SPIM16 EECS Project 1 Specification

ASSEMBLY APPLICATION FORMAT

Assembly File (.asm)

- Program Lines
  - Zero or One Label, plus
  - Zero or One Instruction, plus
  - Zero or One Comment
- Labels – Symbol Definitions
  - Starts with a letter, plus
  - Zero or more symbol characters, plus
  - A single colon ':'
- Symbol Characters
  - Letter
  - Digit
  - Underscore
- Instructions
  - Starts with a name, plus
  - One or more operands separated by whitespace
- Operands (depends on instruction format)
  - A Register, OR
  - An Immediate Value, OR
  - A Label
- Comments
  - Begin with pound sign '#'
  - Continues to end of line
- Load/Store
  - Source/destination register first operand
  - Immediate value second operand
  - Base address register third operand surrounded by parenthesis

NOTE: All assembler input is to be considered valid and well formed. Only one instruction exists per line.

Usage

```
assembler filename
```

FILE OUTPUT

- Symbols File (.sym), Object File (.o), RAM File (.raw)
- The .raw file is a copy of the object file in upper case hexadecimal text, one instruction per line
- Symbol addresses are hexadecimal and 4 characters wide
- Symbol file lines have one definition followed by the usage list separated by tabs per line
- The initial line for the .raw file must be (for use with simulation software): `v2.0 raw`
- Initial header files are given. Do not change the header files, instead create and add to the source files.
- Code project in ANSI-C. See my ANSI-C tutorial for help

BINARY INSTRUCTION FORMATS

Register Format (R)

```
# Assembly Code
operation rd, rs, rt

# Machine Code
oooo ssss tttt dddd
```

Register-Immediate Format (RI)

```
# Assembly Code
operation rt, rs, immediate
operation rt, immediate(rs)

# Machine Code
oooo ssss tttt iiii
```

Immediate Format (I)

```
# Assembly Code
operation rs, immediate
operation rs, label

# Machine Code
oooo ssss iiii iiii
```

Jump Format (J)

```
# Assembly Code
operation immediate
operation label

# Machine Code
oooo iiii iiii iiii
```

GENERAL APPROACH

The Symbol Table

- Read through the .asm file
- Label address is address of next instruction starting with 0
- Only instruction lines increment the address
- Address increments by instruction width in bytes (2)
- Scan for symbol label definitions and label uses (operands)
- Create a table entry for each label definition
- Create a symbol use entry for each label use and add them to the appropriate symbol entry
- Create a new symbol entry with an `address` of `0xFFFF` and `defined` value of `'n'` for a use that has not yet been defined
- Change the `address` to the correct address and set `defined` to `'y'` when you find a label definition and a symbol entry already exists

The Assembler

- Read through the .asm file a second time
- This time translate each assembly instructions into upper case hexadecimal text for the .raw file and binary machine code for the .o file
- Again, scan for any label uses, but not definitions, and lookup the appropriate address value in the symbol table