Warm-Up Problem

- Write a MIPS program that determines where it was loaded (i.e., determines the value of its alpha) and returns it in $3.

CS 241 Lecture 22

Linkers

With thanks to Brad Lushman, Troy Vasiga, Kevin Lanctot, and Carmen Bruni

Linkers

- How do we resolve situations where we have labels in different files?
- One option is to cat all such files together, but why should we have to reassemble these files more than once?
- Could we not assemble the files first and then cat?

Linkers

- Almost, but remember, only one piece of code can be at 0x0 at a time. So, these assembled files need to be MERL files and not just MIPS files.
- But concatenating two MERL files does not give a valid MERL file!

But Wait! There's More!

- Still worse, we haven't really resolved the issue of labels in different files!
- We need to modify our assembler: when we encounter a .word where the label is not in the file, we need to use a placeholder (0x0 is fine) and indicate that we cannot run this program until the value of the id is given.

But Wait! There's More!

- For example, below, a.asm on the left and b.asm on the right (recall asm for assembly):

- a.asm

```
lis  $3
.word label
jr $31
```

b.asm

```
label: sw $4, -4($30)
```

- you cannot run a.asm without linking with b.asm.
- We will need to extend our MERL file format so that it can notify us when we need to assemble with multiple files.

Resiliency

- Naively this works, but we make typos. Consider this:
- lis $3
  .word bananana
  banana:
- Did we make a mistake? Did we mean .word banana, or did we mean for bananana to be provided by another MERL file?

Resiliency

- How could we recognize such errors? Without any other changes, our assembler will believe that a label banana exists somewhere and would load this with a placeholder (which might not be what we want!)
- How can we tell our assembler what is an error and what is intentional?
- Hint: You've already been doing this!

New (sort of…) Directive

- .import id is the directive that tells the assembler which symbols to link in.
- This will not assemble to a word of MIPS (so would not be counted in the symbol table).
- Errors occur if the label id is not in the current file and there is no .import id in the file.

New (sort of…) Directive

- We need to add entries in the MERL symbol table
- Previously we used the code 0x1 for relocation entries, but this isn't a relocation entry!
- New format code: 0x11 for External Symbol Reference (ESR).

ESR Entry

What needs to be in an ESR entry?

1. Where the symbol is being used
2. The name of said symbol.

Format:

```
0x11 ;Format code
;location used
;length of name of symbol (n)
;1st ASCII character of name of symbol
;2nd ASCII character of name of symbol
;...
;nth ASCII character of name of symbol
```

The Other Side

- What about if labels are duplicated? Suppose we have a c.asm along with our other two files that has:

```
label: add $1, $0, $0
; more code here
beq $1, $0, label
```

- Here, we want label to not be exported; rather it should be self-contained.

Exporting

- `.export label` will make `label` available for linking with other files. As with `.import`, it does not translate to a word in MIPS. It tells the assembler to make an entry in the MERL symbol table.
- The assembler makes an ESD, or an External Symbol Definition, for these types of words. It follows this format:

```
0x05 ; Format code
; address the symbol represents
; length of name of symbol (n)
; 1st ASCII character of name of symbol
; 2nd ASCII character of name of symbol
; ...
; nth ASCII character of name of symbol
```

Now, our MERL file contains the code, the address that need relocating, as well as the addresses and names of each ESR and ESD. Our linker now has everything it needs to do its job

Linking Algorithm 1/4

```
// Step 1: Check for duplicate export errors
for each ESD in m1.table {
  if there is an ESD with the same name in m2.table {
    ERROR (duplicate exports)
  }
}

// Step 2: Combine the code segments for the linked file
// The code for m2 must appear after the code for m1.
// We treat linked_code as an array of words containing just the
//   concatenation of the code segments
linked_code = concatenate m1.code and m2.code
```

Linking Algorithm 2/4

```
// Step 3: Relocate m2's table entries
reloc_offset = end of m1.code - 12
for each entry in m2.table {
  add reloc_offset to the number stored in the entry
}

// Step 4: Relocate m2.code
// It is essential that this happen after Step 3
for each relocation entry in m2.table {
  index = (address to relocate - 12) / word size
  add relocation offset to linked_code[index]
}
```

Linking Algorithm 3/4

```
// Step 5: Resolve imports for m1
for each ESR in m1.table {
  if there is an ESD in m2.table with a matching name {
    index = (address of ESR - 12) / word size
    overwrite linked_code[index] with the exported label value
    change the ESR to a REL
  }
}

// Step 6: Resolve imports for m2
Repeat Step 5 for imports from m2 and exports for m1
```

Linking Algorithm 4/4

```
// Step 7: Combine the tables for the linked file
linked_table = concatenate modified m1.table and modified m2.table

// Step 8: Compute the header information
endCode = 12 + linked_code size in bytes
endModule = endCode + linked_table size in bytes

// Step 9: Output the MERL file
output merl cookie
output endModule
output endCode
output linked_code
output linked_table
```

Linking Algorithm 1/4

```
// Step 1: Check for duplicate export errors
for each ESD in m1.table {
  if there is an ESD with the same name in m2.table {
    ERROR (duplicate exports)
  }
}

// Step 2: Combine the code segments for the linked fi
// The code for m2 must appear after the code for m1.
// We treat linked_code as an array of words containir
//   concatenation of the code segments
linked_code = concatenate m1.code and m2.code
```

Linking Algorithm 2/4

```
// Step 3: Relocate m2's table entries
reloc_offset = end of m1.code - 12
for each entry in m2.table {
  add reloc_offset to the number stored in the entry
}

// Step 4: Relocate m2.code
// It is essential that this happen after Step 3
for each relocation entry in m2.table {
  index = (address to relocate - 12) / word size
  add relocation offset to linked_code[index]
}
```

Linking Algorithm 3/4

```
// Step 5: Resolve imports for m1
for each ESR in m1.table {
  if there is an ESD in m2.table with a matching name {
    index = (address of ESR - 12) / word size
    overwrite linked_code[index] with the exported label value
    change the ESR to a REL
  }
}

// Step 6: Resolve imports for m2
Repeat Step 5 for imports from m2 and exports for m1
```

Linking Algorithm 4/4

```
// Step 7: Combine the tables for the linked file
linked_table = concatenate modified m1.table and

// Step 8: Compute the header information
endCode = 12 + linked_code size in bytes
endModule = endCode + linked_table size in bytes

// Step 9: Output the MERL file
output merl cookie
output endModule
output endCode
output linked_code
output linked_table
```

```
; m1.merl
0x00: 10000002
0x04: 0000006c
0x08: 00000034
; m1.asm
.import foo
.export bar
sw $31, -4($30)

lis $29
.word foo
jalr $29

lis $3
.word bar
lw $3, 0($3)

lw $31, -4($30)
jr $31

bar: .word 0
```

```
; m1.table
0x34: 00000001;REL
0x38: 00000020
0x3c: 00000011;ESR foo
0x40: 00000014
0x44: 00000003
0x48: 00000066
0x4c: 0000006f
0x50: 0000006f
0x54: 00000005;ESD bar
0x58: 00000030
0x5c: 00000003
0x60: 00000062
0x64: 00000061
0x68: 00000072
```

```
; m2 merl
0x00: 10000002
0x04: 00000078
0x08: 00000040
; m2.asm
.export foo
.import bar
foo:
lis $2
.word -1
lis $28
.word loop
lis $29
.word bar

loop:
lw $3, 0($29)
add $3, $1, $3
sw $3,0($29)
add $1, $1, $2
bne $1, $0, skip
jr $31
skip:
jr $28
```

```
; m2.table
0x40: 00000001;REL
0x44: 00000018
0x48: 00000011;ESR bar
0x4c: 00000020
0x50: 00000003
0x54: 00000062
0x58: 00000061
0x5c: 00000072
0x60: 00000005;ESD foo
0x64: 0000000c
0x68: 00000003
0x6c: 00000066
0x70: 0000006f
0x74: 0000006f
```

Let’s Link!

- We’ll link the two files on the previous slides on the board