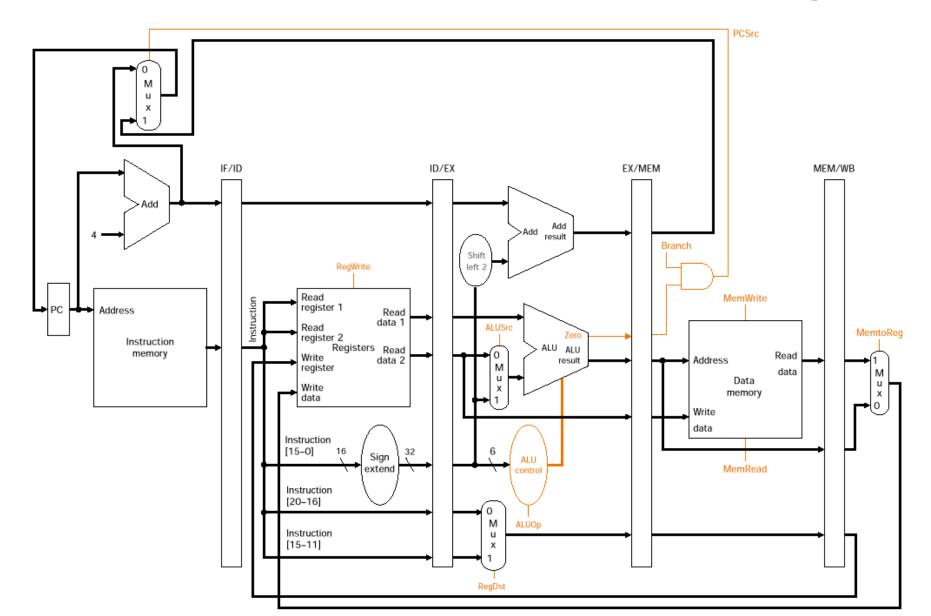
Lecture – 18

Pipelined Datapath with Control Signals



Control Signal for the Pipeline Datapath

Instruction opcode	ALUOp	Instruction operation	Function code	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	set on less than	0111

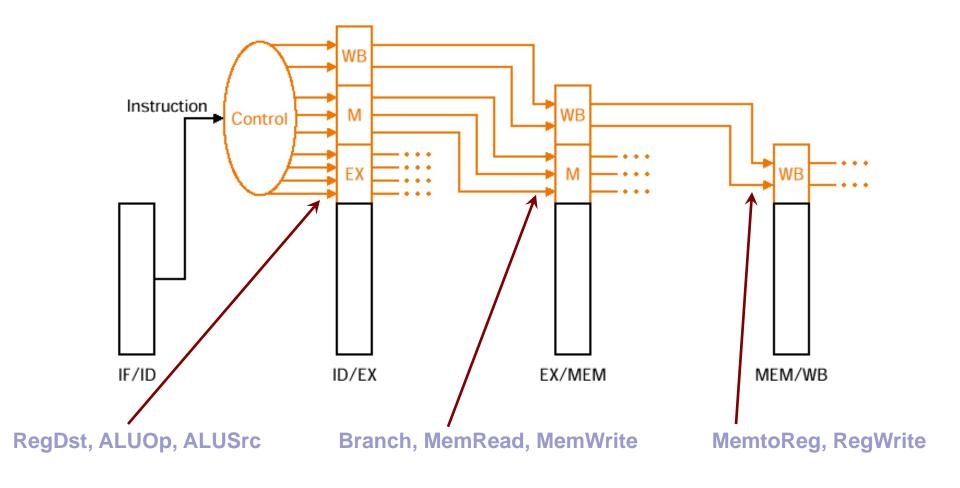
Control Signal for the Pipeline Datapath

Signal name	Effect when deasserted (0)	Effect when asserted (1)
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 rd 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.

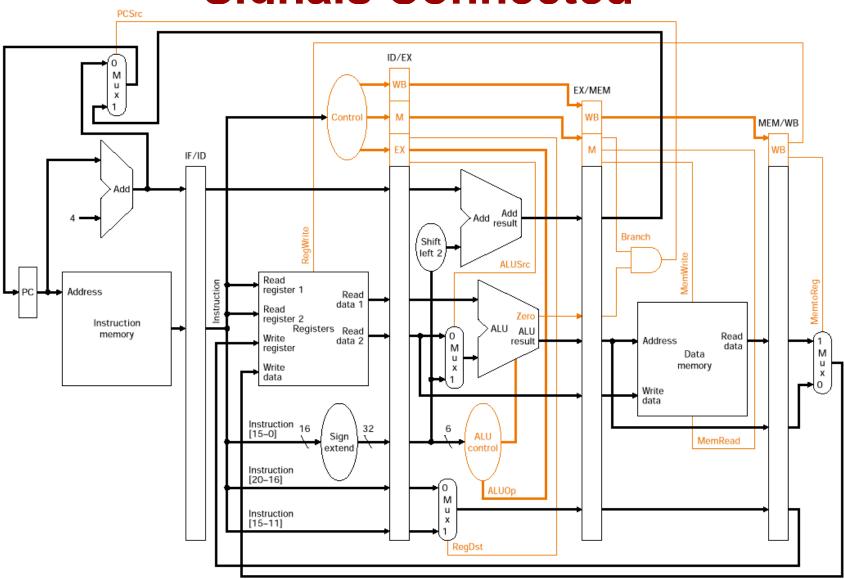
Control Signal for the Pipeline Datapath

	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
Instruction	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
1 w	0	0	0	1	0	1	0	1	1
SW	X	0	0	1	0	0	1	0	Х
beq	X	0	1	0	1	0	0	0	Х

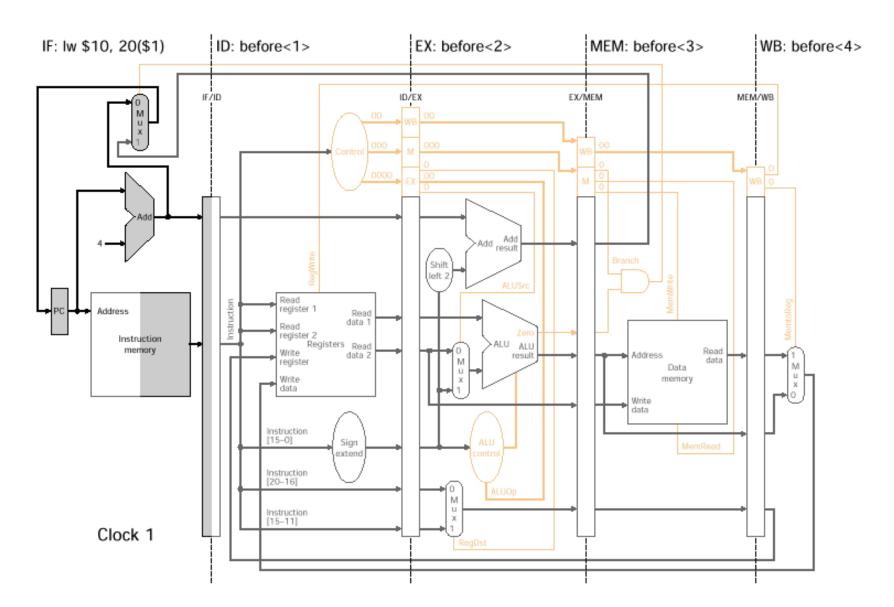
Control Signals Generation

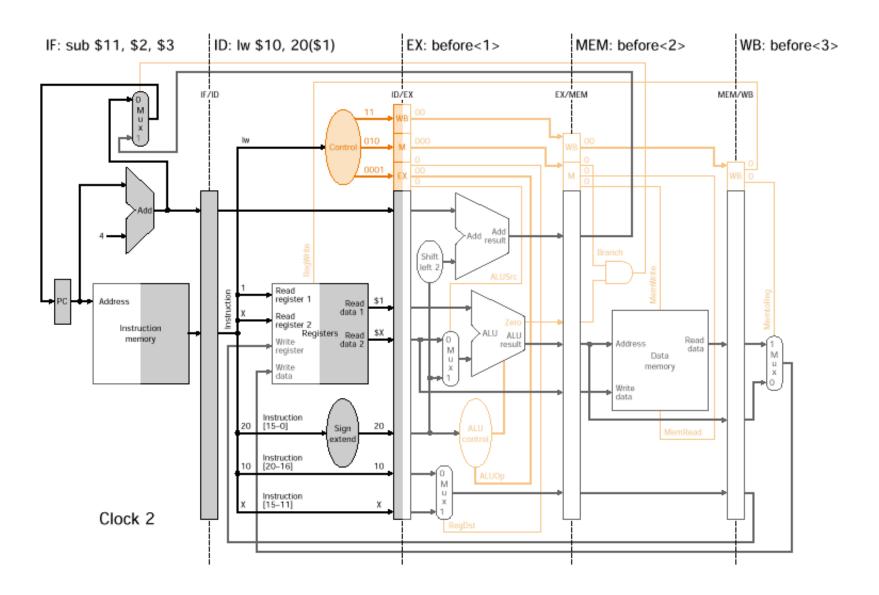


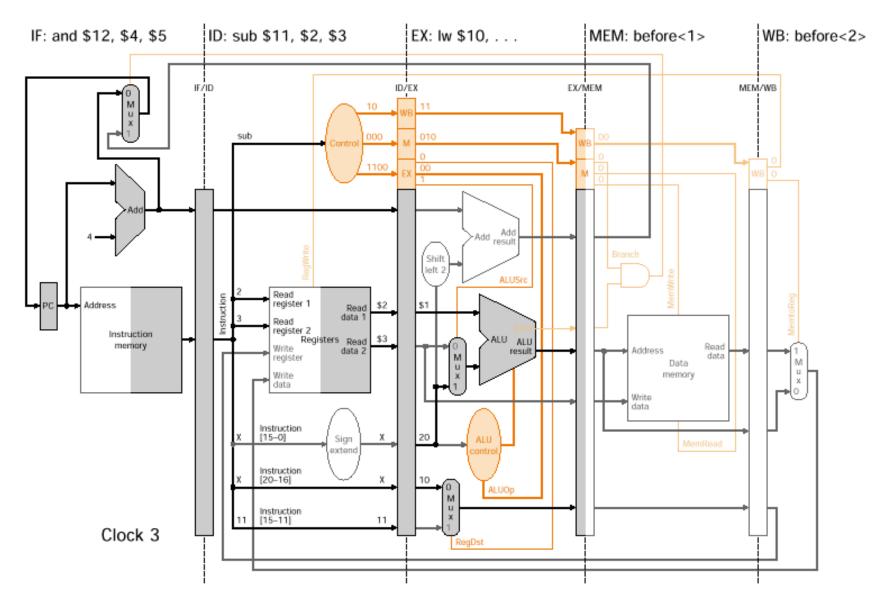
Pipelined Datapath with control Signals Connected

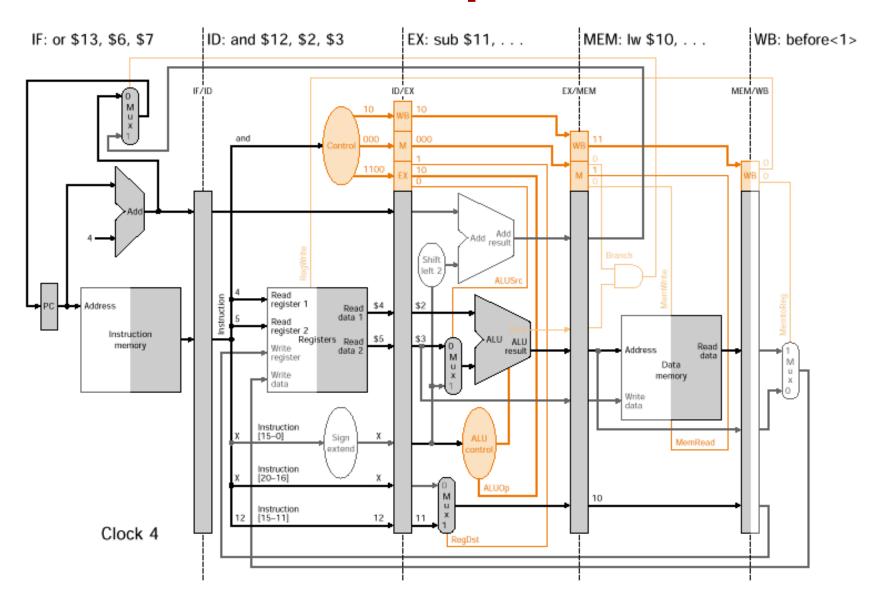


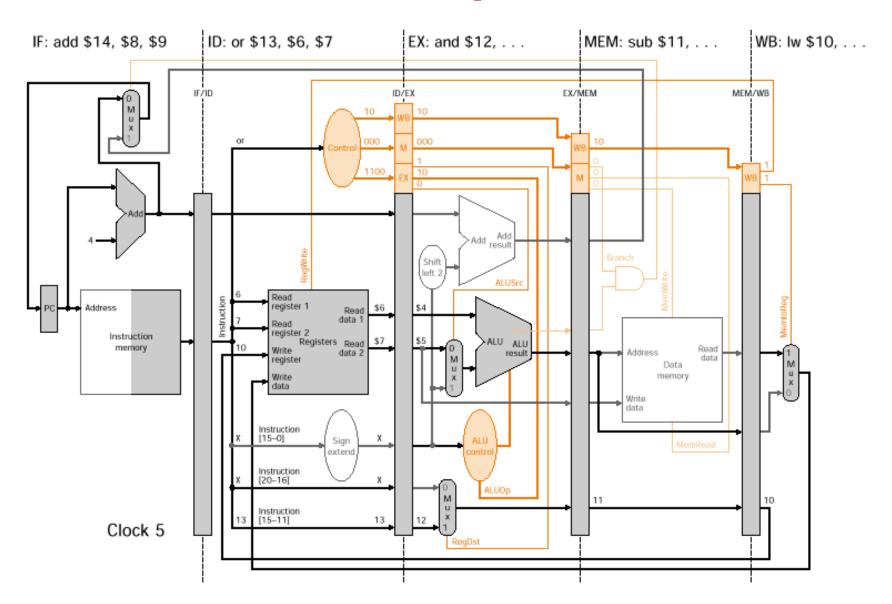
```
Iw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
```

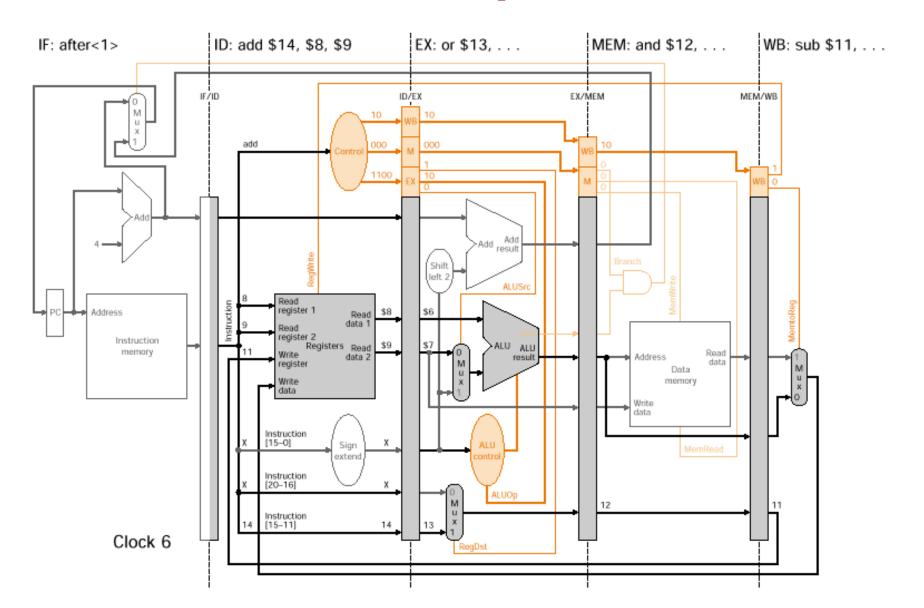


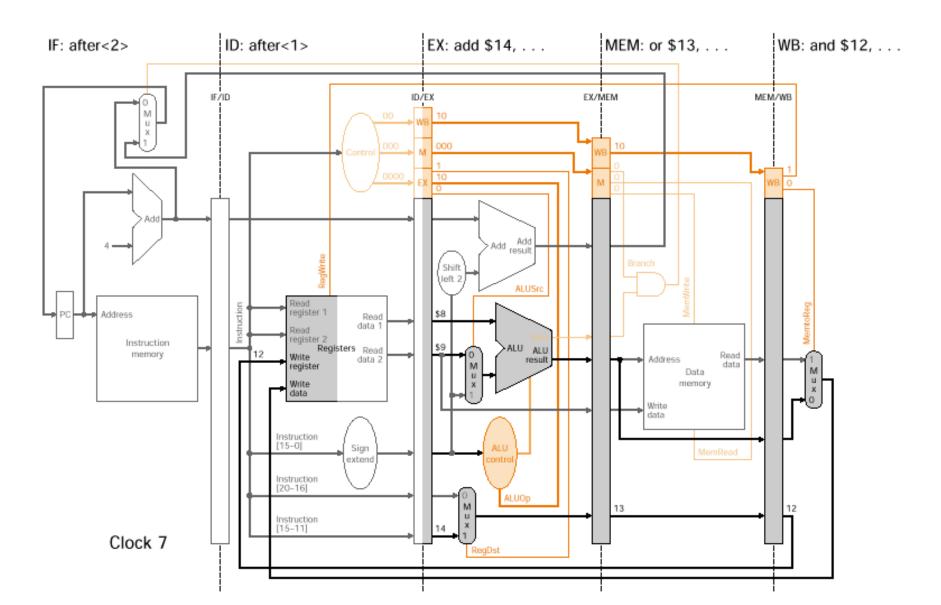


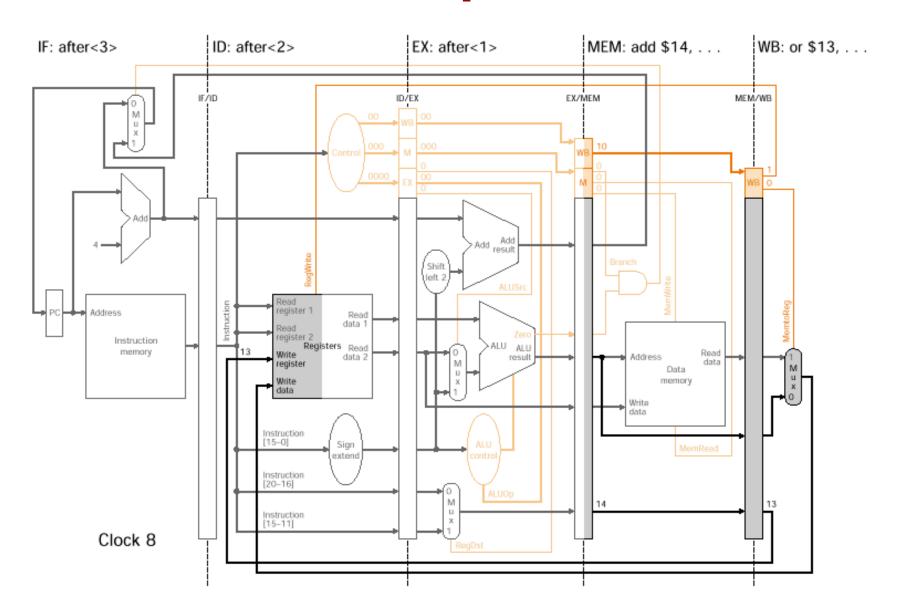


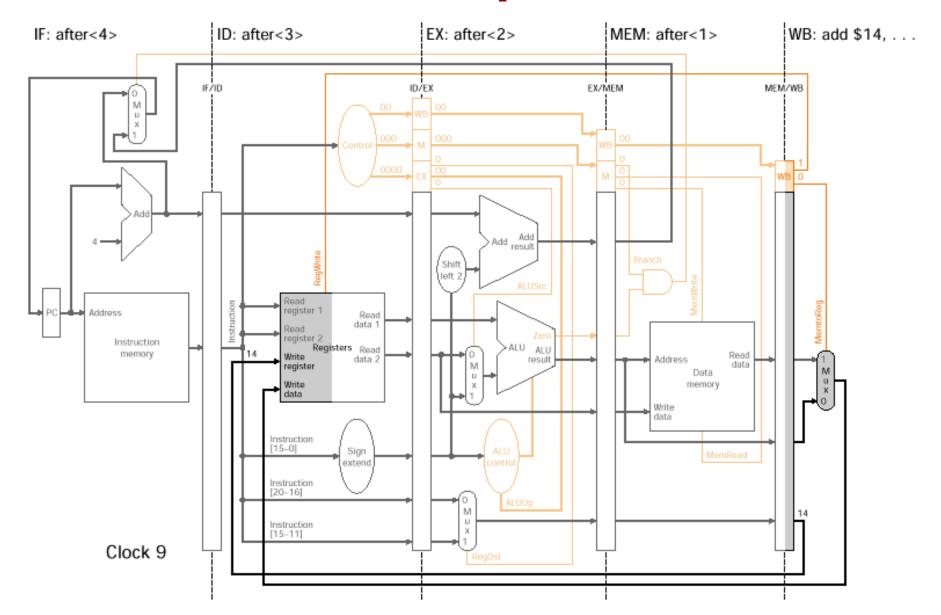












M

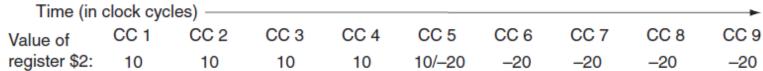
Data Hazards

Consider the following code segments:

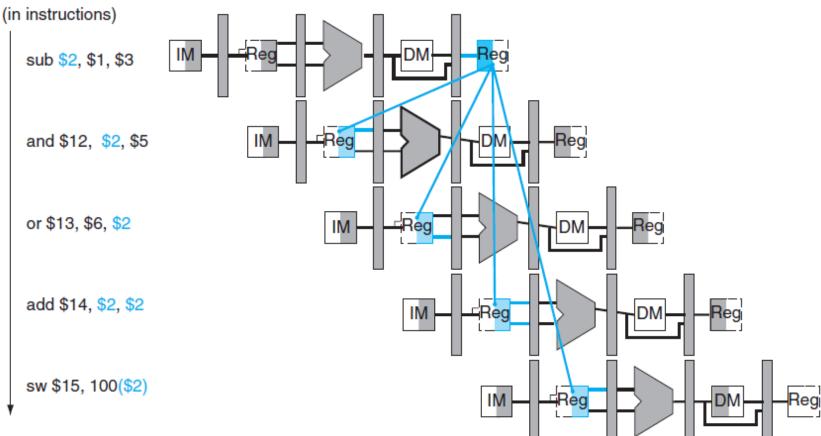
```
sub $2, $1, $3  //$2 written by sub
and $12, $2, $5  //1st operand ($2) depends on sub
or $13, $6, $2  //2nd operand ($2) depends on sub
add $14, $2, $2  //1st ($2) and 2nd ($2) operand depend on sub
sw $15, 100($2)  //base ($2) depends on sub
```

One hazard resolved by the design of register file. Write in the first half of the clock cycle and the read in the second half. So read delivers what is written.

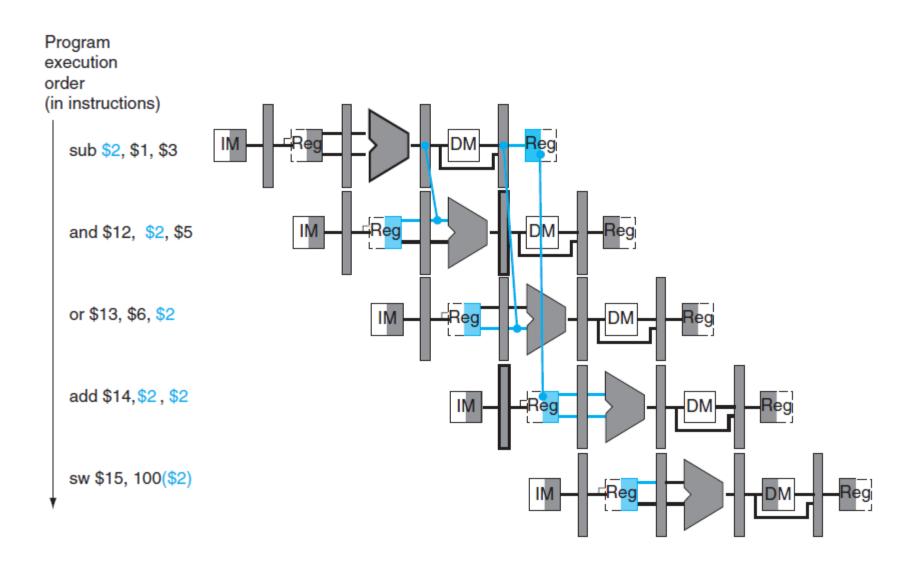
Pipelined Dependences



Program execution order (in instruc



Fixing Data Hazard by Forwarding



м

Data Forwarding

- Data can be forwarded from EX/MEM register and MEM/WB registers.
- Forward by taking the inputs to the ALU from any pipeline register rather than just ID/EX by
 - adding multiplexors to the inputs of the ALU so can pass Rd back to either (or both) of the EX's stage Rs and Rt ALU inputs
 - 00: normal input (ID/EX pipeline registers)
 - 10: forward from previous instr (EX/MEM pipeline registers)
 - 01: forward from instr 2 back (MEM/WB pipeline registers)
 - adding the proper control hardware
- With forwarding can run at full speed even in the presence of data dependencies

Two Pairs of Hazard Conditions

EX/MEM Hazard:

```
    1a.if (EX/MEM.RegisterRd ==ID/EX.RegisterRs)
        forwardA=10 [Between sub and and]
    1b. If (EX/MEM.RegisterRd == ID/EX.RegisterRt)
        forwardB=10
```

MEM/WB Hazard:

```
2a. If (MEM/WB.RegisterRd == ID/EX.RegisterRs) forwardA=01
```

2b. If (MEM/WB.RegisterRd == ID/EX.RegisterRt) forwardB=01 [Between sub and or]



Hazard Conditions

1. EX/MEM hazard:

Forwards the result from the previous instr. to either input of the ALU provided it writes.

2. MEM/WB hazard:

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd == ID/EX.RegisterRs)) ForwardA = 01 if (MEM/WB.RegWrite and (MEM/WB.RegisterRd == ID/EX.RegisterRt)) ForwardB = 01 Forwards the result from the second previous instr. to either input of the ALU provided it writes.



Hazard Conditions

1. EX/MEM hazard:

```
if (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
```

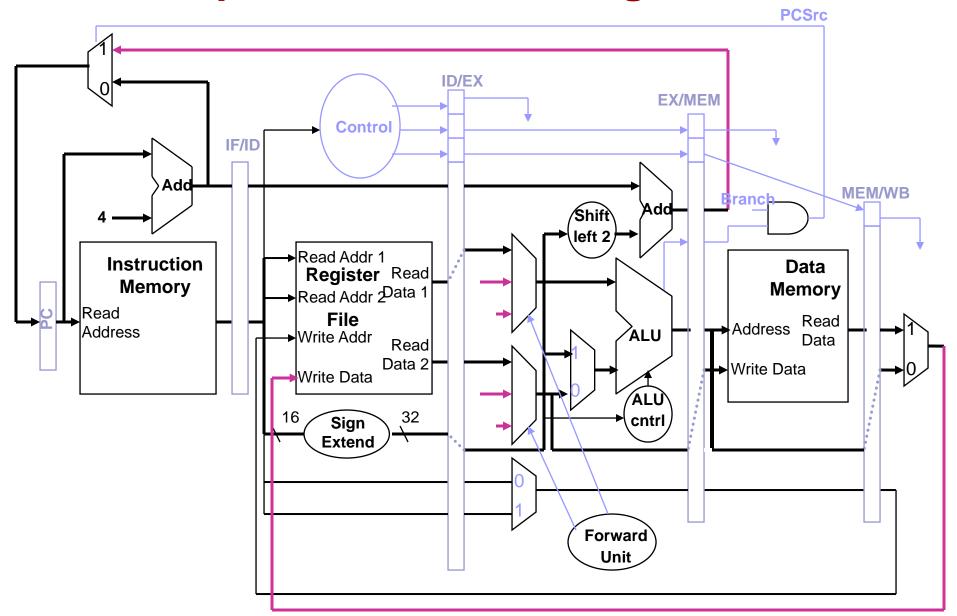
Forwards the result from the previous instr. to either input of the ALU provided it writes and != 0.

2. MEM/WB hazard:

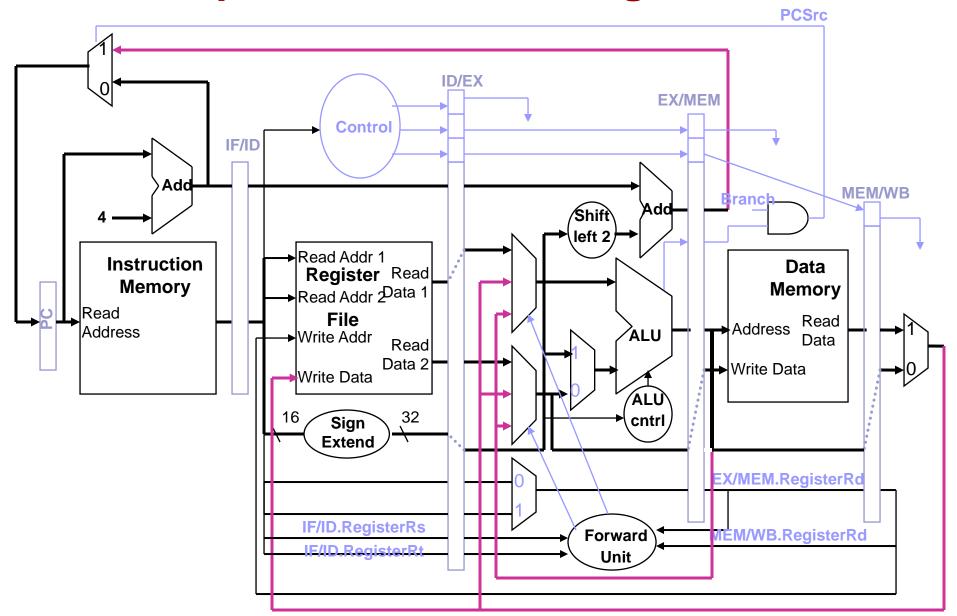
```
if (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd == ID/EX.RegisterRs))
ForwardA = 01
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd == ID/EX.RegisterRt))
ForwardB = 01
```

Forwards the result from the second previous instr. to either input of the ALU provided it writes and != 0

Datapath with Forwarding Hardware



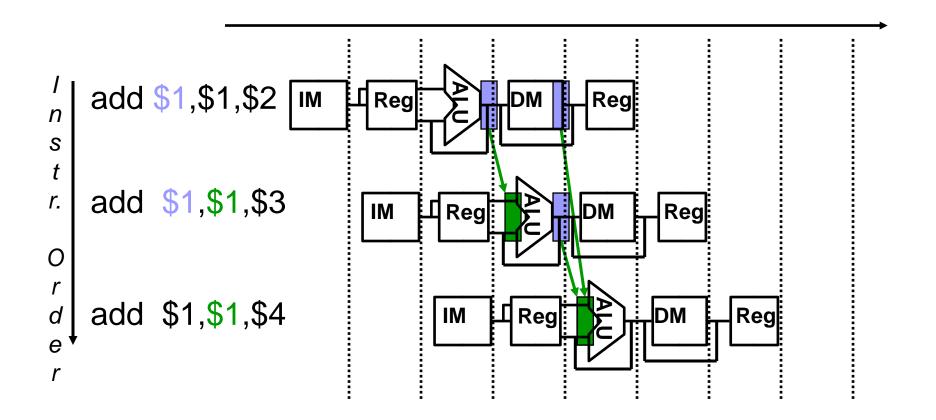
Datapath with Forwarding Hardware



м

Yet Another Complication!

 Another potential data hazard can occur when there is a conflict between the result of the WB stage instruction and the MEM stage instruction – which should be forwarded? More recent result!



٠,

Hazard Conditions

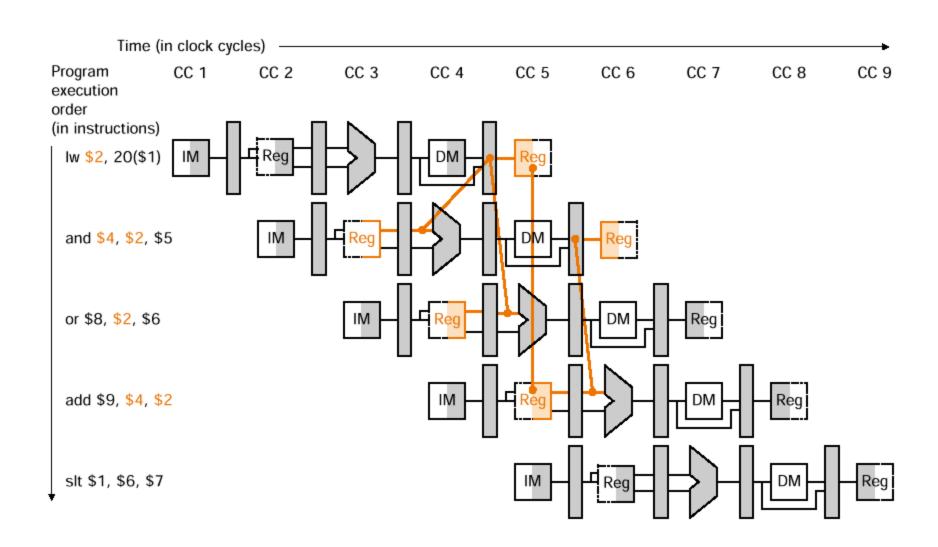
2. MEM/WB hazard:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (EX/MEM.RegisterRd != ID/EX.RegisterRs)
and(MEM/WB.RegisterRd == ID/EX.RegisterRs))
ForwardA = 01
```

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (EX/MEM.RegisterRd != ID/EX.RegisterRt )
and (MEM/WB.RegisterRd == ID/EX.RegisterRt))
ForwardB = 01
```

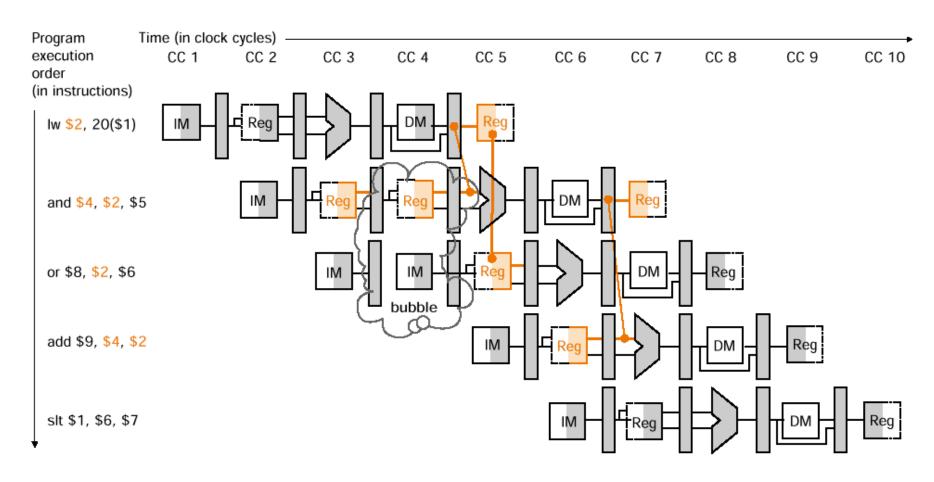
1

Data Hazard Requiring a Stall



7

Data Hazard Requiring a Stall

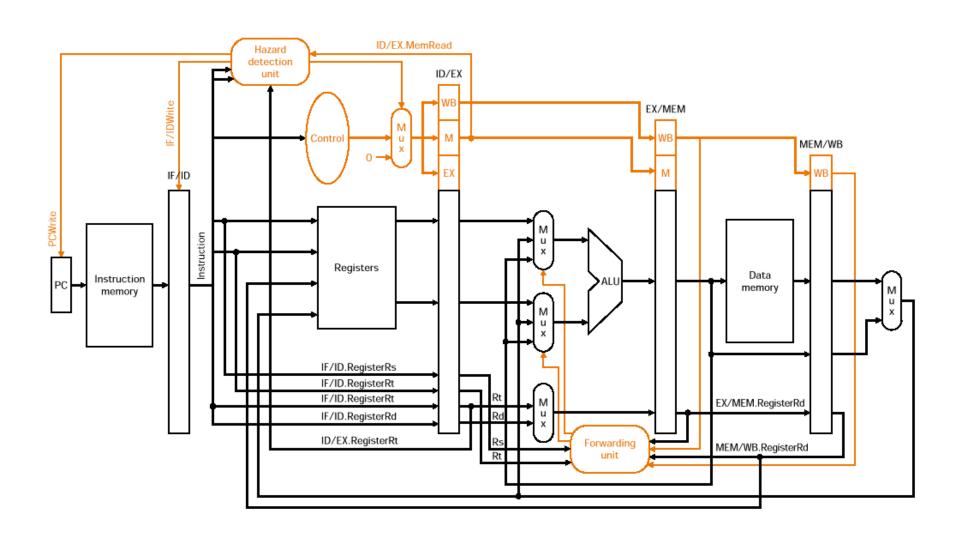




Hazard Detection Unit

- We need hazard detection unit beside the forwarding unit.
- It operates during the ID stage so that it can insert the stall between the load and it's use.
- Condition if (ID/EX.MemRead and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))) stall the pipeline

Stall Logic





Nops

- Both instructions in ID and IF stage must be stalled.
- For this reason, the PC register and the IF/ID register are preserved.
- The EX stage must do nops.
- To insert nops we must deassert all nine control signal in EX, MEM and WB stage.