Scribe Notes

## Updates

- If dying on Pset 2, go to Eddie's marathon OH tomorrow starting 2:30pm
- Grading server is off; will be fixed soon
- Midterm next Thursday! (10/16)

## Machine Programming

- Instruction: a series of bytes that your processor interprets as instructions to complete some task
- Registers: small number of values kept by processor
- Assembly language
    - **.s** files = assembly files

## Assembly File

- each example file contains a single function definition
- How do we figure out the **.c** files from **.s** file?
    - open up "~/cs61-lectures/l11/f00.s" … function does nothing
- **ret** = return from current function (numerically, "c3")
- CPU executes one instruction at a time
    - **2** operands per instruction (source, destination)
- `make f00.o` `// assembles f00.s file`
- `objdump -S f00.o` `// gets hex representation of "ret", which is "c03"`

f01.s … does nothing then returns 0

```
        .file   "f01.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp            // like "push" on a stack
        movl    %esp, %ebp
        movl    $0, %eax        // $0 is source, can't modify const, %eax is
register
        popl    %ebp            // called same amount as "push"
        ret                     // called same amount as "pushl" → stack return to
                                // original state
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

- after optimizing, we don't see register stuff anymore! Why? Eddie's too mysterious to tell us lol

```
        f:
        .LFB0:
                movl    $0, %eax
                ret
```

f02.c … adds 2 ints ("return a + b")

```
        .file   "f02.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    a, %eax
        addl    b, %eax        // addl (src, dest)    same as    dst += src
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

- **NOTE: there are no types anymore.** We are *below* the abstract machine.
- variable names are just placeholders for addresses

f03.c -- unsigned
-- here we have 2 global variables and order of adding them doesn't matter

```
        .file   "f03.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    b, %eax
        addl    a, %eax
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f04.c

- here *a* is int a[ ], while in f03.c, *a* is an int → the effect is the same because of how c arrays work. they both refer to an address
- **Lesson**: you can't determine type in assembly language

- if we compile w/o optimization, very different! ……..

```
        .file   "f04.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    a, %eax
        addl    b, %eax
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f05.c

- **LFB0** = beginning of file
- **LFE0** = end of file
- This code is machine-specific, not C-specific
- What do we think the types are? Address arithmetic with char*

f06.c

- addresses stored in Little Endian (least significant bits stored first)
    - e.g. int x = 1;  hexdump (&x, sizeof(x));
      → 01 00 00 00
- (a & 0xFFFFFFFF) masks off the lower 32 bits of an 8 byte number, which becomes a 32-bit (4-byte) number

```
        .file   "f05.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    a, %eax
        addl    b, %eax
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f06.c

```
	.file	"f06.c"
	.text
	.globl	f
	.type	f, @function
f:
.LFB0:
	movl	b, %eax
	addl	a, %eax
	ret
.LFE0:
	.size	f, .-f
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

f07.c

- return (a-3)

```
	.file	"f07.c"
	.text
	.globl	f
	.type	f, @function
f:
.LFB0:
	movl	a, %eax
	subl	$3, %eax
	ret
.LFE0:
	.size	f, .-f
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

f08.c

- return a + 4294967293 → because this is 32-bit arithmetic, this is “a - 3”

```
	.file	"f08.c"
	.text
	.globl	f
	.type	f, @function
f:
.LFB0:
	movl	a, %eax
	subl	$3, %eax
	ret
.LFE0:
	.size	f, .-f
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

f09.c -- skipped

```
	.file	"f09.c"
	.text
	.globl	f
```

```
	.type	f, @function
f:
.LFB14:
	movl	a, %eax
	subl	$3, %eax
	ret
.LFE14:
	.size	f, .-f
	.section	.rodata.str1.1,"aMS",@progbits,1
.LC0:
	.string	"%d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB15:
	pushl	%ebp
	movl	%esp, %ebp
	andl	$-16, %esp
	subl	$16, %esp
	call	f
	movl	%eax, 8(%esp)
	movl	$.LC0, 4(%esp)
	movl	$1, (%esp)
	call	__printf_chk
	movl	$0, %eax
	leave
	ret
.LFE15:
	.size	main, .-main
	.globl	a
	.bss
	.align 4
	.type	a, @object
	.size	a, 4
a:
	.zero	4
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

f10.c --

-

```
        .file   "f10.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %edx         // x is moved into edx
        movl    y, %eax         // y is moved into eax
        addl    %edx, %eax      // addition
        movl    %eax, a         // put the sum into a
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .globl  g
        .type   g, @function
g:
.LFB1:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %edx
        movl    y, %eax
        subl    %eax, %edx       // subtraction
        movl    %edx, %eax
        movl    %eax, b
        popl    %ebp
        ret
.LFE1:
        .size   g, .-g
        .globl  h
        .type   h, @function
h:
.LFB2:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %edx
        movl    y, %eax
        imull   %edx, %eax        // multiplication
        movl    %eax, c
        popl    %ebp
        ret
.LFE2:
        .size   h, .-h
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

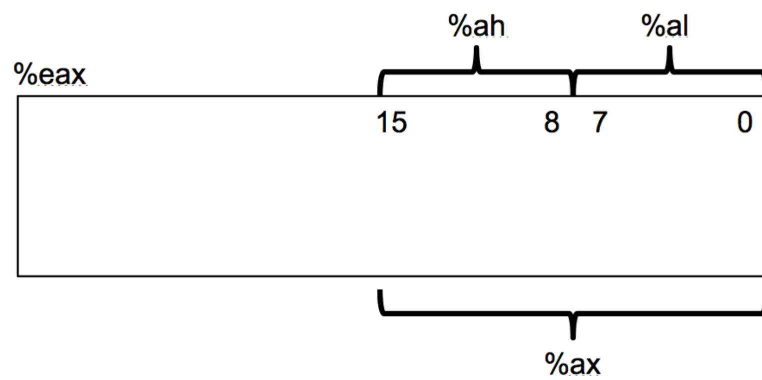
f11.c

```
        .file   "f11.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        movzwl  x, %edx
        movzwl  y, %eax
        addl    %edx, %eax
        movw    %ax, a              // moves a 16-bit qty into a (in x86, word = 16 bits)
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .globl  g
        .type   g, @function
g:                                 // .c version: b = x - y
.LFB1:
        pushl   %ebp
        movl    %esp, %ebp
        movzwl  x, %edx            // moves 16 bits into %edx then extends with 0's to
total 32
        movzwl  y, %eax               bits
        subl    %eax, %edx
        movl    %edx, %eax
        movw    %ax, b
        popl    %ebp
        ret
.LFE1:
        .size   g, .-g
        .globl  h
        .type   h, @function
h:
.LFB2:
        pushl   %ebp
        movl    %esp, %ebp
        movzwl  x, %edx
        movzwl  y, %eax
        imull   %edx, %eax
        movw    %ax, c
        popl    %ebp
        ret
.LFE2:
        .size   h, .-h
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

Sidenote -- registers & register abbreviations

```
%eax → holds the return value
%eax, %ebx, %ecx, and %edx are general registers
%esi
%edi
```

%al → the *low byte* of the %ax register
%ah → the high byte of the %ax register



f12.c

```
	.file	"f12.c"
	.text
	.globl	f
	.type	f, @function
f:
.LFB0:
	pushl	%ebp
	movl	%esp, %ebp
	movzbl	x, %edx
	movzbl	y, %eax
	addl	%edx, %eax
	movb	%al, a			// x and y are bytes (unsigned char)
	popl	%ebp
	ret
.LFE0:
	.size	f, .-f
	.globl	g
	.type	g, @function
g:
.LFB1:
	pushl	%ebp
	movl	%esp, %ebp
	movzbl	x, %edx
	movzbl	y, %eax
	subl	%eax, %edx
	movl	%edx, %eax
	movb	%al, b
	popl	%ebp
	ret
.LFE1:
	.size	g, .-g
	.globl	h
	.type	h, @function
h:
.LFB2:
	pushl	%ebp
	movl	%esp, %ebp
	movzbl	x, %eax
	movzbl	y, %edx
	imull	%edx, %eax
	movb	%al, c
	popl	%ebp
	ret
.LFE2:
	.size	h, .-h
	.ident	"GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
	.section	.note.GNU-stack,"",@progbits
```

f13.c -- skipped

```
	.file	"f13.c"
	.text
	.globl	f
	.type	f, @function
f:
.LFB0:
```

```
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        movl    x, %eax
        movl    y, %ebx
        movl    $0, %edx
        divl    %ebx
        movl    %eax, a
        movl    x, %eax
        movl    y, %ecx
        movl    $0, %edx
        divl    %ecx
        movl    %edx, %eax
        movl    %eax, b
        popl    %ebx
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f14.c --

```
        .file   "f14.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    y, %eax
        andl    x, %eax      // bitwise AND
        movl    %eax, a
        ret
.LFE0:
        .size   f, .-f
        .globl  g
        .type   g, @function
g:
.LFB1:
        movl    y, %eax
        orl     x, %eax       // bitwise OR
        movl    %eax, b
        ret
.LFE1:
        .size   g, .-g
        .globl  h
        .type   h, @function
h:
.LFB2:
        movl    y, %eax
        xorl    x, %eax            // bitwise XOR
        movl    %eax, c
        ret
.LFE2:
        .size   h, .-h
        .globl  k
        .type   k, @function
```

```
k:
.LFB3:
        movl    x, %eax
        notl    %eax
        movl    %eax, d
        ret
.LFE3:
        .size   k, .-k
        .ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section       .note.GNU-stack,"",@progbits
```

f15.c

- most optimized version of **return 0;**
- xor requires 50% less code than mov
  - mov: b8 00 00 00 00
  - xor: 31 c0

```
        .file   "f15.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        xorl    %eax, %eax      // sets %eax to 0 b/c “x xor x” = 0 always
                                 // optimal way to return 0 b/c no data loading into reg
                                 // helps us save 50% of code
        ret
.LFE0:
        .size   f, .-f
        .ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section       .note.GNU-stack,"",@progbits
```

f16.c

```
        .file   "f16.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    x, %eax
        negl    %eax
        movl    %eax, a
        ret
.LFE0:
        .size   f, .-f
        .ident "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section       .note.GNU-stack,"",@progbits
```

f17.c

- compiler optimizes **~x + 1** to **-x**

- `-x == ~x+1                // in 32-bit 2's complement`

```
        .file   "f17.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        movl    x, %eax
        negl    %eax        // "twiddle" (~) x
        movl    %eax, a
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f18.c

- **operations involving 2 registers faster than operations with constants

```
%edx = %eax = x;
%eax << = 10;                // this is the "sall" function, shifting left
a = %edx - %edx
```

What is shifting?

obaBGDE << 3 → obABGDE000 // some bytes may get thrown out

- this file is actually just **multiplication** (a = x * 1023)

```
a = (x<<10) - x              // after subbing in for a = %edx - %edx
```

```
        .file   "f18.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %edx
        movl    %edx, %eax
        sall    $10, %eax           // shift left
        subl    %edx, %eax
        movl    %eax, a
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f19.c

```
        .file   "f19.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %eax
        sall    $10, %eax
        movl    %eax, a
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f20.c

- **shrl** = right shift
  - right shift is division but make sure it's unsigned!
  - moves zeroes to the left side *(number should get smaller with division)*
  - int division & multiplication has signed and unsigned variants (e.g. -1 / 4 returns 0)

```
        .file   "f20.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        pushl   %ebp
        movl    %esp, %ebp
        movl    x, %eax
        shrl    $10, %eax          // logical right shift
        movl    %eax, a
        popl    %ebp
        ret
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
        .section        .note.GNU-stack,"",@progbits
```

f23.c

- integer division is ***much*** more mathematically complicated for the machine