Assemblers, Linkers, and Loaders

Anne Bracy

CS 3410

Computer Science
Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, McKee, and Sirer.

See: P&H Appendix A1-2, A.3-4 and 2.12
When most people say “compile,” they mean the entire process: compile + assemble + link

“It’s alive!”

Executing in Memory process

When most people say “compile” they mean the entire process: compile + assemble + link
From Writing to Running: Command Line

# Compile
csug01> mipsel-linux-gcc -S sum.c

# Assemble
csug01> mipsel-linux-gcc -c sum.s

# Link
csug01> mipsel-linux-gcc -o sum sum.o \${LINKFLAGS}
# -nostartfiles -nodefaultlibs
# -static -mno-xgot -mno-embedded-pic
# -mno-abicalls -G 0 -DMIPS -Wall

# Load
csug01> simulate sum
Sum 1 to 100 is 5050
MIPS program exits with status 0 (approx. 2007 instructions in 143000 nsec at 14.14034 MHz)
#include <stdio.h>

int n = 100;

int main (int argc, char* argv[ ]) {
    int i;
    int m = n;
    int sum = 0;

    for (i = 1; i <= m; i++) {
        sum += i;
    }

    printf ("Sum 1 to %d is %d\n", n, sum);
}

csug03> mipsel-linux-gcc -S sum.c
export PATH=${PATH}:/courses/cs3410/mipsel-linux/bin:/courses/cs3410/mips-sim/bin
or
setenv PATH ${PATH}:/courses/cs3410/mipsel-linux/bin:/courses/cs3410/mips-sim/bin

tells compiler to produce sum.s file
sum.s

.data
.globl n
.align 2
.word 100

.globl main
.align 2
.globl n

.text
.align 2
.globl main

main: addiu $sp,$sp,-48
sw $31,44($sp)
sw $fp,40($sp)  
move $fp,$sp
sw $4,48($fp)
sw $5,52($fp)
al $2,n
lw $2,0($2)
sw $2,28($fp)
sw $0,32($fp)
li $2,1
sw $2,24($fp)

$L2: lw $2,24($fp)
      lw $3,28($fp)
      slt $2,$3,$2
      bne $2,$0,$L3
      lw $3,32($fp)
      lw $2,24($fp)
      addu $2,$3,$2
      sw $2,32($fp)
      lw $2,24($fp)
      addiu $2,$2,1
      sw $2,24($fp)
      lw $2,24($fp)
      lw $2,24($fp)
      la $4,$str0
      lw $5,28($fp)
      lw $6,32($fp)
      jal printf
      move $sp,$fp
      lw $31,44($sp)
      lw $fp,40($sp)
      addiu $sp,$sp,48
      j $31

$str0: .asciiz
   "Sum 1 to %d is %d\n"

.globl printf
.data
.globl n
.align 2
.word 100
.rdata
.align 2
.globl main
.text
.align 2
.globl main
main: addiu $sp,$sp,-48
    sw $31,44($sp)
    sw $fp,40($sp)
    move $fp,$sp
    sw $a0 $4,48($fp)
    sw $a1 $5,52($fp)
    la $v0 $2,n
    lw $2,0($2) $v0=100
    sw $2,28($fp) m=100
    sw $0,32($fp) sum=0
    li $2,1
    sw $2,24($fp) i=1

.prologue
sum.s
prologue
epilogue
$L2: lw $2,24($fp) i=1
    lw $3,28($fp) m=100
    slt $2,$3,$2 if(m < i)
    bne $2,$0,$L3 100 < 1
    lw $3,32($fp)v1=0(sum)
    lw $2,24($fp) v0=1(i)
    addu $2,$3,$2 v0=1(0+1)
    sw $2,32($fp) sum=1
    lw $2,24($fp) i=1
    addiu $2,$2,1 i=2 (1+1)
    sw $2,24($fp) i=2

$L3: la $a0 $4,$str0 str
    call printf
    lw $a1 $5,28($fp)m=100
    lw $a2 $6,32($fp)sum
    jal printf
    move $sp,$fp
    lw $31,44($sp)
    lw $fp,40($sp)
    addiu $sp,$sp,48
    j $31

$str0: .asciiz
    "Sum 1 to %d is %d\n"

.data
.globl n
.align 2
.n: .word 100
.rdata
.globl main
.text
.align 2
.globl main
main: addiu $sp,$sp,-48
    sw $31,44($sp)
    sw $fp,40($sp)
    move $fp,$sp
    sw $a0 $4,48($fp)
    sw $a1 $5,52($fp)
    la $v0 $2,n
    lw $2,0($2) $v0=100
    sw $2,28($fp) m=100
    sw $0,32($fp) sum=0
    li $2,1
    sw $2,24($fp) i=1

.prologue
sum.s
prologue
epilogue
$L2: lw $2,24($fp) i=1
    lw $3,28($fp) m=100
    slt $2,$3,$2 if(m < i)
    bne $2,$0,$L3 100 < 1
    lw $3,32($fp)v1=0(sum)
    lw $2,24($fp) v0=1(i)
    addu $2,$3,$2 v0=1(0+1)
    sw $2,32($fp) sum=1
    lw $2,24($fp) i=1
    addiu $2,$2,1 i=2 (1+1)
    sw $2,24($fp) i=2

$L3: la $a0 $4,$str0 str
    call printf
    lw $a1 $5,28($fp)m=100
    lw $a2 $6,32($fp)sum
    jal printf
    move $sp,$fp
    lw $31,44($sp)
    lw $fp,40($sp)
    addiu $sp,$sp,48
    j $31

$str0: .asciiz
    "Sum 1 to %d is %d\n"

.data
.globl n
.align 2
.n: .word 100
.rdata
.globl main
.text
.align 2
.globl main
main: addiu $sp,$sp,-48
    sw $31,44($sp)
    sw $fp,40($sp)
    move $fp,$sp
    sw $a0 $4,48($fp)
    sw $a1 $5,52($fp)
    la $v0 $2,n
    lw $2,0($2) $v0=100
    sw $2,28($fp) m=100
    sw $0,32($fp) sum=0
    li $2,1
    sw $2,24($fp) i=1

.prologue
sum.s
prologue
epilogue
$L2: lw $2,24($fp) i=1
    lw $3,28($fp) m=100
    slt $2,$3,$2 if(m < i)
    bne $2,$0,$L3 100 < 1
    lw $3,32($fp)v1=0(sum)
    lw $2,24($fp) v0=1(i)
    addu $2,$3,$2 v0=1(0+1)
    sw $2,32($fp) sum=1
    lw $2,24($fp) i=1
    addiu $2,$2,1 i=2 (1+1)
    sw $2,24($fp) i=2

$L3: la $a0 $4,$str0 str
    call printf
    lw $a1 $5,28($fp)m=100
    lw $a2 $6,32($fp)sum
    jal printf
    move $sp,$fp
    lw $31,44($sp)
    lw $fp,40($sp)
    addiu $sp,$sp,48
    j $31

$str0: .asciiz
    "Sum 1 to %d is %d\n"
Assembler

Input: Assembly File (.s)
- assembly instructions, pseudo-instructions
- program data (strings, variables), layout directives

Output: Object File in binary machine code
- MIPS instructions in executable form
  (.o file in Unix, .obj in Windows)

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

```
00100000000001010000000000001010
00000000000000010100101000010000
00100001010010100000000000001111
```
MIPS Assembly Instructions

Arithmetic/Logical
- ADD, ADDU, SUB, SUBU, AND, OR, XOR, NOR, SLT, SLTU
- ADDI, ADDIU, ANDI, ORI, XORI, LUI, SLL, SRL, SLLV, SRLV, SRAV, SLTI, SLTIU
- MULT, DIV, MFLO, MTLO, MFHI, MTHI

Memory Access
- LW, LH, LB, LHU, LBU, LWL, LWR
- SW, SH, SB, SWL, SWR

Control flow
- BEQ, BNE, BLEZ, BLTZ, BGEZ, BGTZ
- J, JR, JAL, JALR, BEQL, BNEL, BLEZL, BGTZL

Special
- LL, SC, SYSCALL, BREAK, SYNC, COPROC
# Pseudo-Instructions

Assembly shorthand, technically not machine instructions, but easily converted into 1+ instructions that are

<table>
<thead>
<tr>
<th>Pseudo-Insns</th>
<th>Actual Insns</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>SLL r0, r0, 0</td>
<td># do nothing</td>
</tr>
<tr>
<td>MOVE reg, reg</td>
<td>ADD r2, r0, r1</td>
<td># copy between regs</td>
</tr>
<tr>
<td>LI reg, 0x45678</td>
<td>LUII reg, 0x4</td>
<td># load immediate</td>
</tr>
<tr>
<td></td>
<td>ORI reg, reg, 0x5678</td>
<td></td>
</tr>
<tr>
<td>BLT reg, reg, label</td>
<td>SLT r1, rA, rB</td>
<td># branch less than</td>
</tr>
<tr>
<td></td>
<td>BNE r1, r0, label</td>
<td></td>
</tr>
</tbody>
</table>

+ a few more...
Symbols and References

Global labels: Externally visible “exported” symbols
- Can be referenced from other object files
- Exported functions, global variables
- Examples: pi, e, username, printf, pick_prime, pick_random

Local labels: Internally visible only symbols
- Only used within this object file
- Static functions, static variables, loop labels, ...
- Examples: randomval, is_prime

```c
int pi = 3;
int e = 2;
static int randomval = 7;
    (external == defined in another file)
extern char *username;
extern int printf(char *str, ...);
int square(int x) { ... }
static int is_prime(int x) { ... }
int pick_prime() { ... }
int pick_random() {
    return randomval;
}
```
Handling forward references

Example:

```
bne $1, $2, L
sll $0, $0, 0
L: addiu $2, $3, 0x2
```

Looking for L

```
Found L
```

The assembler will change this to

```
bne $1, $2, +1
sll $0, $0, 0
addiu $2, $3, $0x2
```

Final machine code

```
0X14220001 # bne
0x00000000 # sll
0x24620002 # addiu
```
Object file

Header
  • Size and position of pieces of file

Text Segment
  • instructions

Data Segment
  • static data (local/global vars, strings, constants)

Debugging Information
  • line number \(\rightarrow\) code address map, *etc.*

Symbol Table
  • External (exported) references
  • Unresolved (imported) references
Object File Formats

Unix

- a.out
- COFF: Common Object File Format
- ELF: Executable and Linking Format
- ...

Windows

- PE: Portable Executable

All support both executable and object files
Objdump disassembly

csug01> mipsel-linux-objdump --disassemble math.o

math.o: file format elf32-tradlittlemips

Disassembly of section .text:

00000000 <pick_random>:
  0: 27bdfff8 addiu sp,sp,-8
  4: afbe0000 sw s8,0(sp)
  8: 03a0f021 move s8,sp
 c: 3c020000 lui v0,0x0
10: 8c420008 lw v0,8(v0)
14: 03c0e821 move sp,s8
18: 8fbe0000 lw s8,0(sp)
1c: 27bd0008 addiu sp,sp,8
20: 03e00008 jr ra
24: 00000000 nop

static int randomval = 7;
int pick_random() { return randomval; }
Objdump symbols

csug01 ~$ mipsel-linux-objdump --syms math.o

symbol format elf32-tradlittlemips

SYMBOL TABLE:

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Symbol</th>
<th>File</th>
<th>Segment Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>l</td>
<td>df</td>
<td><em>ABS</em></td>
<td>00000000 math.c</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.text</td>
<td>00000000 .text</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.data</td>
<td>00000000 .data</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.bss</td>
<td>00000000 .bss</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.mdebug.abi32</td>
<td>00000000 .mdebug.abi32</td>
</tr>
<tr>
<td>00000008</td>
<td>l</td>
<td>O</td>
<td>.data</td>
<td>00000004 randomval</td>
</tr>
<tr>
<td>00000060</td>
<td>l</td>
<td>F</td>
<td>.text</td>
<td>00000028 is_prime local</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.rodata</td>
<td>00000000 .rodata</td>
</tr>
<tr>
<td>00000000</td>
<td>l</td>
<td>d</td>
<td>.comment</td>
<td>00000000 .comment</td>
</tr>
<tr>
<td>00000000</td>
<td>g</td>
<td>O</td>
<td>.data</td>
<td>00000004 pi</td>
</tr>
<tr>
<td>00000000</td>
<td>g</td>
<td>O</td>
<td>.data</td>
<td>00000004 e</td>
</tr>
<tr>
<td>00000000</td>
<td>g</td>
<td>F</td>
<td>.text</td>
<td>00000028 pick_random</td>
</tr>
<tr>
<td>00000028</td>
<td>g</td>
<td>F</td>
<td>.text</td>
<td>00000038 square</td>
</tr>
<tr>
<td>00000088</td>
<td>g</td>
<td>F</td>
<td>.text</td>
<td>0000004c pick_prime</td>
</tr>
<tr>
<td>00000000</td>
<td>g</td>
<td><em>UND</em></td>
<td></td>
<td>00000000 username</td>
</tr>
<tr>
<td>00000000</td>
<td>g</td>
<td><em>UND</em></td>
<td></td>
<td>00000000 printf</td>
</tr>
</tbody>
</table>

external references (undefined)
Separate Compilation & Assembly

Compiler

source files
sum.c

Assembler

assembly files
sum.s

math.c
math.s

math.o

Linker

obj files
sum.o

Executable program

sum

exists on disk

Loader

Executing in Memory process

small change?
→ recompile one module only

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.

http://xkcd.com/303/
Linkers

Linker combines object files into an executable file

- Resolve as-yet-unresolved symbols
- Each has illusion of own address space
  - Relocate each object’s text and data segments
- Record top-level entry point in executable file

End result: a program on disk, ready to execute

- E.g. ./sum Linux
  ./sum.exe Windows
  simulate sum Class MIPS simulator
Static Libraries

*Static Library*: Collection of object files (think: like a zip archive)

Q: Every program contains the entire library?!?
A: No, Linker picks only object files needed to resolve undefined references at link time

e.g. libc.a contains many objects:
- printf.o, fprintf.o, vprintf.o, sprintf.o, snprintf.o, ...
- read.o, write.o, open.o, close.o, mkdir.o, readdir.o, ...
- rand.o, exit.o, sleep.o, time.o, ....
Linker Example: Resolving an External Fn Call

main.o

.text
40 0C000000*
44 21035000
48 1b80050C
4C 8C040000
50 21047002
54 0C000000*

Relocation info Symbol table
00 T main
00 D usr
*UND* printf
*UND* pi
*UND* square
40,JAL, printf
54,JAL, square

... JAL printf → JAL ???
Unresolved references to printf and square

math.o

.text
24 21032040
28 0C000000*
2C 1b301402
30 3C040000
34 34040000

... T square
00 D pi
*UND* printf
*UND* usr
28,JAL, printf

printf.o

.text
... 3C T printf

sum.exe

.text
0040 0000 21032040
0C40023C
1b301402
3C041000
34040004

.data
Entry:0040 0100
text:0040 0000
data:1000 0000

... math
... main
... printf
...
Which symbols are undefined according to both main.o and math.o's symbol table?

A) printf 
B) pi 
C) square 
D) usr 
E) printf & pi
Linker Example: Resolving Global Variables (1)

```
main.o
.text
00 T main
00 D usr
*UND* printf
*UND* pi
*UND* square
40,JAL, printf
4C,LW/gp, pi
54,JAL, square

math.o
.text
24 21032040
28 0C000000
2C 1b301402
30 3C040000
34 34040000

20 T square
00 D pi
*UND* printf
*UND* usr
28,JAL, printf
30,LUI, usr
34,LA, usr

sum.exe
.text
21032040
0C40023C
1b301402
3C041000
34040004

math
0C40023C
21035000
1b80050c
8C048004
21047002
0C400020

main
10201000
21040330
22500102

printf
00000003
Entry:0040 0100
text:0040 0000
data:1000 0000
```

☆ LW $4 "pi" → LW $4 ???
Unresolved reference to pi
**Linker Example: Resolving Global Variables (2)**

### main.o

- **.text**
  - 40 0C000000
  - 44 21035000
  - 48 1b80050c
  - 4C 8C040000
  - 50 21047002
  - 54 0C000000
  - ...

- **Relocation info Symbol table**
  - 00 T main
  - 00 D usr
  - *UND* printf
  - *UND* pi
  - *UND* square
  - 40,JAL, printf
  - 4C,LW/gp, pi
  - 54,JAL, square

- **Starred Note:** LA = LUI/ORI "usr" → ??? Unresolved reference to us

### math.o

- **.text**
  - 24 21032040
  - 28 0C000000
  - 2C 1b301402
  - 30 3C040000
  - 34 34040000
  - 20 T square
  - 00 D pi
  - *UND* printf
  - *UND* usr

  - 28,JAL, printf
  - 30,LUI, usr
  - 34,LA, usr

- **Starred Note:** Relocation info
  - 28,JAL, printf
  - 30,LUI, usr
  - 34,LA, usr

### sum.exe

- **.text**
  - 0040 0000
  - 0040 0100
  - 0040 0200

- **.data**
  - 1000 0000

- **Relocation info**
  - LA num: LUI 1000 ORI 0004

- **Entry:**
  - text: 0040 0000
  - data: 1000 0000

**Notice:** usr gets relocated due to collision with pi
C source files

Compiler
sum.c → sum.s
math.c → math.s

Assembler
sum.s → sum.o
math.s → math.o
io.s → io.o

Linker
sum.o → sum.exe
math.o → sum.exe
io.o → sum.exe
libc.o → sum.exe
libm.o → sum.exe

Executable program:
exists on disk

Loader:
Executing in Memory process
Loaders

*Loader* reads executable from disk into memory

- Initializes registers, stack, arguments to first function
- Jumps to entry-point

Part of the Operating System (OS)
Shared Libraries

Q: Every program contains parts of same library?!?
A: No, they can use shared libraries
  • Executables all point to single shared library on disk
  • final linking (and relocations) done by the loader

Optimizations:
  • Library compiled at fixed non-zero address
  • Jump table in each program instead of relocations
  • Can even patch jumps on-the-fly
Static and Dynamic Linking

Static linking

• Big executable files (all/most of needed libraries inside)
• Don’t benefit from updates to library
• No load-time linking

Dynamic linking

• Small executable files (just point to shared library)
• Library update benefits all programs that use it
• Load-time cost to do final linking
  – But dll code is probably already in memory
  – And can do the linking incrementally, on-demand
Compiler produces assembly files
• (contain MIPS assembly, pseudo-instructions, directives, etc.)

Assembler produces object files
• (contain MIPS machine code, missing symbols, some layout information, etc.)

Linker joins object files into one executable file
• (contains MIPS machine code, no missing symbols, some layout information)

Loader puts program into memory, jumps to 1st insn, and starts executing a process
• (machine code)