Computer Architecture

Paul Mellies

Lecture 1 : General introduction to the course
We are all magnificent meaning machines

We produce meaning all day long
How far can we extend our sense of the world?

And what human society will emerge next?
Computer Architecture is one of the greatest adventures of Humanity in the second part of the 20th Century.

It is also a beautiful story which is very important to study in order to understand what will come next.
Quizz ...

Which american company invented the first microprocessor?

A. Control Data Corporation (CDC)
B. International Business Machines (IBM)
C. Hewlett Packard (HP)
D. Intel Corporation (Intel)
E. Digital Equipment Company (DEC)
Quizz ...

What was the name of this first microprocessor by Intel?

A. Intel Pentium
B. Intel 4004
C. Intel Itanium
D. Intel Xeon
E. Intel Core i7
The Intel 4004 microprocessor

- Put on market in November 1971
- Number of transistors: 2300
- Size: 10 µ-meters
- Clock speed: 740 kHz

See: Stanley Mazor, the History of the Microcomputer -- Invention and Evolution, Proceedings of the IEEE, 1995
Looking for the true foundations of our digital age

« Ask someone who works in online gaming or social networking how an integrated circuit works — a component upon which that person’s job, employer, and industry depends for survival — and you are likely to get a blank stare and perhaps something mumbled about “silicon” and “semiconductor.” This isn’t surprising: after all, the worlds of social networks and code writing are six or seven levels removed from the largely chemical business of making computer chips. It would be like asking someone preparing a Big Mac at a McDonald’s in Prague about cattle feed in Tulsa. »

« This is also true in the media, even in the trade press. The reporter who writes incisively on, say, Apple and its next generation iPhone may have little knowledge about the chips inside that device, other than perhaps the name of the manufacturer of its central processor and perhaps its memory chips. That’s why you read very little these days, other than financial news, about the semiconductor industry. The men and women who once covered the business have mostly retired, and the new generation of technology and business reporters are much more comfortable writing about Twitter or Facebook. »

Michael S. Malone
« The Intel Trinity: How Robert Noyce, Gordon Moore, and Andy Grove Built the World's Most Important Company. »
Do you recognize this gentleman?
Do you recognize this gentleman?

Steve Jobs (1955 - 2011)
But do you recognize these two gentlemen?
But do you recognize these two gentlemen?

Dennis M. Ritchie (1941 - 2011)
Ken Thompson (1943)
Inventors of UNIX
and of the C programming language
And this elegant gentleman?
And this elegant gentleman?

Robert Noyce (1927 - 1990)
A panoramic presentation of the course
Your high-level program

Your electronic device
Your high-level program

What this course is all about!

Your electronic device
Your high-level program

compilation into a low-level program typically assembly code or machine code

Computer Architecture

microprocessor, memory and I/O designed to perform the low-level code in an optimized amount of time, reliability and energy

Your electronic device
Your high-level program

intermediate languages designed to take advantage of the underlying architecture

Computer Architecture

tradeoff to find between the various constraints of time, energy and reliability based on the applications one has in mind

Your electronic device
Electrons

Prerequisite to this course:
Introduction to Computer Science
Software
C or Python as a computational model
Hardware
Digital Logic as a computational model

Algorithm
Programming Language
Operating System
Architecture (ISA)
Microarchitecture
Digital Logic
Circuit

Your application in mind

Prerequisite to this course:
Introduction to Computer Science
Prerequisite to this course:
Introduction to Computer Science

Software
C or Python
as a computational model

Prerequisite to this course:
Introduction to Computer Science

Hardware
Digital Logic
as a computational model
If you study this course hard enough, you will get a much clearer and extensive understanding of the digital age we live in.

Desiderius Erasmus Roterodamus
(1467 - 1536)
The digital abstraction

Every light bulb is either **ON** or **OFF**
The digital abstraction

Every light bulb is either **ON** or **OFF**
Can you count in binary numbers?

0 0 0 0 = 0
Can you count in binary numbers?

0 0 0 1

= 1
Can you count in binary numbers?

$0 \ 0 \ 1 \ 0 = 2$
Can you count in binary numbers?

\[0 \ 0 \ 1 \ 1 = 3\]
Can you count in binary numbers?

\[0100 = 4\]
Can you count in binary numbers?

= 5
Can you count in binary numbers?

= 6
Can you count in binary numbers?

= 7
Can you count in binary numbers?

1 0 0 0 = 8
Can you count in binary numbers?

1001 = 9
Can you count in binary numbers?

= 100101

= 10
Can you count in binary numbers?

= 111101
Can you count in binary numbers?

110011 = 12
Can you count in binary numbers?

1
1
0
1

= 13
Can you count in binary numbers?

1 1 1 0 = 14
Can you count in binary numbers?

1 1 1 1 = 15
There are 10 kinds of people

The people who understand binary numbers and the people who don’t...
Key idea: transistors are switches!
NOT gate

A
V_{DD}

GND

Y

Algorithm
Programming Language
Operating System
Architecture (ISA)
Microarchitecture
Digital Logic
Circuit
A \quad \text{GND} \quad B \quad \text{VDD} \quad \text{NAND gate}

**Circuit**

- Algorithm
- Programming Language
- Operating System
- Architecture (ISA)
- Microarchitecture
- Digital Logic
- Circuit
A NOR gate is a digital logic circuit that produces an output high (VDD) only when both inputs are low (GND). The diagram on the right illustrates the NOR gate with inputs A and B, and output Y. The circuit includes transistors and connections to VDD and GND.
Illustration: the inverter

- NMOS transistor
- PMOS transistor

\[ \text{value} = 0 \]  \quad \text{value} = 1
Illustration: the inverter

- Input value = 0
- Output value = 1

ON

OFF

Value = 0

Value = 1
Illustration: the inverter

- Input value = 1
- Output value = 0
- OFF
- On
NAND gate

A

value = 1

B

value = 0

Y

value = 1

value = 0
NAND gate

A=0  B=0

value = 1

OFF

Y=1

ON

value = 0

OFF

OFF

value = 1

OFF

OFF

value = 0
NAND gate

value = 1

A=0 B=1

Y=1

value = 0

OFF

ON

OFF

ON

value = 0

value = 1
NAND gate

value = 1

A=1  B=1

Y=0

ON

ON

OFF

OFF

value = 0

value = 1

value = 0
Circuit
Digital Logic
Microarchitecture
Architecture (ISA)
Operating System
Programming Language
Algorithm

NOT gate

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
NAND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
NOR gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Digital Logic

OR gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Multiplexor (mux)

<table>
<thead>
<tr>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
Multiplexor (mux)

<table>
<thead>
<tr>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
SR-latch (static memory)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>mem</td>
<td>mem</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### SR-latch (static memory)

![SR-latch circuit diagram]

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>mem</td>
<td>(\overline{mem})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
DRAM (dynamic memory)

Select (word line)

Access transistor

Capacitor

GND

D (bit line)
The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.

Amdahl, Blaauw, Brooks
Architecture of the IBM System / 360
IBM Journal of Research and Development
April 1964

- organization of the data flow = microarchitecture
- logical design = digital logic
- physical implementation = circuit
Schematics of a Princeton architecture
also known as « Von Neumann architecture »
Schematics of an Harvard architecture

where the « data memory » is separated from the « code memory »
A traditional distinction: North Bridge vs. South Bridge

High-speed requirements: Memory, Graphics

Other Input/Output functionalities

North Bridge
- Graphics Controller
- PCIe Controller

South Bridge
- Ethernet Controller
- USB Controller
- USB Controller
- USB Controller
- PCIe Controller

DRAM
- OS
- Doc edit prog
- GCC
- Game prog

Hard Disk
- OS
- Research papers
- Family Photos

Monitors
- Graphics
- PCIe x16

Internet
- Mouse

Keyboard
- Printer

Network devices
- PCIe x1
- PCIe x2
- PCIe x4
### Instruction Set Architecture

<table>
<thead>
<tr>
<th>Add</th>
<th>Add immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add unsigned</td>
<td>Add immediate unsigned</td>
</tr>
<tr>
<td>Subtract</td>
<td>Subtract unsigned</td>
</tr>
<tr>
<td>AND</td>
<td>AND immediate</td>
</tr>
<tr>
<td>OR</td>
<td>OR immediate</td>
</tr>
<tr>
<td>NOR</td>
<td>Shift left logical</td>
</tr>
<tr>
<td>Shift right logical</td>
<td>Load upper immediate</td>
</tr>
<tr>
<td>Load word</td>
<td>Load word</td>
</tr>
<tr>
<td>Store word</td>
<td>Load halfword unsigned</td>
</tr>
<tr>
<td>Oracle</td>
<td>Store halfword</td>
</tr>
<tr>
<td>Load byte unsigned</td>
<td>Load byte unsigned</td>
</tr>
<tr>
<td>Store byte</td>
<td>Store byte</td>
</tr>
<tr>
<td>Load linked (atomic update)</td>
<td>Store linked (atomic update)</td>
</tr>
<tr>
<td>Branch on equal</td>
<td>Branch on not equal</td>
</tr>
<tr>
<td>Jump</td>
<td>Jump link</td>
</tr>
<tr>
<td>Jump register</td>
<td>Jump register</td>
</tr>
<tr>
<td>Set less than</td>
<td>Set less than</td>
</tr>
<tr>
<td>Set less than immediate</td>
<td>Set less than immediate</td>
</tr>
<tr>
<td>Set less than unsigned</td>
<td>Set less than unsigned</td>
</tr>
<tr>
<td>Set less than immediate unsigned</td>
<td>Set less than immediate unsigned</td>
</tr>
</tbody>
</table>

**MIPS Core Instructions**

( Patterson & Hennessy : Fig. 3.26 )
A complete description of the Instruction Set Architecture
A complete description of the Instruction Set Architecture

MIPS Green Sheet

OPCODES, BASE CONVERSION, ASCII SYMBOLS

<table>
<thead>
<tr>
<th>OPCODES</th>
<th>BASE CONVERSION</th>
<th>ASCII SYMBOLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdc2</td>
<td>c.le</td>
<td>sdc2</td>
</tr>
<tr>
<td>swc1</td>
<td>c.ngle</td>
<td>swc1</td>
</tr>
<tr>
<td>ldc2</td>
<td>tne</td>
<td>ldc2</td>
</tr>
<tr>
<td>ldc1</td>
<td>c.ult</td>
<td>ldc1</td>
</tr>
<tr>
<td>pref</td>
<td>tltc</td>
<td>pref</td>
</tr>
<tr>
<td>lwc2</td>
<td>tlt</td>
<td>lwc2</td>
</tr>
<tr>
<td>lwc1</td>
<td>tgeu</td>
<td>lwc1</td>
</tr>
</tbody>
</table>

1. Push along perforation to separate card
2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data Card ("Green Card")

IEEE 754 FLOATING-POINT STANDARD

(-1)^S × (1 + Fraction) × 2^(Exponent - Bias)

where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 - 12</td>
<td>23 - 27</td>
</tr>
</tbody>
</table>

MEMORY ALLOCATION

<table>
<thead>
<tr>
<th>Sup</th>
<th>7ff IEEEmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sgp</td>
<td>1000 8000hex</td>
</tr>
<tr>
<td>pc</td>
<td>0040 0000hex</td>
</tr>
<tr>
<td>sp</td>
<td>7fff fffchex</td>
</tr>
</tbody>
</table>

DATA ALIGNMENT

<table>
<thead>
<tr>
<th>Value of three least significant bits of byte address (Big Endian)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 00 00 hex</td>
</tr>
<tr>
<td>0x00 02 00 hex</td>
</tr>
<tr>
<td>0x00 04 00 hex</td>
</tr>
<tr>
<td>0x00 06 00 hex</td>
</tr>
</tbody>
</table>

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Mark</th>
<th>Exception</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EXCEPTION CODES

<table>
<thead>
<tr>
<th>Number Name</th>
<th>Cause of Exception</th>
<th>Number Name</th>
<th>Cause of Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Int (hardware)</td>
<td>9</td>
<td>Hp Breakpoint Exception</td>
</tr>
<tr>
<td>1</td>
<td>AdEx</td>
<td>10</td>
<td>RI</td>
</tr>
<tr>
<td>2</td>
<td>AdEx</td>
<td>11</td>
<td>CpU</td>
</tr>
<tr>
<td>3</td>
<td>AdEx</td>
<td>12</td>
<td>Ov</td>
</tr>
<tr>
<td>4</td>
<td>AdEx</td>
<td>13</td>
<td>Trap</td>
</tr>
<tr>
<td>5</td>
<td>AdEx</td>
<td>14</td>
<td>Syst</td>
</tr>
</tbody>
</table>

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

<table>
<thead>
<tr>
<th>SIZE PREFIX</th>
<th>SIZE FIX</th>
<th>SIZE SIZE FIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^9, 2^9</td>
<td>10^3, 2^3</td>
<td>10^7, 2^7</td>
</tr>
<tr>
<td>10^12, 2^12</td>
<td>10^6, 2^6</td>
<td>10^10, 2^10</td>
</tr>
<tr>
<td>10^15, 2^15</td>
<td>10^9, 2^9</td>
<td>10^13, 2^13</td>
</tr>
<tr>
<td>10^18, 2^18</td>
<td>10^12, 2^12</td>
<td>10^16, 2^16</td>
</tr>
</tbody>
</table>

The symbol for each prefix is put in text table, except μ is used for micro.
A classification of architectures

<table>
<thead>
<tr>
<th>CISC</th>
<th>VLIW</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086 (Intel 1978) and later x86 architectures</td>
<td>Itanium (Intel 2001)</td>
<td>GPUs more generally</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RISC</th>
<th>VLIW</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRISM (DEC 1985) Alpha (DEC 1992)</td>
<td>Itanium (Intel 2001)</td>
<td>GPUs more generally</td>
</tr>
<tr>
<td>ARM (Acorn 1985)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POWER (IBM 1990)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CISC</th>
<th>Reduced Set Computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>Complex Instruction Set Computing</td>
</tr>
<tr>
<td>VLIW</td>
<td>Very Long Instruction Word</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
</tbody>
</table>
The design principles of the MIPS architecture

P1. Simplicity favors regularity:
Every instruction of the MIPS architecture is of length 4 bytes = 32 bits.

P2. Smaller is faster:
Only 32 registers in the MIPS architecture.

P3. Good design demands good compromises:
The instructions are separated into three different formats R, I and J.

The Register (or R) format

<table>
<thead>
<tr>
<th>opcode</th>
<th>first register source operand</th>
<th>second register source operand</th>
<th>register destination operand</th>
<th>shift amount</th>
<th>function code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>
The design principles of the MIPS architecture

P1. Simplicity favors regularity:
Every instruction of the MIPS architecture is of length 4 bytes = 32 bits.

P2. Smaller is faster:
Only 32 registers in the MIPS architecture.

P3. Good design demands good compromises:
The instructions are separated into three different formats R, I and J.

The Immediate (or I) format

<table>
<thead>
<tr>
<th>opcode</th>
<th>first register source operand</th>
<th>second register source operand</th>
<th>address or immediate constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>
The design principles of the MIPS architecture

P1. Simplicity favors regularity:
   Every instruction of the MIPS architecture is of length 4 bytes = 32 bits.

P2. Smaller is faster:
   Only 32 registers in the MIPS architecture.

P3. Good design demands good compromises:
   The instructions are separated into three different formats R, I and J.

The Jump (or J) format

<table>
<thead>
<tr>
<th>opcode</th>
<th>target address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>
The complete LC-3 Instruction Set Architecture

See the detailed description of the ISA by Patt and Patel, 1984.
The two principles of the stored-program computer are the use of instructions that are indistinguishable from numbers and the use of alterable memory for programs. These principles allow a single machine to aid environmental scientists, financial advisers, and novelists in their specialties.

The selection of a set of instructions that the machine can understand demands a delicate balance among the number of instructions needed to execute a program, the number of clock cycles needed by an instruction, and the speed of the clock. As illustrated in this chapter, three design principles guide the authors of instruction sets in making that delicate balance:

1. Simplicity favors regularity. Regularity motivates many features of the MIPS instruction set: keeping all instructions a single size, always requiring three register operands in arithmetic instructions, and keeping the register fields in the same place in each instruction format.

2. Smaller is faster. The desire for speed is the reason that MIPS has 32 registers rather than many more.

3. Good design demands good compromises. One MIPS example was the compromise between providing for larger addresses and constants in instructions and keeping all instructions the same length.

Inflation of the x86 instruction set over time

The price to pay (among other things) for backward compatibility...
All Haswell models designed to support MMX, SSE, SSE2, SSSE3, SSE4.1, SSE4.2, SSE4.2, F16C, BMI1+BMI2, EIST, Intel 64, XD bit, Intel VT-x and Smart Cache

die size ≈ 177 mm²
clock rate ≈ 3 GHz
22 nm FinFET technology

number of transistors per die
≈ 1 400 000 000
If you want to know more about that issue ...

Please have a look at the Intel manual here:

A famous «law» formulated by Gordon Moore in 1965

Moore's paper may be also seen as part of a technological and ideological fight for integrated circuits.

Moore’s paper may be also seen as part of a technological and ideological fight for integrated circuits.
Illustration: MOS Technology 6502 (1974)

- Die size ≈ 16.6 mm²
- Clock rate ≈ 1 MHz to 2 MHz
- 8 µm NMOS technology
- Number of transistors per die:
  - ≈ 3510 in enhancement mode
  - + 1018 in depletion mode
Digital Archeology: excavating the 6502

The low cost 6502 processor was designed in 1974 and launched the “Home Computers” era. The visual 6502 project started in 1999 and aims at “reverse engineering” the processor.

Read the article: Nikhil Swaminathan. Digging into Technology’s Past. Volume 64 Number 4, July/August 2011 http://archive.archaeology.org/1107/features/mos_technology_6502_computer_chip_cpu.html

Visit the site: http://www.visual6502.org/JSSim

Make the JSSim simulator work and ...

A. count the number of fetches in the first 100 clock cycles
B. guess the function of the Sync pin

Bonus: find the position of the instruction decode table and pinpoint the ROL and ROR commands in it

FOR NEXT RECITATION

A photography of the MOS Technology 6502 design team with a design schematic. Bill Mensch who drew the processor on a drafting board is standing second from the right. 1974.
Have you ever imagined constructing a city layer by layer?

Picture taken from the project Urban Layers
http://io.morphocode.com/urban-layers
Thus very likely to contain bugs and mistakes ...

1974 : MOS 6502
The ROR command of the first 6502 processor did not work. So, the command was simply removed from the manual!

1994  P5 Pentium
A few missing entries in the lookup table for its divide operation algorithm led to an error in the floating point unit, producing inaccurate results in approximately 1 in 9 billion floating point divides/

2011 : Intel Sandy Bridge
The metal line of the SATA port of the Cougar chipset was too small for its purpose and would gradually thin and degrade over time.

2014 : Intel Haswell
The Transactional Synchronization Extension (TSX) or « transactional memory » was bugged in the Haswell processor. The TSX feature was thus disabled via a microcode update.

Hot research topic today:
design formal methods to prove that a given microarchitecture is correct
The Pentium Pro processor and Pentium II processor may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Many times, the effects of the errata can be avoided by implementing hardware or software work-arounds, which are documented in the Pentium Pro Processor Specification Update and Pentium II Processor Specification Update, both of which can be found at www.intel.com. Pentium Pro and Pentium II processors include a feature called "reprogrammable microcode", which allows certain types of errata to be worked around via microcode updates. The microcode updates reside in the system BIOS and are loaded into the processor by the system BIOS during Power-On Self Test, or POST.
The recognition that a program could be written to translate a more powerful language into computer instructions was one of the great breakthroughs in the early days of computing. Programmers today owe their productivity—and their sanity—to the creation of high-level programming languages and compilers that translate programs in such languages into instructions. Figure 1.4 shows the relationships among these programs and languages, which are more examples of the power of abstraction.

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
 temp = v[k];
 v[k] = v[k+1];
 v[k+1] = temp;
}
```

Assembly language program (for MIPS)

```
swap:
multi $2, $5,4
add $2, $4,$2
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $16, 4($2)
jr $31
```

Binary machine language program (for MIPS)

```
0000000010100001000000000100011000
0000000101000010000000000100011000
1000110111100000000000000000000000
1000110000010010000000000000000100
101011100001001000000000000001000
101011011110001000000000000001000
000000111110000000000000000001000
```
## A hierarchy of programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>Designed to facilitate heavy use of math functions</td>
</tr>
<tr>
<td>Perl</td>
<td>Designed for scripting</td>
</tr>
<tr>
<td>Python</td>
<td>Designed to emphasize code readability</td>
</tr>
<tr>
<td>Java</td>
<td>Designed to run securely on any computer</td>
</tr>
<tr>
<td>C</td>
<td>Designed for flexibility and overall system access, including device drivers</td>
</tr>
<tr>
<td>Assembly Language</td>
<td>Human-readable machine language</td>
</tr>
<tr>
<td>Machine Language</td>
<td>Binary representation of a program</td>
</tr>
</tbody>
</table>

Note the specific position of the programming language C at the software / hardware interface.
Compilation from C to MIPS

Original C code:

c = (a+b) - (i+j)

Compiled MIPS code:

add t0, a, b    # temporary register t0 = a+b
add t1, i, j    # temporary register t1 = i+j
sub c, t0, t1   # c = t0 - t1
# Registers in MIPS

There are exactly 32 registers in the MIPS architecture. Each register is able to store exactly one « word ». Here, by « word » one means a sequence of 32 bits. Each MIPS register thus contains exactly 32 bits.

By convention, each register has its own status or function:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register number</th>
<th>Usage</th>
<th>Preserved on call?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>0</td>
<td>The constant value 0</td>
<td>n.a.</td>
</tr>
<tr>
<td>$v0–$v1</td>
<td>2–3</td>
<td>Values for results and expression evaluation</td>
<td>no</td>
</tr>
<tr>
<td>$a0–$a3</td>
<td>4–7</td>
<td>Arguments</td>
<td>no</td>
</tr>
<tr>
<td>$t0–$t7</td>
<td>8–15</td>
<td>Temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$s0–$s7</td>
<td>16–23</td>
<td>Saved</td>
<td>yes</td>
</tr>
<tr>
<td>$t8–$t9</td>
<td>24–25</td>
<td>More temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>Global pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>Stack pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>Frame pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>Return address</td>
<td>yes</td>
</tr>
</tbody>
</table>

For more information, see Figure 2.14 in Patterson & Hennessy.
Compilation from C to MIPS

Original C code:

c = (a+b) - (i+j)

Compiled MIPS code:

add $t0, $s1, $s2  # temp register $t0 = $s1+$s2
add $t1, $s3, $s4  # temp register $t1 = $s2+$s3
sub $s0, $t0, $t1  # $s0 = $t0 - $t1

Here, we make the important assumption that
• the registers $s1 and $s2 contain the values of a and b
• the registers $s2 and $s3 contain the values of i and j
Another example

Original C code:

```c
swap (int v[], int k)
{int temp;
   temp = v[k];
   v = v[k+1];
   v[k+1] = temp;
}
```

Compiled MIPS code:

```mips
multi $2, $5,4  # multiplies by 4 the value of register 5
   # and writes the result in register 2
add $2, $4,$2  # adds the value of register 4 to the value of register 2
   # and writes the result in register 2
lw $15, 0($2)  # loads the value of Mem[$2] in register 15
lw $16, 4($2)  # loads the value of Mem[$2+4] in register 16
sw $16, 0($2)  # stores the value of register 16 in Mem[$2]
sw $15, 4($2)  # stores the value of register 15 in Mem[$2+4]
jr $31  # jumps to the address loaded in register 31
```

Here, we suppose that:
- register 4 contains the address in memory of the array v[] of integers
- register 5 contains the value of the integer k.
A troublesome fact

Almost every C compiler is bugged ...

To convince yourself, please have a look at:

Xuejun Yang, Yang Chen, Eric Eide, John Regehr
Finding and Understanding Bugs in C Compilers
2011 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)
Compilers and operating systems you can trust

Certified Compilers

CompCert: a formally checked C compiler

Certified Operating Systems

seL4 project: a formally checked OS microkernel

For the most curious among you:  

Xavier Leroy  
Proof assistants in computer science research  
https://www.youtube.com/watch?v=PE4v6rVpX2g
Introducing myself ...

Paul Mellies
pam13@nyu.edu

I am a CNRS researcher in Paris working on the interface between logic and programming languages with an interest in the translation (or compilation) of high level programs into low level programs like assembly code or machine code.

One main purpose of my research is to develop the mathematical framework to provide a rigorous meaning (or semantics) to low level code, typically written in machine code or implemented in hardware.
HW 0: introduce yourself!
Due before February 19th (Tuesday) midnight.

In the next five days:
Install Unix on your machine!
Learn your first shell commands using the references which you will find on the webpage of the course:

http://computer-architecture.org

Read the paper by Moore.
Read the paper by Patterson and Ditzel.

HW 1: explore your computer!
Due before February 22nd (Friday) midnight.
Thank you!

Any question?