Computer Science 50

Introduction to Computer Science I

Harvard College

Week 8

David J. Malan
malan@post.harvard.edu

Morse Code

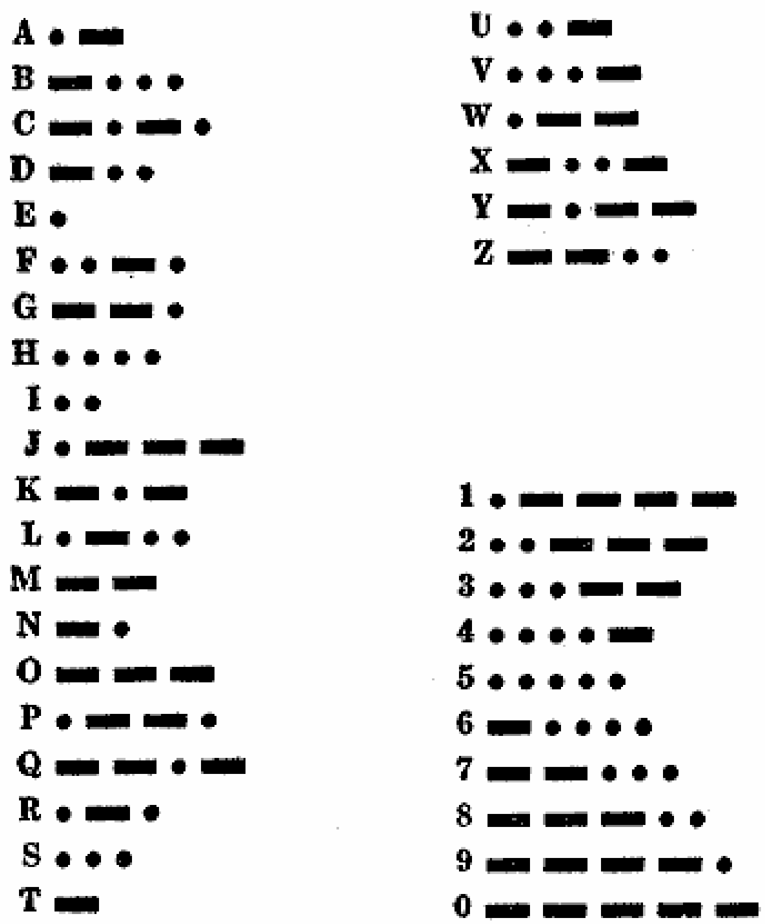
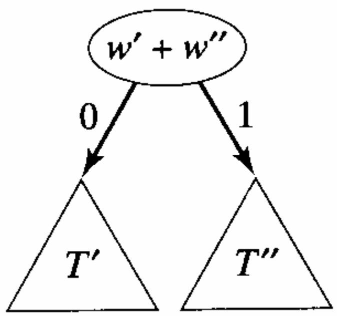


Image adapted from Wikipedia.

Huffman Coding
Immediate Decodability

1) Initialize a list of one-node binary trees containing weights $w_1, w_2, \ldots, w_n$, one for each of the characters $C_1, C_2, \ldots, C_n$.
2) Do the following $n - 1$ times:
   a) Find two trees T' and T'' in this list with roots of minimal weight w' and w''.
   b) Replace these two trees with a binary tree whose root has weight $w' + w''$ and whose subtrees are T' and T''; label the pointers to these subtrees 0 and 1, respectively:



3) The code for character C_i is the bit string labeling the path from root to leaf C_i in the final binary tree.

Pseudocode by Larry Nyhoff.

Huffman Coding

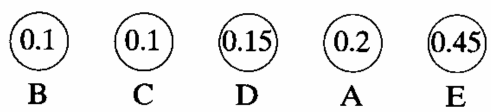
Example

"ECEABEADCAEDEEEECEADEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC
DEDEAEEDCEEAEEE"

character	A	B	C	D	E
frequency	0.2	0.1	0.1	0.15	0.45

Huffman Coding

Example



Figures by Larry Nyhoff.

Huffman Coding

Example

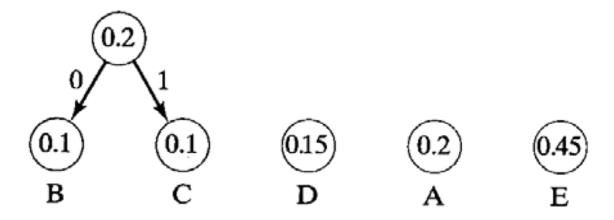


Figure by Larry Nyhoff.

Huffman Coding

Example

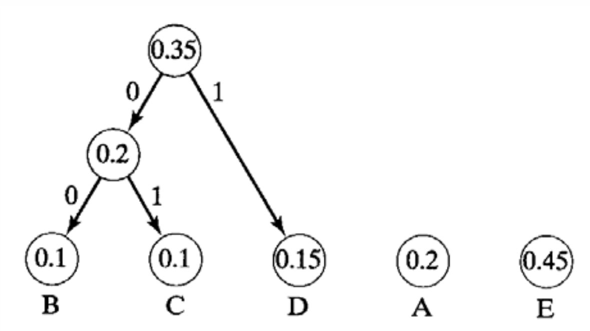


Figure by Larry Nyhoff.

Huffman Coding

Example

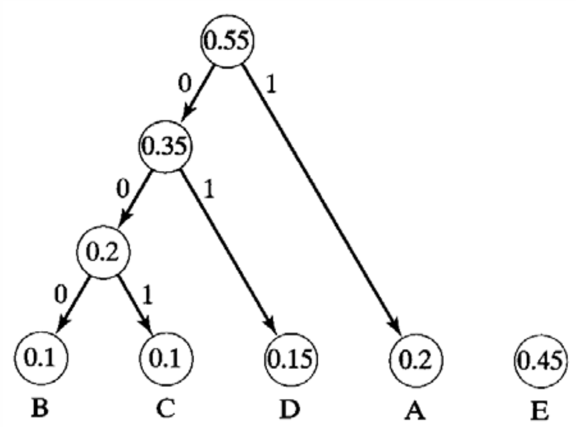


Figure by Larry Nyhoff.

Huffman Coding

Example

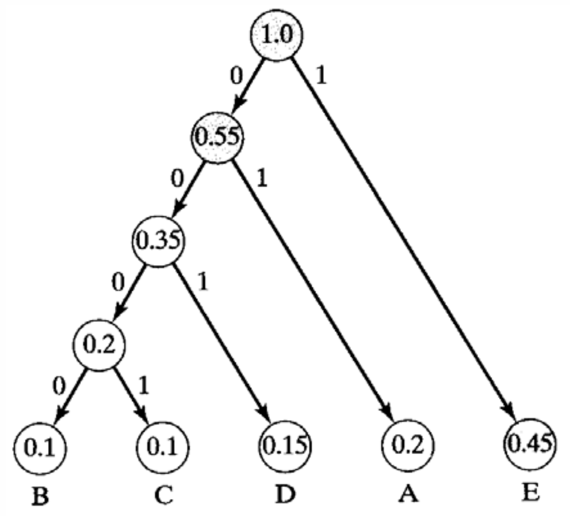


Figure by Larry Nyhoff.

Huffman Coding

Example

Character	Huffman Code
A	
B	
C	
D	
E	

Huffman Coding

Problem?

0 1 0 1 0 1 1 0 1 0

Huffman Coding

In C

```
typedef struct tree
{
     char symbol;
     int frequency;
     struct tree *left;
     struct tree *right;
}
Tree;
```

Bitwise Operators

- `&` bitwise AND
- `|` bitwise OR
- `^` bitwise XOR
- `~` ones complement
- `<<` left shift
- `>>` right shift

Bitwise Operators

AND (&)

		B: 0	B: 1
A	0		
	1		

OR (|)

		B: 0	B: 1
A	0		
	1		

XOR (^)

		B: 0	B: 1
A	0		
	1		

ones complement (~)

A	0	
	1	

see
`binary.c, tolower.c, toupper.c`

Bitwise Operators

Swapping Values

```
int FOO = 1;
int BAR = 4;

                // base-2 value in x          base-2 value in y
int x = FOO;    // 001
int y = BAR;    // 001                                     100

x = x ^ y;      // 001 ^ 100                               100
                // 101
y = x ^ y;      // 101                               101 ^ 100
                //                                         001
x = x ^ y;      // 101 ^ 001                               001
                // 100
```

see
swap2.c

Bitwise Operators
Swapping Values

```
int FOO = 1;
int BAR = 4;

                 // value in x                    value in y
int x = FOO;     // FOO
int y = BAR;     // FOO                                  BAR

x = x ^ y;       // FOO ^ BAR                            BAR
y = x ^ y;       // FOO ^ BAR                (FOO ^ BAR) ^ BAR
                 //                          FOO ^ (BAR ^ BAR)
                 //                                    FOO ^ 0
                 //                                        FOO
x = x ^ y;       // (FOO ^ BAR) ^ FOO                      FOO
                 // FOO ^ BAR ^ FOO
                 // FOO ^ FOO ^ BAR
                 // (FOO ^ FOO) ^ BAR
                 // 0 ^ BAR
                 // BAR
```

see
`swap2.c`

Underneath the Hood

Software

- Pre-Processing
- Compiling
- Assembling
- Linking
- Executing

From Source Code to Object Code

source code

```
#include <stdio.h>

int
main(int argc, char * argv[])
{
    printf("hello, world\n");
}
```

compile

assembly code

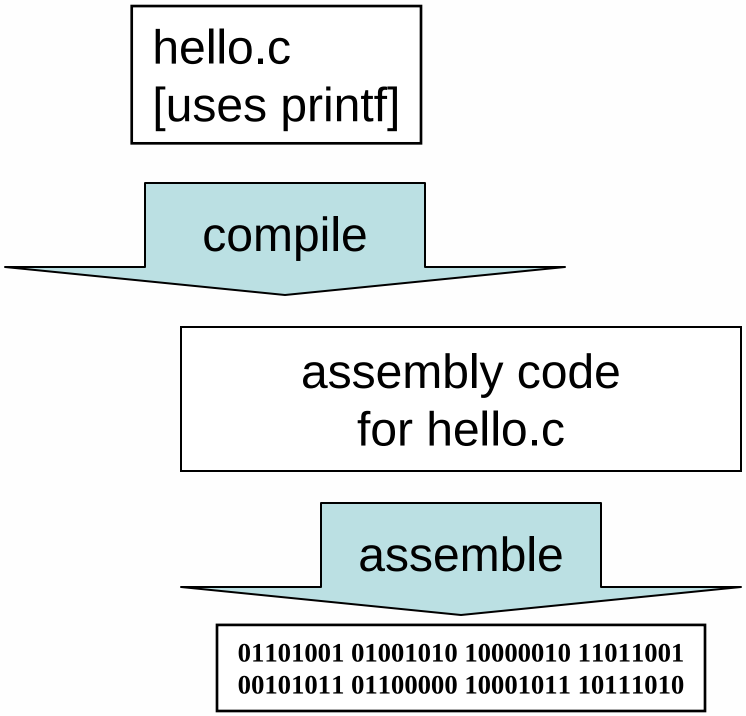
```
# hello.asm -- prints hello world
#   r2: string to be printed

   lc  r2, $hello   # text location
   sys r2, 4        # put string
   sys r0, 0        # halt

_data_:

hello: .data 'h','e','l','l','o',' '
       .data 'w','o','r','l','d','\n',0
```

assemble

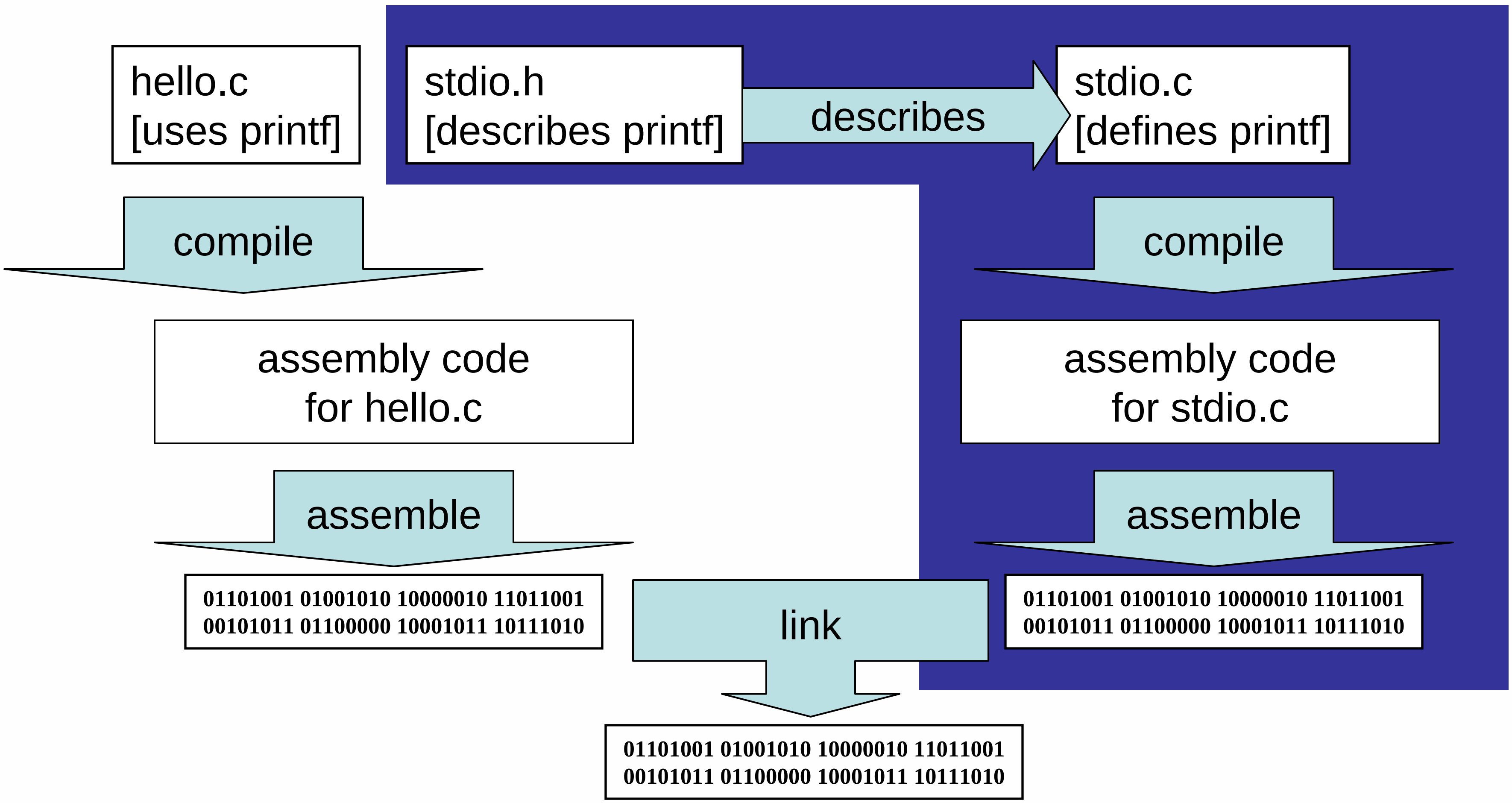
object code

```
01101001 01001010 10000010 11011001
00101011 01100000 10001011 10111010
```

Linking against Libraries

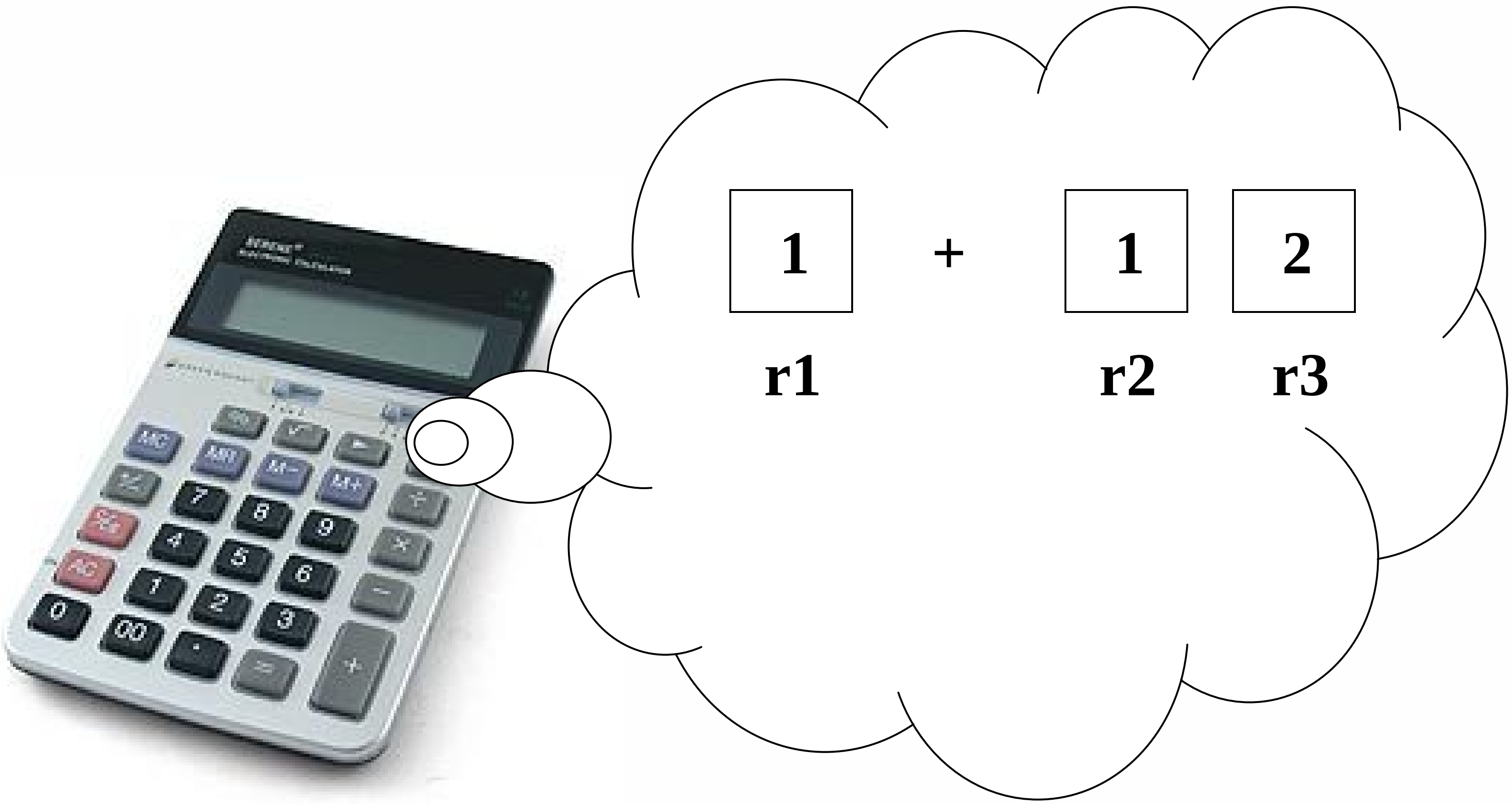


Linking against Libraries



Underneath the Hood

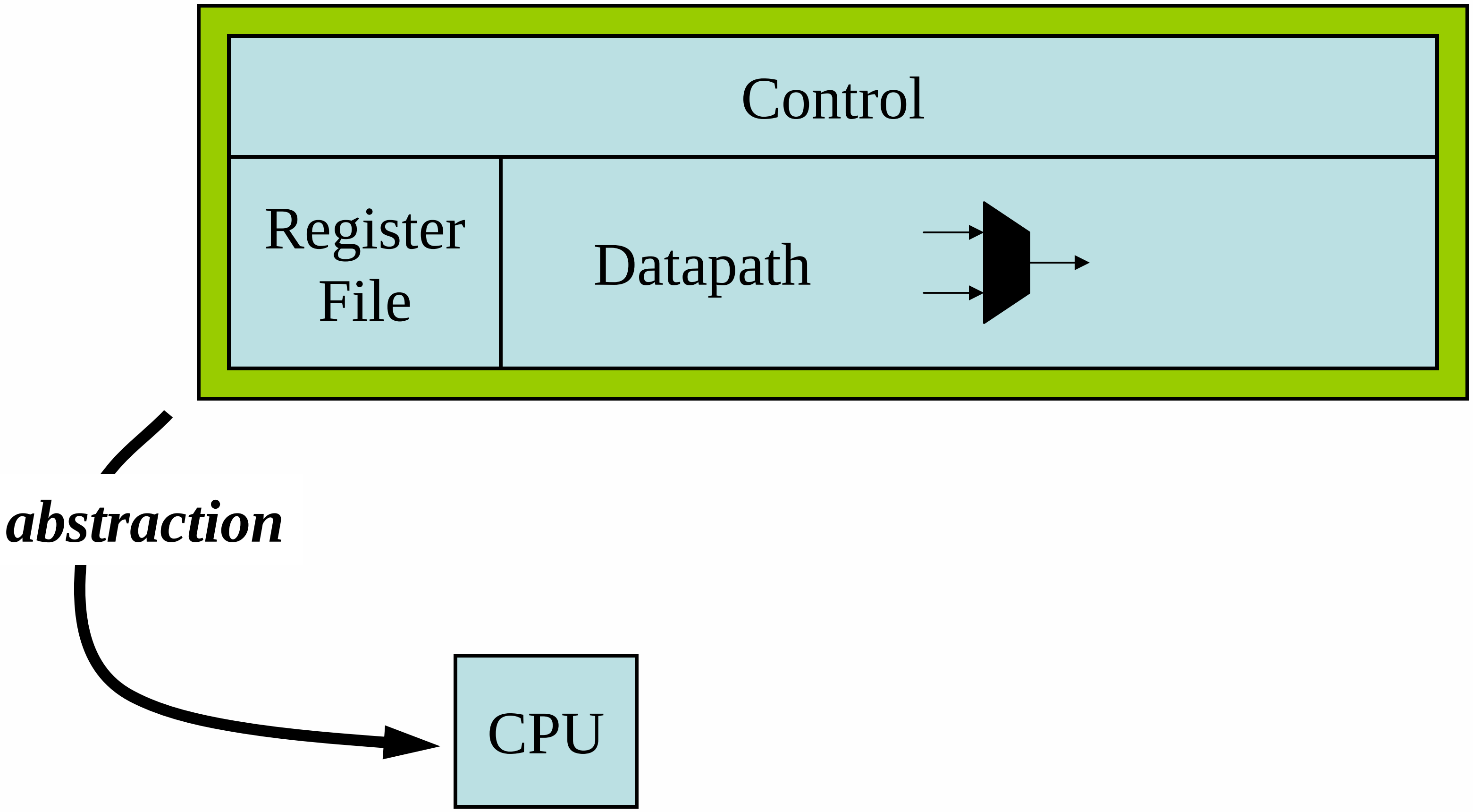
Hardware



What's in the Box?

- Registers
  - very fast temporary memory
  - few of them (16 or 32)
- Program counter
  - special register for tracking the next instruction to execute
- Memory
  - storage for program code and data
- Control unit
  - translates instructions into commands for registers and datapath
- Datapath
  - carries out basic operations (arithmetic, logical)
- I/O devices
  - supports flow of data into and out of the machine

What's in the CPU?



What can a CPU do?

Ant-8's Instruction Set

:: Arithmetic

- `add dst,src1,src2`
- `sub dst,src1,src2`
- `mul dst,src1,src2`
- `inc dst,const8`

:: Load constant

- `lc dst,const8`

:: Bitwise/Logical

- `and dst,src1,src2`
- `nor dst,src1,src2`
- `shf dst,src1,src2`

:: Load and store

- `ld1 dst,base,uconst4`
- `st1 src,base,uconst4`

:: Branch

- `beq tgt,src1,src2`
- `bgt tgt,src1,src2`
- `jmp uconst8`

:: System

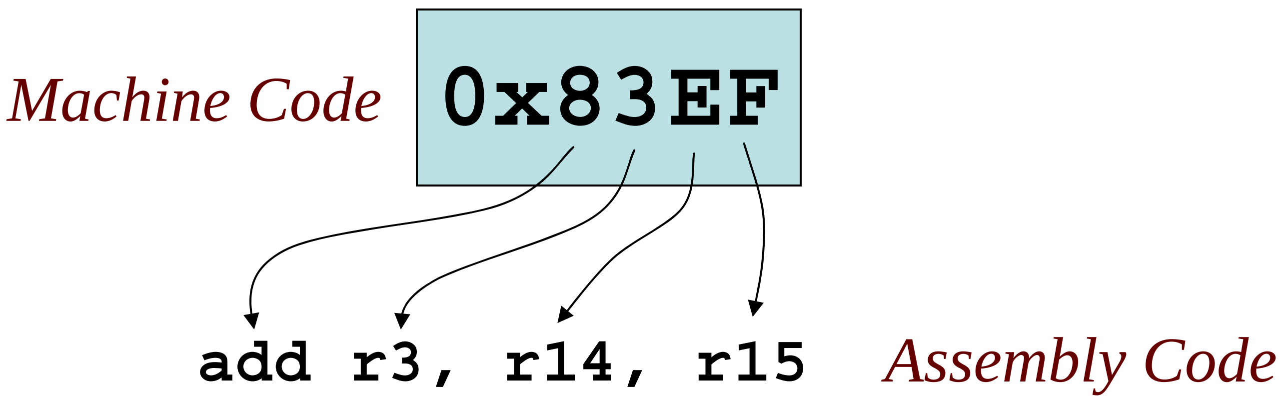
- `hlt`
- `in dst,channel`
- `out src,channel`

Machine Code

Ant-8

- All Ant instructions are 16 bits wide
  - first 4 bits encode the opcode
  - format of other 12 bits depend upon opcode

- Example:



From C to Ant-8

```
/* C code */
int test1 = 77;
int test2 = 96;
int totalPoints = test1 + test2;
```

```
# ANT code
    lc   r5, $test1     # address of test1 in mem
    ld1  r14, r5, 0     # value of test1
    ld1  r15, r5, 1     # value of test2
    add  r3, r14, r15   # sum of test scores
    st1  r3, r5, 2      # store in mem
    hlt                 # end of program code
_data_:
test1:          .byte 77
test2:          .byte 96
totalPoints:    .byte 0
# end of assembly file
```

Computer Science 50

Introduction to Computer Science I

Harvard College

Week 8

David J. Malan
malan@post.harvard.edu