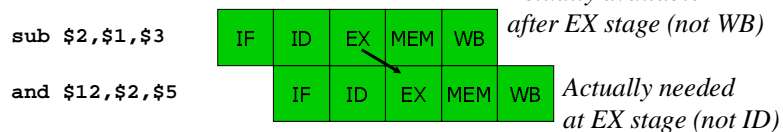
- ❑ Or, detect and “stall” the pipeline
- ❑ Problem: this really slows us down!

Data Hazards: Forwarding

While result not written back until WB:

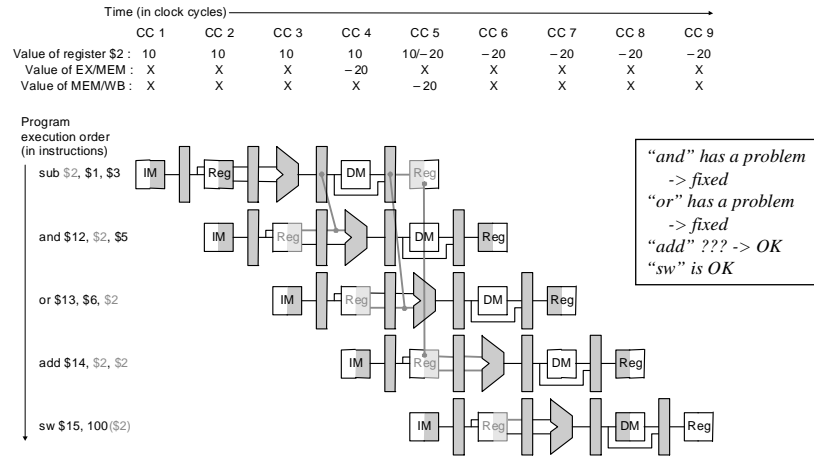


It is calculated earlier – in EX:

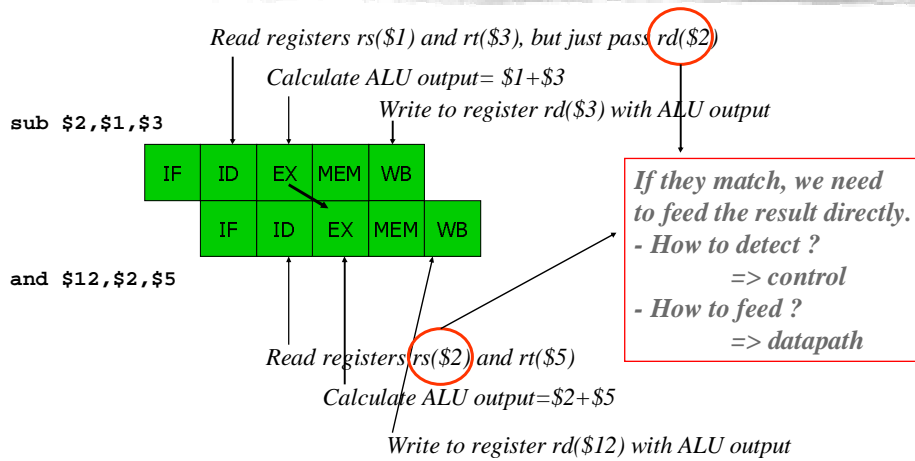


Add forwarding hardware to allow, e.g., EX’s output (located in EX/MEM pipeline register) to be EX’s input.

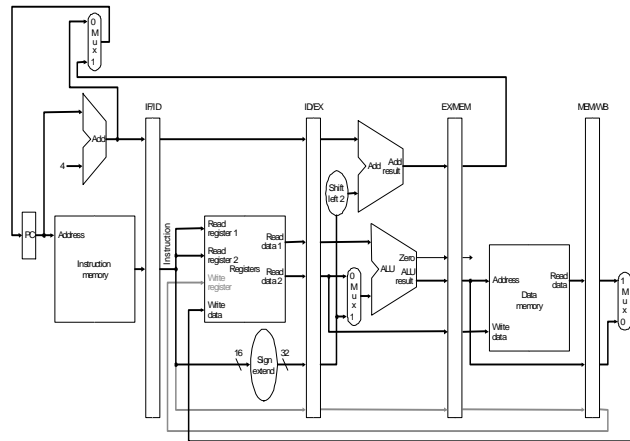
Forwarding : All 2 Cases



Forwarding : Implementation



Forwarding : Implementation



*Additional datapath
for forwarding ?*

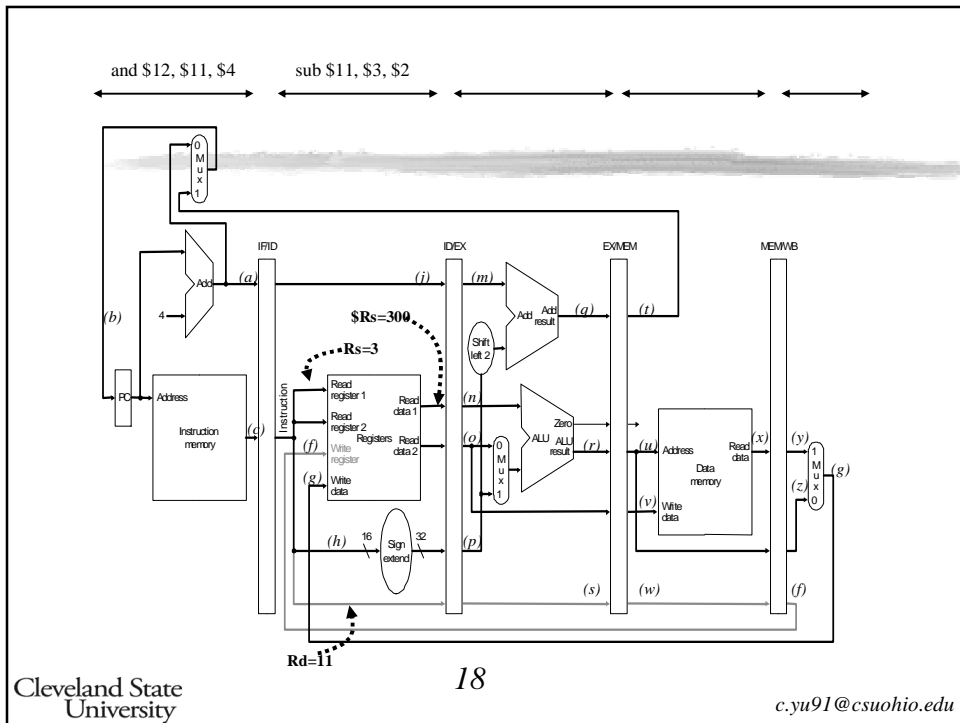
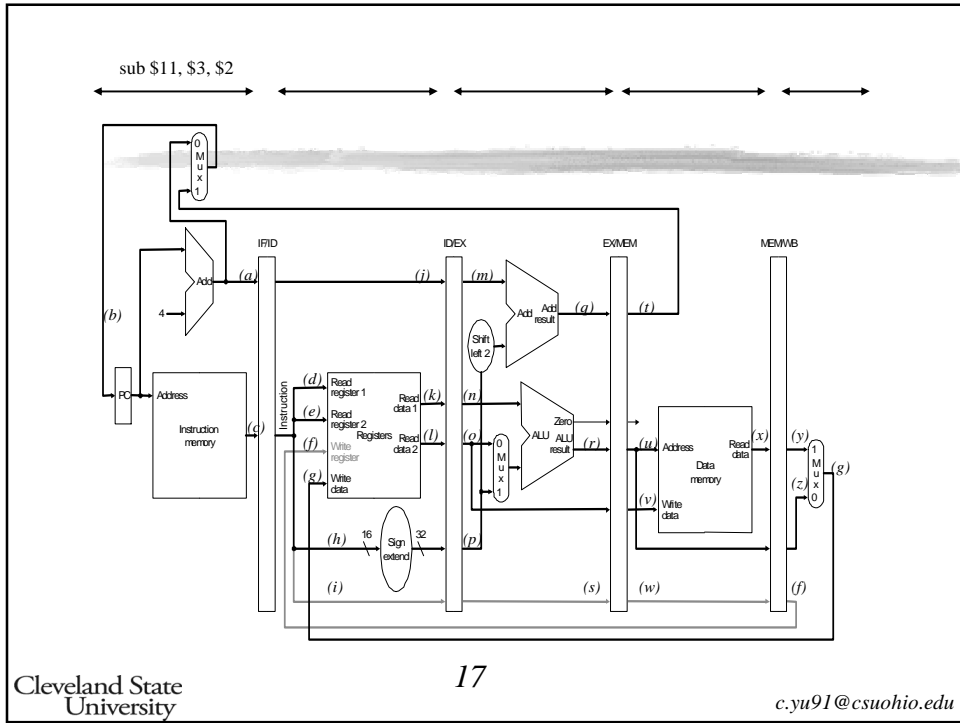
*How to control the
forwarding datapath ?*

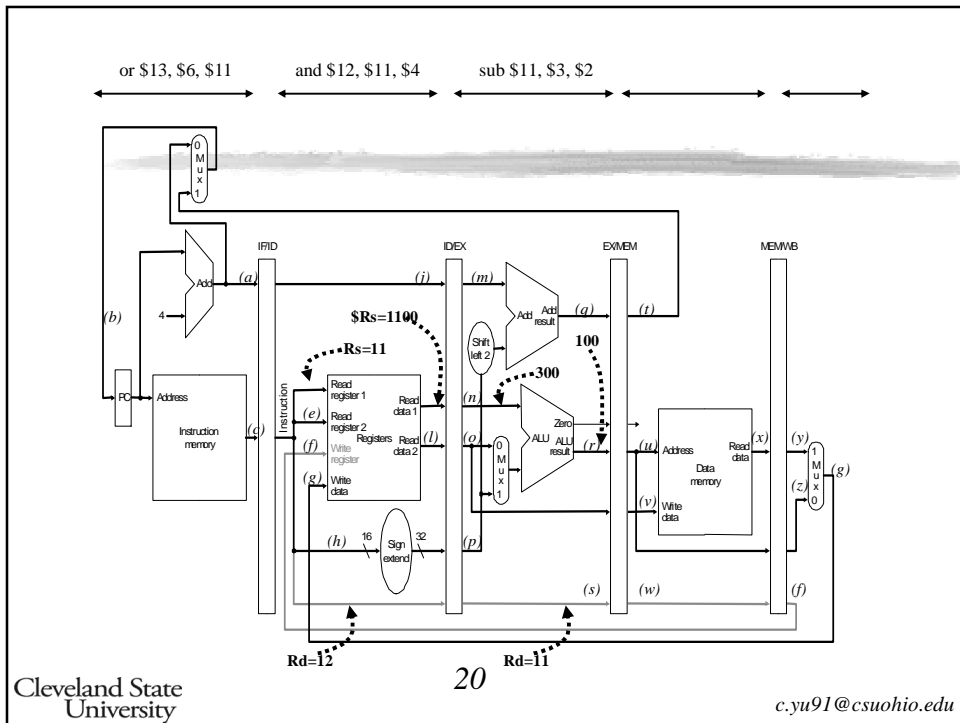
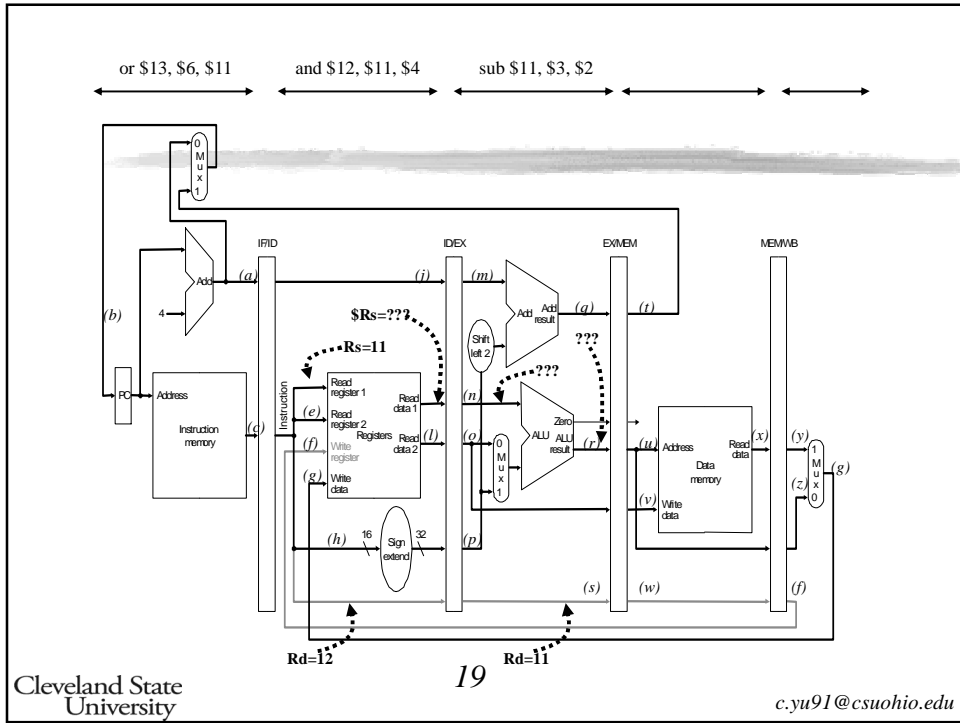
Data Hazards (again)

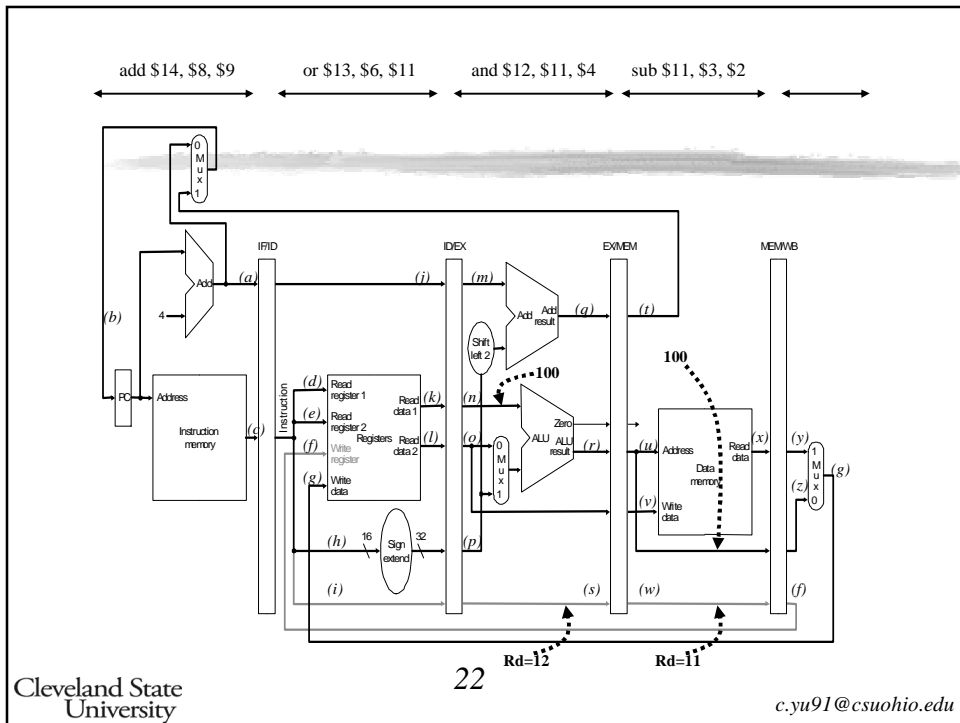
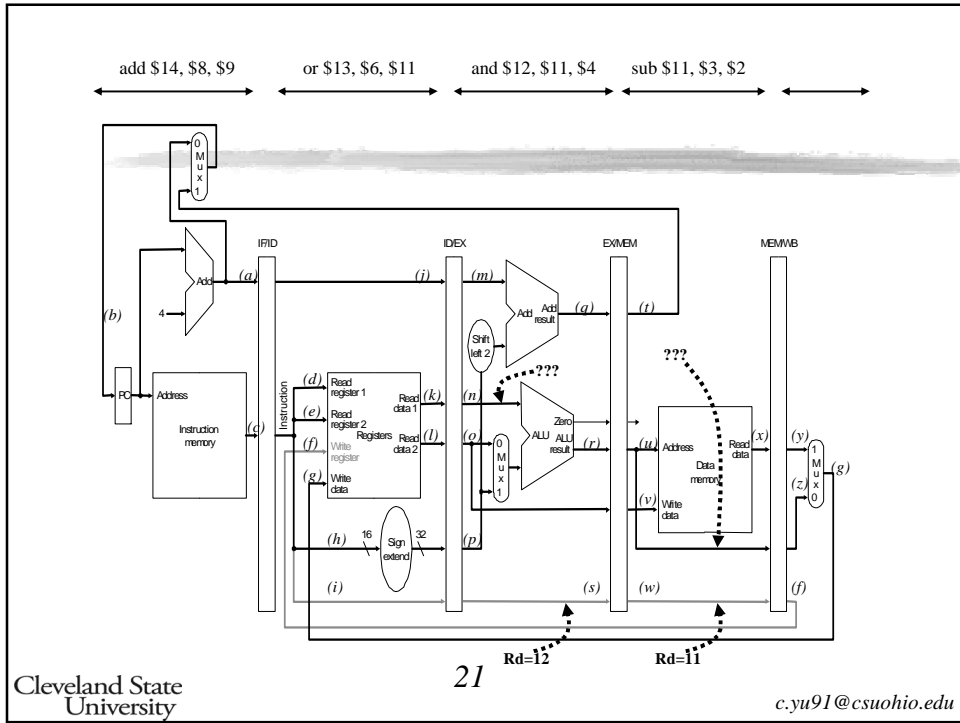
- ❑ Needed data still being computed by previous instruction

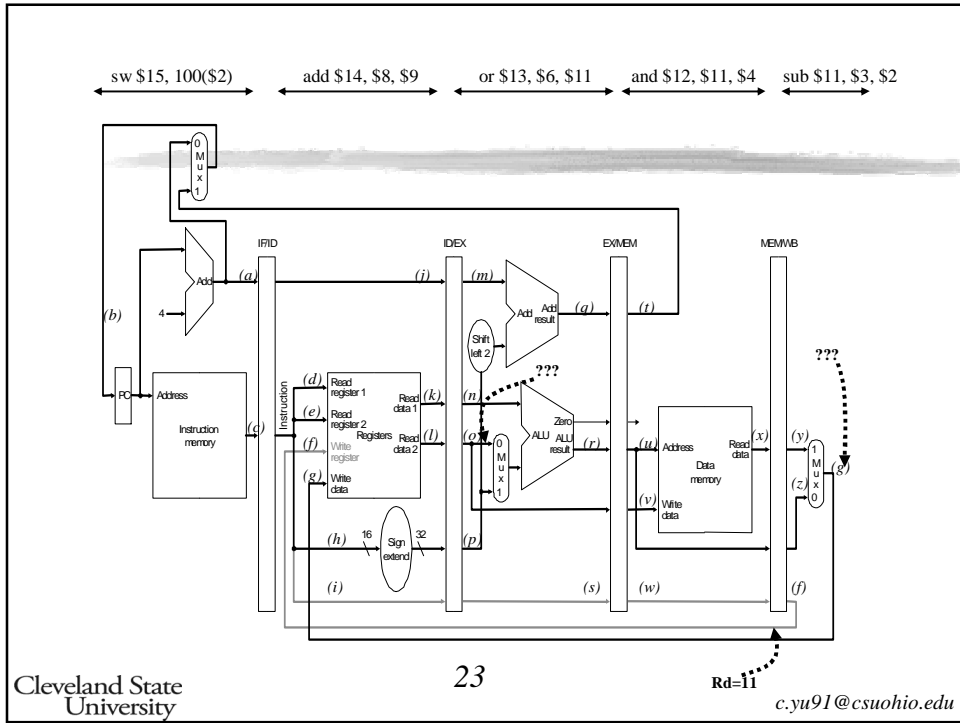
```

sub  $11, $3, $2
and  $12, $11, $4
or   $13, $6, $11
add  $14, $8, $9
sw   $15, 100($2)
    
```

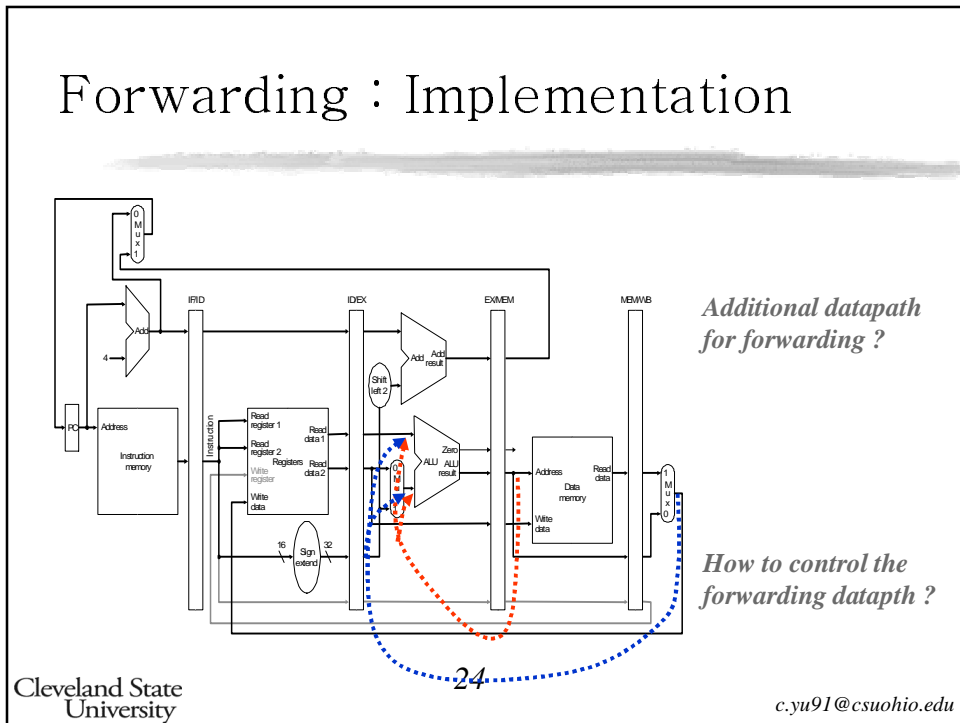




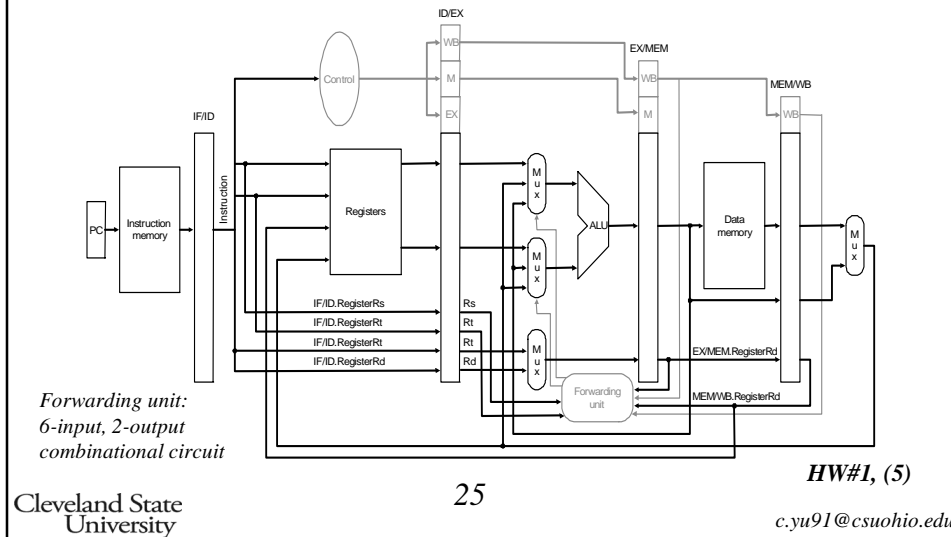




Forwarding : Implementation

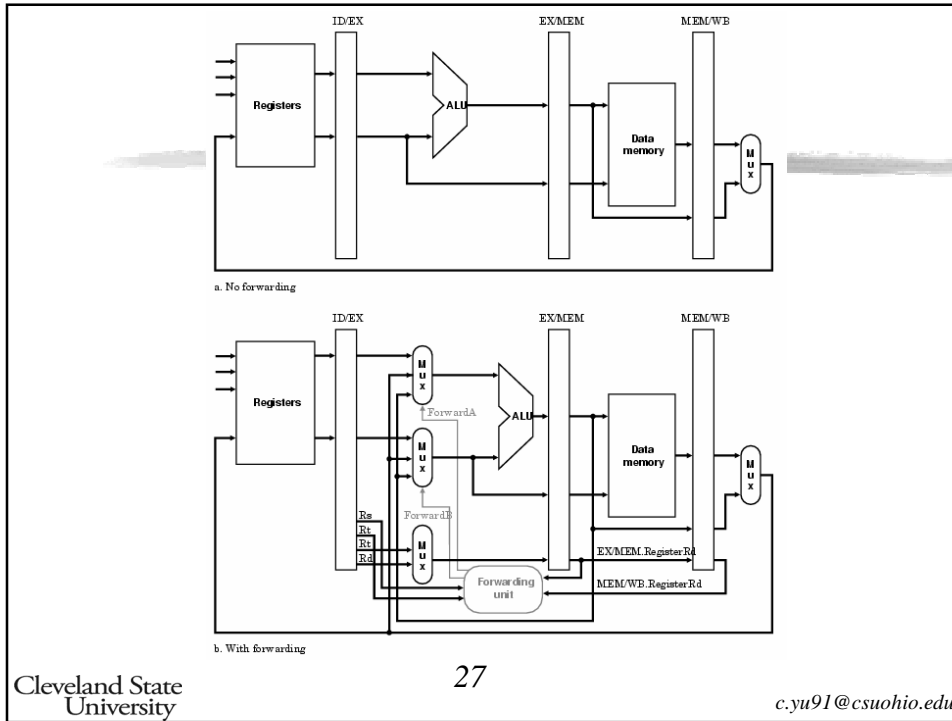


Forwarding : Forwarding Unit



Forwarding : Other Case

- ❑ Use temporary results, don't wait for them to be written
 - Register file forwarding to handle read/write to same register
 - ALU forwarding
- ❑ In the previous example,
 - EX/MEM.Rd = ID/EX.Rs
 - How about "EX/MEM.Rd = ID/EX.Rt" ?
 - They are the case: EX/MEM → EX: Forwarding to the next instruction
- ❑ Anything more?
 - In MIPS pipeline, one more forwarding case:
 - MEM/WB → EX: Forwarding to the instruction after the next.
 - sub \$2, \$1, \$3
 - and \$12, \$2, \$5
 - or \$13, \$6, \$2
 - MEM/WB.Rd = ID/EX.Rs and MEM/WB.Rd = ID/EX.Rt



Forwarding Control

Control logic

ForwardA =

- 10 if (EX/MEM.Rd = ID/EX.Rs) <- get operand from EX/MEM
- 01 if (MEM/WB.Rd = ID/EX.Rs) <- get operand from MEM/WB
- 00, otherwise <- get operand from ID/EX

ForwardB =

- 10 if (EX/MEM.Rd = ID/EX.Rt) <- get operand from EX/MEM
- 01 if (MEM/WB.Rd = ID/EX.Rt) <- get operand from MEM/WB
- 00, otherwise <- get operand from ID/EX

Forwarding Control

- ❑ RegWrite must be active & dest. reg. is not \$0
- ❑ Control logic
 - ForwardA =
 - 10 if ((EX/MEM.Rd = ID/EX.Rs) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0))
 - 00, otherwise
 - ForwardB =
 - 10 if ((EX/MEM.Rd = ID/EX.Rt) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rt) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0))
 - 00, otherwise

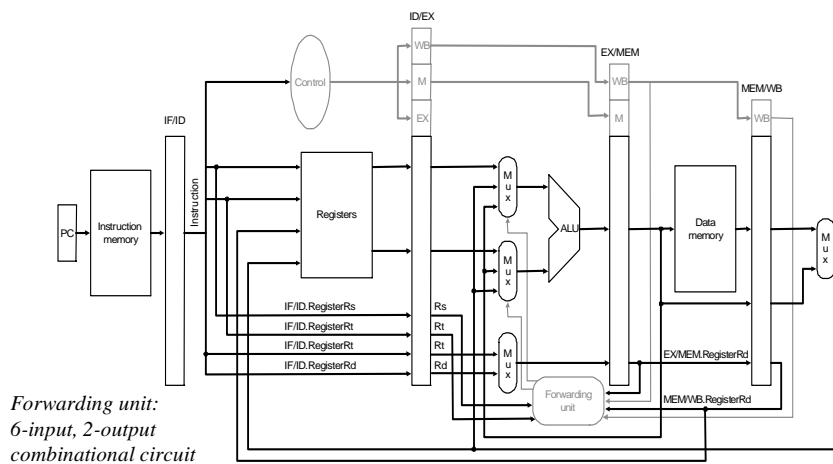
Forwarding Control

- ❑ Control logic
 - ForwardA =
 - 10 if ((EX/MEM.Rd = ID/EX.Rs) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0))
 - 00, otherwise
 - If a list of codes is
 - Add \$1, \$2, \$3 : at WB → MEM/WB.Rd=1
 - Add \$1, \$1, \$4 : at MEM → EX/MEM.Rd=1
 - Add \$1, \$1, \$5 : at EX → ID/EX.Rs=1
 - When the third inst. is at EX stage, the two conditions are all met. Then, ForwardA=10 or 01???
 - Should be forwarded from the most recent result, which is "10"
 - Therefore, ForwardA=01
 - if ((MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rs))

Forwarding Control

- ❑ RegWrite must be active & dest. reg. is not \$0
- ❑ Control logic
 - ForwardA =
 - 10 if ((EX/MEM.Rd = ID/EX.Rs) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rs))
 - 00, otherwise
 - ForwardB =
 - 10 if ((EX/MEM.Rd = ID/EX.Rt) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rt) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rt))
 - 00, otherwise

Forwarding : Forwarding Unit



Forwarding unit:
6-input, 2-output
combinational circuit

Where does ALUSrc go?

