CS 33:
Introduction to Computer Organization

Week 4

Pexex Lab: Getting started

- The objective:
  - Examine the execution of an actual program with a debugger.
  - Learn more about more complicated ways of handling arithmetic.
  - Examine the effect of compiling code with -fwrapv, -fsanitize=undefined, and default.
- Essentially, we're forcing you to learn gdb and using debuggers.

  https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

Pexex Lab: Getting started

- You will be examining the execution of Emacs, the text editor.
- Emacs also provides an interpreter for handling “Elisp”, a functional programming language that allows for simple computation.

Pexex Lab: Getting started

- You will concern yourselves with the following files which are located on the SEASnet server.
- Yup, for this one, you've really got no choice but to use SEASnet, preferably lnxsrv09.
- Emacs executable located in:
  - ~eggert/bin64/bin/emacs-24.5
- Emacs source code located in:
  - ~eggert/src/emacs-24.5/

Pexex Lab: Getting started

- In particular, you will examine the execution of the following invocation of Emacs:
  - ~eggert/bin64/bin/emacs-24.5 -batch -eval '(print (* 6997 -4398042316799 179))'
- You will run this command with gdb via the following:
  - gdb --args ~eggert/bin64/bin/emacs-24.5 -batch -eval '(print (* 6997 -4398042316799 179))'
- Or...
  - gdb ~eggert/bin64/bin/emacs-24.5
  - (gdb) r -batch -eval '(print (* 6997 -4398042316799 179))'

Pexex Lab: gdb quick start

- After opening gdb:
- Run the program
  - run (or simply 'r')
- Run the program with arguments
  - r arg1 arg2
- This will run the executable to completion.

Pexex Lab: gdb quick start

- Set break point at function foo:
  - break foo
- Set break point at instruction address 0x100
  - break *0x100
  - Note the asterisk. Without it, it will look for a function called 0x100. Please don't write a function called 0x100.
- When program is run or “continued”, it will run until it hits a break point in which it will stop.
- Resume a program that is stopped
  - continue (or just 'c')

Pexex Lab: gdb quick start

- When a program is break-ed you can step through each instruction either for each assembly instruction or each high level C instruction.
- Execute the next instruction, stepping INTO functions:
  - stepi (or just 'si')
- Execute the next line of source code, stepping INTO functions:
  - step (or just 's')

Pexex Lab: gdb quick start

- Execute the next instruction, stepping OVER functions:
  - nexti (or just 'ni')
- Execute the next line of source code, stepping OVER functions:
  - next (or just 'n')
- Stepping INTO functions means any time you call another function, you will descend into the function. Stepping OVER functions meaning stepping over the function as if it were a single instruction.

Pexex Lab: gdb quick start

- Examining the assembly instructions of function foo:
  - disassemble foo (or just “disas foo”)
- If you are stopped at some instruction in “foo”, the disassembled assembly will also show you where execution is paused at:

```
   0x54352e <arith_driver+46>: 49 89 d7 mov  %rdx,%r15
=> 0x543531 <arith_driver+49>: 49 89 f5 mov  %rsi,%r13
   0x543534 <arith_driver+52>: 41 89 fe mov  %edi,%r14d
```

- This means arith_driver+46 has been executed but arith_driver+49 has not yet been executed.

Pexex Lab: gdb quick start

- disas /m <function name>
  - Display assembly for <function name> prefaced with the corresponding lines of C.
- Using next(i) or step(i), you can also set the debugger to print each instruction as it's executed:
  - set disassemble-next-line on

Pexex Lab: gdb quick start

- Examining registers:
- Print the current state of all registers:
  - info registers
- Print the state of a particular register:
  - info registers $rdi

Pexex Lab: gdb quick start

- Print out contents at some memory address with the “x” command.
  - x [addr]

Pexex Lab: gdb quick start

- For more options, use x/nfu [addr] where
  - n specifies how much memory to display (in terms of the unit specified by u), default 1
  - f specifies the format (ie decimal, hexadecimal, etc.), default hex
  - u specifies the unit size for each address (words, bytes, etc.), default word

Pexex Lab: gdb quick start

- Also, print out memory based on an address stored in an register:
- Ex: x/20xw $rsp
  - Print out 20 words(w) in hex(x) starting from the address stored in %rsp.

Pexex Lab: TODO

- Set a break point at Ftimes.
- For each instruction until Ftimes completes, examine what each instruction is doing, checking the status of the registers and memory as necessary.
- Answer the questions.

Pexex Lab: TODO

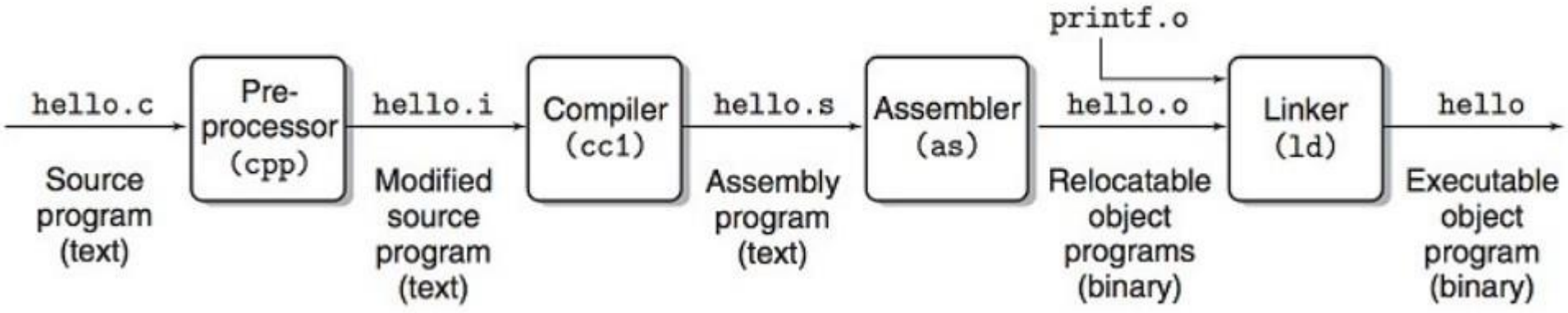
- testovf function
- gcc -S -fno-asynchronous-unwind-tables <ADDITIONAL FLAGS> testovf.c
  - This produces testovf.s which you can read with any text editor.
- gcc -c <ADDITIONAL FLAGS> testovf.c
  - This produces testovf.o, which you will have to read using “objdump -d testovf.o”. To save the output of object dump, use “objdump -d testovf.o > testovf.txt”

Pexex Lab: TODO

- gcc <ADDITIONAL FLAGS> testovf.c
  - This produces a.out, which you can examine with “gdb a.out”.
- If you use the -S or -c options, you don't need to include a “main” function. However, if you compile to completion (no -S or -c), you will need to include a main function.
- What is all this?

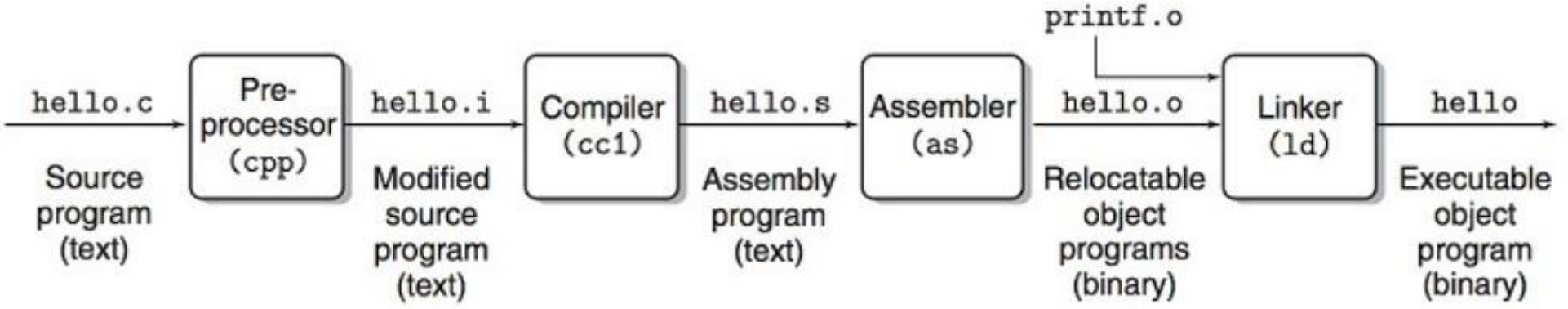
Pexex Lab: TODO

- Recall from Chapter 1 that compilation occurs over several different steps.



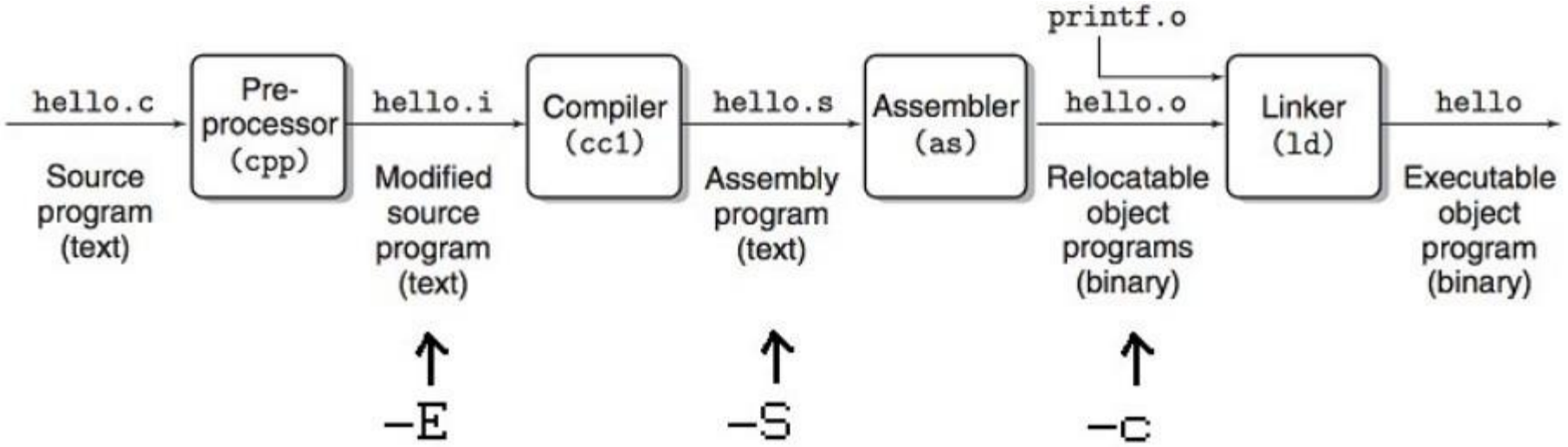
- The result of the Pre-processor step is a modified source with the preprocessor directives(#define, #include) replaced

Pexex Lab: TODO



- The result of the Compiler step is compiled code, which is readable assembly
- The result of the Assembler step is the assembled code which is a binary file.
- Finally, the result of the linker is a fully executable file.
- gcc allows you to compile up to certain steps

Pexex Lab: TODO



- Ex: gcc -E [filename] will get you the modified source file
- Note: Using -E and -S will get you files that you can read with a text editor. To read the output of -c, use objdump.
- To read/disassemble the final executable, use gdb