Reorder Buffer Implementation (Pentium Pro)

Hardware data structures

- **retirement register file (RRF)**
  (~ IBM 360/91 physical registers)
  - physical register file that is the same size as the architectural registers
  - one-to-one relationship

- **reorder buffer (ROB)**
  (~ R10K active list)
  - provides in-order instruction commit
  - circular queue with head & tail pointers
  - holds 40 “executing” instructions in program order (dispatched but not yet committed)
  - field for either integer or FP result when it is computed
  - a result value is put in its register when its producing instruction reaches the head of the buffer & is removed

Reorder Buffer Implementation (Pentium Pro)

- **register alias table (RAT)**
  (~ R10K map table)
  - provides register renaming
    - important because very few GPRs in the x86 architecture
  - indicates whether a source operand of a new instruction points to the reorder buffer or the physical register file
  - do an associative search of ROB destination registers for the new source operands to do the data hazard check

- **reservation station**
  (~ IBM 360/91 reservation stations, R10000 instruction buffer)
  - holds instructions waiting to execute
  - result values go back to the reservation station so dependent instructions have source operand values
  - provides forwarding to reduce RAW hazards
  - provides out-of-order execution

Pentium Pro Execution

In-order issue

- decode instructions
- enter instructions into reorder buffer for in-order completion
- rename registers via register alias table
- detect structural hazards for reservation station

Out-of-order execution

- one reservation station, multiple entries
- RAW hazards checked
- separate integer, FP, memory units
- result goes to reservation station & reorder buffer

In-order completion

- this & previous instructions have completed
- write “G”PR registers
- rollback on interrupts

Pentium Pro

8 stage fetch & decode pipeline (minimum)

BTB access (1 stage)

instruction fetch & align for decoding (2.5 stages)

decode & uop generation (2.5 stages)

register renaming & instruction issue to reservation stations (4 stages minimum)

4 stage integer pipeline (minimum)

execute, resolve branch (1 stage)

write registers (3 stages minimum)

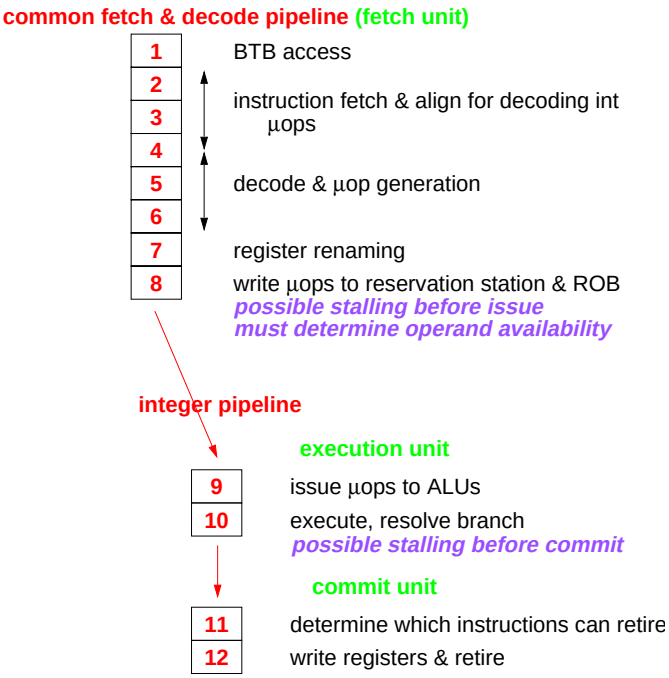
6 stage load pipeline (minimum)

address calculation & to memory reorder buffer (1 stage minimum)

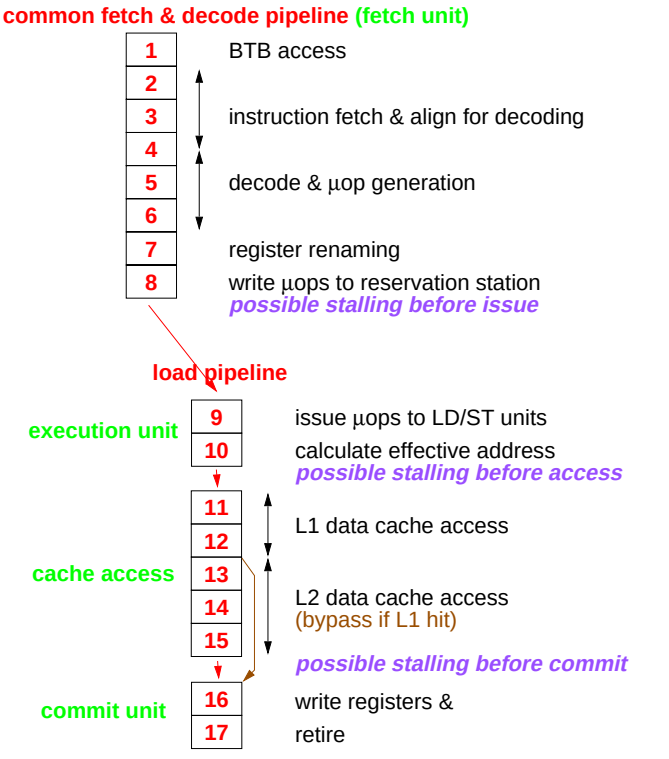
integrated L1 & L2 data cache access

pipelined FP add & multiply

Intel P6 (Pentium Pro)



Intel P6



Intel P6

Some bandwidth constraints: maximum for one cycle

- 16 bytes fetched
- 3 instructions decoded
- 6 μops to the reorder buffer
- 6 μops dispatched to functional units
  - 1 load & 1 store access to the L1 data cache
- 1 cache result returned
- 3 μops committed

if

- good instruction mix
- right instruction order
- operands available
- functional units available
- load & store to different cache banks
- all previous instructions already committed

Making Tomasulo Precise

Reorder buffer

- contains:
  - instruction type (register operation, store, branch)
  - destination field (register, memory, none)
  - value
- replaces load & store buffers
- replaces renaming function of the reservation stations
  - operand fields in reservation stations point to reorder buffer location
  - results tagged with reorder buffer entry number

Tomasulo's Algorithm: Execution Steps

Tomasulo functions

(assume the instruction has been fetched)

- **issue &read**
  - structural hazard detection for entry in reservation station & **reorder buffer**
    - issue if no hazard
    - stall if hazard
  - read registers for source operands
    - **can come from either registers or reorder buffer**
    - **RAT indicates which**
  - **allocate entry in reorder buffer**
- **execute**
  - RAW hazard detection
  - snoop on common data bus for missing operands
    - **reservation station tag is entry number in reorder buffer**
  - dispatch to functional unit when obtain both operand values
- **write result**
  - broadcast result & **reorder buffer entry** (tag) on the common data bus to reservation stations & **reorder buffer**

Tomasulo's Algorithm: Execution Steps

- **commit**
  - **retire the instruction at the head of the reorder buffer**
  - **update register with result in reorder buffer or do a store**
  - **remove instruction from reorder buffer**
  - **if a mispredicted branch, restart with right-path instructions**

Example in the Book 1

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Commit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
ld F2, 45(R3)	yes	yes		
multd F0, F2, F4	yes			
subd F8, F6, F2	yes			
divd F10, F0, F6	yes			
addd F6, F8, F2	yes			

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	yes	subd	**(ROB 1)**			ROB 2
Add2	yes	addd			ROB 4	ROB 2
Add3	no					
Mult1	yes	multd		(F4)	ROB 2	
Mult2	yes	div		**(ROB 1)**	ROB 3	

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
(ROB 3)	(ROB 2)		(ROB 6)	(ROB 4)	(ROB 5)	

Cycle after first load has committed

Example in the Book 2

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Com-mit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~ld F2, 45(R3)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
multd F0, F2, F4	yes	**yes**		
subd F8, F6, F2	yes	**yes**		
divd F10, F0, F6	yes			
addd F6, F8, F2	yes			

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	**no**	subd				
Add2	yes	addd		**(ROB 2)**	ROB 4	
Add3	no					
Mult1	yes	multd	**(ROB 2)**	(F4)		
Mult2	yes	divd		**(ROB 1)**	ROB 3	

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
(ROB 3)	**ROB 2**		(ROB 6)	(ROB 4)	(ROB 5)	

Cycle after second load has committed

Example in the Book 3

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Com-mit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~ld F2, 45(R3)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
multd F0, F2, F4	yes	yes		
subd F8, F6, F2	yes	yes	yes	
divd F10, F0, F6	yes			
addd F6, F8, F2	yes	yes		

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	no	subd				
Add2	yes	addd	(ROB 4)	(ROB 2)		
Add3	no					
Mult1	yes	multd	(ROB 2)	(F4)		
Mult2	yes	divd		(ROB 1)	ROB 3	

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
(RBO 3)	ROB 2		(ROB 6)	(ROB 4)	(ROB 5)	

Cycle after subd has written its result in the reorder buffer but can't commit

Example in the Book 4

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Commit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~ld F2, 45(R3)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
multd F0, F2, F4	yes	yes		
subd F8, F6, F2	yes	yes	yes	
divd F10, F0, F6	yes			
addd F6, F8, F2	yes	yes	yes	

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	no	subd				
Add2	no	addd				
Add3	no					
Mult1	yes	multd	(ROB 2)	(F4)		
Mult2	yes	divd		(ROB 1)	ROB 3	

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
(ROB 3)	ROB 2		(ROB 6)	(ROB 4)	(ROB 5)	

Cycle after addd has written its result in the reorder buffer but can't commit

Example in the Book 5

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Commit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~ld F2, 45(R3)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~multd F0, F2, F4~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
subd F8, F6, F2	yes	yes	yes	
divd F10, F0, F6	yes	yes		
addd F6, F8, F2	yes	yes	yes	

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	no	subd				
Add2	no	addd				
Add3	no					
Mult1	no	multd				
Mult2	yes	divd	(ROB 3)	(ROB 1)		

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
ROB 3	ROB 2		(ROB 6)	(ROB 4)	(ROB 5)	

Cycle after multd has written its result in the reorder buffer & committed

Example in the Book 6

~ Reorder Buffer				
Instruction	**Issue**	**Execute**	**Write Result**	**Commit**
~~ld F6, 34(R2)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~ld F2, 45(R3)~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~multd F0, F2, F4~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
~~subd F8, F6, F2~~	~~yes~~	~~yes~~	~~yes~~	~~yes~~
divd F10, F0, F6	yes	yes		
addd F6, F8, F2	yes	yes	yes	

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	no	subd				
Add2	no	addd				
Add3	no					
Mult1	no	multd				
Mult2	yes	divd	(ROB 3)	(ROB 1)		

Register Status (Q_i)

F0	F2	F4	F6	F8	F10	F12...
ROB 3	ROB 2		(ROB 6)	ROB 4	(ROB 5)	

Cycle after subd has committed

Next: complete divd, commit divd, commit addd

Limits

Limits on out-of-order execution

- amount of ILP in the code
- scheduling window size
- number & types of functional units
- need to do associative searches & its effect on cycle time