ASSEMBLER REPORT

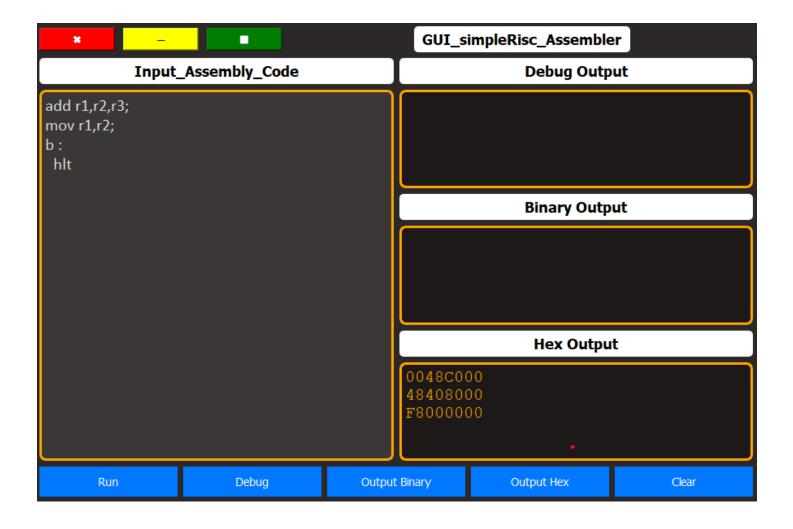
git page

GUI ASSEMBLER FOR SIMPLERISC

Flow of the assembler

- Reading the input file line to line
- Parsing the text and tokenizing the instruction into opcode and opernads ,labels seperately
- Operand, Opcode and immediate value validation .
- Implemeting the relative adressing by reading the input file only once .
- Binary file generation
- Hexadecimal file generation
- GUI using the python libraries

GUI LOOK:



Rules to be followed

1.Register naming:

- Registers must be named as **rX**, where X is a valid number between **0-15**.
- Valid register names: **r3, r14**
- Invalid register names: r03, r28

2.commenting format:

- you can use ';' for commenting lines
- your comment for a single instruction should only occupy one line .

EXAMPLE:

Valid: add r1,r2,r3; adding two numbers and storing

```
add r1,r2,r3; adding two numbers and .....
invalid:
```

3.Labelling format:

- whatever you write just before the ':' is treated as the label.
- There is no restriction for label lenghth.

EXAMPLE:

```
1. New: addr1,r2,r3;

2.new:
add r1,r2,r3

3.new:; new label (this also works but no use)
```

invalid: new add r1,r2,r3;

4.Input Instruction format:

- you should use only small letters for opcode names, registers
- there should be space between opcode and operands
- the operands should be seperated from each other by ','
- the end of the instruction need not be a '; 'you can just leave like that with empty space

EXAMPLE:

```
1. add r1,r2,r3

Valid: 2.add r1,r2,r3;

3.add r1, r2, r3
```

addr2,r3,r4;

invalid: ADDr2,r3,r4

add R3,R4,R4;

5.Immediate value formatting:

- It should be written in the standard format starting with 'Ox'
- The hexadecimal number should contains the alphabets in capitals only .
- The hexadecimal value should be valid and it should contain charcaters '0-9' or 'A-F'

EXAMPLE:

add r1,r2,0xA000

Valid:

add r1,r2,0x5

add r2,r3,oxA00

invalid: add r2,r3,0Xabcd

add r1, r2, A000

6. Formating for ld and st instructions:

- You should use the box format for memory '[]'.
- The offset format should be same as the immediate adressing .

EXAMPLE:

Valid ld r1,0x5[r2]

7. Immediate value position in operands:

- The position of the immediate value should always be at the last considering the no of operands .
- The sense of the source and destination is always preserved

EXAMPLE:

add r1,r2,0x5

Valid: mov r1,0x5

mov r1,r2;(for r1 as destination, r2 as a source)

add r2,0x5,r3

invalid:

mov 0x5,r1

Rest all the rules are same as that we follow in simple risc

Functionalities implemented:

- Making the assembler custiomizable for future addition of instructions by using maps in c++.
- Better standardisation of the instruction using structure which contains opcode,operand_1,operand_2,operand_3,adress as a single datatype stored ina dynamic array.
- memory allocation of program is dynamic (no usage of fixed sizes for arrays)
- No limit for label name characters, number of line of the code
- Mapped special instructions like which contains modifiers **addu** with valid encoding format.
- Validate register addressing modes to ensure correct syntax and operation.
- Handling the branch instruction encoding for positive and negative offset by 2'somplement .
- stored the encoded format into binary file with 32 bits format, .hex file with 8 digit number

Constraints:

- The instructions should be all in small letters.
- the immediate value always should start with Ox.
- No implementation address and opcode showing in the output .(A000:000000..)

Problems tackled:

- Perfoming the labelling with **name,address** fromat into map by reading the input file only once
- Seperation of the different operand instructions effeciently with maps
- Implemting a Debug mechanism adressing all mechanism with line number mentioned
- classifying **ld** , **st** insctructons as three opernads even though they are parsed as the two operand type while reading
- effective **seperation** using for robust differentiation of label, comment, instruction by using the **getline()** function with condition on **delimeters (:,;)** and classifying the type of instruction into types (only label, label+ instruction, instruction only, instruction + comment, label + instruction+ comment, label + comment)
- Effective **dfferentiation** between the **opcode,operands** while parsing by placing ',' between them for easy tokenisation.
- Storing address for each instruction based on line number for effective offset encoding .

For using the gui_asssembler refer the git page and follow the guidlines provided.

AUTHOR:

P. Siva Dhanush Reddy