EECS 470

Tomasulo's Algorithm

Lecture 4 – Winter 2024

Slides developed in part by Profs. Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, and Wenisch of Carnegie Mellon University, Purdue University, University of Michigan, University of Pennsylvania, and University of Wisconsin.

Announcements

Programming assignment #1 due today

Homework #2 posted by end of the day

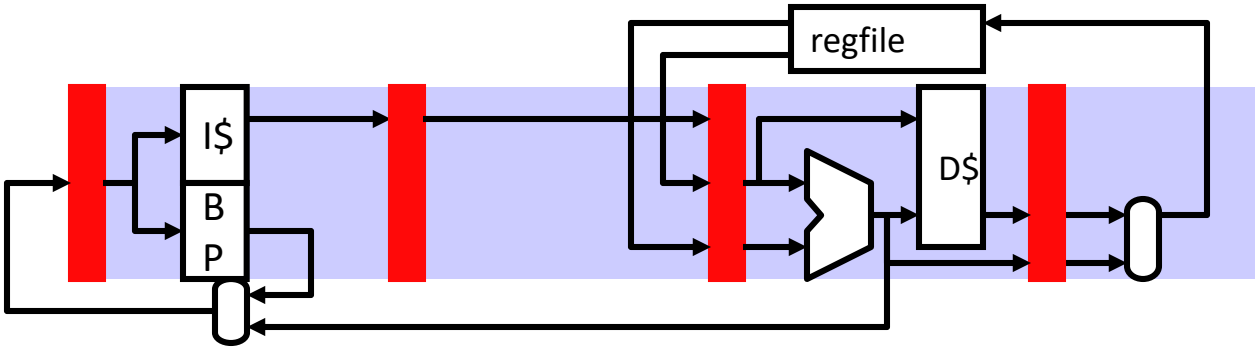
- Due Friday 2/2
- Can do some of it now, and should be able to do all by Tuesday of next week.

Readings

H & P Chapter 3.4-3.5

Basic Anatomy of an OoO Scheduler

New Pipeline Terminology



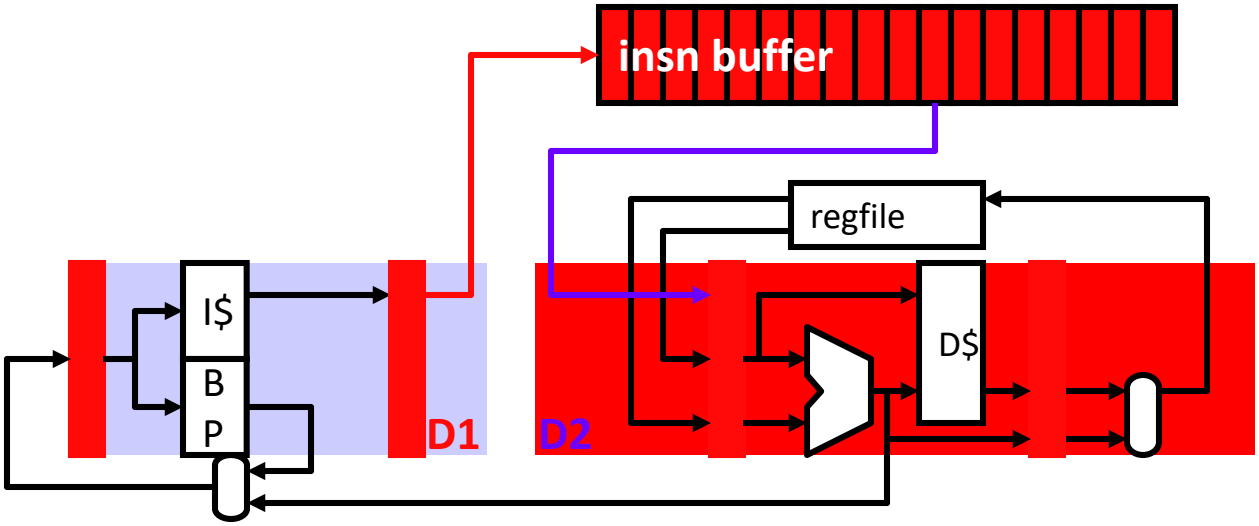
- **In-order pipeline**
  - Often written as F,D,X,W (multi-cycle X includes M)
  - Example pipeline: 1-cycle int (including mem), 3-cycle pipelined FP

New Pipeline Diagram

Insn	D	X	W
`ldf X(r1),f1`	c1	c2	c3
`mulf f0,f1,f2`	c3	c4+	c7
`stf f2,Z(r1)`	c7	c8	c9
`addi r1,4,r1`	c8	c9	c10
`ldf X(r1),f1`	c10	c11	c12
`mulf f0,f1,f2`	c12	c13+	c16
`stf f2,Z(r1)`	c16	c17	c18

- Alternative pipeline diagram (we will see two approaches in class)
  - Down: instructions executing over time
  - Across: pipeline stages
  - In boxes: the specific cycle of activity, for that instruction
  - Basically: stages ↔ cycles
  - Convenient for out-of-order

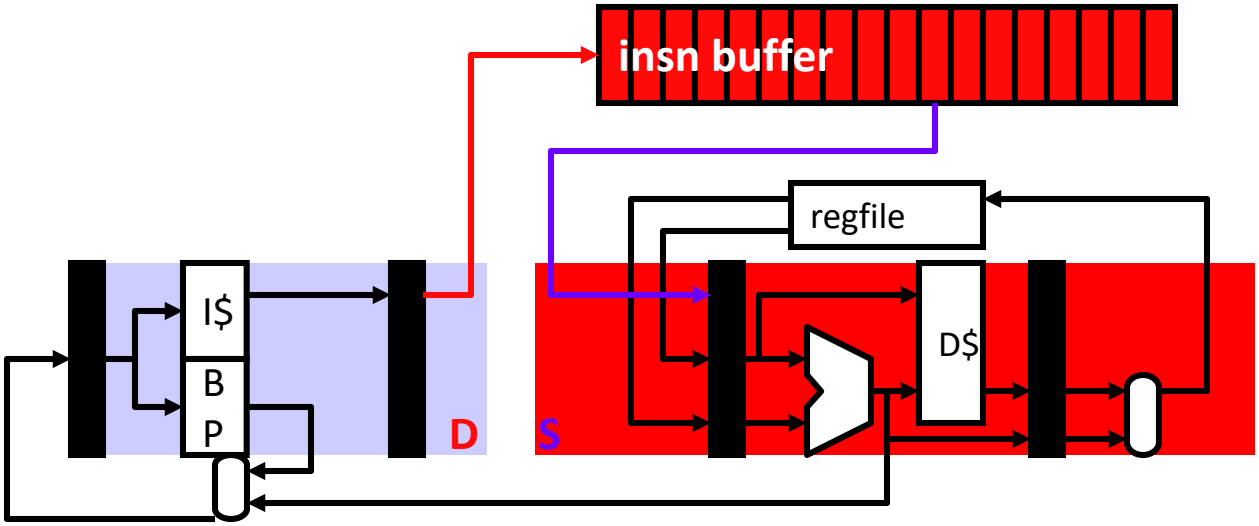
Anatomy of OoO: Instruction Buffer



- **Insn buffer** (many names for this buffer)
  - Basically: a bunch of latches for holding insns
  - Candidate pool of instructions

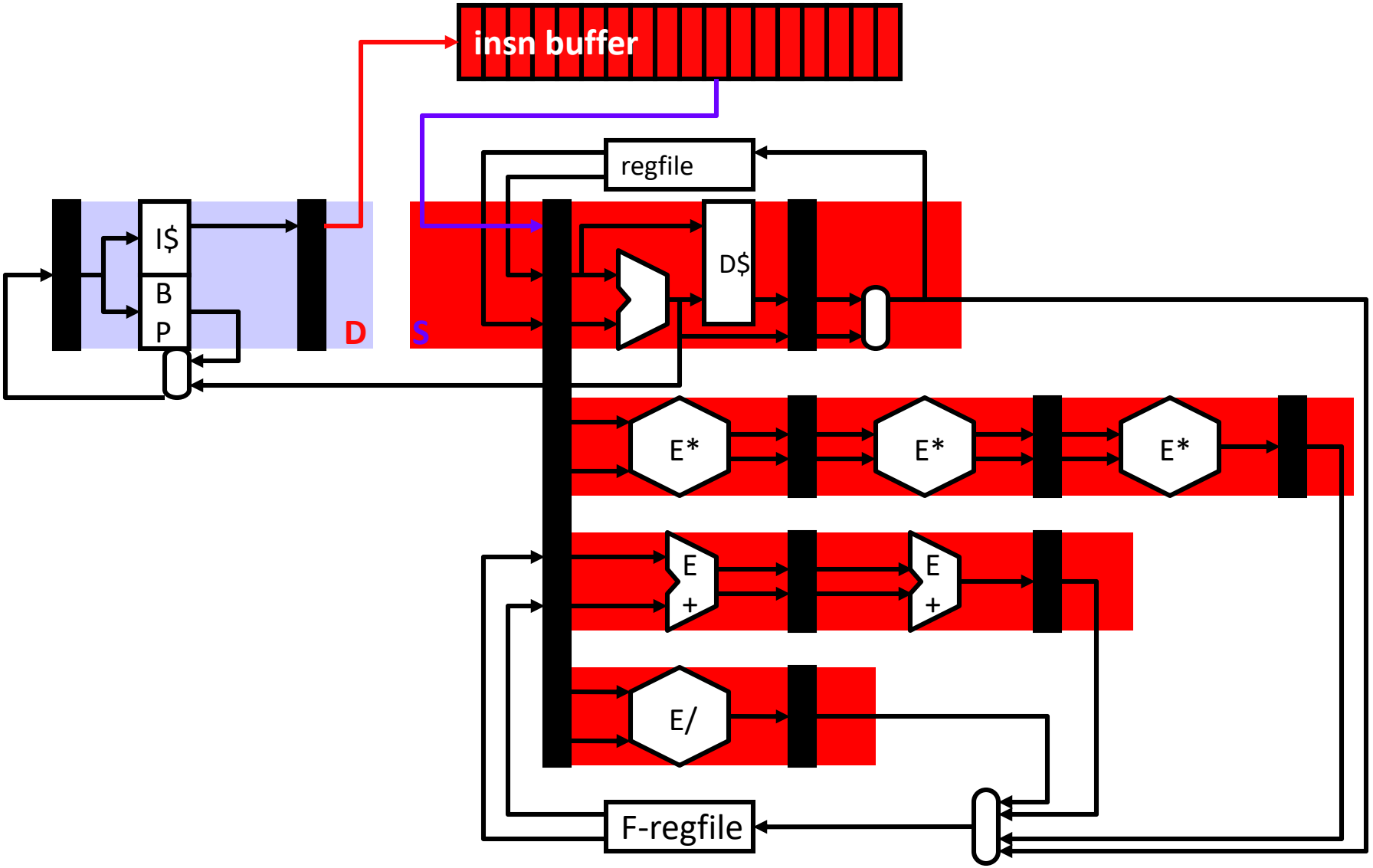
- Split D into two pieces
  - Accumulate decoded insns in buffer **in-order**
  - Buffer sends insns down rest of pipeline **out-of-order**

Anatomy of OoO: Dispatch and Issue



- **Dispatch (D)**: first part of decode
  - Allocate slot in insn buffer
    - New kind of structural hazard (insn buffer is full)
  - In order: **stall** back-propagates to younger insns
- **Issue (S)**: second part of decode
  - Send insns from insn buffer to execution units
  - Out-of-order: **wait** doesn't back-propagate to younger insns

Dispatch and Issue with Floating-Point



Dynamic Scheduling Algorithms

- **Register scheduler**: scheduler driven by register dependences

- Book covers two register scheduling algorithms
  - Scoreboard: No register renaming → limited scheduling flexibility
  - Tomasulo: Register renaming → more flexibility, better performance
  - We focus on Tomasulo's algorithm in the lecture
  - No test questions on scoreboarding
    - Do note that it is used in certain GPUs.

- Big simplification in this lecture: **memory scheduling**
  - Pretend register algorithm magically knows memory dependences
  - A little more realism later in the term

Key OoO Design Feature: Issue Policy and Issue Logic

- Issue
  - If multiple instructions are ready, which one to choose? **Issue policy**
    - Oldest first? Safe
    - Longest latency first? May yield better performance
  - **Select logic**: implements issue policy
    - Most projects use random.

Eliminating False Dependencies with Register Renaming

True Data dependencies

- True data dependency
  - RAW – Read after Write

R1=R2+R3

R4=R1+R5

- True dependencies prevent reordering
  - (Mostly) unavoidable

False Data Dependencies

- False or Name dependencies
  - WAW – Write after Write

    R1=R2+R3

    R1=R4+R5
  - WAR – Write after Read

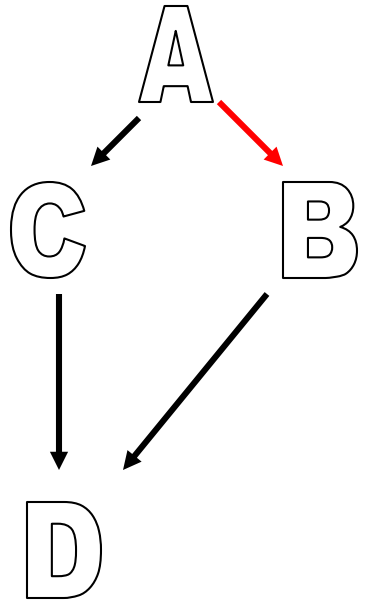
    R2=R1+R3

    R1=R4+R5
- False dependencies prevent reordering
  - Can they be eliminated? (Yes, with renaming!)

Data Dependency Graph: Simple example

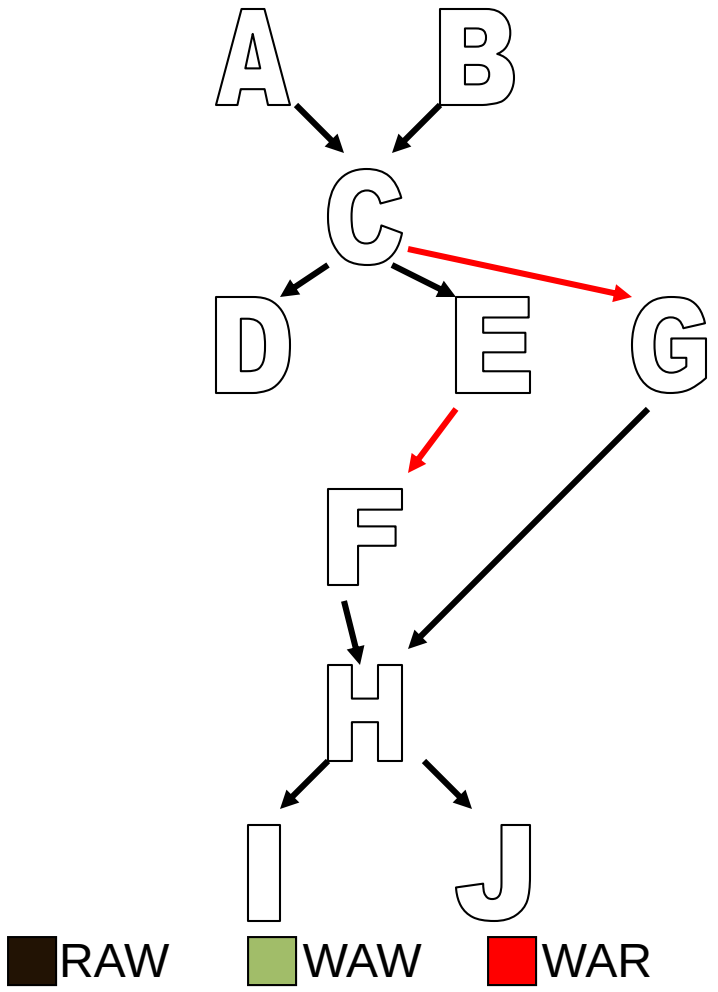
```
R1=MEM[R2+0]   // A
R2=R2+4        // B
R3=R1+R4       // C
MEM[R2+0]=R3   // D
```



RAW WAW WAR

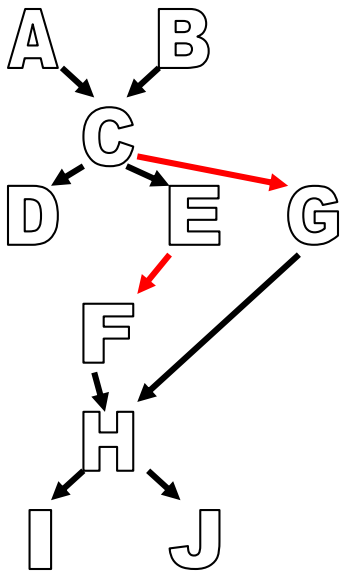
Data Dependency Graph: More complex example

```
R1=MEM[R3+4]     // A
R2=MEM[R3+8]     // B
R1=R1*R2         // C
MEM[R3+4]=R1     // D
MEM[R3+8]=R1     // E
R1=MEM[R3+12]    // F
R2=MEM[R3+16]    // G
R1=R1*R2         // H
MEM[R3+12]=R1    // I
MEM[R3+16]=R1    // J
```



Eliminating False Dependencies

```
R1=MEM[R3+4]     // A
R2=MEM[R3+8]     // B
R1=R1*R2         // C
MEM[R3+4]=R1     // D
MEM[R3+8]=R1     // E
R1=MEM[R3+12]    // F
R2=MEM[R3+16]    // G
R1=R1*R2         // H
MEM[R3+12]=R1    // I
MEM[R3+16]=R1    // J
```



- Well, logically there is no reason for F-J to be dependent on A-E. So…..
  - ABFG
  - CH
  - DEIJ
  - Should be possible.
- But that would cause either C or H to have the wrong reg inputs
- How do we fix this?
  - Remember, the dependency is really on the *name* of the register
  - So… change the register names!

Register Renaming Concept

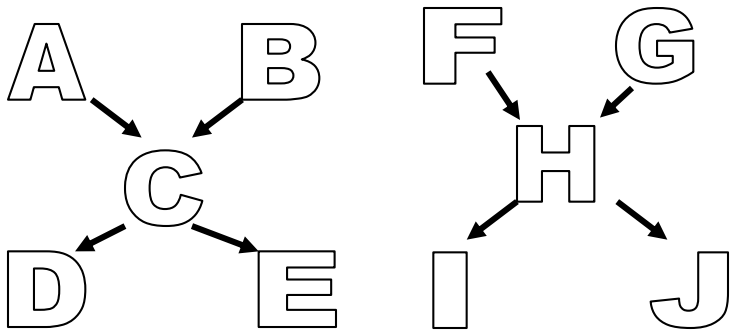
- The register names are arbitrary
- The register name only needs to be consistent between writes.

R1= …..
…. = R1 ….
….= … R1
R1= ….

The value in R1 is “alive” from when the value is written until the last read of that value.

So after renaming, what happens to the dependencies?

```
P1=MEM[R3+4]     //A
P2=MEM[R3+8]     //B
P3=P1*P2         //C
MEM[R3+4]=P3     //D
MEM[R3+8]=P3     //E
P4=MEM[R3+12]    //F
P5=MEM[R3+16]    //G
P6=P4*P5         //H
MEM[R3+12]=P6    //I
MEM[R3+16]=P6    //J
```



RAW WAW WAR

Register Renaming Approach

- Every time an architected register is written we assign it to a physical register
  - Until the architected register is written again, we continue to translate it to the physical register number
  - Leaves **RAW** dependencies intact
- It is really simple, let's look at an example:
  - Names: **`r1,r2,r3`**
  - Locations: **`p1,p2,p3,p4,p5,p6,p7`**
  - Original mapping: **`r1`**→**`p1`**, **`r2`**→**`p2`**, **`r3`**→**`p3`**, **`p4`**–**`p7`** are "free"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

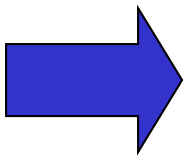
Orig. insns

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

Renamed insns

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

```
R1=MEM[P7+4]      // A        P1=MEM[R3+4]
R2=MEM[R3+8]      // B        P2=MEM[R3+8]
R1=R1*R2          // C        P3=P1*P2
MEM[R3+4]=R1      // D        MEM[R3+4]=P3
MEM[R3+8]=R1      // E        MEM[R3+8]=P3
R1=MEM[R3+12]     // F        P4=MEM[R3+12]
R2=MEM[R3+16]     // G        P5=MEM[R3+16]
R1=R1*R2          // H        P6=P4*P5
MEM[R3+12]=R1     // I        MEM[R3+12]=P6
MEM[R3+16]=R1     // J        MEM[R3+16]=P6
```



Arch	V?	Physical
1	1	
2	1	
3	1	

Tomasulo's Scheduling Algorithm

Tomasulo's Scheduling Algorithm

- **Tomasulo's algorithm**
  - **Reservation stations (RS)**: instruction buffer
  - **Common data bus (CDB)**: broadcasts results to RS
  - Register renaming: removes WAR/WAW hazards

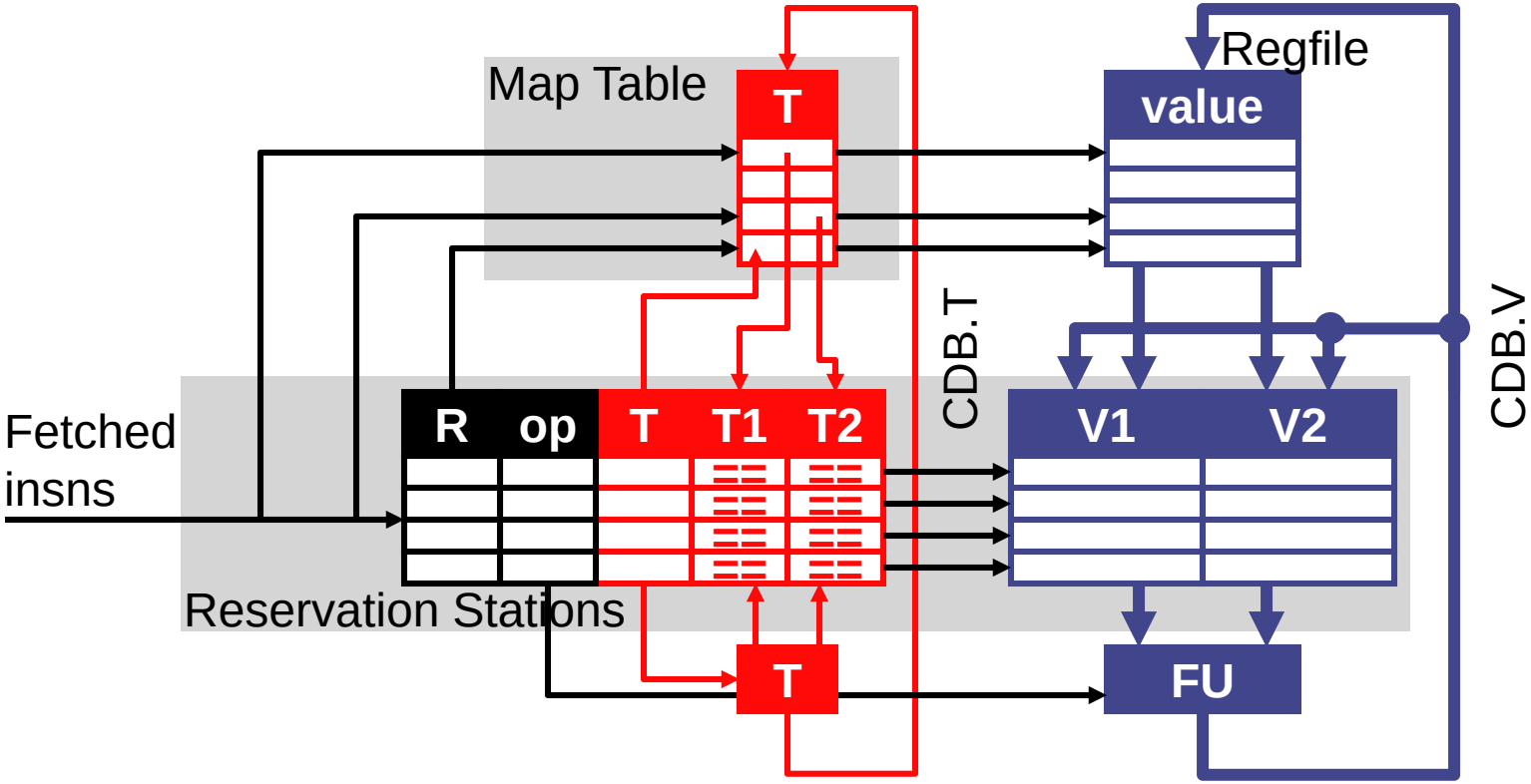
- First implementation: IBM 360/91 [1967]
  - Dynamic scheduling for FP units only
  - Bypassing

- Our example: "Simple Tomasulo"
  - Dynamic scheduling for everything, including load/store
  - No bypassing
  - 5 RS: 1 ALU, 1 load, 1 store, 2 FP (3-cycle, pipelined)

Tomasulo Data Structures

- Reservation Stations (RS#)
  - **FU**, **busy**, **op**, **R**: destination register name
  - **T**: destination register tag (RS# of this RS)
  - **T1**,**T2**: source register tags (RS# of RS that will produce value)
  - **V1**,**V2**: source register values
- Rename Table/Map Table/RAT
  - **T**: tag (RS#) that will write this register
- Common Data Bus (CDB)
  - Broadcasts <RS#, value> of completed insns
- Tags interpreted as ready-bits++
  - T==0 → Value is ready somewhere
  - T!=0 → Value is not ready, wait until CDB broadcasts T

Simple Tomasulo Data Structures

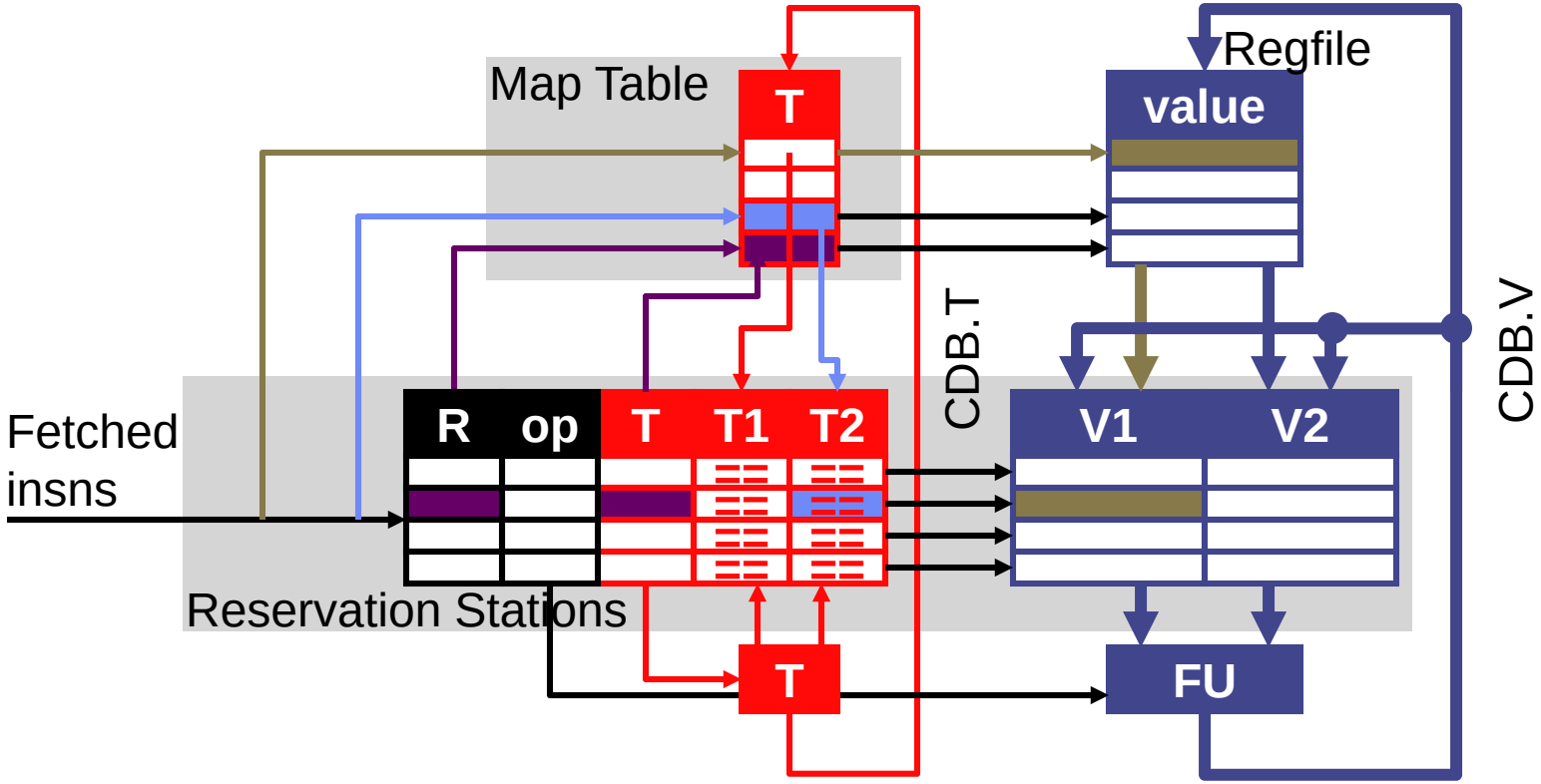


- Insn fields and status bits
- Tags
- Values

Simple Tomasulo Pipeline

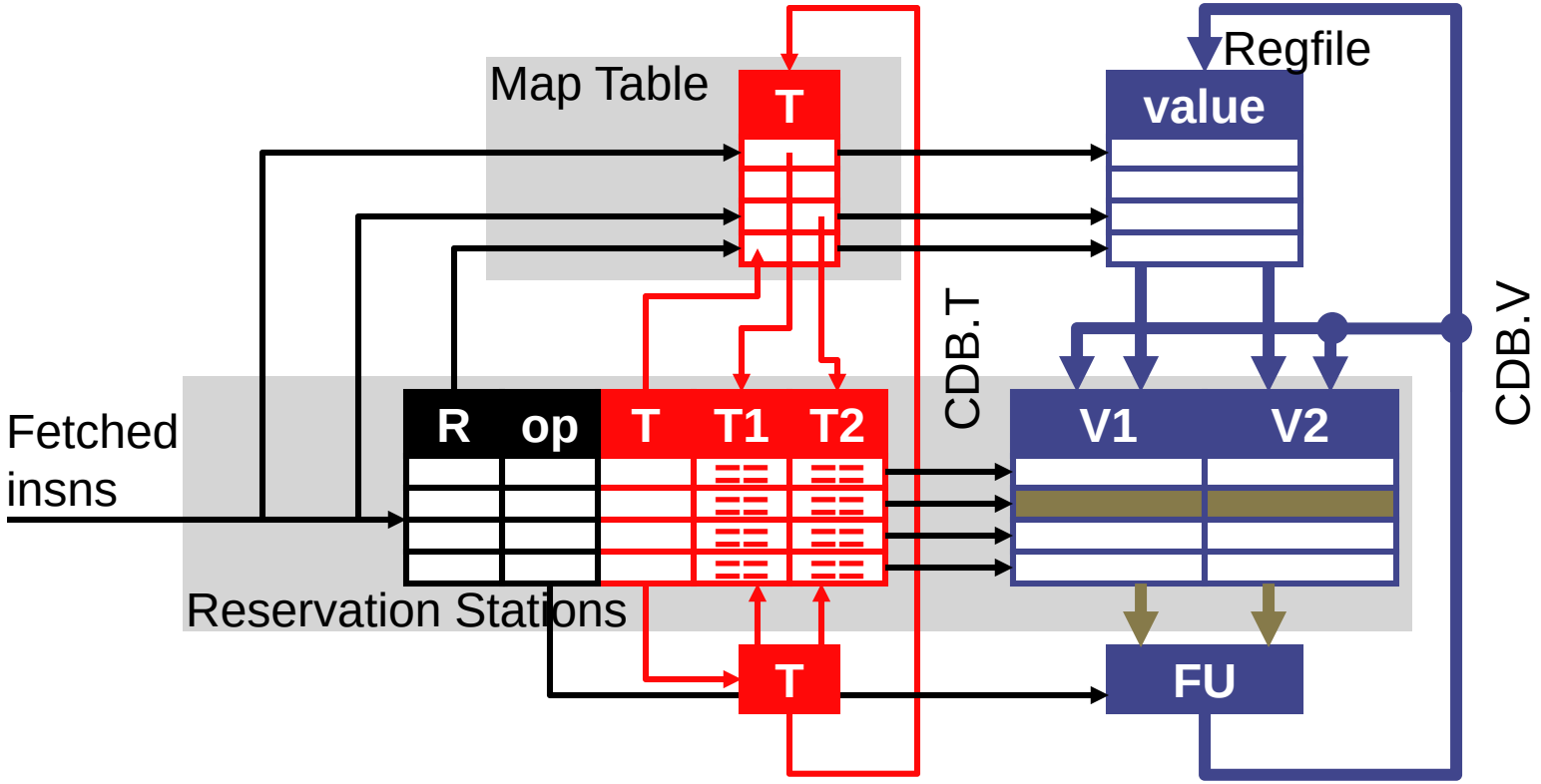
- New pipeline structure: F, **D**, **S**, X, **W**
  - **D (dispatch)**
    - **Structural** hazard ? **stall** : allocate RS entry
  - **S (issue)**
    - **RAW** hazard ? **wait** (monitor CDB) : go to execute
  - **W (writeback)**
    - **W**rite register (sometimes…), free RS entry
    - W and RAW-dependent S in same cycle
    - W and structural-dependent D in same cycle

Tomasulo Dispatch (D)



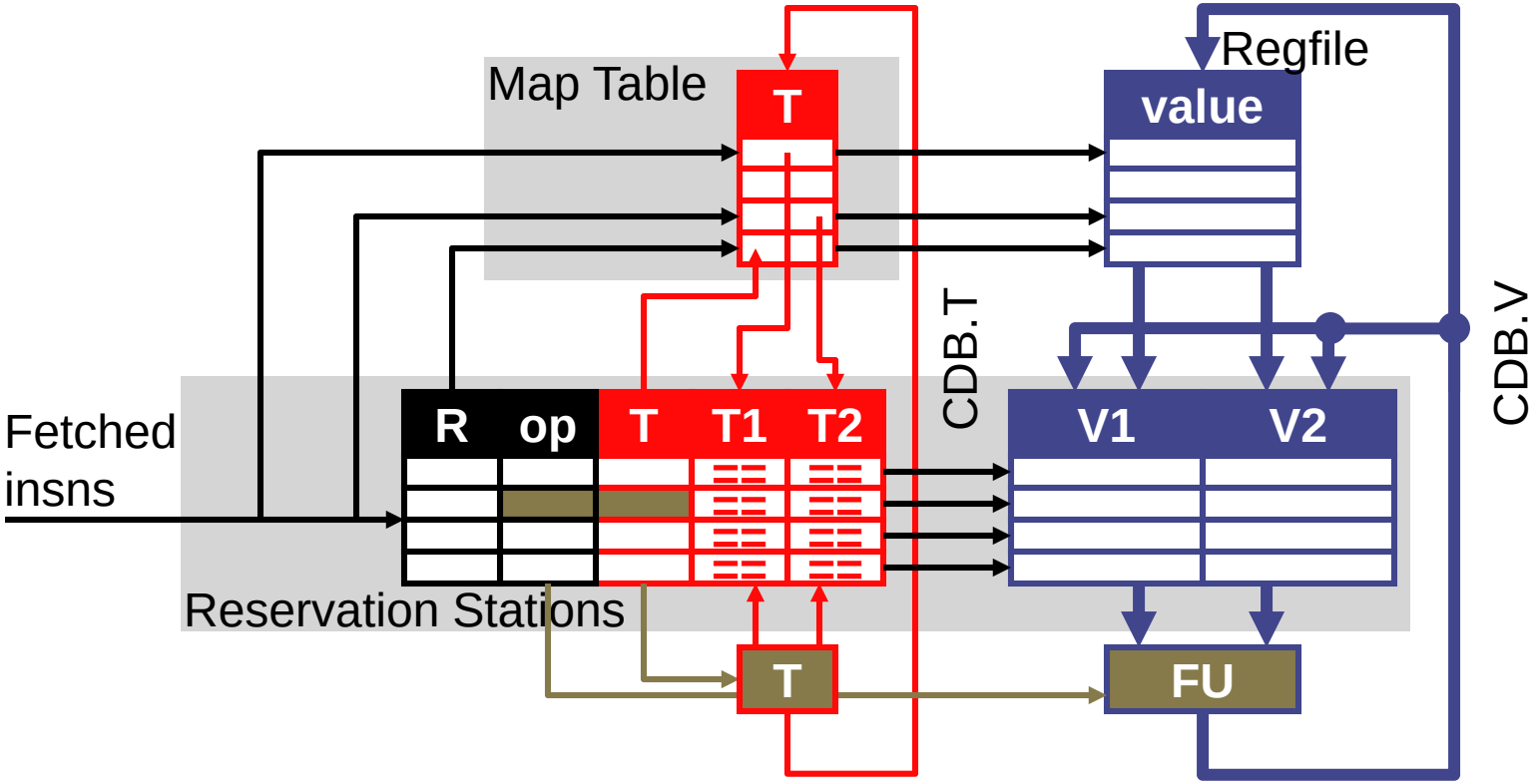
- Stall for structural (RS) hazards
  - Allocate RS entry
  - Input register ready ? read value into RS : read tag into RS
  - Rename output register to RS # (represents a unique value "name")

Tomasulo Issue (S)

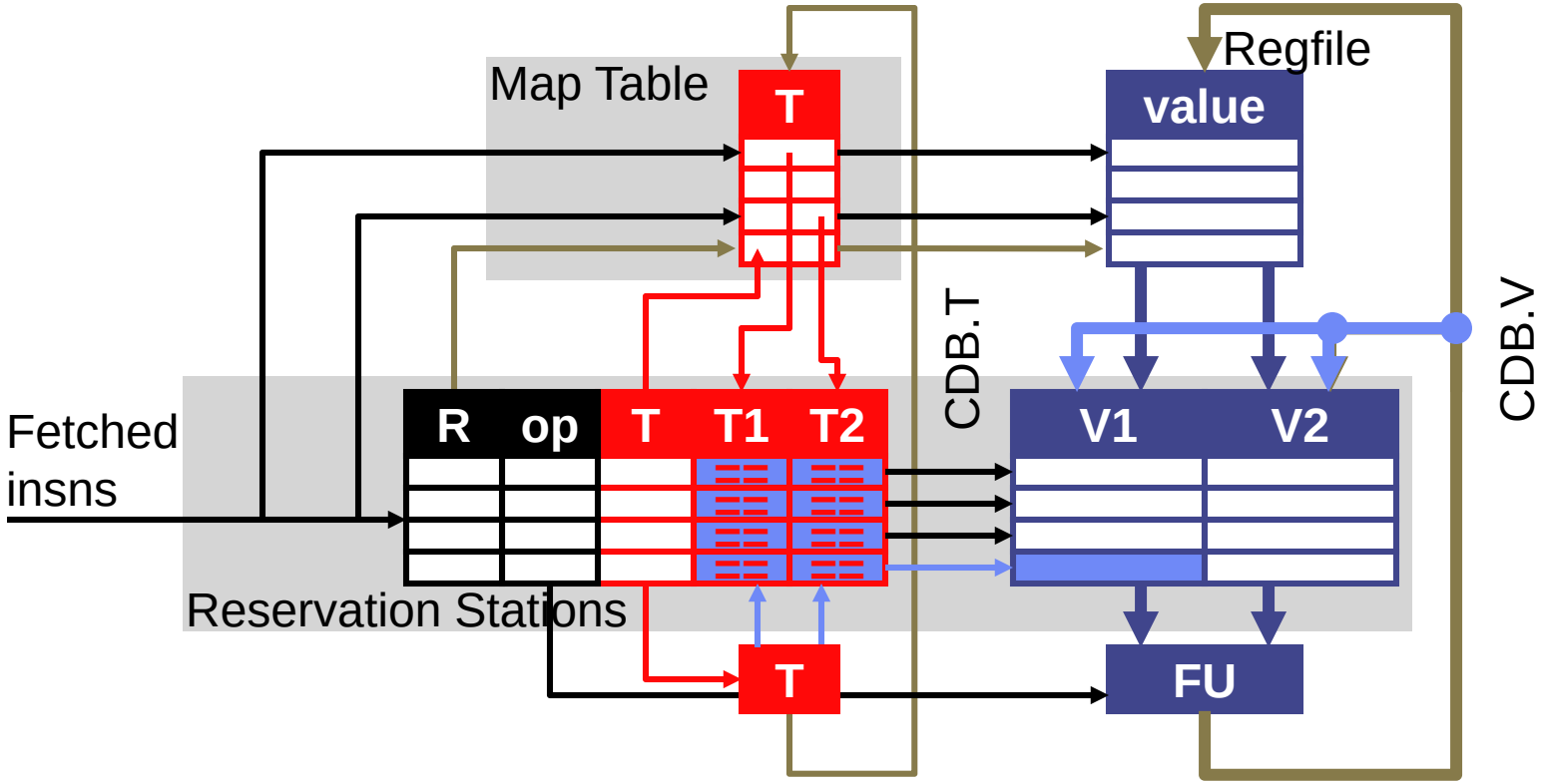


- Wait for RAW hazards
  - Read register values from RS

Tomasulo Execute (X)

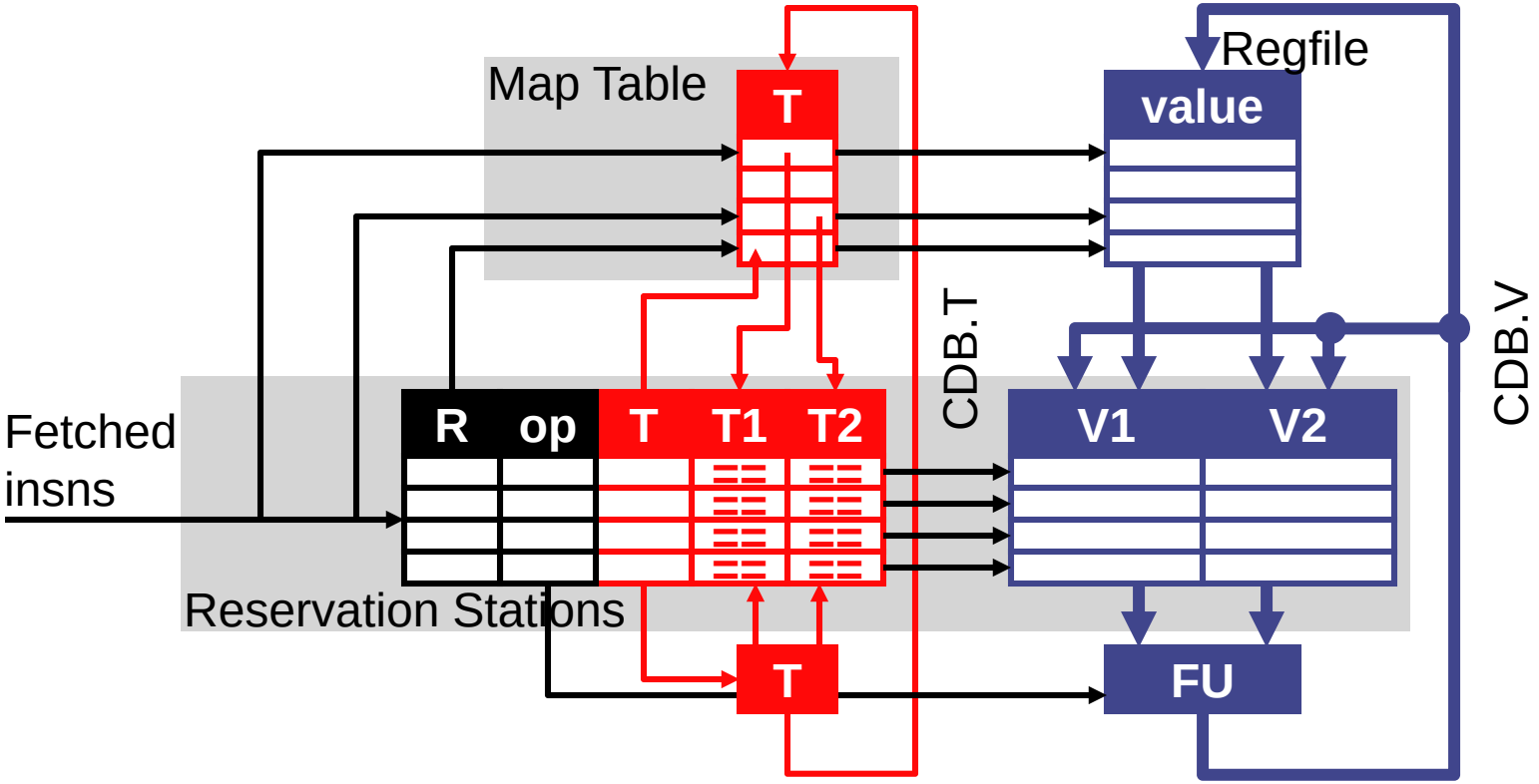


Tomasulo Writeback (W)



- Wait for structural (CDB) hazards
  - if Map Table rename still matches ? Clear mapping, write result to regfile
  - CDB broadcast to RS: tag match ? clear tag, copy value
  - Free RS entry

Register Renaming for Tomasulo

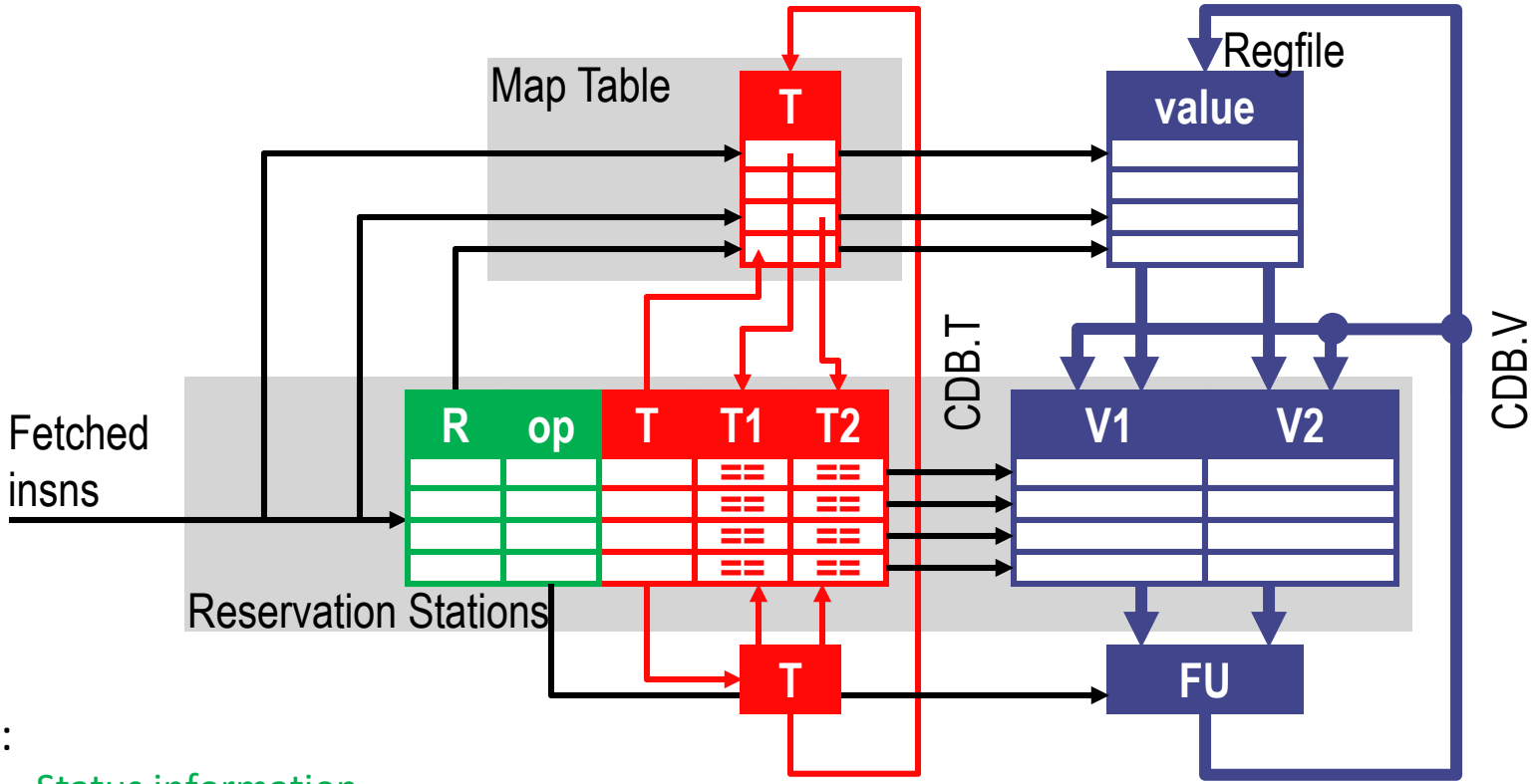


- What in Tomasulo implements **register renaming**?
  - **Value copies in RS (V1, V2)**
  - Insn stores correct input values in its own RS entry
  - \+ Future insns can overwrite master copy in regfile, doesn't matter

Value/Copy-Based Register Renaming

- Tomasulo-style register renaming
  - Called "**value-based**" or "**copy-based**"
  - **Names**: architectural registers
  - **Storage locations**: register file and reservation stations
    - Values can and do exist in both
    - Register file holds master (i.e., most recent) values
    - \+ RS copies eliminate WAR hazards
  - Storage locations referred to internally by RS# tags
    - Register table translates names to tags
    - Tag == 0 value is in register file
    - Tag != 0 value is not ready and is being computed by RS#
  - CDB broadcasts values with tags attached
    - So insns know what value they are looking at

Simple Tomasulo Data Structures



- RS:
  - Status information
    - R: Destination Register
    - op: Operand (add, etc.)
  - Tags
    - T1, T2: source operand tags
  - Values
    - V1, V2: source operand values

- Map table (also RAT: Register Alias Table)
  - Maps registers to tags
- Regfile (also ARF: Architected Register File)
  - Holds value of register if no value in RS

Tomasulo Data Structures
(Timing Free Example)

CDB	
T	V

Map Table	
Reg	T
r0	
r1	
r2	
r3	
r4	

Reservation Stations								
T	FU	busy	R	op	T1	T2	V1	V2
1								
2								
3								
4								
5								

ARF	
Reg	V
r0	
r1	
r2	
r3	
r4	

Instruction
r0=r1*r2
r1=r2*r3
r2=r4+1
r1=r1+r1

Tomasulo Data Structures
(Timing Free Example #2)

CDB	
T	V

Map Table	
Reg	Tag
r0	
r1	
r2	
r3	
r4	

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1								
2								
3								
4								
5								

ARF	
Reg	V
r0	
r1	
r2	
r3	
r4	

Instruction
r0=r1*r2
r1=r0*r3
r0=r4+1
r1=r1+r0

Questions

- Where can we get values for a given instruction from?
  - A)
  - B)
- When do we update the ARF? (This is a bit tricky)
  - How do we know there isn't anyone else who needs the value we overwrite?
- What do we do on a branch mispredict?

Example with timing

- This set of slides is here for you to look over outside of class.
- I generally prefer to not worry about timing issues too much at this point.
  - They are implementation-specific and get more involved than I think is useful.
  - That said, a number of students get the general case better if they have a specific case to look at.

Example: Tomasulo with timing

Insn Status				
Insn	D	S	X	W
`ldf X(r1),f1`				
`mulf f0,f1,f2`				
`stf f2,Z(r1)`				
`addi r1,4,r1`				
`ldf X(r1),f1`				
`mulf f0,f1,f2`				
`stf f2,Z(r1)`				

Map Table	
Reg	T
f0	
f1	
f2	
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	no						
4	FP1	no						
5	FP2	no						

Tomasulo: Cycle 1

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1			
mulf f0,f1,f2				
stf f2,Z(r1)				
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	no						
5	FP2	no						

allocate

Tomasulo: Cycle 2

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2		
mulf f0,f1,f2	c2			
stf f2,Z(r1)				
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo: Cycle 3

Insn Status				
Insn	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	
`mulf f0,f1,f2`	c2			
`stf f2,Z(r1)`	c3			
`addi r1,4,r1`				
`ldf X(r1),f1`				
`mulf f0,f1,f2`				
`stf f2,Z(r1)`				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo: Cycle 4

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4		
stf f2,Z(r1)	c3			
addi r1,4,r1	c4			
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V
RS#2	[f1]

ldf finished (W)
clear f1 RegStatus
CDB broadcast

Reservation Stations									
T	FU	busy	op	R	T1	T2		V1	V2
1	ALU	yes	addi	r1	-	-		[r1]	-
2	LD	no							
3	ST	yes	stf	-	RS#4	-		-	[r1]
4	FP1	yes	mulf	f2	-	RS#2		[f0]	CDB.V
5	FP2	no							

allocate

free

RS#2 ready → grab CDB value

Tomasulo: Cycle 5

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5		
ldf X(r1),f1	c5			
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	no						

allocate

Tomasulo: Cycle 6

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5	c6	
ldf X(r1),f1	c5			
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	
f2	RS#4RS#5
r1	RS#1

CDB	
T	V

no D stall on WAW: scoreboard would overwrite f2 RegStatus
anyone who needs old f2 tag has it

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

allocate

Tomasulo: Cycle 7

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7		
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	RS#1

CDB	
T	V
RS#1	[r1]

no W wait on WAR: scoreboard would anyone who needs old r1 has RS copy

D stall on store RS: structural

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	RS#1	-	CDB.V
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

addi finished (W)
clear r1 RegStatus
CDB broadcast

RS#1 ready → grab CDB value

Tomasulo: Cycle 8

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8		
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#4	[f2]

`mulf` finished (W)
- don't clear `f2` RegStatus
  - already overwritten by 2nd `mulf` (`RS#5`)
- CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	CDB.V	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

`RS#4` ready → grab CDB value

Tomasulo: Cycle 9

Insn Status				
Insn	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8
`stf f2,Z(r1)`	c3	c8	c9	
`addi r1,4,r1`	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c7	c8	c9
`mulf f0,f1,f2`	c6	c9		
`stf f2,Z(r1)`				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#2	[f1]

2nd `ldf` finished (W)
clear `f1` RegStatus
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	-	-	[f2]	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	CDB.V

RS#2 ready → grab CDB value

Tomasulo: Cycle 10

Insn Status				
Insn	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8
`stf f2,Z(r1)`	c3	c8	c9	c10
`addi r1,4,r1`	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c7	c8	c9
`mulf f0,f1,f2`	c6	c9	c10	
`stf f2,Z(r1)`	c10			

Map Table	
Reg	T
f0	
f1	
f2	RS#5
r1	

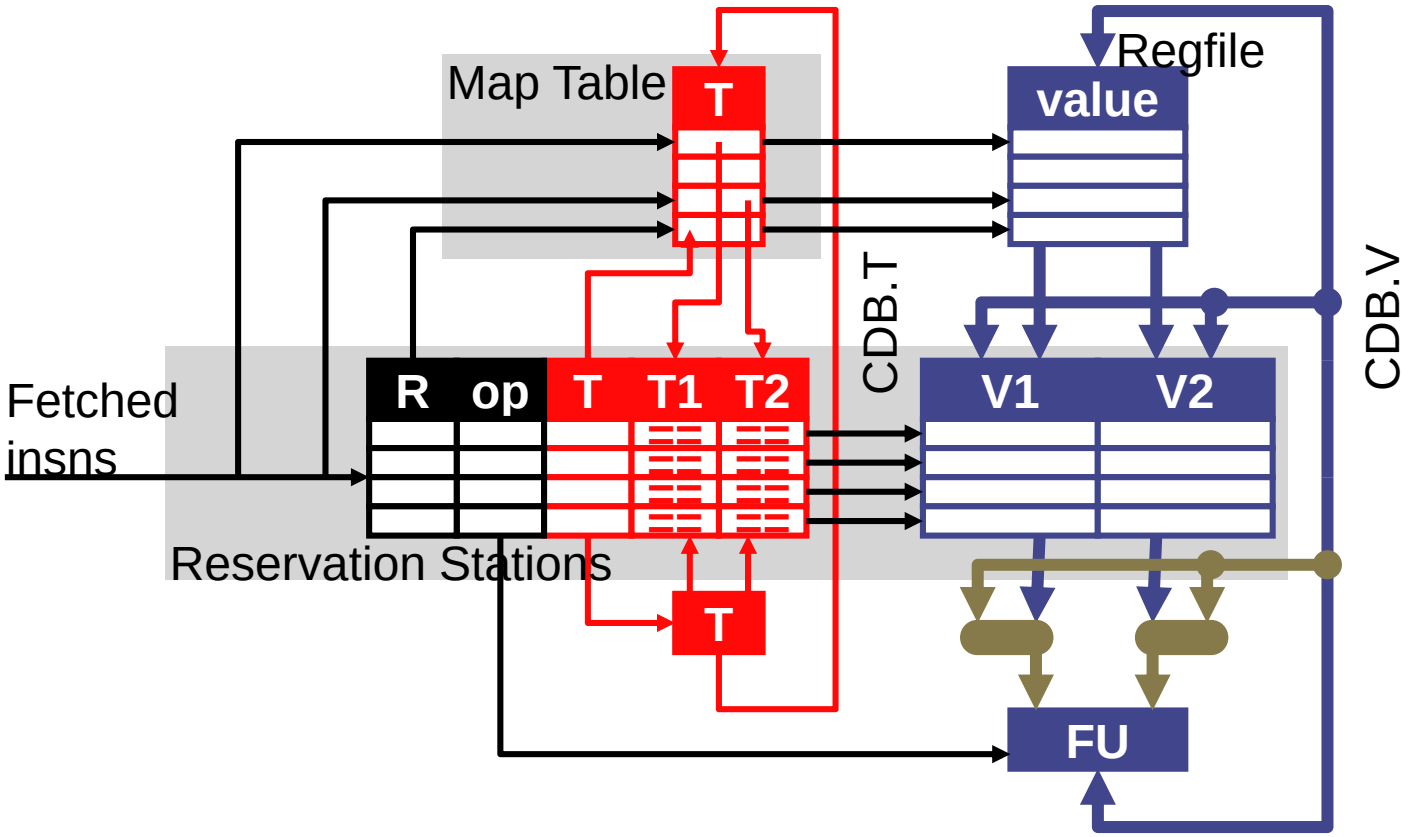
CDB	
T	V

`stf` finished (W)
no output register → no CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	RS#5	-	-	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	-	[f0]	[f1]

free → allocate

Can We Add Bypassing?



- Yes, but it's more complicated than you might think
  - Scheduler must work in advance of computation
  - Requires knowledge of the latency of instructions, not always possible
  - Accurate bypass is a key advancement in scheduling in last 10 years

Can We Add Superscalar?

- Dynamic scheduling and multiple issue are orthogonal
  - E.g., Pentium4: dynamically scheduled 5-way superscalar
  - Two dimensions
    - **N**: superscalar width (number of parallel operations)
    - **W**: window size (number of reservation stations)

- What do we need for an **N**-by-**W** Tomasulo?
  - RS: **N** tag/value w-ports (D), **N** value r-ports (S), **2N** tag CAMs (W)
  - Select logic: **W**→**N** priority encoder (S)
  - MT: **2N** r-ports (D), **N** w-ports (D)
  - RF: **2N** r-ports (D), **N** w-ports (W)
  - CDB: **N** (W)
  - Which are the expensive pieces?

Superscalar Select Logic

- Superscalar select logic: W→N priority encoder
  - Somewhat complicated (N^2 logW)
  - Can simplify using different RS designs
- **Split design**
  - Divide RS into N banks: 1 per FU?
  - Implement N separate W/N→1 encoders
  - \+ Simpler: N * logW/N
  - – Less scheduling flexibility
- **FIFO design** [Palacharla+]
  - Can issue only head of each RS bank
  - \+ Simpler: no select logic at all
  - – Less scheduling flexibility (but surprisingly not that bad)

Dynamic Scheduling Summary

- Dynamic scheduling: out-of-order execution
  - Higher pipeline/FU utilization, improved performance
  - Easier and more effective in hardware than software
    - + More storage locations than architectural registers
    - + Dynamic handling of cache misses
- Instruction buffer: multiple F/D latches
  - Implements large scheduling scope + "passing" functionality
  - Split decode into in-order dispatch and out-of-order issue
    - Stall vs. wait
- Dynamic scheduling algorithms
  - Scoreboard: no register renaming, limited out-of-order
  - Tomasulo: copy-based register renaming, full out-of-order

Are we done?

- When can Tomasulo go wrong?
  - Lack of instructions to choose from!!
    - Need a really really really good branch predictor
  - Exceptions!!
    - No way to figure out relative order of instructions in RS

And… a bit of terminology

- Issue can be thought of as a two-stage process: "wakeup" and "select".
  - When the RS figures out it has it's data and is ready to run it is said to have "woken up" and the process of doing so is called ***wakeup***
    - But there may be a structural hazard—no EX unit available for a given RS
      - When?
  - Thus, in addition to be woken up, and RS needs to be selected before it can go to the execute unit (EX stage).
    - This process is called ***select***