The von Neumann Machine

1945: John von Neumann
  - Wrote a report on the stored program concept, known as the First Draft of a Report on EDVAC
  - also Alan Turing… Konrad Zuse… Eckert & Mauchly…

The basic structure proposed in the draft became known as the “von Neumann machine” (or model).
  - a memory, containing instructions and data
  - a processing unit, for performing arithmetic and logical operations
  - a control unit, for interpreting instructions
The von Neumann Machine

Abstraction of von Neumann Architecture

- **MEMORY**
  - MAR
  - MDR
  - stores both program instructions and data

- **INPUT**
  - Keyboard
  - Mouse
  - Scanner
  - Card reader
  - Disk

- **PROCESSING UNIT**
  - ALU
  - TEMP
  - decodes current instruction, manages processing unit to carry out instruction

- **CONTROL UNIT**
  - PC
  - IR
  - program counter: points to the next instruction to be fetched
  - instruction register: stores current instruction

- **OUTPUT**
  - Monitor
  - Printer
  - LED
  - Disk
Intel x86 History

Totally dominate laptop/desktop/server market

Evolutionary design
- Backwards compatible up until 8086, introduced in 1978
- Added more features as time goes on

Complex instruction set computer (CISC)
- Many different instructions with many different formats
  - But, only small subset encountered with Linux programs
- Hard to match performance of Reduced Instruction Set Computers (RISC)
- But, Intel has done just that!
  - In terms of speed. Less so for low power.
## Intel x86 History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>29K</td>
<td>5-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80386</td>
<td>1985</td>
<td>275K</td>
<td>16-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium 4F</td>
<td>2004</td>
<td>125M</td>
<td>2800-3800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core i7</td>
<td>2008</td>
<td>731M</td>
<td>2667-3333</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel x86 History

Architectures
- X86-16
- X86-32/IA32
  - MMX
  - SSE
  - SSE2
  - SSE3
- X86-64 / EM64t
  - SSE4

Processors
- 8086
- 286
- 386
- 486
- Pentium
- Pentium MMX
- Pentium III
- Pentium 4
- Pentium 4E
- Pentium 4F
- Core 2 Duo
- Core i7
Intel x86 History

Intel Attempted Radical Shift from IA32 to IA64
- Totally different architecture (Itanium)
- Executes IA32 code only as legacy
- Performance disappointing

AMD Stepped in with Evolutionary Solution
- x86-64 (now called “AMD64”)

Intel Felt Obligated to Focus on IA64
- Hard to admit mistake or that AMD's approach is better

2004: Intel Announces EM64T extension to IA32
- Extended Memory 64-bit Technology
- Almost identical to x86-64!

All but low-end x86 processors support x86-64
- But, lots of code still runs in 32-bit mode
von Neumann View of x86

CPU
- PC
- Registers
- Condition Codes

Memory
- Addresses
- Object Code
- Program Data
- OS Data
- Stack

Data
Instructions

x86 Overview 7
High-level x86 CPU

Programmer-Visible State

- **PC**: Program counter
  - Address of next instruction
  - Called “EIP” (IA32) or “RIP” (x86-64)

- **Register file**
  - Heavily used program data

- **Condition codes**
  - Store status information about most recent arithmetic operation
  - Used for conditional branching
IA32 Integer Registers

<table>
<thead>
<tr>
<th>CPU</th>
<th>Registers</th>
<th>16-bit virtual registers (backwards compatibility)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Condition Codes</td>
<td>%eax %ax %ah %al</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%ecx %cx %ch %cl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%edx %dx %dh %dl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%ebx %bx %bh %bl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%esi %si</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%edi %di</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%esp %sp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%ebp %bp</td>
</tr>
</tbody>
</table>
## x86-64 Integer Registers

<table>
<thead>
<tr>
<th>x86</th>
<th>x64</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>%rax</td>
</tr>
<tr>
<td>%rbx</td>
<td>%rbx</td>
</tr>
<tr>
<td>%rcx</td>
<td>%rcx</td>
</tr>
<tr>
<td>%rdx</td>
<td>%rdx</td>
</tr>
<tr>
<td>%rsi</td>
<td>%rsi</td>
</tr>
<tr>
<td>%rdi</td>
<td>%rdi</td>
</tr>
<tr>
<td>%rsp</td>
<td>%esp</td>
</tr>
<tr>
<td>%rbp</td>
<td>%ebp</td>
</tr>
</tbody>
</table>

- Extend existing registers. Add 8 new ones.
- Make %ebp/%rbp general purpose
High-level x86 Memory

- **Memory**
  - Byte addressable array
  - Code, user data, (some) OS data
  - Includes stack used to support procedures

```
Memory

Object Code
Program Data
OS Data

Stack
```
Programming the Machine

// C code
... 
int imax(int first, int second) {
    if (first >= second) 
        return first;
    return second;
}

But the hardware only "understands" binary representations
Programming the Machine

```c
int imax(int first, int second) {
    if ( first >= second )
        return first;
    return second;
}
```

```bash
gcc -O0 -c -m64 -std=c99 imax.c
```

But who wants to program in binary?
int imax(int first, int second) {
    if (first >= second)
        return first;
    return second;
}

Solution:
translate high-level language code into intermediate-level code which is more human-friendly,
then translate that "assembly" code into the machine's language.

```assembly
.imax:
pushq %rbp
movq %rsp, %rbp
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
movl -4(%rbp), %eax
cmpl -8(%rbp), %eax
jl .L4
movl -4(%rbp), %eax
jmp .L5
.L4:
movl -8(%rbp), %eax
.L5:
popq %rbp
ret
```
### Explanation: moving data

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
<th>Src,Dest</th>
<th>C Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>movl</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Imm</strong></td>
<td>Reg</td>
<td>movl $0x4, %eax</td>
<td>tmp = 0x4;</td>
</tr>
<tr>
<td></td>
<td>Mem</td>
<td>movl $-147, (%rax)</td>
<td>*p = -147;</td>
</tr>
<tr>
<td>Reg</td>
<td>Reg</td>
<td>movl %eax, %edx</td>
<td>tmp2 = tmp1;</td>
</tr>
<tr>
<td></td>
<td>Mem</td>
<td>movl %eax, (%rdx)</td>
<td>*p = tmp;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movl (%rax), %edx</td>
<td>tmp = *p;</td>
</tr>
</tbody>
</table>