A Lesson on Instruction Set Architectures (ISAs)

Application software

Systems software (OS, compiler)

ISA

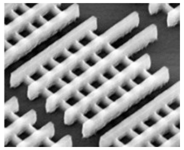
Digital design and architecture

Circuits and devices

Andrew DeOrio

awdeorio@umich.edu

andrewdeorio.com



Today's lesson

- Based on material from my sophomore/junior level Introduction to Computer Organization course (EECS 370 at Michigan)

- Slides online at andrewdeorio.com

- In previous lectures, we covered:
  - Basic programming in C
  - Representing numbers in binary and hexadecimal
  - Computer history

Review: programming in C

- Program

```
/* hello.c */
#include <stdio.h>

int main() {
  printf("hello world!\n");
  return 0;
}
```

- Compile

```
$ gcc hello.c
```

- Run

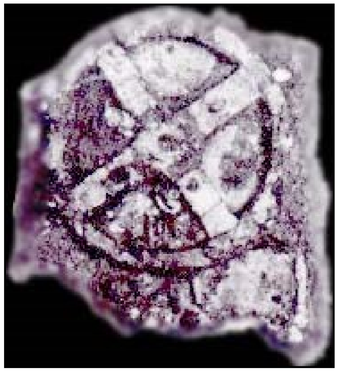
```
$ ./a.out
hello world!
```

Review: numbers in binary

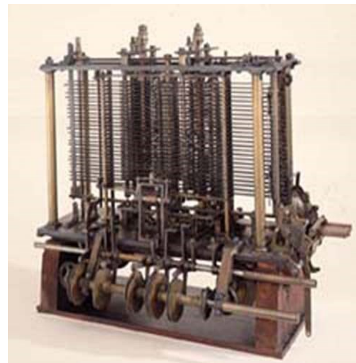
- 42 in base 10 (AKA decimal)
  $= 40 + 2$
  $= 4*10^1 + 2*10^0$
  $= 42_{10}$
- 42 in base 2 (AKA binary)
  $= 32 \quad + \quad 0 \quad + \quad 8 \quad + \quad 0 \quad + \quad 2 \quad + \quad 0$
  $= 1*2^5 + 0*2^4 + 1*2^3 + 0 *2^2 + 1*2^1 + 0*2^0$
  $= 101010_2$
- 42 in base 16 (AKA hexadecimal)
  $= 32 \quad + \quad 10$
  $= 2*16^1 + 10*16^0$
  $= 2A_{16}$
  or
  $= 0010\ 1010_2$
  $= \quad 2 \qquad A$
  $= 0x2A$

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

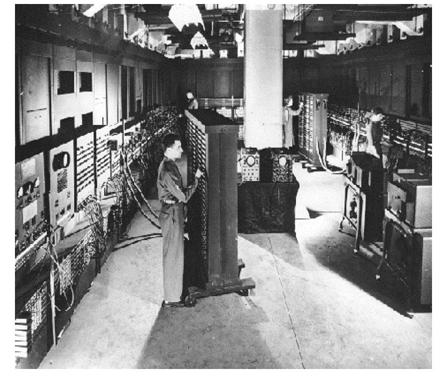
Review: computer history



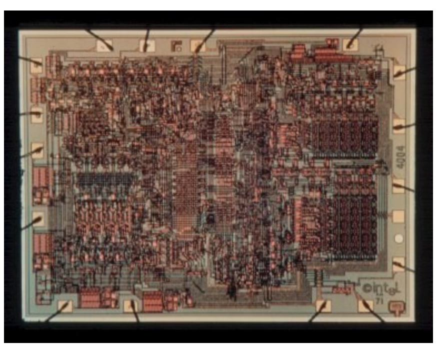
Astrolavos computed phases of the moon



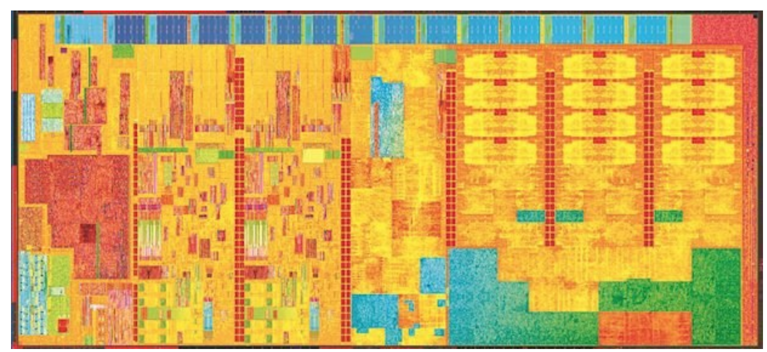
Babbage's analytical engine. Mechanical general purpose computer.



ENIAC electronic computer, using vacuum tubes

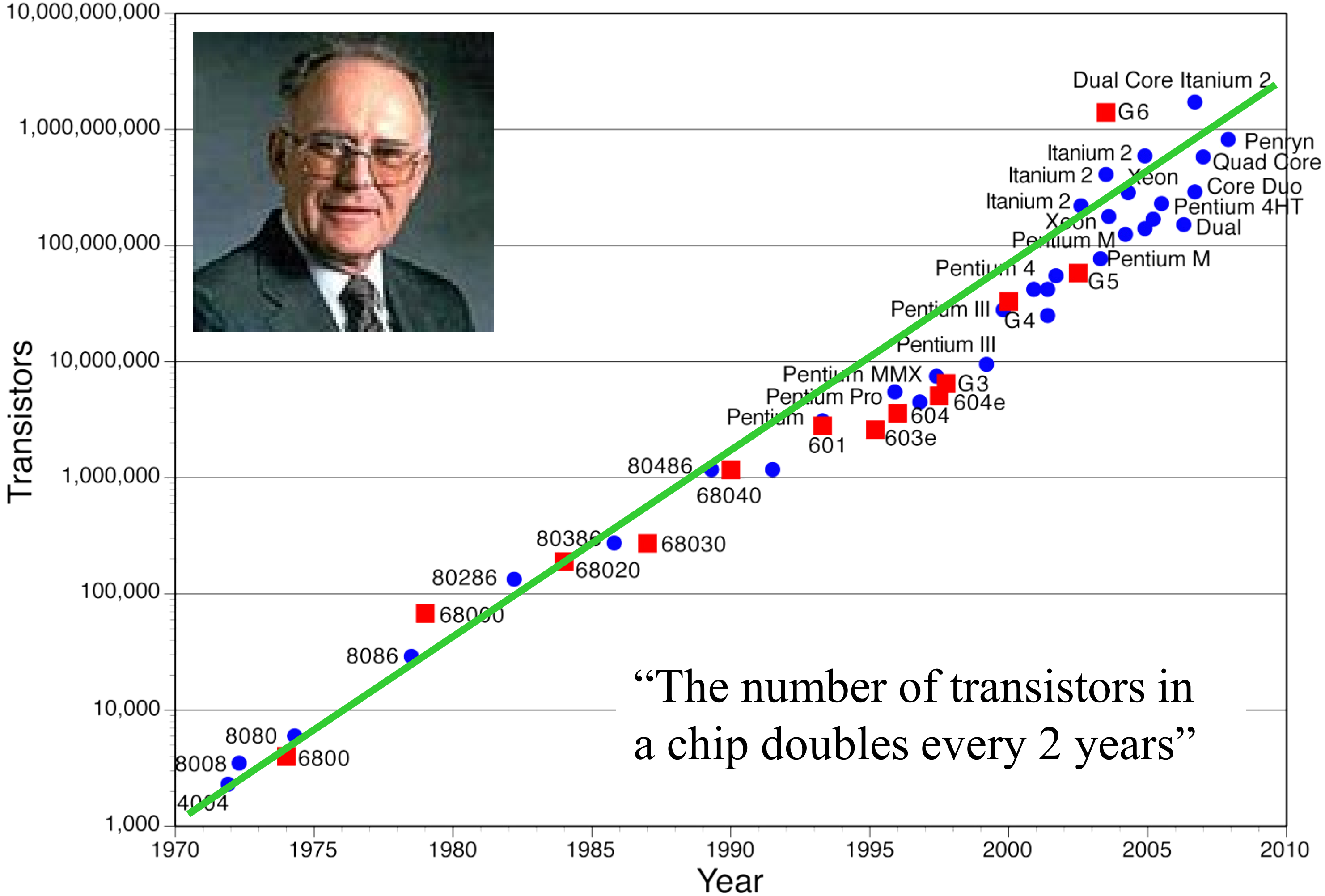


1970: Intel 4004, first microprocessor, 2,250 transistors



2014: Intel Core i7 (Broadwell), 1.4B transistors

Moore’s law


Transistors
Year
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010
Dual Core Itanium 2
G6
Itanium 2
Itanium 2
Itanium 2
Quad Core
Core Duo
Pentium 4HT
Dual
Pentium M
G5
G4
Pentium III
G3
604e
604
603e
601
80486
68040
68030
68020
80286
8086
8080
8008
6800
“The number of transistors in
a chip doubles every 2 years”

iPhone 6 teardown

A8: 2B
transistors

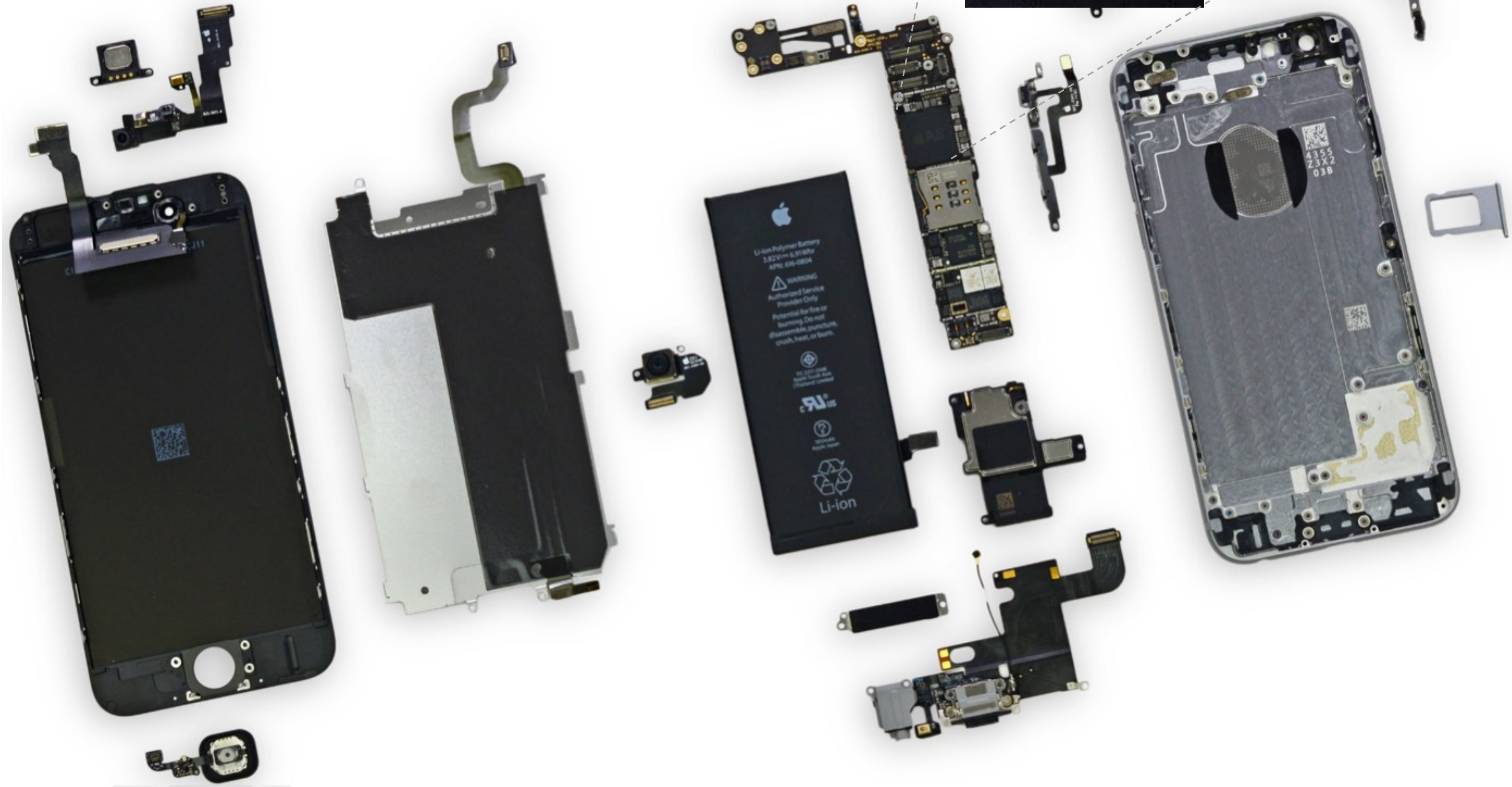


Photo credit: ifixit.com

Abstraction

A8: 2B transistors

- How do you put together 2B transistors?
- Use abstraction to simplify design
- Separate lower-level details from higher-level details
- The abstraction we'll focus on today is called the *Instruction Set Architecture*, or ISA

- Put another way, we need a language that lets SW talk to HW

Abstraction

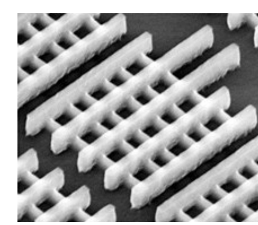
Application software

Systems software (OS, compiler)

Instruction Set Architecture (ISA)

Digital design and architecture

Circuits and devices



Abstraction

- Alice is a hardware engineer, Bob is a software engineer
- Alice and Bob agree on an ISA

- Alice creates a chip
  - Implements the ISA

- Bob writes software
  - Uses the ISA

- They put it together and the iPhone 6 works

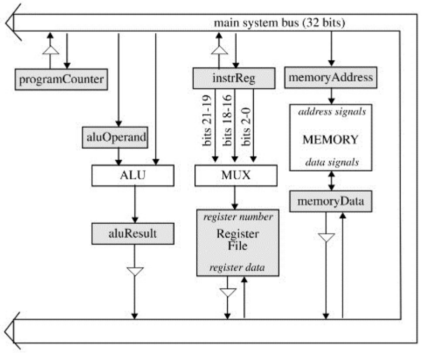
Cartoon credit: xkcd.com

ISA examples

- Core i7 implements the x86 ISA

- Apple A8 implements ARM ISA

- The Little Computer implements the LC2K ISA
  - Example ISA for this class



RISC and CISC

- How to encode instructions?
- CISC: Complex Instruction Set Computer
  - Instructions can vary in size
  - Example: x86 (most laptops)

- RISC: Reduced Instruction Set Computer
  - All instructions are same length
  - Example: ARM (many cell phones)

Instructions

- The language that hardware and software use to talk to each other is the ISA
- The vocabulary consists of *instructions*
- Examples of *instructions*
  - Add two numbers
    `add  R2  R3  R1`
  - Check to see if two numbers are equal
    `beq  R4  R5  5`
  - Copy a piece of data
    `lw   R6  R7   5`

Software program to instructions

A program that converts input C-programs to output assembly programs
Example: `gcc -S`

A program that converts input assembly programs to output binary programs
Example: `as` or (part of) `gcc`

```
int main() {
 int a=0, b=0;
 int c = a + b;
}
```

C program

Compile

```
lw    R2   R0   R0
lw    R3   R0   R0
add   R2   R3   R1
```

Assembly code

Assemble

```
00011100011000
00001111000111
00010100101100
```

Machine code

Storing instructions

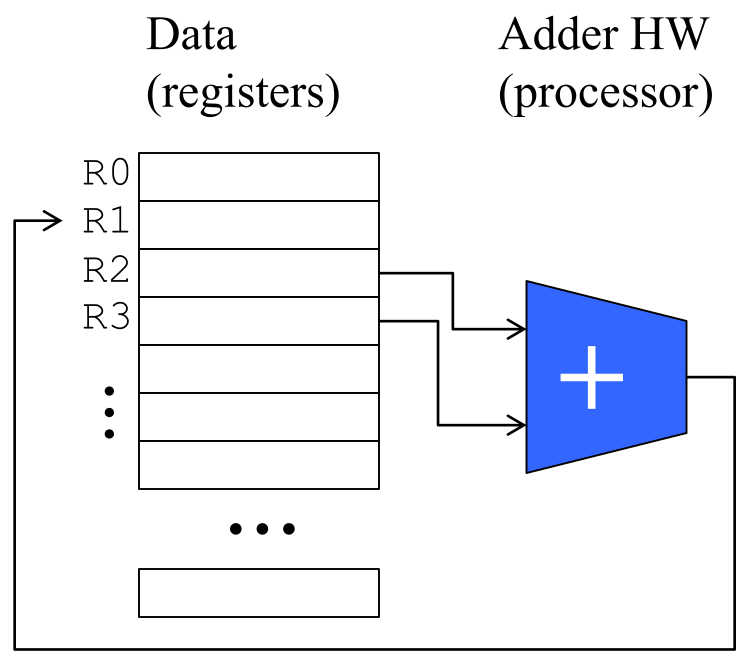
- First, break up the instruction into fields
- Opcode – what instruction to perform
- Source (input) operands
- Destination (output) operand

opcode	src1	src2	dest
add	**R2**	**R3**	**R1**

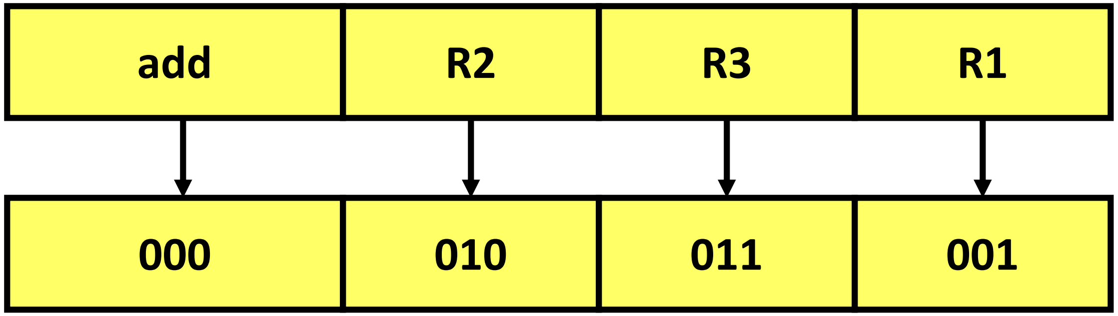
Translation: value in R1 = contents of R2 + R3

Storing operands

- Operands are stored in registers
  `add R2 R3 R1`
- A register is a physical piece of hardware that stores one number
- `add` is performed by physical hardware on chip
- Registers are small and fast
- In future lessons, we'll learn how main memory extends this storage



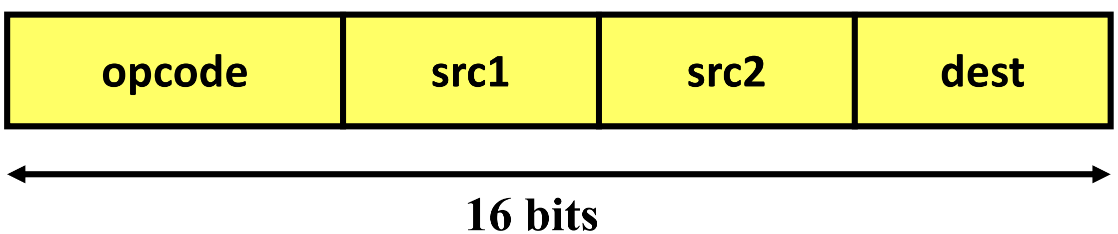
Assembly instruction encoding



- m bits can encode 2^m different values
- n values can be encoded in $\lceil \log_2(n) \rceil$ bits

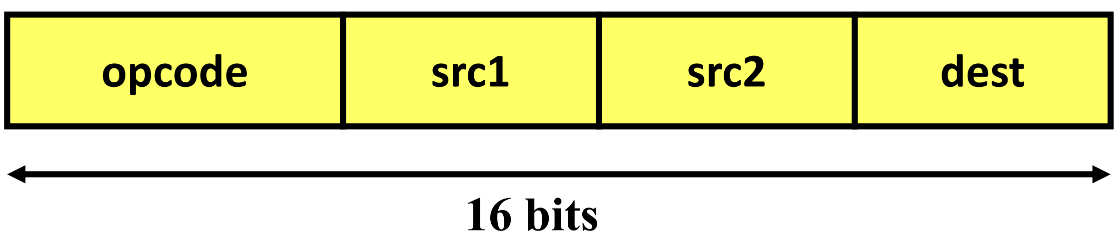
Exercise

- What is the maximum number of registers that can be designed in an ISA with:
  - 16-bit instructions
  - 100 opcodes
  - All instructions have 2 source operands and 1 destination operand



Solution

- What is the maximum number of registers that can be designed in an ISA with:
  - 16-bit instructions
  - 100 opcodes
  - All instructions have 2 source operands and 1 destination operand



1. num opcode bits = $\lceil \log_2(100) \rceil = 7$
2. num bits for operands = $16 - 7 = 9$
3. num bits per operand = $9 / 3 = 3$
4. maximum number of registers = $2^3 = 8$

News break

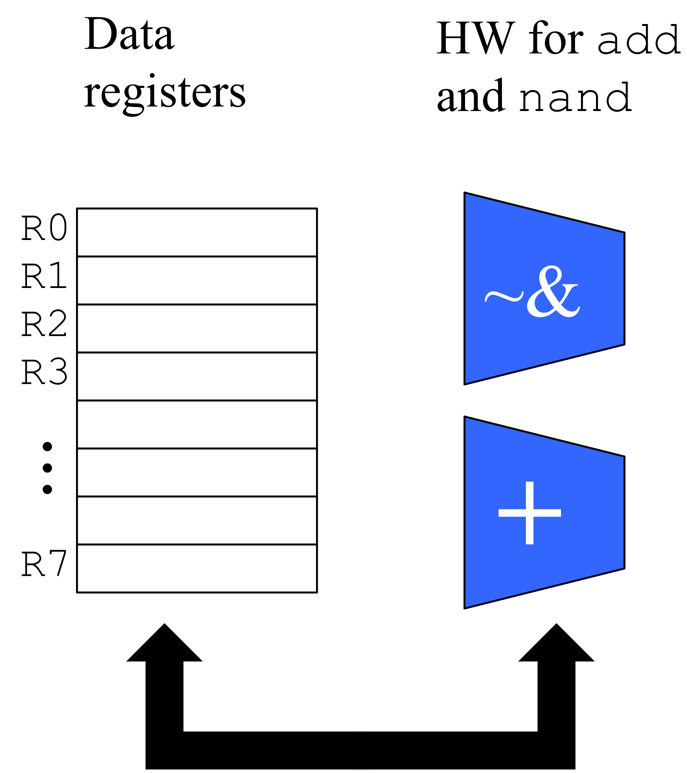
- Let’s take a break

- What’s going on in the computer-related news?

LC2K ISA

- Little Computer 2000 (LC2K) is a small example computer and Instruction Set Architecture
- 32-bit processor
  - Instructions are 32 bits
  - Registers are 32 bits
- 8 registers
- Supports 65,536 ($=2^{16}$) storage locations (memory addresses)
- 8 instructions
  - `add, nand, lw, sw, beq, jalr, halt, noop`

A portion of the LC2K hardware



LC2K assembly instructions

Instruction	Example	Meaning in code	Meaning in English
`add`	`add 1 2 3`	`R3 = R1 + R2`	Add two numbers
`nand`	`nand 1 2 3`	`R3 = ~(R1 & R2)`	Negative *and*
`lw`	Instructions for copying data. We'll learn about these next time.		
`sw`			
`beq`	Control flow instructions used by loops, `if`-statements and functions. Also a future lesson.		
`jalr`			
`halt`	`halt`		Stop the program
`noop`	`noop`		Do nothing

Storing operands

- Example program
- Assume all registers are initialized to zero
- Remember: last register is the destination

```
nand 1 1 2
add  1 2 3
```

R0	0
R1	0
R2	0
R3	0
	0
⋮	0
	0
R7	0

Storing operands solution

- Example program
- Assume all registers are initialized to zero
- Remember: last register is the destination

```
nand 1 1 2
add  1 2 3
```

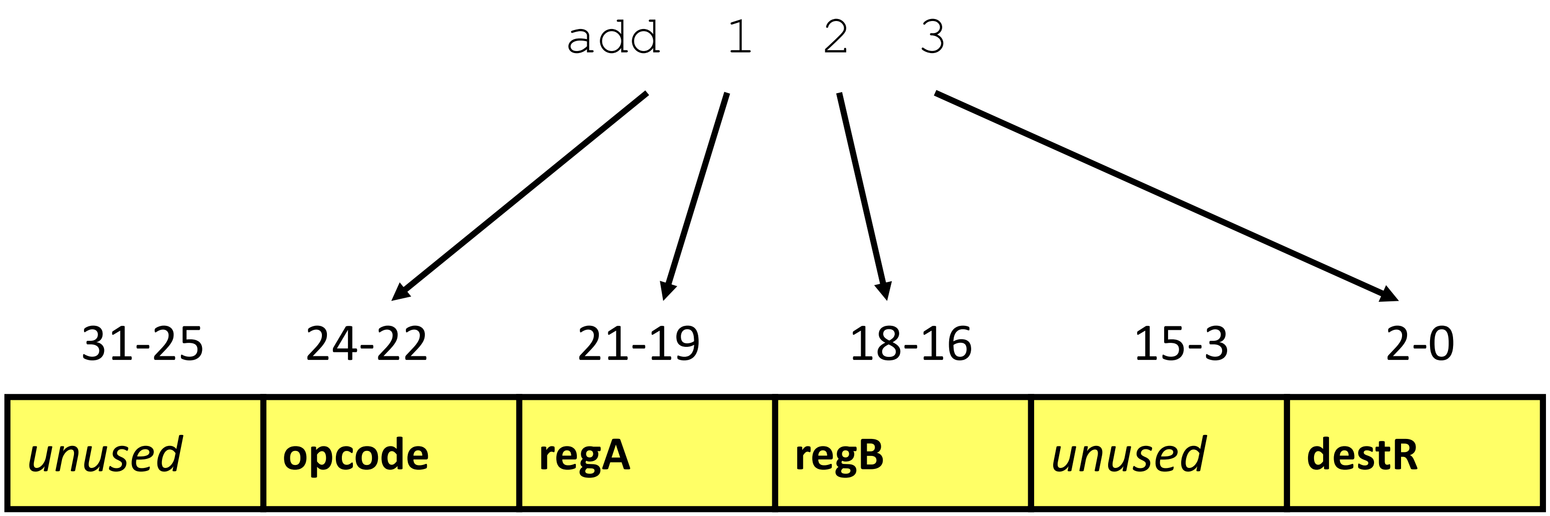
R0	0
R1	0
R2	0xFFFF FFFF
R3	0xFFFF FFFF
	0
⋮	0
	0
R7	0

Assembly instruction encoding

- Instructions are stored in the same way as data
- Instructions are encoded as numbers
- How to encode it? Use the Architecture Reference Manual
- Excerpt from the LC2K manual
- `add srcA srcB dest`
  - bits 24-22: opcode 000
  - bits 21-19: src A operand
  - bits 18-16: src B operand
  - bits 15-3: unused (0's)
  - bits 2-0:   dest operand

Instruction encoding

- Instructions are stored in the same way as data
- Instructions are encoded as numbers
- How to encode it? Use the Architecture Reference Manual



Instruction formats

- Positional organization of bits in LC2K
  - Implies nothing about bit values!!!
- Both `add` and `nand` are encoded this way

31-25	24-22	21-19	18-16	15-3	2-0
unused	**opcode**	**regA**	**regB**	*unused*	**destR**

- Opcode encodings
  - `add` (000), `nand` (001)
- Register values
  - Just encode the register number (R2 = 010)

Encoding exercise

- `nand  3  4  7     //R7 = ~(R3 & R4)`
- `nand` opcode is 001

31-25	24-22	21-19	18-16	15-3	2-0
unused	**opcode**	**regA**	**regB**	*unused*	**destR**

Encoding exercise solution

- `nand  3  4  7      //R7 = ~(R3 & R4)`

31-25	24-22	21-19	18-16	15-3	2-0
unused	**opcode**	**regA**	**regB**	*unused*	**destR**
0000000	001	011	100	0000000000000	111

```
bin: 0000000 001 011 100 0000000000000 111
bin: 0000 0000 0101 1100 0000 0000 0000 0111
hex     0    0    5    C    0    0    0    7
hex: 0x005C0007
```

Recall our abstraction

- Hardware engineer Alice and software engineer Bob
- Alice and Bob agree on an ISA

- Alice creates a chip
  - Implements the ISA

- Bob writes software
  - Uses the ISA

- They put it together and the iPhone 6 works

Cartoon credit: xkcd.com

Abstraction exercise

- For each of these scenarios, does the software need to change? Does the hardware need to change?
- Change `nand` instruction to `and`
- Add a new `mul` (multiply) instruction
- New piece of hardware that can `add` faster
- New operating system that is better at multitasking

Abstraction exercise solution

- For each of these scenarios, does the software need to change? Does the hardware need to change?
- Change `nand` instruction to `and`
  - Both software and hardware
- Add a new `mul` (multiply) instruction
  - Hardware changes. Software optional.
- New piece of hardware that can `add` faster
  - Hardware only
- New operating system that is better at multitasking
  - Software only

Cartoon credit: xkcd.com

The power of ISAs

- We can upgrade the hardware and the old software still works!
  - Example: iPhone 5 to iPhone 5s you can still run your old applications.

- We can upgrade the software and the old hardware still works!
  - Example: iOS7 to iOS8 upgrade on iPhone 5/5s. You can still use your old hardware.

Next time

- Storage and memory
  - How do the load and store instructions work?
- Control instructions
  - How does the `beq` instruction work?
  - How do we implement loops and `if`-statements?

Programming assignment #1

- Coming soon
- Write an assembler to convert input (assembly language program) to output (machine code version of program)

- Write a behavioral simulator to run the machine code version of the program (printing the contents of the registers and memory after each instruction executes

- Write an efficient assembly language program to multiply two numbers