Instrution of MIPS Assembler

Command Line Interface (console.py)

```
only support assemble via command line
```

directly use

```
pass the file(.bin) througn argv
```

```
> python console.py <your_binary_file_path>
```

make use of sublime

```
add a new build system.
open sublime -> 'Tools' -> 'Build System' -> 'New Build System'
add following .json text
```

```
{
    "cmd": ["python","[your_console.py_path]", "$file"],
    "file_regex": "^(...*?):([0-9]*):([0-9]*): (...*?)$",
    "working_dir": "$file_path",
    "selector": "source.asm"
}
```

Graphic Interface (graphics.py)

Packup

```
the provided source code must run within a python environment. if you are
willing to you can also freeze it to a ".exe" executable file

simply run:
```

```
> python setup.py build
```

Shortcuts

- open file <ctrl-o>
- close file <ctrl-c>
- save file <ctrl-s>
- assemble <ctrl-b>
- disassemble <ctrl-d>
- switch edit mode <ctrl-e>

Menu

- File
    i. open file: close the current file and open a new one
    ii. close file: close the current file
    iii. save: save the change you made to current file, if no opened file call "save as"
    iv. save as: save as a new file
- Edit
    i. enter_edit: start editting, since you can't change the opened file by default
    ii. exit_edit: exit editting mode
- Run
    i. assemble: if a ".asm" file is opened, it will be assembled

ii. disassemble: if a ".bin" file is opened, it will be disassembled

Button

- you can choose either ".bin" or ".coe" as the output of assemble

MIPS Syntax

segment declare

```
this assmble is mainly work for FPGA using, so you have to specify where to put your code or data.
```

- .text: <xxxx> declare a code segment start at xxxxh
- .data: <xxxx> declare a data segment start at xxxxh

supported instruction

- Rtype: **add addu sub subu and or xor nor jr jalr sllv srlv srav slt sltu sll srl sra**
- Itype: **addi addiu ori andi xori lui slti sltiu lw lb lbu lh lhu sw sh sb beq bne bgtz blez**
- Jtype: **j jal**
- Pesudo: **la move li**

data declare

```
little endian
```

- 8 bits
  i. .byte "hello"
    - **in memory : 68 65 6c 6c \ 6f 00 00 00**
  ii. .byte "hello", 0x12, 0x34
    - **in memory : 68 65 6c 6c \ 6f 12 34 00**
  iii. .byte 0xff,0x12
    - **in memory : ff 12 00 00**
- 16 bits
  i. .word "hello", 0x1234
    - **in memory : 68 65 6c 6c \ 6f 00 34 12**
  ii. .word 0x1234
    - **in memory : 34 12 00 00**
  iii. .word 0x1234, 0x5678
    - **in memory : 34 12 78 56**
- 32 bits
  i. .dword "hello"
    - **in memory : 68 65 6c 6c \ 6f 00 00 00**
  ii. .dword 0x12345678
    - **in memory : 78 56 34 12**
  iii. .dword 6789abcd
    - **in memory : 67 89 ab cd**