Revised: 3/18/2020

Number Systems

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- Number Systems
- Binary System
  - Binary Addition
  - Binary Subtraction
- Octal System
- Hexadecimal System
- Two’s Complement Representation
- Units of Information
  - Powers of Two
- Bonus: CPU Performance

Number Systems

- A **number system** is a form of notation for expressing numbers using a certain set of symbols or digits.

- The system we use on a daily basis is the **decimal system** which uses the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 to represent both integers (numbers without a fraction) and non-integers (numbers with a fraction).

- There are other number systems aside from decimal, three of which are commonly used in computing: the **binary**, **octal**, and **hexadecimal** systems.

Number Systems

- The number of unique digits used by a number system is referred to as its **base** or **radix**.
- The decimal system, with its ten digits, is also called the **base-10 number system**.
- The binary system (**base-2 number system**) only has two digits: 0 and 1.

Number Systems

- The octal system (**base-8 number system**) uses eight digits: 0, 1, 2, 3, 4, 5, 6, and 7 for representing numbers.

- The hexadecimal system (**base-16 number system**) uses sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Number Systems

- All four number systems are **positional number systems**, where the placement of a digit corresponds to a power of the base.

- For example, there are certain positions in the decimal system commonly called the “one’s place”, the “ten’s place”, the “hundred’s place” and so on.

- Though the digit 5 appears twice in the decimal numeral 5345, the 5 in the thousand’s place has a different magnitude from the 5 in the one’s place.

Number Systems

- The one's place is actually the position that corresponds to 10^0.

- The ten's place is the position corresponding to 10^1, the hundred's place is 10^2, the thousand's place is 10^3, and so on.

- A similar concept applies to the other three numeral systems.

Number Systems

$$5 \quad 3 \quad 4 \quad 5$$

$$10^3 \quad 10^2 \quad 10^1 \quad 10^0$$

$$(5 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$$

$$(5 \times 1000) + (3 \times 100) + (4 \times 10) + (5 \times 1)$$

$$\mathbf{5000} + \mathbf{300} + \mathbf{40} + \mathbf{5} = 5345$$

Number Systems

- Non-integer values are expressed using a **radix point** (or *decimal point* in base-10).

- The decimal system still uses powers of 10 for positions appearing to the right of the decimal point (commonly called the “tenths”, “hundredths”, thousandths”, etc. places).

Number Systems

$$1 \quad 0 \quad 7 \quad . \quad 3 \quad 4$$

$$10^2 \quad 10^1 \quad 10^0 \qquad 10^{-1} \quad 10^{-2}$$

$$(1 \times 10^2) + (0 \times 10^1) + (7 \times 10^0) + (3 \times 10^{-1}) + (4 \times 10^{-2})$$

$$(1 \times 100) + (0 \times 10) + (7 \times 1) + (3 \times 0.1) + (4 \times 0.01)$$

$$100 + 0 + 7 + 0.3 + 0.04 = 107.34$$

Binary System

- The binary numeral system uses just two digits, 0 and 1, for representing both integer and non-integer numbers.

- Of course, 0 and 1 have equivalents in decimal, but what about equivalents of the numbers 2, 3, or 4?
  - How do we represent numbers larger than 1 in binary?

Binary System

- Let’s first look at this from a decimal perspective.

- There is no digit beyond 9 in decimal, but we know 10 (“ten”) comes after 9. However, 10 is not a digit- it’s *two* digits.

- After counting up from 10 through 99, what happens next? *Three* digits are now needed to represent the numbers 100 through 999, after which *four* digits would be needed.
  - The same principle applies in binary.

Binary System

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

Binary System

- It's important to note that 10 or 11 in binary is not called "ten" or "eleven"
  - Those terms only exist in decimal.

- 10 would be read as "*one zero*" and 11 would be read as "*one one*".

- A special notation is often used when working with numbers expressed in different numeral systems to avoid confusion as to whether 1000 means "*one thousand*" (decimal) or "*one zero zero zero*" (binary).

Binary System

$$1000_{10} \qquad 1000_{2}$$

- The two numbers above are written with their base/radix in subscript.

- The first number, with a radix of 10, is to be interpreted as the base-10 (decimal) number "*one thousand*".
- The radix of the second number identifies it to be the base-2 (binary) number "*one zero zero zero*", which is the equivalent of the number 8 in decimal.

Binary System

- The positions of digits in a binary numeral represent powers of two.

- The bit furthest to the right is the **least significant bit** or ***LSB*** as it has the lowest positional value.
- The left-most 1 bit is the **most significant bit** or ***MSB***.

- By multiplying each digit by its power of two (expressed in decimal), the binary number can be converted to its decimal equivalent.

Binary System

MSB **LSB**

$$\mathbf{1}\ 0\ 0\ \mathbf{1}\ \mathbf{1}\ 0\ 0\ 0$$

$$\mathbf{2^7}\quad 2^6\quad 2^5\quad \mathbf{2^4}\quad \mathbf{2^3}\quad 2^2\quad 2^1\quad 2^0$$

$$(\mathbf{1 \times 2^7}) + (0 \times 2^6) + (0 \times 2^5) + (\mathbf{1 \times 2^4}) + (\mathbf{1 \times 2^3}) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$$

$$(\mathbf{1 \times 128}) + (0 \times 64) + (0 \times 32) + (\mathbf{1 \times 16}) + (\mathbf{1 \times 8}) + (0 \times 4) + (0 \times 2) + (0 \times 1)$$

$$\mathbf{128} + 0 + 0 + \mathbf{16} + \mathbf{8} + 0 + 0 + 0 = \boxed{\mathbf{152}_{10}}$$

Binary System

- A decimal number can be converted to binary using repeated division by two, until zero is reached.
  - The result of each division will have a remainder of 1 or 0.

- The remainders are what make up the decimal number's binary equivalent, the first of which will be the LSB.
- The next examples demonstrates converting the decimal numbers 18 and 145 to binary.

Binary System

18 ÷ 2 = 9 r **0**

9 ÷ 2 = 4 r **1**

4 ÷ 2 = 2 r **0**

2 ÷ 2 = 1 r **0**

1 ÷ 2 = 0 r **1**

$\mathbf{18}_{10} = \mathbf{10010}_2$

Binary System

145 ÷ 2 = 72 r **1**

72 ÷ 2 = 36 r **0**

36 ÷ 2 = 18 r **0**

18 ÷ 2 = 9 r **0**

9 ÷ 2 = 4 r **1**

4 ÷ 2 = 2 r **0**

2 ÷ 2 = 1 r **0**

1 ÷ 2 = 0 r **1**

$\mathbf{145_{10} = 10010001_2}$

Binary Addition

- To demonstrate adding binary numbers, we'll start with a simple example.
  - Adding the binary numbers 01 and 10 (or 1_{10} and 2_{10}, respectively):

$$\begin{array}{r} 0\,1 \\ +\,1\,0 \\ \hline \end{array} \qquad \begin{array}{r} 0\;\;1 \\ +\,1\;\;0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0\;\;1 \\ +\,1\;\;0 \\ \hline 1\;\;1 \end{array}$$

  - Start by adding the LSB's first.
    - 1 and 0 are added together resulting in 1.
  - 0 and 1 are then added together, also resulting in 1.
  - Thus, the result of the addition is 11_2 (or 3_{10}).

Binary Addition

- This example demonstrates what happens when 01_2 (1_{10}) is added with 11_2 (3_{10}).

$$\begin{array}{r} 0\;1 \\ \underline{+\;1\;1} \end{array} \qquad\qquad \begin{array}{r} {\scriptstyle 1} \\ 0\;1 \\ \underline{+\;1\;1} \\ 0 \end{array}$$

  - Start by adding the LSB's first.
    - 1 and 1 are added together resulting in 10 (or 2_{10}).
    - The 0 is placed in the final result and the 1 is carried over.

Binary Addition

- This example demonstrates what happens when 01_2 (1_{10}) is added with 11_2 (3_{10}).

$$\begin{array}{r} 0\ 1 \\ \underline{+\ 1\ 1} \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 0\ 1 \\ \underline{+\ 1\ 1} \\ 0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\ \ {\scriptstyle 1} \\ 0\ 1 \\ \underline{+\quad 1\ 1} \\ 0\ 0 \end{array}$$

  - Next, 1, 0, and 1 are added together resulting, again, in 10.
  - The 0 is placed in the final result and the 1 is carried over

Binary Addition

- This example demonstrates what happens when 01_2 (1_{10}) is added with 11_2 (3_{10}).

$$\begin{array}{r} 0\,1 \\ +\,1\,1 \\ \hline \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 0\,1 \\ +\,1\,1 \\ \hline 0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\;{\scriptstyle 1} \\ 0\,1 \\ +\,1\,1 \\ \hline 0\,0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\;{\scriptstyle 1} \\ 0\,0\,1 \\ +\,0\,1\,1 \\ \hline 1\,0\,0 \end{array}$$

  - Finally, the carried 1 is added with 0 and 0 (not previously shown), which results in 1.
  - The calculated sum of these two binary numbers is 100_2 (or 4_{10}).

Binary Addition

- The result of the last example contained more bits that the numbers added together.
  - This is a situation that results in **arithmetic overflow**.

- Consider a particular computer system that limits us to using only four bits for storing numbers in memory.
  - What will happen when 1001_2 and 1010_2 are added?

Binary Addition

$$\begin{array}{r} {}^{1}\;\;1\;0\;0\;1 \\ +\;\;\;\;1\;0\;1\;0 \\ \hline 1\;\underline{0\;0\;1\;1} \end{array}$$

- The result of the addition is 10011_2 but this system is only allowing us four bits of space.
- Therefore, only four bits of this result (beginning with the LSB) will be stored.
  - There is not enough room for the extra bit to be included

Binary Subtraction

- To demonstrate subtracting binary numbers, we'll start with a simple example.
  - Subtracting the binary numbers 11 and 01 (or 3_{10} and 1_{10}, respectively):

$$\begin{array}{rr} & 1\ 1 \\ - & 0\ 1 \\ \hline & \end{array} \qquad \begin{array}{rr} & 1\ 1 \\ - & 0\ 1 \\ \hline & 0 \end{array} \qquad \begin{array}{rr} & 1\ 1 \\ - & 0\ 1 \\ \hline & 1\ 0 \end{array}$$

  - Start by subtracting the LSB's first.
    - 1 is subtracted from 1, resulting in 0.
  - 0 is subtracted from 1, resulting in 1.
  - Thus, the result of the subtraction is 10_2 (or 2_{10}).

Binary Subtraction

- This example demonstrates what happens when 101_2 (5_{10}) is subtracted from 110_2 (6_{10}).

$$
\begin{array}{r}
1\,1\,0 \\
-\,1\,0\,1 \\
\hline
\end{array}
\qquad\qquad
\begin{array}{rccc}
 & & 0 & 10 \\
 & 1 & \cancel{1} & \cancel{0} \\
- & 1 & 0 & 1 \\
\hline
 & & & 1
\end{array}
$$

- To subtract 1 from 0, we need to borrow from the next column.
  - 1 is subtracted from 10, resulting in 1

Binary Subtraction

- This example demonstrates what happens when 101_2 (5_{10}) is subtracted from 110_2 (6_{10}).

$$\begin{array}{r} 1\,1\,0 \\ -\,1\,0\,1 \\ \hline \end{array} \qquad \begin{array}{rccc} & & 0 & 10 \\ & 1 & \cancel{1} & \cancel{0} \\ - & 1 & 0 & 1 \\ \hline & & & 1 \end{array} \qquad \begin{array}{rccc} & & 0 & 10 \\ & 1 & \cancel{1} & \cancel{0} \\ - & 1 & 0 & 1 \\ \hline & & 0 & 1 \end{array} \qquad \begin{array}{rccc} & & 0 & 10 \\ & 1 & \cancel{1} & \cancel{0} \\ - & 1 & 0 & 1 \\ \hline & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array}$$

  - No borrows are needed for the remaining subtractions.
  - The calculated difference of these two binary numbers is 001_2 (or 1_{10}).

Octal System

- The octal numeral system uses eight digits, 0 through 7, for representing numbers.
- In octal, after 7 would come 10, 11, 12, 13 (*one zero*, *one one*, *one two*, *one three*) and so on.
- After 17 would come 20, then 21, then 22 (*two zero*, *two one*, *two two*) and so on.

Octal System

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- The octal system might not seem particularly useful at first but take another look at the table above and the relationship between the individual octal digits and their binary equivalents
  - Each octal digit can be represented with just **three bits**.

Octal System

$$100101101001_2$$

$$\underbrace{\underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{101}_{5}\ \underbrace{001}_{1}}_{4551_8}$$

- This relationship allows for an easy conversion from a long string of bits to a more manageable number.
  - Plus, it's much harder to mis-type (and much easier to read) 4551 than 100101101001.

Hexadecimal System

- The hexadecimal number system uses the sixteen digits 0 through 9 and A through F for representing numbers.

- It might be a little strange to see letters being used for representing numbers but that's because we are so used to using decimal, which has no digits beyond 9.

Hexadecimal System

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Hexadecimal	Binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

- After F, the hexadecimal system continues with two digits- 10 through 1F (*one zero* through *one F*), then 20 through 2F (*two zero* through *two F*), and so on.
- Upon reaching FF, the system will continue by using three digits, beginning at 100 (*one zero zero*).

Hexadecimal System

$$110101101011_2$$

$$1101\ 0110\ 1011$$

$$D\quad 6\quad B$$

$$D6B_{16}$$

- You may notice a similarity to the benefit seen when using the octal system.
- In this case, each hexadecimal digit can be represented using **four bits**

Two's Complement Representation

- Just about every computer architecture today uses two's complement representation to represent signed integers.

- **Unsigned integers** are non-negative integers
  - $0 \leq i \leq +\infty$

- **Signed integers** are integers that can be positive or negative
  - $-\infty \leq i \leq +\infty$

Two's Complement Representation

- The range of possible negative and non-negative numbers that can be represented using two's complement is:

$$-2^{b-1} \leq n \leq 2^{b-1} - 1$$

  - where *b* is the number of bits used.
- Using four-bit numbers, two's complement can be used to represent the integers -8 (-2^3) through 7 ($2^3 - 1$).
- With five bits, two's complement can represent the integers -16 (-2^4) through 15 ($2^4 - 1$).

Two’s Complement Representation

- The process to convert a binary number to two’s complement notation is fairly straight-forward.
- First, “flip” all of the bits in the binary number
  - All 0’s become 1’s and 1’s become 0’s.
- Then, add 1.
  - The *one’s complement*

Two’s Complement Representation

$$\begin{array}{r} 1\ 0\ 1 \\ \downarrow\ \downarrow\ \downarrow \\ 0\ 1\ 0 \\ \underline{+\quad\ 1} \\ 0\ 1\ 1 \end{array}$$

Binary	Two’s Complement Representation
000	000
001	111
010	110
011	101
100	100
101	**011**
110	010
111	001

Two’s Complement Representation

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	0	000	0
001	1	111	-1
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	011	3
110	6	010	2
111	7	001	1

Two’s Complement Representation

- All zeros is always 0

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	**0**	**000**	**0**
001	1	111	-1
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	011	3
110	6	010	2
111	7	001	1

Two's Complement Representation

- The largest value is always 0 followed by all 1's

Binary	Unsigned Integer	Two's Complement	Signed Integer
000	0	000	0
001	1	111	-1
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	**011**	**3**
110	6	010	2
111	7	001	1

Two's Complement Representation

- The smallest value is always 1 followed by all 0's

Binary	Unsigned Integer	Two's Complement	Signed Integer
000	0	000	0
001	1	111	-1
010	2	110	-2
011	3	101	-3
100	4	**100**	**-4**
101	5	011	3
110	6	010	2
111	7	001	1

Two’s Complement Representation

- Every negative number begins with a 1
  - It is the negative version of it’s unsigned equivalent

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	0	000	0
001	1	**111**	**-1**
010	2	**110**	**-2**
011	3	**101**	**-3**
100	4	**100**	**-4**
101	5	011	3
110	6	010	2
111	7	001	1

Two's Complement Representation

- Every positive number (and zero) begins with a 0

Binary	Unsigned Integer	Two's Complement	Signed Integer
000	0	**000**	**0**
001	1	111	-1
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	**011**	**3**
110	6	**010**	**2**
111	7	**001**	**1**

Two’s Complement Representation

- All ones always represents -1

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	0	000	0
001	1	**111**	**-1**
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	011	3
110	6	010	2
111	7	001	1

Two’s Complement Representation

- To test how the two’s complement representations will work using addition, we’ll add the two’s complement representations of -2 and -2 together.
  - The result should be the two’s complement representation of -4.

$$
\begin{array}{rcccc}
 & 1 & 1 & & \\
 & & 1 & 1 & 0 \\
+ & & 1 & 1 & 0 \\
\hline
 & 1 & \underline{1} & \underline{0} & \underline{0}
\end{array}
$$

  - Since we are working with 3-bit numbers, only the first three bits of the result are used.

Two’s Complement Representation

$$\begin{array}{rccc} & 1 & 1 & \\ & 1 & 1 & 0 \\ + & 1 & 1 & 0 \\ \hline 1 & \underline{\mathbf{1}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} \end{array}$$

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	0	000	0
001	1	111	-1
010	2	**110**	**-2**
011	3	101	-3
100	4	**100**	**-4**
101	5	011	3
110	6	010	2
111	7	001	1

Two’s Complement Representation

- Adding the two’s complement representations of -1 and 3 together.
  - The result should be the two’s complement representation of 2.

$$
\begin{array}{ccccc}
 & 1 & 1 & 1 & \\
 & & 1 & 1 & 1 \\
+ & & 0 & 1 & 1 \\
\hline
 & 1 & \underline{\mathbf{0}} & \underline{\mathbf{1}} & \underline{\mathbf{0}}
\end{array}
$$

Binary	Unsigned Integer	Two’s Complement	Signed Integer
000	0	000	0
001	1	**111**	**-1**
010	2	110	-2
011	3	101	-3
100	4	100	-4
101	5	**011**	**3**
110	6	**010**	**2**
111	7	001	1

Units of Information

- Units of information measure the capacities of storage devices and communication channels.

- The smallest unit of information is the **bit**.
  - Can be either 0 or 1
  - Abbreviated as **b**

- A group of eight bits is referred to as a **byte**.
  - Sometimes also called an *octet*
  - Abbreviated as **B**

Units of Information

- Multiples of bits and bytes can be expressed using SI prefixes (which use powers of ten, shown below) or IEC prefixes (which use powers of two, shown on the next slide)

Prefix	Abbreviation	Value
Kilo	k or K	10^3
Mega	M	10^6
Giga	G	10^9
Tera	T	10^{12}
Peta	P	10^{15}
Exa	E	10^{18}
Zetta	Z	10^{21}
Yotta	Y	10^{24}

Units of Information

- IEC binary prefixes

Prefix	Abbreviation	Value
Kibi	Ki	2^{10}
Mebi	Mi	2^{20}
Gibi	Gi	2^{30}
Tebi	Ti	2^{40}
Pebi	Pi	2^{50}
Exbi	Ei	2^{60}
Zebi	Zi	2^{70}
Yobi	Yi	2^{80}

Units of Information

- Why the two systems?

- The SI prefixes are the standard for the metric system.

- Most memory and storage device capacities are powers of two.
  - SI prefixes were repurposed for this.
  - For example, M was *understood* to mean 2^{20} instead of 10^6
    - A difference of ~4.9%

- To avoid ambiguity, we have the two systems.

Powers of Two

- The following table shows the powers of two up to 2^{10}

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Commit these to memory

Powers of Two

- Useful properties of exponents:

$$x^y \times x^z = x^{y+z}$$

$$x^y \div x^z = x^{y-z}$$

$$x^{y^z} = x^{y \times z}$$

Powers of Two

- $2^{10} = 1Ki$
- $2^{11} = 2^1 \times 2^{10} = 2 \times 2^{10} = 2Ki$
- $2^{12} = 2^2 \times 2^{10} = 4 \times 2^{10} = 4Ki$
- $2^{13} = 2^3 \times 2^{10} = 8 \times 2^{10} = 8Ki$
- $2^{19} = 2^9 \times 2^{10} = 512 \times 2^{10} = 512Ki$

Powers of Two

- $2^{20} = 1Mi$
- $2^{21} = 2^1 \times 2^{20} = 2 \times 2^{20} = 2Mi$
- $2^{22} = 2^2 \times 2^{20} = 4 \times 2^{20} = 4Mi$
- $2^{23} = 2^3 \times 2^{20} = 8 \times 2^{20} = 8Mi$
- $2^{29} = 2^9 \times 2^{20} = 512 \times 2^{20} = 512Mi$

Powers of Two

- $2^{30} = 1Gi$
- $2^{31} = 2^1 \times 2^{30} = 2 \times 2^{30} = 2Gi$
- $2^{32} = 2^2 \times 2^{30} = 4 \times 2^{30} = 4Gi$
- $2^{33} = 2^3 \times 2^{30} = 8 \times 2^{30} = 8Gi$
- $2^{39} = 2^9 \times 2^{30} = 512 \times 2^{30} = 512Gi$

Powers of Two

- $2^{40} = 1Ti$
- $2^{41} = 2^1 \times 2^{40} = 2 \times 2^{40} = 2Ti$
- $2^{42} = 2^2 \times 2^{40} = 4 \times 2^{40} = 4Ti$
- $2^{43} = 2^3 \times 2^{40} = 8 \times 2^{40} = 8Ti$
- $2^{49} = 2^9 \times 2^{40} = 512 \times 2^{40} = 512Ti$

Powers of Two – Quick Conversion

$$\mathbf{2^{11}} = 2^1 \times 2^{10} = 2 \times 2^{10} = \boldsymbol{2Ki}$$

- The first digit of the exponent is the magnitude
  - 1 = Ki, 2 = Mi, 3 = Gi, 4 = Ti, etc.

$$2^{\mathbf{1}1} = 2^1 \times \mathbf{2^{10}} = 2 \times 2^{10} = 2\boldsymbol{Ki}$$

- The second digit of the exponent is two raised to that power

$$2^{1\mathbf{1}} = \mathbf{2^1} \times 2^{10} = 2 \times 2^{10} = \mathbf{2}Ki$$

Powers of Two – Quick Conversion

$$\mathbf{2^{43}} = 2^3 \times 2^{40} = 8 \times 2^{40} = \boldsymbol{8Ti}$$

- The first digit is the magnitude
  - 1 = Ki, 2 = Mi, 3 = Gi, 4 = Ti, etc.

$$2^{\mathbf{4}3} = 2^3 \times \mathbf{2^{40}} = 8 \times 2^{40} = 8\boldsymbol{Ti}$$

- The second digit is two raised to that power

$$2^{\mathbf{43}} = \mathbf{2^3} \times 2^{40} = 8 \times 2^{40} = \mathbf{8}Ti$$

Powers of Two

- $\boldsymbol{1Ki} \times \boldsymbol{4Ki} = 2^{10} \times 2^{12} = 2^{10+12} = 2^{22} = 2^{2} \times 2^{20} = \boldsymbol{4Mi}$

- $\boldsymbol{64Ki} \times \boldsymbol{32Ki} = 2^{16} \times 2^{15} = 2^{16+15} = 2^{31} = 2^{1} \times 2^{30} = \boldsymbol{2Gi}$

- $\boldsymbol{8Gi} \div \boldsymbol{32Ki} = 2^{33} \div 2^{15} = 2^{33-15} = 2^{18} = 2^{8} \times 2^{10} = \boldsymbol{256Ki}$

- $\boldsymbol{2Ki^4} = {2^{11}}^{4} = 2^{11\times 4} = 2^{44} = 2^{4} \times 2^{40} = \boldsymbol{16Ti}$

CPU Performance

- Almost all processors have a clock that determines when instructions are executed by it.
- These time intervals are measured in **clock cycles**
  - Also called clock ticks, clocks, or cycles
- Sometimes, the term **clock period** refers to
  - Complete clock cycle (usually measured in picoseconds; ps; 10^{-12} seconds)
  - Clock rate (the inverse of the clock cycle)

CPU Performance

- $Clock\ Cycle\ = \frac{second}{cycle}$

- $Clock\ Rate\ = \frac{cycle}{second}$
  - The unit for $\frac{cycle}{second}$ is Hertz (abbreviated Hz)

CPU Performance

- Example: What is the clock rate of a processor with a clock cycle of 250ps (picoseconds)?
  - $250ps = 250 * 10^{-12}s = 0.00000000025s$

- $Clock\ Cycle = \frac{250 \times 10^{-12} seconds}{cycle} = 0.00000000025 \frac{seconds}{cycle}$

- $Clock\ Rate = \frac{1\ cycle}{250 \times 10^{-12} seconds} = 4{,}000{,}000{,}000 \frac{cycles}{second} = 4.0\ \mathrm{GHz}$

CPU Performance

- Example: What is the clock cycle of a processor with a clock rate of 3.8GHz?

- $Clock\ Rate\ =\ 3{,}800{,}000{,}000\ \dfrac{cycles}{second}$

- $Clock\ Cycle\ =\ \dfrac{1\ second}{3{,}800{,}000{,}000\ cycles}\ =\ 2.63\ \times\ 10^{-10}\ \dfrac{seconds}{cycle}\ =$

$263\ \times\ 10^{-12}\dfrac{seconds}{cycle}$ = $263\dfrac{picoseconds}{cycle}$ or simply $263\ ps$

CPU Performance

- CPU performance can be measured in terms of **execution time**- How long it takes for a processor to run a program.

- CPU Execution time (seconds) = $clock\ cycles\ \times \boldsymbol{clock\ cycle\ time}$

- CPU Execution time (seconds) = $clock\ cycles\ \times \frac{\boldsymbol{seconds}}{\boldsymbol{cycle}}$

CPU Performance

- The clock rate could also be used, since it is the inverse of the clock cycle time.

- CPU Execution time (seconds) = $\dfrac{Clock\ cycles}{\boldsymbol{Clock\ rate}}$

- CPU Execution time (seconds) = $\dfrac{Clock\ cycles}{\dfrac{\boldsymbol{cycles}}{\boldsymbol{second}}}$

CPU Performance

- A program requires 10 x 10^9 cycles to complete. The program is executed by a 2GHz processor. How long will it take the processor to execute the program?

- CPU Execution time = $10 \ \times 10^9 \ cycles \times \frac{1}{2 * 10^9} \frac{s}{cycles}$ = $\frac{10 \times 10^9}{2 \times 10^9} \ s$ = 5 s

- CPU Execution time = $\dfrac{10 \times 10^9 \ cycles}{\dfrac{2 \times 10^9 cycles}{s}}$ = 5 s

CPU Performance

- It takes a 2GHz processor 15 seconds to execute a program. How many clock cycles did the program need in order to complete?
- 15 s = $\boldsymbol{x}\ cycles \times \frac{1}{2 \times 10^9} \frac{s}{cycles}$

$$\boldsymbol{x}\ cycles = \frac{15\ s}{\frac{1}{2\ \times 10^9} \frac{s}{cycles}} = 3\ \times\ 10^{10}\ cycles$$

CPU Performance

- It takes a 2GHz processor 15 seconds to execute a program. How many clock cycles did the program need in order to complete?
- 15 s = $\dfrac{\boldsymbol{x}\ cycles}{\dfrac{2 \times 10^9\ cycles}{s}}$ =

$$\boldsymbol{x}\ cycles = 15\ \cancel{s}\ \times\ \frac{2 \times 10^9\ cycles}{\cancel{s}} = 3\ \times\ 10^{10}\ cycles$$

CPU Performance

- CPU performance can also be measured in terms of *instruction performance*- or, the average number of clock cycles required for each instruction.
    - **CPI: C**ycles **p**er **i**nstruction

- Clock Cycles = $Number\ of\ instructions\ \times\ CPI$

CPU Performance

- A program has 50 instructions and is executed on a processor with a CPI of 1.5. How many clock cycles are required by the program?

- Clock Cycles = $50\ instructions\ \times\ 1.5\ \frac{cycles}{instruction}\ =\ 75\ clock\ cycles$

CPU Performance

- A program has 100 instructions and is executed on a 3GHz processor with a CPI of 1.5. How long will it take the program to execute?

- Clock Cycles = $100\ instructions\ \times\ 1.5\ \frac{cycles}{instruction}\ =\ 150\ clock\ cycles$

- Then use an execution time formula:

$$\frac{150\ cycles}{\frac{3*10^{9}\ cycles}{s}} =\ .00000005\ s\ =\ 5\ \times\ 10^{8}\ s$$

CPU Performance

- The formula:

$$\boldsymbol{Clock\ Cycles\ = Instructions\ \times\ CPI}$$

- Can be substituted in the execution time formulas for a more general formula:

$$CPU\ Execution\ time\ =\ \boldsymbol{Clock\ cycles}\ \times\ clock\ cycle\ time$$

$$\boxed{CPU\ Execution\ time\ = \boldsymbol{Instructions}\ \times\ \boldsymbol{CPI}\ \times\ clock\ cycle\ time}$$

$$CPU\ Execution\ time\ = \frac{\boldsymbol{Clock\ cycles}}{Clock\ rate}$$

$$\boxed{CPU\ Execution\ time\ = \frac{\textbf{Instructions}\ \times\ \textbf{CPI}}{Clock\ rate}}$$

CPU Performance

- Using either, we can derive the number of instructions:

$$CPU\ Execution\ time\ =\ Instructions\ \times\ CPI\ \times\ clock\ cycle\ time$$

$$\boldsymbol{Instructions}\ =\ \frac{CPU\ Execution\ time}{CPI\ \times\ clock\ cycle\ time}$$

$$CPU\ Execution\ time\ =\ \frac{\text{Instructions}\ \times\ \text{CPI}}{Clock\ rate}$$

$$\boldsymbol{Instructions}\ =\ \frac{CPU\ Execution\ time\ \times\ clock\ cycle}{CPI}$$

CPU Performance

- Instructions Per Second (IPS)

$$IPS = \frac{Clock\ Rate}{CPI}$$