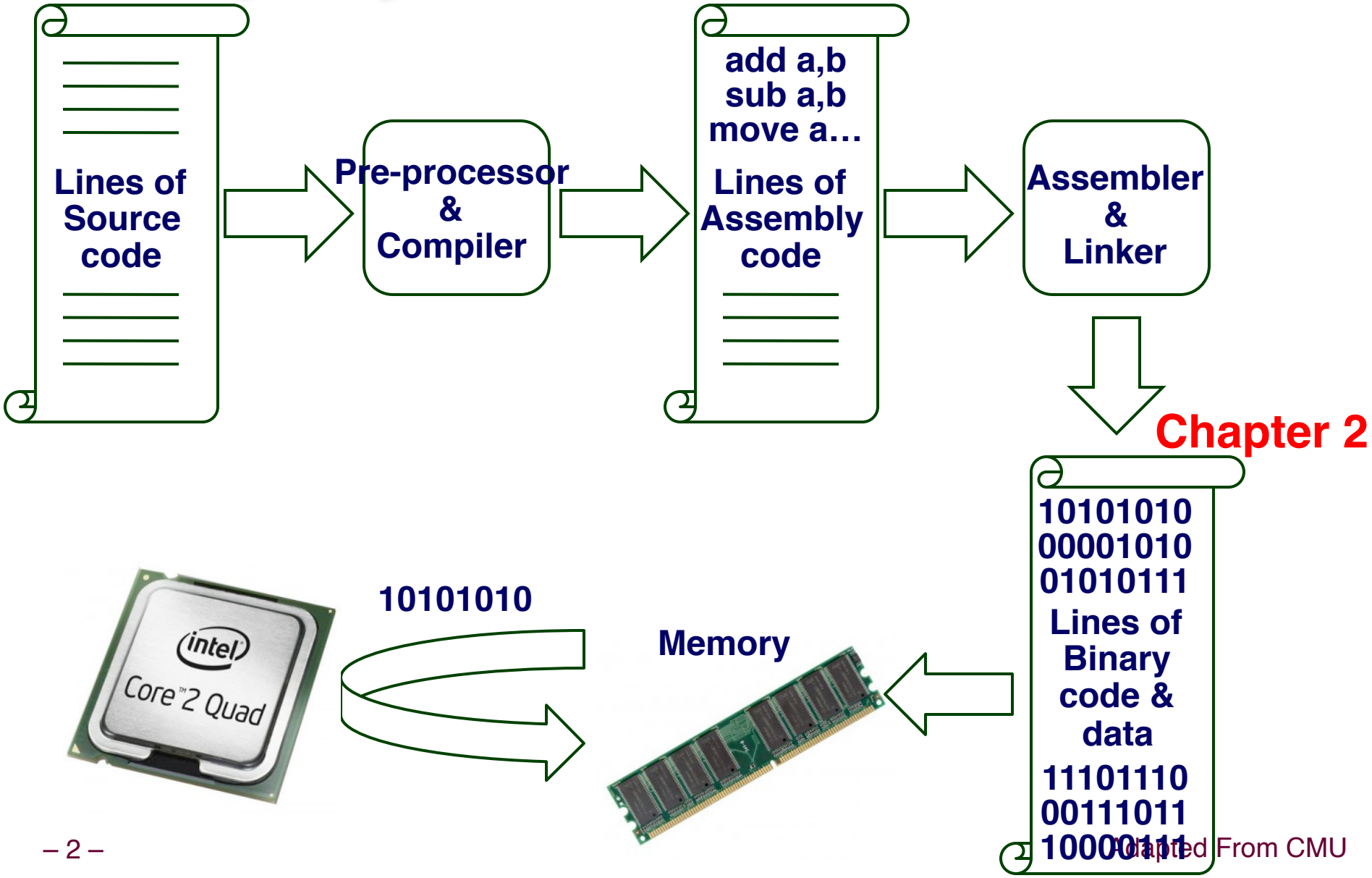
Chapter 2: Bits and Bytes

Topics

- Why bits?
- Binary Logic
- Representing information as bits
  - Binary/Hexadecimal
  - Byte representations
    - » ints, floats, double precision

Recap: Systems In a Nutshell



Adapted From CMU

Decimal Representation

Decimal representation (Base 10) is what we humans naturally gravitate to, given 10 digits (fingers)!

- **7126 base 10 = 7126_{10}**

  $= 7000 + 100 + 20 + 6$

  $= 7*10^3 + 1*10^2 + 2*10^1 + 6*10^0$

By convention, we write the most significant digit on the left, followed by the second most significant digit, then the 3rd most significant digit, etc., until we reach the least significant digit

Binary Representation – Why Bits?

$0\ 1\ 0\ 1_2 = 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

$= 0 + 4 + 0 + 1 = 5_{10}$

Each binary digit is called a 'bit'

Base 2 or Binary is easier to represent via electronic circuits than Base 10

- Digital transistor circuits can easily represent a '0' and a '1'
  - A '1' = +5 volts, a '0' = 0 volts, so only 2 voltage levels
  - Don't have to implement 10 different voltage levels as in base 10
- Easy to implement Base 2 arithmetic
  - much of Base 10 arithmetic using Base 2 arithmetic (integers are exact, but floats are approximations)
- Easy to implement digital logic (AND, OR, etc.) with binary

Digital Logic via Boolean Algebra

Developed by George Boole in 19th Century

- Algebraic representation of logic
  - Encode "True" as 1 and "False" as 0

And "&"

- A&B = 1 only when both A=1 and B=1

	&	B=0	B=1
A=0		0	0
A=1		0	1

A	B	&
0	0	0
1	0	0
0	1	0
1	1	1

Truth Tables (two forms)

Common Digital Logic Operations

And "&"

- A&B = 1 only when both A=1 and B=1

A \ & B	0	1
0	0	0
1	0	1

Or "|"

- A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not "~"

- ~A = 1 only when A=0

~	
0	1
1	0

Exclusive-Or (Xor) "^"

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

Other operators: NAND (Not AND), NOR (Not OR), etc.