COMP4611 Tutorial 6
Instruction Level Parallelism

Instruction Level Parallelism

- Definition
  - An implementation technique whereby multiple instructions are overlapped in execution
- Two separate approaches to exploiting ILP
  - Hardware support to help discover and exploit the parallelism dynamically
  - Software technology to find parallelism statically

Instruction Level Parallelism

- Few possibilities in a basic block
  - A straight-line code sequence
    - no branches in except to the entry
    - no branches out except at the exit
  - Blocks are small (6-7 instructions)
  - Instructions are likely to depend upon one another
- Goal: Exploit ILP across multiple basic blocks
  - Example: *loop-level parallelism*

```
for (i = 1000; i > 0; i=i-1)
    x[i] = x[i] + s;
```

Basic Scheduling

```
for (i = 1000; i > 0; i=i-1)
    x[i] = x[i] + s;
```

Sequential MIPS Assembly Code

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

Pipelined execution:

```
Loop:   LD      F0, 0(R1)     1
        stall                 2
        ADDD    F4, F0, F2    3
        stall                 4
        stall                 5
        SD      0(R1), F4     6
        SUBI    R1, R1, #8    7
        stall                 8
        BNEZ    R1, Loop      9
        stall                10
```

Data dependency

Scheduled pipelined execution:

```
Loop:   LD      F0, 0(R1)     1
        SUBI    R1, R1, #8    2
        ADDD    F4, F0, F2    3
        stall                 4
        BNEZ    R1, Loop      5
        SD      8(R1), F4     6
```

Loop Unrolling

Definition:

Replicates the loop body multiple times

```
Loop:   LD      F0, 0(R1)
        ADDD    F4, F0, F2
        SD      0(R1), F4
        SUBI    R1, R1, #8
        BEQZ    R1, Exit
        LD      F6, 0(R1)
        ADDD    F8, F6, F2
        SD      0(R1), F8
        SUBI    R1, R1, #8
        BEQZ    R1, Exit
        LD      F10, 0(R1)
        ADDD    F12, F10, F2
        SD      0(R1), F12
        SUBI    R1, R1, #8
        BEQZ    R1, Exit
        LD      F14, 0(R1)
        ADDD    F16, F14, F2
        SD      0(R1), F16
        SUBI    R1, R1, #8
        BNEZ    R1, Loop
Exit:
```

Use different registers to avoid unnecessary constraints

Pros:
- Larger basic block
- More scope for scheduling
- Eliminating dependencies

Cons:
- Increases code size

Comment:
- Often a precursor step for other optimizations

Possible hazards – Data hazard

i preceding j

- RAW *(read after write)*
  - *j tries to read a source before i writes it, so j incorrectly* gets the *old value.*

Longer operation latency more frequent stalls

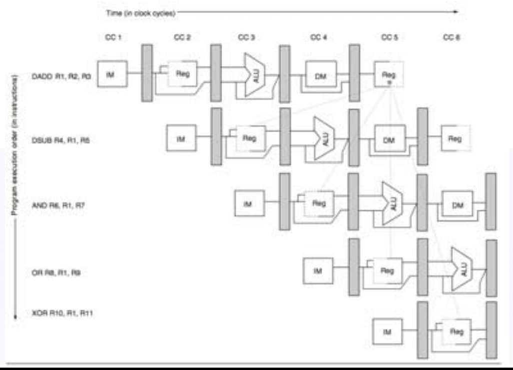
- WAW *(write after write)*
  - *j tries to write an operand before it is written by i*

Happen when
1.write in more than one pipe stage
2.one instruction proceed when previous one is stalled

- WAR *(write after read)*
  - *j tries to write a destination before it is read by i, and i incorrectly gets the new value.*

Happen when instructions are reordered

Possible hazards – Data hazard

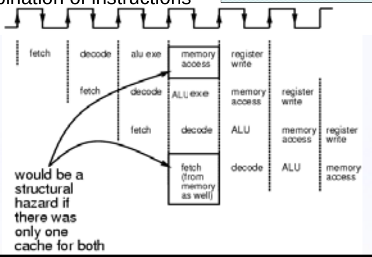


7

Possible hazards – Structural hazard

- Structural hazard
  - Resource conflicts for some combination of instructions

Happen when the overlapped execution of instructions requires to use the same functional unit.



8

Possible hazards – Control hazard

- Control hazard
  - A control hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination.

A branch is either taken or not taken

– Taken: PC<=PC+4+immediate

– Not Taken: PC<=PC+4

9

Example 1

- Consider a machine with multi-cycle functional units that is an extension of the 5 stage pipeline machine (IF ID EX MEM WB).
- Integer operations have 1 EX stage, FP Add has 3 EX stages and FP multiply has 8 EX stages.
- Each FP unit is pipelined and the pipeline supports full forwarding.
- All other stages in the pipeline complete in one cycle. Branches are resolved in the ID stage.
- For WAW hazards, assume that they are resolved through stalls and not through aborting the first instruction.
- List all of the hazards that cause stalls in the following code segment and explain why they occur.

10

Example 1

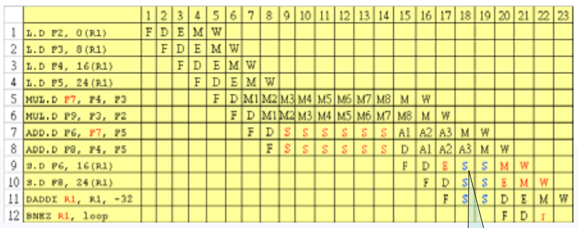
```
Loop :     1) L.D F2, 0(R1)
           2) L.D F3, 8(R1)
           3) L.D F4, 16(R1)
           4) L.D F5, 24(R1)
           5) MUL.D F7, F4, F3
           6) MUL.D F9, F3, F2
           7) ADD.D F6, F7, F5
           8) ADD.D F8, F4, F5
           9) S.D F6, 16(R1)
           10) S.D F8, 24(R1)
           11) DADDI R1, R1, -32
           12) BNEZ R1, loop
```

RAW hazard on F7

RAW hazard on R1

11

Example 1



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	L.D F2, 0(R1)	F	D	E	M	W																		
2	L.D F3, 8(R1)		F	D	E	M	W																	
3	L.D F4, 16(R1)			F	D	E	M	W																
4	L.D F5, 24(R1)				F	D	E	M	W															
5	MUL.D F7, F4, F3					F	D	M1	M2	M3	M4	M5	M6	M7	M8	M	W							
6	MUL.D F9, F3, F2						F	D	M1	M2	M3	M4	M5	M6	M7	M8	M	W						
7	ADD.D F6, F7, F5							F	D	S	S	S	S	S	S	A1	A2	A3	M	W				
8	ADD.D F8, F4, F5								F	S	S	S	S	S	S	D	A1	A2	A3	M	W			
9	S.D F6, 16(R1)															F	D	E	S	S	M	W		
10	S.D F8, 24(R1)																F	D	S	S	E	M	W	
11	DADDI R1, R1, -32																	F	S	S	D	E	M	W
12	BNEZ R1, loop																				F	D	I	

- line 7 RAW hazard on F7
- line 12 RAW hazard on R1

Structural hazard for M

12

Exercise

- The following loop is a dot product:

```
foo:     L.D    F0, 0(R1)     ;load X[i]
         L.D    F4, 0(R2)     ;load Y[i]
         MUL.D  F0, F0, F4    ;multiply X[i]*Y[i]
         ADD.D  F2, F0, F2    ;add sum=sum+X[i]*Y[i]
         ADDUI  R1, R1, #-8   ;decrement X index
         ADDUI  R2, R2, #-8   ;decrement Y index
         BNEZ   R1, foo       ;loop if not done
```

- Assume
  - the pipeline latencies from Table 1,
  - a 1-cycle delay branch.
  - a single-issue pipeline.
  - the running sum in F2 is initially 0.
  - despite the fact that the loop is not parallel, it can be scheduled with no delays.

13

Table 1

Instruction producing result	*Instruction using result*	*Latency in clock cycles*
FP ALU op	**FP ALU op**	3
FP ALU op	**Store double**	2 (ignore the structural hazard)
Load double	**FP ALU op**	1
Load double	**Store double**	0
Integer op	**Integer op**	0

14

Exercise (cont.)

- Unroll the loop a sufficient number of times to schedule it without any delays. Show the delay after eliminating any unnecessary branch maintenance instructions.
- Hint: an scheduling of the code is needed

15

Exercise: Data Dependency

```
L.D     F0, 0(R1)
L.D     F4, 0(R2)
stall
MUL.D   F0, F0, F4
stall
stall
stall
ADD.D   F2, F0, F2
ADDUI   R1, R1, #-8
ADDUI   R2, R2, #-8
BNEZ    R1, foo
stall
```

1 clock cycle

3 clock cycles

16

Unrolling Twice

```
L.D     F0, 0(R1)             L.D     F6, -8(R1)
L.D     F4, 0(R2)             L.D     F8, -8(R2)
stall                         stall
MUL.D   F0, F0, F4            MUL.D   F6, F6, F8
Stall                         Stall
Stall                         Stall
stall                         Stall
ADD.D   F2, F0, F2            ADD.D   F2, F6 , F2
~~ADDUI   R1, R1, #-8~~       ADDUI   R1, R1, #-16
~~ADDUI   R2, R2, #-8~~       ADDUI   R2, R2, #-16
BNEZ    R1, foo
stall
```

Using different registers

17

New method

Solution
- Calculate the partial sum for even and odd elements separately
- Combine the result in the end. (F2, F10)

```
Foo:   L.D     F0, 0(R1)
       L.D     F6, -8(R1)
       L.D     F4, 0(R2)
       L.D     F8, -8(R2)
       MUL.D   F0, F0, F4
       MUL.D   F6, F6, F8
       ADDUI   R1, R1, #-16
       ADDUI   R2, R2, #-16
       ADD.D   F2, F0, F2
       BNEZ    R1, foo
       ADD.D   F10, F6, F10

Bar:   ADD.D   F2, F2, F10
```

Loop body takes 11 cycles

18

Dynamic Scheduling

- Advantages
  - Enables handling some cases when dependencies are unknown at compile time
  - Allows code that was compiled with one pipeline in mind to run efficiently on a different pipeline
  - Allows **Out-of-order** execution, **Out-of-order** completion

```
DIVD    F0, F2, F4
ADDD    F10, F0, F8          (stall)
SUBD    F12, F8, F14         (have to wait)
```

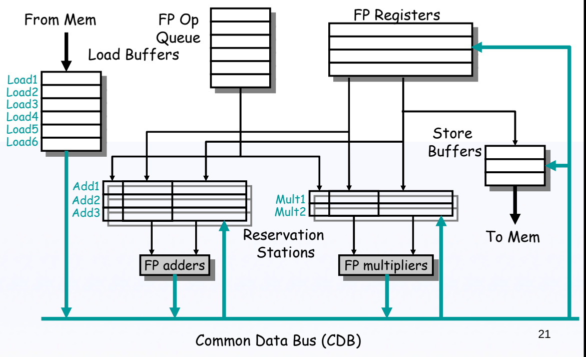
19

Tomasulo Algorithm

- Designed to overcome long memory access and floating point delays.
- RAW hazards are avoided by executing an instruction only when its operands are available.
- WAR and WAW hazards arised from name dependencies, are eliminated by ***register renaming***.
- Registers in instructions are replaced by values or pointers to ***reservation stations***.
- The ***Common Data Bus (CDB)*** is used to bypass the registers and pass the results from the reservation stations directly to the functional units.

20

Tomasulo Organization



21

Reservation Station Components

- **Busy:** Indicates reservation station or FU is busy
- **Op:** Operation to perform in the unit (e.g., + or –)
- **Vj, Vk:** Value of Source operands
- **Qj, Qk:** Reservation stations producing source registers (value to be written)
  - Note: Qj,Qk=0 => ready
- **A:** effective address

22

Three Stages of Tomasulo Algorithm

1. **Issue** —get instruction from FP Op Queue
   - If reservation station free (no structural hazard), control issues the instruction & sends operands (renames registers).
2. **Execute** —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 CDB to all awaiting units; mark reservation station available

23

Tomasulo Example (Cycle 0)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

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

24

Tomasulo Example (Cycle 1)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

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

Tomasulo Example (Cycle 2)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*		Load2		Load1					

Tomasulo Example (Cycle 3)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*	Mult1	Load2		Load1					

Tomasulo Example (Cycle 4)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*	Mult1	Load2		M(A1)	Add1				

Tomasulo Example (Cycle 5)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*	Mult1	M(A2)		M(A1)	Add1	Mult2			

Tomasulo Example (Cycle 6)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

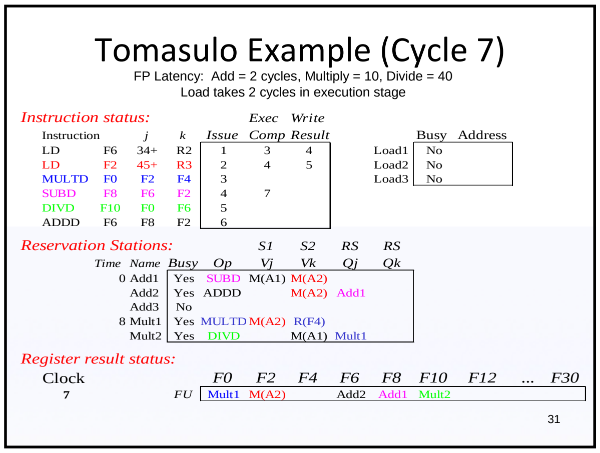
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	*FU*	Mult1	M(A2)		Add2	Add1	Mult2			



Tomasulo Example (Cycle 7)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

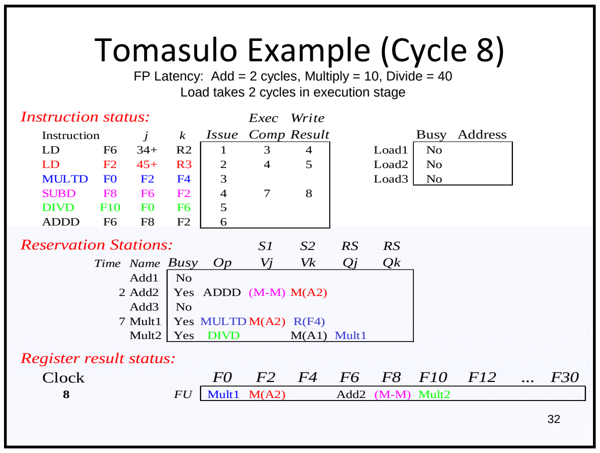
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	Mult1	M(A2)		Add2	Add1	Mult2			



Tomasulo Example (Cycle 8)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

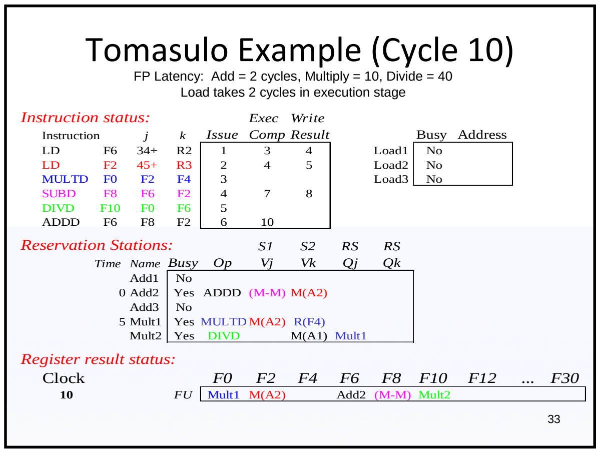
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			



Tomasulo Example (Cycle 10)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

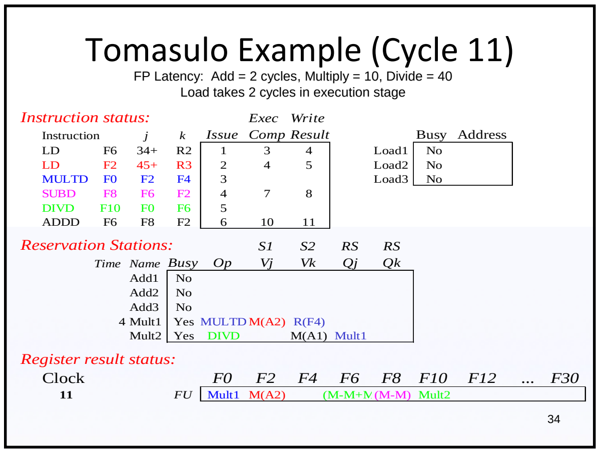
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			



Tomasulo Example (Cycle 11)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

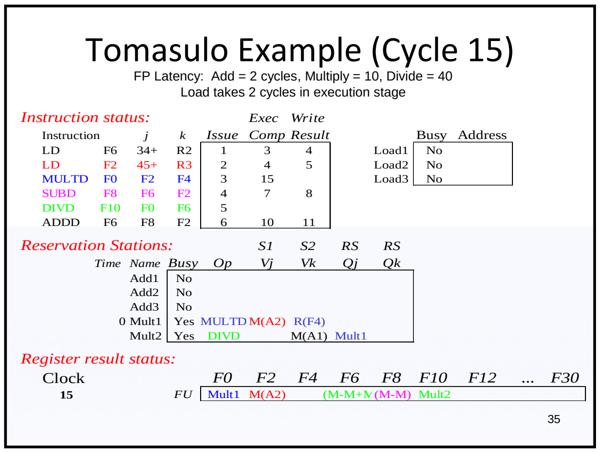
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			



Tomasulo Example (Cycle 15)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

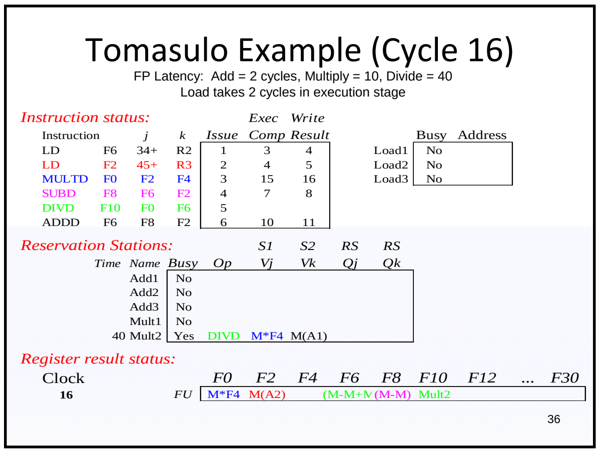
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			



Tomasulo Example (Cycle 16)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

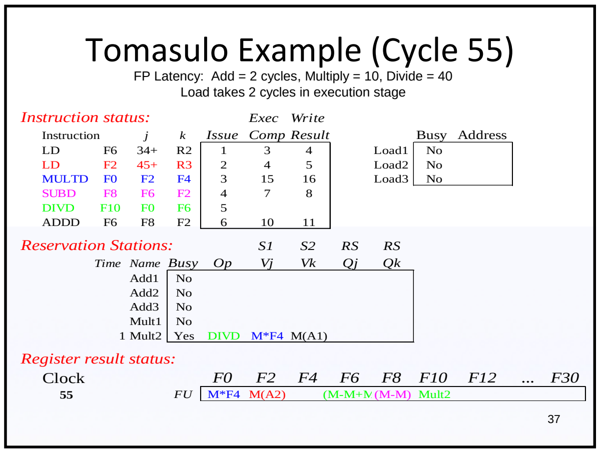
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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



Tomasulo Example (Cycle 55)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

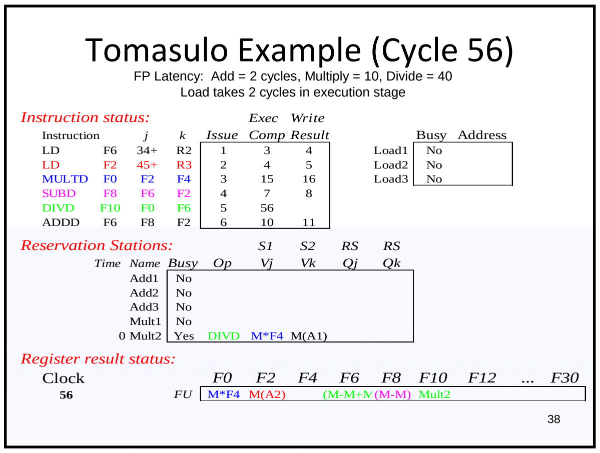
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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



Tomasulo Example (Cycle 56)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

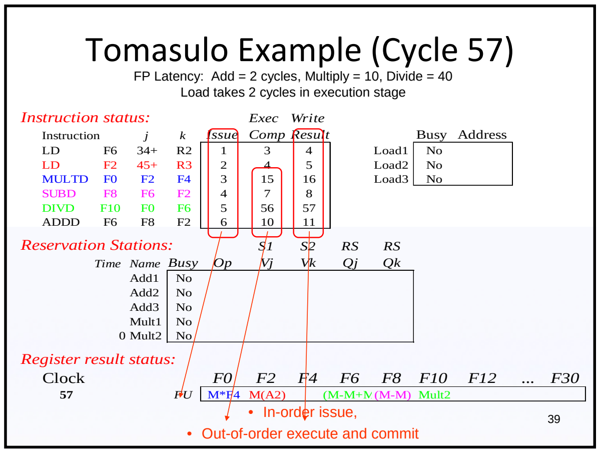
	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS 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	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			



Tomasulo Example (Cycle 57)

FP Latency: Add = 2 cycles, Multiply = 10, Divide = 40
Load takes 2 cycles in execution stage

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	No					

Register result status:

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

- In-order issue,
- Out-of-order execute and commit