CS 241, Lecture 23: Linkers

Note everything in this lecture was already covered in my lecture 22 notes. This is mostly rehashing/additions.

1 Warm Up Problem

Given the MIPS code:

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

what is the appropriate MERL file?

Answer:

```
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
```

2 Loading

- Recall the point of loading was to put things in different addresses, instead of just starting everything at 0x0.
- This requires two passes: one to record the size of hte file, where we start counting addresses at 0x0c instead of 0x0, and record the location of .word id instructions, and the second to output the header, MIPS machine code, and relocation table.
- To sum up the algorithm (as I wrote this before):
  - Start by reading in a MERL file. Read one line to skip the cookie.
  - Store the end length of the MERL file into `endMod`, store the code length - 12 into `codeLen`, where we removed the header.

- Store the free RAM into `alpha`, based on `codeLen`. We assume this can be done with some function; this is not taught.
- Loop through codeLen, and store `read_line()` into `MEM[alpha + i]`.
- Now, from `codeLen + 12` to `endMod`, then store `read_line` into a variable we call `format`.
- If `format == 1` then the next line is something we have to add. `read_line()` again go forward by `alpha` and back by the header len.
- This gives us `MEM[alpha + rel - 12] += alpha - 12`
- If `format != 1` then clearly something is wrong, and ERROR out.
- Then, increment our counter for this while loop by 8 as we just read through 2 lines.

3 Linkers

- We put a placeholder, 0x0, to indicate that we cannot run a program that relies on a linked file without that id value.

3.1 External Symbol Reference(ESR)

- What `.import` creates
- An ESR entry contains two things: where a symbol is being used, and the name of the symbol
- The format is:

```
0x11 ;Format code
;location used
;length of the name of the symbol
;each ASCII char of the symbol
```

3.2 External Symbol Definition(ESD)

- The ESD follows a similar format:

```
0x05 ;Format code
;address the symbol represents
;length of the name of the symbol
;each ASCII char of the symbol
```

4 Course Summary — How do we write a compiler?

So... what have we learned? How **do** we write a compiler now?

- First, we need to scan our code.
  - Tokenizing is just regular languages.
    - DFAs, NFAs, ϵ-NFAs
  - We use a simplified maximal munch algorithm to parse.
- Then, we need to parse.
  - This needs a grammar!
    - CFGs
    - This also points back to regular languages, as every CFG is a RegLang.
  - Remember LR1, LL1, LR0, and SLR1. First and Follow.
  - Top-down and bottom-up.
- Type/error check.
  - See CS 245. Inference rules... are great.
- Then, code generation.
  - Stack management - register \$29 and \$30
  - Caller or callee when saving
  - Label collison
- Assemble.
- Linking and loading.