Control Signal for Pipeline Processor

Outline

- A pipelined datapath
- Pipelined control
- Data hazards and forwarding
- Data hazards and stalls
- Branch hazards

Pipeline Control

- Three steps are needed
  - Step 1: Begin with the same control signal with single-cycle datapath control
  - Group control lines into five groups according to pipeline stage
    - Since control signals are associated with components active during a single pipeline stage
  - Set control signals during each pipeline stage

Pipelined Datapath with Control – Step 1

Start with Same control signals as the single-cycle datapath

PCSrc
0 M u x 1
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
Add
Add result
Branch
Shift left 2
RegWrite
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALUSrc
0 M u x 1
Zero
ALU
ALU result
MemWrite
Address
Data memory
Write data
Read data
MemRead
MemtoReg
1 M u x 0
Instruction [15– 0]
16
Sign extend
32
6
ALU control
ALUOp
Instruction [20– 16]
Instruction [15– 11]
0 M u x 1
RegDst

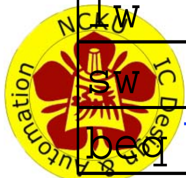
Pipeline Control Signals- Step 2

- Group control lines into five groups according to pipeline stage
  - instruction fetch / PC increment (IF)
  - instruction decode / register fetch (ID)
  - execution / address calculation (EX)
  - memory access (M)
  - write back (WB)

Nothing to control as instruction memory read and PC write are always enabled (IF, ID)

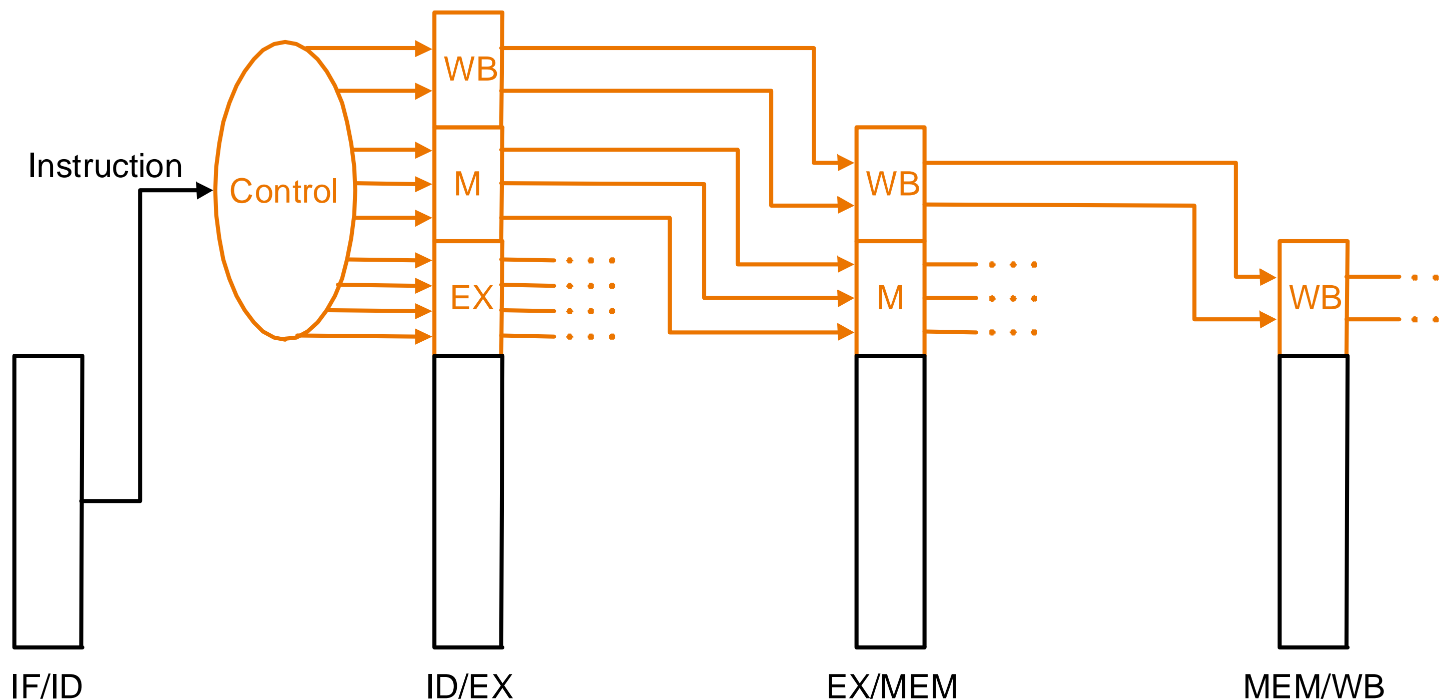
(the following table) (EX, M, WB)

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X



Pipeline Control Implementation

- *Pass control signals along just like the data* – extend each pipeline register to hold needed control bits for succeeding stages



- *Note*: The 6-bit *funct field* of the instruction required in the EX stage to generate ALU control can be retrieved as the 6 least significant bits of the immediate field which is sign-extended and passed from the IF/ID register to the ID/EX register

Pipelined Datapath with Control – Step 3

PCSrc
0 Mux 1
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
WB
M
WB
Add
4
Add
Add result
Shift left 2
Branch
RegWrite
MemWrite
MemtoReg
ALUSrc
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0 Mux 1
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
1 Mux 0
Instruction [15–0]
16
Sign extend
32
6
ALU control
MemRead
Instruction [20–16]
ALUOp
0 Mux 1
Instruction [15–11]
RegDst

Control signals come from the control and are portions of the pipeline registers

An Example

```
lw   $10, 20($1)
sub  $11, $2, $3
and  $12, $4, $5
or   $13, $6, $7
add  $14, $8, $9
```

Cycle 1

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: lw $10, 20($1) | ID: before<1> | EX: before<2> | MEM: before<3> | WB: before<4>

Clock 1

Cycle 2

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: sub $11, $2, $3 | ID: lw $10, 20($1) | EX: before<1> | MEM: before<2> | WB: before<3>

IF/ID ID/EX EX/MEM MEM/WB

Control: WB 11, M 010, EX 0001

Clock 2

Cycle 3

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: and $12, $4, $5 | ID: sub $11, $2, $3 | EX: lw $10, . . . | MEM: before<1> | WB: before<2>

Clock 3

Cycle 4

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: or $13, $6, $7	ID: and $12, $2, $3	EX: sub $11, . . .	MEM: lw $10, . . .	WB: before<1>

Clock 4

Cycle 5

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: add $14, $8, $9 | ID: or $13, $6, $7 | EX: and $12, . . . | MEM: sub $11, . . . | WB: lw $10, . . .

Clock 5

Cycle 6

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: after<1>

ID: add $14, $8, $9

EX: or $13, . . .

MEM: and $12, . . .

WB: sub $11, . . .

IF/ID
ID/EX
EX/MEM
MEM/WB

0 M u x 1
Add
4
PC
Address
Instruction memory
Instruction
add
Control
10
000
1100
WB
M
EX
10
000
1
10
0
WB
10
0
0
0
M
1
WB
0
RegWrite
8
9
11
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
$8
$9
$6
$7
Shift left 2
Add
Add result
ALUSrc
0 M u x 1
ALU
ALU result
Zero
Branch
MemWrite
Address
Data memory
Read data
Write data
MemRead
MemtoReg
1 M u x 0
X
Instruction [15– 0]
Sign extend
X
ALU control
ALUOp
X
Instruction [20– 16]
X
0 M u x 1
RegDst
14
Instruction [15– 11]
14
13
12
11

Clock 6

Cycle 7

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: after<2>

ID: after<1>

EX: add $14, . . .

MEM: or $13, . . .

WB: and $12, . . .

Clock 7

Cycle 8

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: after<3> | ID: after<2> | EX: after<1> | MEM: add $14, . . . | WB: or $13, . . .

Clock 8

Cycle 9

```
lw  $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or  $13, $6, $7
add $14, $8, $9
```

IF: after<4>

ID: after<3>

EX: after<2>

MEM: after<1>

WB: add $14, . . .

IF/ID
ID/EX
EX/MEM
MEM/WB

Control
WB
M
EX
00
000
0000
RegWrite
Branch
MemWrite
MemtoReg
MemRead
ALUSrc
ALUOp
RegDst

PC
Address
Instruction memory
Add
4
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
14
Instruction [15–0]
Sign extend
Instruction [20–16]
Instruction [15–11]
Shift left 2
Add result
ALU
Zero
ALU result
ALU control
Mux
Address
Data memory
Read data
Write data

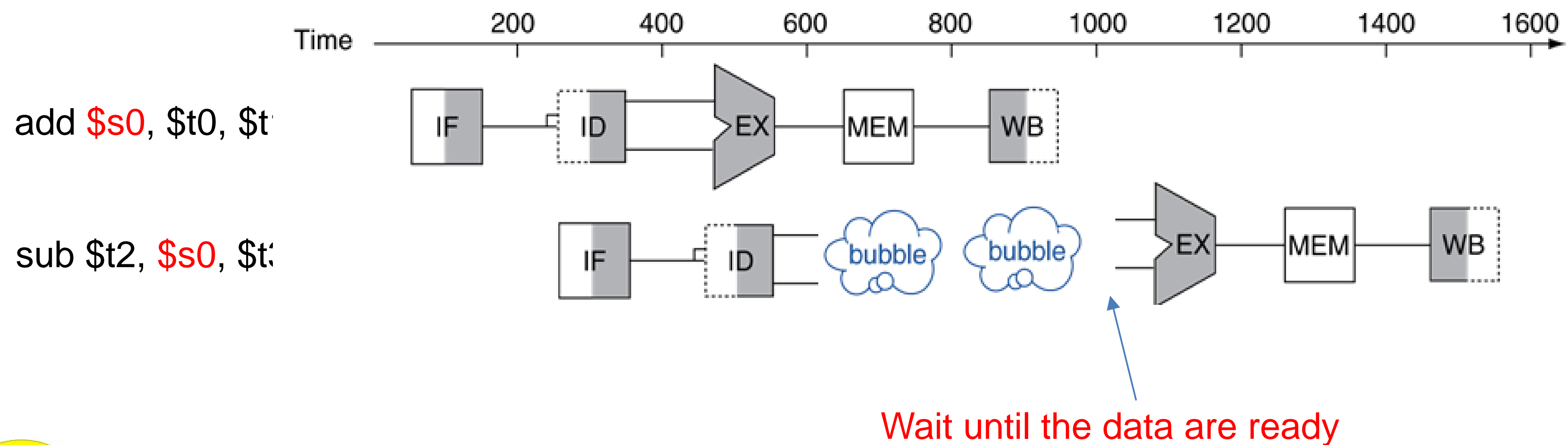
Clock 9

Outline

- A pipelined datapath
- Pipelined control
- Data hazards and forwarding
- Data hazards and stalls
- Branch hazards

Pipeline Hazards

- Pipeline Hazards:
  - Structural hazards, Data hazards, Control hazards
- Hazards can be resolved by waiting (add stalls/bubble)
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards



Data Hazards in ALU Instructions

- Consider this sequence:

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

- We can resolve hazards with forwarding
  - How do we detect when to forward?

Data Hazards

Time (in clock cycles)

Value of register $2:	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
	10	10	10	10	10/-20	-20	-20	-20	-20

Program execution order (in instructions)

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

$2 is written over here

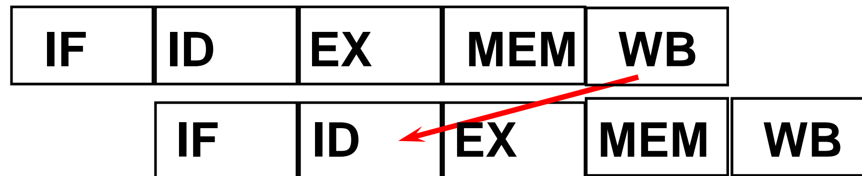
Data Hazards

No Hazards

Three Types of Data Dependency (RAW, WAR, WAW)

- RAW (read after write):
  i2 tries to read operand before i1 writes it

W R R
sub $2 $1 $3
add $4 $3 $2
W R R

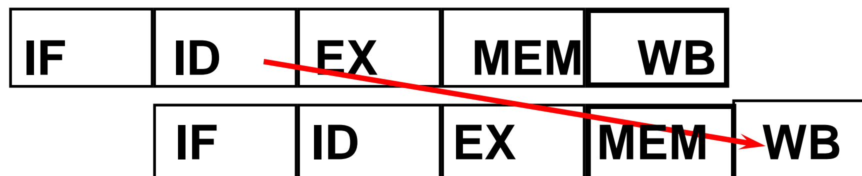


RAW is Data Dependence also cause data hazard

- WAR (write after read):
  i2 tries to write operand before i1 reads it

add $4 $2 $3
sub $2 $1 $3

  – WAR is not a issue in MIPS 5-stage pipeline because
    all instructions take 5 stages, and reads are always in stage 2, and writes are always in stage 5, the following instruction never corrupt the previous instruction



Do not cause stall !!!
WAR is not data hazard

Types of Data Dependency (RAW, WAR, WAW)

Three types: (inst. i1 followed by inst. i2)

W R R

sub $2 $1 $3
sub $2 $1 $3

W R R

- WAW (write after write):
  i2 tries to write operand before i1 writes it
  - WAW is not an issue in MIPS 5-stage pipeline because all instructions take 5 stages, and writes are always in stage 5

Do not cause stall !!!
WAW is not data hazard

Quick summary:
Three data dependency: RAW, WAR, WAR
only RAW may cause data hazard in MIPS

Exercise

- Identify the data dependency (RAW, WAW, WAR) in the following instruction

```
lw $1,40($6)
add $6, $2, $2
sw $6, 50($1)
```

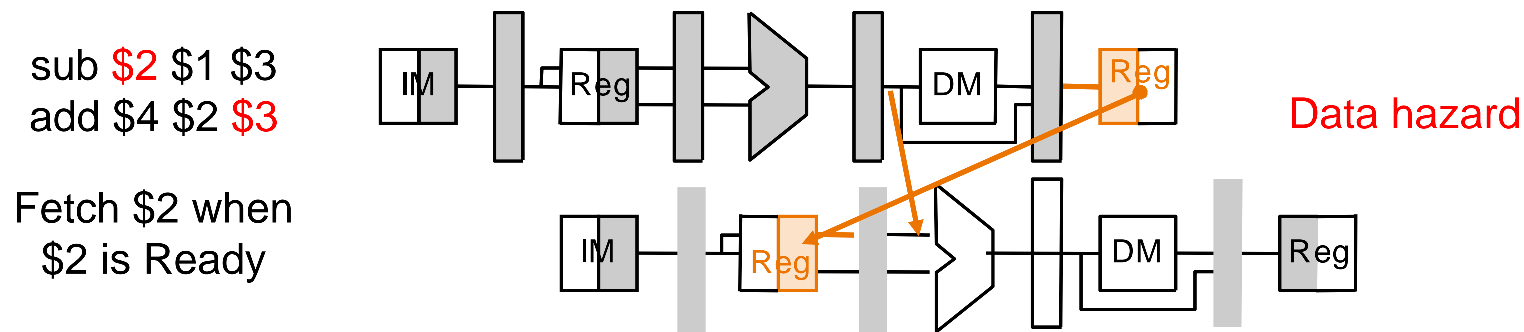
Ans:

```
lw $1,40($6)
   W      R
add $6, $2, $2
    W   R   R
sw $6, 50($1)
   R      R
```

I1 to I2: WAR on $6
I1 to I3: RAW on $1
I2 to I3: RAW on $6

Hardware Solution: Forwarding

- Idea: fetch “fresh” data as early as possible



Forwarding to remove hazard

- Two steps:

Step 1: Detect data hazard:

Is the datum just produced required by the following inst.?

Step 2: Forward intermediate data to resolve hazard

If yes, then forward the requested datum to the requesting inst. immediately.

Data Hazards

Time (in clock cycles)

Value of register $2:	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
	10	10	10	10	10/ -20	-20	-20	-20	-20

Forwarding to solve Data Hazards

Program execution order (in instructions)

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

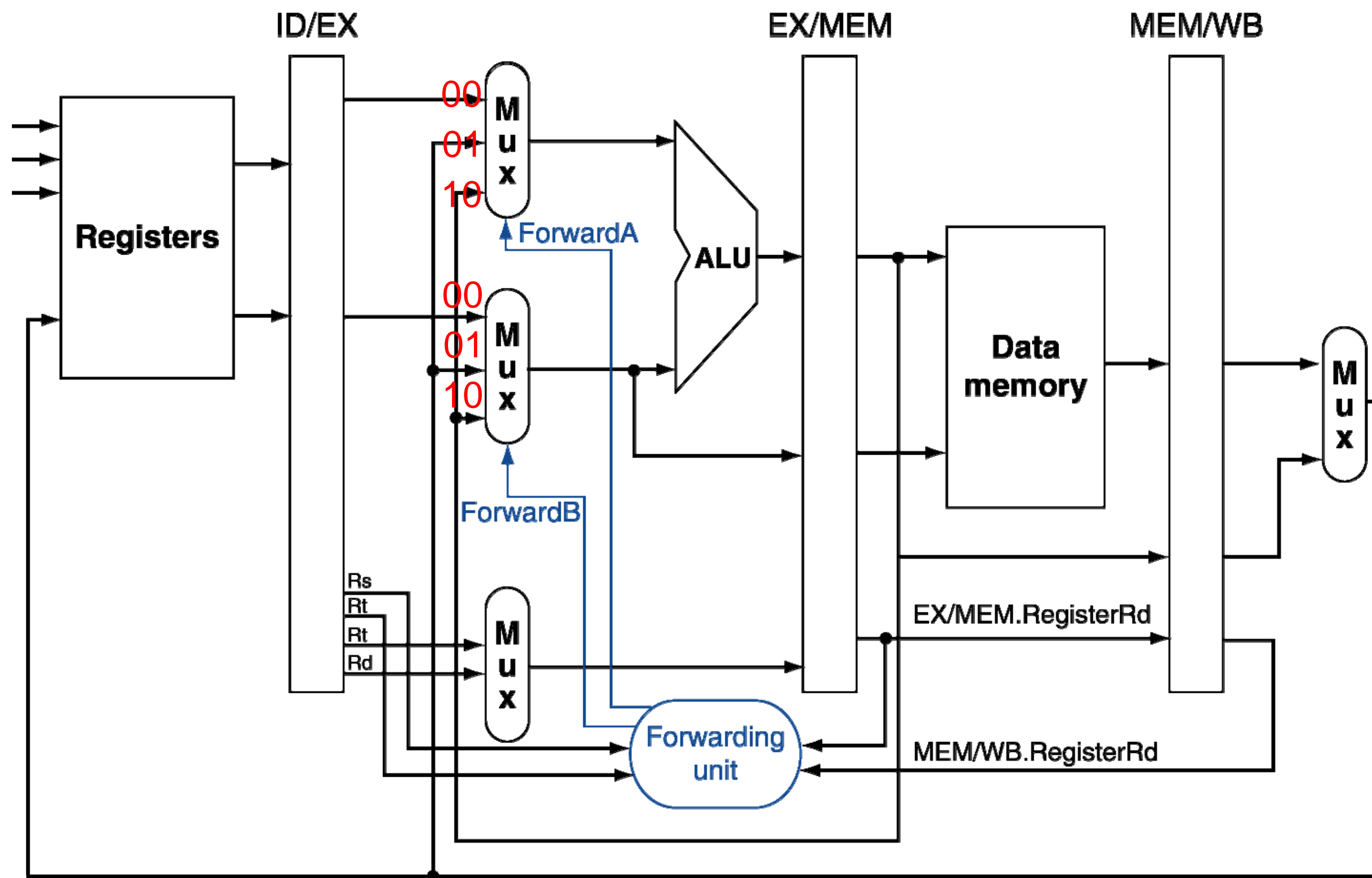
sw $15, 100($2)

Data Hazards

RAW, but No Hazards

Forwarding Hardware: Multiplexor Control

- Add Forwarding unit and ForwardA and ForwardB control signal to control mux (See next slides)



Forwarding Hardware: Multiplexor Control

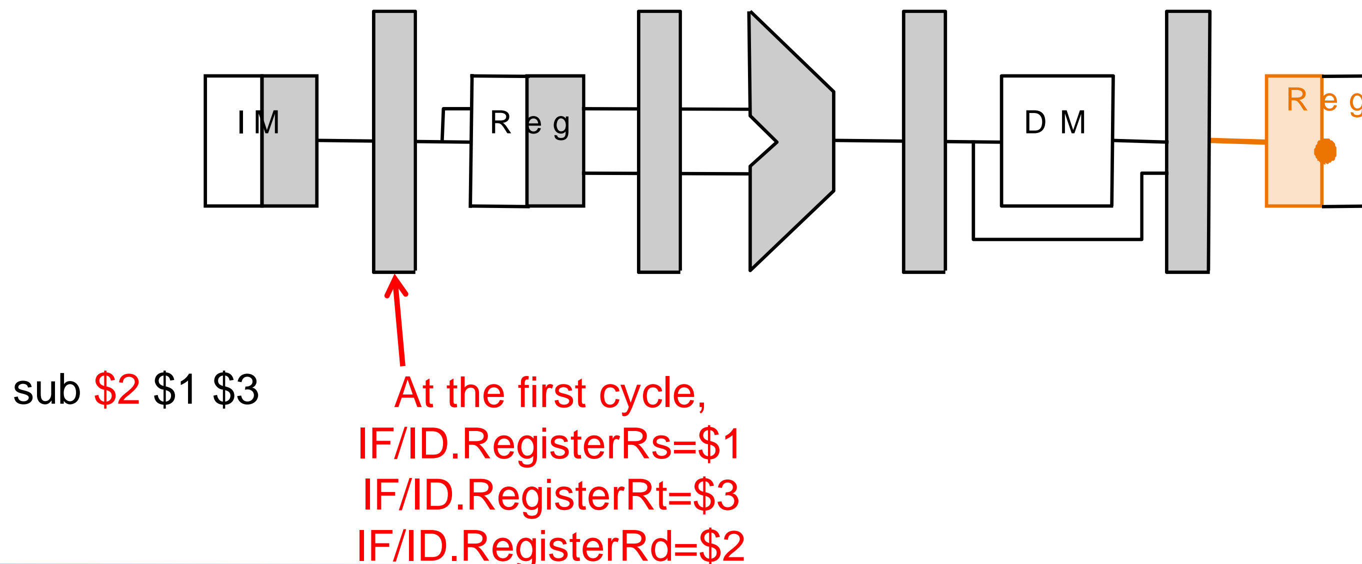
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand (Rs) comes from the register file
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from prior ALU result
ForwardA = 01	MEM/WB	*The first ALU operand is forwarded from data memory or an earlier ALU result

Mux control	Source	Explanation
ForwardB = 00	ID/EX	The second ALU operand (Rt) comes from the register file
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from prior ALU result
ForwardB = 01	MEM/WB	*The second ALU operand is forwarded from data memory or an earlier ALU result

* Depending on the selection in the rightmost multiplexor (see datapath with control diagram)

Detecting the Need to Forward

- Register numbers are passed along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- E.g.: ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt

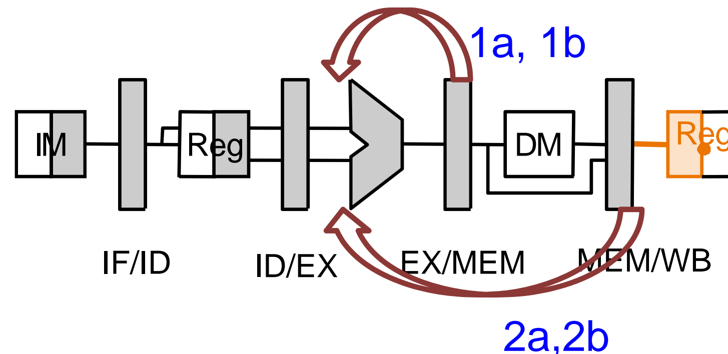


Detecting Data hazard

- Data hazards when
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from EX/MEM pipeline reg (1a, 1b)

Fwd from MEM/WB pipeline reg (2a, 2b)




```
 sub $2, $1, $3
and $12, $2, $5
```

Hazard is detected when the and is in EX stage and the sub is in MEM stage because

EX/MEM.RegisterRd = ID/EX.RegisterRs = $2 (1a)

Detecting EX hazard

- Forwarding is needed only if earlier instruction will write to a register!
  - Check if EX/MEM.RegWrite, MEM/WB.RegWrite is 1

```
 sub $2, $1, $3
and $12, $2, $5
```

- And only if Rd for that instruction is not $zero
  - When Rd=$zero, result is always zero
  - Check if EX/MEM.RegisterRd ≠ 0,MEM/WB.RegisterRd ≠ 0

```
 sub $0, $1, $3
and $12, $0, $5
```

Quick Summary

EX hazard:

If(EX/MEM.RegWrite and (EX/MEM.RegisterRd !=0)
and (EX/MEM.RegisterRd=ID/EX.RegisterRs)) ForwardA =10
If(EX/MEM.RegWrite and (EX/MEM.RegisterRd !=0)
and (EX/MEM.RegisterRd=ID/EX.RegisterRt)) ForwardB=10

MEM hazard:

If(MEM/WB.RegWrite and (MEM/WB.RegisterRd !=0)
and (MEM/WB.RegisterRd=ID/EX.RegisterRs)) ForwardA =01
If(MEM/WB.RegWrite and (MEM/WB.RegisterRd !=0)
and (MEM/WB.RegisterRd=ID/EX.RegisterRt)) ForwardB=01

Datapath with Forwarding

ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
Control
IF/ID
PC
Instruction memory
Instruction
Registers
M u x
ForwardA
ALU
M u x
ForwardB
Data memory
M u x
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
Rs
Rt
Rt
Rd
M u x
EX/MEM.RegisterRd
Forwarding unit
MEM/WB.RegisterRd

Outline

- A pipelined datapath
- Pipelined control
- Data hazards and forwarding
- Data hazards and stalls
- Branch hazards

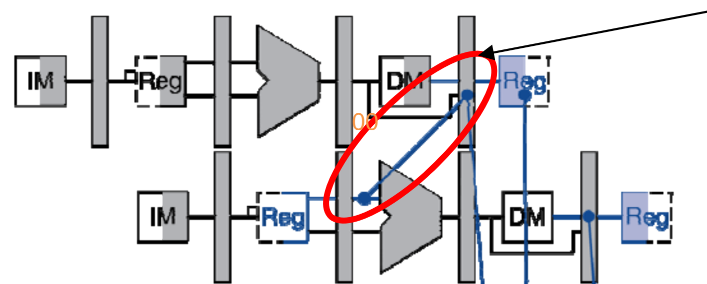
Load-Use Data Hazard

- One stall is needed for load-use data hazard

=> similar to add a nop (no operation) instruction

```
lw   $2, 20($1)

and $4, $2, $5
```

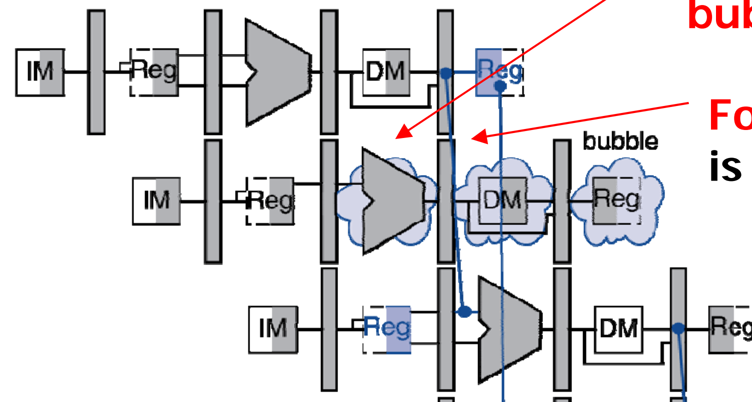


Need to stall for one cycle

```
lw   $2, 20($1)

and becomes nop

and $4, $2, $5
```



Hazard detection unit inserts a 1-cycle bubble in the pipeline

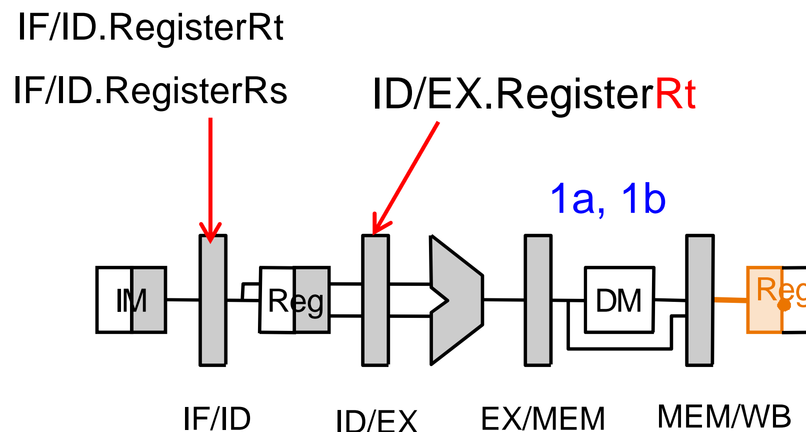
Forward data when it is ready

Hazard Detection Logic to Stall

- Recall: lw instruction format: `lw   Rt, offset(Rs)`
- Hardware to check if stall is needed

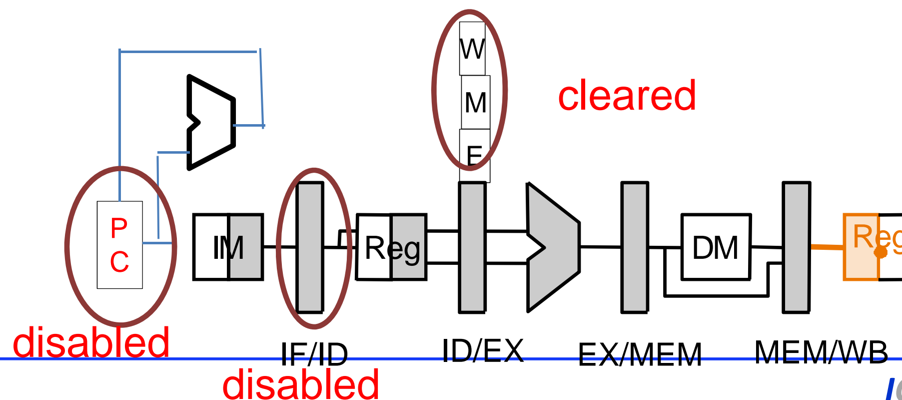
Destination Register in lw instruction.

```
if ( ID/EX.MemRead                        // if the instruction in the EX stage is a load…
   and ( ( ID/EX.RegisterRt = IF/ID.RegisterRs )       // and the destination register
     or  ( ID/EX.RegisterRt = IF/ID.RegisterRt ) ) )  // matches either source register
              // of the instruction in the ID stage, then  stall the pipeline
```



How to stall the Pipeline

- *Only Stall 1 clock cycle* after the load
  - the forwarding unit can resolve the dependency
- How hardware stalls the pipeline 1 cycle:
  - Disable write on IF/ID register => this will cause the instruction in the ID stage to repeat, i.e., *stall*
  - Disable write on PC => this will cause the instruction in the IF stage to repeat, i.e., *stall*
  - Changes all the EX, MEM and WB control fields in the ID/EX pipeline register to 0 => , so the instruction just behind the load becomes a nop – a bubble is said to have been inserted into the pipeline



Adding Hazard Detection Unit

Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
0
M u x
WB
M
EX
PC
Instruction memory
Instruction
Registers
ALU
Data memory
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit
EX/MEM.RegisterRd
MEM/WB.RegisterRd

Datapath with forwarding hardware, the hazard detection unit and controls wires – certain details, e.g., branching hardware are omitted to simplify the drawing

Cycle 2

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```

and $4, $2, $5

lw $2, 20($1)

before<1>

before<2>

before<3>

Hazard detection unit

ID/EX.MemRead

ID/EX

EX/MEM

MEM/WB

IF/ID

IF/IDWrite

PCWrite

Control

Mux

WB

M

EX

11

0

1

X

PC

Instruction memory

Instruction

Registers

$1

$X

1

X

2

ALU

Data memory

Forwarding unit

ID/EX.RegisterRt

Clock 2

Cycle 3

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```

Cycle 4

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```


Clock 4

Cycle 5

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```

add $9, $4, $2 | or $4, $4, $2 | and $4, $2, $5 | bubble | lw $2, . . .

Hazard detection unit

ID/EX.MemRead

ID/EX

IF/IDWrite

PCWrite

IF/ID

EX/MEM

MEM/WB

Control

M u x

WB M EX

10 10 0 0 11

PC

Instruction memory

Instruction

Registers

$4 $2 $2 $5

M u x

ALU

Data memory

Forwarding unit

ID/EX.RegisterRt

Clock 5

Cycle 6

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```

after<1> | add $9, $4, $2 | or $4, $4, $2 | and $4, . . . | bubble

Hazard detection unit
ID/EX.MemRead
IF/IDWrite
PCWrite
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
PC
Instruction memory
Instruction
Registers
ALU
Data memory
Forwarding unit
ID/EX.RegisterRt
WB M EX
M u x

Clock 6

Cycle 7

```
lw  $2, 20($1)
and $4, $2, $5
or  $4, $4, $2
add $9, $4, $2
```


Clock 7

Backup Slides