Bits, Bytes, and Integers

CENG331 - Computer Organization

Adapted from slides of the textbook: http://csapp.cs.cmu.edu/

Hello World!

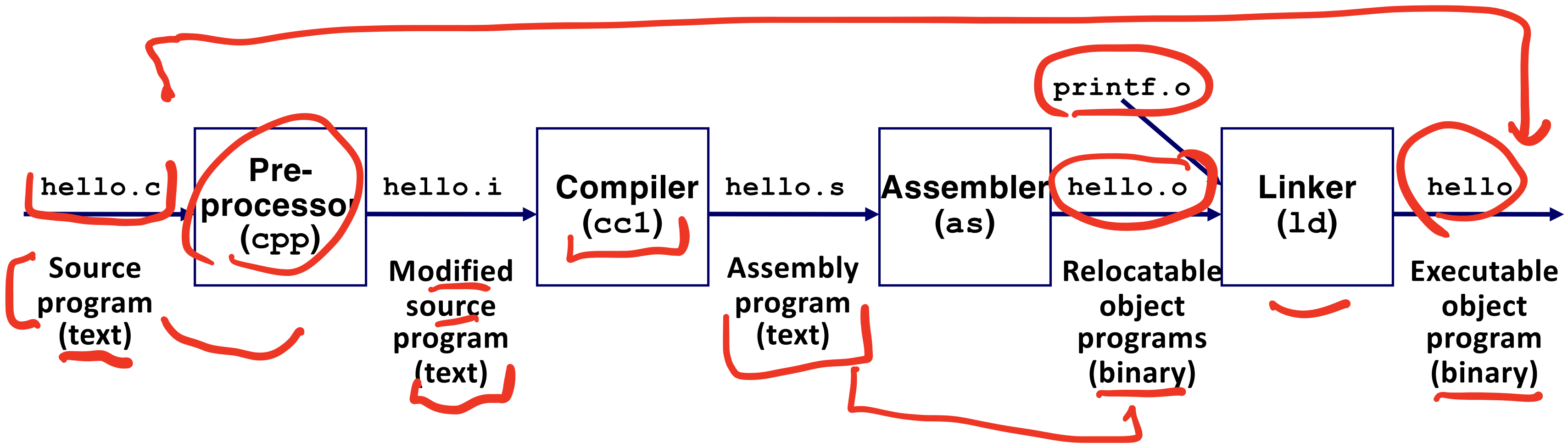
- **What happens under the hood?**

hello.c

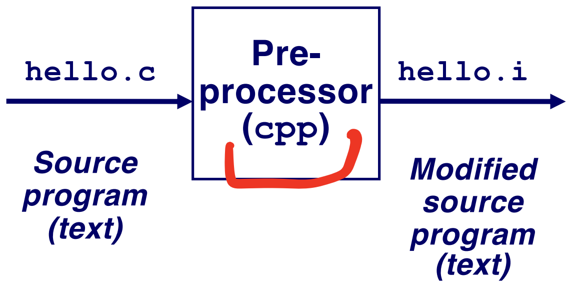
```
#include <stdio.h>


int main() {
  printf("Hello World");
}
```

Compilation of hello.c



Preprocessing




```
#include <stdio.h>

int main() {
    printf("Hello
    World");
}
```

C

#define

cpp hello.c > hello.i

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
..........................................
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;
..........................................
int main() {
    printf("Hello World");
}
```

C

Compiler

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
……………………………..
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;
……………………………..
int main() {
    printf("Hello World");
}
```

C

hello.i → **Compiler (cc1)** → hello.s

Text Text

Assembly

gcc -Wall -S hello.i > hello.s

```
.file    "hello.c"
         .section
.rodata
.LC0:
         .string "Hello
World"
         .text
.globl main
         .type   main,
@function
main:
         pushl   %ebp
         movl    %esp, %ebp
         subl    $8, %esp
         andl    $-16, %esp
         movl    $0, %eax
         addl    $15, %eax
         addl    $15, %eax
         shrl    $4, %eax
         sall    $4, %eax
         subl    %eax, %esp
         subl    $12, %esp
         pushl   $.LC0
         call    printf
         addl    $16, %esp
         leave
         ret
         .size   main, .-
main
         .section
.note.GNU-
stack,"",@progbits
         .ident  "GCC:
(GNU) 3.4.1"
```

Assembler

```
.file    "hello.c"
         .section
.rodata
.LC0:
         .string "Hello
World"
         .text
.globl main
         .type   main,
@function
main:
         pushl   %ebp
         movl    %esp, %ebp
         subl    $8, %esp
         andl    $-16, %esp
         movl    $0, %eax
         addl    $15, %eax
         addl    $15, %eax
         shrl    $4, %eax
         sall    $4, %eax
         subl    %eax, %esp
         subl    $12, %esp
         pushl   $.LC0
         call    printf
         addl    $16, %esp
         leave
         ret
         .size   main, .-
main
         .section
.note.GNU-
stack,"",@progbits
         .ident  "GCC:
(GNU) 3.4.1"
```

Text

hello.s → **Assembler (as)** → hello.o

as hello.s -o hello.o

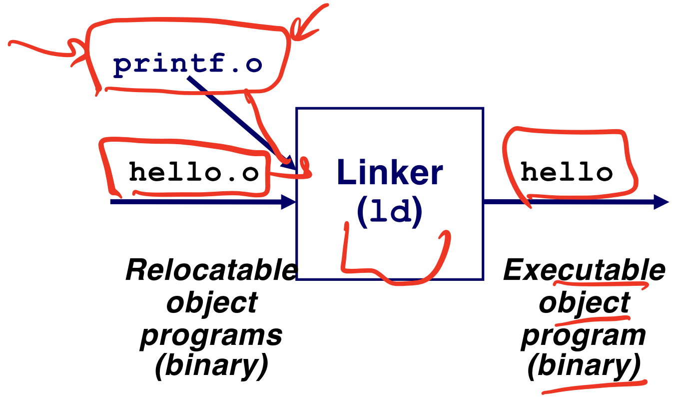
Binary

Object

```
0000500 nul nul nul nul esc nul nul nul  ht nul nul nul nul nul nul nul
0000520 nul nul nul nul   d etx nul nul dle nul nul nul  ht nul nul nul
0000540 soh nul nul nul eot nul nul nul  bs nul nul nul   % nul nul nul
0000560 soh nul nul nul etx nul nul nul nul nul nul nul   d nul nul nul
0000600 nul nul nul nul nul nul nul nul nul nul nul nul eot nul nul nul
0000620 nul nul nul nul   + nul nul nul  bs nul nul nul etx nul nul nul
0000640 nul nul nul nul   d nul nul nul nul nul nul nul nul nul nul nul
0000660 nul nul nul nul eot nul nul nul nul nul nul nul   0 nul nul nul
0000700 soh nul nul nul stx nul nul nul nul nul nul nul   d nul nul nul
0000720  ff nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0000740 nul nul nul nul   8 nul nul nul soh nul nul nul nul nul nul nul
0000760 nul nul nul nul   p nul nul nul nul nul nul nul nul nul nul nul
0001000 nul nul nul nul soh nul nul nul nul nul nul nul   H nul nul nul
0001020 soh nul nul nul nul nul nul nul nul nul nul nul   p nul nul nul
0001040   2 nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0001060 nul nul nul nul dc1 nul nul nul etx nul nul nul nul nul nul nul
0001100 nul nul nul nul   " nul nul nul   Q nul nul nul nul nul nul nul
0001120 nul nul nul nul soh nul nul nul nul nul nul nul soh nul nul nul
0001140 stx nul nul nul nul nul nul nul nul nul nul nul   , stx nul nul
0001160  sp nul nul nul  nl nul nul nul  bs nul nul nul eot nul nul nul
0001200 dle nul nul nul  ht nul nul nul etx nul nul nul nul nul nul nul
0001220 nul nul nul nul   L etx nul nul nak nul nul nul nul nul nul nul
0001240 nul nul nul nul soh nul nul nul nul nul nul nul nul nul nul nul
0001260 nul nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0001300 nul nul nul nul nul nul nul nul eot nul   q del nul nul nul nul
0001320 nul nul nul nul nul nul nul nul etx nul soh nul nul nul nul nul
0001340 nul nul nul nul nul nul nul nul etx nul etx nul nul nul nul nul
0001360 nul nul nul nul nul nul nul nul etx nul eot nul nul nul nul nul
0001400 nul nul nul nul nul nul nul nul etx nul enq nul nul nul nul nul
0001420 nul nul nul nul nul nul nul nul etx nul ack nul nul nul nul nul
0001440 nul nul nul nul nul nul nul nul etx nul bel nul  ht nul nul nul
0001460 nul nul nul nul   . nul nul nul dc2 nul soh nul  so nul nul nul
0001500 nul nul nul nul nul nul nul nul dle nul nul nul nul   h   e   l
0001520   l   o   .   c nul   m   a   i   n nul   p   r   i   n   t   f
0001540 nul nul nul nul  sp nul nul nul soh enq nul nul   % nul nul nul
0001560 stx  ht nul nul
0001564
```

od –a hello.o

Linker




```
0000500 nul nul nul nul esc nul nul nul  ht nul nul nul nul nul nul nul
0000520 nul nul nul nul   d etx nul nul dle nul nul nul  ht nul nul nul
0000540 soh nul nul nul eot nul nul nul  bs nul nul nul   % nul nul nul
0000560 soh nul nul nul etx nul nul nul nul nul nul nul   d nul nul nul
0000600 nul nul nul nul nul nul nul nul nul nul nul nul eot nul nul nul
0000620 nul nul nul nul   + nul nul nul  bs nul nul nul etx nul nul nul
0000640 nul nul nul nul   d nul nul nul nul nul nul nul nul nul nul nul
0000660 nul nul nul nul eot nul nul nul nul nul nul nul   0 nul nul nul
0000700 soh nul nul nul stx nul nul nul nul nul nul nul   d nul nul nul
0000720  ff nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0000740 nul nul nul nul   8 nul nul nul soh nul nul nul nul nul nul nul
0000760 nul nul nul nul   p nul nul nul nul nul nul nul nul nul nul nul
0001000 nul nul nul nul soh nul nul nul nul nul nul nul   H nul nul nul
0001020 soh nul nul nul nul nul nul nul nul nul nul nul   p nul nul nul
0001040   2 nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0001060 nul nul nul nul dc1 nul nul nul etx nul nul nul nul nul nul nul
0001100 nul nul nul nul   " nul nul nul   Q nul nul nul nul nul nul nul
0001120 nul nul nul nul soh nul nul nul nul nul nul nul soh nul nul nul
0001140 stx nul nul nul nul nul nul nul nul nul nul nul   , stx nul nul
0001160  sp nul nul nul  nl nul nul nul  bs nul nul nul eot nul nul nul
0001200 dle nul nul nul  ht nul nul nul etx nul nul nul nul nul nul nul
0001220 nul nul nul nul   L etx nul nul nak nul nul nul nul nul nul nul
0001240 nul nul nul nul soh nul nul nul nul nul nul nul nul nul nul nul
0001260 nul nul nul nul nul nul nul nul nul nul nul nul soh nul nul nul
0001300 nul nul nul nul nul nul nul nul eot nul   q del nul nul nul nul
0001320 nul nul nul nul nul nul nul nul etx nul soh nul nul nul nul nul
0001340 nul nul nul nul nul nul nul nul etx nul etx nul nul nul nul nul
0001360 nul nul nul nul nul nul nul nul etx nul eot nul nul nul nul nul
0001400 nul nul nul nul nul nul nul nul etx nul enq nul nul nul nul nul
0001420 nul nul nul nul nul nul nul nul etx nul ack nul nul nul nul nul
0001440 nul nul nul nul nul nul nul nul etx nul bel nul  ht nul nul nul
0001460 nul nul nul nul   . nul nul nul dc2 nul soh nul  so nul nul nul
0001500 nul nul nul nul nul nul nul nul dle nul nul nul nul   h   e   l
0001520   l   o   .   c nul   m   a   i   n nul   p   r   i   n   t   f
0001540 nul nul nul nul  sp nul nul nul soh enq nul nul   % nul nul nul
0001560 stx  ht nul nul
0001564
```

gcc hello.o –o hello

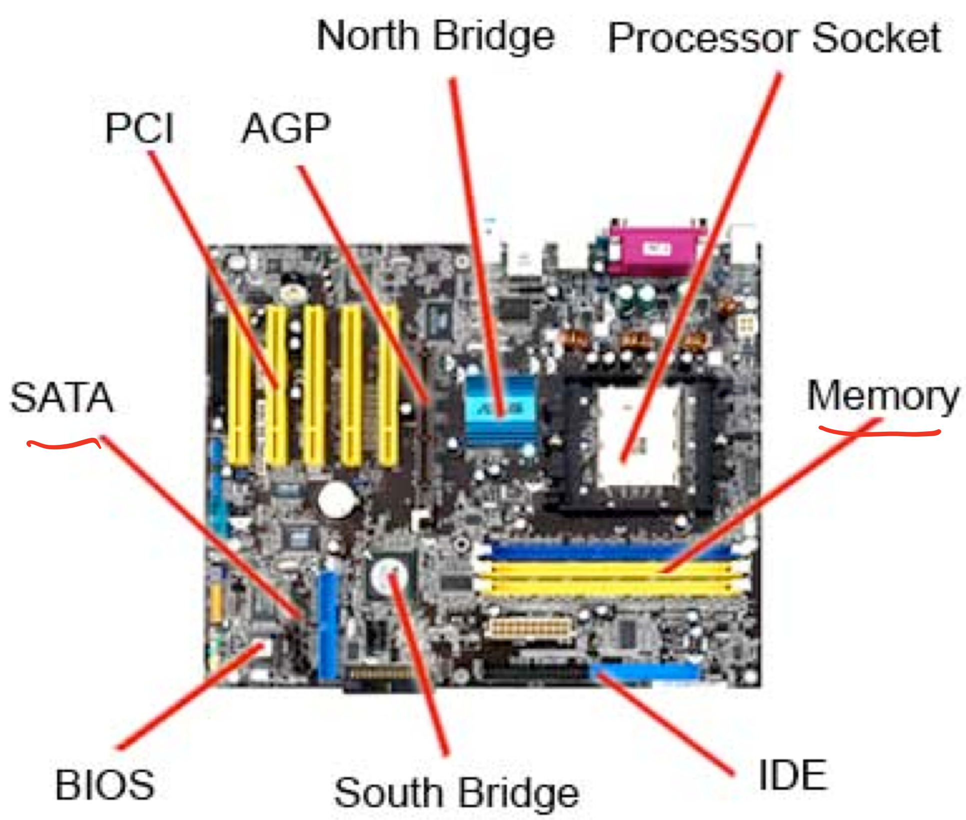
```
0000000 del   E   L   F soh soh soh nul nul nul nul nul nul nul nul nul
0000020 stx nul etx nul soh nul nul nul   @ stx eot  bs   4 nul nul nul
0000040   X  cr nul nul nul nul nul nul   4 nul  sp nul bel nul   ( nul
0000060   ! nul  rs nul ack nul nul nul   4 nul nul nul   4 nul eot  bs
0000100   4 nul eot  bs   ` nul nul nul   ` nul nul nul enq nul nul nul
0000120 eot nul nul nul etx nul nul nul dc4 soh nul nul dc4 soh eot  bs
0000140 dc4 soh eot  bs dc3 nul nul nul dc3 nul nul nul eot nul nul nul
0000160 soh nul nul nul soh nul nul nul nul nul nul nul nul nul eot  bs
0000200 nul nul eot  bs  bs eot nul nul  bs eot nul nul enq nul nul nul
0000220 nul dle nul nul soh nul nul nul  bs eot nul nul  bs dc4 eot  bs
0000240  bs dc4 eot  bs nul soh nul nul eot soh nul nul ack nul nul nul
0000260 nul dle nul nul stx nul nul nul dc4 eot nul nul dc4 dc4 eot  bs
0000300 dc4 dc4 eot  bs   H nul nul nul   H nul nul nul ack nul nul nul
0000320 eot nul nul nul eot nul nul nul   ( soh nul nul   ( soh eot  bs
0000340   ( soh eot  bs  sp nul nul nul  sp nul nul nul eot nul nul nul
0000360 eot nul nul nul   Q   e   t   d nul nul nul nul nul nul nul nul
0000400 nul nul nul nul nul nul nul nul nul nul nul nul ack nul nul nul
0000420 eot nul nul nul   /   l   i   b   /   l   d   -   l   i   n   u
0000440   x   .   s   o   .   2 nul nul eot nul nul nul dle nul nul nul
0000460 soh nul nul nul   G   N   U nul nul nul nul nul stx nul nul nul
0000500 stx nul nul nul enq nul nul nul etx nul nul nul ack nul nul nul
0000520 enq nul nul nul soh nul nul nul etx nul nul nul nul nul nul nul
0000540 nul nul nul nul nul nul nul nul stx nul nul nul nul nul nul nul
..........................................................................
```

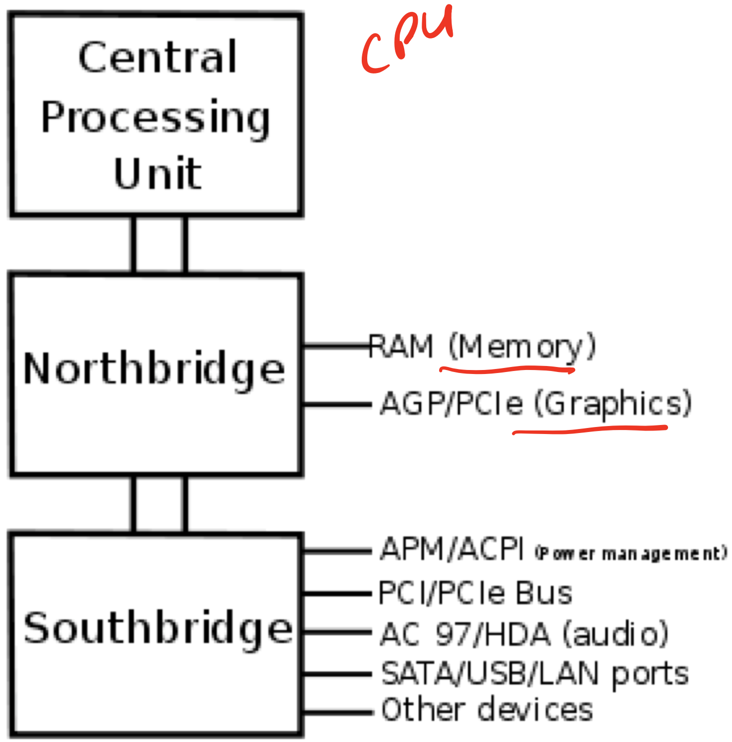
od –a hello

Finally…

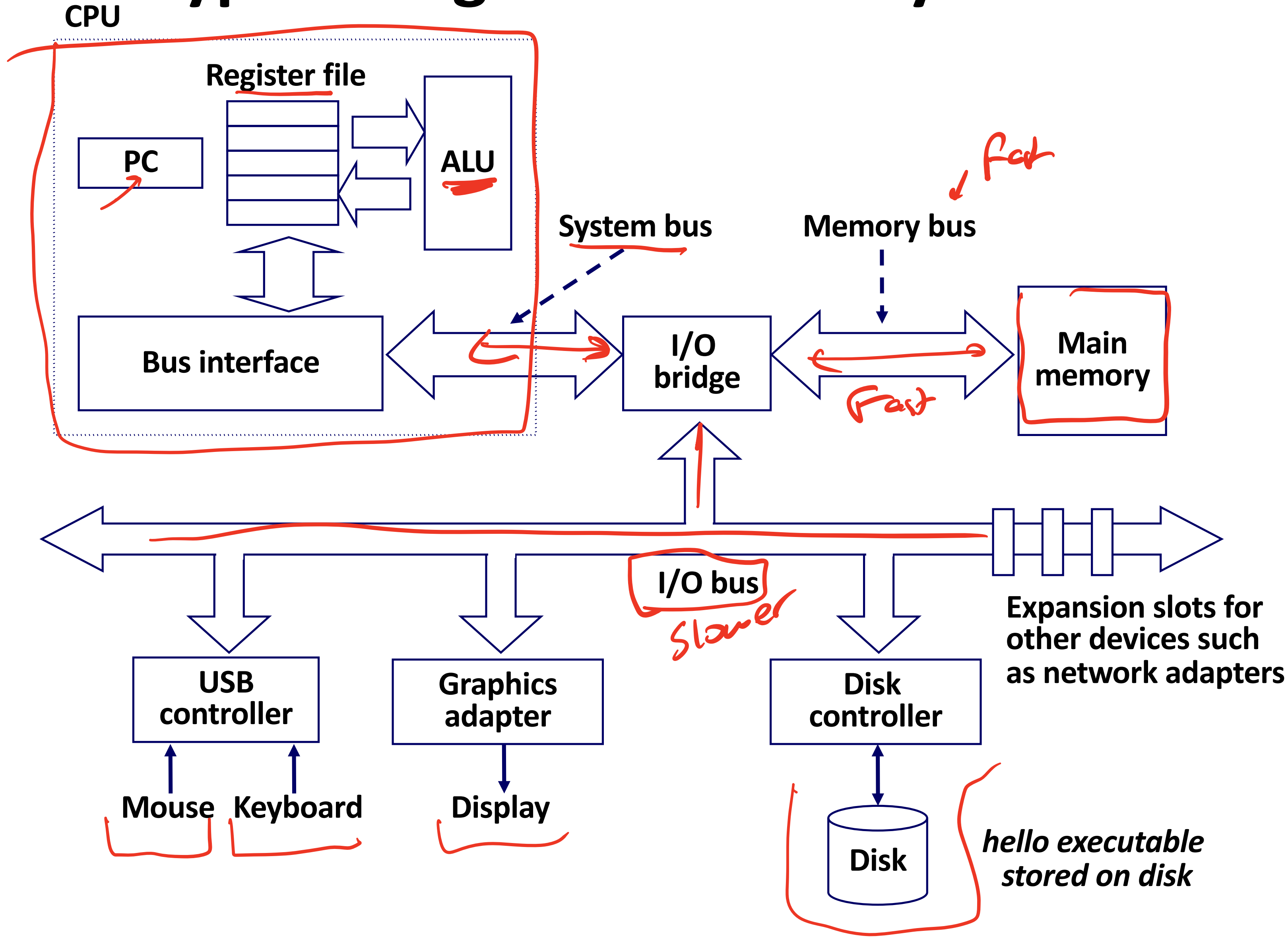
```
$ gcc hello.o -o hello
$ ./hello
Hello World$
```

How do you say “Hello World”?

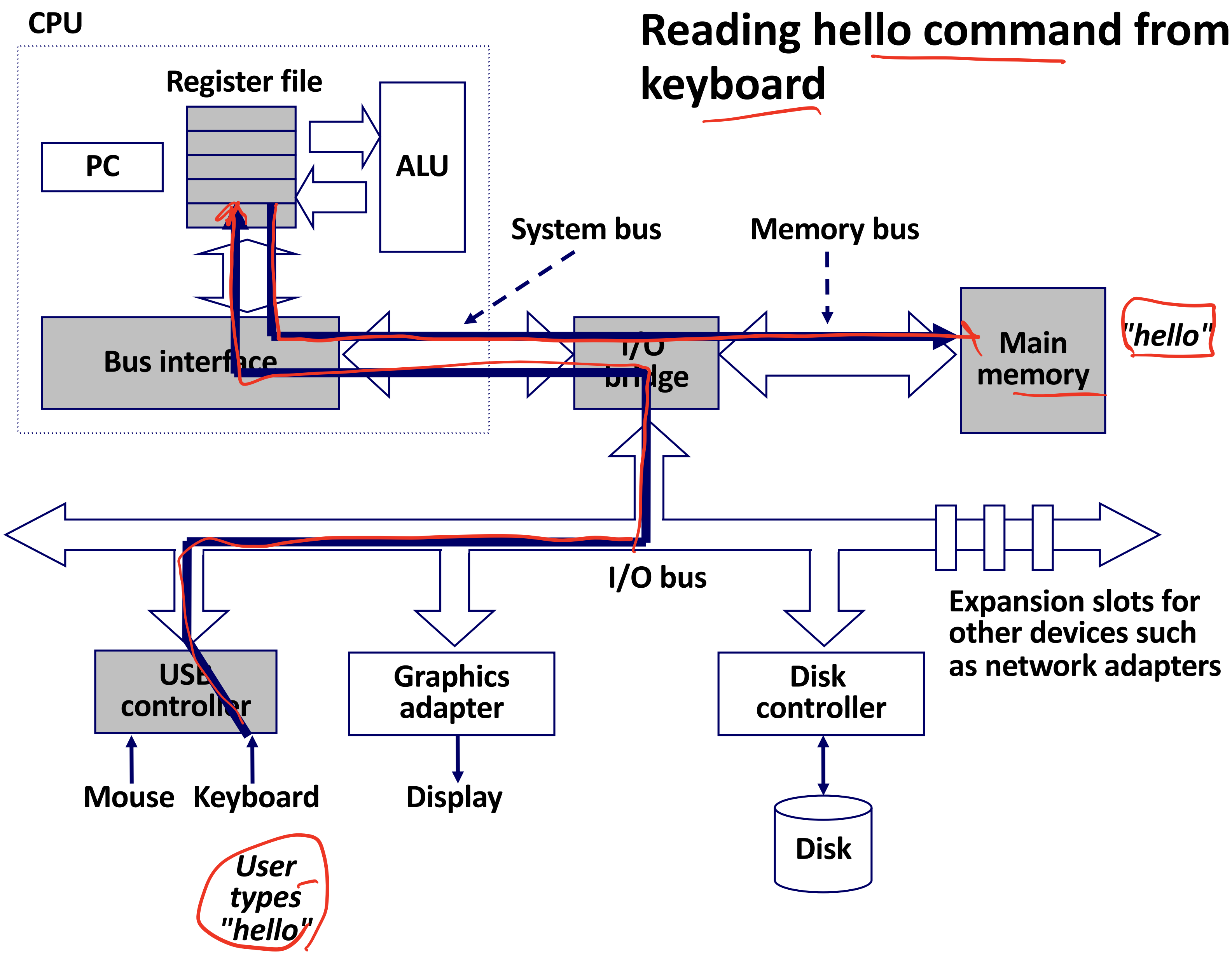




Typical Organization of System

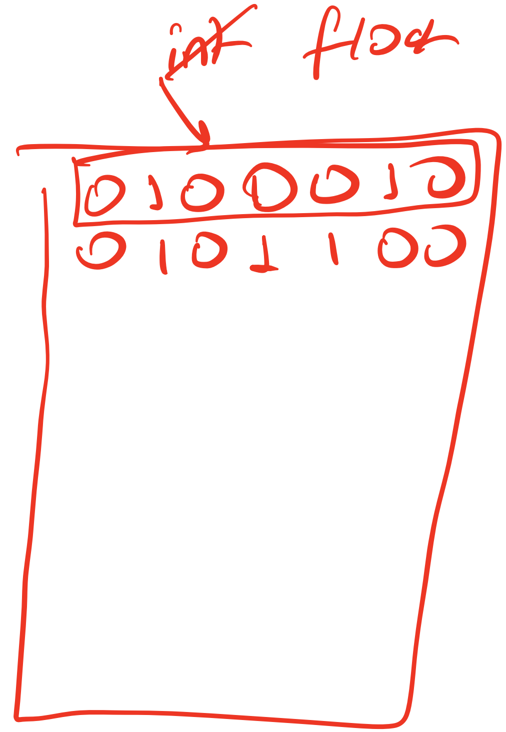


Reading hello command from keyboard



Today: Bits, Bytes, and Integers

- **Representing information as bits**
- **Bit-level manipulations**
- **Integers**
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- **Representations in memory, pointers, strings**



For example, can count in binary

- **Base 2 Number Representation**
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
  - Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

Encoding Byte Values

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters '0' to '9' and 'A' to 'F'
    - Write $FA1D37B_{16}$ in C as
      - 0xFA1D37B
      - 0xfa1d37b

        » Convert in 4-bit groups

1111 1010 0001 1101 ...

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Example Data Representations

word-size

"64" word size

Intel

8086
80286

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	–	–	10/16
pointer Address	4	8	8

32 bit

64-bit

Today: Bits, Bytes, and Integers

- Representing information as bits
- **Bit-level manipulations**
- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings

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 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

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

<table>
<tr><th>|</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

Not

- ~A = 1 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

General Boolean Algebras

- **Operate on Bit Vectors**
  - Operations applied bitwise

```
               OR                   XOR
  01101001        01101001             01101001
& 01010101      | 01010101           ^ 01010101          ~ 01010101
  --------        --------             --------            --------
  01000001        01111101             00111100            10101010
```

- **All of the Properties of Boolean Algebra Apply**

Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$

    6th 5th 3rd 0th bit

    - 01101001 { 0, 3, 5, 6 }
    - *76543210*
    - 01010101 { 0, 2, 4, 6 }
    - *76543210*
- **Operations**

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }

Bit-Level Operations in C

- **Operations &, |, ~, ^ Available in C**
  - Apply to any "integral" data type
    - `long`, `int`, `short`, `char`, `unsigned`
  - View arguments as bit vectors
  - Arguments applied bit-wise
- **Examples (Char data type)**
  - `~0x41 = 0xBE`
    - $\sim 01000001_2 = 10111110_2$
  - `~0x00 = 0xFF`
    - $\sim 00000000_2 = 11111111_2$
  - `0x69 & 0x55 = 0x41`
    - $01101001_2 \ \& \ 01010101_2 = 01000001_2$
  - `0x69 | 0x55 = 0x7D`
    - $01101001_2 \mid 01010101_2 = 01111101_2$

~0x41

0100 0001

$\underbrace{1011}_{B}$ $\underbrace{1110}_{E}$

Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - `&&`, `||`, `!`
    - View 0 as "False"
    - Anything nonzero as "True"
    - Always return 0 or 1
    - Early termination
- **Examples (char data type)**
  - `!0x41 = 0x00`
  - `!0x00 = 0x01`
  - `!!0x41 = 0x01`
  - `0x69 && 0x55 = 0x01`
  - `0x69 || 0x55 = 0x01`
  - `p && *p` (avoids null pointer access due to lazy evaluation)

~~0x41 = 0x41

TRUE TRUE TRUE

!TRUE = FALSE → 0x00

!FALSE = TRUE → 0x01

p && *p → 0

p = 0x0

Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - `&&, ||, !`
    - View 0 as "False"
    - Anything nonzero as "True"
    - Always return 0 or 1
    - Early termination

Watch out for && vs. & (and || vs. |)... one of the more common oopsies in C programming

- **Examples (char data type)**
  - `!0x41 = 0x00`
  - `!0x00 = 0x01`
  - `!!0x41 = 0x01`
  - `0x69 && 0x55 = 0x01`
  - `0x69 || 0x55 = 0x01`
  - `p && *p` (avoids null pointer access)

Shift Operations

- **Left Shift: x << y**
  - Shift bit-vector x left y positions
    - – Throw away extra bits on left
    - Fill with 0's on right
- **Right Shift: x >> y**
  - Shift bit-vector x right y positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on right
- **Undefined Behavior**
  - Shift amount < 0 or ≥ word size

Argument x	01100010
x << 3	00010000
Log. >> 2	00011000
Arith. >> 2	00011000

Argument x	10100010
x << 3	00010000
Log. >> 2	00101000
Arith. >> 2	11101000

Discarded

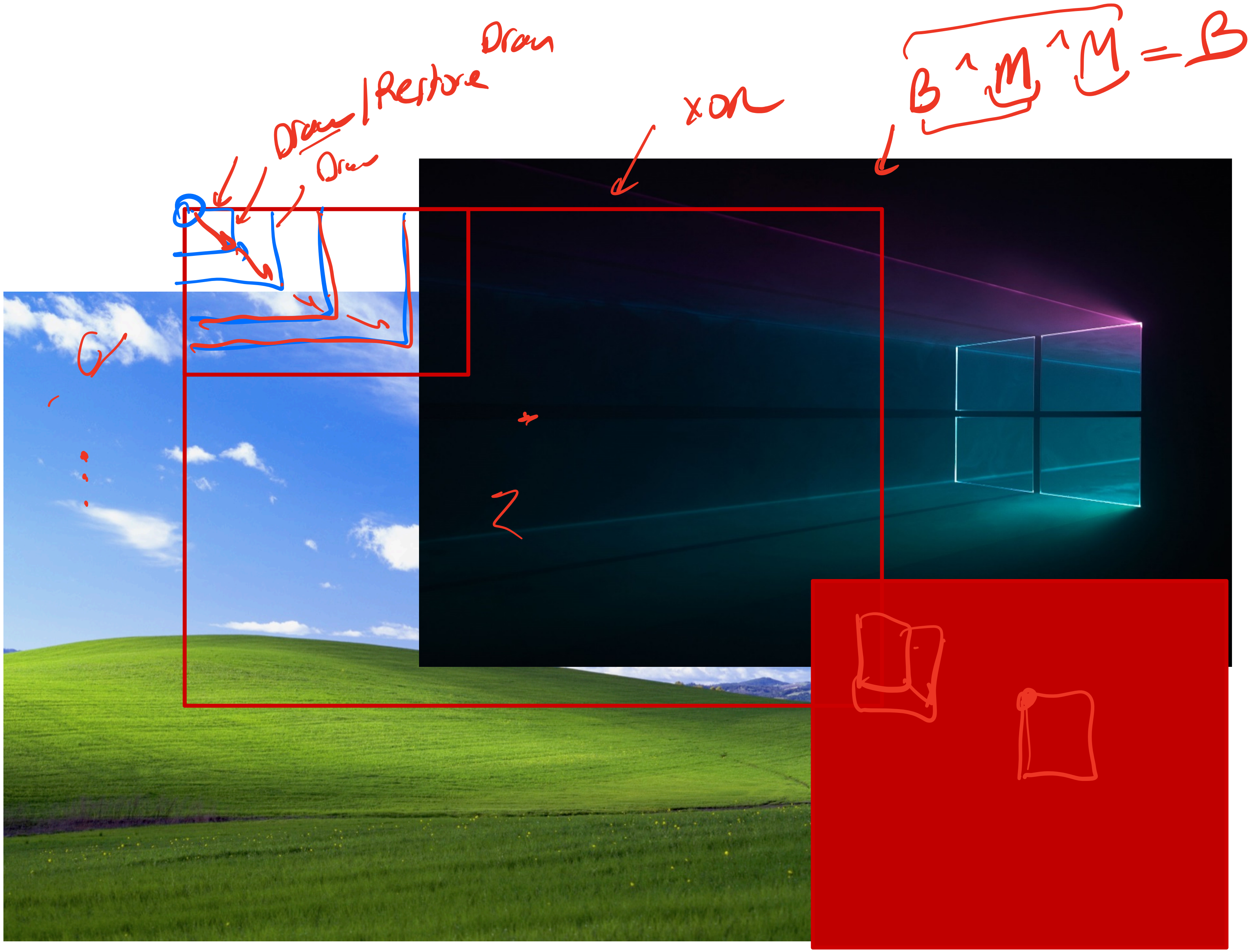
C, however, has only one right shift operator, >>. Many C compilers choose which right shift to perform depending on what type of integer is being shifted; often signed integers are shifted using the arithmetic shift, and unsigned integers are shifted using the logical shift.

Swap

```
void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y =  temp;
}
```

Challenge: Can you code swap function without using a temporary variable?

Hint: Use bitwise XOR (^)


Draw / Restore
Draw
Draw
xor
B ^ M ^ M = B
Z

Today: Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- **Integers**
  - **Representation: unsigned and signed**
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings
- Summary

Encoding Integers

4 byte = $x =$ 0100 1010 0011 1101 ← Most Significant Bit MSB

$w = 32$, x_i, x_1, x_0

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

```
short int x =  15213;
short int y = -15213;
```

$$\text{Bits2Unsigned}(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

$$B2U(X) \geq 0$$

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

$\emptyset$ ≥ 0 < 0

Encoding Integers

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

```
short int x =  15213;
short int y = -15213;
```

- **C `short` 2 bytes long**

	Decimal	Hex	Binary	
x	15213	3B 6D	00111011 01101101	x
y	-15213	C4 93	11000100 10010011	~x+1

- **Sign Bit**
  - For 2's complement, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative

Two-complement Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

For short $w = 16$

Weight	15213		-15213	
2^0 1	1 x_0	1	1	1
2^1 2	0 x_1	0	1	2
2^2 4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
2^{w-1} $\boxed{-32768}$	0	0	1	-32768
Sum		**15213**		**-15213**

Numeric Ranges

Think Binary

$B2U_w(x) = \sum_{i=0}^{w-1} x_i \cdot 2^i$

- **Unsigned Values**
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1

$x = \overbrace{11\cdots1}^{w} = \sum_{i=0}^{w-1} 1 \cdot 2^i$

$+\ 1 = 0\cdots0001$

$x + 1 = \underbrace{10\cdots000}_{w+1} = 1 \cdot 2^w$

$\Rightarrow x = 2^w - 1$

$w = 16 \quad UMax_{short} = 2^{16} - 1$

Numeric Ranges

$$B2T_w(X) = \underbrace{-x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i}$$

- **Two's Complement Values**
  - *TMin* = -2^{w-1}
    100…0
  - *TMax* = $2^{w-1} - 1$
    011…1
- **Other Values**
  - Minus 1
    111…1

$TMin = \overset{w}{1}00\ldots0$

$= -2^{w-1}$

$TMax = 011\ldots1$

$= \sum_{i=0}^{w-2} 2^i$

$= 2^{w-1} - 1$

$$X = \underbrace{\underbrace{1}_{-2^{w-1}} + \underbrace{1\ldots1}_{(2^{w-1}-1)}}_{= -1}$$

$-1 \equiv 1\ldots1$