RISC Design:
Pipeline Hazards

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CP-226: Computer Architecture

Lecture 14 (13 March 2013)

CADSL

Branch Hazard

- Consider heuristic – branch not taken.
- Continue fetching instructions in sequence following the branch instructions.
- If branch is taken (indicated by *zero* output of ALU):
  - Control generates *branch* signal in ID cycle.
  - *branch* activates *PCSource* signal in the MEM cycle to load PC with new branch address.
  - *Three instructions in the pipeline must be flushed if branch is taken – can this penalty be reduced?*

Branch Not Taken

Branch to Z
A
B
C
D
Z

cycle b	cycle b+1	cycle b+2	cycle b+3	cycle b+4
Branch fetched	Branch decoded	Branch decision	PC keeps D (br. not taken)	
	A fetched	A decoded	A executed	A continues
		B fetched	B decoded	B executed
			C fetched	C decoded
				D fetched

Branch Taken

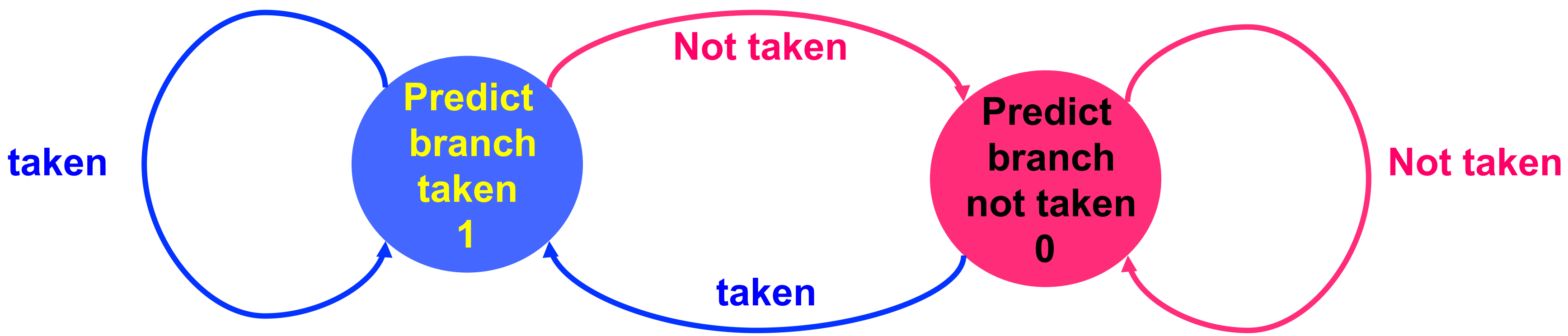
Branch to Z
A
B
C
D
Z

cycle b	cycle b+1	cycle b+2	cycle b+3	cycle b+4
Branch fetched	Branch decoded	Branch decision	PC gets Z (br. taken)	
	A fetched	A decoded	A executed	Nop
		B fetched	B decoded	Nop
			C fetched	Nop
				Z fetched

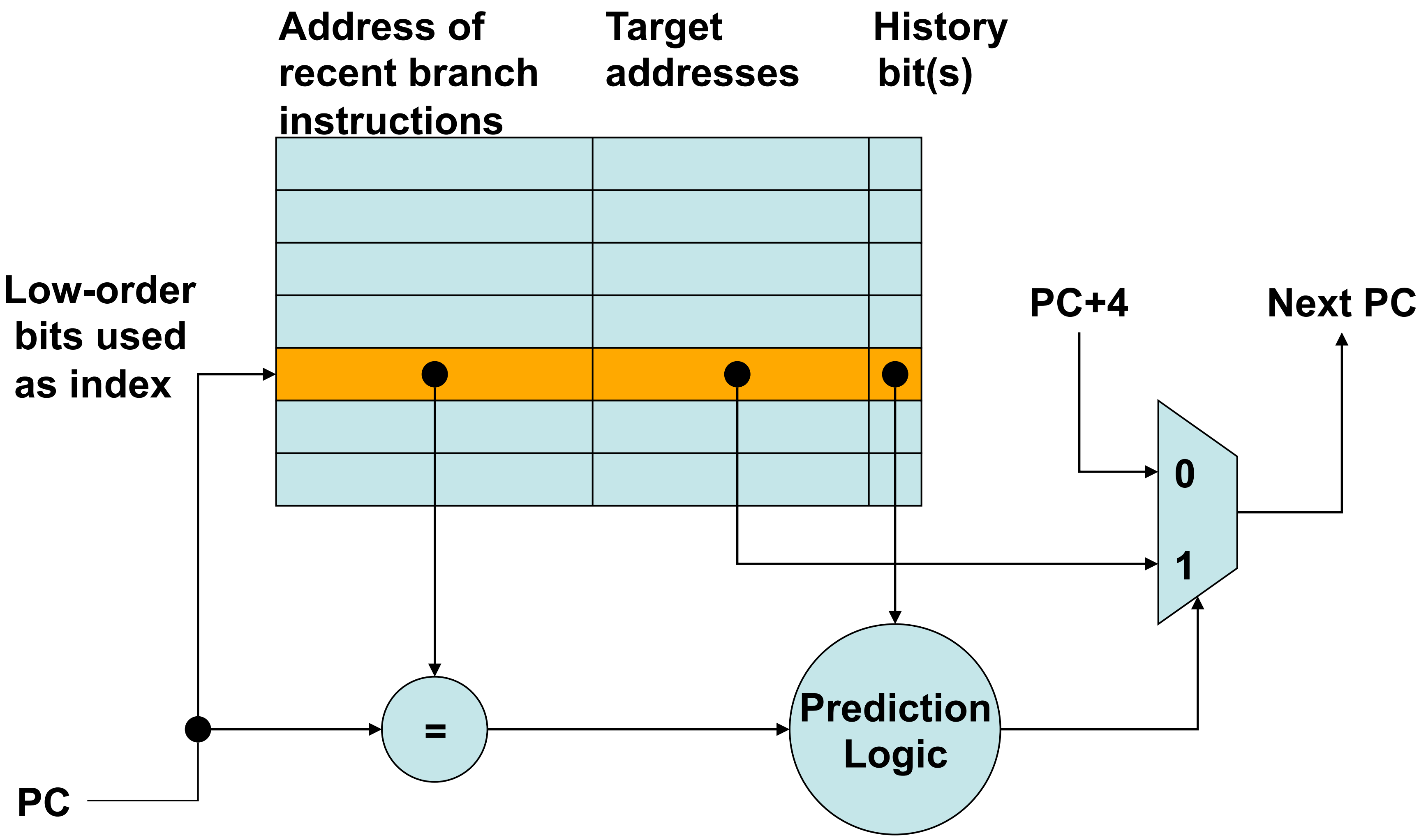
Three instructions are flushed if branch is taken

Branch Prediction

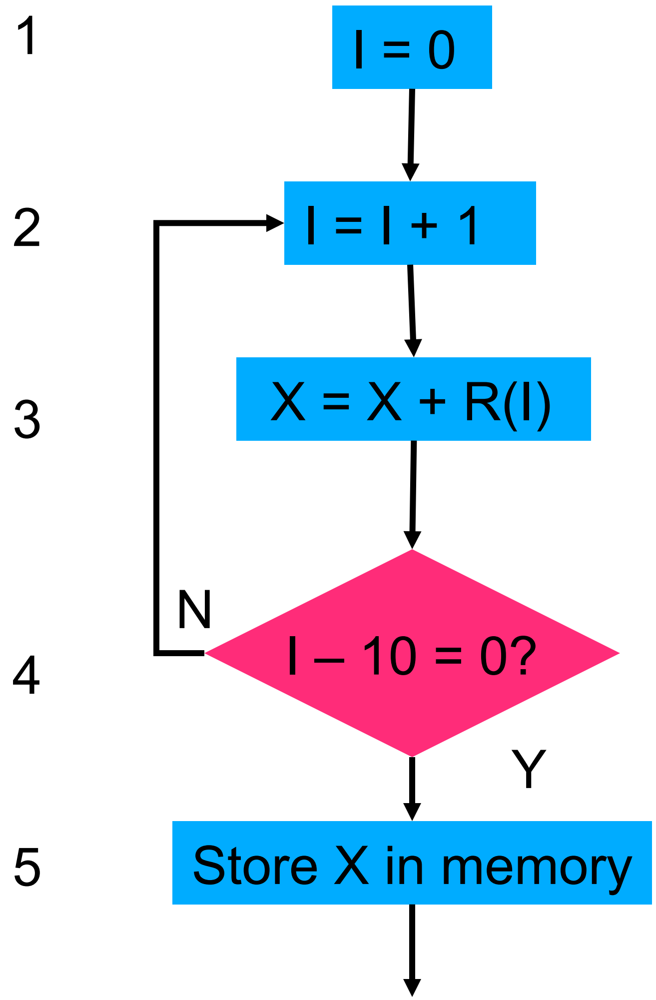
- Useful for program loops.
- A one-bit prediction scheme: a one-bit buffer carries a "history bit" that tells what happened on the last branch instruction
  - History bit = 1, branch was taken
  - History bit = 0, branch was not taken



Branch Prediction



Branch Prediction for a Loop



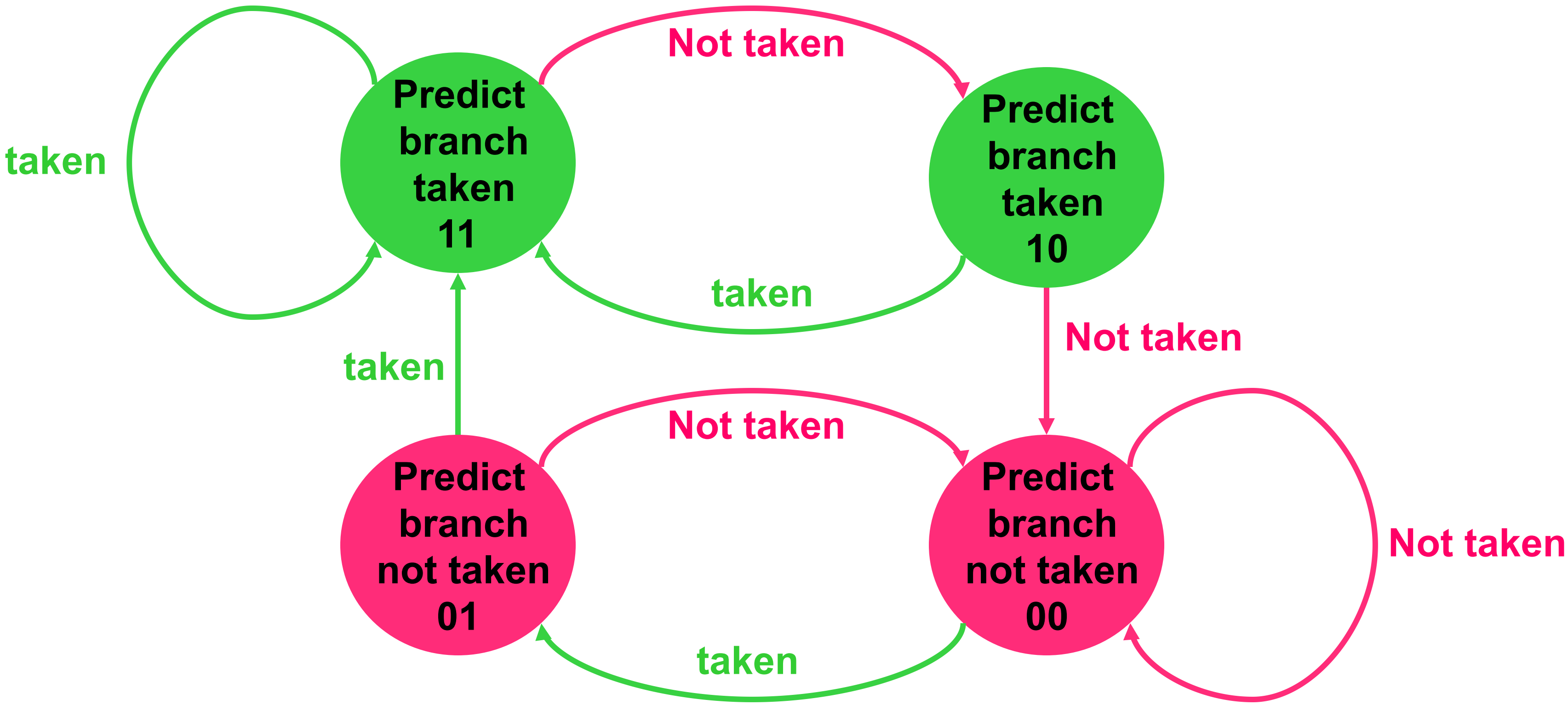
Execution of Instruction 4

Execu-tion seq.	Old hist. bit	Next instr.			New hist. bit	Prediction
		Pred.	I	Act.		
1	0	5	1	2	1	Bad
2	1	2	2	2	1	Good
3	1	2	3	2	1	Good
4	1	2	4	2	1	Good
5	1	2	5	2	1	Good
6	1	2	6	2	1	Good
7	1	2	7	2	1	Good
8	1	2	8	2	1	Good
9	1	2	9	2	1	Good
10	1	2	10	5	0	Bad

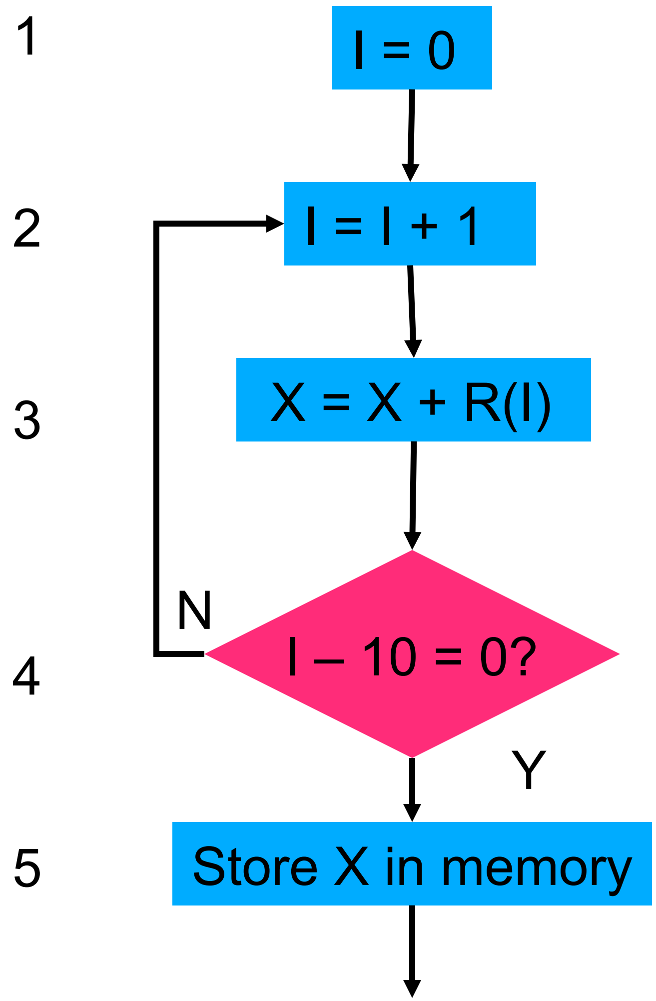
h.bit = 0 *branch not taken*, h.bit = 1 *branch taken*.

Two-Bit Prediction Buffer

- Can improve correct prediction statistics.



Branch Prediction for a Loop



Execution of Instruction 4

Execu-tion seq.	Old Pred. Buf	Next instr.			New pred. Buf	Prediction
		Pred.	I	Act.		
1	10	2	1	2	11	Good
2	11	2	2	2	11	Good
3	11	2	3	2	11	Good
4	11	2	4	2	11	Good
5	11	2	5	2	11	Good
6	11	2	6	2	11	Good
7	11	2	7	2	11	Good
8	11	2	8	2	11	Good
9	11	2	9	2	11	Good
10	11	2	10	5	10	Bad

Summary: Hazards

- Structural hazards
  - Cause: resource conflict
  - Remedies: (i) hardware resources, (ii) stall (bubble)
- Data hazards
  - Cause: data unavailablity
  - Remedies: (i) forwarding, (ii) stall (bubble), (iii) code reordering
- Control hazards
  - Cause: out-of-sequence execution (branch or jump)
  - Remedies: (i) stall (bubble), (ii) branch prediction/pipeline flush, (iii) delayed branch/pipeline flush

Limits of Pipelining

- IBM RISC Experience
  - Control and data dependences add 15%
  - Best case CPI of 1.15, IPC of 0.87
  - Deeper pipelines (higher frequency) magnify dependence penalties
- This analysis assumes 100% cache hit rates
  - Hit rates approach 100% for some programs
  - Many important programs have much worse hit rates

Processor Performance

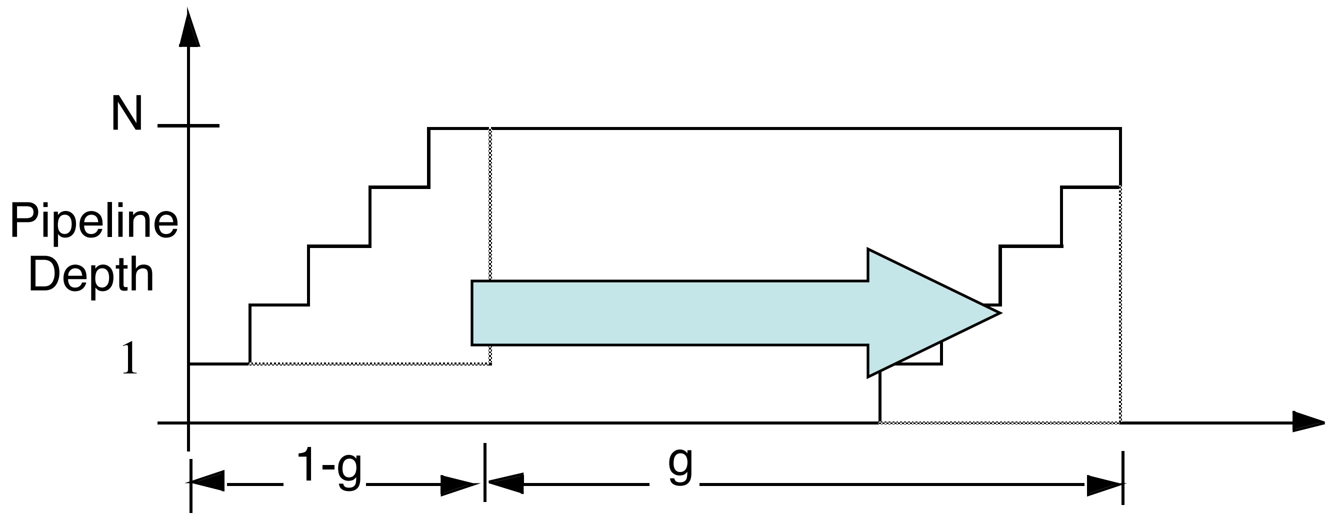
$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

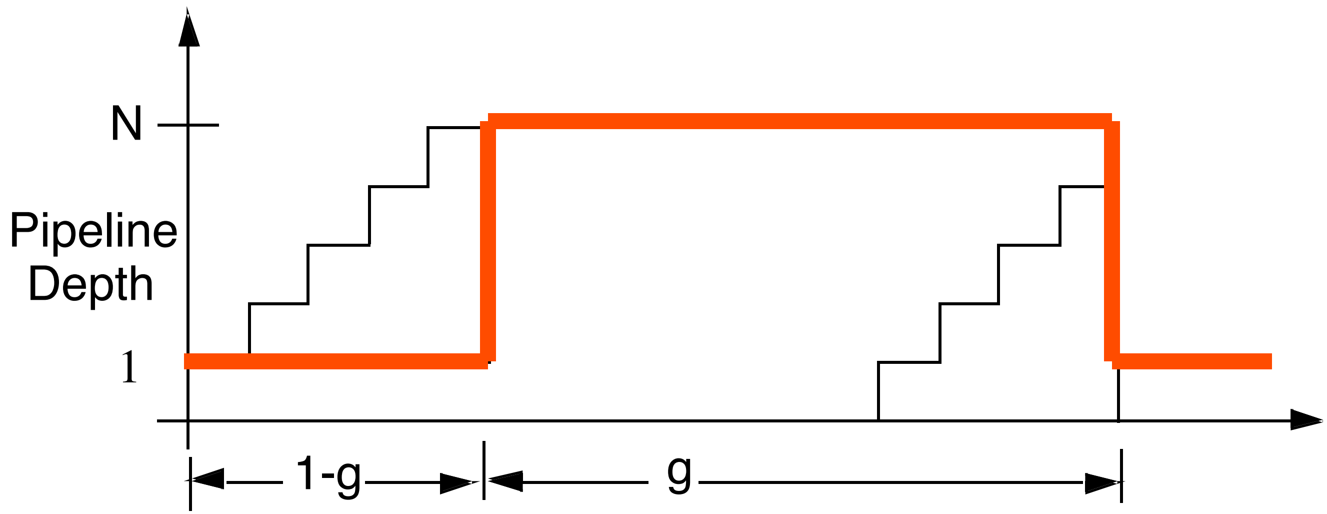
- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)
- In the 2000's (decade of multicore):
  - Marginal CPI improvement

Pipelined Performance Model



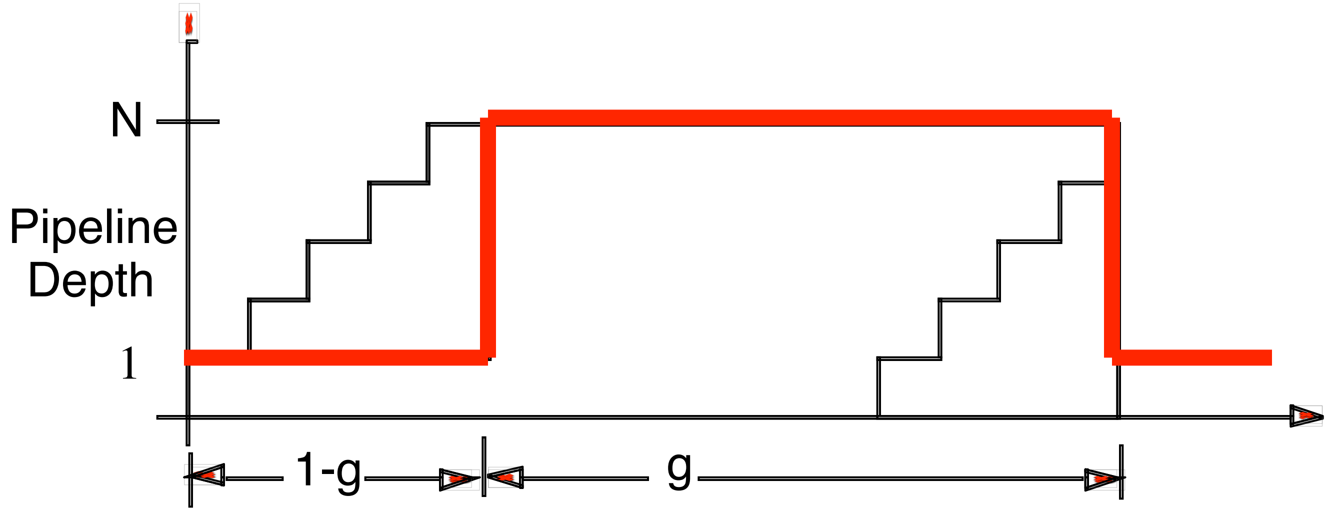
- g = fraction of time pipeline is filled
- 1-g = fraction of time pipeline is not filled (stalled)

Pipelined Performance Model



- g = fraction of time pipeline is filled
- 1-g = fraction of time pipeline is not filled (stalled)

Pipelined Performance Model



- Tyranny of Amdahl's Law [Bob Colwell]
  - When g is even slightly below 100%, a big performance hit will result
  - Stalled cycles are the key adversary and must be minimized as much as possible

Limits on Instruction Level Parallelism (ILP)

Weiss and Smith [1984]	1.58
Sohi and Vajapeyam [1987]	1.81
Tjaden and Flynn [1970]	1.86 (Flynn's bottleneck)
Tjaden and Flynn [1973]	1.96
Uht [1986]	2.00
Smith et al. [1989]	2.00
Jouppi and Wall [1988]	2.40
Johnson [1991]	2.50
Acosta et al. [1986]	2.79
Wedig [1982]	3.00
Butler et al. [1991]	5.8
Melvin and Patt [1991]	6
Wall [1991]	7 (Jouppi disagreed)
Kuck et al. [1972]	8
Riseman and Foster [1972]	51 (no control dependences)
Nicolau and Fisher [1984]	90 (Fisher's optimism)

Superscalar Proposal

- Go beyond single instruction pipeline, achieve IPC > 1
- Dispatch multiple instructions per cycle
- Provide more generally applicable form of concurrency (not just vectors)
- Geared for sequential code that is hard to parallelize otherwise
- Exploit fine-grained or instruction-level parallelism (ILP)

Thank You