

CSEN
702

Microprocessors

Lectures
13

Instruction Level Parallelism (part 5)

Dynamic scheduling-Tomasulo

Reading

Textbook Chapter 3

Part 1

A tomasulo loop based example

Consider the following loop

- A loop that multiplies the values of an array by a scalar stored in F2. (1.33 for example)
- R1 is pre-loaded with value 80.

```
LOOP:    L.D    F0, 0(R1)
          MUL.D  F4, F0, F2
          S.D    F4, 0(R1)
          SUBI   R1, R1, 8
          BNEZ   R1, LOOP
```

Assumptions

- Multiply takes 4 cycles to execute.
- The first load takes 8 cycles because of a cache miss.
- Successive loads take 1 clock cycle.
- Two loads or two stores cannot be executed at the same time.

Whole system status initially

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result

Reg. file	
	Q _i
F0	
F2	0
F4	
F6	
F8	
F10	

Clock

R1
value

80

Load buffers

Store buffers

	Busy	Address		Busy	Address	V	Q
L1	0		S1	0			
L2	0		S2	0			
L3	0		S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	0						
	M2	0					4	

Cycle 1

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1		

Reg. file		Clock
	Q _i	
F0	L1	R1 value 80
F2	0	
F4		
F6		
F8		
F10		

Load buffers

	Busy	Address
L1	1	80
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	0			
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	0						
	M2	0					6	

Cycle 2

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		

Reg. file		Clock
	Q _i	2
F0	L1	
F2	0	R1 value
F4	M1	
F6		
F8		
F10		
		80

Load buffers

	Busy	Address
L1	1	80
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	0			
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	0					7	

Cycle 3

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		

Reg. file	
	Q _i
F0	L1
F2	0
F4	M1
F6	
F8	
F10	

Clock

3

R1 value

80

Load buffers

	Busy	Address
L1	1	80
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	0					8	

Cycle 4

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		

Reg. file		Clock
	Q_i	
F0	L1	4
F2	0	R1 value
F4	M1	
F6		80
F8		
F10		

Since this is integer operation, it doesn't enter the tumosalo's architecture

Load buffers

	Busy	Address
L1	1	80
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	0					9	

Cycle 5

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		

Reg. file	
	Q _i
F0	L1
F2	0
F4	M1
F6	
F8	
F10	

Clock

5

R1
value

72

Load buffers

	Busy	Address
L1	1	80
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	0					10	

Cycle 6

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6		

Reg. file		Clock <div>6</div>
	Q _i	
F0	L2	R1 value <div>72</div>
F2	0	
F4	M1	
F6		
F8		
F10		

Discussion

What happened here?
L2 overrides L1 in the register file. Is this a problem?

Load buffers

	Busy	Address
L1	1	80
L2	1	72
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	0					11	

Cycle 7

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6		
2	MUL	F4	F0	F2	7		

Reg. file		Clock
	Q _i	7
F0	L2	
F2	0	R1 value
F4	M2	
F6		
F8		
F10		72

Discussion

What happened here?
M2 overrides M1 in the register file. Is this a problem?

Load buffers

	Busy	Address
L1	1	80
L2	1	72
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	0			
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	1	MUL		R[F2]	L2	12	

Cycle 8

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6		
2	MUL	F4	F0	F2	7		
2	S.D	F4	0	R1	8		

Reg. file	
	Q _i
F0	L2
F2	0
F4	M2
F6	
F8	
F10	

Clock

8

R1 value

72

Load buffers

	Busy	Address
L1	1	80
L2	1	72
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	1	MUL		R[F2]	L2	13	

Cycle 9

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



In this cycle, SUBI is working and the load from iteration 1 finished execution (it needed 8 cycles as stated)

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6		
2	MUL	F4	F0	F2	7		
2	S.D	F4	0	R1	8		

Reg. file		Clock
	Q _i	
F0	L2	9
F2	0	R1 value 72
F4	M2	
F6		
F8		
F10		

Load buffers

	Busy	Address
L1	1	80
L2	1	72
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
	M1	1	MUL		R[F2]	L1		
	M2	1	MUL		R[F2]	L2	14	

Cycle 10

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



In this cycle, BNEZ is dispatched, Load from iteration 1 writes result on CDB, load from iteration 2 starts executing

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2		
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	
2	MUL	F4	F0	F2	7		
2	S.D	F4	0	R1	8		

Reg. file	
	Q _i
F0	L2
F2	0
F4	M2
F6	
F8	
F10	

Clock

10

R1
value

64

Load buffers

	Busy	Address
L1	0	
L2	1	72
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
4	M1	1	MUL	MEM[80]	R[F2]			
	M2	1	MUL		R[F2]	L2	15	

Cycle 11

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```



Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...	
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7		
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11		

Reg. file	
	Q _i
F0	L3
F2	0
F4	M2
F6	
F8	
F10	

Clock

11

R1 value

64

In this cycle:

- L.D of third iteration is issued because we have load empty buffers.
- L.D of iteration 2 finishes (it needs 1 cycle now)

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	1	64

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
3	M1	1	MUL	MEM[80]	R[F2]			
4	M2	1	MUL	MEM[72]	R[F2]		16	

Cycle 12

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

- In this cycle:
- MUL.D of iteration 2 starts executing.
 - Can we issue MUL.D of the third iteration?

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...	
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...	
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11	12	

Reg. file		Clock
	Q_i	12
F0	L3	R1 value
F2	0	
F4	M2	64
F6		
F8		
F10		

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	1	64

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
2	M1	1	MUL	MEM[80]	R[F2]			
3	M2	1	MUL	MEM[72]	R[F2]		17	

Cycle 13

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

In this cycle:

- Can we issue S.D of iteration 3?
- → NO! once an instruction in the queue is stuck, we don't issue after it. We must adhere to FIFO order

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...	
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...	
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11	12	13

Clock

13

Reg. file

	Q _i
F0	0
F2	0
F4	M2
F6	
F8	
F10	

mem[64]

R1 value

64

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
1	M1	1	MUL	MEM[80]	R[F2]			
2	M2	1	MUL	MEM[72]	R[F2]		18	

Cycle 14

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

In this cycle:

- the first MUL.D finished execution!

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...14	
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...	
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11	12	13

Reg. file	
	Q _i
F0	0
F2	0
F4	M2
F6	
F8	
F10	

Clock

14

R1 value

64

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	0	

Store buffers

	Busy	Address	V	Q
S1	1	80		M1
S2	1	72		M2
S3	0			

Time

Time remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
0	M1	1	MUL	MEM[80]	R[F2]			
1	M2	1	MUL	MEM[72]	R[F2]		19	

Cycle 15

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

In this cycle:

- the first MUL.D writes result
- Second MUL.D finishes execution.

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...14	15
1	S.D	F4	0	R1	3		
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...15	
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11	12	13

Reg. file	
	Q _i
F0	0
F2	0
F4	M2
F6	
F8	
F10	

Clock

15

R1 value

64

Load buffers

	Busy	Address
L1	0	
L2	0	
L3		

Store buffers

	Busy	Add.	V	Q
S 1	1	80	MEM[80]*1.33	
S 2	1	72		M2
S	0			

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
0	M1	0						
0	M2	1	MUL	MEM[72]	R[F2]		20	

Cycle 16

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

- In this cycle:
- multiply of iteration 2 writes result
 - multiply of iteration 3 can now be **issued**!
 - S1 can start executing

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
1	L.D	F0	0	R1	1	2...9	10
1	MUL	F4	F0	F2	2	11...14	15
1	S.D	F4	0	R1	3	16	
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...15	16
2	S.D	F4	0	R1	8		
3	L.D	F0	0	R1	11	12	13
3	MUL	F4	F0	F2	16		

Reg. file		Clock
	Q_i	16
F0	0	R1 value 64
F2	0	
F4	M1	
F6		
F8		
F10		

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	0	

Store buffers

	Busy	Add.	V	Q
S 1	1	80	MEM[80]*1.33	
S 2	1	72	MEM[72]*1.33	
S	0			

Time remaining

		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
4	M1	1	MUL	mem[64]	1.33			
0	M2	0					21	

Cycle 17

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

- In this cycle:
- Now this S.D of iteration 3 can be issued (it was stuck before because MUL was stuck)
 - Store of iter. 1 writes back, store of iter.2 executes

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	0	

Store buffers

	Busy	Add.	V	Q
S 1				
S 2	1	72	MEM[72]*1.33	
S	1	64		M1

Iteration #	Instruction	j	k	Issue	Execution Complete	Write Result
3	S.D	F4	0	R1	17	
1	MUL	F4	F0	F2	2	11...14
1	S.D	F4	0	R1	3	16
2	L.D	F0	0	R1	6	10
2	MUL	F4	F0	F2	7	12...15
2	S.D	F4	0	R1	8	17
3	L.D	F0	0	R1	11	12
3	MUL	F4	F0	F2	16	17..

Reg. file	
	Q _i
F0	0
F2	0
F4	M1
F6	
F8	
F10	

Clock

17

R1 value

64

Time

Time remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
3	M1	1	MUL	mem[64]	1.33			
0	M2	0					22	

Cycle 18

```
LOOP:  L.D F0, 0(R1)
        MUL.D F4, F0, F2
        S.D F4, 0(R1)
        SUBI R1, R1, 8
        BNEZ R1, LOOP
```

- In this cycle:
- Store of iteration 2 writes back

Iteration #	Instruction		j	k	Issue	Execution Complete	Write Result
3	S.D	F4	0	R1	17		
1	MUL	F4	F0	F2	2	11...14	15
1	S.D	F4	0	R1	3	16	17
2	L.D	F0	0	R1	6	10	11
2	MUL	F4	F0	F2	7	12...15	16
2	S.D	F4	0	R1	8	17	18
3	L.D	F0	0	R1	11	12	13
3	MUL	F4	F0	F2	16	17..	

Reg. file		Clock
	Q_i	17
F0	0	
F2	0	R1
F4	M1	value
F6		64
F8		
F10		

Load buffers

	Busy	Address
L1	0	
L2	0	
L3	0	

Store buffers

	Busy	Add.	V	Q
S 1				
S 2				
S	1	64		M1

Time

remaining		Busy	op	vj	Vk	Qj	Qk	A
	A1	0						
	A2	0						
	A3	0						

		Busy	op	vj	Vk	Qj	Qk	A
2	M1	1	MUL	mem[64]	1.33			
0	M2	0					23	

And so on..

- As you noticed, issue is always in order, but execution is out of order.
- Instructions are re-ordered dynamically, executed out of order, and all types of hazards are avoided and handled.

Part 2

Loads and stores

Loads and stores special cases


- Loads and stores can execute out of order, **only** if they access different memory addresses.
- What happens when loads and stores access the same memory?

```
LD  F0, 10  (R1)
SD  F1, 12  (R2)
LD  F2, 14  (R3)
SD  F3, 16  (R4)
```

- Assume R1=10, R2=8, R3=6, R4=4
→ All of them map to the **same** address.

Out of order loads and stores

```
LD  F0, 10 (R1)
SD  F1, 12 (R2)
LD  F2, 14 (R3)
SD  F3, 16 (R4)
```



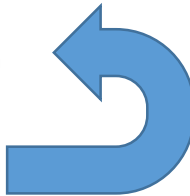
Case 1

Load is before the store and we **interchange** them:

Result: F0 would be loaded a new value rather than old → WAR hazard

Out of order loads and stores

```
LD  F0, 10 (R1)
SD  F1, 12 (R2)
LD  F2, 14 (R3)
SD  F3, 16 (R4)
```




Case 2

Store is before the load and we **interchange** them:

Result: F2 would be loaded an old value rather than new → RAW hazard

Out of order loads and stores

```
LD  F0, 10 (R1)
SD  F1, 12 (R2)
LD  F2, 14 (R3)
SD  F3, 16 (R4)
```



Case 3

Store is before the store and we **interchange** them:

Result: the memory address would get F1 at the end of the program rather than F3 → WAW hazard

Out of order loads and stores

Solution for the Loads :

Check if there are any uncompleted stores that precedes the load in program order with the same effective address.

```
LD F0, 10 (R1)
SD F1, 12 (R2)
LD F2, 14 (R3)
SD F3, 16 (R4)
```



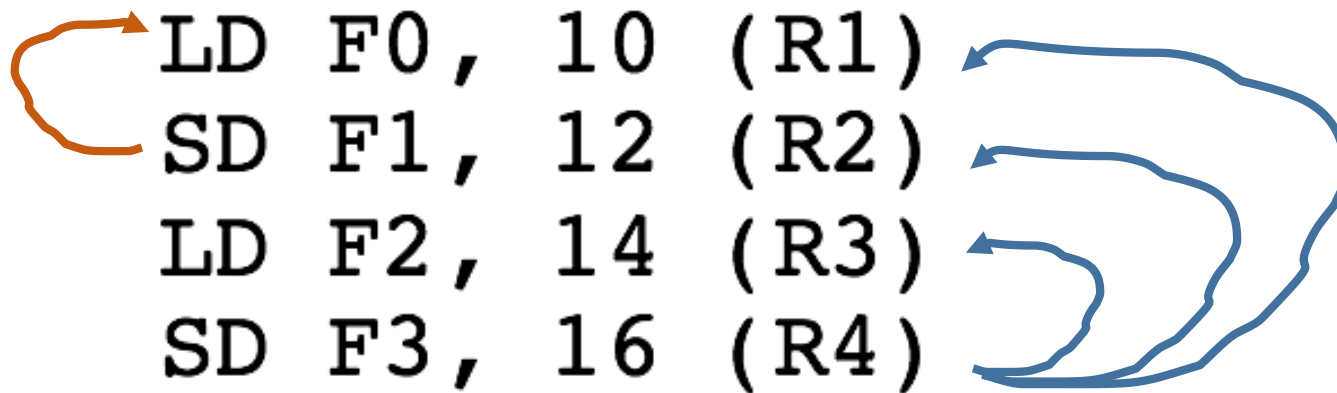
This load
must wait
for the
preceding
store.

Out of order loads and stores

Solution for the stores:

Wait until uncompleted loads and stores that precede in program order and share the same effective address.

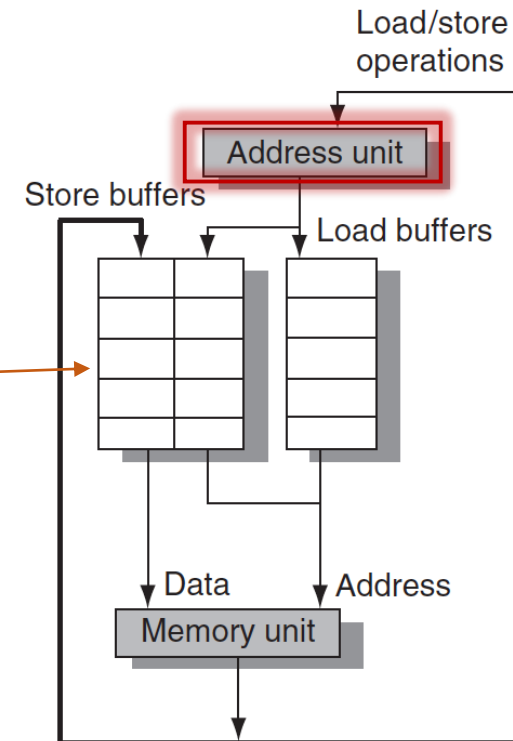
This store must wait for the preceding load.



This store must wait for all these.

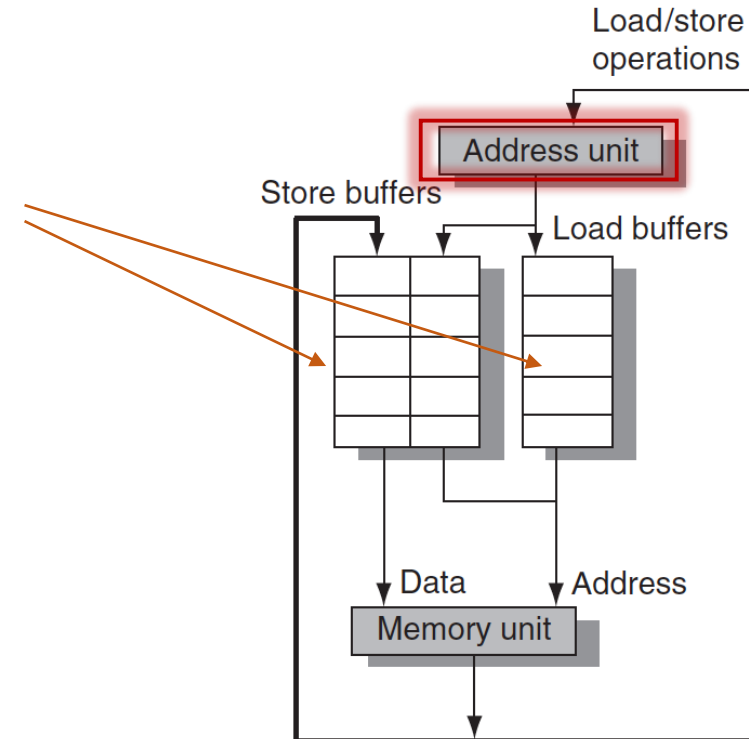
Implementations: **load**

- Effective address calculations must be done in program order using the address unit.
- When a **load** finishes the effective address calculation, it's not issued right away but the **A** field of all the active stores in the **store buffer** are checked.
- If there's a conflict (same address), the load is not issued to the load buffer.



Implementations: **store**

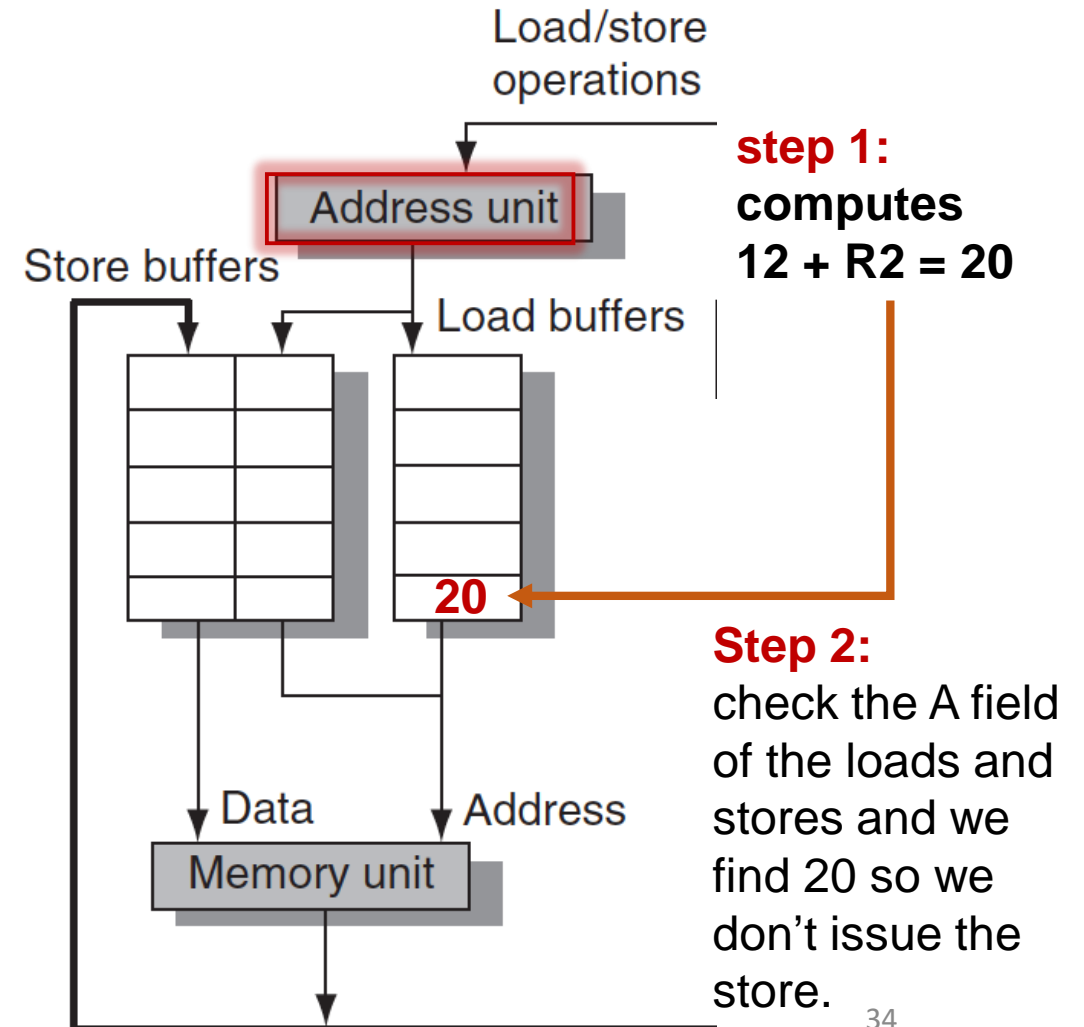
- When a store finishes the effective address calculation, it's not issued right away but the **A** field of all the active stores **and** loads in the **store and load buffer** are checked.
- If there's a conflict (same address), the store is not issued to the store buffer.



Example

```
LD F0, 10 (R1)
SD F1, 12 (R2)
```

- Assume load was issued but not executed yet so it's in the load buffer and its A field is = 20
- And we want to issue the store.



Tomasulo in practice

- Tomasulo's scheme was unused for many years after the 360/91, but was widely adopted in multiple-issue processors starting in the 1990s for several reasons:
 1. Although Tomasulo's algorithm was designed before caches, the presence of caches, with the inherently unpredictable delays, has become one of the major motivations for dynamic scheduling.

Out-of-order execution allows the processors to continue executing instructions **while awaiting the completion of a cache miss**, thus hiding all or part of the cache miss penalty.

Tomasulo in practice

2. As processors became more aggressive in their issue capability and designers are concerned with the performance of difficult-to-schedule code, techniques such as register renaming, dynamic scheduling, and speculation became more important.
3. It can achieve high performance without requiring the compiler to target code to a specific pipeline structure, a valuable property in the era of shrink wrapped mass market software.

***Many modern processors implement dynamic scheduling schemes that are derivative of Tomasulo's original algorithm, including popular Intel x86-64 chips.**

End of lecture 13