Lecture 11: Instruction Level Parallelism & Dynamic Scheduling

What is ILP?

- fine-grained parallelism
- measure of inter-instruction dependency in an application
  - ✓ ILP assumes a unit-cycle operation, infinite resources, prefect front-end
  - ✓ ILP != IPC
    - IPC = # instructions / # cycles
  - ✓ ILP is the upper bound of attainable IPC
- limited by
  - ✓ data dependency
  - ✓ control dependency
- enabled and improved by RISC
  - ✓ more ILP of a RISC over CISC does not imply a better overall performance
  - ✓ CISC can be implemented like a RISC

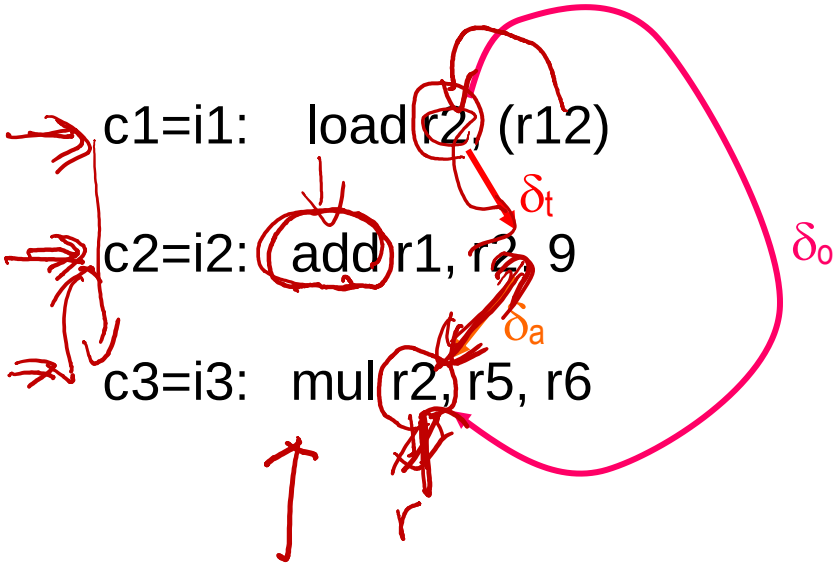
ILP Example

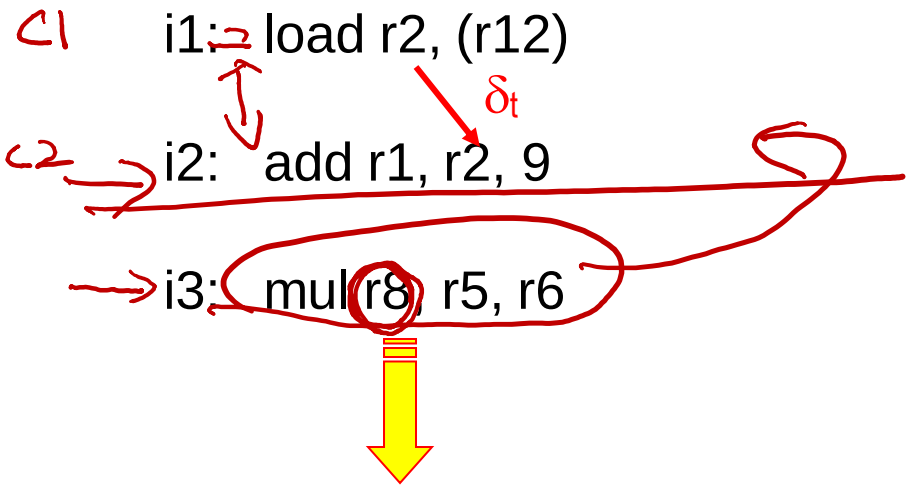
true dependency forces sequentiality

ILP = 3/3 = 1

false dependency removed

ILP = 3/2 = 1.5

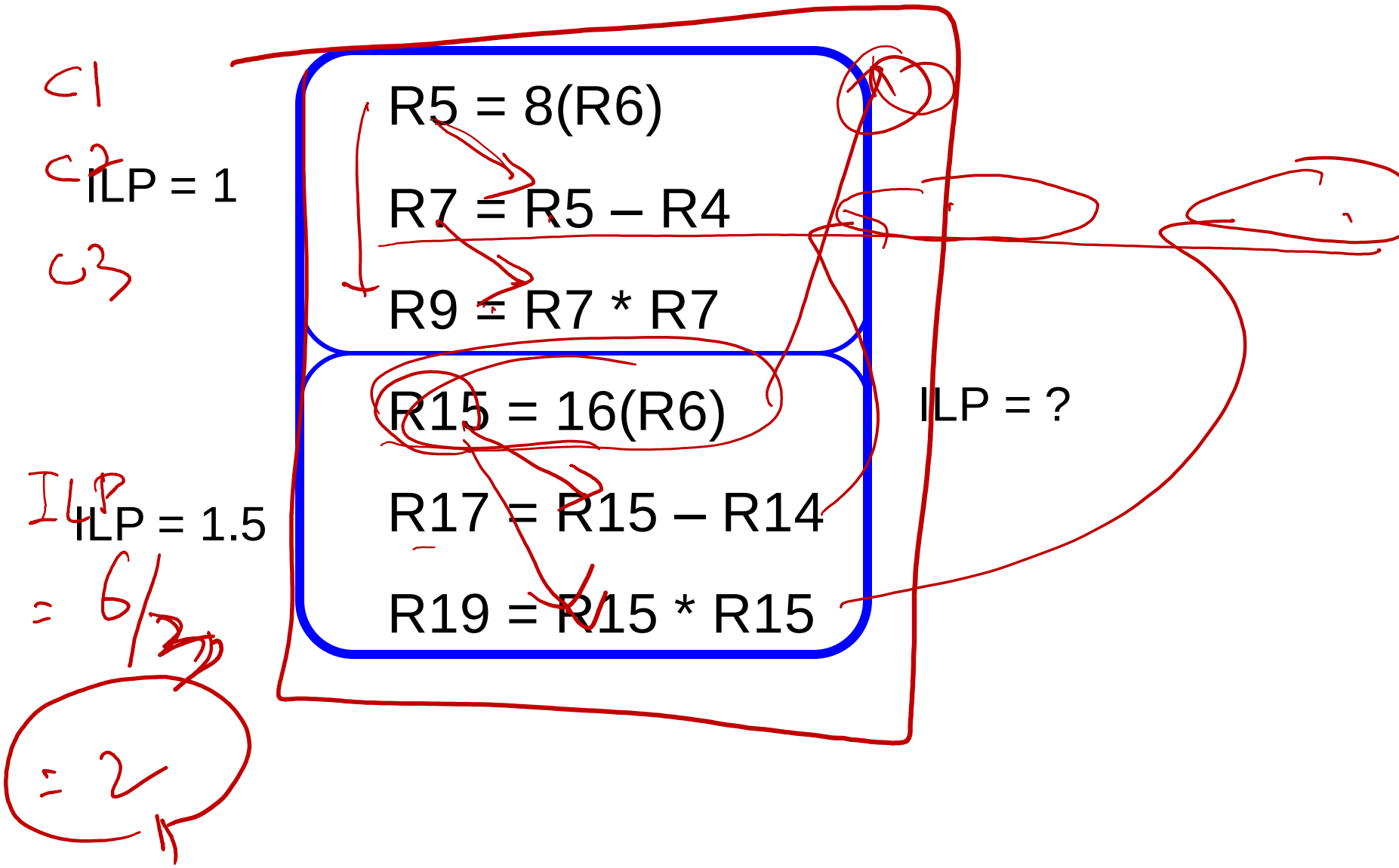




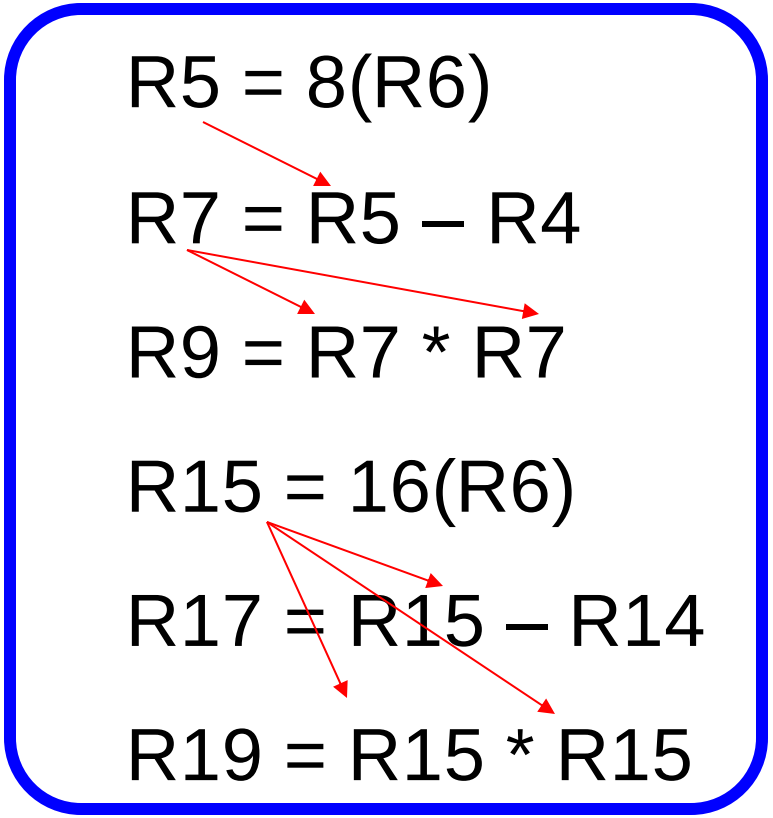
c1: load r2, (r12)

c2: add r1, r2, #9 mul r8, r5, r6

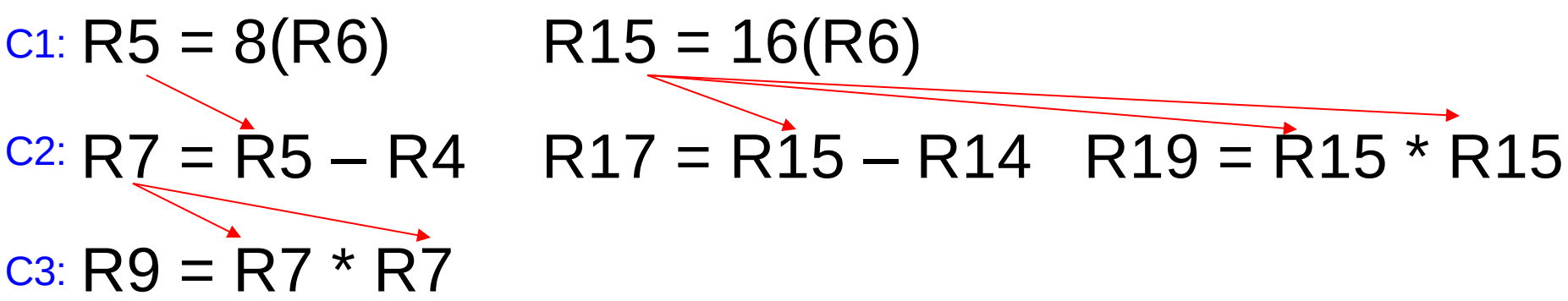
Window in Search of ILP



Window in Search of ILP



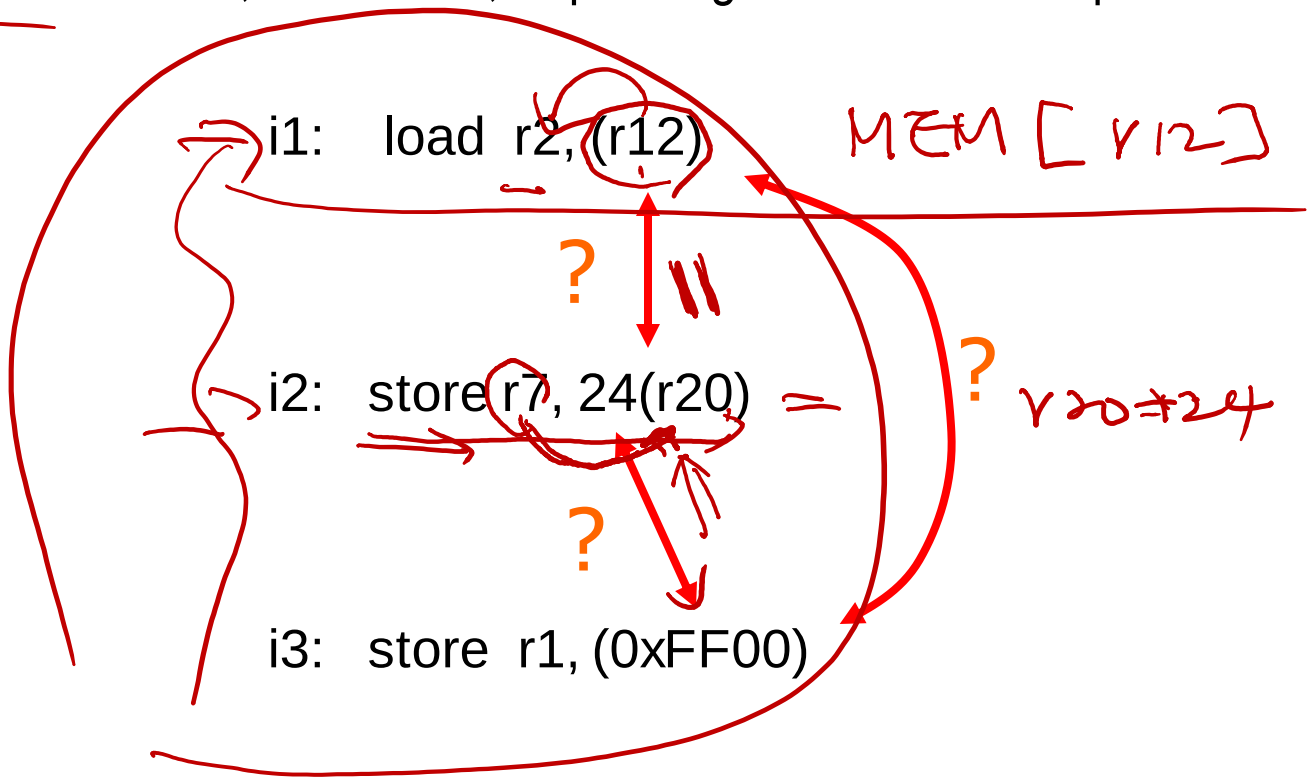
Window in Search of ILP



- ILP = 6/3 = 2 better than 1 and 1.5
- larger window gives more opportunities
  - ✓ who exploit the instruction window?
  - ✓ what limits the window?

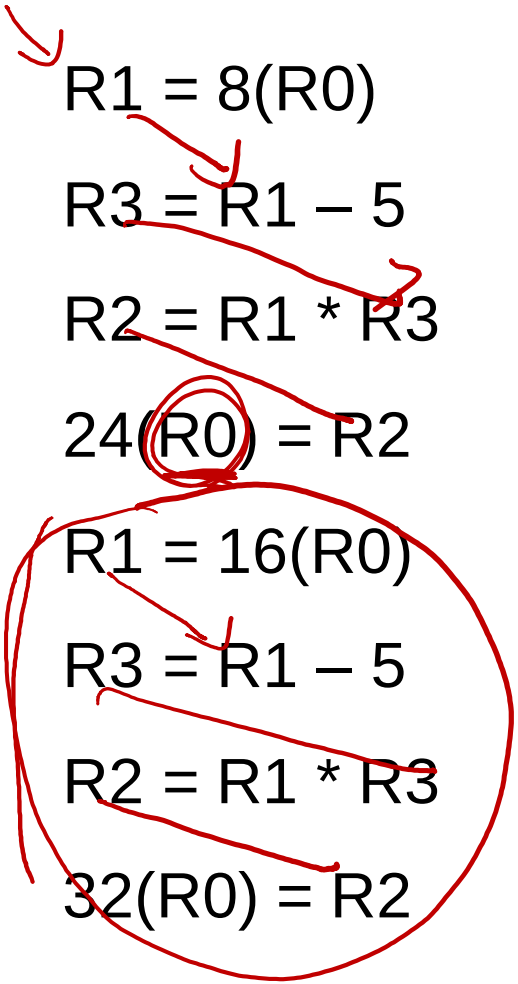
Memory Dependency

- ambiguous dependency also forces "sequentiality"
- to increase ILP
  - ✓ needs dynamic memory disambiguation mechanisms that are either safe or recoverable
- ILP could be 1, could be 3, depending on the actual dependence



ILP, Another Example

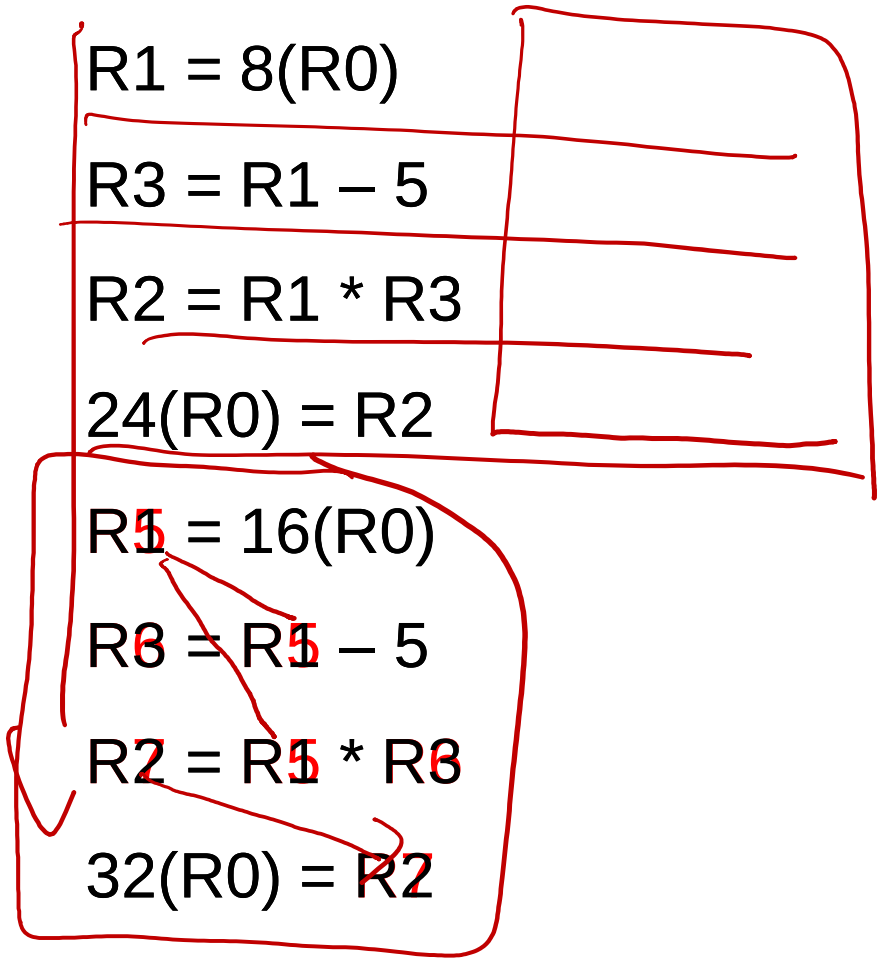
when only 4 registers available



ILP =

ILP, Another Example

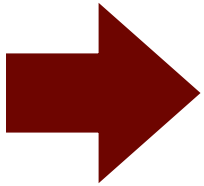
when more registers (or register renaming) available



ILP = 2

Basic Blocks

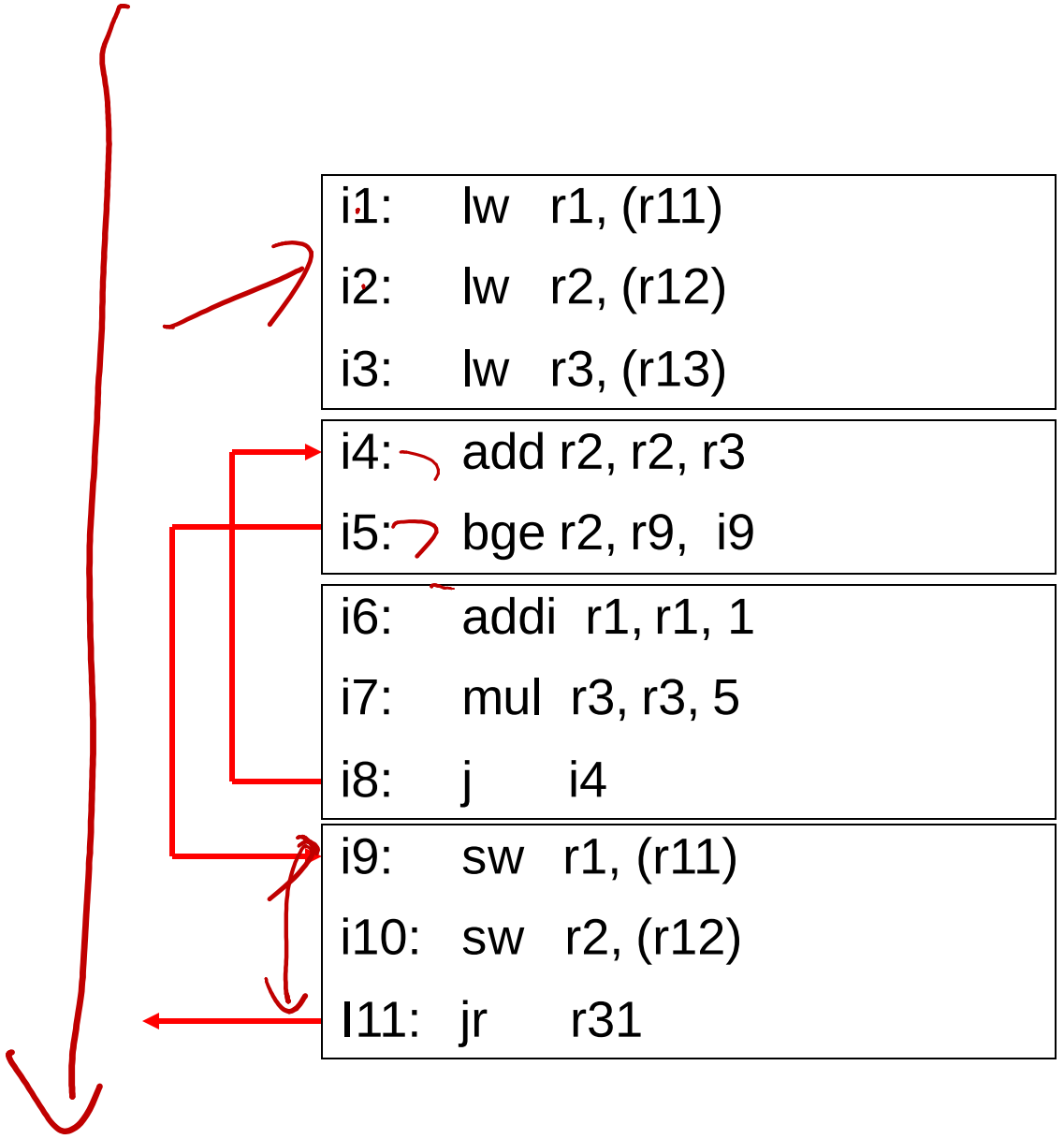
```
a = array[i];
b = array[j];
c = array[k];
d = b + c;
while (d<t) {
  a++;
  c *= 5;
  d = b + c;
}
array[i] = a;
array[j] = d;
```



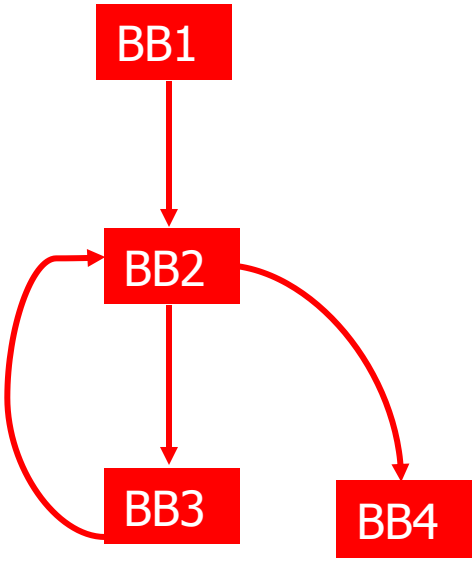
```
i1:   lw   r1, (r11)
i2:   lw   r2, (r12)
i3:   lw   r3, (r13)
i4:   add r2, r2, r3
i5:   bge r2, r9,  i9
i6:   addi  r1, r1, 1
i7:   mul  r3, r3, 5
i8:   j      i4
i9:   sw   r1, (r11)
i10:  sw   r2, (r12)
I11:  jr     r31
```

Basic Blocks

```
a = array[i];
b = array[j];
c = array[k];
d = b + c;
while (d<t) {
  a++;
  c *= 5;
  d = b + c;
}
array[i] = a;
array[j] = d;
```



Control Flow Graph



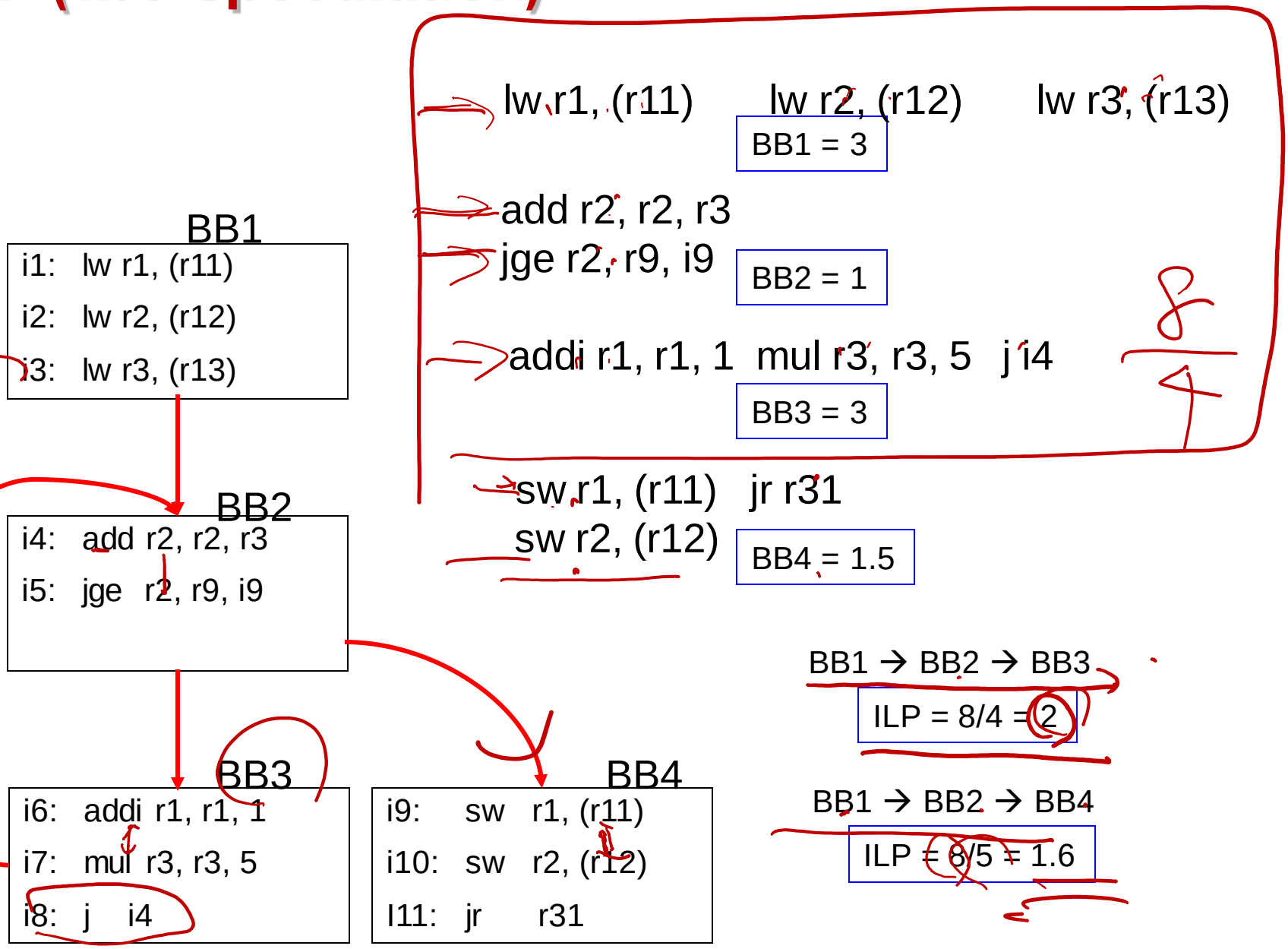

```
i1:   lw r1, (r11)
i2:   lw r2, (r12)
i3:   lw r3, (r13)
```

```
i4:   add r2, r2, r3
i5:   jge  r2, r9, i9
```

```
i6:   addi r1, r1, 1
i7:   mul  r3, r3, 5
i8:   j    i4
```

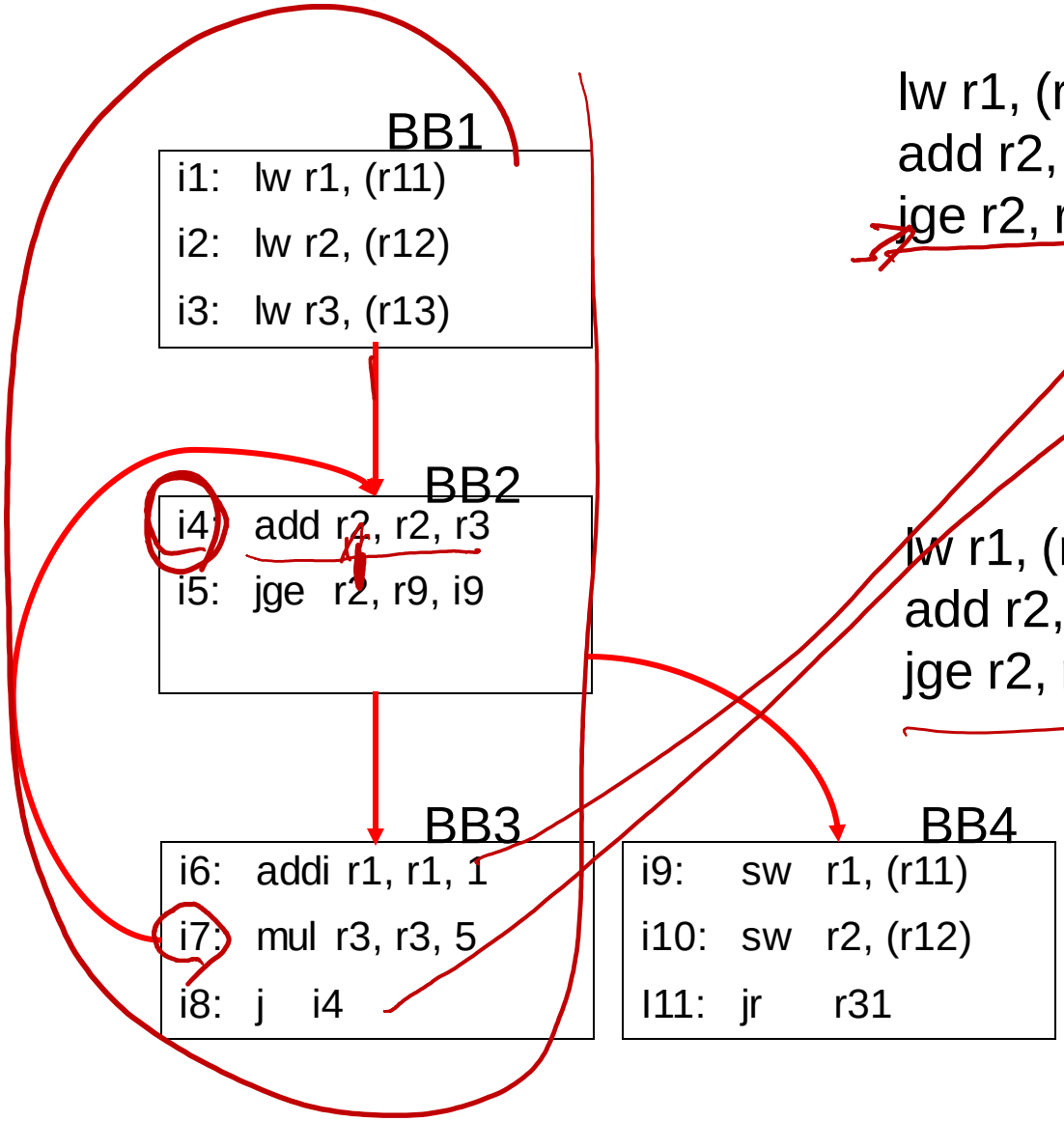
```
i9:    sw   r1, (r11)
i10:   sw   r2, (r12)
I11:   jr    r31
```

ILP (w/o Speculation)



ILP (w/ Speculation)

no control dependence



BB1 → BB2 → BB3

```
lw r1, (r11)      lw r2, (r12)      lw r3, (r13)
add r2, r2, r3    addi r1, r1, 1    mul r3, r3, 5
jge r2, r9, i9    j i4
```

ILP = 8/3 = 2.67

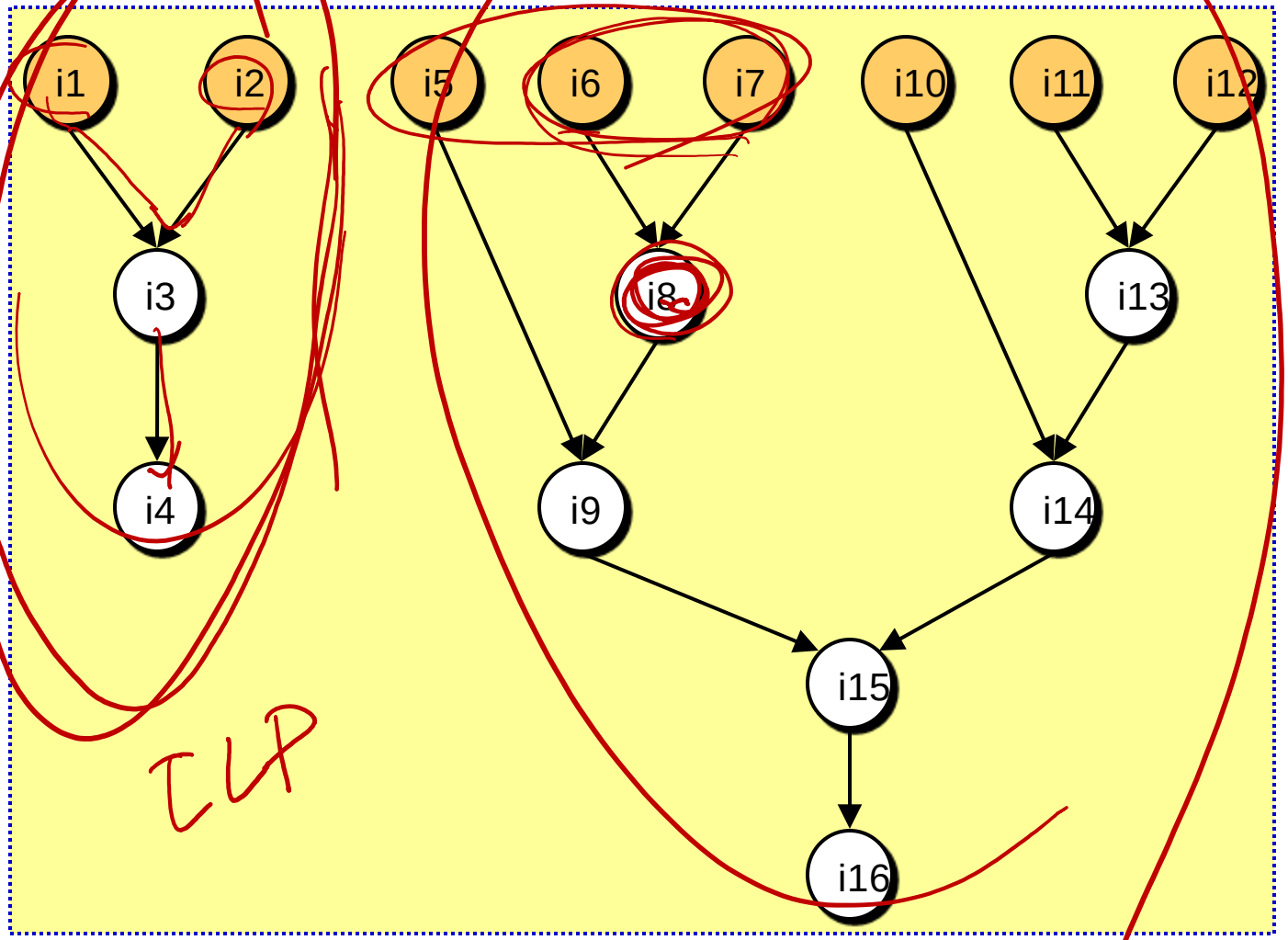
BB1 → BB2 → BB4

```
lw r1, (r11)      lw r2, (r12)      lw r3, (r13)
add r2, r2, r3    sw r1, (r11)
jge r2, r9, i9    sw r2, (r12)      jr r31
```

ILP = 8/3 = 2.67

Data Flow Execution Model

```
i1:  r2  = 4(r22)
i2:  r10 = 4(r25)
i3:  r10 = r2 + r10
i4:  4(r26) = r10
i5:  r14 = 8(r27)
i6:  r6  = (r22)
i7:  r5  = (r23)
i8:  r5  = r6 – r5
i9:  r4  = r14 * r5
i10: r15 = 12(r27)
i11: r7  = 4(r22)
i12: r8  = 4(r23)
i13: r8  = r7 – r8
i14: r8  = r15* r8
i15: r8  = r4 – r8
i16: (r28) = r8
```



Data Flow Execution Model

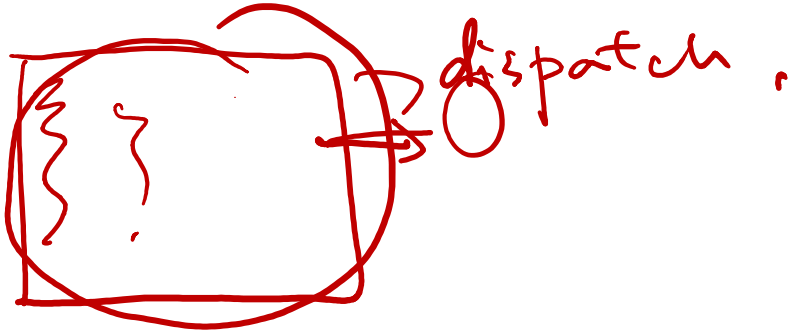
- to exploit maximal ILP
- an instruction can be executed immediately after
  - ✓ all source operands are ready
  - ✓ execution unit available
  - ✓ destination is ready (to be written)

Dynamic Scheduling

- to exploit ILP at run-time, execute instructions OOO
- HW will
  - ✓ maintain true dependency (data flow manner)
  - ✓ find ILP within an instruction window (pool)
    - need an accurate branch predictor
- pros
  - ✓ scalable performance: allows code to be compiled on one platform, but also run efficiently on another
  - ✓ handle cases where dep is unknown at compile-time
- cons
  - ✓ hardware complexity (main argument from the VLIW/EPIC camp)

OOO Execution

- OOO execution ≡ out-of-order completion
- OOO execution ≠ out-of-order retirement (commit)
- no (speculative) instruction allowed to retire until it is confirmed on the right path
- fetch, decode, issue (i.e., front-end) are still done in the program order



IBM 360

- IBM 360 introduced
  - ✓ 8-bit = 1 byte
  - ✓ 32-bit = 1 word
  - ✓ byte-addressable memory
- Tomasulo algorithm
  - ✓ dynamic scheduling
  - ✓ register renaming

Dr. Blauwy

Tomasulo Algorithm

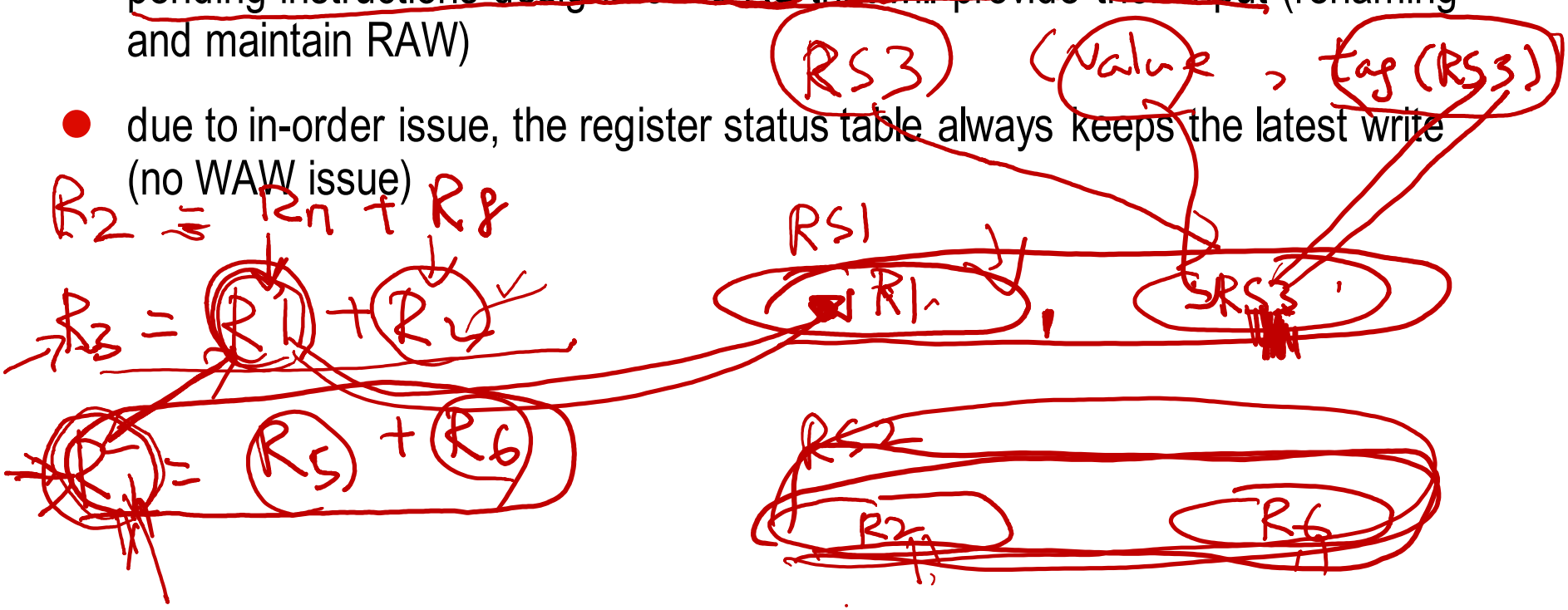
- goal: high performance without special compilers
  - ✓ dynamic scheduling done completely by HW
  - ✓ generally use "supercalar processor" for such category as opposed to "VLIW" or "EPIC"
- why study? Lead to Pentium Pro/II/III/4, Core, Alpha 21264, MIPS R10000, HP 8000, PowerPC 604

IBM 360/91 FPU w/ Tomasulo Algorithm

- not to stall FP instructions due to long latency
  - ✓ two function units —— FP Add + FP Mult/Div
- three new mechanisms
  - ✓ reservation stations (RS)
    - 3 in FP add, 2 in FP mult/div
    - register name is discarded when issue to reservation station
  - ✓ tags
    - 4-bit tag for one of the 11 possible sources (5 RSs + 6 FLB for loads)
    - written for unavailable sources whose results are being generated by one of the sources (5 RS or 6 FLB)
    - new tag assignment eliminates false dependency
  - ✓ common data bus (CDB), driven by
    - 11 sources: 5 RS + 6 FLB
    - 17 destinations: 2*5 RS + 3 SDB + 4 FLR

Basic Principles

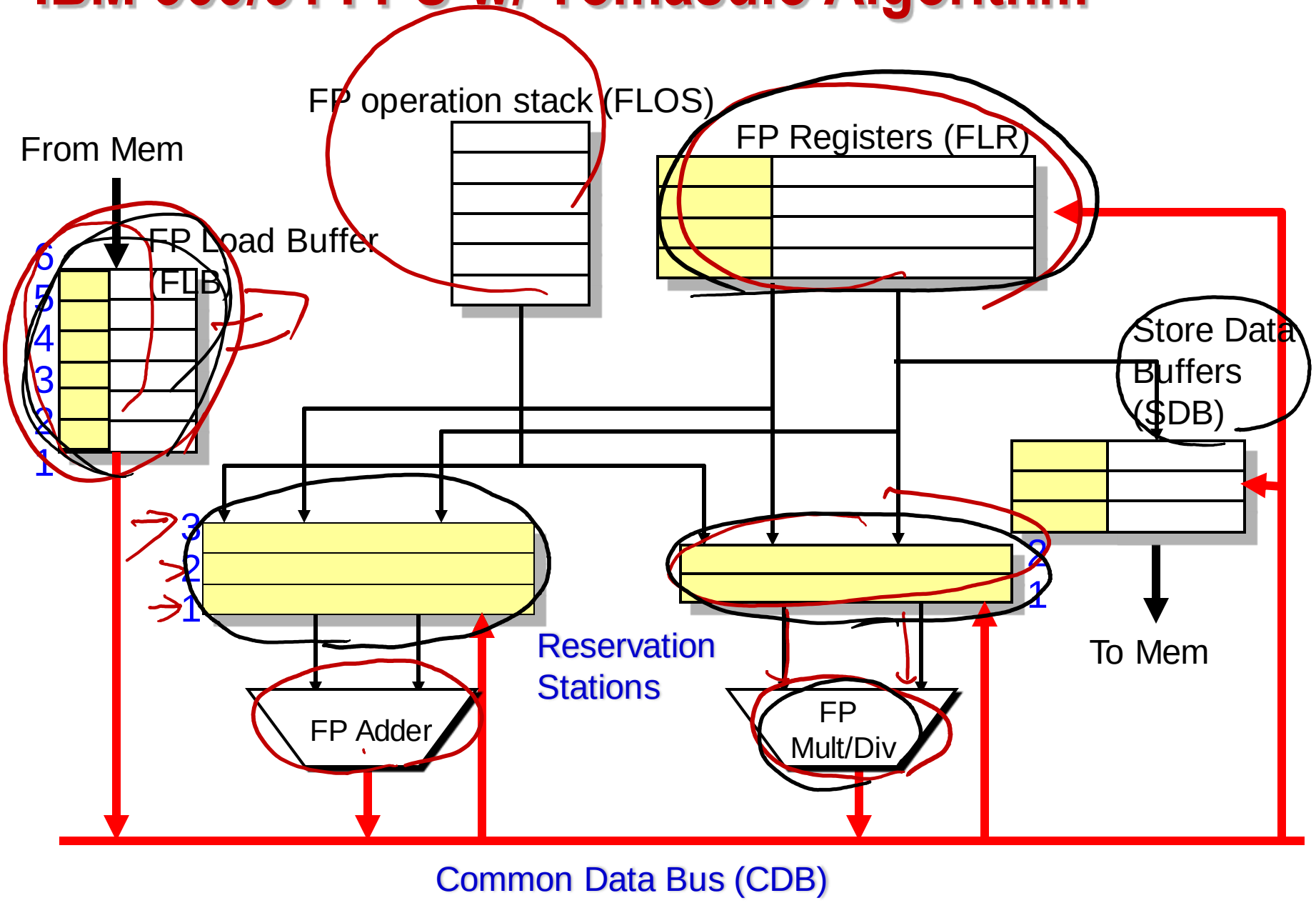
- do not rely on a centralized register file !
- RS fetches and buffers an operand as soon as it is available via CDB
  - ✓ eliminating the need to get it from a register (No WAR)
  - ✓ data flow execution model
- pending instructions designate the RS that will provide their input (renaming and maintain RAW)
- due to in-order issue, the register status table always keeps the latest write (no WAW issue)



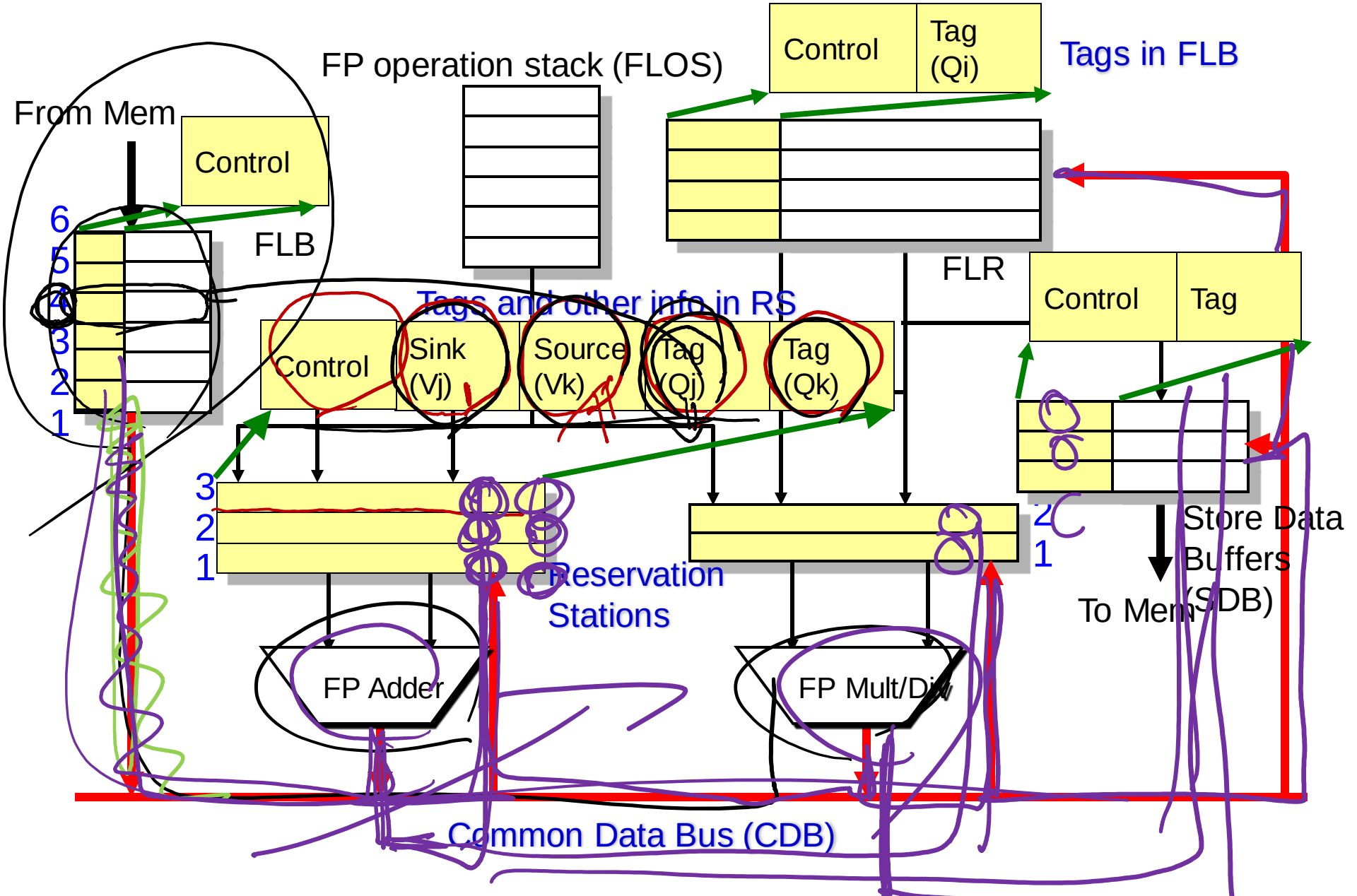
Key Representation

- Op — operation to perform in the units
- Vj — value of source 1 (called SINK in 360/91)
- Vk — value of source 2 (called SOURCE in 360/91)
- Qj — RS (tag) will produce source 1
- Qk — RS (tag) will produce source 2
- A(ddress) — hold info for the memory address generation for a load or store
- Qi — whose value should be stored into the register

IBM 360/91 FPU w/ Tomasulo Algorithm



IBM 360/91 FPU w/ Tomasulo Algorithm



RAW Example:

i: R2 ← R0 + R4 (2)
j: R8 ← R0 + R2 (2)

Cycle #0:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4			10.0
8			7.8

Cycle #1: Issue i

RS	Tag	Sink	Tag	Src
1	0	6.0	0	10.0
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4			10.0
8			7.8

Cycle #2: Issue j

RS	Tag	Sink	Tag	Src
1	0	6.0	0	10.0
2	0	6.0	1	
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4			10.0
8	1	2	---

RAW Example:

i: R2 ← R0 + R4 (2)
j: R8 ← R0 + R2 (2)

Cycle #3: Broadcasts tag and result: CDB_a=<RS1,16.0>

RS	Tag	Sink	Tag	Src
1				
2	0	6.0	0	16.0
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			16.0
4			10.0
8	1	2	---

Cycle #5: Broadcasts tag and result: CDB_a=<RS2,22.0>

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			16.0
4			10.0
8			22.0

WAR Example:

i: R4 ← R0 x R8 (3)
j: R0 ← R4 x R2 (3)
k: R2 ← R2 + R8 (2)

Cycle #0:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4			10.0
8			7.8

Cycle #1: Issue i

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4	1	4	---
8			7.8

Cycle #2: Issue j

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5	4		0	3.5

Multiplier/Divider

FLR	Busy	Tag	Data
0	1	5	---
2			3.5
4	1	4	---
8			7.8

WAR Example:

i: R4 ← R0 x R8 (3)
j: R0 ← R4 x R2 (3)
k: R2 ← R2 + R8 (2)

Cycle #3: Issue k

RS	Tag	Sink	Tag	Src
1	0	3.5	0	7.8
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5	4	---	0	3.5

Multiplier/Divider

FLR	Busy	Tag	Data
0	1	5	---
2	1	1	---
4	1	4	---
8			7.8

Cycle #4: Broadcasts CDB_m=<RS4,46.8>;

RS	Tag	Sink	Tag	Src
1	0	3.5	0	7.8
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5	0	46.8	0	3.5

Multiplier/Divider

FLR	Busy	Tag	Data
0	1	5	---
2	1	1	---
4			46.8
8			7.8

Cycle #5: Broadcasts CDB_a=<RS1,11.3>

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5	0	46.8	0	3.5

Multiplier/Divider

FLR	Busy	Tag	Data
0	1	5	---
2			11.3
4			46.8
8			7.8

WAR Example:

i: R4 ← R0 x R8 (3)
j: R0 ← R4 x R2 (3)
k: R2 ← R2 + R8 (2)

Cycle #7: Broadcasts CDB_m=<RS5,163.8>

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			163.8
2			11.3
4			46.8
8			7.8

WAW Example:

i: R4 ← R0 x R8 (3)
j: R2 ← R0 + R4 (2)
k: R4 ← R0 + R8 (2)

Cycle #0:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4			10.0
8			7.8

Cycle #1: Issue i

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4	1	4	---
8			7.8

Cycle #2: Issue j

RS	Tag	Sink	Tag	Src
1	0	6.0	4	---
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4	1	4	---
8			7.8

WAW Example:

i: R4 ← R0 x R8 (3)
j: R2 ← R0 + R4 (2)
k: R4 ← R0 + R8 (2)

Cycle #3: Issue k

RS	Tag	Sink	Tag	Src
1	0	6.0	4	---
2	0	6.0	0	7.8
3				

Adder

RS	Tag	Sink	Tag	Src
4	0	6.0	0	7.8
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4	1	2	---
8			7.8

Cycle #4: Broadcasts CDB_m=<RS4,46.8>

RS	Tag	Sink	Tag	Src
1	0	6.0	0	46.8
2	0	6.0	0	7.8
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4	1	2	---
8			7.8

Cycle #5: Broadcasts CDB_a=<RS2,13.8>

RS	Tag	Sink	Tag	Src
1	0	6.0	0	46.8
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2	1	1	---
4			13.8
8			7.8

WAW Example:

i: R4 ← R0 x R8 (3)
j: R2 ← R0 + R4 (2)
k: R4 ← R0 + R8 (2)

Cycle #6: Broadcasts CDB_a=<RS1,52.8>

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			52.8
4			13.8
8			7.8

Yet Another Example

i: R2 ← R2 + R4 (2)
j: R8 ← R0 × R2 (3)
k: R2 ← R8 × R2 (3)

Cycle #0:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			6.0
2			3.5
4			10.0
8			7.8

Cycle #1:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #2:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

i: R2 ← R2 + R4 (2)
j: R8 ← R0 × R2 (3)
k: R2 ← R8 × R2 (3)

Cycle #3:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #4:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #5:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

i: R2 ← R2 + R4 (2)
j: R8 ← R0 × R2 (3)
k: R2 ← R8 × R2 (3)

Cycle #6:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #7:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #8:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

i: R2 ← R2 + R4 (2)
j: R8 ← R0 × R2 (3)
k: R2 ← R8 × R2 (3)

Cycle #9:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #10:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Cycle #11:

RS	Tag	Sink	Tag	Src
1				
2				
3				

Adder

RS	Tag	Sink	Tag	Src
4				
5				

Multiplier/Divider

FLR	Busy	Tag	Data
0			
2			
4			
8			

Issues in Tomasulo Algorithm

- CDB at high speed?
- precise exception issues
- speculative instructions
  - ✓ branch prediction enlarges instruction window
  - ✓ how to rollback when mispredicted?

Tomasulo Summary

- prevents register as bottleneck
- avoids/removes WAR, WAW hazards
- lasting contributions
  - ✓ dynamic scheduling
  - ✓ register renaming (in what way does the register name change?)
  - ✓ load/store disambiguation

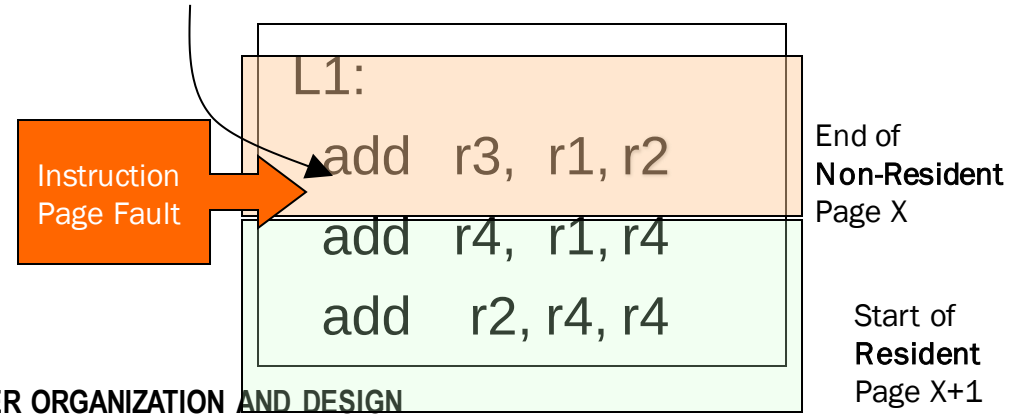
Major Challenges

- exceptions/interrupts
  - ✓ can't identify a particular point in the program at which an interrupt/exception occurs
  - ✓ how do you know where to go back to after an interrupt handler completes?
- interaction with pipelined ALUs and cache memories
  - ✓ reservation station couldn't be released until instruction completes,
  - ✓ cache miss can take many cycles
  - ✓ would need many reservation stations – large instruction window

Issue w/ Imprecise Interrupt

- add instructions take one cycle
  - ✓ e.g.,
    - Load (left side) induces a “data page fault”;
    - Add (right side) induces an “instruction page fault”
  - ✓ if out-of-order completion is allowed
    - r10, r12, (or r3, r4) … will be modified
    - wrong values will be used by the re-issued load
  - ✓ interrupt classes
    - program interrupts (exceptions or traps)
    - external interrupts (asynchronous)

```
lw    r5,  8(r10)
add   r10, r9, r8
add   r12, r10, r7
```



Precise Interrupts

- to reflect a sequential architecture model $\Rightarrow$ serially correct (think about a single issue, non-pipelined processor)
- keep precise state of an execution
  - ✓ all instructions before the interrupted instruction must be completed
  - ✓ state should appear as if no instruction issued after the interrupted instruction
  - ✓ interrupted PC should be presented to the interrupt handler (restartable)
- similar to branch misprediction handling
- out-of-order execution makes the ordering hard
  - ✓ undo what comes after an interrupt

Why Supporting Precise Interrupts

- need to maintain a precise state (for recovery)
  - ✓ need to know where to resume execution when interrupt handler finishes
  - ✓ debugging -- present consistent view to programmer at breakpoints/errors
  - ✓ emulation of instructions in software

Support Precise Interrupt

- buffer results
- reconstruct the scenario (state) as sequential execution
- restart from saved PC with saved PC state

Announcement

- next lecture: dynamic scheduling (speculative execution)
  - ✓ Ch. 3.4 – 3.8 (HP2)
- MP assignment
  - ✓ MP3 checkpoint 1 due on 3/11 5pm

Tomasulo Example (H&P Text)

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					

These are RS, we have only one FU for each type (MUL, ADD, LD). We reduce Load from 6 to 3 for simplicity. SDB is not shown either

Register result status:

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

Assumption

- INT (load) — 1 cycle
- MULT — 10 cycles
- ADD — 2 cycles
- DIVIDE — 40 cycles

Tomasulo Example Cycle 1

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	Qi				Load1					

Tomasulo Example Cycle 2

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	Qi		Load2		Load1					

RS enables multiple outstanding loads

Load is calculating the effective address

Tomasulo Example Cycle 3

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	Qi	Mult1	Load2		Load1					

Regs names are “renamed” in RSs; MULT issued vs. scoreboard

Load1 completing; what is waiting for Load1?

Tomasulo Example Cycle 4

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	Qi	Mult1	Load2		M(A1)	Add1				

Load1 write to CDB; Load2 completing; what is waiting for Load2?

Tomasulo Example Cycle 5

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		R(F6)	Mult1	

Register result status:

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

Tomasulo Example Cycle 6

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		R(F2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	Qi	Mult1			Add2	Add1	Mult2			

R(F6) was entered in Cycle 5

Issue ADDD here vs. scoreboard?

Tomasulo Example Cycle 7

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		R(F2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	Qi	Mult1			Add2	Add1	Mult2			

Add1 completing; what is waiting for it?

Tomasulo Example Cycle 8

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	M1-M2	R(F2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	Qi	Mult1			Add2	M1-M2	Mult2			

Tomasulo Example Cycle 9

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					
1	Add2	Yes	ADDD	M1-M2	R(F2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1			Add2		Mult2			

Tomasulo Example Cycle 10

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	M1-M2	R(F2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1			Add2		Mult2			

Add2 completing; what is waiting for it?

Tomasulo Example Cycle 11

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		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1		(M1-M2+M(A2))			Mult2			

Write result of ADDD here vs. scoreboard?

All quick instructions complete in this cycle!

Tomasulo Example Cycle 12

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					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Mult2			

Tomasulo Example Cycle 13

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					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1					Mult2			

Tomasulo Example Cycle 14

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					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1					Mult2			

Tomasulo Example Cycle 15

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		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1					Mult2			

Tomasulo Example Cycle 16

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	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	M*F4					Mult2			

Faster than light computation (skip a couple of cycles)

Tomasulo Example Cycle 55

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	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU						Mult2			

Tomasulo Example Cycle 56

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	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU						Mult2			

Mult2 is completing; what is waiting for it?

Tomasulo Example Cycle 57

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					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU						Result			

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