

2026

CS 332

Subject o

NJIT

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department



The yellowish/brownish notes are copyright the textbook author.

The bluish notes are copyright the instructor and are add-ons and provide more explanation or additional details



Subject 0

Math + CS Review

(draws from Document 3)



Subject 0

Module 1 : Math Review

(draws from Document 3)



1. Set vs Sequence

Set: $\{10, 20, 30\}$

order does not matter ; no repetitions

Sequence: $\langle 20, 10, 10 \rangle$

order matters;

repetitions ok

Example

$A = \{10, 20, 30\}$ is the same as $B = \{30, 20, 10\}$ i.e. $A = B$

$C = \langle 20, 10, 10 \rangle$ is not the same as $D = \langle 20, 10, 20 \rangle$ i.e. $C \neq D$

What is the first element of A ? No notion of order in a set!

What is the second element of C? 10

2. Exponentials (Powers)



$\text{pow}(x,y)$, x^y , $x^{}y$: multiply y instances of x together**

Properties of powers (all same base and in this course base is 2!)

Multiplication: $2^n * 2^m = 2^{(n+m)}$

Division: $2^n / 2^m = 2^{(n - m)}$

Exponentiation: $(2^n)^m = 2^{(n*m)}$

Example:

$\text{pow}(2,0) = 2^{**}0 = 2^0 = 1,$ $2^{10} = 2*2*2*2*2*2*2*2*2*2 = 1024 \sim 10^3$

$\text{pow}(2,1) = 2^{**}1 = 2^1 = 2$ $2^{20} = (2^{10})*(2^{10}) = 1048576 \sim 10^6$

$\text{pow}(2,8) = 2^{**}8 = 256$ $2^{30} = (2^{20})*(2^{10}) \sim 10^9$

Exercise: Memorize powers of 2 from power 1 through 16 i.e. $2^{}1$ to $2^{**}16$**



Floor and Ceiling:

floor(x) : largest integer less than or equal to x

ceiling(x) is smallest integer greater than or equal to x

Example:

floor(3.5) is 3

ceil(3.5) is 4

floor(-3.5) is -4

ceil(-3.5) is -3

floor(3)=ceil(3)=3.



$\lg(n)$ is the number we need to raise (base) two to get n

i.e. $2^{\lg(n)} = n$

Properties of logarithms.

Multiplication: $\lg(a * b) = \lg(a) + \lg(b)$

Division: $\lg(a/b) = \lg(a) - \lg(b)$

Exponentiation: $\lg(a^{**}k) = k * \lg(a)$

Example: $\lg(8) = \lg(2^{**}3) = 3 * \lg(2) = 3 * 1 = 3$ also: $2^{**}3 = 8$ implies $\lg(8) = 3$

$\lg(1024) = \lg(2^{**}10) = 10 * \lg(2) = ?$

Example.

$\lg(n)$ is $\lg(n)$ since $2^{**}\lg(n)$ gives n by definition (default)

$\lg(2)$ is 1 since $2^{**}1$ gives 2

$\lg(8)$ is 3 since $2^{**}3$ gives 8

$\lg(1)$ is 0 since $2^{**}0$ gives 1

How much is $\lg(1024)$?



5. Byte and bit

ALERT: there is no space between the magnitude and the unit!

1bit

bit is an acronym for **binary digit**: a bit has two values 0 or 1 (low/high, down/up)

Description:

1bit

100bit

1B

a byte is denoted by a capital B and is the minimal amount of info stored in a computer or that one can '**bite**' from the memory of a computer; we read from memory or write into memory 1B or multiples, never 5bit or bit multiple!

Description:

10B

100B

Nowadays, a byte is a sequence of 8 bit, older computers used to 'bite' 6bit, 12bit, 16bit at a time. Because of this do not get surprised if you hear talk of an **octet**! (prefix oct as also in octal implies 8)

1b is nonsense (trade publications use it however...)

Do not disregard this discussion. It might **bite** you later in this course (pun intended)

6. Bit in a Byte (ordering)



A byte nowadays is a sequence of 8 bit but this was not so always

(so the term octet is in use in networking)

A byte: **10110110**

Leftmost bit is 1 might or might not be 'most significant' bit

Rightmost bit is a 0 'least 'significant' bit

4th bit from left is 1 4th bit from right is a 0

Read byte in Little Endian: **0,1,1,0,1,1,0,1** (right to left)

Read byte in Big Endian: **1,0,1,1,0,1,1,0** (left to right)

Suggestion: Try to avoid using the term 'significant bit'



SI system of units (aka Metric System)

1K is one degree Kelvin !

1k is 1000 (lower case k is kilo)

1KiB is $1024B = 2^{10}B$ (kibibyte)

1M is 1000000 (capital M is Mega)

1MiB is $1048576B = 2^{20}B$ (mebibyte)

1G is 1000000000 (capital G is Giga)

1GiB is $2^{30}B$ (gibibyte)

1T is 10^{12} (capital T is Tera)

1TiB is $2^{40}B$ (tebibyte)

Never Use in a Homework or an Exam: 1KB , 1kB , 1K

1MB of RAM might or might not mean $1,048,576 = 2^{20}B$

1GB or RAM might or might not mean $2^{30}B$

1GB of disk space according to Seagate or Western Digital means $1,000,000,000B$

OTHER AGGREGATIONS

1B : 8 bit , 1 octet : 8bit, 1 half-word: 2B : 16 bit

1 word: 4B : 32 bit, 1double word: 8B:64bit

8. Bite a byte lunacy!



- ¹ One megabyte (MB) is equal to one million bytes, one gigabyte (GB) is equal to 1,000MB (one billion bytes), and one terabyte (TB) is equal to 1,000GB (one trillion bytes) when referring to storage capacity. Accessible capacity will vary from the stated capacity due to formatting, system software, and other factors.
- ² MTBF and AFR specifications are based on a sample population and are estimated by statistical measurements and acceleration algorithms under typical operating conditions for this drive model. MTBF and AFR ratings do not predict an individual drive's reliability and do not constitute a warranty.

- ³ Advanced Format drive: 4K (4096-byte) physical sectors
- ⁴ Portion of buffer capacity used for drive firmware
- ⁵ MiB/s is 2^{20} bytes, MB/s is 10^6 bytes
- ⁶ Excludes command overhead
- ⁷ SATA models: 8KB Queue Depth = 1 @ 40 IOPS, SAS models: 4KB Queue Depth = 4 @ Max IOPS
- ⁸ Idle specification is based on use of Idle_A

Western Digital.

5601 Great Oaks Parkway
 San Jose, CA 95119, USA
US (Toll-Free): 800.801.4618
International: 408.717.6000
www.wdc.com/dc-hc530
www.wdc.com/dc-support



14TB | 7200 RPM | SATA 6Gb/s and SAS 12Gb/s

Funny Note:

In (British) English, a billion is a million millions...

In (American) English, a billion is a thousand millions ...

Pay attention to locale to

disambiguate!!!!



Time Period (T): Duration of a periodic event (cycle) in (usually) seconds

Frequency (f): Number of times a periodic event (cycle) occurs in one second.

Relationship between T(in s) and f of a periodic event: $T * f = 1$

Unit of time: second and denoted by **s**. $1s = 1,000ms = 1,000,000\mu s = 1,000,000,000ns$

Unit of frequency: cycle/s denoted by **Hz**. $1GHz = 1,000MHz = 1,000,000kHz = 1,000,000,000Hz$
(Hz is read Hertz)

Example. Periodic event occurs 5 times in a second $\Rightarrow f = 5\text{cycle/s} = 5\text{Hz}$

$T * f = 1 \Rightarrow T = 1/f = 1/5\text{cycle/s} = 0.2s/\text{cycle} \Rightarrow T = 0.2s$ or $0.2s/\text{cycle}$

Thus, one event takes 0.2s to complete

Speed of Light: 300,000km/s. **Light travels 1ft in roughly 1ns**

CPU: Periodic event is (elementary) instruction's execution time

A clock is attached to the CPU and ticks periodically. Period (distance of two consecutive ticks) is T .

T is large enough for an elementary instruction to complete its execution

Time of (elementary) instruction's execution $\leq T$

The frequency of the CPU's clock is $f = 1/T$ (#of elem. instructions /s)

$f = 1\text{GHz}$ for a CPU implies $1,000,000,000$ instr/s

implies $T = 1/f = 10^{*(-9)} \text{ s/instr} = 1\text{ns/instr}$

$f = 2\text{GHz}$ for a CPU implies $T = 0.5\text{ns/instr}$

Note: In modern CPUs several instructions take more than one cycle each. For example a

`MOV Y, X, 1,000,000` : Move 1,000,000B from address X to address Y will take more time than one cycle!

(implication: block of bytes from X .. X+999,999 moves to Y .. Y+999,999 respectively, a total of 1,000,000B

`MOV Y,X, 1,000` : A Move that only moves 1,000B will take less time than the previous MOVE operation!



Variable:	Value can change	
Constant:	Value never changes	eg 3.14
Data type (d.t.):	Set of values a variable takes	
Primitive data types (built-in):	byte, boolean, char, int, float	
Composite data type:	array, struct, objects (e.g. class)	
primitive ('built-in') d.t. int, char, float , double,	vs	composite (aggregate) d.t. BigInteger, int[10], struct, class
weakly typed languages eg interpreted Python, Perl	vs	strongly typed languages eg compiled C, C++, Java

12. Parameter vs Argument



$f(x) = x * x;$ x is the unknown, the indeterminate,

In computing x is called the **parameter** of f

$f(20) ;$ 20 is not a parameter, it is the **argument** of f

$y=20; f(y);$ y is not a parameter, its value is the **argument** of f
but sometimes we can also say that y is the argument of f .



left operand
23 - 15
 right operand
 o
p
e
r
a
t
o
r

Operator: - (binary operator)

Operation: Subtraction

Left operand: 23

Right operand: 15

- 15
 right operand
 o
p
e
r
a
t
o
r

Operator: - (Unary operator)

Operation: Negation

Right operand: 15

OPERATOR OVERLOAD: - for negation and subtraction!

Alternative: sub(23,15) (prefix notation)

23-15 (infix notation)

23, 15, - (postfix notation)

-, 23,15 (prefix notation)

Operator Overload: Symbol - can be unary (negation) or binary (subtraction)

Moreover x++ vs ++x

x-- vs --x

Alternative: minus(15) (prefix notation)



14. Base vs Radix

Base of an integer: It is its radix!

Radix-10 (aka base-10 aka **denary** aka decimal).

Use digits: **{0,1,2,3,4,5,6,7,8,9}**

i.e. **10 digits**

Radix-2 (aka base-2, aka **binary**)

Use digits: **{0,1}**

i.e. **2 digits**

Radix-8 (aka base-8, aka **octal**)

Use digits: **{0,1,2,3,4,5,6,7}**

i.e. **8 digits**

Radix-16 (aka base-16, aka **hexadecimal**)

Use digits: **{0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f}**

i.e. **16 'digits'**

or : **{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}**

Caution:

1101	can be read one thousand one hundred and one (implies denary)	Reading -1
	can be read one one zero one radix-2 (implies binary)	Reading-2



Number of bit to minimally represent D is $b(D)$ where

$$b(D) = \text{floor}(\lg(D)) + 1 \quad \text{or} \quad b(D) = \text{ceiling}(\lg(D+1))$$

Example.

$$8 \quad \text{floor}(\lg(8)) + 1 = \text{floor}(3) + 1 = 4 \quad \text{ceil}(\lg(8+1)) = \text{ceil}(3.x) = 4 \quad b(8) = 4 \quad \text{bin}(8, 4) = 1000 \quad (4\text{bit})$$

$$7 \quad \text{floor}(\lg(7)) + 1 = \text{floor}(2.x) + 1 = 3 \quad \text{ceil}(\lg(7+1)) = \text{ceil}(\lg(8)) = 3 \quad b(7) = 3 \quad \text{bin}(7, 3) = 111 \quad (3\text{bit})$$

Number of same # bit to represent all integers

o .. D is the

Number of bits to represent largest i.e.

$b(D)$

It is also if D is a power of D the floor or ceiling can be dropped!

Example:

$D=7$; all integer numbers o.. D can have the same bit length 3

$D=7$

000,001,010,011, 100,101,110,111 in three-bit writing

o.. 7



Integer is a shortcut for integer number

Non-negative or **Unsigned** integer means positive integer or zero

Natural number refers sometimes to a positive, sometimes to a non-negative integer

This course: all integers are unsigned integers unless explicitly told otherwise.

Example 1. An index of an array takes values from 0 to a maximum value say 15

The number of indexes (indices) in that range is : 16.

The 16 indexes / indexes 0..15 can be represented with $\lg(16)=4\text{bit}$. (0000 .. 1111)

On the previous page we derived that these integers can be presented in $b(15)=4\text{bit}$ as well!!

Example 2. An index of an array will use 12bit.

How many indexes can we represent?

$2^{**12}=4096\text{indexes}$

What is their range?

0 through 4095

Using 12 bit for each one of them

or 000000000000 .. 111111111111



17. Notation overload

a :	the value of (unsigned) integer number that we 'think about' (radix-10)	
$ a $:	magnitude, a without a sign; for non-negative integers $ a =a$	
$\text{rdx}(a, \text{base}, n)$:	the digit-sequence of a in a given base using n digits; leading os possible	
$\text{rdx}(a, \text{base}, o)$:	the digit-sequence of a in a give base without any leading zeroes	
$\text{rdx}(a, 2, n)$	is $\text{bin}(a, n)$	$b(a, 2) = b(2) = \# \text{bit in representation of } a \text{ (no leading os)}$
$\text{rdx}(a, 8, n)$	is $\text{oct}(a, n)$	$b(a, 8) = \# \text{digit in octal rep. of } a$
$\text{rdx}(a, 16, n)$	is $\text{hex}(a, n)$	$b(a, 16) = \# \text{digit in hex rep. of } a$
$\text{rdx}(a, 10, n)$	is $\text{den}(a, n)^*$	$b(a, 10) = \# \text{digit in hex rep. of } a$
$l(a)$:	number of digit in string-view of a ; $l(\text{bin}(a, n))$ is n	

*den is for denary (i.e. ten-based). Ten from Den.



18. Tables

Denary	Hexadecimal		Binary (4bit)	Octal	Binary (3bit)
0	0		0000	0	000
1	1		0001	1	001
2	2		0010	2	010
3	3		0011	3	011
4	4		0100	4	100
5	5		0101	5	101
6	6		0110	6	110
7	7		0111	7	111
8	8		1000		
9	9		1001		
10	A	a	1010		
11	B	b	1011		
12	C	c	1100		
13	D	d	1101		
14	E	e	1110		
15	F	f	1111		

19. Default Notation



1234: Denary unless otherwise explicitly told

0b1101: Binary

0o1234: Octal

01234: Treacherous octal (leading zero) **DO NOT USE IT!**

0x123a: Hexadecimal

0X123A: Hexadecimal



Convert integer a written in radix-10 into radix-10 : Obvious!

1. Obtain a i.e. $a = a_{n-1} \dots a_1 a_0$ $l(a)=n$

2. Generate powers of 10 starting with $10^{**0} \dots 10^{**(n-1)}$ $10^{** (n-1)}$ $\dots$ 10^{**1}
 10^{**0}

corresponding to $a_0 \dots a_{n-1}$ a_{n-1} $\dots$ a_1 a_0

3. Multiply digits with corresponding powers, and

4. Add partial results $|a| = a = a_{n-1} * 10^{**(n-1)} + \dots a_1 * 10^{**1} + a_0 * 10^{**0}$

5. Output $|a|=a$

Example: Convert $a=5678$ written in radix-10 into radix-10 :Obvious!

1. Obtain a i.e. $a = 5\ 6\ 7\ 8$ $|a|=a=5678$, $l(a)=b(a,10)=n=4$

2. Generate powers of 10 starting with $10^{**0} \dots 10^{**3}$ 10^{**3} 10^{**2} 10^{**1} 10^{**0}
corresponding to $a_0 \dots a_3$ 5 6 7 8

3. Multiply digits with corresponding powers, and

4. Add partial results $|a| = a = 5 * 10^{**3} + 6 * 10^{**2} + 7 * 10^{**1} + 8 * 10^{**0}$

5. Output $|a|=a$ $|a|=a=5000+600+70+8=5678$



Convert a written in radix-2 into radix-10

1. Obtain **a** i.e. **a = a_{n-1} .. a₁ a₀** $l(a)=b(a,2)=n$

2. Generate powers of 2 starting with $2^{**0} .. 2^{**}(n-1)$

$2^{**}(n-1) \quad .. \quad 2^{**1} \quad 2^{**0}$

corresponding to $a_0 .. a_{n-1}$

$a_{n-1} \quad .. \quad a_1 \quad a_0$

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a| = a = a_{n-1} * 2^{**}(n-1) + ... a_1 * 2^{**1} + a_0 * 2^{**0}$$

5. Output $|a|=a$

Example : Convert a=1101 written in radix-2 into radix-10

1. Obtain **a** i.e. **a = 1101** $rdx(a,2,4)=bin(a,4)=1101 \quad l(a)=b(a,4)=n=4$

2. Generate powers of 10 starting with $2^{**0} .. 2^{**3}$

$2^{**3} \quad 2^{**2} \quad 2^{**1} \quad 2^{**0}$

corresponding to $a_0 .. a_{n-1}$

$1 \quad 1 \quad 0 \quad 1$

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a| = a = 1 * 2^{**3} + 1 * 2^{**2} + 0 * 2^{**1} + 1 * 2^{**0}$$

5. Output $|a|=a$

$$|a|=a= 8+ 4 +1 = 13$$



Convert a written in radix-8 into radix-10

1. Obtain **a** i.e. **a = a_{n-1} .. a₁ a₀** $l(a)=b(a,8)=n$

2. Generate powers of 8 starting with $8^{**0} .. 8^{**}(n-1)$

$8^{**}(n-1)$.. 8^{**1} 8^{**0}

corresponding to $a_0 .. a_{n-1}$

a_{n-1} .. a_1 a_0

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a|=a = a_{n-1} * 8^{**}(n-1) + ... a_1 * 8^{**1} + a_0 * 8^{**0}$$

5. Output $|a|=a$

Example: Convert a=1101 written in radix-8 into radix-10

1. Obtain **a** i.e. **a = 1101** $rdx(a,8,4)=oct(a,4)=1101$

$$l(a)=b(a,8)=n=4$$

2. Generate powers of 10 starting with $8^{**0} .. 8^{**3}$

8^{**3} 8^{**2} 8^{**1} 8^{**0}

corresponding to $a_0 .. a_{n-1}$

1 1 0 1

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a|=a = 1 * 8^{**3} + 1 * 8^{**2} + 0 * 8^{**1} + 1 * 8^{**0}$$

5. Output $|a|=a$

$$|a|=a = 512+64+1=577$$



Convert a written in radix-16 into radix-10

1. Obtain **a** i.e. **a = a_{n-1} .. a₁ a₀** $l(a)=b(a,16)=n$

2. Generate powers of 16 starting with $16^{**0} .. 16^{**}(n-1)$

$16^{**}(n-1) \quad .. \quad 16^{**1} \quad 16^{**0}$

corresponding to $a_0 .. a_{n-1}$

$a_{n-1} \quad .. \quad a_1 \quad a_0$

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a| = a = a_{n-1} * 16^{**}(n-1) + ... a_1 * 16^{**1} + a_0 * 16^{**0}$$

5. Output $|a|=a$

Example: Convert a=110A written in radix-16 into radix-10

1. Obtain **a** i.e. **a = 110A** $rdx(a,16,4)=HEX(a,4)=110A$

$$l(a)=b(a,16)=n=4$$

2. Generate powers of 10 starting with $16^{**0} .. 16^{**3}$

$16^{**3} \quad 16^{**2} \quad 16^{**1} \quad 16^{**0}$

corresponding to $a_0 .. a_{n-1}$

$1 \quad 1 \quad 0 \quad A$

3. Multiply digits with corresponding powers, and

4. Add partial results

$$|a| = a = 1 * 16^{**3} + 1 * 16^{**2} + 0 * 8^{**1} + 10 * 16^{**0}$$

5. Output $|a|=a$

$$|a|=a = 4096 + 256 + 10 = 4362$$



Algorithm **Binary-2-Hex**

1. Group into groups of four bit right to left (pad with leading zeroes leftmost group if < 4 bit)
2. Convert each group into denary, [eg see DHOB]
3. and result into hex.

Binary	Groups of 4	Into denary	Into Hexadecimal		
01000101->	`0100`0101->	`4`5 ->	45	0x45	0X45
11101011->	`1110`1011->	`14`11 ->	EB	0xeb	0XEB
1000101->	`0100`0101->	`4`5 ->	45	0x45	0X45

a, A is 10	b , B is 11	c, C is 12
d, D is 13	e , E is 14	f, F is 15



Algorithm **Binary-2-Octal**

1. Group into groups of three bit right to left (pad with leading zeroes leftmost group if < 3 bit)
2. Convert each group into denary
3. and result is in octal!

Binary	Groups of 3	Into denary	and it is in Octal	
01000101->	`001`000`101->	`1`0`5->	105	0o105
11101011->	`011`101`011->	`3`5`3 ->	353	0o353



Algorithm Denary-2-Binary (n) left-2-right

1. Generate all powers of 2 less than or equal to n right (smallest) to left (largest).
2. temp=n;
3. from leftmost to rightmost power,
4. if temp < power generate 0 for bit position and temp remains the same
5. else generate 1 for bit position and temp=temp-power
6. print generated bit
7. move right

n=89

	64	32	16	8	4	2	1	
	1	0	1	1	0	0	1	
	temp<64?	temp<32?	temp<16?	temp<8?	temp<4?	temp<2?	temp<1?	
temp:	89 ->	25 ->	25 ->	9 ->	1 ->	1 ->	1 ->	0



Algorithm Denary-2-Binary (n) right-2-left

1. temp=n
2. generate bit of n right (least significant) to left (most significant)
3. Is temp odd? if yes generate 1 and temp= (temp-1)/2
4. else generate 0 and temp= temp/2;
5. If temp ==0 stop
6. else goto line 3

n=89

	64	32	16	8	4	2	1	
	1	0	1	1	0	0	1	
0	1	2	5	11	22	44	89	temp
stop	is it odd?	is it odd?	is it odd?	is it odd?	is it odd?	is it odd?	is it odd?	



0.a.1.	$\text{pow}(2, 2)$	$= 4$	$2*2$
0.a.2.	$2**3$	$= 8$	$2*2*2$
0.a.3.	2^{20}	$= 1048576$	
0.a.4	2^{20}	$\sim 10^3$	$2^{20} = 2^{10} * 2^{10} = 1024*1024$
0.a.5.	$2**20$	$= 1048576$	0.a.3 0.a.4 ($=1024*1024$)
0.a.6.	$\lg(4)$	$= 2$	$2*2 = 4 \Rightarrow \lg(4)$ is the exp 2.
0.a.7.	$\lg(1048576)$	$= 20$	$2^{20} = 1048576$ by 0.a.5
0.a.8.	$\lg(10^6)$	~ 20	$\lg(10^6) \sim \lg(1048576) = 20$
0.a.9	$\lg(10^{12})$	~ 40	$\lg(10^{12}) = \lg(10^6 * 10^6) = 20 + 20$
0.a.10	$\text{ceil}(10)$	$= 10$	definition



Convert: Put (0x,0X,0o,0b) prefix in the result

Evaluate: Do not put prefix in the result

- 0.b.1. Evaluate `rdx(26,10,0)` = 26
- 0.b.2. Evaluate `rdx(26,10,5)` = 00026
- 0.b.3. Evaluate `rdx(26,2,0)` = 11010
- 0.b.4. Convert denary 26 into binary = 0b11010
- 0.b.5. Evaluate `rdx(26,16,0)` = 1a (default)
- 0.b.6. Evaluate `HEX(26,0)` = 1A
- 0.b.7. Convert denary 26 into hexadecimal = 0x1a or 0X1A
- 0.b.8. Evaluate `rdx(26,16,4)` = 001a
- 0.b.9. Convert denary 26 into octal = 0o32
- 0.b.10. Convert `z=0x3A7` into `rdx(z,8,0)` = 0o1647
- 0.b.11. Convert `z=0x3A7` into `rdx(z,2,16)` = 0b0000001110100111

Hint: it might be easier to solve 0.b.11 before 0.b.10



Subject 0

Module 2 : CS Review

(Arrays and Pointers)



M Memory

11111101
01010101
00001111
00000001
11111111
00011101
00101010
00000011

Formal Definition:

Wrong course! → ECE

Informal Definition Memory is a sequence of byte(s).

Informal Abstraction Memory is an array of bytes.

Thus:

```
byte[] M = new byte[ 2**3 ];
```

length of M

```
M.length // # of elements
```

size of M (in B)

```
M.length * sizeof(byte)
```

Array M has length 8.

Array M is of size 8B.

M stands for Memory.

Memory stands for Main Memory

Secondary Memory refers to 'disk space'

Observation 1: Every byte of main memory has an associated address (also called index or offset)

Humans do not like binary numbers; they prefer to use for addresses numbers in other (than binary) radix systems. Computers love binary numbers they are hardwired to use them!

Observation 2: Every byte of main memory has a stored value, known as the contents of the byte associated with the corresponding address

Humans do not like binary numbers; they prefer to describe the contents of memory in a number representation other than binary

It is treacherous to ask. What is the address of byte 00001111 ?

Address 2 has contents 00001111, and

Address 7 has contents 00001111.

Address 7, Offset 7 from address 0, and Index 7 of M are synonyms with respect to this memory M.

M Memory

0	11111101
1	01010101
2	00001111
3	00000001
4	11111111
5	00011101
6	00101010
7	00001111



30. Address / Offset / Index

Different ways to indicate the same

'offset 2' means 'offset 2 from 0' i.e. 'address 0+2'

Address i.e. index i contains $M[i]$

Offset i i.e. address $0+i$ contains $M[0+i]$ i.e. $M[i]$

Multiple ways of saying the same thing

Memory at offset 2 contains 00001111

Memory at offset 2 is 00001111

Memory at address 2 contains 00001111

The contents of address 2 are 00001111

2 is also an index of an array

$M[2]$ contains 00001111

M is (Main)Memory

M Memory

0	11111101
1	01010101
2	00001111
3	00000001
4	11111111
5	00011101
6	00101010
7	00001111



31. Read and Write Operations

Memory Read : $A \rightarrow M[A]$

Input: Address A

Output: Contents of address A i.e. $M[A]$

M Memory

0	11111101
1	01010101
2	00001111
3	00000001
4	11111111
5	00011101
6	00101010
7	00001111

Memory Write : $(A, D) \rightarrow M[A] = D$

Input: Address A, and byte value D

Output: Write D into memory address A i.e. $M[A] = D$

Example: Memory Read with $A=2$,

//byte 00001111 is transferred from memory into the ...CPU...

Example Memory Write with $(A, D) = (7, 00000011)$

//byte 00001111 of address 7 will be overwritten by 00000011



Create array X in memory (length(X) and datatype(X) given)

```
byte [] X = new byte[5]; // length(X) is 5, datatype(X) is byte
```

What does this mean say in C or C++ or other languages?

1. datatype(X)=byte, length(X)=5, sizeof(byte)=1B are computed (i.e. evaluated).

2 Allocator finds free memory of space size(X) = length(X) * sizeof(byte)=5*1=5B

3. X is set to an address, the starting address of the free memory allocated to X.

Example: size(X)=5B, and X := 532. Sometimes X is known as the base address of X.

Memory

M

530	11111101
531	01010101
532	00001111
533	00000001
534	11111111
535	00011101
536	00101010
537	00000011

X+0	X[0]
X+1	X[1]
X+2	X[2]
X+3	X[3]
X+4	X[4]

What is the address in memory of X[k]?

This is easy to compute

$X + k * \text{sizeof}(\text{datatype}(X)) = X + k * \text{sizeof}(\text{byte})$

where X is the (starting) address of X i.e. 532. Therefore

X[0] is stored at address $X + 0 * 1 = 532$

X[1] is stored at address $X + 1 * 1 = 533$

X[2] is stored at address $X + 2 * 1 = 534$

etc

X[4] is stored at address $X + 4 * 1 = 536$

Memory

M

530	11111101
531	01010101
532	00001111
533	00000001
534	11111111
535	00011101
536	00101010
537	00000011

Create array XYin memory (length(Y) and datatype(Y) given)

```
int [] Y = new int[2];
```

What does this mean say in C or C++ or other languages.

1. $\text{datatype}(Y)=\text{int}$, $\text{length}(Y)=2$, $\text{sizeof}(\text{int})=4\text{B}$ are computed (i.e. evaluated).
- 2 Allocator find free memory of space $\text{size}(Y) = \text{length}(Y) * \text{sizeof}(\text{int})=2*4=8\text{B}$
3. Y is set to an address, the starting address of the free memory allocated to Y.

Example: $\text{size}(Y)=8\text{B}$, and $Y := 530$

$Y+0*4$ $Y[0]$ 1stB

$Y[0]$ 2ndB

$Y[0]$ 3rdB

$Y[0]$ 4thB

$Y+1*4$ and

so

on for

$Y[1]$

What is the address in memory of $Y[k]$?

This is easy to compute

$$Y + k * \text{sizeof}(\text{datatype}(Y)) = Y+k*\text{sizeof}(\text{int})$$

where Y is the (starting) address of Y i.e. 530. Therefore

$Y[0]$ is stored at address $Y+0*4=530$

$Y[1]$ is stored at address $Y+1*4=534$



34. Little Big Endian Problems

530	11111101
531	01010101
532	00001111
533	00000001

Q1 What is the integer stored in
memory addresses 530-533? // i.e Y[o]

Case1 (Little Endian). [Intel uses it]

00000001 00001111 01010101 11111101

Least significant byte -> Low Memory Address

Most significant byte -> High Memory Address

Case2 (Big Endian). [PowerPC uses it]

11111101 01010101 00001111 00000001

Least significant byte -> High Memory Address

Most significant byte -> Low Memory Address



Memory

M

Textbook Memory

Textbook twist

**Chapter 1 (and
 as needed later):**



Memory
 addresses store
 2B not 1B !!!

530	FD
531	44
532	0F
533	01
534	FF
535	1D
536	26
537	03



1B

530	00FD
531	0044
532	000F
533	0001
534	00FF
535	001D
536	0026
537	0003



3hex digit = 12bit

4hex digit = 2B

Textbook : One address (e.g. 531) is associated with 2B (00 and 44)

Moreover an address is 12bit i.e. 3 hex digit (eg 531)

36. Memorize those numbers



Latency: time to read one (first byte) from ...

Time to read one byte from cache memory (typical) **20ns**

Time to read one byte from main memory (typical) **80ns**

Time to read one byte from a HDD (hard disk drive) **20ms**

(we read a HDD sector at a time and extract byte from it)

Time* to execute one (elementary) instruction nowadays **0.25-1ns**

*: **depends on CPU's frequency f**

Terms cache, main memory to be defined next chapter!



Subject 0

Module 3 : CS Review

(Pages and Paging)



An ancient scroll was a book of one page.

References were simple: Read scroll line (number) 571 i.e. $L=571$

A scroll

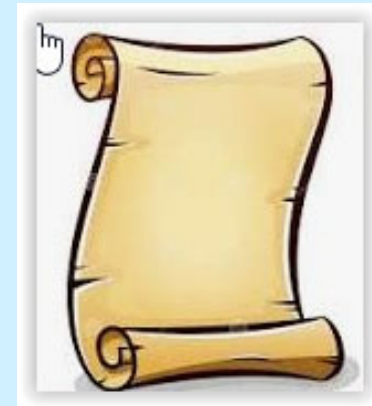
Source: Wikipedia

Modern books have pages. In well-organized books every page has the same number of lines as any other page. This is known as page size

Page size s is number of lines per page !

Page number P is a number/label on every page that identifies that page

Page offset T is a line number/offset within a page that identifies the line of that page



What if a scroll becomes organized as a book ?

Say page size s is 32. (Every page has thus 32 lines that use page offsets 0 .. 31.)

Question: What is the page number and line number (page offset) of scroll line $L=571$ in a book who page size is 32 (lines / page). That is how can I find in the book scroll line $L=571$?

Answer: Assuming page numbers start from 0 just like line numbers / page offsets,

Step 1: Integer division (quotient and remainder) of L by s : Divide L by s . ($L=571, s=32$.)

Step 2: Obtain P, T such that $L = P * s + T \Rightarrow 571 = 17 * 32 + 27,$

$$P = L/s = 571/32 = 17 \text{ and } T = L \% s = 571 \% 32 = 27$$

$$L=571 \text{ implies } (P, T) = (\text{PageNumber}, \text{PageOffset}) = (17, 27)$$

Step 3: Quotient P is the page number and remainder T is the page offset (line number).

Step 4: Go to page $P=17$ and in that page to page offset (line number) $T=27$



The benefit of having a page size that is a power of 2. $s=32$ is a power of 2 : $32 = 2^{**5}$

Integer division is replaced with bit manipulation

Step 1. Find $\lg(\text{PageSize})=\lg(s)$ $s=32 \Rightarrow \lg(s)=5$

Step 2. Write down $L=571$ in binary **1000111011**

Step 3. Isolate $\lg(s)$ rightmost bit: It is the page offset **1000111011**

Step 4. Remaining (leftmost) bit is page number: **1000111011**

Binary **10001** is denary $P=17!$

Binary **11011** is denary $T=27!$

$L=571$ $(P,T)=(\text{PageNumber}, \text{PageOffset})=(17,27)$

Step 5. Go to page **17** offset **27** (within page 17)



39. Paging (also in Sub 7)

We have N bytes of MainMemory (with addresses $0..N-1$) and we want to arrange them into N/S pages of page size S i.e. $N = (N/S) \times S$

1. Memory Size: $N=2^{**}n$ N always a power of two; easy to deal with
2. #Bit for addresses $0..N-1$: n $\lg(N)=n$; slide 15
3. # pages: N/S page number range $0 \dots N/S-1$
4. #bit to describe a page: $\lg(N/S)=\lg N - \lg S = n - \lg S$ $\lg(N/S)$; see slide 15
5. Page size: S #of B per page
6. #B in a page S range of offsets: $0..S-1$
7. #bit to describe a B in a page: $\lg(S)$ slide 15 again

the bit that describe a B in a page are called an offset (aka index of a B in a page)

How much is $4+7$? $4+7$ is equal to 2. Joke... for line 4 + line 7 is equal to line 2



$$\lg(N) = n = (n - \lg S) + \lg S$$

Practice 1.1



1.1.1. <code>pow(2,10)</code>	= ?	1.1.11 <code>lg(2**20)</code>	= ?
1.1.2. <code>2**8</code>	= ?	1.1.12 <code>lg(8)</code>	= ?
1.1.3. <code>2^10</code>	= ?	1.1.13 <code>lg(256)</code>	= ?
1.1.4 <code>2^12</code>	= ?	1.1.14 <code>lg(65536)</code>	= ?
1.1.5. <code>2**16</code>	= ?	1.1.15 <code>lg(1000)</code>	~ ?
1.1.6. <code>pow(2,20)</code>	~ ?	1.1.16 <code>lg(10**9)</code>	~ ?
1.1.7. <code>pow(2,30)</code>	~ ?	1.1.17 <code>lg(4096)</code>	= ?
1.1.8. <code>2**0</code>	= ?	1.1.18 <code>lg(2)</code>	= ?
1.1.9 <code>ceil(4.5)</code>	= ?	1.1.19 <code>lg(1)</code>	= ?
1.1.10 <code>floor(5)</code>	= ?	1.1.20 <code>lg(2**k)</code>	= ?

Practice 1.2



1.2.1. Write ten bit compactly =

1.2.2. Write ten byte compactly =

1.2.3. How many KiB in 4096B? =

1.2.4. How many MiB in 4096KiB =

1.2.5. How many KiB in 1048576B =

1.2.x. Convert 64 into hexadecimal = 0x40

1.2.6. Convert 64 into binary =

1.2.7. Convert 64 into octal =

1.2.8. Convert 127 into hexadecimal =

1.2.9. Convert 512 into hexadecimal =

1.2.10. Convert 256 into binary =

Practice 1.3



1.3.1. Evaluate $\text{rdx}(126, 10, 0)$ =

1.3.2. Evaluate $\text{rdx}(126, 10, 5)$ =

1.3.3. Evaluate $\text{rdx}(126, 2, 0)$ =

1.3.4. Convert denary 126 into binary =

1.3.5. Evaluate $\text{rdx}(126, 16, 0)$ =

1.3.6. Evaluate $\text{HEX}(126, 0)$ =

1.3.7. Convert denary 126 into hexadecimal =

1.3.8. Evaluate $\text{rdx}(126, 16, 4)$ =

1.3.9. Convert denary 126 into octal =

1.3.10. Convert $z=0xA37$ into $\text{rdx}(z, 8, 0)$ =

1.3.11. Convert $z=0xA37$ into $\text{rdx}(z, 2, 16)$ =

Canvas Quizzes: A Self Assessment 0 with answers to most questions



Whenever a question refers to bit or B or aggregates of B ALWAYS include the UNIT immediately after the magnitude (no space in-between). Likewise for time or other measures.

1.4.1 How many bit in one byte?

Correct answer: 8bit

Incorrect answer: 8

Incorrect answer: 8b

1.4.2 How many byte in 4KiB?

Correct answer: 4096B

1.4.3 Write 4096B in KiB form.

Correct answer: 4KiB

1.4.4 How many millisecond in one second?

Correct answer: 1000ms

Canvas Quizzes: A Self Assessment 0 with answers to most questions

Practice 2.1



2.1.1. Go to Slide 33 (Page 40) left side. A 32-bit integer is stored starting at address 0x534.

What is its value in binary? =

2.1.2. What is its leftmost bit ? =

2.2.3. What is its rightmost bit? =

Canvas Quizzes: A Self Assessment 0 with answers to most questions



Paging is being used with pagesize 100B. The following address $L=234567$ is given. For 3.1.1-3.1.2 provide answers in denary.

3.1.1. What is the pagenumber P for L ? =

3.1.2. What is the pageoffset T for L ? =

Paging is being used with pagesize 4096B. The following address $L=234567$ is given For 3.1.3-3.1.4 provide answers in binary.

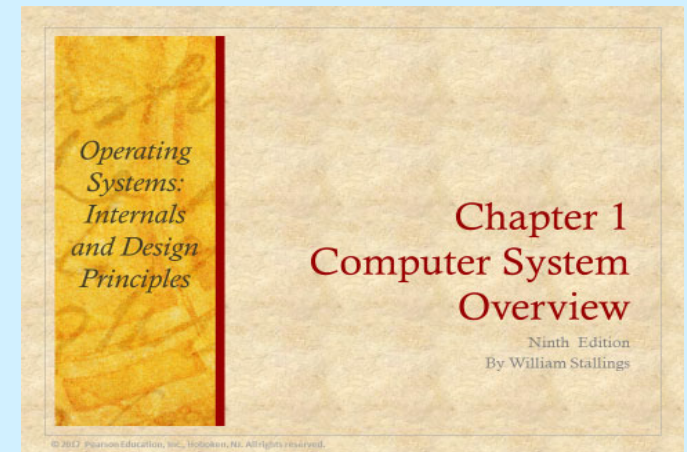
3.1.3. What is the pagenumber P for L ? =

3.1.4. What is the pageoffset T for L ? =



Primarily textbook author-based (figures from textbook)

Textbook slides look different (brownish)

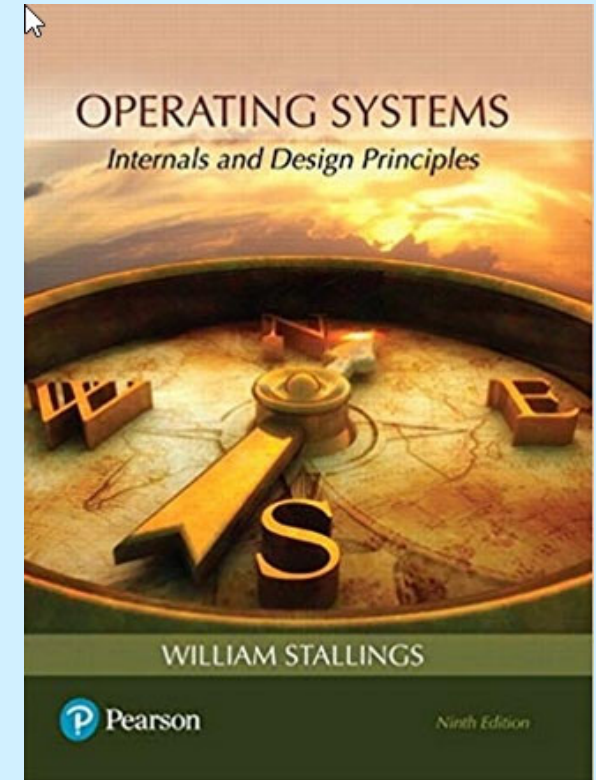


Also instructor notes

My notes look like this collection of slides! (blueish)

Textbook

OS is a course THAT requires YOU to read the
TEXTBOOK



NJIT

The logo features the letters 'NJIT' in a white, serif font. A white, curved line starts under the 'J' and sweeps upwards and to the right, ending under the 'T'.

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department