Solution to Tomasulo's + Speculation Example

This example shows the operation of the reorder buffer (ROB) in a dynamically scheduled processor that allows speculation, in addition to the structures we previously discussed—reservation stations and register result status table. This solution shows the state of those areas as well as the pipeline at every point.

Recall that the major differences with speculation are: (1) instructions are issued both to reservation stations and the ROB, which ensures in-order commit, (2) renamed registers are renamed to ROB entries, not reservation stations, and (3) only the ROB is allowed to write the register file, when an instruction commits; at instruction completion, the value is broadcast to dependent reservation stations and to the ROB, not the register file.

In this example, we show two loop iterations (the code is listed in the pipeline diagram), assuming 2 cycle latencies for adds and loads and 6 cycles for multiplication. The hardware we use is shown below:

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	N						
Load2	N						
INT1	N						
INT2	N						
Mult1	N						
Mult2	N						

Register result status table

F0	F2	F4	R1	R2

Pipeline diagram

Cycle	1	2	3	4	5
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					

Cycles 1-2: The first instruction to issue is the L.D instruction, in cycle 2. This instruction fills reservation station Load1 as well as ROB entry 1. The destination register (F0) is renamed to ROB1. We also fetch the MUL.D in this cycle.

Reorder buffer

	Op	Dest	Value	Ready
1	L.D	F0		N
2				
3				
4				
5				
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D	[R1]				0
Load2	N						
Int1	N						
Int2	N						
Mult1	N						
Mult2	N						

Register result status table

F0	F2	F4	R1	R2
ROB1				

Pipeline diagram

Cycle	1	2	3	4	5
`L.D F0,0(R1)`	IF	IS			
`MUL.D F4,F0,F2`		IF			
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					

Cycle 3: The L.D calculates its address in this cycle. We issue the MUL.D to Mult1 and ROB entry 2; since this instruction depends on the load, it will stall until the load completes. The S.D is fetched.

Reorder buffer

	Op	Dest	Value	Ready
1	L.D	F0		N
2	MUL.D	F4		N
3				
4				
5				
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D					0+[R1]
Load2	N						
INT1	N						
INT2	N						
Mult1	Y	MUL.D		[F2]	ROB1		
Mult2	N						

Register result status table

F0	F2	F4	R1	R2
ROB1		ROB2		

Pipeline diagram

Cycle	1	2	3	4	5
`L.D F0,0(R1)`	IF	IS	EX		
`MUL.D F4,F0,F2`		IF	IS		
`S.D F4,0(R1)`			IF		
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					

Cycle 4: The L.D accesses memory. The S.D is issued; this instruction needs no reservation station, but it will occupy entry 3 in the ROB. Note that the destination for this instruction is an address, not a register, and that address will not be calculated until the next cycle. We do have to indicate that the value will come from ROB2. The MUL.D stalls, waiting for the L.D to complete, and the DADDIU is fetched.

Reorder buffer

	Op	Dest	Value	Ready
1	L.D	F0		N
2	MUL.D	F4		N
3	S.D		ROB2	N
4				
5				
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D					0+[R1]
Load2	N						
INT1	N						
INT2	N						
Mult1	Y	MUL.D		[F2]	ROB1		
Mult2	N						

Register result status table

F0	F2	F4	R1	R2
ROB1		ROB2		

Pipeline diagram

Cycle	1	2	3	4	5
`L.D F0,0(R1)`	IF	IS	EX	M	
`MUL.D F4,F0,F2`		IF	IS	S	
`S.D F4,0(R1)`			IF	IS	
`DADDIU R1,R1,#-8`				IF	
`BNE R1,R2,Loop`					
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					

Cycle 5: The L.D completes in this cycle, which allows the MUL.D to start execution. Note that the completion of the load clears the reservation station, but does not clear the appropriate entry in the register result status table, as the result of the load remains in the ROB until the instruction commits. The ROB entry with the L.D, however, is marked as ready to commit; as the oldest instruction in the ROB, it will commit in the next cycle. The S.D can calculate its address in this cycle—although it is dependent on the MUL.D, that value isn't necessary until the actual store takes place, when the instruction commits. The DADDIU issues to INT1 and ROB4. The BNE is fetched.

Reorder buffer

	Op	Dest	Value	Ready
1	**L.D**	**F0**	**M[0+R1]**	**Y**
2	MUL.D	F4		N
3	S.D	**0+R1**	ROB2	N
4	**DADDIU**	**R1**		**N**
5				
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	**N**						
Load2	N						
INT1	**Y**	**DADDIU**	**[R1]**	**-8**			
INT2	N						
Mult1	**Y**	**MUL.D**	**[F0]**	**[F2]**			
Mult2	N						

Register result status table

F0	F2	F4	**R1**	R2
ROB1		ROB2	**ROB4**	

Pipeline diagram

Cycle	1	2	3	4	5
`L.D F0,0(R1)`	IF	IS	EX	M	**WB**
`MUL.D F4,F0,F2`		IF	IS	S	**EX1**
`S.D F4,0(R1)`			IF	IS	**EX**
`DADDIU R1,R1,#-8`				IF	**IS**
`BNE R1,R2,Loop`					**IF**
`L.D F0,0(R1)`					
`MUL.D F4,F0,F2`					
`S.D F4,0(R1)`					
`DADDIU R1,R1,#-8`					
`BNE R1,R2,Loop`					

Cycle 6: The first instruction commits in this cycle. Since the L.D is the oldest instruction in the ROB, and it's ready to commit, it writes its result to the register file and clears the corresponding ROB entry. The MUL.D now becomes the head of the ROB; no other instructions will commit until this instruction finishes execution.

The BNE issues in this cycle; much like the store, this instruction does not require a reservation station, but it does occupy an entry in the ROB. Committing a branch implies that all instructions following that branch are non-speculative, so keeping these instructions in the ROB is extremely important.

We assume that the BNE can be accurately predicted, so we fetch the second L.D in this cycle.

The MUL.D continues executing. The S.D reaches its MEM stage, but performs no actual work, as it does not have the necessary register value and will not actually access memory until commit. The DADDIU begins execution in this cycle.

Reorder buffer

	Op	Dest	Value	Ready
~~1~~	~~L.D~~	~~F0~~	~~M[0+R1]~~	~~Y~~
2	MUL.D	F4		N
3	S.D	0+R1	ROB2	N
4	DADDIU	R1		N
5	BNE	--		N
6				
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	N						
Load2	N						
INT1	Y	DADDIU	[R1]	-8			
INT2	N						
Mult1	Y	MUL.D	[F0]	[F2]			
Mult2	N						

Register result status table

F0	F2	F4	R1	R2
		ROB2	ROB4	

Pipeline diagram

Cycle	1	2	3	4	5	6
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2
`S.D F4,0(R1)`			IF	IS	EX	S
`DADDIU R1,R1,#-8`				IF	IS	EX1
`BNE R1,R2,Loop`					IF	IS
`L.D F0,0(R1)`						IF
`MUL.D F4,F0,F2`						
`S.D F4,0(R1)`						
`DADDIU R1,R1,#-8`						
`BNE R1,R2,Loop`						

Cycle 7: The MUL.D and DADDIU continue executing in this cycle; the DADDIU will complete execution in this cycle and will be able to broadcast its result in the following cycle. All other issued instructions are dependent on those two instructions—the S.D depends on the MUL.D; the BNE depends on the DADDIU—and must therefore stall. We issue the second L.D in this cycle; this instruction also depends on the DADDIU, so we mark the Qj field for the load with the number of the ROB entry (ROB4) that will eventually contain the correct value. We also fetch the second MUL.D.

Reorder buffer

	Op	Dest	Value	Ready
1				
2	MUL.D	F4		N
3	S.D	0+R1	ROB2	N
4	DADDIU	R1		N
5	BNE	--		N
6	L.D	F0		N
7				
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D			ROB4		0
Load2	N						
INT1	Y	DADDIU	[R1]	-8			
INT2	N						
Mult1	Y	MUL.D	[F0]	[F2]			
Mult2	N						

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB2	ROB4	

Pipeline diagram

Cycle	1	2	3	4	5	6	7
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C	
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3
`S.D F4,0(R1)`			IF	IS	EX	S	S
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2
`BNE R1,R2,Loop`					IF	IS	S
`L.D F0,0(R1)`						IF	IS
`MUL.D F4,F0,F2`							IF
`S.D F4,0(R1)`							
`DADDIU R1,R1,#-8`							
`BNE R1,R2,Loop`							

Cycle 8: The DADDIU completes in this cycle, broadcasting its result on the CDB and clearing its reservation station. This instruction is now ready to commit, although it will not do so for several cycles. The result of this instruction is therefore stored in the ROB until the instruction commits, and any dependent instructions looking to read this value will get it from the ROB. The BNE and L.D instructions both depend on the DADDIU; they can now begin execution. The BNE calculates its correct outcome, while the L.D calculates its address.

We issue the second MUL.D in this cycle. Note that when we do so, we rename F4 to ROB7, overwriting the previous entry for that register in the register result status table. This means that when the first MUL.D commits, it will not actually update the register file because a newer instruction that writes the same register is currently in flight.

Reorder buffer

	Op	Dest	Value	Ready
1				
2	MUL.D	F4		N
3	S.D	0+R1	ROB2	N
4	**DADDIU**	**R1**	**[R1]+(-8)**	**Y**
5	BNE	--		N
6	L.D	F0		N
7	**MUL.D**	**F4**		**N**
8				
9				
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D					**0+[R1] +(-8)**
Load2	N						
INT1	**N**						
INT2	N						
Mult1	Y	MUL.D	[F0]	[F2]			
Mult2	**Y**	**MUL.D**		**[F2]**	**ROB6**		

Register result status table

F0	F2	**F4**	R1	R2
ROB6		**ROB7**	ROB4	

Pipeline diagram

Cycle	1	2	3	4	5	6	7	**8**
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C		
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	**EX4**
`S.D F4,0(R1)`			IF	IS	EX	S	S	**S**
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	**WB**
`BNE R1,R2,Loop`					IF	IS	S	**EX**
`L.D F0,0(R1)`						IF	IS	**EX**
`MUL.D F4,F0,F2`							IF	**IS**
`S.D F4,0(R1)`								**IF**
`DADDIU R1,R1,#-8`								
`BNE R1,R2,Loop`								

<u>Cycle 9:</u> We issue the second store in this cycle, which once again takes up only a ROB entry and not a reservation station. This instruction must wait for the second multiply to complete.

The BNE, having determined its outcome in the previous cycle, is now marked as being ready to commit. Both this instruction and the DADDIU are now waiting to commit; they are not technically stalled, as they are not consuming execution resources, so we mark their status as "—" in the pipeline diagram.

The first multiply continues execution. The second load accesses memory in this cycle and will write back its result in the following cycle. The first store and second multiply remain stalled.

<u>Reorder buffer</u>

	Op	Dest	Value	Ready
1				
2	MUL.D	F4		N
3	S.D	0+R1	ROB2	N
4	DADDIU	R1	[R1]+(-8)	Y
5	**BNE**	**--**		**Y**
6	L.D	F0		N
7	MUL.D	F4		N
8	**S.D**		**ROB7**	**N**
9				
10				

<u>Reservation stations:</u>

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	Y	L.D					0+[R1] +(-8)
Load2	N						
INT1	N						
INT2	N						
Mult1	Y	MUL.D	[F0]	[F2]			
Mult2	Y	MUL.D		[F2]	ROB6		

<u>Register result status table</u>

F0	F2	F4	R1	R2
ROB6		ROB7	ROB4	

<u>Pipeline diagram</u>

Cycle	1	2	3	4	5	6	7	8	9
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C			
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	EX4	**EX5**
`S.D F4,0(R1)`			IF	IS	EX	S	S	S	**S**
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	WB	**--**
`BNE R1,R2,Loop`					IF	IS	S	EX	**--**
`L.D F0,0(R1)`						IF	IS	EX	**M**
`MUL.D F4,F0,F2`							IF	IS	**S**
`S.D F4,0(R1)`								IF	**IS**
`DADDIU R1,R1,#-8`									**IF**
`BNE R1,R2,Loop`									

Cycle 10: The second L.D completes in this cycle. This completion allows the second multiply to begin execution; it also leads to the clearing of the reservation station and the marking of this instruction as ready to commit. The value loaded from memory is stored in ROB entry 6, where the load resides.

We issue the second DADDIU; once again, we rename a register (R1) with a value that has not been committed by an earlier instruction (in this case, the first DADDIU). That first DADDIU will therefore not write the register file when it commits. Note that the second DADDIU reads the value of R1 from the ROB, not the register file, which is why the Vj field of that instruction has the value "[R1]+(-8)," the same value to be committed by the first add.

Reorder buffer

	Op	Dest	Value	Ready
1				
2	MUL.D	F4		N
3	S.D	0+R1	ROB2	N
4	DADDIU	R1	[R1]+(-8)	Y
5	BNE	--		Y
6	L.D	F0	M[R1-8]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1		N
10				

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	N						
Load2	N						
INT1	Y	DADDIU	[R1]+(-8)	-8			
INT2	N						
Mult1	Y	MUL.D	[F0]	[F2]			
Mult2	Y	MUL.D	[F0]	[F2]			

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	1	2	3	4	5	6	7	8	9	10
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C				
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6
`S.D F4,0(R1)`			IF	IS	EX	S	S	S	S	S
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	WB	--	--
`BNE R1,R2,Loop`					IF	IS	S	EX	--	--
`L.D F0,0(R1)`						IF	IS	EX	M	WB
`MUL.D F4,F0,F2`							IF	IS	S	EX1
`S.D F4,0(R1)`								IF	IS	EX
`DADDIU R1,R1,#-8`									IF	IS
`BNE R1,R2,Loop`										IF

Cycle 11: The first MUL.D finishes execution in this cycle and broadcasts its results on the CDB. This broadcast leads to both the multiply and the first S.D being marked as ready to commit, since the store is waiting for the result of the multiply. We also clear the corresponding reservation station.

We issue the final instruction in this sequence, the second BNE, in this cycle. The branch requires only a ROB entry.

Reorder buffer

	Op	Dest	Value	Ready
1				
2	MUL.D	F4	[F0]*[F2]	Y
3	S.D	0+R1	[F0]*[F2]	Y
4	DADDIU	R1	[R1]+(-8)	Y
5	BNE	--		Y
6	L.D	F0	M[R1-8]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1		N
10	BNE	--		N

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk	A
Load1	N						
Load2	N						
INT1	Y	DADDIU	[R1]+(-8)	-8			
INT2	N						
Mult1	N						
Mult2	Y	MUL.D	[F0]	[F2]			

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	1	2	3	4	5	6	7	8	9	10	11
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C					
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB
`S.D F4,0(R1)`			IF	IS	EX	S	S	S	S	S	--
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	WB	--	--	--
`BNE R1,R2,Loop`					IF	IS	S	EX	--	--	--
`L.D F0,0(R1)`						IF	IS	EX	M	WB	--
`MUL.D F4,F0,F2`							IF	IS	S	EX1	EX2
`S.D F4,0(R1)`								IF	IS	EX	S
`DADDIU R1,R1,#-8`									IF	IS	EX1
`BNE R1,R2,Loop`										IF	IS

Cycle 12: The first multiply commits in this cycle. Note, however, that the most up-to-date value of F4—the destination register for this instruction—will come from ROB7, so this instruction does not update the register file. Note that we now have a string of five instructions, starting with this MUL.D, ready to commit, so a new instruction will commit in each of the next several cycles.

The second multiply and add instructions continue execution. The second store and branch instructions depend on these two instructions, and therefore stall.

Reorder buffer

	Op	**Dest**	**Value**	**Ready**
1				
~~2~~	~~MUL.D~~	~~F4~~	~~[F0]*[F2]~~	~~Y~~
3	S.D	0+R1	[F0]*[F2]	Y
4	DADDIU	R1	[R1]+(-8)	Y
5	BNE	--		Y
6	L.D	F0	M[R1-8]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1		N
10	BNE	--		N

Reservation stations:

Name	**Busy?**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**A**
Load1	N						
Load2	N						
INT1	Y	DADDIU	[R1]+(-8)	-8			
INT2	N						
Mult1	N						
Mult2	Y	MUL.D	[F0]	[F2]			

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C						
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C
`S.D F4,0(R1)`			IF	IS	EX	S	S	S	S	S	--	--
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	WB	--	--	--	--
`BNE R1,R2,Loop`					IF	IS	S	EX	--	--	--	--
`L.D F0,0(R1)`						IF	IS	EX	M	WB	--	--
`MUL.D F4,F0,F2`							IF	IS	S	EX1	EX2	EX3
`S.D F4,0(R1)`								IF	IS	EX	S	S
`DADDIU R1,R1,#-8`									IF	IS	EX1	EX2
`BNE R1,R2,Loop`										IF	IS	S

Cycle 13: The first store commits in this cycle and will now be allowed to write to memory.

The second DADDIU writes its result to the CDB in this cycle, which allows the second branch to determine its correct outcome. The MUL.D continues executing, while the S.D stalls while waiting for the multiply.

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
~~3~~	~~S.D~~	~~0+R1~~	~~[F0]*[F2]~~	~~Y~~
4	DADDIU	R1	[R1]+(-8)	Y
5	BNE	--		Y
6	L.D	F0	M[R1-8]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1	**[R1]+(-16)**	**Y**
10	BNE	--		N

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk
Load1	N					
Load2	N					
INT1	**N**					
INT2	N					
Mult1	N					
Mult2	Y	MUL.D	[F0]	[F2]		

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
L.D F0,0(R1)	IF	IS	EX	M	WB	C							
MUL.D F4,F0,F2		IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C	
S.D F4,0(R1)			IF	IS	EX	S	S	S	S	S	--	--	**C**
DADDIU R1,R1,#-8				IF	IS	EX1	EX2	WB	--	--	--	--	**--**
BNE R1,R2,Loop					IF	IS	S	EX	--	--	--	--	**--**
L.D F0,0(R1)						IF	IS	EX	M	WB	--	--	**--**
MUL.D F4,F0,F2							IF	IS	S	EX1	EX2	EX3	**EX4**
S.D F4,0(R1)								IF	IS	EX	S	S	**S**
DADDIU R1,R1,#-8									IF	IS	EX1	EX2	**WB**
BNE R1,R2,Loop										IF	IS	S	**EX**

Cycle 14: The first DADDIU commits in this cycle, but is not allowed to update the register file, as the instruction in ROB9 (the second DADDIU) will produce a more up-to-date version of R1.

The second branch is marked as ready to commit in this cycle. The second multiply continues execution.

Note that we have removed the first cycle from the pipeline diagram for space reasons; we will continue to do so for all remaining cycles.

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
3				
4	~~DADDIU~~	~~R1~~	~~[R1]+(-8)~~	~~Y~~
5	BNE	--		Y
6	L.D	F0	M[0+R1]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1	[R1]+(-16)	Y
10	BNE	--		Y

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk
Load1	N					
Load2	N					
INT1	N					
INT2	N					
Mult1	N					
Mult2	Y	MUL.D	[F0]	[F2]		

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	2	3	4	5	6	7	8	9	10	11	12	13	14
L.D F0,0(R1)	IS	EX	M	WB	C								
MUL.D F4,F0,F2	IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C		
S.D F4,0(R1)		IF	IS	EX	S	S	S	S	S	--	--	C	
DADDIU R1,R1,#-8			IF	IS	EX1	EX2	WB	--	--	--	--	--	C
BNE R1,R2,Loop				IF	IS	S	EX	--	--	--	--	--	--
L.D F0,0(R1)					IF	IS	EX	M	WB	--	--	--	--
MUL.D F4,F0,F2						IF	IS	S	EX1	EX2	EX3	EX4	EX5
S.D F4,0(R1)							IF	IS	EX	S	S	S	S
DADDIU R1,R1,#-8								IF	IS	EX1	EX2	WB	--
BNE R1,R2,Loop									IF	IS	S	EX	--

Cycle 15: The first branch commits in this cycle. We assume this instruction was predicted correctly and allow later instructions to commit. If the branch had been mispredicted, we would have identified that misprediction in its EX stage, and all other instructions in the ROB, as well as the pipeline, would be flushed. Instruction fetch would then restart at the correct instruction after the branch.

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
3				
4				
~~5~~	~~BNE~~	~~--~~		~~Y~~
6	L.D	F0	M[0+R1]	Y
7	MUL.D	F4		N
8	S.D	0+R1	ROB7	N
9	DADDIU	R1	[R1]+(-16)	Y
10	BNE	--		Y

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk
Load1	N					
Load2	N					
INT1	N					
INT2	N					
Mult1	N					
Mult2	Y	MUL.D	[F0]	[F2]		

Register result status table

F0	F2	F4	R1	R2
ROB6		ROB7	ROB9	

Pipeline diagram

Cycle	3	4	5	6	7	8	9	10	11	12	13	14	15
`L.D F0,0(R1)`	EX	M	WB	C									
`MUL.D F4,F0,F2`	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C			
`S.D F4,0(R1)`	IF	IS	EX	S	S	S	S	S	--	--	C		
`DADDIU R1,R1,#-8`		IF	IS	EX1	EX2	WB	--	--	--	--	--	C	
`BNE R1,R2,Loop`			IF	IS	S	EX	--	--	--	--	--	--	C
`L.D F0,0(R1)`				IF	IS	EX	M	WB	--	--	--	--	--
`MUL.D F4,F0,F2`					IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6
`S.D F4,0(R1)`						IF	IS	EX	S	S	S	S	S
`DADDIU R1,R1,#-8`							IF	IS	EX1	EX2	WB	--	--
`BNE R1,R2,Loop`								IF	IS	S	EX	--	--

Cycle 16: The second load commits this cycle. As we can see from the register result status table (as shown in the previous cycle), this instruction holds the most up-to-date value for F0 and will therefore be allowed to update the register file.

The second multiply writes its result to the CDB this cycle, meaning that both the multiply and the store that needs its result are now ready to commit.

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
3				
4				
5				
~~6~~	~~L.D~~	~~F0~~	~~M[R1-8]~~	~~Y~~
7	**MUL.D**	**F4**	**[F0]*[F2]**	**Y**
8	S.D	0+R1	**[F0]*[F2]**	**Y**
9	DADDIU	R1	[R1]+(-16)	Y
10	BNE	--		Y

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk
Load1	N					
Load2	N					
INT1	N					
INT2	N					
Mult1	N					
Mult2	**N**					

Register result status table

F0	F2	F4	R1	R2
		ROB7	ROB9	

Pipeline diagram

Cycle	4	5	6	7	8	9	10	11	12	13	14	15	16
L.D F0,0(R1)	M	WB	C										
MUL.D F4,F0,F2	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C				
S.D F4,0(R1)	IS	EX	S	S	S	S	S	--	--	C			
DADDIU R1,R1,#-8	IF	IS	EX1	EX2	WB	--	--	--	--	--	C		
BNE R1,R2,Loop		IF	IS	S	EX	--	--	--	--	--	--	C	
L.D F0,0(R1)			IF	IS	EX	M	WB	--	--	--	--	--	C
MUL.D F4,F0,F2				IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB
S.D F4,0(R1)					IF	IS	EX	S	S	S	S	S	--
DADDIU R1,R1,#-8						IF	IS	EX1	EX2	WB	--	--	--
BNE R1,R2,Loop							IF	IS	S	EX	--	--	--

Cycles 17-20: All remaining instructions commit during these cycles, one per cycle. The appropriate register result status table entries are cleared in cycle 17 (when the MUL.D commits) and cycle 19 (when the DADDIU commits).

Reorder buffer

	Op	Dest	Value	Ready
1				
2				
3				
4				
5				
6				
~~7~~	~~MUL.D~~	~~F4~~	~~[F0]*[F2]~~	~~Y~~
~~8~~	~~S.D~~	~~0+R1~~	~~[F0]*[F2]~~	~~Y~~
~~9~~	~~DADDIU~~	~~R1~~	~~[R1]+(-16)~~	~~Y~~
~~10~~	~~BNE~~	~~--~~		~~Y~~

Reservation stations:

Name	Busy?	Op	Vj	Vk	Qj	Qk
Load1	N					
Load2	N					
INT1	N					
INT2	N					
Mult1	N					
Mult2	N					

Register result status table

F0	F2	F4	R1	R2
		~~ROB7~~	~~ROB9~~	

Pipeline diagram

Cycle	8	9	10	11	12	13	14	15	16	17	18	19	20
`L.D F0,0(R1)`													
`MUL.D F4,F0,F2`	EX4	EX5	EX6	WB	C								
`S.D F4,0(R1)`	S	S	S	--	--	C							
`DADDIU R1,R1,#-8`	WB	--	--	--	--	--	C						
`BNE R1,R2,Loop`	EX	--	--	--	--	--	--	C					
`L.D F0,0(R1)`	EX	M	WB	--	--	--	--	--	C				
`MUL.D F4,F0,F2`	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C			
`S.D F4,0(R1)`	IF	IS	EX	S	S	S	S	S	--	--	C		
`DADDIU R1,R1,#-8`		IF	IS	EX1	EX2	WB	--	--	--	--	--	C	
`BNE R1,R2,Loop`			IF	IS	S	EX	--	--	--	--	--	--	C

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
`L.D F0,0(R1)`	IF	IS	EX	M	WB	C														
`MUL.D F4,F0,F2`		IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C								
`S.D F4,0(R1)`			IF	IS	EX	S	S	S	S	S	--	--	C							
`DADDIU R1,R1,#-8`				IF	IS	EX1	EX2	WB	--	--	--	--	--	C						
`BNE R1,R2,Loop`					IF	IS	S	EX	--	--	--	--	--	--	C					
`L.D F0,0(R1)`						IF	IS	EX	M	WB	--	--	--	--	--	C				
`MUL.D F4,F0,F2`							IF	IS	S	EX1	EX2	EX3	EX4	EX5	EX6	WB	C			
`S.D F4,0(R1)`								IF	IS	EX	S	S	S	S	S	--	--	C		
`DADDIU R1,R1,#-8`									IF	IS	EX1	EX2	WB	--	--	--	--	--	C	
`BNE R1,R2,Loop`										IF	IS	S	EX	--	--	--	--	--	--	C

The full pipeline diagram for all 20 cycles. As we've discussed, instructions are allowed to execute and complete out of order, but they must commit in order to allow easy recovery from branch mispredictions and maintain precise exceptions.