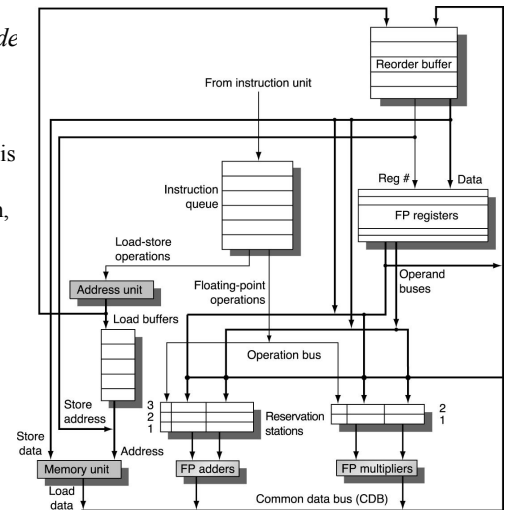


HW support for More ILP

- *Speculation*: allow an instruction to issue that is dependent on branch predicted to be taken *without* any consequences (including exceptions) if branch is not actually taken (“HW undo”)
- Often combined with dynamic scheduling
- Tomasulo: separate *speculative* bypassing of results from real bypassing of results
 - When instruction no longer speculative, write results (*instruction commit*)
 - execute out-of-order but commit in order

Hardware Speculative Execution

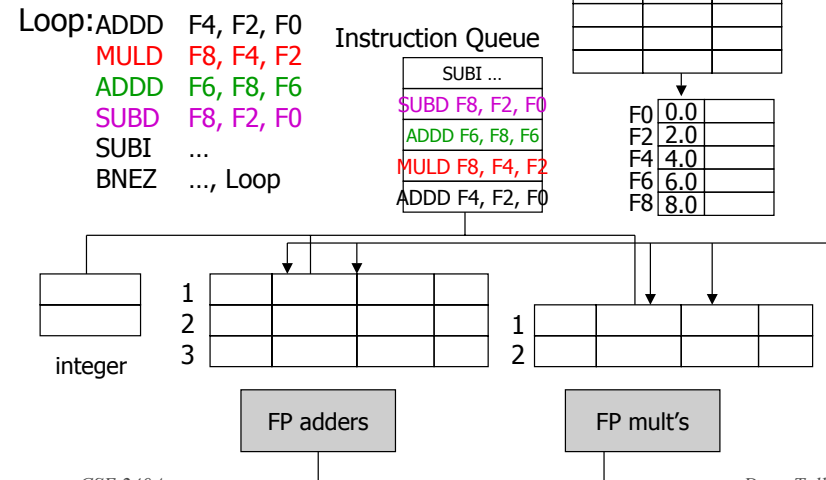
- Need HW buffer for results of uncommitted instructions: *reorder buffer*
 - Reorder buffer can be operand source
 - Once operand commits, result is found in register
 - 3 fields: instr. type, destination, value
 - Use reorder buffer number instead of reservation station
 - Instructions commit in order
 - As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions



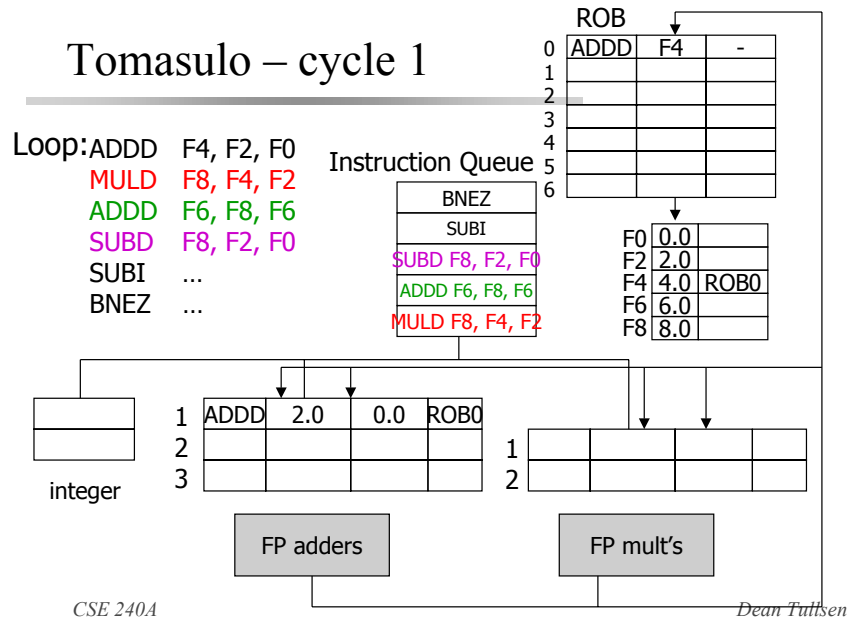
Four Steps of Speculative Tomasulo Algorithm

1. **Issue**—get instruction from FP Op Queue
If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination. Operands may be read from register file or reorder buffer.
2. **Execution**—operate on operands (EX)
When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute
3. **Write result**—finish execution (WB)
Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. **Commit**—update register with reorder result
When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.

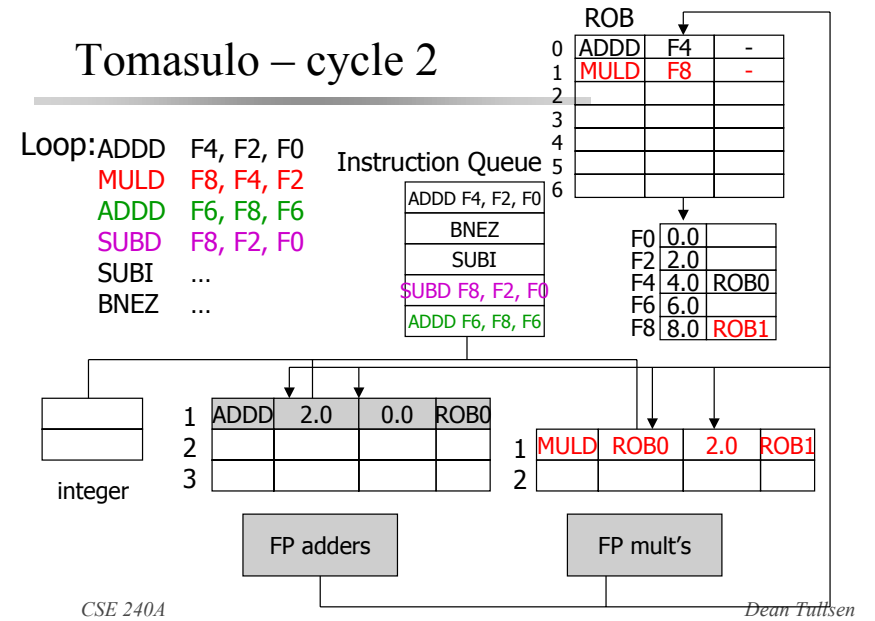
Tomasulo – cycle 0



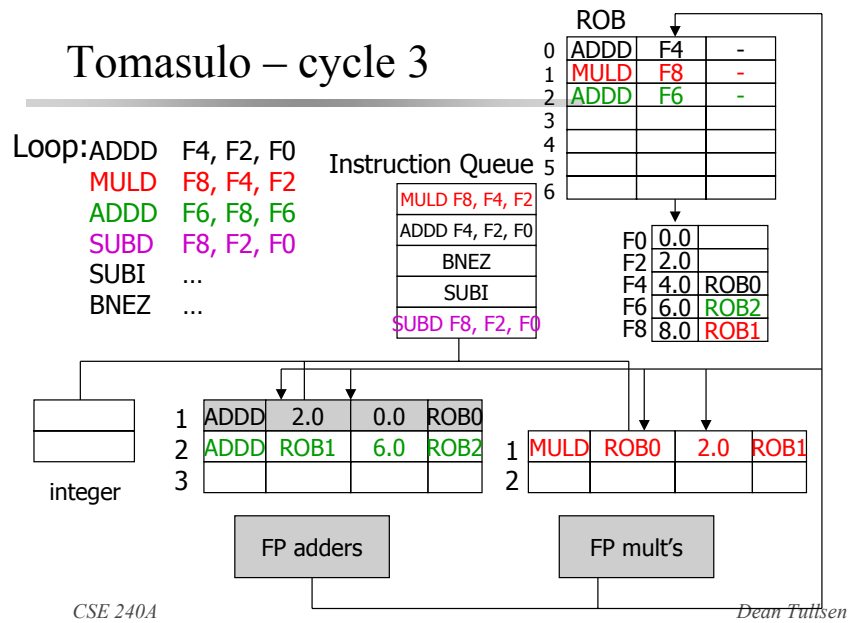
Tomasulo – cycle 1



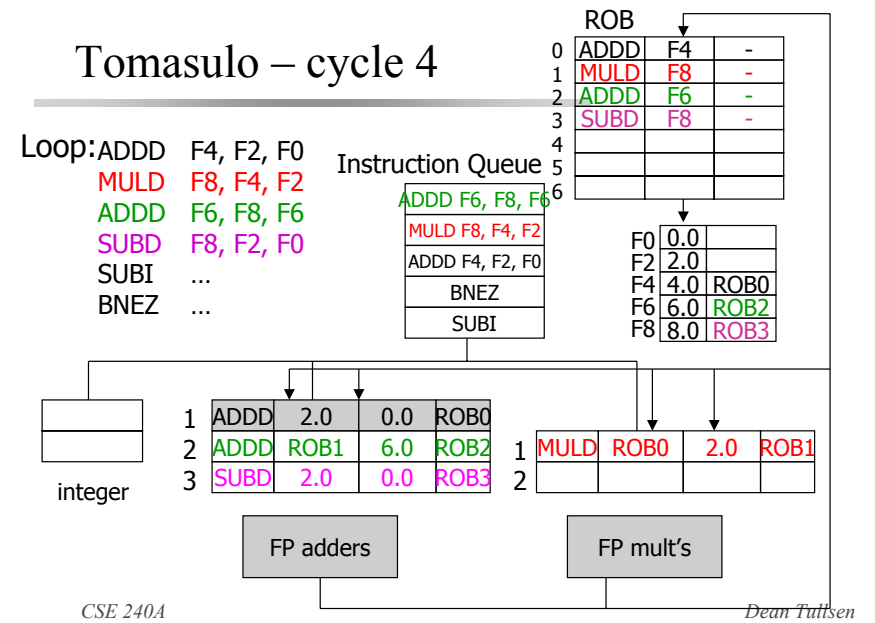
Tomasulo – cycle 2



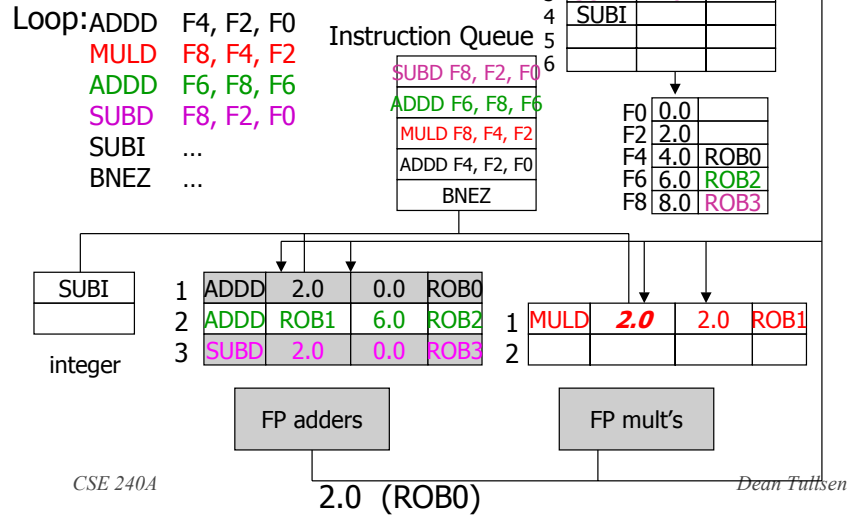
Tomasulo – cycle 3



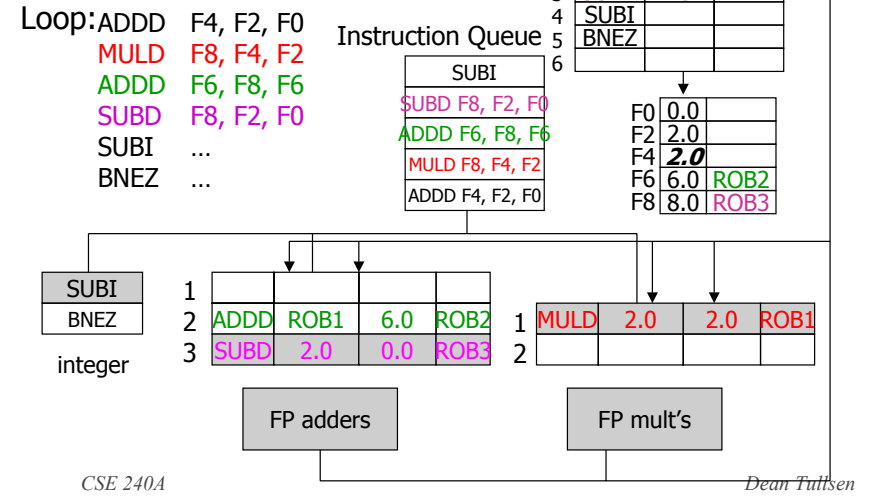
Tomasulo – cycle 4



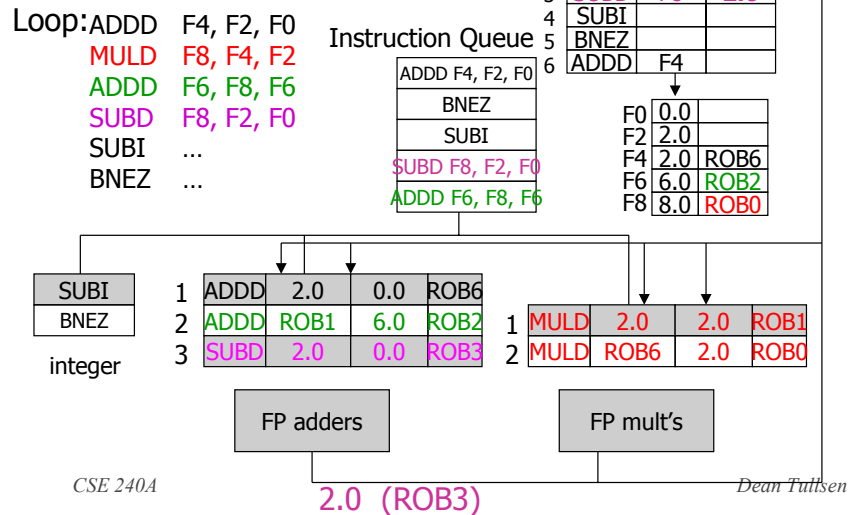
Tomasulo – cycle 5



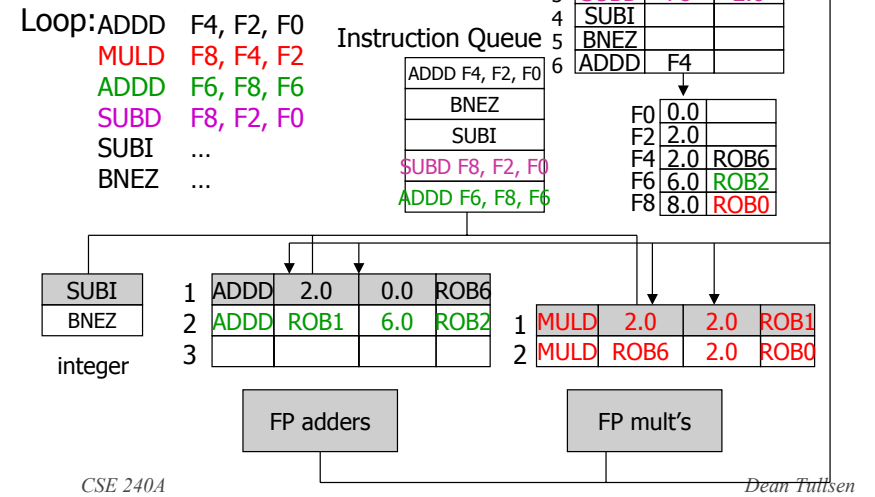
Tomasulo – cycle 6



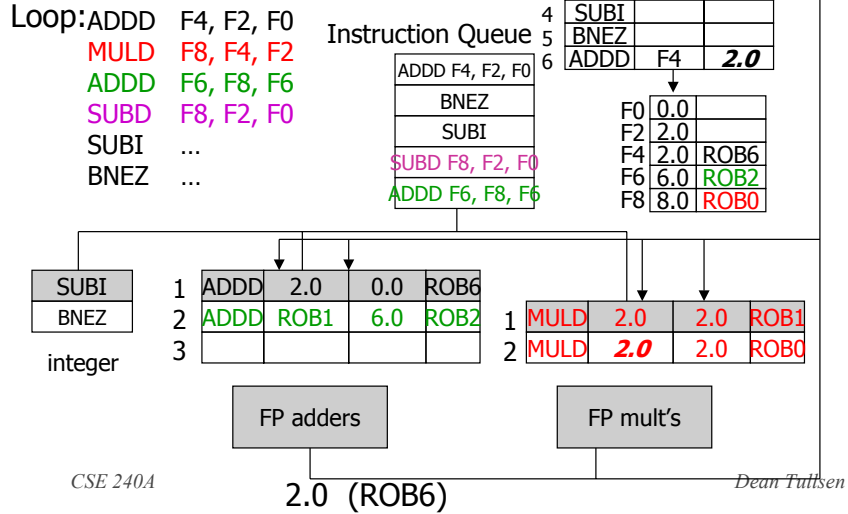
Tomasulo – cycle 8



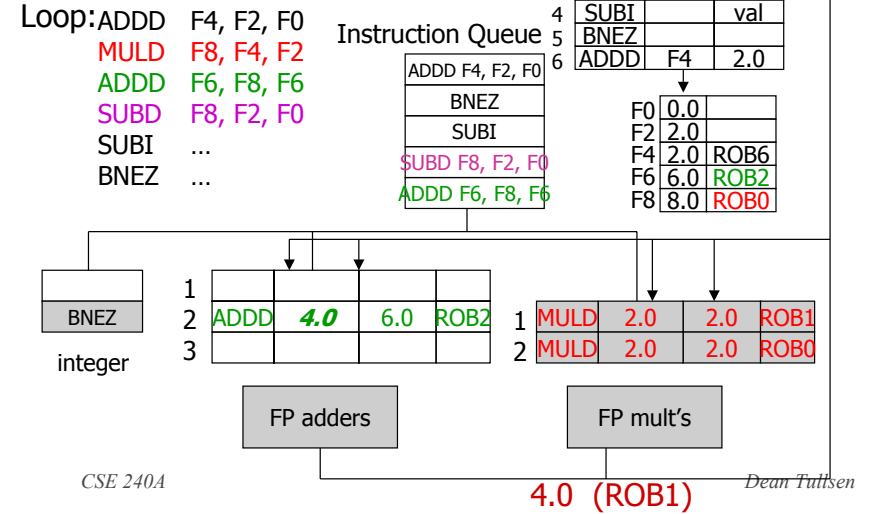
Tomasulo – cycle 9



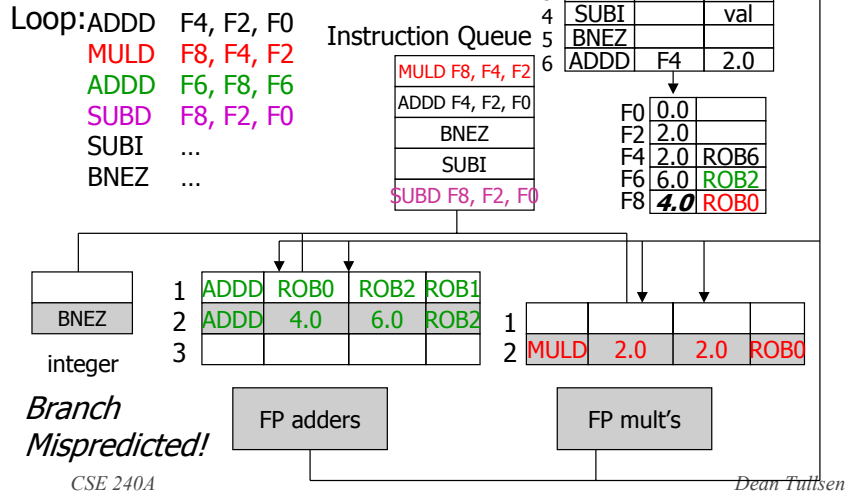
Tomasulo – cycle 11



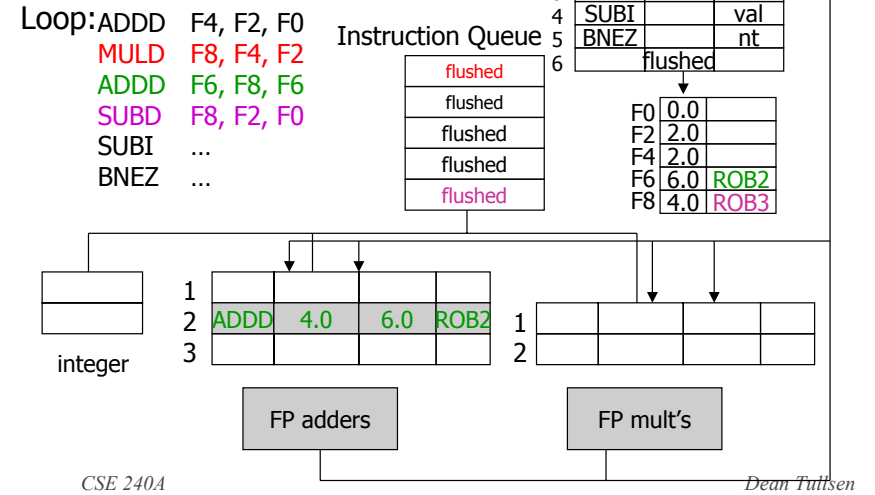
Tomasulo – cycle 15



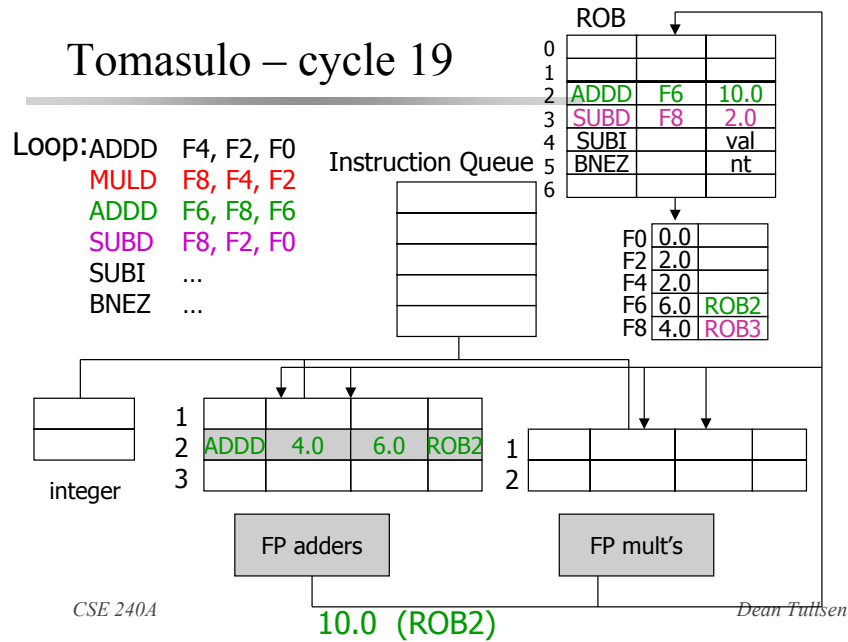
Tomasulo – cycle 16



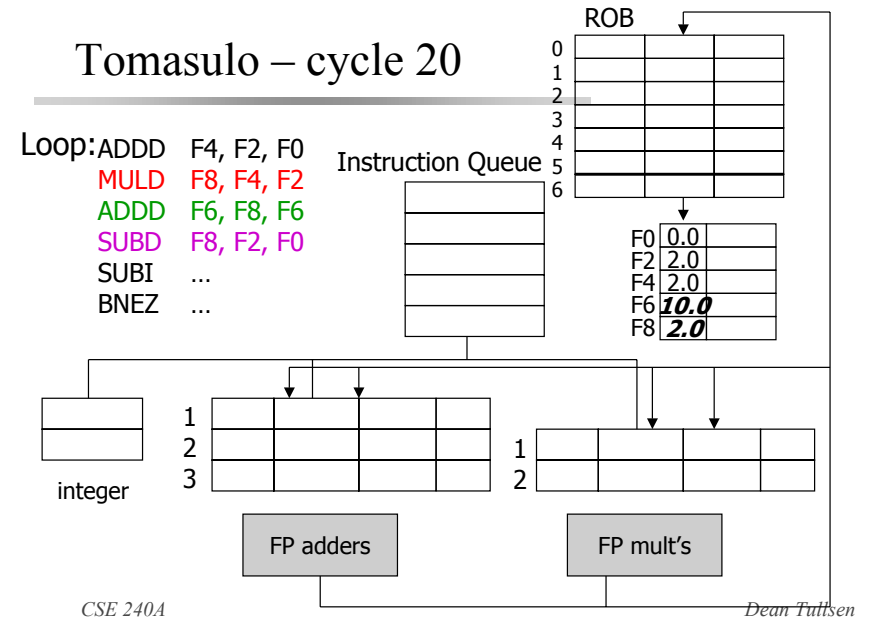
Tomasulo – cycle 17



Tomasulo – cycle 19



Tomasulo – cycle 20

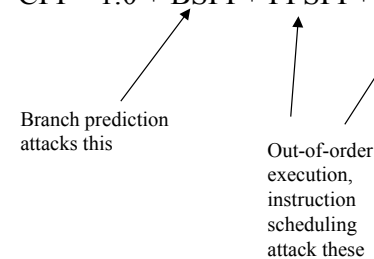


Speculative Execution

- The re-order buffer and in-order commit allow us to flush the speculative instructions from the machine when a misprediction is discovered.
- ROB is another possible source of operands
- ROB can provide precise exceptions in an out-of-order machine
- ROB allows us to ignore exceptions on speculative code.
- Compiler speculation vs. hardware speculation?

Now what?

- $CPI = 1.0 + BSPI + FPSPI + LdSPI$



Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar
 - variable number of instructions issued each cycle
 - parallelism detected in hardware
- Very Long Instruction Words (VLIW)
 - fixed number of instructions issued each cycle
 - parallelism scheduled by the compiler
 - IA-64 (Merced, Itanium)

CSE 240A

Dean Tullsen

Multiple Instruction Issue

or

*The insts go marching two by two, hurrah,
hurrah...*

CSE 240A

Dean Tullsen

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar MIPS: 2 instructions, 1 FP & 1 anything else
 - Fetch 64-bits/clock cycle; Int on left, FP on right
 - Can only issue 2nd instruction if 1st instruction issues

Type	Pipe Stages					
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction		IF	ID	EX	MEM	WB
FP instruction		IF	ID	EX	MEM	WB
Int. instruction			IF	ID	EX	MEM WB
FP instruction			IF	ID	EX	MEM WB

CSE 240A

Dean Tullsen

Superscalar MIPS Implications

- More ports for FP registers to do FP load & FP op in a pair
- 1 cycle load delay expands to 3 instructions in SS
 - instruction in right half can't use it, nor instructions in next slot
- Branch delay also expands to 3

CSE 240A

Dean Tullsen

Unrolled Loop that Minimizes Stalls for Scalar

1	Loop: LD	F0, 0(R1)	LD to ADDD: 1 Cycle
2	LD	F6, -8(R1)	ADDD to SD: 2 Cycles
3	LD	F10, -16(R1)	
4	LD	F14, -24(R1)	
5	ADDD	F4, F0, F2	
6	ADDD	F8, F6, F2	
7	ADDD	F12, F10, F2	
8	ADDD	F16, F14, F2	
9	SD	0(R1), F4	
10	SD	-8(R1), F8	
11	SD	-16(R1), F12	
12	SUBI	R1, R1, #32	
13	BNEZ	R1, LOOP	
14	SD	8(R1), F16 ; 8-32 = -24	

CSE 240A **14 clock cycles, or 3.5 per iteration**

Dean Tullsen

Loop Unrolling in Superscalar

	Integer instruction	FP instruction	Clock cycle
Loop:	LD F0, 0(R1)		1
	LD F6, -8(R1)		2
	LD F10, -16(R1)	ADDD F4, F0, F2	3
	LD F14, -24(R1)	ADDD F8, F6, F2	4
	LD F18, -32(R1)	ADDD F12, F10, F2	5
	SD 0(R1), F4	ADDD F16, F14, F2	6
	SD -8(R1), F8	ADDD F20, F18, F2	7
	SD -16(R1), F12		8
	SD -24(R1), F16		9
	SUBI R1, R1, #40		10
	BNEZ R1, LOOP		11
	SD -32(R1), F20		12

- Unrolled 5 times to avoid delays
- 12 clocks, or 2.4 clocks per iteration

CSE 240A

Dean Tullsen

Dynamic Scheduling in Superscalar

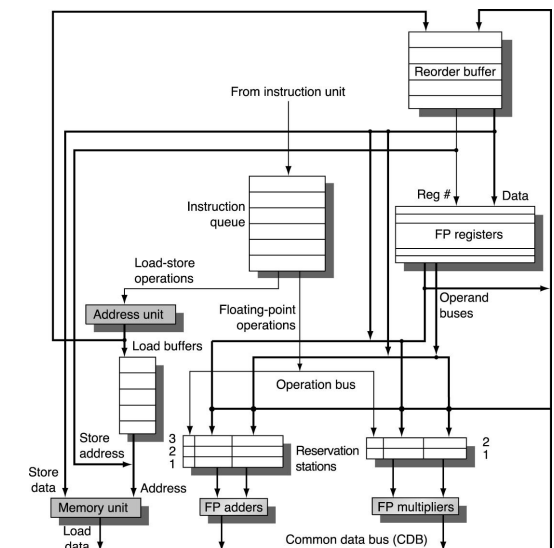
- Dependencies stop instruction issue in In-order SS
- Code compiled for scalar version will run poorly on SS
 - May want code to vary depending on how superscalar
- Simple approach: Combine Tomasulo with the ability to fetch and issue multiple instructions simultaneously
 - simplified if we don't issue instructions that read the same register file in the same cycle.
 - requires multiple CDBs
 - can complicate updating of register bookkeeping if instructions are dependent, but can be done

CSE 240A

Dean Tullsen

Superscalar Dynamic Issue

- Issues/complications?



CSE 240A

Performance of Dynamic SS

Iteration no.	Instructions	Issues	Executes	Writes result
clock-cycle number				
1	LD F0,0(R1)	1	2	4
1	ADDD F4,F0,F2	1	5	8
1	SD 0(R1),F4	2	9	
1	SUBI R1,R1,#8	3	4	5
1	BNEZ R1,LOOP	4	5	
2	LD F0,0(R1)	5	6	8
2	ADDD F4,F0,F2	5	9	12
2	SD 0(R1),F4	6	13	
2	SUBI R1,R1,#8	7	8	9
2	BNEZ R1,LOOP	8	9	

- 4 clocks per iteration

Branches, Decrements still take 1 clock cycle

Limits of Superscalar

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
 - Exactly 50% FP operations
 - No hazards
- If more instructions issue at same time, greater difficulty of decode and issue
 - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue

Superscalar Key Points

- Only way to get CPI < 1 is multiple instruction issue
- SS requires duplicated hardware, more dependence checking
- Without duplication of functional units, will see limited improvement
- SS combined with dynamic scheduling can be powerful

VLIW Processors

- Very Long Instruction Word
- N-wide VLIW issues packets of N instructions simultaneously. Compiler guarantees independence of those N instructions.

```
add r5, r4, r1 | multd f6, f4, f2 | lw r2, 0(r7) | sub r8, r6, r1 | beqz r9, label
sub r11, r5, r1 | addd f8, f0, f6 | sw r5, 8(r7) | nop | beqz r2, l2
```

