CSE 401 - Compilers
Section 7

2/28/2013
12:30 - MEB 238
1:30 - EE 037

Any Lingering Midterm Questions?

(I also have office hours on Friday)

State of the Project

Project 1 Grading

Grade and comments posted via Catalyst.

Interpreting my comments (-1 point per bullet):

- "Line comment at EOF": Your regex for line comments doesn't work if the line comment the last thing in the file (without a newline before the EOF)
- "Block comment body can't end in "*"": Your regex for block comments doesn't allow the block comment body to end in a "*" character
- "Integers starting with 0": Your regex for integer literals recognizes non-zero integers that start with 0. This is wrong because real Java would interpret these literals as octals, and MiniJava should be a subset of Java.
- "No example [error] output": You didn't provide example output for factorial or a program with an error. 1 point off for each missing example. Another point off if I couldn't find a test program that should produce errors.

Email me with questions (steinz@cs) or come to office hours on Friday.

Project 2 Grading

- TestAST implemented as requested
- Factorial AST printout looks right
- Description of grammar changes
  - No reduce-reduce conflicts
  - Described shift-reduce conflicts
- Description of language extensions (if any)

Anything else I should be looking for?

Project 3

- TestSemantics implemented as requested
- Symbol tables are printed and "look right"
  - Give us some example output
- Type information stored somewhere
- Requested error checks implemented
  - Grading will be test driven
- Error messages printed and "look good"
  - Give us some example output
- INFO file
  - Describe any extensions, additional checks, etc.
  - Describe surprises (anything not implemented?)
  - Any changes to the scanner/parser?

Project 4

Test Driven Development
Source Control

Suggested implementation strategy:

- Expressions, main, and println
- Object creation and simple method calls
- Method parameters and variables
- Control flow
- Class variables and inheritance
- Arrays and anything else (extensions, ...)

Bootstrapping

A small C program that will call your asm code

- Link your .s output file with gcc
- Debug with gdb
- demo.s example in the project writeup

Implements IO and memory management

- Interface with the system here

Does this still feel like magic?

demo.s (+ Fact$ tags)

```
   ...
asm_main: # your main method
   ...
Fact$fact: # method implementation
   ...

   .data
Fact$$: # method table
   .quad 0 # no superclass
   .quad Fact$fact # method pointer
```

Project 4 Testing

Building:

- Your compiler
  - .java -> .s
- gcc
  - .s + bootstrap code -> executable

You'll probably want to script this

Pass args to ant: ant -Dfile=test1.java test-file
Using an arg in build.xml: input="${file}"

Some Review

Objects Representation

Field storage

- Accessed via an offset
  - "this" is passed to methods implicitly
- Contain fields of superclass
  - Even if shadowed (for parent methods)

Pointer to a (per-class) method dispatch table

- Tables built at compile time
- Subclass table starts with parent table
- Runtime lookup when method is called

Object Creation

Steps:

- Get memory
  - Initialization (Java, C, ...)?
- Store method table pointer
- Call a constructor
  - May call superclass' constructor
- Return a pointer

x64

Register and calling conventions in lecture slides

- Arguments passed via registers
- Keep %rsp 16 byte aligned
- Follow conventions (or calls into boot.c might break)!
- Only need to support passing upto 6 args

Symbols in Linux (s) vs Windows, OS X (_s)

- Make sure your code works on Linux (attu)
- Ideally, just develop on Linux (attu, VM)

A Stack Machine

The easiest way to generate working code

Leave results in %rax (the top of the stack)

Push intermediaries when needed
Pop what you push

Some high-level PLs are
stack-oriented too (Joy, Forth,

Other Project Tips

Your compiler can generate comments

- Helps track where the asm comes from

Use semantic names for labels

- while1, if1, else1, etc.
- Consider matching up if/else numbers

Build incrementally and test

Start early

Questions?

x64 Syntax Reminder

```
; Intel/Microsoft prologue
;
push   rbp
mov    rbp,rsp
sub    rsp,16

; Store rdi to frame ptr-8
movq   [rbp-8],rdi
```

```
#   GNU/AT&T  prologue
#
pushq  %rbp
movq   %rsp, %rbp
subq   $16, %rsp

#   Store rdi to frame ptr-8
movq   %rdi,-8(%rbp)
```