CS 61C Quest Notes

Jeffrey Shen 30345555 77

Bits

- N bits is at most 2^n things
- Numerals: Binary (2), Decimal (10), Hexadecimal (16)
- ASCII: for all characters in English Language, 7 bits

Bit Conversions

Binary → Decimal, Hex → Decimal

- Add up powers of 2/16

$0b101$
$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$

Binary ↔ Hex

1. Pad left 0s to make multiple of 4
2. Read off groups of 4, using table
3. Drop any leading 0s

Decimal → Binary, Decimal → Hex

1. From left to right, find largest power of 2/16
2. If it fits, subtract and repeat w/ next digit

Decimal	Hex (0x...)	Binary (0b...)
00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Overflow: number is too large to be represented, positive or negative

Bit Operations

&	AND	x & y	extract bits, check neither are 1, turn off bits
|	OR	x | y	combine w/ mask, check either are 1, turn on bits
^	XOR (not equal)	x ^ y	flip bits w/ mask
~	Complement (flip)	~x	flip bits
<<	shift left	x << n	multiply by 2
>>	shift right	x >> n	divide by 2

1<<n is n 0s to right of 1

Number Representations

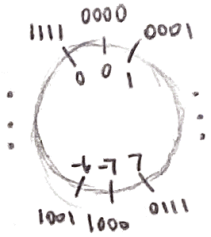
N bits

① Sign and Magnitude $[-(2^{N-1}-1), 2^{N-1}-1]$

Negation: Leftmost bit is sign bit, 0: positive 1: negative

- 2 zeros

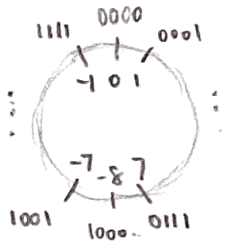
② One's Complement $[-(2^{N-1}-1), 2^{N-1}-1]$



Negation: Flip the bits

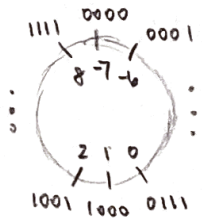
- 2 zeros, leftmost bit tells sign

③ Two's Complement $[-2^{N-1}, 2^{N-1}-1]$



Negation: Flip bits and add one

④ Bias Notation $[b, 2^N - 1 + b]$



b = -7

- Shift on unsigned notation

x + b = n

x: unsigned, b: bias, n: number represented

⑤ Unsigned $[0, 2^N - 1]$

- No negative numbers, max at $2^N - 1$

C

- Allows us to exploit underlying architecture, created in 1970s, C99
- Files first pass through C Pre-Processor where macros replace functions

Variable types

Ex) char: 8 bits (1 byte)
int: 32 bits (4 byte)
int *: 32 bits depending on machine [index into memory array]

False values: '\0', 0, NULL

Pointer Arithmetic: increments in sizeof(pointer_type)

Structs: structured groups of variables

sizeof: returns number of bytes

Pointers (type *var)

- Stores an address
- dereference operator (*): gets value at address
- C passes parameter by value, pass pointer
- If a variable is not initialized, it holds garbage
- Arrow Notation: var → x same as (*var).x

&var	= 0xA
var	= 0xB
*var	= 3

Memory Allocation

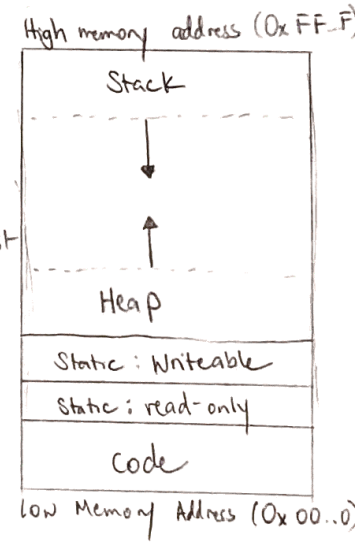
malloc(byte-size): returns void * pointer, initalizes with default garbage

free(ptr): must free any malloc'ed pont, once

realloc(ptr, size): reallocate memory

- After allocating memory, check if pointer is NULL

Memory Management



1. Stack: function local variables, strings allocated as arrays (LIFO)
2. Heap: dynamically allocated memory (malloc, calloc, realloc)
   Not necessarily contigous, fragmentation is an issue, circular Linked List
3. Static: global variables, statically allocated strings, basically permanent memory
4. Code: machine instructions

NOTE: - memory of pointers and what they point to may be in different memory

- char* s1 = "CS61C" s1[0] is in static, read-only
- char s2[] = "CS61C" s2[0] is in stack

Tips

- strlen(char*): # chars (bytes) in string not including terminator
- Big Endian: lowest address on left
- Little Endian: lowest address on right
- "\n" after print
- arr is pointer to first element in array

C Problems

- Draw out diagram w/ addresses
- Check if malloc or parameters are NULL