SIC/XE Assembler

A complete two-pass assembler for the SIC/XE (Simplified Instructional Computer - Extended) architecture.

Team Members

- **Ilyas** - Pass 1, Data Structures
- **Nadja** - Pass 2, Output Generation

Language & Requirements

Language: Python 3.11 or higher

Dependencies:

- Python standard library only (no external packages required)

Optional (for testing):

- pytest 7.4+ (for running unit tests)

Project Structure

```
sicxe-assembler/
├── assembler.py          # Main assembler program
├── input_processor.py    # Source file parsing
├── pass1.py              # Pass 1 implementation
├── pass2.py              # Pass 2 implementation
├── data_structures.py    # OPTAB, SYMTAB, LITTAB
├── output_generator.py   # Object file generation
├── README.md             # This file
├── tests/                # Test files
│   ├── test1.asm
│   ├── test2.asm
│   └── ...
└── output/               # Generated object files
```

Development Environment

IDE: Visual Studio Code 1.83+

- Python extension recommended

- Git integration enabled

Version Control: Git/GitHub

Testing Framework: pytest (optional)

Installation

1. **Clone the repository:**

```bash
git clone https://github.com/yourusername/sicxe-assembler.git
cd sicxe-assembler
```

2. **Verify Python version:**

```bash
python --version  # Should be 3.11+
```

3. **No additional installation required** (uses Python standard library only)

How to Compile

No compilation needed - Python is interpreted. All `.py` files are source code.

How to Run

Basic Usage

```bash
python assembler.py <input_file.asm> [output_file.obj]
```

Examples

1. **Basic assembly (auto-generates output filename):**

```bash
```

```
python assembler.py test1.asm
# Generates: test1.obj
```

2. **Specify output filename:**

```
bash

python assembler.py test1.asm my_program.obj
```

3. **With verbose output:**

```
bash

python assembler.py test1.asm --verbose
# Shows symbol table, object code for each line
```

Command-Line Options

```
usage: assembler.py [-h] [-o OUTPUT] [-v] input

SIC/XE Two-Pass Assembler

positional arguments:
  input              Input assembly source file (.asm)

optional arguments:
  -h, --help         Show this help message and exit
  -o OUTPUT          Output object file (default: input_name.obj)
  -v, --verbose      Show detailed assembly process
  --symtab           Display symbol table
  --no-output        Run assembler without generating object file (checking only)
```

Input File Format

Assembly source files should be in standard SIC/XE format:

```
assembly
```

```
COPY   START  1000
FIRST  LDA    ALPHA
       ADD    BETA
       STA    GAMMA
ALPHA  RESW   1
BETA   RESW   1
GAMMA  RESW   1
       END    FIRST
```

Format rules:

- Labels: Start in column 1, max 6 characters
- Mnemonic: Separated by whitespace from label
- Operand: Separated by whitespace from mnemonic
- Comments: Start with '.' or ';'

Output Files

The assembler generates:

1. **Object file (.obj)** - Contains:
    - H record (Header)
    - T records (Text/object code)
    - M records (Modification for relocation)
    - E record (End)
2. **Listing file (.lst)** - Optional, shows:
    - Line numbers
    - Addresses
    - Object code
    - Source code

Example Output

```
H^COPY  ^001000^001077
T^001000^1D^141033^482039^28203F^...
M^000004^05
```

```
M^00000D^05
E^001000
```

Features Implemented

✅ Pass 1:

- Symbol table construction
- Location counter management
- All directives (START, END, RESW, RESB, WORD, BYTE)
- Literal processing
- Program length calculation
- Error detection (duplicate symbols, invalid labels)

✅ Pass 2:

- Object code generation for all 59 SIC/XE instructions
- All addressing modes:
  - Immediate (#)
  - Indirect (@)
  - Indexed (,X)
  - PC-relative
  - Base-relative
- Format 1, 2, 3, 4 instructions
- nixbpe flag generation
- Displacement calculation

✅ Output Generation:

- Header record
- Text records (60-byte limit)
- Modification records
- End record

Testing

Run all tests:

```bash
pytest tests/
```

Run specific test:

```bash
python assembler.py tests/test1.asm
```

Test cases provided:

1. `test1.asm` - Simple program with basic instructions
2. `test2.asm` - All directives
3. `test3.asm` - Addressing modes
4. `test4.asm` - PC-relative addressing
5. `test5.asm` - Base-relative addressing
6. `test6.asm` - Format 4 and literals

Troubleshooting

Problem: "Command not found: python"

- Try `python3` instead of `python`
- Ensure Python is in your PATH

Problem: "No module named 'xxx'"

- All required modules should be in the same directory
- Check that all `.py` files are present

Problem: Assembly errors

- Check input file format (labels, spacing)
- Verify instruction mnemonics are valid

- Check for undefined symbols

Problem: Object file format errors

- Ensure END directive is present
- Check that program doesn't exceed address space

Error Messages

The assembler provides detailed error messages:

```
ERROR (Line 5): Duplicate symbol 'ALPHA'
ERROR (Line 12): Undefined symbol 'BETA'
ERROR (Line 8): Invalid mnemonic 'ADDD'
ERROR (Line 15): Displacement out of range for PC-relative
```

Supported Instructions

All 59 standard SIC/XE instructions including:

- Arithmetic: ADD, SUB, MUL, DIV, ADDF, SUBF, MULF, DIVF
- Logic: AND, OR, COMP, COMPR
- Load/Store: LDA, LDB, LDCH, STA, STB, STCH, etc.
- Jump: J, JEQ, JGT, JLT, JSUB, RSUB
- And many more...

Limitations

- No macro processing (MACRO/MEND not supported)
- No CSECT support (single control section only)
- Limited expression evaluation in operands
- No optimization (manual Format 3/4 selection required)

Project Statistics

- **Lines of Code:** ~2,500
- **Modules:** 5 main classes

- **Test Cases:** 6 programs
- **Instructions Supported:** 59

Contact

For questions or issues:

- Ilyas: [email]
- Nadja: [email]

License

Academic project for Computer Structures course.

Version History

- **v1.0** (Dec 15, 2025) - Initial release
    - Two-pass assembler complete
    - All addressing modes supported
    - Object file generation working

Last Updated: December 15, 2025