EE 459/500 – HDL Based Digital Design with Programmable Logic

Lecture 19
Pipeline Design

References:

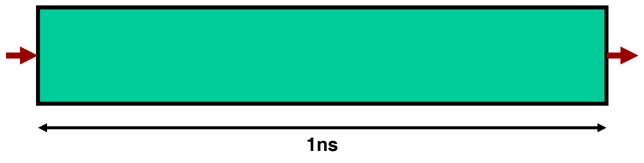
Chapter 11 of M. Morris Mano and Charles Kime, Logic and Computer Design Fundamentals, Pearson Prentice Hall, 4th Edition, 2008.

Overview

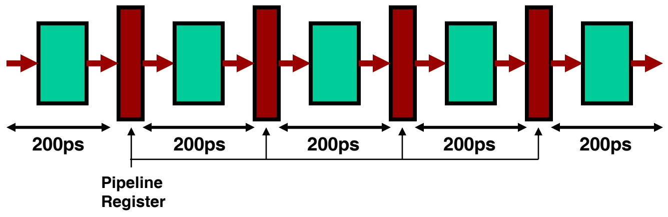
- Pipelining concept
- Pipelined design of Simple Computer
  - Basic 5-stage pipe
    - Speedup of pipelined vs non-pipelined implementations
  - Pipeline hazards
    - Structural, data, control
- Parallel digital systems

Pipelining a digital system

- Key idea: break big computation into pieces



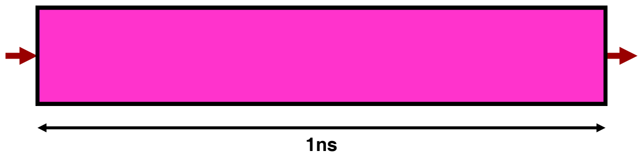
- Separate each piece with a pipeline register



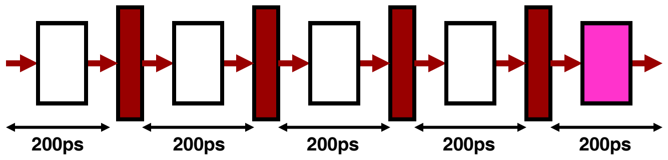
3

Pipelining a digital system

- Why do this? Because it's faster for repeated computations



Non-pipelined:
1 operation finishes every 1ns



Pipelined:
1 operation finishes every 200ps

4

Pipelining

- Pipelining increases **throughput**, but not total computation time of a task
  - Answer available every 200ps, BUT
  - A single computation still takes 1ns
- Limitations:
  - Computations must be divisible into stage size
  - Pipeline registers add overhead

5

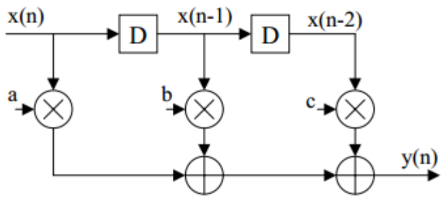
Pipelining

- **Pipelining transformation** leads to a reduction in the critical path, which can be exploited to increase the clock speed or to reduce power consumption at same speed.
- In parallel processing, multiple outputs are computed in parallel in a clock period. Therefore, the **effective clock speed** is increased by the level of parallelism.

6

Example: 3-tap FIR digital filter

Example 1: Consider a 3-tap FIR filter: y(n)=ax(n)+bx(n-1)+cx(n-2)



T_M : *multiplication – time*

T_A : *Addition – time*

– The critical path (or the minimum time required for processing a new sample) is limited by 1 multiply and 2 add times. Thus, the "sample period" (or the "sample frequency") is given by:

$$T_{sample} \geq T_M + 2T_A$$

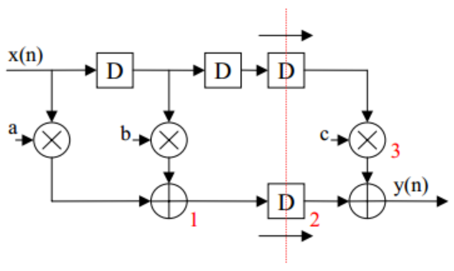
$$f_{sample} \leq \frac{1}{T_M + 2T_A}$$

7

Example: 3-tap FIR digital filter

- The pipelined implementation: By introducing 2 additional latches in Example 1, the critical path is reduced from T_M+2T_A to T_M+T_A.The schedule of events for this pipelined system is shown in the following table. You can see that, at any time, 2 consecutive outputs are computed in an interleaved manner.

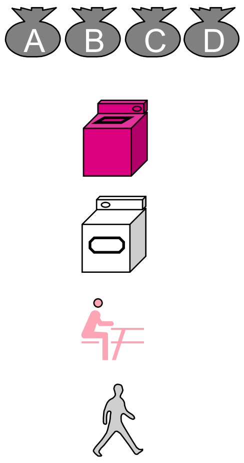
Clock	Input	Node 1	Node 2	Node 3	Output
0	x(0)	ax(0)+bx(-1)	—	—	—
1	x(1)	ax(1)+bx(0)	ax(0)+bx(-1)	cx(-2)	y(0)
2	x(2)	ax(2)+bx(1)	ax(1)+bx(0)	cx(-1)	y(1)
3	x(3)	ax(3)+bx(2)	ax(2)+bx(1)	cx(0)	y(2)



8

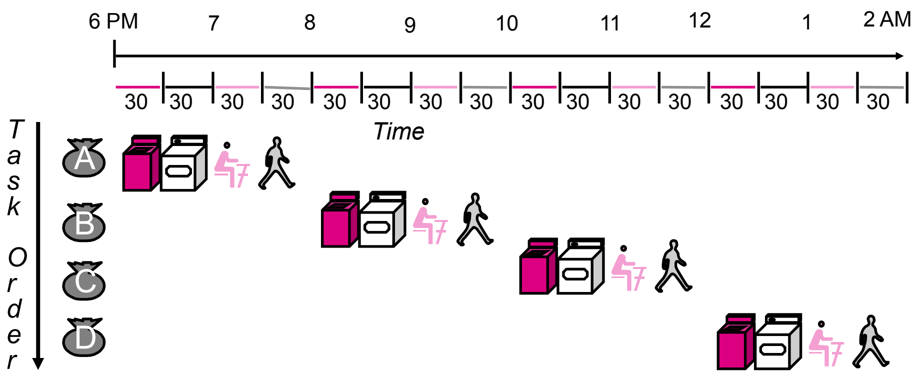
The Laundry Analogy

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 30 minutes
- "Folder" takes 30 minutes
- "Stasher" takes 30 minutes to put clothes into drawers



9

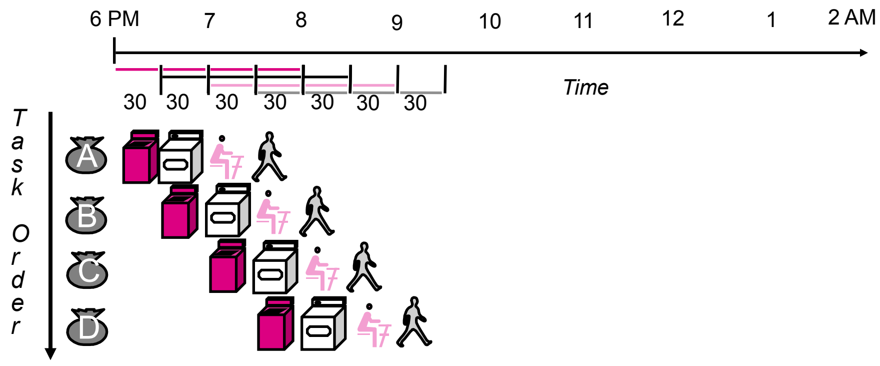
If we do laundry sequentially...



- Time Required: 8 hours for 4 loads

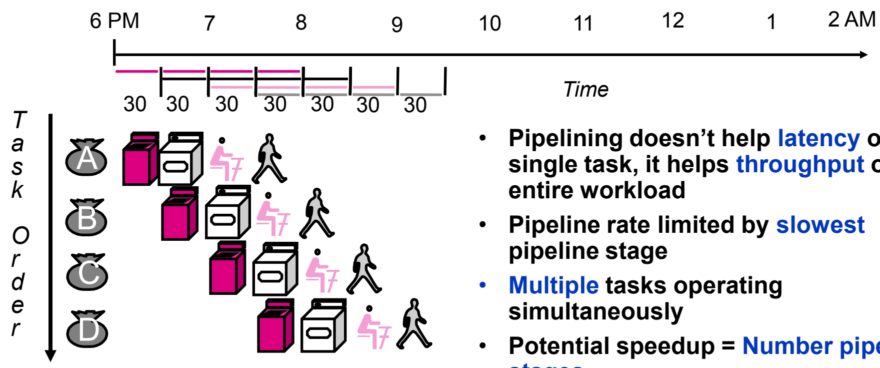
10

To Pipeline, We Overlap Tasks



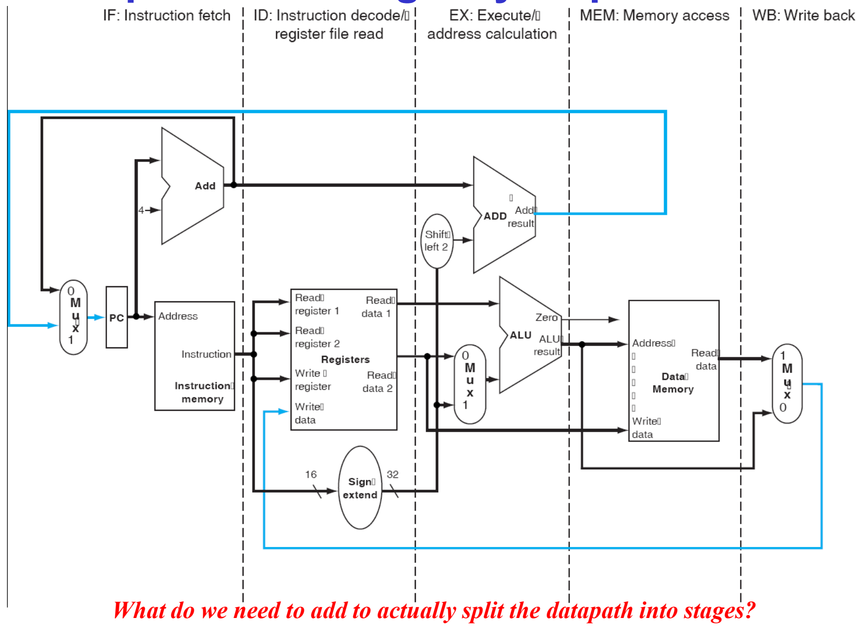
- Time Required: 3.5 Hours for 4 Loads

To Pipeline, We Overlap Tasks



- **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**

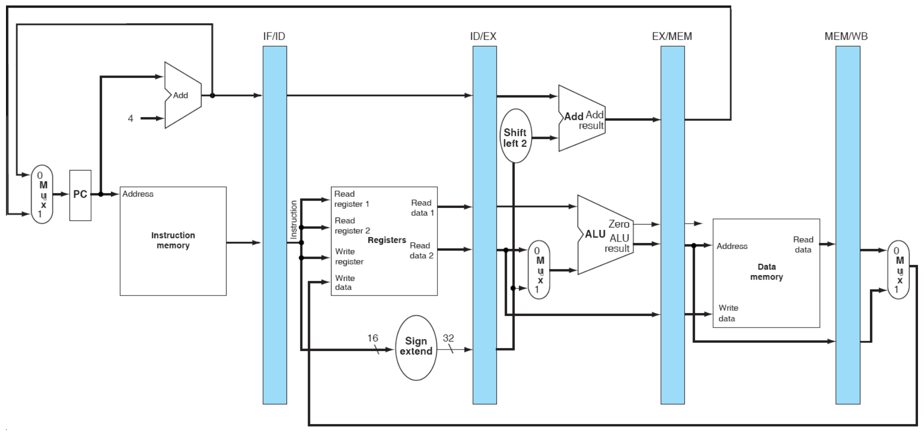
Example: basic single-cycle processor



What do we need to add to actually split the datapath into stages?

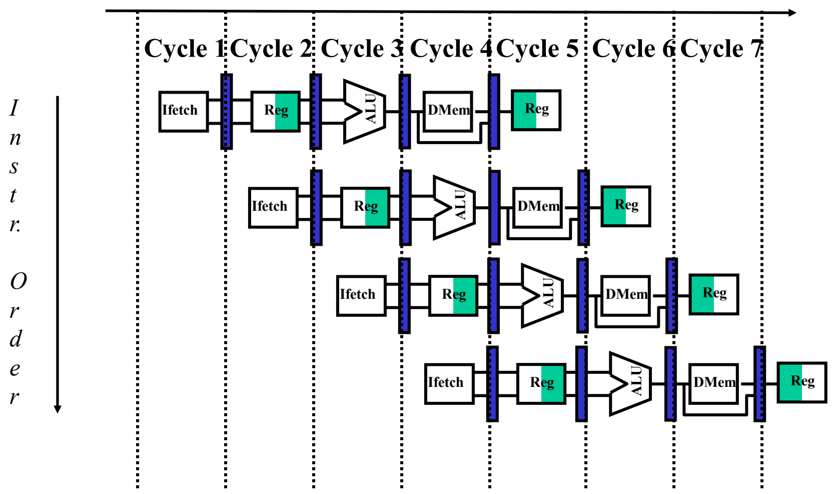
13

Example: basic pipelined processor



14

Example: The Basic Pipeline For MIPS



15

Overview

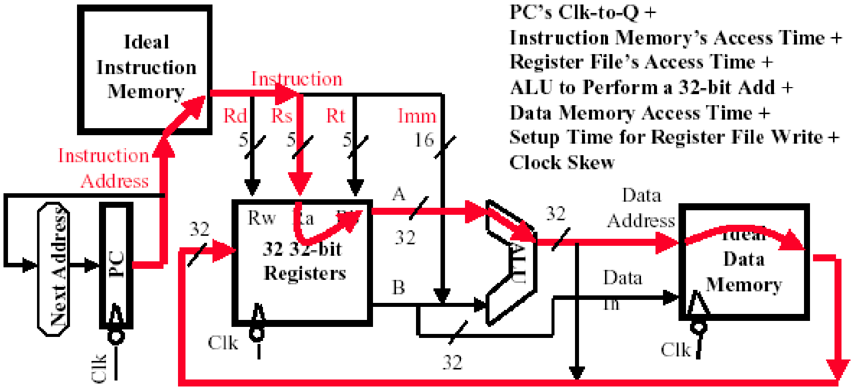
- Pipelining concept
- Pipelined design of Simple Computer
  - Basic 5-stage pipe
    - Speedup of pipelined vs. non-pipelined implementations
  - Pipeline hazards
    - Structural, data, control
- Parallel digital systems

16

Abstract View of Critical Path

- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."

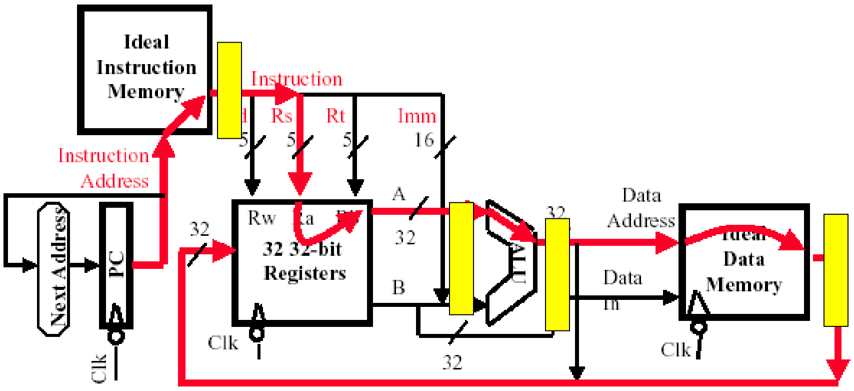
Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew



17

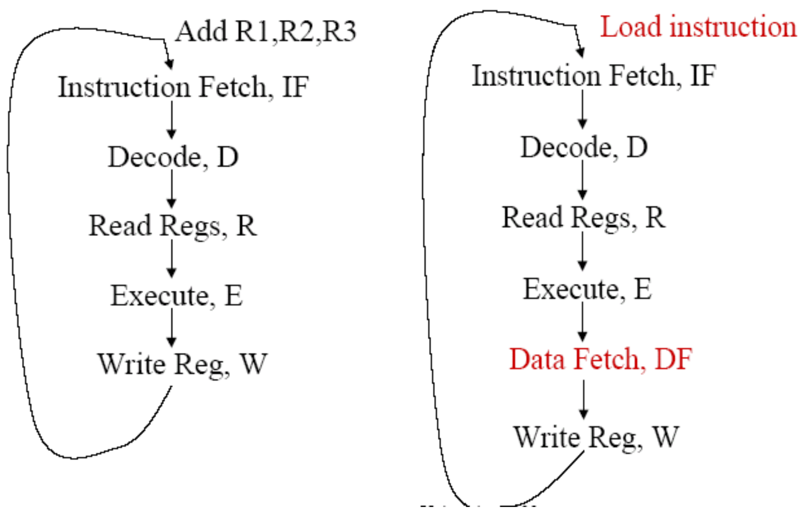
Pipelined critical path

- Critical path is longest path between stage registers



18

Steps in Instruction Processing



Un-pipelined (Non-overlapped) Implementation

- Consider loads with DF stage

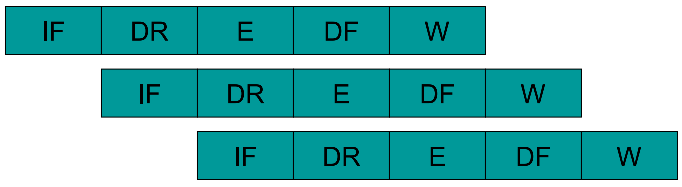
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
I1	IF	DR	E	DF	W					
I2						IF	DR	E	DF	W
I3										
…										

Pipelined Implementation

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
I1	IF	DR	E	DF	W					
I2		IF	DR	E	DF	W				
I3			IF	DR	E	DF	W			
I4				IF	DR	E	DF	W		
I5					IF	DR	E	DF	W	
I6						IF	DR	E	DF	W

5-stage Pipeline

- CPU stages
  - IF: Instruction fetch
  - DR: Instruction decode & Register read
  - E: Execute
  - DF: Data fetch (Memory load/store)
  - W: Write Back Registers
- Another set of mnemonic names
  - IF, ID, E, MEM, WB

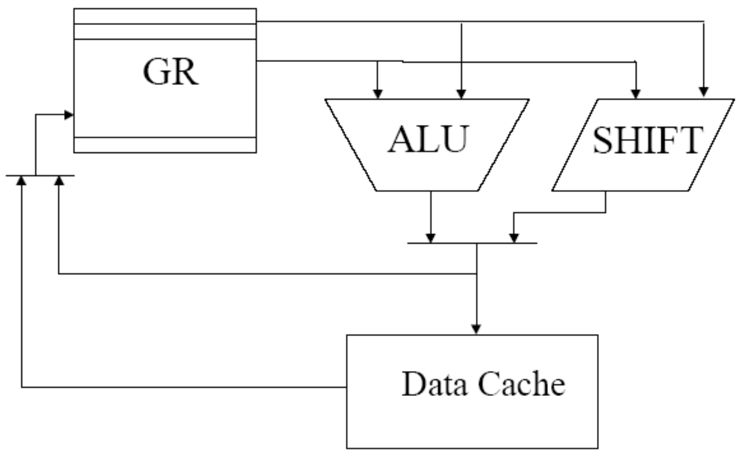


Computer Pipelines

- Execute billions of instruction, so **throughput** is what matters
- Throughput versus latency
  - + Throughput increases
  - - Latency for a single instruction increases
    - May have to wait longer for single instruction to complete
- Allows much faster clock cycle
- RISC pipeline architecture features:
  - All instructions same length
  - Registers located in same place in instruction format
  - Memory operands only in loads and stores

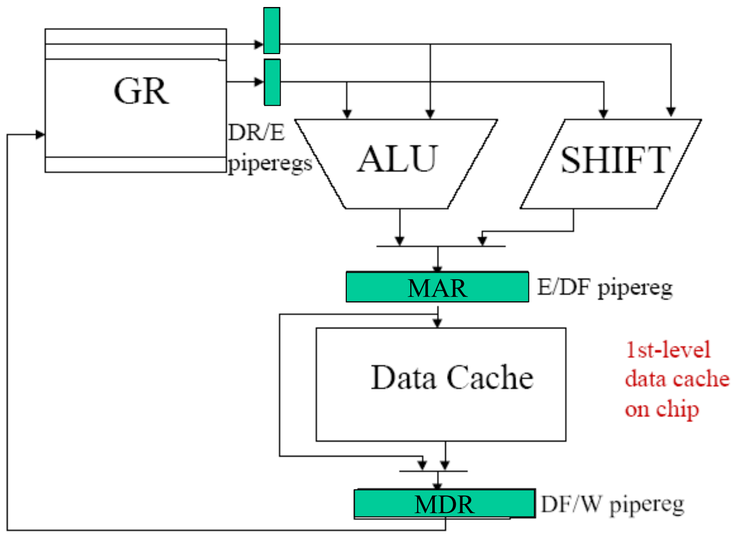
23

Unpipelined Datapath



24

Pipelined Datapath



25

Overview

- Pipelining concept
- Pipelined design of Simple Computer
  - Basic 5-stage pipe
    - Speedup of pipelined vs. non-pipelined implementations
  - Pipeline hazards
    - Structural, data, control
- Parallel digital systems

26

Pipelining Hazards

- Hazards cause the pipe to stall because of some conflict in the pipe (prevents the next instruction in pipe from executing in its turn)
- Types of hazards
  - Structural: contention for same hardware resource
  - Data: dependency on earlier instruction for the correct sequencing of register reads and writes
  - Control: branch/jump instructions stall the pipe until get correct target address into PC

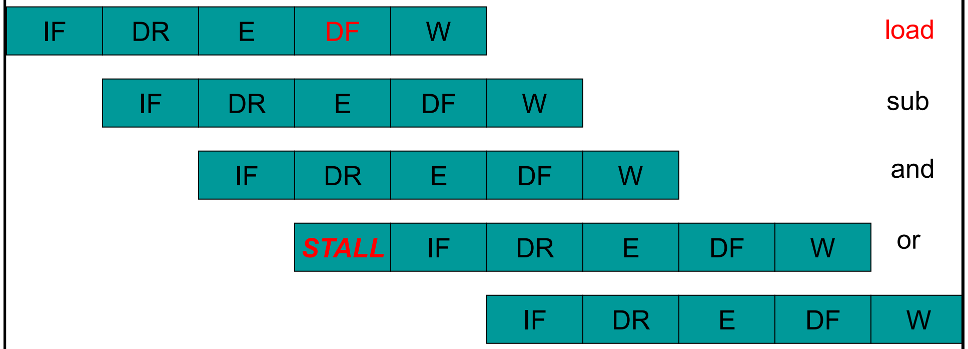
27

Structural Hazards

- Resource conflicts in the pipeline
- Examples
  - Single memory port shared for instruction and data access
  - Register file without a separate write port

28

Structural Hazards



29

Structural Hazards

- IF and DF compete for single memory port
- Ideal Machine
  - No stalls, 1 cycle per instruction
- Assume 30% of instructions access data
  - With structural hazard, 1.3 cycles per instruction
  - Performance has gone down by 30%
- Solutions:
  - Pipeline stall (insert bubble)
  - Have 2 memory ports for shared instruction-data cache-memory (expensive)
  - Have separate instruction cache-memory and data cache-memory

30

Three Generic Data Hazards (I) - RAW

- Instr$_1$ followed by Instr$_2$

```
add r1, r3, r2
add r4, r5, r1
```

- Instr$_2$ tries to read operand before Instr$_1$writes it
  - Can be due to true "data dependency" (data must be produced before it can be consumed)
  - Or can be due to pipeline staging (data already produced, but not yet written to general register file

31

Data Hazards (II) - WAR

- Instr$_1$ followed by Instr$_2$

```
ld  r1, (r3)+
add r3, r4, r1
```

- Instr$_2$ tries to write operand before Instr$_1$ reads it
  - Instr$_1$ gets wrong operand
  - Can't happen in the 5-stage RISC pipeline we just covered
    - All instruction take 5 stages
    - Reads are always in stage 2
    - Writes are always in stage 5

32

Data Hazards (III) - WAW

- Instr$_1$ followed by Instr$_2$

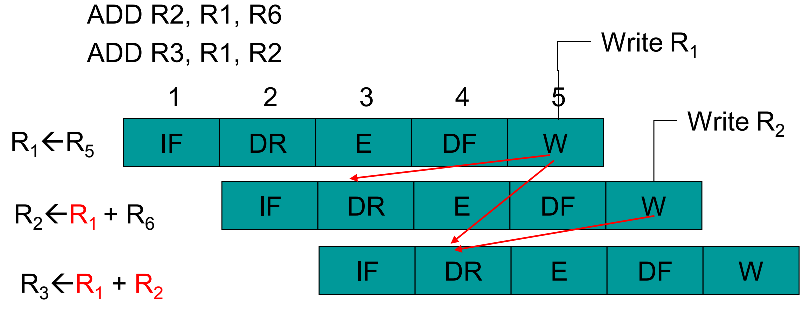
```
mul r1, r0, r2
add r1, r5, r6
```

- Instr$_2$ tries to write operand before Instr$_1$ writes it
  - Leaves wrong result (Instr$_1$, not Instr$_2$)
  - Can't happen in our 5-stage pipeline because
    - All instructions take 5 stages
    - Writes are always in stage 5

Data Hazards

- Overlapping instructions cause dependencies on data (RAW)

e.g., MOVA R1, R5
ADD R2, R1, R6
ADD R3, R1, R2



Data Hazards Remedy - SW

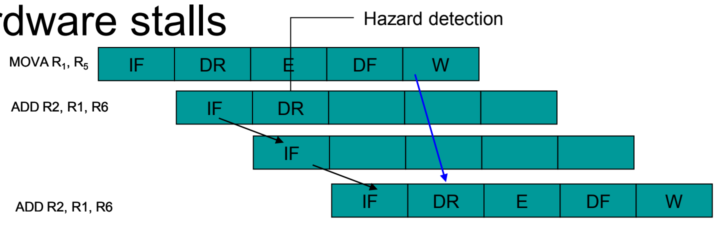
- Software delay (compiler or machine code programming to insert NOPs)

```
MOVA R1, R5
NOP
NOP
ADD R2, R1, R6
NOP
NOP
ADD R3, R1, R2
```

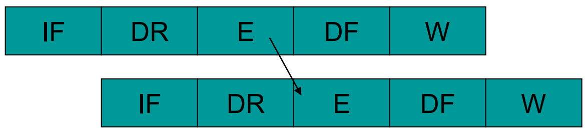
35

Data Hazards Remedy - HW

- Hardware stalls



- Hardware Data Forwarding
  - Add an extra path connecting ALU outputs to ALU inputs on the next clock



36

Data Forwarding (Reg. Bypassing)

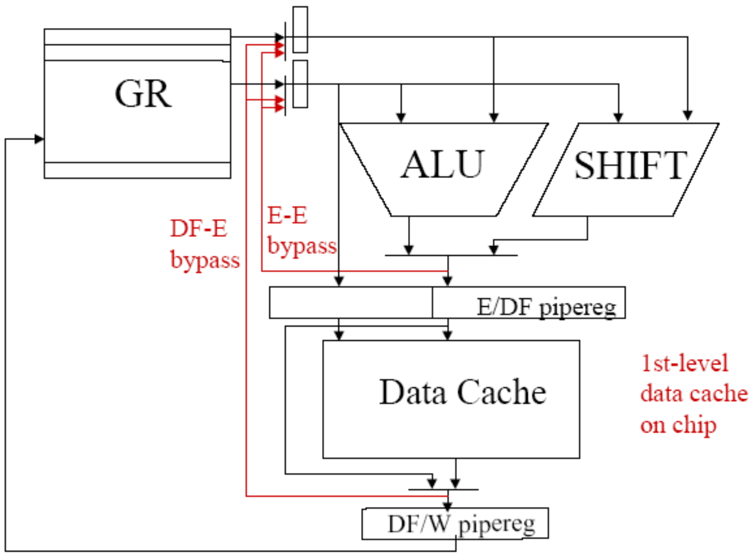
```
add r1,r2,r3    |IF|DR|E |DF|W |
                   E-E byp
sub r4,r1,r3       |IF|DR|E |DF|W |
                     DF-E byp
and r6,r1,r7          |IF|DR|E |DF|W |
                                *
or   r8,r1,r9            |IF|DR|E |DF|W |

xor r10,r1,r11              |IF|DR|E |DF|W |
```

* Write regs in 1st. half of cycle, Read regs in 2nd. half

37

Pipelined Datapath – with data forwarding



38

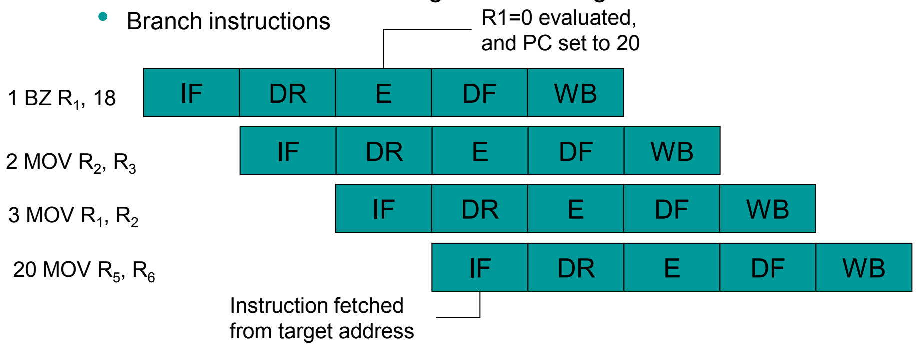
Control Hazards

- For branch or jump instruction, the correct instruction to execute is not known in time (at the start of the IF stage of the instruction after the Branch)
  - Condition not yet determined (for conditional branch instruction)
  - Target instruction address not yet calculated

39

Control Hazards

- Conflicts that arise from changes in the Program Counter
  - Branch instructions



- IF of 2 instructions following branch happens before you know whether or not to branch and where to branch
- ISA may allow code to run useful instructions
- Can use branch prediction to improve performance

40

Control Hazard Solutions

- Solution 1: stall
- The instructions after the branch are stalled, until the branch condition is checked and target address is generated

```
Add    |IF|DR|E |DF|W |
Beq       |IF|DR|E |DF|W |
and          |- |- |IF|DR|E |DF|W |
xor             |- |- |IF|DR|E |DF|W |
```

2 (stall) penalty cycles,

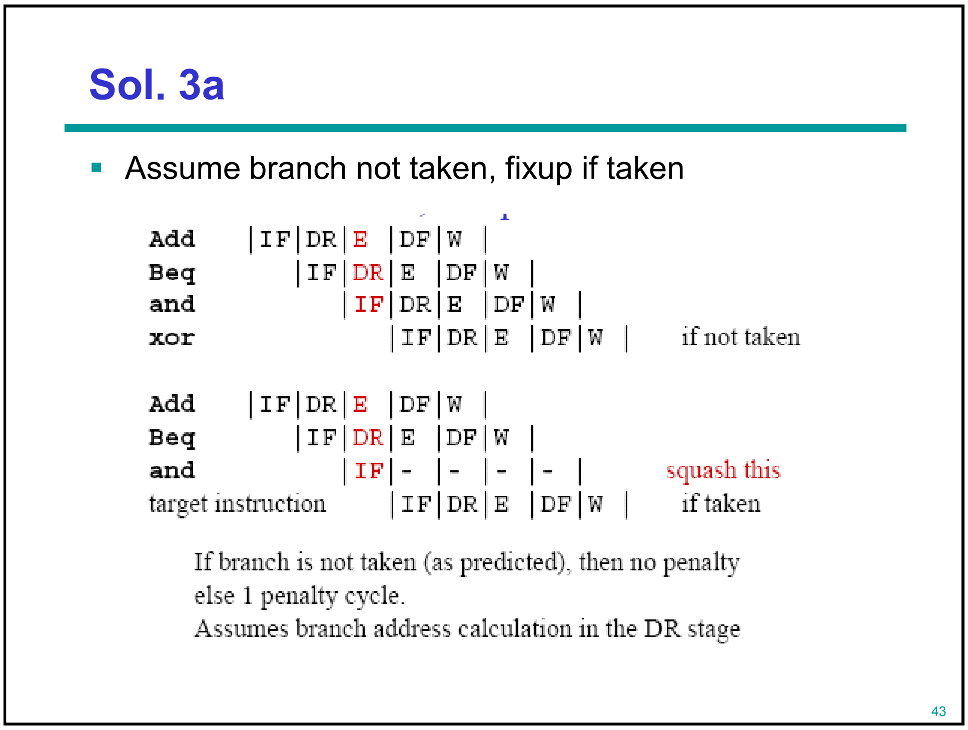
Sol 2:

- Perform target address calculation earlier in DR stage

```
Add    |IF|DR|E |DF|W |
Beq       |IF|DR|E |DF|W |
and          |- |IF|DR|E |DF|W |
xor             |- |IF|DR|E |DF|W |
```

Reduced penalty cycles from 2 to 1,

Need a separate branch adder in the DR stage to calculate PC+displacement



Sol. 3a

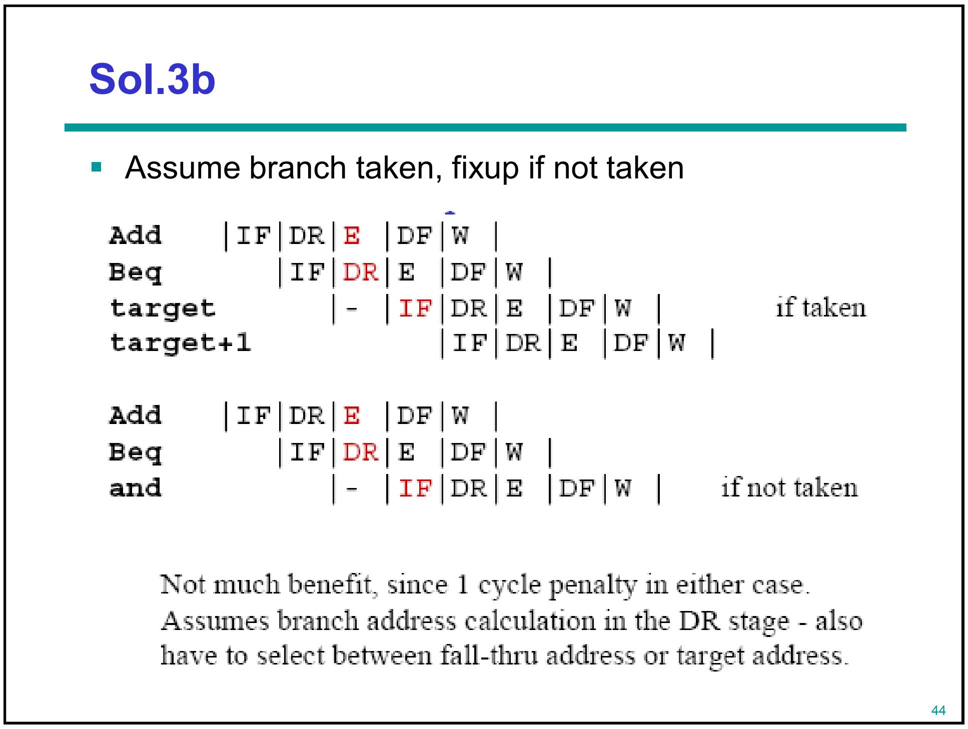
- Assume branch not taken, fixup if taken

```
Add     |IF|DR|E  |DF|W  |
Beq        |IF|DR|E  |DF|W  |
and           |IF|DR|E  |DF|W  |
xor              |IF|DR|E  |DF|W  |     if not taken

Add     |IF|DR|E  |DF|W  |
Beq        |IF|DR|E  |DF|W  |
and           |IF|-  |-  |-  |-  |      squash this
target instruction |IF|DR|E  |DF|W  |     if taken
```

If branch is not taken (as predicted), then no penalty
else 1 penalty cycle.
Assumes branch address calculation in the DR stage

43



Sol.3b

- Assume branch taken, fixup if not taken

```
Add       |IF|DR|E  |DF|W  |
Beq          |IF|DR|E  |DF|W  |
target          |-  |IF|DR|E  |DF|W  |        if taken
target+1               |IF|DR|E  |DF|W  |

Add       |IF|DR|E  |DF|W  |
Beq          |IF|DR|E  |DF|W  |
and             |-  |IF|DR|E  |DF|W  |     if not taken
```

Not much benefit, since 1 cycle penalty in either case.
Assumes branch address calculation in the DR stage - also
have to select between fall-thru address or target address.

44

Sol.4

- Delayed branch (ISA change)

```
Add            |IF|DR|E  |DF|W  |
Beq                |IF|DR|E  |DF|W  |
(delay slot)           |IF|DR|E  |DF|W  |
fallthru or target         |IF|DR|E  |DF|W  |
```

ISA specifies that the branch, if taken, only takes place after a delay of x instructions. (Above, x=1)

Branch adder in DR stage to calculate PC+displ, or PC+4+displ (depending on ISA definition)

Solutions to Control Hazards

- Micro-architecture solution
  - Stall the pipes
  - Calculate target address and condition earlier in pipeline (DR)
  - Assume branch always goes untaken (taken) and fix up pipe if it is actually taken (untaken)
- ISA solution
  - Have delay branches
    - Branch target takes place after n (delay) instructions
    - For n penalty cycles, must have (n-1) stages between IF and the stage where target and condition are determined
  - Have instruction which separate target address calculation and condition generation from actual branching, so these can be executed earlier (e.g., IA-64)
- Branch prediction

Handling Hazards - Summary

- Avoid some hazards "by design"
  - Eliminate DF-IF structural hazard by having separate I-cache and D-cache
  - Eliminate WAR by always fetching operands earlier in pipe (DR)
  - Eliminate WAW by doing all Ws in order (last stage, static) – not always the best micro-architecture decisions though!
  - Delayed branch in ISA to reduce control hazard penalty
- Detect and resolve remaining ones
  - Stall or forward (if possible)

48

Pipelining Cautions

- Superpipelining can cause long latencies
  - A large number of pipeline stages
  - High frequency (GHz)
    - Clock limited by slowest stage in the pipe
- Long pipes can also cause more stalls
- Dependencies can be tolerated as long as there is work to overlap the dependency

49

Overview

- Pipelining concept
- Pipelined design of Simple Computer
  - Basic 5-stage pipe
    - Speedup of pipelined vs. non-pipelined implementations
  - Pipeline hazards
    - Structural, data, control
- Parallel digital systems

Parallelism in CPUs

- Instruction Level Parallelism
  - Superscalar
    - Multiple functional units in a CPU support multiple instructions fetch and issue simultaneously
  - VLIW: single instruction, but multiple executions

Pipelined CPUs

```
mult r1, r2, r3
add r4, r5, r6
```

3 cycle multiply execute stage

IF	ID	E	E	E	MEM	WB	
	IF	ID	***STALL***	***STALL***	E	MEM	WB

52

Superscalar CPUs

```
mult r1, r2, r3
add r4, r5, r6
```

IF	ID	E	E	E	MEM	WB
IF	ID	E	MEM	WB		

3 cycle multiply execute stage, with multiple ALUs

53

4-Way Superscalar

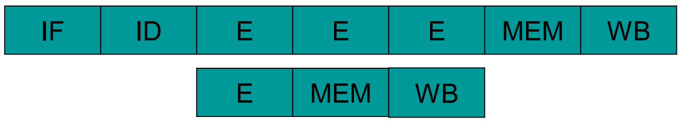
- Today's microprocessors are typically 4-way superscalar

Cycle	1	2	3	4	5	6	7	8
	IF	DR	E	DF	W			
	IF	DR	E	DF	W			
	IF	DR	E	DF	W			
	IF	DR	E	DF	W			
		IF	DR	E	DF	W		
		IF	DR	E	DF	W		
		IF	DR	E	DF	W		
		IF	DR	E	DF	W		
			IF	DR	E	DF	W	
			IF	DR	E	DF	W	
			IF	DR	E	DF	W	
			IF	DR	E	DF	W	
				IF	DR	E	DF	W
				IF	DR	E	DF	W
				IF	DR	E	DF	W
				IF	DR	E	DF	W

54

VLIW CPUs

```
mult r1, r2, r3; add r4, r5, r6
```



VLIW: very large instruction word
Single instruction fetched, multiple operations executed at the same time

55

4-way VLIW

Cycle	1	2	3	4	5	6	7	8
	IF	DR	E E E E	DF	W W W W			
		IF	DR	E E E E	DF	W W W W		
			IF	DR	E E E E	DF	W W W W	
				IF	DR	E E E E	DF	W W W W

Summary

- Pipelining is essential
- Parallel computing is the future