EECS 252 Graduate Computer Architecture

Lec 7 – Instruction Level Parallelism

David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley
http://www.eecs.berkeley.edu/~pattrsn
http://vlsi.cs.berkeley.edu/cs252-s06

Review from Last Time

• 4 papers: All about where to draw line between HW and SW
• IBM set foundations for ISAs since 1960s
  – 8-bit byte
  – Byte-addressable memory (as opposed to word-addressable memory)
  – 32-bit words
  – Two’s complement arithmetic (but not the first processor)
  – 32-bit (SP) / 64-bit (DP) Floating Point format and registers
  – Commercial use of microcoded CPUs
  – Binary compatibility / computer family
• B5000 very different model: HLL only, stack, Segmented VM
• IBM paper made case for ISAs good for microcoded processors ⇒ leading to CISC
• Berkeley paper made the case for ISAs for pipeline + cache microprocessors (VLSI) ⇒ leading to RISC
• Who won RISC vs. CISC? VAX is dead. Intel 80x86 on desktop, RISC in embedded, Servers x86 and RISC

Outline

• ILP
• Compiler techniques to increase ILP
• Loop Unrolling
• Static Branch Prediction
• Dynamic Branch Prediction
• Overcoming Data Hazards with Dynamic Scheduling
• (Start) Tomasulo Algorithm
• Conclusion

Recall from Pipelining Review

• Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls
  – Ideal pipeline CPI: measure of the maximum performance attainable by the implementation
  – Structural hazards: HW cannot support this combination of instructions
  – Data hazards: Instruction depends on result of prior instruction still in the pipeline
  – Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps)
Instruction Level Parallelism

- Instruction-Level Parallelism (ILP): overlap the execution of instructions to improve performance
- 2 approaches to exploit ILP:
  1. Rely on hardware to help discover and exploit the parallelism dynamically (e.g., Pentium 4, AMD Opteron, IBM Power) and
  2. Rely on software technology to find parallelism, statically at compile-time (e.g., Itanium 2)
- Next 4 lectures on this topic

Basic Block (BB) ILP is quite small
- BB: a straight-line code sequence with no branches in except to the entry and no branches out except at the exit
- average dynamic branch frequency 15% to 25%
  => 4 to 7 instructions execute between a pair of branches
- Plus instructions in BB likely to depend on each other

To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks
- Simplest: loop-level parallelism to exploit parallelism among iterations of a loop. E.g.,
  \[
  \text{for } (i=1; i<=1000; i=i+1) \\
  x[i] = x[i] + y[i];
  \]

Loop-Level Parallelism

- Exploit loop-level parallelism to parallelism by “unrolling loop” either by
  1. dynamic via branch prediction or
  2. static via loop unrolling by compiler
    (Another way is vectors, to be covered later)
- Determining instruction dependence is critical to Loop Level Parallelism
- If 2 instructions are
  - parallel, they can execute simultaneously in a pipeline of arbitrary depth without causing any stalls (assuming no structural hazards)
  - dependent, they are not parallel and must be executed in order, although they may often be partially overlapped

Data Dependence and Hazards

- Instr\textsubscript{j} is data dependent (aka true dependence) on Instr\textsubscript{i}
  1. Instr\textsubscript{j} tries to read operand before Instr\textsubscript{i} writes it
  2. or Instr\textsubscript{j} is data dependent on Instr\textsubscript{k} which is dependent on Instr\textsubscript{i}
- If two instructions are data dependent, they cannot execute simultaneously or be completely overlapped
- Data dependence in instruction sequence \(\Rightarrow\) data dependence in source code \(\Rightarrow\) effect of original data dependence must be preserved
- If data dependence caused a hazard in pipeline, called a Read After Write (RAW) hazard
ILP and Data Dependencies, Hazards

- HW/SW must preserve **program order**: order instructions would execute in if executed sequentially as determined by original source program
  - Dependences are a property of programs
- Presence of dependence indicates **potential** for a hazard, but actual hazard and length of any stall is property of the pipeline
- Importance of the data dependencies
  1) indicates the possibility of a hazard
  2) determines order in which results must be calculated
  3) sets an upper bound on how much parallelism can possibly be exploited
- HW/SW goal: exploit parallelism by preserving program order only where it affects the outcome of the program

Name Dependence #1: Anti-dependence

- **Name dependence**: when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name; 2 versions of name dependence
  - Instr\textsubscript{J} writes operand **before** Instr\textsubscript{I} reads it
  - Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”
  - If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard

Name Dependence #2: Output dependence

- Instr\textsubscript{J} writes operand **before** Instr\textsubscript{I} writes it.
  - Called an “output dependence” by compiler writers. This also results from the reuse of name “r1”
  - If anti-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order
  - if p1 {
    - S1;
  };
  - if p2 {
    - S2;
  }
  - S1 is control dependent on p1, and S2 is control dependent on p2 but not on p1.
Control Dependence Ignored

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program
- Instead, 2 properties critical to program correctness are
  1) exception behavior and
  2) data flow

Exception Behavior

- Preserving exception behavior
  ⇒ any changes in instruction execution order must not change how exceptions are raised in program
  (⇒ no new exceptions)
- Example:
  \[
  \begin{align*}
  &\text{DADDU} & R2, R3, R4 \\
  &\text{BEQZ} & R2, L1 \\
  &\text{LW} & R1, 0 (R2) \\
  \end{align*}
  \]
  \[L1:\]
  - (Assume branches not delayed)
  - Problem with moving LW before BEQZ?

Data Flow

- Data flow: actual flow of data values among instructions that produce results and those that consume them
  - branches make flow dynamic, determine which instruction is supplier of data
- Example:
  \[
  \begin{align*}
  &\text{DADDU} & R1, R2, R3 \\
  &\text{BEQZ} & R4, L \\
  &\text{DSUBU} & R1, R5, R6 \\
  &\text{L:} & \ldots \\
  &\text{OR} & R7, R1, R8
  \end{align*}
  \]
  - OR depends on DADDU or DSUBU?
  - Must preserve data flow on execution

CS 252 Administrivia

- 1 Page project writeups Due this Sunday
  - students working on the RAMP project should go to 253 Cory or 387 Soda to update their cardkey access for 125 Cory
  - RAMP Blue meeting today at 3:30 in 6th floor Soda Alcove
- Reading Assignment: Chapter 2 today, Chapter 3 following next Wednesday
  - Try 30 minute discussion after one hour lecture (similar to ISA discussion)
  - Send email to TA by Friday, will be posted on Saturday, review before discussion on Monday
  - Read pages 1-35 (> ½ of paper is figures)
  - In your comments, rank in order of importance alias analysis, branch prediction, jump prediction, register renaming, and speculative execution
  - In your comments, mention what are limits to this study of limits of ILP?
Computers in the News

Who said this?
A. Jimmy Carter, 1979
B. Bill Clinton, 1996
C. Al Gore, 2000
D. George W. Bush, 2006

"Again, I'd repeat to you that if we can remain the most competitive nation in the world, it will benefit the worker here in America. People have got to understand, when we talk about spending your taxpayers' money on research and development, there is a correlating benefit, particularly to your children. See, it takes a while for some of the investments that are being made with government dollars to come to market. I don't know if people realize this, but the Internet began as the Defense Department project to improve military communications. In other words, we were trying to figure out how to better communicate, here was research money spent, and as a result of this sound investment, the Internet came to be.

The Internet has changed us. It's changed the whole world."

Software Techniques - Example

• This code, add a scalar to a vector:
  for (i=1000; i>0; i=i–1)
  \[ x[i] = x[i] + s; \]
• Assume following latencies for all examples
  – Ignore delayed branch in these examples

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in cycles</th>
<th>stalls between in cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>Another FP ALU op</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>Store double</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Integer op</td>
<td>Integer op</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

FP Loop: Where are the Hazards?

• First translate into MIPS code:
  - To simplify, assume 8 is lowest address

Loop:
  L.D F0,0(R1);F0=vector element
  ADD.D F4,F0,F2;add scalar from F2
  S.D 0(R1),F4;store result
  DADDUI R1,R1,-8;decrement pointer 8B (DW)
  BNEZ R1,Loop ;branch R1!=zero
FP Loop Showing Stalls

1 Loop: L.D F0,0(R1) ;F0=vector element
2 stall
3 ADD.D F4,F0,F2 ;add scalar in F2
4 stall
5 stall
6 S.D 0(R1),F4 ;store result
7 DADDUI R1,R1,-8 ;decrement pointer 8B (DW)
8 stall ;assumes can’t forward to branch
9 BNEZ R1,Loop ;branch R1!=zero

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

9 clock cycles: Rewrite code to minimize stalls?

Revised FP Loop Minimizing Stalls

1 Loop: L.D F0,0(R1)
2 ADDUI R1,R1,-8
3 ADD.D F4,F0,F2
4 stall
5 stall
6 S.D 0(R1),F4 ;altered offset when move DSUBUI
7 BNEZ R1,Loop

Swap DADDUI and S.D by changing address of S.D

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

7 clock cycles, but just 3 for execution (L.D, ADD.D, S.D), 4 for loop overhead; How make faster?

Unroll Loop Four Times (straightforward way)

1 Loop: L.D F0,0(R1)
3 ADD.D F4,F0,F2
6 S.D 0(R1),F4 ;drop DSUBUI & BNEZ
7 L.D F6,-8(R1)
9 ADD.D F8,F6,F2
12 S.D -8(R1),F8 ;drop DSUBUI & BNEZ
13 L.D F10,-16(R1)
15 ADD.D F12,F10,F2
18 S.D -16(R1),F12 ;drop DSUBUI & BNEZ
19 L.D F14,-24(R1)
21 ADD.D F16,F14,F2
24 S.D -24(R1),F16
25 DADDUI R1,R1,#-32 ;alter to 4*8
26 BNEZ R1,Loop

Unrolled Loop Detail

- Do not usually know upper bound of loop
- Suppose it is n, and we would like to unroll the loop to make k copies of the body
- Instead of a single unrolled loop, we generate a pair of consecutive loops:
  - 1st executes \((n \mod k)\) times and has a body that is the original loop
  - 2nd is the unrolled body surrounded by an outer loop that iterates \((n/k)\) times
- For large values of n, most of the execution time will be spent in the unrolled loop

27 clock cycles, or 6.75 per iteration
( Assumes R1 is multiple of 4)
Unrolled Loop That Minimizes Stalls

1. Loop: L.D F0,0(R1)
2. L.D F6,-8(R1)
3. L.D F10,-16(R1)
4. L.D F14,-24(R1)
5. ADD.D F4,F0,F2
6. ADD.D F8,F6,F2
7. ADD.D F12,F10,F2
8. ADD.D F16,F14,F2
9. S.D 0(R1),F4
10. S.D -8(R1),F8
11. S.D -16(R1),F12
12. DSUBUI R1,R1,#32
13. S.D 8(R1),F16 ; 8-32 = -24
14. BNEZ R1,LOOP

14 clock cycles, or 3.5 per iteration

5 Loop Unrolling Decisions

- Requires understanding how one instruction depends on another and how the instructions can be changed or reordered given the dependences:
  1. Determine loop unrolling useful by finding that loop iterations were independent (except for maintenance code)
  2. Use different registers to avoid unnecessary constraints forced by using same registers for different computations
  3. Eliminate the extra test and branch instructions and adjust the loop termination and iteration code
  4. Determine that loads and stores in unrolled loop can be interchanged by observing that loads and stores from different iterations are independent
     • Transformation requires analyzing memory addresses and finding that they do not refer to the same address
  5. Schedule the code, preserving any dependences needed to yield the same result as the original code

3 Limits to Loop Unrolling

1. Decrease in amount of overhead amortized with each extra unrolling
   • Amdahl’s Law
2. Growth in code size
   • For larger loops, concern it increases the instruction cache miss rate
3. Register pressure: potential shortfall in registers created by aggressive unrolling and scheduling
   • If not be possible to allocate all live values to registers, may lose some or all of its advantage
   • Loop unrolling reduces impact of branches on pipeline; another way is branch prediction

Static Branch Prediction

- Lecture 3 showed scheduling code around delayed branch
- To reorder code around branches, need to predict branch statically when compile
- Simplest scheme is to predict a branch as taken
  • Average misprediction = untaken branch frequency = 34% SPEC
- More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:

- More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:
Dynamic Branch Prediction

• Why does prediction work?
  – Underlying algorithm has regularities
  – Data that is being operated on has regularities
  – Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems

• Is dynamic branch prediction better than static branch prediction?
  – Seems to be
  – There are a small number of important branches in programs which have dynamic behavior

Dynamic Branch Prediction

• Performance = \( f(\text{accuracy}, \text{cost of misprediction}) \)

• Branch History Table: Lower bits of PC address index table of 1-bit values
  – Says whether or not branch taken last time
  – No address check

• Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iterations before exit):
  – End of loop case, when it exits instead of looping as before
  – First time through loop on next time through code, when it predicts exit instead of looping

Solution: 2-bit scheme where change prediction only if get misprediction twice

• Red: stop, not taken
• Green: go, taken
• Adds \textit{hysteresis} to decision making process

BHT Accuracy

• Mispredict because either:
  – Wrong guess for that branch
  – Got branch history of wrong branch when index the table

• 4096 entry table:
Correlated Branch Prediction

- Idea: record $m$ most recently executed branches as taken or not taken, and use that pattern to select the proper $n$-bit branch history table.

- In general, $(m,n)$ predictor means record last $m$ branches to select between $2^m$ history tables, each with $n$-bit counters.
  - Thus, old 2-bit BHT is a $(0,2)$ predictor.

- Global Branch History: $m$-bit shift register keeping T/NT status of last $m$ branches.

- Each entry in table has $m n$-bit predictors.

Correlating Branches

(2,2) predictor

- Behavior of recent branches selects between four predictions of next branch, updating just that prediction.

Global Branch History:

- $m$-bit shift register keeping T/NT status of last $m$ branches.

- Each entry in table has $m n$-bit predictors.

Tournament Predictors

- Multilevel branch predictor
- Use $n$-bit saturating counter to choose between predictors
- Usual choice between global and local predictors.
Tournament Predictors

Tournament predictor using, say, 4K 2-bit counters indexed by local branch address. Chooses between:

• Global predictor
  – 4K entries index by history of last 12 branches \(2^{12} = 4\text{K}\)
  – Each entry is a standard 2-bit predictor

• Local predictor
  – Local history table: 1024 10-bit entries recording last 10 branches, index by branch address
  – The pattern of the last 10 occurrences of that particular branch used to index table of 1K entries with 3-bit saturating counters

Comparing Predictors (Fig. 2.8)

• Advantage of tournament predictor is ability to select the right predictor for a particular branch
  – Particularly crucial for integer benchmarks.
  – A typical tournament predictor will select the global predictor almost 40% of the time for the SPEC integer benchmarks and less than 15% of the time for the SPEC FP benchmarks

Pentium 4 Misprediction Rate
(per 1000 instructions, not per branch)

≈ 6% misprediction rate per branch SPECint (19% of INT instructions are branch)
≈ 2% misprediction rate per branch SPECfp (5% of FP instructions are branch)

Branch Target Buffers (BTB)

• Branch target calculation is costly and stalls the instruction fetch.
• BTB stores PCs the same way as caches
• The PC of a branch is sent to the BTB
• When a match is found the corresponding Predicted PC is returned
• If the branch was predicted taken, instruction fetch continues at the returned predicted PC
Dynamic Branch Prediction Summary

- Prediction becoming important part of execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
  - Either different branches (GA)
  - Or different executions of same branches (PA)
- Tournament predictors take insight to next level, by using multiple predictors
  - usually one based on global information and one based on local information, and combining them with a selector
  - In 2006, tournament predictors using \( \approx 30K \) bits are in processors like the Power5 and Pentium 4
- Branch Target Buffer: include branch address & prediction

Outline

- ILP
- Compiler techniques to increase ILP
- Loop Unrolling
- Static Branch Prediction
- Dynamic Branch Prediction
- Overcoming Data Hazards with Dynamic Scheduling
- (Start) Tomasulo Algorithm
- Conclusion

Advantages of Dynamic Scheduling

- **Dynamic scheduling** - hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behavior
- It handles cases when dependences unknown at compile time
  - it allows the processor to tolerate unpredictable delays such as cache misses, by executing other code while waiting for the miss to resolve
- It allows code that compiled for one pipeline to run efficiently on a different pipeline
- It simplifies the compiler
- **Hardware speculation**, a technique with significant performance advantages, builds on dynamic scheduling (next lecture)
**HW Schemes: Instruction Parallelism**

- Key idea: Allow instructions behind stall to proceed
  
  - `DIVD F0,F2,F4`
  - `ADDD F10,F0,F8`
  - `SUBD F12,F8,F14`

- Enables **out-of-order execution** and allows **out-of-order completion** (e.g., `SUBD`)
  
  - In a dynamically scheduled pipeline, all instructions still pass through issue stage in order (in-order issue)

- Will distinguish when an instruction **begins execution** and when it **completes execution**; between 2 times, the instruction is **in execution**

- Note: Dynamic execution creates WAR and WAW hazards and makes exceptions harder

---

**Dynamic Scheduling Step 1**

- Simple pipeline had 1 stage to check both structural and data hazards: Instruction Decode (ID), also called Instruction Issue
  
  - `Issue`—Decode instructions, check for structural hazards

- Split the ID pipe stage of simple 5-stage pipeline into 2 stages:
  
  - `Read operands`—Wait until no data hazards, then read operands

---

**A Dynamic Algorithm: Tomasulo’s**

- For IBM 360/91 (before caches!)
  
  - ⇒ Long memory latency

- Goal: High Performance without special compilers

- Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
  
  - This led Tomasulo to try to figure out how to get more effective registers — renaming in hardware!

- Why Study 1966 Computer?

- The descendants of this have flourished!
  
  - Alpha 21264, Pentium 4, AMD Opteron, Power 5, ...

---

**Tomasulo Algorithm**

- Control & buffers distributed with Function Units (FU)
  
  - FU buffers called “reservation stations”; have pending operands

- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming
  
  - Renaming avoids WAR, WAW hazards
  
  - More reservation stations than registers, so can do optimizations compilers can’t

- Results to FU from RS, not through registers, over **Common Data Bus** that broadcasts results to all FUs
  
  - Avoids RAW hazards by executing an instruction only when its operands are available

- Load and Stores treated as FUs with RSs as well

- Integer instructions can go past branches (predict taken), allowing FP ops beyond basic block in FP queue
**Tomasulo Organization**

From Mem → FP Op Queue → Load Buffers → FP Registers → Reservation Stations → Common Data Bus (CDB) → Store Buffers → To Mem

- **FP adders**
- **FP multipliers**

**Reservation Station Components**

- **Op**: Operation to perform in the unit (e.g., + or –)
- **Vj, Vk**: Value of Source operands
  - Store buffers has V field, result to be stored
- **Qj, Qk**: Reservation stations producing source registers (value to be written)
  - Note: Qj, Qk = 0 => ready
  - Store buffers only have Qi for RS producing result
- **Busy**: Indicates reservation station or 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.

**Three Stages of Tomasulo Algorithm**

1. **Issue**—get instruction from FP Op Queue
   - If reservation station free (no structural hazard), control issues instr & sends operands (renames registers).
2. **Execute**—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch Common Data Bus for result
3. **Write result**—finish execution (WB)
   - Write on Common Data Bus to all awaiting units; mark reservation station available
   - **Normal data bus**: data + destination (“go to” bus)
   - **Common data bus**: data + source (“come from” bus)
     - 64 bits of data + 4 bits of Functional Unit source address
     - Write if matches expected Functional Unit (produces result)
     - Does the broadcast
   - **Example speed**: 3 clocks for Fl.pt. +, -; 10 for *; 40 clks for /

**Instruction Example**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>S1</th>
<th>S2</th>
<th>RS</th>
<th>RS</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Load1</td>
<td>No</td>
</tr>
<tr>
<td>LD F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Load2</td>
<td>No</td>
</tr>
<tr>
<td>MULTD F0 F2</td>
<td>34</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Load3</td>
<td>No</td>
</tr>
<tr>
<td>ADDD F6 F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status**

- **Clock cycle counter**
- **FU count down**
- **3 FP Adder R.S.**
- **2 FP Mult R.S.**
Tomasulo Example Cycle 1

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>jk</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>Load1: Yes 34+R2</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>Load2: No</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Load3: No</td>
</tr>
<tr>
<td>SUBID</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations:

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register result status:

Clock 4

FU: Load1

Tomasulo Example Cycle 2

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>jk</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>Load1: Yes 34+R2</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>Load2: Yes 45+R3</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Load3: No</td>
</tr>
<tr>
<td>SUBID</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations:

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register result status:

Clock 4

FU: Load2

Note: Can have multiple loads outstanding

Tomasulo Example Cycle 3

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>jk</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>Load1: Yes 34+R2</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>Load2: Yes 45+R3</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Load3: No</td>
</tr>
<tr>
<td>SUBID</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations:

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>Yes</td>
<td>MULTD</td>
<td>R(F4)</td>
<td>Load2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register result status:

Clock 4

FU: Multi1

Multi1: Load2

Load2 completing; what is waiting for Load2?

Tomasulo Example Cycle 4

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>jk</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>Load1: No</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>Load2: Yes 45+R3</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Load3: No</td>
</tr>
<tr>
<td>SUBID</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations:

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>Yes</td>
<td>SUBID</td>
<td>M(A1)</td>
<td>Load2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register result status:

Clock 4

FU: Multi1

Multi1: Load2

M(A1): Add1

Load2 completing; what is waiting for Load2?

• Note: registers names are removed (“renamed”) in Reservation Stations; MULT issued

• Load1 completing; what is waiting for Load1?
### Tomasulo Example Cycle 5

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1 3 4</td>
<td>Load1</td>
<td>No</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>2 4 5</td>
<td>Load2</td>
<td>No</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>3</td>
<td>Load3</td>
<td>No</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

- S1: Add1
- S2: No
- RS: No
- RS: No

**Register result status:**

- Clock 0: Add1
- Clock 1: No
- Clock 2: Yes
- Clock 3: No
- Clock 4: No
- Clock 5: No

**Busy Address:**

- Add1: M(A1) M(A2)
- ADDD: M(A2)

• Timer starts down for Add1, Mult1

2/8/2006  
CS252 S06 Lec7 ILP  
57

### Tomasulo Example Cycle 6

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1 3 4</td>
<td>Load1</td>
<td>No</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>2 4 5</td>
<td>Load2</td>
<td>No</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>3</td>
<td>Load3</td>
<td>No</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

- S1: Add1
- S2: No
- RS: No
- RS: No

**Register result status:**

- Clock 0: Add1
- Clock 1: Add2
- Clock 2: No
- Clock 3: No
- Clock 4: No
- Clock 5: Mult1 Mult2

### Tomasulo Example Cycle 7

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1 3 4</td>
<td>Load1</td>
<td>No</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>2 4 5</td>
<td>Load2</td>
<td>No</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>3</td>
<td>Load3</td>
<td>No</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

- S1: Add1
- S2: Yes
- RS: No
- RS: No

**Register result status:**

- Clock 0: Add1
- Clock 1: Add2
- Clock 2: No
- Clock 3: No
- Clock 4: Mult1 Mult2

• Add1 (SUBD) completing; what is waiting for it?

2/8/2006  
CS252 S06 Lec7 ILP  
59

### Tomasulo Example Cycle 8

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1 3 4</td>
<td>Load1</td>
<td>No</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>2 4 5</td>
<td>Load2</td>
<td>No</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>3</td>
<td>Load3</td>
<td>No</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

- S1: Add1
- S2: No
- RS: No
- RS: No

**Register result status:**

- Clock 0: Add1
- Clock 1: Add2
- Clock 2: No
- Clock 3: No
- Clock 4: Mult1 Mult2

• Issue ADDD here despite name dependency on F6?
### Tomasulo Example Cycle 9

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+ R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
<td></td>
</tr>
<tr>
<td>LD F2 45+ R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
<td></td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>Load3 No</td>
<td></td>
</tr>
<tr>
<td>SUBID F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6 F8 F2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>Yes</td>
<td>ADDD (M-M) M(A2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Multi1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

- Clock: 9

| FU | Multi1 M(A2) | Add2 (M-M) Multi2 |

- Add2 (ADDD) completing; what is waiting for it?

---

### Tomasulo Example Cycle 10

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+ R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
<td></td>
</tr>
<tr>
<td>LD F2 45+ R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
<td></td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>Load3 No</td>
<td></td>
</tr>
<tr>
<td>SUBID F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6 F8 F2</td>
<td>6</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 Add2</td>
<td>Yes</td>
<td>ADDD (M-M) M(A2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Multi1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

- Clock: 10

| FU | Multi1 M(A2) | Add2 (M-M) Multi2 |

- Add2 (ADDD) completing; what is waiting for it?

- All quick instructions complete in this cycle!

---

### Tomasulo Example Cycle 11

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+ R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
<td></td>
</tr>
<tr>
<td>LD F2 45+ R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
<td></td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>Load3 No</td>
<td></td>
</tr>
<tr>
<td>SUBID F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6 F8 F2</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Multi1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

- Clock: 11

| FU | Multi1 M(A2) | Add2 (M-M) Multi2 |

- Write result of ADDD here?

---

### Tomasulo Example Cycle 12

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+ R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
<td></td>
</tr>
<tr>
<td>LD F2 45+ R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
<td></td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>Load3 No</td>
<td></td>
</tr>
<tr>
<td>SUBID F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6 F8 F2</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Multi1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

- Clock: 12

| FU | Multi1 M(A2) | Add2 (M-M) Multi2 |

- Write result of ADDD here?

- All quick instructions complete in this cycle!
### Tomasulo Example Cycle 13

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+</td>
<td>R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
</tr>
<tr>
<td>LD F2 45+</td>
<td>R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>Load3 No</td>
</tr>
<tr>
<td>SUBD F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>F0 F2 F4 F6 F8 F10 F12 ... F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td></td>
<td>Mult1 M(A2) (M-M+N(M-M) Mult2</td>
</tr>
</tbody>
</table>

- Mult1 (MULTD) completing; what is waiting for it?

### Tomasulo Example Cycle 14

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+</td>
<td>R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
</tr>
<tr>
<td>LD F2 45+</td>
<td>R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>Load3 No</td>
</tr>
<tr>
<td>SUBD F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>F0 F2 F4 F6 F8 F10 F12 ... F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td></td>
<td>Mult1 M(A2) (M-M+N(M-M) Mult2</td>
</tr>
</tbody>
</table>

- Just waiting for Mult2 (DIVD) to complete

### Tomasulo Example Cycle 15

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
<th>Busy Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6 34+</td>
<td>R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1 No</td>
</tr>
<tr>
<td>LD F2 45+</td>
<td>R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2 No</td>
</tr>
<tr>
<td>MULTD F0 F2 F4</td>
<td>3</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>Load3 No</td>
</tr>
<tr>
<td>SUBD F8 F6 F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10 F0 F6</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 Mult1</td>
<td>Yes</td>
<td>MULTD M(A2) R(F4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD M(A1) Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>FU</th>
<th>F0 F2 F4 F6 F8 F10 F12 ... F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td>Mult1 M(A2) (M-M+N(M-M) Mult2</td>
</tr>
</tbody>
</table>

- Mult1 (MULTD) completing; what is waiting for it?
Faster than light computation (skip a couple of cycles)

Tomasulo Example Cycle 55

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+ R2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+ R3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2 F4</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6 F2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0 F6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8 F2</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

Reservation Stations: S1 S2 RS RS

Register result status: Clock 55 FU M*F4 M(A2) (M-M+M(M-M) Mult2)

• Mult2 (DIVD) is completing; what is waiting for it?

Tomasulo Example Cycle 56

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+ R2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+ R3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2 F4</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6 F2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0 F6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8 F2</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

Reservation Stations: S1 S2 RS RS

Register result status: Clock 56 FU M*F4 M(A2) (M-M+M(M-M) Mult2)

Tomasulo Example Cycle 57

Instruction status: Exec Write

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+ R2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+ R3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2 F4</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6 F2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0 F6</td>
<td>5</td>
<td>56</td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8 F2</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

Reservation Stations: S1 S2 RS RS

Register result status: Clock 57 FU M*F4 M(A2) (M-M+M(M-M) Result)

• Once again: In-order issue, out-of-order execution and out-of-order completion.
Why can Tomasulo overlap iterations of loops?

• Register renaming
  – Multiple iterations use different physical destinations for registers (dynamic loop unrolling).

• Reservation stations
  – Permit instruction issue to advance past integer control flow operations
  – Also buffer old values of registers - totally avoiding the WAR stall

• Other perspective: Tomasulo building data flow dependency graph on the fly

Tomasulo’s scheme offers 2 major advantages

1. Distribution of the hazard detection logic
   – distributed reservation stations and the CDB
   – If multiple instructions waiting on single result, & each instruction has other operand, then instructions can be released simultaneously by broadcast on CDB
   – If a centralized register file were used, the units would have to read their results from the registers when register buses are available

2. Elimination of stalls for WAW and WAR hazards

Tomasulo Drawbacks

• Complexity
  – delays of 360/91, MIPS 10000, Alpha 21264, IBM PPC 620 in CA:AQA 2/e, but not in silicon!

• Many associative stores (CDB) at high speed

• Performance limited by Common Data Bus
  – Each CDB must go to multiple functional units ⇒high capacitance, high wiring density
  – Number of functional units that can complete per cycle limited to one!
    » Multiple CDBs ⇒ more FU logic for parallel assoc stores

• Non-precise interrupts!
  – We will address this later

And In Conclusion … #1

• Leverage Implicit Parallelism for Performance: Instruction Level Parallelism
• Loop unrolling by compiler to increase ILP
• Branch prediction to increase ILP
• Dynamic HW exploiting ILP
  – Works when can’t know dependence at compile time
  – Can hide L1 cache misses
  – Code for one machine runs well on another
And In Conclusion … #2

• Reservations stations: *renaming* to larger set of registers + buffering source operands
  – Prevents registers as bottleneck
  – Avoids WAR, WAW hazards
  – Allows loop unrolling in HW

• Not limited to basic blocks
  (integer units gets ahead, beyond branches)

• Helps cache misses as well

• Lasting Contributions
  – Dynamic scheduling
  – Register renaming
  – Load/store disambiguation

• 360/91 descendants are Intel Pentium 4, IBM Power 5, AMD Athlon/Opteron, …