**Lec. 2 (3binary):**

**OVERFLOW**

2’sC: Taking negative of most negative number.

When adding 2#s of same sign: if carry into leftmost ≠ carry out leftmost then it is over flow.

Also, when you go too far from **zero.**

**Conversions To:**

S/M: Leftmost used for sign. Has (±0).

1’sC: **Flip** 7(0111) to get -7(1000). Has (±0).

2’sC: **Flip+1** 7(0111) to (1000+1)= -7(1001).

Has more neg. than pos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | U/S | S/M | 1C | 2C |
| 1111 | 15 | -7 | **-0** | -1 |
| 1110 | 14 | -6 | -1 | -2 |
| 1101 | 13 | -5 | -2 | -3  **Largest values of any n-bit long number:**  Largest positive: (signed) 2­­n - 1  (unsigned) 2n-1 – 1  Largest negative: (S/M) 11…11 = -2n-1 + 1  (1’sC) 10…00 = -2n-1 + 1  (2’sC) 10…00 = -2n-1.  **Conversions From:**  S/M: Replace leftmost with a sign.  1’sC: **Flip** -7(1000) to get 7(0111).  2’sC: **Flip except last 1 or 0**  -3(1101) to 3(0011). |
| 1100 | 12 | -4 | -3 | -4 |
| 1011 | 11 | -3 | -4 | -5 |
| 1010 | 10 | -2 | -5 | -6 |
| 1001 | 9 | -1 | -6 | -7 |
| 1000 | 8 | **-0** | -7 | -8 |

**Lec. 3 (Oct&Hex):**

**Octal (Base 8):**

Digits: 0 – 7

Used for 3, 6, 9, 12 – bitstring lengths

Usually 3k bit partitions

So, 345 = 011 100 101

**Hexadecimal (Base 16):**

Digits: 0 – 9 & A – F (for 10 – 15)

Used for 4, 8, 12, 16–bitstring lengths

Usually 4k bit partitions

So, 3FC = 0011 1111 1100

**Negation (X to –X):**

The 1’sC of X is 15 – X , aka the 15’sC of X.

(15 – 3A = C5)

The 2’sC of X is 16’sC of X (= [15’sC of X] + 1)

([15 – 3A] + 1 = C6)

**Oct Conversion (5-bit):**

U/S: 31 = 11 001 = 25

2’sC: 31 = -(00 111) = -7

1’sC: 31 = -(00 110) = -6

S/M: 31 = -(1 001) = -9

**Hex Conversion (7-bit):**

U/S: 5D = 101 1101 = 93

2’sC: 5D = -(101 0011) = -35

1’sC: 5D = -(010 0010) = -34

S/M: 5D = -(01 1101) = -29

**Floating point = Sci/notation**

Ex. 6.4 = 110.01 = 1.1001\*22

**Lec. 4 (ASCII/Frac/Float/Sci/IEEE):**

**ASCII (8-bits):**

Digit (0 – 9) = Hex (30 – 39) =Dec (48 – 57)

Letter (A – Z) = Hex (41 – 5A) = Dec (65 – 90)

Letter (a – z) = Hex (61 – 7A) = Dec (97 – 122)

Space = Hex 20 = Dec 32

**Convert to Binary Whole#:**

Divide by 2 w/R then bottom-up

**Convert to Binary Fraction#:**

Divide by 2 w/R then Top-down

**Convert to Whole# Binary:**

Mult by 2n

**Convert to Fraction# Binary:**

Mult by 2-n

**IEEE Overflow:**

Exp. Too large. E = 254-127=127

**IEEE Underflow:**

Exp too small. E = 1-127 = -126

**32-bit IEEE:**

**S =** Sign-bit, **E =** 8-bit exponent, **F =** 23-bit fraction

Ex. 1100 0101 1011 0100 0(16).

S = 1 (neg), E = 1000 1011 =139 - (127) = 12,

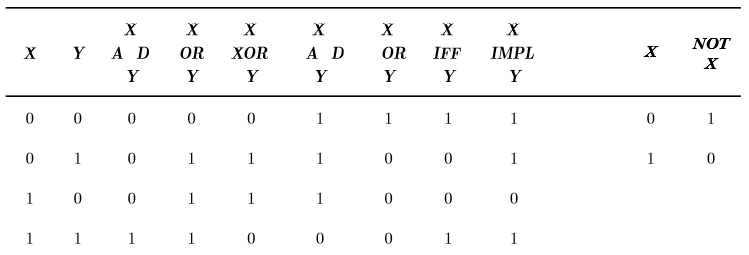
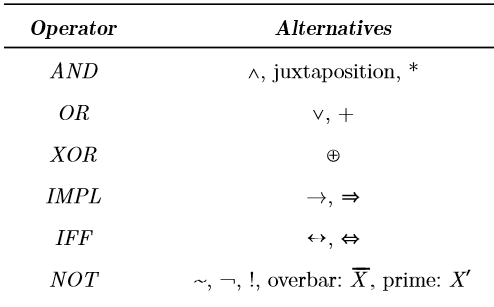
F = 0110 1000(16) = 1.011010(18). (prepend 1. to F)

**So, Float = - 1.011010(18)\*212.**

**Unicode (16-bits)**

**ASCII C** have null terminator ‘\0’.

**Lec. 5 (Boolean Logic):**



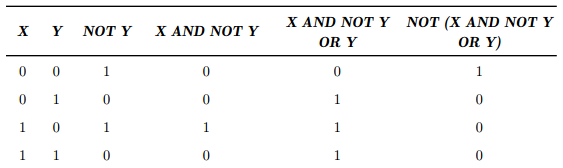
**Precedence (high to low):**

NOT, AND/NAND, OR/XOR, IMPL(≤)/IFF.

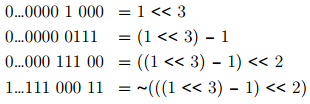
Tautology = always true

Contradiction = always false

Contingency = mix



Style 1:

**Lec. 6 (Bit Opr. & shift):**

Left Shift Zero Fill: 1110 => 1100

Left Circular Shift: 1110 => 1101

R-shiftw/0(logical R): 0111 => 0011

R-shiftw/sign(arithmetic R):

(positive same as logical R)

(When negative): 10011(-13)=>11001(-7)(2C)

R-circular: same

Left-shift (zero-fill):

Mask = ((1<<8) – 1); //last 8bits=1. Zero else.

1<<k = 2k.

Right: X>>5 //shifts X(of 1s) right 5 bits.

**Lec. 7 (pointers&structs): Lec. 8 (pntr&array):**

&b[0] + 1 is an integer-width (4 bytes) larger than &b[0].

&b[0] + k == &b[k]. & &b[k] - j == &b[k-j].

b + i == &b[0] + i – implies \*(b+i) = \*&b[i] = b[i].

p = &b[2] 🡪 p+1 = &b[3] & p-1 = &b[1].

\*b = 2 🡪 b[0] = 2; //So, &x[2]=2+x=x+2=&2[x] (same address)

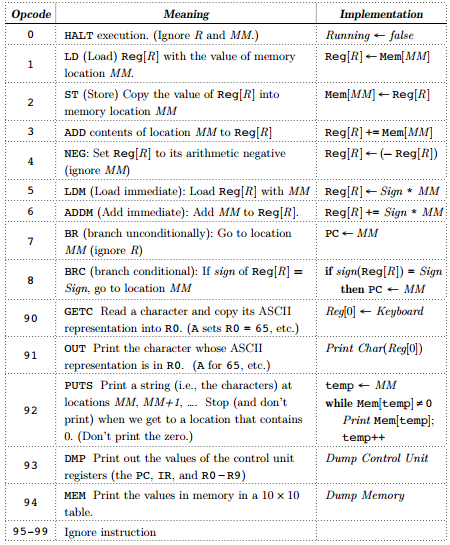
2[x]=x[2] (same value at address)

Int \*p = &x 🡪 &x is address of x

\*p=7 🡪 value at x is now 7.

P = address. \*p = value @address.

“%p” – prints address



typedef struct {

double real\_part;

double imag\_part;

} Complex;

void set\_cpx(Complex \*x, double a, double b) {

(\*x).real\_part = a; //or x->real\_part = a;

(\*x).imag\_part = b; //or x->imag\_part = b;

}//use printf(“%f”); use params (Complex \*x)

(a) p == &b[0] (b) q == p+2 (c) \*p == \*q-10 (d) \*p == \*(q-10)

(e) p[0] == p[1] (f) q == &p[2] (g) \*p == \*(p+1) (h) p != p+1

a, b, c, e, f, g, h are true; d may cause a runtime error because that address might be illegal.

**Lec. 9 (von Neumann comp):**

>3 main parts: CPU, Memory, I/O devices

>Different: programs are stored as data in memory

>Decoding of instruction in instruction register

>PC incremented during fetch instr, after reading intr from memory. Points to next instr.

>Phases of instr. Cycle: Fetch Instruction, Decode Instruction, Evaluate Addresses, Fetch Operands, Execute Instruction, Store Results

>CPU parts: Control Unit & (arith/log)Processing Unit

>MAR: holds location to read/write

>MDR: holds value read from/to write to memory

**Lec. 10 (simple decimal comp):**

5 1 78: LDM R1 <- 78

-5 2 78: LDM R2 <- -78

6 1 89: ADDM R1 <- R1 + 89 = 78 + 89 = 167

-6 2 89: ADDM R2 <- R2 + 89 = -78 + -89 = -167

2 1 45: ST M[45] <- R1 = 167

1 3 45: LD R3 <- M[45] = 167

3 3 45: ADD R3 <- R3 + M[45] = 167 + 167 = 334

4 3 67: NEG R3 <- -R3 = -334

7 8 10: BR 10

8 1 12: BRC 12 if R1 = 167 > 0: Yes

-8 2 14: BRC 14 if R2 = -167 < 0: Yes

9 0 11: I/O 0: Read char