Scoreboard Summary

- Speedup 1.7 from compiled code; 2.5 for hand-coded
- Limitations of 6600 scoreboard:
  - No forwarding hardware
  - Limited to instructions in basic block (small *window*)
    - why?
  - Small number of functional units (structural hazards)
    - insts to same fu cannot be reordered
  - Wait for WAR hazards (after EX, before WB)
  - Prevent WAW hazards (in ID)

Another Dynamic Algorithm: Tomasulo Algorithm

- For IBM 360/91 about 3 years after CDC 6600
- Goal: High Performance without special compilers
- Differences between IBM 360 & CDC 6600 ISA
  - IBM has only 2 register specifiers/instr vs. 3 in CDC 6600
  - IBM has 4 FP registers vs. 8 in CDC 6600
  - Implications?

Differences between Tomasulo Algorithm & Scoreboard

- Control & buffers distributed with Function Units vs. centralized in scoreboard; called “reservation stations”

  => instrs schedule themselves
- Registers in instructions replaced by pointers to reservation station buffer

  scoreboard => registers primary operand storage

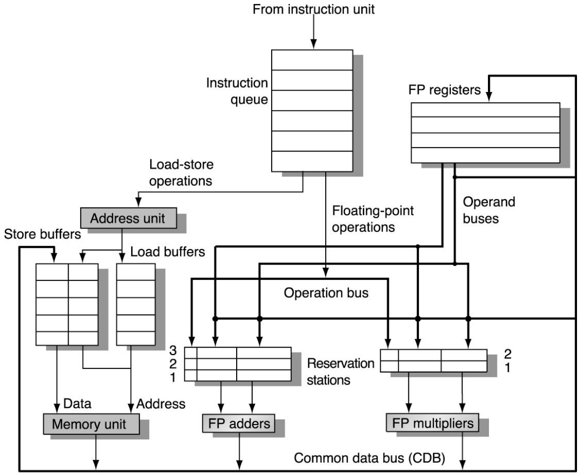
  Tomasulo => reservation stations as operand storage
- HW renaming of registers to avoid WAR, WAW hazards

  Scoreboard => both source registers read together (thus one could not be overwritten while we wait for the other).

  Tomasulo => each register read as soon as available.
- Common Data Bus broadcasts results to all FUs

  RS’s (FU’s), registers, etc. responsible for collecting own data off CDB
- Load and Store Queues treated as FUs as well

Tomasulo Organization



Reservation Station Components

Op—Operation to perform in the unit (e.g., + or –)

Qj, Qk—Reservation stations producing source registers

Vj, Vk—Value of Source operands

Rj, Rk—Flags indicating when Vj, Vk are ready

Busy—Indicates reservation station is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   If reservation station free, the scoreboard issues instr & sends operands (renames registers).
2. Execution—operate on operands (EX)
   When both operands ready then execute;
   if not ready, watch CDB for result
3. Write result—finish execution (WB)
   Write on Common Data Bus to all waiting units;
   mark reservation station available.

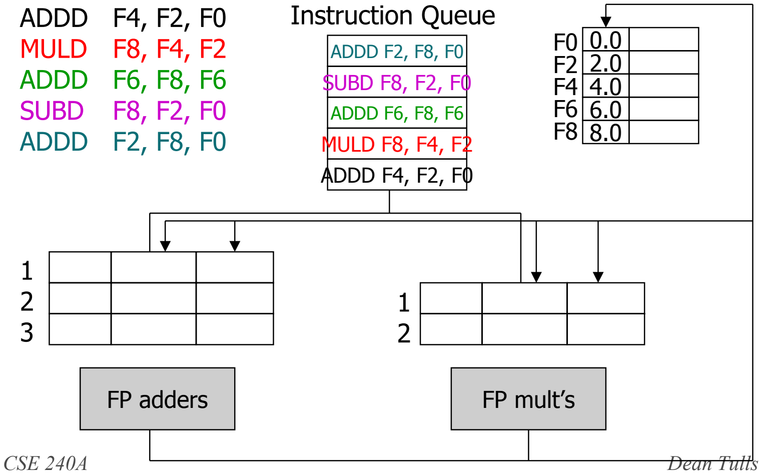
Tomasulo Example

```
ADDD   F4, F2, F0
MULD   F8, F4, F2
ADDD   F6, F8, F6
SUBD   F8, F2, F0
ADDD   F2, F8, F0
```

Multiply takes 10 clocks, add/sub take 4

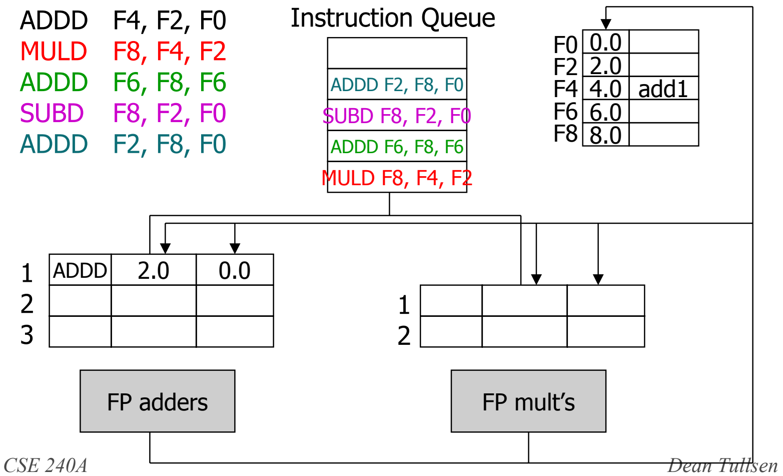
Tomasulo – cycle 0

```
ADDD   F4, F2, F0
MULD   F8, F4, F2
ADDD   F6, F8, F6
SUBD   F8, F2, F0
ADDD   F2, F8, F0
```



Tomasulo – cycle 1

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Instruction Queue

ADDD F2, F8, F0
SUBD F8, F2, F0
ADDD F6, F8, F6
MULD F8, F4, F2

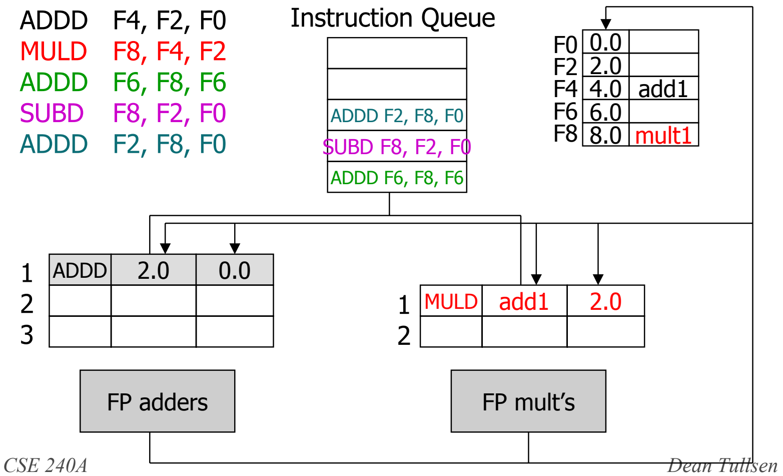
F0	0.0	
F2	2.0	
F4	4.0	add1
F6	6.0	
F8	8.0	

1	ADDD	2.0	0.0
2			
3			

1			
2			

Tomasulo – cycle 2

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Instruction Queue

ADDD F2, F8, F0
SUBD F8, F2, F0
ADDD F6, F8, F6

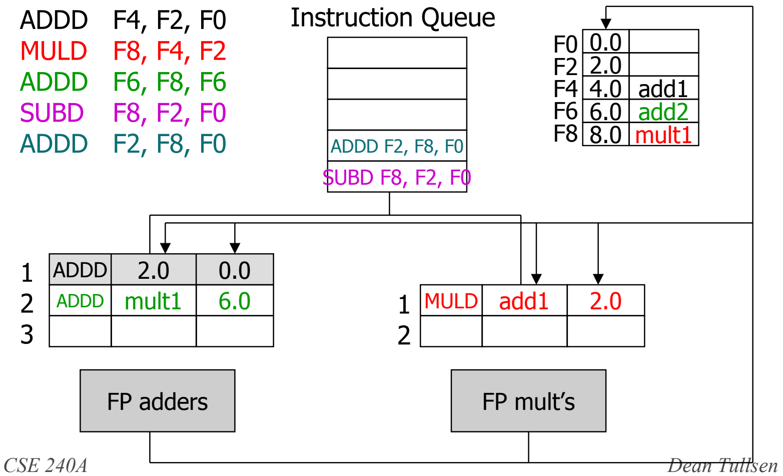
F0	0.0	
F2	2.0	
F4	4.0	add1
F6	6.0	
F8	8.0	mult1

1	ADDD	2.0	0.0
2			
3			

1	MULD	add1	2.0
2			

Tomasulo – cycle 3

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Instruction Queue

ADDD F2, F8, F0
SUBD F8, F2, F0

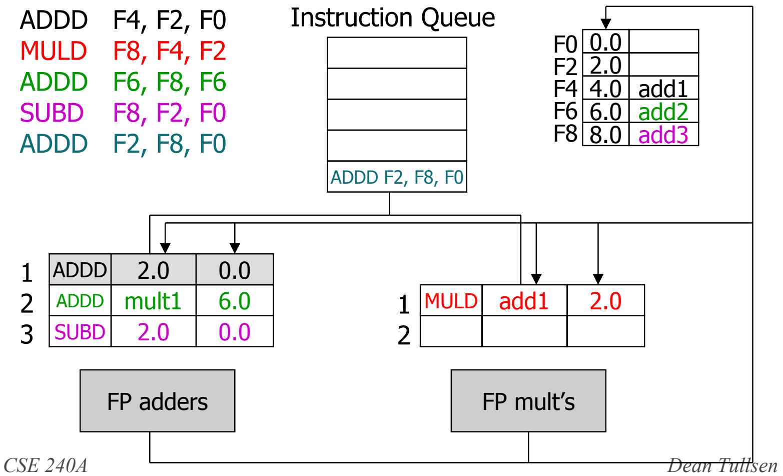
F0	0.0	
F2	2.0	
F4	4.0	add1
F6	6.0	add2
F8	8.0	mult1

1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3			

1	MULD	add1	2.0
2			

Tomasulo – cycle 4

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



Instruction Queue

ADDD F2, F8, F0

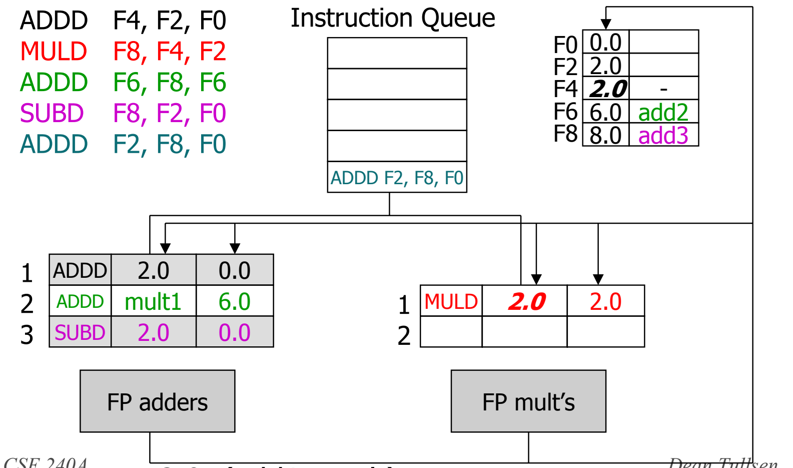
F0	0.0	
F2	2.0	
F4	4.0	add1
F6	6.0	add2
F8	8.0	add3

1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3	SUBD	2.0	0.0

1	MULD	add1	2.0
2			

Tomasulo – cycle 5

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



F0	0.0	
F2	2.0	
F4	***2.0***	-
F6	6.0	add2
F8	8.0	add3

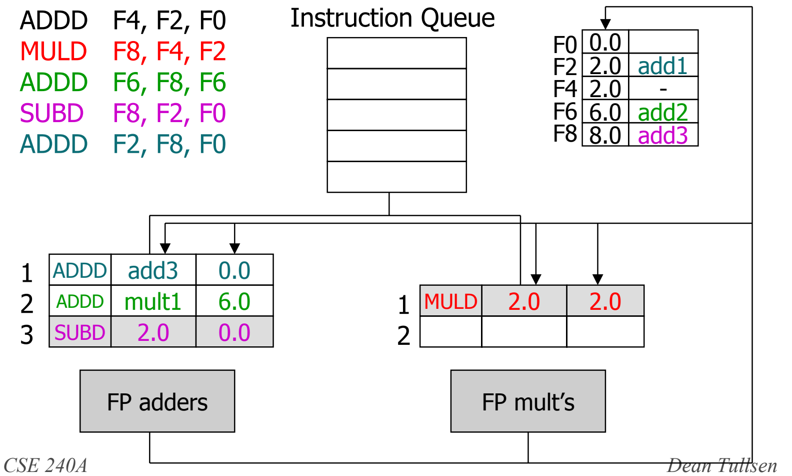
1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3	SUBD	2.0	0.0

1	MULD	***2.0***	2.0
2			

2.0 (add1 result)

Tomasulo – cycle 6

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



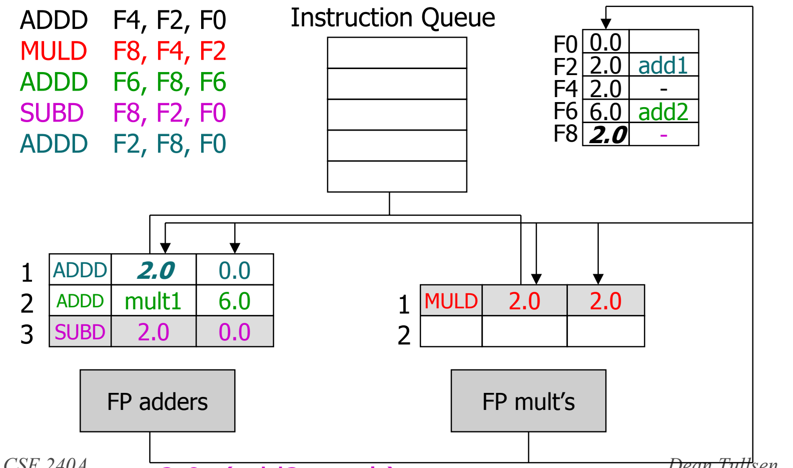
F0	0.0	
F2	2.0	add1
F4	2.0	-
F6	6.0	add2
F8	8.0	add3

1	ADDD	add3	0.0
2	ADDD	mult1	6.0
3	SUBD	2.0	0.0

1	MULD	2.0	2.0
2			

Tomasulo – cycle 8

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



F0	0.0	
F2	2.0	add1
F4	2.0	-
F6	6.0	add2
F8	***2.0***	-

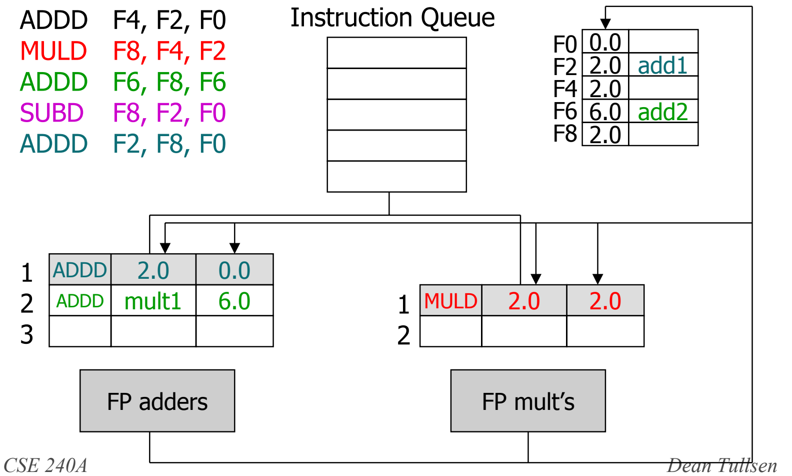
1	ADDD	***2.0***	0.0
2	ADDD	mult1	6.0
3	SUBD	2.0	0.0

1	MULD	2.0	2.0
2			

2.0 (add3 result)

Tomasulo – cycle 9

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0



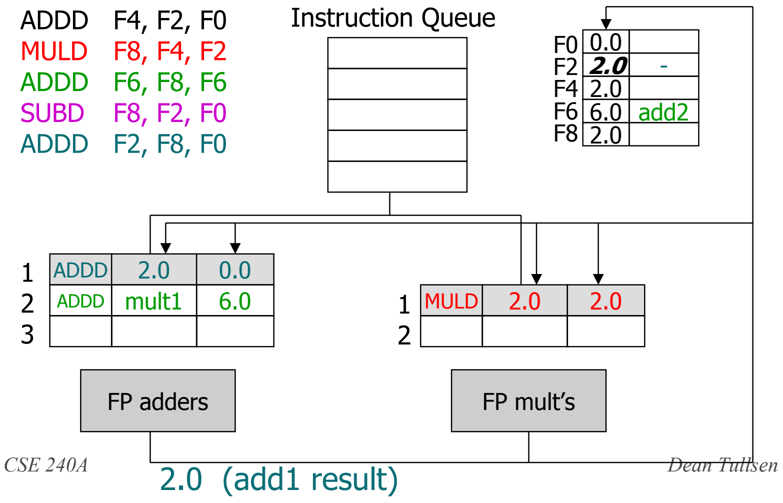
F0	0.0	
F2	2.0	add1
F4	2.0	
F6	6.0	add2
F8	2.0	

1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3			

1	MULD	2.0	2.0
2			

Tomasulo – cycle 12

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



Instruction Queue

F0	0.0	
F2	***2.0***	-
F4	2.0	
F6	6.0	add2
F8	2.0	

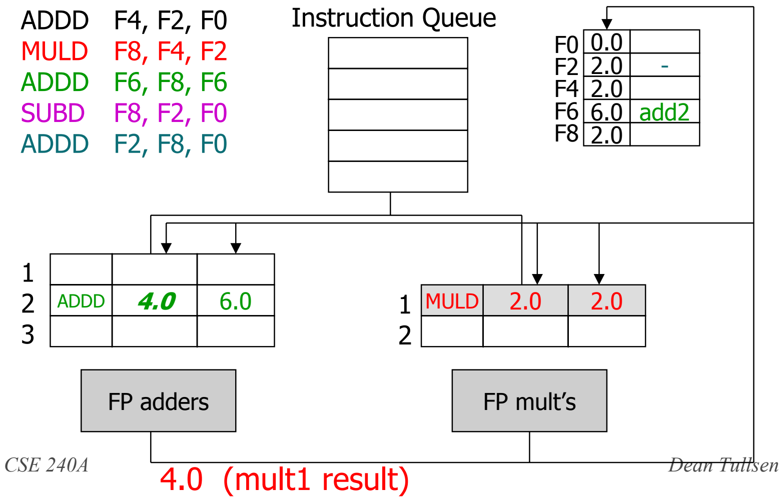
1	ADDD	2.0	0.0
2	ADDD	mult1	6.0
3			

1	MULD	2.0	2.0
2			

2.0 (add1 result)

Tomasulo – cycle 15

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



Instruction Queue

F0	0.0	
F2	2.0	-
F4	2.0	
F6	6.0	add2
F8	2.0	

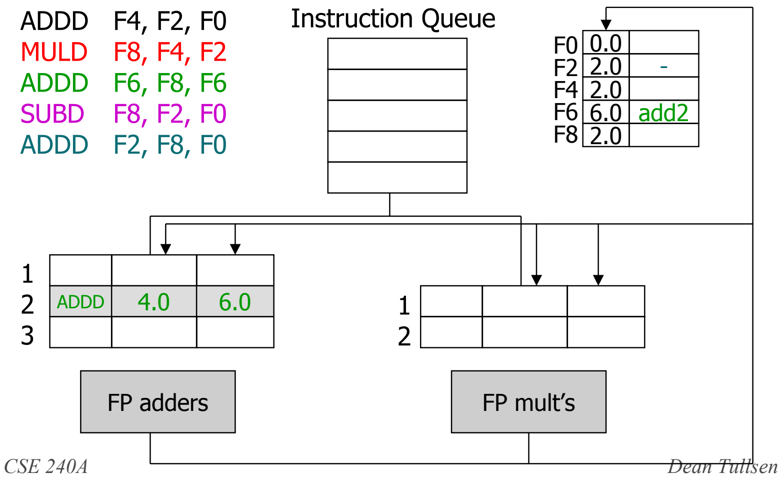
1			
2	ADDD	***4.0***	6.0
3			

1	MULD	2.0	2.0
2			

4.0 (mult1 result)

Tomasulo – cycle 16

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



Instruction Queue

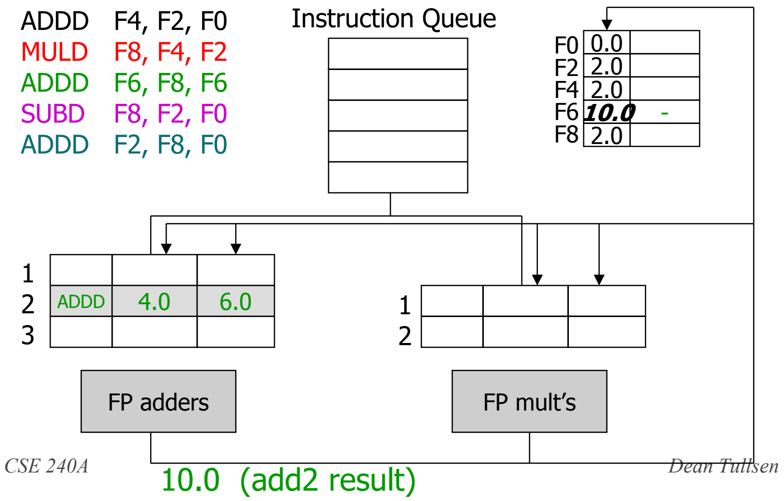
F0	0.0	
F2	2.0	-
F4	2.0	
F6	6.0	add2
F8	2.0	

1			
2	ADDD	4.0	6.0
3			

1			
2			

Tomasulo – cycle 19

ADDD	F4, F2, F0
MULD	F8, F4, F2
ADDD	F6, F8, F6
SUBD	F8, F2, F0
ADDD	F2, F8, F0



Instruction Queue

F0	0.0	
F2	2.0	
F4	2.0	
F6	***10.0***	-
F8	2.0	

1			
2	ADDD	4.0	6.0
3			

1			
2			

10.0 (add2 result)

Tomasulo Summary

- Prevents Register as bottleneck
- Avoids WAR, WAW hazards of Scoreboard
- Allows loop unrolling in HW
- Not limited to basic blocks (provided branch prediction)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming (in what way does the register name *change*?)
  - Load/store disambiguation

Scoreboard vs. Tomasulo, the score

	Scoreboard	Tomasulo
issue	when FU free	when RS free
read operands	from reg file	from reg file, CDB
write operands	to reg file	to CDB
structural hazards	functional units	reservation stations
WAW, WAR hazards	problem	no problem
register renaming	no	yes
instructions completing	no limit	1 / cycle (per CDB)
instructions beginning ex.	1 (per set of read ports)	no limit

Modern Architectures

- Alpha 21264+, MIPS R10K+, Pentium 4 use an ***instruction queue***.
- They use explicit register renaming. Registers are not read until instruction issues (begins execution). Register renaming ensures no conflicts.

```
Div   R5, R4, R2
Add  R7, R5, R1
Sub   R5, R3, R2
Lw    R7, 1000(R5)
```

⇨

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
…	

Modern Architectures

- Alpha 21264+, MIPS R10K+, Pentium 4 use an ***instruction queue***.
- Uses explicit register renaming. Registers are not read until instruction issues (begins execution). Register renaming ensures no conflicts.

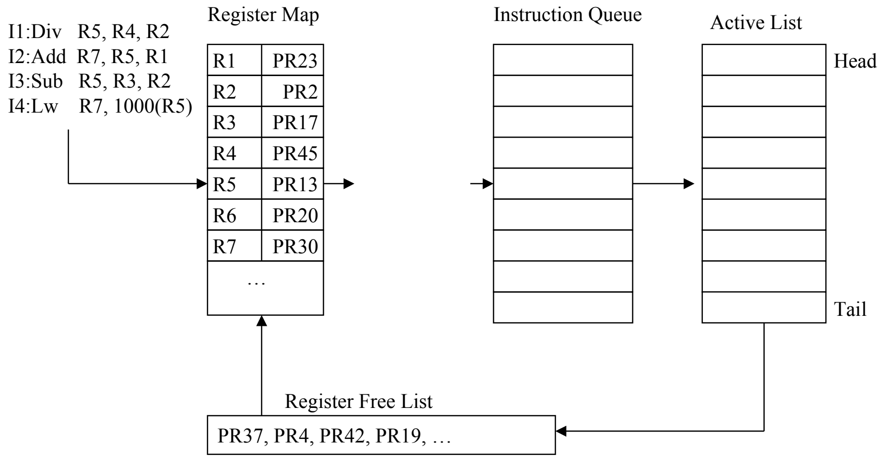
```
Div   R5, R4, R2
Add  R7, R5, R1
Sub   R5, R3, R2
Lw    R7, 1000(R5)
```

⇨

```
Div   PR37, PR45, PR2
Add  PR4, PR37, PR23
Sub   PR42, PR17, PR2
Lw    PR19, 1000(PR42)
```

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	**PR42**
R6	PR20
R7	**PR19**
…	

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

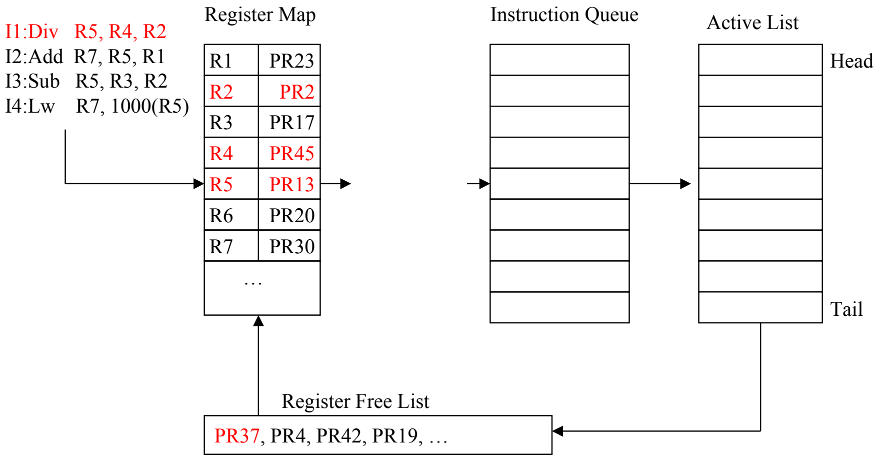
Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
…	

Register Free List

PR37, PR4, PR42, PR19, …

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

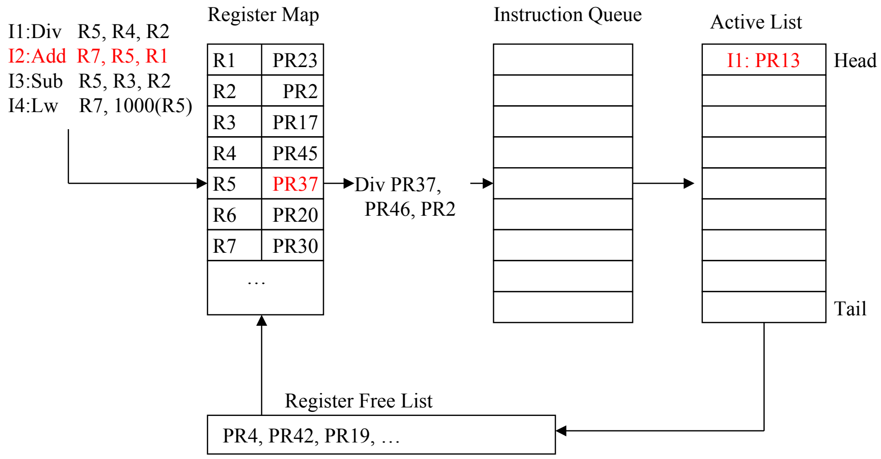
Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR13
R6	PR20
R7	PR30
…	

Register Free List

PR37, PR4, PR42, PR19, …

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	PR30
…	

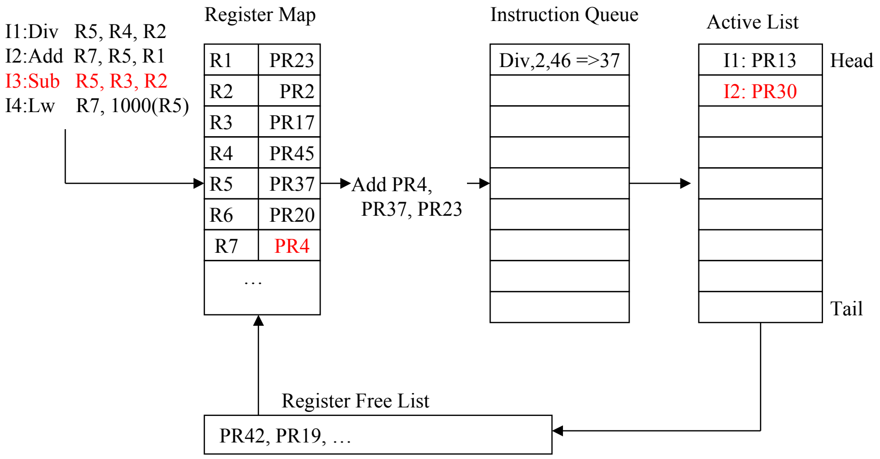
Div PR37, PR46, PR2

I1: PR13

Register Free List

PR4, PR42, PR19, …

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR37
R6	PR20
R7	PR4
…	

Add PR4, PR37, PR23

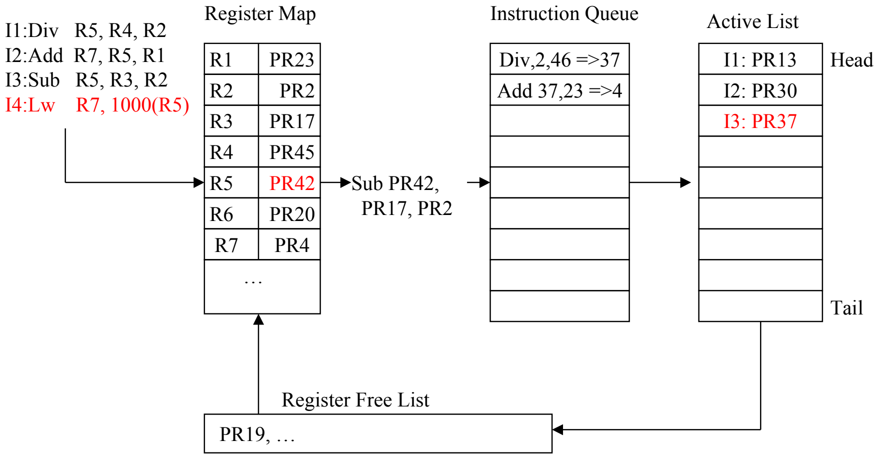
Div,2,46 =>37

I1: PR13
I2: PR30

Register Free List

PR42, PR19, …

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map	
R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR4
…	

Sub PR42, PR17, PR2

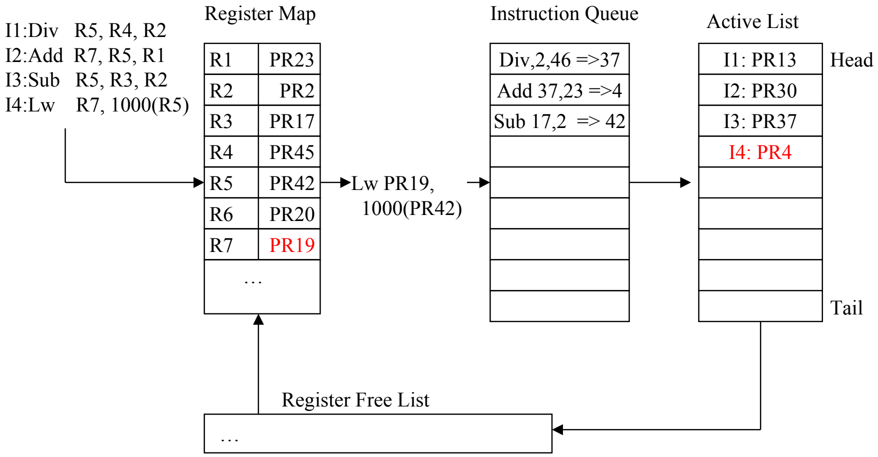
Instruction Queue
Div,2,46 =>37
Add 37,23 =>4

Active List	
I1: PR13	Head
I2: PR30	
I3: PR37	
	Tail

Register Free List

PR19, …

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map	
R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

Lw PR19, 1000(PR42)

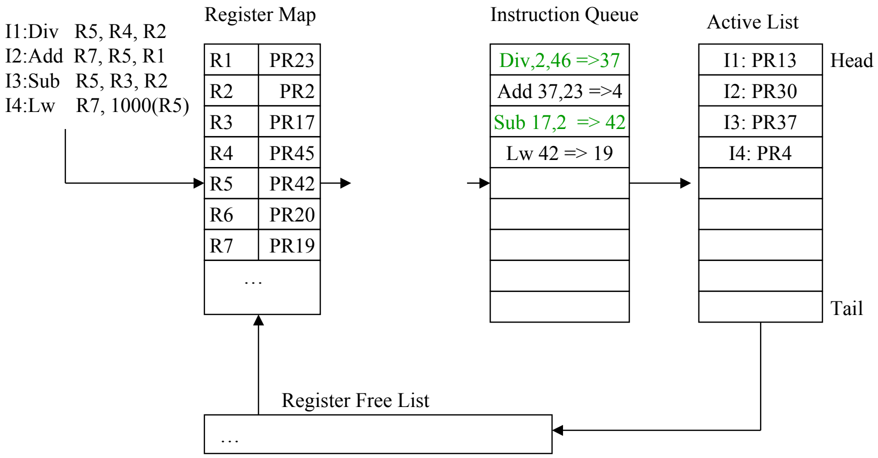
Instruction Queue
Div,2,46 =>37
Add 37,23 =>4
Sub 17,2 => 42

Active List	
I1: PR13	Head
I2: PR30	
I3: PR37	
I4: PR4	
	Tail

Register Free List

…

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map	
R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

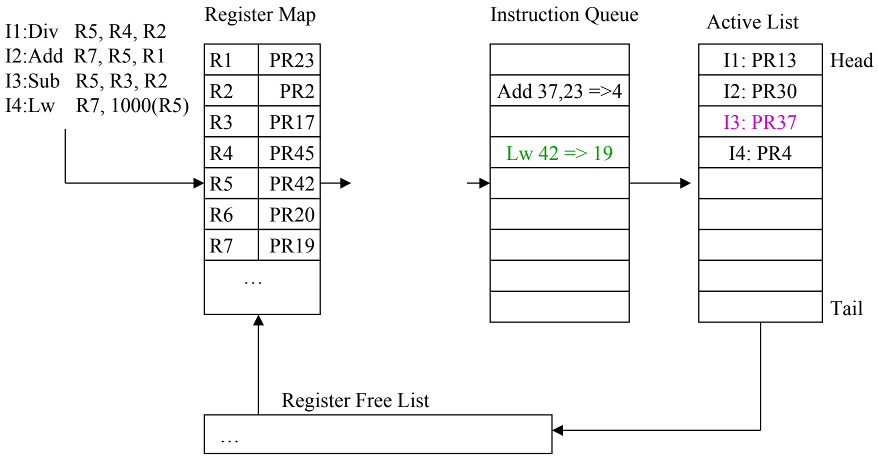
Instruction Queue
Div,2,46 =>37
Add 37,23 =>4
Sub 17,2 => 42
Lw 42 => 19

Active List	
I1: PR13	Head
I2: PR30	
I3: PR37	
I4: PR4	
	Tail

Register Free List

…

MIPS R10000, some detail



I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)

Register Map	
R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

Instruction Queue
Add 37,23 =>4
Lw 42 => 19

Active List	
I1: PR13	Head
I2: PR30	
I3: PR37	
I4: PR4	
	Tail

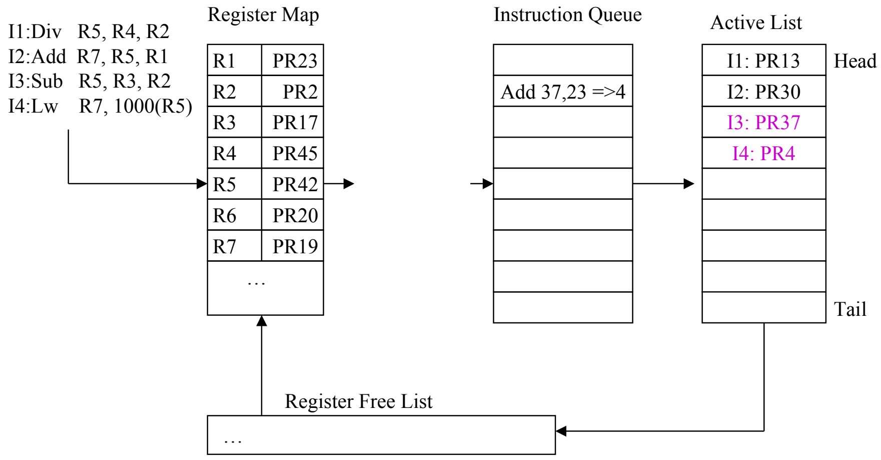
Register Free List

…

I3, producing register 42, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail

I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)



Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

Instruction Queue

Add 37,23 =>4

Active List

I1: PR13 Head
I2: PR30
I3: PR37
I4: PR4
Tail

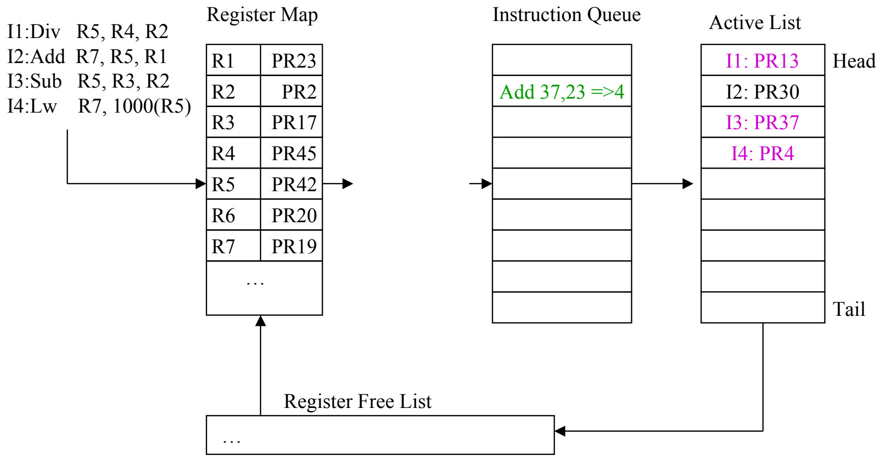
Register Free List

…

I4, producing register 19, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail

I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)



Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

Instruction Queue

Add 37,23 =>4

Active List

I1: PR13 Head
I2: PR30
I3: PR37
I4: PR4
Tail

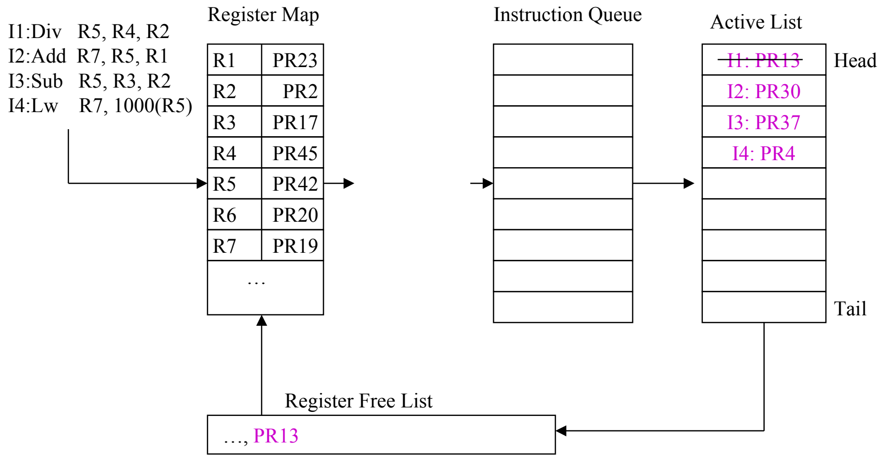
Register Free List

…

I1, producing register 37, completes, broadcasts a completion signal to IQ

MIPS R10000, some detail

I1:Div R5, R4, R2
I2:Add R7, R5, R1
I3:Sub R5, R3, R2
I4:Lw R7, 1000(R5)



Register Map

R1	PR23
R2	PR2
R3	PR17
R4	PR45
R5	PR42
R6	PR20
R7	PR19
…	

Instruction Queue

Active List

~~I1: PR13~~ Head
I2: PR30
I3: PR37
I4: PR4
Tail

Register Free List

…, PR13

I2, producing register 4, completes, broadcasts a completion signal to IQ
I1 commits.

Dynamic Scheduling Key Points

- Dynamic scheduling is code motion in HW.
- Dynamic scheduling can do things SW scheduling (static scheduling) cannot.
- Scoreboard, Tomasulo have various tradeoffs
- Register renaming eliminates WAW, WAR dependencies.
- To get cross-iteration parallelism, we need to eliminate WAW, WAR dependencies.