Computer Architecture

Dynamic Scheduling: Register Renaming, Tomasulo

Debadatta Mishra, CSE, IITK

Scoreboard: issues (recap)

- Provides dynamic scheduling capability.
- In-order issue, out-of-order execution and completion
- Supports scheduling/dispatching instructions out-of-order

1. Instruction window limited by size of basic blocks
2. Locking FUs before resolving RAW dependencies
3. Stalls to handle WAR and WAW
4. No support for forwarding through bypass network

Dynamic scheduling: Tomasulo

- Provides dynamic scheduling capability.
- In-order issue, out-of-order execution and completion
- Supports scheduling/dispatching instructions out-of-order

1	Instruction window limited by size of basic blocks	Lays the foundation (+one step)
2	Locking FUs before resolving RAW dependencies	Solved using reservation stations
3	Stalls to handle WAR and WAW	Register renaming through using RS
4	No support for forwarding through bypass network	Broadcast using common data bus

Register renaming

```
I1: add r1,r2,r4
I2: sub r5,r1,r7
I3: sw r5,100(r3)
I4: or r7,r8,r9
I5: mul r5,r8,r10
```

- What are the dependencies and possible hazards?

Register renaming

```
I1: add r1,r2,r4
I2: sub r5,r1,r7
I3: sw r5,100(r3)
I4: or r7,r8,r9
I5: mul r5,r8,r7
```

- What are the dependencies and possible hazards?
- True dependences (RAW): `{add,sub}`, `{sub,sw}` and `{or,mul}`
- Anti-dependences (WAR): `{sub,or}` and `{sw,mul}`
- Output dependencies: `{sub,mul}`

- Register renaming can solve hazards by renaming all destination registers
- All source operands of subsequent instructions matching the renamed register renamed
- Can be effectively done with additional registers maintaining the usage status

Register renaming

```
I1: add r1,r2,r4          I1: add r1,r2,r4
I2: sub r5,r1,r7          I2: sub r11,r1,r7
I3: sw r5,100(r3)         I3: sw r5,100(r3)
I4: or r7,r8,r9           I4: or r12,r8,r9
I5: mul r5,r8,r7          I5: mul r5,r8,r7
```

- Assume two additional registers r11 and r12
- Register r5 and r7 are renamed to r11 and r12, respectively. WAR and WAW taken care of
- Is there any issues with the renaming?

Register renaming (static)

```
I1: add r1,r2,r4        I1: add r1,r2,r4
I2: sub r5,r1,r7        I2: sub r11,r1,r7
I3: sw r5,100(r3)       I3: sw r11,100(r3)
I4: or r7,r8,r9         I4: or r12,r8,r9
I5: mul r5,r8,r7        I5: mul r5,r8,r12
```

- Assume two additional registers r11 and r12
- Register r5 and r7 are renamed to r11 and r12, respectively. WAR and WAW taken care of
- Is there any issues with the renaming? Yes, True dependencies must remain the same (for correctness)

Register renaming (dynamic)

Register Alias Table (RAT)

r1	p11
r2	p32
r3	p7
…	…
r16	p7

Physical Registers

RegID	Value
p1	0x123
p2	0x0
…	…
p32	0x200

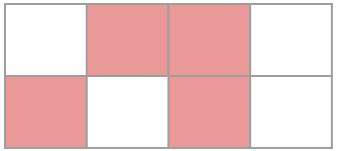
Register Status

- Register alias table (a.k.a rename table) is used to access the registers from the register file (physical registers)
- Register status is used to find unused registers

Register renaming example (dynamic)

Register Alias Table (RAT)

r1	p2
r2	p3
r3	p5
r4	p7



Register Status

Physical Registers

RegID	Value
p1	0x0
p2	0x1000
p3	0xAB00
p4	0x200
p5	0xA
p6	0x0
p7	0x98
p8	0x0

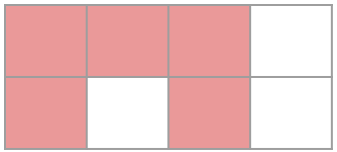
```
add r1,r2,r4
sub r2,r1,r2
mul r1,r2,r3
```

- Four architectural registers and eight physical registers

Register renaming example (dynamic)

Register Alias Table (RAT)

r1	p1
r2	p3
r3	p5
r4	p7



Register Status

Physical Registers

RegID	Value
p1	0x0
p2	0x1000
p3	0xAB00
p4	0x200
p5	0xA
p6	0x0
p7	0x98
p8	0x0

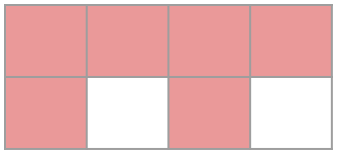
```
add r1,r2,r4    add p1,p3,p7
sub r2,r1,r2
mul r1,r2,r3
```

- Mapping of r1 is changed r1 ⟶ p2 to r1 ⟶ p1
- Free p2 ?

Register renaming example (dynamic)

Register Alias Table (RAT)

r1	p1
r2	**p4**
r3	p5
r4	p7



Register Status

Physical Registers

RegID	Value
p1	0x0
p2	0x1000
p3	0xAB00
p4	0x200
p5	0xA
p6	0x0
p7	0x98
p8	0x0

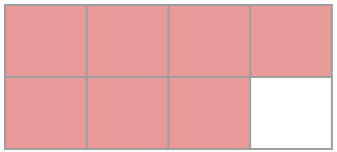
```
add r1,r2,r4    add p1,p3,p7
sub r2,r1,r2    sub p4,p1,p3
mul r1,r2,r3
```

- Mapping of r2 is changed r2 ⟶ p3 to r2 ⟶ p4
- Source operands replaced with mapping before changing the dest. mapping

Register renaming example (dynamic)

Register Alias Table (RAT)

r1	p6
r2	p4
r3	p5
r4	p7



Register Status

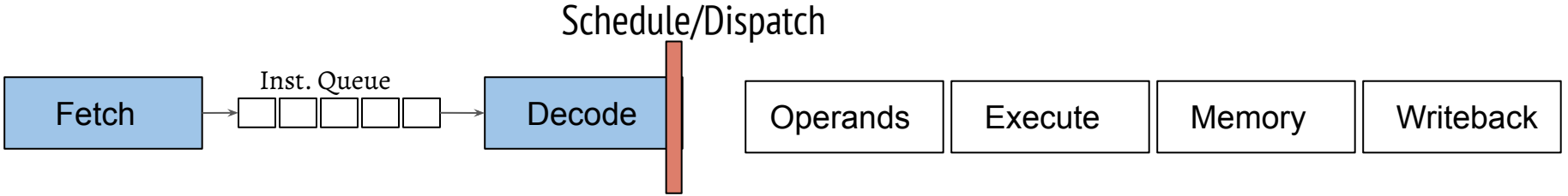
Physical Registers

RegID	Value
p1	0x0
p2	0x1000
p3	0xAB00
p4	0x200
p5	0xA
p6	0x0
p7	0x98
p8	0x0

```
add r1,r2,r4    add p1,p3,p7
sub r2,r1,r2    sub p4,p1,p3
mul r1,r2,r3    mul p6,p4,p5
```

- RAW preserved, WAR and WAW eliminated
- When and how to free registers i.e., update register status?

Dynamic Scheduling: Overall scheme

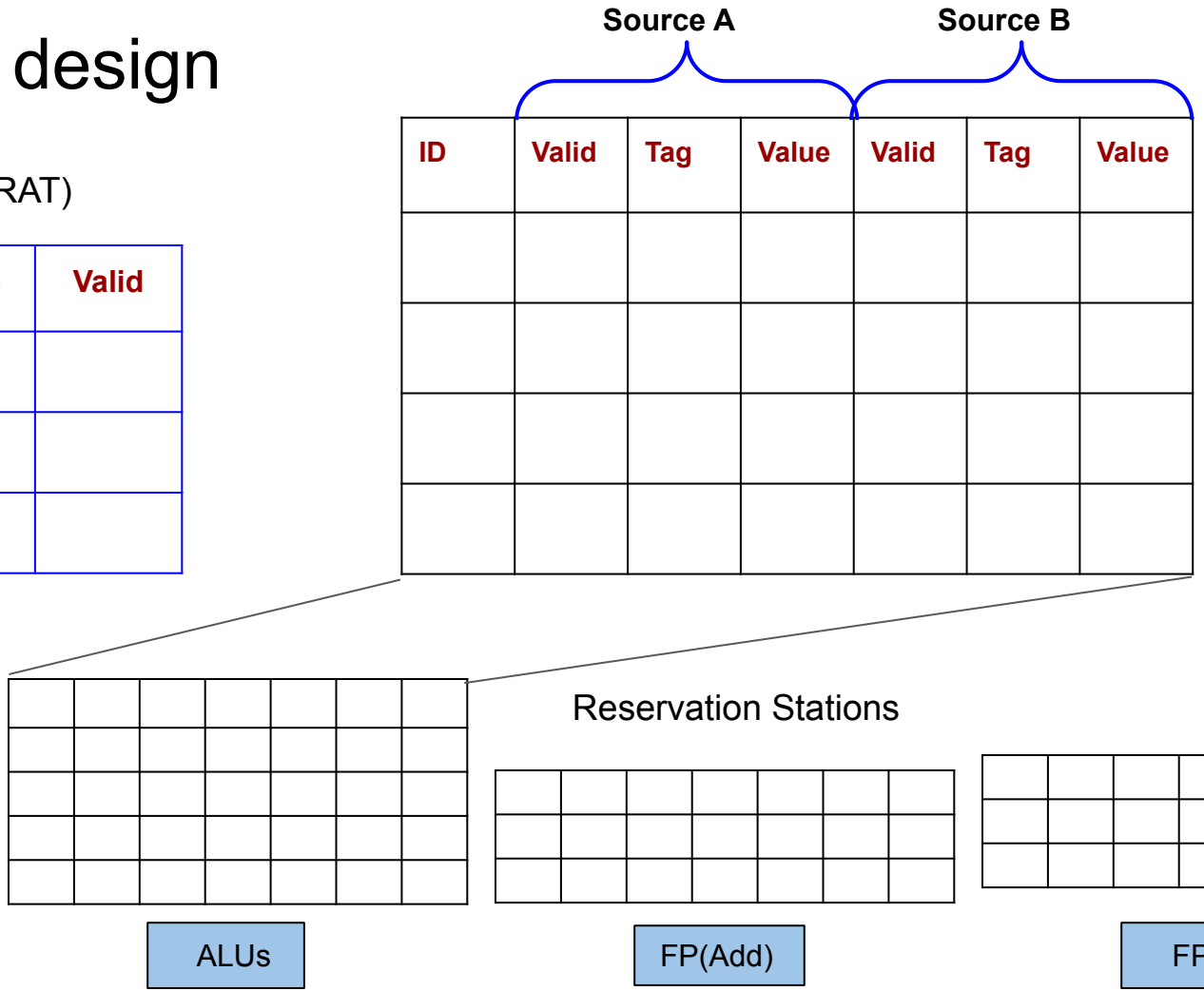


- Considering the five stage pipeline, in which stage, the dynamic scheduling should take place and why? Decode; dependency can be determined only after decoding the instructions
- What are the changes to the pipeline structure? Decode and RF access should be two different stages. Typically, an instruction queue between the fetch and decode.
- Tomasulo uses register renaming using reservation stations (RS)
- Data forwarding through a common data bus (CDB)

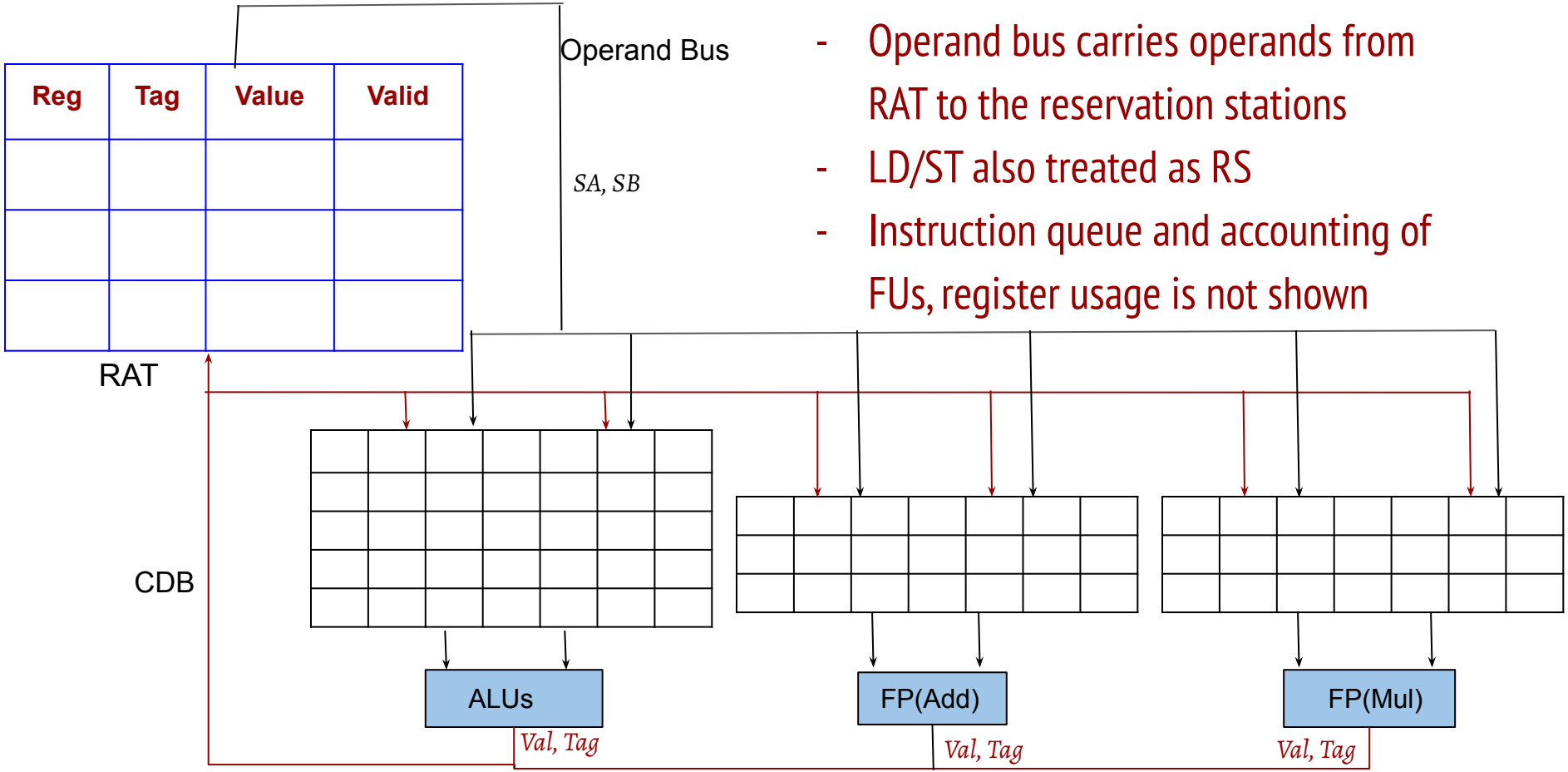
Tomasulo's design

Register Alias Table (RAT)

Reg	Tag	Value	Valid

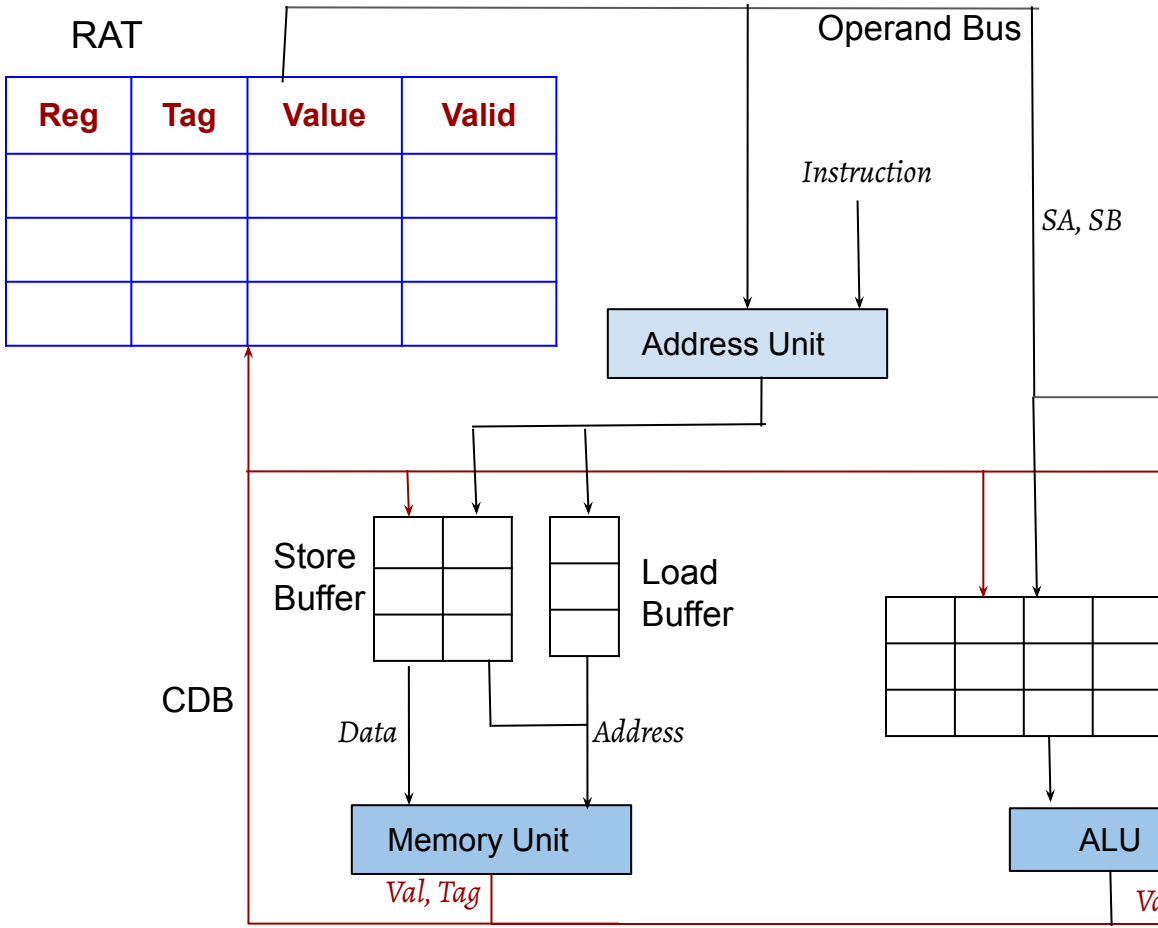


Tomasulo's design



- Operand bus carries operands from RAT to the reservation stations
- LD/ST also treated as RS
- Instruction queue and accounting of FUs, register usage is not shown

Tomasulo's design (with load/store)



- Load/store buffers are special reservation station
- Assumption: Address unit serializes LD/ST

Tomasulo Algorithm

- Issue
  - Pick the instruction from the instruction queue and check for structural hazard
  - Issue the instruction to the matching RS with the source operands
  - Destination operand in the RAT is assigned the tag of reservation station

Tomasulo Algorithm

- Issue
  - Pick the instruction from the instruction queue and check for structural hazard
  - Issue the instruction to the matching RS with the source operands
  - Destination operand in the RAT is assigned the tag of reservation station
- Execute
  - If operands are available, the FU starts execution (multiple instructions can be ready)
  - For load and store, the address unit is used to calculate the effective address before placing the load/store request in the load/store buffers

Tomasulo Algorithm

- Issue
  - Pick the instruction from the instruction queue and check for structural hazard
  - Issue the instruction to the matching RS with the source operands
  - Destination operand in the RAT is assigned the tag of reservation station
- Execute
  - If operands are available, the FU starts execution (multiple instructions can be ready)
  - For load and store, the address unit is used to calculate the effective address before placing the load/store request in the load/store buffers
- Write result
  - When a FU finishes execution, it broadcasts the tag and value in the CDB.
  - The RAT and RS compare the tag and update the value with the broadcasted value

Example

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- 3 FP Add/Sub, 2 FP Mul/Div
- Execution time in cycles: Integer Op = 1, FP add = 2, FP multiply = 10 and FP divide = 40

Example: Cycle 0

Inst	Issue	Execute	WriteRes

AU	Tag	Operands
Ad		

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	0					
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0			
f2			
f4			
f6			
f8			
f10			

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

				I1

Instruction Queue

Initially, all reservation stations are empty. Assuming two load buffers, one addr. unit

Tracking usage of reservation stations not shown

Example: Cycle 1

Inst	Issue	Execute	WriteRes
I1	1		

AU	Tag	Operands
Ad	L1	r2 , imm

Load Buff	Tag	Address
1	L1	INVAL
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	0					
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	0	F0	1
f2	0	F2	1
f4	0	F4	1
f6	L1	F6	0
f8	0	F8	1
f10	0	F10	1

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

			I2	I1

Instruction Queue

I1 issued, f6 tag is updated in RAT

Load buffer entry (L1) is locked

EA becomes valid in the next CC

Example: Cycle 2

Inst	Issue	Execute	WriteRes
I1	1	2(R1)	
I2	2		

AU	Tag	Operands
Ad	L2	r3 , imm

Load Buff	Tag	Address
1	L1	r2+34
2	L2	INVAL

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	0					
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	0	F0	1
f2	L2	F2	0
f4	0	F4	1
f6	L1	F6	0
f8	0	F8	1
f10	0	F10	1

I2 issued, f2 tag updated in RAT

First load (I1) is ready for MU

MU will load in the next CC

			I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Example: Cycle 3

Inst	Issue	Execute	WriteRes
I1	1	3	
I2	2	3(R1)	
I3	3		

AU	Tag	Operands

Load Buff	Tag	Address
1	L1	r2+34
2	L2	r3+45

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0	L2	-	1	-	F4
M2	0					
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	L2	F2	0
f4	0	F4	1
f6	L1	F6	0
f8	0	F8	1
f10	0	F10	1

		I3	I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

I1 finished loading from memory unit. Result will be broadcasted in the next CC

Second load (I2) is ready for MU

I3 issued, op-B (f4) is copied, op-A (f2) is not ready yet

Example: Cycle 4

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	
I3	3		
I4	4		

AU	Tag	Operands

Load Buff	Tag	Address
1		
2	L2	r3+45

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0	L2	-	1	-	F4
M2	0					
A1	0	L1	-	0	L2	-
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	L2	F2	0
f4	0	F4	1
f6	L1	IF6	0
f8	A1	F8	0
f10	0	F10	1

I1 writes result i.e., loaded value broadcasted over CDB, tag match and value updation in RAT and RS

Second load (I2) data is loaded

I4 issued, op-A becomes available in the next CC and op-B (f2) is not ready yet

	I4	I3	I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Example: Cycle 5

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3		
I4	4		
I5	5		

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0	L2	-	1	-	F4
M2	0	M1	-	1	-	IF6
A1	1	-	IF6	0	L2	-
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	L2	IF2	0
f4	0	F4	1
f6	0	IF6	1
f8	A1	F8	0
f10	M2	F10	0

Loaded value (I2) is broadcasted over CDB in this cycle. I3 and I4 RS entries updated and ready in next CC

I5 issued, op-A is not ready (M1), op-B (f6) is copied to RS

	I5	I4	I3	I2

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Example: Cycle 6

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	6(R9)	
I4	4	6(R1)	
I5	5		
I6	6		

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	1	-	IF6	1	-	IF2
A2	0	A1	-	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	IF6	0
f8	A1	F8	0
f10	M2	F10	0

I3 and I4 start execution during this cycle (R* shows remaining)

I6 issued, op-A is not ready (A1), op-B (f2) is copied to RS

	I6	I5	I4	I3

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Example: Cycle 7

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	7(R8)	
I4	4	7	
I5	5		
I6	6		

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	1	-	IF6	1	-	IF2
A2	0	A1	-	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	IF6	0
f8	A1	F8	0
f10	M2	F10	0

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

	I6	I5	I4	I3

Instruction Queue

I4 finish execution in the cycle

Example: Cycle 8

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	8(R7)	
I4	4	7	8
I5	5		
I6	6		

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	1	-	IF6	1	-	IF2
A2	0	A1	-	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	IF6	0
f8	A1	SF8	0
f10	M2	F10	0

	I6	I5	I4	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

I4 broadcasts the result (SF8) with tag = A1 over the CDB. The values of matching tag in RS and RAT updated in this CC

Example: Cycle 9

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	9(R6)	
I4	4	7	8
I5	5		
I6	6	9(R1)	

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	0					
A2	1	-	SF8	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	IF6	0
f8	0	SF8	1
f10	M2	F10	0

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

		I6	I5	I3

Instruction Queue

I6 can now start execution

Example: Cycle 10

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	10(R5)	
I4	4	7	8
I5	5		
I6	6	10	

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	0					
A2	1	-	SF8	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	IF6	0
f8	0	SF8	1
f10	M2	F10	0

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

		I6	I5	I3

Instruction Queue

I6 finish execution in this CC

Example: Cycle 11

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	11(R4)	
I4	4	7	8
I5	5		
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	0					
A2	1	-	SF8	1	-	IF2

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	A2	AF6	0
f8	0	SF8	1
f10	M2	F10	0

		I6	I5	I3

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Result of I6 (AF6) broadcasted over CDB. RAT updated

Example: Cycle 12

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	12(R3)	
I4	4	7	8
I5	5		
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

			I5	I3

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

I3 is still in execution, I5 still waiting in the RS

Example: Cycle 15

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	
I4	4	7	8
I5	5		
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	-	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	F0	0
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

			I5	I3

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

I3 finishes execution

Example: Cycle 16

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5		
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	1	-	IF2	1	-	F4
M2	0	M1	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	M1	MF0	0
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

			I5	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Results from M1 (MF0) is broadcasted over the CDB. RAT and RS are updated in this CC

Example: Cycle 17

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	17(39)	
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	1	-	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	0	MF0	1
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

				I5

Instruction Queue

I5 starts execution in this cycle

Example: Cycle 56

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	56	
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	1	-	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	0	MF0	1
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

				I5

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

I5 finish execution in this CC

Example: Cycle 57

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	56	57
I6	6	10	11

AU	Tag	Operands

Load Buff	Tag	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2	1	-	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	Tag	Value	Valid
f0	0	MF0	1
f2	0	IF2	1
f4	0	F4	1
f6	0	AF6	1
f8	0	SF8	1
f10	M2	F10	0

				I5

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Results of I5 (DF6) broadcasted with tag M2. RAT updated

Tomasulo: Observations

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	56	57
I6	6	10	11

- Issue is in-order, execution and commit (write back) are out-of-order
- FU is locked only when all operands are ready
- How WAW hazards handled?
- How to handle memory dependencies?

Tomasulo: Observations

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	56	57
I6	6	10	11

- Issue is in-order, execution and commit (write back) are out-of-order
- FU is locked only when all operands are ready
- How WAW hazards handled? During issue, tag of the destination register is updated to the RS ID of the latter instruction. Thus, completion of former instruction out of order will not update the register value
- How to handle memory dependencies?

Tomasulo: Observations

Inst	Issue	Execute	WriteRes
I1	1	3	4
I2	2	4	5
I3	3	15	16
I4	4	7	8
I5	5	56	57
I6	6	10	11

- Issue is in-order, execution and commit (write back) are out-of-order
- FU is locked only when all operands are ready
- How WAW hazards handled? During issue, tag of the destination register is updated to the RS ID of the latter instruction. Thus, completion of former instruction out of order will not update the register value
- How to handle memory dependencies?

Tomasulo: Handling memory dependencies

- Load and store can be performed OoO, if they are performed on disjoint addresses
- Load-Store dependencies are not known till effective address is calculated
- Approach 1: Allow load only if no stores are issued ⟶ Significant performance loss

Tomasulo: Handling memory dependencies

- Load and store can be performed OoO, if they are performed on disjoint addresses
- Load-Store dependencies are not known till effective address is calculated
- Approach 1: Allow load only if no stores are issued ⟶ Significant performance loss
- Approach 2: Allow issue of loads and stores, serialize after address calculation
  - Check the effective address of load in the store buffer
  - If there is a matching entry, stall the load till the store completes
  - Note that, the matching store may be dependent on other instructions

Tomasulo: Handling memory dependencies

- Load and store can be performed OoO, if they are performed on disjoint addresses
- Load-Store dependencies are not known till effective address is calculated
- Approach 1: Allow load only if no stores are issued ⟶ Significant performance loss
- Approach 2: Allow issue of loads and stores, serialize after address calculation
  - Check the effective address of load in the store buffer
  - If there is a matching entry, stall the load till the store completes
  - Note that, the matching store may be dependent on other instructions
- Approach 3: Allow issue of load stores, forward result from store buffer to load buffer
  - Similar to the previous approach, but why wait till store completes?
  - Load may read the data from the store buffer. Challenges?

Tomasulo: Handling memory dependencies

- Load and store can be performed OoO, if they are performed on disjoint addresses
- Load-Store dependencies are not known till effective address is calculated
- Approach 1: Allow load only if no stores are issued ⟶ Significant performance loss
- Approach 2: Allow issue of loads and stores, serialize after address calculation
  - Check the effective address of load in the store buffer
  - If there is a matching entry, stall the load till the store completes
  - Note that, the matching store may be dependent on other instructions
- Approach 3: Allow issue of load stores, forward result from store buffer to load buffer
  - Similar to the previous approach, but why wait till store completes?
  - Load may read the data from the store buffer. Challenges? Address matching, alignment checks, handling partial matches