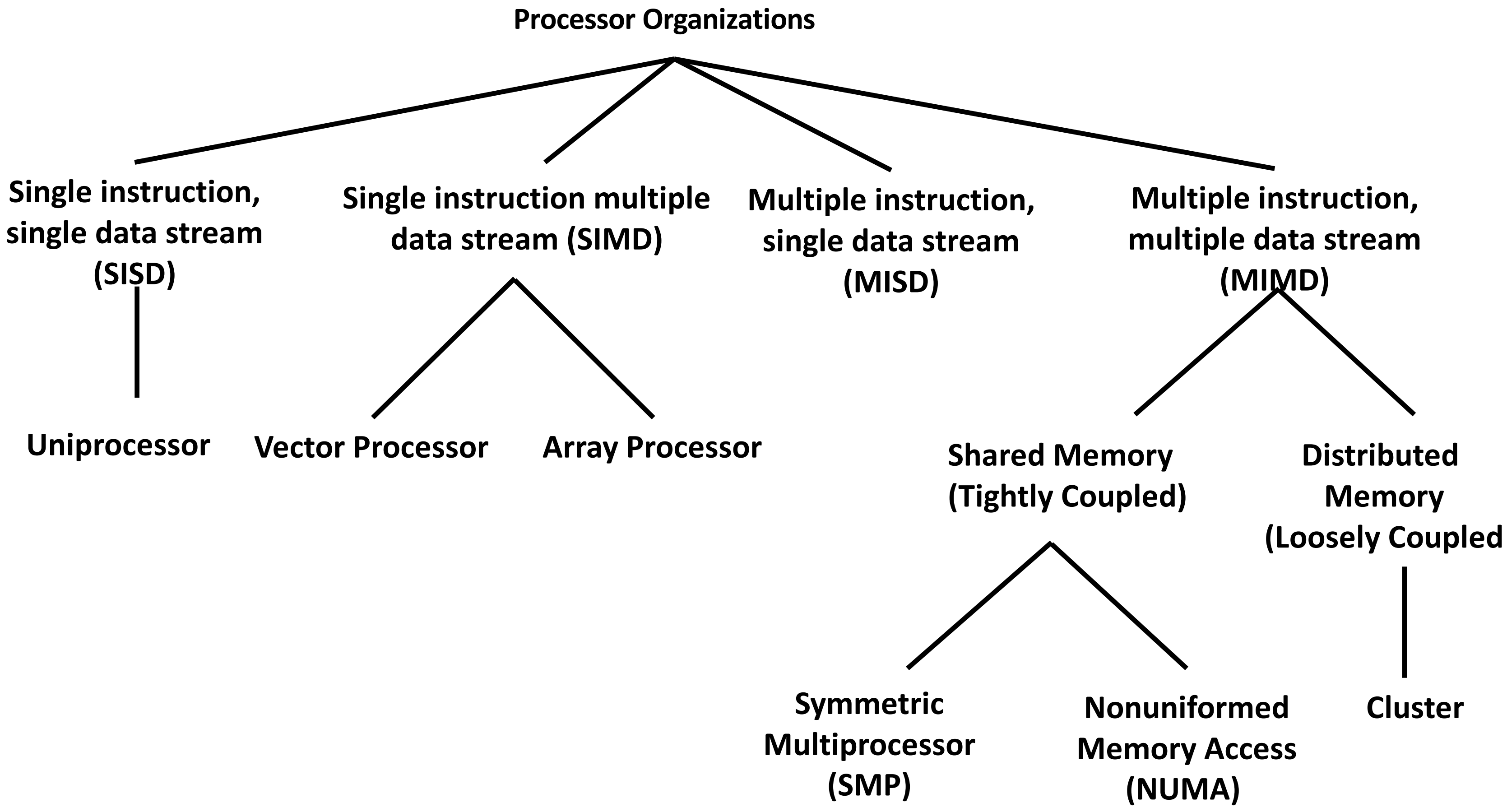
CSE 520
Computer Architecture II

Complex Pipelining: Superscalar

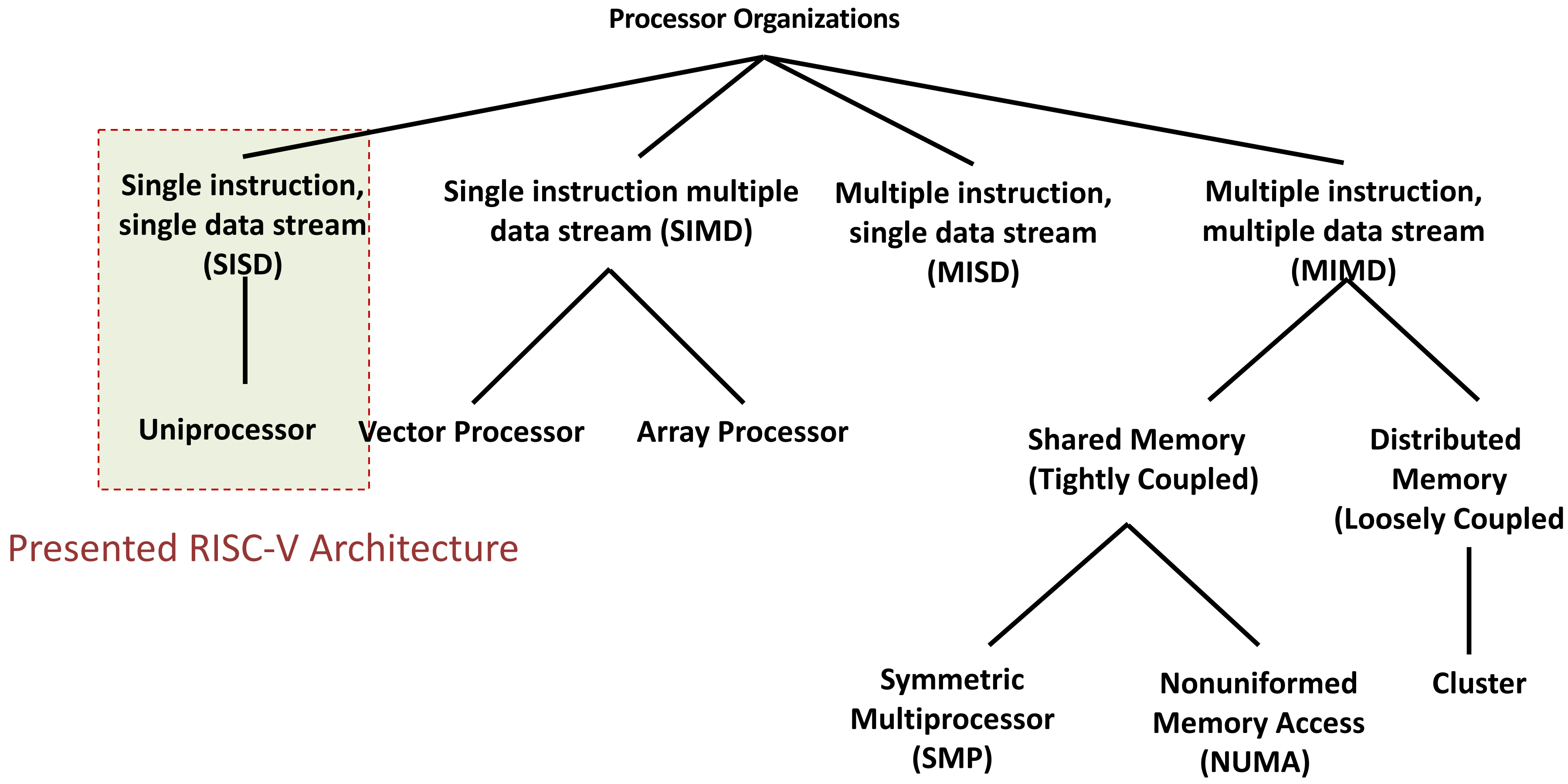
Prof. Michel A. Kinsy

Architecture Taxonomy



Parallelism Paradigms: Instruction level, Data level and Task level Parallelisms

Architecture Taxonomy



Parallelism Paradigms: Instruction level, Data level and Task level Parallelisms

Performance Driven Design

- Pipelining was introduced to improve performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Additional techniques for improvement
  - Duplication of resources
  - Out of order issue hardware
  - Windowing to decouple execution from decode
  - Register Renaming capability

Performance Driven Design

- Pipelining was introduced to improve performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Pipelining becomes complex when we want high performance in the presence of:
  - Long latency or partially pipelined floating-point units
  - Multiple function and memory units
  - Memory systems with variable access time

Superscalar Architectures

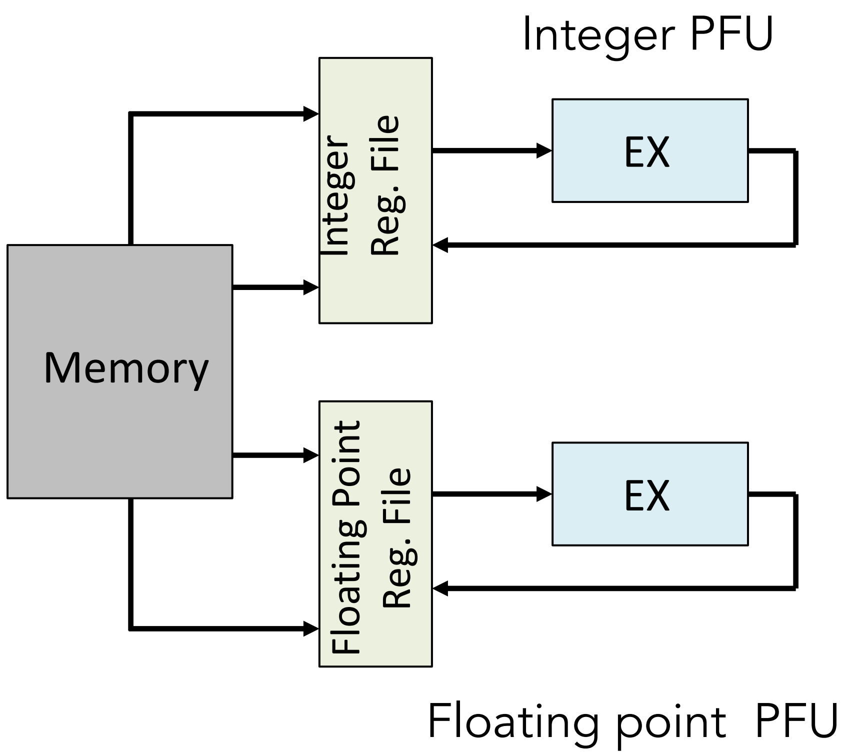
- The term **superscalar** was first coined in 1987
  - It refers to a machine that is designed to improve the performance of the execution of scalar instructions
  - In most applications, the bulk of the operations are on scalar quantities
- The superscalar approach represents the next step in the evolution of high-performance general-purpose processors
  - The essence of the superscalar approach is the ability to execute instructions independently and concurrently in different pipelines
  - The concept can be further exploited by allowing instructions to be executed in an order different from the program order

Superscalar Implementation

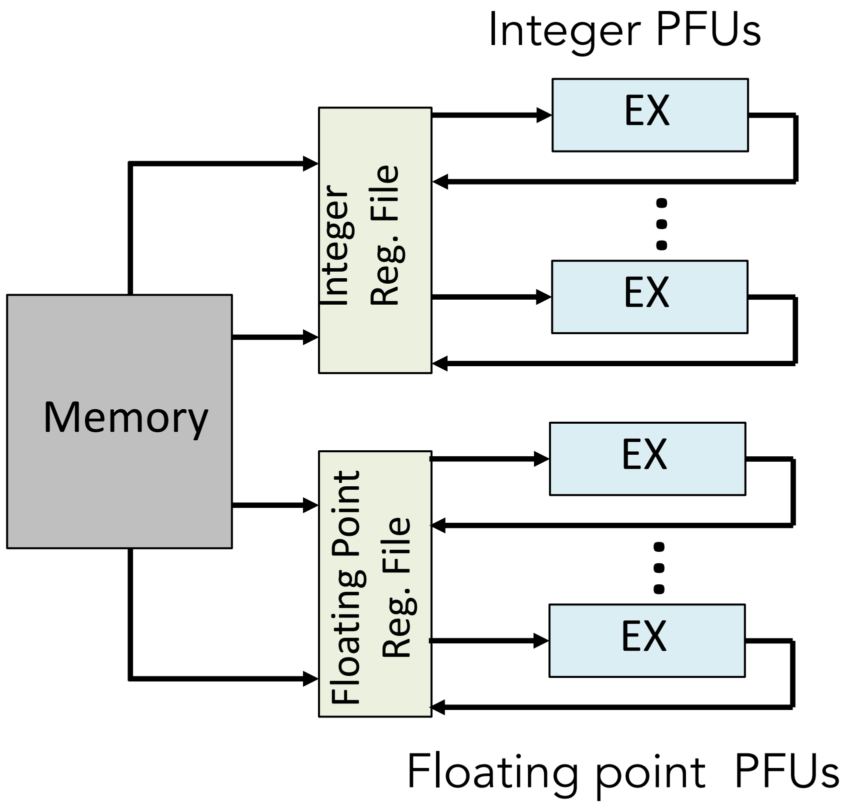
- Simultaneously fetch multiple instructions
  - Superscalar architecture exploits ILP (Instruction Level Parallelism)
- Logic to determine true dependencies involving register values
- Mechanisms to communicate these values
- Mechanisms to initiate multiple instructions in parallel
- Resources for parallel execution of multiple instructions
- Mechanisms for committing process state in correct order

Scalar vs. Superscalar

- Pipelined Functional Unit (PFU)



Scalar Architecture

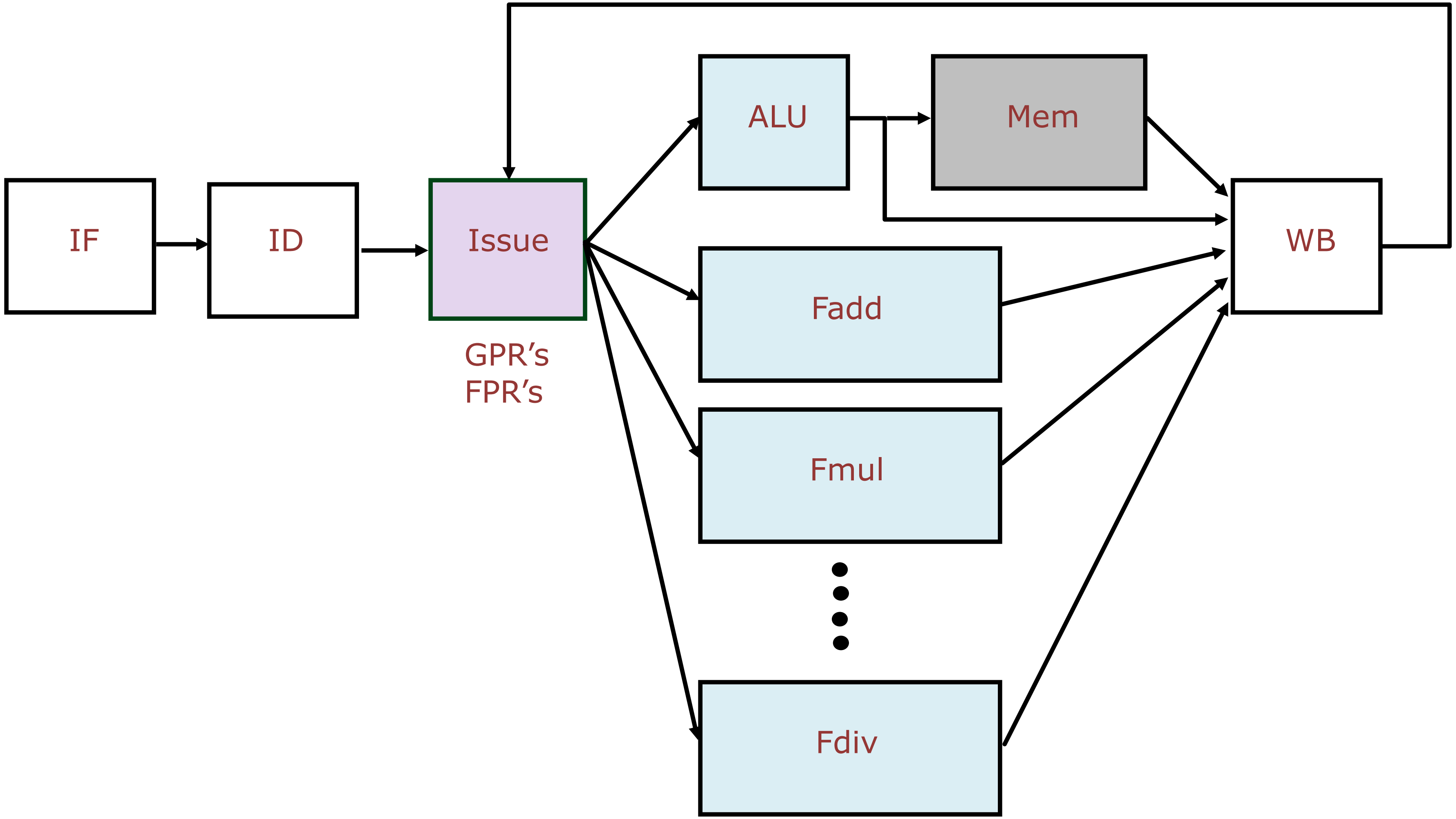


Superscalar Architecture

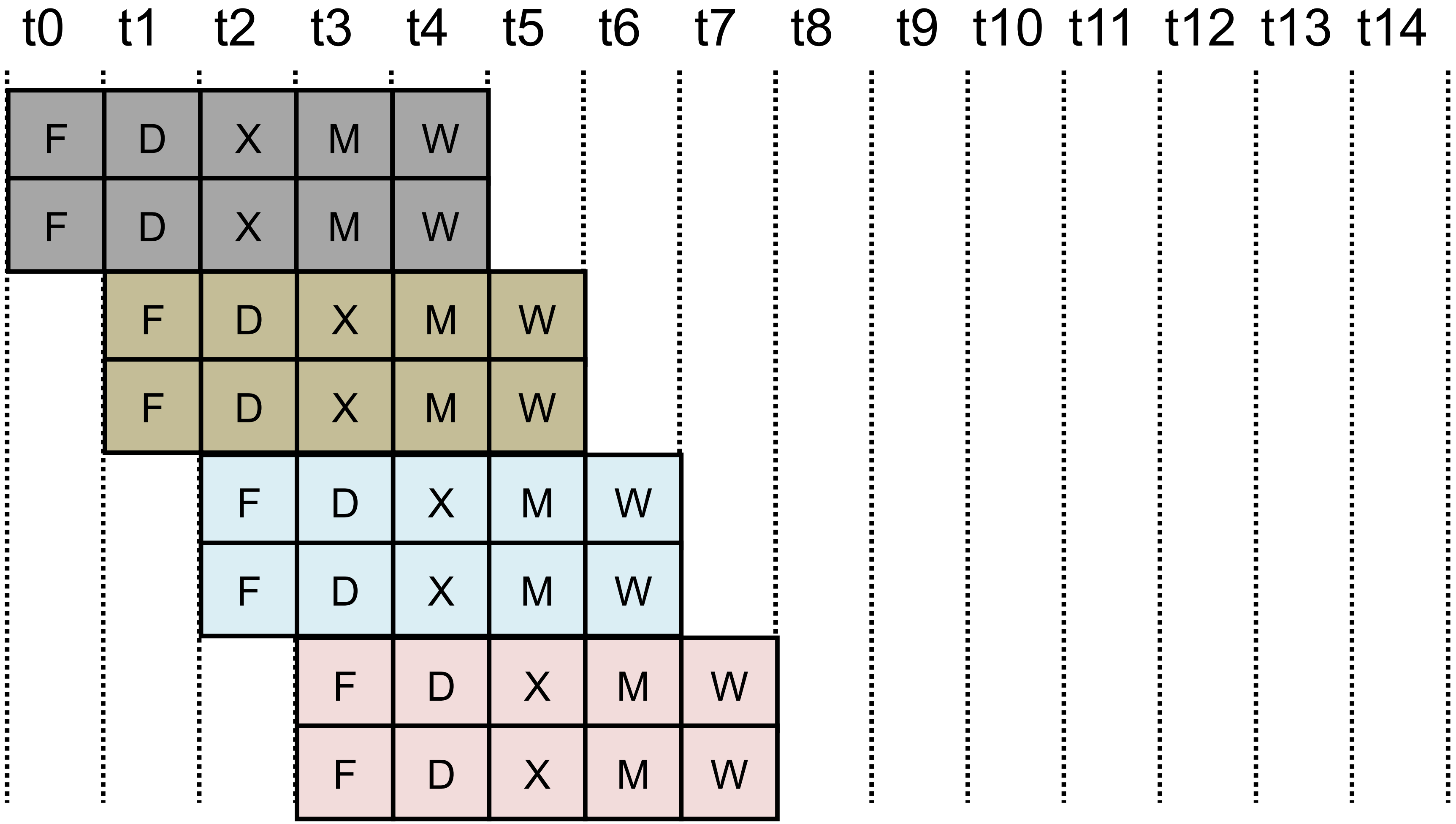
Instruction Level Parallelism

- ILP refers to the degree to which the instructions can be executed parallel
- To achieve it:
  - Compiler based optimization
  - Hardware techniques
- Limited by
  - Data dependency
  - Procedural dependency
  - Resource conflicts

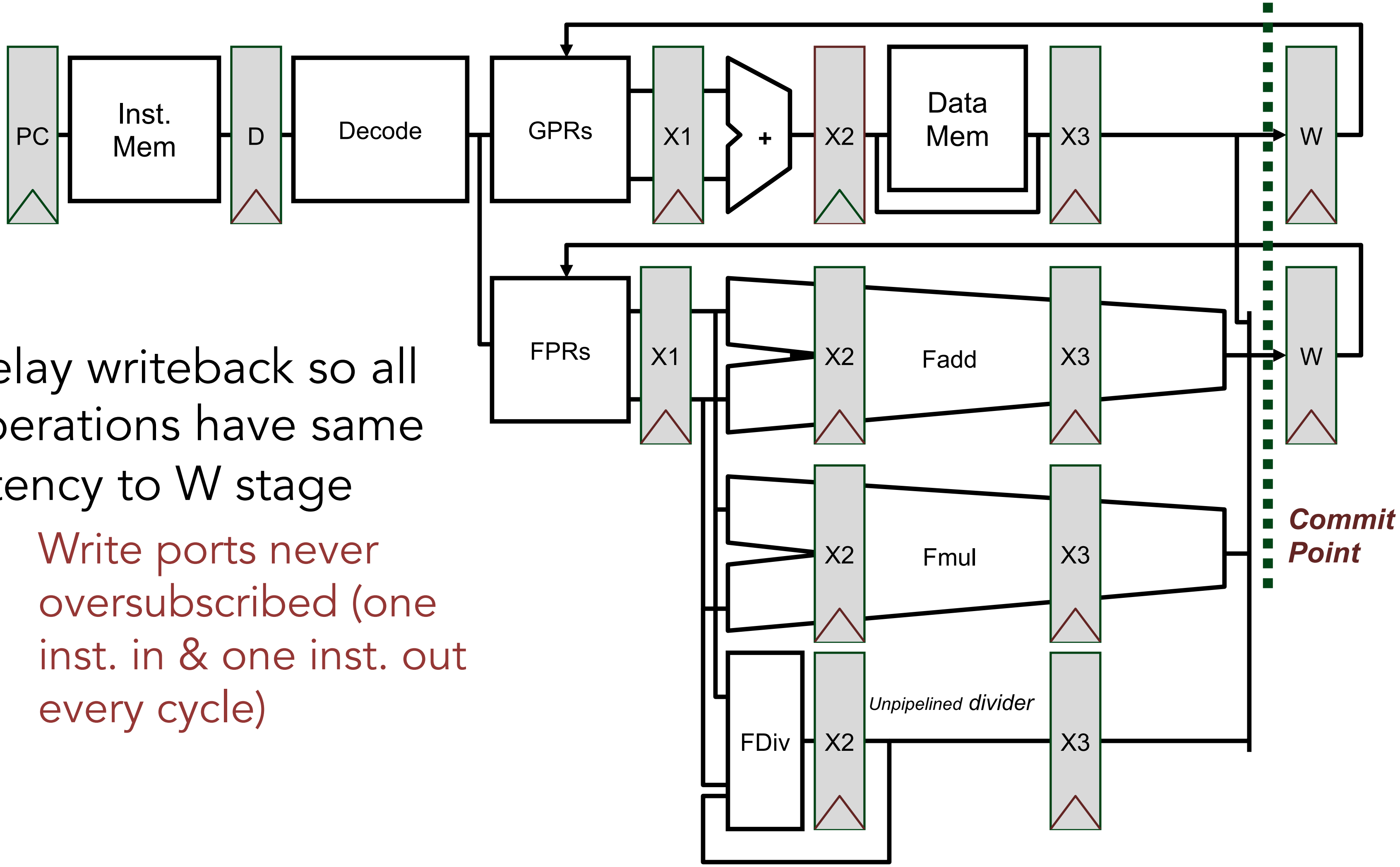
Complex Pipeline Structure



Superscalar Execution

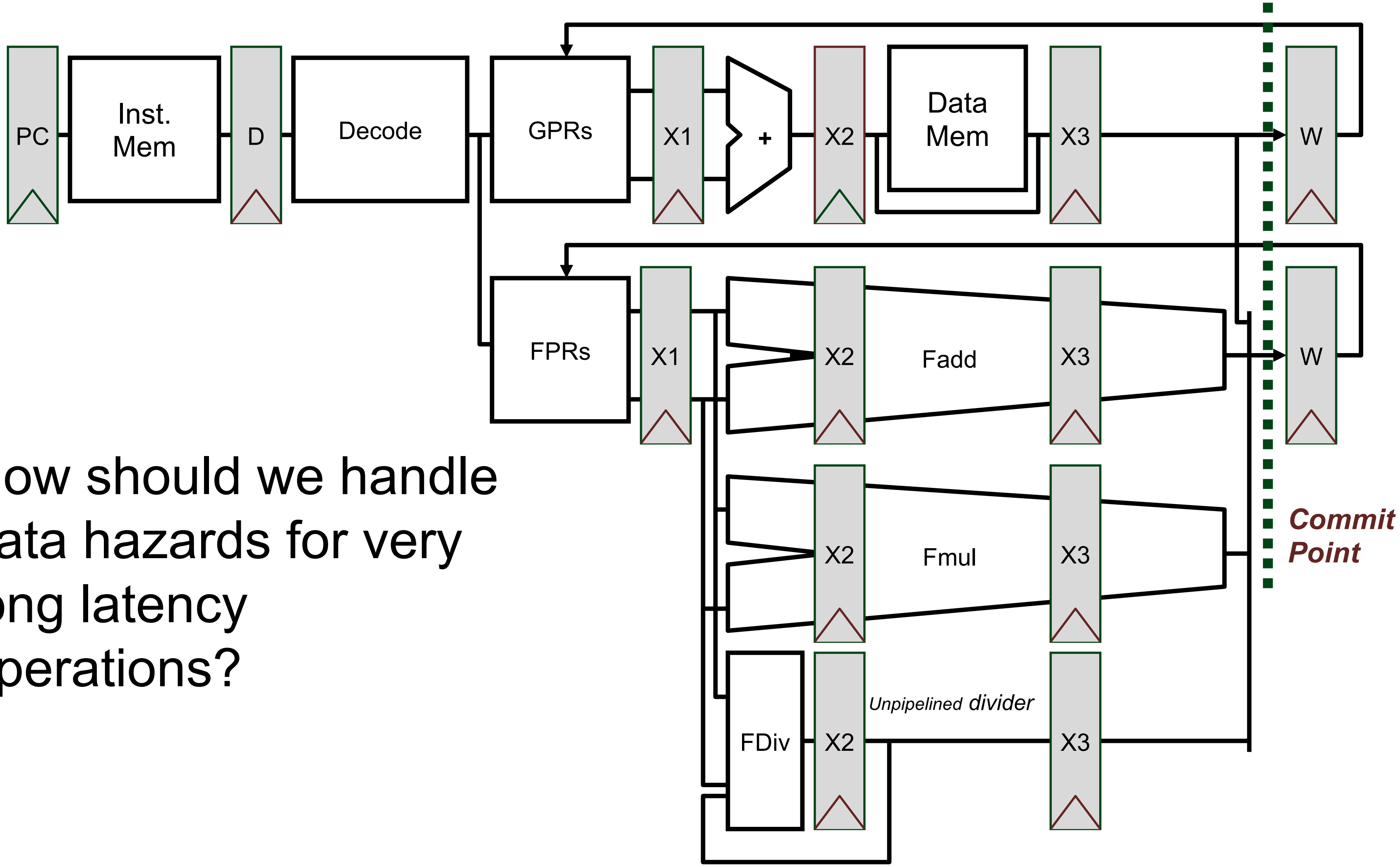


Complex In-Order Pipeline



- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)

Complex In-Order Pipeline



- How should we handle data hazards for very long latency operations?

Superscalar In-Order Pipeline

- Fetch two instructions per cycle; issue both simultaneously if one is integer/memory and other is floating-point (dependences?)
- Inexpensive way of increasing throughput
  - Alpha 21064 (1992) & MIPS R5000 series (1996)
- The idea can be extended to wider issue but register file ports and bypassing costs grow quickly
  - Example 4-issue UltraSPARC

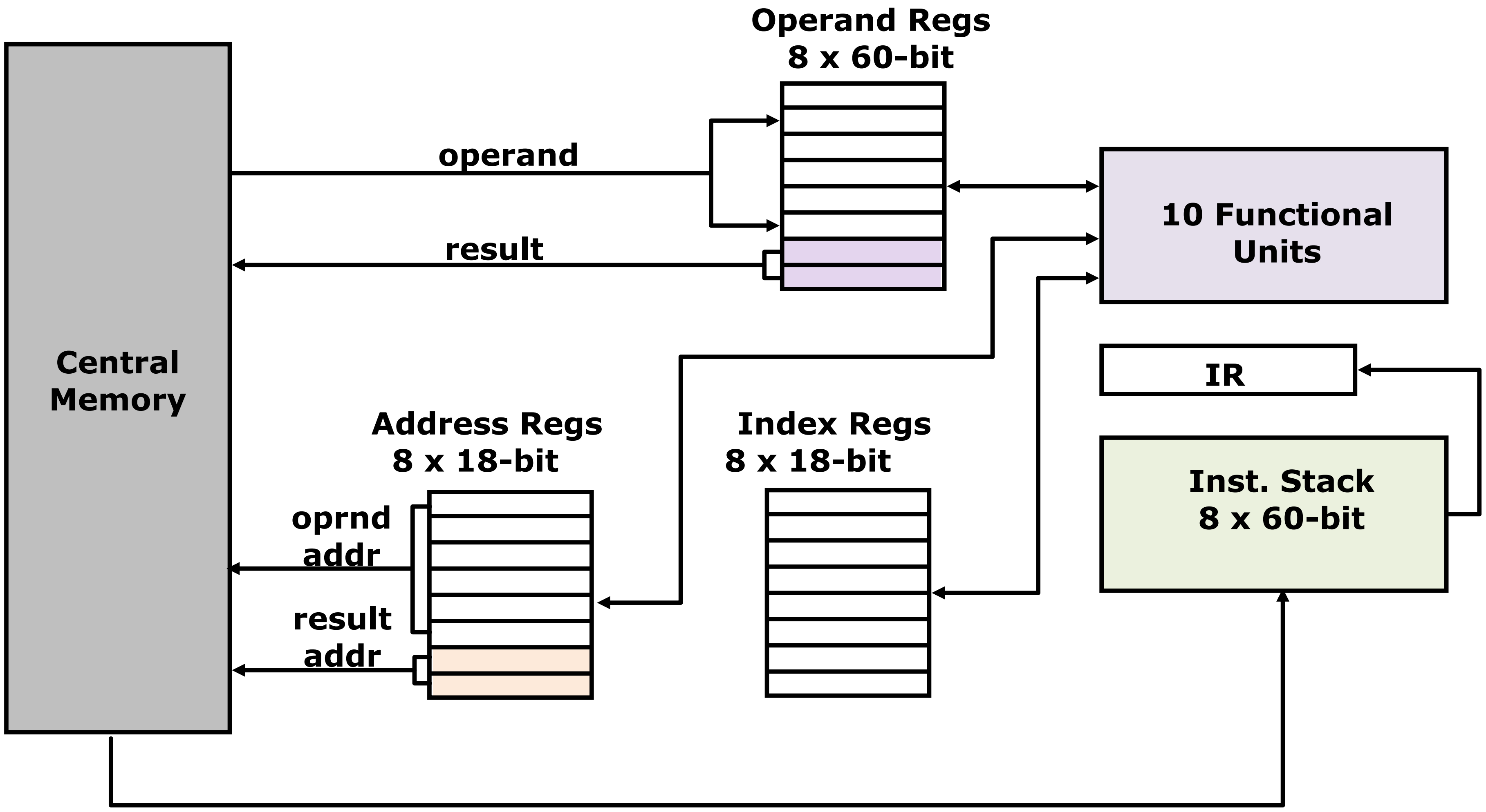
CDC 6600 by Seymour Cray

- Year 1963
- A fast pipelined machine with 60-bit words
  - 128K word main memory capacity, 32 banks
- Ten functional units (parallel, unpipelined)
  - Floating Point: adder, 2 multipliers, divider
  - Integer: adder, 2 incrementers, ...

CDC 6600 by Seymour Cray

- Hardwired control (no microcoding)
  - Dynamic scheduling of instructions using a scoreboard
- Ten Peripheral Processors for Input/Output
  - A fast multi-threaded 12-bit integer ALU
- Very fast clock, 10 MHz (FP add in 4 clocks)
- >400,000 transistors, 750 sq. ft., 5 tons, 150 kW, novel freon-based technology for cooling
- Fastest machine in world for 5 years (until 7600)
  - Over 100 sold ($7-10M each)

CDC 6600: Datapath



A Load/Store Architecture

- Separate instructions to manipulate three types of reg:
  - 8 60-bit data registers (X)
  - 8 18-bit address registers (A)
  - 8 18-bit index registers (B)
- All arithmetic and logic instructions are reg-to-reg

6	3	3	3
opcode	i	j	k

Ri ←(Rj) op (Rk)

- Only Load and Store instructions refer to memory!

6	3	3	18
opcode	i	j	disp

Ri ←[(Rj) + disp]

CDC6600: Vector Addition

- Implicit operations:
  - Touching address registers 1 to 5 initiates a load
  - 6 to 7 initiates a store
  - very useful for vector operations

A_i = address register
B_i = index register
X_i = data register

```
        B0 ← - n
loop:   JZE  B0, exit
        A0 ← B0 + a0        load X0
        A1 ← B0 + b0        load X1
        X6 ← X0 + X1
        A6 ← B0 + c0        store X6
        B0 ← B0 + 1
        jump  loop
```

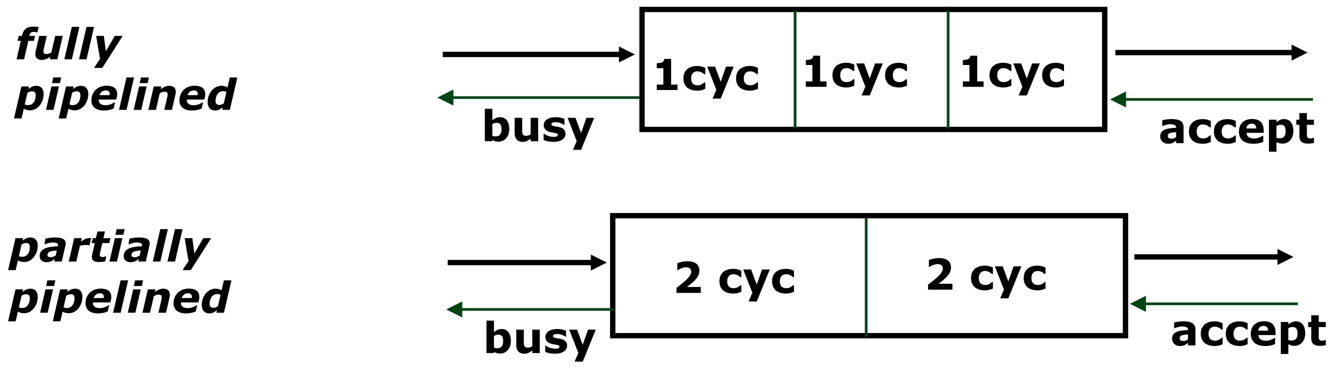
Floating Point ISA

- Interaction between the Floating point datapath and the Integer datapath is determined largely by the ISA
- RISC-V ISA
  - Separate register files for FP and Integer instructions the only interaction is via a set of move instructions (some ISA's don't even permit this)
  - Separate load/store for FPR's and GPR's but both
  - Use GPR's for address calculation
  - Separate conditions for branches
    - FP branches are defined in terms of condition codes

Floating Point Unit

- Much more hardware than an integer unit
- Single-cycle floating point unit is a bad idea - why?
  - It is common to have several floating point units
  - It is common to have different types of FPU's
    - Fadd, Fmul, Fdiv, ...
  - An FPU may be pipelined, partially pipelined or not pipelined
  - To operate several FPU's concurrently the register file needs to have more read and write ports

Function Unit Characteristics



- Function units have internal pipeline registers
  - Operands are latched when an instruction enters a function unit
  - Inputs to a function unit (e.g., register file) can change during a long latency operation

Complex Pipeline Control Issues

- Structural conflicts at the execution stage if some FPU or memory unit is not pipelined and takes more than one cycle
- Structural conflicts at the write-back stage due to variable latencies of different function units
- Out-of-order write hazards due to variable latencies of different function units

Effects of Superscalar Execution

- The outcomes of conditional branch instructions are usually predicted in advance to stalling or flushing
- Instructions are initiated for execution in parallel based on the availability of operand data, rather than their original program sequence
  - Dynamic instruction scheduling
- Upon completion instruction results are serialized back to the original program order

Instruction Dispatch Policy

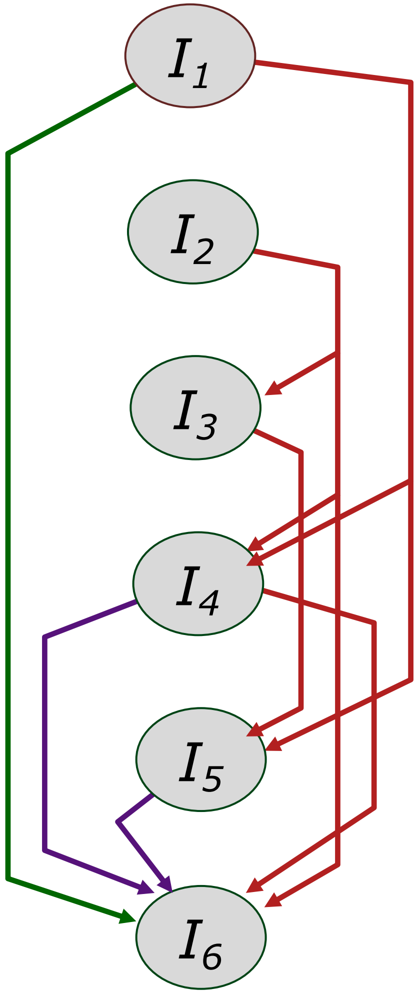
- Selection Rule
  - Specifies when instructions are considered executable
    - e.g. Dataflow principle of operation
  - Those instructions whose operands are available are executable
- Arbitration Rule
  - Needed when more instructions are eligible for execution than can be disseminated
    - e.g. choose the "oldest" instruction
- Dispatch order
  - Determines whether a non-executable instruction prevents all subsequent instructions from being dispatched

Dependencies

- Data Dependency
  - RAW, WAR, WAW
- Procedural Dependency
  - Cannot execute instructions after a (conditional) branch in parallel with instructions before a branch
- Structural Dependency
  - Two or more instructions requiring access to the same resource at the same time
    - e.g. functional units, registers, bus

Data Hazards

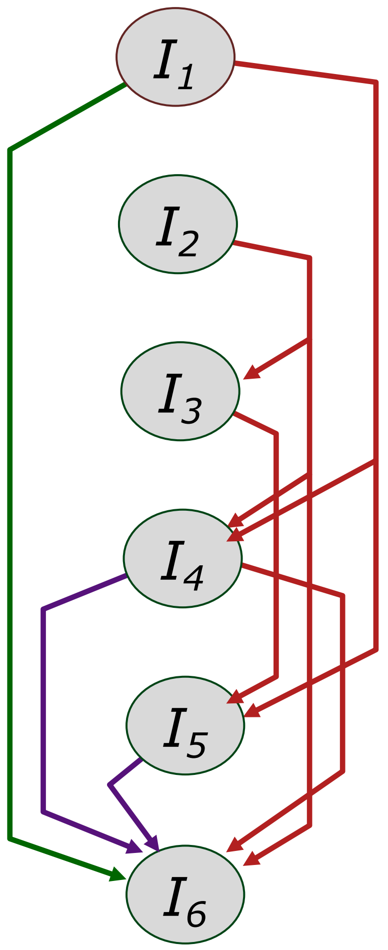
I_1	ADD	x6,	x6,	x4
I_2	LW	x2,	44(x3)	
I_3	SUB	x5,	x2,	x4
I_4	AND	x8,	x6,	x2
I_5	SUB	x10,	x5,	x6
I_6	ADD	x6,	x8,	x2



RAW Hazards
WAR Hazards
WAW Hazards

Data Hazards & Execution

					Latencies
I_1	ADD	x6,	x6,	x4	3
I_2	LW	x2,	44(x3)		2
I_3	SUB	x5,	x2,	x4	3
I_4	AND	x8,	x6,	x2	1
I_5	SUB	x10,	x5,	x6	3
I_6	ADD	x6,	x8,	x2	3



Let us assume that ADD and SUB use the same functional unit

Program Order of Execution

- Instruction Issue Policies
  - Order in which instructions are fetched
  - Order in which instructions are executed
  - Order in which instructions update registers and memory values (order of completion)
- Standard Categories
  - In-order issue with in-order completion
  - In-order issue with out-of-order completion
  - Out-of order issue with out-of-order completion

Program Order of Execution

- Order of the program
  - The order created by the compiler
- Order in which instructions are fetched
  - Could be different than program order due to prefetching and trace execution
- Order in which instructions are executed
  - Parallelism exploitation and data availability can lead to an order of execution
- Order in which instructions change registers and memory
  - Commitment or retiring

In-Order Issue & In-Order Execution

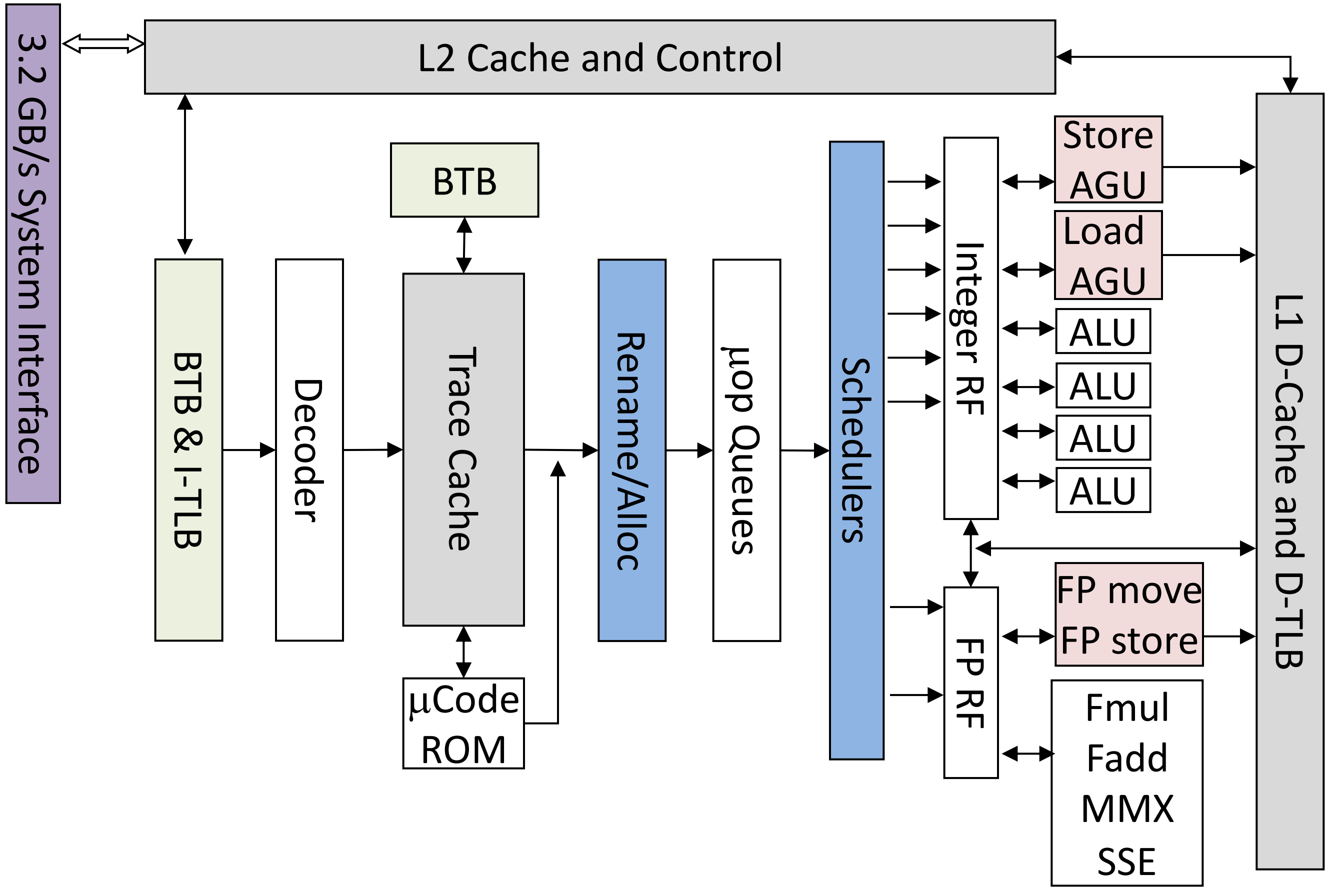
Decode		Execute			Commit		Cycle
I_1	I_2						1
I_3	I_4	I_1	I_2				2
I_5	I_6	I_1	I_2				3
		I_1					4
					I_1	I_2	5
		I_3		I_4			6
		I_3					7
		I_3					8
					I_3	I_4	9
		I_5					10
		I_5					11
		I_5					12
		I_6					13

In-Order Issue & Out-Order Execution

Decode		Execute			Commit		Cycle
I_1	I_2						1
I_3	I_4	I_1	I_2				2
I_5	I_6	I_1	I_2				3
		I_1				I_2	4
		I_3			I_1		5
		I_3		I_4			6
		I_3				I_4	7
					I_3		8
		I_5					9
		I_5					10
		I_5					11
		I_6					12
		I_6					13

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive



Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

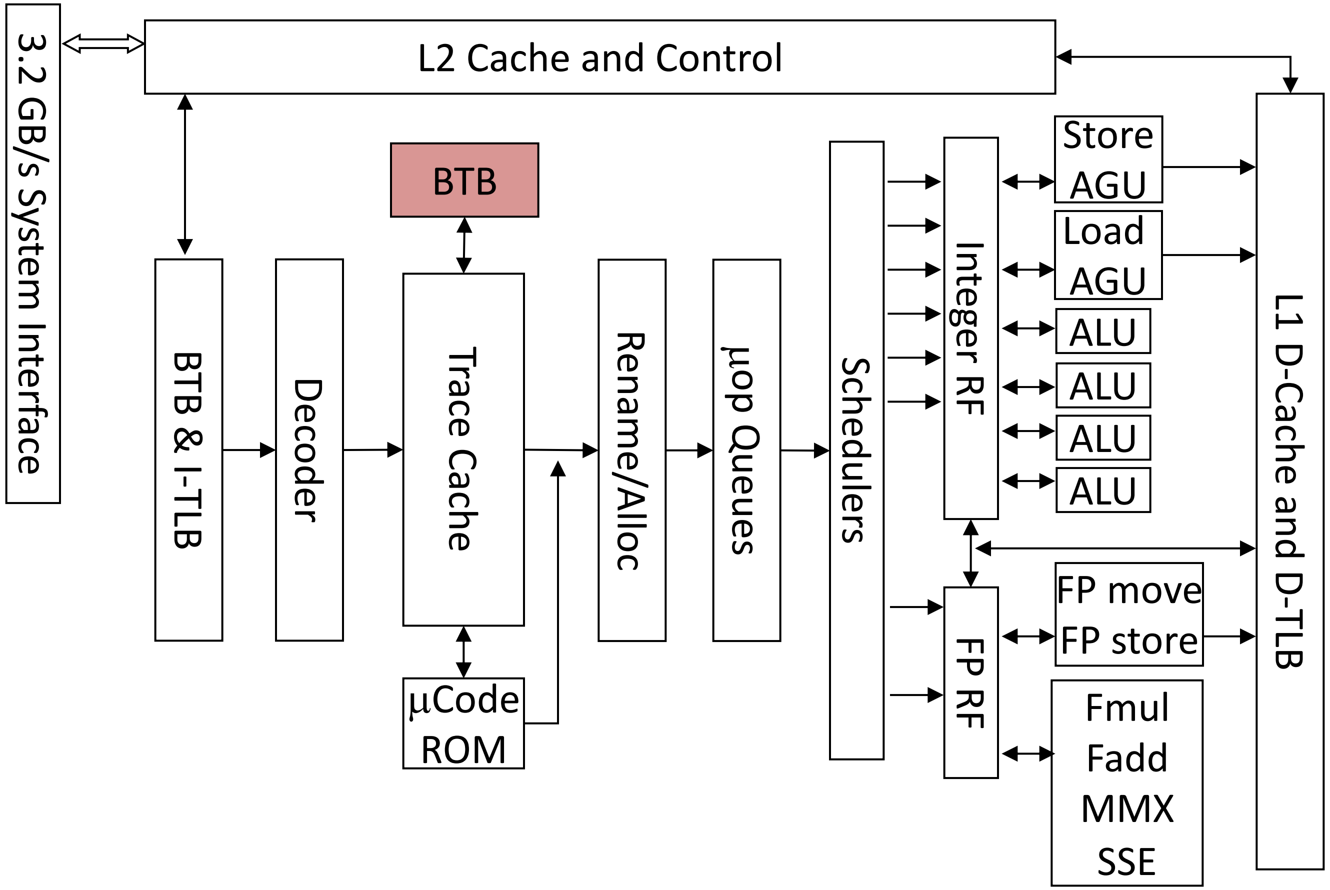
- The Pentium 4 has a 20 stage pipeline
- This deep pipeline increases
  - Performance of the processor
  - Frequency of the clock
  - Scalability of the processor
- Also, it provides
  - High Clock Rates
  - Frequency headroom to above 1GHz

TC Nxt IP

- Trace Cache: Next Instruction Pointer
- Held in the BTB (branch target buffer)
- And specifies the position of the next instruction to be processed
- Branch Prediction takes over
  - Previously executed branch: BHT has entry
  - Not previously executed or Trace Cache has invalidated the location: Calculate Branch Address and send to L2 cache and/or system bus

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

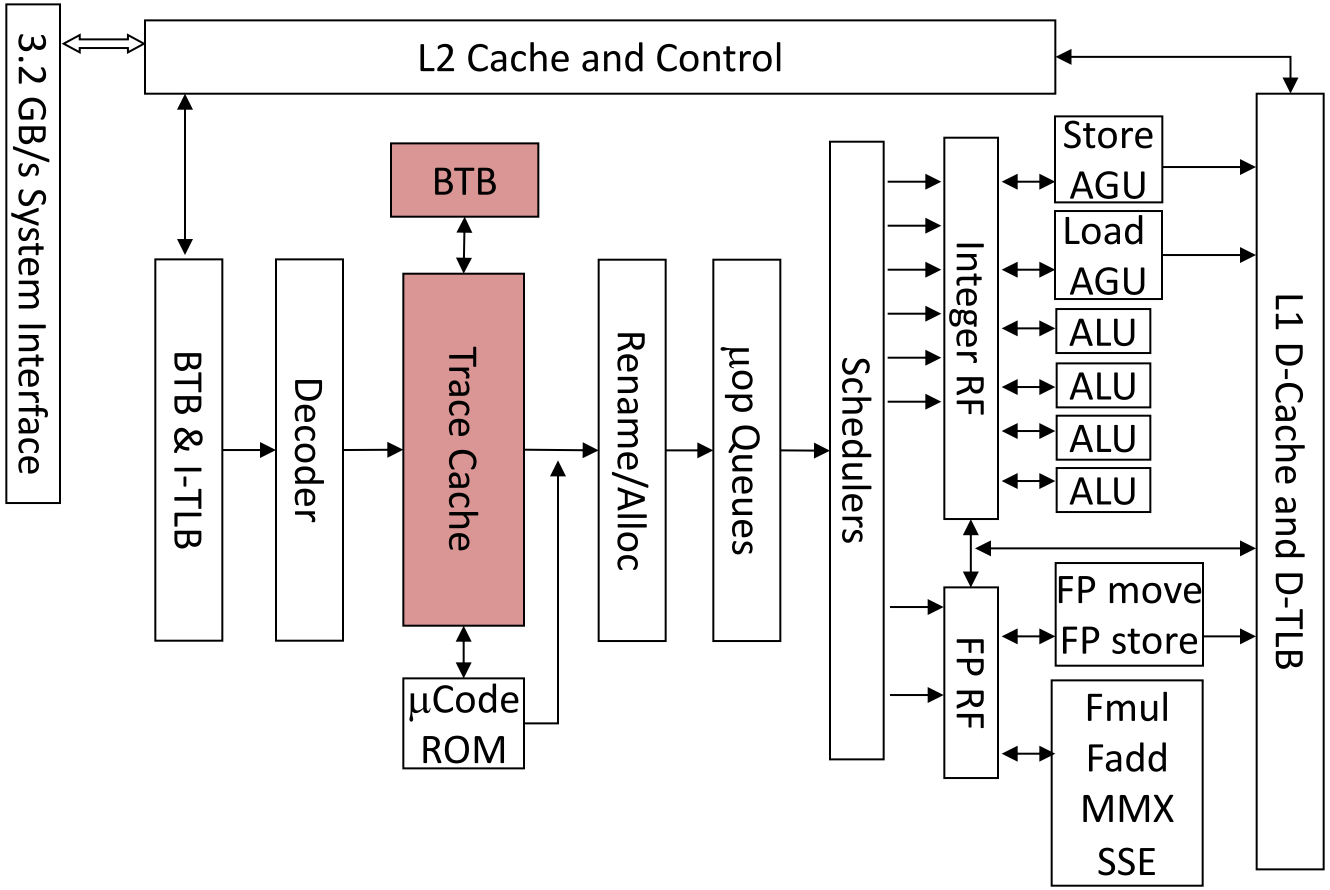


Trace Cache (TC) Fetch

- Reading μops (from Execution TC) requires two clock cycles
- The TC holds up to 12K μops and can output up to three μops per cycle to the Rename/Allocator
- Storing μops in the TC removes:
  - Decode-costs on frequently used instructions
  - Extra latency to recover on a branch misprediction

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

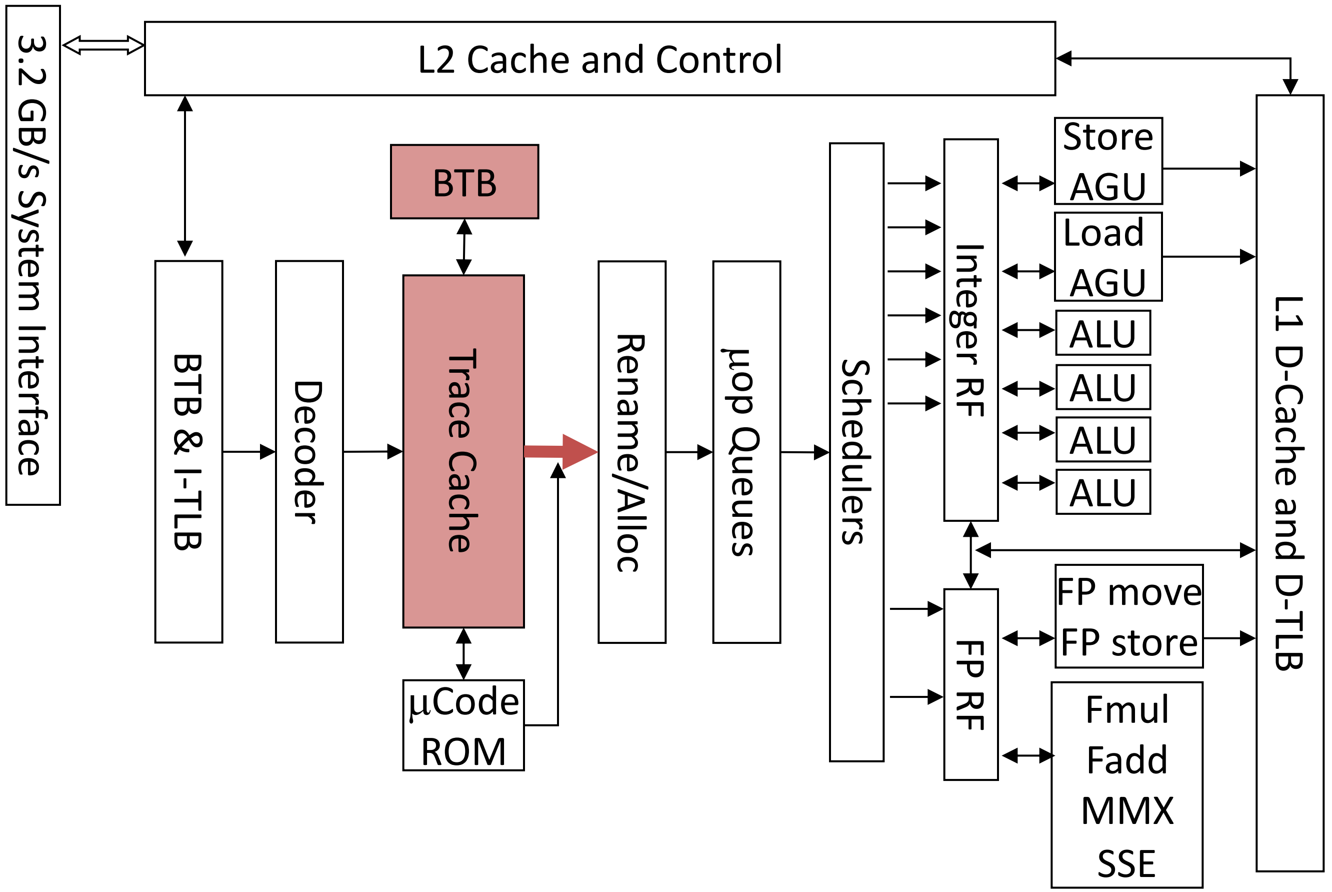


Wire Drive

- This stage of the pipeline occurs multiple times
- WD only requires one clock cycle
- During this stage, up to three μops are moved to the Rename/Allocator
  - One load
  - One store
  - One manipulate instruction

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

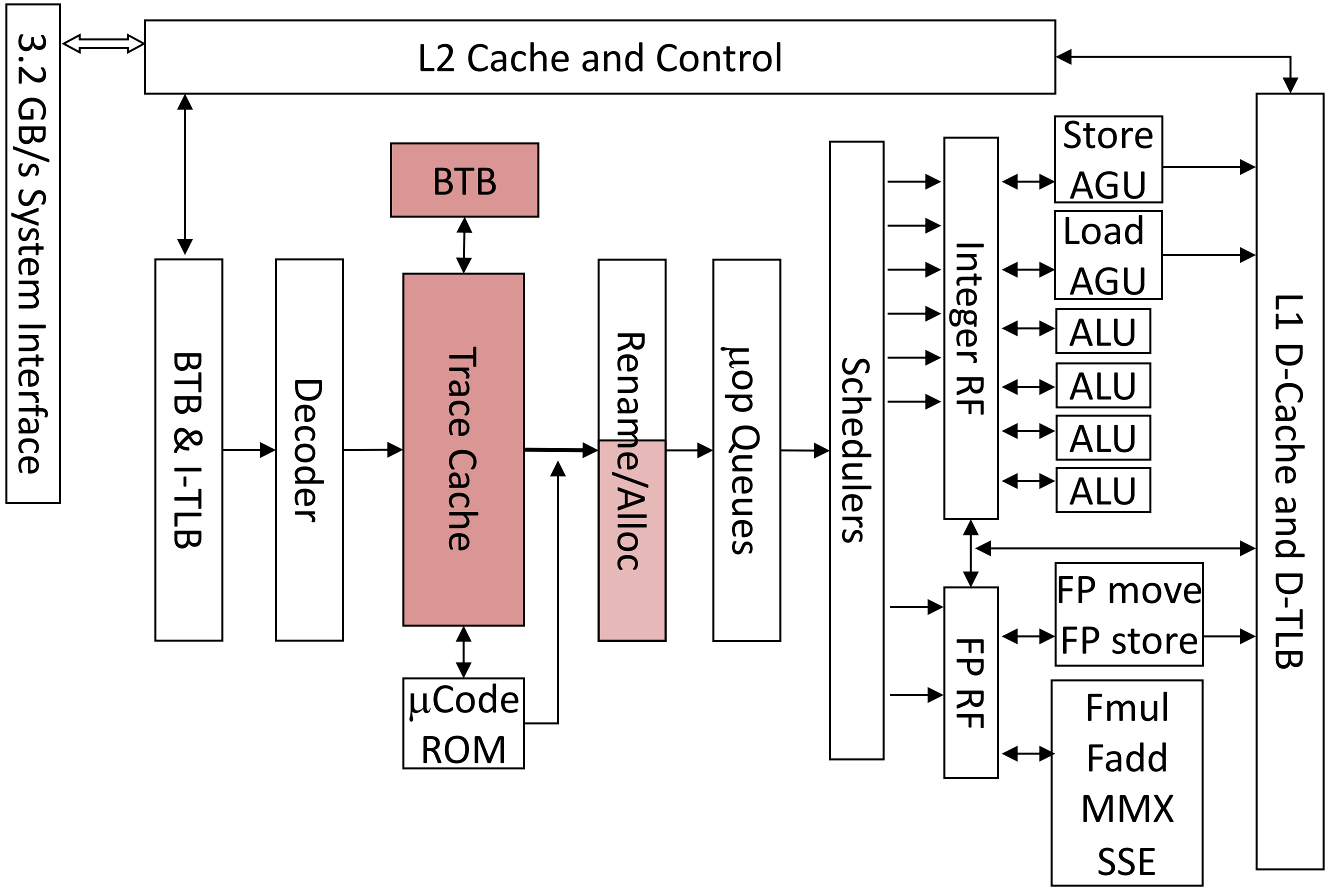


Allocate

- This stage determines what resources are needed by the µops.
- Decoded µops go through a one-stage Register Allocation Table (RAT)
- IA-32 instruction register references are renamed during the RAT stage

Pentium 4: A Superscalar CISC Architecture

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

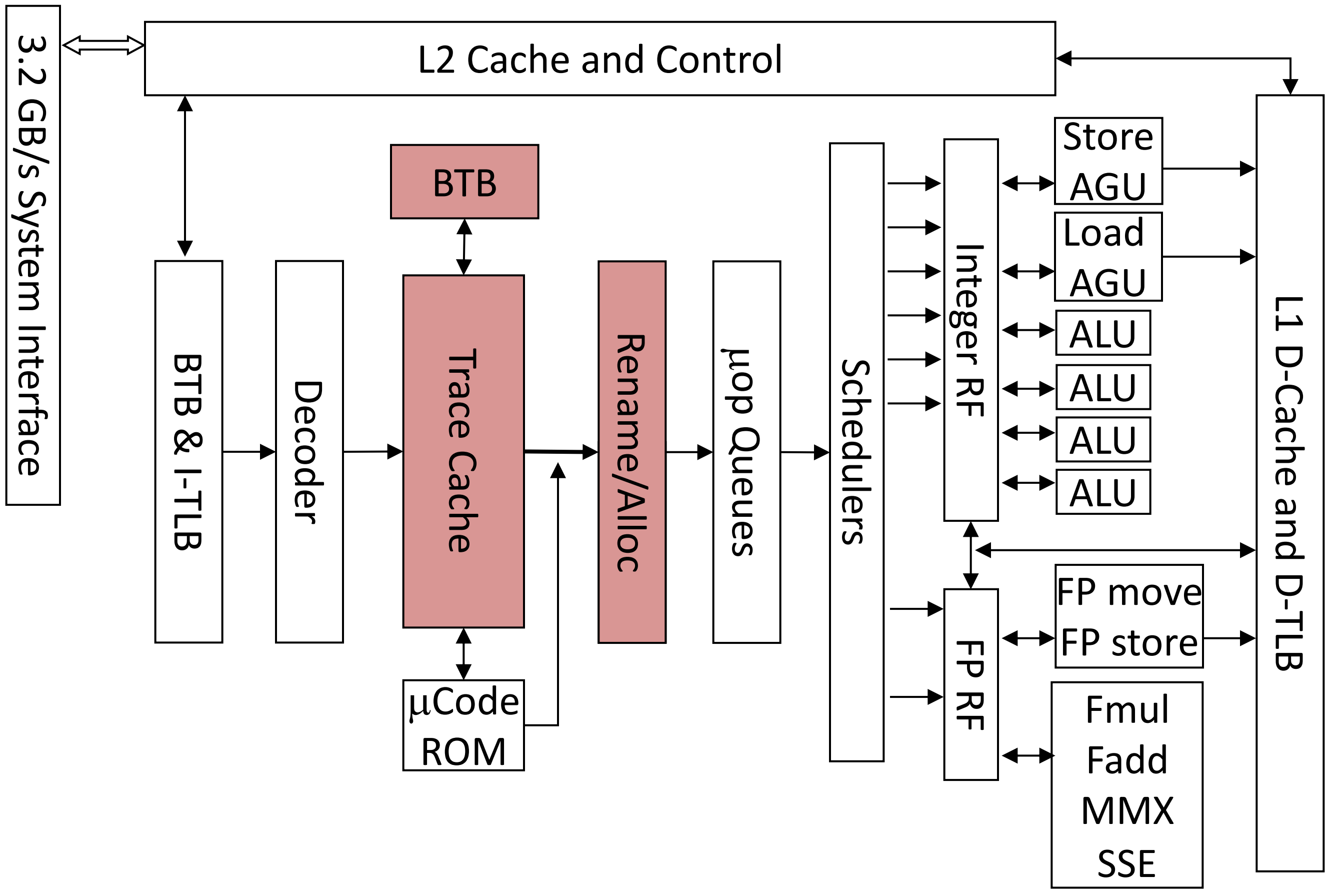


Renaming Registers

- This stage renames logical registers to the physical register space
  - It allows the small, 8-entry,architecturally defined IA-32 register file to be dynamically expanded to use the 128 physical registers in the Pentium 4 processor
- In the MicroBurst Architecture there are 128 registers with unique names
- Basically, any references to original IA-32 general purpose registers are renamed to one of the internal physical registers
- It removes false register name dependencies between instructions allowing the processor to execute more instructions in parallel
- Parallel execution helps keep all resources busy

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

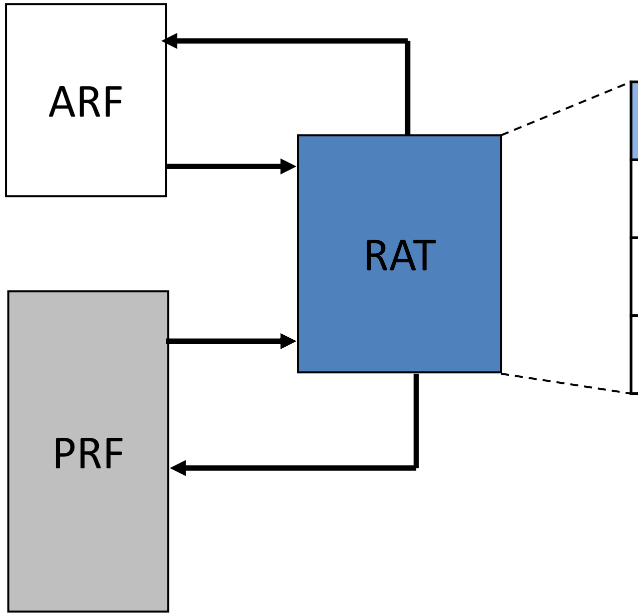


Hardware Register Renaming

- Give processor more registers than specified by the ISA
  - Temporarily map ISA registers ("logical" or "architected" registers) to the physical registers to avoid overwrites
  - Help eliminate WAR and WAW dependencies
    - WAR and WAW dependencies are from reusing registers
- Components:
  - Mapping mechanism
  - Physical registers
  - Allocated vs. free registers
  - Allocation/deallocation mechanism

Hardware Register Renaming

- Architected Register File (ARF)
  - Visible to the outside world i.e., programmer
- Physical Register File (PRF)
- Register Allocation/Alias Table (RAT)

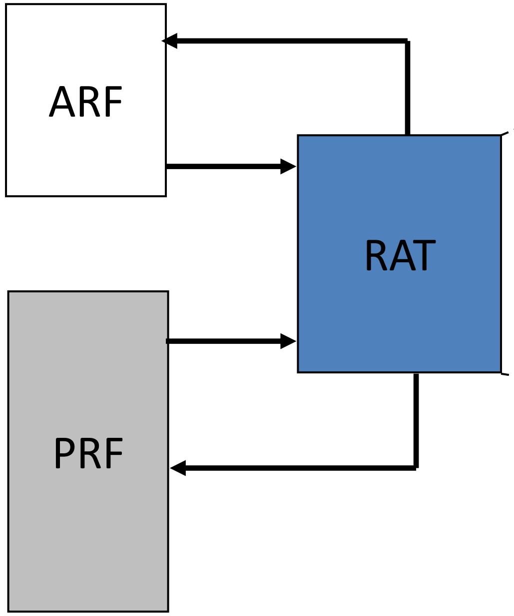


ARF	PRF	In-Use

I_1	ADD	x6,	x6,	x4	3
I_2	LW	x2,	44(x3)		2
I_3	SUB	x5,	x2,	x4	3
I_4	AND	x8,	x6,	x2	1
I_5	SUB	x10,	x5,	x6	3
I_6	ADD	x6,	x8,	x2	3

Hardware Register Renaming

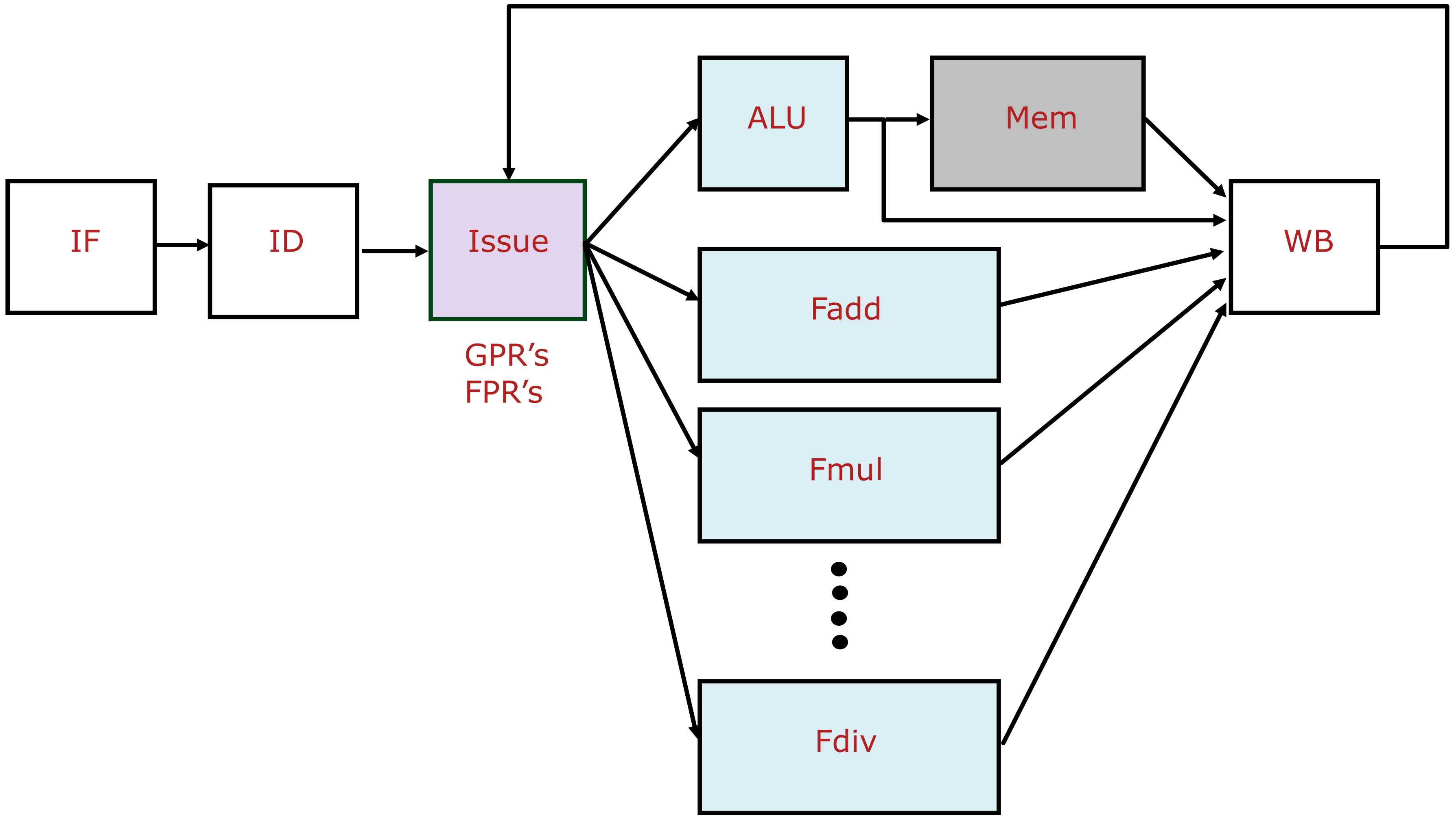
- Architected Register File (ARF)
  - Visible to the outside world i.e., programmer
- Physical Register File (PRF)
- Register Allocation Table (RAT)



ARF	PRF	In-Use
x6	x6a	I_1
x4	x4b	I_3
x6	x6b	I_6

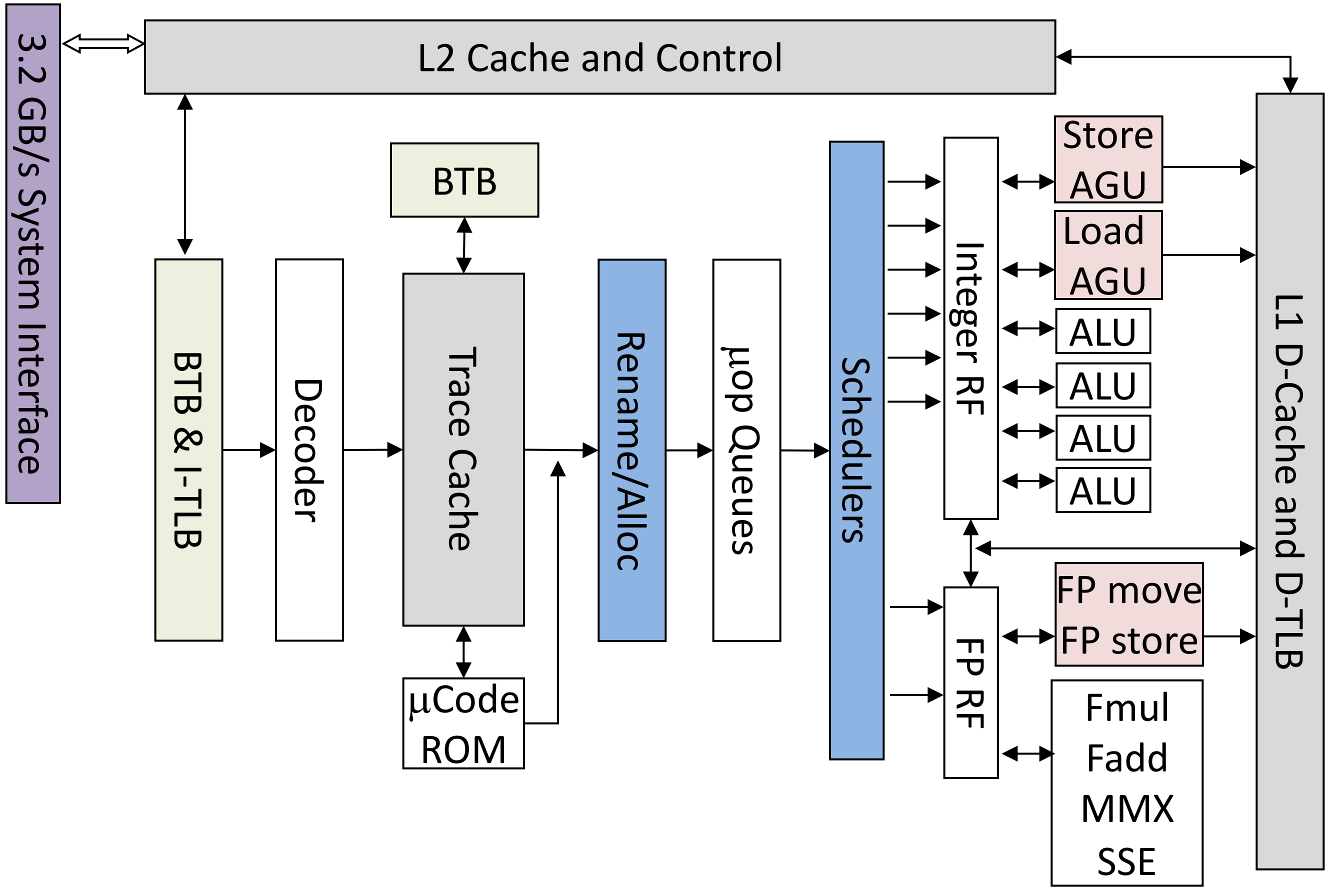
I_1	ADD	x6,	x6a	x4	3
I_2	LW	x2,	44(x3)		2
I_3	SUB	x5,	x2,	x4b	3
I_4	AND	x8,	x6,	x2	1
I_5	SUB	x10,	x5,	x6	3
I_6	ADD	x6b,	x8,	x2	3

Complex Pipeline Structure



Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

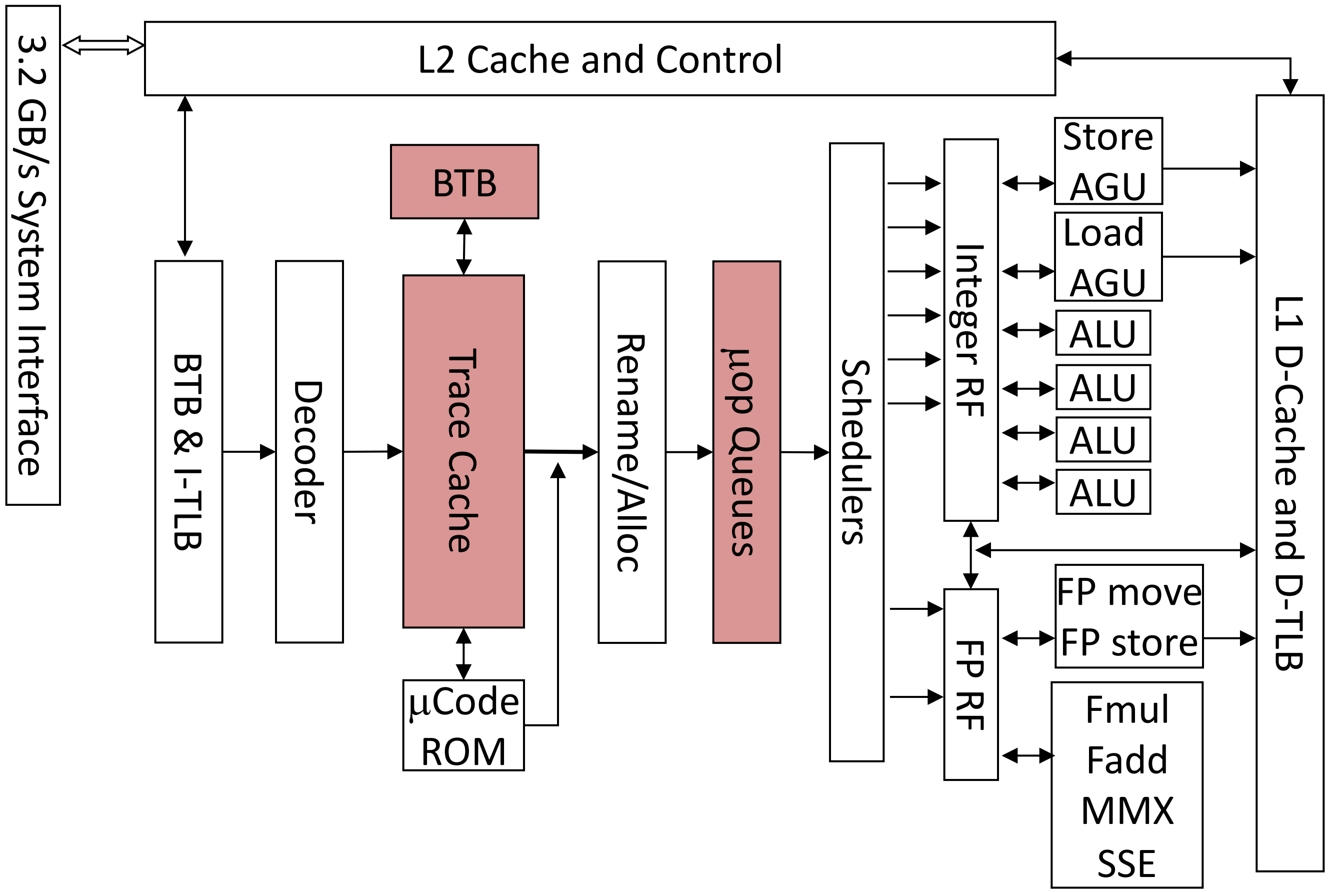


Que

- Also known as the µops "pool".
- µops are put in the queue before they are sent to the proper execution unit.
- Provides record keeping of order commitment/retirement to ensure that µops are retired correctly.
- The queue combined with the schedulers provides a function similar to that of a reservation station.

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

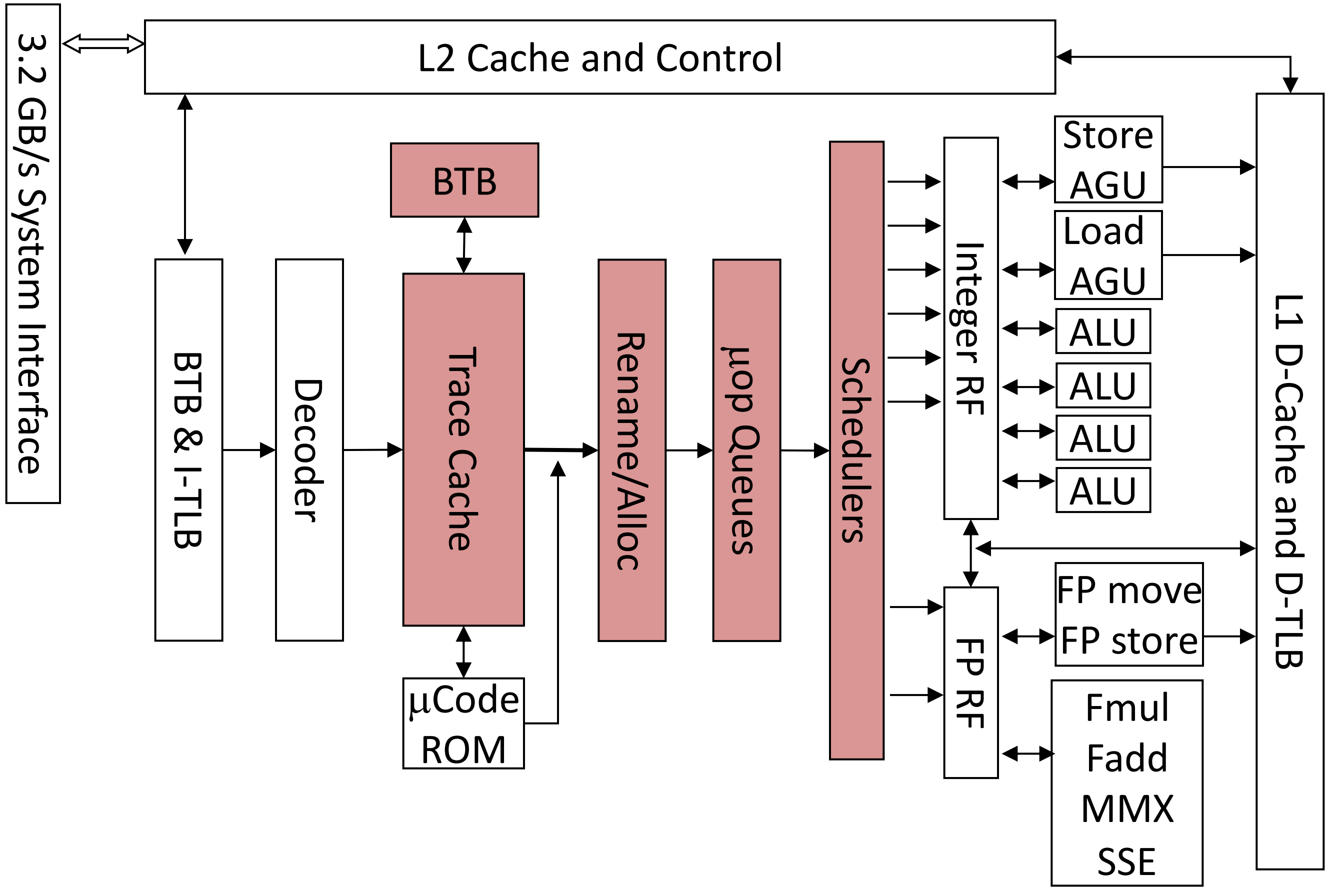


Schedulers

- Ensures μops execute in the correct sequence
- Disperses μops in the queue (or pool) to the proper execution units.
- The scheduler looks to the pool for requests, and checks the functional units to see if the necessary resources are available.

Pentium 4: A Superscalar CISC Architecture

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

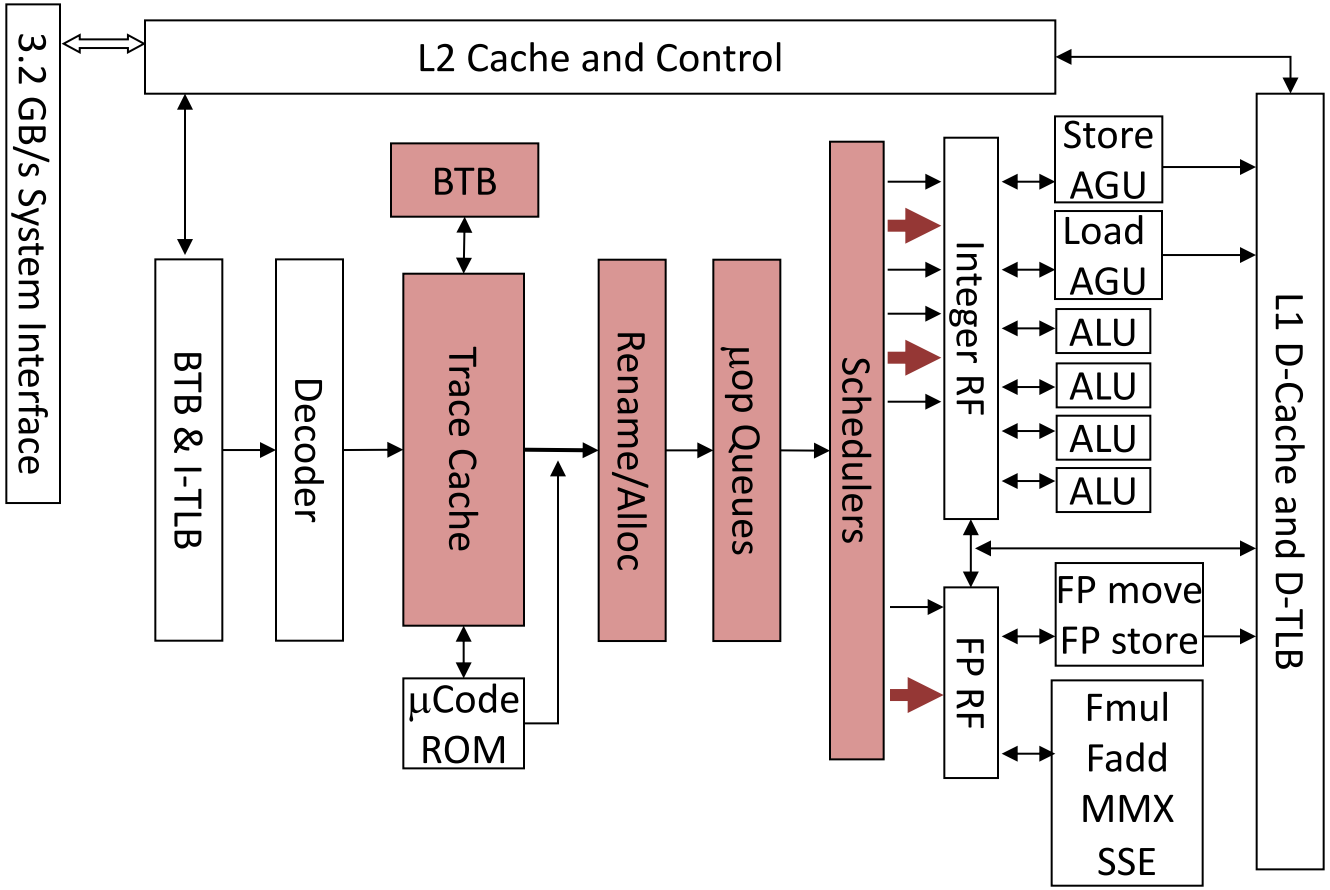


Dispatch

- This stage takes two clock cycles to send each μops to the proper execution unit.
- Logical functions are allowed to execute in parallel, which takes half the time, and thus executes them out of order.
- The dispatcher can also store results back into the queue (pool) when it executes out of order.

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

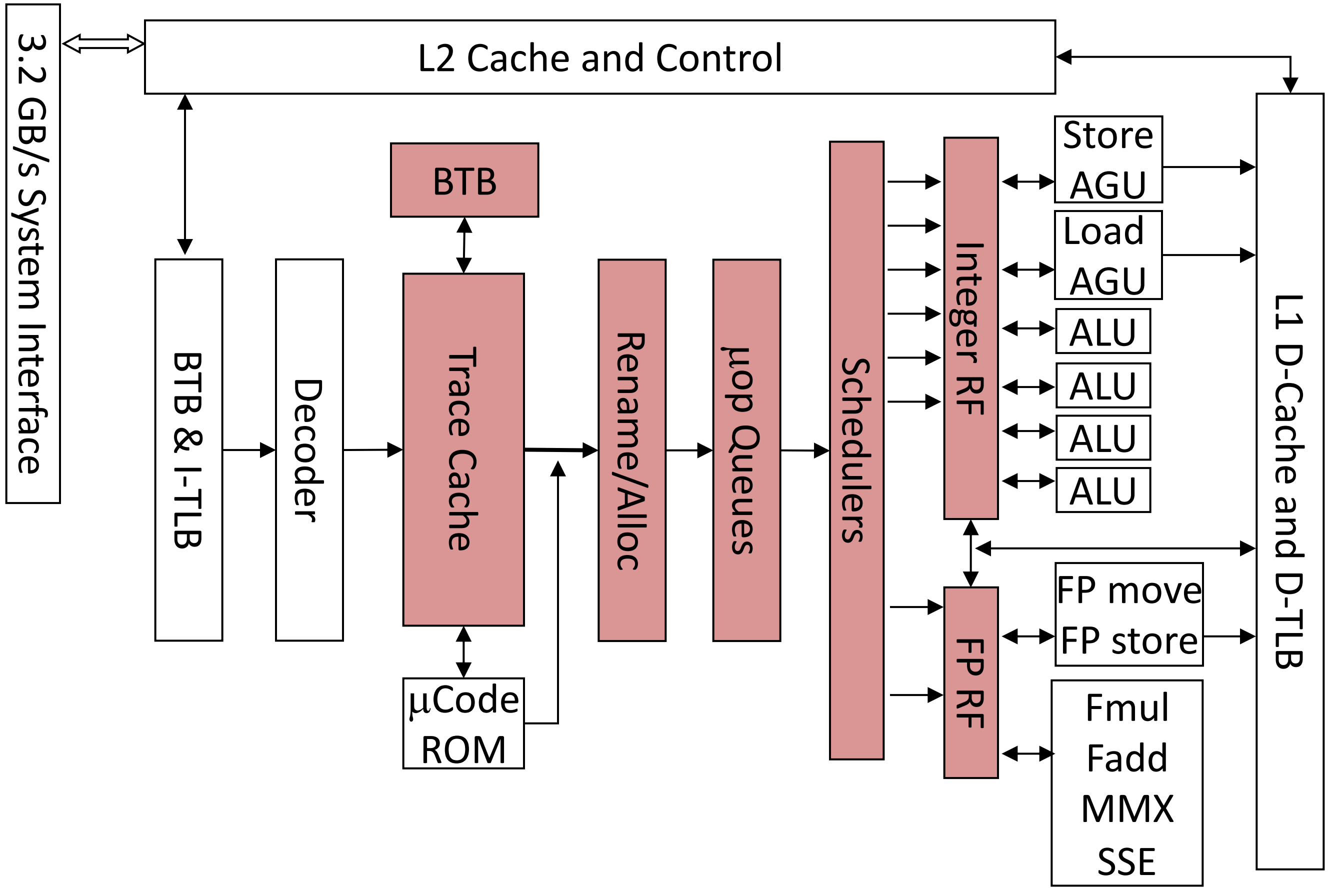


Register File

- Internal registers are read

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive

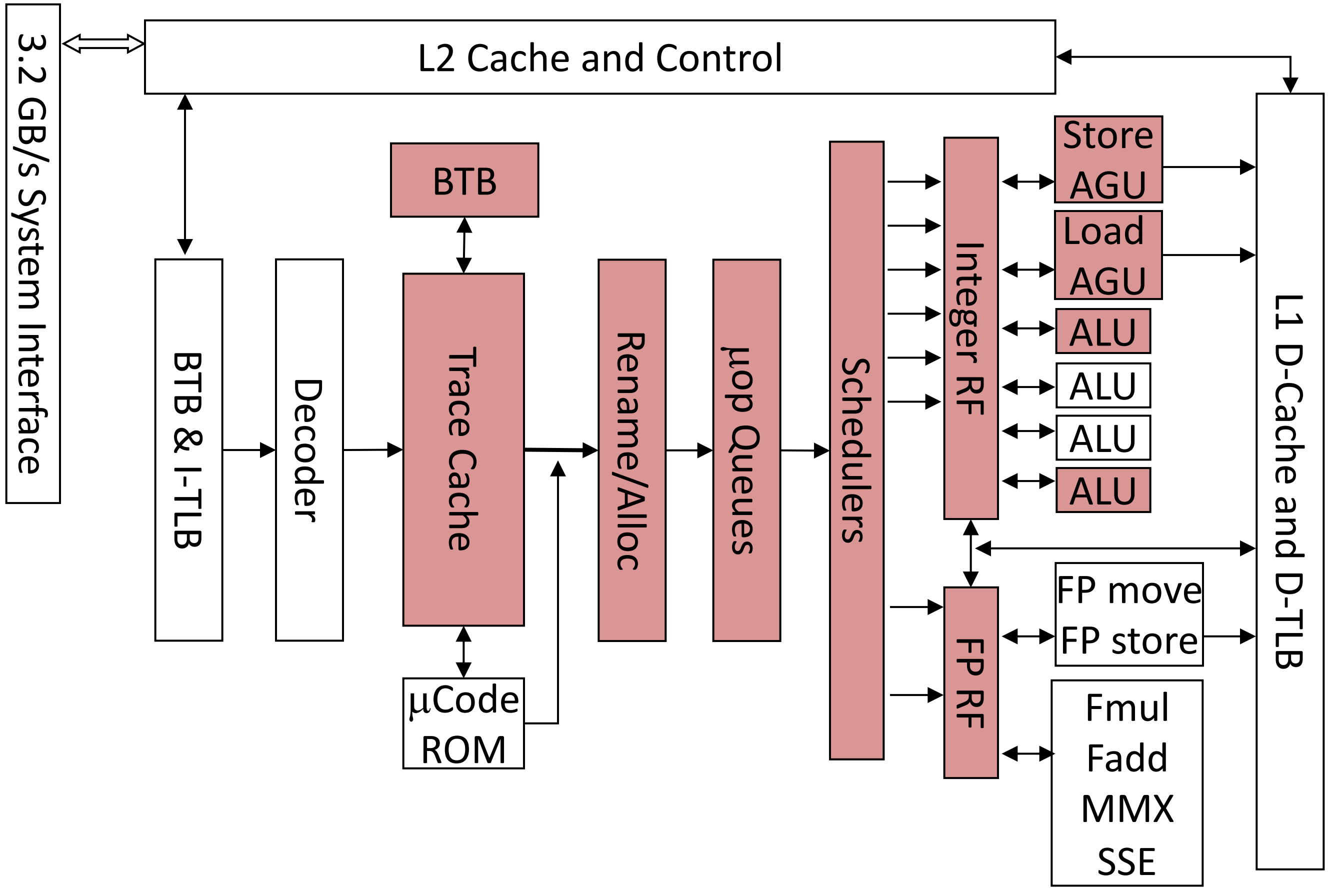


Execution

- μops will be executed on the proper execution engine by the processor
- The number of execution engines limits the amount of execution that can be performed.
- Integer and floating point unites comprise this limiting factor

Pentium 4: A Superscalar CISC Architecture

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive



Retirement

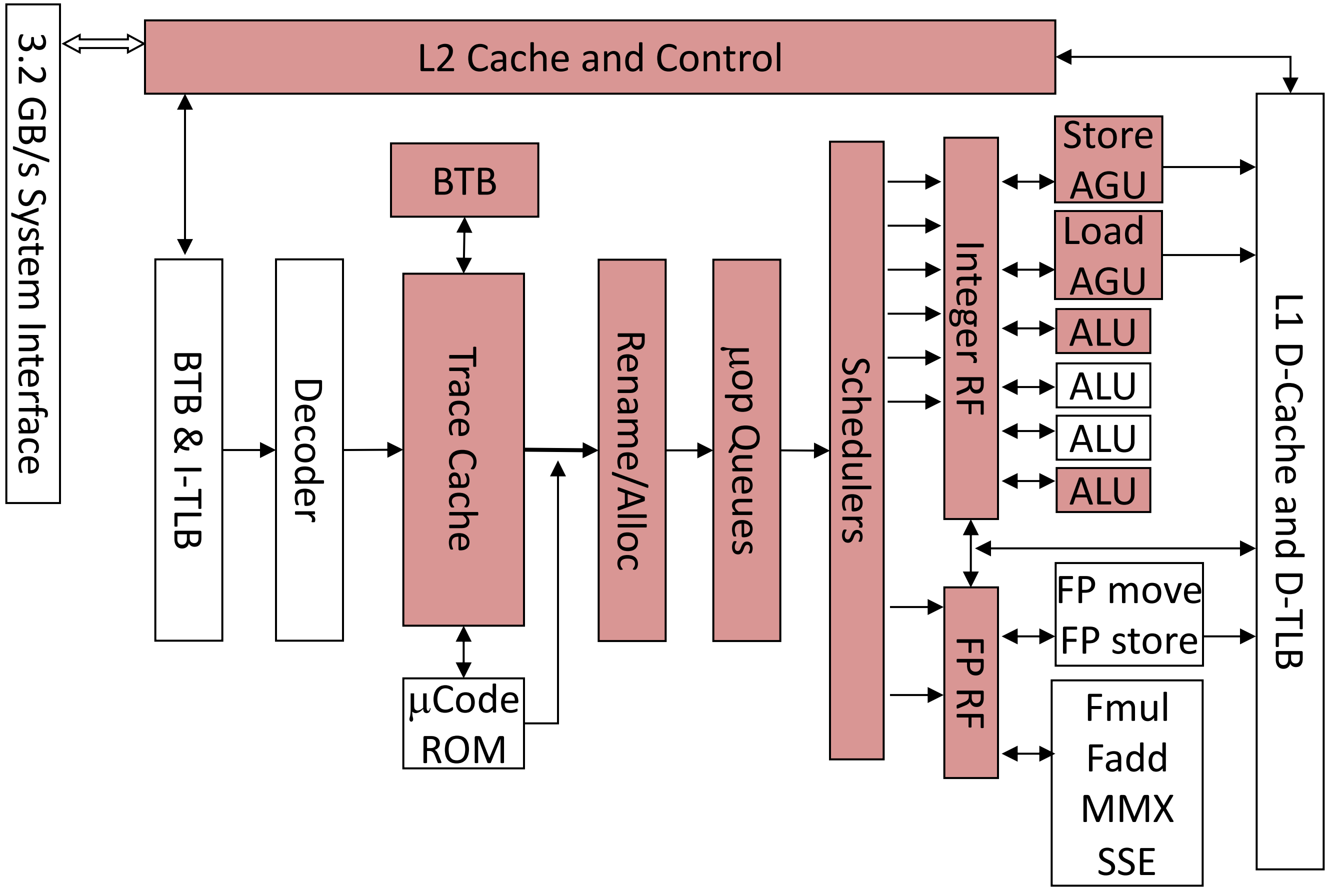
- During this stage results are written back to memory or actual IA-32 registers that were referred to before renaming took place.
- This unit retires all instructions in their original order, taking all branches into account.
- Three µops may be retired in one clock cycle
- The processor detects and recovers from mispredictions in this stage.
- Also, a reorder buffer (ROB) is used:
  - Updates the architectural state
  - Manages the ordering of exceptions

Flags, Branch Check, & Wire Drive

- Flags
  - One clock cycle is required to set or reset any flags that might have been affected.
- Branch Check
  - Brach operations compares the result of the branch to the prediction
  - The P4 uses a BHT and a BTB
- Wire Drive
  - One clock cycle moves the result of the branch check into the BTB and updates the target address after the branch has been retired.

Pentium 4: A Superscalar CISC Architecture

1 2	3 4	5	6	7 8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	BrCk	Drive



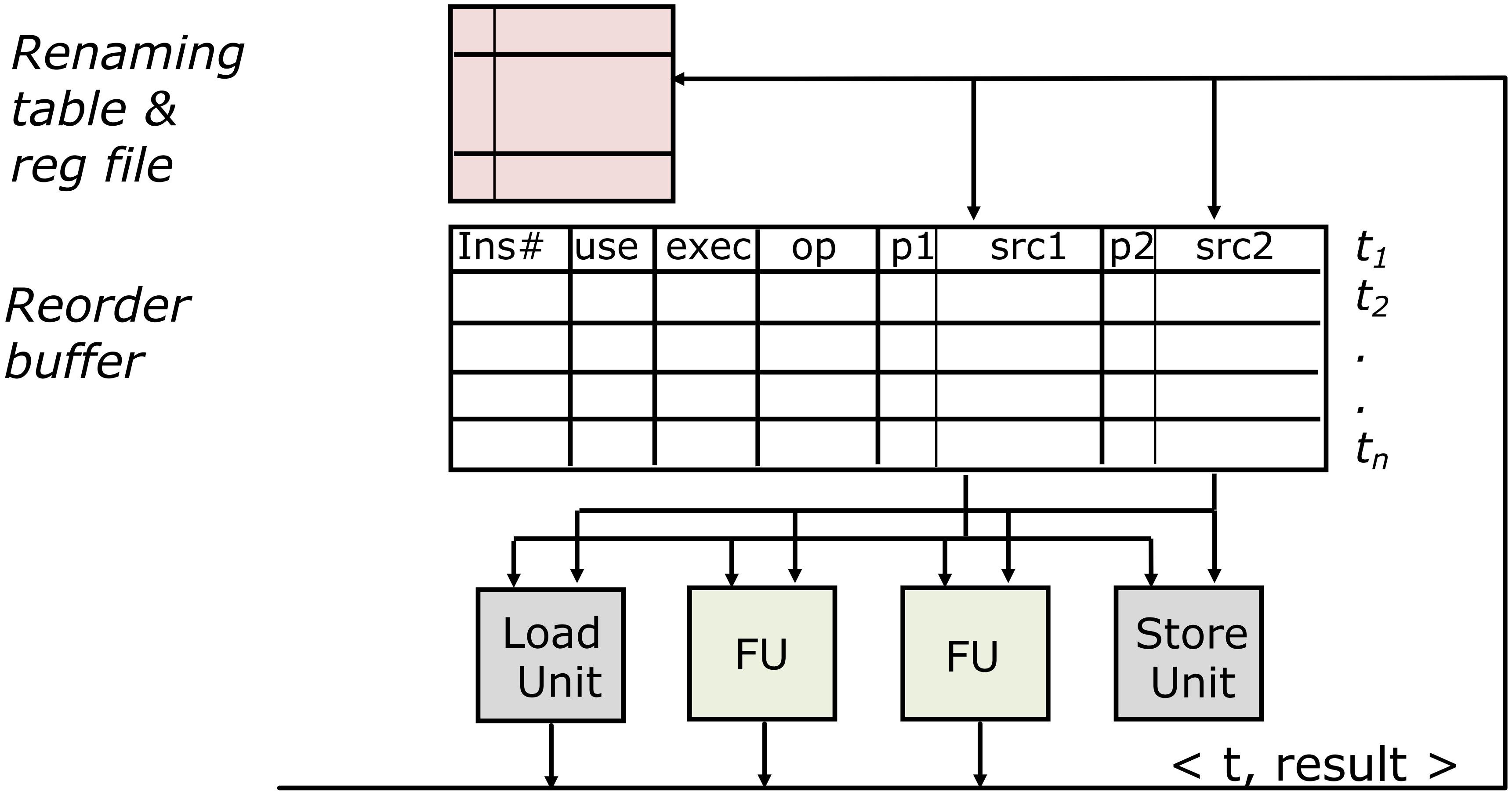
Pentium 4: A Superscalar CISC Architecture

- Stages 1-9
  - 1-2 (BTB&I-LTB, F/t): Fetch (64-byte) instructions, static branch prediction, split into 4 (118-bit) micro-ops
  - 3-4 (TC): Dynamic branch prediction with 4 bits, sequencing micro-ops
  - 5: Feed into out-of-order execution logic
  - 6 (R/a): Allocating resources (126 micro-ops, 128 registers)
  - 7-8 (R/a): Renaming registers and removing false dependencies
  - 9 (micro-opQ): Re-ordering micro-ops

Pentium 4: A Superscalar CISC Architecture

- Stages 10-20
  - 10-14 (Sch): Scheduling (FIFO) and dispatching (6) micro-ops whose data is ready towards available execution unit
  - 15-16 (RF): Register read
  - 17 (ALU, Fop): Execution of micro-ops
  - 18 (ALU, Fop): Compute flags
  - 19 (ALU): Branch check – feedback to stages 3-4
  - 20: Retiring instructions

Renaming and Reorder Buffer



Renaming & Out-of-order Issue

Renaming table

	p	data
x1		
x2		t1/v1
x3		
x4		t2/t5
x5		
x6		t3
x7		
x8		t4/v4

Reorder buffer

Ins#	use	exec	op	p1	src1	p2	src2	
1	0/1	0/1	lw					t_1
2	1/0	0/1	lw					t_2
3	1	0	add	0/1	t2/v2	1/1	t1/v1	t_3
4	1/0	0/1	sub	1	v1	1	v1	t_4
5	1	0	and	1	v1	0/1	t4/v4	t_5
								.
								.

1	*lw*	*x2, 34(x1)*
2	*lw*	*x4, 4C(x3)*
3	*Add*	*x6, x4, x2*
4	*sub*	*x8, x2, x2*
5	*and*	*x4, x2, x8*
6	*add*	*x10, x6,x4*

Renaming & Out-of-order Issue

Renaming table

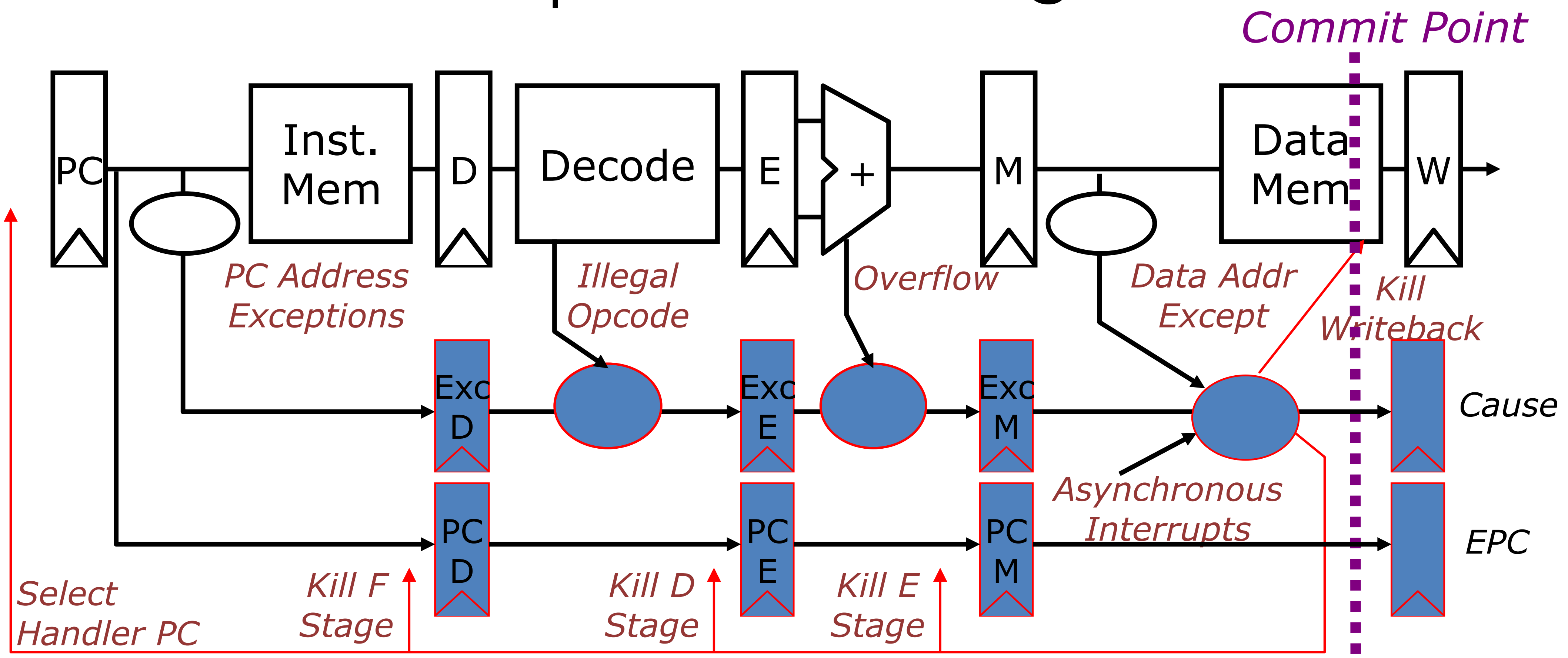
	p	data
x1		
x2		t1/v1
x3		
x4		t2/t5
x5		
x6		t3
x7		
x8		t4/v4

1	*lw*	*x2, 34(x1)*
2	*lw*	*x4, 4C(x3)*
3	*Add*	*x6, x4, x2*
4	*sub*	*x8, x2, x2*
5	*and*	*x4, x2, x8*
6	*add*	*x10, x6,x4*

Reorder buffer

Ins#	use	exec	op	p1	src1	p2	src2	
1	0/1	0/1	lw					t_1
2	1/0	0/1	lw					t_2
3	1	0	add	0/1	t2/v2	1/1	v1/t1	t_3
4	1/0	0/1	sub	1	v1	1	v1	t_4
5	1	0	and	1	v1	0/1	t4/v4	t_5
								.
								.

Exception Handling



- Hold exception flags in pipeline until commit point (M stage)
  - Inject external interrupts at commit point
  - If exception at commit Update Cause/EPC registers, kill all stages and fetch at handler PC

Precise Interrupts

- It must appear as if an interrupt is taken between two instructions (I_i and I_{i+1})
- The effect of all instructions up to and including I_i is totally complete
- No effect of any instruction after I_i has taken place
- The interrupt handler either aborts the program or restarts it at I_{i+1}

Execution Concurrency Limits

- Which features of an ISA limit the number of instructions in the pipeline?
  - Number of Registers
- Which features of a program limit the number of instructions in the pipeline?
  - Control transfers

Next Class

- Complex Pipelining: VLIW