What is a Computer Program?

- Description of algorithms and data structures to achieve a specific objective
- Could be done in any language, even a natural language like English
- Programming language: A Standard notation for writing programs
- Examples: C, Java, Intel assembly language
- An extreme example: Machine language
- Hence the need for program translators
  - Example: gcc

6

High level vs Low Level

English:

"To calculate the simple interest, multiply the principal amount by the rate of interest by the number of years"

C:

```
W = X * Y * Z;
```

Even better C statement:

```
SimpleInterest = Principal * Rate * Years;
```

High level vs Low Level …

Machine language:
(transformed to a human readable form)

```
movss   x(%rip), %xmm1
movss   y(%rip), %xmm0
mulss   %xmm0, %xmm1
movss   z(%rip), %xmm0
mulss   %xmm1, %xmm0
movss   %xmm0, w(%rip)
```

C program is made up of statements
Machine language program is made up of machine instructions

What does gcc do for you?

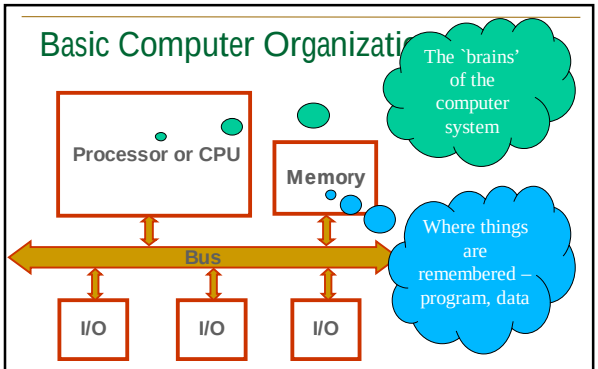
% gcc hello.c

"source": Program that you wrote in the C language and typed into the file hello.c

"executable": File generated by gcc. The a.out file contains an equivalent program in machine language that can be executed on a computer system

9

Basic Computer Organizati



10

Program = Instructions + Data

- We will next learn more about data

11

There are different kinds of data

How does one piece of data differ from another?

- Constant vs Variable
- Basic vs Structured
- Of different types
  - Character
  - Integer (unsigned, signed)
  - Real
  - Others (boolean, complex, …)

12

Data differing in their lifetimes

- Lifetime: Interval between time of creation and end of existence
- How long can the lifetime of a datum be?
- We will consider 3 possible lifetimes

13

Program Data: Different Lifetimes.

1. Lifetime = Execution time of program
   - Initialized/uninitialized data
   - Must be indicated in executable file
   - The space in memory for all of this data can be assigned when program execution starts (Static Allocation)

14

Program Data: Different Lifetimes.

1. Lifetime = Execution time of program
2. Lifetime = Time between explicit creation of data & explicit deletion of data
   - Dynamic memory allocation
   - In C you create new data using a function like malloc()
   - The memory space for this data is managed dynamically when the malloc/free is executed (Heap allocation)

15

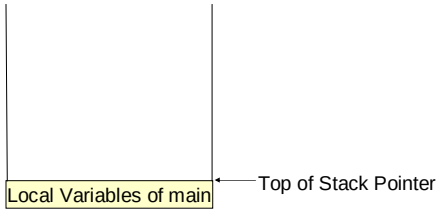
Program Data: Different Lifetimes..

1. Lifetime = Execution time of program
2. Lifetime = Time between explicit creation of data & explicit deletion of data
3. Lifetime = During execution of a function (i.e., time between function call and return)
   - Local variables, parameters of the function
   - The memory space for this data is assigned when the function is called and reclaimed on return from the function (Stack allocation)
   - Stack: Like a pile of books on a table

16

Stack allocated: Function Local Variables

When the program starts executing

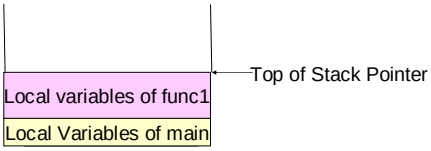


What if main() then calls function func1()?

17

Stack allocated: Function Local Variables.

While executing in function func1()

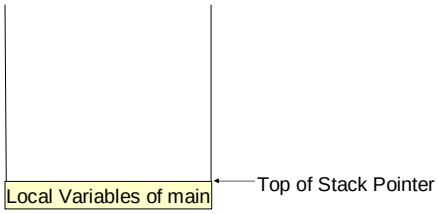


What happens on return from the call to func1()?

18

Stack allocated: Function Local Variables..

Executing in main() once again



19

Recursion vs Iteration

Example: Compute N! , N > 0

```
{       int N, fact, j;
        for (fact=1, j=1; j <= N; j++)
                fact = fact * j;
}
```

```
int factorial (int j)
{       if (j == 0) return(1);
        return (j * factorial(j-1));
}
```

```
void main()
{       int N;
        factorial(N);
}
```

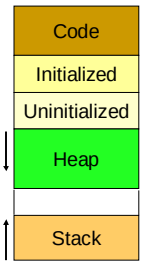
j ☐
fact ☐
N 100

J=0
J=1
• • •
J=98
J=99
J=100

N 100

During program execution

Code (machine language program)
Data (initialized and uninitialized)
Code and Data don't change in size while the program is executing
Heap (for dynamically allocated data)
Stack (for function local variables)
Heap and Stack change in size as program executes



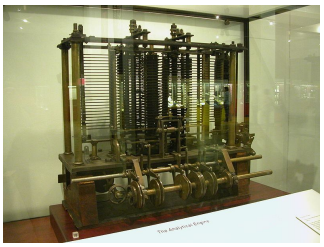
21

Digital Computers

As opposed to mechanical computers or analog computers

22

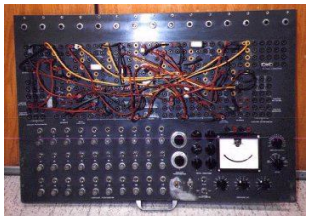
Babbage's Analytical Engine



Mechanical computers used mechanical parts like gears and levers to calculate

wikipedia

Analog Computer



Analog computers used continuously varying quantities like voltage

In digital computers, discretely varying values are used instead

http://userwww.sfsu.edu/~hl/c.heath.analog.html

Digital Computers..

In current digital computers, values vary between 2 discrete values

0 and 1

high and low

0 Volts and 3 Volts

25

Digital Circuits

Electrical circuits in which voltages only take on a discrete number of values

There are digital circuits that can do calculations, others that can be used to remember values, etc

26

Example: Binary Addition

- Binary: Base 2 number system
- You can generalize the decimal number system, which uses 10 digits (0,1,…,9) to work with any radix or base

$$(d_{n-1}d_{n-2}...d_2d_1d_0)_{ten} = d_{n-1}\times 10^{n-1} + d_{n-2}\times 10^{n-2} + ... + d_2\times 10^2 + d_1\times 10^1 + d_0\times 10^0$$
$$= \sum_{i=0}^{i=n-1} d_i \times 10^i$$

- Binary digits (called bits): 0 and 1

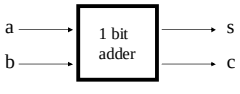
$$(b_{n-1}b_{n-2}...b_2b_1b_0)_{two} = b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + ... + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0$$
$$= \sum_{i=0}^{i=n-1} b_i \times 2^i$$

- Example: Decimal 47 is Binary 101111

Binary Addition .. 1 bit adder

- A circuit that can add one bit to another bit
- The different cases

0 + 0	= 0	= 0 0
0 + 1	= 1	= 0 1
1 + 0	= 1	= 0 1
1 + 1	???	= 1 0

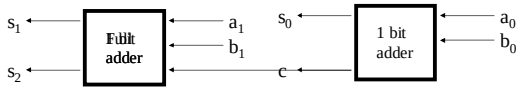


One bit adder has 2 inputs and 2 outputs

Binary Addition .. 2 bit adder

A circuit that can add one 2 bit value (a_1a_0) to another 2 bit value (b_1b_0)

$$\begin{array}{r} a_1\ a_0 \\ +\ b_1\ b_0 \\ \hline s_2\ s_1\ s_0 \end{array}$$



This idea can be extended to design a 32 bit adder or a 64 bit adder

How is Data Represented?

- On a digital computer
- Binary
  - Base 2 number system
  - Two values: 0 and 1
  - Bit (Notation: b); Byte (Notation: B) 8 bits
  - Other notation: K, M, G, T, P etc
    - K: 2^{10}, M: 2^{20}, G: 2^{30}, etc
    - 1G = 1,073,741,824
    - "2 GB of RAM", "1 TB hard disk drive"

30

Character Data

- Typically represented using the ASCII code
- ASCII: American Standard Code for Information Interchange
- Each character is represented by a unique 8 bit ASCII code word
- Example: 'a' is represented by 01100001, '1' is represented by 00110001

31

How is Data Represented?

- Character data: ASCII code
- Integer data
  - In computer systems, you usually find support for both "signed integers" and "unsigned integers"
    - e.g., C programming

      int x; Can take +ve or -ve whole number values

      unsigned int y; Can take on +ve whole number values

32

Unsigned Integer Data

- Representation: Binary number system

$$(b_{n-1}b_{n-2}..b_2b_1b_0)_{two} = b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + ... + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0$$

$$= \sum_{i=0}^{i=n-1} b_i \times 2^i$$

- e.g., Decimal 1000 is represented as 1111101000
  - i.e. 0000001111101000 in 16 bits
  - 00000000000000000000001111101000 in 32 bits

33

Aside: <<

- C << operator
  - e.g., y = x << 3;
- "Left Shift by 3 bits"
  - Shifts each bit b_i to the left by 3 bits
  - The 3 bits on the extreme left go away
  - 3 Zero's come in at the right extreme end
- e.g., in 8 bits, 11001101 << 3 is 01101000
- Claim: Shifting an unsigned int left by one bit is the same as multiplication by 2
  - Qualification: If the product can be computed
  - i.e., all of the bits that go away are Zero

34

Aside: << ..

Proof: Consider n bit value $I = (b_{n-1}b_{n-2}...b_2b_1b_0)_{two}$

Shifted left by 1 bit and $b_{n-1} = 0$ we get

$$(b_{n-2}b_{n-3}...b_1b_0 0)_{two}$$

$$= \sum_{i=0}^{i=n-2} b_i \times 2^{i+1}$$

$$= 2\sum_{i=0}^{i=n-2} b_i \times 2^i$$

$$= 2I$$

Signed Integer Data

2s Complement Representation

The *n* bit quantity

least significant bit

$$x_{n-1}x_{n-2}\cdots x_0$$

represents the signed integer value

$$-x_{n-1}2^{n-1}+\sum_{i=0}^{n-2}x_i 2^i$$

Example: In 8 bits -128 + 64 + 32 + 16 + 2 + 1

13 is represented as 00001101

-13 is represented as 11110011

36

How is Data Represented?

- Character data: ASCII code
- Signed Integer data: 2s complement
- Real data

37

Real data

- Real numbers: points on the infinitely long real number line
  - There are an infinitely many points between any two points on the real number line

38

Real Data: Floating Point Representation

IEEE Floating Point Standard (IEEE 754)

32 bit value with 3 components (*s*, *e*, *f*)

1. *s* (1 bit sign)
2. *e* (8 bit exponent)
3. *f* (23 bit fraction)

represents the value

$$(-1)^s \times 1.f \times 2^{e-127}$$

39

Example: IEEE Single Float

Consider the decimal value 0.5

- Equal to 0.1 in binary 1.0×2^{-1}

  $(-1)^s \times 1.f \times 2^{e-127}$
- s: 0, e: 126, f: 000…000
- In 32 bits,

0 01111110 00000000000000000000000

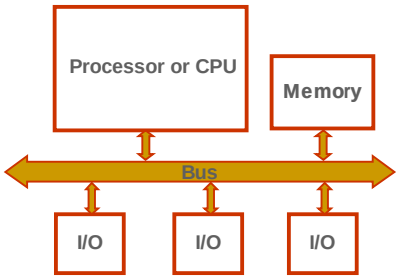
40

Basic Computer Organization

- Main parts of a computer system:
  - Processor: Executes programs
  - Main memory: Holds program and data
  - I/O devices: For communication with outside
- Machine instruction: Description of primitive operation that machine hardware is able to execute e.g. ADD these two integers
- Instruction Set: Complete specification of all the kinds of instructions that the processor hardware was built to execute

41

Basic Computer Organization



42

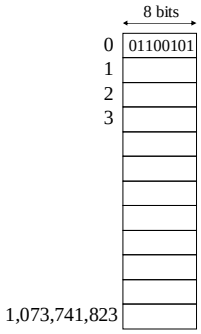
Aside: About Memory

- What is memory?
  - Devices that can remember things
- There are different kinds of memory in a computer system
  - Some remember by the state an electrical circuit is in e.g., SRAM
  - Others remember by the amount of electrical charge stored in a capacitor e.g., DRAM – "Memory"
  - Yet others remember by magnetic or optical properties e.g., Hard disk drive/Mag Tape, CD/DVD
- They can vary substantially in their speed and capacity

43

Main Memory.

- Holds instructions and data
- View it as a sequence of fixed sized locations, each referred to by a unique **memory address**
- In many computers, the size of each memory location is 1 Byte



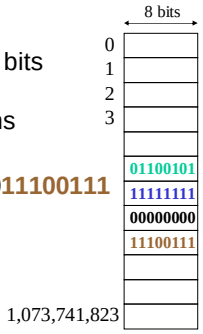
(if memory is of size 1GB)

44

Main Memory..

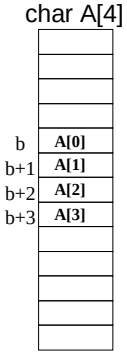
Values that occupy more than 8 bits would occupy more than one, neighbouring memory locations

e.g., 32 bit signed integer 01100101111111110000000011100111 would occupy 4 neighbouring memory locations, maybe as shown

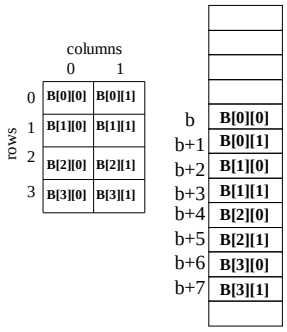


Data Structures: Arrays

char A[4]



char B[4][2]



Data Structures: Linked Lists

```
struct node
 { char data;          / a character: 8 bits in size
   struct node *next;  / a pointer: 32  bits in size
 }

struct node *head;   / a pointer: 32  bits in size
                     / in memory at address b
```