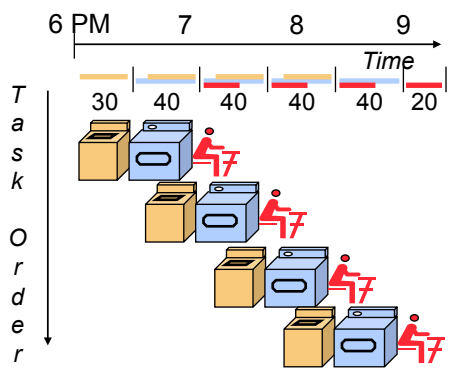
Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- *Washer* takes 30 minutes, *Dryer* takes 40 minutes, *Folder* takes 20 minutes



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

(1)

Computer Pipelines

- MIPS desirable features:
  - all instructions same length,
  - registers located in same place in instruction format,
  - memory operands only in loads or stores
- We will first review a non-pipelined MIPS architecture.
- Review – you should know about:
  - multiplexors,
  - register files,
  - ALU's
  - arithmetic Vs logical shifts
  - program counter (PC), status word (PS) and instruction register (IR),
  - Memory address registers (MAR) and memory data registers (MDR)
  - combinational Vs sequential circuits

(2)

Implementing the MIPS architecture

Arithmetic/logic instructions
1) fetch instruction
2) read registers
3) compute (use ALU)
4) write to register

Memory instructions
1) fetch instruction
2) read registers
3) compute address (use ALU)
4) write/read to/from memory
5) write to register (for load)

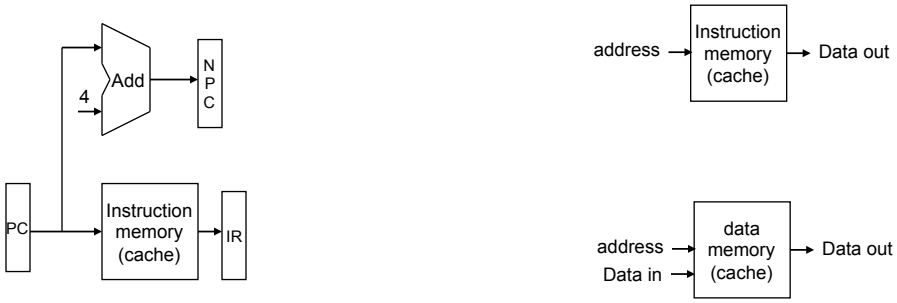
Branch instructions
1) fetch instruction
2) read registers
3) compute branch address (use ALU)
4) evaluate branch condition
5) update the PC (condition satisfied)

Increment the PC

(3)

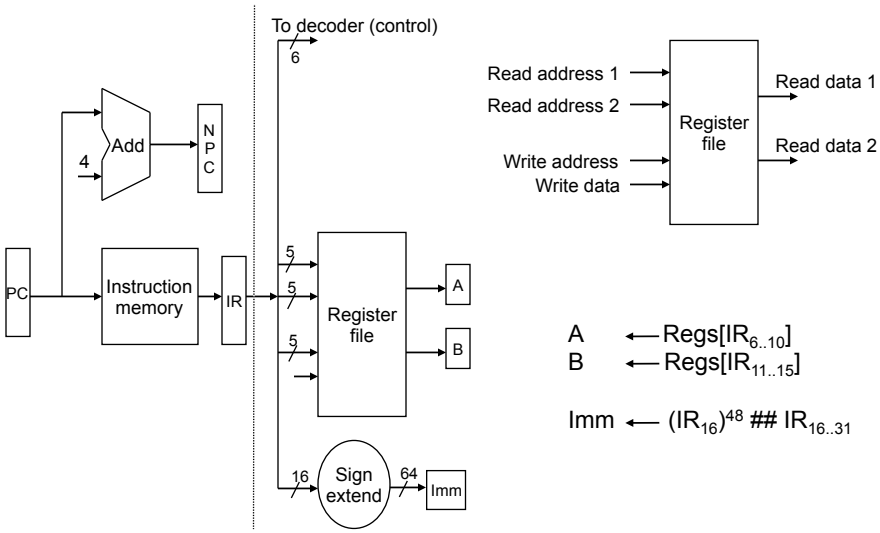
Instruction fetch

IR ← Mem[PC]
NPC ← PC + 4



(4)

Register read

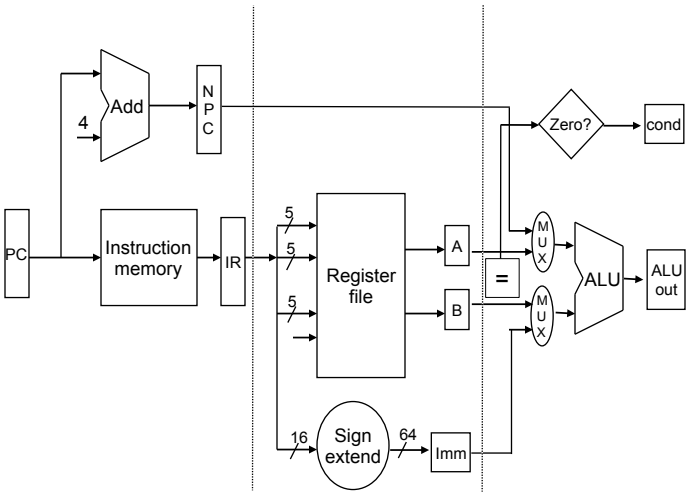


A ← Regs[$IR_{6..10}$]
B ← Regs[$IR_{11..15}$]

Imm ← $(IR_{16})^{48}$ ## $IR_{16..31}$

(5)

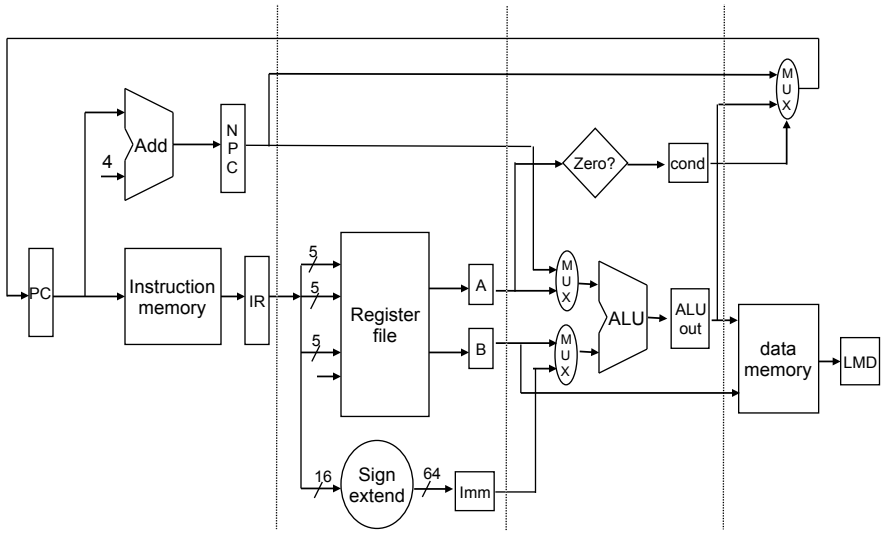
Use ALU and evaluate branch condition



(the multiplexors settings and ALU function depend on the op-code)

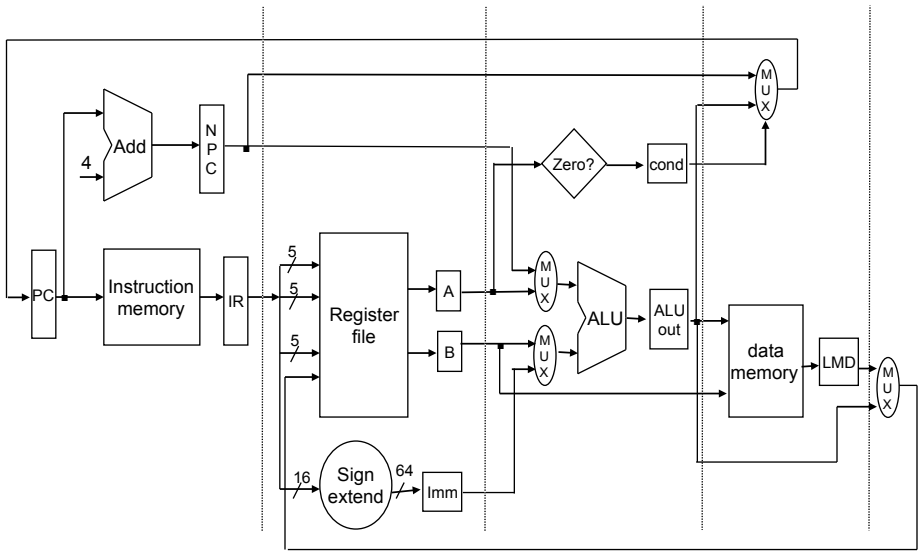
(6)

Use memory



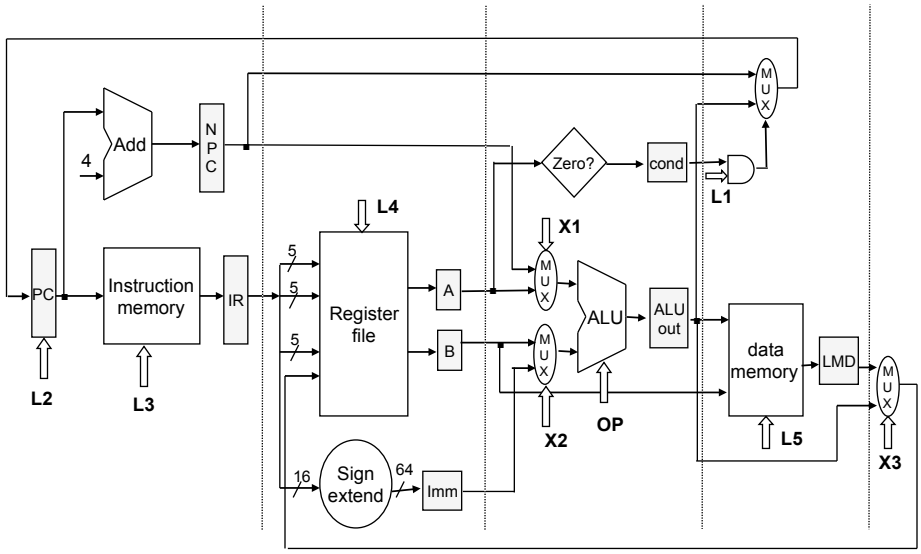
(7)

Write to register



(8)

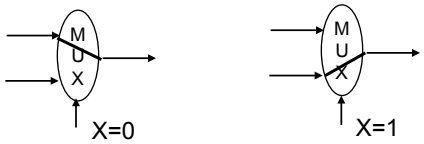
The control signals



(9)

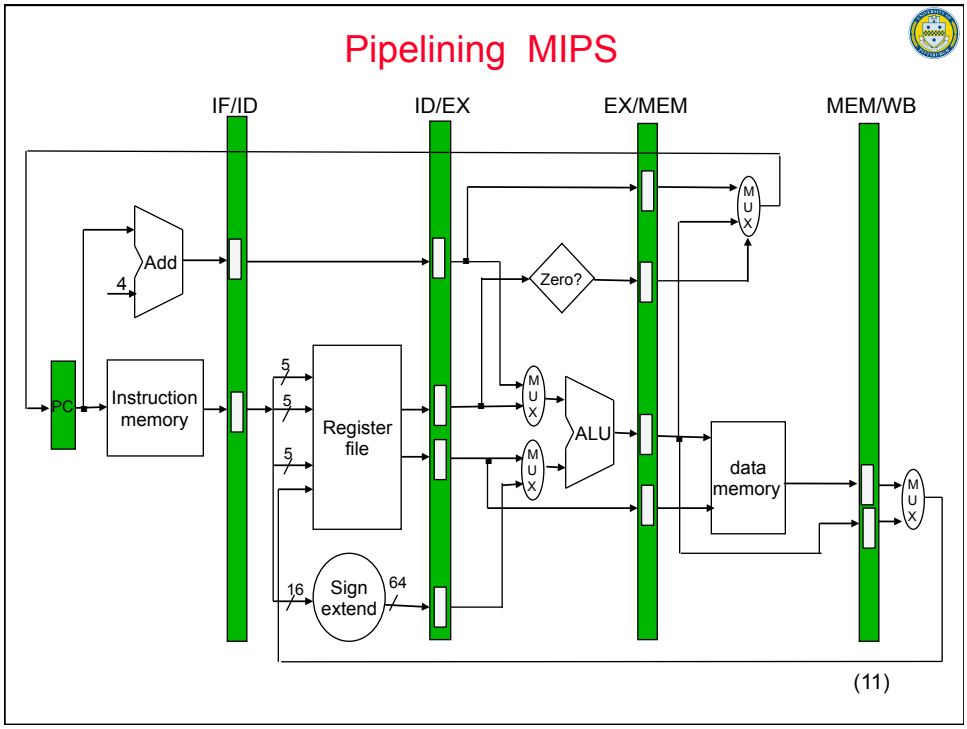
The control signals

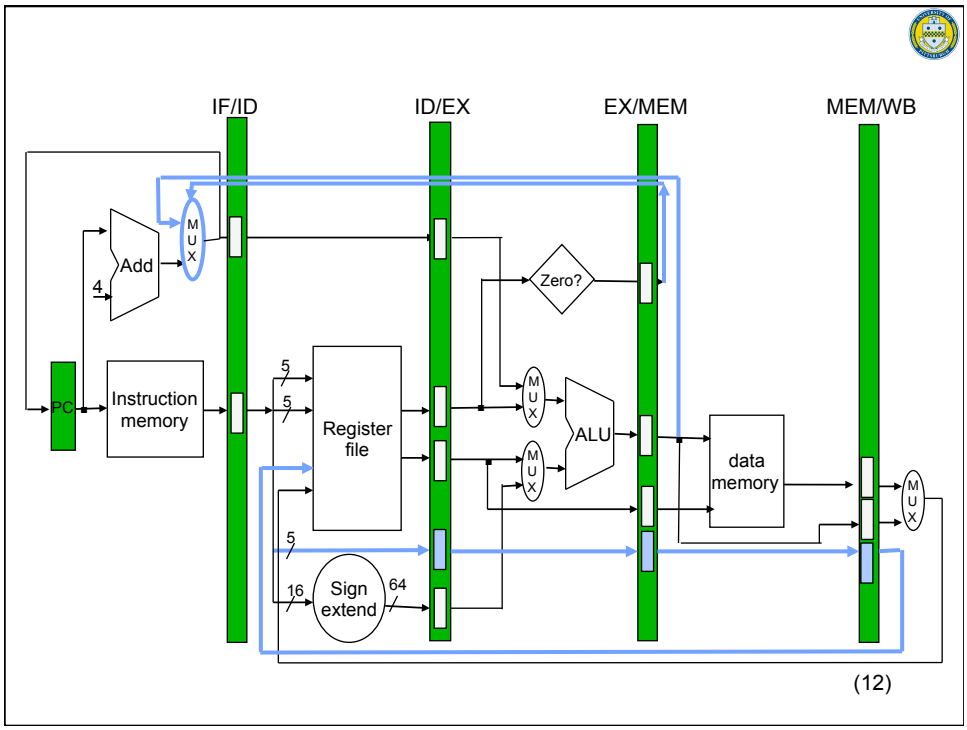
- X1, X2, X3 = select multiplexor input (one bit each)



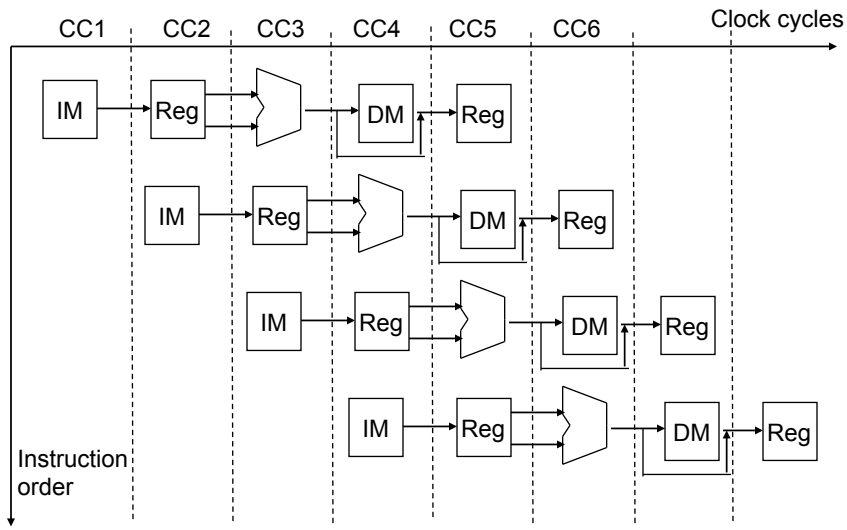
- L1 = set if the instruction is a branch (one bit)
- L2 = loads the PC (one bit)
- L3 = read the instruction memory (one bit)
- L4 = read/write register (two bits)
  00 = no-op, 01 = read, 10 = write
- L5 = read/write the data memory (two bits)
  00 = no-op, 01 = read, 10 = write
- OP = ALU control (?? Bits)
  0..0 = no-op, 1..1 = add, 10.. = others

(10)


Pipelining MIPS
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Instruction memory
5
5
5
Register file
16
Sign extend
64
Zero?
M U X
ALU
data memory
(11)


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M U X
PC
Instruction memory
5
5
Register file
5
16
Sign extend
64
Zero?
ALU
data memory
(12)

Visualizing pipelining



Pipelining puts additional demands on memory bandwidth, (13)

Visualizing pipelining

	CC1	CC2	CC3	CC4	CC5	CC6	Clock cycles
lw	IM	Reg	Alu	Mem	WB		
add		IM	Reg	Alu	Mem	WB	
sw			IM	Reg	Alu	Mem	WB

Clock cycles	**IM**	**Reg**	**Alu**	**Mem**	**WB**
	lw				
	add	lw			
	sw	add	lw		
		sw	add	lw	
			sw	add	lw
				sw	add

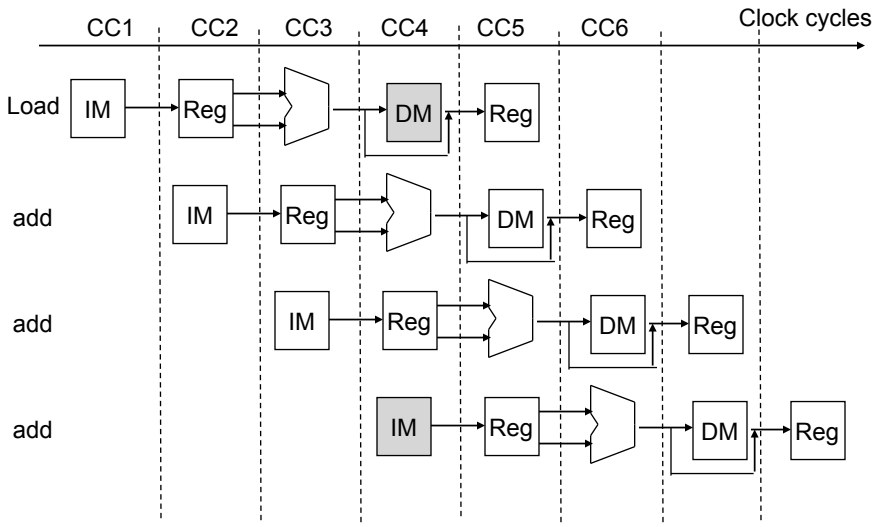
(14)

Limits to pipelining:

- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support a combination of instructions.
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline.
  - Control hazards: Pipelining of branches & other instructions *stall* the pipeline until the hazard *bubbles* in the pipeline

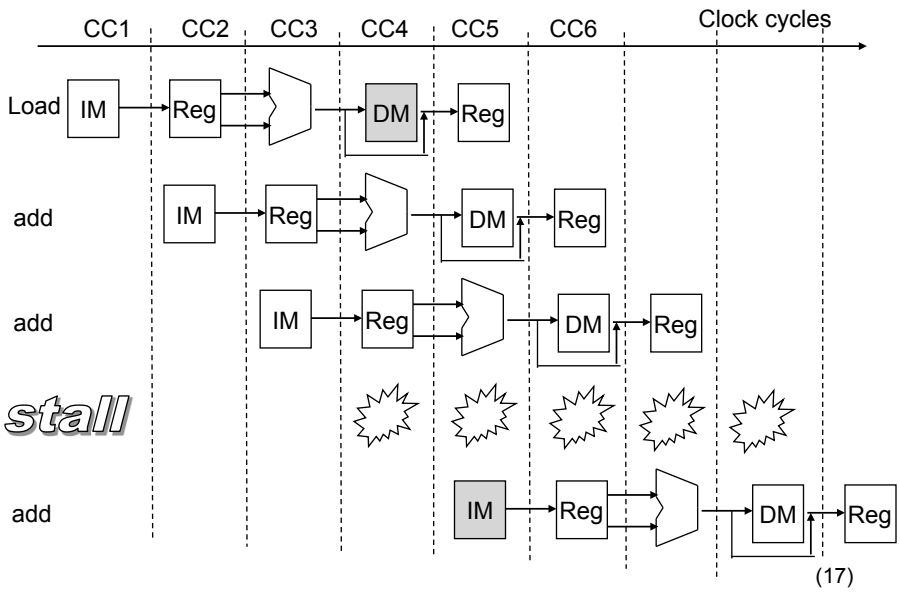
(15)

Structural Hazards (assuming a single memory)



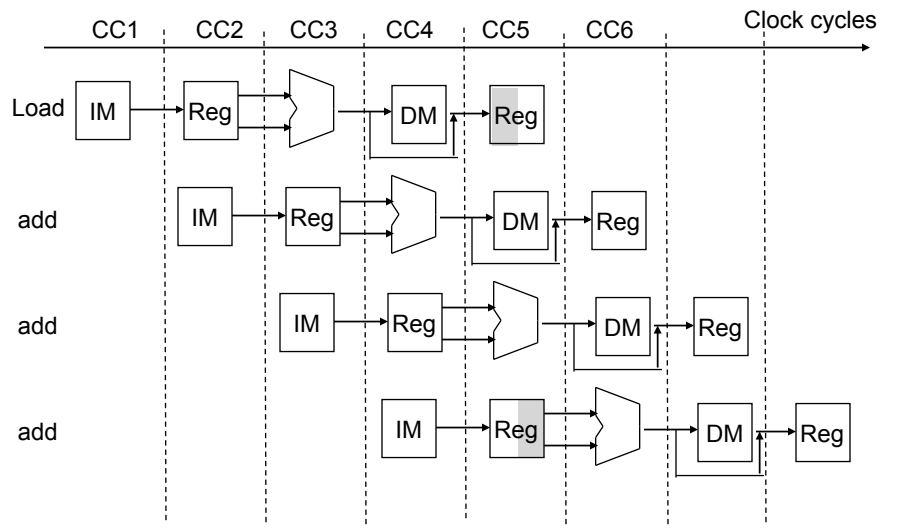
(16)

Structural Hazards (assuming a single memory)



(17)

Structural hazard in register file



(18)

Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{stall cycles per instruction}$$

$$\text{Speedup} = \frac{CPI_{unpipelined}}{CPI_{pipelined}} \times \frac{\text{Clock Cycle}_{unpipelined}}{\text{Clock Cycle}_{pipelined}}$$

Example:

- Machine A: pipelined (with some depth) and dual ported memory
- Machine B: pipelined (same depth as A), but single ported memory, and a 1.05 times faster clock rate
- Ideal CPI = 1 for both, and loads are 40% of instructions executed

$$\text{SpeedUp}_A = \text{Pipeline Depth}$$

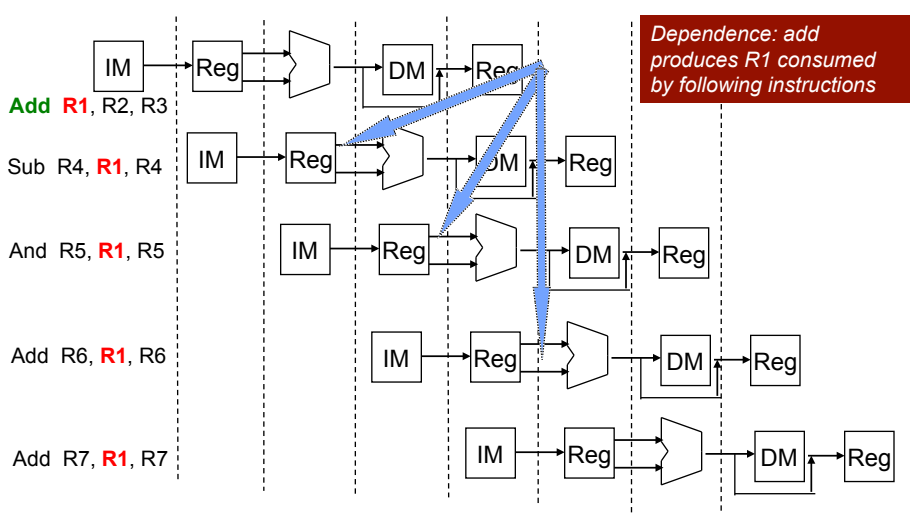
$$\text{SpeedUp}_B = (\text{Pipeline Depth}/1.4) \times 1.05$$

$$= 0.75 \times \text{Pipeline Depth}$$

$\text{SpeedUp}_A / \text{SpeedUp}_B = 1.33$

(19)

Register-to-register data Hazards



(20)

Three types of Data Dependence

$Instr_i$* is later in the pipeline than *$Instr_j$

j depends on i for operand R

1. **Read After Write (RAW)**

   $Instr_j$ tries to read operand *before* *$Instr_i$* writes it

2. **Write After Read (WAR)**

   $Instr_j$ tries to write operand *before* *$Instr_i$* reads it

   – Gets wrong operand
   – Can't happen in MIPS 5 stage pipeline (why?)

3. **Write After Write (WAW)**

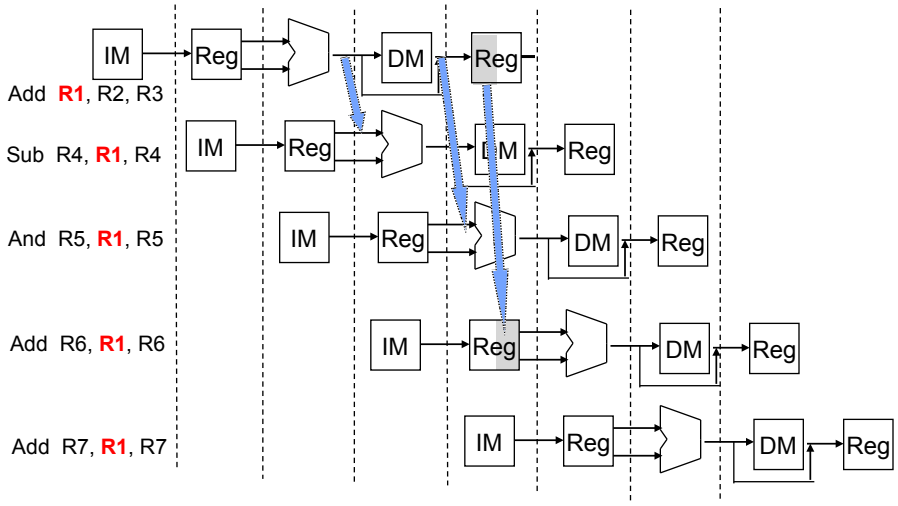
   $Instr_j$ tries to write operand *before* *$Instr_i$* writes it

   – Leaves wrong result
   – Can't happen in MIPS 5 stage pipeline (why?)

Will see WAR and WAW in later more complicated pipes

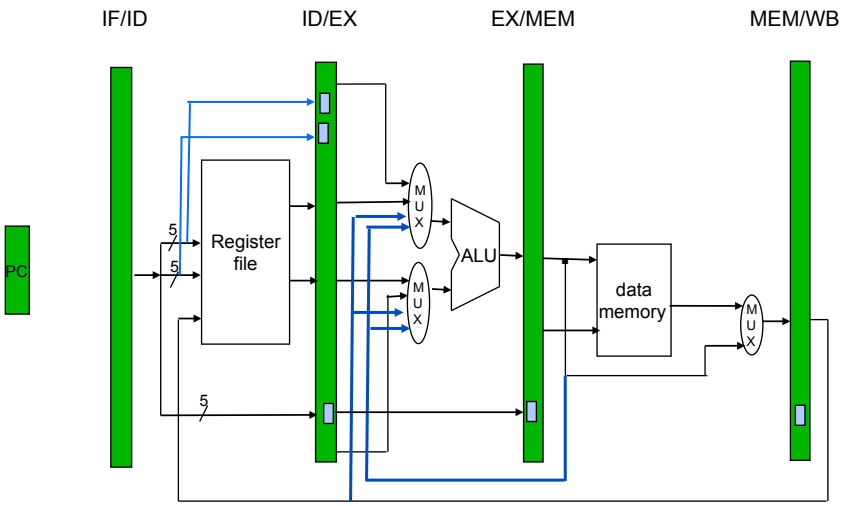
(21)

Forwarding to Avoid Data Hazard



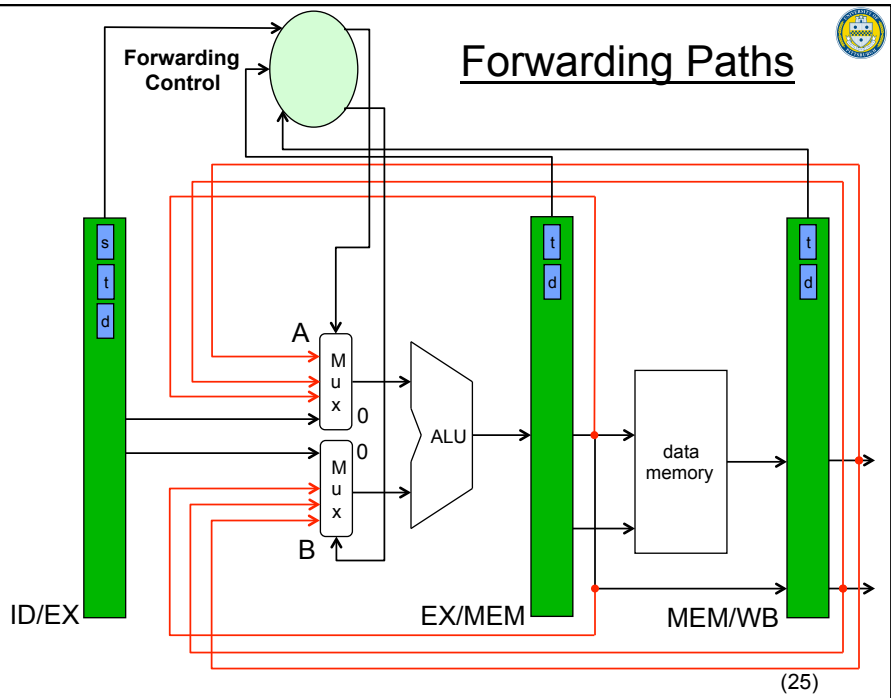
(22)

HW Change for Forwarding



(23)

Forwarding Paths



(25)

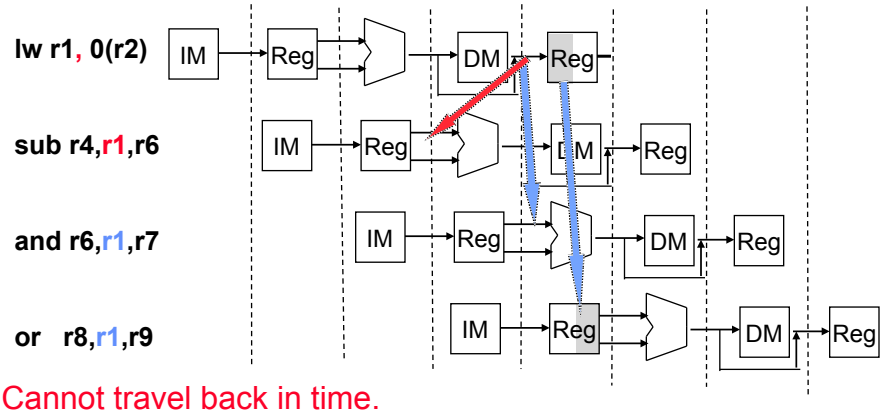
Detection & Activation for Forwarding Control

Src Type	Detection Condition	Input	Action	Priority
R-R	EX/MEM.rd == ID/EX.rs	Mux A	1	1
R-R	EX/MEM.rd == ID/EX.rt	Mux B	1	1
R-R	MEM/WB.rd == ID/EX.rs	Mux A	2	2
R-R	MEM/WB.rd == ID/EX.rt	Mux B	2	2
Imm	EX/MEM.rt == ID/EX.rs	Mux A	1	1
Imm	EX/MEM.rt == ID/EX.rt	Mux B	1	1
Imm	MEM/WB.rt == ID/EX.rs	Mux A	2	2
Imm	MEM/WB.rt == ID/EX.rt	Mux B	2	2
Ld	MEM/WB.rt == ID/EX.rs	Mux A	3	2
Ld	MEM/WB.rt == ID/EX.rs	Mux B	3	2

Src Type = Producer instruction opcode type
Action = Mux setting; if no match, then Mux selection is 0
Priority = Which detection condition takes precedence (note, multiple can match)

(26)

Data Hazard Even with Forwarding



Cannot travel back in time.
Need to stall (interlock) the pipe if:

- The instruction in ID/EX is a lw
- The instruction in IF/ID uses register Rs1 and/or Rs2
- Rd for the instruction in ID/EX = Rs1 or Rs2

How can we interlock (stall) the pipeline?

(27)

Interlock on Load-Use

- **Detect a load followed by a dependent use**

Src Type	Dst Type	Condition
Ld	Register-Register	ID/EX.rt == IF/ID.rs
Ld	Register-Register	ID/EX.rt == IF/ID.rt
Ld	Immediate	ID/EX.rt == IF/ID.rs

- **Insert the bubble on detection**
  - Disable write (no updates): PC, ID/IF register
  - Clear the ID/EX register (inserting a nop)

(28)

Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:

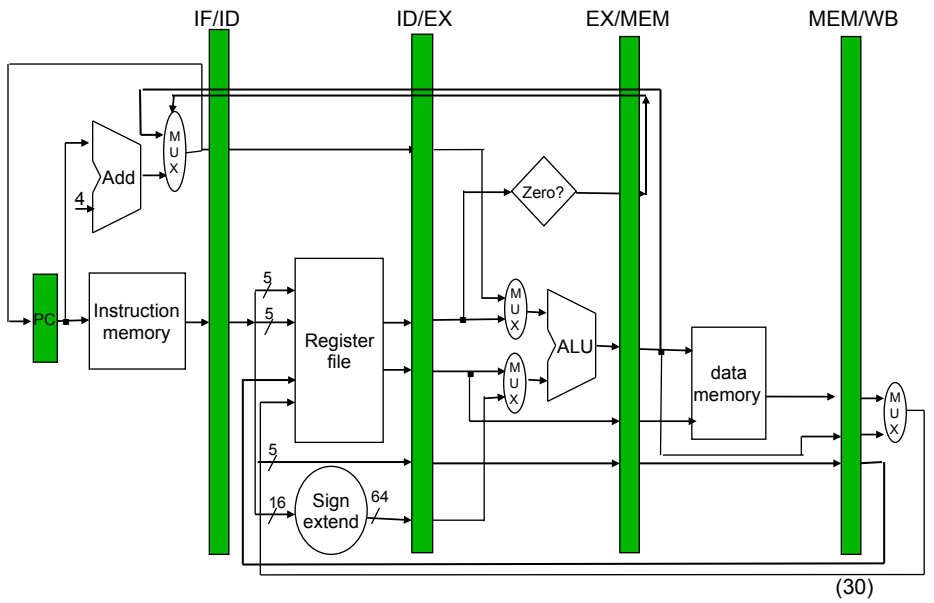
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```

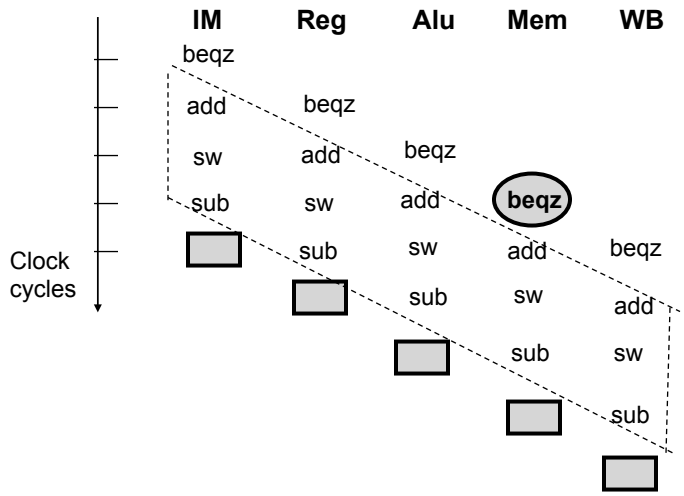
(29)

Control Hazard on Branches



(30)

Control Hazard on Branches



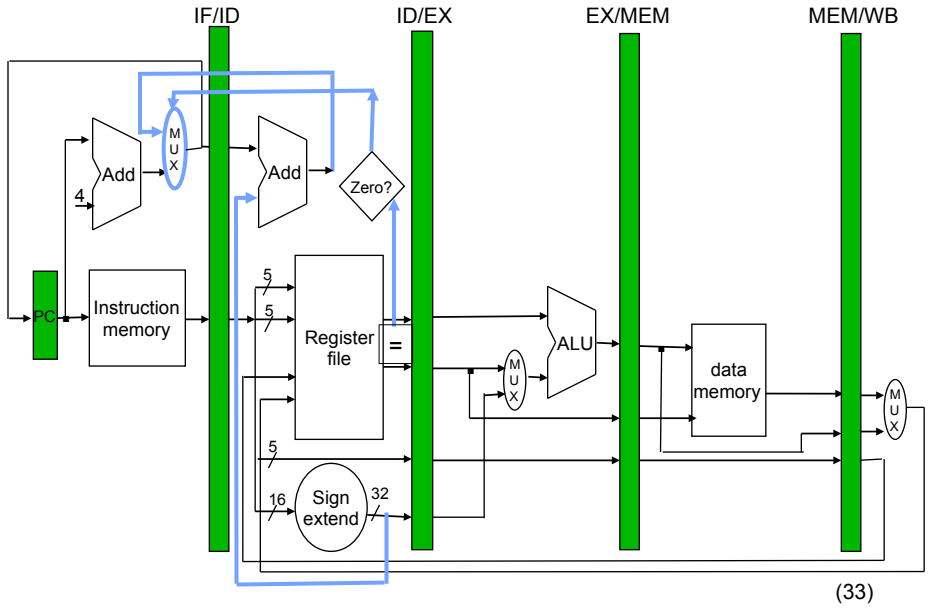
(31)

Branch Stall Impact

- If CPI = 1, 10% branch, Stall 3 cycles => new CPI = 1.3
- Two part solution:
  1. Determine branch taken or not sooner, AND
  2. Compute taken branch address earlier
- MIPS branch tests if two registers are equal
- MIPS solution
  - Move Zero test to ID/EX stage
  - Adder to calculate branch target address in ID/EX stage
  - 1 clock cycle penalty for branch versus 3

(32)

Early determination of Branch condition and target



(33)

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken (how?)
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken

- 53% MIPS branches taken on average
- But haven't calculated branch target address in MIPS
  - » MIPS still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

(34)

Four Branch Hazard Alternatives

#4: Delayed Branch

- Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay of length *n*

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- Delay Slot Instruction (DSI)
- MIPS uses this

(35)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R2,R2,4
     addi R5,R5,-1
     bne  R5,R0,top
     delay slot instruction
     add
     ...
```

(36)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R2,R2,4
     addi R5,R5,-1
     bne  R5,R0,top
     nop
     add
     ...
```

Dependence chain

(37)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R2,R2,4
     addi R5,R5,-1
     bne  R5,R0,top
     nop
     add
     ...
```

Dependence chain

(38)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R2,R2,4
     addi R5,R5,-1
     bne  R5,R0,top
     nop
     add
     ...
```

Dependence chain

(39)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R5,R5,-1
     bne  R5,R0,top
     addi R2,R2,4
     add
     ...
```

Dependence chain

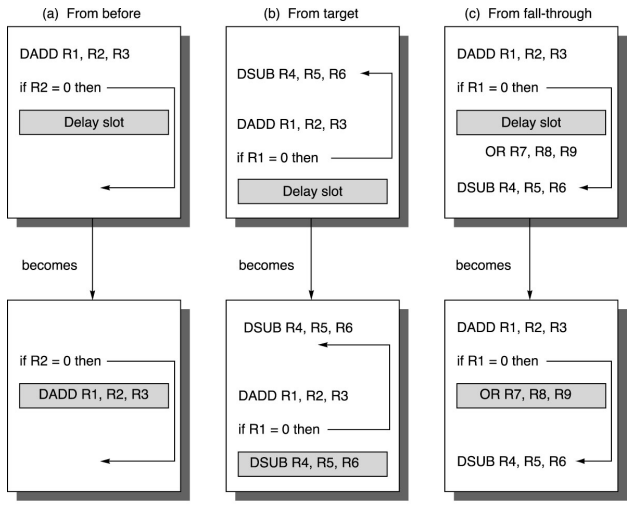
(40)

Example of DSI

```
top: lw   R1,0(R2)
     lw   R3,4(R2)
     add  R1,R1,R3
     sw   R1,0(R2)
     addi R2,R2,-4
     bne  R2,R0,top
     nop
     add
     ...
```

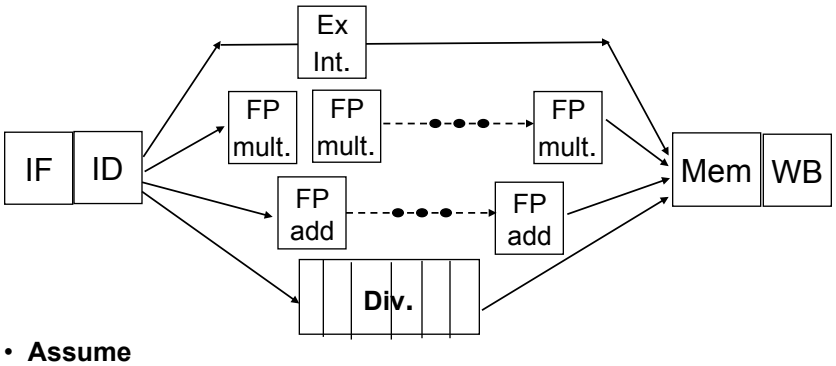
Can we move the addi?

(41)



- Where do we get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: should repeat instruction for correctness
  - From fall through: correct if R7 is dead after the branch

Multi-cycle pipelines



- **Assume**
  - 4-stage, pipelined, FP add
  - 7-stage, pipelined, FP multiply
  - 25 stage, non-pipelined divide unit
- **Latency** of an instruction, *I*, in a pipeline, *P*, is the number of bubbles that has to exist in *P* if the instruction following *I* wants to use the result of *I*.

(44)

Hazards:

- Two divide instructions will stall the pipe (structural hazards).
- May have more than one register write in one cycle (why?)
  - increase number of ports, or stall the pipeline (interlock)
- May have WAW hazard (why?)
- Out-of-order completion causes problems with exceptions,
- The long pipes causes more RAW hazards (why?)

To deal with structural Hazards:

- Stall a conflicting instruction at the ID stage
  - Use a shift register to keep track of the utilization of a stage that may suffer from structural hazard (ex. Input ports of registers)
- Stall a conflicting instruction when entering the Mem stage
  - may give priority to longer instructions to reduce RAW hazards.

(45)

Examples

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4, 0(R2)	IF	ID	EX	MEM	WB												
Mul.D F0, F4, F6		IF	ID	-	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
Add.D F2, F0, F8			IF	-	ID	-	-	-	-	-	-	A1	A2	A3	A4	MEM	WB
S.D F2, 0(R2)					IF	-	-	-	-	-	-	ID	EX	-	-	-	MEM

Stall due to RAW hazards

instruction	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	Mem	WB
…		IF	ID	EX	Mem	WB					
…			IF	ID	EX	Mem	WB				
Add.D F2,F4,F6				IF	ID	A1	A2	A3	A4	Mem	WB
…					IF	ID	EX	Mem	WB		
…						IF	ID	EX	Mem	WB	
L.D F2, 0(R2)							IF	ID	Ex	Mem	WB

Structural hazards

(46)

To deal with WAW:

- WAW occurs only if a useless instruction is executed
  - If there is a use in between the writes, then a RAW will stall the pipe (if no forwarding is used -- in this case forwarding is not possible)
- May detect hazard and hold the second instruction,
- May detect hazard and prevent the first instruction from writing
- May detect hazard and replace the first instruction with a no-op

Can all hazards be detected at the ID stage?

To maintain precise exceptions:

- May ignore the problem, or
- buffer the results and enforce the order of writes, or
- let the trap handling routine enforce the preciseness (software approach), or
- delay the issue (stall the pipe) to enforce in-order completion.

(47)

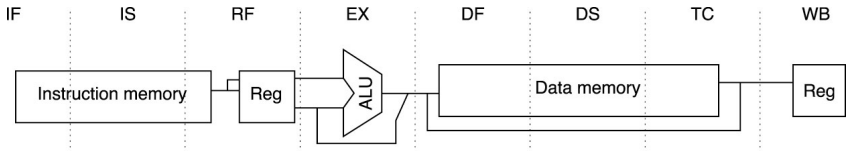
Instruction set design and pipelining

- Variable instruction length and execution time leads to
  - imbalance among stages,
  - complicate hazard detection and precise exceptions
- Caches have similar effects (imbalance pipes)
  - may freeze the entire pipeline on a cache miss
- Complex addressing modes
  - may change register values
  - may require multiple memory access
- self modifying instructions causes pipeline problems
- Implicitly set condition codes complicates pipeline control hazards

(48)

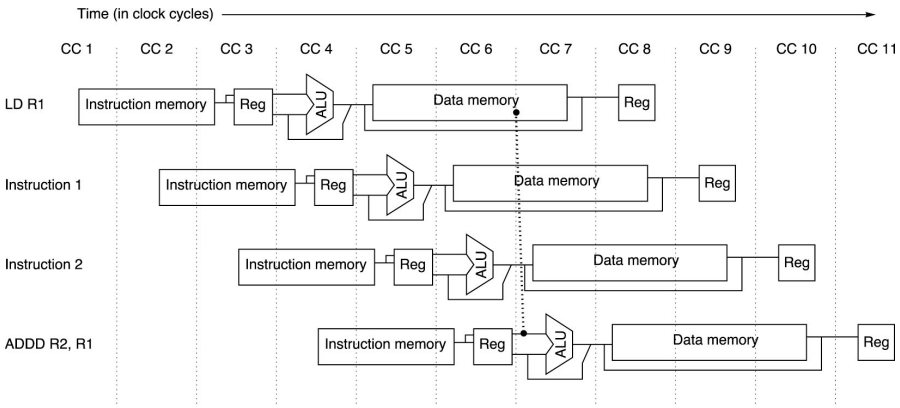
The MIPS R4000

- 64-bit instruction set (MIPS-3 ISA)
- Decomposes memory access into stages (super-pipelining)



- Uses a deeper pipeline
  - IF -- first half of instruction fetch
  - IS -- second half of instruction fetch
  - RF - instruction decode and register fetch (and check cache tag)
  - EX - execution: effective address calculation, ALU operation, branch target computation, branch condition evaluation
  - DF - first half of data fetch
  - DS - second half of data fetch
  - TC - check cache tag (to determine if it is a hit)
  - WB write back

(49)



Forwarding from memory output is required to instructions that are 3 or 4 cycles later (sooner instructions have to stall)

(50)

The MIPS R4000 pipeline

- Branch delay is 3 cycles
- The architecture supports a single-cycle delayed branch

The floating-point pipeline:

- Three units: adder (5 stages), divider(36 non-pipelined stages) and multiplier (9 stages)

- Eight hardware units are shared among stages
  - A (add mantissa),
  - D (divide),
  - E(exception test),
  - M and N (first and second stages of multiplier),
  - R (rounding),
  - S (shift),
  - U (unpack)

(51)