BITS, BYTES, AND INTEGERS

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

Adapted from Bryant and O'Hallaron,
Computer Systems:
A Programmer's Perspective, Third Edition

TYPES IN C : POINTERS

WORK IT OUT

- Create a program that will print out numbers from 1 to 100 backwards

- Create a program that
  - will create an array of integers with 100 elements in it
  - set each of the elements to twice its index value.
  - output the address of each element of the array

- Create a function that doubles the value of its argument. Use pointers to make the changes persist even after the function call is complete.

TYPES IN C : POINTERS

WORK IT OUT

- Create a program that will print out numbers from 1 to 100 backwards

```c
#include <stdio.h>

int main(){

    for(int i = 100; i >=1 ; i--){
        printf("%d\n",i);
    }

    return 0;
}
```

TYPES IN C : POINTERS

WORK IT OUT

- Create a program that will print out numbers from 1 to 100 backwards

```c
#include <stdio.h>

int main(){

    for(int i = 100; i >=1 ; i--){
        printf("%d\n",i);
    }

    return 0;
}
```

```
$ ./workItOut_01 | tail
10
9
8
7
6
5
4
3
2
1
```

TYPES IN C : POINTERS

WORK IT OUT

- Create a program that
  - will create an array of integers with 100 elements in it
  - set each of the elements to twice its index value.
  - output the address of each element of the array

```c
#include <stdio.h>

int main(){

    int a[100];

    for(int i = 0; i < 100 ; i++){
        a[i] = 2 * i;
    }

    for(int i = 0; i < 100 ; i++){
        printf("%p\n",&(a[i]));
    }

    return 0;
}
```

TYPES IN C : POINTERS

WORK IT OUT

- Create a program that
  - will create an array of integers with 100 elements in it
  - set each of the elements to twice its index value.
  - output the address of each element of the array

```c
#include <stdio.h>

int main(){

    int a[100];

    for(int i = 0; i < 100 ; i++){
        a[i] = 2 * i;
    }

    for(int i = 0; i < 100 ; i++){
        printf("%p\n",&(a[i]));
    }

    return 0;
}
```

```
$ ./workItOut_02 | tail
0x7fff5bad1318
0x7fff5bad131c
0x7fff5bad1320
0x7fff5bad1324
0x7fff5bad1328
0x7fff5bad132c
0x7fff5bad1330
0x7fff5bad1334
0x7fff5bad1338
0x7fff5bad133c
```

TYPES IN C : POINTERS

WORK IT OUT

```c
#include <stdio.h>

void doubleItNoChange(int x){
    x = x * 2;
}

void doubleItChange(int *x){
    *x = *x * 2;
}

int main(){
    int y = 5;
    printf("y starts as %d\n",y);
    doubleItNoChange(y);
    printf("then y is %d\n",y);
    doubleItChange(&y);
    printf("then y is %d\n",y);
    return 0;
}
```

- Create a function that doubles the value of its argument. Use pointers to make the changes persist even after the function call is complete.

TYPES IN C : POINTERS

WORK IT OUT

- Create a function that doubles the value of its argument. Use pointers to make the changes persist even after the function call is complete.

```c
#include <stdio.h>

void doubleItNoChange(int x){
    x = x * 2;
}

void doubleItChange(int *x){
    *x = *x * 2;
}

int main(){
    int y = 5;
    printf("y starts as %d\n",y);
    doubleItNoChange(y);
    printf("then y is %d\n",y);
    doubleItChange(&y);
    printf("then y is %d\n",y);
    return 0;
}
```

```
$ ./workItOut_03
y starts as 5
then y is 5
then y is 10
```

0 1

BITS, BYTES, INTEGERS

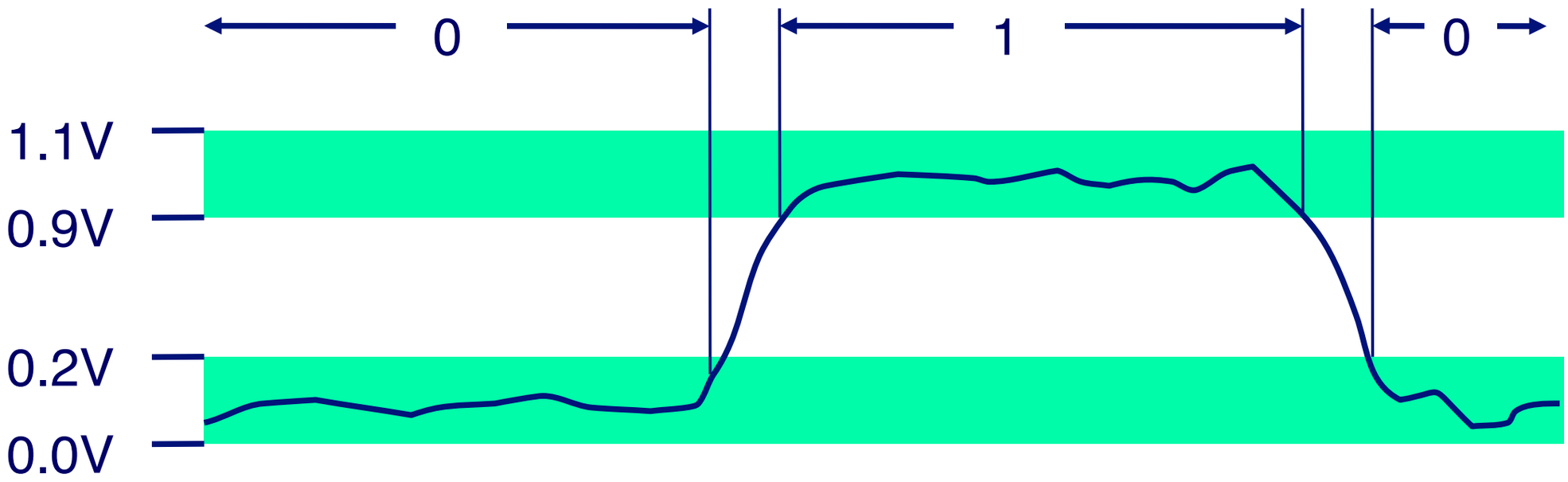
BASICS

- bit
- byte
- bytes that are interpreted becomes information
  - instructions
    - machine language
- data “types”
  - unsigned integers
  - signed integers (two’s-complement)
  - floating point
  - characters

BITS, BYTES, INTEGERS

BASICS

- Why bits?
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



BITS, BYTES, INTEGERS

BASICS

- bytes are limited so computations can have errors
  - overflow

```c
#include <stdio.h>

int main(){


    printf("%d\n", 200*300*400*500);
    printf("%d\n", (200*300)*(400*500));
    printf("%d\n", 200*(300*400)*500);


    return 0;
}
```

BITS, BYTES, INTEGERS

BASICS

- bytes are limited so computations can have errors
  - overflow

```c
#include <stdio.h>

int main(){


    printf("%d\n", 200*300*400*500);
    printf("%d\n", (200*300)*(400*500));
    printf("%d\n", 200*(300*400)*500);


    return 0;
}
```

```
-884901888
-884901888
-884901888
```

BITS, BYTES, INTEGERS

BASICS

- bytes are limited so computations can have errors
  - underflow

```c
#include <stdio.h>

int main(){
    double x = 10.0;
    double y = 1E-318;

    printf("%e\n", y);

    double z = y/x;
    printf("%e\n", z);

    z = z/x;
    printf("%e\n", z);

    z = z/x;
    printf("%e\n", z);

    z = z/x;
    printf("%e\n", z);

    z = z/x;
    printf("%e\n", z);

    z = z/x;
    printf("%e\n", z);

    return 0;
}
```

$$1\text{E-}318 = 1 * 10^{-318} = \frac{1}{10^{318}}$$

```
9.999987e-319
9.999889e-320
9.999889e-321
9.980126e-322
9.881313e-323
9.881313e-324
0.000000e+00
```

BITS, BYTES, INTEGERS

BINARY

- Despite limitations we can count in binary just like we can in decimal
  - We can represent
    - 15213_{10} as 11101101101101_2
    - 1.20_{10} as $1.00110011001100110011[0011]..._2$
    - $1.5213 \text{ X } 10^4$ as $1.1101101101101_2 \text{ X } 2^{13}$

BITS, BYTES, INTEGERS

BINARY

- 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 FA1D37B16 in C as
      - 0xFA1D37B
      - 0xfa1d37b

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

WORK IT OUT

CONVERSIONS

- What is 0x39A7F8 in binary
- What is 1100100101111011 in hexadecimal
- What is 0xD5E4C in binary
- What is 10011011100111110110101

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

WORK IT OUT

CONVERSIONS

- What are the following in hexadecimal
  - $0000\ 0000_2$
  - $0001\ 0000_2$
  - $0001\ 0000_2$
  - $0101\ 0101_2$
  - $1001\ 1001_2$
  - $0001\ 0001_2$
  - $0010\ 0010_2$
  - $0100\ 0100_2$
  - $1000\ 1000_2$

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

BITS, BYTES, INTEGERS

TYPICAL SIZES VARY ON DIFFERENT MACHINES

- Data representations in bytes

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	4	8	8

BITS, BYTES, INTEGERS

{BIG, LITTLE} ENDIAN

- Data representations in bytes
- Suppose
  - `int *p = 0x100;`
  - `(*p) = 0x01234567;`

Big endian

	0x100	0x101	0x102	0x103	
...	01	23	45	67	...

Little endian

	0x100	0x101	0x102	0x103	
...	67	45	23	01	...

BITS, BYTES, INTEGERS

{BIG, LITTLE} ENDIAN

- So which format does your computer use?
    - Write a program which will show you the answer

BITS, BYTES, INTEGERS

{BIG, LITTLE} ENDIAN

```c
#include <stdio.h>

int main(){

    int test = 0x01234567;

    int *test_pointer = &test;

    char *mystery_pointer = (char *)test_pointer;

    if( (*mystery_pointer) == 0x01){
        printf("Big Endian\n");
    }
    else{
        printf("Little Endian\n");
    }

    printf("First Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Second Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Third Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Fourth Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));

    return 0;
}
```

BITS, BYTES, INTEGERS

{BIG, LITTLE} ENDIAN

```c
#include <stdio.h>

int main(){

    int test = 0x01234567;

    int *test_pointer = &test;

    char *mystery_pointer = (char *)test_pointer;

    if( (*mystery_pointer) == 0x01){
        printf("Big Endian\n");
    }
    else{
        printf("Little Endian\n");
    }

    printf("First Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Second Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Third Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));
    mystery_pointer++;
    printf("Fourth Byte at \t %p is %.2x\n", mystery_pointer,(*mystery_pointer));

    return 0;
}
```

```
Little Endian
First Byte at     0x7fff5e7b8358 is 67
Second Byte at    0x7fff5e7b8359 is 45
Third Byte at     0x7fff5e7b835a is 23
Fourth Byte at    0x7fff5e7b835b is 01
```

BITS, BYTES, INTEGERS

{BIG, LITTLE} ENDIAN

Machine	Value	Type	Bytes (hex)
Linux 32	12,345	`int`	39 30 00 00
Windows	12,345	`int`	39 30 00 00
Sun	12,345	`int`	00 00 30 39
Linux 64	12,345	`int`	39 30 00 00
Linux 32	12,345.0	`float`	00 e4 40 46
Windows	12,345.0	`float`	00 e4 40 46
Sun	12,345.0	`float`	46 40 e4 00
Linux 64	12,345.0	`float`	00 e4 40 46
Linux 32	`&ival`	`int *`	e4 f9 ff bf
Windows	`&ival`	`int *`	b4 cc 22 00
Sun	`&ival`	`int *`	ef ff fa 0c
Linux 64	`&ival`	`int *`	b8 11 e5 ff ff 7f 00 00

BITS, BYTES, INTEGERS

WHAT ABOUT CHARACTERS?

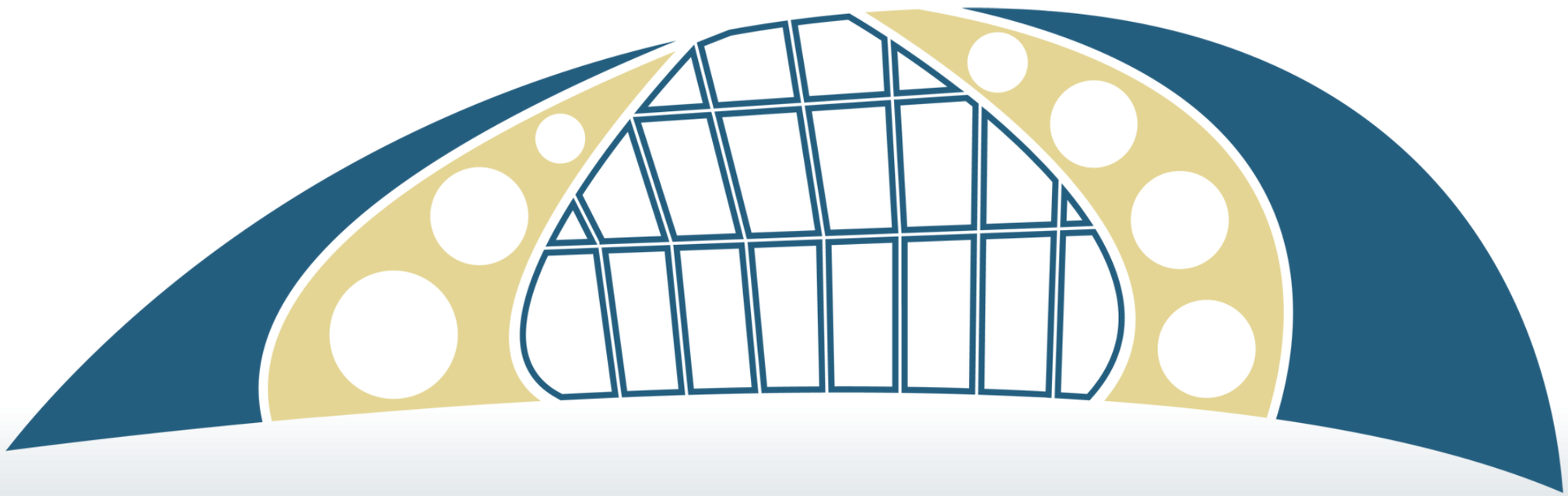
```c
#include <stdio.h>
int main(){
    char test[] = "01234567";
    char *test_pointer = test;

    printf("First Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Second Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Third Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Fourth Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));

    return 0;
}
```

BITS, BYTES, INTEGERS

WHAT ABOUT CHARACTERS?

```
First Byte at    0x7fff5able34f is 0x30
Second Byte at   0x7fff5able350 is 0x31
Third Byte at    0x7fff5able351 is 0x32
Fourth Byte at   0x7fff5able352 is 0x33
```

```c
#include <stdio.h>

int main(){

    char test[] = "01234567";

    char *test_pointer = test;

    printf("First Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Second Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Third Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));
    test_pointer++;
    printf("Fourth Byte at \t %p is 0x%.2x\n", test_pointer,(*test_pointer));

    return 0;
}
```



WESTMONT INSPIRED

COMPUTING LAB