CSE 490/590 Computer Architecture

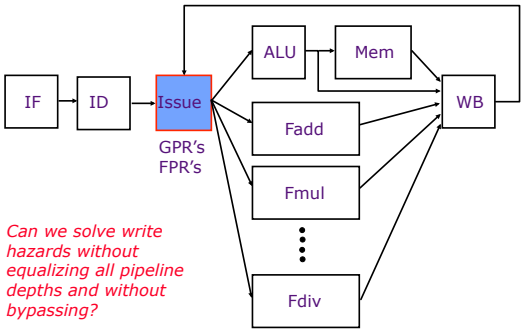
Complex Pipelining II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

- Complex pipelining
  - Multiple functional units with variable access time
- Types of data hazards
  - RAW, WAR, WAW
- Dependency graph
  - How instructions are dependent on each other
  - Basis for out-of-order

Complex Pipeline



Can we solve write hazards without equalizing all pipeline depths and without bypassing?

When is it Safe to Issue an Instruction?

Suppose a data structure keeps track of all the instructions in all the functional units

The following checks need to be made before the Issue stage can dispatch an instruction

- Is the required function unit available?
- Is the input data available? ⇒ RAW?
- Is it safe to write the destination? ⇒ WAR? WAW?
- Is there a structural conflict at the WB stage?

Types of Data Hazards

Consider executing a sequence of

$r_k \leftarrow r_i$ op r_j

type of instructions

Data-dependence

$r_3 \leftarrow r_1$ op r_2
$r_5 \leftarrow r_3$ op r_4
Read-after-Write (RAW) hazard

Anti-dependence

$r_3 \leftarrow r_1$ op r_2
$r_1 \leftarrow r_4$ op r_5
Write-after-Read (WAR) hazard

Output-dependence

$r_3 \leftarrow r_1$ op r_2
$r_3 \leftarrow r_6$ op r_7
Write-after-Write (WAW) hazard

A Data Structure for Correct Issues

Keeps track of the status of Functional Units

Name	*Busy*	*Op*	*Dest*	*Src1*	*Src2*
Int					
Mem					
Add1					
Add2					
Add3					
Mult1					
Mult2					
Div					

The instruction i at the Issue stage consults this table

FU available?	check the busy column
RAW?	search the dest column for i's sources
WAR?	search the source columns for i's destination
WAW?	search the dest column for i's destination

An entry is added to the table if no hazard is detected;
An entry is removed from the table after Write-Back

Simplifying the Data Structure Assuming In-order Issue

Suppose the instruction is not dispatched by the Issue stage if a RAW hazard exists or the required FU is busy, and that operands are latched by functional unit on issue:

Can the dispatched instruction cause a
WAR hazard ?
NO: Operands read at issue
WAW hazard ?
YES: Out-of-order completion

Simplifying the Data Structure ...

No WAR hazard
⇒ no need to keep *src1* and *src2*

The Issue stage does not dispatch an instruction in case of a WAW hazard
⇒ a register name can occur at most once in the *dest* column

WP[reg#] : a bit-vector to record the registers for which writes are pending
These bits are set to true by the Issue stage and set to false by the WB stage
⇒ Each pipeline stage in the FU's must carry the *dest* field and a flag to indicate if it is valid
"the (we, ws) pair"

Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU's availability. (FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set to true by the Issue stage and set to false by the WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?	Busy[FU#]
RAW?	WP[src1] or WP[src2]
WAR?	*cannot arise*
WAW?	WP[dest]

Scoreboard Dynamics

	Functional Unit Status Int(1)	Add(1)	Mult(3)		Div(4)			WB	Registers Reserved for Writes	
t0	I_1				f6				f6	
t1	I_2 f2				f6				f6, f2	
t2						f6		f2	f6, f2	I_2
t3	I_3		f0				f6		f6, f0	
t4				f0				f6	f6, f0	I_1
t5	I_4				f0 f8				f0, f8	
t6					f8			f0	f0, f8	I_3
t7	I_5	f10				f8			f8, f10	
t8							f8	f10	f8, f10	I_5
t9								f8	f8	I_4
t10	I_6	f6							f6	
t11								f6	f6	I_6

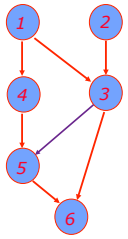
I_1	DIVD	f6,	f6,	f4
I_2	LD	f2,	45(r3)	
I_3	MULTD	f0,	f2,	f4
I_4	DIVD	f8,	f6,	f2
I_5	SUBD	f10,	f0,	f6
I_6	ADDD	f6,	f8,	f2

CSE 490/590 Administrivia

- Midterm on Friday, 3/4
- Review on Wednesday, 3/2
- Project 1 deadline: Friday, 3/11
- Office hours this week: Wed after class until 2pm

In-Order Issue Limitations: *an example*

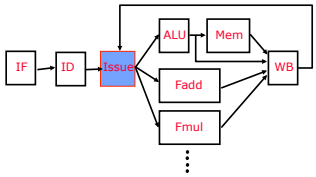
					latency
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		long
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4,	F2,	F8	4
6	ADDD	F10,	F6,	F4	1



In-order: 1 (2,1) . . . . . 2 3 4 4 3 5 . . . 5 6 6

In-order restriction prevents instruction 4 from being dispatched

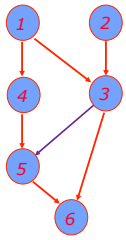
Out-of-Order Issue



- Issue stage buffer holds multiple instructions waiting to issue.
- Decode adds next instruction to buffer if there is space and the instruction does not cause a WAR or WAW hazard.
  - Note: WAR possible again because issue is out-of-order (WAR not possible with in-order issue and latching of input operands at functional unit)
- Any instruction in buffer whose RAW hazards are satisfied can be issued *(for now at most one dispatch per cycle)*. On a write back (WB), new instructions may get enabled.

Issue Limitations: *In-Order and Out-of-Order*

					latency
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		long
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4,	F2,	F8	4
6	ADDD	F10,	F6,	F4	1



In-order: 1 (2,1) . . . . . 2 3 4 4 3 5 . . . 5 6 6
Out-of-order: 1 (2,1) 4 4 . . . . 2 3 . . 3 5 . . . 5 6 6

Out-of-order execution did not allow any significant improvement!

How many instructions can be in the pipeline?

Which features of an ISA limit the number of instructions in the pipeline?

Number of Registers

Out-of-order dispatch by itself does not provide any significant performance improvement!

Overcoming the Lack of Register Names

Floating Point pipelines often cannot be kept filled with small number of registers.

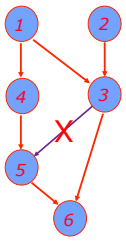
IBM 360 had only 4 floating-point registers

Can a microarchitecture use more registers than specified by the ISA without loss of ISA compatibility ?

Robert Tomasulo of IBM suggested an ingenious solution in 1967 using on-the-fly *register renaming*

Instruction-level Parallelism via Re*naming*

					latency
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		long
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4',	F2,	F8	4
6	ADDD	F10,	F6,	F4'	1



In-order: 1 (2,1) . . . . . 2 3 4 4 3 5 . . . 5 6 6
Out-of-order: 1 (2,1) 4 4 5 . . . 2 (3,5) 3 6 6

Any antidependence can be eliminated by renaming.
(renaming ⇒ additional storage)
Can it be done in hardware? yes!

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252