

January 28, 2025

## Announcements

- Talk topic, due Friday
- HW1 Thu

Review

---

## Goals

- towards first principles
- assembly

## Moore's Law

Moore's Law: The number of transistors on microchips has doubled every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World  
in Data



## Issue: More transistors = faster?

$$\frac{\text{Work}}{\text{s}} = \cancel{\text{Clock Frequency}} \times \underline{\text{Work/Clock}}$$

Data source: Wikipedia ([wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/w/index.php?title=Transistor_count&oldid=1000000000))

[OurWorldInData.org](http://OurWorldInData.org) - Research and data to make progress against the world's largest problems

# Dennard Scaling of MOSFETs

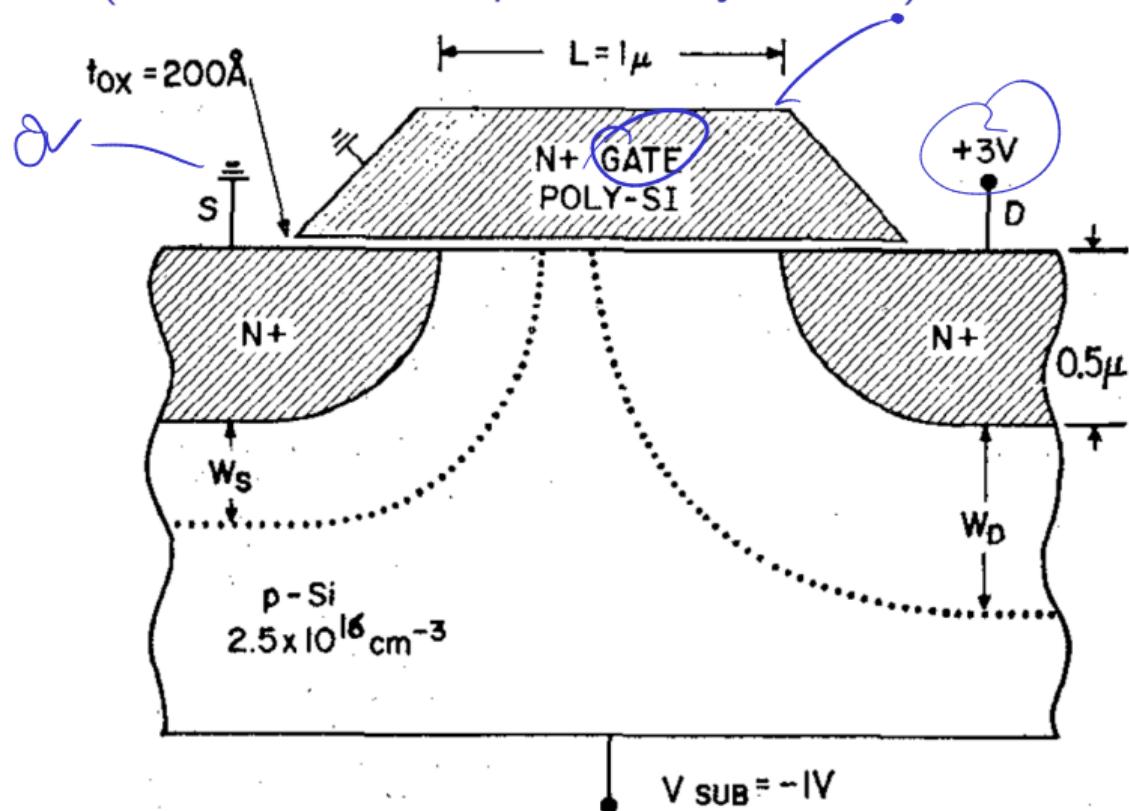
1D	Parameter	Factor
Dimension	$1/\kappa$	$\Rightarrow$ area $\sim \frac{1}{\kappa^2}$
Voltage	$1/\kappa$	
Current	$1/\kappa$	
Capacitance	$1/\kappa$	
Delay Time	$1/\kappa$	
Power dissipation/circuit	$1/\kappa^2$	
Power density	1	

[Dennard et al. '74, via Bohr '07]

Frequency = Delay time<sup>-1</sup>

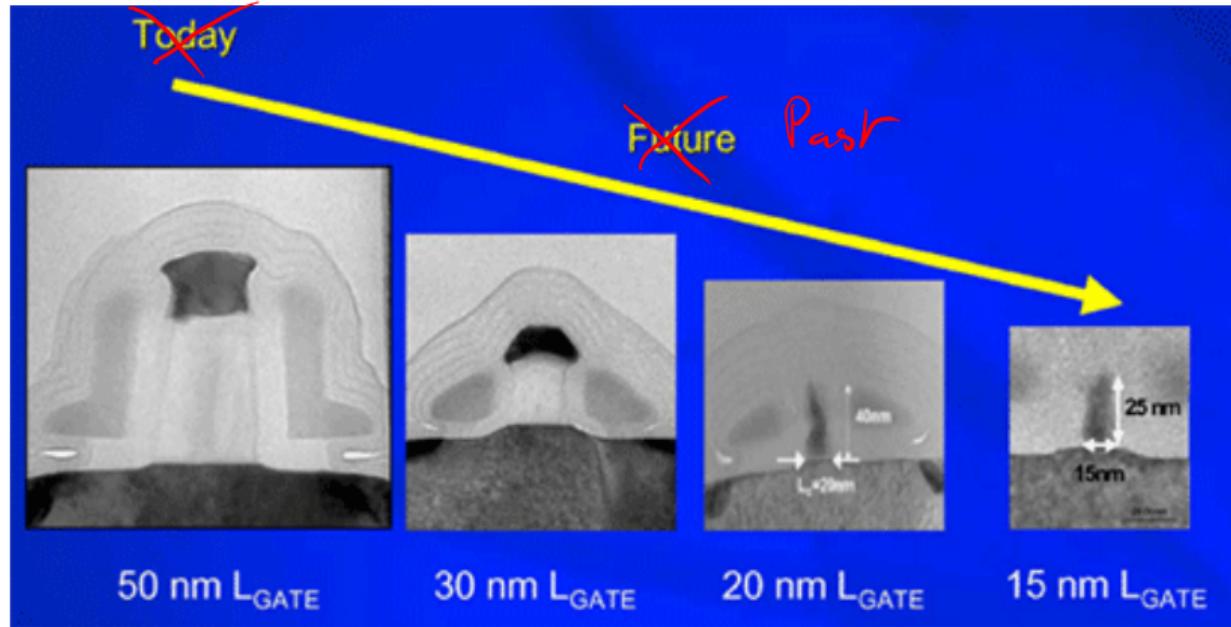


## MOSFETs ("CMOS" – "complementary" MOS): Schematic



[Dennard et al. '74]

# MOSFETs: Scaling



[Intel Corp.]

- ▶ 'New' problem at small scale:
  - Sub-threshold leakage (due to low voltage, small structure)
  - Dennard scaling is over – and has been for a while.

## Peak Architectural Instructions per Clock: Intel

CPU	IPC	Year
Pentium 1	1.1	1993
Pentium 3	1.9	1999
Pentium 4 (Gallatin)	1.9	20
Pentium D	2	2005
Pentium M	2.5	2003
Core 2	3	2006
Sandy Bridge...	<4	2011
Skylake	<4	2015
Golden Cove	<6	2021
Lion Cove	<8	2024

[Charlie Brej <http://brej.org/blog/?p=15>, Wikipedia, Intel]

Context: [Lemire: simdjson achieved IPC, '19](#)

Discuss: How do we get out of this dilemma?

# The Performance Dilemma

- ▶ IPC: Brick-ish Wall
- ▶ Clock Frequency: Brick Wall

Ideas:



- SIMD
- SPMW
- single instruction multiple data
- single program multiple data

Question: What is the *conceptual* difference between those ideas?

shared mem  
dist. mem

## The Performance Dilemma: Another Look

- ▶ **Really:** A crisis of the 'starts-at-the-top-ends-at-the-bottom' programming model
- ▶ **Tough luck:** Most of our codes are written that way ↪
- ▶ **Even tougher luck:** Everybody on the planet is *trained* to write codes this way ↪

So:

- ▶ **Need:** Different tools/abstractions to write those codes

# Outline

## Introduction

Notes

Notes (unfilled, with empty boxes)

Notes (source code on Github)

About This Class

Why Bother with Parallel Computers?

## Lowest Accessible Abstraction: Assembly

Architecture of an Execution Pipeline

Architecture of a Memory System

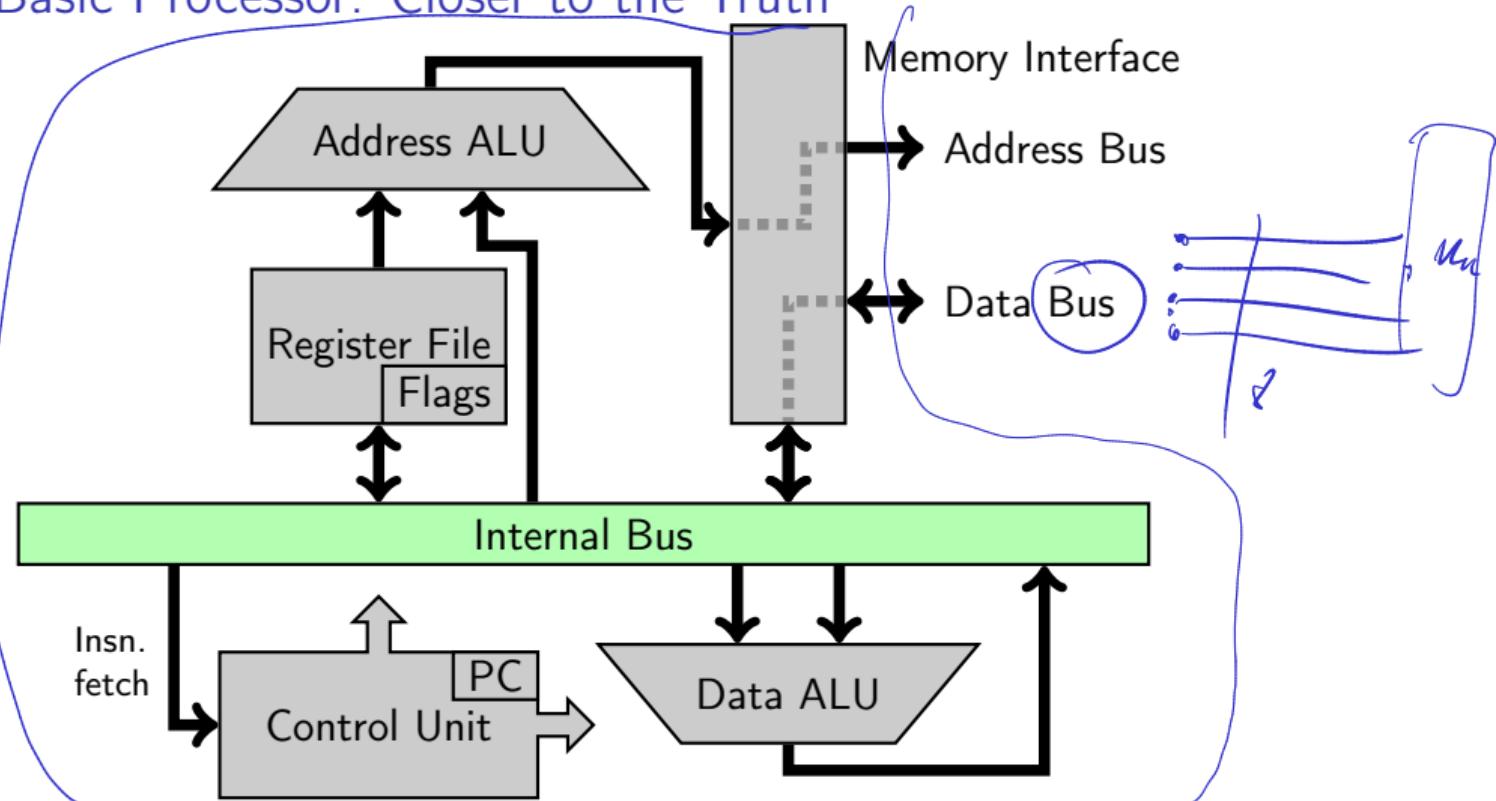
Shared-Memory Multiprocessors

## Machine Abstractions

## Performance: Expectation, Experiment, Observation

## Performance-Oriented Languages and Abstractions

# A Basic Processor: Closer to the Truth



- ▶ loosely based on Intel 8086
- ▶ What's a bus?

$\times 86$  ; 196, 286, 386  
486

# A Very Simple Program

```
int a = 5;  
int b = 17;  
int z = a * b;
```

little Endian ?  
↓

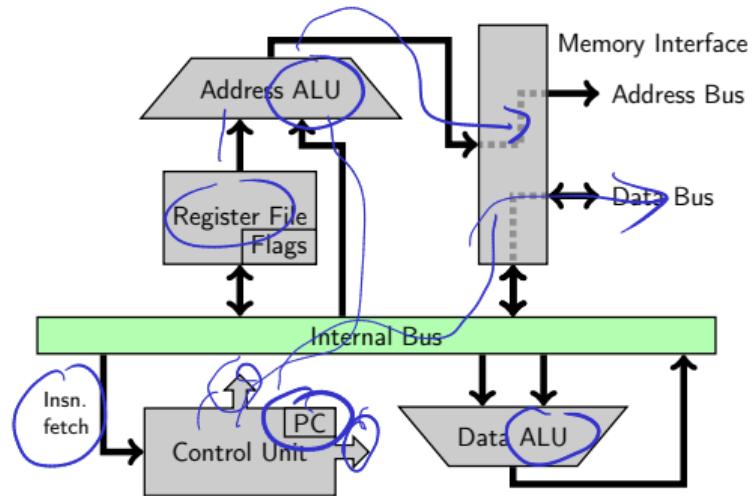
4:	c7 45 f4 05 00 00 00	movl
b:	c7 45 f8 11 00 00 00	movl
12:	8b 45 f4	mov
15:	0f af 45 f8	imul
19:	89 45 fc	mov
1c:	8b 45 fc	mov

\$0x5,-0xc(%rbp)  
\$0x11,-0x8(%rbp)  
-0xc(%rbp),%eax  
-0x8(%rbp),%eax  
%eax,-0x4(%rbp)  
-0x4(%rbp),%eax

Things to know:

- ▶ Question: Which is it?
  - ▶ <opcode> <src>, <dest>
  - ▶ <opcode> <dest>, <src>
- ▶ Addressing modes (Immediate, Register, Base plus Offset)
- ▶ 0xHexadecimal

## A Very Simple Program: Another Look



4: c7 45 f4 05 00 00 00 movl \$0x5,-0xc(%rbp)  
b: c7 45 f8 11 00 00 00 movl \$0x11,-0x8(%rbp)  
12: 8b 45 f4 mov -0xc(%rbp),%eax  
15: 0f af 45 f8 imul -0x8(%rbp),%eax  
19: 89 45 fc mov %eax,-0x4(%rbp)  
1c: 8b 45 fc mov -0x4(%rbp),%eax

## A Very Simple Program: Intel Form

4:	c7 45 f4 05 00 00 00	mov	0	DWORD PTR [rbp-0xc],0x5
b:	c7 45 f8 11 00 00 00	mov		DWORD PTR [rbp-0x8],0x11
12:	8b 45 f4	mov		eax,DWORD PTR [rbp-0xc]
15:	0f af 45 f8	imul		eax,DWORD PTR [rbp-0x8]
19:	89 45 fc	mov		DWORD PTR [rbp-0x4],eax
1c:	8b 45 fc	mov		eax,DWORD PTR [rbp-0x4]

- ▶ “Intel Form”: (you might see this on the net)  
<opcode> <sized dest>, <sized source>
- ▶ Previous: “AT&T Form”: (we’ll use this)
- ▶ Goal: Reading comprehension.
- ▶ Don’t understand an opcode?

[https://en.wikipedia.org/wiki/X86\\_instruction\\_listings](https://en.wikipedia.org/wiki/X86_instruction_listings)

# Assembly Loops

```
int main()
{
    int y = 0, i;
    for (i = 0;
        y < 10; ++i)
        y += i;
    return y;
}
```

0:	55
1:	48 89 e5
4:	c7 45 f8 00 00 00 00 00
b:	c7 45 fc 00 00 00 00 00
12:	eb 0a
14:	8b 45 fc
17:	01 45 f8
1a:	83 45 fc 01
1e:	83 7d f8 09
22:	7e f0
24:	8b 45 f8
27:	c9
28:	c3

push	%rbp
mov	%rsp,%rbp
movl	\$0x0,-0x8(%rbp)
movl	\$0x0,-0x4(%rbp)
jmp	1e <main+0x1e>
mov	-0x4(%rbp),%eax
add	%eax,-0x8(%rbp)
addl	\$0x1,-0x4(%rbp)
cmpl	\$0x9,-0x8(%rbp)
jle	14 <main+0x14>
mov	-0x8(%rbp),%eax
leaveq	
retq	

Things to know:

- ▶ Condition Codes (Flags): Zero, Sign, Carry, etc.
- ▶ Call Stack: Stack frame, stack pointer, base pointer
- ▶ ABI: Calling conventions

Application binary interface

## Demos

### Demo: intro/Assembly Reading Comprehension

Demo: Source-to-assembly mapping

Code to try:

```
int main()
{
    int y = 0, i;
    for (i = 0; y < 100; ++i)
        y += i*i;
    return y;
}
```

Also try <https://godbolt.org> for direct source-to-assembly mapping