Instruction Scheduling

Introduction

1. Instruction allows us to execute multiple instructions per cycle, allowing for greater speeds even in the presence of data dependencies

Improving IPC

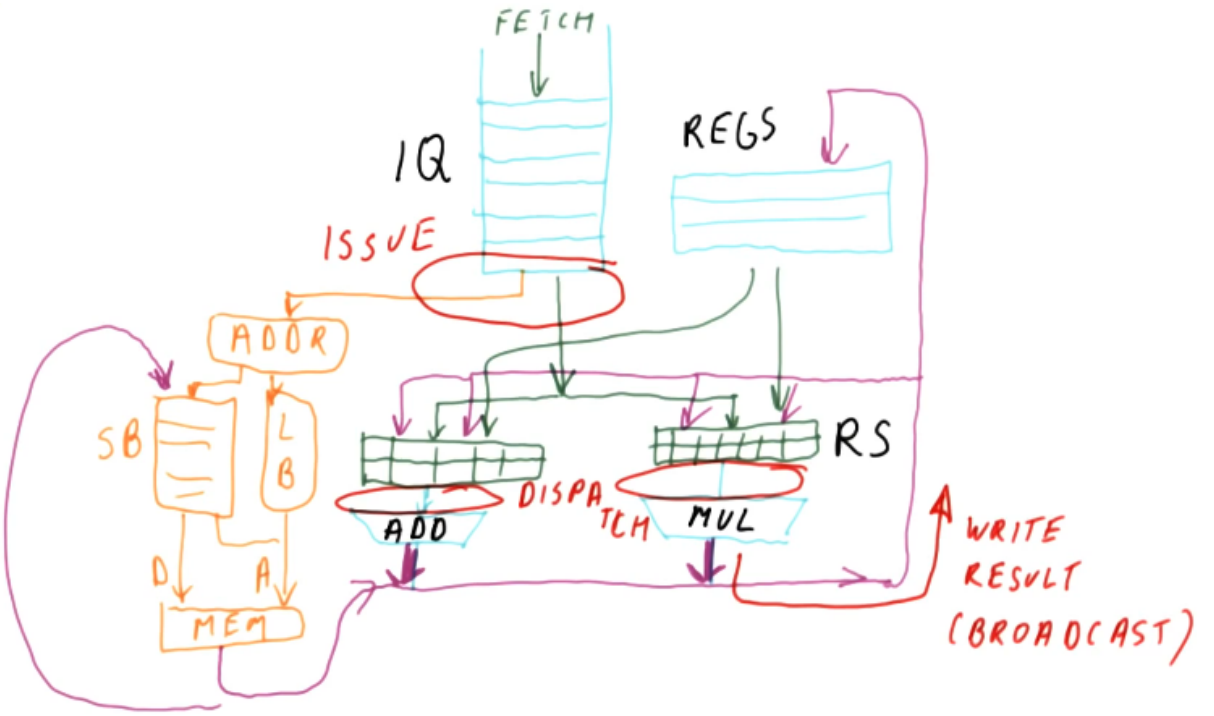
1. ILP is typically >> 1
2. Need to reduce control dependencies -> branch prediction
3. WAR/WAW data dependencies -> register renaming
4. RAW data dependencies -> out of order execution
5. Structural dependencies -> invest in wider-issue processors

Tomasulo's Algorithm

1. Used in IBM 360 (more than 40 years old)
2. Determine which instructions have inputs ready
3. Performs register renaming
4. Very similar to what we use today
5. Differences between Tomasulo and today's implementation
    - Tomasulo only operated on floating point instructions, today it's applied to all instructions
    - Tomasulo used fewer instructions in the "window" while today, we apply it to 100s of instructions
    - Tomasulo provided little support for exception handling. This is fully supported in modern processors

The Big Picture

1. IQ: Instruction queue
2. RS: Reservation station
3. RF (Regs): Register file
4. Addr: Address generation unit
5. RAT: Register aliasing table
6. SB: Store buffer
7. LB: Load buffer
8. IQ -> RS: Issue
9. RS -> Execution: Dispatch
10. Execution -> Bus: Write result (broadcast)
11. Tomasulo didn't reorder loads and stores



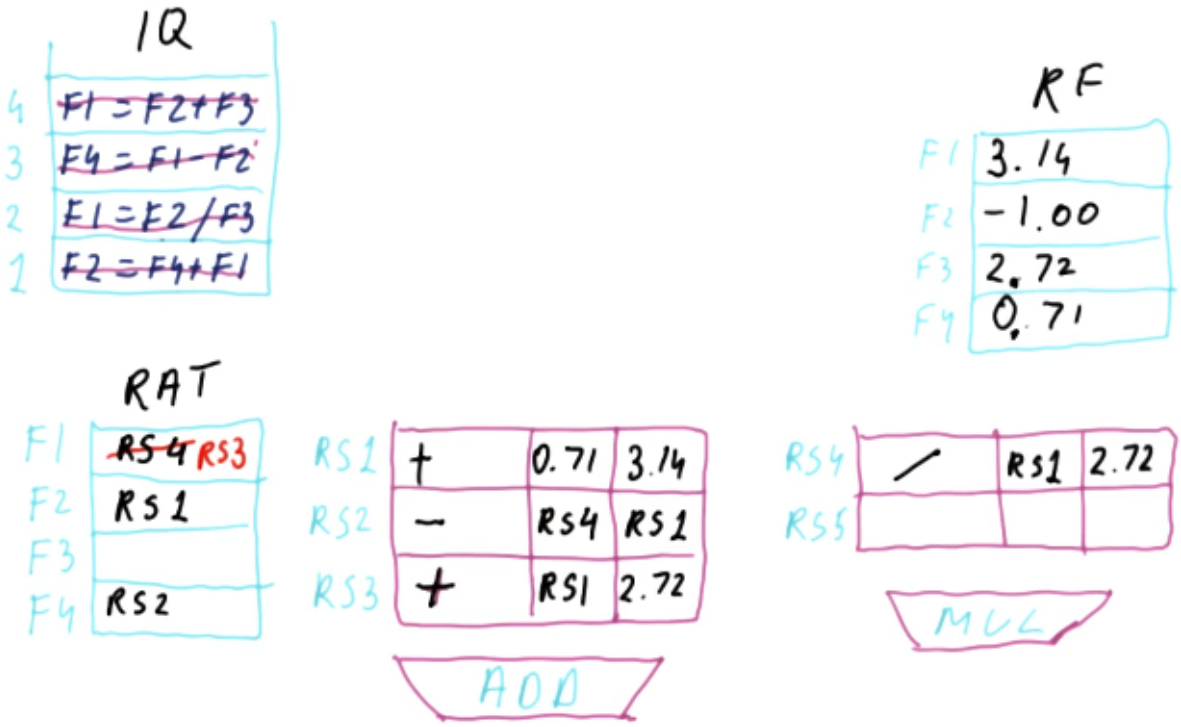
Tomasulo's Algorithm - The Big Picture

Issue

1. Take next (in program order) instruction from IQ
2. Determine where inputs come from
3. Get available reservation station (of correct kind, ADD vs MUL)
4. Put instruction in RS
5. Tag destination register

Issue Example

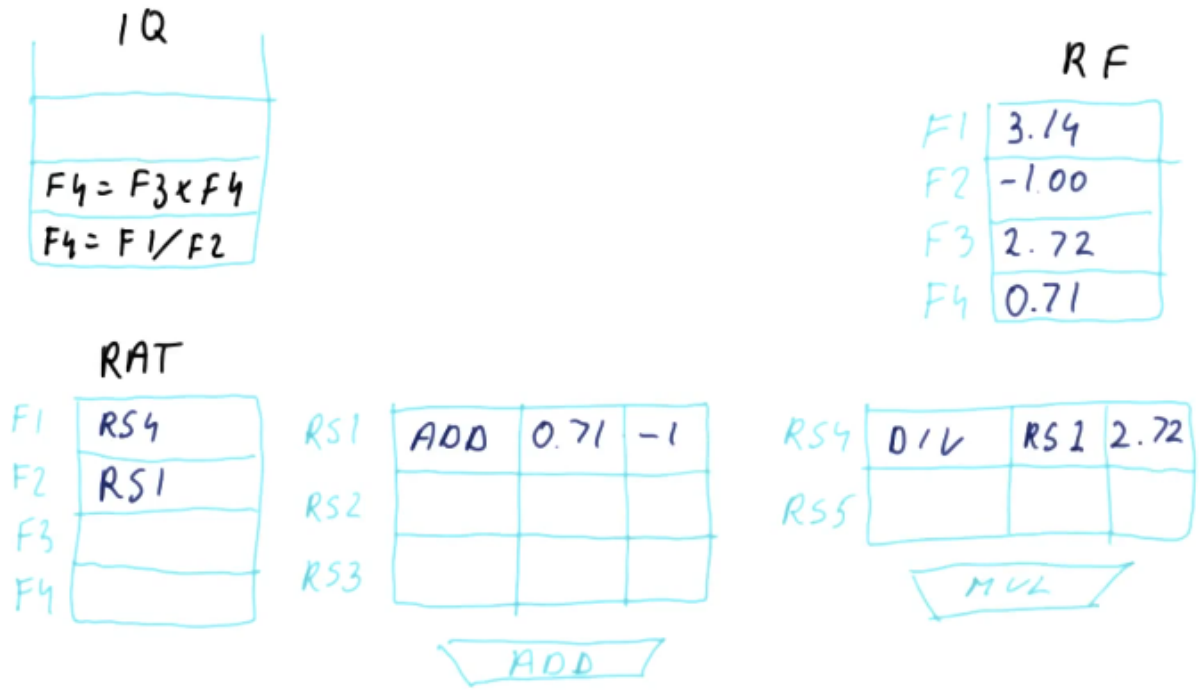
1. Take an instruction from the instruction buffer
2. Find if our inputs are ready or not
3. If ready, place instruction in reservation station (assuming one is available)
4. Update RAT to indicate that future instructions should read from the reservation station instead of the register file



Issue Example

Issue Quiz

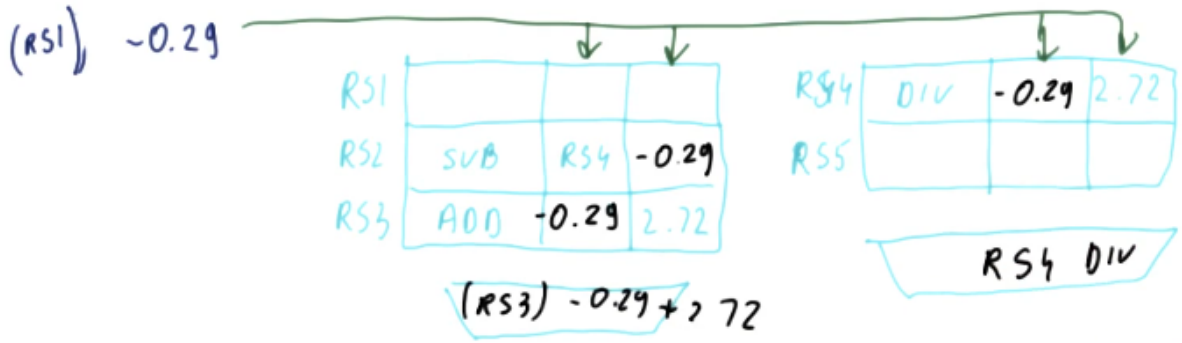
1. Issue the next two instructions for the following example:
   - RS5: DIV, RS4, RS1, F4 <- RS5
   - F4 = F3 * F4 can't be issued since the multiply/divide reservation station is full



Issue Quiz

Dispatch

1. Dispatch: Latch values back into the reservation station
2. First, mark the reservation station as available
3. Then, replace all instances waiting for that value with the value
4. Find all reservation stations that have all of their operands
5. Give each execution unit a ready instruction, if possible



Dispatch Quiz

More Than 1 Instruction Ready

1. When multiple instructions have all of their operands, how do we decide which to send to the execution unit?

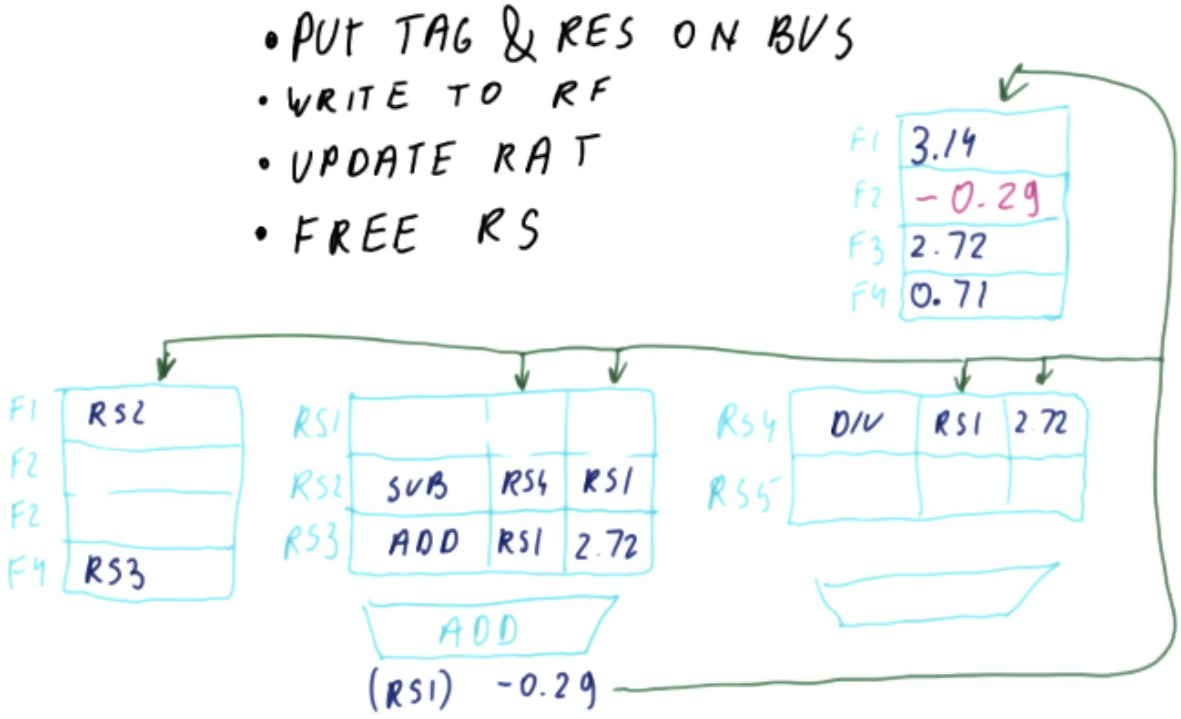
2. Ideally, we dispatch the instruction that allows us to resolve future data dependencies as early as possible
    - Requires knowledge of the future, which is hard
3. Can't do it perfectly, but some useful heuristics
    - Oldest first (this is what is typically used)
    - Most dependencies first (count how many other instructions need the value)
    - Random

Dispatch Quiz

1. What are the possible reasons an instruction might have all of its operands but still be in the reservation station?
    - It was issued in the previous cycle (true)
    - Another instruction was dispatched to the execution unit (true)
    - There is an older instruction missing operands, which prevents the newer instruction from dispatching (false)

Write Result (Broadcast)

1. First, put the tag and RES on the bus
2. Write to register file
3. Update register aliasing table
4. Free the reservation station



Broadcast

More Than 1 Broadcast

1. Multiple execution units could be ready in the same cycle
    - If there's only one bus, which goes first?

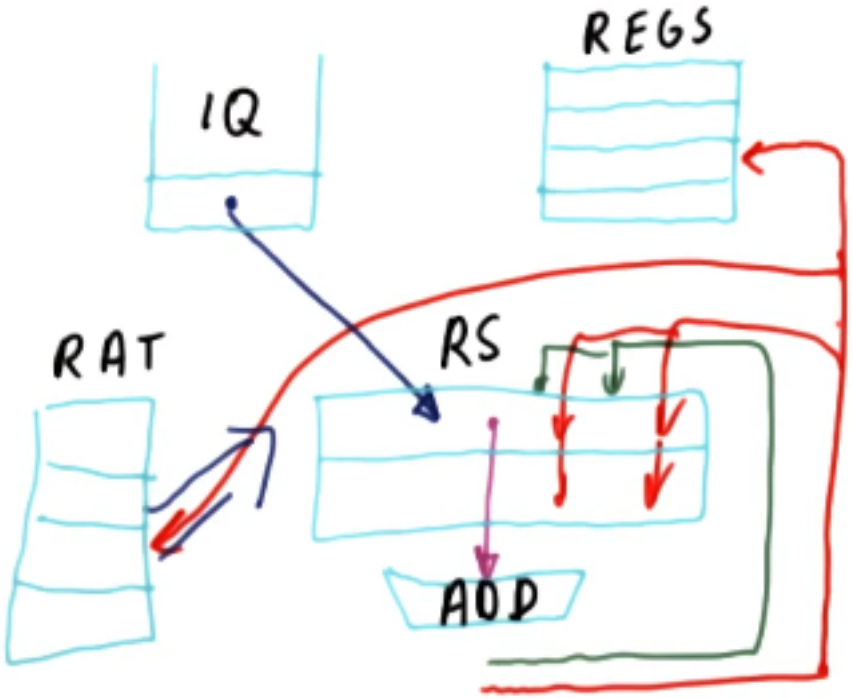
2. Could have a separate bus for each execution unit, but requires significantly more hardware
    - Not typically implemented due to increased cost
3. Common heuristic for picking a unit: Slowest first
    - This is due to the fact that there are likely to be more instructions waiting on the result
    - Multiply/divide is typically slower than add/subtract

Broadcast Stale Result

1. It's possible that the RAT/RF is not depending on the result of an execution unit at the time it completes ("stale")
2. In this case, we still use the value to write back to the reservation stations
3. However, if the tag doesn't match any entry in the RF or RAT, we simply do nothing
    - This handles the case where all of the instructions depending on the result are already in the RS, so there's no need for additional work

Review Part 1

1. For one instruction:
    - Issue
    - Capture results from other instructions
    - Dispatch
    - Write result
2. Every cycle, each of these things is happening:
    - Issue
    - Capture
    - Dispatch
    - Write result

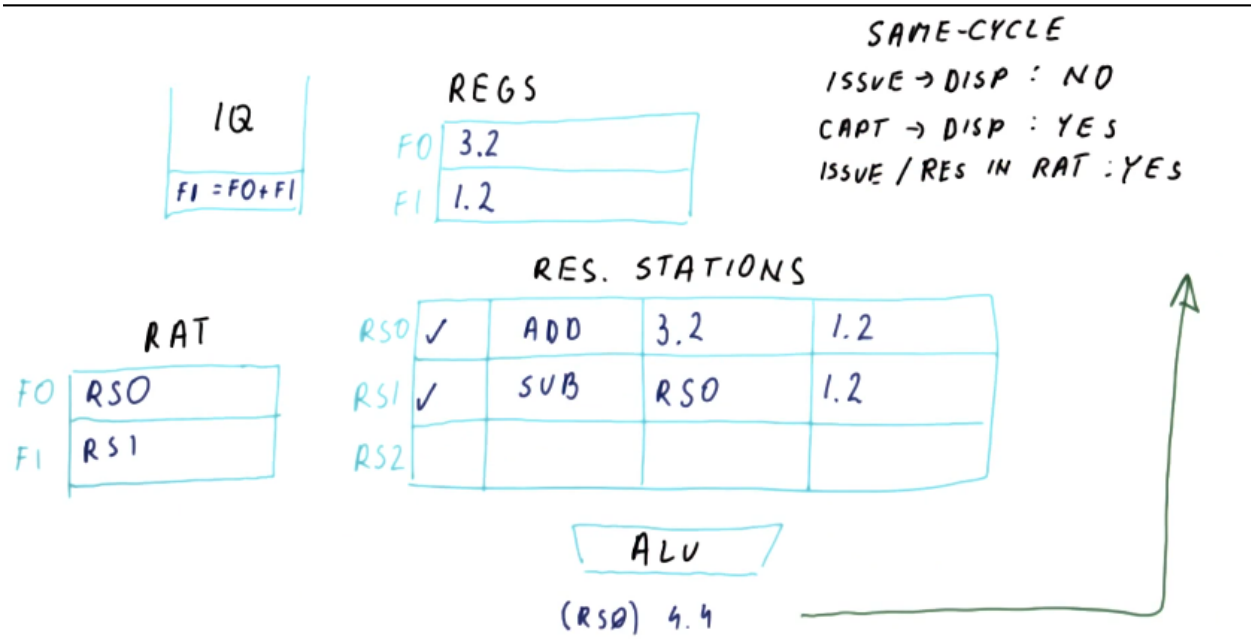


Review

Review Part 2

1. Interesting cases:
   - Can we do same-cycle issue -> dispatch? No
     - Writing to RS and dispatching happen simultaneously, so it isn't ready
   - Can we do same-cycle capture -> dispatch? No
     - Same reason as above, need to wait until next cycle
   - Can we update RAT entry for issue and write result in the same cycle? Yes
     - Need to ensure the issuing instruction is written last since future instructions depend on that value
     - Since it's simple (just write the issue value), we can handle this case

One Cycle Quiz Part 1



One Cycle Quiz

1. What are the contents of the RAT and RF at the end of the next cycle?

Reg	RAT
F0	F0
F1	RS2

Reg	RF
F0	4.4
F1	1.2

One Cycle Quiz Part 2

1. What is in the reservation stations at the end of the next cycle?

RS	Used	Instr	Value 1	Value 2
RS0		ADD	3.2	1.2
RS1	X	SUB	4.4	1.2
RS2	X	ADD	4.4	RS1

One Cycle Quiz Part 3

1. What will dispatch in this cycle?
    - Only RS1 will dispatch

Tomasulo's Algorithm Quiz

1. Which is NOT true about Tomasulo's algorithm?
    - It issues instructions in program order (true)
    - It dispatches instructions in program order (false)
    - It writes results in program order (false)

Load and Store Instructions

1. Similar to data dependencies in logical instructions, memory instructions can also have dependencies
    - RAW: SW to A, then LW from A
    - WAR: LW then SW
    - WAW: SW, SW to same address
2. What do we do about this?
    - In Tomasulo's algorithm, we do loads and stores in-order
    - Alternatively, identify dependencies and reorder (complicated)
        - Modern processors take this approach

Long Example Cycles 1-2

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1	✓	L		134			
LD2							
AD1							
AD2							
AD3							
ML1							
ML2							

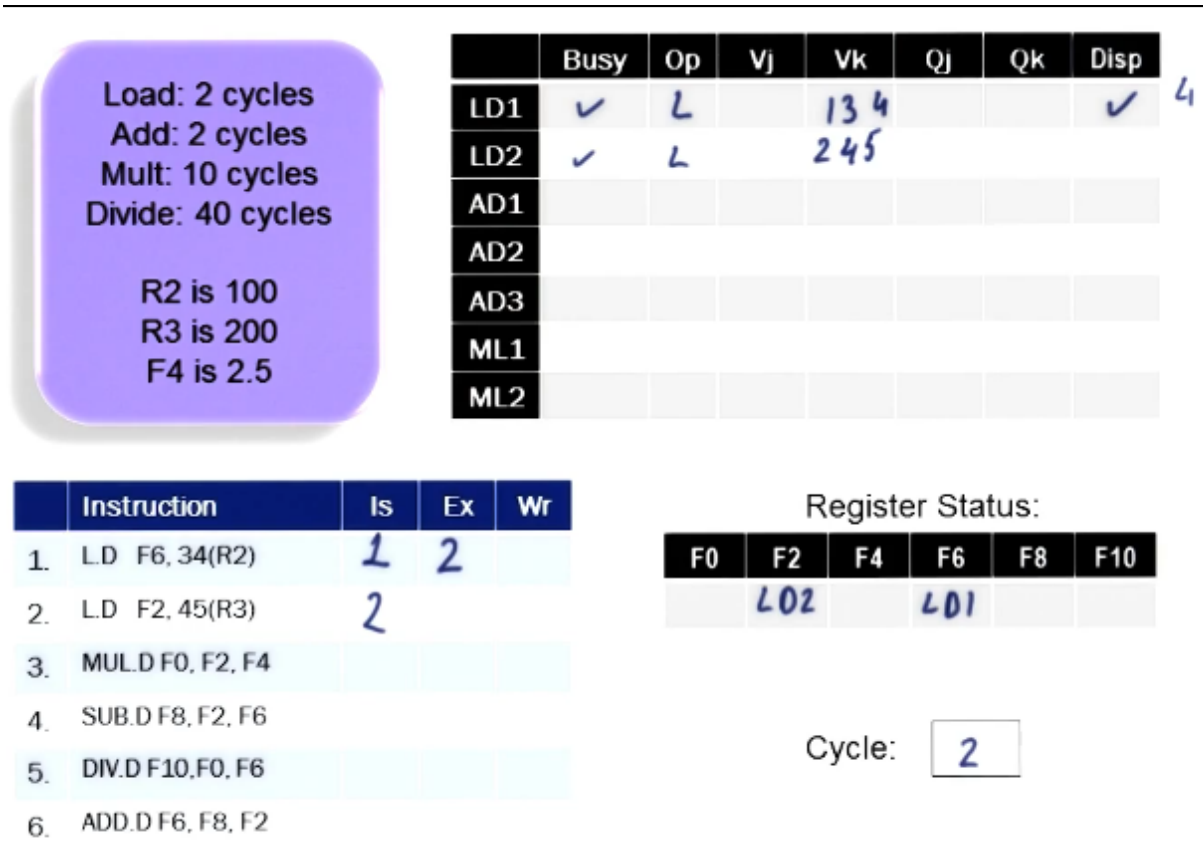
	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1		
2.	L.D F2, 45(R3)			
3.	MUL.D F0, F2, F4			
4.	SUB.D F8, F2, F6			
5.	DIV.D F10,F0, F6			
6.	ADD.D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
			LD1		

Cycle: 1

Cycle 1



Cycle 2

Long Example Cycles 3-4

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp	
LD1	✓	L		134			✓	4
LD2	✓	L		245				4
AD1								
AD2								
AD3								
ML1	✓	M		2.5	LD2			
ML2								

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	
2.	L.D F2, 45(R3)	2		
3.	MUL.D F0, F2, F4	3		
4.	SUB.D F8, F2, F6			
5.	DIV.D F10,F0, F6			
6.	ADD.D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
ML1	LD2		LD1		

Cycle: 3

Cycle 3

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2	✓	L		245			✓
AD1	✓	S		7.1	LD2		
AD2							
AD3							
ML1	✓	M		2.5	LD2		
ML2							

F6 7.1

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	
3.	MUL.D F0, F2, F4	3		
4.	SUB.D F8, F2, F6	4		
5.	DIV.D F10,F0, F6			
6.	ADD.D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
ML1	LD2			AD1	

Cycle: 4

Cycle 4

Long Example Cycles 5-6

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2	✓	L		245			✓
AD1	✓	S		7.1	LD2		
AD2							
AD3							
ML1	✓	M		2.5	LD2		
ML2	✓	D		7.1	ML1		

F6 7.1

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	
3.	MUL.D F0, F2, F4	3		
4.	SUB.D F8, F2, F6	4		
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
ML1	LD2			AD1	ML2

Cycle: 5

Cycle 5

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2	✓	L		245			✓
AD1	✓	S	-2.5	7.1			
AD2	✓	A		-2.5	AO1		
AD3							
ML1	✓	M	-2.5	2.5			
ML2	✓	D		7.1	ML1		

F2 -2.5
F6 7.1

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3		
4.	SUB.D F8, F2, F6	4		
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
ML1			AO2	AO1	ML2

Cycle: 6

Cycle 6

Long Example Cycles 7-9

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp	
LD1								
LD2								
AD1	✓	S	-2.5	7.1			✓	9
AD2	✓	A		-2.5	AO1			
AD3								
ML1	✓	M	-2.5	2.5			✓	17
ML2	✓	D		7.1	ML1			

F2 -2.5
F6 7.1

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	
4.	SUB.D F8, F2, F6	4	7	
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
ML1			AO2	AO1	ML2

Cycle: 7

Cycle 7

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp	
LD1								
LD2								
AD1								
AD2	✓	A	-9.6	-2.5				
AD3								
ML1	✓	M	-2.5	2.5			✓	17
ML2	✓	D		7.1	ML1			

F2 -2.5
F6 7.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
ML1			AD2		ML2

Cycle: 9

Cycle 9

Long Example Cycles 10-end

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp	
LD1								
LD2								
AD1								
AD2	✓	A	-9.6	-2.5			✓	12
AD3								
ML1	✓	M	-2.5	2.5			✓	17
ML2	✓	D		7.1	ML1			

F2 -2.5
F6 7.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6	10	

Register Status:

F0	F2	F4	F6	F8	F10
ML1			AD2		ML2

Cycle: 10

Cycle 10

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1							
AD2							
AD3							
ML1	✓	M	-2.5	2.5		✓	17
ML2	✓	D		7.1	ML1		

F2 -2.5
F6 -12.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6	10	12

Register Status:

F0	F2	F4	F6	F8	F10
ML1					ML2

Cycle: 12

Cycle 12

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1							
AD2							
AD3							
ML1							
ML2	✓	D	?	7.1			

F0 ?
F2 -2.5
F6 -12.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	17
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5		
6.	ADD.D F6, F8, F2	6	10	12

Register Status:

F0	F2	F4	F6	F8	F10
					ML2

Cycle:

Cycle 17

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1							
AD2							
AD3							
ML1							
ML2	✓	D	?	7.1		✓	58

F0 ?
F2 -2.5
F6 -12.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	17
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5	18	
6.	ADD.D F6, F8, F2	6	10	12

Register Status:

F0	F2	F4	F6	F8	F10
					ML2

Cycle: 18

Cycle 18

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1							
AD2							
AD3							
ML1							
ML2							

F0 ?
F2 -2.5
F6 -12.1
F8 -9.6
F10 ??

	Instruction	Is	Ex	Wr
1.	L.D F6, 34(R2)	1	2	4
2.	L.D F2, 45(R3)	2	4	6
3.	MUL.D F0, F2, F4	3	7	17
4.	SUB.D F8, F2, F6	4	7	9
5.	DIV.D F10,F0, F6	5	18	58
6.	ADD.D F6, F8, F2	6	10	12

Register Status:

F0	F2	F4	F6	F8	F10

Cycle: 58

Cycle 58

Timing Example

1. Assumptions:
   - Load: 2 cycles
   - Add: 2 cycles
   - Multiply: 10 cycles
   - Divide: 40 cycles
2. After tracking timing in the table, double check to ensure no two instructions are broadcasting in the same cycle

Number	Instruction	Issue	Execute	Write	Comment
1	LD F6,34(R2)	1	2	4	
2	LD F2,45(R3)	2	4	6	
3	MUL F0,F2,F4	3	7	17	F2 from 2
4	SUB F8,F2,F6	4	7	9	F2 from 2
5	DIV F10,F0,F6	5	18	58	F0 from 3
6	ADD F6,F8,F2	6	10	12	F8 from 4

Tomasulo Timing Quiz

1. Assumptions:
   - Latency:
     - Load: 1 cycle
     - Add: 1 cycle
     - Multiply: 5 cycles
   - Number of reservation stations
     - Load: 1
     - Add: 2
     - Multiply: 2
   - Same cycle:
     - Issue -> Dispatch: No
     - Capture -> Dispatch: No
     - RS freed -> RS allocate: No
2. Need to be aware of when all ADD reservation stations are in use; can't issue new instructions when this is the case

Number	Instruction	Issue	Dispatch	Write
1	LD F6, 0(R2)	1	2	3
2	MUL F2,F0,F1	2	3	8
3	ADD F6,F2,F6	3	9	10
4	ADD F6,F2,F6	4	11	12
5	ADD F1,F1,F1	11	12	13
6	ADD F1,F3,F4	13	14	15

Conclusion

1. Tomasulo's algorithm shows how a processor can reorder and rename instructions to work around dependencies
2. How do we handle exceptions?
   - Divide by zero
   - Memory protection exception