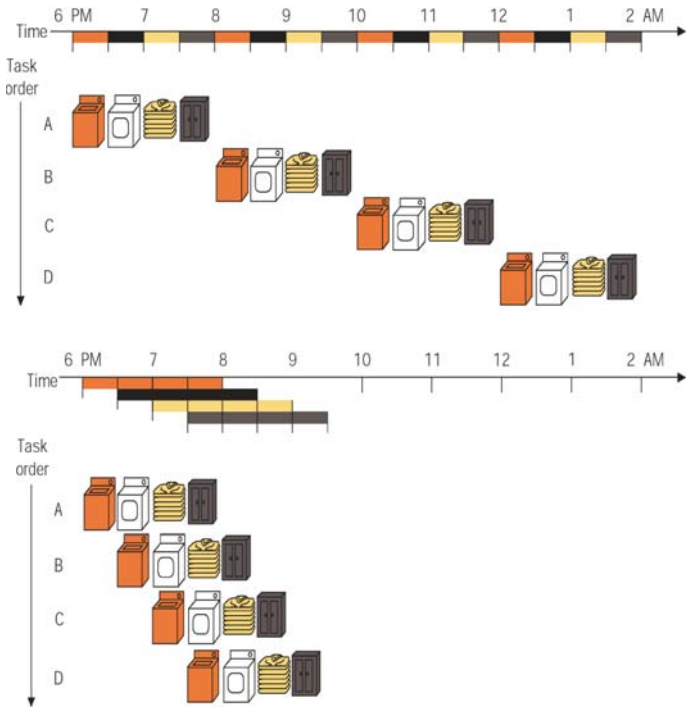
Chapter Four – II (1/5)

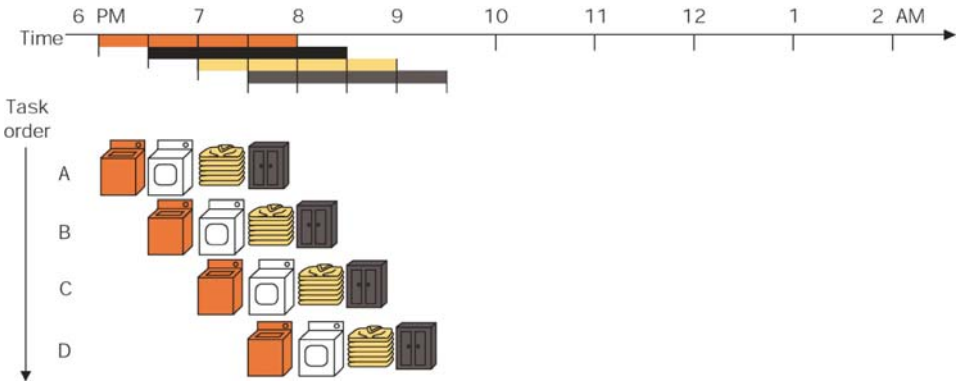
Pipelining is Natural

- Pipelining provides a method for executing multiple instructions at the same time.
- Laundry Example:
  - Four types of resources: Washer, Dryer, Folding, Storing
  - Each stage take 0.5 hours.
- Sequential executions of four jobs take 8 hours.
- Pipelined executions of four jobs take 3.5 hours.

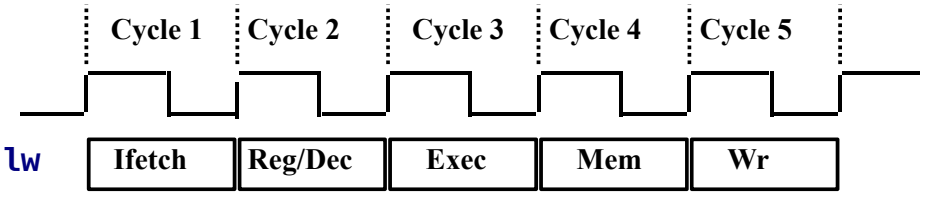


Pipelining Lessons

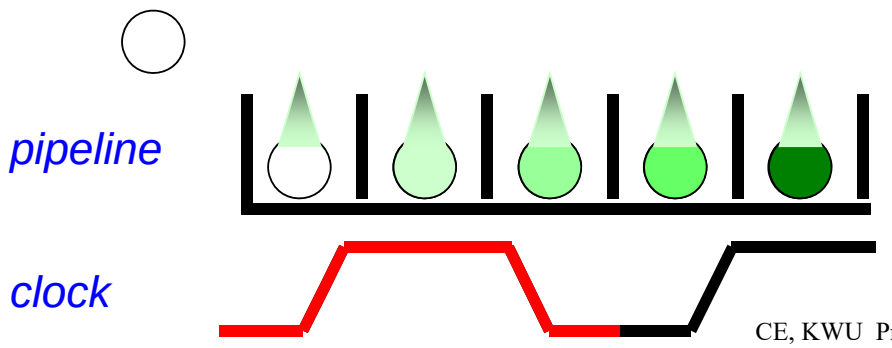
- Pipelining doesn't help latency of single task, it helps throughput of entire workload.
- Pipeline rate is limited by the slowest pipeline stage
- Multiple tasks operates simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" and "drain" the pipeline reduces speedup
- Stall for dependences reduces speedup



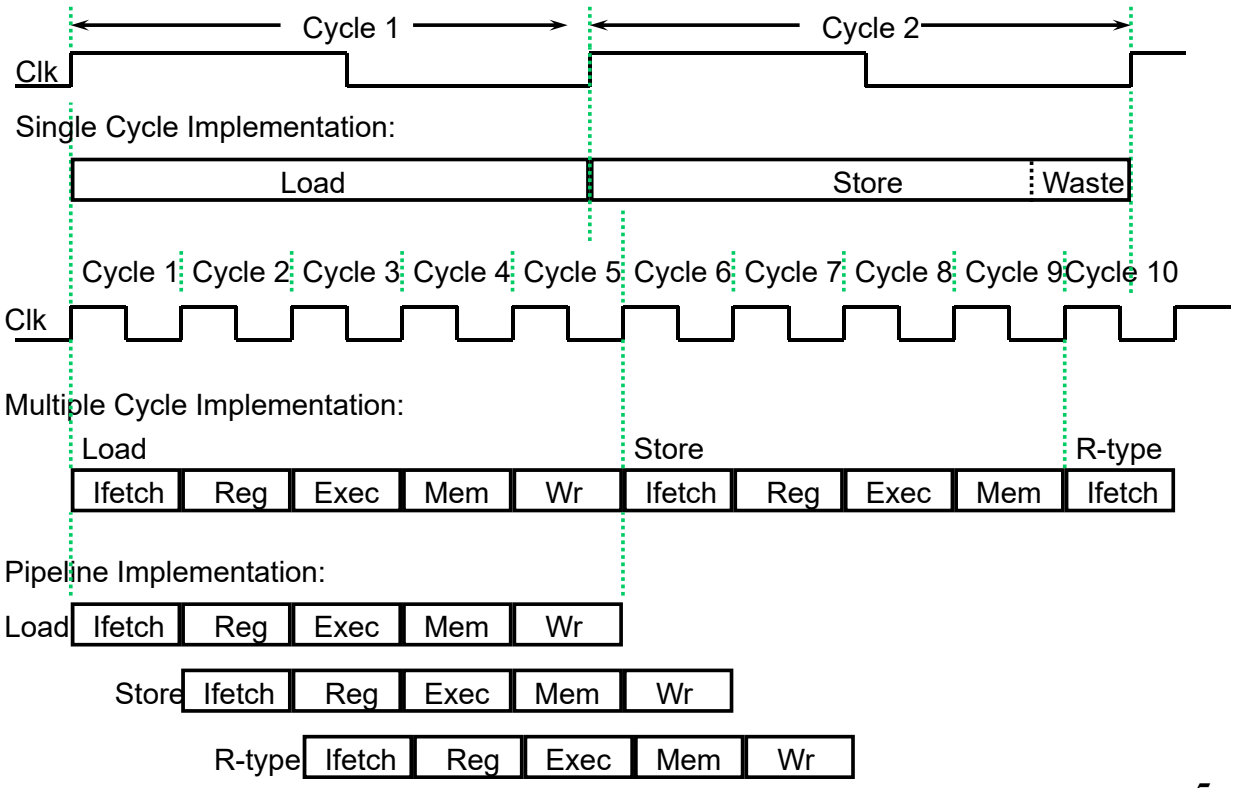
The Five Stages of An Instruction



- Ifetch: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- Reg/Dec: Registers Fetch and Instruction Decode
- Exec: Calculate the memory address
- Mem: Read the data from the Data Memory
- Wr: Write the data back to the register file



Single Cycle, Multiple Cycle, vs. Pipeline



Pipeline Performance

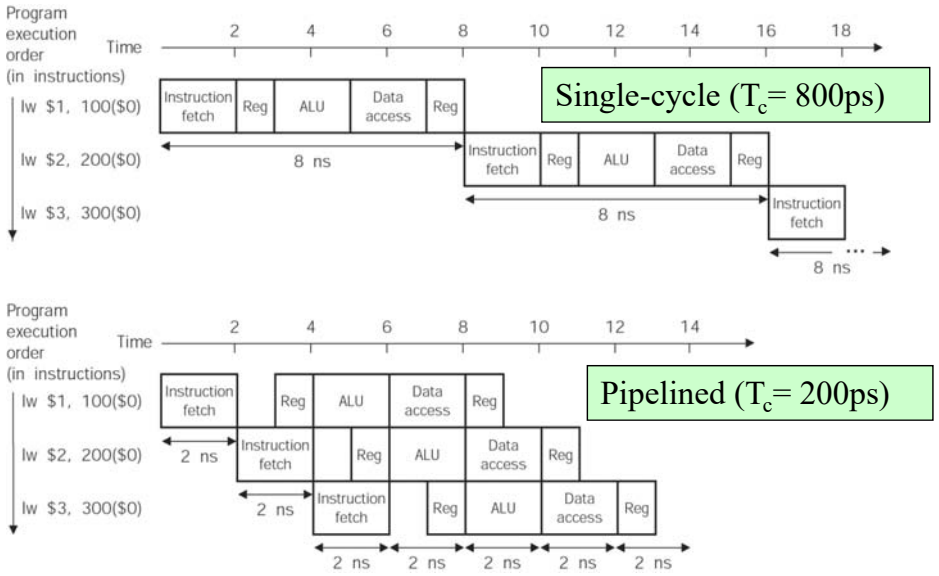
- **Assume time for stages is**
  - **100ps for register read or write**
  - **200ps for other stages**

- **Compare pipelined datapath with single-cycle datapath**

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

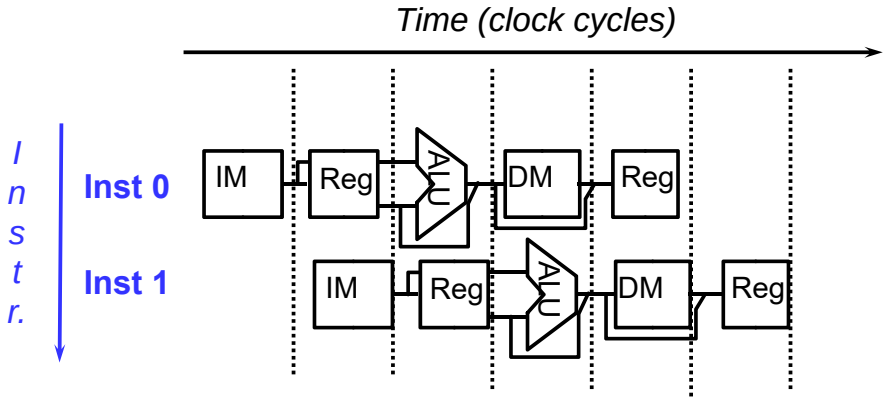
Pipelining

- Improve performance by increasing instruction throughput.



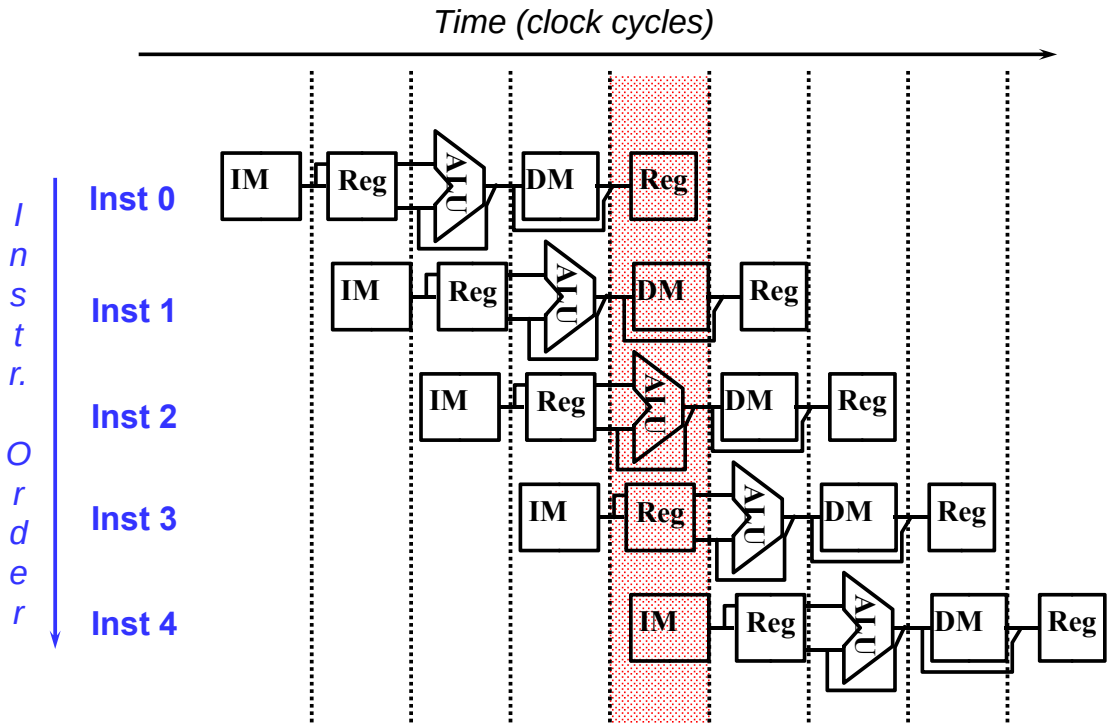
Ideal speedup is number of stages in the pipeline. Do we achieve this?

Graphically Representing Pipelines



- Can help with answering questions like:
  - How many cycles does it take to execute this code?
  - What is the ALU doing during cycle 4?
  - Are two instructions trying to use the same resource at the same time?

Why Pipeline? Because the resources are there!



Why Pipeline?

- Suppose
  - 100 instructions are executed
  - The single cycle machine has a cycle time of 45 ns
  - The multicycle and pipeline machines have cycle times of 10 ns
  - The multicycle machine has a CPI of 4.6
- Single Cycle Machine
  - 45 ns/cycle x 1 CPI x 100 inst = 4500 ns
- Multicycle Machine
  - 10 ns/cycle x 4.6 CPI x 100 inst = 4600 ns
- Ideal pipelined machine
  - 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns
- Ideal pipelined vs. single cycle speedup
  - 4500 ns / 1040 ns = 4.33
- What has not yet been considered?

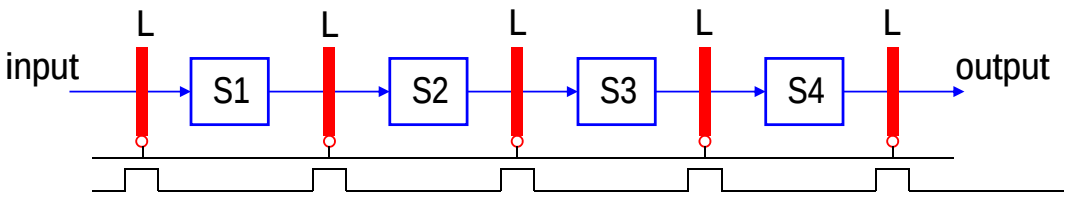
Pipelining and ISA Design

- **What makes it easy**
  - **all instructions are the same length**
  - **just a few instruction formats**
  - **memory operands appear only in loads and stores**
  - **Operands must be aligned in memory**

- **MIPS ISA designed for pipelining**
  - **All instructions are 32-bits**
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - **Few and regular instruction formats**
    - Can decode and read registers in one step
  - **Load/store addressing**
    - Can calculate address in 3rd stage, access memory in 4th stage
  - **Alignment of memory operands**
    - Memory access takes only one cycle

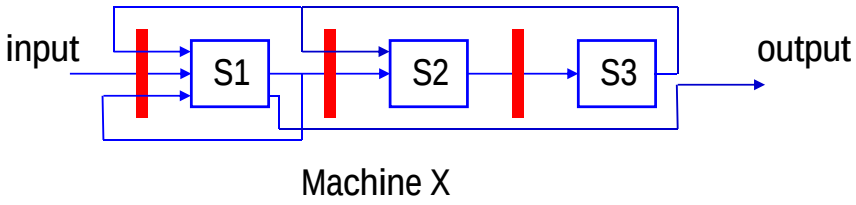
Pipeline Design

- **Pipelining increases throughput**
- **Linear pipeline is straightforward**



Non-linear Pipeline

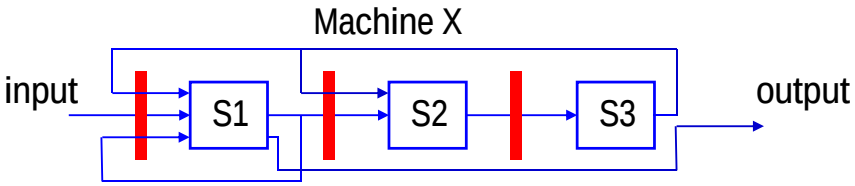
- **Stages might have different latencies**
- **Multiple paths out of one stage to other stages**
- **Multi-functional**



Machine X

- Generalized form of pipeline definition

Reservation Table

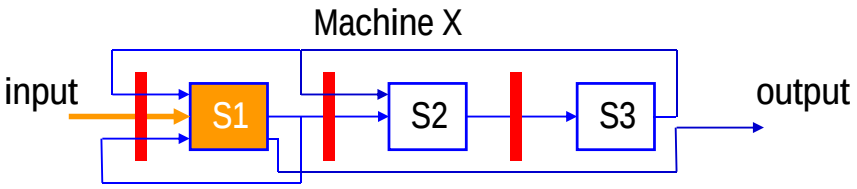


Reservation Table

Time →

Stage ↓	0	1	2	3	4	5	6	7
S1								
S2								
S3								

Reservation Table

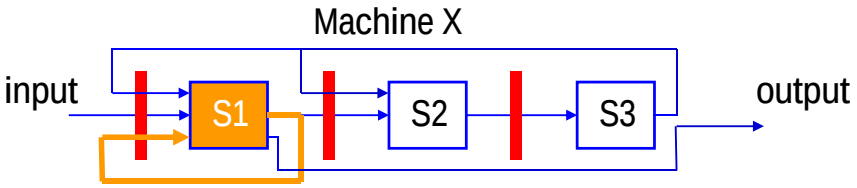


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X							
S2								
S3								

Reservation Table

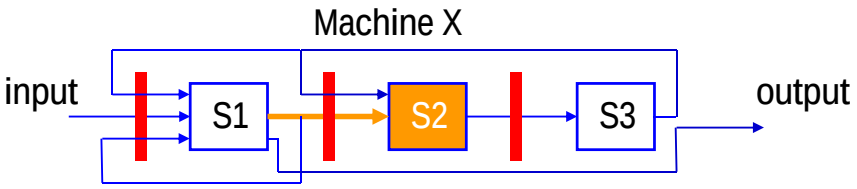


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X						
S2								
S3								

Reservation Table

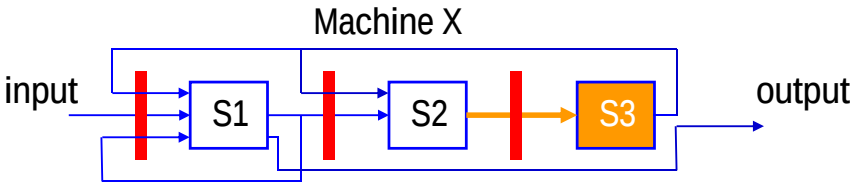


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X						
S2			X					
S3								

Reservation Table

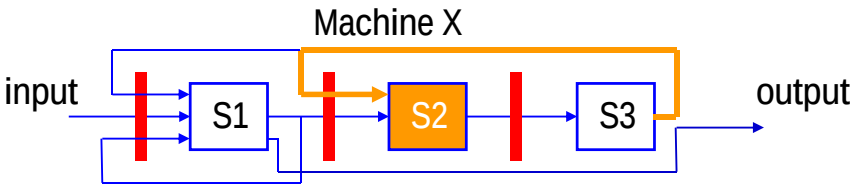


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X						
S2			X					
S3				X				

Reservation Table

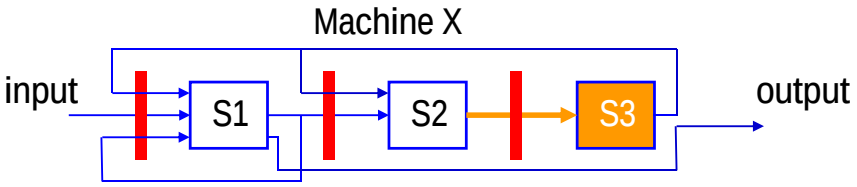


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X						
S2			X		X			
S3				X				

Reservation Table

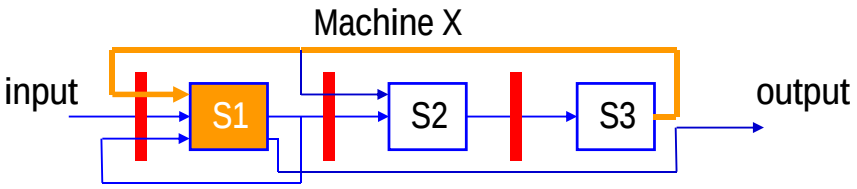


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X						
S2			X		X			
S3				X		X		

Reservation Table

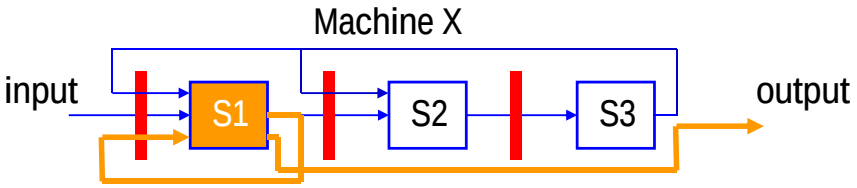


Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X					X	
S2			X		X			
S3				X		X		

Reservation Table



Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Collision Test for Initiating New Iterations

T=	0	1	2	3	4	5	6	7	8	9
S1	X	X					X	X		
S2			X		X					
S3				X		X				

O	O					O	O
		O		O			
			O		O		

- **Collision vector**
  - **A d-bit binary sequence where d is the compute time of the reservation table (e.g., how long an instruction is gonna stay in the pipeline)**
  - **The d bits are labeled 0 to (d-1) from left to right with a 0 indicating no collision and a 1 indicating collision**
- **Goal: use collision vector to schedule events in pipeline**

Collision Vector of Machine G

Lat = 0	0	1	2	3	4	5	6	7
S1	1 2	1 2					1 2	1 2
S2			1 2		1 2			
S3				1 2		1 2		

Collision Vector = 1xxxxxxx

Lat = 1	0	1	2	3	4	5	6	7
S1	1	1 2	2				1	1 2
S2			1	2	1	2		
S3				1	2	1	2	

Collision Vector = 11xxxxxx

Lat = 2	0	1	2	3	4	5	6	7
S1	1	1	2	2			1	1
S2			1		1 2		2	
S3				1		1 2		2

Collision Vector = 111xxxxx

Lat = 3	0	1	2	3	4	5	6	7
S1	1	1		2	2		1	1
S2			1		1	2		2
S3				1		1	2	

Collision Vector = 1110xxxx

Collision Vector of Machine G

Lat = 4	0	1	2	3	4	5	6	7
S1	1	1			2	2	1	1
S2			1		1		2	
S3				1		1		2

Collision Vector = 11100xxx

Lat = 5	0	1	2	3	4	5	6	7
S1	1	1				2	1 2	1
S2			1		1			2
S3				1		1		

Collision Vector = 111001xx

Lat = 6	0	1	2	3	4	5	6	7
S1	1	1					1 2	1 2
S2			1		1			
S3				1		1		

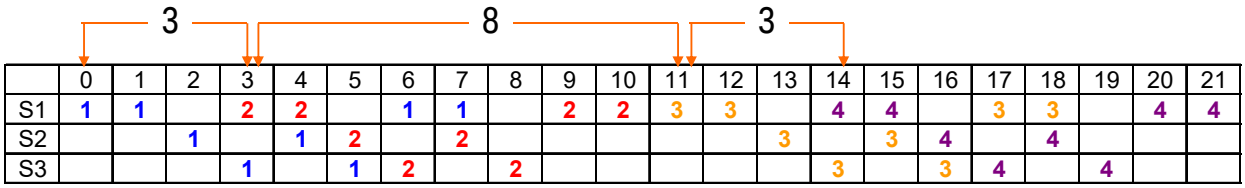
Collision Vector = 1110011x

Lat = 7	0	1	2	3	4	5	6	7
S1	1	1					1	1 2
S2			1		1			
S3				1		1		

Collision Vector = 11100111

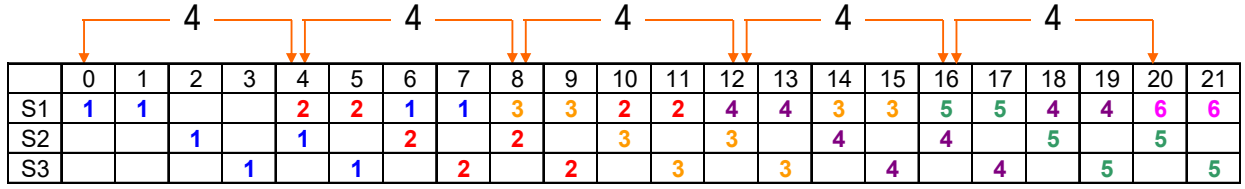
Pipeline Schedules of Machine G

- **How to achieve maximum throughput?**



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S1	1	1		2	2		1	1		2	2	3	3		4	4		3	3		4	4
S2			1		1	2		2						3		3	4		4			
S3				1		1	2		2						3		3	4		4		

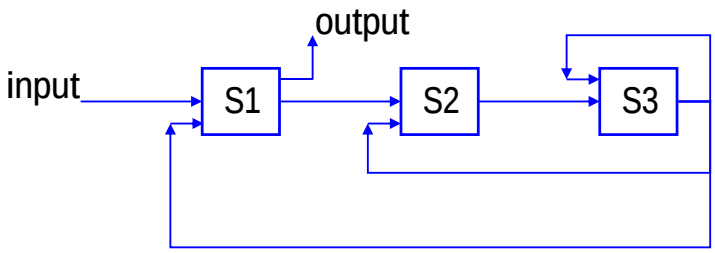
Greedy initiation
Latency sequence = <3, 8, 3, 8, ….>
Average latency = (3+8)/2 = 5.5 cycles



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S1	1	1			2	2	1	1	3	3	2	2	4	4	3	3	5	5	4	4	6	6
S2			1		1		2		2		3		3		4		4		5		5	
S3				1		1		2		2		3		3		4		4		5		5

Optimal initiation
Latency sequence = <4, 4, 4, 4, ….>
Average latency = 4 cycles

Improving Throughput by Delay Insertion



	0	1	2	3	4
S1	X				X
S2		X		X	
S3			X	X	

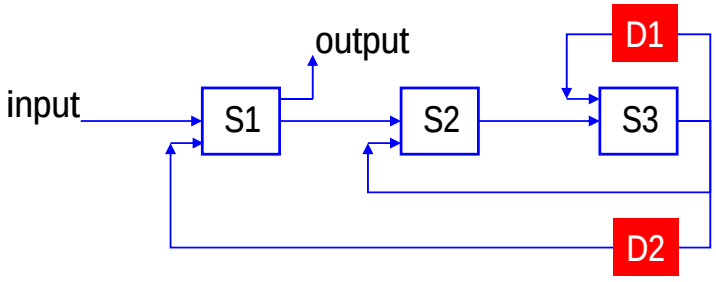
Initial Collision Vector

11101

Pipeline Latency = 5 cycles

- MAL=3; Can we do better than 3?

Delay Insertion



	0	1	2	3	4	5	6
S1	X					D2 →	X
S2		X		X			
S3			X		D1 → X		

Initial Collision Vector

1010001

Pipeline Latency = 7 cycles

- New latency sequence = <3,1,3,1,3,1,…>
- MAL = (3+1)/2 = 2

Pipelining Hazards

- **What makes it hard?**
  - **structural hazards: suppose we had only one memory**
  - **control hazards: need to worry about branch instructions**
  - **data hazards: an instruction depends on a previous instruction**

- **We'll talk about modern processors and what really makes it hard:**
  - **exception handling**
  - **trying to improve performance with out-of-order execution, etc.**

Can pipelining get us into trouble?

- **Yes: Pipeline Hazards**
  - **Structural hazards: attempt to use the same resource two different ways at the same time**
    - E.g., two instructions try to read the same memory at the same time
  - **Data hazards: attempt to use item before it is ready**
    - instruction depends on result of prior instruction still in the pipeline

      add r1, r2, r3

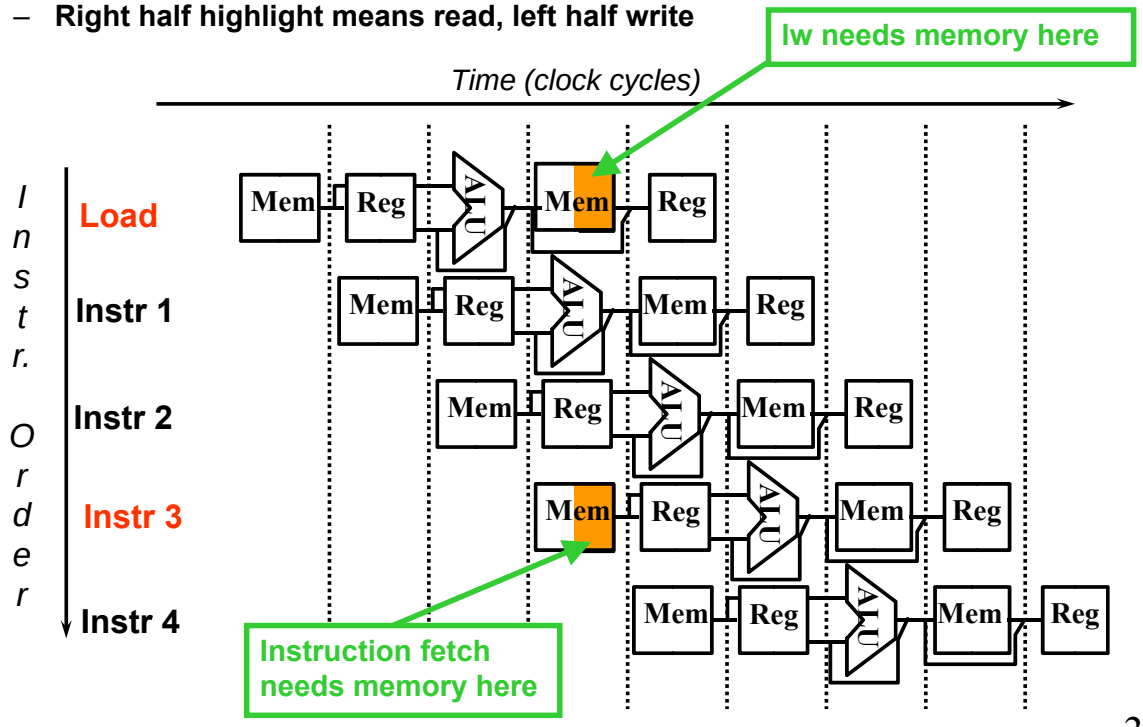
      sub r4, r2, r1
  - **Control hazards: attempt to make a decision before condition is evaulated**
    - branch instructions

      beq r1, loop

      add r1, r2, r3
- **Can always resolve hazards by waiting**
  - **pipeline control must detect the hazard**
  - **take action (or delay action) to resolve hazards**

Structural Hazards

- A Case of a Single Memory
  - Detection is easy in this case!
  - Right half highlight means read, left half write



Structural Hazards limit performance

- Example:
  - If 1.3 memory accesses per instruction and only one memory access per cycle then
    - average CPI should be more than 1.3
    - otherwise resource is more than 100% utilized (Impossible!)
- Solution 1:
  - Use separate instruction and data memories
- Solution 2:
  - Allow memory to read and write more than one word per cycle
- Solution 3:
  - Stall