Little's Law

- N = ʎ x T
- N: Average number of jobs waiting
- ʎ: Rate of jobs arrival
- T: average time for a job to get done after arriving

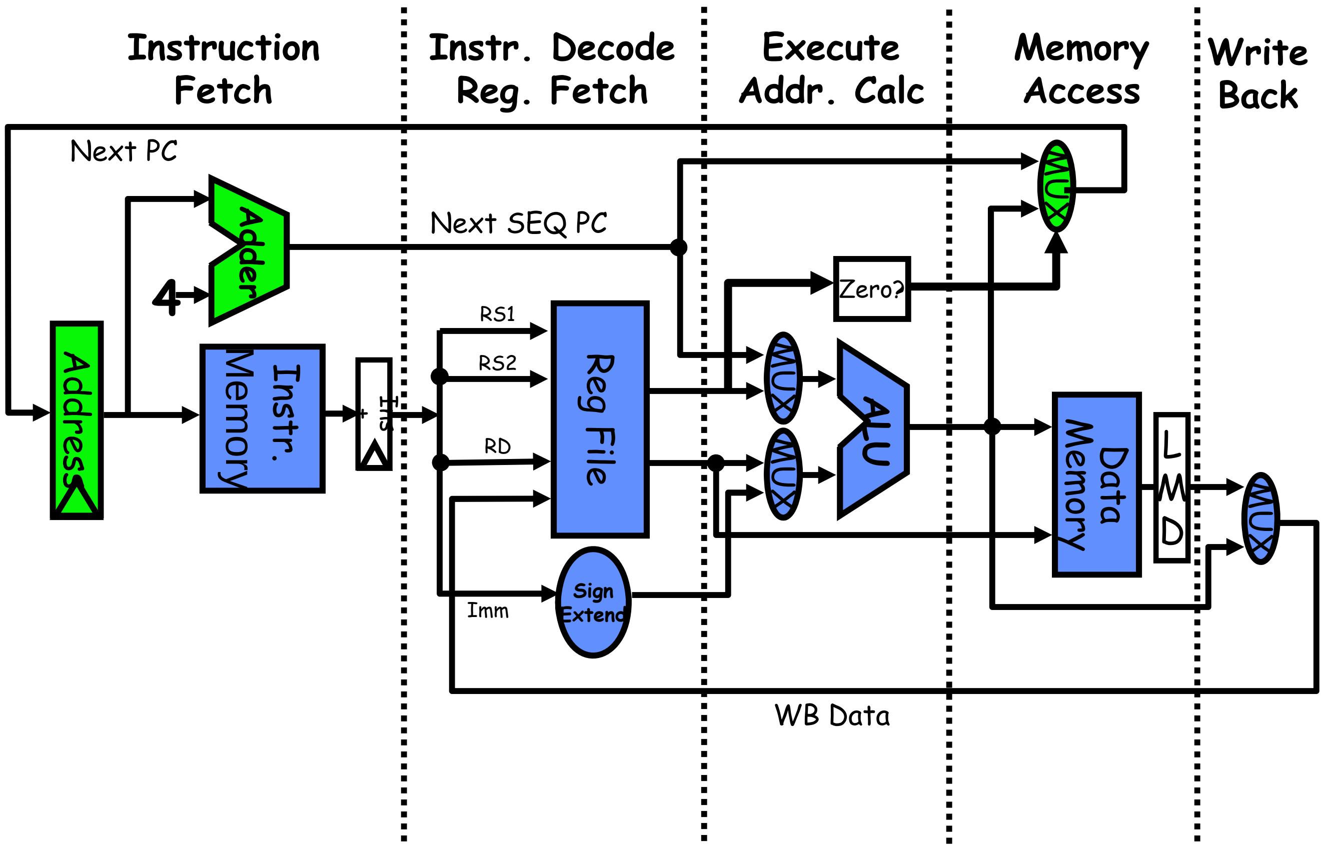
- Q.1 10 queries arrive per second, each query needs 300ms to be serviced, how many queries wait?
- Q.2 if there are 10 jobs waiting on average in a queue and a jobs get serviced in 2s, at what rate the jobs arrive?
- Q.3 if there is a cache miss every 20 cycles, and a miss waits in a queue to be serviced for 100 cycles, how many jobs wait in a queue on average? How big of a queue you will use?

Κεφ. 2: Παραλληλισμός μεταξύ Εντολών
Instruction Level Parallelism (ILP)

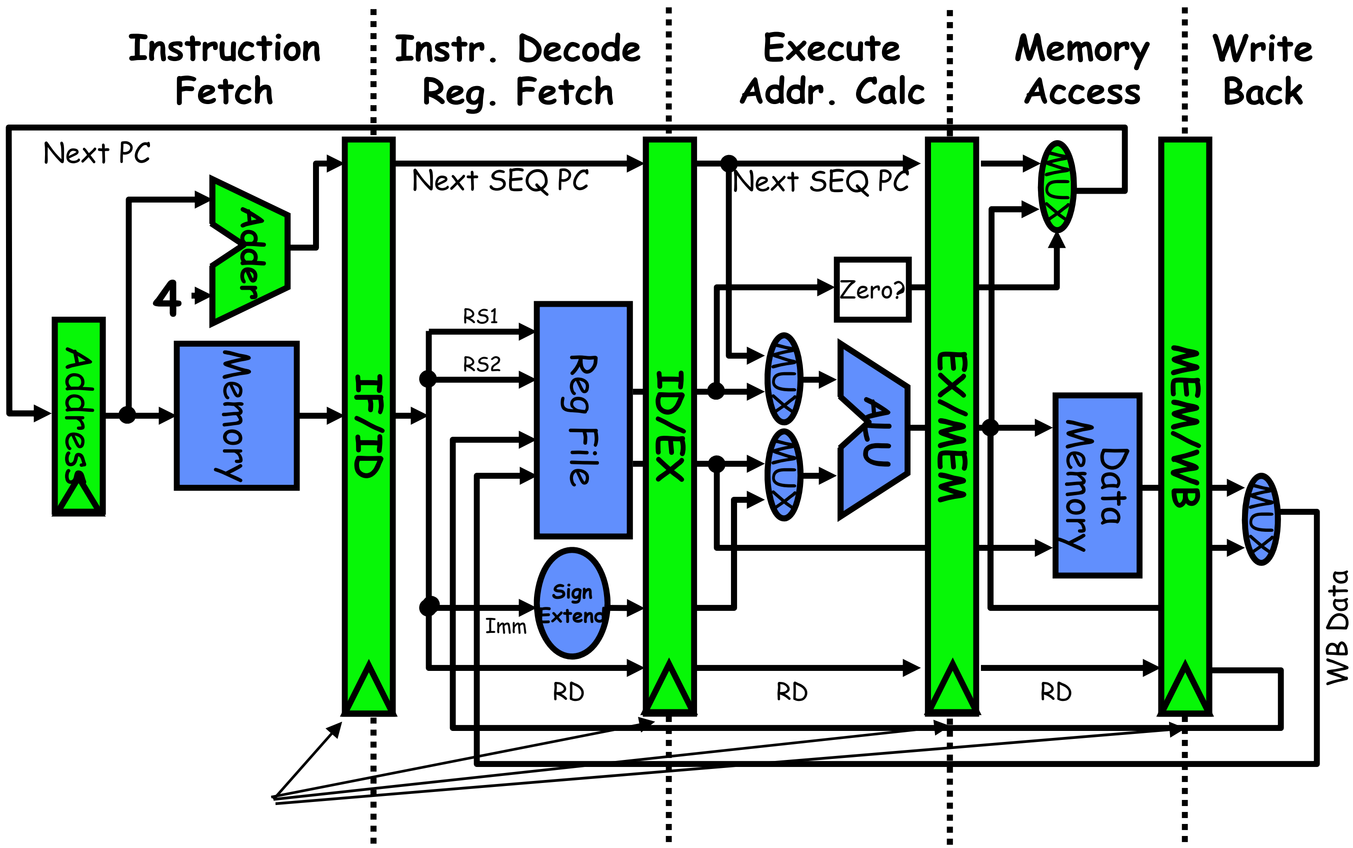
Types of Processor Microarchitecture

- Pipeline/Non-pipelined Processors
- In-Order/Out-of-Order Processors
- Scalar/Superscalar Processors
  - Dynamically and Statically scheduled
- Vector Processors
  - Operations that enable Single operation on multiple data
  - SIMD
- Multicore Processors
- Multithreaded Processors
  - A core can execute multiple threads
- Accelerators
  - Graphics Processor Units, Neural Nets, Encryption/Decryption, Compression, …

Non-Pipelined



Pipelined

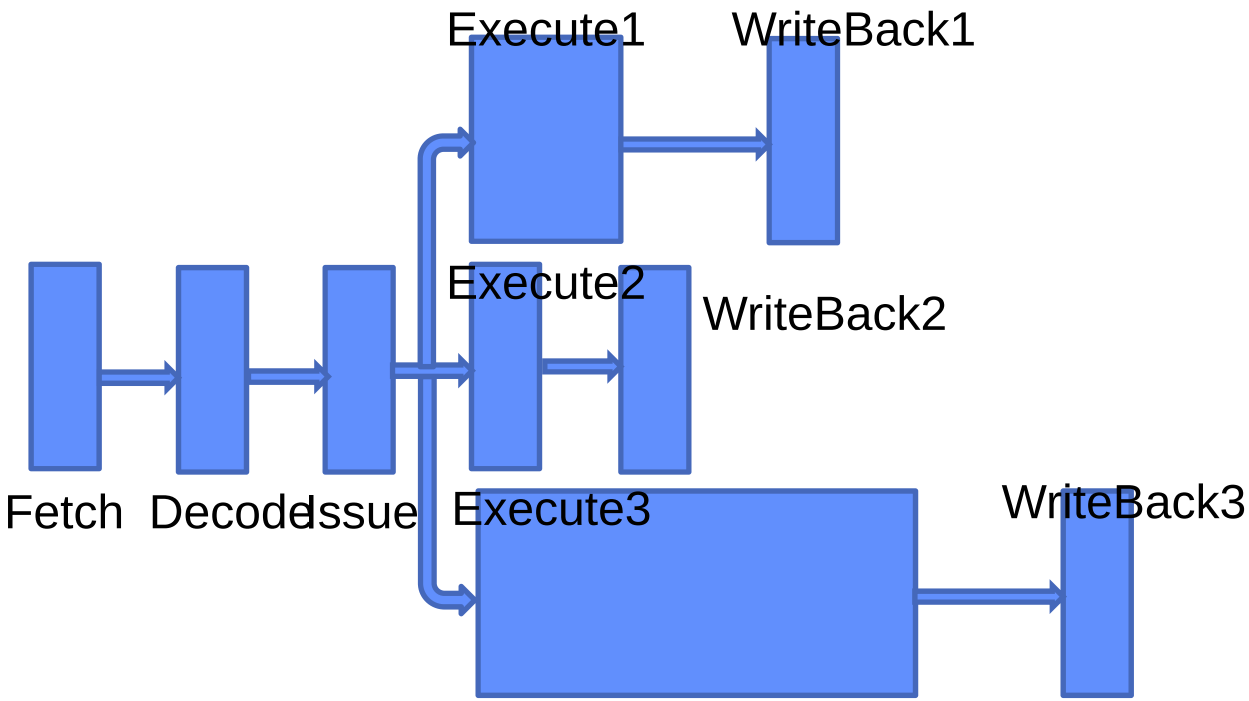


Instruction Fetch
Instr. Decode Reg. Fetch
Execute Addr. Calc
Memory Access
Write Back
Next PC
Adder
4
Address
Memory
IF/ID
Next SEQ PC
RS1
RS2
Reg File
Sign Extend
Imm
RD
ID/EX
Next SEQ PC
Zero?
MUX
ALU
RD
EX/MEM
MUX
Data Memory
RD
MEM/WB
MUX
WB Data

In-Order

- Processors that execute instructions in program-order
- Instructions flow through the pipeline in the same order they appear in the program

Out-of-Order

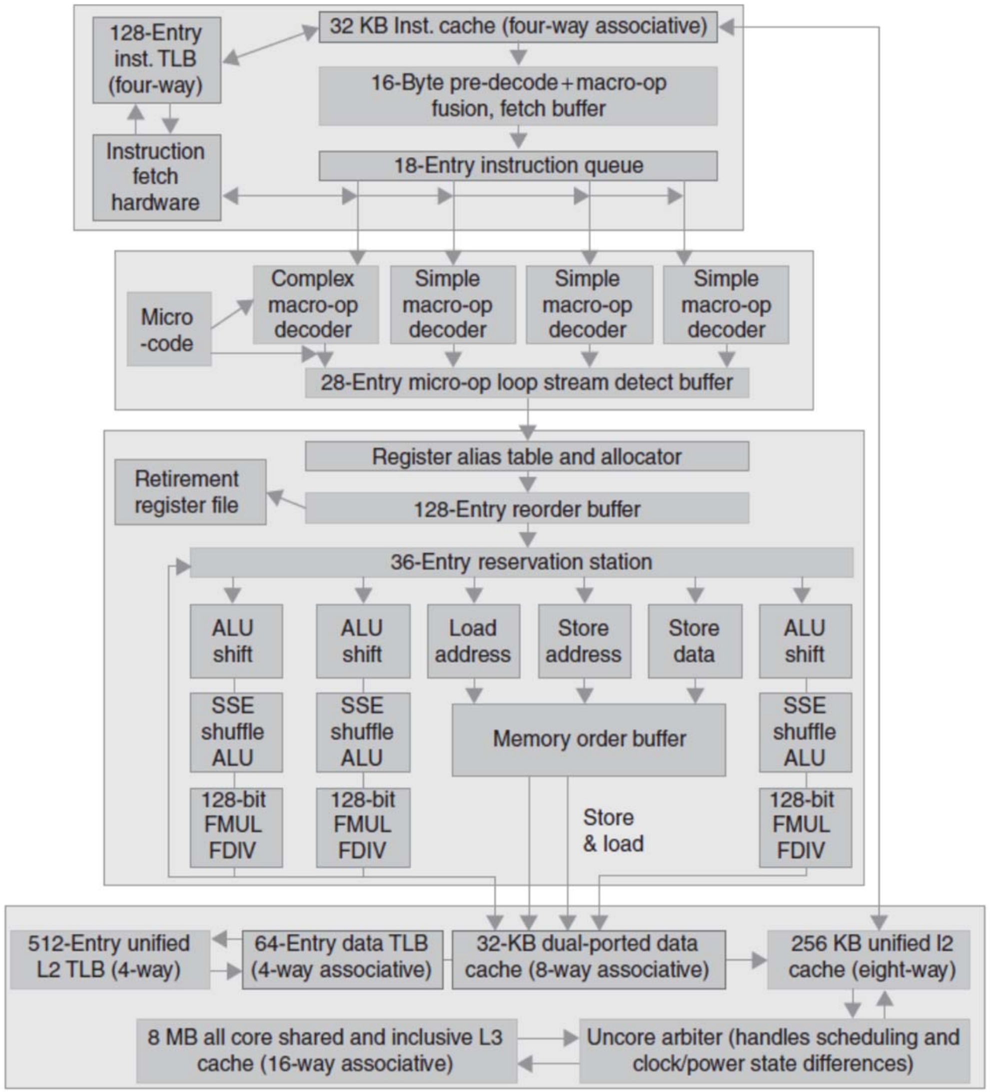

Execute1
WriteBack1
Execute2
WriteBack2
Fetch
Decode
Issue
Execute3
WriteBack3

Out-of-Order

- Instructions can execute in different order than what they appear in the dynamic program order
  - Front-end common
  - Back-end multiple paths in the pipeline
- Allow this while ensuring precise exceptions (program order) and correct dataflow
  - » Register Renaming
  - » Reorder-Buffer

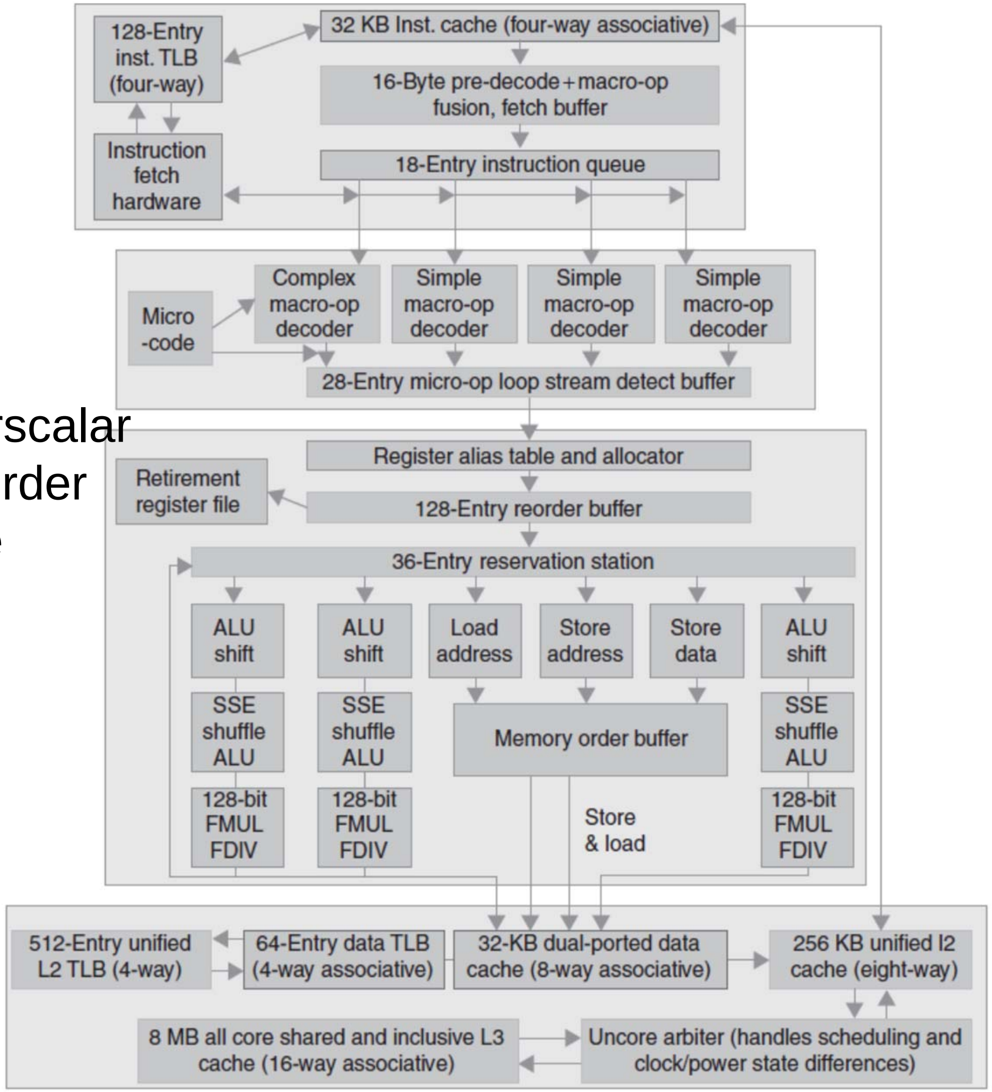
Superscalar Processors

- Fetch/Issue/Execute/Commit multiple instructions
- Both in-order and out-of-order superscalar processors
- Scalar Processors: single-wide processors
- Speculation: Execute instructions that do not appear in dynamic program



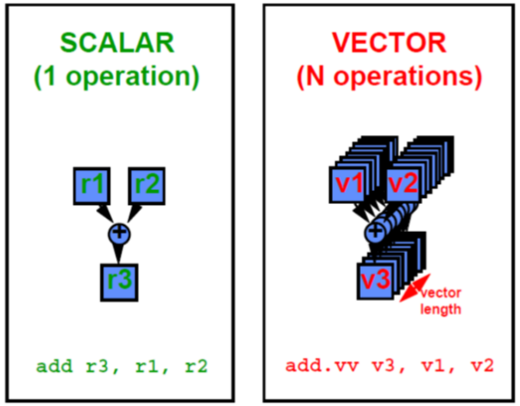
128-Entry inst. TLB (four-way)
32 KB Inst. cache (four-way associative)
16-Byte pre-decode + macro-op fusion, fetch buffer
Instruction fetch hardware
18-Entry instruction queue
Micro -code
Complex macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
28-Entry micro-op loop stream detect buffer
Register alias table and allocator
Retirement register file
128-Entry reorder buffer
36-Entry reservation station
ALU shift
ALU shift
Load address
Store address
Store data
ALU shift
SSE shuffle ALU
SSE shuffle ALU
Memory order buffer
SSE shuffle ALU
128-bit FMUL FDIV
128-bit FMUL FDIV
Store & load
128-bit FMUL FDIV
512-Entry unified L2 TLB (4-way)
64-Entry data TLB (4-way associative)
32-KB dual-ported data cache (8-way associative)
256 KB unified l2 cache (eight-way)
8 MB all core shared and inclusive L3 cache (16-way associative)
Uncore arbiter (handles scheduling and clock/power state differences)

Intel Superscalar Out-of-Order Core



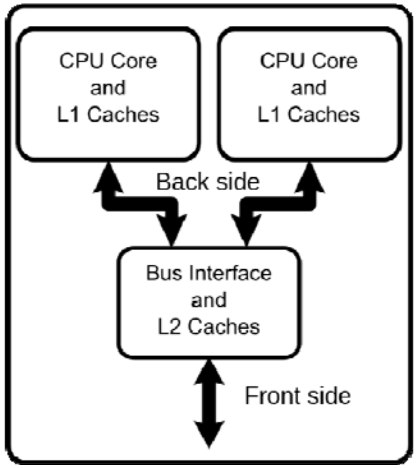
Vector Processors

- Instructions for operating on multiple-data
- Hardware support
  - Multiple execution units, wide-busses
  - Multi-cycle operation with help of FSM to read-execute-write over multip data

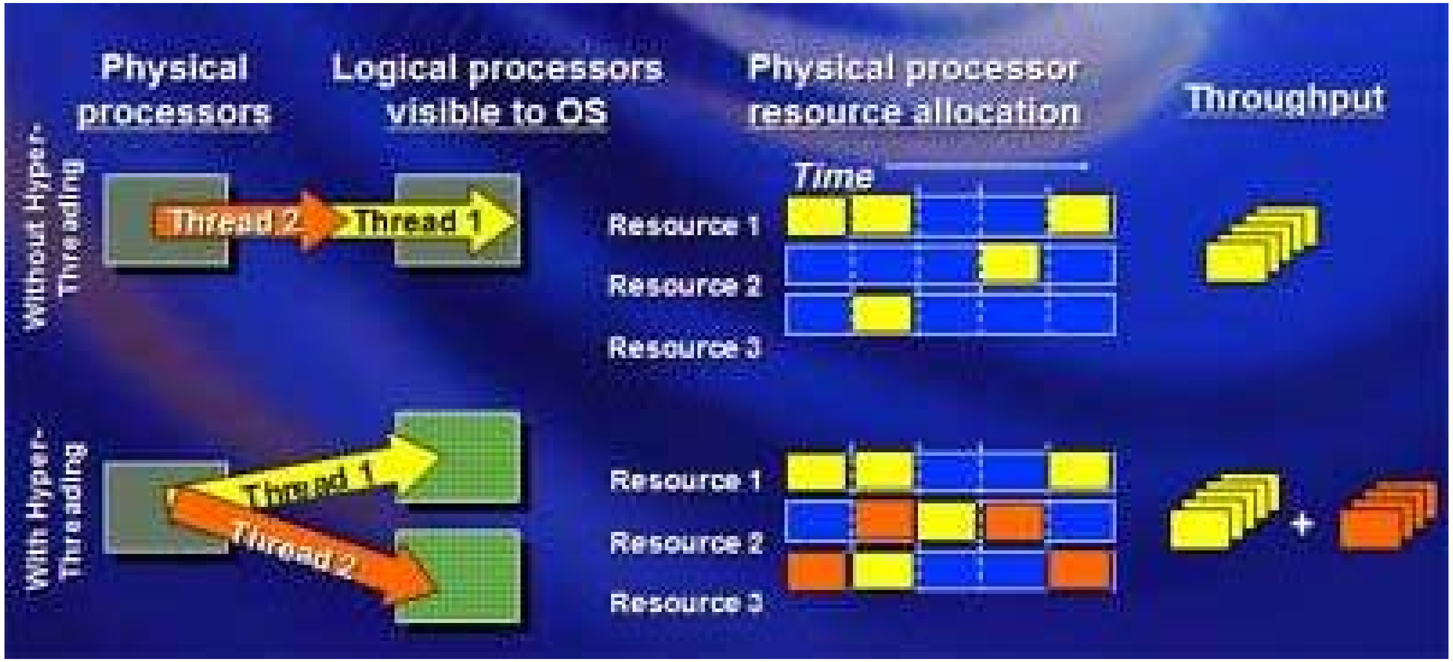


Multicore Processors

- Processor with more than one core
- Can execute multiple programs concurrently (as many as cores)
- Can execute parallel programs
  - Each program consists of multiple threads
- Resource sharing outside core: LLC, Memory, Busses memory



Multithreaded Cores



- A single core can execute multiple threads
  - Hypethreading or Simultaneous Multithreading
- Can execute multiple programs concurrently (as many as cores) or multiple-threads of a parallel program
  - Resource sharing with-in core: Caches, Predictors, Execution units, buses
  - Per core resources: PC, Register File

Execution Time and CPI

- T = I x CPI x CT
- Pipeline CPI = Ideal CPI +
  Structural Stalls +
  Data Hazard Stalls +
  Control Stalls
  - Ideal pipeline CPI:
    - » maximum performance possible
  - Structural hazards:
    - » HW cannot support this combination of instructions
  - Data hazards:
    - » dependence on a result of prior instruction still in the pipeline
    - » RAW (true data dependence), WAW and WAR
  - Control hazards:
    - » Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps)

Μέθοδοι για αύξηση της επίδοσης: παραλληλισμός

- Instruction level parallelism (ILP)
- Thread Level Parallelism
  - requires programming changes
- Data Level Parallelism
- Program Level Parallelism
  - run many programs

Μέθοδοι για αύξηση ILP

- Instruction level parallelism

- Dynamic ILP:Επίλυση Εξαρτήσεων και Κινδύνων Δυναμικά at run time-(HW)

- Static ILP: στατικές μέθοδοι ILP στατικά at compile time (Compiler - SW)

- Metric of ILP: IPC=1/CPI

- Improve IPC without other overheads
  - CT (and I)
  - Power cost
  - Design complexity

How to uncover the ILP

- Violate program order and control dependences
- But maintain correctness
  - same dataflow and exception behavior
- Do above effectively AND efficiently!

Program Order

- program order: order instructions would execute in if executed sequentially 1 at a time as determined by original source program (assembly level)
- Goal: exploit parallelism by preserving appearance of program order
  - *modify order in manner that cannot be observed by programmer*
  - must not affect the outcome of the program (τι είναι outcome?)

1. ld r1,4(r2)
2. add r2, r1, r3
3. sub r1, r4,r5
4. or r3, r1, 1

Simple pipeline: if ld misses then every one stalls

But sub and or have no dependence on the ld? Why wait?

Renaming to help ILP

» ld r1,4(r2)
» add r2, r1, r3
» sub r1, r4,r5
» or r3, r1, 1

if sub executes earlier then add may get wrong value for r1 (violate dataflow). But this is not a true data dependence.

Rename and change program order

» ld r1,4(r2)
» add r2, r1, r3
» sub r6, r4,r5
» or r3, r6, 1

- Can increase ILP: sub can execute before or in parallel
- Renaming used by both by hw and sw ILP techniques

Register Renaming

- Preserve RAW
- Remove name dependence for regs (WAR, WAW)
- Maintain Dataflow and enable reordering and increase performance

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order

```
if p1 {
  S1;
};
if p2 {
  S2;
}
```

- S1 is control dependent on p1, and S2 is control dependent on p2 but not on p1.

Control Dependence Ignored (temporarily)

- Control dependence need not be respected all the time
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, ONLY if can do so without affecting correctness of the program
  - I.e. create a speculative program order that gets verified (execute instructions that may not belong in program)

- "Ignoring" control and modifying state is called SPECULATION

Example Program Order:

```
        DADDU         R2,R3,R4
        BEQZ          R2,L1
        LW            R1,0(R4)
  L1:
        DSUB          R2,R2,R1

  with ooo + speculation execution order can become
        LW            R1,0(R4)
        DADDU         R2,R3,R4
        BEQZ-Check    R2,L1 -- may need to cancel instructions after BEQZ
        DSUB          R2,R2,R1
```

Data Flow

- **Data flow**: actual flow of data values among instructions that produce results and those that consume them
  - branches make flow dynamic, determine which instruction is supplier of data
- Example:

```
DADDU       R1,R2,R3
BEQZ R4,L
DSUBU       R1,R5,R6
L:    …
OR          R7,R1,R8
```

- OR depends on DADDU or DSUBU?
  - Must preserve data flow on execution
  - Need recovery when violating control dependences
- True Data Dependences set performance limits

Exception Behavior

- Exceptions: abnormal program behavior (overflow, divide by zero, page fault, etc)

- Preserving exception behavior => any changes in instruction execution order must not change how exceptions are raised in program (=> no new exceptions)
- Example:

```
        DADDU       R2,R3,R4
        BEQZ        R2,L1
        SW          R1,0(R2)
L1:
```

- Problem with moving SW before BEQZ?

These lectures...

- How to reorder instructions and speculate in hardware BUT STILL maintain correctness (dataflow and exceptions)
- HW
  - Dynamic Scheduling (out-of-order issue/execution)
  - Renaming
  - Reservation Stations
  - Reorder Buffer
  - Prediction/Speculation
  - Memory Dependences
- *SW*
  - *Static scheduling*
  - *Loop unrolling*
  - *Trace Scheduling*

Dynamic Scheduling

- Decide which instruction executes dynamically
  - Determines which instructions are ready to execute
  - Which of the ready instructions are selected for execution
- KEY: able to track dependences between instructions and know how long (cycles) an instruction takes to complete

HW Scheme for violating program order (OOO)

- Key idea: Allow instructions after stall to proceed

```
DIVD  F0,F2,F4
ADDD  F10,F0,F8
SUBD  F12,F8,F14
```

- Enables out-of-order execution
  and allows out-of-order completion (write result)
- Will distinguish when an instruction *begins execution* and when it *completes execution*; between 2 times, the instruction is *in execution*

Basic Pipeline Stages

- Fetch
- Decode
- Issue
- Execute
- Writeback

Dynamic Scheduling and Pipeline

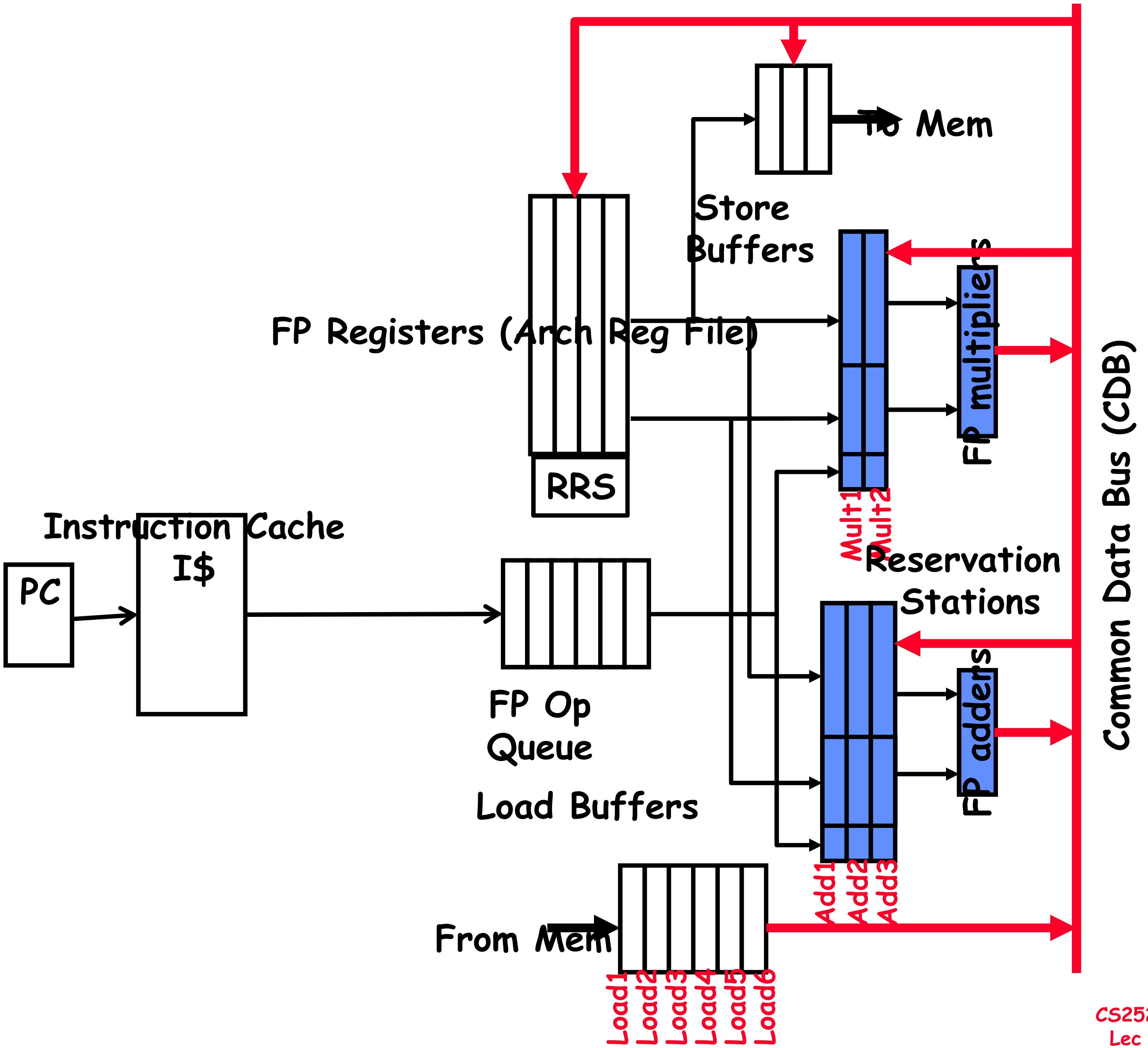
- Split the Decode pipe stage of simple 5-stage pipeline into 2 stages:
- *Decode/Rename*
  - Decode instructions, check for structural hazards
- *Queue/Issue*
  - Wait until no data hazards
  - when ready and selected issue
- In a dynamically scheduled pipeline, all instructions pass through fetch/decode stages in order (in-order decoding/renaming). Why???

A Dynamic Algorithm: Tomasulo's Algorithm

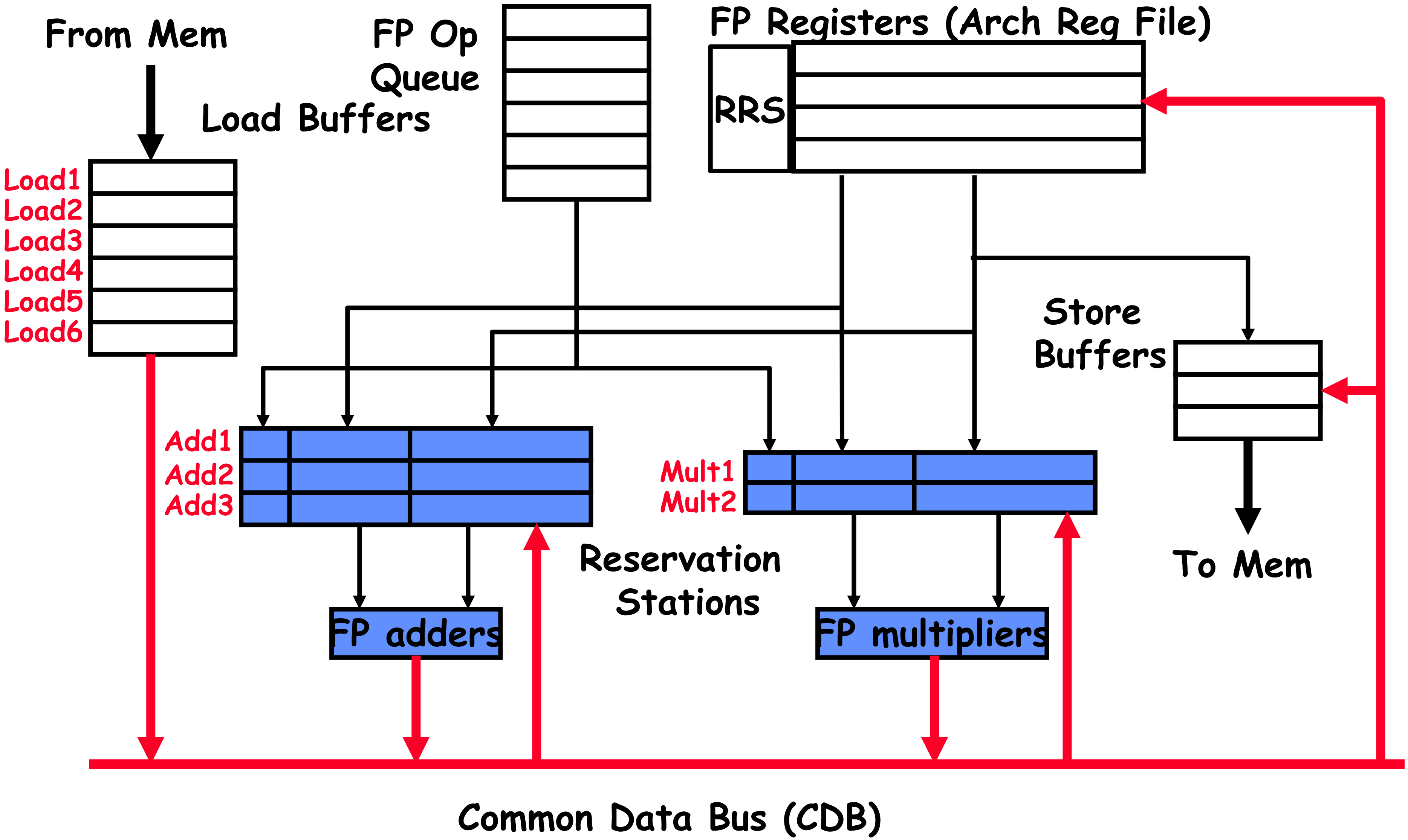
- For IBM 360/91 (before caches!)
- Goal: High Performance without special compilers
- Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
  - This led Tomasulo to try to figure out how to get more effective registers — renaming in hardware!
- Why Study 1966 Computer?
- The descendants of this have flourished!
  - Alpha 21264, HP 8000, MIPS 10000, Pentium IV, PowerPC , Core, A15, Opteron, Power8,…

Tomasulo Algorithm

- Control & buffers distributed with Function Units (FU)
  - FU buffers called "reservation stations"; have pending operands: results of previous instructions
  - Each RS has an identifier
- Registers in instructions replaced by values or pointers to reservation stations (RS) (often called Tags);
  - register renaming ;
  - avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
- Load and Stores treated as FUs with RSs as well


To Mem
Store
Buffers
FP Registers (Arch Reg File)
RRS
FP multipliers
Mult1
Mult2
Reservation
Stations
Common Data Bus (CDB)
Instruction Cache
I$
PC
FP Op
Queue
FP adders
Load Buffers
Add1
Add2
Add3
From Mem
Load1
Load2
Load3
Load4
Load5
Load6

Tomasulo Organization



Reservation Station Components and the Register Result Status

Op: Operation to perform in the unit (e.g., + or –)

Vj, Vk: Value of Source operands

- Store buffers has V field, result to be stored

Qj, Qk: Reservation stations producing source registers (value to be written)

- Note: Qj,Qk=0 => ready
- Store buffers only have Qi for RS producing result

Busy: Indicates reservation station or FU is busy

Register result status (RRS)—Table that indicates which pending instruction will write each register, if one exists. Invalid when no pending instructions that will write that register.

How is the RRS indexed?

What it contains?

Three Stages of Tomasulo Algorithm

1. Rename—get instruction from Op Queue

   If there is reservation station free (no structural hazard) get a RS, issue instr & send operands (or renames registers)

   update destination Register in RRS as pending indicating which RS identifier will be used to broadcast result.

2. Issue/Execute—operate on operands (EX)

   When both operands ready then issue to execute; if not ready, watch Common Data Bus for result

3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting units; and destination if match with RRS; mark reservation station available

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits of data + 4 bits of RS source address
  - Write source operands that matches broadcast RS (produces result)
  - Write also Register File (if matches RS) and flag register value available
  - Does the broadcast
- Example speed:
  3 clocks for Fl .pt. +,-; 2 for load; 10 for * ; 40 clks for /

Tomasulo Example

Instruction stream

Instruction status:

Instruction		j	k	Rename	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	

3 Load/Buffers

Reservation Stations:

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

FU count down

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

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

Clock cycle counter

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	*FU*				Load1					

Tomasulo Example Cycle 2

Instruction status:

Instruction		j	k	Rename	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					

Note: Can have multiple loads outstanding (non-blocking)

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	*FU*	Mult1	Load2		Load1					

- Note: registers names are removed ("renamed") in Reservation Stations; MULT issued
- 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	

Ships in the night

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				

- **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		M(A1)	Mult1	

Register result status:

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

- Timer starts down for Add1, Mult1

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		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			

- Issue ADDD here despite name dependency on F6?

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		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			

- Add1 (SUBD) 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	(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 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	(M-M)	M(A2)		
	Add3	No					
6	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
9	*FU*	Mult1	M(A2)		Add2	(M-M)	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	(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			

- Add2 (ADDD) 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		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			

- Write result of ADDD here?
- All quick instructions complete in this cycle!

Tomasulo Example Cycle 12

Instruction status:

Instruction		j	k	Rename	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		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 13

Instruction status:

Instruction		j	k	*Rename*	*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		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 14

Instruction status:

Instruction		*j*	*k*	*Rename*	*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		M(A1)	Mult1	

Register result status:

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

Tomasulo Example Cycle 15

Instruction status:

Instruction		*j*	*k*	*Rename*	*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			

- **Mult1 (MULTD) completing; what is waiting for it?**

Tomasulo Example Cycle 16

Instruction status:

Instruction		j	k	Rename	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			

- **Just waiting for Mult2 (DIVD) to complete**

skip several cycles

Tomasulo Example Cycle 55

Instruction status:

Instruction		j	k	Rename	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

Instruction status:

Instruction		j	k	*Renam e* (Exec Comp)	*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			

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

Tomasulo Example Cycle 57

Instruction status:

Instruction		j	k	Rename	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	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)	Result			

- In-order rename, out-of-order execution and out-of-order completion.

Tomasulo Loop Example

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

- This time assume Multiply takes 4 clocks
- Assume 1st load takes 8 clocks
  (L1 cache miss), 2nd load takes 1 clock (hit)
- Show multiple iterations

Loop Example

Instruction status:

ITER	Instruction		j	k	Exec Renam	Comp	Write Result
1	LD	F0	0	R1			
1	MULTD	F4	F0	F2			
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

Iter-ation Count

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Added Store Buffers

Reservation Stations:

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

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Instruction Loop

Register result status

Clock	**R1**		F0	F2	F4	F6	F8	F10	F12	...	F30
0	**80**	*Fu*									

Value of Register used for address, iteration control

Loop Example Cycle 1

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

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

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
1	80	Fu	Load1								

Loop Example Cycle 2

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	**80**	*Fu*	Load1		Mult1						

Loop Example Cycle 3

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
3	80	Fu	Load1		Mult1						

- **Implicit renaming sets up data flow graph**

Loop Example Cycle 4

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8 ←
BNEZ	R1	Loop	

Register result status

Clock	**R1**		F0	F2	F4	F6	F8	F10	F12	...	F30
4	**80**	*Fu*	Load1		Mult1						

- **Dispatching SUBI Instruction (not in FP queue)**

Loop Example Cycle 5

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	←

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
5	72	Fu	Load1		Mult1						

- **And, BNEZ instruction (not in FP queue)**

Loop Example Cycle 6

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Memory Disambiguation

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
6	72	Fu	Load2		Mult1						

- Notice that F0 never sees Load from location 80

Loop Example Cycle 7

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Fu	Load2		Mult2						

- Register file completely detached from computation
- First and Second iteration completely overlapped

Loop Example Cycle 8

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
8	72	Fu	Load2		Mult2						

Loop Example Cycle 9

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
9	72	Fu	Load2		Mult2						

- **Load1 completing: who is waiting?**
- **Note: Dispatching SUBI**

Loop Example Cycle 10

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd	M[80]	R(F2)		
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
10	64	Fu	Load2		Mult2						

- Load2 completing: who is waiting?
- Note: Dispatching BNEZ

Loop Example Cycle 11

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	Multd	M[80]	R(F2)		
4	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
11	64	Fu	Load3		Mult2						

- Next load in sequence

Loop Example Cycle 12

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	Multd	M[80]	R(F2)		
3	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
12	64	Fu	Load3		Mult2						

- **Why not issue third multiply?**

Loop Example Cycle 13

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	Multd	M[80]	R(F2)		
2	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
13	**64**	*Fu*	Load3		Mult2						

- **Why not issue third store?**

Loop Example Cycle 14

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	Multd	M[80]	R(F2)		
1	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
14	64	Fu	Load3		Mult2						

- **Mult1 completing. Who is waiting?**

Loop Example Cycle 15

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	
2	SD	F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	**64**	*Fu*	Load3		Mult2						

- **Mult2 completing. Who is waiting?**

Loop Example Cycle 16

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result		Busy	Addr	Fu
1	LD	F0	0	R1	1	9	10	Load1	No		
1	MULTD	F4	F0	F2	2	14	15	Load2	No		
1	SD	F4	0	R1	3			Load3	Yes	64	
2	LD	F0	0	R1	6	10	11	Store1	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	16	Store2	Yes	72	[72]*R2
2	SD	F4	0	R1	8			Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
16	64	Fu	Load3		Mult1						

Loop Example Cycle 17

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*		*Busy*	*Addr*	*Fu*
1	LD	F0	0	R1	1	9	10	Load1	No		
1	MULTD	F4	F0	F2	2	14	15	Load2	No		
1	SD	F4	0	R1	3			Load3	Yes	64	
2	LD	F0	0	R1	6	10	11	Store1	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	16	Store2	Yes	72	[72]*R2
2	SD	F4	0	R1	8			Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
17	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 18

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj (S1)	Vk (S2)	Qj (RS)	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

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

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
18	**64**	*Fu*	Load3		Mult1						

Loop Example Cycle 19

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*		*Busy*	*Addr*	*Fu*
1	LD	F0	0	R1	1	9	10	Load1	No		
1	MULTD	F4	F0	F2	2	14	15	Load2	No		
1	SD	F4	0	R1	3	18	19	Load3	Yes	64	
2	LD	F0	0	R1	6	10	11	Store1	No		
2	MULTD	F4	F0	F2	7	15	16	Store2	Yes	72	[72]*R2
2	SD	F4	0	R1	8	19		Store3	Yes	64	Mult1

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	←

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
19	**56**	*Fu*	Load3		Mult1						

Loop Example Cycle 20

Instruction status:

ITER	Instruction		j	k	Renaming (S1)	Exec Comp (S2)	Write Result (RS)			Busy	Addr	Fu
1	LD	F0	0	R1	1	9	10	11	Load1	Yes	56	
1	MULTD	F4	F0	F2	2	14	15	16	Load2	No		
1	SD	F4	0	R1	3	18	19	20	Load3	Yes	64	
2	LD	F0	0	R1	6	10	11	21	Store1	No		
2	MULTD	F4	F0	F2	7	15	16	22	Store2	No		
2	SD	F4	0	R1	8	19	20	23	Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

LD	F0	0	R1 ←
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

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

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

Advantages of Dynamic Scheduling

- OOO – out of order execution
- No WAW and WAR
  - Building data flow dependency graph on the fly.
  - Logical names translated to physical resources(tags, RS, Physical registers)
    - » Inputs from RRS
    - » Rename and save outputs to RRS
  - One logical register can be mapped to many physical registers
  - Reservation stations (physical destinations) buffer old values of logical registers
  - No WAR and WAW when dependences unknown at compile time
    - » (e.g., because they may involve a memory reference OR a control dependence OR across functions)
  - Allows code that compiled for one pipeline to run efficiently on a different pipeline

Advantages (cntd)

the distribution of the hazard detection logic

- **distributed reservation stations and the CDB**
- **If multiple instructions waiting on single result, & each instruction has other operand, then instructions can be released simultaneously by broadcast on CDB**
- **If a centralized register file were used, the units would have to read their results from the registers when register buses are available.**

Tomasulo Drawbacks

- Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Each CDB must go to multiple functional units $\Rightarrow$high capacitance, high wiring density
  - Number of functional units that can complete per cycle limited to one!
    - » Multiple CDBs $\Rightarrow$ more FU logic for parallel assoc stores
- *PROBLEM: Non-precise interrupts!*

What about Precise Interrupts (Exceptions)?

- State of machine looks as if no instruction beyond faulting instructions was renamed

- Tomasulo had:

  In-order rename

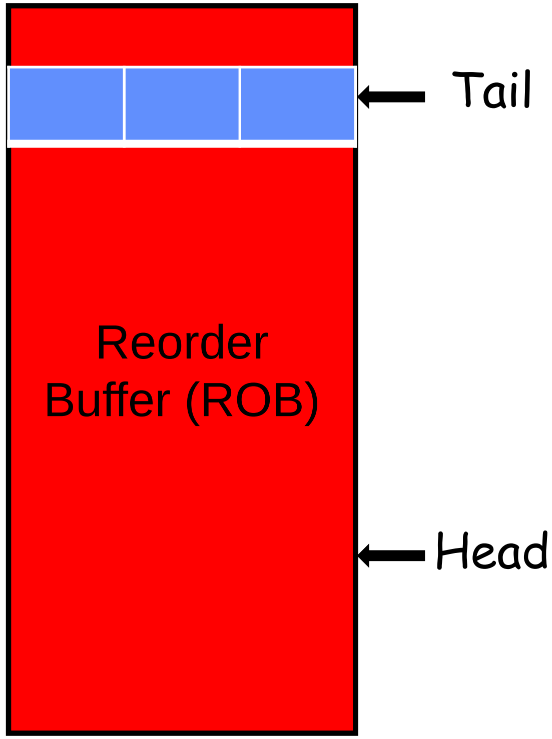
  Out-of-order execution, and

  Out-of-order completion

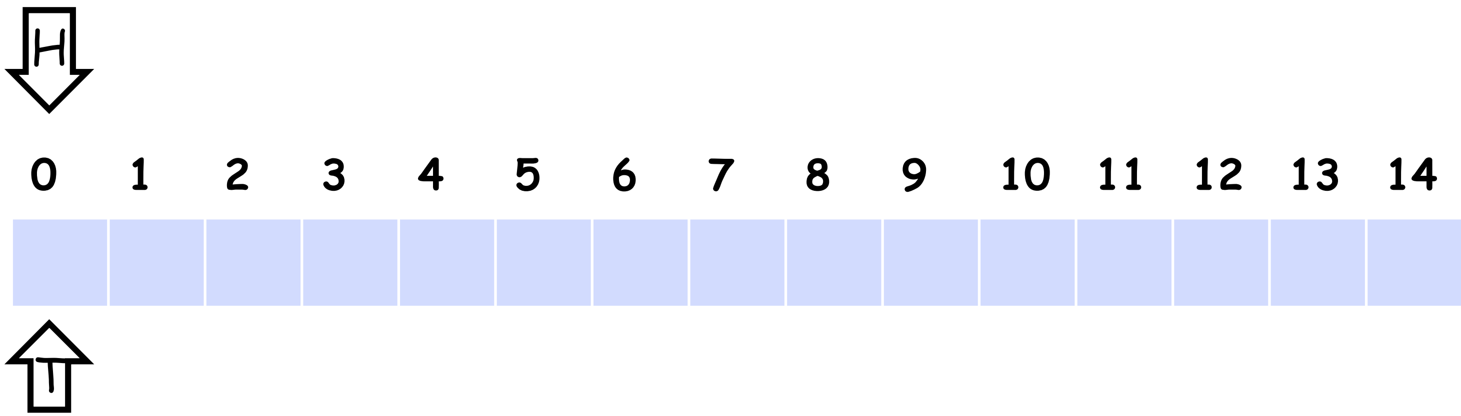
- Need to "fix" the out-of-order completion aspect so that we can find precise breakpoint in instruction stream.

HW support for precise interrupts

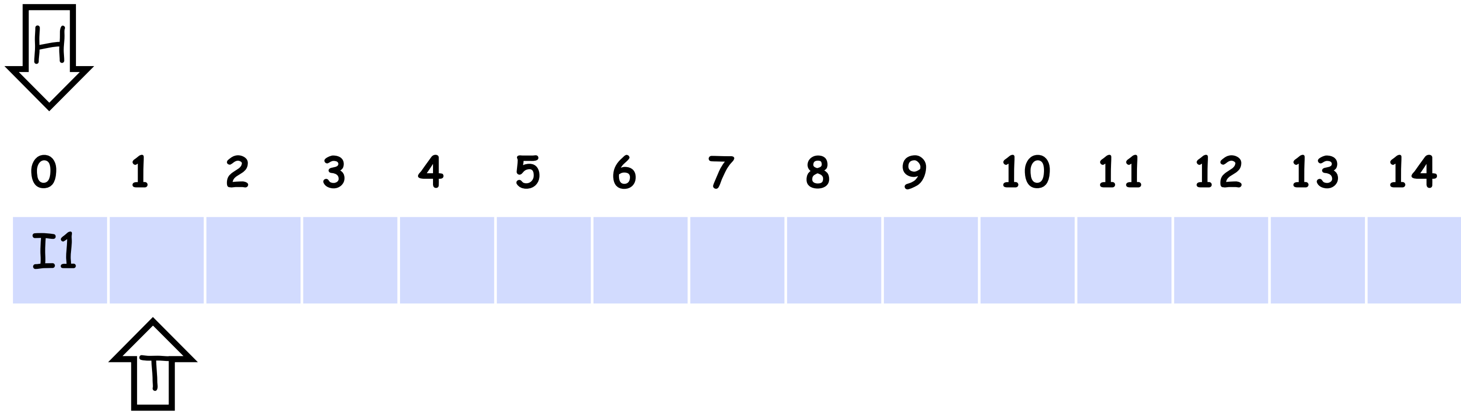
- Need HW buffer for results of uncommitted instructions: *reorder buffer (ROB)* *(instruction enter ROB in program order – head and tail)*
  - fields: instr, destination, value, PC, exception,
  - Use reorder buffer number instead of reservation station when execution completes
  - Supplies operands between execution complete & commit
  - (Reorder buffer can be operand source => more registers like RS)
  - Instructions commit
  - Once instruction commits, result is put into arch register
  - As a result, easy to undo instructions after exceptions or mispredicted branches (reset tail)



```
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1
```



```
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1     ⟸
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1
```



L1:

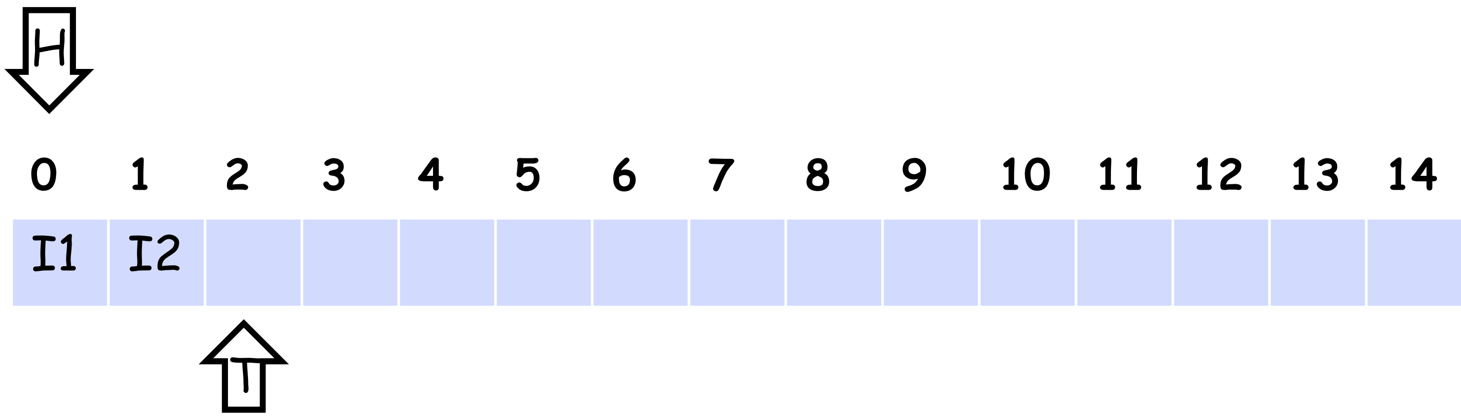
I1. lw r4,0(r8)

I2. add r4,r4,1

I3. sw r4,0(r8) ⟸

I4. add r8,r8,4

I5. bne r8,r9,L1



Several cycles later

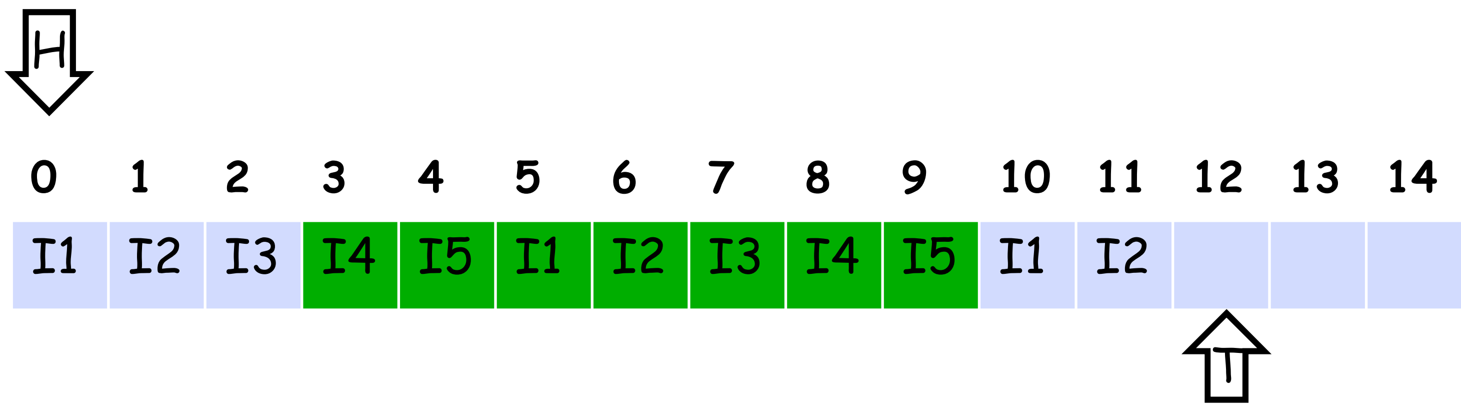
Renaming instructions for 3rd iteration

First load missed in L1

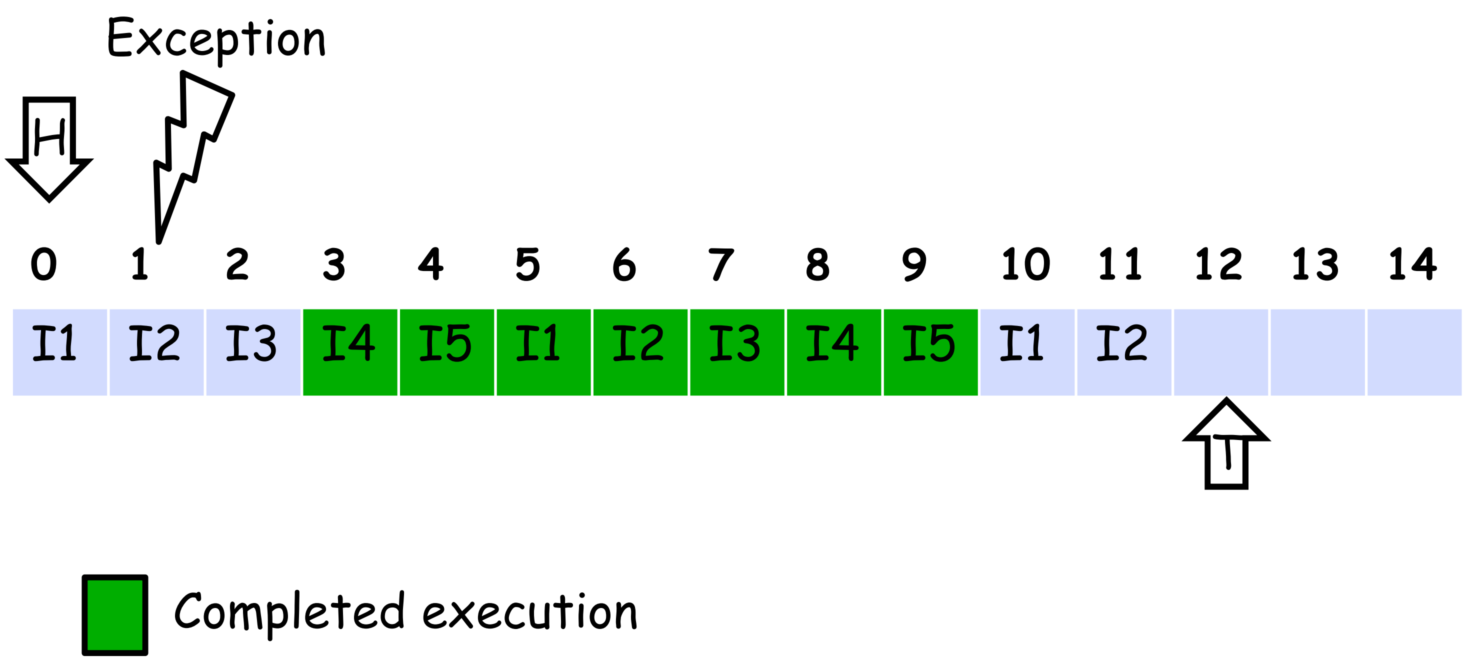
No instruction committed yet

Several instruction completed execution

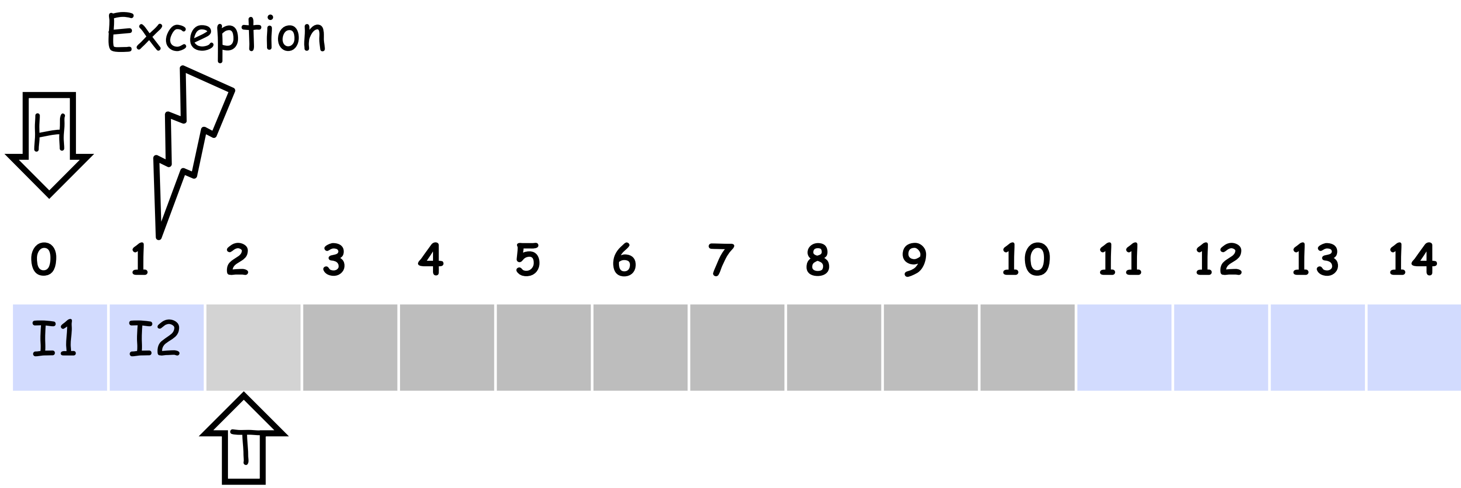
```
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1
```



```
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1
```

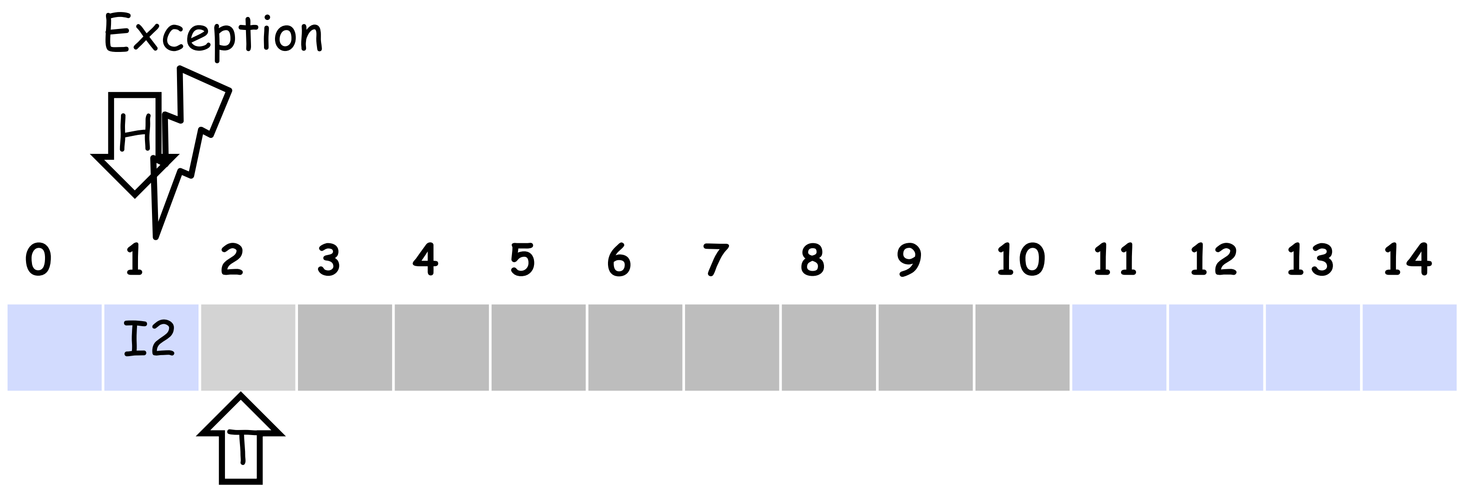


```
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1
```

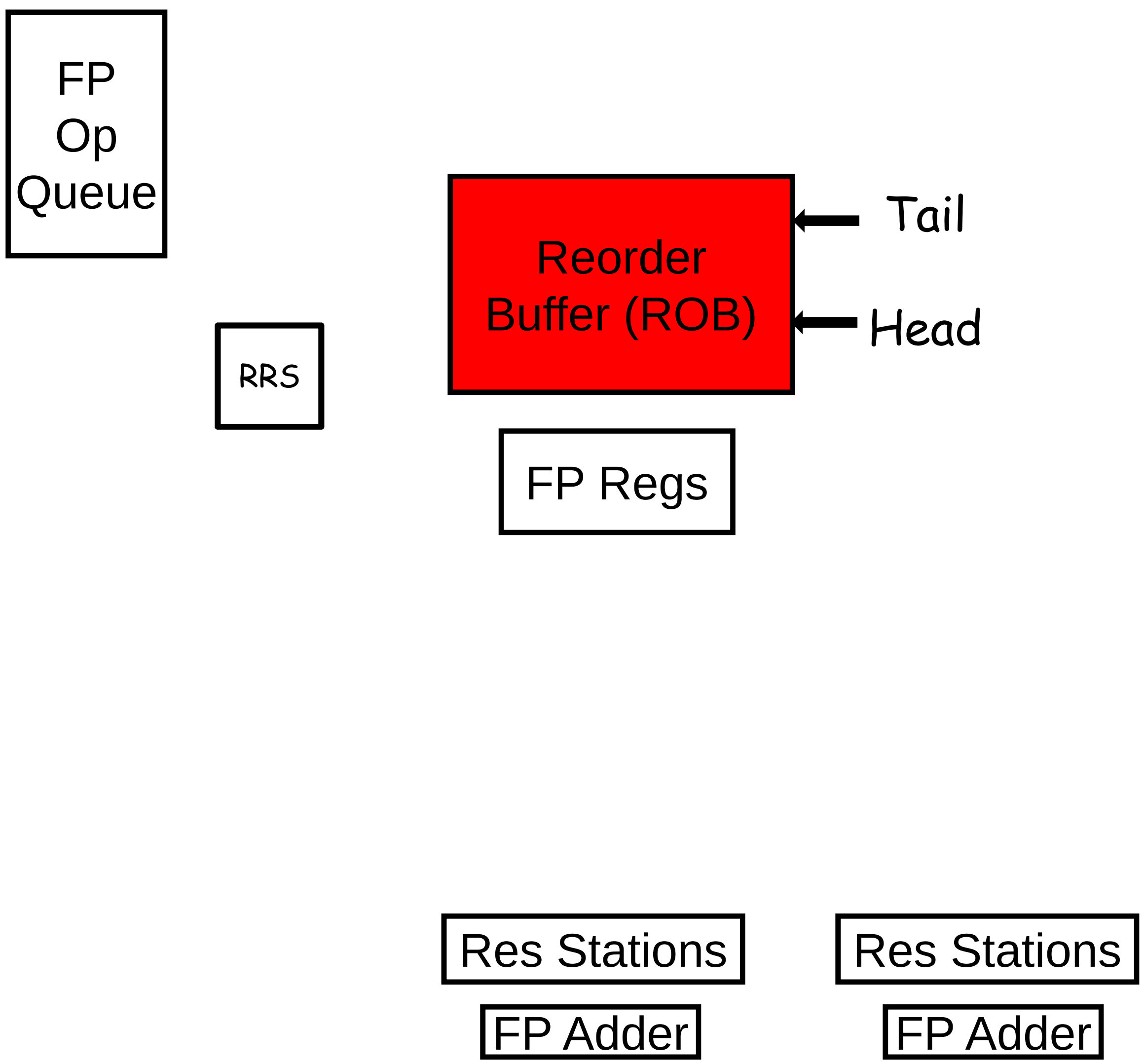


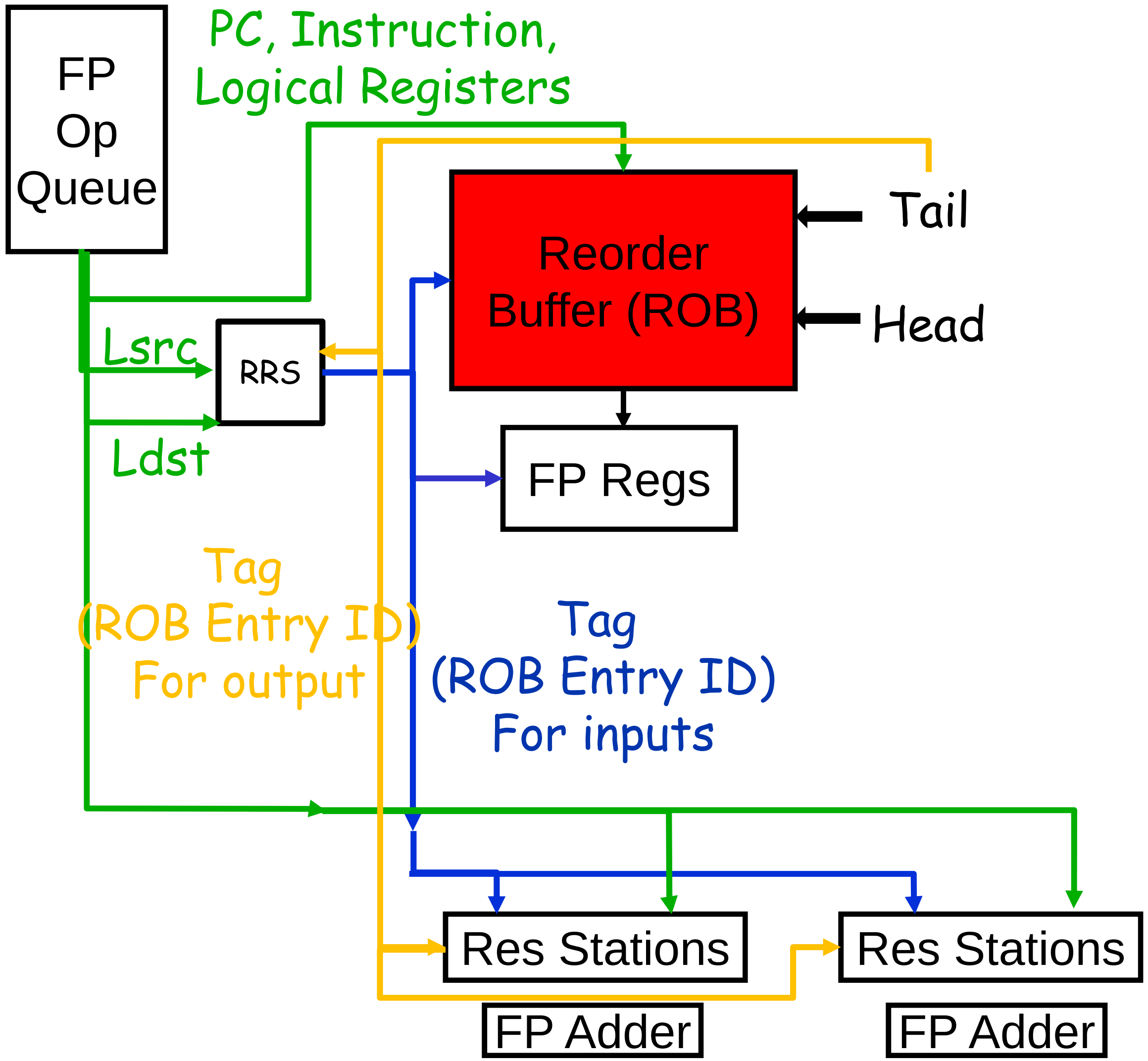
What are we waiting?

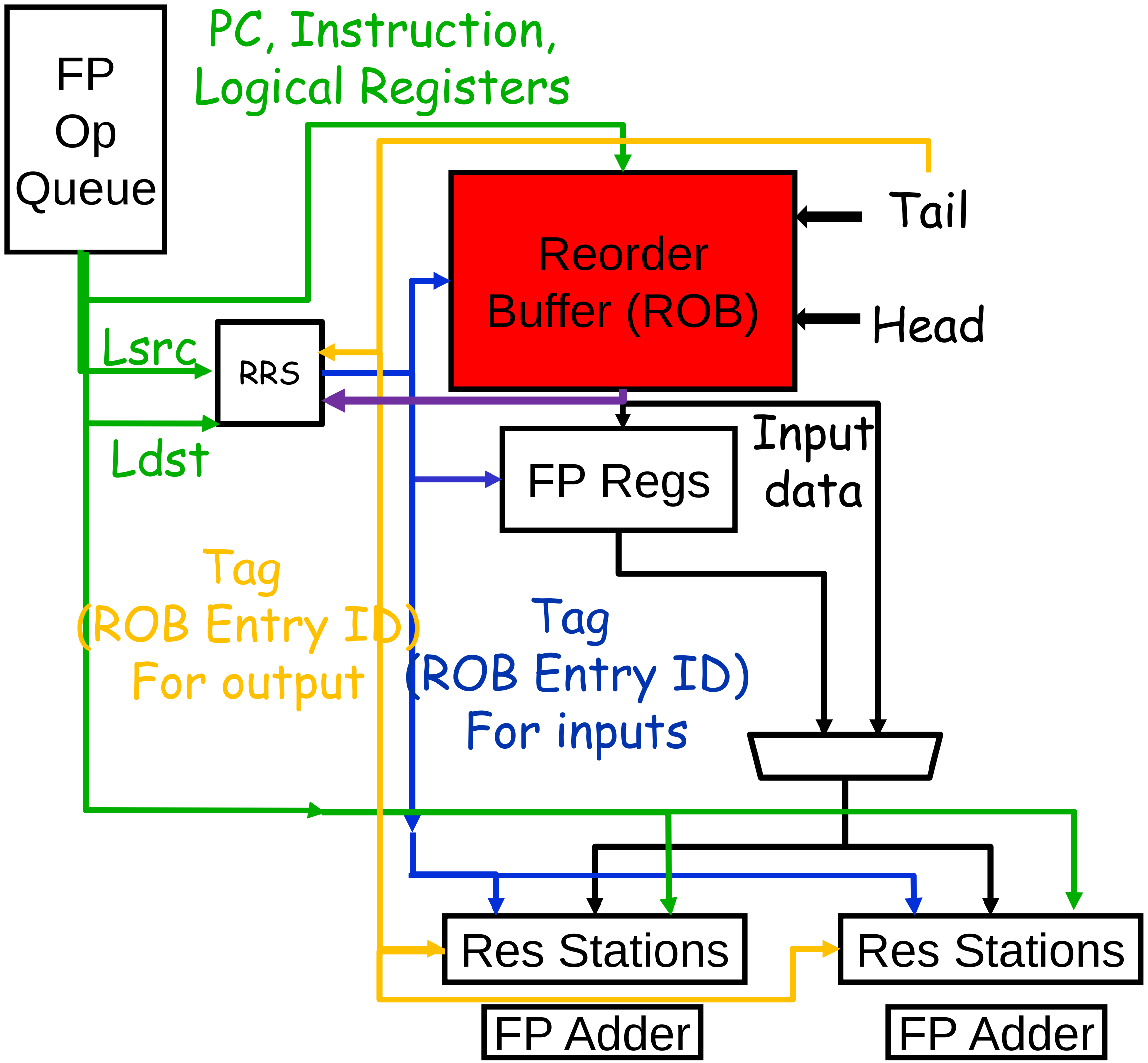
L1:
I1. lw r4,0(r8)
I2. add r4,r4,1 ⟸
I3. sw r4,0(r8)
I4. add r8,r8,4
I5. bne r8,r9,L1

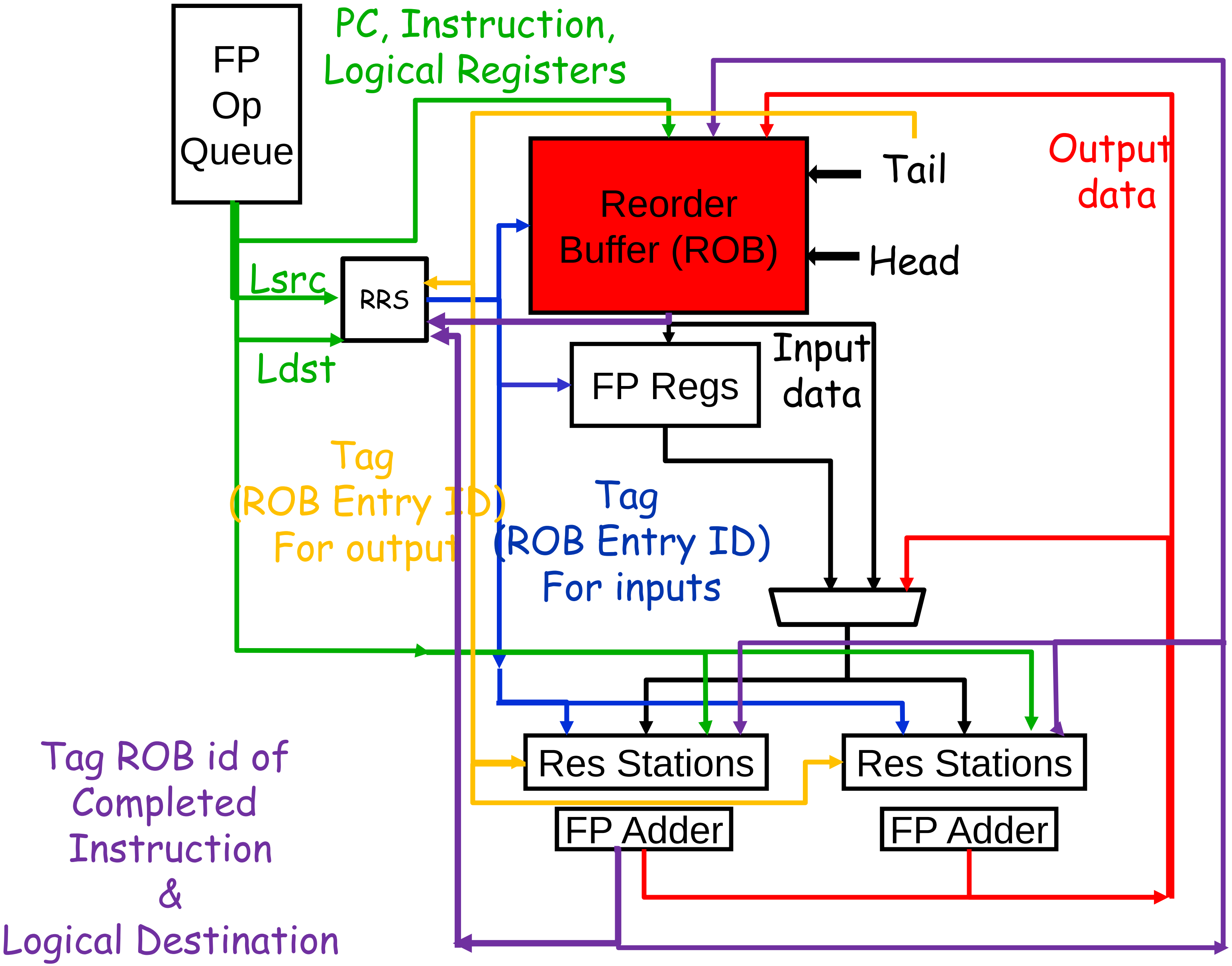


I1 committed
All registers up to date except r4, how to recover RRS?


FP Op Queue
RRS
Reorder Buffer (ROB)
Tail
Head
FP Regs
Res Stations
FP Adder
Res Stations
FP Adder


FP
Op
Queue
PC, Instruction,
Logical Registers
Reorder
Buffer (ROB)
Tail
Head
Lsrc
RRS
Ldst
FP Regs
Tag
(ROB Entry ID)
For output
Tag
(ROB Entry ID)
For inputs
Res Stations
Res Stations
FP Adder
FP Adder


FP
Op
Queue
PC, Instruction,
Logical Registers
Reorder
Buffer (ROB)
Tail
Head
Lsrc
RRS
Ldst
FP Regs
Input
data
Tag
(ROB Entry ID)
For output
Tag
(ROB Entry ID)
For inputs
Res Stations
Res Stations
FP Adder
FP Adder


FP Op Queue
PC, Instruction, Logical Registers
Reorder Buffer (ROB)
Tail
Head
Output data
Lsrc
RRS
Ldst
FP Regs
Input data
Tag (ROB Entry ID) For output
Tag (ROB Entry ID) For inputs
Res Stations
Res Stations
FP Adder
FP Adder
Tag ROB id of Completed Instruction & Logical Destination

Four Steps of Speculative Tomasulo Algorithm

1. **Issue**—get instruction from FP Op Queue
   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. **Execution**—operate on operands (EX)
   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. **Write result**—finish execution (WB)
   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. **Commit**—update register with reorder result
   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch/exception flushes reorder buffer (sometimes called "graduation")

- How do you find the latest version of a register?
  - use the register result status buffer to track which specific reorder buffer has received the value
- When do we release RS (queue entry)?
- How about RRS (in case of misprediction, wrong path effects)?
  - Commit all instructions until mispredicted branch (RRS up to date with no pending instructions)
- Mispredicted branch earlier than commit?

Relationship between precise interrupts and speculation:

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

Register renaming, physical registers versus Reorder Buffers

- Alternative to Reorder Buffer is a larger virtual set of registers and register renaming
- Physical registers hold both architecturally visible registers + temporary values
  - replace functions of reorder buffer and reservation stations
  - Allocate a physical register at renaming

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- 2-way Superscalar: process 2 instructions in parallel
  - Fetch 64-bits/clock cycle

Pipe Stages

IF	ID	EX	MEM	WB		
IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB
		IF	ID	EX	MEM	WB

Challenges of Instruction-Level Parallelism (ILP)

- Hardware more complicate
  - need multiple buses with associated matching logic at every reservation station.
  - need multiple forwarding paths
  - Need multiple of everything
  - we know how to handle this up to 8
- Where to find these instructions?
  - Basic Block (BB) ILP is quite small
  - BB: a straight-line code sequence with no branches in except to the entry and no branches out except at the exit
  - average dynamic branch frequency 15% to 25% => 4 to 7 instructions execute between a pair of branches
- Control Instructions
  - Need to fetch across basic blocks (i.e. Branch Prediction)
  - Need independent instructions (depends on program behavior)
- Memory Instructions another limitation
  - We don't know if they depend on each other until we know the address!

Control Speculation and Conditional Branch Predictors

Notes from Veerle Desmet

Conditional Branches

```
for (i=0; i<50; i++)
          {
      /* a loop... */
          }
/* next statements */
```

```
if (i > 0)
   /* something */
       else
 /* something else */
```

How frequent do conditional branches occur?

1/8

Pipelined architectures

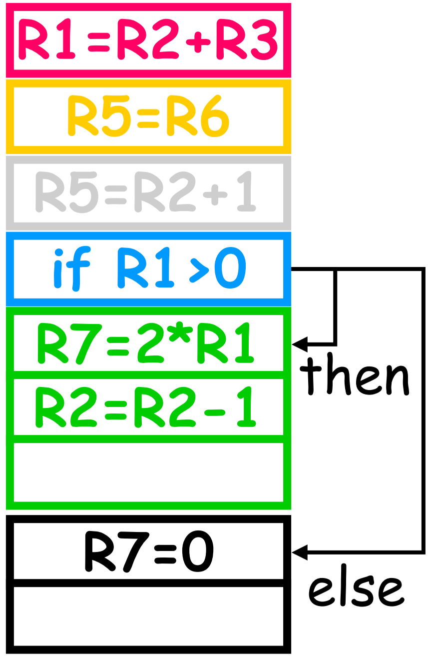
Parallel versus sequential:

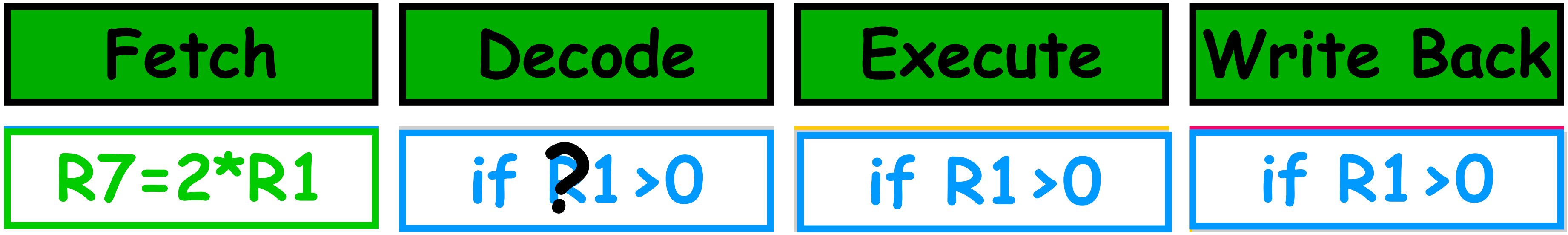
Fetch	Decode	Execute	Write Back
R1>0	R5=R6	R7=2*R1	R4=R3-1

- Constant flow of instructions possible
- Faster applications
- Limitation due to conditional branches

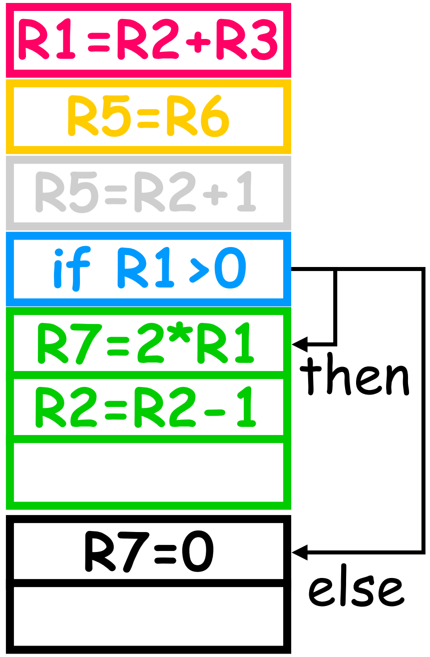
Problem: Branches

- Branches introduce bubbles
- Affects pipeline throughput

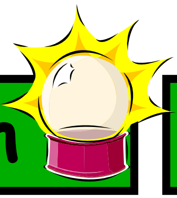




Solution: Prediction



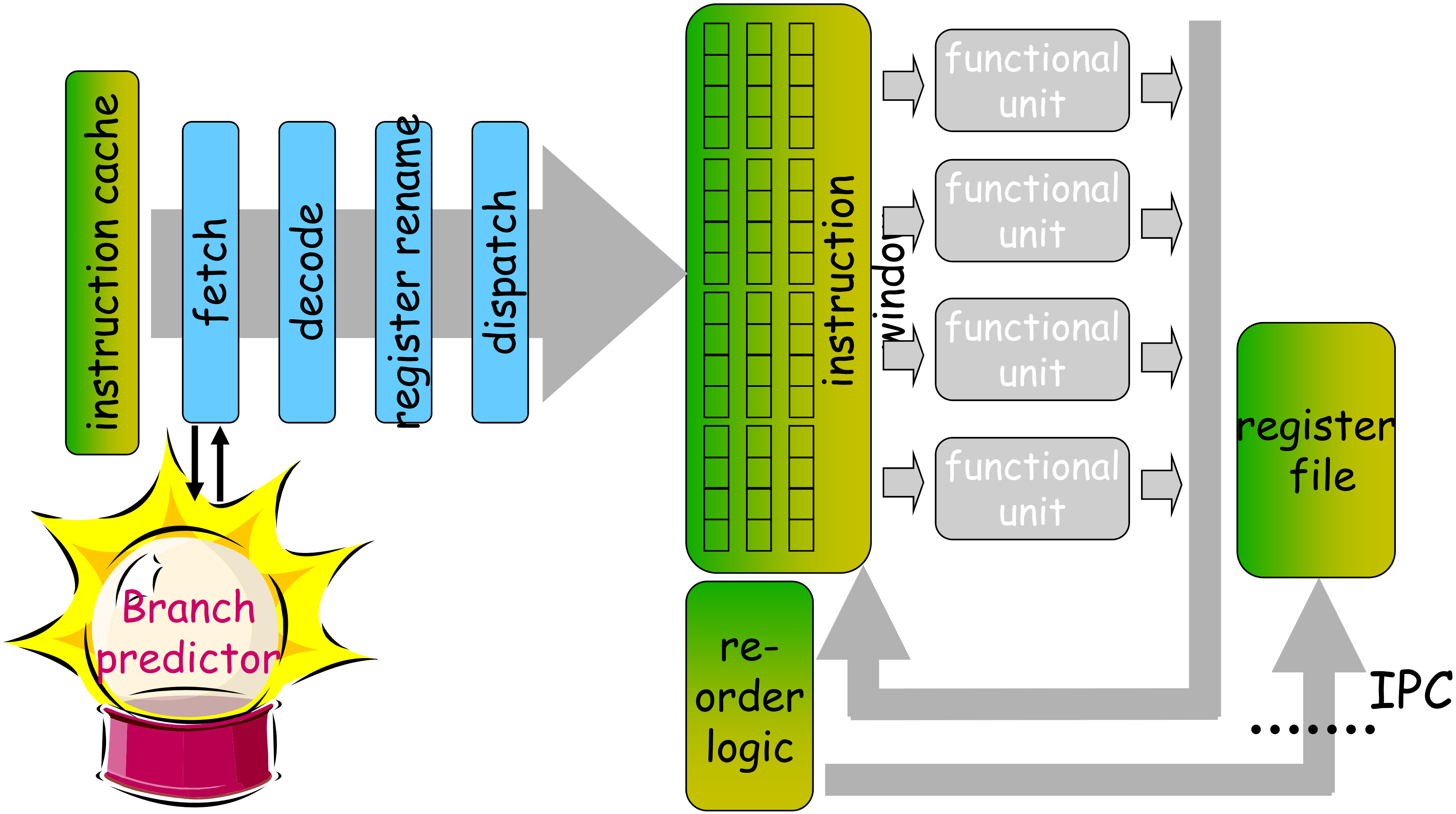
- Fetch those instructions that are likely to be executed



Fetch	Decode	Execute	Write Back
R2=R2-1	R7=2*R1	if R1>0	R5=R2+1

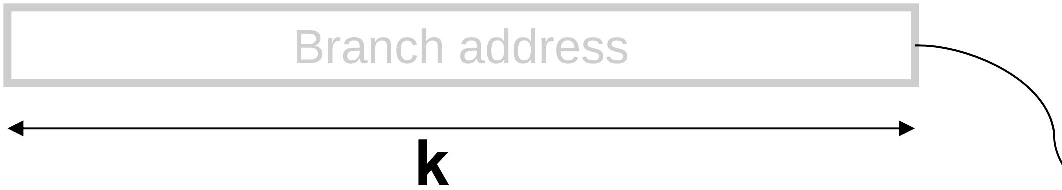
correct prediction = gain
misprediction = penalty

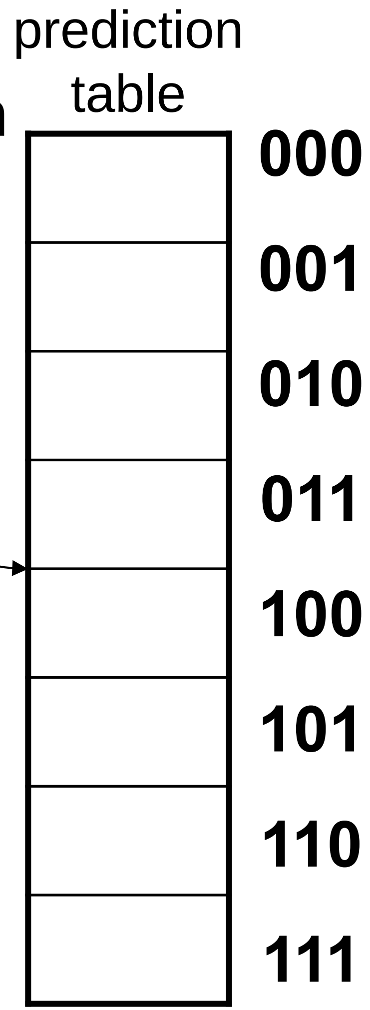
Nowaday's Architecture



Bimodal Branch Predictor

- Predict outcome of condition
  - eg if or else, taken or not taken
  - based on unique branch address

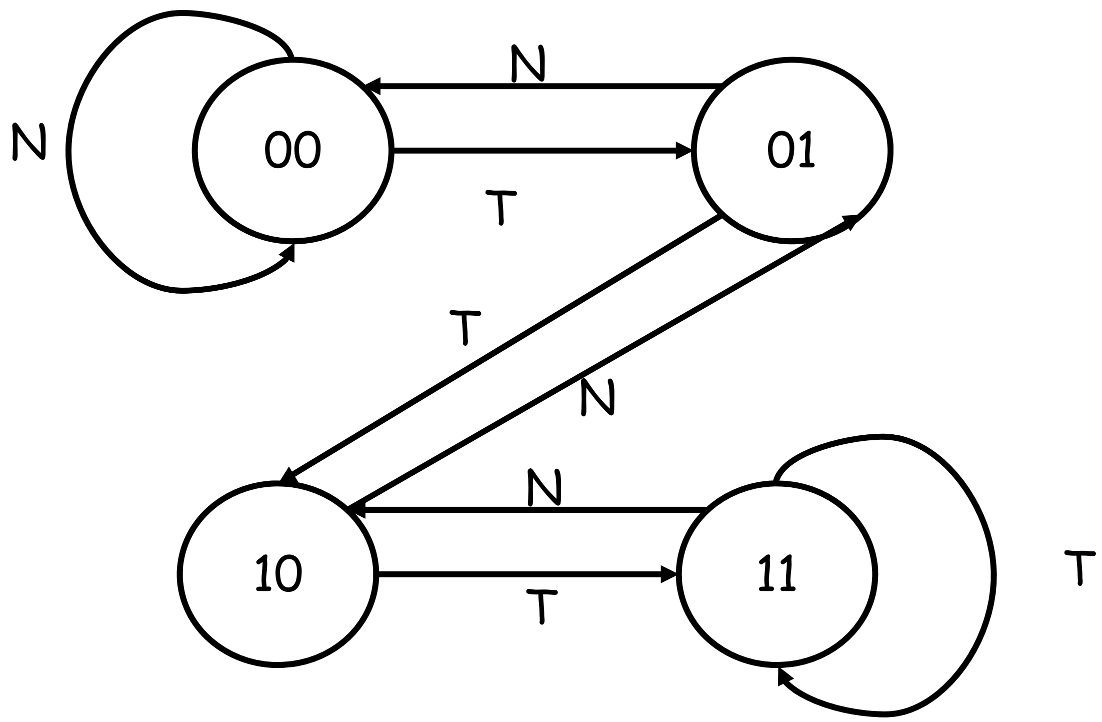




- Entry a 2-bit saturating counter

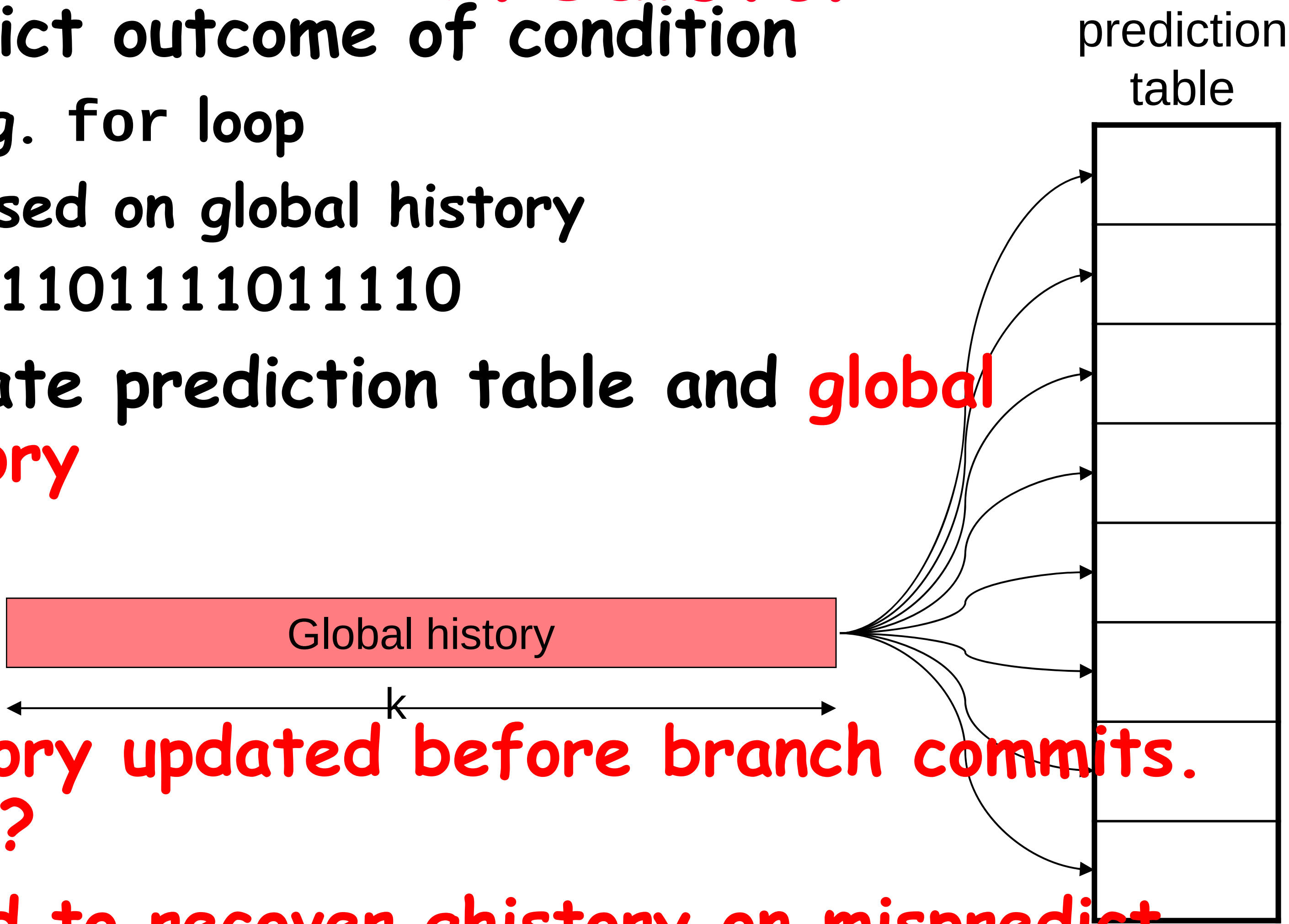
  ++ on taken OR -- on not taken
- Predict taken if counter > 1 else not taken
- Update prediction table when branch commits. What entry?

2-bit Saturating Counter



Global History Branch Predictor

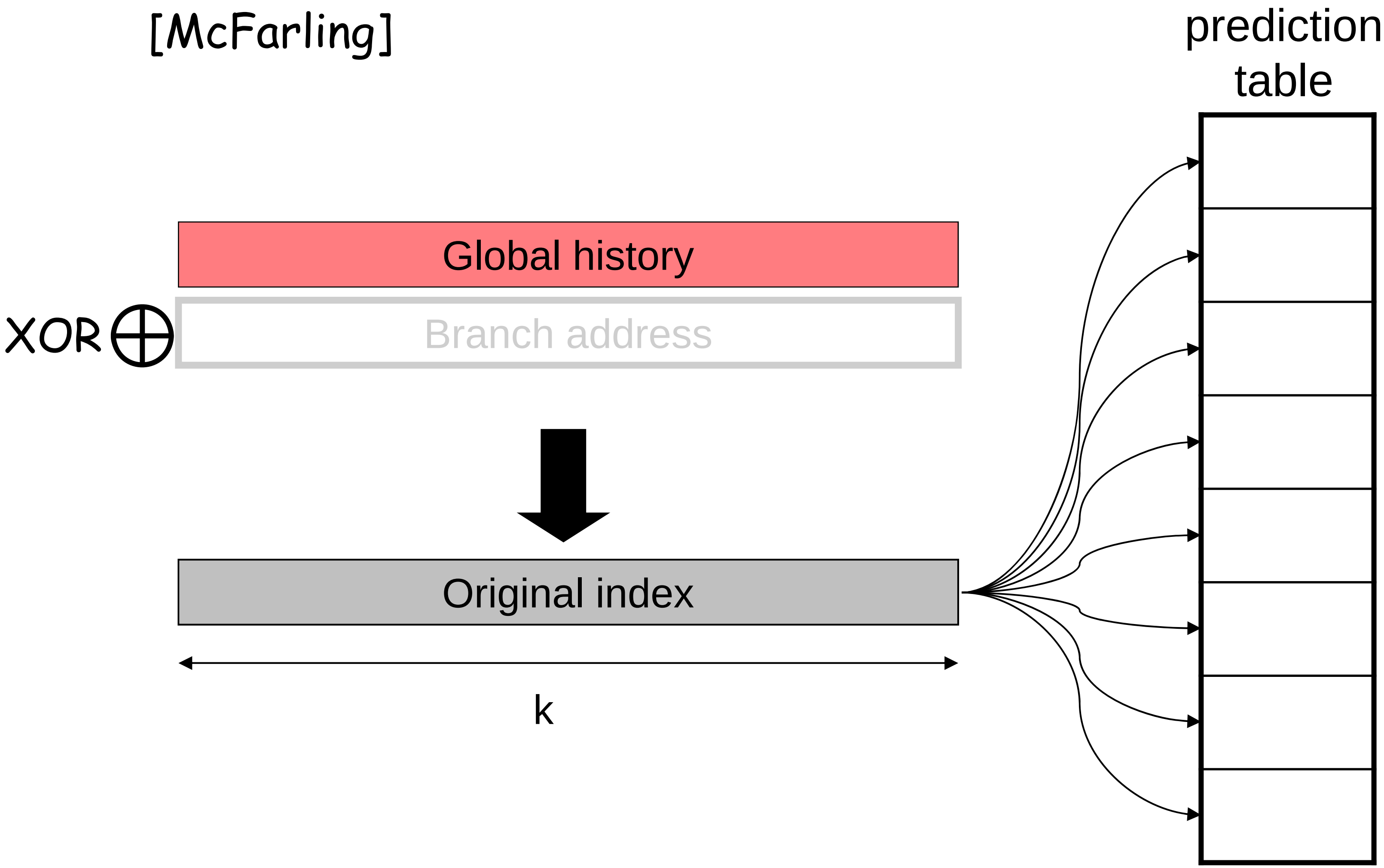
- Predict outcome of condition
  - e.g. for loop
  - based on global history
  - 11110111011110
- Update prediction table and global history



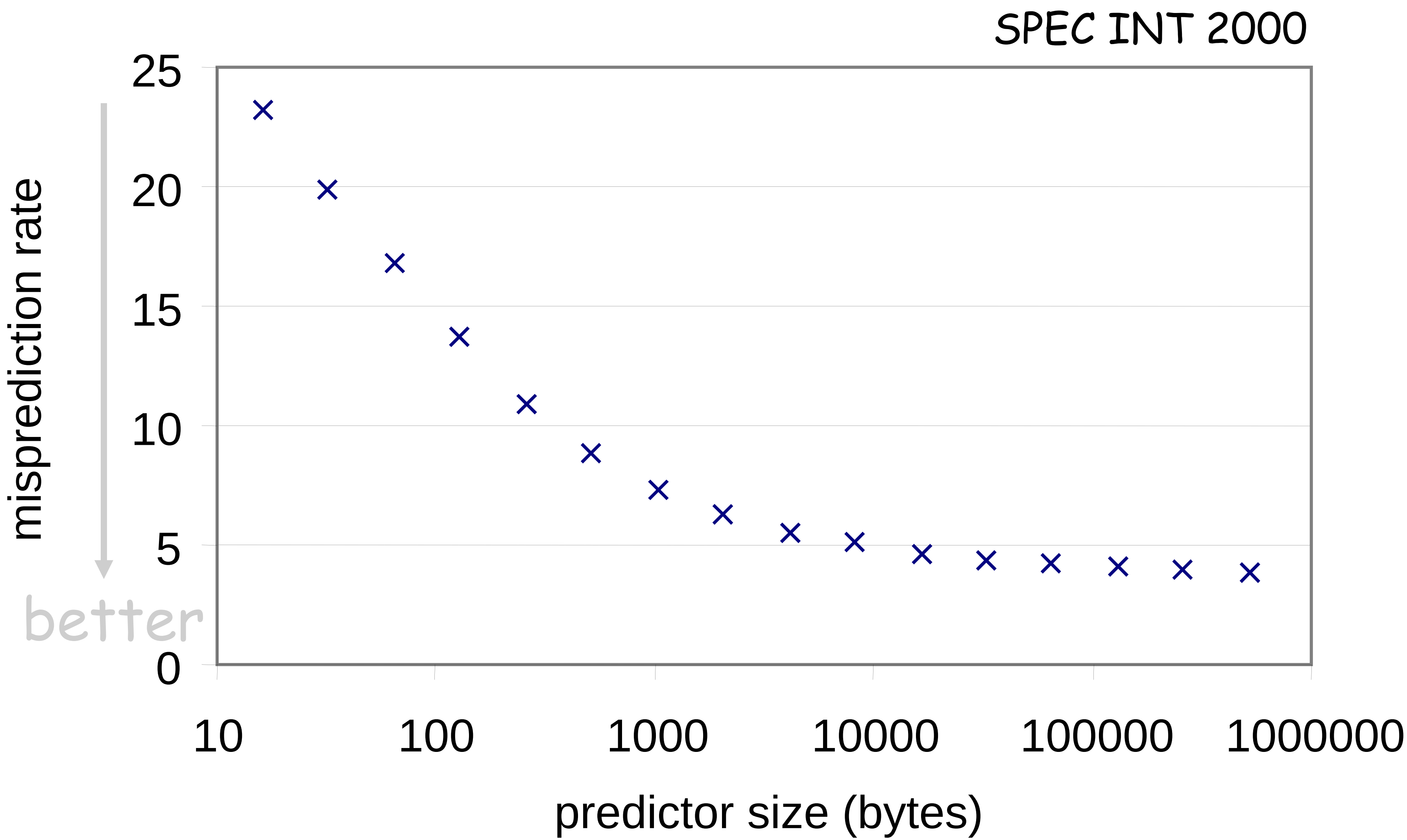
- History updated before branch commits. Why?
- Need to recover ghistory on mispredict

Gshare Branch Predictor

[McFarling]

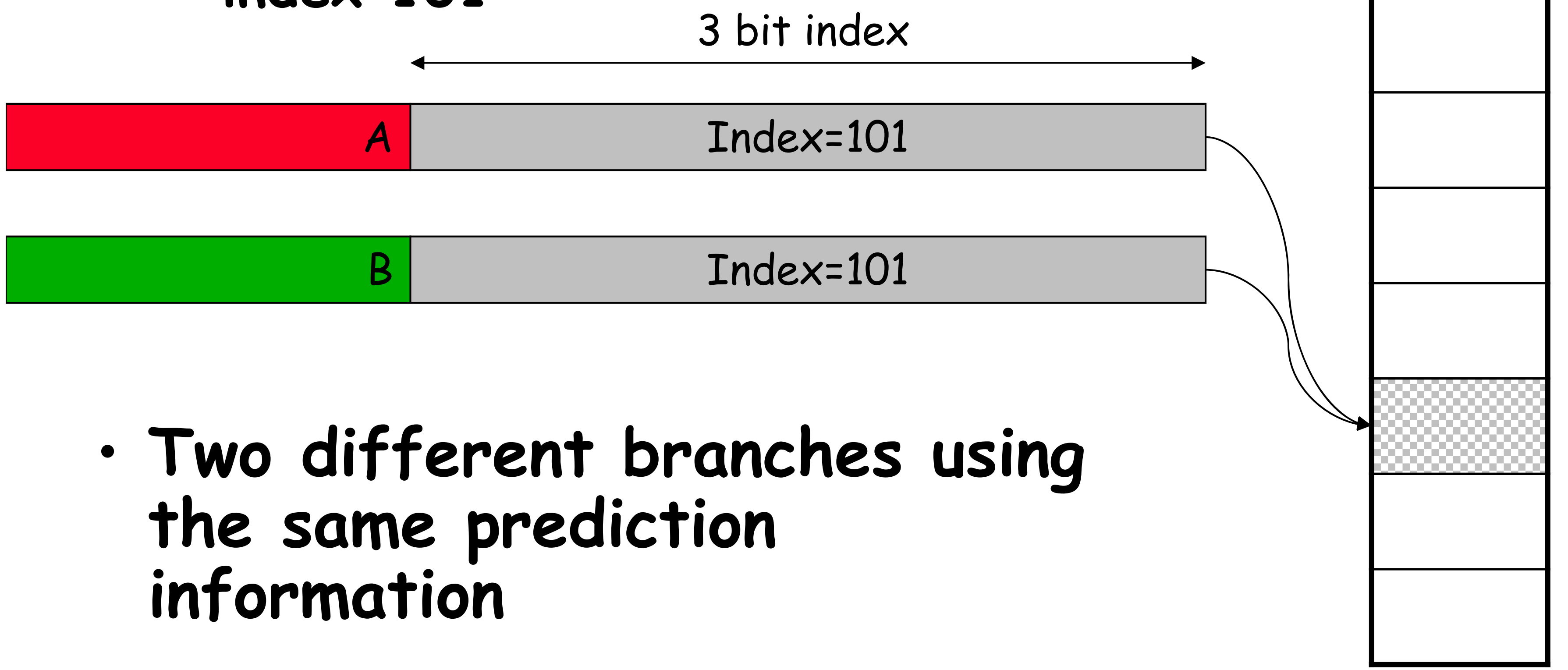


Misprediction rate: gshare



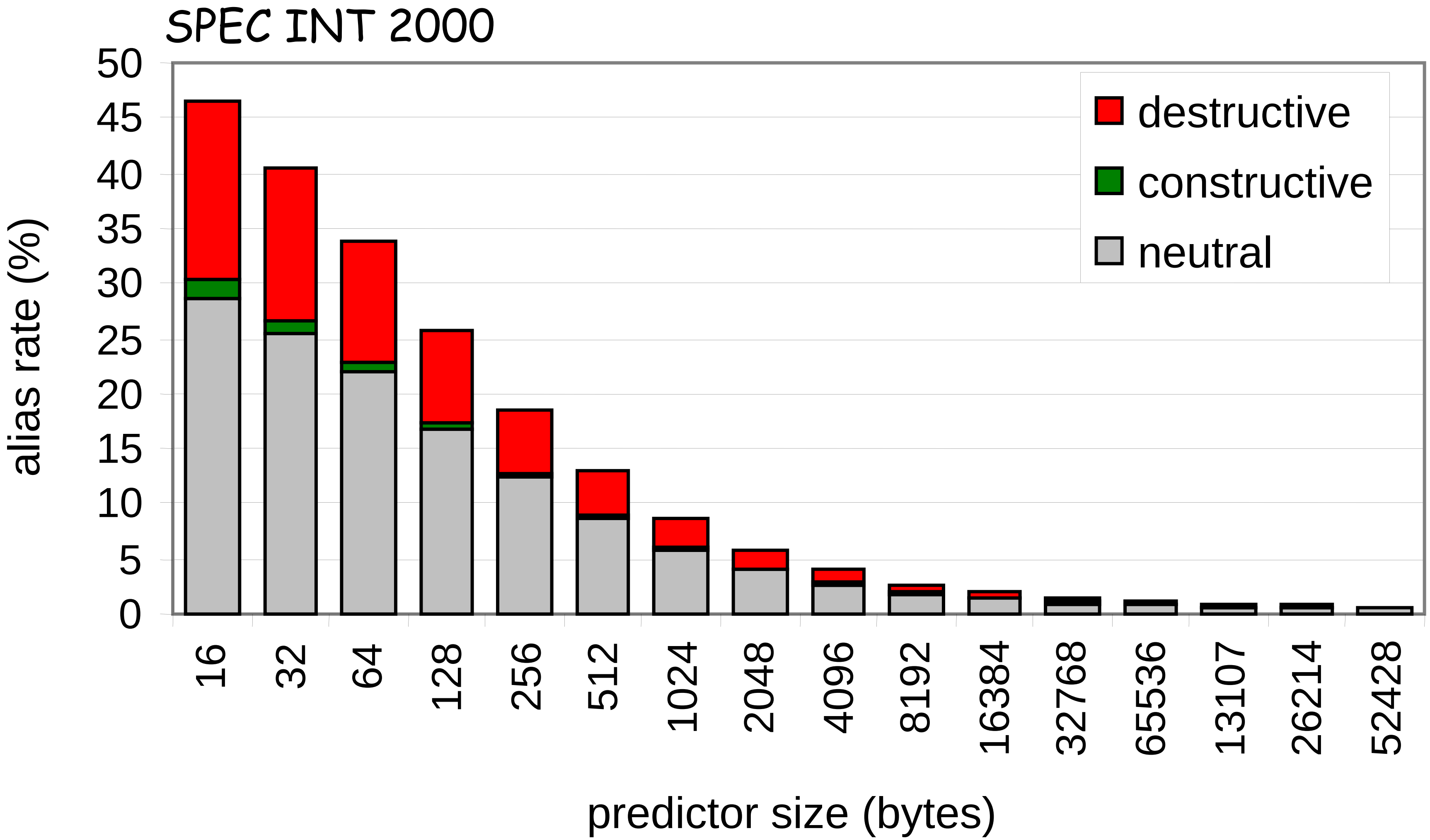
Aliasing

- Resource limitations:
  - 8 entries, index = 3 bits
  - index 101



- Two different branches using the same prediction information

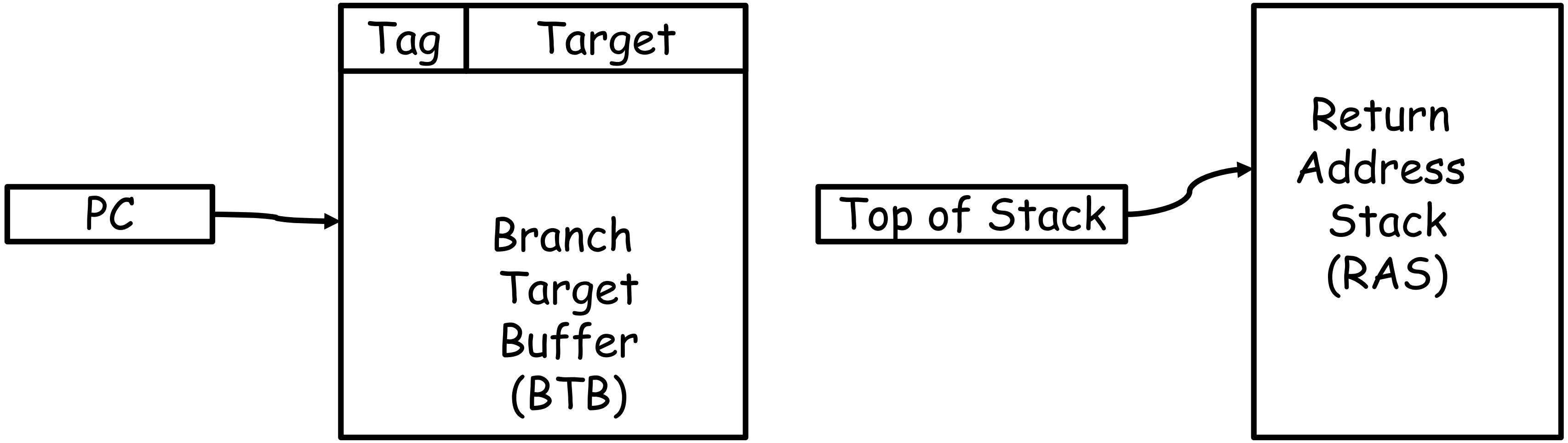
Aliasing


SPEC INT 2000
50
45
40
35
30
25
20
15
10
5
0
alias rate (%)
destructive
constructive
neutral
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
13107
26214
52428
predictor size (bytes)

Other Predictors found in processors

- Branch Target Buffer: target of taken branches
  - bnz r4, L1
  - what is the taken PC?
  - Predict helps not to wait to determine PC + L1
  - Indexed with PC
  - Update on a mispredict
- Return Address Stack: return address from function
  - ret (jr 31)
  - What address to return at?
  - It is stored in stack takes time to read. Predict helps to avoid waiting for r31
  - Push return address on calls
  - Predict: Pop on returns
  - Recover TOS on mispredict
- Indirect Jump Predictor: target of indirect jumps
  - jr R2
  - What is the address in R2?
  - Indexed similar to conditional (history based)
  - Update on mispredict at commit, history updated speculatively

Control Flow Predictors



Push return address on call
Pop on return

When an instruction branch is taken its target is saved in BTB
Next time to avoid having to wait for instruction to be fetched before we can determine its target we use BTB
If PC matches Tag use Target

Whether an instruction is a call or return and when is a call the return address are known at the fetch stage

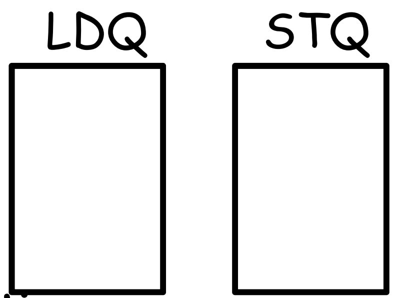
Every instruction checkpoints TOS and on mispredicts it recover TOS

Register Renaming with Merged Register File

Memory Dependences

st r3, 0(r4)
ld r2,64(r5)

- Ok to execute ld before store if 64+r5 != r4
- When do you know for sure that 64+r5 != r4?
  - Too late in the pipeline
- Memory Dependence Speculation
  - Predict whether or no dependence for a load
- Out of Order loads/stores might cause dependency violations in memory
  - If wrong squash
- Load/Store Queue is employed to:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
    - » Previous stores in program order
  - Store checks addresses in Load Queue to avoid WAR,WAW
    - » Subsequent loads in program order
  - Loads and stores commit when they are at the head of the ROB



Memory Dependence Predictor

- Table Indexed with the Pc of a load
- Each entry 1-bit
- Prediction read table
  - 1 speculate
  - 0 wait for previous stores
- Update:
  - If speculate and wrong set bit to 0
  - Every so many cycles reset all predictor bits to 1
- More advanced technique: memory renaming
  - Renamed Ldst to a different register (actually physical registers)

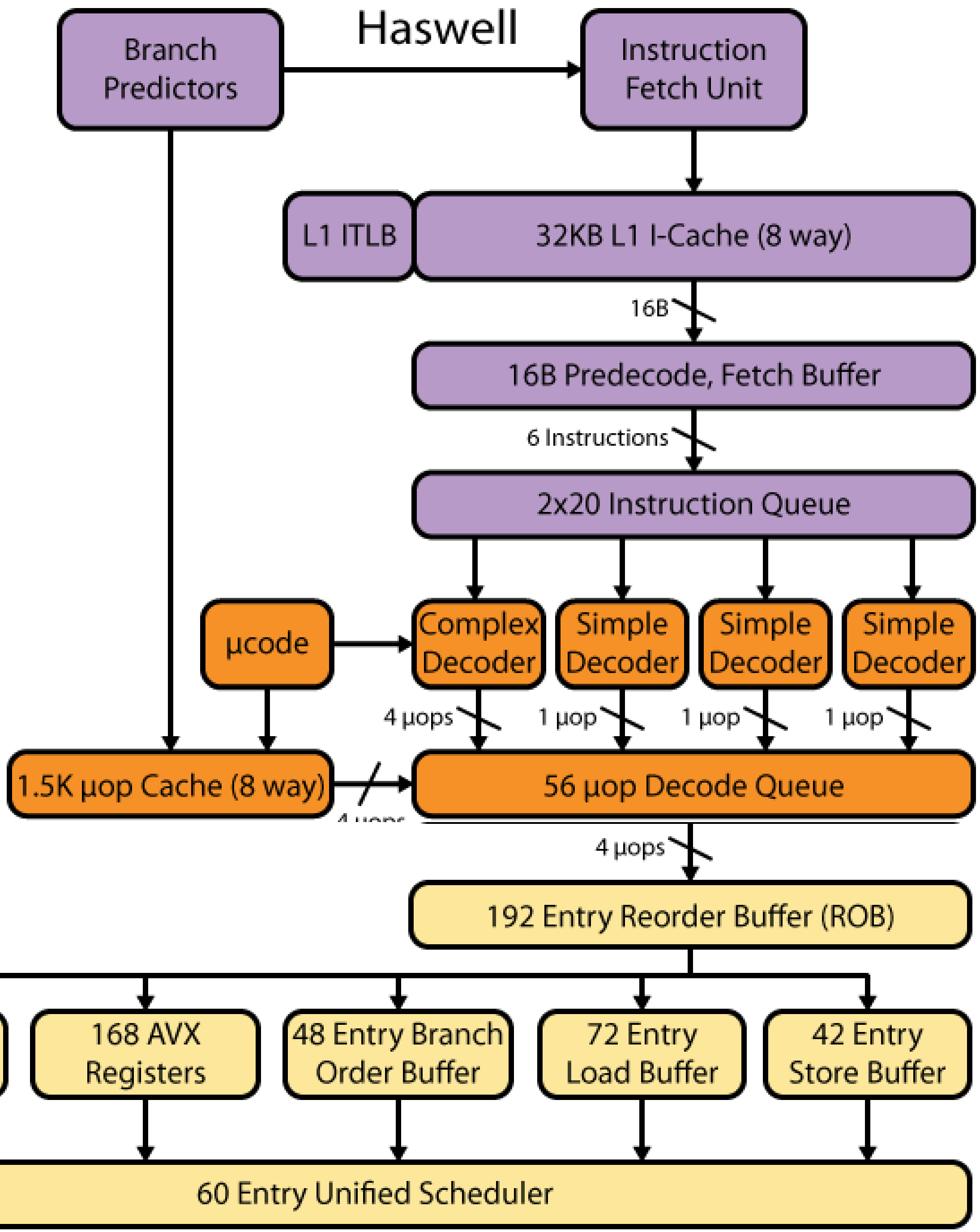
```
add r4,r5,1
stw r4,0(r8)
Lw  r3,4(r10)
bnz r3,L1
```

⇒

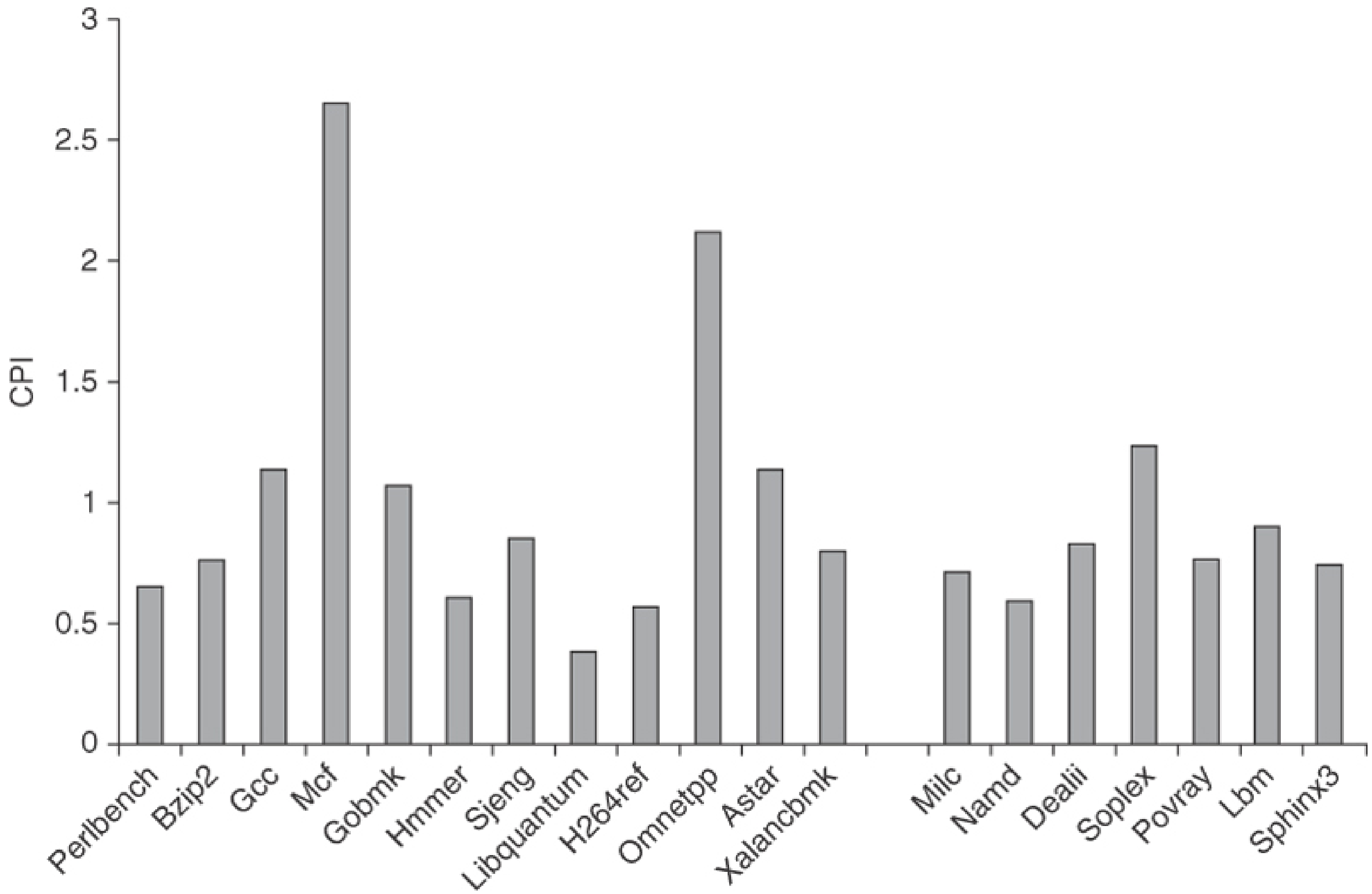
```
add r4,r5,1
stw r4,0(r8)
Lw  r3,4(r10)
bnz r4,L1
```

- Generalize to all instructions (to delta?) – (Project)

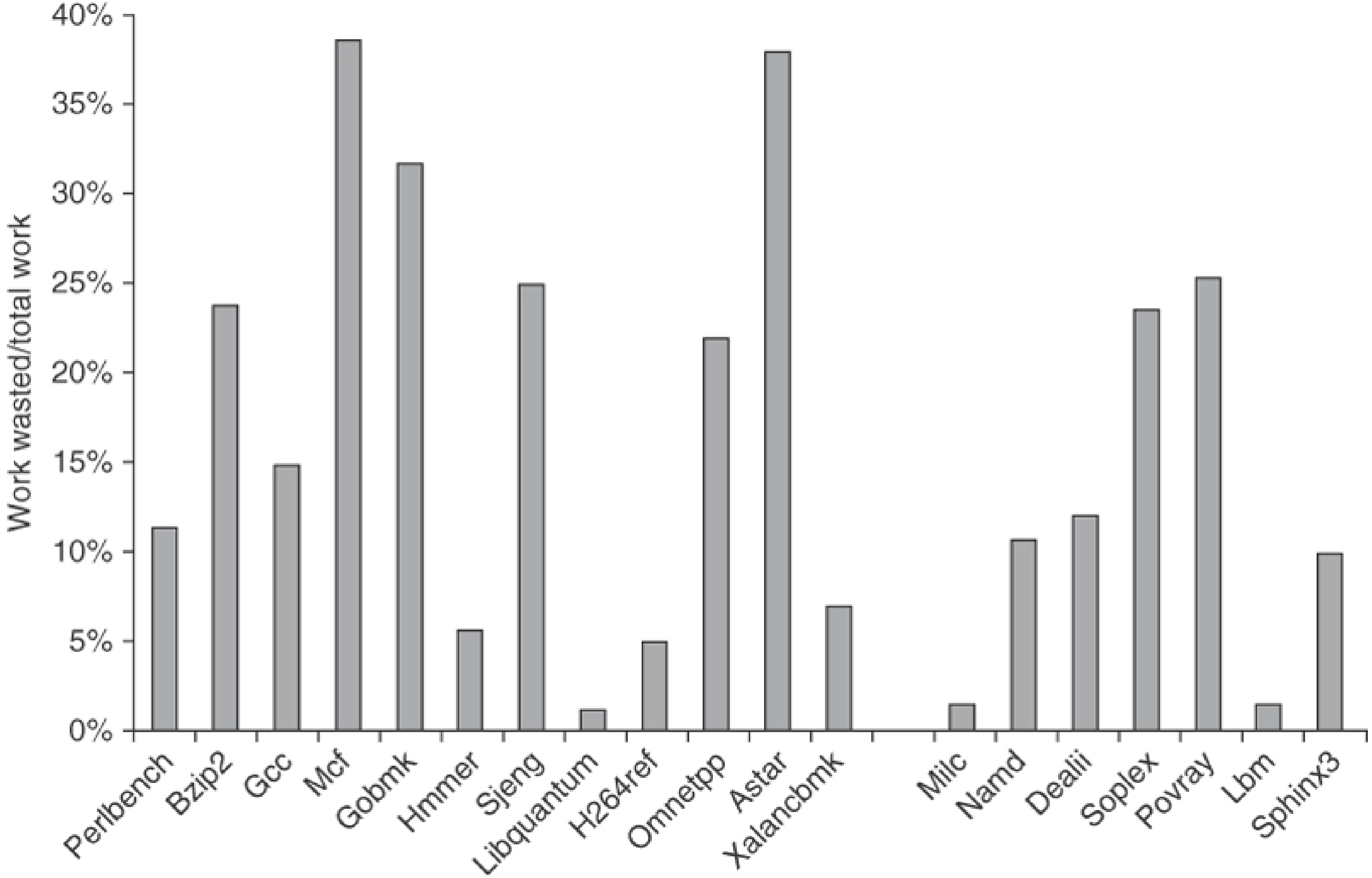
Intel ooo core



CPI for SPEC CPU2006 (CPI<1)

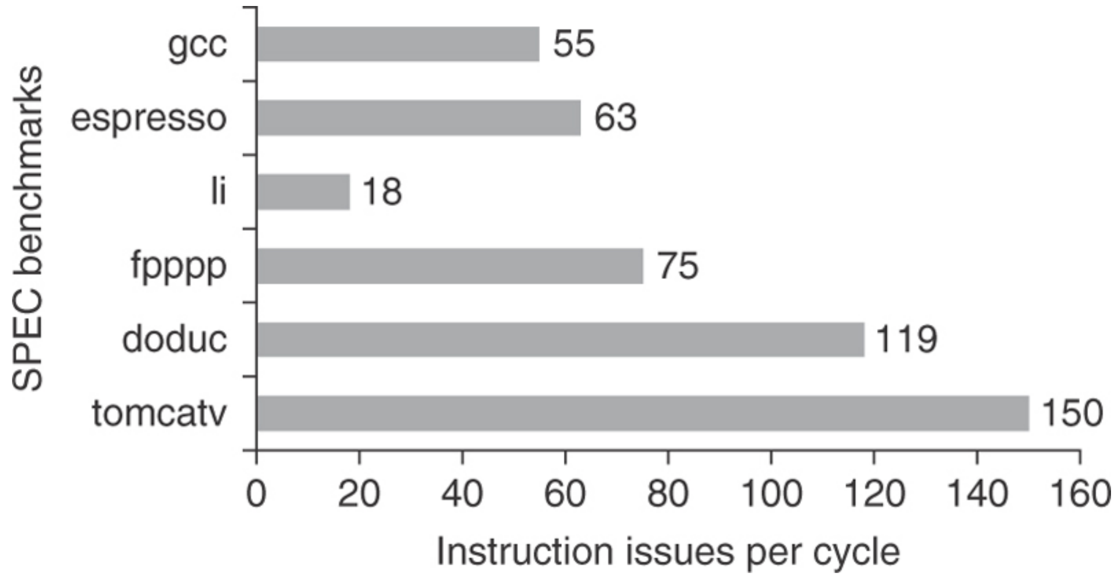


Wasted Work (due to mispredictions)



Ultimate Limits of ILP

- Infinite fetch, perfect branch prediction, memory dependence prediction, registers, functional units.
- How much ILP?



- ILP with realistic constraints more limited
- Alternative to ILP?

Projects

1. Determine performance of state of the art branch predictor
   1. Analyze where it mispredicts
2. Defective branch predictor performance
3. Machine Learning based Prediction
4. Opportunity for using a 3-bit counter as 1-bit, 2-bit or 3-bit – (set dueling)
5. How often no pending stores in a page (track them at TLB)
   1. (write-read analysis)
   2. (write-write analysis)
   3. Fine vs coarse check
6. How often all lines of page in IL1, DL1, L2
7. How many accesses in a set/page between evictions? Regular?
8. Distance between write-miss and read? (selective write-allocate/no-write-allocate)
9. Prefetching based on Deterministic Stream (not speculative)
10. Register Renaming for Delta Reuse
11. Tagged Memory

1
2
3
4
5
6
7
8
9
10
11 Παύλος