Welcome to 6.004!

Computation Structures

Fall 2019
6.004 Course Staff

Instructors

Daniel Sanchez
sanchez@csail.mit.edu

Silvina Hanono Wachman
silvina@mit.edu

Jason Miller
jasonm@csail.mit.edu
6.004 Course Staff

Instructors

Daniel Sanchez
sanchez@csail.mit.edu

Silvina Hanono Wachman
silvina@mit.edu

Jason Miller
jasonm@csail.mit.edu

Teaching Assistants

Kathy Camenzind
Brian Chen
Quan Nguyen
Ileana Rugina
Kendall Garner

Felipe Moreno
Domenic Nutile
Sebastian Bartlett
Computing Devices Then...

ENIAC, 1943  30 tons, 200KW, ~1000 ops/sec
Computing Devices Now

Typical 2019 laptop
1kg, 10W, 10 billion ops/s
Computing Devices Now

Typical 2019 laptop
1kg, 10W, 10 billion ops/s
An Introduction to the Digital World

Computer programs

Devices
Materials
Atoms
An Introduction to the Digital World

Computer programs

Digital design
Combinational and sequential circuits

Devices
Materials
Atoms
An Introduction to the Digital World

Computer programs

Computer architecture
Processors, caches, pipelining

Digital design
Combinational and sequential circuits

Devices
Materials
Atoms
An Introduction to the Digital World

Computer programs

Computer systems
Operating systems, virtual memory, I/O

Computer architecture
Processors, caches, pipelining

Digital design
Combinational and sequential circuits

Devices
Materials
Atoms
The Power of Engineering Abstractions

Good abstractions let us reason about behavior while shielding us from the details of the implementation.

- Virtual machines
- Instruction set + memory
- Digital circuits
- Bits, Logic gates
The Power of Engineering Abstractions

Good abstractions let us reason about behavior while shielding us from the details of the implementation.

**Corollary:** implementation technologies can evolve while preserving the engineering investment at higher levels.

- Virtual machines
- Instruction set + memory
- Digital circuits
- Bits, Logic gates
The Power of Engineering Abstractions

Good abstractions let us reason about behavior while shielding us from the details of the implementation.

*Corollary*: implementation technologies can evolve while preserving the engineering investment at higher levels.

Leads to hierarchical design:
- Limited complexity at each level ⇒ shorten design time, easier to verify
- Reusable building blocks

Virtual machines

Instruction set + memory

Digital circuits

Bits, Logic gates
Course Outline

- Module 1: Assembly language
  - From high-level programming languages to the language of the computer
Course Outline

- Module 1: Assembly language
  - From high-level programming languages to the language of the computer

- Module 2: Digital design
  - Combinational and sequential circuits
Course Outline

- Module 1: Assembly language
  - From high-level programming languages to the language of the computer

- Module 2: Digital design
  - Combinational and sequential circuits

- Module 3: Computer architecture
  - Simple and pipelined processors
  - Caches and the memory hierarchy
Course Outline

- **Module 1: Assembly language**
  - From high-level programming languages to the language of the computer

- **Module 2: Digital design**
  - Combinational and sequential circuits

- **Module 3: Computer architecture**
  - Simple and pipelined processors
  - Caches and the memory hierarchy

- **Module 4: Computer systems**
  - Operating system and virtual memory
  - Parallelism and synchronization
Our Focus: Programmable General-Purpose Processors
Our Focus: Programmable General-Purpose Processors

- Microprocessors are the basic building block of computer systems
  - Understanding them is crucial even if you do not plan to work as a hardware designer
Our Focus: Programmable General-Purpose Processors

- Microprocessors are the basic building block of computer systems
  - Understanding them is crucial even if you do not plan to work as a hardware designer
- Microprocessors are the most sophisticated digital systems that exist today
  - Understanding them will help you design all kinds of hardware
Our Focus: Programmable General-Purpose Processors

- Microprocessors are the basic building block of computer systems
  - Understanding them is crucial even if you do not plan to work as a hardware designer
- Microprocessors are the most sophisticated digital systems that exist today
  - Understanding them will help you design all kinds of hardware

By the end of the term you will have designed a simple processor from scratch!
We Rely on Modern Design Tools

- We will use RISC-V, a simple and modern instruction set
We Rely on Modern Design Tools

- We will use RISC-V, a simple and modern instruction set

- We will design hardware using Minispec, a new hardware description language built for 6.004
  - Based on Bluespec, but heavily simplified

![Diagram showing the workflow from Minispec source to compiled circuit (gates)]
Course Mechanics

- 2 lectures/week: handouts, videos, and reference materials on website
- 2 recitations/week: work through tutorial problems using skills and concepts from previous lecture
Course Mechanics

- 2 lectures/week: handouts, videos, and reference materials on website
- 2 recitations/week: work through tutorial problems using skills and concepts from previous lecture
- 7 mandatory lab exercises
  - Online submission + check-off meetings in lab
  - Due throughout the term (7 free late days meant to give you flexibility, cover short illnesses, etc.; see website)
Course Mechanics

- 2 lectures/week: handouts, videos, and reference materials on website
- 2 recitations/week: work through tutorial problems using skills and concepts from previous lecture
- 7 mandatory lab exercises
  - Online submission + check-off meetings in lab
  - Due throughout the term (7 free late days meant to give you flexibility, cover short illnesses, etc.; see website)
- One open-ended design project
  - Due at the end of the term
Course Mechanics

- 2 lectures/week: handouts, videos, and reference materials on website
- 2 recitations/week: work through tutorial problems using skills and concepts from previous lecture
- 7 mandatory lab exercises
  - Online submission + check-off meetings in lab
  - Due throughout the term (7 free late days meant to give you flexibility, cover short illnesses, etc.; see website)
- One open-ended design project
  - Due at the end of the term
- 3 quizzes: Oct 17, Nov 14, Dec 5 (7:30-9:30pm)
  - If you have a conflict, contact us to schedule a makeup
Recitation Mechanics

- 12 recitation sections on Wed & Fri
  - If you have a conflict with your assigned section, choose a different one—no need to let us know
Recitation Mechanics

- 12 recitation sections on Wed & Fri
  - If you have a conflict with your assigned section, choose a different one—no need to let us know

- Recitation attendance is mandatory (worth 5% of your grade)
  - Everyone has 4 excused absences
Recitation Mechanics

- 12 recitation sections on Wed & Fri
  - If you have a conflict with your assigned section, choose a different one—no need to let us know

- Recitation attendance is mandatory (worth 5% of your grade)
  - Everyone has 4 excused absences

- Recitations will review lecture material and problems associated with each lecture
  - We recommend you work on the problems before recitation
  - We will post solutions after recitation
Grading

- 80 points from labs,
- 20 points from design project,
- 90 points from quizzes,
- 10 points from recitation attendance
Grading

- 80 points from labs,
  20 points from design project,
  90 points from quizzes,
  10 points from recitation attendance

- Fixed grade cutoffs:
  - A: Points $\geq 165$
  - B: Points $\geq 145$
  - C: Points $\geq 125$
  - F: Points $< 125$ or not all labs complete
Online and Offline Resources

- The course website has up-to-date information, handouts, and references to supplemental reading: http://6004.mit.edu
Online and Offline Resources

- The course website has up-to-date information, handouts, and references to supplemental reading: http://6004.mit.edu

- We use Piazza extensively
  - Fastest way to get your questions answered
  - All course announcements are made on Piazza
Online and Offline Resources

- The course website has up-to-date information, handouts, and references to supplemental reading: http://6004.mit.edu

- We use Piazza extensively
  - Fastest way to get your questions answered
  - All course announcements are made on Piazza

- We will hold regular office hours in the lab (room 32-083) to help you with lab assignments, infrastructure, and any other questions
Online and Offline Resources

- The course website has up-to-date information, handouts, and references to supplemental reading: http://6004.mit.edu

- We use Piazza extensively
  - Fastest way to get your questions answered
  - All course announcements are made on Piazza

- We will hold regular office hours in the lab (room 32-083) to help you with lab assignments, infrastructure, and any other questions

32-083 Combination Lock:__________________________
We Want Your Feedback!

- Your input is crucial to fine-tune this offering of the course and improve future versions

- Periodic informal surveys

- Any time: Email us or post on Piazza
Binary Number Encoding and Arithmetic
Digital systems represent and process information using **discrete symbols** or digits.
Digital Information Encoding

- Digital systems represent and process information using **discrete symbols** or digits.

- These are typically **binary digits** (bits): 0 and 1.
Digital Information Encoding

- Digital systems represent and process information using **discrete symbols** or digits

- These are typically binary digits (bits): 0 and 1

- We can implement operations like +, >, AND, etc. on binary numbers in hardware very efficiently
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

\[
\begin{array}{cccccccccccc}
2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
\]
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

What value does 011111010000 encode?
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

<table>
<thead>
<tr>
<th></th>
<th>2^11</th>
<th>2^10</th>
<th>2^9</th>
<th>2^8</th>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What value does 011111010000 encode?

\[ V = 0 \cdot 2^{11} + 1 \cdot 2^{10} + 1 \cdot 2^9 + \ldots \]

\[ = 1024 + 512 + 256 + 128 + 64 + 16 = 2000 \]
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What value does 011111010000 encode?

\[ V = 0 \times 2^{11} + 1 \times 2^{10} + 1 \times 2^9 + \ldots \]

\[ = 1024 + 512 + 256 + 128 + 64 + 16 = 2000 \]

Smallest number?  Largest number?
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What value does 011111010000 encode?

\[ V = 0 \times 2^{11} + 1 \times 2^{10} + 1 \times 2^9 + \ldots \]
\[ = 1024 + 512 + 256 + 128 + 64 + 16 = 2000 \]

Smallest number? 0    Largest number?
Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an N-bit number is given by the following formula:

\[ v = \sum_{i=0}^{N-1} 2^i b_i \]

<table>
<thead>
<tr>
<th>2^11</th>
<th>2^10</th>
<th>2^9</th>
<th>2^8</th>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What value does 011111010000 encode?

\[ V = 0 \times 2^{11} + 1 \times 2^{10} + 1 \times 2^9 + \ldots \]

\[ = 1024 + 512 + 256 + 128 + 64 + 16 = 2000 \]

Smallest number? 0
Largest number? 2^(N-1)
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

<table>
<thead>
<tr>
<th>$2^{11}$</th>
<th>$2^{10}$</th>
<th>$2^{9}$</th>
<th>$2^{8}$</th>
<th>$2^{7}$</th>
<th>$2^{6}$</th>
<th>$2^{5}$</th>
<th>$2^{4}$</th>
<th>$2^{3}$</th>
<th>$2^{2}$</th>
<th>$2^{1}$</th>
<th>$2^{0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

<table>
<thead>
<tr>
<th>Hexadecimal - base 16</th>
<th>2^11</th>
<th>2^10</th>
<th>2^9</th>
<th>2^8</th>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 0</td>
<td>1000 - 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001 - 1</td>
<td>1001 - 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010 - 2</td>
<td>1010 - A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011 - 3</td>
<td>1011 - B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100 - 4</td>
<td>1100 - C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101 - 5</td>
<td>1101 - D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110 - 6</td>
<td>1110 - E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111 - 7</td>
<td>1111 - F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

September 5, 2019
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

Hexadecimal - base 16

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
</tr>
</tbody>
</table>

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

Hexadecimal - base 16

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
</tr>
</tbody>
</table>
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

Hexadecimal - base 16

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

<table>
<thead>
<tr>
<th>Hexadecimal - base 16</th>
<th>2^11 2^10 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 0</td>
<td>1000 - 8</td>
</tr>
<tr>
<td>0001 - 1</td>
<td>1001 - 9</td>
</tr>
<tr>
<td>0010 - 2</td>
<td>1010 - A</td>
</tr>
<tr>
<td>0011 - 3</td>
<td>1011 - B</td>
</tr>
<tr>
<td>0100 - 4</td>
<td>1100 - C</td>
</tr>
<tr>
<td>0101 - 5</td>
<td>1101 - D</td>
</tr>
<tr>
<td>0110 - 6</td>
<td>1110 - E</td>
</tr>
<tr>
<td>0111 - 7</td>
<td>1111 - F</td>
</tr>
</tbody>
</table>
Hexadecimal Notation

Long strings of bits are tedious and error-prone to transcribe, so we often use a higher-radix notation, choosing the radix so that it’s simple to recover the original bit string.

A popular choice is to transcribe numbers in base-16, called hexadecimal. Each group of 4 adjacent bits is represented as a single hexadecimal digit.

Hexadecimal - base 16

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Base 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 0</td>
<td>1000 - 8</td>
</tr>
<tr>
<td>0001 - 1</td>
<td>1001 - 9</td>
</tr>
<tr>
<td>0010 - 2</td>
<td>1010 - A</td>
</tr>
<tr>
<td>0011 - 3</td>
<td>1011 - B</td>
</tr>
<tr>
<td>0100 - 4</td>
<td>1100 - C</td>
</tr>
<tr>
<td>0101 - 5</td>
<td>1101 - D</td>
</tr>
<tr>
<td>0110 - 6</td>
<td>1110 - E</td>
</tr>
<tr>
<td>0111 - 7</td>
<td>1111 - F</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccccccccccc}
2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0 & b & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
0b011111010000 = 0x7D0
\]
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10

\[
\begin{array}{c}
14 \\
+ 7
\end{array}
\]
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10

\[
\begin{array}{c}
1 \\
14 \\
+ 7 \\
\hline
1
\end{array}
\]
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10

\[
\begin{align*}
14 & \quad + \quad 7 \\
\hline
1 & \\
\end{align*}
\]

\text{carry}
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10

14
+ 7
\[ \underline{21} \]
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>
Binary Addition and Subtraction

1. Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>01</td>
</tr>
</tbody>
</table>

1 ➔ carry
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>101</td>
</tr>
</tbody>
</table>

1 carry
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>0101</td>
</tr>
</tbody>
</table>
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10 | Base 2
---|---
14 | 111
+ 7 | + 111
---|---
21 | 10101

*carry*
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 + 7 = 21</td>
<td>111 + 111 = 10101</td>
</tr>
</tbody>
</table>

**Carry**
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

\[
\begin{array}{cccccc}
\text{Base 10} & & & & & \\
\text{14} & + & \text{7} & \rightarrow & \text{carry} & \text{21} \\
\text{Base 2} & & & & & \\
\text{1110} & + & \text{111} & \rightarrow & \text{10101} \\
\end{array}
\]

-1

\[
\begin{array}{cccc}
\text{14} & - & \text{7} & \rightarrow & \text{7} \\
\end{array}
\]
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10 | Base 2
---|---
14 + 7 | 111 + 111
---|---
21 | 10101

carry

-1 borrow

14 - 7
---
7
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10.

**Base 10** | **Base 2**
---|---
14 + 7 | 111 + 111
| 21 | 1110 + 10101

- **carry**
- **borrow**

September 5, 2019
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>23</td>
<td>10101</td>
</tr>
</tbody>
</table>

**carry**

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>110</td>
</tr>
</tbody>
</table>

**borrow**
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- Addition with carry:

<table>
<thead>
<tr>
<th>1 - carry</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1100</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- Subtraction with borrow:

<table>
<thead>
<tr>
<th>-1 - borrow</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 + 7</td>
<td>1110 + 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- Carry: 1
- Borrow: -1
## Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- **carry**

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>111</td>
</tr>
</tbody>
</table>

- **borrow**
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10 | Base 2
---|---
14 + 7 = 21 | 1110 + 111 = 10101
14 - 7 = 07 | 1110 - 111 = 0111

1 carry

-1 borrow
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

Addition example:

- **carry**: 1

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>-1</td>
<td>-111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

Subtraction example:

- **borrow**: 1

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- Addition: 14 \( + 7 = 21 \)
- Subtraction: 14 \( - 7 = 07 \)

\( \text{carry} \)

\( -1 \) \( \text{borrow} \)

- Addition: 1110 \( - 111 = 0111 \)
- Subtraction: 1110 \( - 101 = 011 \)
**Binary Addition and Subtraction**

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 + 7</td>
<td>1110 + 111</td>
</tr>
<tr>
<td><strong>21</strong></td>
<td><strong>10101</strong></td>
</tr>
</tbody>
</table>

1. *carry*

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 - 7</td>
<td>1110 - 111</td>
</tr>
<tr>
<td><strong>07</strong></td>
<td><strong>0111</strong></td>
</tr>
</tbody>
</table>

-1. *borrow*
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

1 carry

-1 borrow

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

- 101

011

110
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

-1 carry

-1 borrow

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

-1 borrow

-1

011

101

110
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

**Carry**: 1

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

**Borrow**: 1

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 101</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

**Borrow**: 1

September 5, 2019
MIT 6.004 Fall 2019
L01-19
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

Base 10

<table>
<thead>
<tr>
<th>14</th>
<th>+</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base 2

<table>
<thead>
<tr>
<th>1110</th>
<th>+</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What does this mean?

Base 10

<table>
<thead>
<tr>
<th>14</th>
<th>-</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base 2

<table>
<thead>
<tr>
<th>1110</th>
<th>-</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-1

011

???

110
Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

-1  borrow

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>- 7</td>
<td>- 111</td>
</tr>
<tr>
<td>07</td>
<td>0111</td>
</tr>
</tbody>
</table>

What does this mean?

-2^3 + 0b110

-1  011

-1  101

??? 110
# Binary Addition and Subtraction

- Addition and subtraction in base 2 are performed just like in base 10

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>+ 7</td>
<td>+ 111</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
</tbody>
</table>

- What does this mean?
  - \(-2^3 + 0b110\)
  - We need a way to represent negative numbers!
If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition).

- This is known as an overflow.
Binary Modular Arithmetic

- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an overflow

- Common approach: Ignore the extra bit
  - Gives rise to **modular arithmetic**: With N-bit numbers, equivalent to following all operations with \( \text{mod } 2^N \)
### Binary Modular Arithmetic

- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an **overflow**
- Common approach: Ignore the extra bit
  - Gives rise to **modular arithmetic**: With N-bit numbers, equivalent to following all operations with $\text{mod } 2^N$
  - Visually, numbers “wrap around”:
Binary Modular Arithmetic

- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an overflow
- Common approach: Ignore the extra bit
  - Gives rise to modular arithmetic: With N-bit numbers, equivalent to following all operations with mod $2^N$
  - Visually, numbers “wrap around”:
Binary Modular Arithmetic

- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an overflow
- Common approach: Ignore the extra bit
  - Gives rise to modular arithmetic: With N-bit numbers, equivalent to following all operations with mod $2^N$
  - Visually, numbers “wrap around”:

Example: $(3 - 5) \mod 2^3$?
If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)

- This is known as an overflow

Common approach: Ignore the extra bit

- Gives rise to modular arithmetic: With N-bit numbers, equivalent to following all operations with $\text{mod } 2^N$
- Visually, numbers “wrap around”:

Example: $(3 - 5) \text{ mod } 2^3$ ?
Binary Modular Arithmetic

- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an overflow
- Common approach: Ignore the extra bit
  - Gives rise to modular arithmetic: With N-bit numbers, equivalent to following all operations with mod $2^N$
  - Visually, numbers “wrap around”:

  Example: $(3 - 5) \mod 2^3$?
Encoding Negative Integers

Attempt #1: Use a sign-magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “−”) separately from its magnitude (using decimal digits).

We could use the same approach for binary representations:

```
1 1 1 1 1 1 1 0 1 0 0 0 0 0
```

= -2000
Encoding Negative Integers

Attempt #1: Use a sign-magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “-“) separately from its magnitude (using decimal digits).

We could use the same approach for binary representations:

```
1 1 1 1 1 1 1 0 1 0 0 0 0 0
```

“0” for “+”
“1” for “-”

= -2000
Encoding Negative Integers

Attempt #1: Use a sign-magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “-”) separately from its magnitude (using decimal digits).

We could use the same approach for binary representations:

```
1 1 1 1 1 1 1 0 1 0 0 0 0
```

“0” for “+”
“1” for “-”

What issues might this encoding have?
Encoding Negative Integers

Attempt #1: Use a sign-magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “-”) separately from its magnitude (using decimal digits).

We could use the same approach for binary representations:

```
0 1 1 1 1 1 1 0 1 0 0 0 0 0
```

“0” for “+”
“1” for “-”

What issues might this encoding have?
Two representations for 0 (+0, -0)
Encoding Negative Integers

Attempt #1: Use a sign-magnitude representation for decimal numbers, encoding the sign of the number (using “+” and “-”) separately from its magnitude (using decimal digits).

We could use the same approach for binary representations:

```
1 1 1 1 1 1 1 0 1 0 0 0 0
```

“0” for “+”
“1” for “-”

What issues might this encoding have?

- Two representations for 0 (+0, -0)
- Addition and subtraction use different algorithms and are more complex than with unsigned numbers
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic?
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic? Yes!
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic?

Yes!
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic? Yes!
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic? Yes!

Diagram showing binary representation of numbers with negative values marked in red.
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic? **Yes!**
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic? Yes!

This is called two’s complement encoding.
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the \( N \)-bit representation has negative weight:

\[
v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i
\]

- Negative numbers have “1” in the high-order bit
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- Negative numbers have “1” in the high-order bit
- *Most negative number?* 10...0000
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- Negative numbers have “1” in the high-order bit
- Most negative number? 10...0000 -2^{N-1}
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- Negative numbers have “1” in the high-order bit
- **Most negative number?** 10...0000  \(-2^{N-1}\)
- **Most positive number?** 01...1111
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- Negative numbers have “1” in the high-order bit
- **Most negative number?** 10...0000 \(-2^{N-1}\)
- **Most positive number?** 01...1111 \(+2^{N-1} - 1\)
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- **Negative numbers have “1” in the high-order bit**
- **Most negative number?** \(10...0000\) \(-2^{N-1}\)
- **Most positive number?** \(01...1111\) \(+2^{N-1} - 1\)
- **If all bits are 1?** \(11...1111\)
Two’s Complement Encoding

In two’s complement encoding, the high-order bit of the N-bit representation has negative weight:

\[ v = -2^{N-1}b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \]

- Negative numbers have “1” in the high-order bit
- **Most negative number?** 10...0000 \(-2^{N-1}\)
- **Most positive number?** 01...1111 \(+2^{N-1} - 1\)
- **If all bits are 1?** 11...1111 \(-1\)
Two’s Complement and Arithmetic

- To negate a number (i.e., compute $-A$ given $A$), we invert all the bits and add one
Two’s Complement and Arithmetic

- To negate a number (i.e., compute \(-A\) given \(A\)), we invert all the bits and add one

  Why does this work?
Two’s Complement and Arithmetic

- To negate a number (i.e., compute $-A$ given $A$), we invert all the bits and add one

  Why does this work?

  $$-A + A = 0 = -1 + 1$$
Two’s Complement and Arithmetic

- To negate a number (i.e., compute \(-A\) given \(A\)), we invert all the bits and add one.

Why does this work?

\[-A + A = 0 = -1 + 1\]

\[-A = (-1 - A) + 1\]
Two’s Complement and Arithmetic

- To negate a number (i.e., compute \(-A\) given \(A\)), we invert all the bits and add one.

Why does this work?

\[-A + A = 0 = -1 + 1\]
\[-A = (-1 - A) + 1\]
\[1 \ldots 1 1\]
\[-A_{n-1} \ldots A_1 A_0\]
Two’s Complement and Arithmetic

- To negate a number (i.e., compute \(-A\) given \(A\)), we invert all the bits and add one.

*Why does this work?*

\[-A + A = 0 = -1 + 1\]

\[-A = (-1 - A) + 1\]

\[
\begin{array}{c}
\downarrow \\
1 \ldots 1 1 \\
- A_{n-1} \ldots A_1 A_0 \\
\hline
\bar{A}_{n-1} \ldots \bar{A}_1 \bar{A}_0
\end{array}
\]
Two’s Complement and Arithmetic

- To negate a number (i.e., compute \(-A\) given \(A\)), we invert all the bits and add one

  Why does this work?

  \[
  -A + A = 0 = -1 + 1 \\
  -A = (-1 - A) + 1
  \]

- To compute \(A - B\), we can simply use addition and compute \(A + (-B)\)
  - Result: Same circuit can add and subtract!
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition
Two’s Complement Example

Compute 3 - 6 using 4-bit 2’s complement addition

3: 0011
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

$0011$
$+ 1010$

$1101$
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[ 0011 + 1010 = 1 \]

September 5, 2019

MIT 6.004 Fall 2019
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

\[
\begin{array}{c}
3: 0011 \\
6: 0110 \\
-6: 1010 \\
\hline
1 \\
0011 \\
+ 1010 \\
\hline
01
\end{array}
\]
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[ \begin{align*}
0011 & \quad 1 \\
0011 & \quad + \\
1010 & \quad \underline{101}
\end{align*} \]
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[ \begin{array}{c}
1 \\
0011 \\
+ \\
1010 \\
\hline \\
1101 \\
\end{array} \]
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: \ 0011$
- $6: \ 0110$
- $-6: \ 1010$

\[
\begin{array}{c}
\phantom{0}0011 \\
\phantom{0}1010 \\
+ \ 1010 \\
\hline
1101 \\
\end{array}
\]

Compute $3 - 2$ using 3-bit 2’s complement addition
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
1 \\
0011 \\
+ \\
1010 \\
\hline
1101
\end{array}
\]

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

\[ 0011 + 1010 = 1101 \]

Compute $3 - 2$ using 3-bit 2’s complement addition

- $3: 011$
- $2: 010$
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

\[
\begin{array}{c}
\phantom{1} \phantom{0011} \phantom{+} \\
0011 \\
\phantom{1} \phantom{0110} \phantom{=} \\
\rule{1cm}{0.1pt} \phantom{1010} \\
\phantom{1} \phantom{+} \\
\hline
1101
\end{array}
\]

Compute $3 - 2$ using 3-bit 2’s complement addition

- $3: 011$
- $2: 010$
- $-2: 110$
Two’s Complement Example

**Compute 3 – 6 using 4-bit 2’s complement addition**
- 3: 0011
- 6: 0110
- -6: 1010

\[ \begin{array}{c}
3: 0011 \\
6: 0110 \\
-6: 1010 \\
\end{array} \]
\[ \begin{array}{c}
1 \\
0011 \\
1010 \\
\end{array} \]
\[ \begin{array}{c}
+ \\
+ \\
+ \\
\end{array} \]
\[ \begin{array}{c}
1101 \\
1101 \\
1101 \\
\end{array} \]

**Compute 3 – 2 using 3-bit 2’s complement addition**
- 3: 011
- 2: 010
- -2: 110

\[ \begin{array}{c}
3: 011 \\
2: 010 \\
-2: 110 \\
\end{array} \]
\[ \begin{array}{c}
011 \\
110 \\
\end{array} \]
\[ \begin{array}{c}
+ \\
+ \\
\end{array} \]
\[ \begin{array}{c}
110 \\
110 \\
\end{array} \]
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition
- $3: 0011$
- $6: 0110$
- $-6: 1010$

$0011$
$+ 1010$
$1101$

Compute $3 - 2$ using 3-bit 2’s complement addition
- $3: 011$
- $2: 010$
- $-2: 110$

$011$
$+ 110$
$1$
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

\[
\begin{array}{c}
0011 \\
+ 1010 \\
\hline
1101
\end{array}
\]

Compute $3 - 2$ using 3-bit 2’s complement addition

- $3: 011$
- $2: 010$
- $-2: 110$

\[
\begin{array}{c}
011 \\
+ 110 \\
\hline
01
\end{array}
\]
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

$$
\begin{array}{r}
0011 \\
+ 1010 \\
\hline
1101
\end{array}
$$

Compute $3 - 2$ using 3-bit 2’s complement addition

- $3: 011$
- $2: 010$
- $-2: 110$

$$
\begin{array}{r}
011 \\
+ 110 \\
\hline
001
\end{array}
$$
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
1 \\
0011 \\
+ 1010 \\
\hline
1101
\end{array}
\]

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

\[
\begin{array}{c}
11 \\
011 \\
+ 110 \\
\hline
1001
\end{array}
\]
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
3: 0011 \\
+ 1010 \\
\hline
1101
\end{array}
\]

Keep only last 3 bits

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

\[
\begin{array}{c}
3: 011 \\
+ 110 \\
\hline
1001
\end{array}
\]

Keep only last 3 bits
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

```
0011
+ 1010
1101
```

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

```
011
+ 110
1001
```

Keep only last 3 bits

What does this 1 mean?
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
3: 0011 \\
6: 0110 \\
-6: 1010
\end{array}
\]

\[
\begin{array}{c}
1 \\
0011 \\
+ 1010
\end{array}
\]

\[
1101
\]

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

\[
\begin{array}{c}
3: 011 \\
2: 010 \\
-2: 110
\end{array}
\]

\[
\begin{array}{c}
11 \\
011 \\
+ 110
\end{array}
\]

\[
1001
\]

Keep only last 3 bits

What does this 1 mean?
Two’s Complement Example

Compute $3 - 6$ using 4-bit 2’s complement addition

- $3: 0011$
- $6: 0110$
- $-6: 1010$

\[
\begin{array}{c}
\text{3: 0011} \\
+ \text{1010} \\
\hline
\text{1101}
\end{array}
\]

Compute $3 - 2$ using 3-bit 2’s complement addition

- $3: 011$
- $2: 010$
- $-2: 110$

\[
\begin{array}{c}
\text{3: 011} \\
+ \text{110} \\
\hline
\text{1001}
\end{array}
\]

Keep only last 3 bits

What does this 1 mean?
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
3: 0011 \\
6: 0110 \\
-6: 1010
\end{array}
+ \begin{array}{c}
1 \\
0011 \\
1010
\end{array}
\hline
1101
\]

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

\[
\begin{array}{c}
3: 011 \\
2: 010 \\
-2: 110
\end{array}
+ \begin{array}{c}
11 \\
011 \\
110
\end{array}
\hline
1001
\]

Keep only last 3 bits

What does this 1 mean?
Two’s Complement Example

Compute 3 – 6 using 4-bit 2’s complement addition

- 3: 0011
- 6: 0110
- -6: 1010

\[
\begin{array}{c}
3: 0011 \\
+ 1010 \\
\hline
1101
\end{array}
\]

Compute 3 – 2 using 3-bit 2’s complement addition

- 3: 011
- 2: 010
- -2: 110

\[
\begin{array}{c}
3: 011 \\
+ 110 \\
\hline
1001
\end{array}
\]

Keep only last 3 bits

What does this 1 mean?

Zero crossing
Summary

- Digital systems encode information using binary for efficiency and reliability

- We can encode unsigned integers using strings of bits; long addition and subtraction are done as in decimal

- Two’s complement allows encoding negative integers while preserving the simplicity of unsigned arithmetic
Thank you!

Next lecture:
Introduction to assembly and RISC-V