January 28, 2025

Announcements

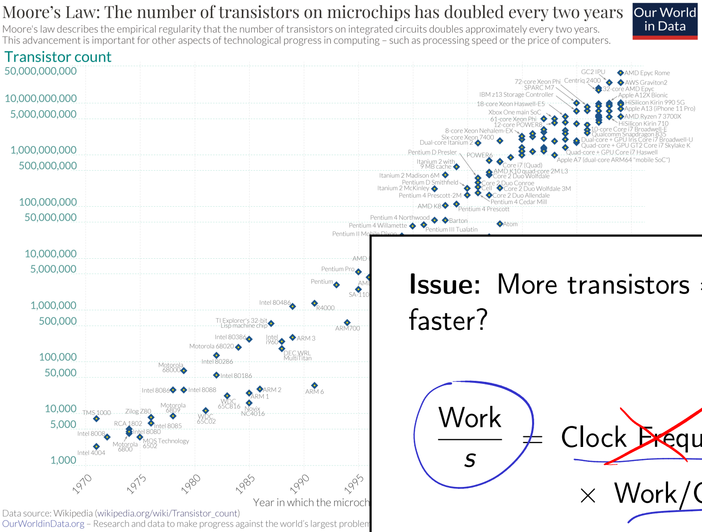
- Talk topic, due Friday
- HW1 Thu

Goals

- towards first principles
- assembly

Review

Moore’s Law



Issue: More transistors = faster?

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

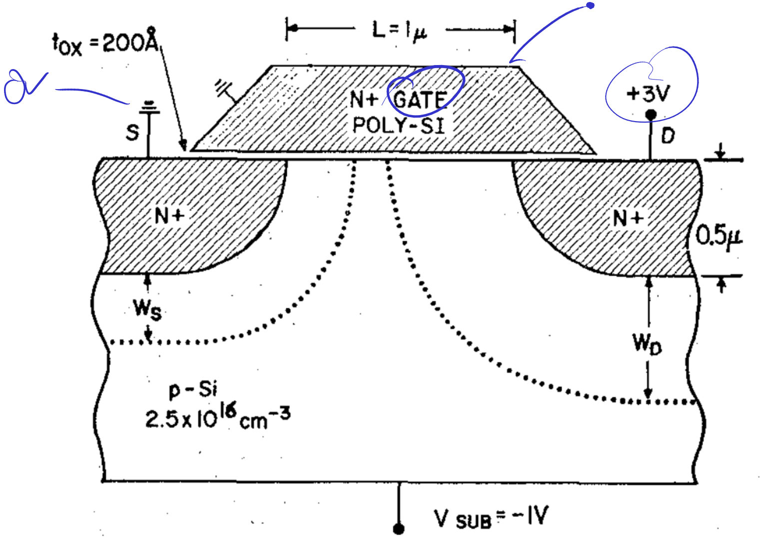
Dennard Scaling of MOSFETs

Parameter	Factor
Dimension	$1/\kappa$
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]

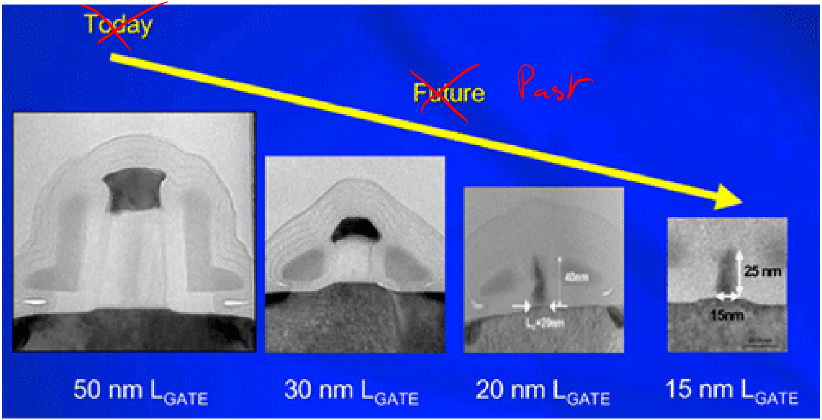
- Frequency = Delay time^{-1}

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 – single instruction multiple data
- SPMD – single program multiple data "MPI"
  - shared me
  - dist. me

key trade-off, control flow

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

The Performance Dilemma: Another Look

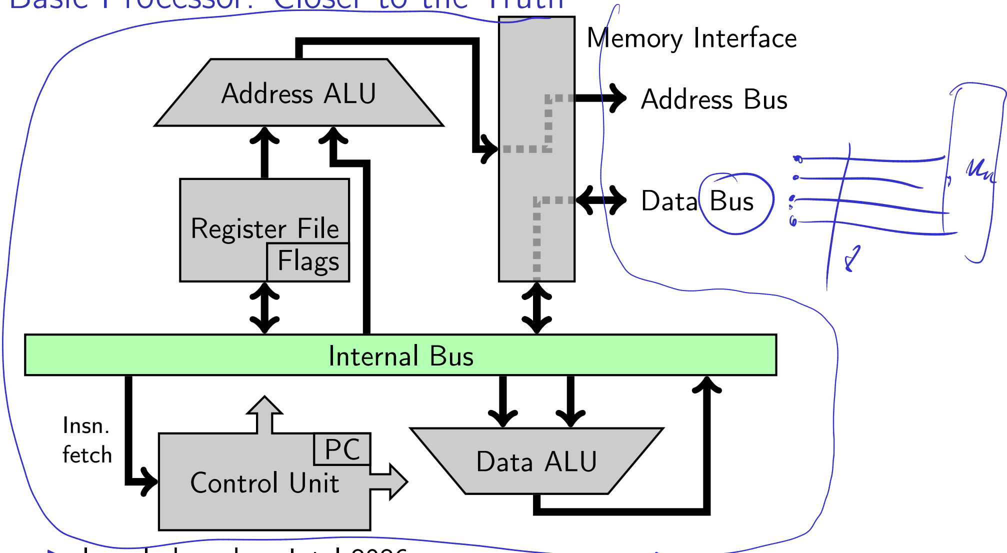
- Really: A crisis of the 'starts-at-the-top-ends-at-the-bottom' prorgramming model
- Tough luck: Most of our codes are written that way
- Even tougher luck: Everybody on the planet is *trained* to write codes this way

So:

- Need: Different tools/abstractions to write those codes

Outline

A Basic Processor: Closer to the Truth



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

A Very Simple Program

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

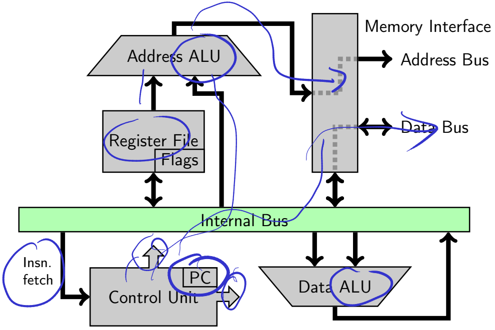
```
4:   c7 45 f4 05 00 00 00 movl  $0x5,-0xc(%rbp)
b:   c7 45 f8 11 00 00 00 movl  $0x11,-0x8(%rbp)
12:  8b 45 f4             mov   -0xc(%rbp),%eax
15:  0f af 45 f8          imul  -0x8(%rbp),%eax
19:  89 45 fc             mov   %eax,-0x4(%rbp)
1c:  8b 45 fc             mov   -0x4(%rbp),%eax
```

little Endian ?

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    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

Assembly Loops

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

```
0:    55                      push   %rbp
1:    48 89 e5                mov    %rsp,%rbp
4:    c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
b:    c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
12:   eb 0a                   jmp    1e <main+0x1e>
14:   8b 45 fc                mov    -0x4(%rbp),%eax
17:   01 45 f8                add    %eax,-0x8(%rbp)
1a:   83 45 fc 01             addl   $0x1,-0x4(%rbp)
1e:   83 7d f8 09             cmpl   $0x9,-0x8(%rbp)
22:   7e f0                   jle    14 <main+0x14>
24:   8b 45 f8                mov    -0x8(%rbp),%eax
27:   c9                      leaveq
28:   c3                      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