

Interpreters, compilers and assembly language

Lecture B07



Lecture notes section B07

2002-01-14

CSEI 303 Part B lecture notes

1

Last time

- Pointers
- Aggregation of data
 - arrays
 - structs
- Characters
- Strings

2002-01-14

CSEI 303 Part B lecture notes

2

In this lecture

- Interpreters
- Machine language
- Assembly language
 - structure
- Compilers
 - what a compiler does
- Stages of code generation
 - compiling, assembling, linking
 - compiling multi-file programs

2002-01-14

CSEI 303 Part B lecture notes

3

Interpreters

- Interpreter is a program which can understand and run programs written in a language
- Shell (`/bin/sh` in Unix) interprets and runs command line
 - loop:
 - read command line from user ("fetch")
 - examine command line ("decode")
 - perform appropriate commands ("execute")
 - repeat loop

2002-01-14

CSEI 303 Part B lecture notes

4

Interpreters

- Advantages of interpreters
 - easy to write
 - flexible
 - easy to add extra functionality
 - easy to debug interpreted program
 - breakpoints and single-stepping
- Problems with interpreters
 - slow
 - every line must be interpreted every time it is run
 - fetch-decode-execute must be performed in software
 - inefficient
 - must allocate memory for interpreter and program

2002-01-14

CSEI 303 Part B lecture notes

5

Interpreters

- What language is the interpreter written in?

2002-01-14

CSEI 303 Part B lecture notes

6

Machine language

- Ultimately, computer code must run on hardware
 - programs must be storable in memory as patterns of bits
 - programs must be expressed in very small, easily-performed instructions
 - this language called machine language

2002-01-14 CSEI 303 Part B lecture notes 7

Machine language

Function to calculate factorial:

```

00110100000000100000000000000001
00101000010000010000000000000001
0001010000000010000000000000100
00000001000001000000000000011000
0000000000000000000100000010010
00100000100001001111111111111111
000100000000000011111111111010
00000111110000000000000001000
  
```

2002-01-14 CSEI 303 Part B lecture notes 8

Assembly language

- Problems with machine language
 - difficult to write
 - difficult to read
- Need a compromise
 - language that humans can cope with
 - supports comments, naming of variables
 - human-readable instructions
 - but is easily converted to machine language
 - 1-to-1 relationship between instructions
 - ideally, conversion done by a computer program that won't make mistakes
 - this language called assembly language

2002-01-14 CSEI 303 Part B lecture notes 9

Assembly language

- Function to calculate factorial:

```

# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li    $v0, 1      # Result.
loop: blt  $a0, 1, end
      mul  $v0, $v0, $a0
      sub  $a0, $a0, 1
      b   loop
end:   j    $ra        # Return.
  
```

2002-01-14 CSEI 303 Part B lecture notes 10

Assembly language

comment: starts at #, extends to end of line.

```

# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li    $v0, 1      # Result.
loop: blt  $a0, 1, end
      mul  $v0, $v0, $a0
      sub  $a0, $a0, 1
      b   loop
end:   j    $ra        # Return.
  
```

2002-01-14 CSEI 303 Part B lecture notes 11

Assembly language

label: identifies a line of the program ...

... so that other lines can refer to it.

```

# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li    $v0, 1      # Result.
loop: blt  $a0, 1, end
      mul  $v0, $v0, $a0
      sub  $a0, $a0, 1
      b   loop
end:   j    $ra        # Return.
  
```

2002-01-14 CSEI 303 Part B lecture notes 12

Assembly language

instruction: one per line (this one performs a subtraction)

```
# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li $v0, 1 # Result.
loop: blt $a0, 1, end
      mul $v0, $v0, $a0
      sub $a0, $a0, 1
      b loop
end: j $ra # Return.
```

2002-01-14

CSEI 303 Part B lecture notes

13

Assembly language

opcode (operation code): describes kind of instruction (here, "branch if less than")

```
# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li $v0, 1 # Result.
loop: blt $a0, 1, end
      mul $v0, $v0, $a0
      sub $a0, $a0, 1
      b loop
end: j $ra # Return.
```

2002-01-14

CSEI 303 Part B lecture notes

14

Assembly language

register: one of a few fast memory words located on CPU

immediate (literal) value: here, number 1

```
# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li $v0, 1 # Result.
loop: blt $a0, 1, end
      mul $v0, $v0, $a0
      sub $a0, $a0, 1
      b loop
end: j $ra # Return.
```

collectively these are called **operands**

2002-01-14

CSEI 303 Part B lecture notes

15

Assembly language

assembler directive: start with . character, one per line (this one says program text follows)

```
# Factorial: compute factorial of
# $a0, return result in $v0.
.text
fact: li $v0, 1 # Result.
loop: blt $a0, 1, end
      mul $v0, $v0, $a0
      sub $a0, $a0, 1
      b loop
end: j $ra # Return.
```

2002-01-14

CSEI 303 Part B lecture notes

16

Assembly language

- Why learn assembly language?
 - convenient halfway stop between high-level language (e.g., C) and machine language
 - break down problem of running C programs into two smaller problems (C ⇔ assembly, assembly ⇔ machine language)
 - sometimes necessary to use in low-level programming or when timing is critical
 - e.g., device drivers

2002-01-14

CSEI 303 Part B lecture notes

17

Assembly language

- Programs already exist which already turn C into machine language
 - compilers
- Most people don't write low-level device drivers
- So, why learn assembly language?

2002-01-14

CSEI 303 Part B lecture notes

18

Assembly language

- Why learn assembly language?
 - to understand how compilers work
 - to understand how to write effective C (and other languages)
 - efficient
 - correct
 - portable
 - to become a more valuable programmer

2002-01-14

CSE1 303 Part B lecture notes

19

Compilers

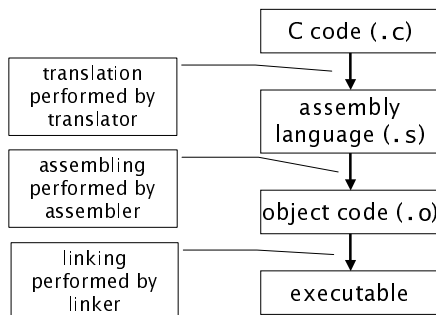
- Compiler is a set of tools (programs) which convert high-level code (e.g., C) to machine language
 - translator
 - C to assembly language
 - assembler
 - assembly language to object code
 - linker
 - object code to executable program
- Most compilers (e.g., Borland C++, GCC) can perform any or all of the above steps
 - options can halt processing at any stage

2002-01-14

CSE1 303 Part B lecture notes

20

Compilers



2002-01-14

CSE1 303 Part B lecture notes

21

Compilers

- Some programs may refer to variables or functions defined elsewhere
 - e.g., printf or strlen functions
- Object code is code which has been compiled as far as it can without resolving these references
 - essentially machine language, but with additional data identifying symbols still needing resolving
- Linking is resolving these references so that they refer to the correct places
 - produces pure machine language executable

2002-01-14

CSE1 303 Part B lecture notes

22

Compilers

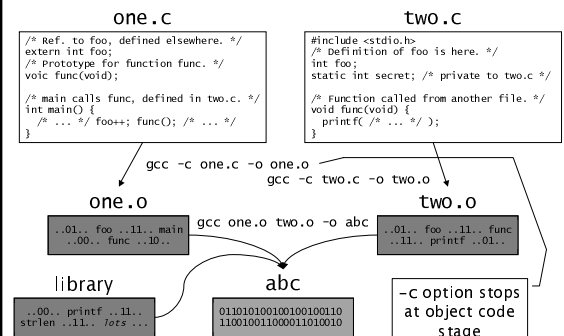
- Programs may be made from many source files
 - some programs are millions of lines long, too long for one file
- Separate files can be compiled to the object code stage independently of the others
 - if one file is changed, other files don't need recompiling
- Multiple object files are linked simultaneously to form executable
 - including libraries written by OS vendor

2002-01-14

CSE1 303 Part B lecture notes

23

Compilers



2002-01-14

CSE1 303 Part B lecture notes

24

Compilers

- To write multi-file C programs
 - if file `one.c` needs to call function in `two.c`, put function prototype in `one.c`
 - `void func(void);`
 - compiler now knows enough about function to compile `one.c`
 - linker will locate the function during linking

2002-01-14

CSE1 303 Part B lecture notes

25

Compilers

- To write multi-file C programs
 - if file `one.c` needs to refer to global variable in `two.c`, declare it in `one.c` with `extern` modifier
 - `extern int foo;`
 - compiler tags variable as being outside file, and does not try to make another variable with the same name in `one.c`
 - linker will associate `one.c`'s variable with `two.c`'s variable during linking

2002-01-14

CSE1 303 Part B lecture notes

26

Compilers

- To write multi-file C programs
 - if file `two.c` wants to keep a global variable or function private (inaccessible outside the file), declare it with the `static` modifier
 - `static int secret;`
 - `static void mine(char* x);`
 - compiler will remove all references to the variable or function in object file
 - other files cannot refer to it, even with `extern` modifier
 - files in same program may each have `static` variable/function of the same name

2002-01-14

CSE1 303 Part B lecture notes

27

Compilers

- To compile multi-file C programs
 - compile each source file to the object code stage
 - with GCC, use `-c` option
 - `gcc -c file1.c -o file1.o`
 - creates object code called `file1.o`
 - link all object files together
 - with GCC, no option needed provided files have `.o` suffix
 - `gcc file1.o file2.o file3.o -o final`
 - creates executable file called `final`
 - link with libraries too at this stage (here, math lib)
 - `gcc file1.o file2.o file3.o -lm -o final`

2002-01-14

CSE1 303 Part B lecture notes

28

Compilers

- To compile multi-file C programs
 - if all files are small, let GCC compile all of them at once to executable file
 - `gcc file1.c file2.c file3.c -o final`
 - not efficient but easier to type

2002-01-14

CSE1 303 Part B lecture notes

29

Covered in this lecture

- Interpreters
- Machine language
- Assembly language
 - structure
- Compilers
 - what a compiler does
- Stages of code generation
 - compiling, assembling, linking
 - compiling multi-file programs

2002-01-14

CSE1 303 Part B lecture notes

30

Going further

- Object file format
 - getting your hands dirty with object code
- make
 - a tool that makes managing multi-file programs easy
 - uses file date stamps to determine which files need updating
 - man make (manual page)

2002-01-14

CSE1 303 Part B lecture notes

31

Next time

- MIPS architecture
 - registers
 - memory
- Running MIPS programs
 - fetch-execute cycle
 - SPIM simulator



Reading:
lecture notes section B08

2002-01-14

CSE1 303 Part B lecture notes

32

Copyright

Copyright © 2001 Deborah Pickett.
No part of this presentation may be
duplicated without permission from
the author.

2002-01-14

CSE1 303 Part B lecture notes

33