Review: Summary

- Instruction Level Parallelism in SW or HW
- Loop level parallelism is easiest to see
- SW parallelism dependencies defined for program, hazards if HW cannot resolve dependencies
- SW dependencies/Compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine
- Scoreboard is one dynamic HW scheme
- Key idea: Allow instructions behind stall to proceed
- Enables out-of-order execution => out-of-order completion => watch for WAR, WAR hazards

© Patterson 1994

FP Loop Showing Stalls

```
1 Loop: LD    F0,0(R1)   ;F0=vector element
2       stall
3       ADDD  F4,F0,F2   ;add scalar in F2
4       stall
5       stall
6       SD    0(R1),F4   ;store result
7       SUBI  R1,R1,8    ;decrement pointer 8B (DW)
8       BNEZ  R1,Loop    ;branch R1!=zero
9       stall            ;delayed branch slot
```

Instruction producing result	*Instruction using result*	*Latency in clock cycles*
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1

- **Rewrite code to minimize stalls?**

© Patterson 1994

Unrolled Loop that minimizes stalls

```
1 Loop: LD     F0,0(R1)
2       LD     F6,-8(R1)
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
```

- 14 clock cycles, or 3.5 per iteration

- **What assumptions made when moved code?**
- OK to move store past SUBI even though changes register
- OK to move loads before stores: get right data?
- When is it safe for compiler to do such changes?

© Patterson 1994

3 Parts of the Scoreboard

1. Instruction status—which of 4 steps the instruction is in.
2. Functional unit status—Indicates the state of the functional unit (FU). 9 fields for each functional unit:
   - Busy—Indicates whether the unit is busy or not
   - Op—Operation to perform in the unit (e.g., + or –)
   - Fi—Destination register
   - Fj, Fk—Source-register numbers
   - Qj, Qk—Functional units producing source registers Fj, Fk
   - Rj, Rk—Flags indicating when Fj, Fk are ready
3. Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

© Patterson 1994

4 Stages of Scoreboard Control

1. **Issue—decode instructions & check for structural hazards**

   If functional unit free & no WAW, the scoreboard issues instr & updates bookeeping.

2. **Read operands—wait until no data hazards, then read operands (ID2)**

   If no RAW, read the operands and begin execution.

3. **Execution—operate on operands (EX)**

   FU begins execution. When the result is ready, it notifies the scoreboard that it has completed execution.

4. **Write result—finish execution (WB)**

   If no WAR hazards, it writes results & updates bookeeping of RAW instructions.

© Patterson 1994

Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← 'D'; Fj(FU)← 'S1'; Fk(FU)← 'S2'; Qj← Result('S1'); Qk← Result('S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

© Patterson 1994

Scoreboard Summary

- Speedup 1.7 from compiler; 2.5 by hand BUT slow memory (no cache)
- Limitations of 6600 scoreboard
  - No forwarding
  - Limited to instructions in basic block (small *window*)
  - Number of functional units(structural hazards)
  - wait for WAR hazards
  - prevent WAW hazards

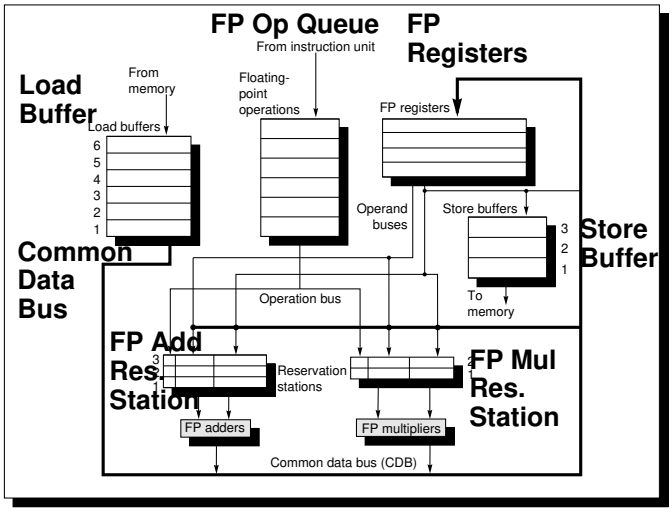
© Patterson 1994

Another Dynamic Algorithm: Tomasulo Algorithm

- For IBM 360/91 about 3 years after CDC 6600
- Goal was to have 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
- Differences between Tomasulo Algorithm & Scoreboard
  - Control & buffers distributed with Function Units vs. centralized in scoreboard; called “reservation stations”
  - Registers in instructions replaced by pointers to reservation station buffer
  - HW renaming of registers to avoid WAR, WAW hazards
  - Common Data Bus broadcasts results to all FUs
  - Load and Stores treated as FUs as well

© Patterson 1994

Tomasulo Organization



© Patterson 1994

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 Fj, Fk are ready

Busy—Indicates reservation station and FU 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.

© Patterson 1994

3 Stages of Tomasolu 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 awaiting units

© Patterson 1994