18-447 Lecture 15: A Whirlwind Tour of Modern Microarchitectures

James C. Hoe
Dept of ECE, CMU
March 18, 2009

Announcements: Project, project, project,
Midterm, midterm, midterm
This lecture won't be covered on the midterm or the final

Handouts: H11 Project 3 (on Blackboard)
The Microarchitecture of Superscalar Processors, Smith and Sohi, Proceedings of IEEE, 12/1995. (on Blackboard)
Practice Midterm

Performance Factors

$$T_{wall\text{-}clock} = T_{cyc} \times CPI \times No.Instructions$$

max. combinational delay

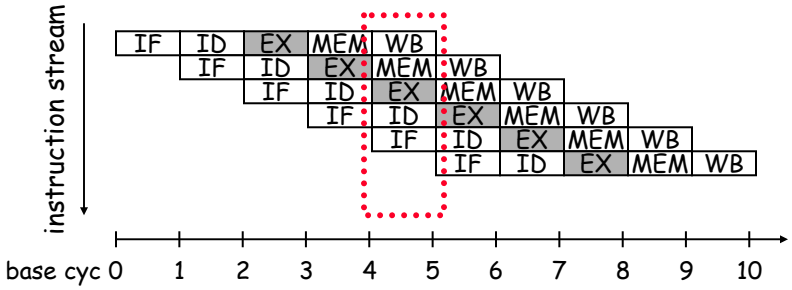
cycles-per-instruction

ISA and compilers

Going after IPC

Scalar Pipeline (baseline)

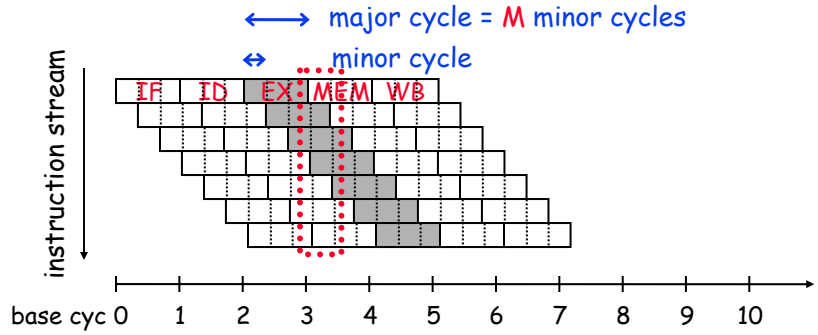
- Operation Latency = 1
- Peak IPC = 1
- Instruction-Level Parallelism = 1



Superpipelined Machine

Superpipelined Execution

- OL = 1 baseline cycle (M minor cycles)
- Peak IPC = M per baseline cycle (1 per minor cycle)
- ILP = M



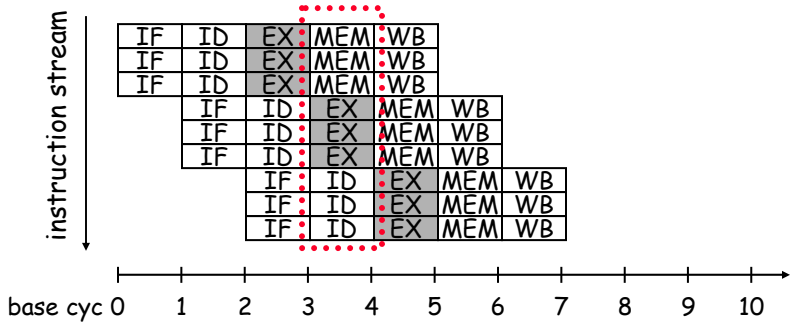
Superscalar Machines

Superscalar (Pipelined) Execution

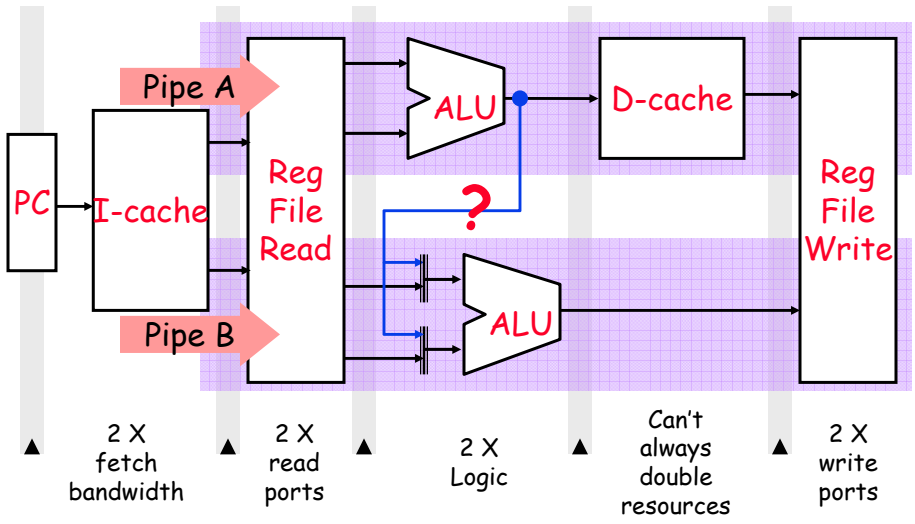
OL = 1 baseline cycles

Peak IPC = N per baseline cycle

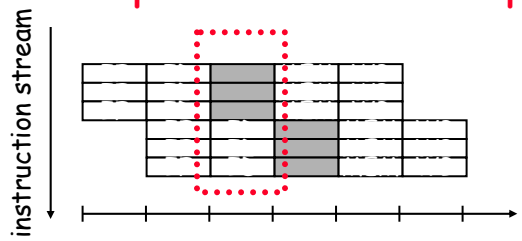
ILP = N



Superscalar Datapath



Superscalar and Superpipelined



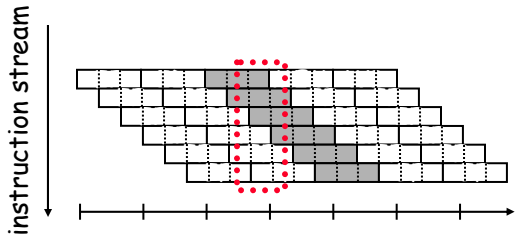
Superscalar Parallelism

Operation Latency: 1

Issuing Rate: N

Superscalar Degree: N

vs.



Superpipeline Parallelism

Operation Latency: 1

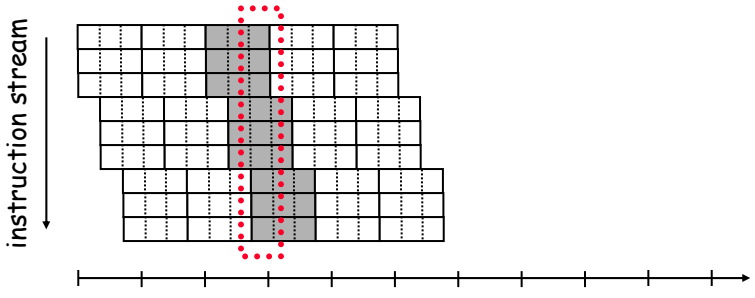
Issuing Rate: M

Superpipelined Degree: M

Achieving peak performance on each architecture depends on finding N or M independent instructions per cycle

Limitations of Inorder Pipelines

- CPI of inorder pipelines degrades very sharply if the machine parallelism is increased beyond a certain point, i.e. when NxM approaches average distance between dependent instructions
- Even with forwarding, pipeline may never be full due to frequent dependency stalls!!



ILP: Instruction-Level Parallelism

- ILP is is a measure of the amount of inter-dependencies between instructions
- Average ILP = no. instruction / no. cyc required

  code1: ILP = 1

  i.e. must execute serially

  code2: ILP = 3

  i.e. can execute at the same time

```
code1:  r1 ← r2 + 1
        r3 ← r1 / 17
        r4 ← r0 - r3
```

```
code2:  r1 ← r2 + 1
        r3 ← r9 / 17
        r4 ← r0 - r10
```


Removing False Dependencies

- Anti and output dependencies are false dependencies

$$r_3 \leftarrow r_1 \text{ op } r_2$$
$$r_5 \leftarrow r_3 \text{ op } r_4$$
$$r_3 \leftarrow r_6 \text{ op } r_7$$

- The dependence is on the register name rather than data
- Given infinite number of registers, anti and output dependencies can always be eliminated

Register Renaming: Example

Original

r1 ← r2 / r3
r4 ← r1 * r5
r1 ← r3 + r6
r3 ← r1 - r5

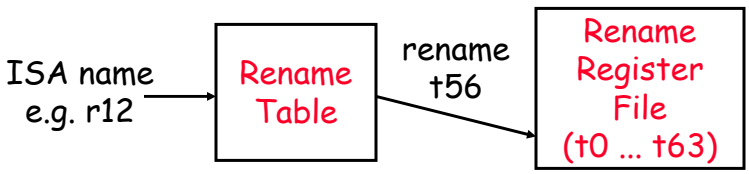
ILP=1

Renamed

r1 ← r2 / r3
r4 ← r1 * r5
r8 ← r3 + r6
r9 ← r8 - r5

ILP=2

Hardware Register Renaming



- maintain bindings from ISA reg. names to rename registers
- When issuing an instruction that updates 'rd':
  - allocate an unused rename register tx
  - recording binding from 'rd' to tx
- When to remove a binding? When to de-allocate a rename register?

r1 ← r2 / r3
r4 ← r1 * r5
r1 ← r3 + r6

Out-of-Order Execution

- Renaming eliminates WAW and WAR
- In a RAW dependent instruction pair, the reader must wait for the result from the writer
- How to get more ILP?

```
r1 ⇐ r2 + 1
r3 ⇐ r1 / 17      ILP=1
r4 ⇐ r0 - r3
r11⇐ r12 + 1      ILP=2
r13⇐ r19 / 17
r14⇐ r0 - r20
```


Dataflow Execution Ordering

- Maintain a window of many pending instructions (a.k.a. Issue Buffer)
- Dispatch instructions out-of-order
  - find instructions whose operands are available
  - give preference to older instructions
  - A completing instruction may enable other pending instructions (RAW)
- Need to remember how to put things back in order (Reorder Buffer)

Instruction Reorder Buffer

- At today's clock frequency, on a memory load
  - a cache hit (best case) takes 4~7 cyc
  - a L1 cache miss takes a few 10s of cycles
  - an off-chip cache miss takes a few 100s of cycles
- ROB is a program-order instruction bookkeeping structure
  - instructions must enter and leave in program order
  - holds 10s to 100s of "in-flight" instructions in various stages of execution
  - re-sorts all instructions on exit to appear to complete in program order
  - supports precise exception for any in-flight instruction

Control Dependencies

Suppose we have an infinitely wide datapath, perfect renaming and an infinitely large issue buffer, what is the limit now??

- Control transfer instructions (branches and jumps) occupy 14% of an avg. instruction mix

Average run length between branches = ??

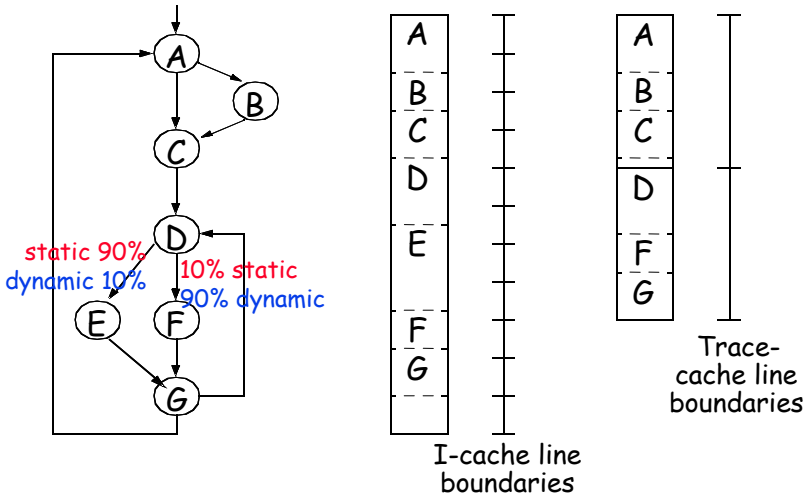
How do we keep the Issue Buffer filled?

Branch Prediction

- Guess the outcome of a branch instruction
- Static Prediction
  - Let the compiler include a hint with each branch
  - 90% of backward branches are taken (loop)
  - 50% of forward branches are taken (if-else)
- Dynamic History-Based Prediction
  - Past behavior is a good predictor of future actions
  - Predicts not only the direction of the branch but the target of the branch

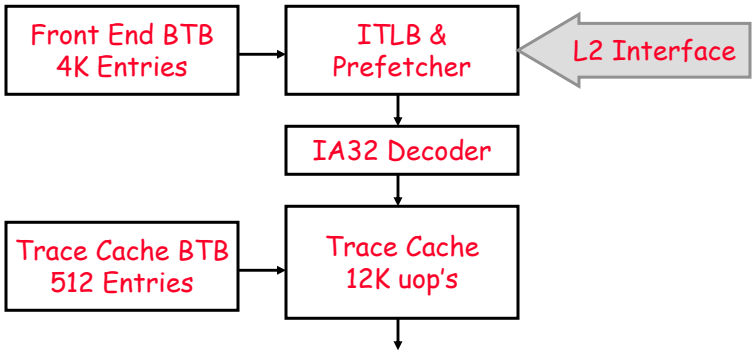
  Modern techniques get better than 95% accuracy
- Trace Caching

Trace Caching



Intel P4 Trace Cache

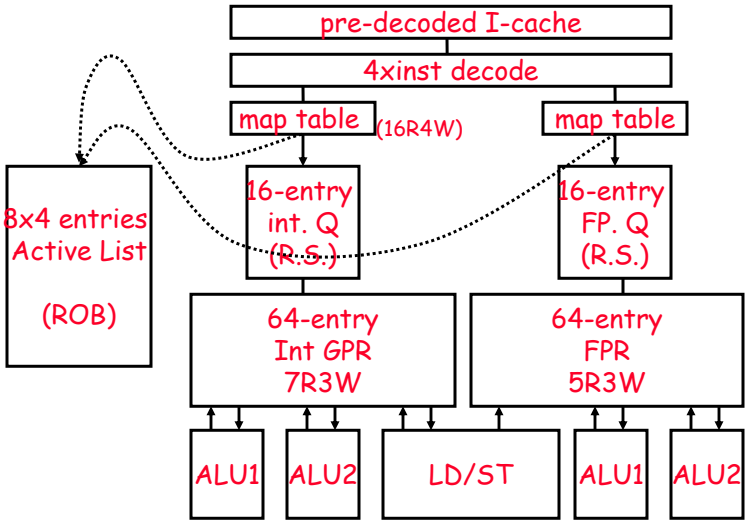
- A 12K-uop trace cache replaces the L1 I-cache
- 6-uop per trace line, can include branches
- Trace cache returns 3-uop per cycle
- IA-32 decoder can be simpler and slower



Speculative Execution

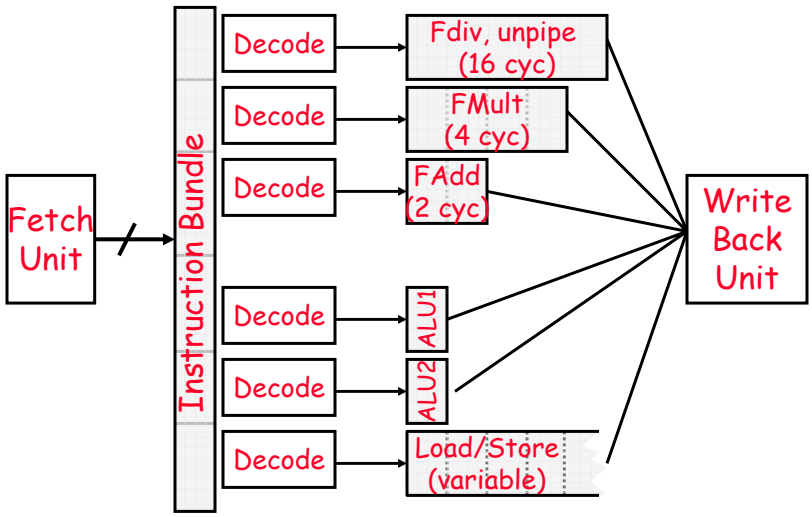
- Instructions after a predicted branch are speculative
  - ⇒ Must have ways to undo their effects
- Maintain separate copies of
  - In-order State: a check-point state up to just before the first speculated instruction
  - Speculative State: include all state changes after check-point, possibly multiple predicted branches
- Commit - admit known-to-be good speculative state changes into the in-order state
- Rewind - discard all, or part of, the speculative state

MIPS R10000

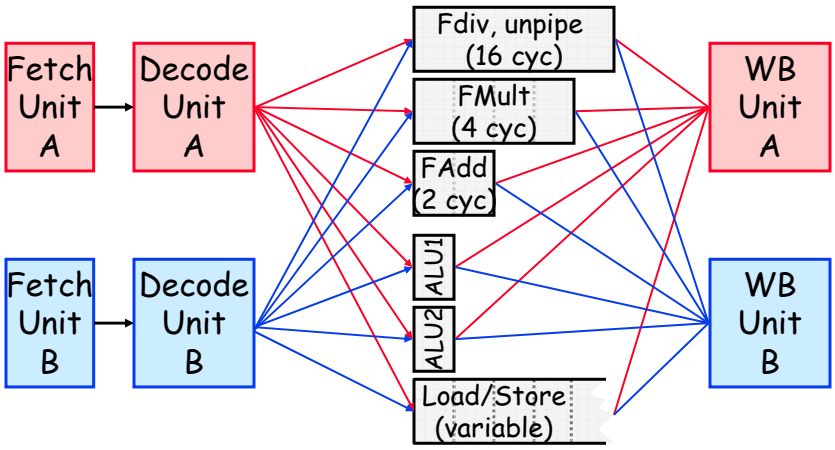


Read [Yeager 1996, IEEE Micro] if you are really interested

In Vogue: Very Long Instruction Words

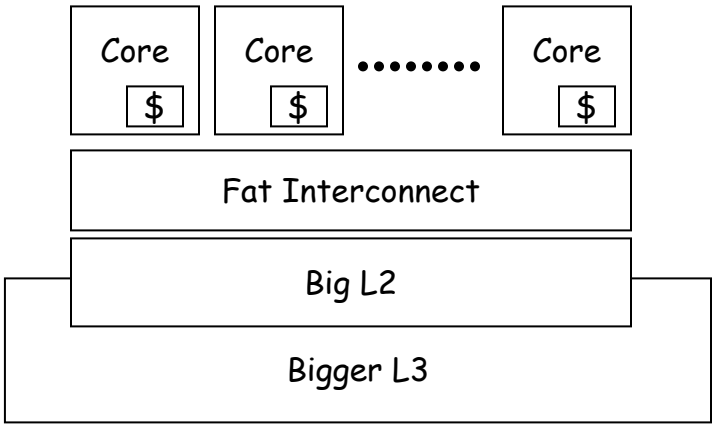


In Vogue: Simultaneous Multi-Threading



How do you get more performance by letting two threads share the same functional units?

In Vogue: Chip-Multiprocessor



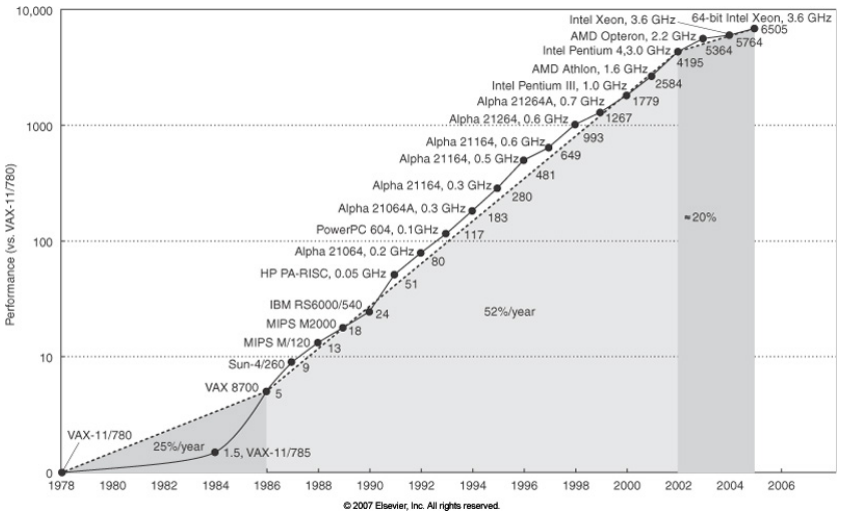
It is a power issue. How to get more performance without increasing clock frequency.....

State of the Art

	AMD Opteron 8360SE	Intel Xeon X7460	Intel Itanium 9050	IBM P5	IBM P6	Fijitsu SPARC 7	SUN T2
cores/threads	4x1	6x1	2x2	2x2	2x2	4x2	8x8
Clock (GHz)	2.5	2.67	1.60	2.2	5	2.52	1.8
Issue Rate	3 (x86)	4 (rop)	6	5	7	4	2
Pipeline depth	12/17	14	8	15	13	15	8/12
Out-of-order	72(rop)	96(rop)	inorder	200	limited	64	inorder
on-chip$ (MB)	2+2	9+16	1+12	1.92	8	6	4
Trans (10^6)	463	1900	1720	276	790	600	503
Power (W)	105	130	104	100	>100	135	95
SPECint 2006 per-core/total	14.4/170	22/274	14.5/1534	10.5/197	15.8/1837	10.5/2088	--/142
SPECfp 2006 per-core/total	18.5/156	22/142	17.3/1671	12.9/229	20.1/1822	25.0/1861	--/111

Microprocessor Report, Oct 2008

Moore's Law?



One thing to remember is that without the extensive instruction reordering and speculation in modern pipelines, we would not be able to run the CPU at muliti-GHz against 50ns DRAM cycles

New Challenges: Power

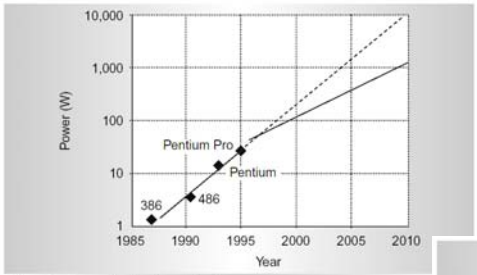


Figure 8. Power dissipation projections.

Think about power and current density! The chips are not getting any bigger.

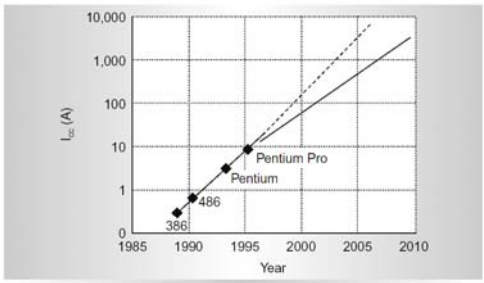
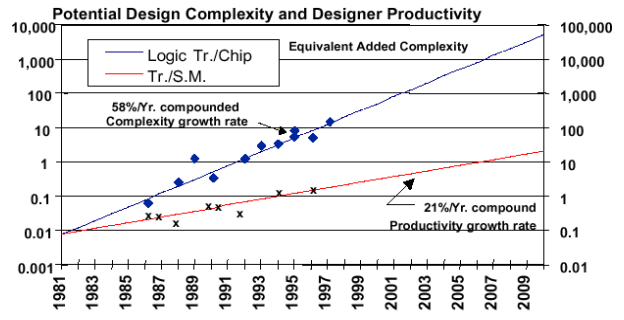


Figure 9. Supply current projections.

[Shekhar Borkar, IEEE Micro, July 1999]

New Challenges: Complexity



Year	Technology	Chip Complexity	Frequency	3 Yr. Design Staff	Staff Cost*
1997	250 nm	13 M Tr.	400	210	90 M
1998	250 nm	20 M Tr.	500	270	120 M
1999	180 nm	32 M Tr.	600	360	160 M
2002	130 nm	130 M Tr.	800	800	360 M

* @ $150K / StaffYr. (In 1997 Dollars)

[International Roadmap for Semiconductor 1999: Design]

New Challenges: Reliability

- Soft Errors: errors not caused by a permanent defect in hardware, e.g., energy from a cosmic ray strike flipping a bit in a register or SRAM
- Memory arrays are protected by parity or error-correcting code (ECC)
- What about registers and logic in deep submicron?
  - low node capacitance
  - low voltage
  - lots and lots of them on a chip

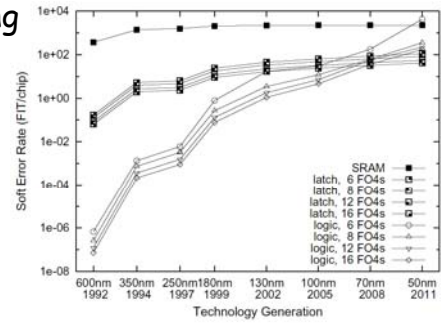


Figure 7. SER/chip for SRAM/latches/logic

[P.Shivakumar, et al. DSN'02]