
CPE 631 Lecture 10: Instruction Level Parallelism and Its Dynamic Exploitation

Aleksandar Milenković, milenka@ece.uah.edu
Electrical and Computer Engineering
University of Alabama in Huntsville

CPE
631
©AM

Outline

- Tomasulo's algorithm

14/02/2005

UAH-CPE631

2

Techniques to exploit parallelism

Technique (Section in the textbook)	Reduces
Forwarding and bypassing (Section A.2)	Data hazard (DH) stalls
Delayed branches (A.2)	Control hazard stalls
Basic dynamic scheduling (A.8)	DH stalls (RAW)
Dynamic scheduling with register renaming (3.2)	WAR and WAW stalls
Dynamic branch prediction (3.4)	CH stalls
Issuing multiple instruction per cycle (3.6)	Ideal CPI
Speculation (3.7)	Data and control stalls
Dynamic memory disambiguation (3.2, 3.7)	RAW stalls w. memory
Loop Unrolling (4.1)	CH stalls
Basic compiler pipeline scheduling (A.2, 4.1)	DH stalls
Compiler dependence analysis (4.4)	Ideal CPI, DH stalls
Software pipelining and trace scheduling (4.3)	Ideal CPI and DH stalls
Compiler speculation (4.4)	Ideal CPI, and D/CH stalls

14/02/2005

UAH-CPE631

3

Dynamically Scheduled Pipelines

CPE 631 ©AM

Scoreboard Limitations

- Amount of parallelism among instructions
 - can we find independent instructions to execute
- Number of scoreboard entries
 - how far ahead the pipeline can look for independent instructions (we assume a window does not extend beyond a branch)
- Number and types of functional units
 - avoid structural hazards
- Presence of antidependences and output dependences
 - WAR and WAW stalls become more important

14/02/2005 UAH-CPE631 5

CPE 631 ©AM

Tomasulo's Algorithm

- Used in IBM 360/91 FPU (before caches)
- Goal: high FP performance without special compilers
- Conditions:
 - Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
 - Long memory accesses and long FP delays
 - 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, HP 8000, MIPS 10000, Pentium III, PowerPC 604, ...

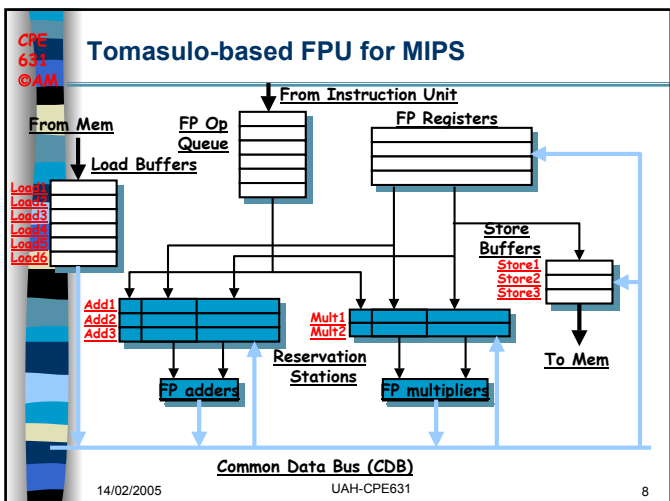
14/02/2005 UAH-CPE631 6

CPE 631 ©AM

Tomasulo's Algorithm (cont'd)

- Control & buffers **distributed** with Function Units (FU)
 - FU buffers called "**reservation stations**" => buffer the operands of instructions waiting to issue;
- Registers in instructions replaced by values or pointers to reservation stations (RS) => **register 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
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches, allowing FP ops beyond basic block in FP queue

14/02/2005 UAH-CPE631 7



CPE 631 ©AM

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 => source operand is already available in Vj/Vk
 - 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.

14/02/2005 UAH-CPE631 9

CPE 631 ©AM

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 it 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: 2 clocks for FI .pt. +,-; 10 for *; 40 clks for /

14/02/2005 UAH-CPE631 10

CPE 631 ©AM

Tomasulo Example

Instruction stream

Instruction	j	k	Issue	Comp	Result
LD	F6	34+	R2		
LD	F2	45+	R3		
MULTD	F0	F2	F4		
SLD	F8	F6	F2		
DIVD	F10	F0	F6		
ADDD	F6	F8	F2		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
0									

Annotations: 3 Load/Buffers, 3 FP Adder R.S., 2 FP Mult R.S., FU count down, Clock cycle counter

14/02/2005 UAH-CPE631 11

CPE 631 ©AM

Tomasulo Example Cycle 1

Instruction stream:

Instruction	j	k	Issue	Comp	Result
LD	F6	34+	R2		
LD	F2	45+	R3		
MULTD	F0	F2	F4		
SLD	F8	F6	F2		
DIVD	F10	F0	F6		
ADDD	F6	F8	F2		

Reservation Stations:

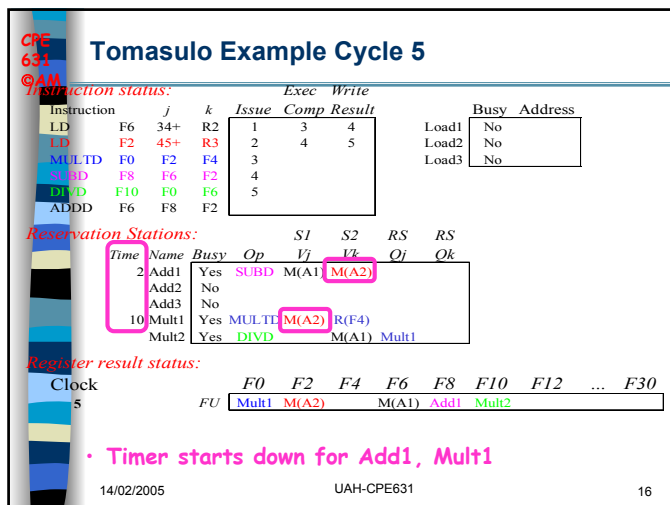
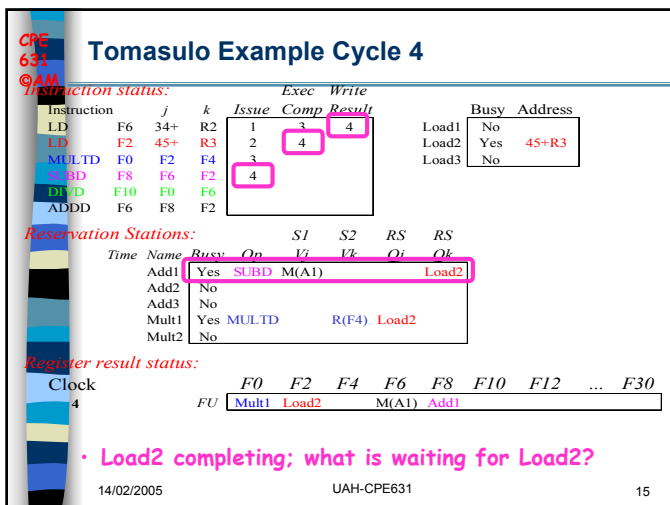
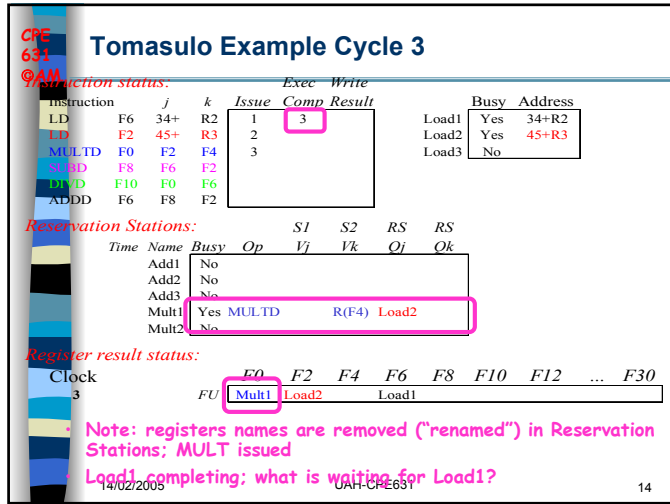
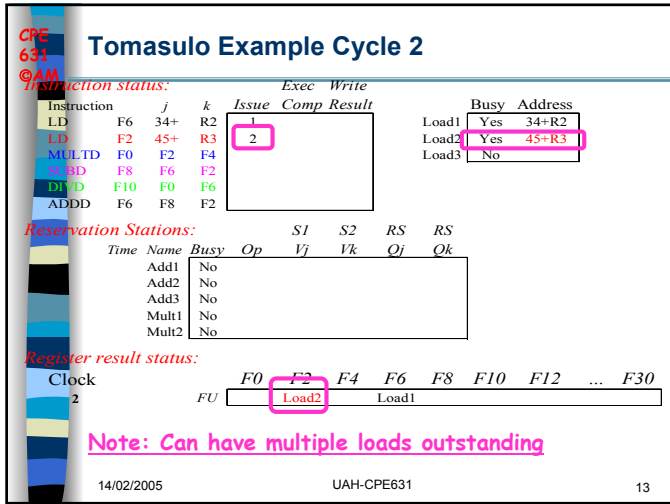
Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

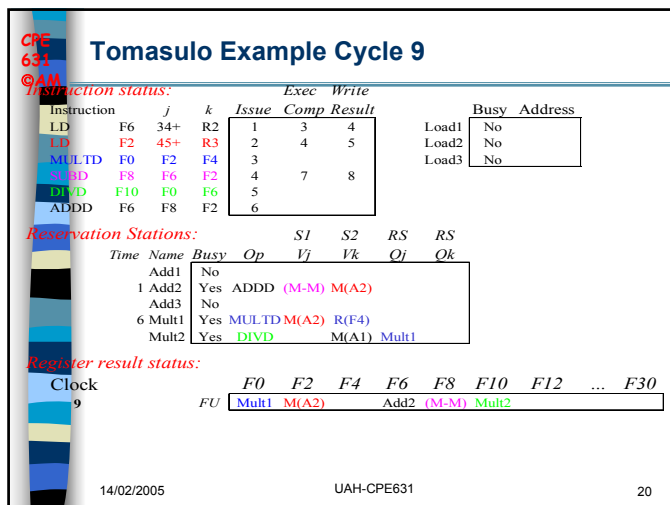
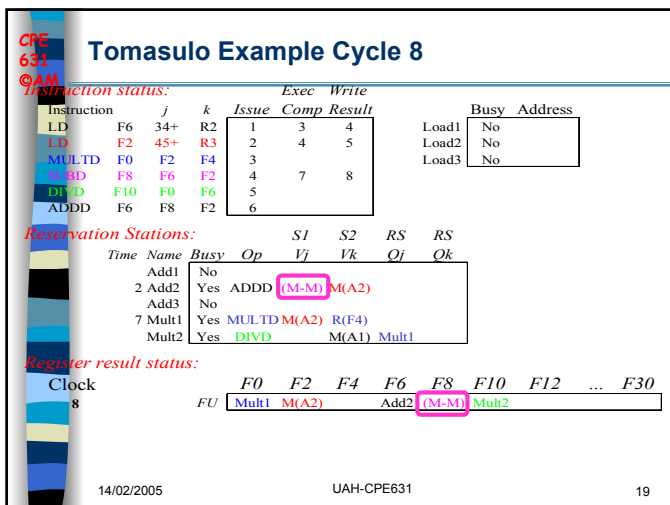
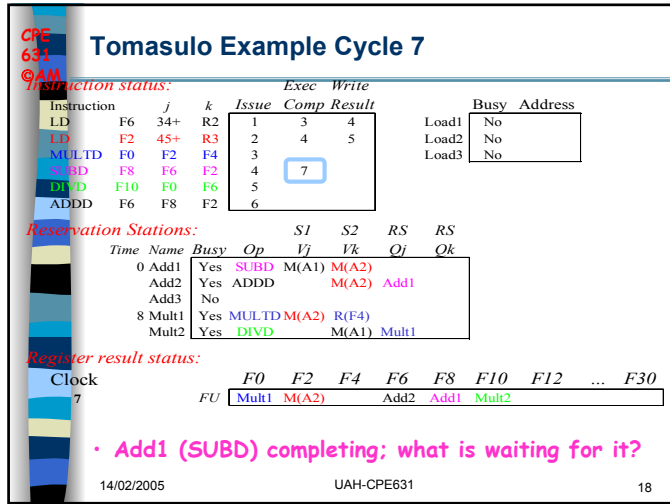
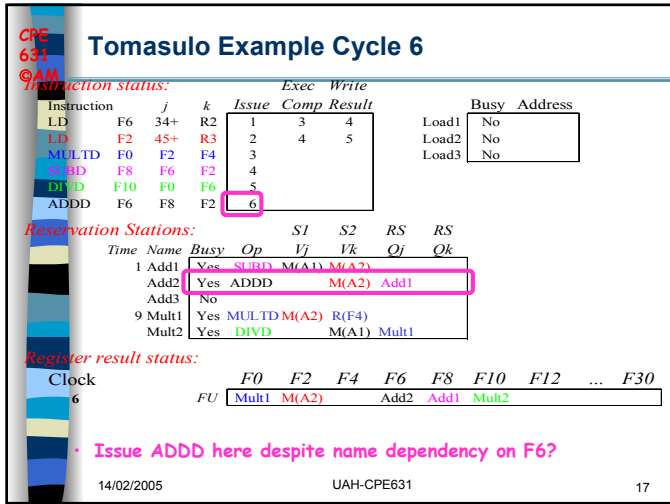
Register result status:

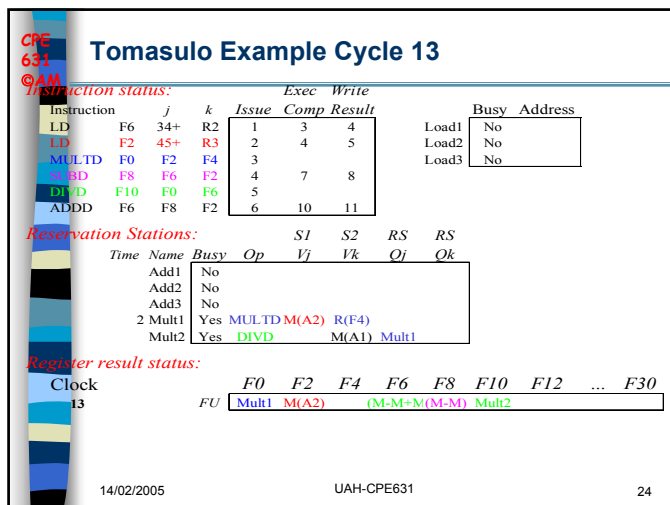
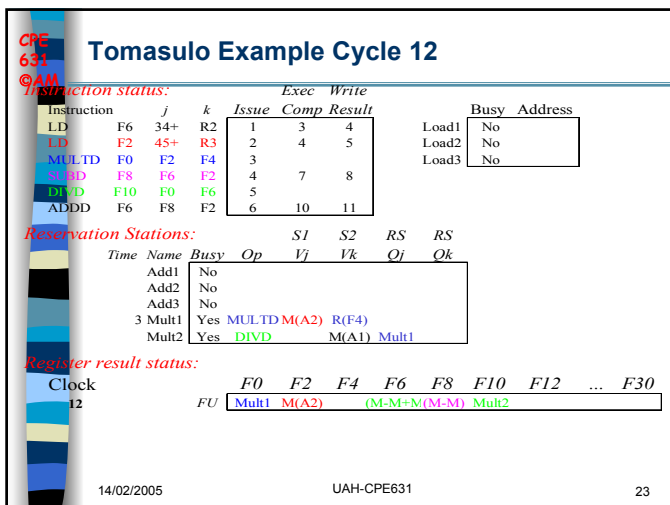
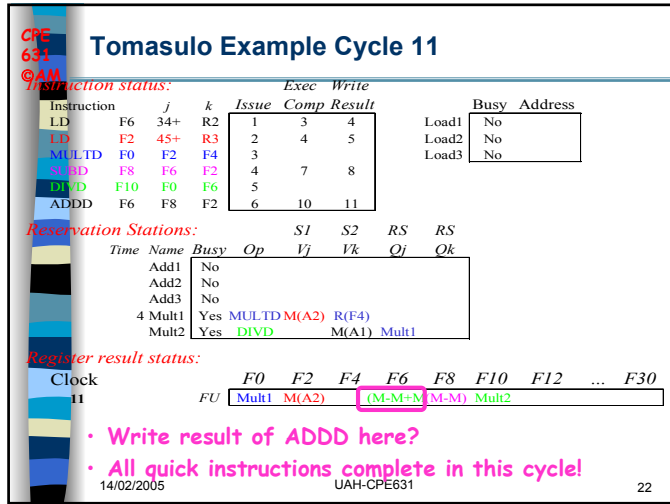
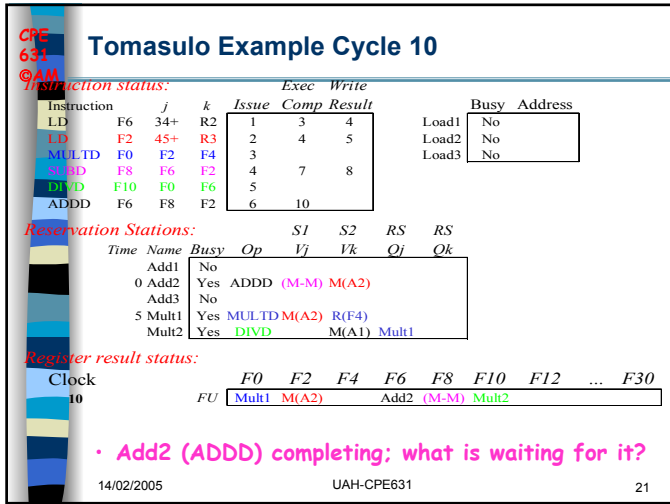
Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
1				Load1					

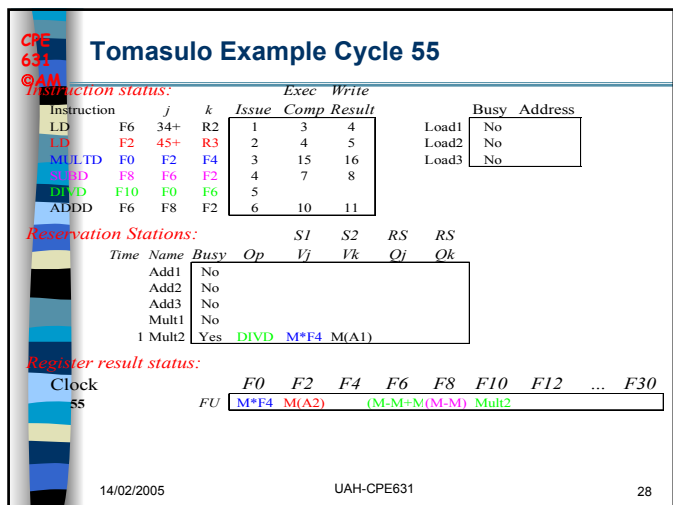
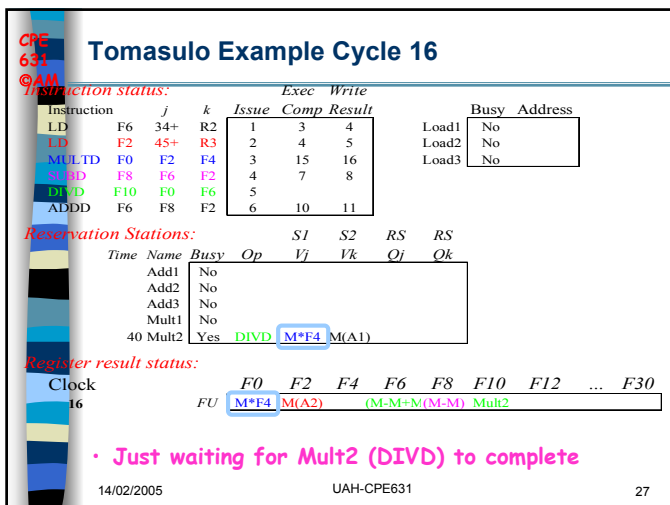
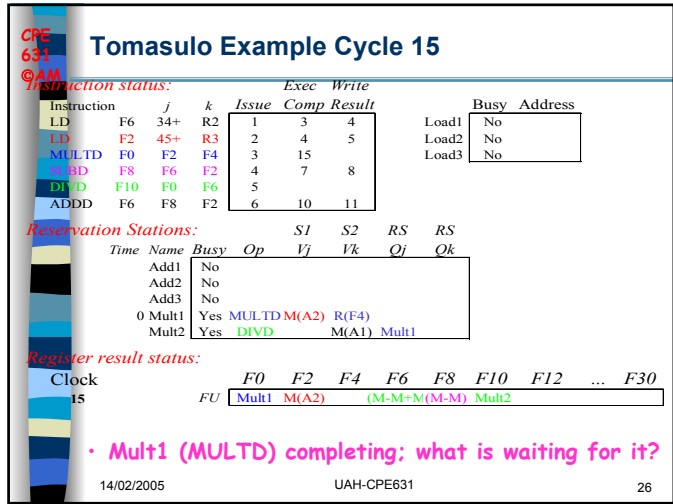
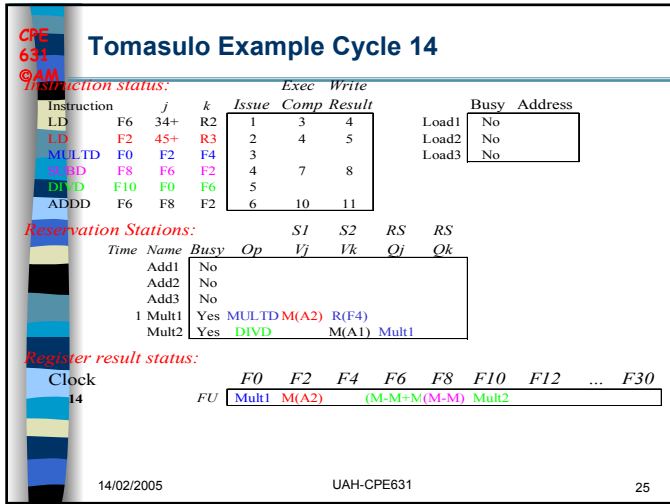
Annotations: 1, Yes 34+R2, Load1

14/02/2005 UAH-CPE631 12









CPE 631
9AM

Tomasulo Example Cycle 56

Instruction status:

Instruction	j	k	Exec		Write	Load1	Load2	Load3	Busy	Address
			Issue	Comp						
LD	F6	34+	R2	1	3	4			No	
LD	F2	45+	R3	2	4	5			No	
MULTD	F0	F2	F4	3	15	16			No	
SUBD	F8	F6	F2	4	7	8			No	
DIVD	F10	F0	F6	5	56				No	
ADDD	F6	F8	F2	6	10	11			No	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	M*F4	M(A2)		(M-M+M(M-M))	Mult2				

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

14/02/2005 UAH-CPE631 29

CPE 631
9AM

Tomasulo Example Cycle 57

Instruction status:

Instruction	j	k	Exec		Write	Load1	Load2	Load3	Busy	Address
			Issue	Comp						
LD	F6	34+	R2	1	3	4			No	
LD	F2	45+	R3	2	4	5			No	
MULTD	F0	F2	F4	3	15	16			No	
SUBD	F8	F6	F2	4	7	8			No	
DIVD	F10	F0	F6	5	56	57			No	
ADDD	F6	F8	F2	6	10	11			No	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	M*F4	M(A2)		(M-M+M(M-M))	Result				

• Once again: In-order issue, out-of-order execution and out-of-order completion.

14/02/2005 UAH-CPE631 30

CPE 631
9AM

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

14/02/2005 UAH-CPE631 31

CPE 631
9AM

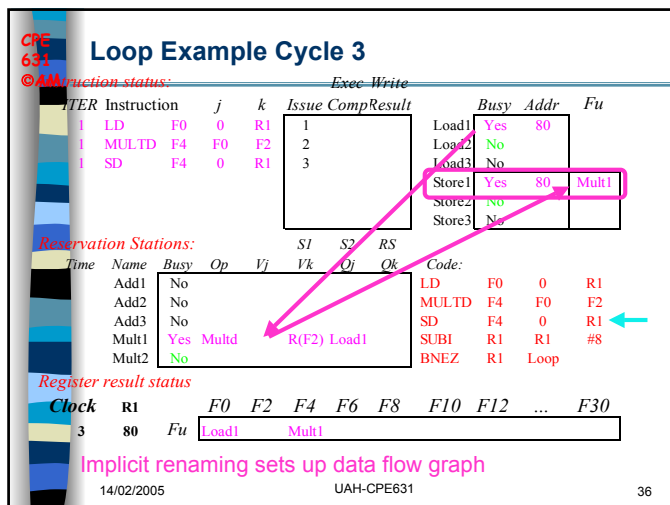
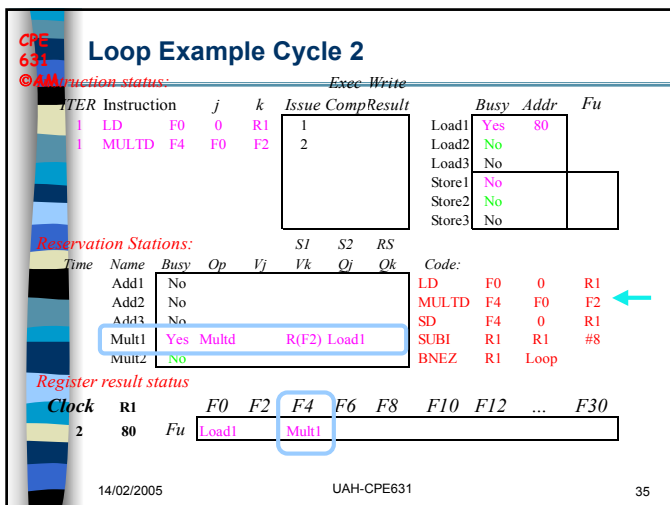
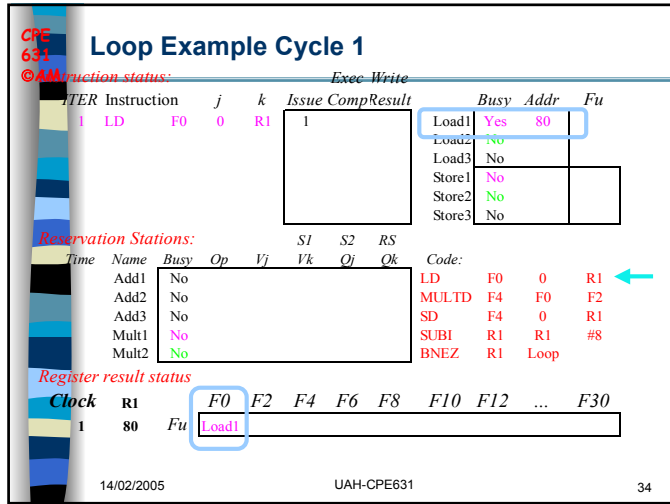
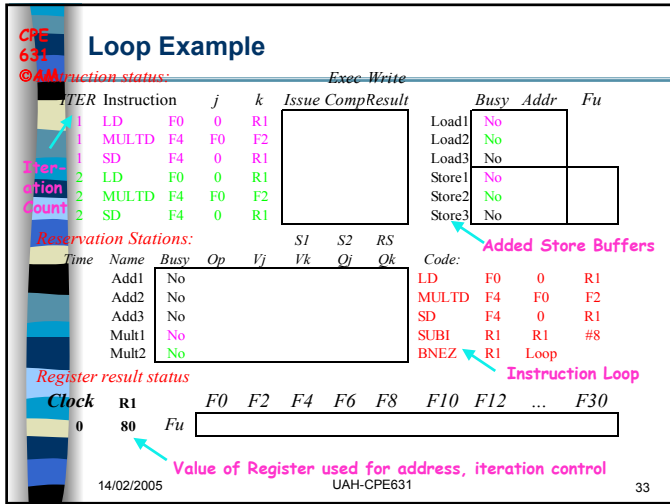
Tomasulo Loop Example

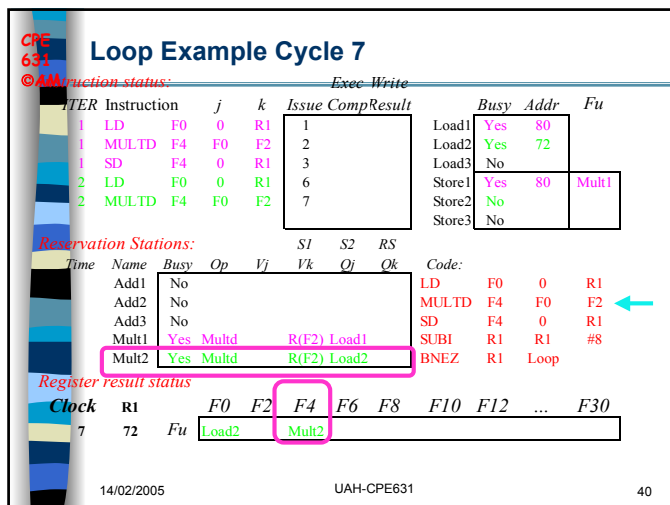
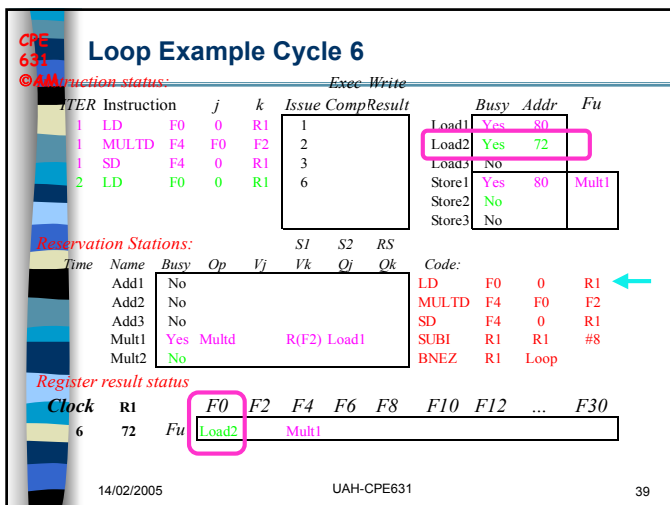
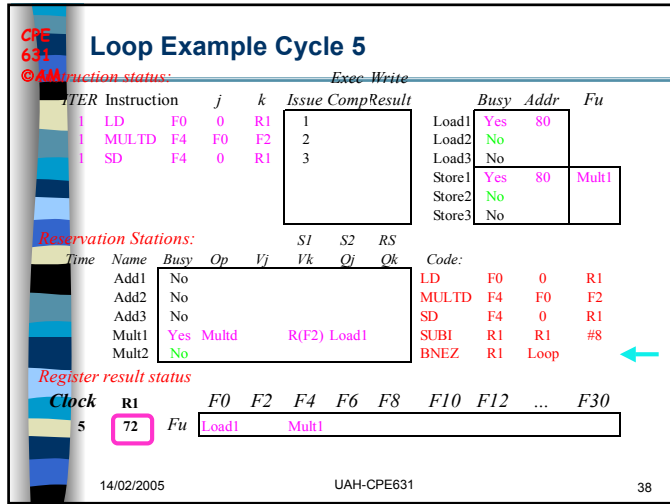
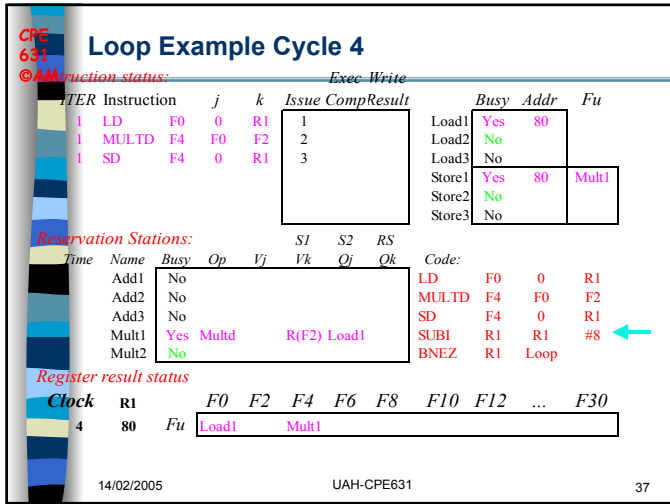
```

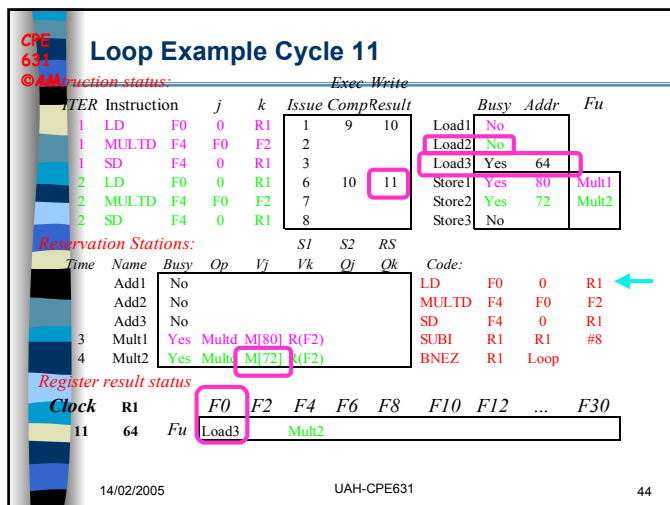
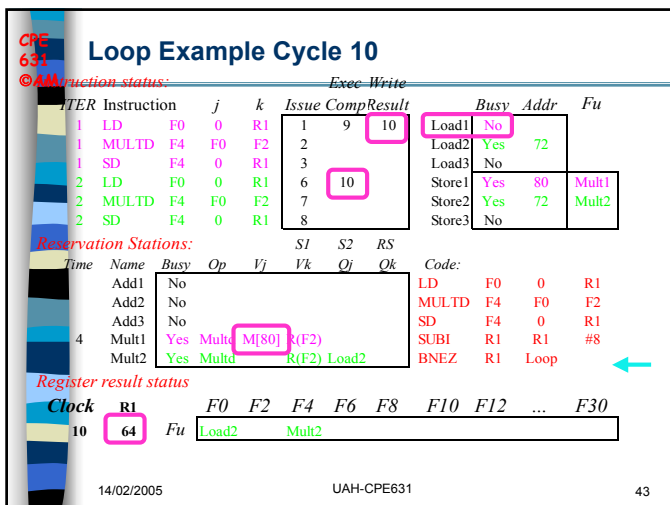
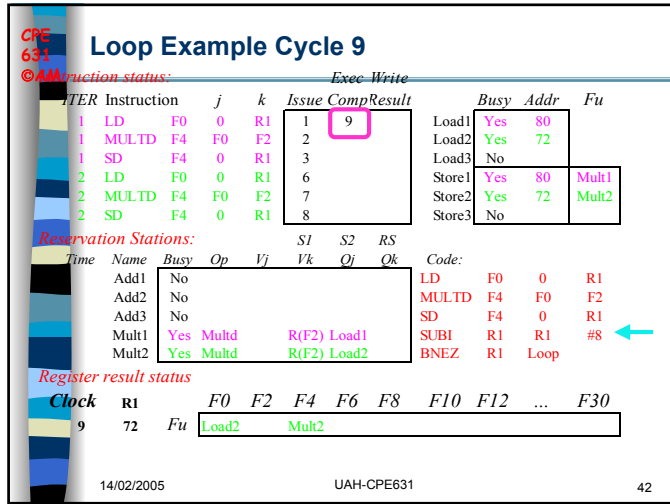
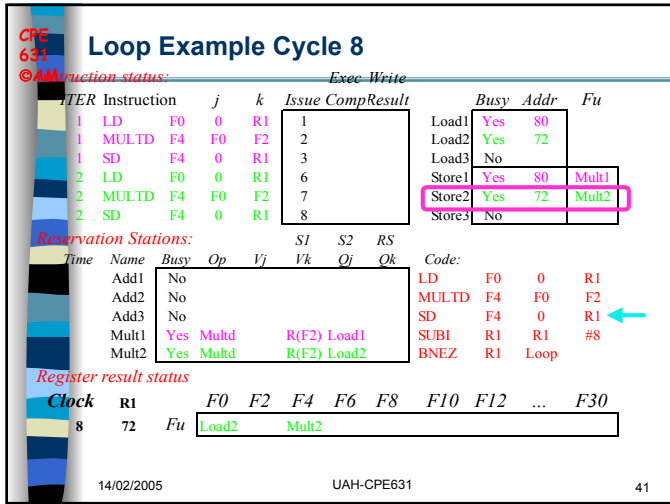
Loop: LD    F0    0(R1)
      MULTD F4    F0    F2
      SD    F4    0    R1
      SUBI  R1    R1    #8
      BNEZ R1    Loop
  
```

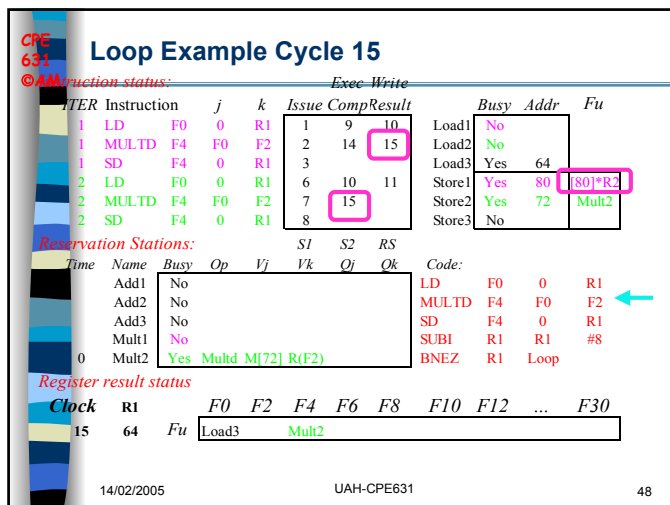
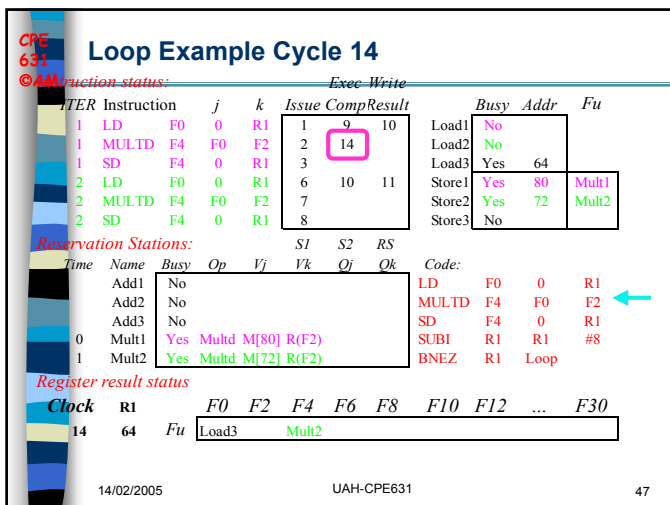
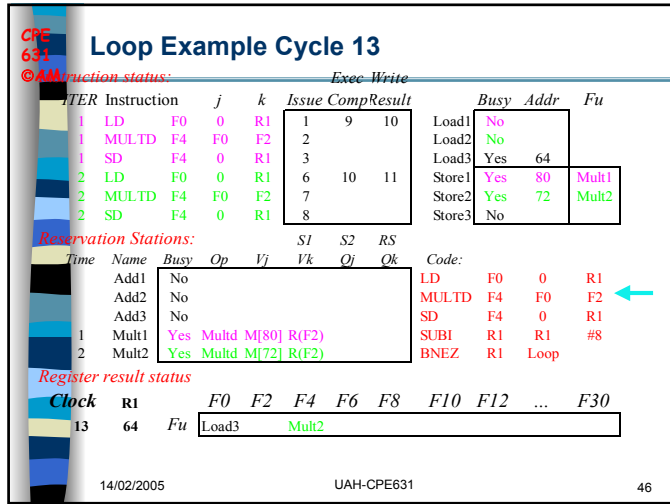
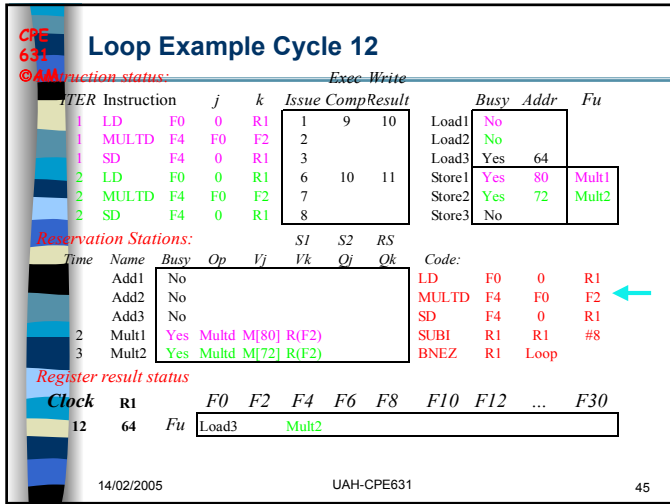
- This time assume Multiply takes 4 clocks
- Assume 1st load takes 8 clocks (L1 cache miss), 2nd load takes 1 clock (hit)
- To be clear, will show clocks for SUBI, BNEZ
 - Reality: integer instructions ahead of Fl. Pt. Instructions
- Show 2 iterations

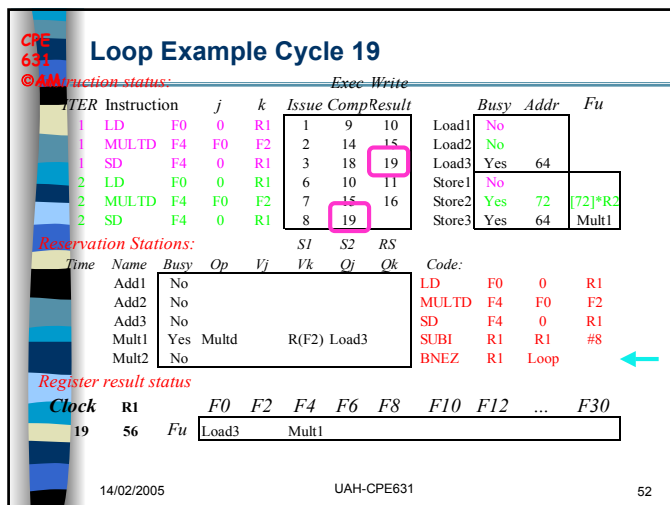
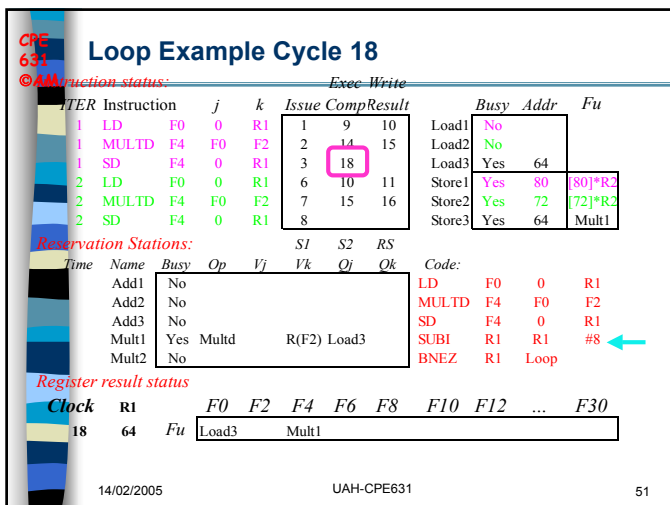
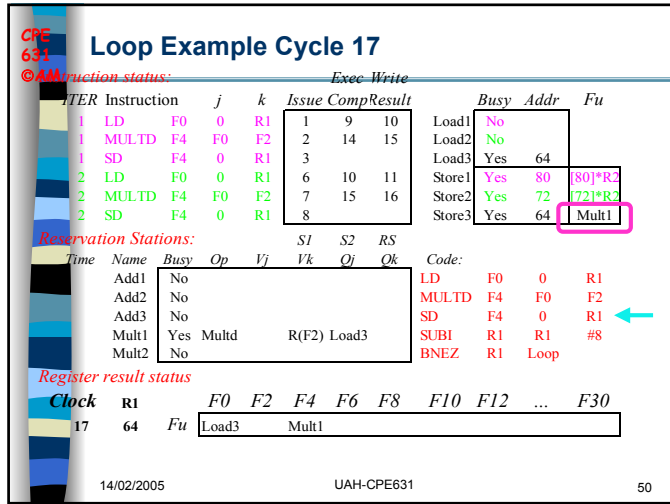
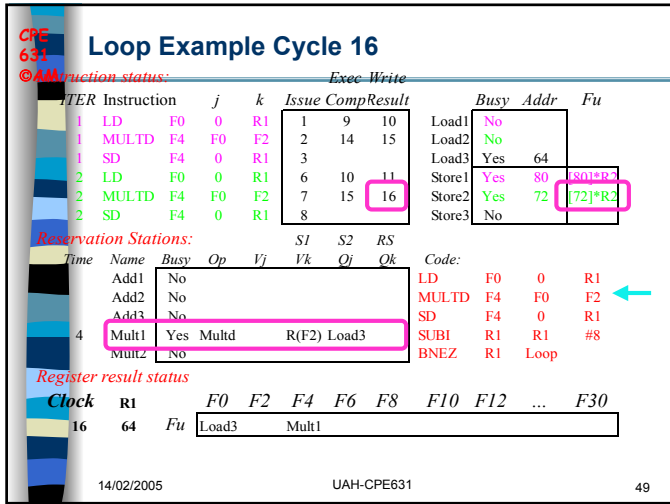
14/02/2005 UAH-CPE631 32











CPE 631 ©AM

Loop Example Cycle 20

Instruction status: Exec Write

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD	F0	0	R1	1	9	10	Load1	Yes 56
1	MULTD	F4	F0	F2	2	14	15	Load2	No
1	SD	F4	0	R1	3	18	19	Load3	Yes 64
2	LD	F0	0	R1	6	10	11	Store1	No
2	MULTD	F4	F0	F2	7	15	16	Store2	No
2	SD	F4	0	R1	8	19	20	Store3	Yes 64 Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Oj	Ok	Code:
Add1	No							LD F0 0 R1
Add2	No							MULTD F4 F0 F2
Add3	No							SD F4 0 R1
Mult1	Yes	Multd		R(F2)	Load3			SUBI R1 R1 #8
Mult2	No							BNEZ R1 Loop

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Fu	Load1	Mult1						

Once again: In-order issue, out-of-order execution and out-of-order completion

14/02/2005 UAH-CPE631 53

CPE 631 ©AM

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 that we saw in the scoreboard
- Other perspective: Tomasulo building data flow dependency graph on the fly

14/02/2005 UAH-CPE631 54

CPE 631 ©AM

Tomasulo's scheme offers 2 major advantages

- (1) the 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) the elimination of stalls for WAW and WAR hazards

14/02/2005 UAH-CPE631 55

CPE 631 ©AM

What about Precise Interrupts?

- Tomasulo had:
 - In-order issue, out-of-order execution, and out-of-order completion
- Need to "fix" the out-of-order completion aspect so that we can find precise breakpoint in instruction stream

14/02/2005 UAH-CPE631 56

CPE 631 ©AM

Relationship between precise interrupts and speculation

- Speculation is a form of guessing
- Important for branch prediction:
 - Need to "take our best shot" at predicting branch direction
- If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly:
 - This is exactly same as precise exceptions!
- Technique for both precise interrupts/exceptions and speculation: in-order completion or commit

14/02/2005 UAH-CPE631 57

CPE 631 ©AM

HW support for precise interrupts

- Need HW buffer for results of uncommitted instructions: reorder buffer
 - 3 fields: instr, destination, value
 - Use reorder buffer number instead of reservation station when execution completes
 - Supplies operands between execution complete & commit
 - (Reorder buffer can be operand => more registers like RS)
 - Instructions commit
 - Once instruction commits, result is put into register
 - As a result, easy to undo speculated instructions on mispredicted branches or exceptions

14/02/2005 UAH-CPE631 58

CPE 631 ©AM

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 (this stage sometimes called "dispatch")
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; checks RAW (sometimes called "issue")
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. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

14/02/2005 UAH-CPE631 59

CPE 631 ©AM

What are the hardware complexities with reorder buffer (ROB)?

- How do you find the latest version of a register?
 - (As specified by Smith paper) need associative comparison network
 - Could use future file or just use the register result status buffer to track which specific reorder buffer has received the value
- Need as many ports on ROB as register file

14/02/2005 UAH-CPE631 60

Summary

- Reservations stations: implicit register renaming to larger set of registers + buffering source operands
 - Prevents registers as bottleneck
 - Avoids WAR, WAW hazards of Scoreboard
 - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Today, helps cache misses as well
 - Don't stall for L1 Data cache miss (insufficient ILP for L2 miss?)
- Lasting Contributions
 - Dynamic scheduling
 - Register renaming
 - Load/store disambiguation
- 360/91 descendants are Pentium III; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264