Warm Up Problem

Please do your Teaching Evaluations! Go to
`https://evaluate.uwaterloo.ca`
Write the MERL file for the following code:

```
lis $1
.word 0x1000
lis $2
.word A
A: jr $2
beq $0, $1, B
B: jr $31
```

Warm Up Problem Solution

```
beq $0, $0, 2
.word 0x30 ;file length in bytes
.word 0x28 ;code length + header!!
lis $1
.word 0x1000
lis $2
.word A
A: jr $2
beq $0, $1, B
B: jr $31
.word 0x1 ;format code
.word 0x18; where .word A is
```

Note 0x30 is 48 and 0x24 is 40 and 0x18 is 24.

CS 241 Lecture 23

Linking

With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot

Loading

Recall that we were trying to make our files relocatable so that they don't always need to start at memory address 0.

Requires two passes:

- Pass 1: Record the size of the file, start counting addresses at `0x0c` instead of `0x0` and record the location of `.word id` instructions
- Pass 2: Output the header, MIPS machine code and relocation table.

In what follows next, assume that the input is a merl file that is ready to be relocated. The output to the function is that the code is correctly loaded starting from α.

Loader Relocation Algorithm

```
read_line() //skip cookie!
endMod <-- read_line() //End of MERL file
codeLen <- read_line() - 12 //No header in codeLe
alpha <- findFreeRAM(codeLen)
for(int i = 0; i <codeLen; i+= 4)
  MEM[alpha + i] <- read_line()
i <- codeLen + 12 //start of reloc. table
while (i < endMod)
  format <- read_line()
  if (format == 1)
    rel <-- read_line()
    //go forward by alpha and back by header len
    //alpha + rel - 12 since we don't load header
    MEM[alpha + rel - 12] += alpha - 12
  else
    ERROR
  i += 8
```

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

- 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?
- Sure but remember only one piece can be at `0x0` at a time - so these assembled files need to be MERL files and not just MIPS files.
- Worse still, recall that 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 print a place holder, `0x0` in our case, and indicate that we cannot run this program until the value of the `id` is given.
- For example, below, `a.asm` on the left and `b.asm` on the right (recall `asm` for assembly):

`a.asm`

```
lis $3
.word label
```

`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 so that it can notify us when we need to assemble with multiple files.

Error Checking

- Naively this works, but we make typos. Consider this:

```
lis $3
.word banana
bananas:
```

  Did we make a mistake? Maybe we meant `.word bananas`? 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 command that asks for which `id` we should be linking in.
- This will not assemble to a word of MIPS.
- Errors occur if the label `id` is not in the current file *and* there is no `.import id` in the file.
- We need to add entries in the MERL symbol table
- Include 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
```

Still Problems

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.

We include another assembler directive and another MERL entry type to handle this, namely the `.export` directive.

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 addresses 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.

Linker Algorithm 1 of 3

Algorithm 1 Algorithm to link two merl files m_1 and m_2

1: $\alpha = m_1.codeLen - 12$
2: Relocate m_2 by α.
3: Add α to each entry of m_2's symbol table
4: **if** intersection of labels of the exports of m_1 and m_2 are non-empty **then**
5: ERROR
6: **end if**
7: //(See next page...)

Note: the -12 above is because we do not load m_2's header so we need to shift m_2 by -12.

Linker Algorithm 2 of 3

Algorithm 2 Algorithm to link two merl files m_1 and m_2 con't

```
 1: for ⟨addr1, label⟩ in m1's imports do
 2:     if there exists a ⟨addr2, label⟩ in m2's exports then
 3:         m1.code[addr1] = addr2
 4:         Remove ⟨addr1, label⟩ from m1's imports
 5:         Add addr1 to m1's relocates:
 6:     end if
 7: end for
 8: for ⟨addr2, label⟩ in m2's imports do
 9:     if there exists a ⟨addr1, label⟩ in m1's exports then
10:         m2.code[addr2] = addr1
11:         Remove ⟨addr2, label⟩ from m2's imports
12:         Add addr2 to m2's relocates:
13:     end if
14: end for
15: // See last page
```

Linker Algorithm 3 of 3

Algorithm 3 Algorithm to link two merl files m_1 and m_2 con't

1: imports = union of m_1 and m_2's imports
2: exports = union of m_1 and m_2's exports
3: relocates = union of m_1 and m_2's relocates
4: output 0x10000002 (MERL cookie)
5: output total codeLen + total (import, export, relocates) + 12
6: output total codeLen + 12
7: output m_1 code
8: output m_2 code
9: output Imports, exports, relocates