Marathon!

Monday, May 11, 2015 3:00 PM

Encoding in Binary
1_10 = 1_2
4_10 = 2^2 = 100
42 = 42

$2^{\cdots}\ 2^3\ 2^2\ 2^1\ 2^0$

1	0
2	1
4	2
8	3
16	4
32	5

$$\begin{array}{r} 42 \\ -\,32 \\ \hline 10 - 8 = 2 \end{array}$$

42_2 1 0 1 0 1 0

$$\begin{array}{r} 42 \\ +\ 7 \\ \hline 49 \end{array} \qquad \begin{array}{r} 101010 \\ +\ 111 \\ \hline 110001 \end{array}$$

32 + 16 = 48 + 1 = 49

w = 5

$\underbrace{1\ 1\ 1\ 1\ 1}_{2^5 - 1} = 31$

$$\begin{array}{r} 11111 \\ +\ 11111 \\ \hline 1\,|\,11110 \end{array}$$

2^w

$A + B = (A + B)\ \%\ 2^w$

<u>Commutative:</u>a + b = b + a
<u>Associative:</u> (a + b) + c = c + (b + a)
<u>Identity:</u> a + 0 = a
<u>Inverse:</u> inverse(a) = -a, a + (-a) = 0

$$\begin{array}{r} 26 \\ \times\ 5 \\ \hline 130 \end{array} \qquad \begin{array}{r} 11010 \\ \times\ 101 \\ \hline 11010 \\ 00000\ \\ 11010\ \ \\ \hline 10000010 \end{array} = 128 + 2 = 130$$

0 1 1 0 = 6

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$
16 8 4 2 1

$$\begin{array}{r} 0110 \\ +\ 1010 \\ \hline 1\,0000 \end{array} = 0$$

$0x\_\_\_\_\_\_\_\_{}_{16}$ ↓ "$\_\_\_\_\_\_\_\_{}_2$"

$0x1$ = 0001

$0xA = 10_{10} = 1010$
$0xB = 11_{10} = 1011$
$0xC = 12_{10} = 1100$
$0xD = 13_{10} = 1101$
$0xE = 14_{10} = 1110$
$0xF = 15_{10} = 1111$

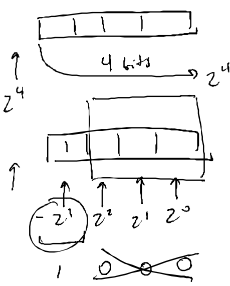
$0xAABB$ = 1010 1010 1011 1011

$52_{10} = 110100_2 = 0x34$

Declaration	32 bit	64 bit	
char	1	1	
short	2	2	
int	4	4	
long int	**4**	**8**	
char *	**4**	**8**	
float	4	4	
double	8	8	

---

double	8	8	



$V_{min} = 0$, $V_{max} = 2^w - 1$

4 bits $\rightarrow 2^4 - 1$

2^4

$T_{min} = -2^{w-1}$ $T_{max} = 2^{w-1} - 1$

$-2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$\sim X + 1 = -X$

$-1 = 1111$

$-8 + 4 + 2 + 1 = 8 + 7 = -1$

$\sim 1111 = 0000$

$+ \quad 1$

1 ✓

$\downarrow 2^{w-1}$

$-1 = 1|111$
$5 = +0|101$
$1|0100 = 4$

$0111 = 7$
$0111 = 7$
$1110 = 14$
$-8 + 4 + 2 = -2$

$7 + 7 - 2^w$
$= 7 + 7 \boxed{-2(2^{w-1})}$
$= 7 + 7 - 16 = -2$

-2^w $\quad -2^{w-1}$
$1000 = -8$
$1000 = -8$
$\boxed{0000 \neq -16}$

$A + B + 2^w$
$= A + B \boxed{+2(2^{w-1})}$
$-8 + -8 + 16 = 0$

Converting between Signed and Unsigned in C

- Most numbers are signed by default : unsigned 0x1234u

In C, <u>converting between unsigned and signed retains the underlying bit representation.</u>
<u>In a comparison containing an unsigned value, both numbers are implicitly cast to unsigned.</u>

int s1, s2;
unsigned u1, u2; /* "unsigned int" */
s1 = (int) u1;
s1 = u1;

(-1 < 0) = True
(-1 < 0U) = 2^w - 1 < 0 = 0 False

Some nonintuitive evaluations: $\quad 1000\ldots$

2147483647U > -2147483647 - 1 = -2147483648 = F $\quad -\infty \; +$

2^31 = 2147483648 $\quad 1000\ldots1$

2147483647 > (int) 2147843649U = T $\quad 2^{31} \quad + \quad 1 =$

(unsigned) -1 > -2 = T

$1111\ldots = -1$
$1111\ldots0 = -2 \quad -1 - 1 = -2$
$+1$

$1111\ldots > 1111\ldots0 \longrightarrow T$

<u>Expanding the Bit Reprsentation of a Number</u>

In C, when <u>converting an unsigned number to a larger data type</u>, we add leading zeroes to the representation.

$\underbrace{0000}_{+}\,\underbrace{1101}_{+}$

For <u>two's complement numbers</u>, converting to a larger data type involves *sign extension* which copies the most significant bit to the newly added bits.

$1 \; 1 \; 1 \; 1 \; 1 \; 1 \; 0 \; 1$
$-128 + 64 + 32 + 16 + 8 \;\; 4 \;\; 0 \;\; 1 = -3$

1 byte 1111 1111

1 1 1 1 1 1 0 1

$\underbrace{-128 + 64}_{= -64} + 32 + 16 + \underbrace{8}_{-8}\ 4\ 0\ 1 = -3$

$= -3$

15th 1111 1111

4th 1111 1111 . . . .

$\underbrace{\qquad}_{+}$

0 0 0 0 1111 1111

$\underbrace{\qquad}_{+}$

char a = -1;
unsigned b = (unsigned) a;

(unsigned) a -> (unsigned) (int) a
NOT (unsigned) (unsigned short) a

int c = -1;
char d = c;

~~X~~ . 1111 1111 = -1

unsigned X = 0xFFFFFFFF;
unsigned short US = X;
printf("%d\n",US); /* prints 65535 */
short ss = X;
printf("%d\n",ss); /* prints -1 */

~~X~~ | 1111 1111

char

---

n m

0x00	0x01	0x02	0x03
0xFF	0xFF	0xFF	0xFF

$=_{int}$ "-1"

short n = -1;
short m = -1

Big Endian

0x00	0x01	0x02	0x03
0x12	0x34	0x56	0x78

int a = 0x12345678

MSB

Little Endian

0x00	0x01	0x02	0x03
0x78	0x56	0x34	0x12

?

0x12345678

Desired

0x AABB|CCDD

Little Endian

0x00	0x01	0x02	0x03
0xDD	0xCC	0xBB	0xAA

int

0x00

short

0xCCDD

int *p;
p = (int *) malloc(sizeof(int));
*p = 5;
printf("%d\n", *p); /* prints 5 */
free(p);

Big Endian

0xAABB|CCDD

0x00	0x01	0x02	0x03
0xAA	0xBB	0xCC	0x00

int

short

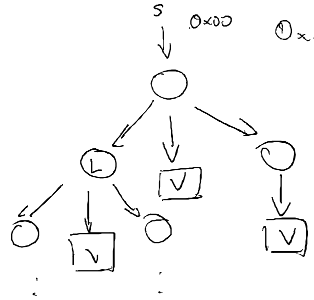
0x00

0xAABB

pb = 0xFE → p = 0x00 → 5

int a = 5;
int *p = &a;
int **pb = &p;
int ***pc = &pb;
printf("%d\n",***pc) /* prints 5 */;

S



S → I → L → V

Array Declarations are actually pointers which point to the beginning (first element) of a block of memory.
We can dereference any pointer using array notation.

int arr[] = {5, 8, 12};

0x00 0x04 0x08

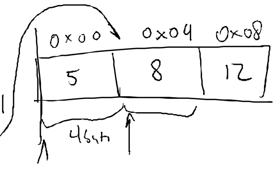
ret
old ebp

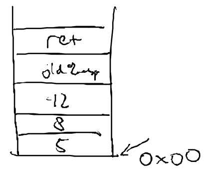
Array Declarations are actually pointers which point to the beginning (first element) of a block of memory.
We can dereference any pointer using array notation.

```
int arr[] = {5, 8, 12};

int *p = arr;
p = p + 1;
```

$= 0\times00 + 4 \times 1$

$= 0\times04$





arr[1] = arr + 4 × 1 bytes

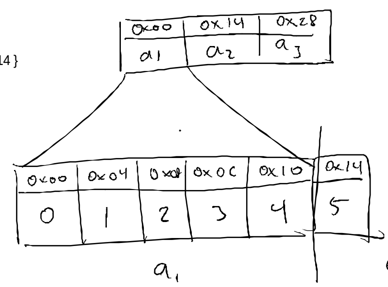
int 0x08

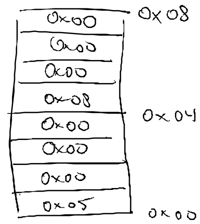
Little E

int 0x05

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x05	0x00	0x00	0x00	0x08	0x00	0x00	0x00

```
int myarray[3][5] = {
 a1 { 0, 1, 2, 3, 4 },
 a2 { 5, 6, 7, 8, 9 },
 a3 { 10, 11, 12, 13, 14 }
};
```





& = bitwise AND, | = bitwise OR, ^ = bitwise XOR, ~ = bitwise NOT

```
  1011        1011      11111111      0011
& 0110      ^ 0110    ^ 01011000    ^ 1110
  ----        ----      --------      ----
  0010        1101      10100111      1101
```

&& = logical AND, || = logical OR, ! = logical NOT

(A && !B) || (B && !A) = (A || B) && !(A && B) = logical XOR

```
   0110    T                        0110   T
|| 0000    F      T || F = T     && 0000   F
   ----                             ----
...0001    T                     ...0000   F
```

!!a = 1 or 0

For logical operations, **as soon as the result of an expression is determined, any remaining arguments are not evaluated.**

a && 5/a
a && a++

if(pointer)
    evaluate

Shift Operations
- Logical Left <<
- Logical Right >>
- Arithmetic Right >>
    - Signed right shifts in C are typically arithmetic.
    - Right shifting arithmetically always rounds down.

0011 <<3 → 1000

0x A5 << 4 → 0x50

1100 >>(Log) 2 → 0011

1100 >>(arit) 2 → 1111

0110 << 1 → 1100 = 12

6×2^1

0110 << 2 → 1000 = 8

$6 \times 2^2 = 24$

unsigned 1100 >> 2 → 0011 = 3 ✓

1111 >> 2 → 0011 = 3

$\frac{15}{4} = 3.$

signed 1111 >> 2 → 1111 = -1

$$\frac{15}{4} = 3.$$

Signed $1111 >> 2 \rightarrow 1111 = -1$

$$\frac{-1}{4} = -.25 \rightarrow -1$$

Fractional Binary Numbers

$\_\ .\ \_\ \_\ \_$: $10^0 \quad 10^{-1} \quad 10^{-2} \quad 10^{-3}$

$\_\ \_\ \_\ .\ \_\ \_\ \_$: $2^2 \quad 2^1 \quad 2^0 \quad 2^{-1} \quad 2^{-2} \quad 2^{-3}$

$\frac{1}{2^1} \quad \frac{1}{2^2} \quad \frac{1}{2^3}$

$\frac{1}{2} \quad \frac{1}{4} \quad \frac{1}{8}$

1011.1011_2

$8+2+1 \qquad \frac{1}{2} \quad \frac{1}{8} \ \frac{1}{16}$

$11 \ \frac{11}{16}$

$$\underset{8+2+1}{.1011} = \frac{11}{16}$$

Floating Point Numbers!

$$V = (-1)^s * M * 2^E$$

| s | exp | M frac |

k bits, n bit

E "desired exponent", $\text{Bias} = 2^{k-1} - 1$

8 bit format: s = 1 bit, e = 4 bits, frac = 3 bits

$$2^{4-1} - 1 = 7$$

① Denormalized Values

| s | $\underbrace{0000}_{\text{zero}}$ | frac |

$$V = (-1)^s * M * 2^E$$

$E = 1 - \text{Bias}$
$= -6$

$M = .\text{frac}$
$.101 = \frac{5}{8}$

Store +0

s	e	f
0	0000	000

$$V = (-1)^s \times M \times 2^E$$
$$= (-1)^0 \times 0 \times 2^{-6}$$
$$= 0$$

store $\frac{3}{512}$

$\frac{1}{2} \ \frac{1}{4} \ \frac{1}{8} \ \frac{1}{16} \ \frac{1}{32} \ \frac{1}{64} \ \frac{1}{128} \ \frac{1}{256} \ \frac{1}{512}$

$$.000000011 = .\underbrace{011}\ \times 2^{-6}$$

$$\frac{2}{512} + \frac{1}{512} = \frac{3}{512}$$

s e f

$\frac{2}{512} + \frac{1}{512} = \frac{3}{512}$

s	e	f
0	0000	011

Largest Possible Denormalized

s	e	
0	0000	111

$V = (-1)^s \times M \times 2^E$

$= (-1)^0 \times .111 \times 2^{-6}$

$= .000\,000\,111 = \frac{7}{512}$

$\uparrow \frac{1}{512}$

Smallest Nonzero Positive Denormalized

0	0000	001

$\rightarrow V = (-1)^0 \times .001 \times 2^{-6}$

$= .000\,000\,001 = \frac{1}{512}$

Normalized Values

$0 < e < 2^k - 1$

0000   1111

$E = e - Bias$

$\uparrow$ stored in representation

s	0011	.010

$M = 1 + f$

$M = 1 + .010_2 = 1.010_2$

$1\frac{1}{4}$

s	e	f
0	0111	010

$\overbrace{1.\underbrace{010}_{f}}^{1\frac{1}{4}} \times 2^E$

$E = 0$

$1.010_2 \times 2^0 = 1.010_2$

$E = e - Bias$

$0 = e - Bias$

$Bias = e$

$V = (-1)^0 \times 1.010_2 \times 2^{7-7}$

$= 1.010 \times 2^0$

$= 1.010 = 1\frac{1}{4}$ ✓

Storing $-3\frac{1}{2}$

s	e	f
1	1000	110

$\underbrace{11}_{3}.\underbrace{1}_{\frac{1}{2}} = 3\frac{1}{2}$

$1.\overbrace{11}^{f} \times 2^1$ ✓

$E = 1$

$E = e - Bias$

$1 = e - Bias$

$1 + Bias = e$

$1 + 7 = 8$

1000

$V = (-1)^1 \times 1.110 \times 2^{8-7}$

$= (-1) \times 1.110 \times 2^1$

$= (-1) \times \underbrace{11}_{3}.\underbrace{1}_{\frac{1}{2}} = -\frac{3}{2}$ ✓

Smallest Possible Nonzero Normalized

1000

Smallest Possible Nonzero Normalized

| 0 | 0 001 | 000 |

e

$M = 1.f = 1.000$

$f = 000$

$V = (-1)^0 \times 1.000 \times 2^{e-Bias}$

$= 1 \times 2^{1-7}$

$= 1 \times 2^{-6} = \frac{1}{64}$

$.000001$

1×2^{-6}

$.1 \times 2^{-5}$

Storing Largest Possible Normalized

s e f

| 0 | 1110 | 111 |

$8+4+2 = 14$

$E = e - Bias$

$M = 1.f$

$V = (-1)^0 \times 1.111 \times 2^E$

$= 1.111 \times 2^{14-7}$

$= 1.111 \times 2^7$

1- Bias Emin

$2^{-6} = 1 \times 2^{-6}$

| 0 | 0001 | 000 |

1.111 0000

1111 0000 = 240

Special Values

s e f

$-\infty$ | 1 | 1111 | 000 |

∞ multiply 2 large

divide by zero

$f = 000$

| 0 | 1111 | 010 |

$\sqrt{-1}$, $\infty - \infty$

Rounding

$1.000\,011 \xrightarrow{\text{round}} 1.000$

leading 1

288

1110 0100

$1.\underbrace{110}_{f}\,\underbrace{0100}_{x} \times 2^7$

e

| 0 | 1110 | 110 |

$E = 7 = e - Bias$

$7 + Bias = e$

$7 + 7 = 14$

$V = (-1)^0 \times 1.110 \times 2^7$

$= 1.110 \times 2^7$

$= 1110\ 0000$

128 64 32 16 8 4 2 1

= 1110 0000

128 64 32 16 8 4 2 1

$$\begin{array}{r} 128 \\ +64 \\ \hline 192 \end{array} \qquad \begin{array}{r} 192 \\ +32 \\ \hline 224 \end{array}$$

$\underline{-222}$

E+

1101 1110

$1.101\ 1110 \times 2^7$

$7 = E = e - Bias$
$= e - 7$
$= 14$

rounded → 1.110

e

| 1 | 1110 | 110 |

$V = (-1)^1 \times 1.110 \times 2^{14-7}$

$= (-1) \times 1.110 \times 2^7$

$- 1110\ 0000_2$

$= -224$

½ 1 0

1.001100

if lsb is 1, round up
if lsb is 0, round down

$\frac{-7}{1024}$

.0000000111

$.01\,1\,1 \rightarrow ? \times 2^{-6}$

.100

| 1 | 0000 | 100 | $= V = (-1)^1 \times .100 \times 2^{-6}$

$= .000\ 0001$

$= \frac{-8}{1024} = \frac{-4}{512}$

$\frac{5}{1024}$

$.0\ 000000101_2$

$.010\cancel{1} \times 2^{-6}$

$.010 = f$

1.01 * 2^-8
.010 1 * 2^-6

.011 1 * 2^-6
.01(2) 0
.0(2)0 0
.1000

| 0 | 0000 | 010 |

$.010 \times 2^{-6}$

$.00000001 = \frac{1}{256} < \frac{5}{1024}$

.0000001111 ↗ .000001

1.???

$E = e - Bias$

$1 - 7 = -6$

Floating Point Operations: Viewing two floating point values x and y as real numberes, and some operation _ defined over real numbers, the computation should yield *Round*(x _ y)

6 1/2 * (-3 1/8) = ?

Floating Point Operations: Viewing two floating point values x and y as real numberes, and some operation _ defined over real numbers, the computation should yield *Round*(x _ y)

6 1/2 * (-3 1/8) = ?

6 1/2 $\quad 110.1 = \underbrace{1.101} \times 2^2$

s	e	f
0	1001	101

$E = 2 = e - Bias$

$2 = e - 7$

$9 = e$

$V = (-1)^0 \times 1.101 \times 2^E$, $E = e - Bias$

1.101×2^2 $\quad = 9 - 7$

$= 110.1$ $\quad = 2$

$-3\frac{1}{8}$ $\quad 11.001$

$1.100|1 \times 2^1 \longrightarrow 1.100$

$M = 1 + f$

$.100$

1	1000	100

$E = 1 = e - Bias$

$8 = e$

$V = (-1)^1 \times 1.100 \times 2^1$

$= (-1)(1.100)(2)$

$= -11.00 = -3$

$6.5 \times -3 = -19.5$

10011.1

$1.00111 \rightarrow 1.010 \times 2^4$

$E = 4 = e - Bias$

$11 = e$

	e	
1	1011	010

$V = (-1)^s \times (1.010) \times 2^4$

$= -10100_2$

16 8 4 2 1

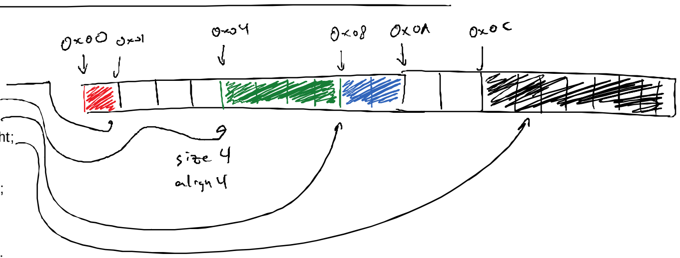
$16 + 4 = -20$

Operations with floating point values are commutative, but not associative. If there is a mix between floating point and integer values in an operation, the integer value is converted to floating point first. We are guaranteed f*f >= 0

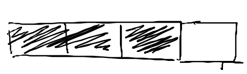
To\From	int	float	double	
int	OK	R0, OF	R0, OF	
float	R	OK	R, OF	
double	OK	OK	OK	

Structs

```
struct animal {
      char initial;
      int weight;
      short age;
      double height;
};

struct animal cat;
cat.initial = 'm';
cat.weight = 12;
cat.age = 8;
cat.height = 10.5;
```




```
typedef struct {
      short a;
      char b;
} struct1;

struct1 s;
```



```
typedef struct {
      char a;
      char b;
      char c;
} sturct2;
```

Union

```
union U3 {
      char c;
      int i[2];
      double v;
};

union U4 {
      char c[7];
      int a;
}
```

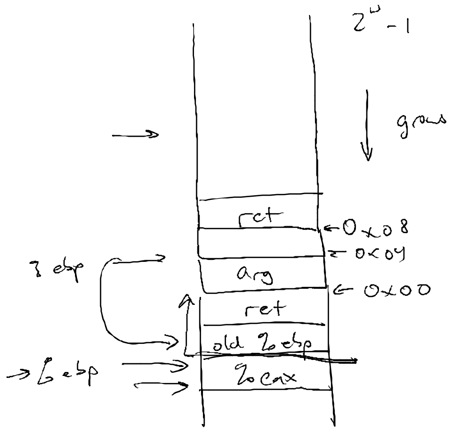
1. Translate the following assembly into a C function (assume one argument, an unsigned int), and then determine what it returns:

```
func:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ebx  ✓
  movl $0,%eax       ✓
  movl $0,%ecx       ✓
.L13:
  leal (%eax,%eax),%edx
  movl %ebx,%eax
  andl $1,%eax
  orl %edx,%eax
  shrl %ebx
  addl $1,%ecx
  cmpl $32,%ecx
  jne .L13
  popl %ebx
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int func (unsigned x) {
    int b = x;
    int result = 0;
    int c = 0;
do {
    int d = 2 * result;
    result = b;
    result = result & 0x01;
    result = result | d;
    b = b >> 1;
    c++;
} while (c != 32);
    return result;
}
```

```
int func (unsigned x) {
    int b = x;
    int result = 0;
    for (int c = 0; c != 32; c++) {
        int d = 2 * result;
        result = (b & 0x01) | d;
        b = b >> 1;
    }
    return result;
}
```

What is the Stack?



%ebp → "pointer to base of current stack frame"

%esp

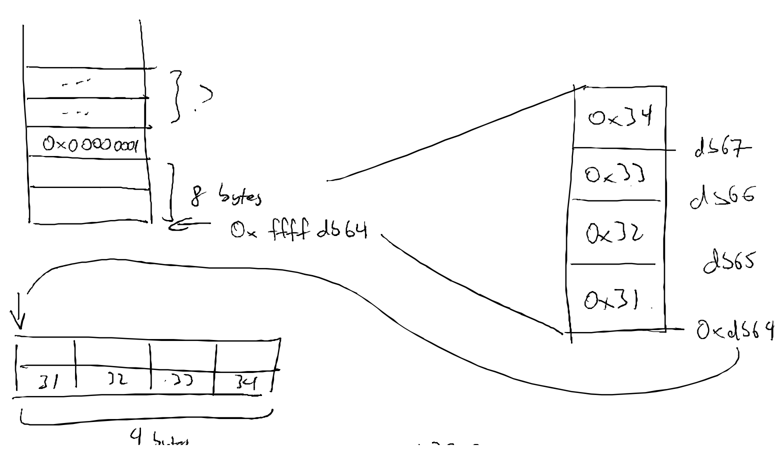
%eip

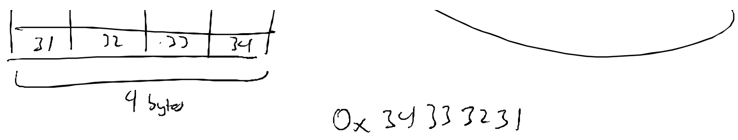
addl %ebx, %ecx

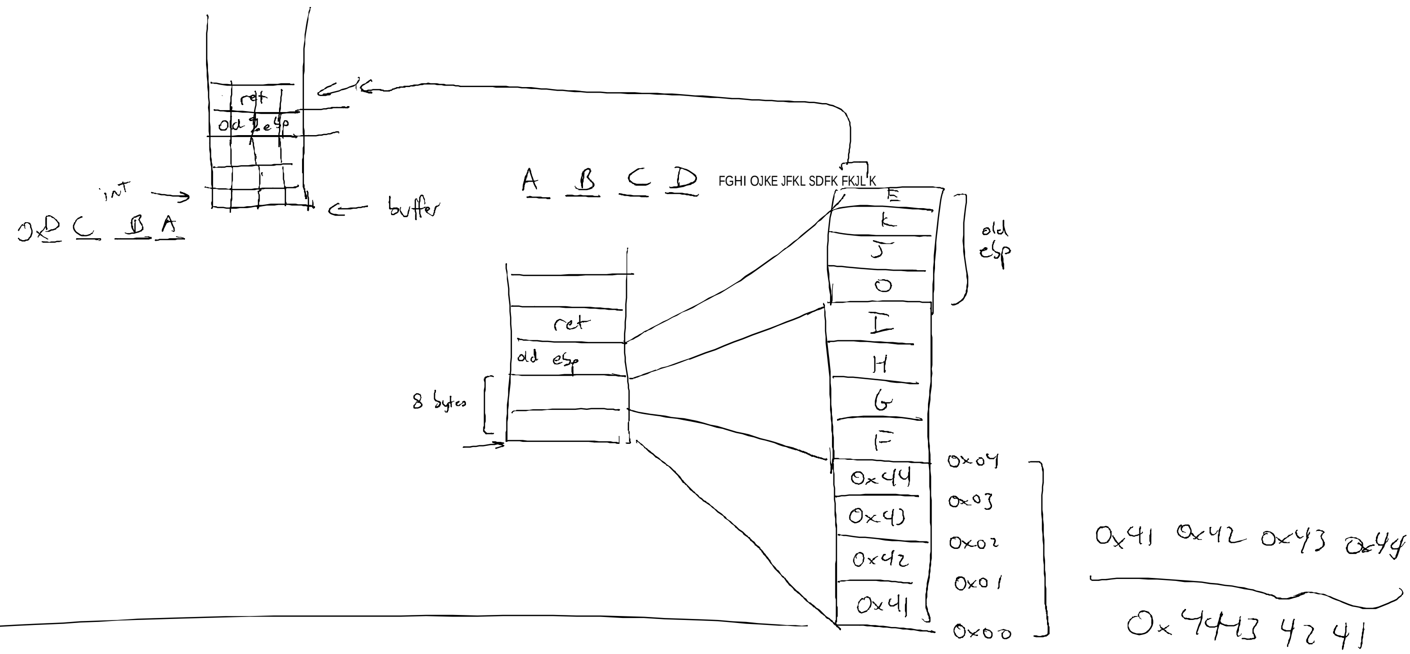
call func

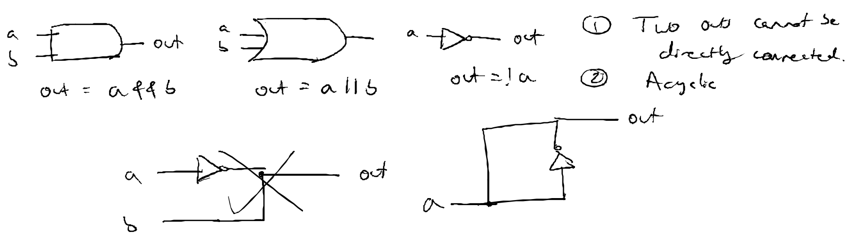
→ pushl %eax

1. restore %esp
2. restore %ebp
3. ret









HCL : Hardware Control Language

bool a, b, c...
int A, B, C

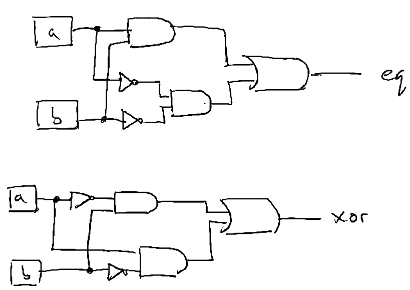
&& || !, ==, !=, <, <=, >=, >

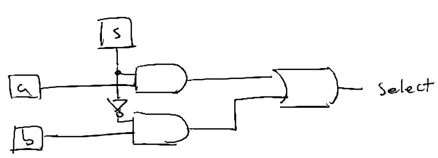
bool eq = (a && b) || (!a && !b)

bool xor = (!a && b) || (a && !b)

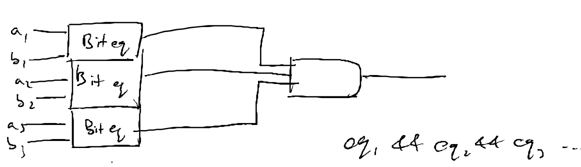
= (a || b) && !(a && b)

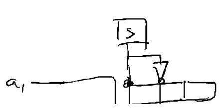
bool select = (s && a) || (!s && b)





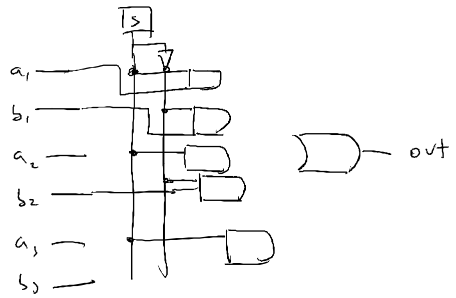
bool eq = (A == B);



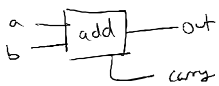


[

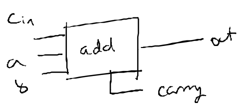
select_1 : expr_1




```
[
    select_1 : expr_1
    select_2 : expr_2
    select_k : expr_k
]

int Out = [
    s: A;
    1: B;
]
                                    00 : A
int Out4 = [                        01 : B
    !s1 && !s2 : A;                 10 : C
    !s1 && s2: B;                   11 : D
    s1 && !s2: C;
    1: D;
]

int Out4 = [
    !s1 && !s2 : A;
    !s1 : B;
    !s2 : C;
    1 : D;
]

int Min3 = [
    A <= B && A <= C : A;
    B <= A && B <= C : B;
    1 : C;
]
```



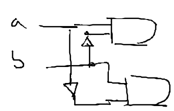
bool out = (a && !b) || (!a && b);
bool carry = (a && b);

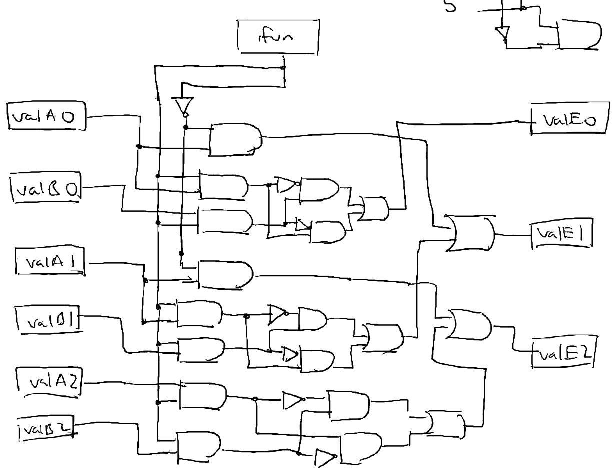


bool out = (a && b && c) || (a && !b && !c) || (!a && b && !c) || (!a && !b && c);
bool out = (a && ((b && c) || (!b && !c))) || (!a && ((b && !c) || (!b && c)))
bool out = (a && !(b xor c)) || (!a && (b xor c))
bool out = a xor b xor c

bool carry = (a && b && c) || (a && b && !c) || (a && !b && c) || (!a && b && c);
bool carry = (a && ((b && c) || (b && !c) || (!b && c))) || (!a && b && c);
bool carry = (a && (b || c)) || (!a && b && c);

Set Membership
bool s1 = input in { 2, 3 };
bool s0 = input in { 1, 3 };





ret

fetch icode : ifun ← M_1[PC]
valP ← PC + 1

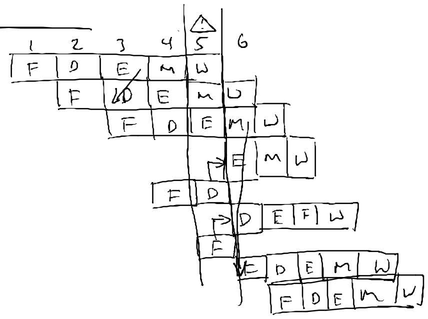
decode valA ← R[%esp]
valB ← R[%esp]

execute valE ← valB + 4;

mem valM ← M_4[valA]

wb R[%esp] ← valE
PC PC ← valM

```
irmovl $0, %eax
andl   %eax, %eax
jne    target

ret

irmovl $8, %ecx
irmovl $5, %ecx
halt
```



Condition	F	D	E	M	W						
Processing ret	stall	bubble	normal	normal	normal						
Load/Use Hazard	stall	stall	bubble	normal	normal						
Combination B	stall	stall	bubble	normal	normal						
	1	2	3	4	5	6	7	8	9	10	11
popl %esp	F	D	E	M	W						
				E	M	W					
ret		F	D	D	E	M	W				
					D	E	M	W			
						D	E	M	W		
							D	E	M	W	
instr			F	F	F	F	F	D	E	M	W
			1) processing ret	1) processing ret	1) processing ret	1) processing ret					
			2) load use								