CS 110
Computer Architecture

Lecture 2: *Introduction to C, Part I*

Instructor:
Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

- Everything is a Number
- Compile vs. Interpret
- Administrivia
- Quick Start Introduction to C
- Pointers
- And in Conclusion, …

Agenda

- Everything is a Number
- Compile vs. Interpret
- Administrivia
- Quick Start Introduction to C
- Pointers
- And in Conclusion, …

BIG IDEA: Bits can represent anything!!

- Characters?
  - 26 letters $\Rightarrow$ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation
    $\Rightarrow$ 7 bits (in 8) ("ASCII")
  - standard code to cover all the world's languages $\Rightarrow$ 8,16,32 bits ("Unicode")
    www.unicode.com

- Logical values?
  - 0 $\rightarrow$ False, 1 $\rightarrow$ True
- colors ? Ex: ***Red (00)*** ***Green (01)*** ***Blue (11)***
- locations / addresses? commands?
- MEMORIZE: N bits $\Leftrightarrow$ at most 2^N things

Key Concepts

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, …
- Integer and floating-point operations can lead to results too big/small to store within their representations: *overflow*/*underflow*

Number Representation

- Value of i-th digit is $d \times Base^{i}$ where i starts at 0 and increases from right to left:
- $123_{10} = 1_{10} \times 10_{10}^{2} + 2_{10} \times 10_{10}^{1} + 3_{10} \times 10_{10}^{0}$

  $= 1x100_{10} + 2x10_{10} + 3x1_{10}$

  $= 100_{10} + 20_{10} + 3_{10}$

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We use 1_{two}, 5_{ten}, 10_{hex} to be clearer (vs. 1_{2}, 4_{8}, 5_{10}, 10_{16})

Number Representation

- Hexadecimal digits:
  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- $FFF_{hex} = 15_{ten} \times 16_{ten}^{2} + 15_{ten} \times 16_{ten}^{1} + 15_{ten} \times 16_{ten}^{0}$
  $= 3840_{ten} + 240_{ten} + 15_{ten}$
  $= 4095_{ten}$
- $1111\ 1111\ 1111_{two} = FFF_{hex} = 4095_{ten}$
- May put blanks every group of binary, octal, or hexadecimal digits to make it easier to parse, like commas in decimal

Signed and Unsigned Integers

- C, C++, and Java have *signed integers*, e.g., 7, -255:

```
int x, y, z;
```

- C, C++ also have *unsigned integers*, e.g. for addresses
- 32-bit word can represent 2^{32} binary numbers
- Unsigned integers in 32 bit word represent 0 to 2^{32}-1 (4,294,967,295) (4 Gig)

Unsigned Integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
... ...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 2{,}147{,}483{,}649_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2{,}147{,}483{,}650_{ten}$
... ...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

Signed Integers and Two's-Complement Representation

- Signed integers in C; want ½ numbers <0, want ½ numbers >0, and want one 0
- *Two's complement* treats 0 as positive, so 32-bit word represents 2^{32} integers from -2^{31} (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - Book lists some other options, all of which are worse
  - Every computer uses two's complement today
- *Most-significant bit* (leftmost) is the *sign bit*, since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant, bit 0 is least significant

Two's-Complement Integers

Sign Bit

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
... ...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2{,}147{,}483{,}646_{ten}$
... ...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Ways to Make Two's Complement

- For N-bit word, complement to 2_{ten}^{N}
  - For 4 bit number $3_{ten}=0011_{two}$, two's complement

    (i.e. -3_{ten}) would be

    $16_{ten}-3_{ten}=13_{ten}$ or $10000_{two} - 0011_{two} = 1101_{two}$
- Here is an easier way:
  - Invert all bits and add 1
  - Computers actually do it like this, too

3_{ten} 0011_{two}

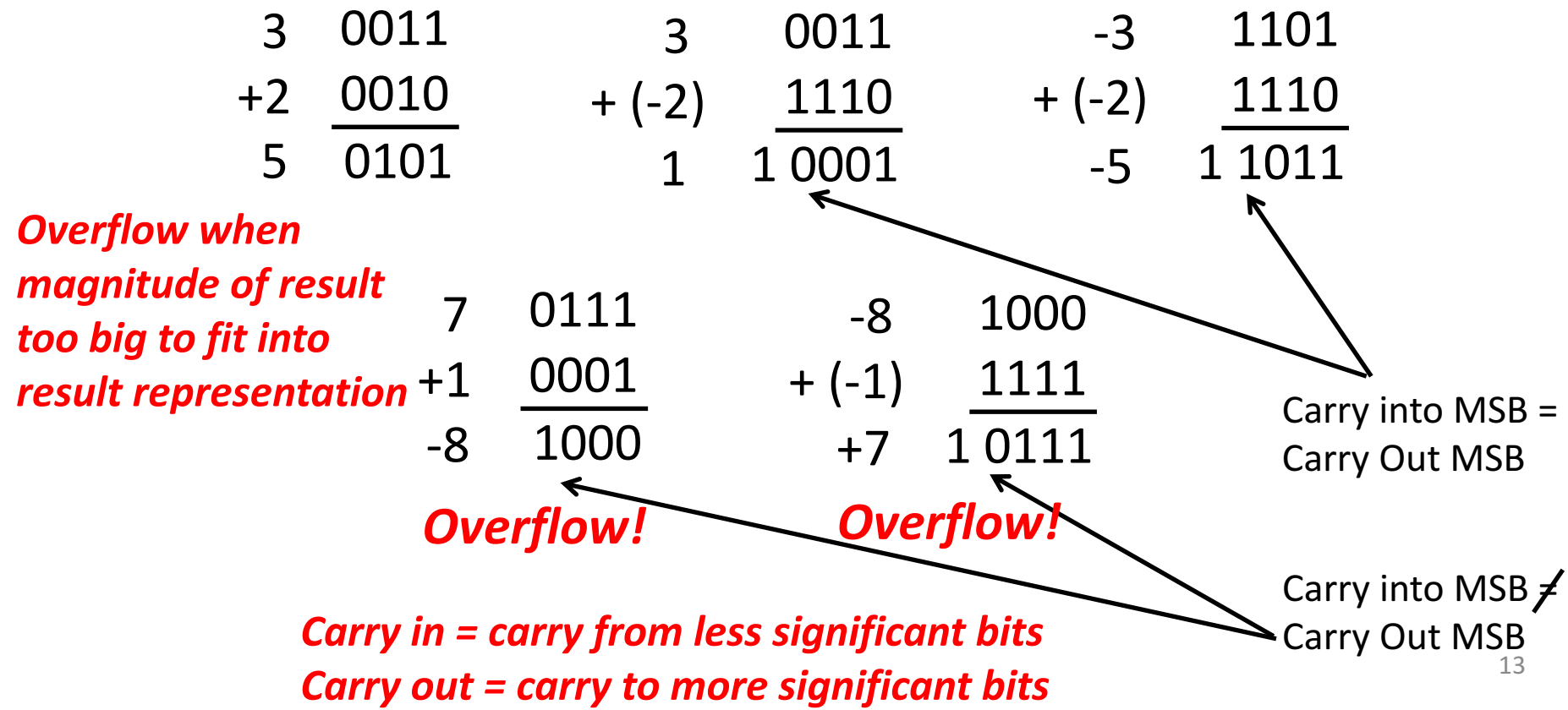
Bitwise complement 1100_{two}

$+ \quad 1_{two}$

-3_{ten} 1101_{two}

Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented



Suppose we had a 5-bit word. What integers can be represented in two's complement?

A -32 to +31

B 0 to +31

C -16 to +15

D -15 to +16

Suppose we had a 5-bit word. What integers can be represented in two's complement?

A -32 to +31

B 0 to +31

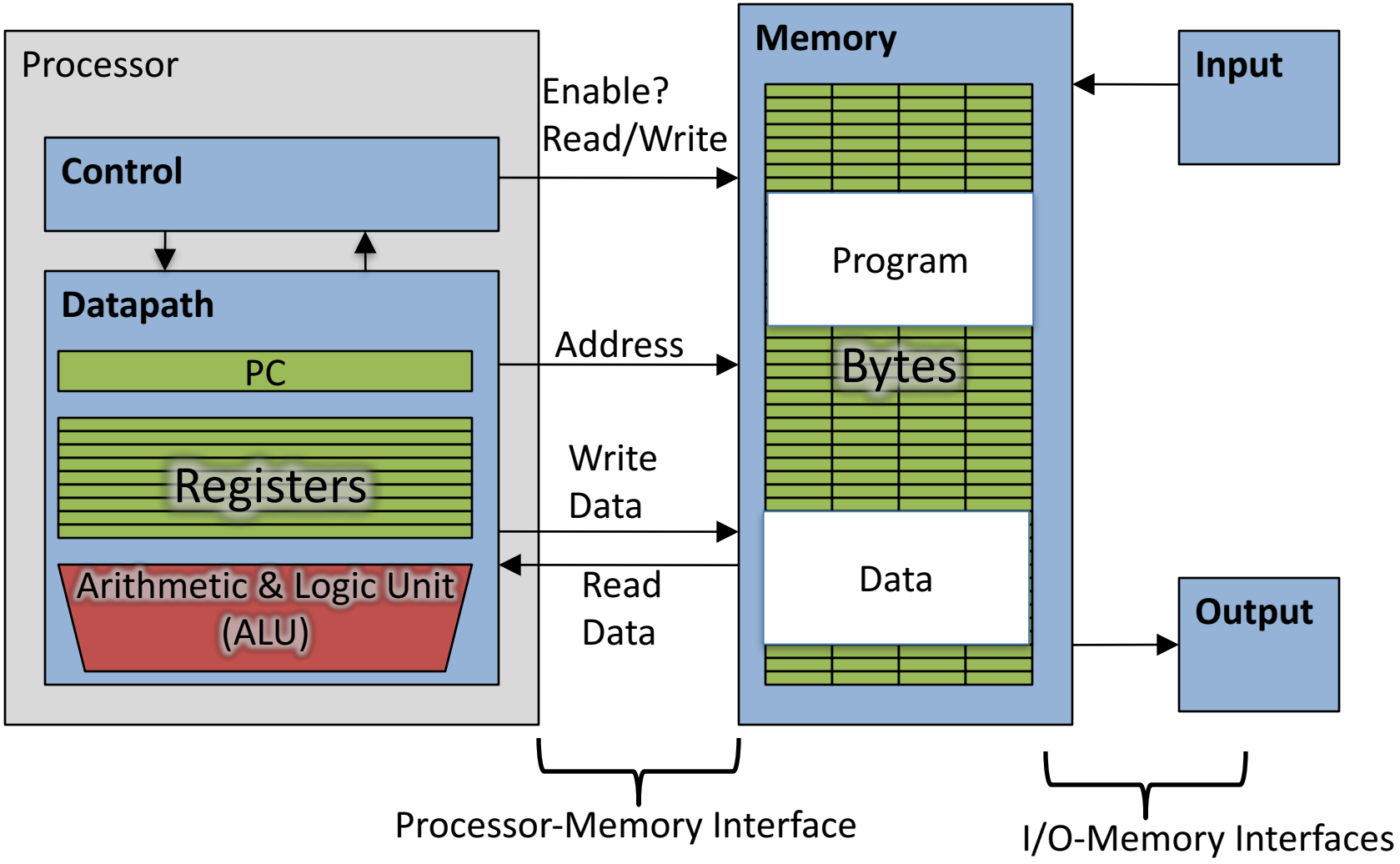
C -16 to +15

D -15 to +16

Agenda

- Everything is a Number
- Compile vs. Interpret
- Administrivia
- Quick Start Introduction to C
- Pointers
- And in Conclusion, …

Components of a Computer



Great Idea: Levels of Representation/Interpretation

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

We are here!

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw    t0, 0(s2)
lw    t1, 4(s2)
sw    t1, 0(s2)
sw    t0, 4(s2)
```

Assembler

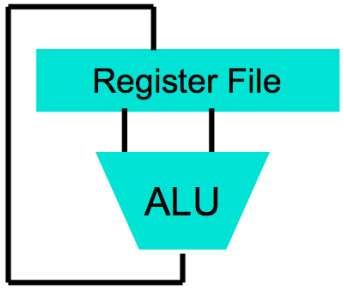
Machine Language Program (RISC-V)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Anything can be represented as a *number*, i.e., data or instructions

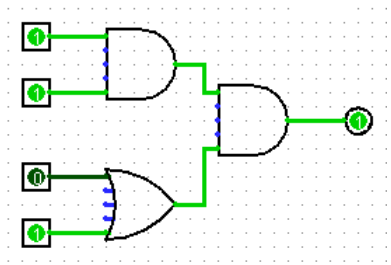
Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

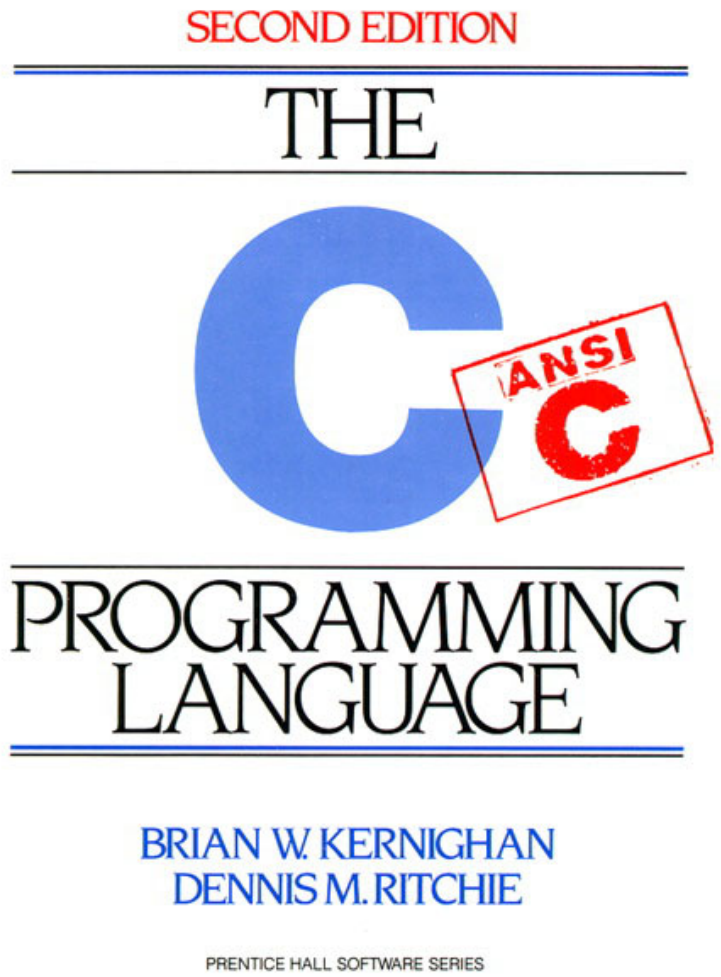


Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



Introduction to C
“The Universal Assembly Language”



Intro to C

- *C is not a "very high-level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*

  – Kernighan and Ritchie

- Enabled first operating system not written in assembly language: *UNIX* - A portable OS!

Intro to C

- Why C?: *we can write programs that allow us to exploit underlying features of the architecture – memory management, special instructions, parallelism*
- C and derivatives (C++/Obj-C/C#) still one of the most popular application programming languages after >40 years!

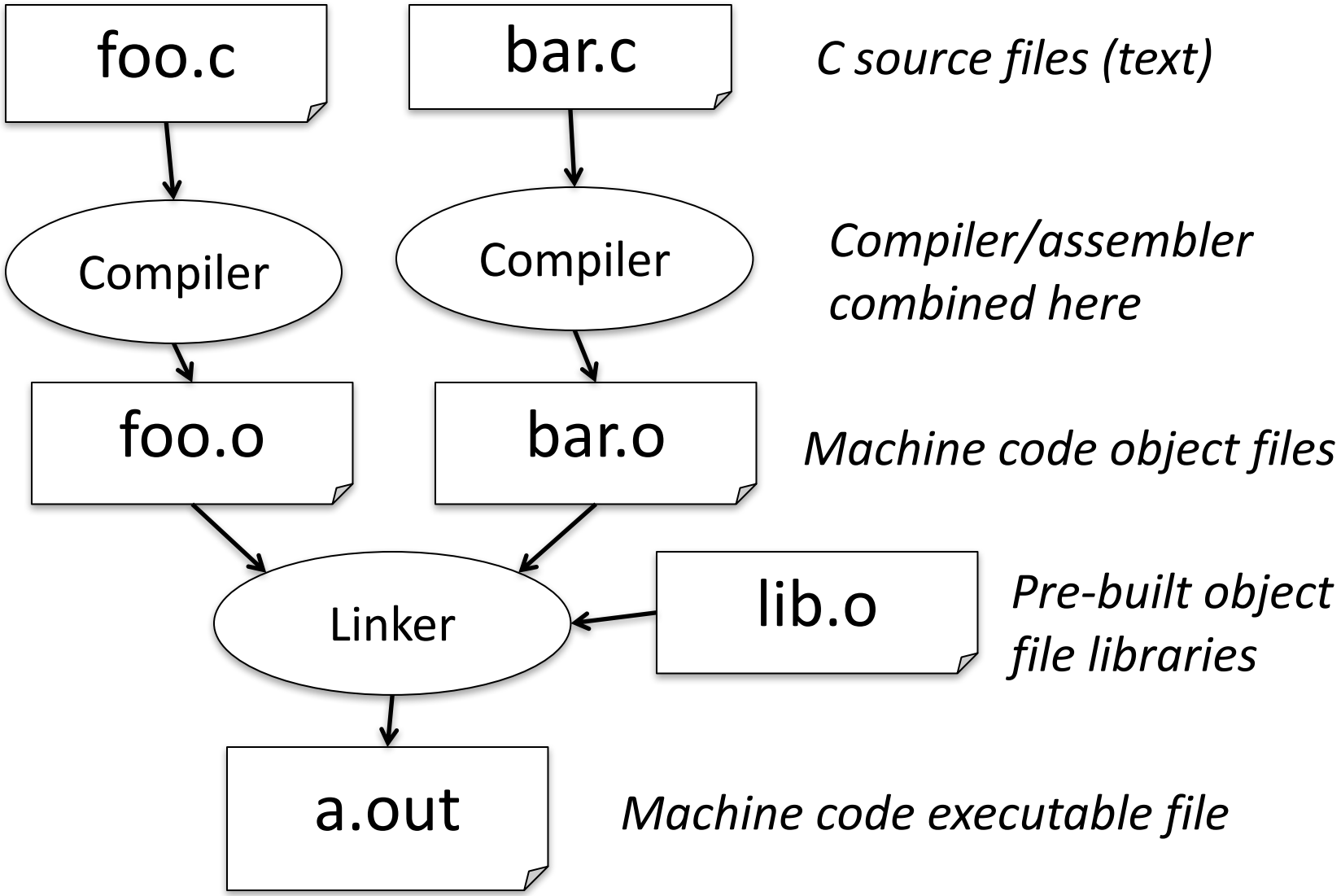
Disclaimer

- You will not learn how to fully code in C in these lectures! You'll still need your C reference for this course
  - K&R is a must-have
    - Check online for more sources
- Key C concepts: Pointers, Arrays, Implications for Memory management
- We will use ANSI C89 – original "old school" C
  - Because it is closest to Assembly

Compilation: Overview

- C *compilers* map C programs into architecture-specific machine code (string of 1s and 0s)
  - Unlike *Java*, which converts to architecture-independent *bytecode*
  - Unlike *Python* environments, which *interpret* the code
  - These differ mainly in exactly when your program is converted to low-level machine instructions ("levels of interpretation")
  - For C, generally a two part process of compiling .c files to .o files, then linking the .o files into executables;
  - Assembling is also done (but is hidden, i.e., done automatically, by default); we'll talk about that later

C Compilation Simplified Overview (more later in course)



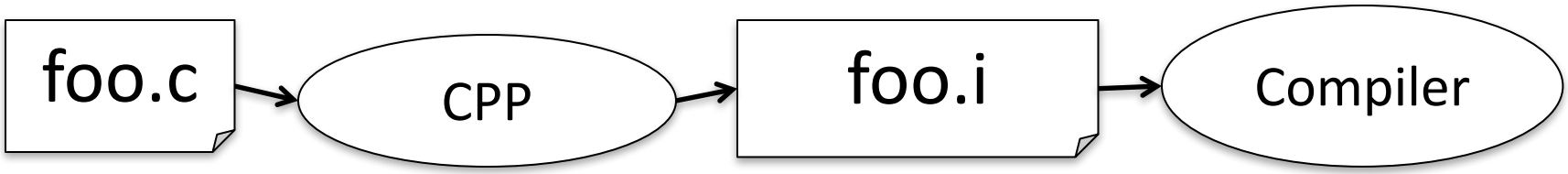
Compilation: Advantages

- Excellent run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- Reasonable compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled

Compilation: Disadvantages

- Compiled files, including the executable, are architecture-specific, depending on processor type (e.g., MIPS vs. RISC-V) and the operating system (e.g., Windows vs. Linux)
- Executable must be rebuilt on each new system
  - I.e., "porting your code" to a new architecture
- "Change → Compile → Run [repeat]" iteration cycle can be slow during development
  - but Make tool only rebuilds changed pieces, and can do compiles in parallel (linker is sequential though -> Amdahl's Law)

C Pre-Processor (CPP)



- C source files first pass through macro processor, CPP, before compiler sees code
- CPP replaces comments with a single space
- CPP commands begin with “#”
- #include “file.h” /* Inserts file.h into output */
- #include <stdio.h> /* Looks for file in standard location */
- #define M_PI (3.14159) /* Define constant */
- #if/#endif /* Conditional inclusion of text */
- Use -save-temps option to gcc to see result of preprocessing
- Full documentation at: **http://gcc.gnu.org/onlinedocs/cpp/**

Typed Variables in C

```
int   variable1   = 2;
float variable2   = 1.618;
char  variable3   = 'A';
```

- Must declare the type of data a variable will hold
  - Types can't change

Type	Description	Examples
int	integer numbers, including negatives	0, 78, -1400
unsigned int	integer numbers (no negatives)	0, 46, 900
long	larger signed integer	-6,000,000,000
char	single text character or symbol	'a', 'D', '?'
float	floating point decimal numbers	0.0, 1.618, -1.4
double	greater precision/big FP number	10E100

Integers: Python vs. Java vs. C

Language	sizeof(int)
Python	>=32 bits (plain ints), infinite (long ints)
Java	32 bits
C	Depends on computer; 16 or 32 or 64

- C: `int` should be integer type that target processor works with most efficiently
- Only guarantee: sizeof(`long long`) ≥ sizeof(`long`) ≥ sizeof(`int`) ≥ sizeof(`short`)
  - Also, `short` >= 16 bits, `long` >= 32 bits
  - All could be 64 bits

Consts and Enums in C

- Constant is assigned a typed value once in the declaration; value can't change during entire execution of program

```
const float golden_ratio = 1.618;
const int days_in_week = 7;
```

- You can have a constant version of any of the standard C variable types
- Enums: a group of related integer constants. Ex:

```
enum cardsuit {CLUBS,DIAMONDS,HEARTS,SPADES};
enum color {RED, GREEN, BLUE};
```

Compare "`#define PI 3.14`" and "`const float pi=3.14`" – which is true?

A: Constants "PI" and "pi" have same type

B: Can assign to "PI" but not "pi"

C: Code runs at same speed using "PI" or "pi"

D: "pi" takes more memory space than "PI"

E: Both behave the same in all situations

Agenda

- Everything is a Number
- Compile vs. Interpret
- Administrivia
- Quick Start Introduction to C
- Pointers
- And in Conclusion, …

Administrivia

- Find a partner for the lab. Inform your lab TA about your partner selection during lab 1. Partner teams should be 2 persons – there can be exactly one 3 person lab team per lab.
- Labs start next week! Check your schedule! You cannot get checked without a partner!
- The tasks for Lab 1 are posted on the website. Prepare for it over the weekend.

Administrivia

- We have registered you for Autolab yesterday night.
- HW1 is available on Autolab. Due next Thursday.
- Website is down. Hopefully will be up soon. Slides will be posted on piazza for now. Maybe Lab 1, too.
- Register in piazza! Will be part of your grade!
  - There are also apps for your phone…

Agenda

- Everything is a Number
- Compile vs. Interpret
- Administrivia
- Quick Start Introduction to C
- Pointers
- And in Conclusion, …

Typed Functions in C

```
int number_of_people ()
{
  return 3;
}

float dollars_and_cents ()
{
  return 10.33;
}

int sum ( int x, int y)
{
   return x + y;
}
```

- You have to *declare* the type of data you plan to return from a function
- Return type can be any C variable type, and is placed to the left of the function name
- You can also specify the return type as **`void`**
  - Just think of this as saying that no value will be returned
- Also necessary to declare types for values passed into a function
- Variables and functions MUST be declared before they are used

Structs in C

- Structs are structured groups of variables, e.g.,

```
typedef struct {
  int length_in_seconds;
  int year_recorded;
} Song;
```

Dot notation: **`x.y = value`**

```
Song song1;

song1.length_in_seconds =  213;
song1.year_recorded     = 1994;

Song song2;

song2.length_in_seconds =  248;
song2.year_recorded     = 1988;
```

A First C Program: Hello World

Original C:

```
main()
{
  printf("\nHello World\n");
}
```

ANSI Standard C:

```
#include <stdio.h>

int main(void)
{
  printf("\nHello World\n");
  return 0;
}
```

C Syntax: `main`

- When C program starts
  - C executable a.out is loaded into memory by operating system (OS)
  - OS sets up stack, then calls into C runtime library,
  - Runtime 1st initializes memory and other libraries,
  - then calls your procedure named main ()
- We'll see how to retrieve command-line arguments in main() later…

A Second C Program: Compute Table of Sines

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int    angle_degree;
    double angle_radian, pi, value;
    /* Print a header */
    printf("\nCompute a table of the
   sine function\n\n");

    /* obtain pi once for all       */
    /* or just use pi = M_PI, where */
    /* M_PI is defined in math.h    */
    pi = 4.0*atan(1.0);
    printf("Value of PI = %f \n\n",
   pi);
    printf("angle     Sine \n");

    angle_degree = 0;
    /* initial angle value */
    /* scan over angle     */
    while (angle_degree <= 360)
    /* loop until angle_degree > 360 */
       {
          angle_radian = pi*angle_degree/180.0;
          value = sin(angle_radian);
          printf (" %3d      %f \n ",
                    angle_degree, value);
          angle_degree = angle_degree + 10;
          /* increment the loop index */
       }
    return 0;
}
```

Second C Program Sample Output

```
Compute a table of the sine
   function

Value of PI = 3.141593

angle       Sine
   0      0.000000
  10      0.173648
  20      0.342020
  30      0.500000
  40      0.642788
  50      0.766044
  60      0.866025
  70      0.939693
  80      0.984808
  90      1.000000
 100      0.984808
 110      0.939693
 120      0.866025
 130      0.766044
 140      0.642788
 150      0.500000
 160      0.342020
 170      0.173648
 180      0.000000
 190     -0.173648
 200     -0.342020
 210     -0.500000
 220     -0.642788
 230     -0.766044
 240     -0.866025
 250     -0.939693
 260     -0.984808
 270     -1.000000
 280     -0.984808
 290     -0.939693
 300     -0.866025
 310     -0.766044
 320     -0.642788
 330     -0.500000
 340     -0.342020
 350     -0.173648
 360     -0.000000
```

C Syntax: Variable Declarations

- *All* variable declarations must appear before they are used (e.g., at the beginning of the block)
- A variable may be initialized in its declaration; if not, it holds garbage!
- Examples of declarations:
  - Correct:
```
{
    int a = 0, b = 10;
    ...
```
  - Incorrect:
```
    for (int i = 0; i < 10; i++)
}
```

Newer C standards are more flexible about this, more later

C Syntax : Control Flow (1/2)

- Within a function, remarkably close to Java constructs in terms of control flow
  - **`if-else`**
    - **`if (expression) statement`**
    - **`if (expression) statement1 else statement2`**
  - **`while`**
    - **`while (expression) statement`**
    - **`do statement while (expression);`**

C Syntax : Control Flow (2/2)

- **for**
  - **for (initialize; check; update) statement**
- **switch**
  - **switch (expression){**

```
case const1:    statements
case const2:    statements
default:        statements
}
```

  - **break**