CS 354
Machine Organization and Programming

Week 8b

Michael Doescher
Spring 2021

Casting Review and Endianness
Bitwise Operations
Binary Arithmetic
File Input/Output
Introduction to Assembly

Bitwise operators

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

Bitwise operators: OR

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
X = 1010   (10)
Y = 0110    (6)
    1110   (14)


X | Y
```

Bitwise operators: AND

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
X = 1010   (10)
Y = 0110    (6)
    0010    (2)


X & Y
```

Bitwise operators: NOT

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
x = 1010   (10)
~x  0101    (5)
```

Bitwise operators: EX-OR

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
X = 1010   (10)
Y = 0110    (6)
    1100   (12)

X ^ Y

Same = 0
Different = 1
```

Bitwise operators: LEFT SHIFT

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
X = 1001 0010
    0010 0000

X << 4  // shift 4 bits to the
left and fill with 0s

Only have storage for 8 bits here

Bits that are in the 4 left most
positions are lost
```

Bitwise operators: RIGHT SHIFT

OR	|
AND	&
NOT	~
EX-OR	^
LEFT SHIFT	<<
RIGHT SHIFT	>>

```
signed X = 1001 0010
           1111 1001

signed Y = 0110 0101
           0000 0110

Arithmetic Shift
X >> 4  // shift 4 bits to the
right and fill with msb


Logical Shift
unsigned X = 1001 0010
             0000 1001

X >> 4  // shift 4 bits to the
right and fill with 0
```

Why?

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

```
Extract the 4 least significant bits
i.e. 0010

X = 1001 0010
```

Bit Masks

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Extract the 4 least significant bits

And with a mask to clear some of the
bits and retain others


X = 1001 0010
   &0000 1111
```

Bit Masks

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Extract the 4 least significant bits

And with a mask to clear some of the
bits and retain others


X = 1001 0010
   &0000 1111
    0000 0010
```

Bit Masks

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Extract the 4 most significant bits
i.e. return 0000 1001


X = 1001 0010
```

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

```
Extract the 4 most significant bits
i.e. return 0000 1001


X =    1001 0010
>> 4   1111 1001
```

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

```
Extract the 4 most significant bits
i.e. return 0000 1001


X =   1001 0010
>> 4  1111 1001
    & 0000 1111
```

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

```
Extract the 4 most significant bits
i.e. return 0000 1001


X =    1001 0010
>> 4   1111 1001
     & 0000 1111
       0000 1001
```

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

Each of these is 1 byte or has values between 0 and 255 (00 to FF)

Bit Masks

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

Each of these is 1 byte or has values between 0 and 255 (00 to FF)

Alpha represents transparency
00 = transparent
FF = opaque

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

Each of these is 1 byte or has values between 0 and 255 (00 to FF)

Alpha represents transparency
00 = transparent
FF = opaque

These color data are frequently packed into a single 4-byte int to save space.

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

Each of these is 1 byte or has values between 0 and 255 (00 to FF)

Alpha represents transparency
00 = transparent
FF = opaque

These color data are frequently packed into a single 4-byte int to save space.

To get the value of blue we just need to extract the b_8-b_{15} bits

Bit Masks

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

To get the value of the green channel we just need to extract the b_{16}-b_{23} bits

```
Color = 0x00 FF 00 FF
```

Bit Masks

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

In computer graphics colors are represented by 4 numbers red, green, blue, and alpha

To get the value of the green channel we just need to extract the b_{16}-b_{23} bits

```
Color = 0x00 FF 00 FF
```

Right shift 16 and mask

```
Green = color >> 16
Green = Green && 0x00 00 00 FF
```

Bit Fields and Flags

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

Bit Flags are a very efficient way of storing Boolean data. I can just use one bit for each piece of data that I want to store.

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

Bit Flags are a very efficient way of storing Boolean data. I can just use one bit for each piece of data that I want to store.

```
int old = 1<<0;               // 00001
int employed = 1<<1;          // 00010
int parent = 1<<2;            // 00100
int married = 1<<3;           // 01000
int drives_ferrari = 1<<4; // 10000
```

Bit Fields and Flags

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

Bit Flags are a very efficient way of storing Boolean data. I can just use one bit for each piece of data that I want to store.

```
int old = 1<<0;            // 00001
int employed = 1<<1;       // 00010
int parent = 1<<2;         // 00100
int married = 1<<3;        // 01000
int drives_ferrari = 1<<4; // 10000

int mike = old | employed | parent;
Mike = 00111
```

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

Bit Flags are a very efficient way of storing Boolean data. I can just use one bit for each piece of data that I want to store.

```
int old = 1<<0;              // 00001
int employed = 1<<1;         // 00010
int parent = 1<<2;           // 00100
int married = 1<<3;          // 01000
int drives_ferrari = 1<<4;  // 10000

int mike = old | employed | parent;
Mike = 00111

if (Mike & old)
      printf(“Wow you’re old”);
```

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles
```

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles

Trick for multiplication / division by
powers of 2
```

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles

14*2 = 28
0000 1110
      * 2
0001 1100
```

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles
```

```
14*2 = 28
0000 1110
      * 2
0001 1100
```

$14 = 2^3 + 2^2 + 2^1$

$28 = 2^4 + 2^3 + 2^2$

Bit Fields and Flags

Bit Masks - Bit Extraction

Multiplication and Division

Bit Flags

Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles
```

```
14*2 = 28
0000 1110
      * 2
0001 1100
```

$14 = 2^3 + 2^2 + 2^1$

$28 = 2^4 + 2^3 + 2^2$

$14 * 2 = (2^3 + 2^2 + 2^1)*2$

$14 * 2 = 2^3*2 + 2^2*2 + 2^1*2$

Bit Fields and Flags

Bit Masks - Bit Extraction
Multiplication and Division
Bit Flags
Packed ints

```
Add / subtract -> Fast 1-2 cpu cycles
Mult           -> Slow 6 cpu cycles
Division       -> Very slow 30-60 cycles

14*2 = 28
0000 1110
       * 2
0001 1100
```

$14 = 2^3 + 2^2 + 2^1$

$28 = 2^4 + 2^3 + 2^2$

$14 * 2 = (2^3 + 2^2 + 2^1)*2$

$14 * 2 = 2^3*2 + 2^2*2 + 2^1*2$

This is the same as just moving all of bits one place to the left.