# CS 241 – Week 4 Tutorial

Writing an Assembler Pt. II

Spring 2015

## Summary

- Outputting bytes
- Outputting instructions
- MERL
- Relocating

## Problems

1. Write pseudocode for a function called `output_word` that takes a 32-bit integer as input and outputs each of its four bytes to standard output.

   You can assume a function called `output_byte` is available. This function takes an integer as input and first checks if the integer is small enough that it can be represented in 8 bits. If so, it outputs the corresponding byte; otherwise it produces an error.

   How would you use this above function (in conjunction with the symbol table) to assemble the `.word after` directive in the above program?

2. Write a function called `assemble_add` that assembles instructions of the form `add $d, $s, $t`. It should take three integer parameters (the numbers $d$, $s$ and $t$) and return a single integer representing the binary encoding of the add instruction.

   Can we generalize the `assemble_add` function for other instructions? How?

## MERL

MERL, which stands for **M**IPS **E**xecutable **R**elocatable **L**inkable (file format), describes a MIPS executable program that can be relocated and run at an arbitrary address $\alpha$ and can be linked with any number of other MERL programs. In order to faciliate these features, MERL files contain a table full of "sticky notes" that describe certain properties of the MERL file.

A MERL file will always begin with a 3 word header. The first value is the *cookie*, which has the value 0x10000002. Note that this value actually corresponds to the instruction `beq $0, $0, 2`, which gives the desirable property that any MERL file is executable, as it will cause the rest of the header to be skipped when executed. The second word is a pointer *file length*, that tells how long the entire file is in bytes. The

third word is a pointer *code length* that points to the word after the end of the code. Essentially, the code length allows us to easily locate the sticky notes.

There are 3 kinds of sticky notes in a MERL file:

- REL - a relocation entry. A REL entry has the form:
  1. `0x01`
  2. `loc` - a word which is a pointer to the location within the program that must be relocated
- ESD - an external symbol definition. An ESD entry specifies a label which must be visible to other programs, analogous to the `.export` directive in assembly.
  1. `0x05`
  2. `val` - a word which is the value of the label `sym` in the original assembly program.
  3. `len` - a word which is the length of the label `sym`
  4. `len` words, specifying the name of the label `sym` in ASCII.
- ESR - an external symbol reference. An ESR entry specifies a label which the program requires someone else to provide, analogous to the `.import` directive in assembly.
  1. `0x011`
  2. `loc` - a word which is the location of a *use* of the label `sym` in the original assembly program.
  3. `len` - a word which is the length of the label `sym`
  4. `len` words, specifying the name of the label `sym` in ASCII.

# Problems

3. Transform the given assembly language program into an equivalent MERL file (you may leave the file in assembly for readability). Relocate the resulting MERL program to execute at address $\alpha =$ `0x8c`

```
  lis $6
  .word 0x18

  sw $31, -4($30)
  lis $31
  .word 4
  sub $30, $30, $31
  lis $3
  .word proc
  lis $11
  .word 1
loop:
  beq $2, $0, end
  jalr $3
  sub $2, $2, $11
  beq $0, $0, loop

end:
  add $3, $1, $0
```

```
  lis $31
  .word 4
  add $30, $30, $31
  lw $31, -4($30)
  jr $31

proc:
  add $1, $1, $6
  jr $31
```