Credits: Dr. Yousif (Intel), Kubi@UCB, Asanovic @ MITs

Advanced Instruction Re-Ordering - Dynamically Scheduled Pipelines: Tomasulo's Algorithm and Register Renaming

Handout 12

October 26, 2004

Shoukat Ali

shoukat@umr.edu

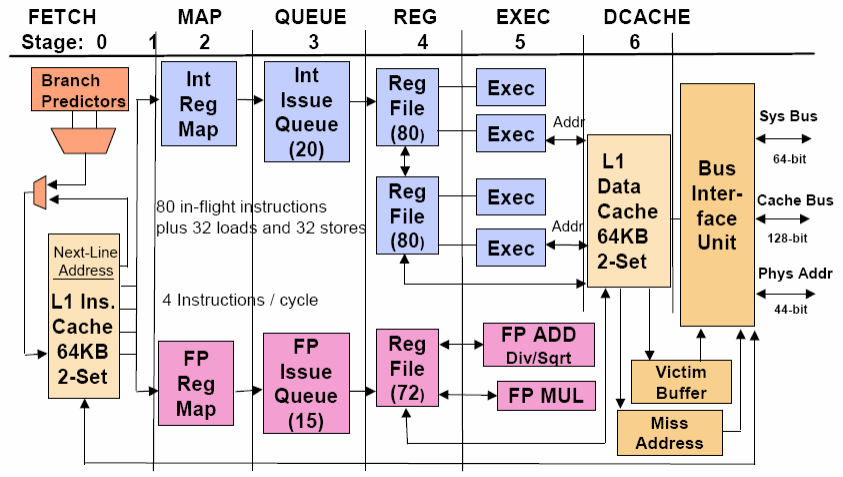
UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

1

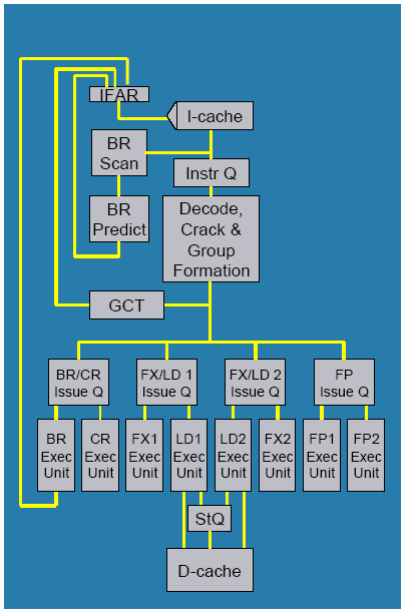
Why So Much Tomasulo's Algorithm?

2

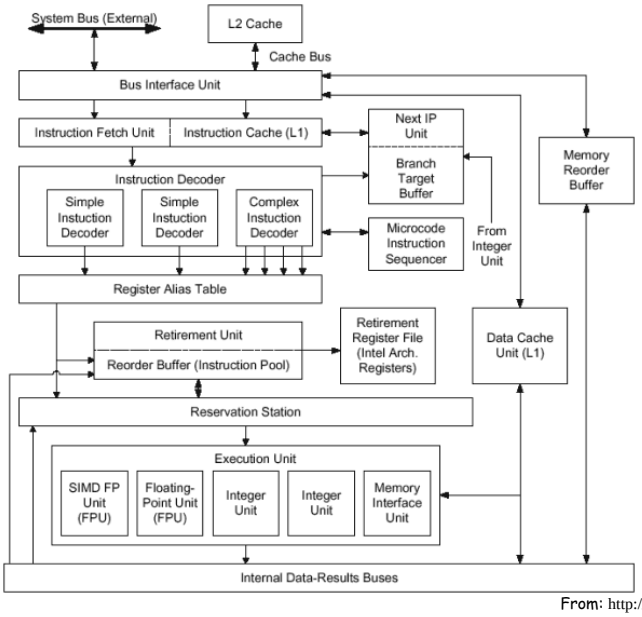
Tomasulo in Alpha 21264



Tomasulo in IBM Power 4



Tomasulo in Intel Pentium 4



From: http://www.digit-life.com/articles/pentium4/

5

Tomasulo Example – Details

- assume
  - floating point add/sub take 2 cycles of **execution**
  - floating point mult takes 10 cycles of **execution**
  - load takes 2 cycles of **execution**
  - floating point divide takes 40 cycles of **execution**
  - three FP adder units, two mult units (mult also does divide)
  - two load units (store units not discussed in this example)
- in following slides,
  - Vj = value of source operand 1
  - Vk = value of source operand 2
    - I know V1 and V2 would have been better symbols!!!
  - Qj = for a pending op, this is # of the RS which will produce Vj
  - Qk = for a pending op, this is # of the RS which will produce Vk

6

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1	No					
Add2	2	No					
Add3	3	No					
Mult1	4	No					
Mult2	5	No					
Load1	6	No					
Load2	7	No					

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0	0	18
2	0	32
4	0	56
6	0	31
8	0	98
10	0	66

7

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1	No					
Add2	2	No					
Add3	3	No					
Mult1	4	No					
Mult2	5	No					
Load1	6	No					
Load2	7	No					

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0	0	18
2	0	32
4	0	56
6	0	31
8	0	98
10	0	66

8

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

9

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

10

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

11

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

12

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

13

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

14

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	*RS Tag*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

15

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	*RS Tag*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

16

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

17

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

18

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

19

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

20

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

21

finishing times

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

IS	RPO	EXE	WR

RS	RS Tag	Busy	Op	Vj	Vk	Qj	Qk
Add1	1						
Add2	2						
Load1	3						
Load2	4						
Add3	5						
Mult1	6						
Mult2	7						

Integer Reg #	Q	V
1	0	213
2	0	50
3	0	100
4	0	417
5	0	87

FP Reg #	Q	V
0		
2		
4		
6		
8		
10		

22

In-Order Issue, Out-Of-Order Execution

Instruction	j	k
LD F6	34	R2
LD F2	45	R3
MULTD F0	F2	F4
SUBD F8	F6	F2
DIVD F10	F0	F6
ADDD F6	F8	F2

finishing times

IS	RPO	EXE	WR
1	1	3	4
2	2	4	5
3	5	15	16
4	5	7	8
5	16	56	57
6	8	10	11

instructions issued in program order

instructions enter execution out of program order

referred to as “in-order issue, out-of-order execution”

23

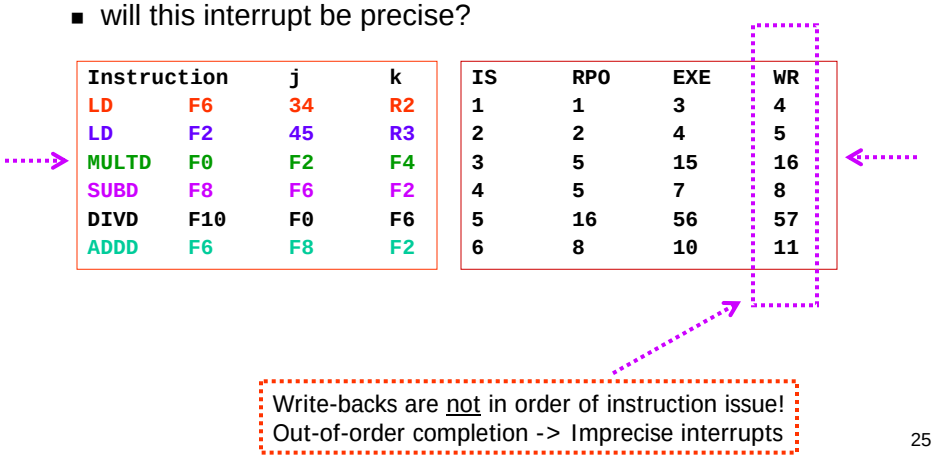
Two Further Concepts Related to Tom’s Alg

- impact of Tomasulo on precise interrupts
- register renaming as used in Tomasulo’s algorithm

24

Impact of Tomasulo on Precise Interrupts

- Tomasulo's algorithm allows in-order issue, out-of-order execution and **out-of-order completion (commission, retirement, graduation)**
- what if multd raises an interrupt in cycle 9?
  - will this interrupt be precise?



Instruction		j	k
LD	F6	34	R2
LD	F2	45	R3
MULTD	F0	F2	F4
SUBD	F8	F6	F2
DIVD	F10	F0	F6
ADDD	F6	F8	F2

IS	RPO	EXE	WR
1	1	3	4
2	2	4	5
3	5	15	16
4	5	7	8
5	16	56	57
6	8	10	11

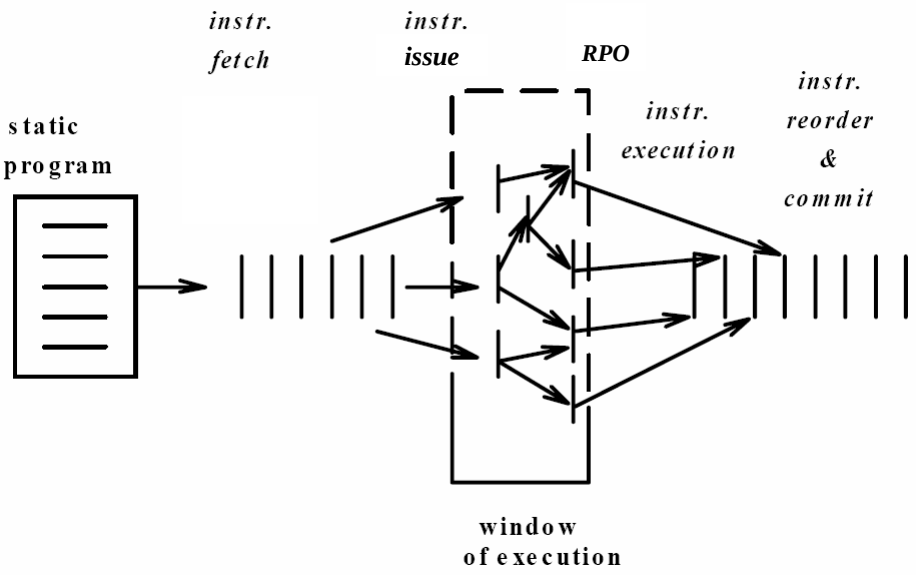
25

Fixing Tom's Alg For Precise Interrupts

- Tomasulo's algorithm must be modified if interrupts are to be precise
- solution: allow instructions to execute out of order, **but** force them to commit in order
  - once write-back (or commit) is in program order, we can enforce precise interrupts as we did for MIPS integer pipeline
- one implementation of above solution = re-order buffer, ROB
- used in PowerPC 603/604/G3/G4, MIPS R10K/R12K, Pentium II/III/4, Alpha 21264

26

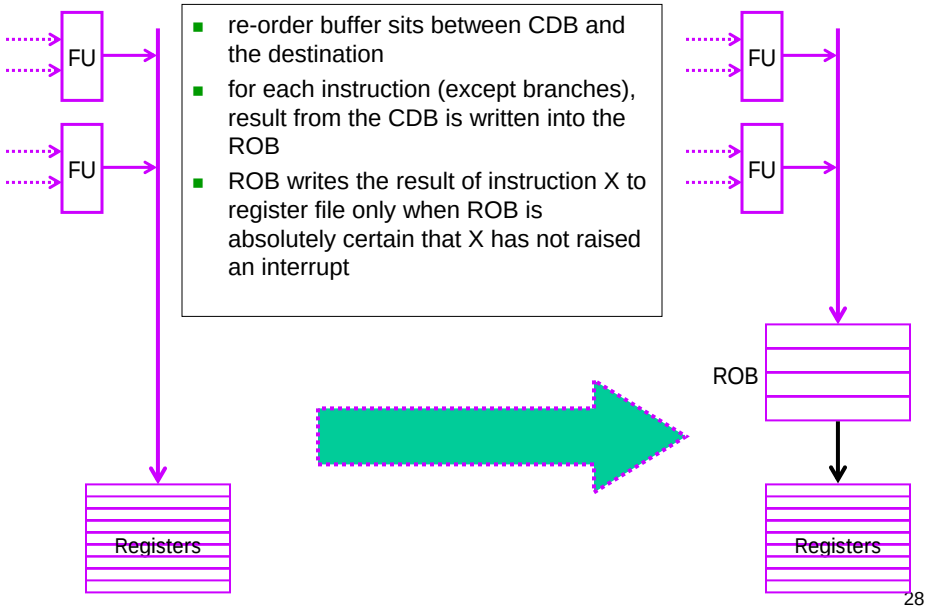
Function of Re-Order Buffer: The Big Picture



27

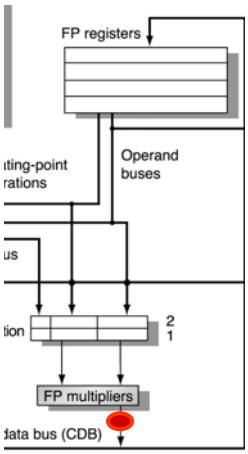
Slide borrowed from Hill, Smith, Sohi, Wood – UW ECE/CS 752 Lecture Notes

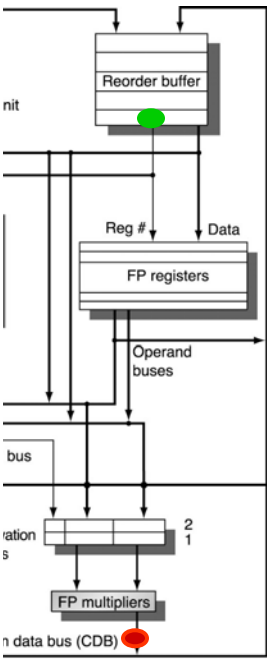
Structure of ROB: The Middle Man



28

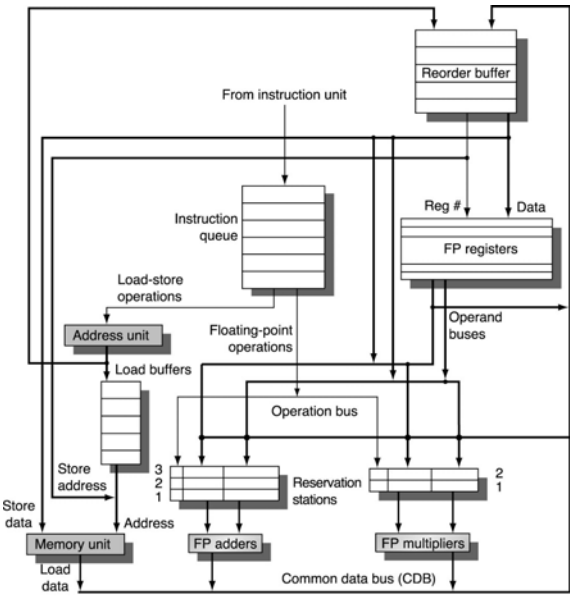
ROB: The Middle Man





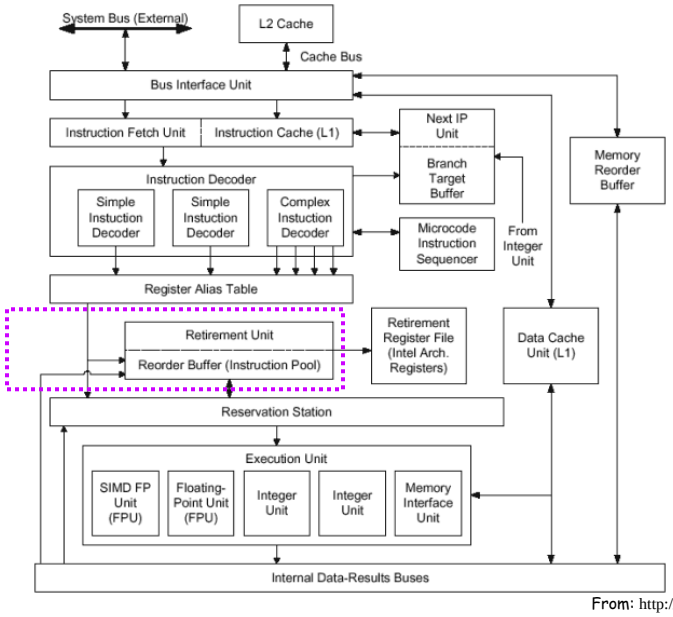
29

ROB In A Typical Dynamically Scheduled Pipeline



30

ROB in Intel Pentium 4



From: http://www.digit-life.com/articles/pentium4/

31

Re-Order Buffer: An Analogy

- imagine a group of students waiting to use a computer lab
- assume we want them to leave the lab in the order they enter
- different students use computers for different times
  - will get done at different times
    - but must leave in the order they arrive

- in-order leaving
  - make students sign up a sheet as they arrive
  - then ensure students leave in the order their names are signed
  - for students that get done sooner than an “earlier-admitted” student, make them wait in an area of lab called “re-order buffer”
    - in re-order buffer, queue up students in the signature order

32

Re-Order Buffer Analogy: Entry Control

- how do we admit a student in the lab?
  - check if there is a free computer in the lab
  - check if there is a blank line on the sign-up sheet
  - if above two NOT met, make student wait outside the lab

33

Real Deal: Re-Order Buffer Function

- re-order buffer holds the result of an instruction between the time the operation associated with an instruction completes and the time instruction commits
- re-orders all these results so that result from the earliest issued instruction is at the head of the queue
- result from the last issued instruction at the tail
- control checks the queue head every clock cycle
  - if no exceptions are posted, inst's result is written to the reg file
  - otherwise, interrupt is processed
    - will discuss that more in the next lecture

34