# Object Oriented Analysis

Joshua Archibald, Steven Lei, Christian Miranda

## Introduction

The purpose of this document is to identify objects in order to build upon the theory of operation, as well as to make future additions/maintenance of this code easier. Each object will have the following attributes:

- Name

  : The name of the object

- Description

  : A brief description of this object and its purpose in this project.

- Base Class

  : Aka: "superclass" - the class from which this class inherits properties (attributes, methods, etc).

- Data Members

  : Any element that is part of the current class's data, not including methods. Can be of primitive or defined types.

- Constructors

  : The method that creates an instance of a class, ie, an object. Includes information on what is instantiated within the class.

- Destructors

  : The method that destroys an instance of a class.

- Methods

  : Operations that can be preformed on this object.

- Operators

  : Primitive operators (eg: `+`, `-`, `/`, `*`, etc) that can be used on this object - what operators are overloaded on this object.

- Type Conversion

  : Classes that this class can be converted to.

- Error Handling

  : Errors produced during method calls/instantiation.

- Helper Functions

: Functions that are used in this class that cannot be called - they are internal to this class.

Additionally, at the end there will be a brief description of the main loop.

## asm_line

### Description

---

This object implements an assembly line. This object contains the information parsed from a `string` line of assembly from an assembly or include `file`.

### Base Class

---

None.

### Data Members

---

- `origin_file_name` - Name of the file the line of assembly originally belonged to as a `string`.
- `line_num` - The line number of the assembly line in its file as an `int`.
- `text` - The assembly line as a `string`.
- `label` - The label in the assembly line as a `string`.
- `op_name` - The operation name in the assembly line as a `string`.
- `operand` - The operand in the assembly line as a `string`.

### Constructors

---

- `construct_asm_line`:
- **Description:** Parses a line of assembly and updates all data.
- **Arguments:** Line of assembly as a `string` and an `isa` object.

### Destructors

---

- `destruct_asm_line`

### Methods

---

### Accessors

---

- `origin_file`
  - **Return Value:** Name of the assembly line's file of origin as a `string`.
- `line_num`
  - **Return Value:** The assembly line's line number in its file.
- `text`
  - **Return Value:** The assembly line as a `string`.
- `label`
  - **Return Value:** The assembly line's label as a `string`.

### Modifiers

---

None.

### General

---

- `assemble`
  - **Description:** Provides the assembled program data corresponding to this assembly line.
  - **Arguments:** `isa` object and `symbol_table` `string` to `int` `map` object.
  - **Return Value:** The program data as an `int`.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

`construct_asm_line` will return `NULL` if a line of assembly is unable to be parsed.

### Helper Functions

---

None.

## isa (Instruction Set Architecture)

### Description

---

This object implements an Instruction Set Architecture (ISA) object. This object contains the relationship between operation names, operands, and assembled program data. This object has the ability to translate `asm_line` object information to program data.

### Base Class

---

None.

### Data Members

---

- `code_map` - Maps operation names as `string` objects to `code_macro` objects. Type `map`.
- `style` - Ordered `list` object that holds the specified order of assembly line elements and their delimiters.

### Constructors

---

- `construct_isa`
  - **Description:** Parses the ISA file and updates all data.
  - **Arguments:** Name of ISA file as a `string`.

### Destructors

---

- `destruct_isa`

### Methods

---

### Accessors

---

None.

### Modifiers

---

None.

### General

---

- `translate`
  - **Description:** Provides the program data corresponding to an operation name and an operand.
  - **Arguments:** An operation name and an operand as `string` objects.
  - **Return Value:** The program data as an `int`.
- `parse_asm`
  - **Description:** Provides an `asm_line` from a string of assembly.
  - **Arguments:** A line of assembly as a `string`.
  - **Return Value:** The line of assembly as an `asm_line`.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

`translate` will return `NULL` if the provided operation name and operand do not have a corresponding piece of program data. `parse_asm` will return `NULL` if the provided assembly line cannot be parsed.

### Helper Functions

---

None.

## code_macro

### Description

---

This object implements a code macro. This object contains the relationship between an operation code, an operand, and their assembled program data.

### Base Class

---

None.

### Data Members

---

- `op_code` - `int` that represents a unique `op name`.
- `operand_template` - `string` template for how the operand should look.
- `prog_data_template` - `string` template for how the program data should look.

### Constructors

---

- `construct_code_macro`
  - **Description:** Constructs the code macro object and updates all data.
  - **Arguments:** Operand template and program data template as `string` objects and the operation code as an `int`.

### Destructors

---

- `destruct_code_macro`

### Methods

---

### Accessors

---

None.

### Modifiers

---

None.

### General

---

- `prog_data`
  - **Description:** Provides the program data corresponding to an operand.
  - **Arguments:** An operand as a `string`.
  - **Return Value:** The program data as an `string`.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

`prog_data` will return `NULL` if the provided operand does not follow the `operand_template`.

### Helper Functions

---

None.

## assembler

### Description

---

This object implements the assembler. This object can assemble assembly and include `file` objects into machine code and listing `file` objects.

### Base Class

---

None.

### Data Members

---

- `pc` - The program counter as an `int`.
- `data_used` - The amount of data memory space used by the program being assembled as an `int`
- `symbol_table` - Maps `labels`, including variable names, as `strings` to their address in program memory as an `int` objects. Type `map`.
- `ref_table` - Maps `labels`, including variable names, as `strings` to the lines in assembly they are defined and referenced as a `list` of `int` objects. Type `map`.
- `isa` - The `isa` object created from an ISA file.
- `asm_files` - Ordered `list` object of assembly and include file names as `string` objects.
- `asm_prog` - Ordered `list` of `asm_line` objects.

### Constructors

---

- `construct_assembler`
  - **Description:** Constructs the assembler object and the `isa` object contained in it. `asm_files` data is also updated.

- **Arguments:** Ordered `list` of file names as `string` objects to be assembled and the name of ISA file as a `string`.

**Destructors**

---

- `destruct_assembler`

**Methods**

---

**Accessors**

---

None.

**Modifiers**

---

None.

**General**

---

- `first_pass`
  - **Description:** Performs the first pass on the assembly and include `file` objects. This updates the `symbol_table`, `ref_table`, `asm_prog`, and the `data_used` data.
  - **Arguments:** None.
  - **Return Value:** None.
- `second_pass`
  - **Description:** Performs the second pass on the `asm_line` objects. This assembles all the code and writes to machine code and listing `file` objects. this updates the `pc`, `asm_prog`, and the `data_used` data.
  - **Arguments:** None.
  - **Return Value:** None.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

Displays the appropriate error message to the user if any method is unsuccessful. In this case the assembly process continues but a machine code `file` object will not be generated.

### Helper Functions

---

None.

## expression

### Description

---

This object implements an expression. This object can evaluate `string` expressions that contain `string` representations of the supported `operator` objects and `int` objects.

### Base Class

---

None.

### Data Members

---

- `expression` - The expression as a `string`.

### Constructors

---

- `construct_expression`
  - **Description:** Constructs the expression object and updates all data.
  - **Arguments:** The expression as a `string`.

### Destructors

---

- `destruct_expression`

### Methods

---

### Accessors

---

None.

### Modifiers

---

None.

### General

---

- `evaluate`
  - **Description:** Provides the `int` equivalent of the expression.
  - **Arguments:** None.
  - **Return Value:** None.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

`evaluate` returns `NULL` if the string expression contains symbols that are not either `int` or supported `operator` objects or the `(` and `)` `char` objects.

### Helper Functions

---

None.

## operator

### Description

---

This object implements an operator. This object can evaluate `int` objects in expressions. These objects respect a priority among other `operator` objects, and has a `string` representation. This object can perform an operation and two `int` objects.

### Base Class

---

None.

### Data Members

---

- `priority` - The operator's priority with respect to other `operator` objects represented as an `int`.
- `symbol` - The operator's `string` representation.
- `evaluate_func_ptr` - A function pointer to the operation between the `int` objects.

### Constructors

---

- `construct_operator`
  - **Description:** Constructs the operator object and updates all data.
  - **Arguments:** The operator's priority and number of operands as `int` objects, the operand symbol as a `string`, and an evaluation function pointer.

### Destructors

---

- `destruct_operator`

### Methods

- `priority`
  - **Return Value:** The operators priority as an `int`.
- `symbol`
  - **Return Value:** The operand's `string` representation.

### Modifiers

---

None.

### General

---

- `evaluate`
  - **Description:** Provides the `int` result of the specified operation on two `int` objects.
  - **Arguments:** Two `int` objects.
  - **Return Value:** `int` result of operation.

### Operators

---

None.

### Type Conversions

---

None.

### Error Handling

---

None.

**Helper Functions**

---

None.

## Main Loop

The main loop is not a class, but rather constructs instances of the classes we have described in this document and operates on them to complete the ultimate goal of producing runnable machine code. These are steps it will take:

1) Use user input to extract the name of the main assembly `file` and the name of the `isa file` as `string` objects.
2) Create an `assembler` object using the extracted objects which creates the `isa` object, creating `code_macro` object's.
3) Run the `first_pass` method of the `assembler` object which uses the `isa` object to update its data members.
4) If errors occur in the first pass end the program. Note that the appropriate error messages are displayed within internal function calls.
5) If no errors occur in the first pass, run the `second_pass` method which uses the `isa` object and internal data to produce machine code and a listing file.
6) End the program. Note that If errors occurred in the previous step no object file or list would be produced.