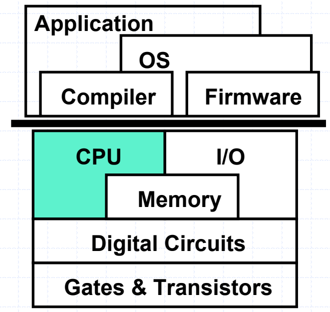
CIS 501
Introduction to Computer Architecture

Unit 8: Dynamic Scheduling I

This Unit: Dynamic Scheduling I



- Dynamic scheduling
  - Out-of-order execution
- Scoreboard
  - Dynamic scheduling with WAW/WAR
- Tomasulo's algorithm
  - Add register renaming to fix WAW/WAR

- Next unit
  - Adding speculation and precise state
  - Dynamic load scheduling

Readings

- H+P
  - None (not happy with explanation of this topic)

The Problem With In-Order Pipelines

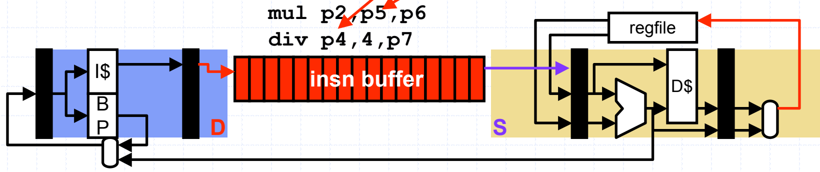
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`addf f0,f1,f2`	F	D	E+	E+	E+	W										
`mulf f2,f3,f2`		F	D	d*	d*	E*	E*	E*	E*	E*	W					
`subf f0,f1,f4`			F	p*	p*	D	E+	E+	E+	W						

- What's happening in cycle 4?
  - `mulf` stalls due to **RAW hazard**
    - OK, this is a fundamental problem
  - `subf` stalls due to **pipeline hazard**
    - Why? `subf` can't proceed into D because `addf` is there
    - That is the only reason, and it isn't a fundamental one

- Why can't `subf` go into D in cycle 4 and E+ in cycle 5?

Dynamic Scheduling: The Big Picture

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



Ready Table

P2	P3	P4	P5	P6	P7	
Yes	Yes					add p2,p3,p4
Yes	Yes	Yes			Yes	sub p2,p4,p5 and div p4,4,p7
Yes	Yes	Yes	Yes		Yes	mul p2,p5,p6
Yes	Yes	Yes	Yes	Yes	Yes	

- Instructions fetch/decoded/renamed into *Instruction Buffer*
  - Also called "instruction window" or "instruction scheduler"
- Instructions (conceptually) check ready bits every cycle
  - Execute when ready

Register Renaming

- To eliminate WAW and WAR hazards
- 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	FreeList	Raw insns	Renamed insns
p1	p2	p3	p4,p5,p6,p7	add r2,r3,r1	add p2,p3,p4
p4	p2	p3	p5,p6,p7	sub r2,r1,r3	sub p2,p4,p5
p4	p2	p5	p6,p7	mul r2,r3,r3	mul p2,p5,p6
p4	p2	p6	p7	div r1,4,r1	div p4,4,p7

- Renaming
  - + Removes **WAW** and **WAR** dependences
  - + Leaves **RAW** intact!

Dynamic Scheduling - OoO Execution

- Dynamic scheduling
  - Totally in the hardware
  - Also called "out-of-order execution" (OoO)
- Fetch many instructions into instruction window
  - Use branch prediction to speculate past (multiple) branches
  - Flush pipeline on branch misprediction
- Rename to avoid false dependencies (WAW and WAR)
- Execute instructions as soon as possible
  - Register dependencies are known
  - Handling memory dependencies more tricky (much more later)
- Commit instructions in order
  - Any strange happens before commit, just flush the pipeline
- Current machines: 100+ instruction scheduling window

Static Instruction Scheduling

- **Issue**: time at which insns execute
- **Schedule**: order in which insns execute
  - Related to issue, but the distinction is important
- **Scheduling**: re-arranging insns to enable rapid issue
  - Static: by compiler
  - Requires knowledge of pipeline and program dependences
    - **Pipeline scheduling**: the basics
  - Requires large scheduling scope full of independent insns
    - **Loop unrolling**, software pipelining: increase scope for loops
    - **Trace scheduling**: increase scope for non-loops

Anything software can do … hardware can do better

Motivation Dynamic Scheduling

- **Dynamic scheduling (out-of-order execution)**
  - Execute insns in non-sequential (non-VonNeumann) order…
    + Reduce RAW stalls
    + Increase pipeline and functional unit (FU) utilization
      - Original motivation was to increase FP unit utilization
    + Expose more opportunities for parallel issue (ILP)
      - Not in-order → can be in parallel
  - …but make it appear like sequential execution
    - Important
    – But difficult
    - Next unit

Before We Continue

- If we can do this in software…
- …why build complex (slow-clock, high-power) hardware?
  + Performance portability
    - Don't want to recompile for new machines
  + More information available
    - Memory addresses, branch directions, cache misses
  + More registers available (??)
    - Compiler may not have enough to fix WAR/WAW hazards
  + Easier to speculate and recover from mis-speculation
    - Flush instead of recover code
  – But compiler has a larger scope
    - Compiler does as much as it can (not much)
    - Hardware does the rest

Going Forward: What's Next

- We'll build this up in steps over the next few weeks
  - "Scoreboarding" - first OoO, no register renaming
  - "Tomasulo's algorithm" - adds register renaming
  - Handling precise state and speculation
    - P6-style execution (Intel Pentium Pro)
    - R10k-style execution (MIPS R10k)
  - Handling memory dependencies
    - Conservative and speculative

- Let's get started!

Dynamic Scheduling as Loop Unrolling

- Three steps of loop unrolling
  - Step I: combine iterations
    - Increase scheduling scope for more flexibility
  - Step II: pipeline schedule
    - Reduce impact of RAW hazards
  - Step III: rename registers
    - Remove WAR/WAW violations that result from scheduling

Loop Example: SAX (SAXPY – PY)

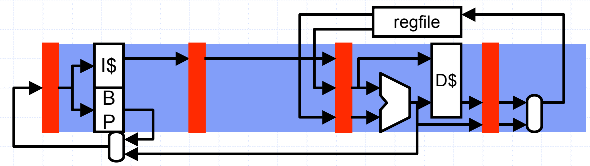
- **SAX** (Single-precision A X)
  - Only because there won't be room in the diagrams for SAXPY

```
for (i=0;i<N;i++)
  Z[i]=A*X[i];

0: ldf X(r1),f1       // loop
1: mulf f0,f1,f2      // A in f0
2: stf f4,Z(r1)
3: addi r1,4,r1       // i in r1
4: blt r1,r2,0        // N*4 in r2
```

- Consider two iterations, ignore branch
  ldf, mulf, stf, addi, ldf, mulf, stf

New Pipeline Terminology



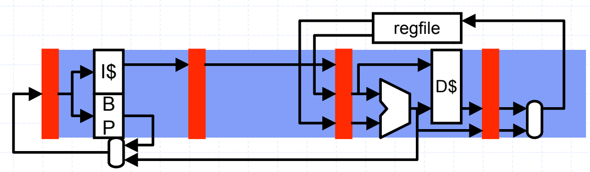
- **In-order pipeline**
  - Often written as F,D,X,W (multi-cycle X includes M)
  - Example pipeline: 1-cycle int (including mem), 3-cycle pipelined FP

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

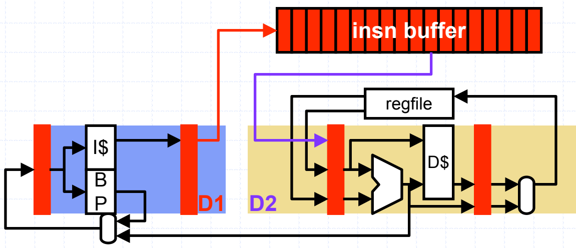
- Alternative pipeline diagram
  - Down: insns
  - Across: pipeline stages
  - In boxes: cycles
  - Basically: stages ↔ cycles
  - Convenient for out-of-order

The Problem With In-Order Pipelines



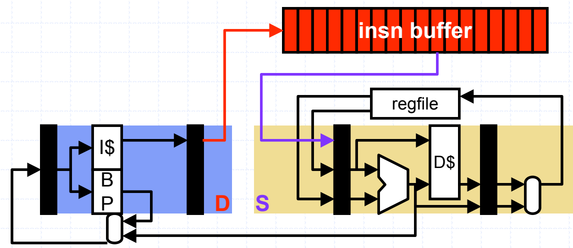
- **In-order pipeline**
  - **Structural hazard**: 1 insn register (latch) per stage
    - 1 insn per stage per cycle (unless pipeline is replicated)
    - Younger insn can't "pass" older insn without "clobbering" it
- **Out-of-order pipeline**
  - Implement "passing" functionality by removing structural hazard

Instruction Buffer



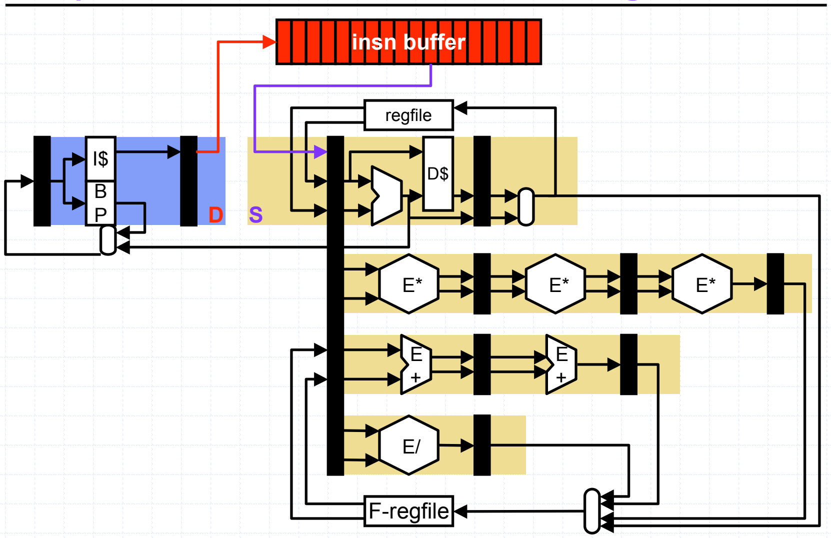
- Trick: **insn buffer** (many names for this buffer)
  - Basically: a bunch of latches for holding insns
  - Implements iteration fusing … here is your scheduling scope
- Split D into two pieces
  - Accumulate decoded insns in buffer **in-order**
  - Buffer sends insns down rest of pipeline **out-of-order**

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

- Three parts to loop unrolling
  - Scheduling scope: insn buffer
  - Pipeline scheduling and register renaming: **scheduling algorithm**
- Look at two register scheduling algorithms
  - **Register scheduler**: scheduler based on register dependences
  - Scoreboard
    - No register renaming → limited scheduling flexibility
  - Tomasulo
    - Register renaming → more flexibility, better performance
- Big simplification in this unit: **memory scheduling**
  - Pretend register algorithm magically knows memory dependences
  - A little more realism next unit

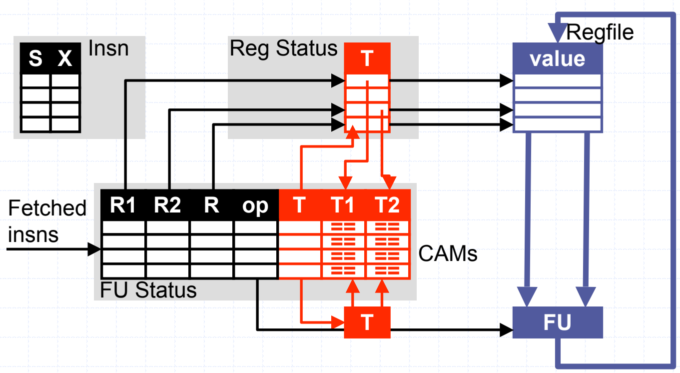
Scheduling Algorithm I: Scoreboard

- **Scoreboard**
  - Centralized control scheme: insn status explicitly tracked
  - Insn buffer: **Functional Unit Status Table (FUST)**
- First implementation: CDC 6600 [1964]
  - 16 separate non-pipelined functional units (7 int, 4 FP, 5 mem)
  - No bypassing
- Our example: "Simple Scoreboard"
  - 5 FU: 1 ALU, 1 load, 1 store, 2 FP (3-cycle, pipelined)

Scoreboard Data Structures

- FU Status Table
  - **FU**, **busy**, **op**, **R**, **R1**, **R2**: destination/source register names
  - **T**: destination register tag (FU producing the value)
  - **T1**,**T2**: source register tags (FU producing the values)
- Register Status Table
  - **T**: tag (FU that will write this register)
- Tags interpreted as ready-bits
  - Tag == 0 → Value is ready in register file
  - Tag != 0 → Value is not ready, will be supplied by T
- Insn status table
  - S,X bits for all active insns

Simple Scoreboard Data Structures

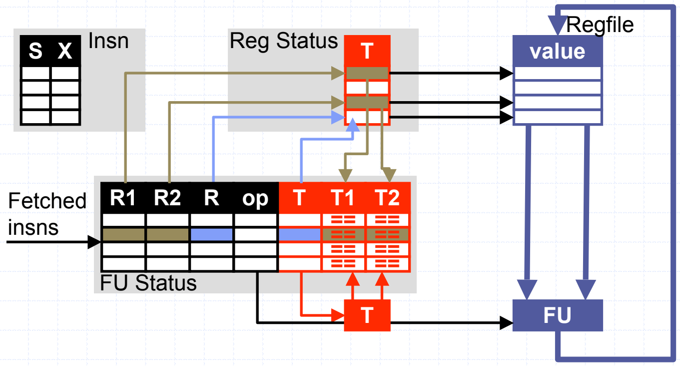


- Insn fields and status bits
- Tags
- Values

Scoreboard Pipeline

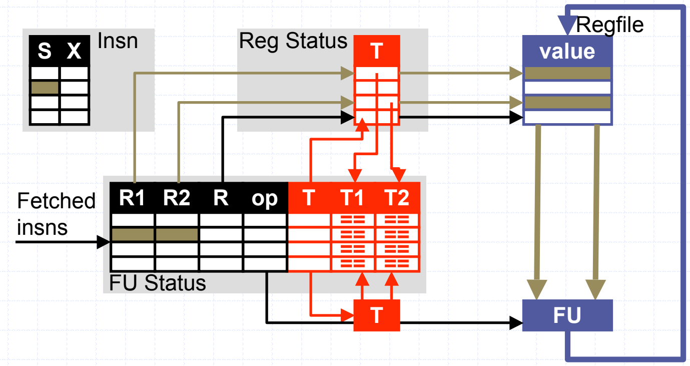
- New pipeline structure: F, **D**, **S**, X, **W**
  - F (fetch)
    - Same as it ever was
  - **D (dispatch)**
    - **Structural** or **WAW** hazard ? **stall** : allocate scoreboard entry
  - **S (issue)**
    - **RAW** hazard ? **wait** : read registers, go to execute
  - X (execute)
    - Execute operation, notify scoreboard when done
  - **W (writeback)**
    - **WAR** hazard ? **wait** : write register, free scoreboard entry
    - W and RAW-dependent S in same cycle
    - W and structural-dependent D in same cycle

Scoreboard Dispatch (D)



- Stall for WAW or structural (Scoreboard, FU) hazards
  - Allocate scoreboard entry
  - Copy Reg Status for input registers
  - Set Reg Status for output register

Scoreboard Issue (S)

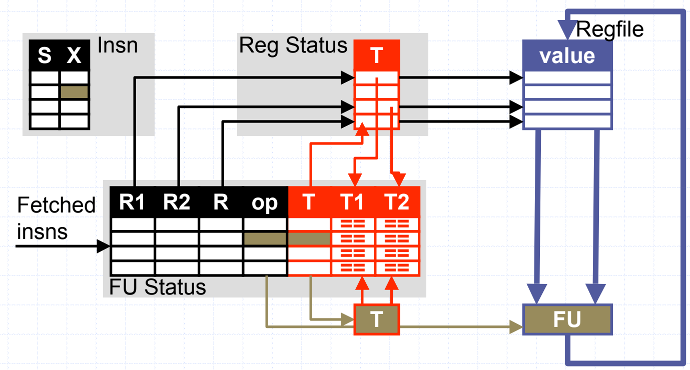


- Wait for RAW register hazards
  - Read registers

Issue Policy and Issue Logic

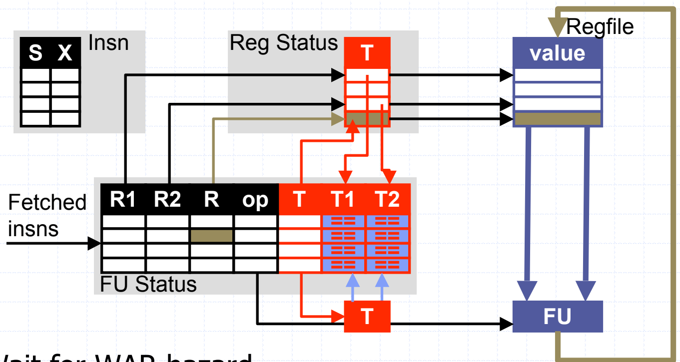
- Issue
  - If multiple insns ready, which one to choose? **Issue policy**
    - Oldest first? Safe
    - Longest latency first? May yield better performance
  - **Select logic**: implements issue policy
    - W→1 priority encoder
    - **W**: window size (number of scoreboard entries)

Scoreboard Execute (X)



- Execute insn

Scoreboard Writeback (W)



- Wait for WAR hazard
  - Write value into regfile, clear Reg Status entry
  - Compare tag to waiting insns input tags, match ? clear input tag
  - Free scoreboard entry

Scoreboard Data Structures

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)				

Reg Status

Reg	T
f0	
f1	
f2	
r1	

FU Status

FU	busy	op	R	R1	R2	T1	T2
ALU	no						
LD	no						
ST	no						
FP1	no						
FP2	no						

Scoreboard: 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)				

Reg Status

Reg	T
f0	
f1	LD
f2	
r1	

FU Status

FU	busy	op	R	R1	R2	T1	T2	
ALU	no							
LD	yes	ldf	f1	-	r1	-	-	allocate
ST	no							
FP1	no							
FP2	no							

Scoreboard: 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)				

Reg Status

Reg	T
f0	
f1	LD
f2	FP1
r1	

FU Status

FU	busy	op	R	R1	R2	T1	T2	
ALU	no							
LD	yes	ldf	f1	-	r1	-	-	
ST	no							
FP1	yes	mulf	f2	f0	f1	-	LD	allocate
FP2	no							

Scoreboard: 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)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP1
r1	

Functional unit status							
FU	busy	op	R	R1	R2	T1	T2
ALU	no						
LD	yes	ldf	f1	-	r1	-	-
ST	yes	stf	-	f2	r1	FP1	-
FP1	yes	mulf	f2	f0	f1	-	LD
FP2	no						

allocate (ST)

Scoreboard: 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)				

Reg Status	
Reg	T
f0	
f1	~~LD~~
f2	FP1
r1	ALU

f1 written → clear

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	yes	addi	r1	r1	-	-	-
LD	no						
ST	yes	stf	-	f2	r1	FP1	-
FP1	yes	mulf	f2	f0	f1	-	~~LD~~
FP2	no						

allocate (ALU)

free (LD)

f0 (LD) is ready → issue mulf

Scoreboard: 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)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP1
r1	ALU

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	yes	addi	r1	r1	-	-	-
LD	yes	ldf	f1	-	r1	-	ALU
ST	yes	stf	-	f2	r1	FP1	-
FP1	yes	mulf	f2	f0	f1	-	-
FP2	no						

allocate (LD)

Scoreboard: 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				
stf f2,Z(r1)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP1
r1	ALU

D stall: WAW hazard w/ `mulf (f2)`
How to tell? `RegStatus[f2]` non-empty

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	yes	addi	r1	r1	-	-	-
LD	yes	ldf	f1	-	r1	-	ALU
ST	yes	stf	-	f2	r1	FP1	-
FP1	yes	mulf	f2	f0	f1	-	-
FP2	no						

Scoreboard: 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	
ldf X(r1),f1	c5			
mulf f0,f1,f2				
stf f2,Z(r1)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP1
r1	ALU

W wait: WAR hazard w/ stf (r1)
How to tell? Untagged r1 in FuStatus
Requires CAM

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	yes	addi	r1	r1	–	–	–
LD	yes	ldf	f1	–	r1	–	ALU
ST	yes	stf	–	f2	r1	FP1	–
FP1	yes	mulf	f2	f0	f1	–	–
FP2	no						

Scoreboard: 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	
ldf X(r1),f1	c5			
mulf f0,f1,f2	c8			
stf f2,Z(r1)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP1 FP2
r1	ALU

first mulf done (FP1)

W wait

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	yes	addi	r1	r1	–	–	–
LD	yes	ldf	f1	–	r1	–	ALU
ST	yes	stf	–	f2	r1	FP1	–
FP1	no						
FP2	yes	mulf	f2	f0	f1	–	LD

f1 (FP1) is ready → issue stf
free
allocate

Scoreboard: 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	c9
ldf X(r1),f1	c5	c9		
mulf f0,f1,f2	c8			
stf f2,Z(r1)				

Reg Status	
Reg	T
f0	
f1	LD
f2	FP2
r1	ALU

r1 written → clear

D stall: structural hazard FuStatus[ST]

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	no						
LD	yes	ldf	f1	–	r1	–	ALU
ST	yes	stf	–	f2	r1	–	–
FP1	no						
FP2	yes	mulf	f2	f0	f1	–	LD

free
r1 (ALU) is ready → issue ldf

Scoreboard: 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	c9
ldf X(r1),f1	c5	c9	c10	
mulf f0,f1,f2	c8			
stf f2,Z(r1)	c10			

Reg Status	
Reg	T
f0	
f1	LD
f2	FP2
r1	

W & structural-dependent D in same cycle

FU Status							
FU	busy	op	R	R1	R2	T1	T2
ALU	no						
LD	yes	ldf	f1	–	r1	–	–
ST	yes	stf	–	f2	r1	FP2	–
FP1	no						
FP2	yes	mulf	f2	f0	f1	–	LD

free, then allocate

In-Order vs. Scoreboard

	In-Order			Scoreboard			
Insn	D	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c1	c2	c3	c4
`mulf f0,f1,f2`	c3	c4+	c7	c2	c4	c5+	c8
`stf f2,Z(r1)`	c7	c8	c9	c3	c8	c9	c10
`addi r1,4,r1`	c8	c9	c10	c4	c5	c6	c9
`ldf X(r1),f1`	c10	c11	c12	c5	c9	c10	c11
`mulf f0,f1,f2`	c12	c13+	c16	c8	c11	c12+	c15
`stf f2,Z(r1)`	c16	c17	c18	c10	c15	c16	c17

- Big speedup?
  - – Only 1 cycle advantage for scoreboard
    - Why? **`addi`** WAR hazard
    - Scoreboard issued **`addi`** earlier (**`c8`** → **`c5`**)
    - But WAR hazard delayed W until **`c9`**
    - Delayed issue of second iteration

In-Order vs. Scoreboard II: Cache Miss

	In-Order			Scoreboard			
Insn	D	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2+	c7	c1	c2	c3+	c8
`mulf f0,f1,f2`	c7	c8+	c11	c2	c8	c9+	c12
`stf f2,Z(r1)`	c11	c12	c13	c3	c12	c13	c14
`addi r1,4,r1`	c12	c13	c14	c4	c5	c6	c13
`ldf X(r1),f1`	c14	c15	c16	c5	c13	c14	c15
`mulf f0,f1,f2`	c16	c17+	c20	c6	c15	c16+	c19
`stf f2,Z(r1)`	c20	c21	c22	c7	c19	c20	c21

- Assume
  - 5 cycle cache miss on first **`ldf`**
  - Ignore FUST structural hazards
  - – Little relative advantage
    - **`addi`** **WAR hazard** (**`c7`** → **`c13`**) stalls second iteration

Scoreboard Redux

- The good
  - + Cheap hardware
    - InsnStatus + FuStatus + RegStatus ~ 1 FP unit in area
  - + Pretty good performance
    - 1.7X for FORTRAN (scientific array) programs
- The less good
  - – No bypassing
    - Is this a fundamental problem?
  - – Limited scheduling scope
    - Structural/**WAW** hazards delay dispatch
  - – Slow issue of truly-dependent (RAW) insns
    - **WAR** hazards delay writeback
  - Fix with **hardware register renaming**

Register Renaming

- **Register renaming (in hardware)**
  - Change register names to eliminate WAR/WAW hazards
  - One of most the beautiful things in computer architecture
  - Key: think of registers (**`r1`**,**`f0`**…) as **names**, not **storage locations**
  - + Can have more locations than names
  - + Can have multiple active versions of same name
- How does it work?
  - **Map-table**: maps names to most recent locations
    - SRAM indexed by name
  - On a write: allocate new location, note in map-table
  - On a read: find location of most recent write via map-table lookup
  - Small detail: must de-allocate locations at some point

Register Renaming Example

- Parameters
  - 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			FreeList	Raw insns	Renamed insns
r1	r2	r3			
p1	p2	p3	p4,p5,p6,p7	add r2,r3,r1	add p2,p3,p4
p4	p2	p3	p5,p6,p7	sub r2,r1,r3	sub p2,p4,p5
p4	p2	p5	p6,p7	mul r2,r3,r3	mul p2,p5,p6
p4	p2	p6	p7	div r1,4,r1	div p4,4,p7

- Renaming
  - + Removes **WAW** and **WAR** dependences
  - + Leaves **RAW** intact!

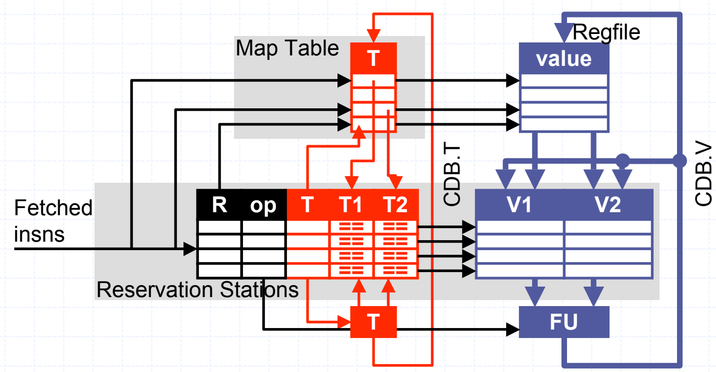
Scheduling Algorithm II: Tomasulo

- **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 (for comparison with Scoreboard)
  - 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
    - That's new
- Map Table
  - **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

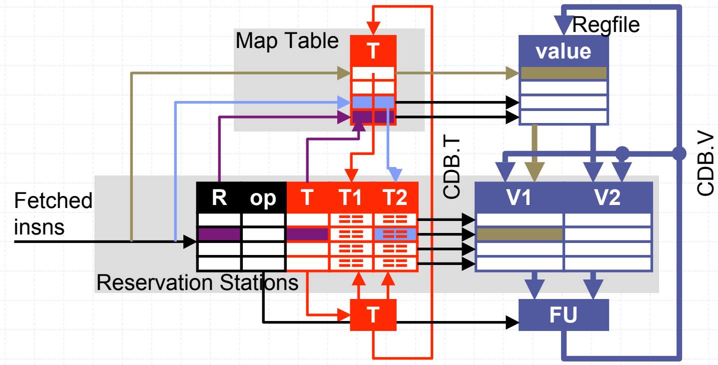


- Insn fields and status bits
- Tags
- Values

Simple Tomasulo Pipeline

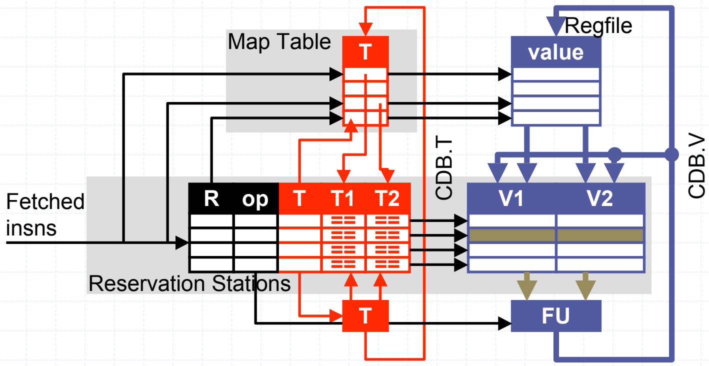
- New pipeline structure: F, **D**, **S**, X, **W**
  - **D (dispatch)**
    - **Structural** hazard ? **stall** : allocate RS entry
  - **S (issue)**
    - **RAW** hazard ? **wait** (monitor CDB) : go to execute
  - **W (writeback)**
    - Write register, free RS entry
    - W and RAW-dependent S in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

Tomasulo Dispatch (D)



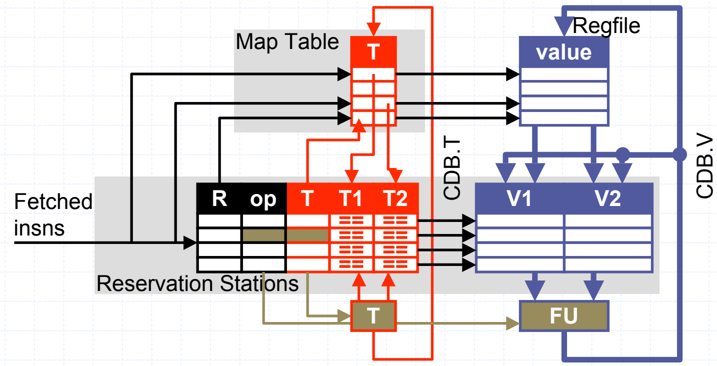
- Stall for structural (RS) hazards
  - Allocate RS entry
  - Input register ready ? read value into RS : read tag into RS
  - Set register status (i.e., rename) for ouput register

Tomasulo Issue (S)

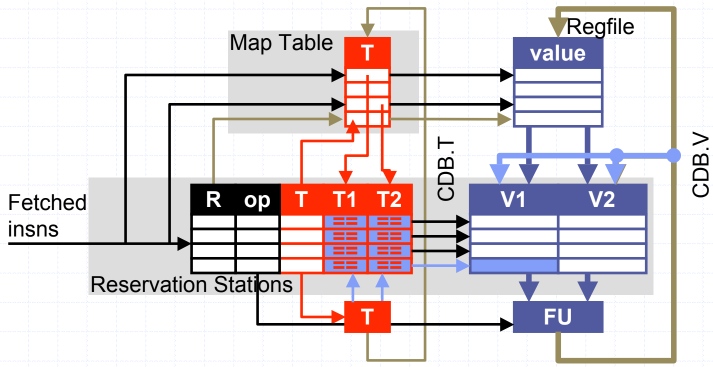


- Wait for RAW hazards
  - Read register values from RS

Tomasulo Execute (X)

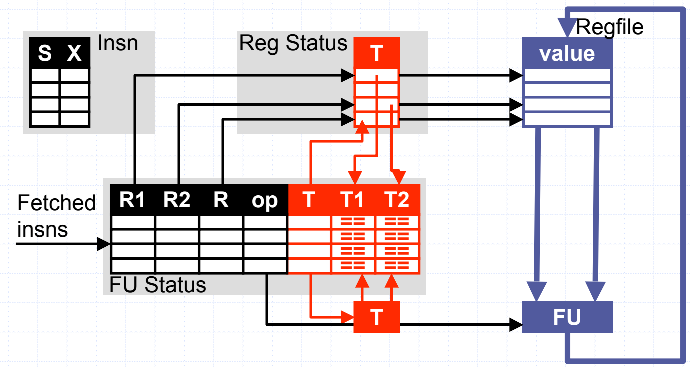


Tomasulo Writeback (W)

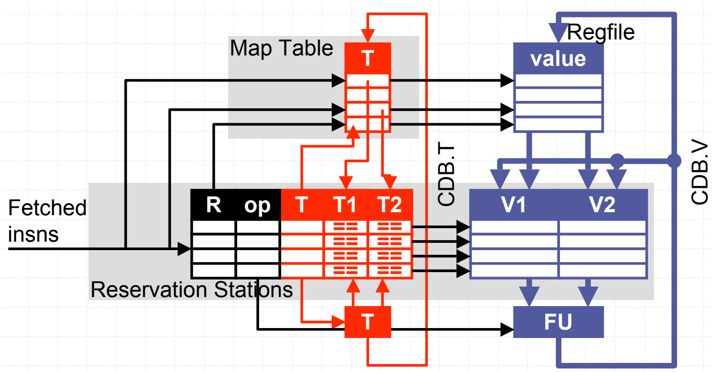


- Wait for structural (CDB) hazards
  - Output Reg Status tag still matches? clear, write result to register
  - CDB broadcast to RS: tag match ? clear tag, copy value
  - Free RS entry

Difference Between Scoreboard…



…And 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

Value-Based Renaming Example

ldf X(r1),f1 (allocated RS#2)

- MT[r1] == 0 → RS[2].V2 = RF[r1]
- MT[f1] = RS#2

mulf f0,f1,f2 (allocate RS#4)

- MT[f0] == 0 → RS[4].V1 = RF[f0]
- MT[f1] == RS#2 → RS[4].T2 = RS#2
- MT[f2] = RS#4

addf f7,f8,f0

- Can write RF[f0] before **mulf** executes, why?

ldf X(r1),f1

- Can write RF[f1] before **mulf** executes, why?
- Can write RF[f1] before first **ldf**, why?

Map Table

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

Reservation Stations

T	FU	busy	op	R	T1	T2	V1	V2
2	LD	yes	ldf	f1	–	–	–	[r1]
4	FP1	yes	mulf	f2	–	RS#2	[f0]	

Tomasulo Data Structures

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]	allocate
3	ST	no							
4	FP1	no							
5	FP2	no							

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]	–	allocate
5	FP2	no							

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]	allocate
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-	
5	FP2	no							

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]	-	allocate
2	LD	no							free
3	ST	yes	stf	-	RS#4	-	-	[r1]	
4	FP1	yes	mulf	f2	-	RS#2	[f0]	CDB.V	RS#2 ready → grab CDB value
5	FP2	no							

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	-	-	allocate
3	ST	yes	stf	-	RS#4	-	-	[r1]	
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]	
5	FP2	no							

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	RS#2
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 mulf 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

Scoreboard vs. Tomasulo

	Scoreboard				Tomasulo			
Insn	D	S	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c4	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8	c2	c4	c5+	c8
`stf f2,Z(r1)`	c3	c8	c9	c10	c3	c8	c9	c10
`addi r1,4,r1`	c4	c5	c6	c9	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c9	c10	c11	c5	c7	c8	c9
`mulf f0,f1,f2`	c8	c11	c12+	c15	c6	c9	c10+	c13
`stf f2,Z(r1)`	c10	c15	c16	c17	c10	c13	c14	c15

Hazard	Scoreboard	Tomasulo
Insn buffer	stall in D	stall in D
FU	wait in S	wait in S
RAW	wait in S	wait in S
WAR	wait in W	none
WAW	stall in D	none

Scoreboard vs. Tomasulo II: Cache Miss

	Scoreboard				Tomasulo			
Insn	D	S	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3+	c8	c1	c2	c3+	c8
`mulf f0,f1,f2`	c2	c8	c9+	c12	c2	c8	c9+	c12
`stf f2,Z(r1)`	c3	c12	c13	c14	c3	c12	c13	c14
`addi r1,4,r1`	c4	c5	c6	c13	c4	c5	c6	c7
`ldf X(r1),f1`	c8	c13	c14	c15	c5	c7	c8	c9
`mulf f0,f1,f2`	c12	c15	c16+	c19	c6	c9	c10+	c13
`stf f2,Z(r1)`	c13	c19	c20	c21	c7	c13	c14	c15

- Assume
  - 5 cycle cache miss on first `ldf`
  - Ignore FUST and RS structural hazards
  - \+ Advantage Tomasulo
    - No `addi` **WAR hazard** (`c7`) means iterations run in parallel

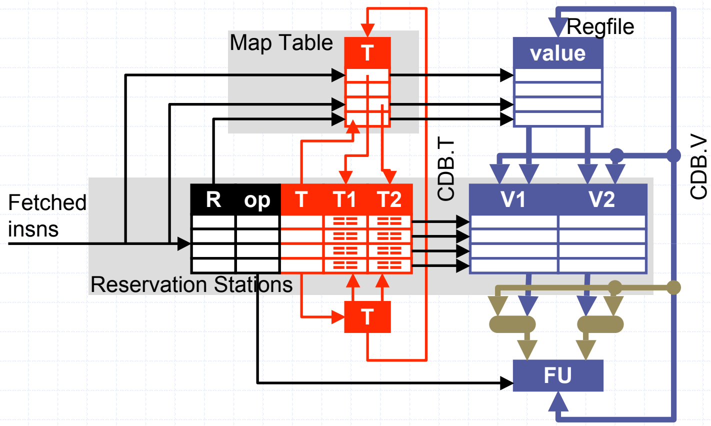
Can We Add Superscalar?

- 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)

Can We Add Bypassing?



- Yes, but it's more complicated than you might think
  - In fact: requires a completely new pipeline

Why Out-of-Order Bypassing Is Hard

	No Bypassing				Bypassing			
Insn	D	S	X	W	D	S	X	W
`ldf X(r1),f1`	c1	c2	c3	c4	c1	c2	c3	c4
`mulf f0,f1,f2`	c2	c4	c5+	c8	c2	c3	c4+	c7
`stf f2,Z(r1)`	c3	c8	c9	c10	c3	c6	c7	c8
`addi r1,4,r1`	c4	c5	c6	c7	c4	c5	c6	c7
`ldf X(r1),f1`	c5	c7	c8	c9	c5	c7	c7	c9
`mulf f0,f1,f2`	c6	c9	c10+	c13	c6	c9	c8+	c13
`stf f2,Z(r1)`	c10	c13	c14	c15	c10	c13	c11	c15

- Bypassing: **`ldf`** X in c3 → **`mulf`** X in c4 → **`mulf`** S in c3
  - But how can **`mulf`** S in c3 if **`ldf`** W in c4? Must change pipeline
- Modern scheduler
  - Split CDB tag and value, move tag broadcast to S
    - **`ldf`** tag broadcast now in cycle 2 → mulf S in cycle 3
  - How do multi-cycle operations work? How do cache misses work?

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