CMPE110 Lecture 12
Out-of-Order Execution

Heiner Litz

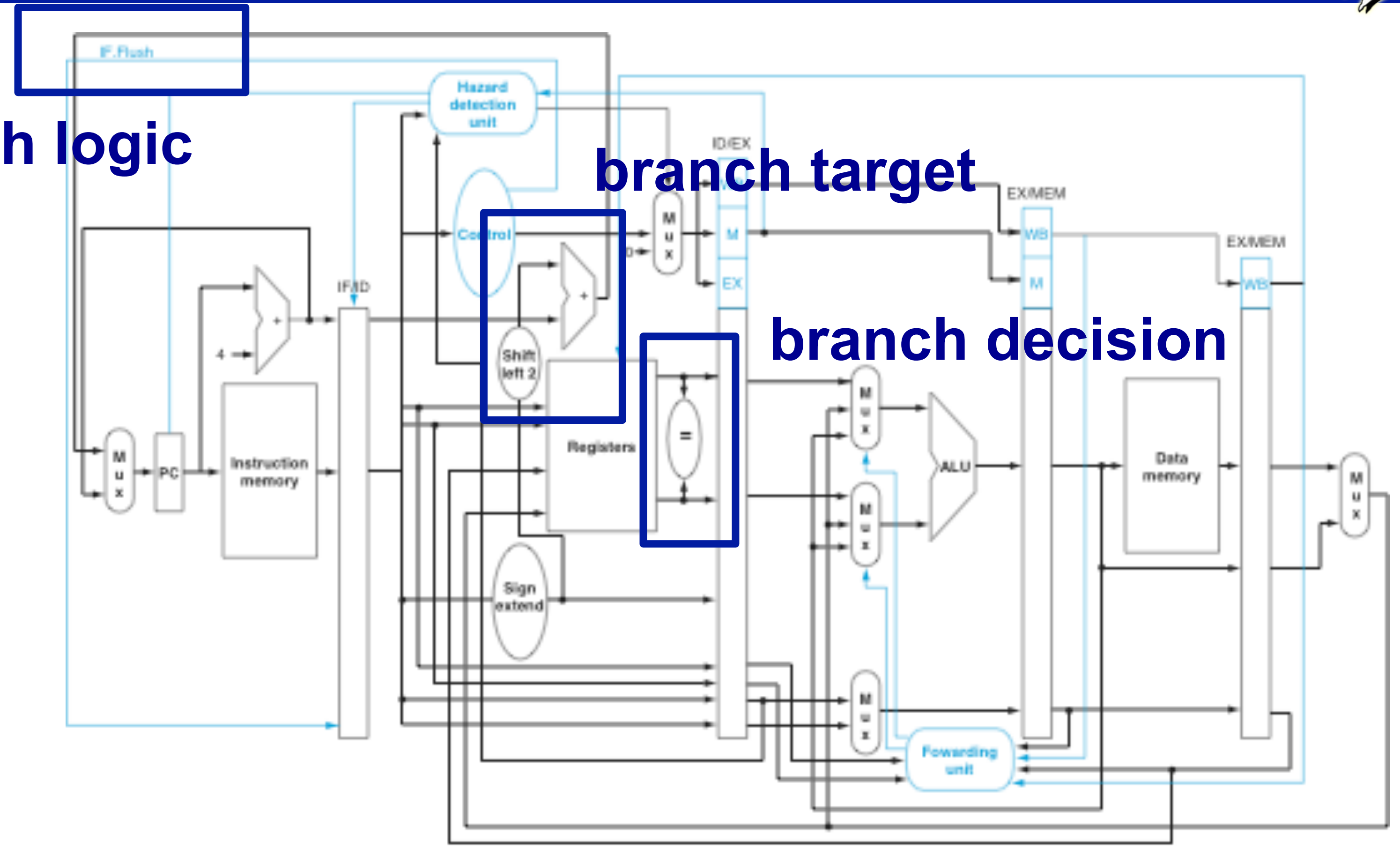
https://canvas.ucsc.edu/courses/12652

Announcements

- Midterm: May 7th
  - Everything covered in lectures, quizzes, HAs
  - Performance, ISA, processor architecture, 5-stage
  - Book Chapters: 2.1-2.8, 2.10, 2.14, 3.2, 4.1-4.10
  - Do I need to memorize the RISC-V ISA?
    - Midterm will have a copy of the cheat sheet
- Next Friday (May 4th):
  - No quiz
  - HA3 Review
  - Q & A

Review

Reducing Branch Delay

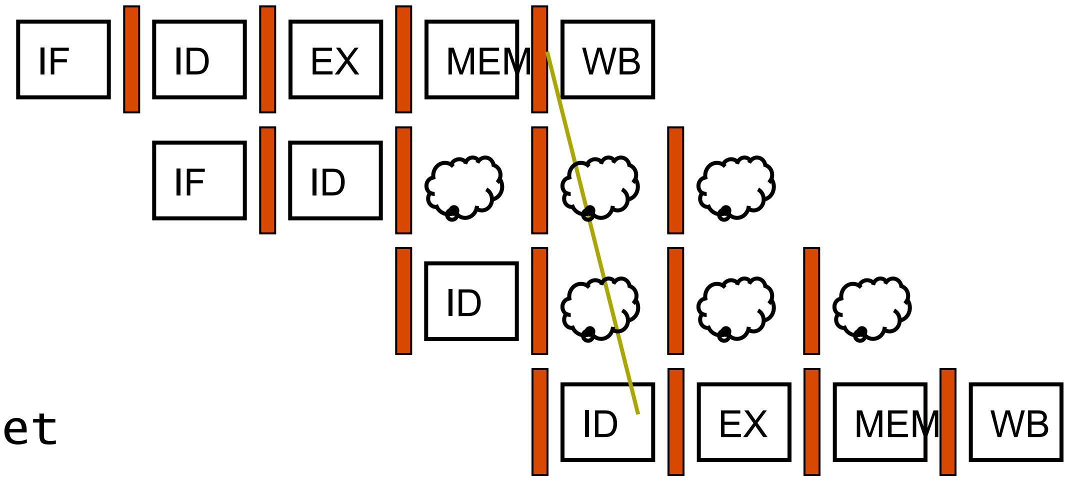


move branch computation earlier in the pipeline
enable flush capability if mispredicted

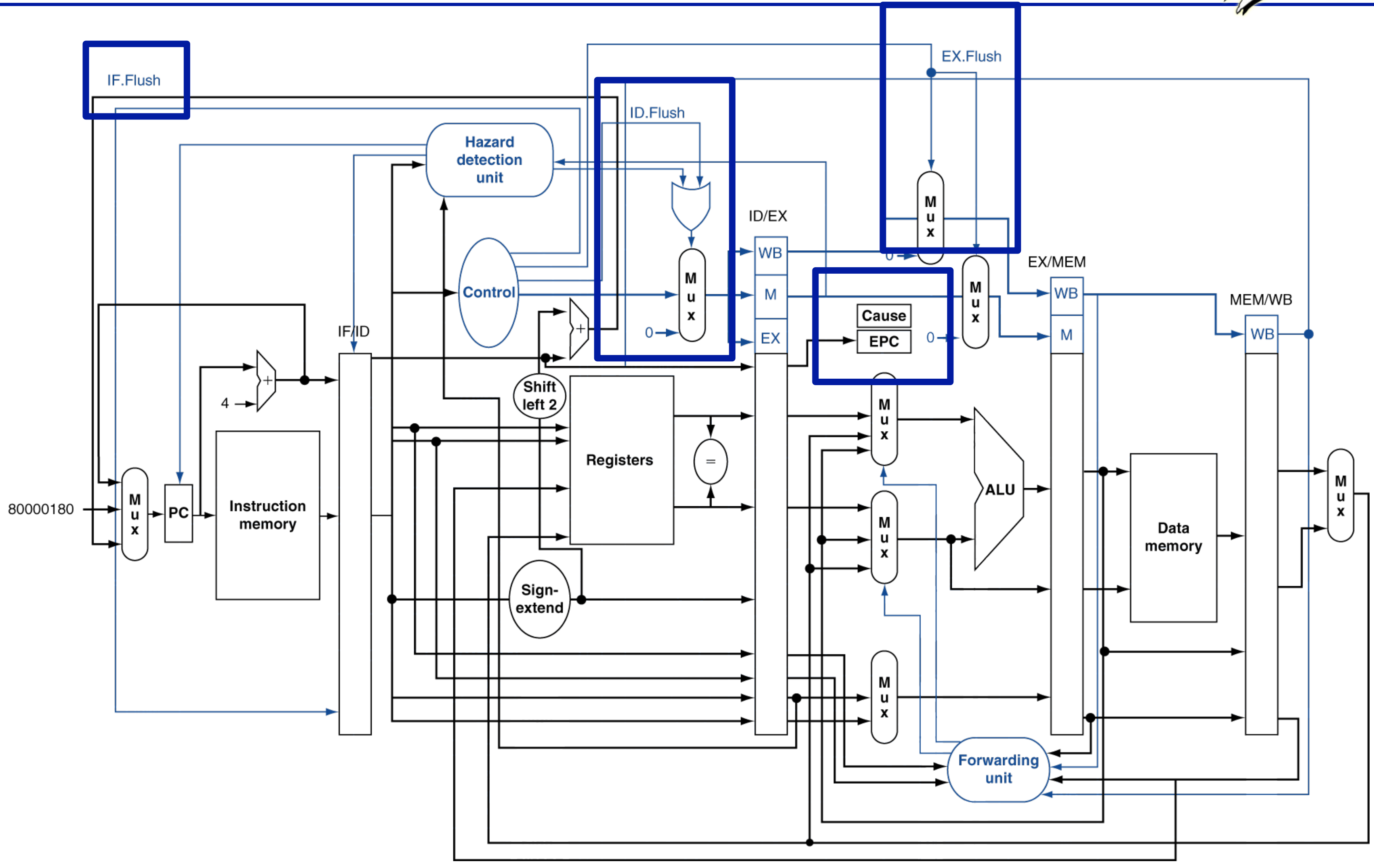
Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



5-stage Pipeline with Exceptions



Beyond the 5-stage Pipeline

Beyond the 5-stage Pipeline

You should take next Fall:

Advanced Computer Architecture CMPE202 (Renau)

Topics: advanced core design, branch prediction, superscalar, symmetric multi-threading

Advanced Parallel Processing CMPE220 (Litz) ☺

Topics: support for parallel programming, GPU architecture, SIMD, many core architectures, accelerators

And a research project

Beyond the 5-stage Pipeline

Desirable properties

- Higher clock frequency
- $\text{CPI}_{\text{base}} < 1$
- Avoid in-order stalls
  - When instruction stalls, independent instructions behind it stall
- Avoid stalls due to branches

How do we go about it?

Modern Processors

High clock frequency → deep pipelines

10 – 20 stages

$CPI_{base} < 1$ → superscalar pipelines

Launch 2, 3, or 4 instructions every cycle

Avoid in-order stalls → out-of-order execution

Re-order instructions based on dependencies

Avoid stalls due to branches → branch prediction

Keep history about direction and target of branches

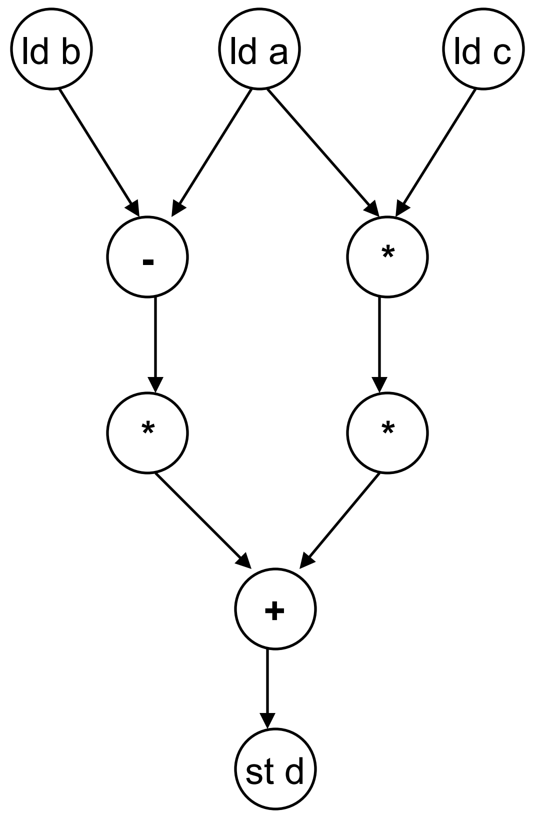
Instruction-Level Parallelism

$$D = 3(a-b) + 7ac$$

- **Sequential execution order**

  ld a
  ld b
  sub a-b
  mul 3(a-b)
  ld c
  mul ac
  mul 7ac
  add 3(a-b)+7ac
  st d

- **Data-flow execution order**



Question

How much ILP is there in common programs?

Limits on Instruction Level Parallelism (ILP)

Weiss and Smith [1984]	1.58
Sohi and Vajapeyam [1987]	1.81
Tjaden and Flynn [1970]	1.86
Tjaden and Flynn [1973]	1.96
Uht [1986]	2.00
Smith et al. [1989]	2.00
Jouppi and Wall [1988]	2.40
Johnson [1991]	2.50
Acosta et al. [1986]	2.79
Wedig [1982]	3.00
Butler et al. [1991]	5.8
Melvin and Patt [1991]	6
Wall [1991]	7
Kuck et al. [1972]	8
Riseman and Foster [1972]	51
Nicolau and Fisher [1984]	90

- These studies made very different assumptions!
- Any guesses on the differences?

Multiple Instruction Issue (Superscalar)

Fetch and execute multiple instructions per cycle => CPI < 1

- Example: fetch 2 instructions/clock cycle
- Dynamically decide if they can go down the pipe together or not (hazard)

Pipe Stages

	IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB	
Ideal CPI = 0.5		IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB
			IF	ID	EX	MEM	WB

Resources: double amount of hardware (FUs, Register file ports)

Issues: hazards, branch delay, load delay

Modern processors: Intel x86: 4-way, ARM: 6-way/8-way

Deeper Pipelining

Increase number of pipeline stages

- Fewer levels of logic per pipe stage
- Higher clock frequency

Example 9-stage pipeline

Pipe Stages

IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB	
	IF1	IF2	ID	EX	MEM1	MEM2	MEM3	WB

Issues: branch delay, load delay: CPI = 1.4 - 2.0, cost of pipeline registers

Modern pipelines

- Pentium: 5 stages
- Original Pentium Pro: 12 stages
- Pentium 4: 31 stages, 3+ GHz
- Tejas: 40-50 stages, 10 GHz, cancelled!
- Core 2: 14 stages

Dynamic Scheduling

Execute instructions out-of-order

Fetch multiple instructions per cycle using branch prediction

Figure out which are independent and execute them in parallel

Example

```
add   $t0, $t1, $t2
or    $t3, $t0, $t2
sub   $t0, $t1, $t2
and   $t5, $t0, $t2
```

Superscalar + Dynamic scheduling

```
add   $t0, $t1, $t2     sub $t0, $t1, $t2
or    $t3, $t0, $t2     and $t5, $t0, $t2
```

What’s wrong with this?

Register Renaming

Rename (map) architectural registers to physical registers in decode stage to get rid of false dependencies

```
add   $t0, $t1, $t2
or    $t3, $t0, $t2
sub   $t0, $t1, $t2
and   $t5, $t0, $t2
```

Superscalar + Dynamic scheduling + register renaming

```
add   $t0A, $t1, $t2    sub $t0B, $t1, $t2
or  $t3, $t0A, $t2      and $t5, $t0B, $t2
```

Need more physical registers than architectural registers

Physical registers are automatically recycled

How do we know which instructions can be scheduled?

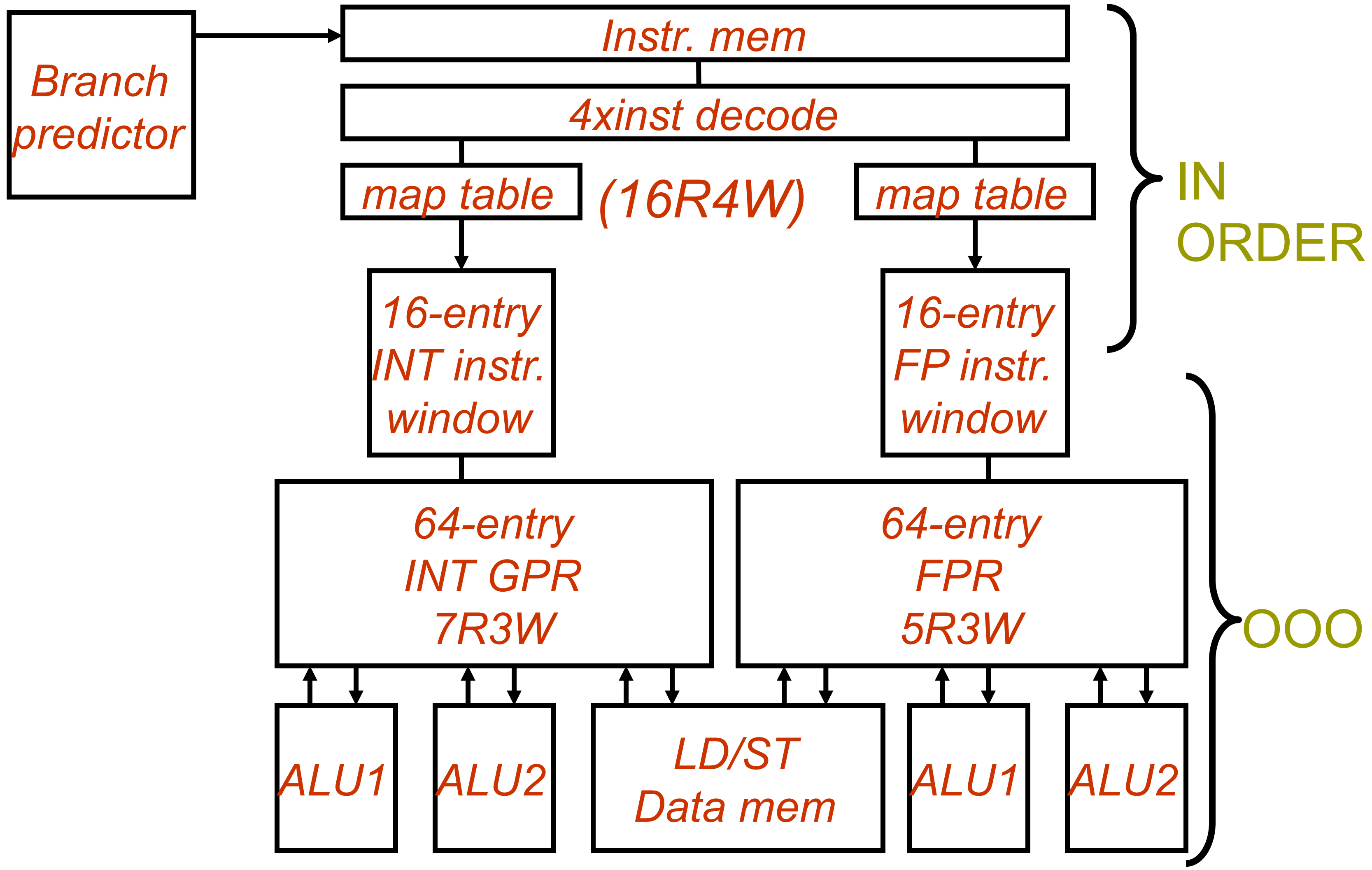
- Need to represent the data flow graph in hardware

Simple Scoreboarding

- Scoreboard: a bit-array, 1-bit for each GPR
  - if the bit is not set, the register has valid data
  - if the bit is set, the register has stale data
    - i.e. some outstanding inst is going to change it
- Dispatch in order: RD ← Fn (RS, RT)
  - if SB[RS] or SB[RT] is set ⇒ RAW, stall
  - if SB[RD] is set is set ⇒ WAW, stall
  - else dispatch to FU, set SB[RD]
    - Assuming a functional unit is available (structural hazard)
- Complete out-of-order
  - update GPR[RD], clear SB[RD]

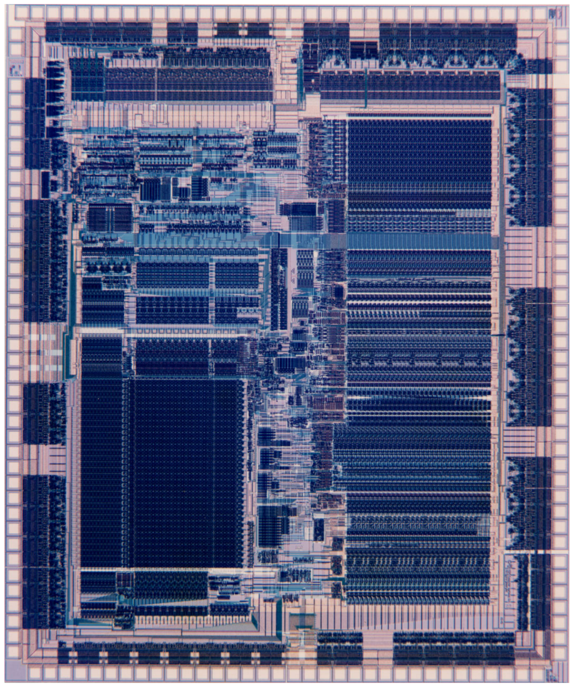
Can be avoided with renaming

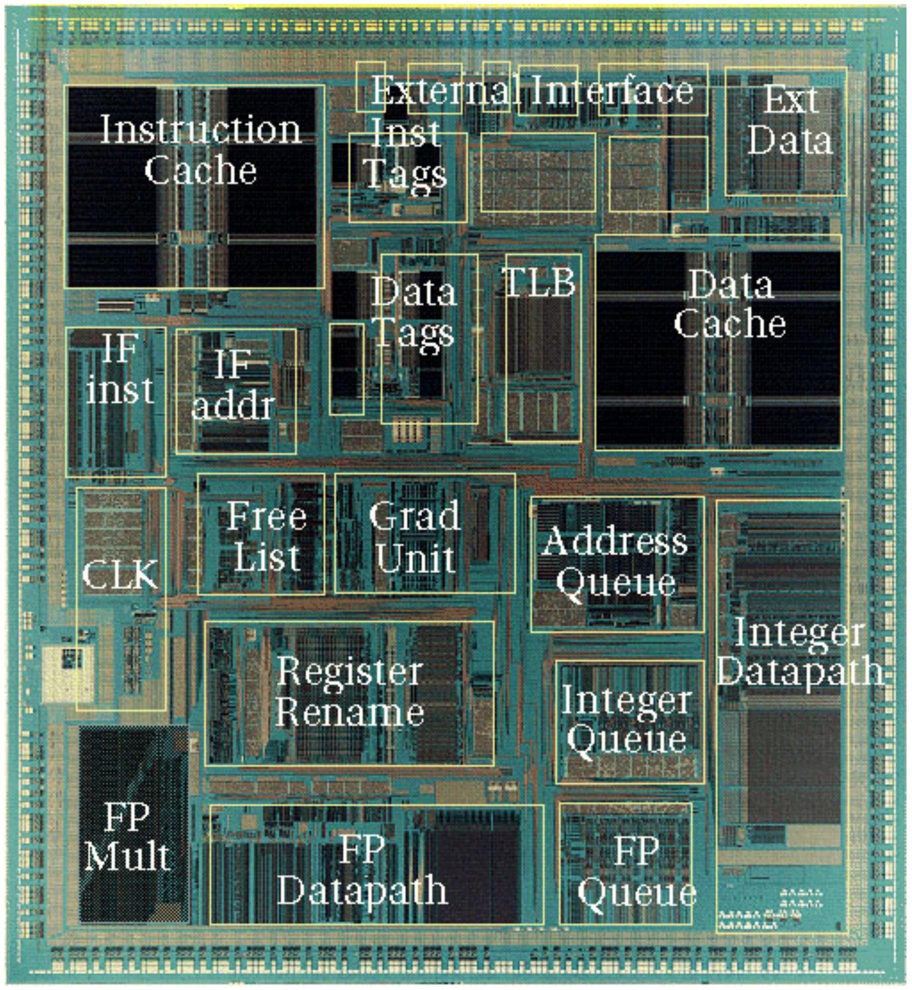
Dynamic Scheduling in a Modern OOO MIPS R10000



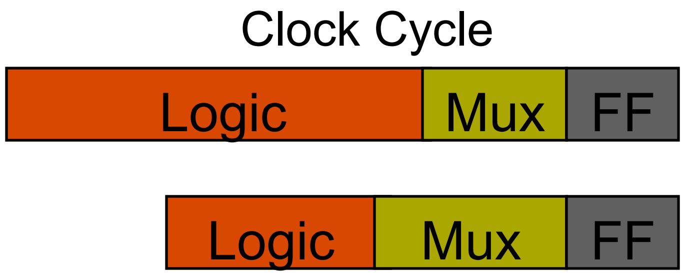
R2000 vs. R10000

- R2000: mid-80's, 32b 5-stage integer pipeline only, 120 K transistors
- R10k: mid-90's, 64b out-of-order superscalar w/ integer & FPU, 64 KB cache, 6.8 M transistors





Limits of Advanced Pipelining



- Limited ILP in real programs
- Pipeline overhead
  - Branch and load delays exacerbated
  - Clock cycle timing limits
- Limited branch prediction accuracy (85%-98%)
  - Even a few percent really hurts with long/wide pipes!
- Memory inefficiency
  - Load delays + # of loads/cycle
  - Caches (next lecture) are not perfect
- Complexity of implementation
  - Expensive to design
  - Why does a superscalar, out-of-order processor burn more power and energy?

Caveats of Out-of-Order Processors

What can go wrong if I reorder instructions?

Tips

- Think of dependencies
- Think of exceptions

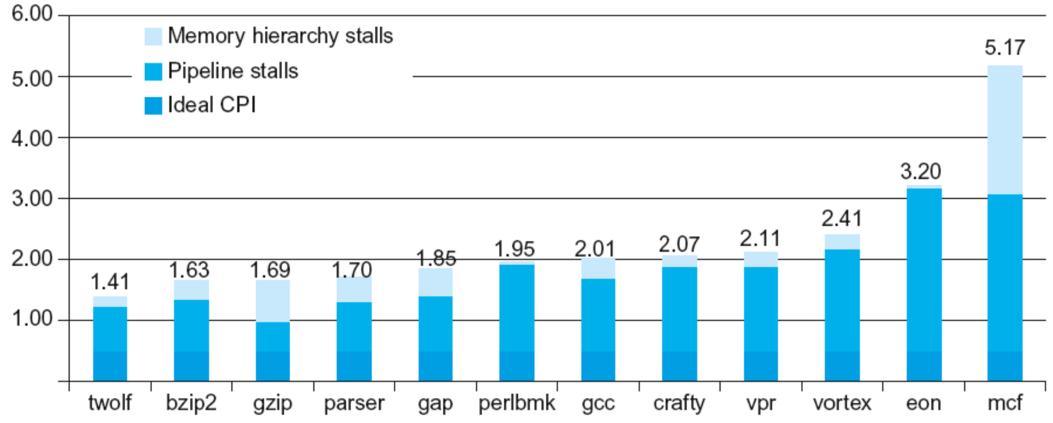
Solutions?

Performance/Power Tradeoffs

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple issue?	Yes	Yes
Peak instructions/clock cycle	2	4
Pipeline stages	14	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	2-level, simple	2-level, aggresive
1st level caches/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-1024 KiB	256 KiB
3rd level caches (shared)	-	2- 8 MB

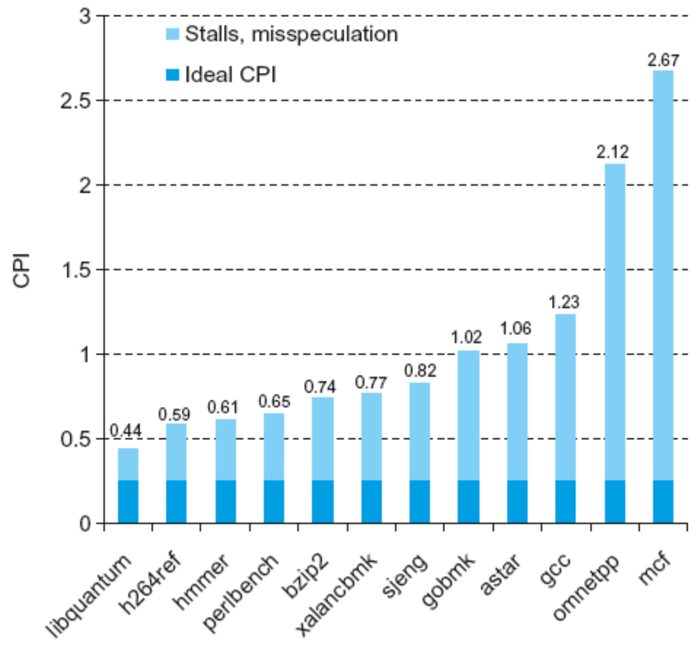
Performance/Power Tradeoffs

ARM A8



Minnespec Benchmarks

Intel i7 920



SPEC 2006 integer benchmarks

Impact of Branch Missprediction

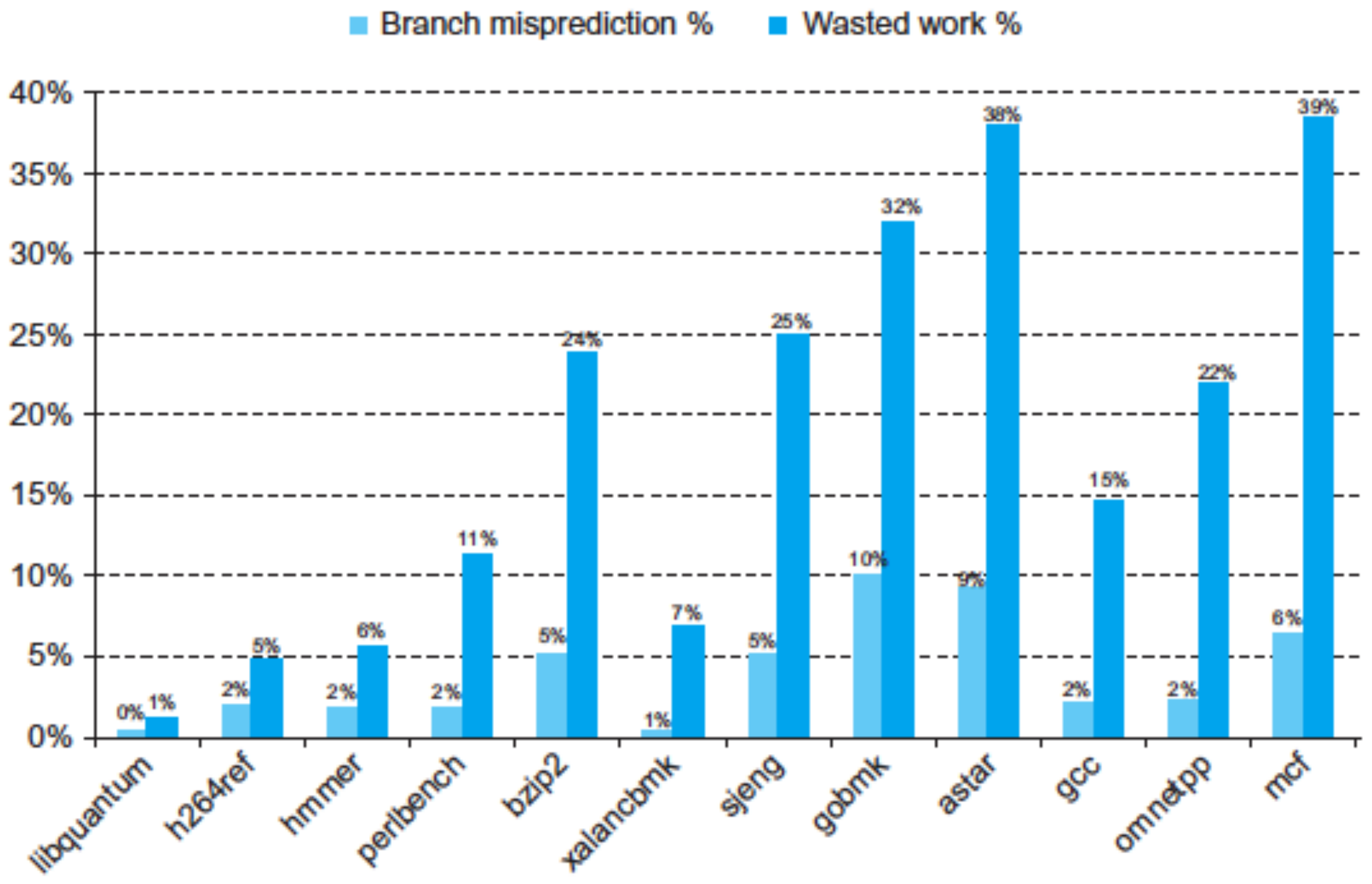
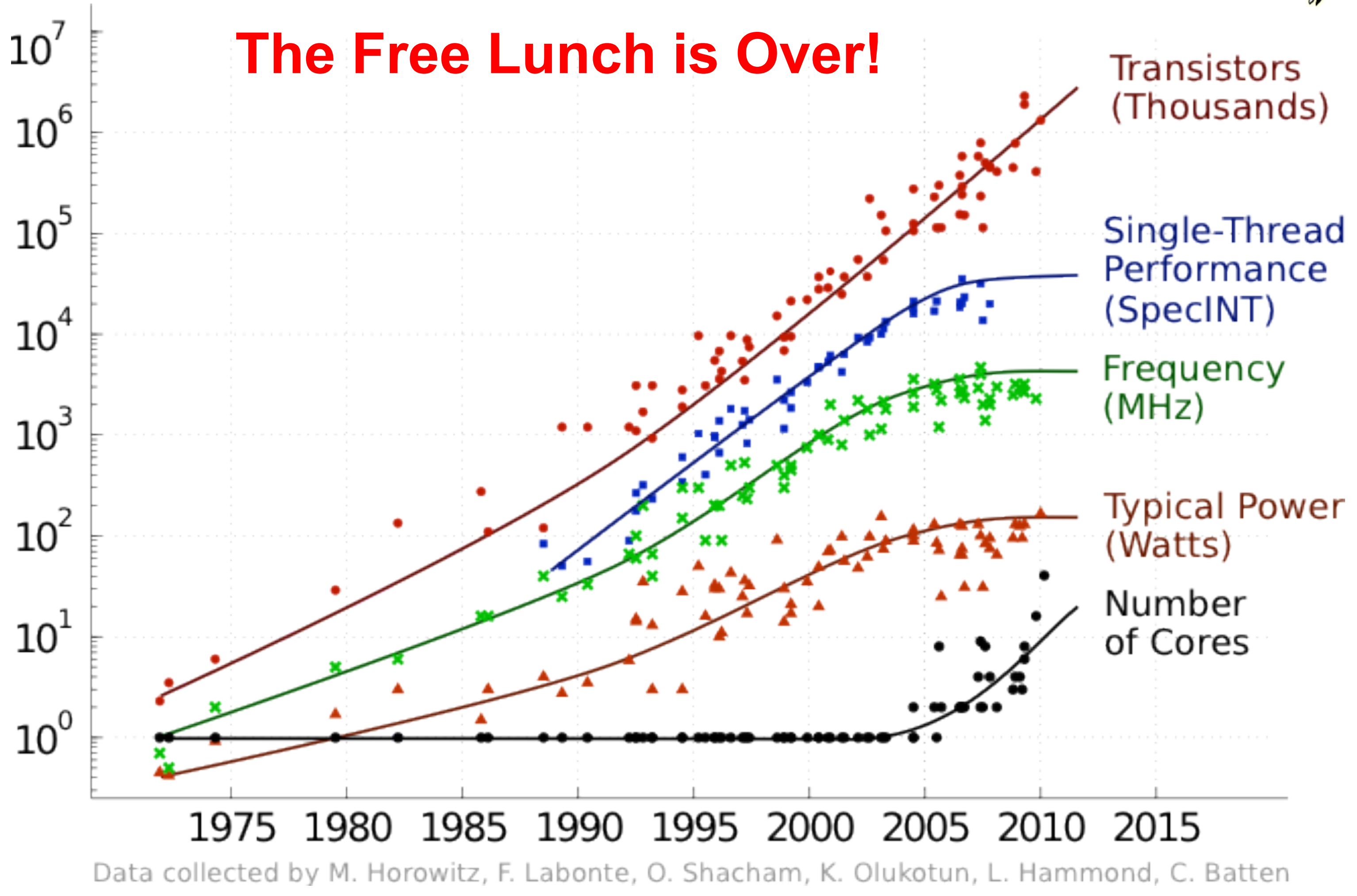


FIGURE 4.79 Percentage of branch mispredictions and wasted work due to unfruitful speculation of Intel Core i7 920 running SPEC2006 integer benchmarks.

End of Uniprocessor Performance Improvements


The Free Lunch is Over!
Transistors (Thousands)
Single-Thread Performance (SpecINT)
Frequency (MHz)
Typical Power (Watts)
Number of Cores
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten