CMPE 110: Computer Architecture

Week 5
Out-of-Order Execution

Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

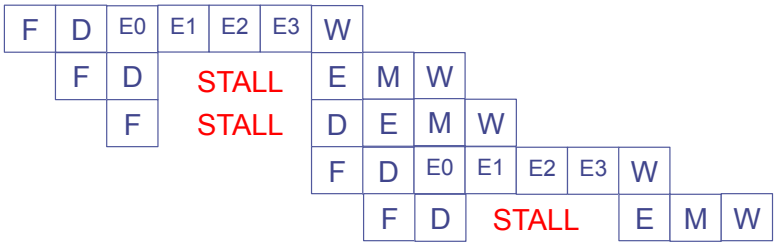
[Adapted in part from Jose Renau, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

Reminder

- Homework 2 will be posted by 4:30pm today
  - Due Wednesday, Nov. 2 midnight
- Midterm1 grade will be posted by today

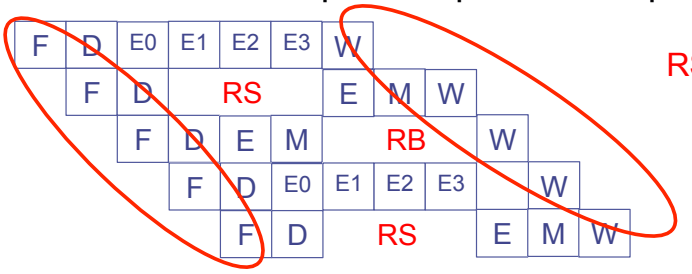
Review: In-order vs. Out-of-order Dispatch

- In order dispatch + precise exceptions:



IMUL R3 ← R1, R2
ADD R3 ← R3, R1
ADD R1 ← R6, R7
IMUL R5 ← R6, R8
ADD R7 ← R3, R5

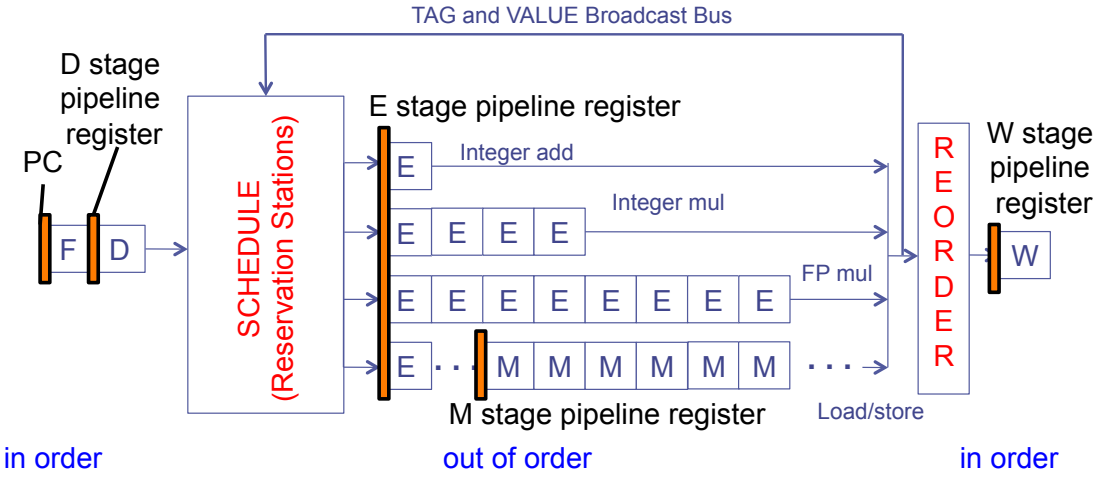
- Out-of-order dispatch + precise exceptions:



RS: wait in reservation station
RB: wait in reorder buffer

- 15 vs. 12 cycles

Review: Two Humps in a Modern Pipeline



- Hump 1: Reservation stations (scheduling window)
- Hump 2: Reordering (reorder buffer, aka instruction window or active window)

Tomasulo's Algorithm: Renaming

- Register rename table (register alias table)

	tag	value	valid?
R0			1
R1			1
R2			1
R3			1
R4			1
R5			1
R6			1
R7			1
R8			1
R9			1

Tomasulo's Algorithm

- If reservation station available before renaming
  - Instruction + renamed operands (registers) inserted into the reservation station
  - Only rename if reservation station is available
- Else stall
- While in reservation station, each instruction:
  - Watches common data bus (CDB) for tag of its sources
  - When tag seen, grab value for the source and keep it in the reservation station
  - When both operands available, instruction ready to be dispatched
- Dispatch instruction to the Functional Unit when instruction is ready
- After instruction finishes in the Functional Unit
  - Arbitrate for CDB
  - Put tagged value onto CDB (tag broadcast)
  - Register file is connected to the CDB
    - Register contains a tag indicating the latest writer to the register
    - If the tag in the register file matches the broadcast tag, write broadcast value into register (and set valid bit)
  - Reclaim rename tag
    - no valid copy of tag in system!

A Modern OoO Design: Intel Pentium 4

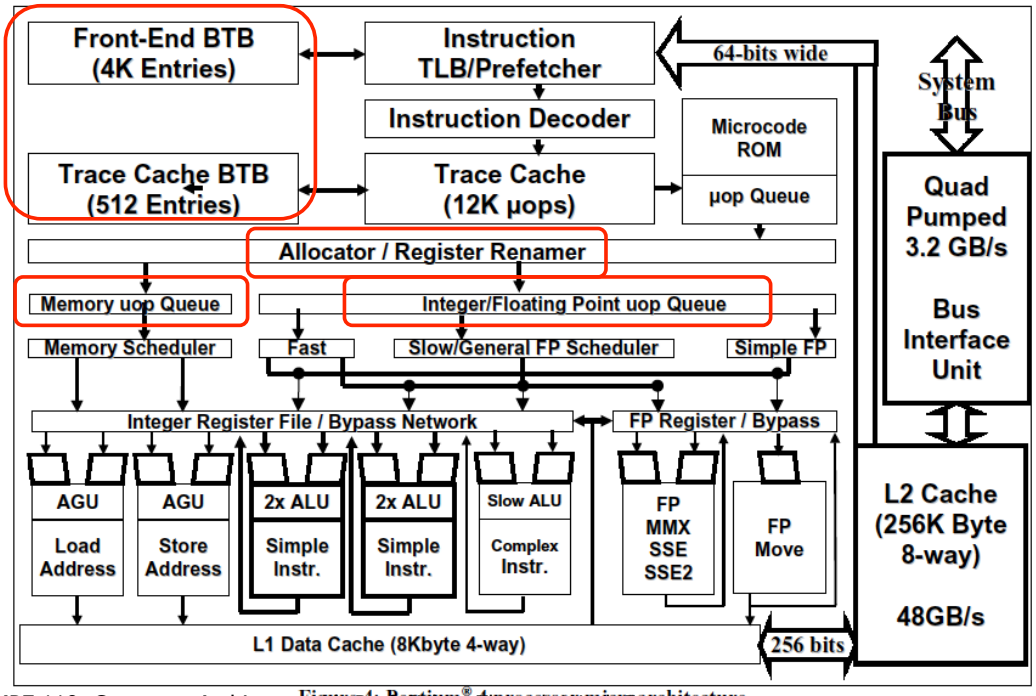
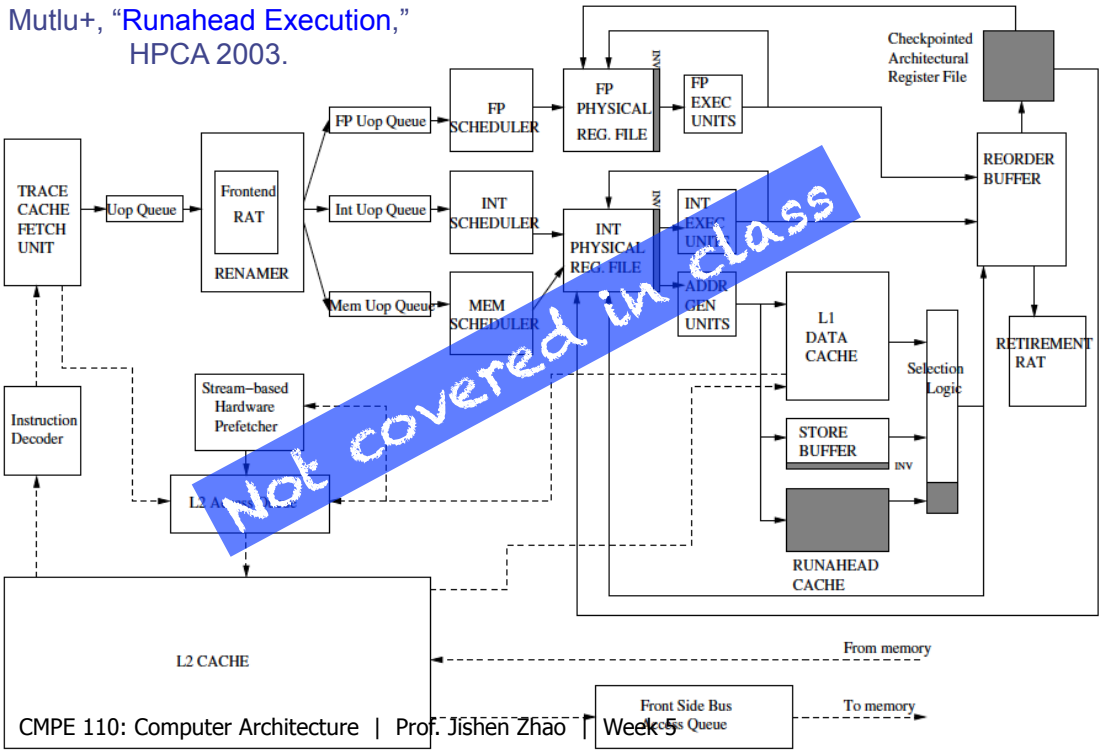


Figure 1: Pentium® 4 processor pipe architecture

Boggs et al., "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, 2001.

Intel Pentium 4 Simplified

Mutlu+, "Runahead Execution," HPCA 2003.



Alpha 21264

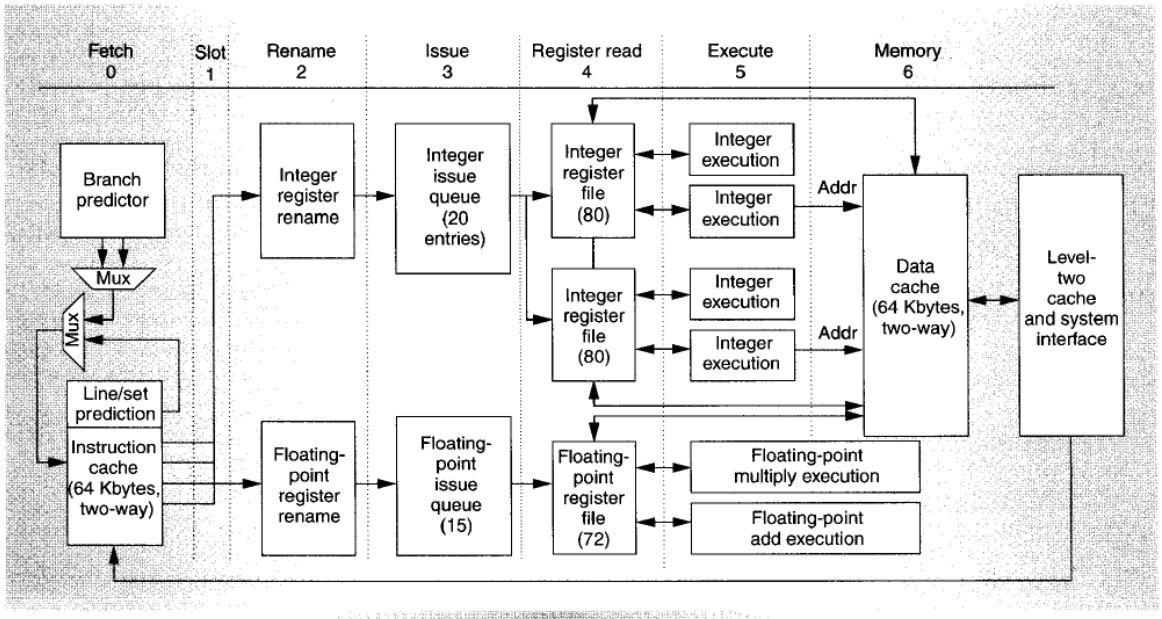
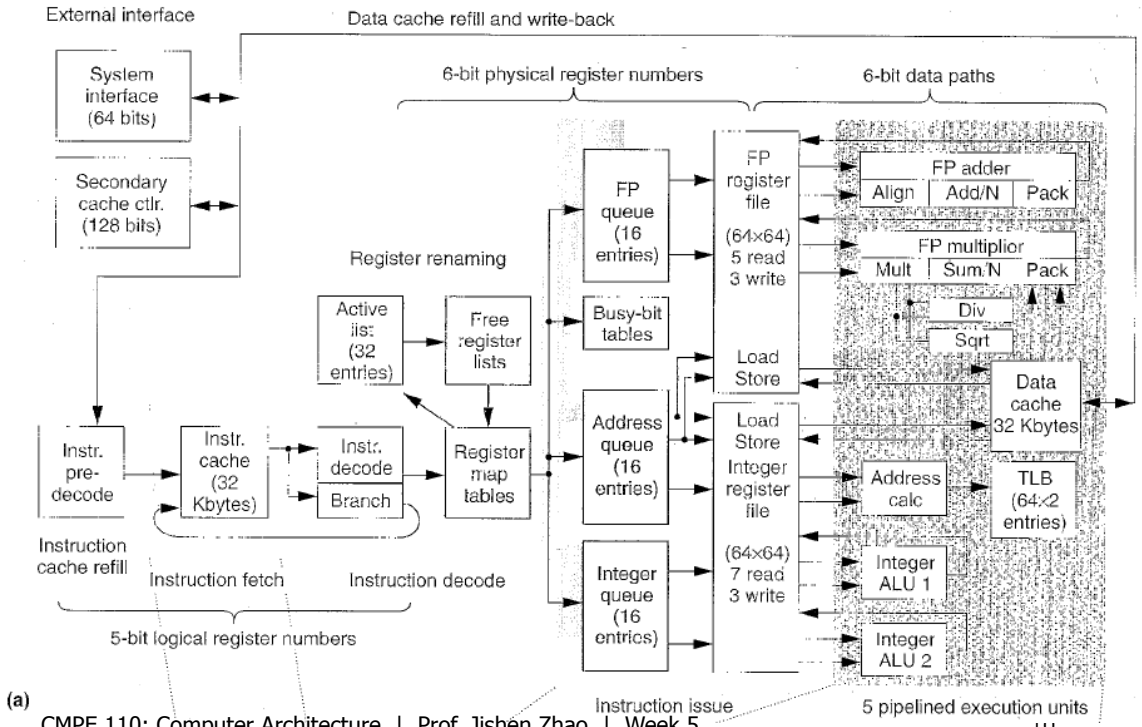


Figure 2. Stages of the Alpha 21264 instruction pipeline.

Kessler, "The Alpha 21264 Microprocessor," IEEE Micro, March-April 1999.

MIPS R10000



Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, April 1996

IBM POWER5

- Kalla et al., "IBM Power5 Chip: A Dual-Core Multithreaded Processor," IEEE Micro 2004.

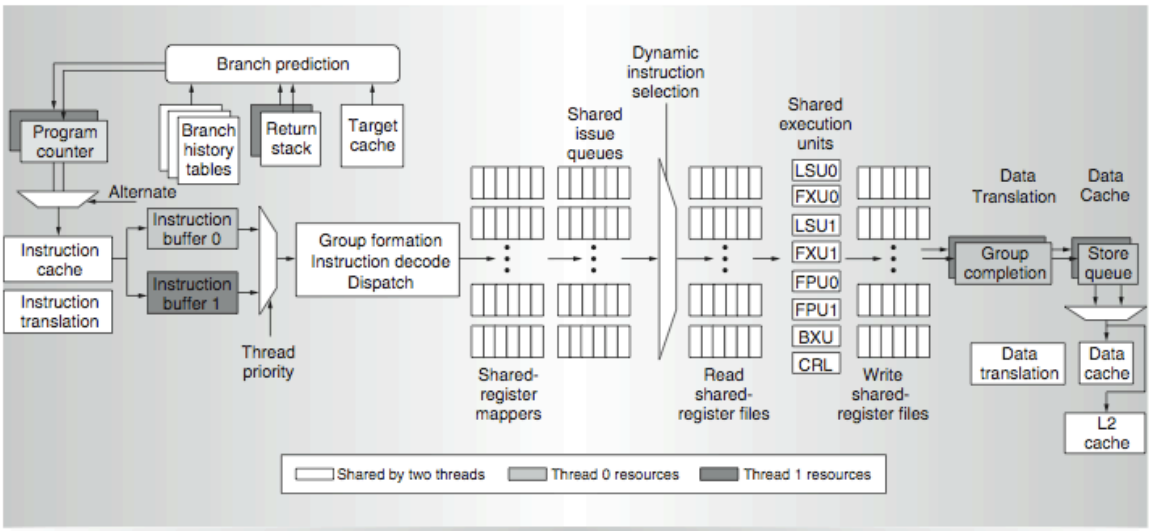


Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

Put it all together: An Exercise

MUL R3 ← R1, R2
ADD R5 ← R3, R4
ADD R7 ← R2, R6
ADD R10 ← R8, R9
MUL R11 ← R7, R10
ADD R5 ← R5, R11

F	D	X	W

- F, D, X, W, i.e., 4-stage pipeline
  - Assume ADD (4 cycle pipelined execute, i.e., X takes 4 cycles), MUL (6 cycle pipelined execute, i.e., X takes 6 cycles)
  - Assume one adder and one multiplier
- How many cycles
  - In a pipelined machine without bypassing (forwarding) →stall on data hazards (Answer: 28 cycles, Quiz 1 question 2)
  - In a pipelined machine with bypassing (Answer: 25 cycles)
  - **In an out-of-order dispatch pipelined with forwarding**

Q1: no solution provided, will be a quiz question

Q2 solution: to distinguish the two different data paths of MUL and ADD, here used E to represent MUL exeuction and X to represent ADD execution

Insn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
MUL R3 <- R1, R2	F	D	E1	E2	E3	E4	E5	E6	W																							
ADD R5 <- R3, R4		F	D	D	D	D	D	D	X1	X2	X3	X4	W																			
ADD R7 <- R2, R6			F	F	F	F	F	F	D	X1	X2	X3	X4	W																			
ADD R10 <- R8, R9									F	D	X1	X2	X3	X4	W																	
MUL R11 <- R7, R10										F	D	D	D	D	E1	E2	E3	E4	E5	E6	W											
ADD R5 <- R5, R11											F	F	F	F	D	D	D	D	D	D	X1	X2	X3	X4	W							

Total: 25 cycles

Q3 solution: to distinguish the two different data paths of MUL and ADD, here used E to represent MUL exeuction and X to represent ADD execution

We use / to represent a WAIT at reservation station; We use // to represent a WAIT at reorder buffer

Insn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
MUL R3 <- R1, R2	F	D	E1	E2	E3	E4	E5	E6	W																							
ADD R5 <- R3, R4		F	D	/	/	/	/	/	X1	X2	X3	X4	W																			
ADD R7 <- R2, R6			F	D	X1	X2	X3	X4	//	//	//	//	//	W																			
ADD R10 <- R8, R9				F	D	X1	X2	X3	X4	//	//	//	//	//	W																	
MUL R11 <- R7, R10					F	D	/	/	/	E1	E2	E3	E4	E5	E6	W																	
ADD R5 <- R5, R11						F	D	/	/	/	/	/	/	/	/	X1	X2	X3	X4	W													

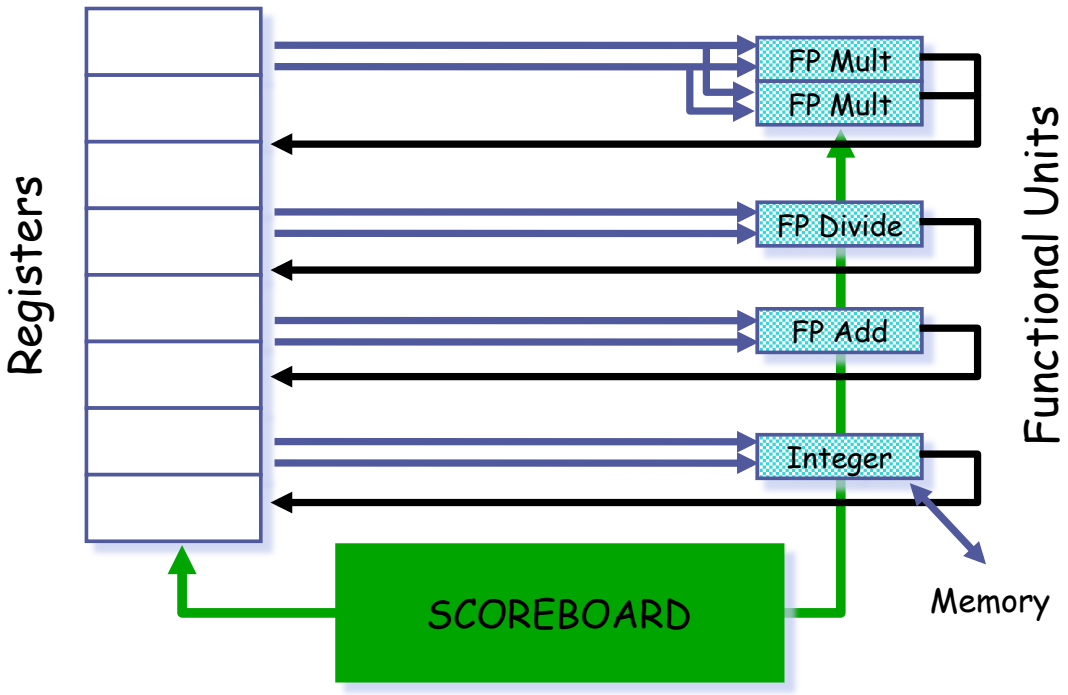
Total: 20 cycles

Note: cycle 10 has a forwarding (bypassing) from Reorder Buffer to execution stage E1

Other than Tomasulo approach

- "Scoreboarding" technique
  - Also a dynamic scheduling scheme
  - Monitors each instruction waiting to be dispatched. Once it determines that all the source operands and the required functional units are available, it dispatches the instruction so that it can be executed.

Scoreboard Architecture(CDC 6600)



Steps with scoreboarding

- Handles all RAW, WAR, and WAW with proper stalls, but allows independent instructions to proceed
- Step 1: Issue (part of original ID stage) – Issue instruction to functional unit iff functional unit is free and no earlier instruction writes to the same destination register (WAW)
- Step 2: Read operands (part of original ID stage) – Wait until source registers become available from earlier instructions through register file (RAW)
- Step 3: Execute (original EXE stage) – Execute instruction and notify scoreboard when done
- Step 4: Write result (original WB stage) – Wait until earlier instructions read operands before writing to register file (WAR)