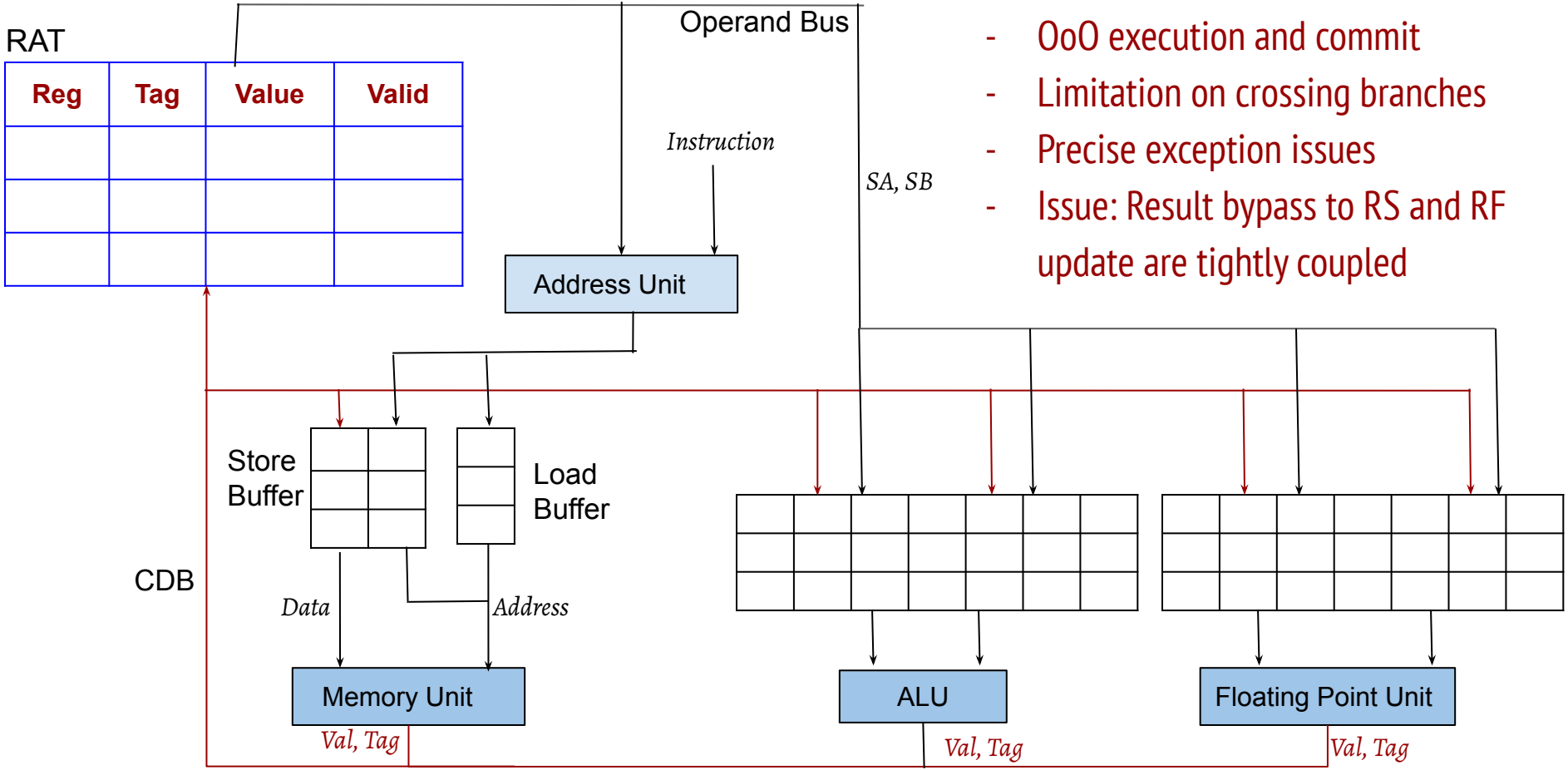
Computer Architecture

Speculative Execution

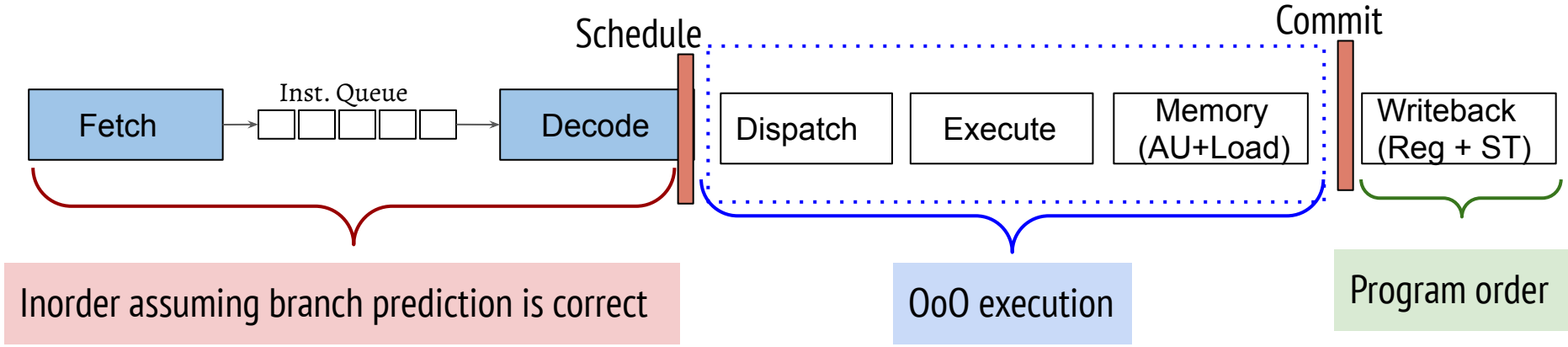
Debadatta Mishra, CSE, IITK

Tomasulo's design (recap)



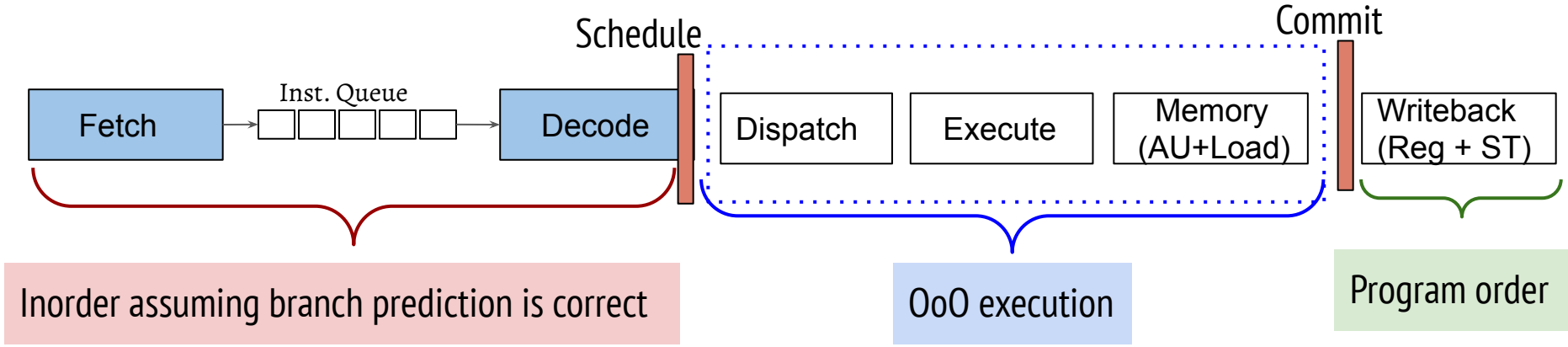
- OoO execution and commit
- Limitation on crossing branches
- Precise exception issues
- Issue: Result bypass to RS and RF update are tightly coupled

Speculative execution



- Instructions are executed speculatively out-of-order maintaining the data dependencies
- Architectural visible state is updated only during instruction commit
- All commits happen in program execution order ⇒ Require some form of ordering (queue like DS)

Speculative execution



- Instructions are executed speculatively out-of-order maintaining the data dependencies
- Architectural visible state is updated only during instruction commit
- All commits happen in program execution order ⇒ Require some form of ordering (queue like DS)
- On a branch misprediction or exception, only the instructions before the branch/exception are committed; speculative execution states are flushed; instruction fetch from the new PC

Reorder Buffer (ROB)

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw  r5,100(r3)
BB3:  or  r7,r8,r9
chk:   beq r5,r7,#loop
```

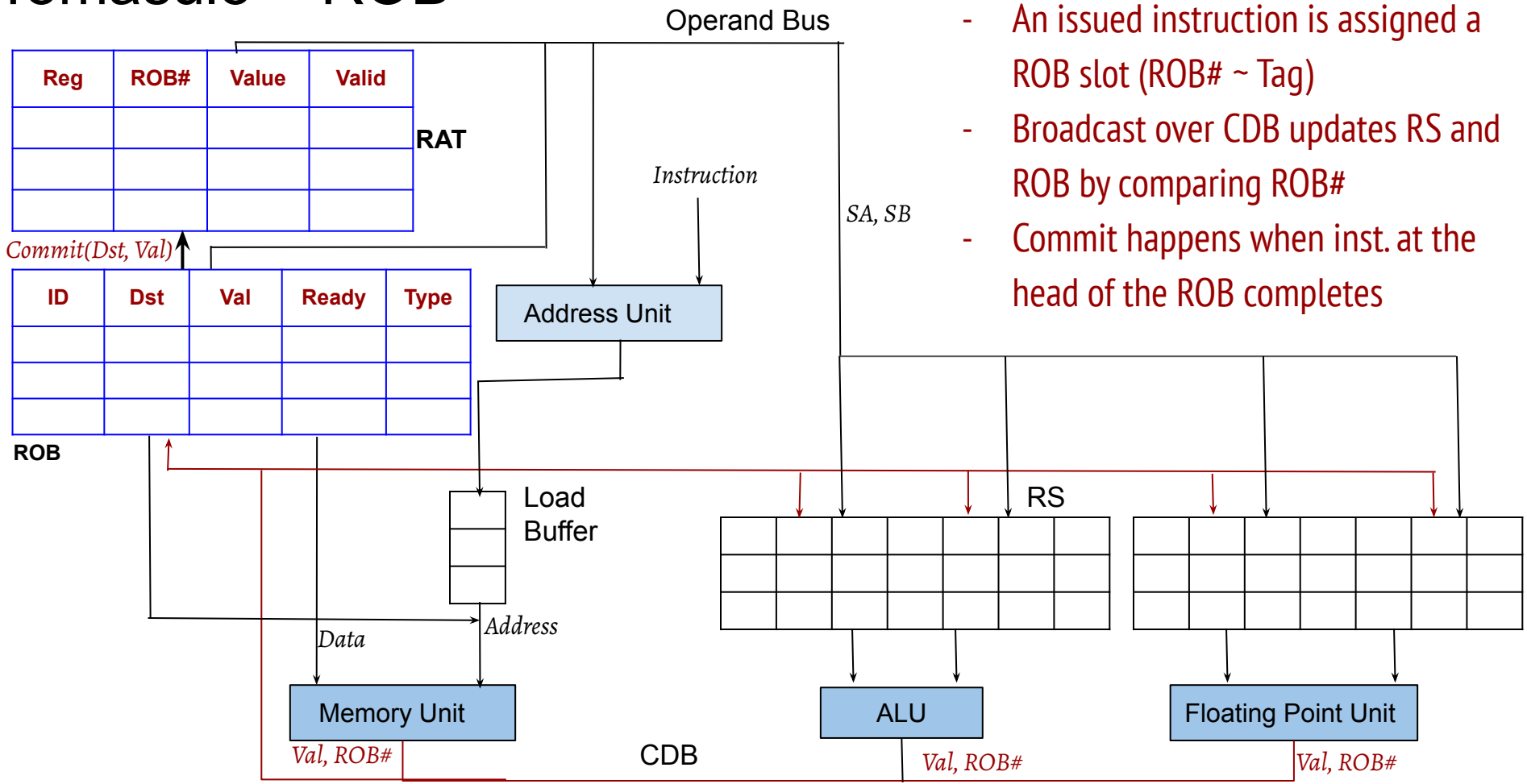
Commit

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R1	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(ALU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6						
7						

- Each entry in the ROB corresponds to one instruction ⇒ Window is determined by ROB size
- ROB entry contains the status of the destination register and information regarding the instruction
- If the the destination is ready, all subsequent instructions can get their source operand from the ROB

Tomasulo + ROB



- An issued instruction is assigned a ROB slot (ROB# ~ Tag)
- Broadcast over CDB updates RS and ROB by comparing ROB#
- Commit happens when inst. at the head of the ROB completes

Tomasulo + ROB Algorithm

- Issue
  - Issue the instruction if both RS and ROB has free slots, stall otherwise
  - Send the operands to RS from register table (if valid) or from ROB (if ready)
  - If any operand is not ready in ROB, use the ROB entry ID as the tag in RS

Tomasulo + ROB Algorithm

- Issue
  - Issue the instruction if both RS and ROB has free slots, stall otherwise
  - Send the operands to RS from register table (if valid) or from ROB (if ready)
  - If any operand is not ready in ROB, use the ROB entry ID as the tag in RS
- 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

Tomasulo + ROB Algorithm

- Issue
  - Issue the instruction if both RS and ROB has free slots, stall otherwise
  - Send the operands to RS from register table (if valid) or from ROB (if ready)
  - If any operand is not ready in ROB, use the ROB entry ID as the tag in RS
- 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
- Write result
  - When a FU finishes execution, it broadcasts the ROB entry tag and value in the CDB.
  - ROB and RS compare the tag and update the stored value with the broadcasted value

Tomasulo + ROB Algorithm

- Issue
  - Issue the instruction if both RS and ROB has free slots, stall otherwise
  - Send the operands to RS from register table (if valid) or from ROB (if ready)
  - If any operand is not ready in ROB, use the ROB entry ID as the tag in RS
- 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
- Write result
  - When a FU finishes execution, it broadcasts the ROB entry tag and value in the CDB.
  - ROB and RS compare the tag and update the stored value with the broadcasted value
- Commit (from the head of ROB)
  - For normal (non-speculative) instruction, register table or memory (for ST) is updated
  - When a branch instruction with misprediction commits, speculative insts are flushed

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	Dst	Value	State

ROB

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	ROB#	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 RS and ROB are empty. Assuming two load buffers, one addr. unit

Tracking usage of reservation stations, ROB not shown

Example: Cycle 1

Inst	Dst	Value	State
I1	f6	INV	Issue

ROB

AU	ROB#	Operands
Ad	1	r2 , imm

Load Buff	ROB#	Address
1	1	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	ROB#	Value	Valid
f0			1
f2			1
f4			1
f6	1	-	0
f8			1
f10			1

			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 issued, f6 tag in RAT = ROB #1

Load buffer entry (L1) is locked

EA becomes valid in the next CC

Example: Cycle 2

Inst	Dst	Value	State
I1	f6	INV	Ex(AU)
I2	f2	INV	Issue

ROB

AU	ROB#	Operands
Ad	2	r2 , imm

Load Buff	ROB#	Address
1	1	r2+34
2	2	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	ROB#	Value	Valid
f0		FO	1
f2	2	F2	0
f4		F4	1
f6	1	F6	0
f8		F8	1
f10		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
```

I2 issued, f2 tag updated in RAT

First load (I1) is ready for MU

MU will load in the next CC

Example: Cycle 3

Inst	Dst	Value	State
I1	f6	INV	Mem
I2	f2	INV	Ex(AU)
I3	f0	INV	Issue

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1	1	r2+34
2	2	r3+45

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	0	2	-	1	-	F4
M2	0					
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	2	F2	0
f4		F4	1
f6	1	F6	0
f8		F8	1
f10		F10	1

	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
```

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) points to ROB #2

Example: Cycle 4

Inst	Dst	Value	State
I1	f6	IF6	WrRes
I2	f2	INV	Mem
I3	f0	INV	Ex(M1)
I4	f8	INV	Issue

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
2	2	r3+45

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	0	2	-	1	-	F4
M2	0					
A1(4)	1	-	IF6	0	2	-
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	2	F2	0
f4		F4	1
f6	1	F6	0
f8	4	F8	0
f10		F10	1

I5	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
```

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

Example: Cycle 5

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	WrRes
I3	f0	INV	Ex(M1)
I4	f8	INV	Ex(A1)
I5	f10	INV	Issue

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	0	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1(4)	1	-	IF6	0	2	IF2
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	2	F2	0
f4		F4	1
f6	-	IF6	1
f8	4	F8	0
f10	5	F10	0

I6	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
```

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 (ROB#3), op-B (f6) is copied

I1 commits, updates RAT

Example: Cycle 6

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r9	Ex(M1)
I4	f8	INV-r1	Ex(A1)
I5	f10	INV	Ex(M2)
I6	f6	INV	Issue

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1(4)	1	-	IF6	1	2	IF2
A2(6)	0	4	-	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

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

I6 issued, op-A is not ready (ROB #4), op-B (f2) is copied to RS

Example: Cycle 7

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r8	Ex(M1)
I4	f8	INV-r0	Ex(A1)
I5	f10	INV	Ex(M2)
I6	f6	INV	Ex(A2)

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1(4)	1	-	IF6	1	2	IF2
A2(6)	0	4	-	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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 finish execution in the cycle (ROB entry tag 4)

Example: Cycle 8

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r7	Ex(M1)
I4	f8	SF8	WrRes
I5	f10	INV	Ex(M2)
I6	f6	INV	Ex(A2)

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1(4)	1	-	IF6	1	2	IF2
A2(6)	0	4	SF8	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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 ROB# tag = 4 over the CDB. The values of matching tag in RS and ROB updated in this CC.

I6 will start execution next cycle

Example: Cycle 9

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r6	Ex(M1)
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	INV-r1	Ex(A2)

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1	0					
A2(6)	1	-	SF8	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

I6 starts execution in this CC

I4 can not be committed

Example: Cycle 10

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r5	Ex(M1)
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	INV-r0	Ex(A2)

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1	0					
A2(6)	1	-	SF8	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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

Example: Cycle 11

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r4	Ex(M1)
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	AF6	WrRes

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1	0					
A2(6)	1	-	SF8	1	-	IF2

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

Result of I6 (AF6) broadcasted over CDB with ROB #6. ROB updated.

Example: Cycle 12

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r3	Ex(M1)
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	AF6	Done

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

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

I6 can not be committed

Example: Cycle 15

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	INV-r0	Ex(M1)
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	AF6	Done

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	-	1	-	IF6
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

I3 finishes execution

Example: Cycle 16

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	MF0	WrRes
I4	f8	SF8	Done
I5	f10	INV	Ex(M2)
I6	f6	AF6	Done

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1(3)	1	2	IF2	1	-	F4
M2(5)	0	3	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	3	FO	0
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	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
```

Results from M1 (MF0) is broadcasted over the CDB with ROB #3. ROB and RS are updated in this CC

Example: Cycle 17 (commit width = 1)

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	MF0	CMT(17)
I4	f8	SF8	Done
I5	f10	INV-r39	Ex(M2)
I6	f6	AF6	Done

ROB

AU	ROB#	Operands

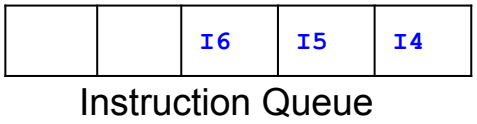
Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2(5)	0	3	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	-	MF0	1
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	4	F8	0
f10	5	F10	0



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
```

If commit width = 1, only one instruction can be committed.

#of WR ports in the RF depends on the commit width

Example: Cycle 17 (commit width > 1)

Inst	Dst	Value	State
I1	f6	IF6	CMT(5)
I2	f2	IF2	CMT(6)
I3	f0	MF0	CMT(17)
I4	f8	SF8	CMT(17)
I5	f10	INV-r39	Ex(M2)
I6	f6	AF6	Done

ROB

AU	ROB#	Operands

Load Buff	ROB#	Address
1		
2		

RS

RSID	A_Valid	A_T	A_Val	B_Valid	B_T	B_Val
M1	0					
M2(5)	0	3	MF0	1	-	IF6
A1	0					
A2	0					

RAT

Reg	ROB#	Value	Valid
f0	-	MF0	1
f2	-	IF2	1
f4		F4	1
f6	6	IF6	0
f8	-	SF8	1
f10	5	F10	0

			I6	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
```

If commit width >=2, both I3 and I4 can commit in the same cycle

With ROB, Tomasulo can perform speculative execution and support precise exception

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw r5,100(r3)
BB3:  or r7,r8,r9
chk:   beq r5,r7,#loop
```

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(ALU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	-	N	N	Ex(ALU)	BB1,ALU

- Assuming the branch predictor predicts "taken", what will be the value of r1 at the ALU while executing the "sub" instruction (ROB entry #7)?

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw r5,100(r3)
BB3:  or r7,r8,r9
chk:   beq r5,r7,#loop
```

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(ALU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	-	N	N	Ex(ALU)	BB1,ALU

- Assuming the branch predictor predicts "taken", what will be the value of r1 at the ALU while executing the "sub" instruction? #R12

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw r5,100(r3)
BB3:  or r7,r8,r9
chk:   beq r5,r7,#loop
```

Assume correct branch prediction (taken). With in-order commit, will there be any issues?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	{#R52}	Y	N	Complete	BB2,ST

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw r5,100(r3)
BB3:  or r7,r8,r9
chk:   beq r5,r7,#loop
```

Assume correct branch prediction (taken). With in-order commit, will there be any issues?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	{#R52}	Y	N	Complete	BB2,ST

- When BB1 (ROB #2) completes and forwards result in the bypass network, BB2(ROB #3) should use that value (not #R52 from ROB #7). Why not? What is the fix?

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw r5,100(r3)
BB3:  or r7,r8,r9
chk:   beq r5,r7,#loop
```

Assume correct branch prediction (taken). With in-order commit, will there be any issues?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#loop	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	{#R52}	Y	N	Complete	BB2,ST

- When BB1 (ROB #2) completes and forwards the result in the bypass network, BB2(ROB #3) should use that value (not #R52!). The first store (ROB #3) should store the result sub instruction (speculative mode correctness). ROB entry for ST maintain additional tag (ROB#) to update the value to be stored

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw  r5,100(r3)
BB3:  or  r7,r8,r9
chk:  beq r5,r7,#loop
FP1:  addD f1,f2,f3
… … … … …
```

Assume branch misprediction (actually not taken). What will be the status of the ROB?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	-	N	N	!Resolve	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	{#R52}	Y	N	Complete	BB2,ST

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw  r5,100(r3)
BB3:  or  r7,r8,r9
chk:  beq r5,r7,#loop
FP1:  addD f1,f2,f3
… … … … …
```

Assume branch misprediction (actually not taken). What will be the status of the ROB?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#FP1	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	#R52	Y	N	Complete	BB2,ST
9	f1	-	N	N	Ex(FU)	FP1,FP

- All instructions executing speculatively will be discarded, How and When?

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw  r5,100(r3)
BB3:  or  r7,r8,r9
chk:  beq r5,r7,#loop
FP1:  addD f1,f2,f3
... ... ... ... ...
```

Assume branch misprediction (actually not taken). What will be the status of the ROB?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#FP1	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	#R52	Y	N	Complete	BB2,ST
9	f1	-	N	N	Ex(FU)	FP1,FP

- All instructions executing speculatively will be discarded, How and When?
- Discarding right-away does not yield a lot of benefits if we do not free RS and ROB

ROB: Hypothetical Scenario

```
loop: add r1,r2,r4
BB1:  sub r5,r1,r7
BB2:  sw  r5,100(r3)
BB3:  or  r7,r8,r9
chk:  beq r5,r7,#loop
FP1:  addD f1,f2,f3
… … … … …
```

Assume branch misprediction (actually not taken). What will be the status of the ROB?

ROB

ID#	Destination (Reg/MAddr)	Value (Reg/Mem)	Ready	Exception	Status	PC, Type and Other control
1	r1	#R11	Y	N	Complete	loop,ALU
2	r5	-	N	N	Ex(FU)	BB1,ALU
3	100+r3	-	N	N	Ex(RAW)	BB2,ST
4	r7	#R7	Y	N	Complete	BB3,ALU
5	-	#FP1	Y	N	Complete	chk,BR
6	r1	#R12	Y	N	Complete	loop,ALU
7	r5	#R52	Y	N	Complete	BB1,ALU
8	100+r3	#R52	Y	N	Complete	BB2,ST
9	f1	-	N	N	Ex(FU)	FP1,FP

- Discarding right-away does not yield a lot of benefits if we do not free RS and ROB
- Maintain information about speculative instructions, squash them when the branch inst. commits

Speculative execution with explicit register renaming

- In Tomasulo+ROB, register renaming takes place implicitly
  - Source operand values are copied to RS
  - Destination operands are allocated in the ROB

Speculative execution with explicit register renaming

- In Tomasulo+ROB, register renaming takes place implicitly
  - Source operand values are copied to RS
  - Destination operands are allocated in the ROB
- As discussed earlier, explicit register renaming requires more physical registers than architecturally visible registers
- Possible design: Two mapping tables (RATs) maintained, one to use operands from uncommitted instructions (like Tomasulo) and another to maintain the committed mapping from architectural register state to physical register state

Speculative execution with explicit register renaming

- In Tomasulo+ROB, register renaming takes place implicitly
  - Source operand values are copied to RS
  - Destination operands are allocated in the ROB
- As discussed earlier, explicit register renaming requires more physical registers than architecturally visible registers
- Possible design: Two mapping tables (RATs) maintained, one to use operands from uncommitted instructions (like Tomasulo) and another to maintain the committed mapping from architectural register state to physical register state
- Register deallocation is challenging
  - Keep track of usage at reservation stations, free when the register is unused
  - A physical register not used by RS and not mapped by architectural register can be reclaimed