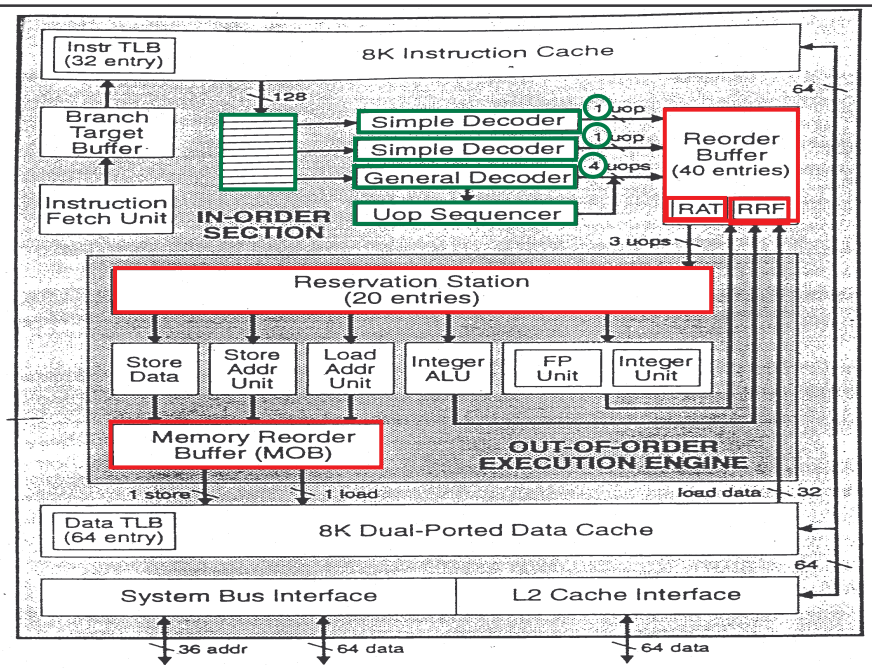
Reorder Buffer Implementation (Pentium Pro)

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 after it has been computed
  - a result value is put in its register in the RRF after its producing instruction reaches the head of the buffer & is removed (i.e., commits)

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
    - if found, consumer instruction points to the producer instruction in the ROB
    - the data hazard check before instruction dispatch
- **reservation station**
  (~ IBM 360/91 reservation stations, R10000 instruction buffer)
  - holds instructions waiting to execute
  - provides forwarding to reduce RAW hazards
    - result values go back to the reservation station (as well as ROB) so dependent instructions have source operand values
  - provides out-of-order execution



Pentium Pro Execution

In-order issue

- decode instructions
- enter **uops** 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
- check source operands for RAW hazards
- check structural hazards for separate integer, FP, memory units
- result goes to reservation station & reorder buffer

In-order completion

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

Pentium Pro

fetch & decode pipeline

- 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** (3 stages minimum)

integer pipeline

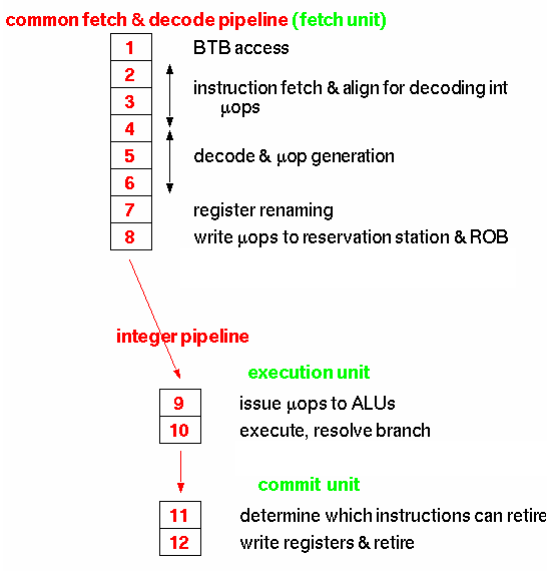
- execute, resolve branch
- write registers & commit

load pipeline

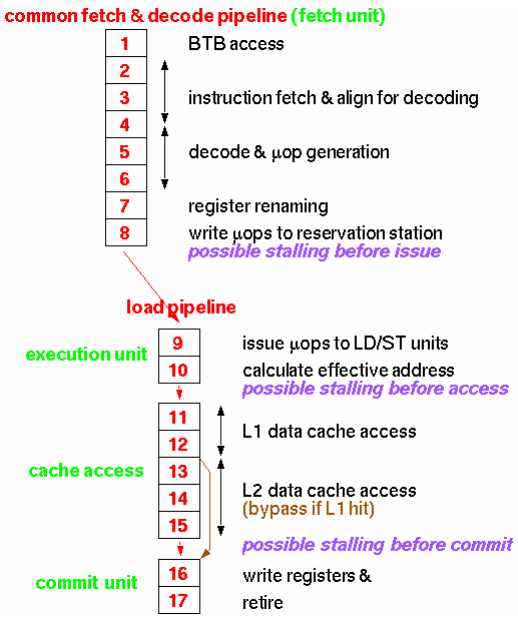
- address calculation & to memory reorder buffer
- integrated L1 & L2 data cache access

pipelined FP add & multiply

Pentium Pro



Pentium Pro



Pentium Pro

Some bandwidth constraints: maximum for one cycle

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

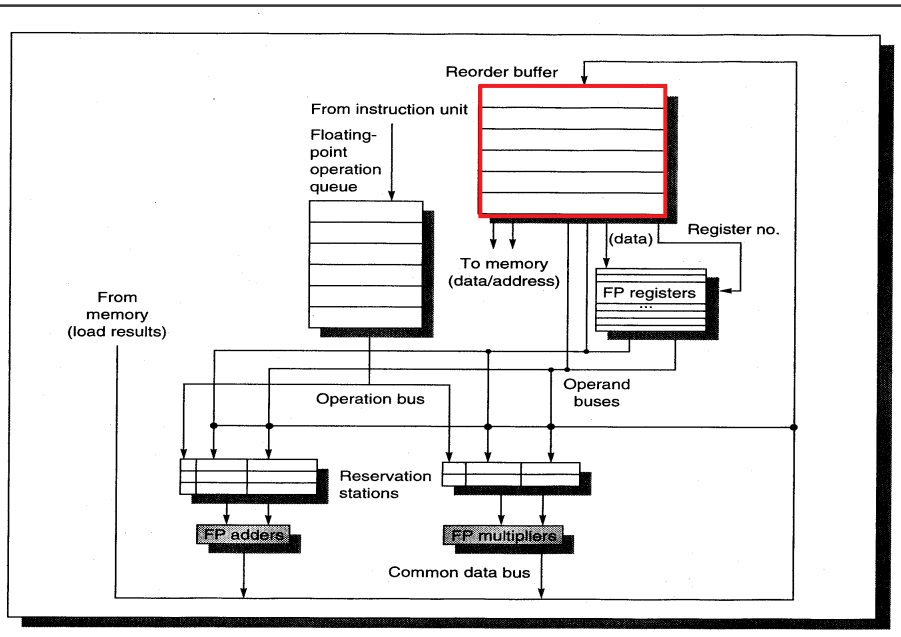
if

- 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)
  - result values
  - ready bit to indicate instruction has executed
- replaces load & store buffers
- does renaming (instead of the reservation stations)
  - operand fields in reservation stations point to reorder buffer location
  - results tagged with reorder buffer entry number
- provides hardware speculation (speculative execution, not just fetch & issue) plus precise interrupts



Tomasulo's Algorithm: Execution Steps

issue & read (assume the instruction has been fetched)

- 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 for source operands
- snoop on common data bus for missing operands
  - **tag in reservation station is entry number in reorder buffer**
- dispatch to functional unit when obtain both operand values

Tomasulo's Algorithm: Execution Steps

write result

- broadcast result & **reorder buffer entry** (tag) on the common data bus to reservation stations & **reorder buffer**

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**

Precise Tomasulo, in action 1

First load has written its result

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 2

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 3

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 4

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 5

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 6

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 7

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 8

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 9

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Precise Tomasulo, in action 10

~ Reorder Buffer

Instruction	Issue	Execute	Write Result	Commit
`ld F6, 34(R2)`	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			

Reservations 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 (Qj)

F0	F2	F4	F6	F8	F10	F12…
ROB 3	ROB 2		ROB 6	ROB 4	ROB 5	

Pool of Physical Registers vs. Reorder Buffer

Think about the advantages and disadvantages of these implementations

- clean-up all done at instruction commit (physical register pool)
  - record that value no longer speculative in register busy table
  - unmap previous mapping for the architectural register
- instruction issue simpler (physical register pool)
  - only look in one place for the source operands (the physical register file)
- book claims that deallocating register is more complicated with a physical register pool
  - have to search for outstanding uses in the active list
  - but not done in practice: wait until the instruction that redefines the architectural register commits

Limits

Limits on out-of-order execution

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