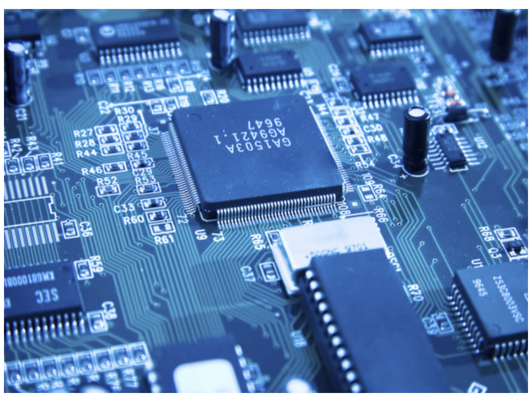
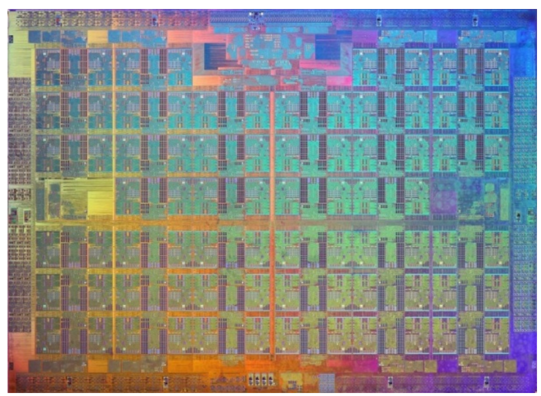
Welcome to 6.004!

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

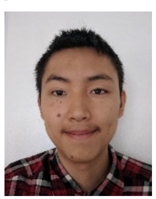
Daniel Sanchez
sanchez@csail.mit.edu

Silvina Hanono Wachman
silvina@mit.edu

Jason Miller
jasonm@csail.mit.edu

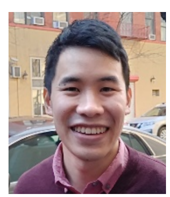
Teaching Assistants

Kathy Camenzind



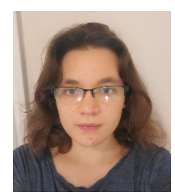
Brian Chen

Felipe Moreno



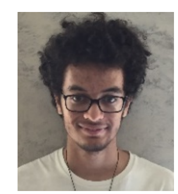
Quan Nguyen

Domenic Nutile



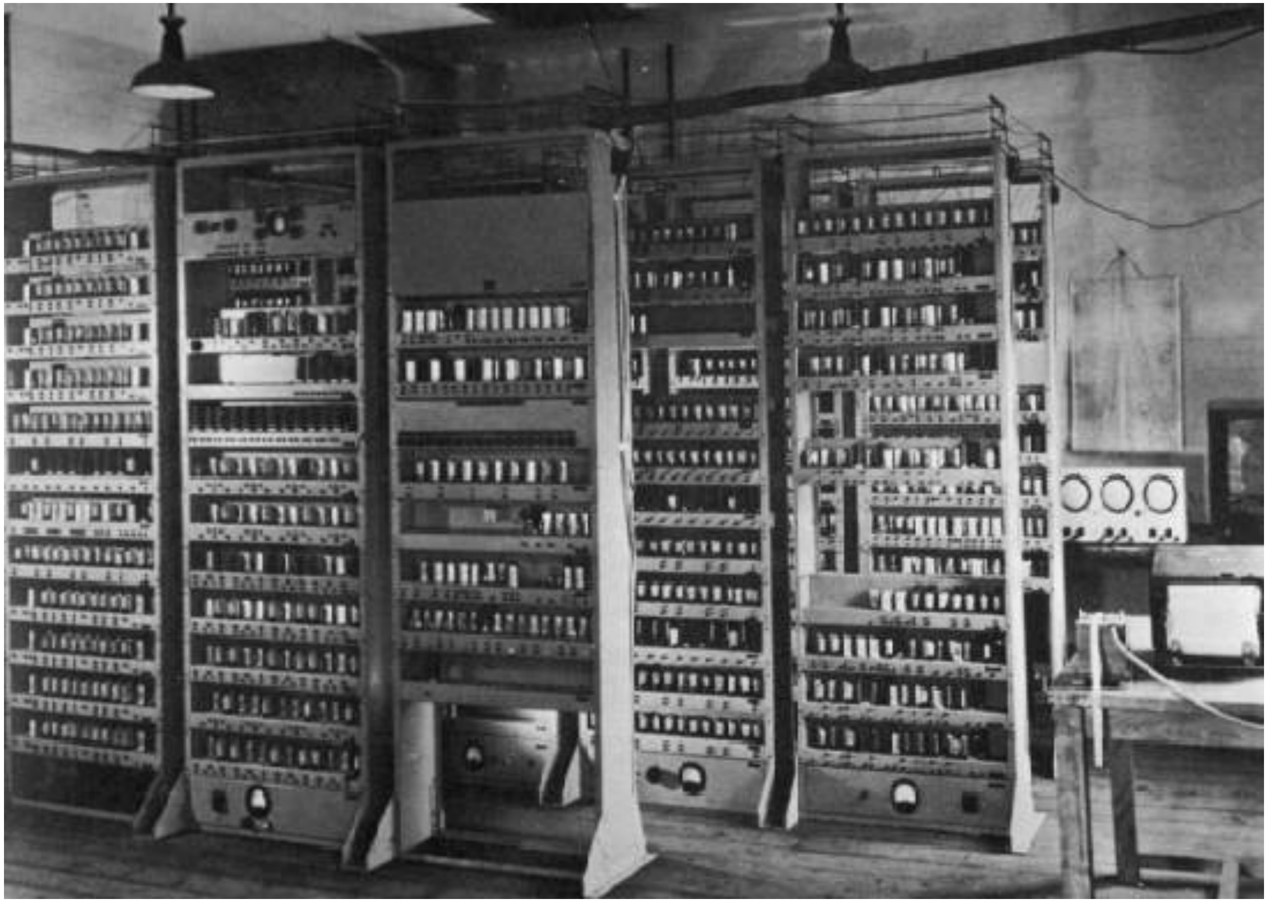
Ileana Rugina

Sebastian Bartlett



Kendall Garner

Computing Devices Then…



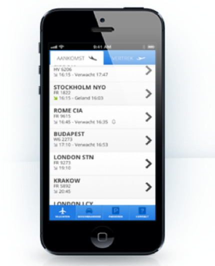
ENIAC, 1943   30 tons, 200KW, ~1000 ops/sec

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

An Introduction to the Digital World

Computer programs

Virtual machines

Computer systems

Operating systems, virtual memory, I/O

Instruction set + memory

Computer architecture

Processors, caches, pipelining

Digital circuits

Digital design

Combinational and sequential circuits

Bits, Logic gates

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

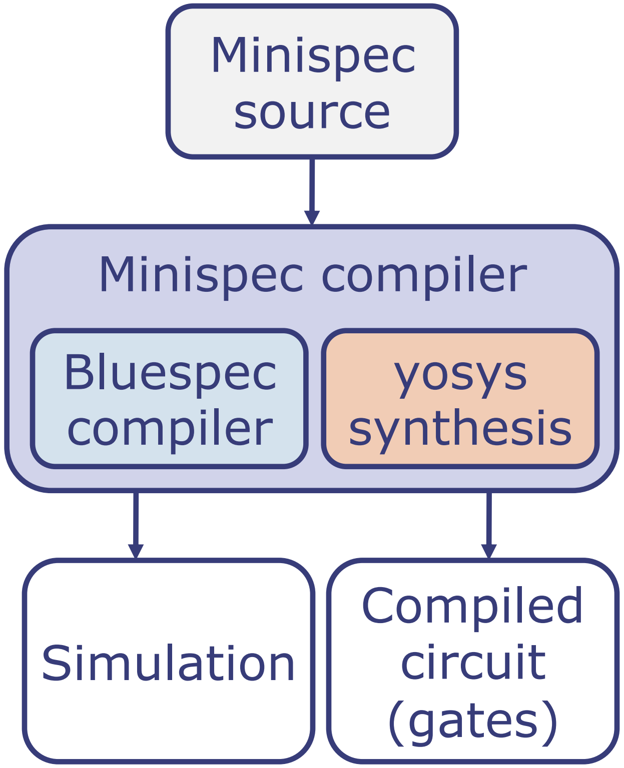
By the end of the term you will have designed a simple processor from scratch!

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



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 >= 165
  - B: Points >= 145
  - C: Points >= 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

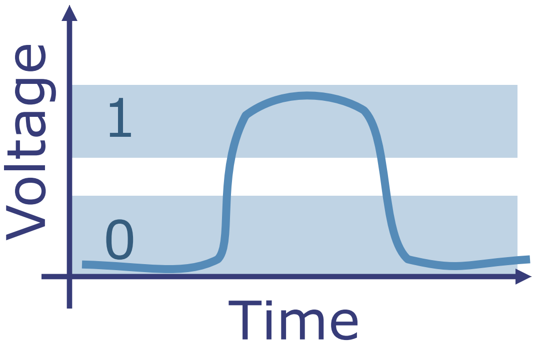
Digital Information Encoding

- 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

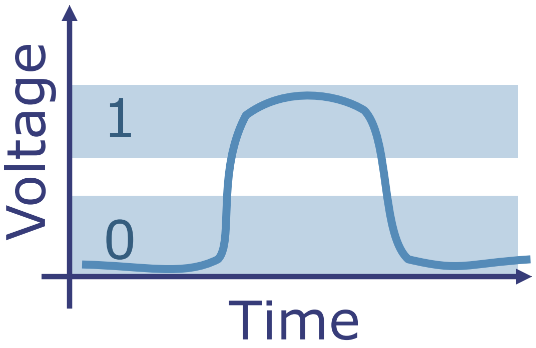
- These are typically binary digits (bits): 0 and 1



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:

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

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	0	1	0	0	0	0

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:

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

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	0	1	0	0	0	0

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:

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

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	0	1	0	0	0	0

What value does 011111010000 encode?

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$

$= 1024 + 512 + 256 + 128 + 64 + 16 = 2000$

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:

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

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	0	1	0	0	0	0

What value does 011111010000 encode?

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$

$= 1024 + 512 + 256 + 128 + 64 + 16 = 2000$

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:

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

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	0	1	0	0	0	0

What value does 011111010000 encode?

$V = 0*2^{11} + 1*2^{10} + 1*2^{9} + \ldots$

$= 1024 + 512 + 256 + 128 + 64 + 16 = 2000$

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:

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

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	0	1	0	0	0	0

What value does 011111010000 encode?

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$

$= 1024 + 512 + 256 + 128 + 64 + 16 = 2000$

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.

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	0	1	0	0	0	0

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

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

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	0	1	0	0	0	0

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

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F

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	0	1	0	0	0	0

$\underbrace{0\ 0\ 0\ 0}_{0}$

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

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F

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	0	1	0	0	0	0

$$0111\,\underbrace{1101}_{D}\,\underbrace{0000}_{0}$$

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

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F

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	0	1	0	0	0	0

$\underbrace{0111}_{7}\ \underbrace{1101}_{D}\ \underbrace{0000}_{0}$

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

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

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	0	1	0	0	0	0

$$\underbrace{0111}_{7}\;\underbrace{1101}_{D}\;\underbrace{0000}_{0}$$

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

$$\begin{array}{r} 14 \\ +\ \ 7 \\ \hline \end{array}$$

Binary Addition and Subtraction

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

Base 10

$$
\begin{array}{r}
{\color{red}\mathit{1}} \\
14 \\
+\;\;7 \\
\hline
{\color{red}\mathit{1}}
\end{array}
$$

Binary Addition and Subtraction

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

Base 10

$$\begin{array}{r} \color{red}{\mathit{1}} \\ 14 \\ +\ \ 7 \\ \hline \color{red}{\mathit{1}} \end{array}$$

carry

Binary Addition and Subtraction

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

Base 10

$$\begin{array}{r} \color{red}{1} \\ 14 \\ +\ \ 7 \\ \hline \color{red}{21} \end{array}$$

carry

Binary Addition and Subtraction

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

Base 10	Base 2
$\begin{array}{r} {\color{red}1} \\ 14 \\ +\ \ 7 \\ \hline {\color{red}21} \end{array}$ *carry*	$\begin{array}{r} 1110 \\ +\ \ 111 \\ \hline \end{array}$

Binary Addition and Subtraction

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

Base 10

carry

$$\begin{array}{r} \color{red}{1} \\ 14 \\ +\ \ 7 \\ \hline \color{red}{21} \end{array}$$

Base 2

$$\begin{array}{r} 1110 \\ +\ \ 111 \\ \hline \color{red}{1} \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2
1 *carry*	*1*
14	1110
+ 7	+ 111
21	*01*

Binary Addition and Subtraction

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

Base 10	Base 2
1 *carry*	*11*
14	1110
+ 7	+ 111
21	*101*

Binary Addition and Subtraction

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

Base 10	Base 2
1 carry	*111*
14	1110
+ 7	+ 111
21	*0101*

Binary Addition and Subtraction

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

Base 10	Base 2
1 ← *carry*	*111*
14	1110
+ 7	+ 111
21	*10101*

Binary Addition and Subtraction

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

Base 10	Base 2
$$\begin{array}{r} \text{1} \quad \textit{carry} \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$$	$$\begin{array}{r} 111 \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$$

$$\begin{array}{r} 14 \\ -\ \ 7 \\ \hline \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2
1 *carry*	*111*
14	1110
+ 7	+ 111
21	*10101*

-1
14
− 7
7

Binary Addition and Subtraction

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

Base 10

carry

$$\begin{array}{r} 1 \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$$

borrow

$$\begin{array}{r} -1 \\ 14 \\ -\ \ 7 \\ \hline 7 \end{array}$$

Base 2

$$\begin{array}{r} 111 \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2

Base 10:

$$\begin{array}{r} \scriptstyle 1 \quad \textit{carry} \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$$

Base 2:

$$\begin{array}{r} \scriptstyle 111 \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$$

$$\begin{array}{r} \scriptstyle -1 \quad \textit{borrow} \\ 14 \\ -\ \ 7 \\ \hline 07 \end{array}$$

Binary Addition and Subtraction

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

Base 10

carry

$$\begin{array}{r} {\color{red}1} \\ 14 \\ +\ \ 7 \\ \hline {\color{red}21} \end{array}$$

borrow

$$\begin{array}{r} {\color{red}-1} \\ 14 \\ -\ \ 7 \\ \hline {\color{red}07} \end{array}$$

Base 2

$$\begin{array}{r} {\color{red}111} \\ 1110 \\ +\ \ 111 \\ \hline {\color{red}10101} \end{array}$$

$$\begin{array}{r} 1110 \\ -\ \ 111 \\ \hline \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2
$\begin{array}{r} \textit{1} \quad \textit{carry} \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$	$\begin{array}{r} 111\ \ \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$
$\begin{array}{r} -1 \quad \textit{borrow} \\ 14 \\ -\ \ 7 \\ \hline 07 \end{array}$	$\begin{array}{r} -1\ \ \\ 1110 \\ -\ \ 111 \\ \hline 1 \end{array}$

Binary Addition and Subtraction

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

Base 10	Base 2
$\begin{array}{r} \scriptstyle 1 \ \text{carry} \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$	$\begin{array}{r} \scriptstyle 111 \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$
$\begin{array}{r} \scriptstyle -1 \ \text{borrow} \\ 14 \\ -\ \ 7 \\ \hline 07 \end{array}$	$\begin{array}{r} \scriptstyle -1-1 \\ 1110 \\ -\ \ 111 \\ \hline 11 \end{array}$

Binary Addition and Subtraction

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

Base 10	Base 2
1 carry	*1 1 1*
14	1110
+ 7	+ 111
2 1	*1 0 1 0 1*
-1 borrow	-1-1-1
14	1110
– 7	– 111
0 7	*1 1 1*

Binary Addition and Subtraction

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

Base 10

carry

$$\begin{array}{r} 1 \\ 14 \\ +\ \ 7 \\ \hline 21 \end{array}$$

borrow

$$\begin{array}{r} -1 \\ 14 \\ -\ \ 7 \\ \hline 07 \end{array}$$

Base 2

$$\begin{array}{r} 111 \\ 1110 \\ +\ \ 111 \\ \hline 10101 \end{array}$$

$$\begin{array}{r} -1-1-1 \\ 1110 \\ -\ \ 111 \\ \hline 0111 \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2	
$$\begin{array}{rr} & 1 \text{ — carry} \\ & 14 \\ + & 7 \\ \hline & 21 \end{array}$$	$$\begin{array}{rr} & 111 \\ & 1110 \\ + & 111 \\ \hline & 10101 \end{array}$$	
$$\begin{array}{rr} & -1 \text{ — borrow} \\ & 14 \\ - & 7 \\ \hline & 07 \end{array}$$	$$\begin{array}{rr} & -1-1-1 \\ & 1110 \\ - & 111 \\ \hline & 0111 \end{array}$$	$$\begin{array}{rr} & 011 \\ - & 101 \\ \hline \end{array}$$

Binary Addition and Subtraction

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

Base 10	Base 2	
1 *carry*	*111*	
14	1110	
+ 7	+ 111	
21	*10101*	
-1 *borrow*	*-1-1-1*	
14	1110	011
− 7	− 111	− 101
07	*0111*	*0*

Binary Addition and Subtraction

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

Base 10	Base 2	
1 (*carry*) 14 + 7 *21*	*111* 1110 + 111 *10101*	
-1 (*borrow*) 14 − 7 *07*	*-1-1-1* 1110 − 111 *0111*	011 − 101 *10*

Binary Addition and Subtraction

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

Base 10	Base 2	
$\begin{array}{r} \color{red}{1} \\ 14 \\ +\ \ 7 \\ \hline \color{red}{21} \end{array}$ carry	$\begin{array}{r} \color{red}{111\ } \\ 1110 \\ +\ \ \ 111 \\ \hline \color{red}{10101} \end{array}$	
$\begin{array}{r} \color{red}{-1} \\ 14 \\ -\ \ 7 \\ \hline \color{red}{07} \end{array}$ borrow	$\begin{array}{r} \color{red}{-1-1-1\ } \\ 1110 \\ -\ \ \ 111 \\ \hline \color{red}{0111} \end{array}$	$\begin{array}{r} 011 \\ -\ \ \ 101 \\ \hline \color{red}{110} \end{array}$

Binary Addition and Subtraction

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

Base 10	Base 2	
1 (*carry*)	*111*	
14	1110	
+ 7	+ 111	
21	*10101*	
-1 (*borrow*)	*-1-1-1*	*-1*
14	1110	011
− 7	− 111	− 101
07	*0111*	*110*

Binary Addition and Subtraction

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

Base 10		Base 2	
1	carry	111	
14		1110	
+ 7		+ 111	
21		10101	
-1	borrow	-1-1-1	-1
14		1110	011
− 7		− 111	− 101
07		0111	??? 110

Binary Addition and Subtraction

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

Base 10	Base 2	
1 *carry*	*111*	*What does this mean?*
14	1110	
+ 7	+ 111	
21	*10101*	
-1 *borrow*	*-1-1-1*	*-1*
14	1110	011
− 7	− 111	− 101
07	*0111*	*???* *110*

Binary Addition and Subtraction

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

Base 10	Base 2	
carry		What does this mean?
1	111	$-2^3 + 0b110$
14	1110	
+ 7	+ 111	
21	10101	
borrow		
-1	-1-1-1	-1
14	1110	011
− 7	− 111	− 101
07	0111	??? 110

Binary Addition and Subtraction

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

Base 10	Base 2	
1 (*carry*) 14 + 7 *21*	*111* 1110 + 111 *10101*	*What does this mean?* *-2^3 + 0b110* *We need a way to represent negative numbers!*
-1 (*borrow*) 14 − 7 *07*	*-1-1-1* 1110 − 111 *0111*	*-1* 011 − 101 *???110*

Binary Modular Arithmetic

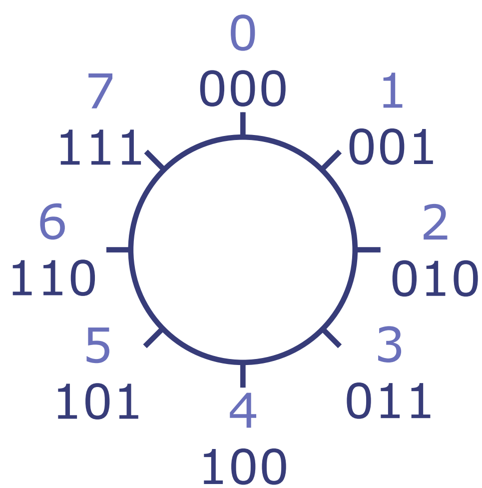
- If we use a fixed number of bits, addition and other operations may produce results outside the range that the output can represent (up to 1 extra bit for addition)
  - This is known as an overflow

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

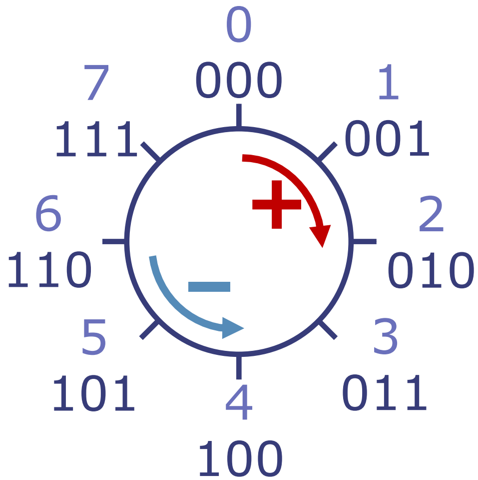
Binary Modular Arithmetic

- 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":



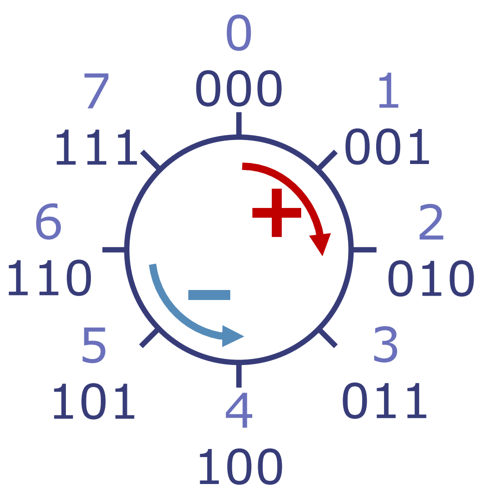
Binary Modular Arithmetic

- 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":

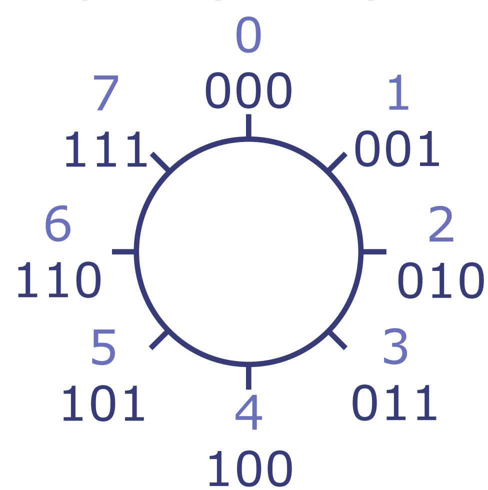


Binary Modular Arithmetic

- 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":

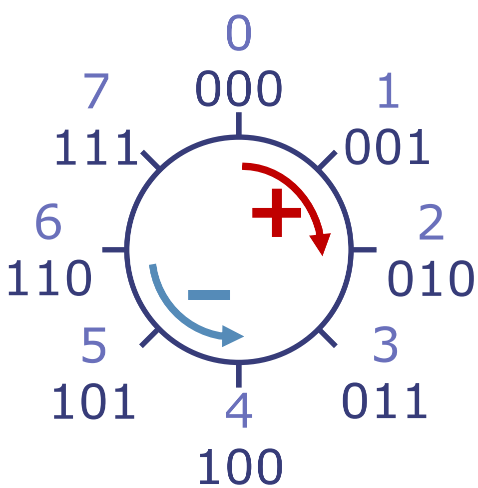


Example: (3 – 5) mod 2^3 ?

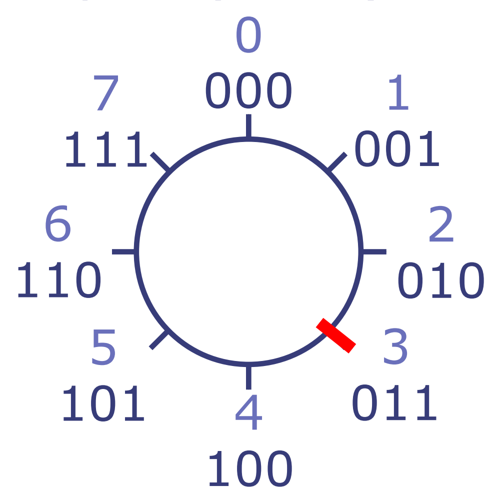


Binary Modular Arithmetic

- 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":

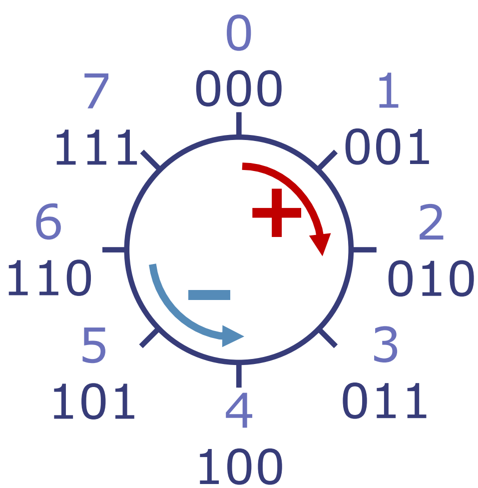


Example: (3 – 5) mod 2^3 ?

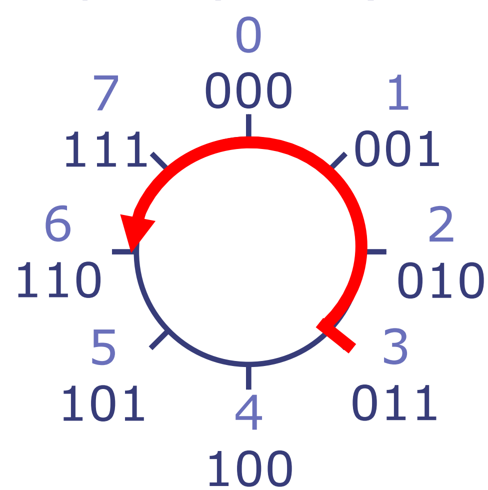


Binary Modular Arithmetic

- 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	0	1	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	0	1	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	0	1	0	0	0	0

"0" for "+"
"1" for "-"

=-2000

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:

1	1	1	1	1	1	0	1	0	0	0	0

"0" for "+"
"1" for "-"

=-2000

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	0	1	0	0	0	0

"0" for "+"
"1" for "-"

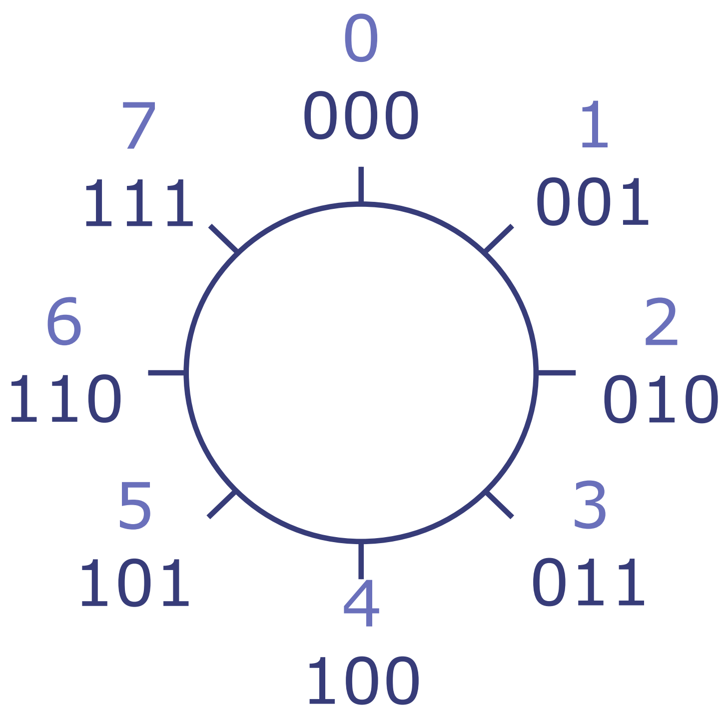
=-2000

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

Deriving a Better Encoding

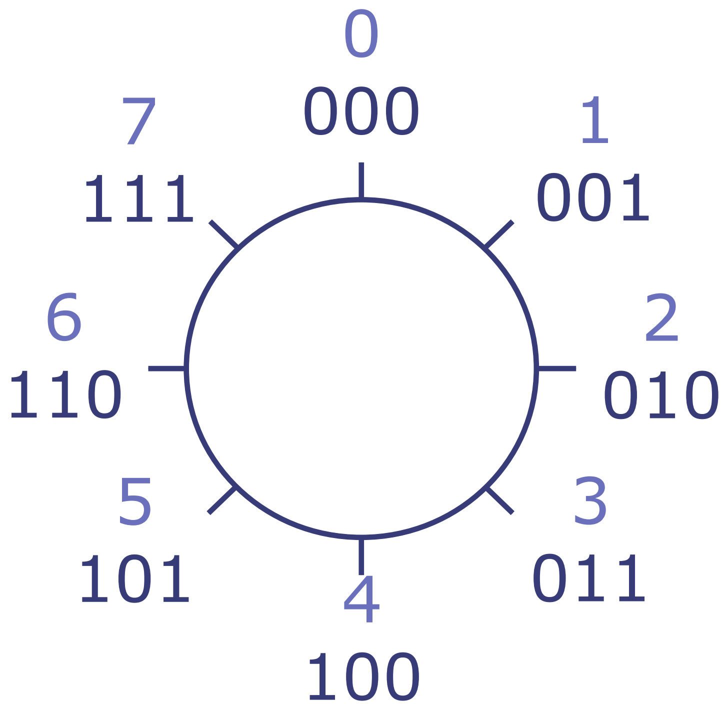
Can you simply relabel some of the digits to represent negative numbers while retaining the nice properties of modular arithmetic?



Deriving a Better Encoding

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

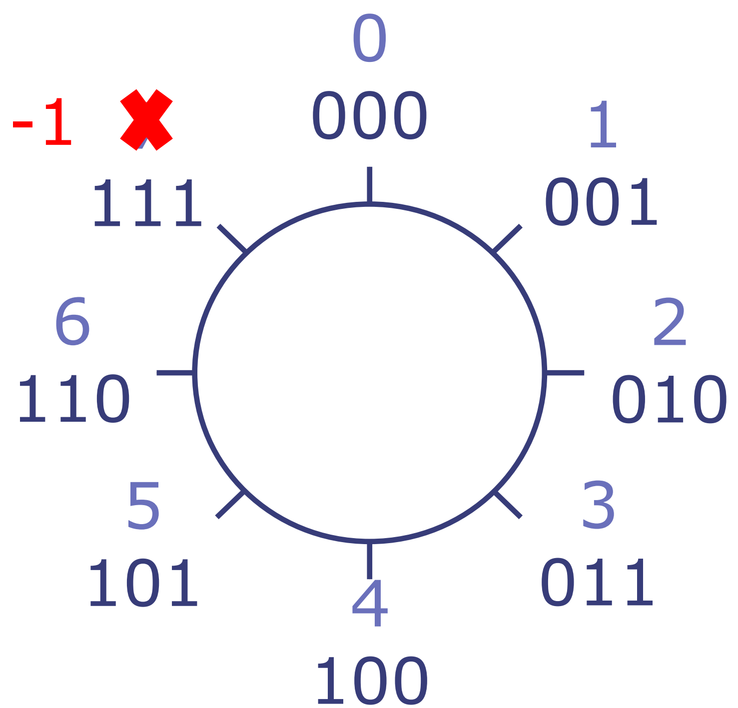
Yes!



Deriving a Better Encoding

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

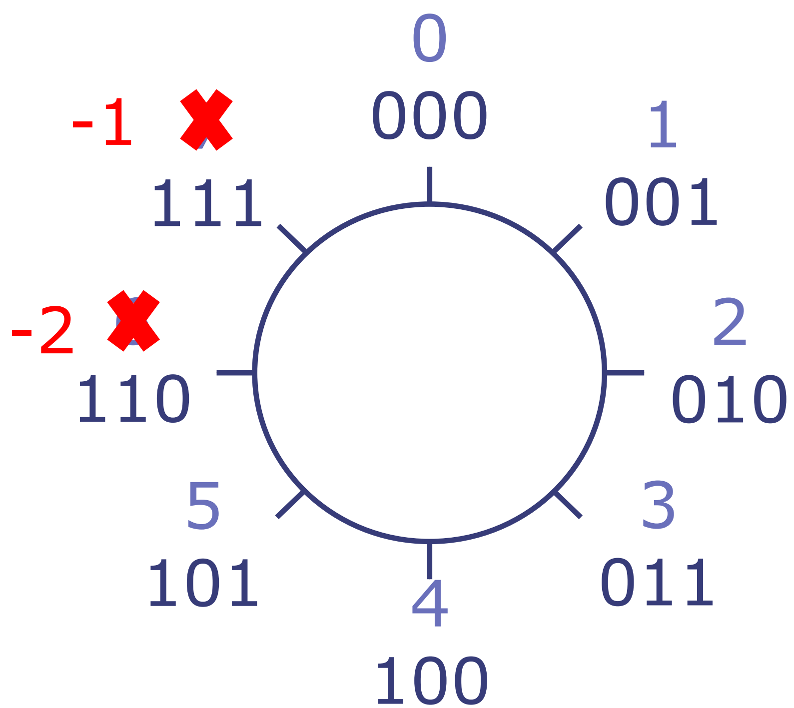
Yes!



Deriving a Better Encoding

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

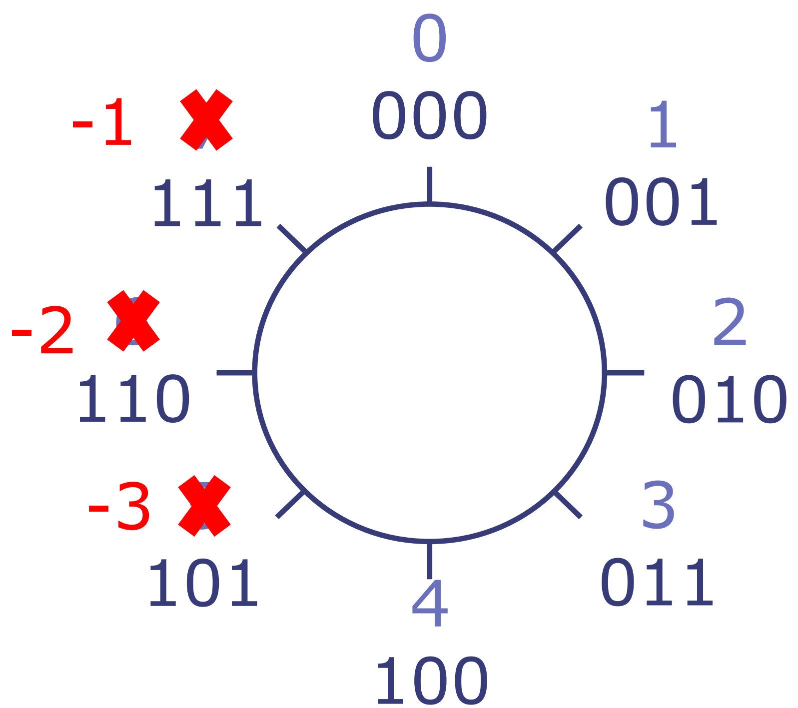
Yes!



Deriving a Better Encoding

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

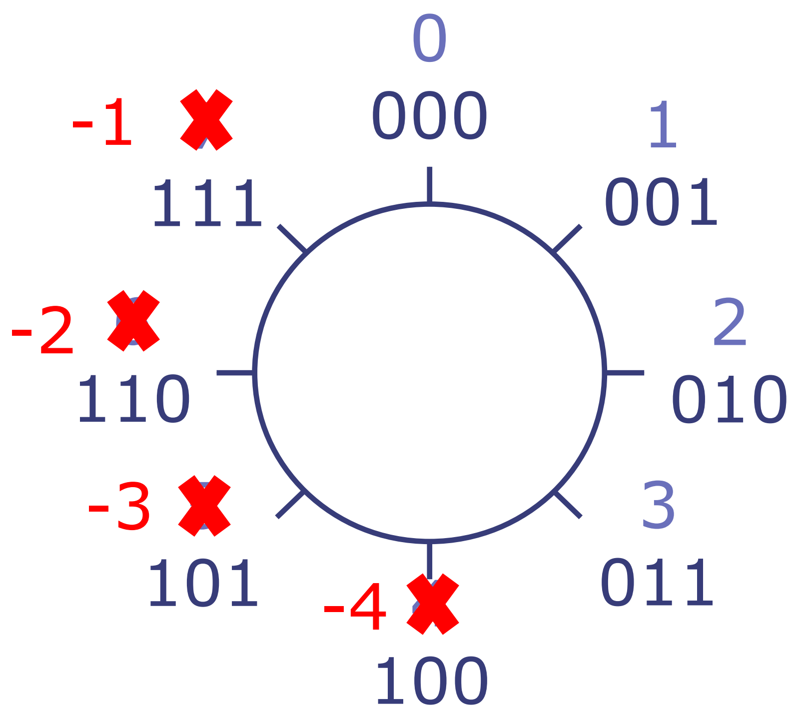
Yes!



Deriving a Better Encoding

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

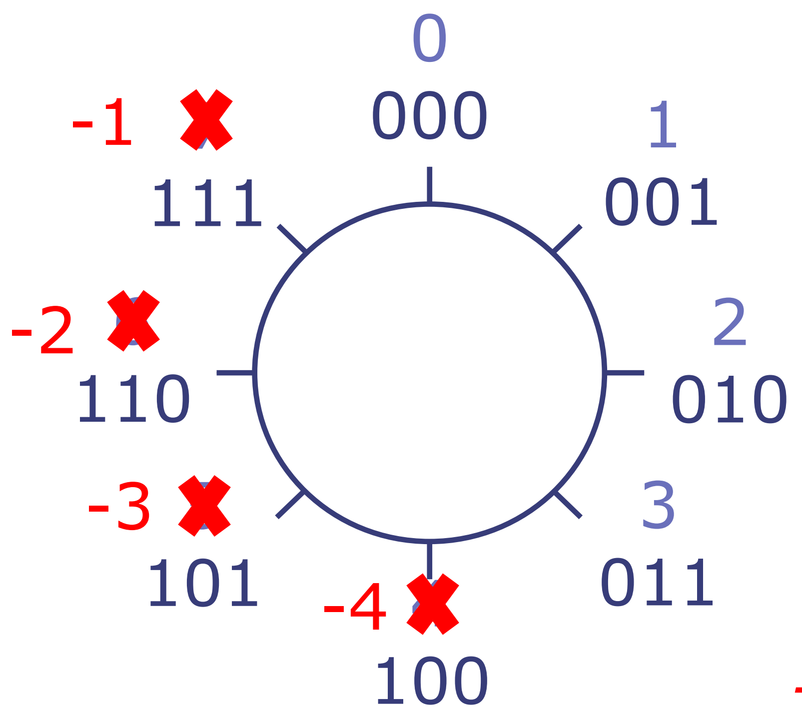
Yes!



Deriving a Better Encoding

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:

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

N bits

-2^{N-1}	2^{N-2}	⋯	⋯	⋯	2^3	2^2	2^1	2^0

Two's Complement Encoding

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

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

N bits

-2^{N-1}	2^{N-2}	…	…	…	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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:

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

N bits

-2^{N-1}	2^{N-2}	···	···	···	2^3	2^2	2^1	2^0

- 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?

$$-A + A = 0 = -1 + 1$$

$$-A = \underbrace{(-1 - A)}_{\begin{array}{r} 1\ \ldots 1\ 1 \\ -\ A_{n-1}\ldots A_1 A_0 \\ \hline \end{array}} + 1$$

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?

$$-A + A = 0 = -1 + 1$$

$$-A = \underbrace{(-1 - A)}_{} + 1$$

$$\begin{array}{r} 1 \ldots 1\ 1 \\ -\ A_{n-1} \ldots A_1 A_0 \\ \hline \overline{A_{n-1}} \ldots \overline{A_1}\,\overline{A_0} \end{array}$$

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?

$$-A + A = 0 = -1 + 1$$

$$-A = \underbrace{(-1 - A)}_{} + 1$$

$$\begin{array}{r} 1 \ldots 1\ 1 \\ -\ A_{n-1} \ldots A_1 A_0 \\ \hline \overline{A_{n-1}} \ldots \overline{A_1}\,\overline{A_0} \end{array}$$

- 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*

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

Two's Complement Example

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

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

$$
\begin{array}{r}
0011 \\
+\ \ 1010 \\
\hline
\mathit{1}
\end{array}
$$

Two's Complement Example

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

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

$$\begin{array}{r} \color{red}{1} \\ 0011 \\ +\ \ 1010 \\ \hline \color{red}{01} \end{array}$$

Two's Complement Example

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

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

$$
\begin{array}{r}
\mathit{1} \\
0011 \\
+\ \ 1010 \\
\hline
\mathit{101}
\end{array}
$$

Two's Complement Example

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

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

$$
\begin{array}{r}
\textit{1} \\
0011 \\
+\ \ 1010 \\
\hline
\textit{1101}
\end{array}
$$

Two's Complement Example

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

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

$$
\begin{array}{r}
\color{red}{\mathit{1}} \\
0011 \\
+\ \ 1010 \\
\hline
\color{red}{\mathit{1101}}
\end{array}
$$

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*

$$\begin{array}{r} \color{red}{1} \\ 0011 \\ +\ \ 1010 \\ \hline \color{red}{1101} \end{array}$$

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*

$$\begin{array}{r} \color{red}{1} \\ 0011 \\ +\ 1010 \\ \hline \color{red}{1101} \end{array}$$

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*

$$
\begin{array}{r}
{\color{red}1} \\
0011 \\
+\ \ 1010 \\
\hline
{\color{red}1101}
\end{array}
$$

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*

$$\begin{array}{r} {\color{red}{\scriptstyle 1}} \\ 0011 \\ +\ \ 1010 \\ \hline {\color{red}{1101}} \end{array}$$

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

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

$$\begin{array}{r} 011 \\ +\ \ 110 \\ \hline \end{array}$$

Two's Complement Example

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

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

$$\begin{array}{r} \textit{1} \\ 0011 \\ +\ \ 1010 \\ \hline \textit{1101} \end{array}$$

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

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

$$\begin{array}{r} 011 \\ +\ \ 110 \\ \hline \textit{1} \end{array}$$

Two's Complement Example

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

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

$$\begin{array}{r} {\scriptstyle 1} \\ 0011 \\ +\ \ 1010 \\ \hline 1101 \end{array}$$

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

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

$$\begin{array}{r} {\scriptstyle 1} \\ 011 \\ +\ \ 110 \\ \hline 01 \end{array}$$

Two's Complement Example

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

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

$$\begin{array}{r} \scriptstyle 1 \\ 0011 \\ +\ 1010 \\ \hline \mathit{1101} \end{array}$$

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

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

$$\begin{array}{r} \scriptstyle 11 \\ 011 \\ +\ 110 \\ \hline \mathit{001} \end{array}$$

Two's Complement Example

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

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

$$\begin{array}{r} \scriptstyle 1 \\ 0011 \\ +\quad 1010 \\ \hline 1101 \end{array}$$

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

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

$$\begin{array}{r} \scriptstyle 11 \\ 011 \\ +\quad 110 \\ \hline 1001 \end{array}$$

Two's Complement Example

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

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

$$\begin{array}{r} {\scriptstyle 1} \\ 0011 \\ +\;\;1010 \\ \hline 1101 \end{array}$$

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

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

$$\begin{array}{r} {\scriptstyle 11} \\ 011 \\ +\;\;110 \\ \hline 1\underbrace{001} \end{array}$$

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*

$$\begin{array}{r} \textit{1} \\ 0011 \\ +\ 1010 \\ \hline \textit{1101} \end{array}$$

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

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

$$\begin{array}{r} \textit{11} \\ 011 \\ +\ 110 \\ \hline \textit{1}\underbrace{\textit{001}} \end{array}$$

What does this 1 mean?

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*

```
   1
   0011
+  1010
  -----
   1101
```

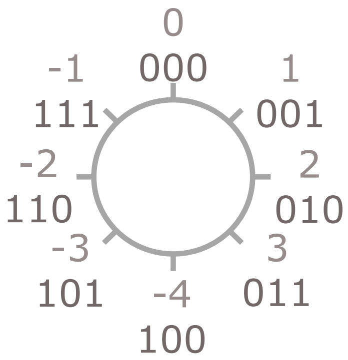
Compute 3 – 2 using 3-bit 2's complement addition

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

```
  11
   011
+  110
  -----
  1001
```

What does this 1 mean?

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*

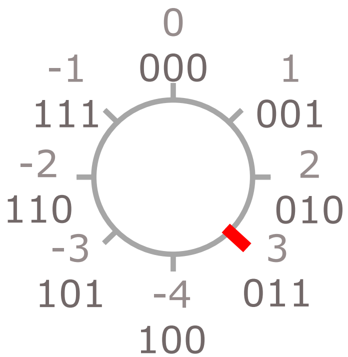
$$\begin{array}{rr} & \scriptstyle 1 \\ & 0011 \\ + & 1010 \\ \hline & 1101 \end{array}$$

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

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

$$\begin{array}{rr} & \scriptstyle 11 \\ & 011 \\ + & 110 \\ \hline & 1\underbrace{001}_{\text{Keep only last 3 bits}} \end{array}$$

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*

```
   1
   0011
+  1010
  -----
   1101
```

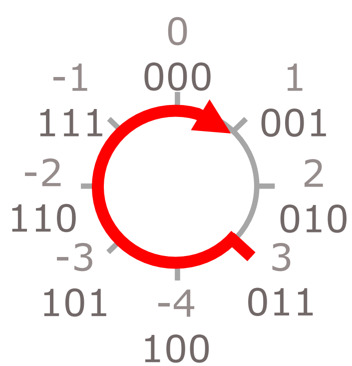
Compute 3 – 2 using 3-bit 2's complement addition

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

```
  11
   011
+  110
  -----
  1001
```

What does this 1 mean?

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*

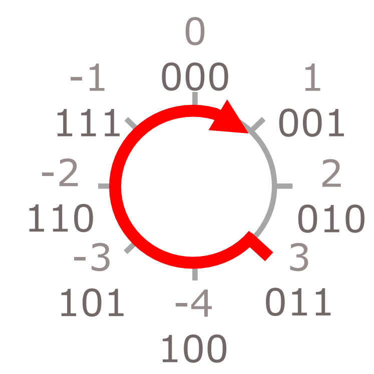
$$\begin{array}{r} \scriptstyle 1 \\ 0011 \\ +\ \ 1010 \\ \hline 1101 \end{array}$$

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

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

$$\begin{array}{r} \scriptstyle 11 \\ 011 \\ +\ \ 110 \\ \hline 1\underbrace{001}_{\text{Keep only last 3 bits}} \end{array}$$

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