CSC 373 Winter 2020 Professor Lytinen
Computer Representations of Instructions, Characters, and Integers

We are almost done with the C book. Text: Bryant and O'Hallaron, Chapter p. 31-95 (sections 2.1, 2.2, and part of 2.3)

1. **Everything in a computer memory is a bit (0 or a 1)**
2. 8 bits = 1 **byte**
3. In general, a byte is the smallest directly addressable chunk of memory.
4. Bytes are further grouped together into larger sized chucks depending on the type of data

Virtual memory

- On a 64-bit machine, the operating system gives a process the illusion that it has 2^{64} bytes of memory (called **virtual memory**).

- This is also referred to as the **address space** of the process

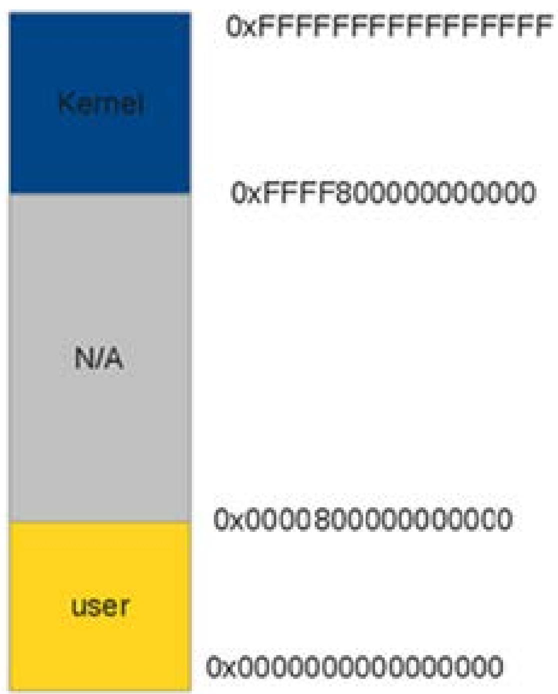
- In reality, much of this data is stored on the disk, or don't exist at all. Copies of parts of the process may also be stored in
  - Main memory
  - The caches (starting with L3)

- For the most part, processes generally have **no control** of precisely where in physical memory data is store (exception: In **assembly language** we can specify that an **int** should be placed in and integer register).. These decisions are made by the hardware and/or the operating system.

- At the process level, all that can be explicitly referenced are the registers (by referring to an instruction with a register operand) and/or a virtual memory address (0 ... (2^{64} - 1))

Division of address space

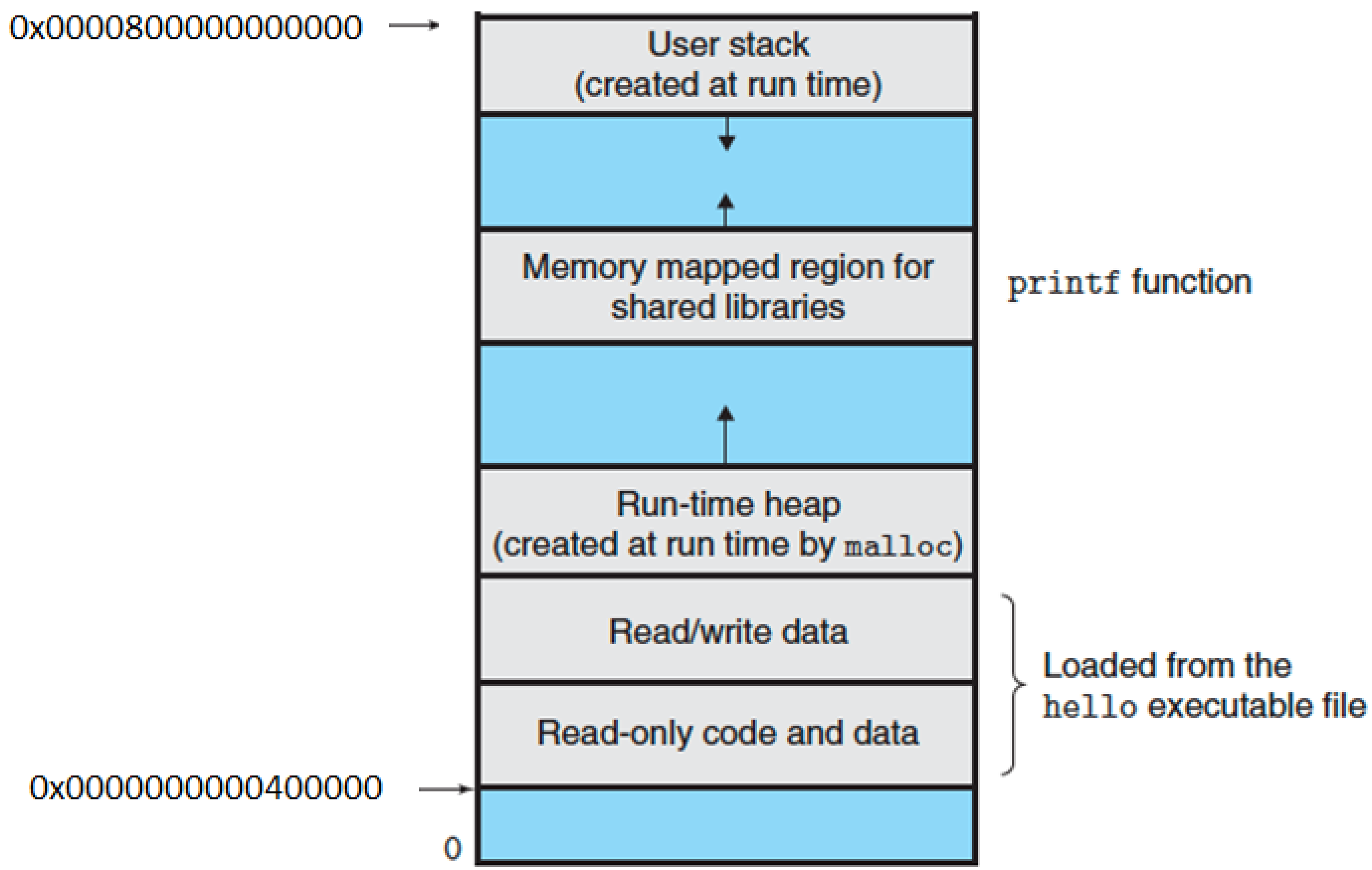
- Source: http://thinkiii.blogspot.com/2014/02/arm64-linux-kernel-virtual-address-space.html
- Not all 2^{64} bytes of the address space of a process are used;
- The "top" portion of the address space contains the OS "kernel" (its essentials); the bottom portion is your program and data. Note that the yellow is 2^{48} bytes, 2^{18} bytes are still unused.



kernel/arch/arm64 implementation

User address space

http://stackoverflow.com/questions/9511982/virtual-address-space-in-the-context-of-programming



Storing programs and data

- **Instructions** are stored in groups of bytes of varying size, depending on the specific instruction
  - Details discussed in Chapter 3; instructions range from 1-15 bytes

- Each **data type** has its own internal representation:

  - int: and its variants: 1-8 bytes (8 - 64 bits), depending on the specific data type. All but unsigned types use the same basic format, called **2s complement**, which can represent an approximately equal range of positive and negative numbers

    In an n-bit number,

    b_0 is the least significant (rightmost) bit, and b_{n-1} is the leftmost bit (possibly the sign bit)

  - The unsigned types do not support negative numbers within their ranges, and have a range of 0 ... (2^n - 1) (n = number of bits)

Storing programs and data

- float, double: floating point numbers have their own representation. We'll discuss floating point briefly (not now).
- char: ASCII (each is 1 byte) or Unicode (each is 2 bytes) encoding. We will assume characters are in ASCII format.
- All pointers are 64-bits; content is interpreted as an address
- How does a program know whether to interpret sometime as an instruction, a char, an int, a float, a pointer, etc?
  - It’s all just 0s and 1s
- At the machine level, the data type is interpreted by the context in which it is used

Representation of Integers

- Base 10 is most natural for humans
- For computers, binary is the only option
- Signed integers: leftmost bit represents the sign (0 means non-negative, 1 means negative)
- remaining bits used for the specific negative or non-negative values

Integers in Base 10

- Base 10:
  - 10 different symbols (0...9)
  - each digit has a different meaning depending on its location relative to other symbols in a number
  - Examples using the digit 1

Number	Number of trailing 0's	Meaning	As spoken in English
1	0	10^0	one
10	1	10^1	ten
100	2	10^2	one hundred
1000	3	10^3	one thousand
10000	4	10^4	ten thousand

Other digits

Number	Number of trailing digits	Meaning of digit	As spoken in English
2	0	$2 * 10^0$	two
50	1	$5 * 10^1$	fifty
400	2	$4 * 10^2$	four hundred

Example: 58204_{10}

10^4	10^3	10^2	10^1	10^0
5	8	2	0	4
$5 * 10^4$	$8 * 10^3$	$2 * 10^2$	$0 * 10^1$	$4 * 10^0$
50000	8000	200	0	4

50000 + 8000 + 200 + 0 + 4 = 58204

Base 2 for non-negative integers

- 2 different symbols (0, 1)
- Each digit represents a different power of 2, depending on where in the number it appears.
- For example, numbers with 1 followed by different numbers of 0's.

Number	Number of trailing 0's	Meaning	As spoken in English
1	0	2^0	one
10	1	2^1	two
100	2	2^2	four
1000	3	2^3	eight
10000	4	2^4	Sixteen

In general, if a 1 is followed by n 0's, then the 1 means 2^n

Converting from binary to base 10

Example: $0000\ 0000\ 1011\ 0011_2$

- In general, we'll pad the binary number with a appropriate number of 0s to conform to the data type's size (e.g., 16 bits for a `short`)

- $1*2^0 + 1*2^1 + 1*2^4 + 1*2^5 + 1*2^7 = 1 + 2 + 16 + 32 + 128 =$ $\mathbf{179_{10}}$

Another Example: $0000\ 0010\ 1100\ 1001_2$

- $1*2^0 +$…

Converting from (non-negative) base 10 to binary
Method 1: powers of two

To convert a base 10 number to n-bit binary:

1. Initialize ***p*** to be $\mathbf{2^{n-2}}$, the largest positive power of 2 be can be represented in ***n*** bits (leftmost reserved for the sign)

2. Repeat until ***p == 0***:
   a) If x >= p then the next digit is 1. Also, subtract p from x
   b) Otherwise, the next bit is 0.
   c) In either case, divide p by 2

Converting from (non-negative) base 10 to binary
Method 1: powers of two

What is 42_{10} as an 8-bit binary number?

X	p	binary digit
42	64	0
42	32	1
10	16	0
10	8	1
2	4	0
2	2	1
0	1	0

$42_{10} = 0101010_2$ (or 00101010_2 including the sign bit)

Exercise
Method 1: powers of two

What is 28 as an 8-bit binary number

x	P	binary digit
28	64	0
28	32	0
28	16	1
12	8	1
4	4	1
0	2	0
0	1	0

28_{10} =

Exercise
Method 1: powers of two

What is 187_{10} in binary, using 16 bits?

x	P	binary digit
1870	16384 (2^14)	0
1870	8192	0
1870	4096	0
1870	2048	0
1870	1024	1
846	512	1
314	256	1

187_{10} =

00000111…

X	P	Binary digit
48	64	0
48	32	1
16	16	1
0	8	0
0	4	0
0	2	0
0	1	0

187_{10} = 0000011101100000

```
// converts an int to a string whose chars are '0' or '1'.
void bits(int x, char b[]) {
  int p=1;
  int half_x = x/2;
  int i=0;
  // find largest power of 2 which is <= x
  while (p <= half_x) {
    p *= 2;
    i++;
  }
  b[i+1] = '\0';
  i=0;
  while (p > 0) {
    if (x >= p) {
      b[i] = '1';
      x -= p;
    }
    else b[i] = '0';
    p /= 2;
    i++;
  }
}
```

```
int main() {
  int x;
  char b[33];
  while (1) {
    printf("Type an int.  To finish, type -1\n");
    scanf("%d", &x);
    bits(x, b);
    if (x < 0) return;
    printf("%s\n", b);
  }
}
```

Converting from (non-negative) base 10 to binary
Method 2: Successive Division

- To convert x to binary:
  - Repeatedly divide x by 2
  - Next digit is the remainder
  - Continue until x >= 1
  - Answer is read from bottom to top
  - Pad with 0's if necessary

- Example: What is 19_{10} in 8-bit binary?

x	x/2 (int division)	Remainder
19	9	1
9	4	1
4	2	0
2	1	0
1	0	1

$19_{10} = 00010011_2$

Exercise: using successive division

Example: What is 37_{10} in binary?

x	x/2	Remainder
37	18	1
18	9	0
9	4	1
4	2	0
2	1	0
1	0	1

37_{10} =
Pad with as many leading 0's as necessary for a datatype

0000 0000 0010 0101

Exercise: using successive division

Example: What is 188_{10} in binary?

x	x/2	Remainder
188	94	0
94	47	0
47	23	1
23	11	1
11	5	1
5	2	1
2	1	0
1	0	1

$188_{10} = 0000\ 0000\ 1011\ 1100_2$

Converting from (non-negative) base 10 to binary
Method 2: Successive Division

```
void bits2(int x, char *str, int len) {
  int i;
  int rem;   // rem is the remainder of dividing x by 2

  str[len] = '\0';
  // fill in the bits right to left
  for (i=31; x > 0; i--) {
    rem = x%2;
    if (rem == 1) str[i] = '1';
    else str[i] = '0';
    x /= 2;
  }
  // add leading 0s to make the number 32 bits
  for ( ; i >= 0; i--)
    str[i] = '0';
}
```

```
int main() {
  int x;
  char bits[33];
  while (1) {
    printf("Type an >0 int, or -1 to finish”);
    scanf("%d", &x);
    if (x < 0) return;
    bits2(x, bits, 33);
    printf("%s\n", bits);
  }
}
```

Range of `int`

- ints are 32 bits in C
- Minimum possible value, discussed below
- Leftmost bit indicates sign (1 = negative, 0 = non-negative)
- This leaves 31 bits, so maximum number is

= 0111 1111 1111 1111 1111 1111 1111
$= 2^{31}-1$

$= 2{,}147{,}483{,}647_{10}$

The unsigned type in C

- Only non-negative integers can be represented
- # bytes depends on type of unsigned integer; e.g.
  `unsigned short` takes up 16 bytes
- Increases the range of non-negative numbers accordingly, at the sacrifice of no sign bit
- Minimum value of unsigned short:
- $0000\ 0000\ 0000\ 0000_2 = 0_{10}$
- Maximum value:
- $1111\ 1111\ 1111\ 1111 = 2^{16} - 1 = 65535$
- Minimum value of `unsigned int`
- $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0$
- Maximum value:
- $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2^{32} - 1 = 4{,}294{,}967{,}295_{10}$

Negative integers

- For signed integers, the leftmost bit represents the **sign** of a number
- Leftmost bit == 1, number is negative. Leftmost bit == 0, number is non-negative
- To make addition easier, computers use **2s complement notation**
- To convert a non-negative number x_2 (x in base 2) to a corresponding negative numbers in 2s complement:
  - flip all bits (i.e., change 0s to 1s, vice versa)
  - add 1
- Example: -29_{10} as a 32-bit int

  $+29_{10} = 2^4 + 2^3 + 2^2 + 2^0$
  $= 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 1101_2$

  Flip the bits, then add 1

  $-29_{10} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0010 + 1$
  $-29_{10} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0011_2$

Negative integers

- Example: -32_{10} as a 16-bit `short`

  $+32_{10} = 2^5 = 0000\ \ 0000\ 0010\ 0000_2$

  Flip the bits, then add 1

  $-32 = 1111\ 1111\ 1101\ 1111_2 + 1 = 1111\ 1111\ 1110\ 0000_2$

- Example: -46_{10} as an 8-bit binary number ????????

From 2s complement to base 10

- Example: 1111 1111 1110 0010_2 (16 bits = a short)

  Subtract 1
  Then flip the bits

  1111 1111 1110 0001
  0000 0000 0001 1110 = -30_{10}

- Example: 1110 0010_2 (8 bits = a char)

The minimum int

- Is it 1111 1111 1111 1111 1111 1111 1111 1111 ?

  Subtract 1, then flip the bits

  0000 0000 0000 0000 0000 0000 0000 0001

  so 1111 1111 1111 1111 1111 1111 1111 1111 = -1_{10}

The minimum int

- How about 1000 0000 0000 0000 0000 0000 0000 0000 ?

  Subtract 1, then flip the bits

```
1000 0000 0000 0000 0000 0000 0000 0000
-                                      1
----------------------------------------
0111 1111 1111 1111 1111 1111 1111 1111
```
then flip bits

1000 0000 0000 0000 0000 0000 0000 0000 $= -2^{31}{}_{10}$

- So the range of integers is $[-2^{31}, 2^{31})$

Exercise

```
#include <stdio.h>

int main() {
  char x = 1;
  while (x > 0) {
     x = (x << 1) + 1;
     printf("%d ", x);
  }
  printf("\n");
}
```

What is the last output of this program?

Binary counting

```
#include <stdio.h>

int main() {
  char x = 0;
  do {
    printf("%d", x);
    x++;
  } while (x != 0);
  printf("\n");
}
```

Output is:

```
0 1 2 3 4 5 6 7 8 9 10 … 120 121 122 123 124 125 126
127 -128 -127 -126 -125 -124 -123 -122 -121 -120 …
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```

Addition of two 1-bit numbers

X	Y	X+Y = X^Y
0	0	0
0	1	1
1	0	1
1	1	0

We really need 2 outputs: sum (S) and carry (C)

X	Y	S	C = X&Y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = what logical operator(s) applied to X and Y?

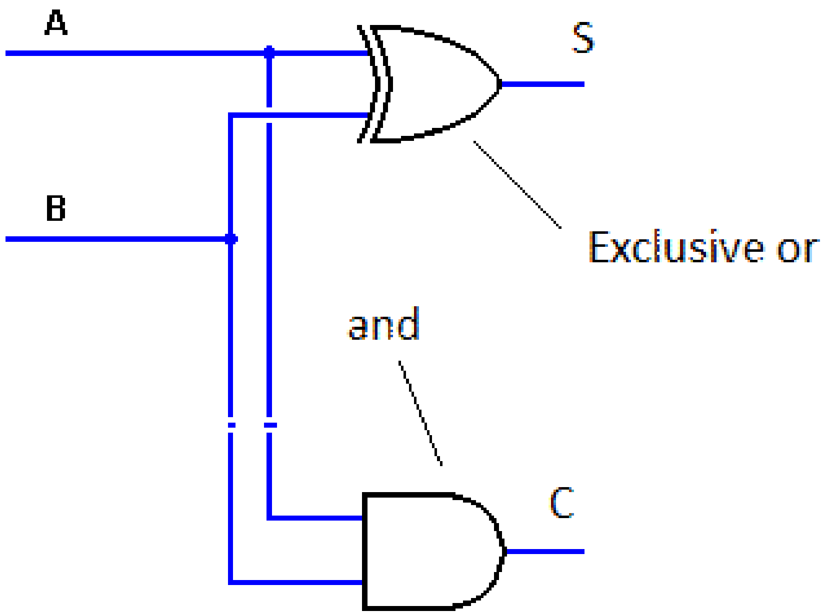
C = what logical operator(s) applied to X and Y?

X	Y	S	X^Y	C	X&Y
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	1	1

S = X ^ Y

C = X & Y

Addition of two 1-bit numbers



Addition of two n-bit numbers

X_i	Y_i	C_{i-1}	X+Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	10
1	0	0	1
1	0	1	10
1	1	0	10
1	1	1	11

Addition of two n-bit numbers: 3 inputs, 2 outputs

X_i	Y_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

S_i = what combination of X_i, Y_i, and C_{i-1}?

C_i = what combination of X_i, Y_i, and C_{i-1}

X_i	Y_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

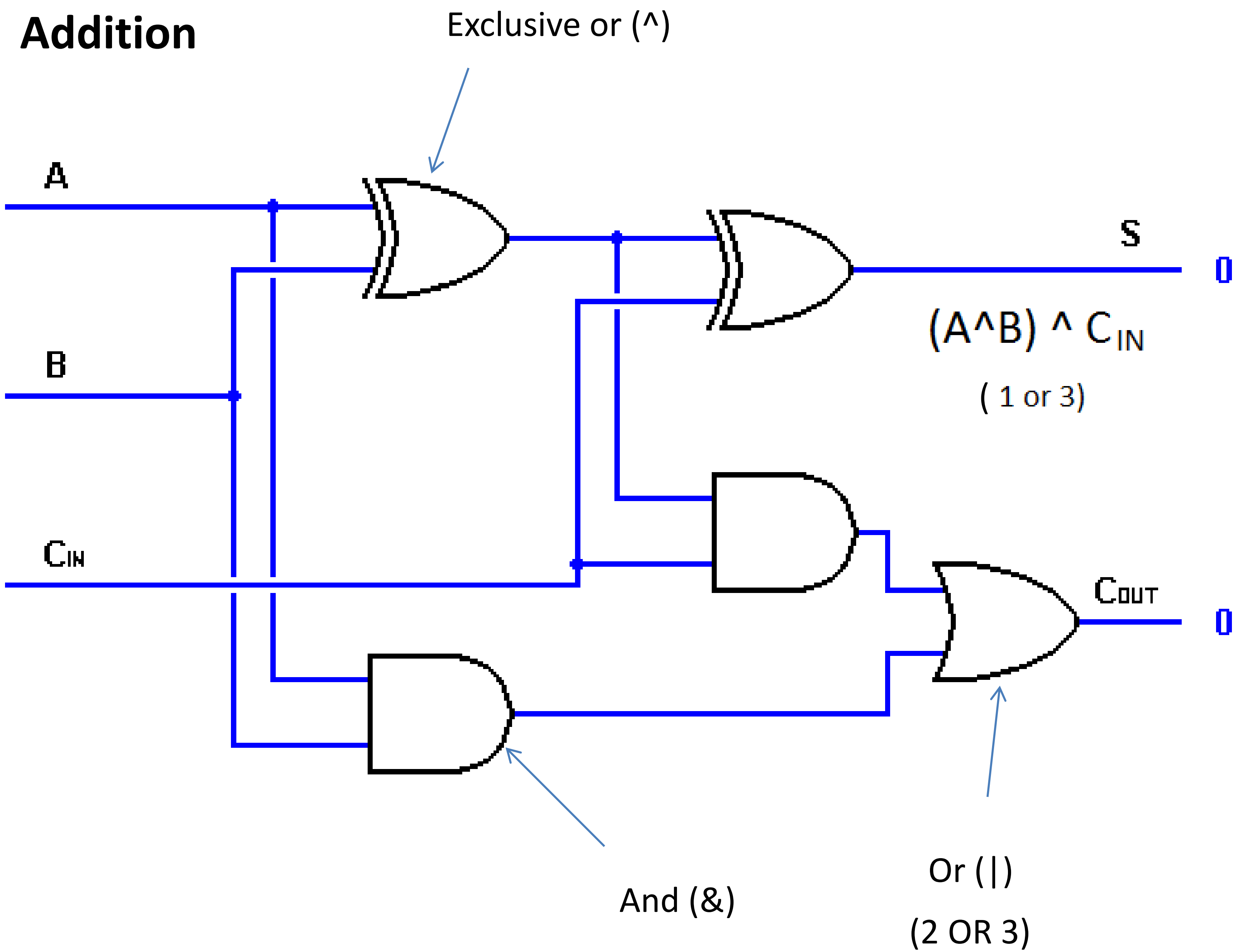
if an odd number of inputs are 1, then $S_i = 1$

if at least 2 inputs are 1, then $C_i = 1$

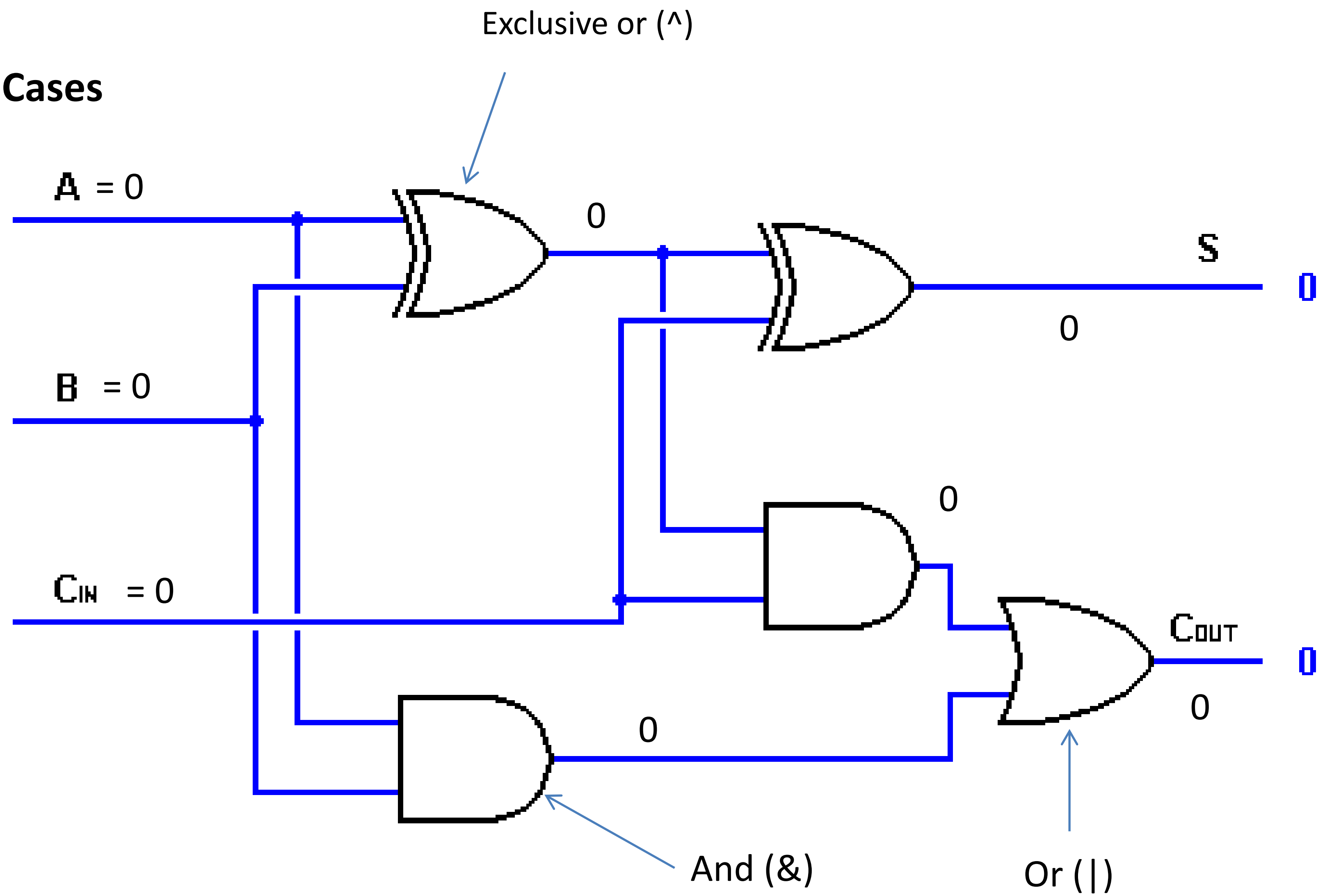
S_i = (X_i ^ Y_l) ^ C_{i-1}

C_i = ((X_i ^ Y_i) & C_{i-1}) | (X_i & Y_i)

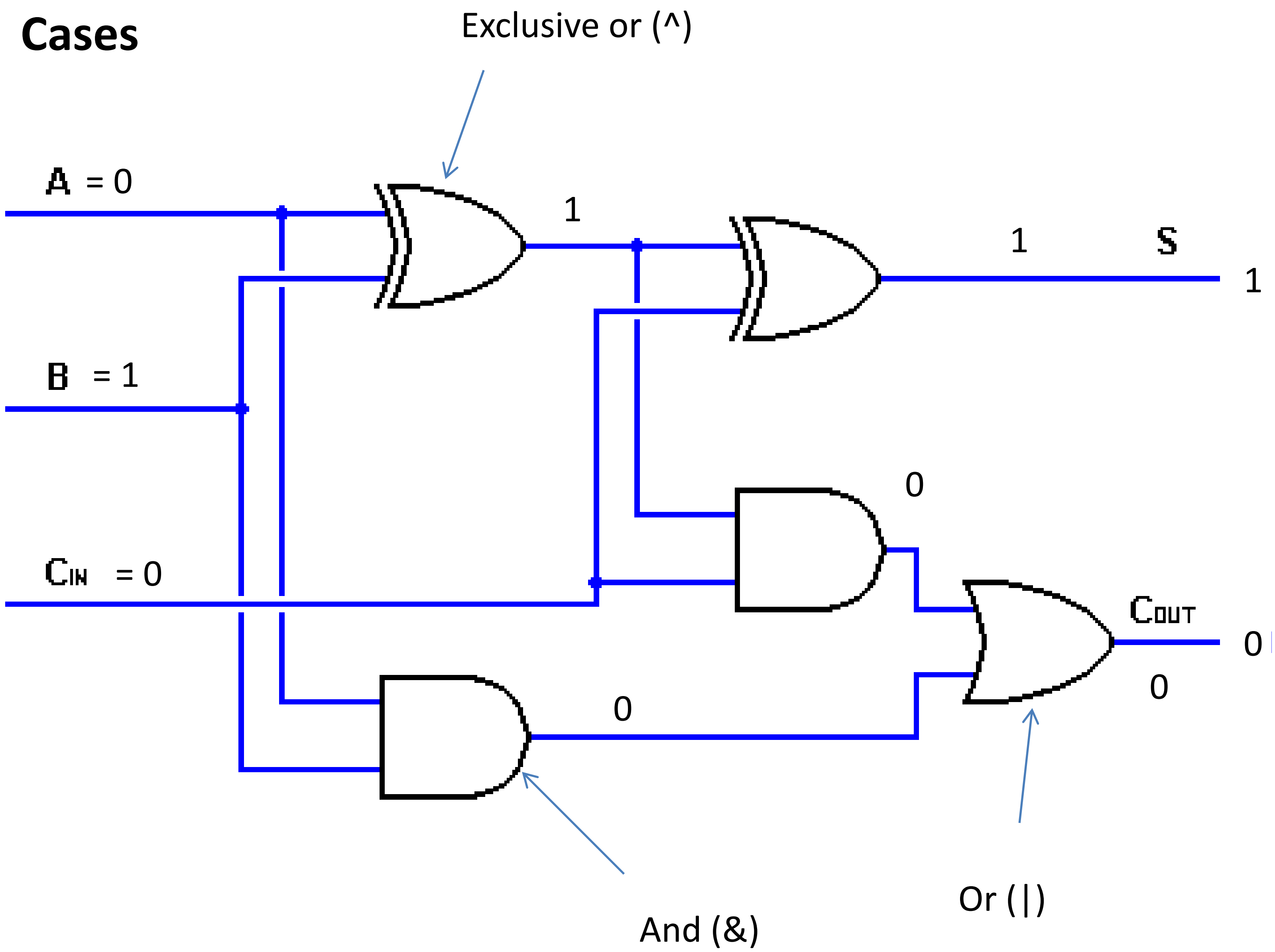
Addition


Exclusive or (^)
A
B
CIN
S
0
$(A \wedge B) \wedge C_{IN}$
(1 or 3)
COUT
0
And (&)
Or (|)
(2 OR 3)

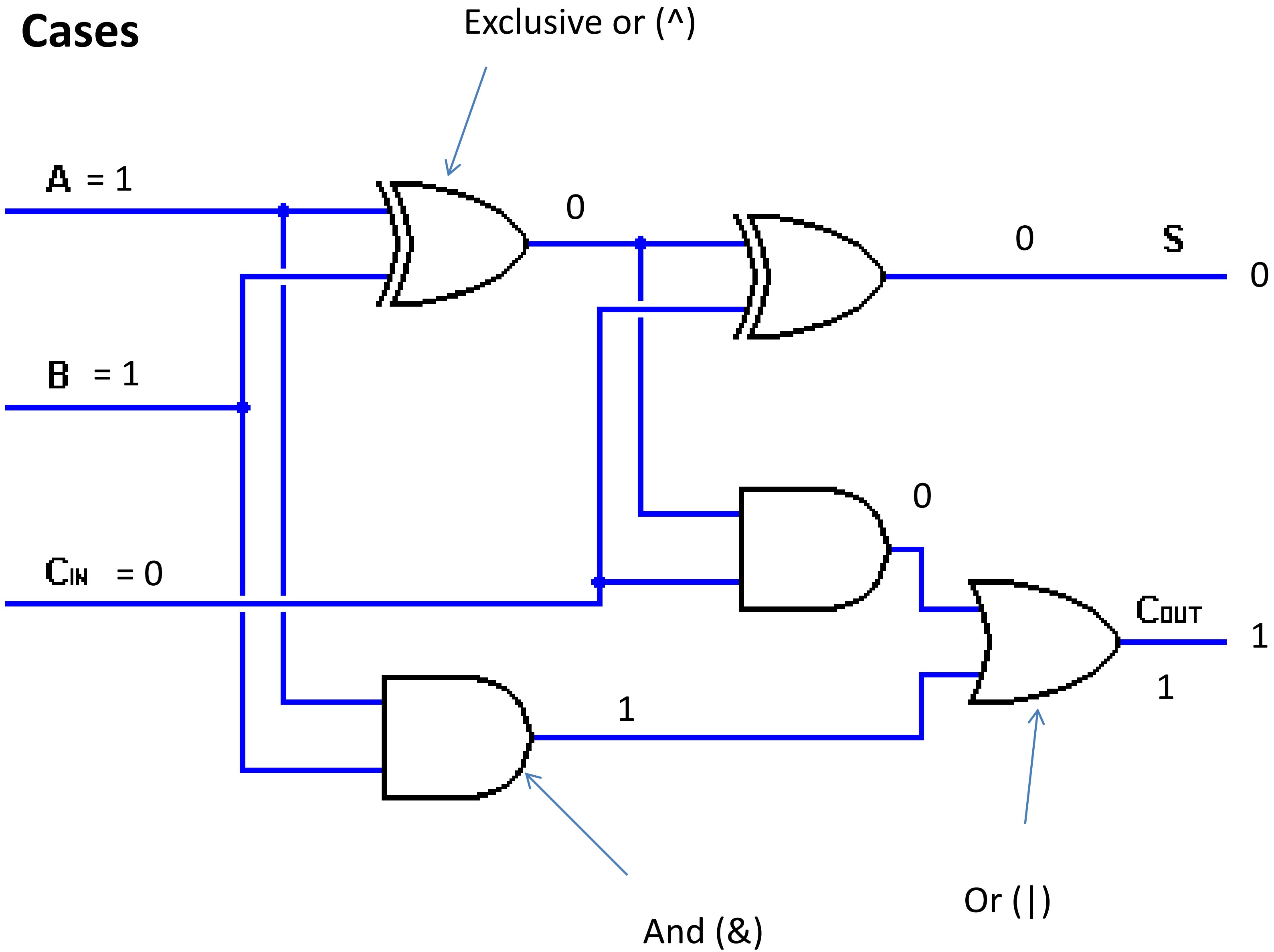
Cases



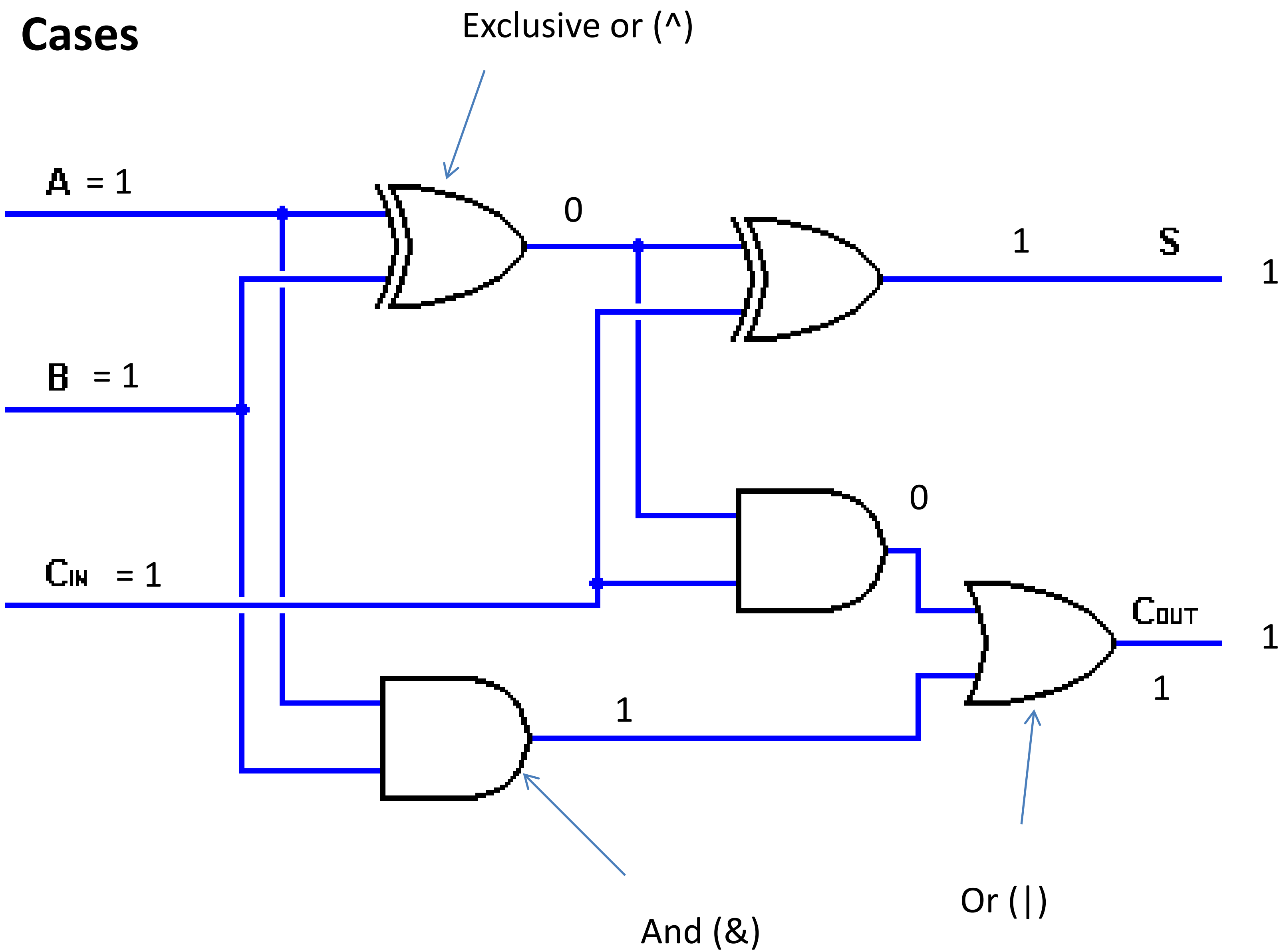
Cases



Cases



Cases



Try adding these numbers

```
  00110100
+ 00011111
```

0	0	1	1	0	1	0	0	A
0	0	0	1	1	1	1	1	B
0	1	1	1	1	0	0	-	C_{in}
0	1	0	1	0	0	1	1	Sum
0	0	1	1	1	1	0	0	C_{Out}

Addition

- Works for both positive and negative numbers, because of 2s complement
- Can be done more efficiently, but:

```
int add(int x, int y) {
    int z; // the answer
    int bitx, bity, x_xor_y, carry=0, sum;
    int i;
    for (i=0; i<32; i++) {
        bitx = (x >> i) & 1; // extract ith bit of x
        bity = (y >> i) & 1;
        x_xor_y = bitx ^ bity;
        sum = x_xor_y  ^ carry;
        carry = (x_xor_y & carry) | (bitx & bity);
        z = z | (sum << i);
    }
    return z;
}
```

Characters: stored as binary numbers

- ASCII code: Standard list of numbers representing 128 characters, requiring 8 bits (1 byte)
  - Actually , it’s 7 bits, but computer can only handle bytes
- A-Z, a-z, 0-9, punctuation, *control characters*
- Note A-Z differs from a-z - this often matters
- Example:

```
int main() {
  char c=0;
  do {
    printf("%u = %c\n", c, c);
    if (c == 127) break;
    c++;
  }
  while (1);
}
```

Hexadecimal (hex)

- Base 16 numbers
- Notice: $16 = 2^4$ (groups of 4 bits)
- 16 Digits (because it’s base 16):

  0,1,2,...,8,9,a,b,c,d,e,f
  or
  0,1,2,...,8,9,A,B,C,D,E,F

- Easy to convert between binary and hex: each hex digits represented in 4 bits

Easy to convert between binary and hex: each hex digits represented in 4 bits

Hex	Binary	Hex	Binary
O	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Used or seen a lot in low-level computing, because it’s more compact than binary, but easy to translate to binary

There is no hex datatype in C

Why? –

Counting in hex

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12...
1D, 1E, 1F, 20, 21, 22... 2F, 30, ...

- Hex numbers are usually signified by an `0x` before the number; e.g., `0x8AF0`
- To print a hex number using printf, it is conventional to use %#x (# causes `0x` to be displayed)

Converting Hex Number to Binary

Each hex digit translates into 4 bits, according to table from earlier.

$0x000093af_{16}$ = **0000 0000 0000 0000 1001 0011 1010 1111**

Both represent 32-bit numbers
8 hex digits = 32 bits

Converting Binary - Hex

- Group binary digits into groups of 4, starting from the right
- If fewer than 4 bits remaining, pad on the left with the necessary 0's
- Follow the table from before

- Example

Convert $0000\ 0000\ 1010\ 0001_2$ to hex

a) break into groups of 4

$0000\ 0000\ 1010\ 0001_2$

b) convert to hex

$00a1_{16}$

Converting Binary – Hex (ignoring 2s complement)

Convert 100001_2 to hex

a) break into groups of 4

$0010\ 0001_2$

b) convert to hex

$2\ 1_{16}$

0

More Exercises

Convert 00100110_2 to hex
Assuming no 2s complement, convert $0010\ 1010\ 0101_2$ to hex

Converting Hex to Decimal

- Each column (digit) is worth some power of 16 (just like each binary digit is worth a different power of two). Assume a hex number x has k digits, and that h_0 is the rightmost digit. Let's call the base 10 number d:

$$D = h_0*1 + h_1*16 + h_2*256 + \ldots + h_{k-1} * 16^{k-1}$$

Example: Convert $AF3_{16}$ to decimal

Converting Hex to Decimal

$D = h_0{*}1 + h_1{*}16 + h_2{*}256 + \ldots + h_{k-1} * 16^{k-1}$

Example: Convert $AF3_{16}$ to decimal (no 2s complement)

Hex Digit	Power of 16	Base 10 value
3	0	$3 * 16^0 = 3$
F	1	$15 * 16^1 = 240$
A	2	$10 * 16^2 = 2560$
	Total (sum the Base 10 values)	$3 + 240 + 2560 = 2803_{10}$

Converting Hex to Decimal

$D = h_0*1 + h_1*16 + h_2*256 + \ldots + h_{k-1} * 16^{k-1}$

Example: Convert $28C_{16}$ to decimal

Hex Digit	Power of 16	Base 10 value
C	0	12
8	1	128
2	2	512
	Total (sum the Base 10 values)	652

Converting Hex to Decimal

```
int from_hex(char *hex) {
  int len = strlen(hex);
  int ans = 0;
  int p = 1;
  int i;
  // power of 16 represented
  // by the leftmost digit
  for (i=0; i<len-1; i++)
    p = p << 4;
  for (i=0; i<len; i++) {
    int digit;
    if ('0' <= hex[i] && '9' >= hex[i])
      digit = hex[i] - '0';
    else if ('a' <= hex[i] && 'f' >= hex[i])
      digit = 10 + hex[i] - 'a';
    else digit = 10 + hex[i] - 'A'
      ans += digit * p;
        p /= 16;
  }
  return ans;
}
```

Converting from Decimal to Hex
Powers of 16 (analogous to powers of 2 algorithm)

Task: convert x_{10} to hexadecimal

Find p, the largest power of 16 <= x. Same table as before (powers of 2).

BUT right column is a hex digit (0-f), not a bit (0-1)

```
// p = largest power of 16 which is <= x
while (p > 0) {
        if (p <= x) {
                d = p/x; // integer division
                x = x – (d*p); }
        p  >>  4;  }
```

Converting from Decimal to Hex
Powers of 16 (analogous to powers of 2 algorithm)

Example: convert 2370_{10} to hex

x	p	hex digit (x/p)
2370	256	9
66	16	4
2	1	2

$2370_{10} = 942_{16}$

Note: 66 = 2370 - (9*256)
2 = 66 – (16*4)

Exercises

1. Convert 333_{10} to hex

x	p	d (hex)
333	256	1
77	16	4
13	1	D

2. Convert 1111_{10} to hex

x	p	d
1111	256	4
87	16	5
7	1	7

3. Convert 8206_{10} to hex

x	p	d
8206	4096 $(= 16^3)$	

Converting from Decimal to Hex
Successive division, again (but this time divide by 16)

convert 2370_{10} to hex

x	x/16	x%16
2370	148	2
148	9	4
9	0	9

$2370_{10} = 942_{16}$

Exercises

1. Convert 283_{10} to hex

x	x/16	x%16
283		

2. Convert 1060_{10} to hex

x	x/16	x%16
1060	66	4

3. Convert 8206_{10} to hex

x	x/16	x%16
8206		

Octal

Octal is base 8; usage is similar to hex, except that only the digits 0-7 are used, and conversion between octal and binary works in groups of three bits instead of four e.g.

Octal symbols

In base 10, we have 10 different symbols (0..9)
In base 8, we need 8 different symbols (0-7)

Octal counting:

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, ..., 17, 20, 21, ...

Octal numbers are usually signified by an 0 before the number; e.g., 0177

To print an octal number using printf, use %#o

Converting Octal Digits – Binary

- Individual octal digits can be converted to binary as follows:

Octal	Binary	Octal	Binary
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

Example: convert 1111000101011_2 to octal

Binary:	001	111	000	101	011_2
Octal:	1	7	0	5	3_8

Permissions in Linux

- Linux divides file permission into 3 types of users:
  - self
  - members of a user group
  - the rest of the world

- Each type is represented by 3 bits

- Each bit represents one type of permission
  - Rightmost bit: execute permission
  - Middle bit: write permission
  - Leftmost bit: read permission

- Therefore, 9 bits (or 3 octal digits)

Examples:
*necessary permission to read files in a directory

Code (binary)	Code (octal)	Permission
000	0	none
100	4	read-only
101	5	read/execute*
111	7	all permissions

The linux chmod command

- Stands for "change mode"
- Changes permissions on a file or directory
- To access a directory, the permission must be set to read and execute (101 or 5_8)
- To access a file, the permission must be set to read-only (100 or 4_8)
- To access a file and write to it, the permission must be set to read-write (110 or 111, 6_8 or 7_8)

The condor.depaul.edu server uses the **Apache** web server.

As configured, users place their Web files in ~/public_html

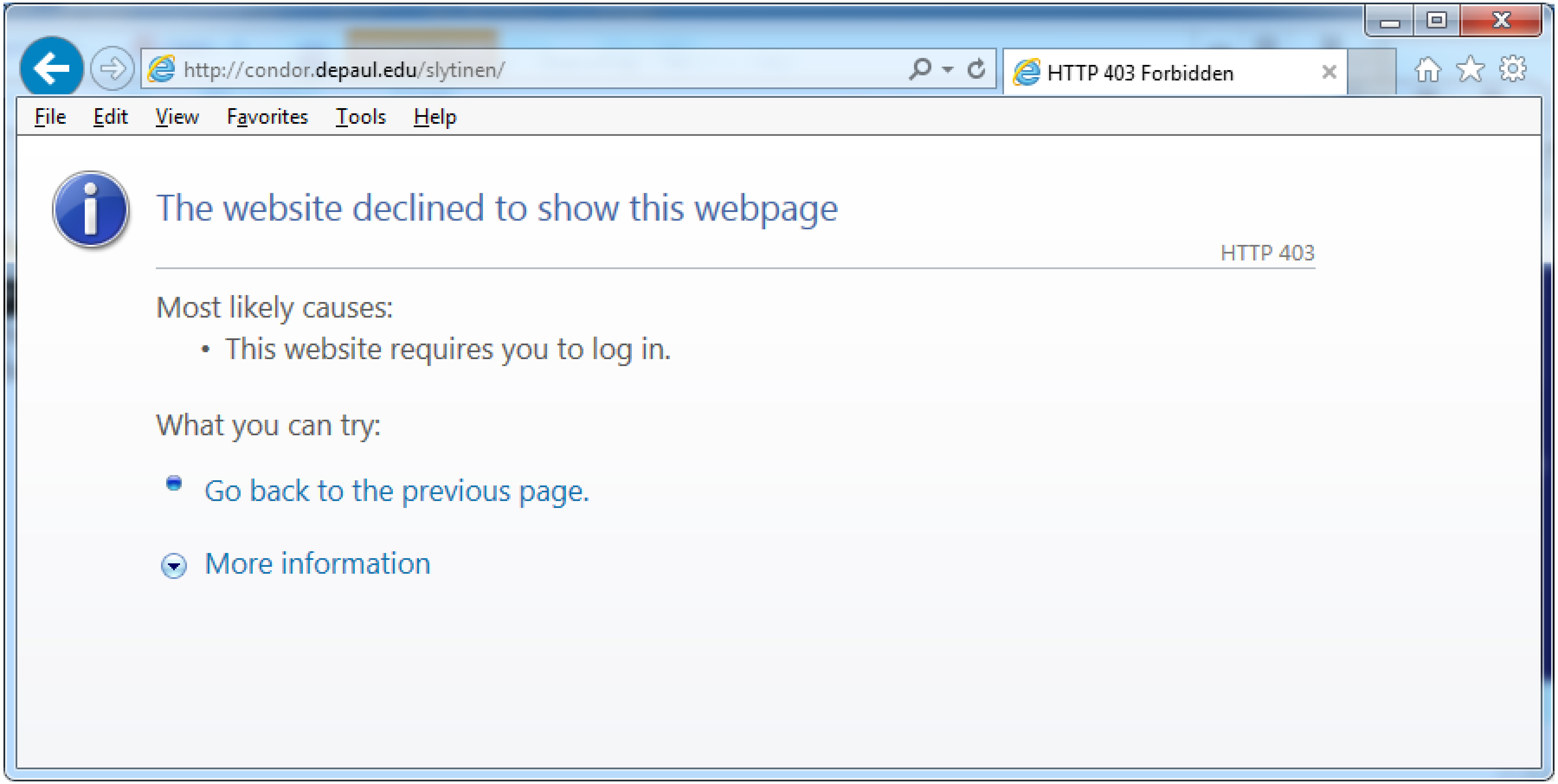
Default homepage is called index.html

Type this URL into a browser window:

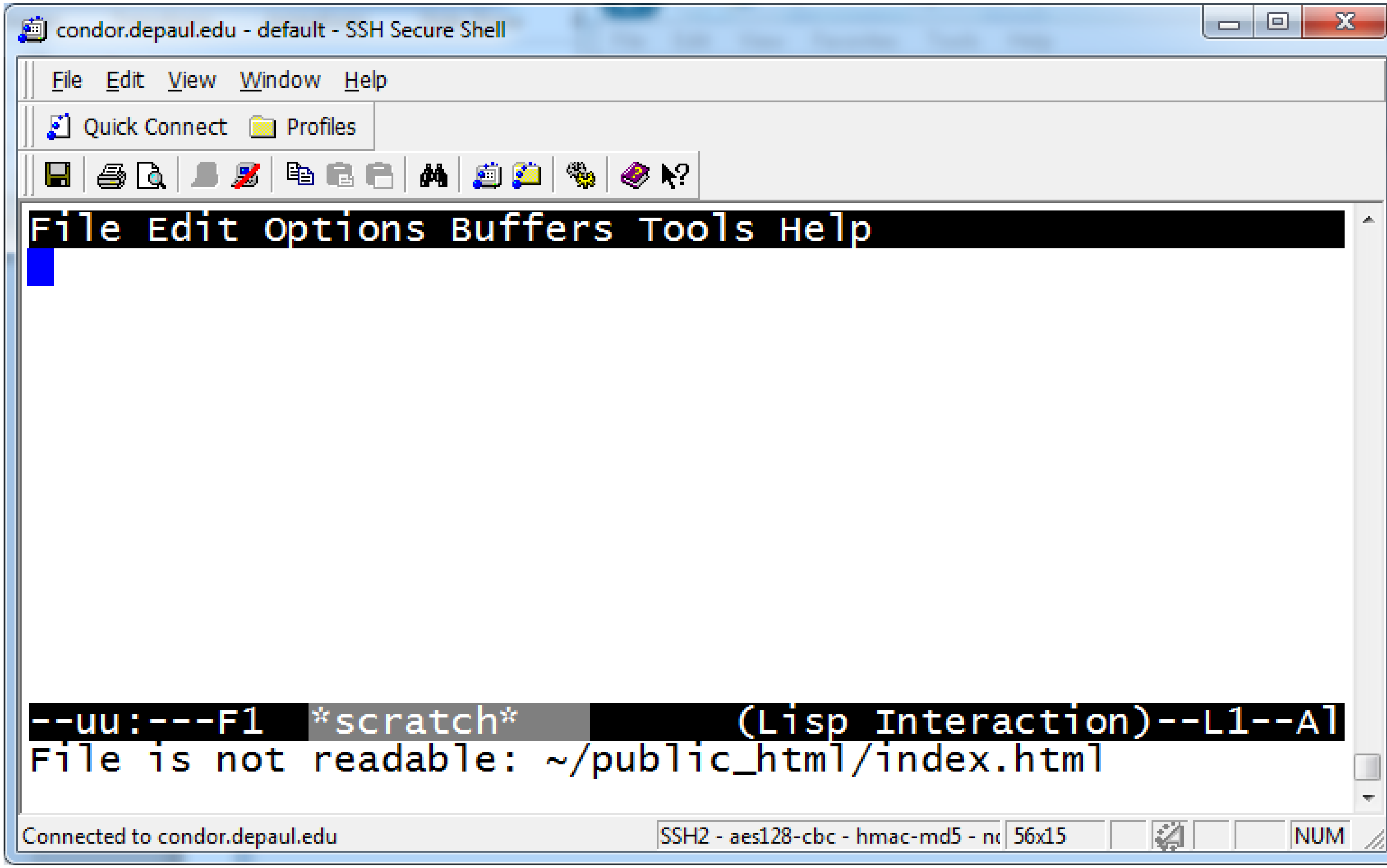
http://condor.dpu.depaul.edu/slytinen/

Now, I will login to condor.depaul.edu and use the `chmod` command:

```
$ cd public_html/
$ chmod 700 index.html
```



$ chmod 000 index.html


condor.depaul.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
File Edit Options Buffers Tools Help
--uu:---F1 *scratch* (Lisp Interaction)--L1--All
File is not readable: ~/public_html/index.html
Connected to condor.depaul.edu
SSH2 - aes128-cbc - hmac-md5 - nc
56x15
NUM