

# Y86 Assembly IDE 使用说明

周芯怡 17307130354

张作柏 17300240035

2019 年 1 月 11 日

## 1 简介

Y86 Assembly IDE提供书写Y86汇编代码与动态调试的基本功能：

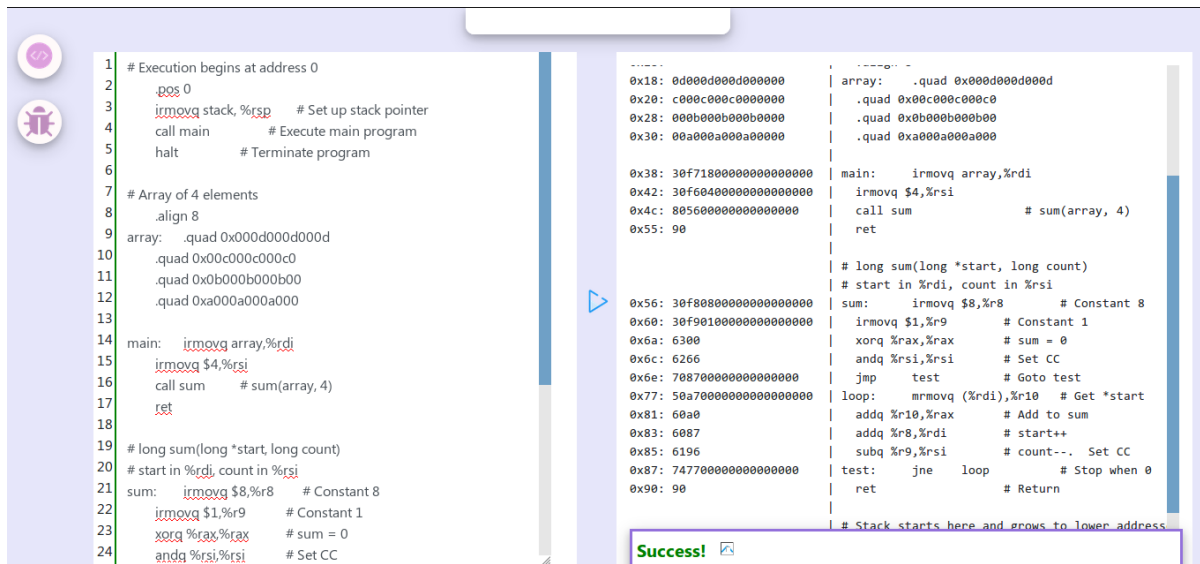
- 支持将汇编代码和指令码以.yo文件的形式输出
- 支持运行中跳转至任意语句
- 支持设置断点
- 支持设置、监视特定值
- 强大的help功能

Y86汇编IDE开发基于Django框架，运行需要安装Python 2.7与Django 1.10。

使用时，需在Y86/Y86\_Assembly\_IDE目录下打开命令行，输入指令 `python manage.py runserver` 。运行成功后，打开浏览器，输入地址 `http://localhost:8000/IDE`即可进入IDE界面。


## 2 界面说明

### 2.1 代码界面



```
1 # Execution begins at address 0
2 .pos 0
3 irmovq stack, %rsp # Set up stack pointer
4 call main # Execute main program
5 halt # Terminate program
6
7 # Array of 4 elements
8 .align 8
9 array: .quad 0x00d000d000d
10 .quad 0x00c000c000c0
11 .quad 0xb000b000b00
12 .quad 0xa000a000a000
13
14 main: irmovq array, %rdi
15 irmovq $4, %rsi
16 call sum # sum(array, 4)
17 ret
18
19 # long sum(long *start, long count)
20 # start in %rdi, count in %rsi
21 sum: irmovq $8, %r8 # Constant 8
22 irmovq $1, %r9 # Constant 1
23 xorq %rax, %rax # sum = 0
24 andq %rsi, %rsi # Set CC
```

```
-----
0x18: 0d000d000d000000 array: .quad 0x00d000d000d
0x20: c000c000c0000000 .quad 0x00c000c000c0
0x28: 00b000b000b00000 .quad 0xb000b000b00
0x30: 0a000a000a000000 .quad 0xa000a000a000
-----
0x38: 30f71800000000000000 main: irmovq array, %rdi
0x42: 30f60400000000000000 irmovq $4, %rsi
0x4c: 80560000000000000000 call sum # sum(array, 4)
0x55: 90 ret
-----
# long sum(long *start, long count)
# start in %rdi, count in %rsi
0x56: 30f80800000000000000 sum: irmovq $8, %r8 # Constant 8
0x60: 30f90100000000000000 irmovq $1, %r9 # Constant 1
0x6a: 6300 xorq %rax, %rax # sum = 0
0x6c: 6266 andq %rsi, %rsi # Set CC
0x6e: 70870000000000000000 jmp test # Goto test
0x77: 50a70000000000000000 loop: mrmovq (%rdi), %r10 # Get *start
0x81: 60a0 addq %r10, %rax # Add to sum
0x83: 6087 addq %r8, %rdi # start++
0x85: 6196 subq %r9, %rsi # count-- Set CC
0x87: 74770000000000000000 test: jne loop # Stop when 0
0x90: 90 ret # Return
-----
# Stack starts here and grows to lower address
```

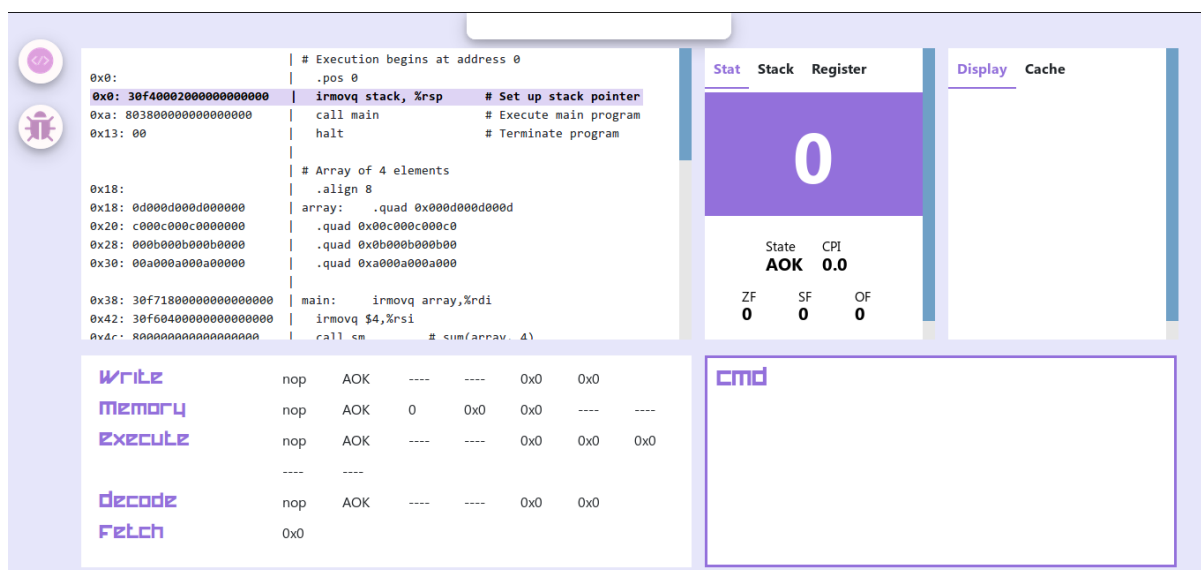
Success! 

如上图，左侧为代码框，用于书写汇编代码，右侧为编译成功后的.yo代码。

为方便阅读，书写代码时会自动添加行号。未进行编译时，左侧边框为黄色，点击编译按钮后，边框变为绿色，表示已完成编译。

右下方**Error Info**一栏显示编译信息：编译成功时，显示Success；编译失败时，显示Failed，并弹出错误列表。错误列表将显示错误的行号，并返回详细错误。双击该错误，可使.yo代码中相应行高亮显示。

## 2.2 Debug界面



点击页面左侧图标可切换至Debug界面。

页面上方为指令输入框，用于输入指令。

左上方为.yo代码框。汇编代码编译成功后，.yo代码会自动同步到此处。代码运行过程中，会高亮当前行，清晰显示代码运行过程。

左下方为流水线寄存器的信息显示，将鼠标放在数字上，可显示对应的含义。指令执行过程中，五个Stage图标会闪烁，亮起表示正在执行，熄灭表示执行完毕，以此来可视化并行过程。

右下方为CMD窗口，返回指令执行信息，可显示输入指令及返回的运行结果。

正中为运行状态窗口，可分别显示运行的状态(Cycle数，Status，CPI，条件码)、栈中元素、寄存器的值。

右上方为显示窗口，可观察设置为Display的变量与Cache当前的存储信息。

## 3 指令说明

### 3.1 #Step指令集

指令	说明
step	格式: step [N] 不带参数时, 执行下一个指令 带参数时, 参数[N]表示前进N步(除非程序中止) 可缩写为s
next	格式: next [N] 不带参数时, 执行下一个指令, 不会进入子函数 带参数时, 参数[N]表示前进N步(除非程序中止) 可缩写为n
continue	格式: continue 继续执行当前程序, 直到遇到断点或中止信号 可缩写为c
finish	格式: finish 执行当前程序, 直到退出当前函数

### 3.2 #Jump指令集

指令	说明
jump	格式: jump <location> 令PC跳至位置<location>处 <location>可以是指令地址, 可以是合法的label
return	格式: return 直接跳出当前函数。
call	格式: call <label> 直接调用label函数。

### 3.3 #Breakpoints指令集

指令	说明
break	格式: break <i>&lt;location&gt;</i> 在位置 <i>&lt;location&gt;</i> 处设置断点。
info breakpoints	格式: info breakpoints 输出所有用户设置的未被删除的断点信息, 包括断点编号、断点状态、断点位置。
enable	格式: enable <i>&lt;NUM&gt;</i> 使编号为 <i>&lt;NUM&gt;</i> 的断点重新发挥作用, 相对指令disable。 若不带任何参数, 则恢复所有断点。
disable	格式: disable <i>&lt;NUM&gt;</i> 使编号为 <i>&lt;NUM&gt;</i> 的断点失去作用, 即遇到该断点时不在停止, 相对指令enable。 若不带任何参数, 则使所有断点失去作用。
delete	格式: delete <i>&lt;NUM&gt;</i> 删除编号为 <i>&lt;NUM&gt;</i> 的断点, 连其编号一同移除, 即之后不再有编号为 <i>&lt;NUM&gt;</i> 的断点。 若不带任何参数, 则删除所有断点。

### 3.4 #Display指令集

指令	说明
display	格式: display <i>&lt;EXP&gt;</i> 在Display框中实时显示EXP变量的值。 几种EXP的格式: REG <i>&lt;REGNAME&gt;</i> 例: REG %rax 显示系统寄存器 <i>&lt;REGNAME&gt;</i> 的值, 注意 <i>&lt;REGNAME&gt;</i> 必须以'%'开头。 MEM <i>&lt;ADDR&gt;</i> 例: MEM 0x200 显示内存地址 <i>&lt;ADDR&gt;</i> 中的值, 注意 <i>&lt;ADDR&gt;</i> 必须是十六进制表示。 STACK <i>&lt;NUMID&gt;</i> 例: STACK 0 显示系统栈项第 <i>&lt;NUMID&gt;</i> 个元素的值, 下标从0开始, 注意 <i>&lt;NUMID&gt;</i> 必须是十进制表示。
undisplay	格式: undisplay <i>&lt;EXP&gt;</i> 在Display框中取消显示EXP变量的值。 几种EXP的格式: REG <i>&lt;REGNAME&gt;</i> 例: REG %rax 显示系统寄存器 <i>&lt;REGNAME&gt;</i> 的值, 注意 <i>&lt;REGNAME&gt;</i> 必须以'%'开头。 MEM <i>&lt;ADDR&gt;</i> 例: MEM 0x200 显示内存地址 <i>&lt;ADDR&gt;</i> 中的值, 注意 <i>&lt;ADDR&gt;</i> 必须是十六进制表示。 STACK <i>&lt;NUMID&gt;</i> 例: STACK 0 显示系统栈项第 <i>&lt;NUMID&gt;</i> 个元素的值, 下标从0开始, 注意 <i>&lt;NUMID&gt;</i> 必须是十进制表示。

### 3.5 #I/O指令集

指令	说明
write	格式: write $\langle FILENAME \rangle$ 将指令码与汇编代码以.yo文件的格式写入文件 $\langle FILENAME \rangle$ 中。
list	格式: list 将指令码与汇编代码以.yo文件的格式显示在屏幕上。
load	格式: load $\langle FILENAME \rangle$ 清除之前所有的代码和状态。 从 $\langle FILENAME \rangle$ 中加载指令码和汇编码, $\langle FILENAME \rangle$ 必须是.yo文件或.ya文件。 当加载.yo类型文件时, 可直接从中提取指令码。 当加载.ya类型文件时, 调用YAS生成指令码, 并记录所有label。
read	格式: read [Enter] $\langle AssemblyCode \rangle$ [Enter] 动态输入汇编代码, 附加在当前代码集合之后, 可以调用之前的label, 可以新建label。

### 3.6 #Others指令集

指令	说明
set	格式: set $\langle EXP \rangle$ $\langle value \rangle$ 几种EXP的格式: REG $\langle REGNAME \rangle$ 例: REG %rax 显示系统寄存器 $\langle REGNAME \rangle$ 的值, 注意 $\langle REGNAME \rangle$ 必须以'%'开头。 MEM $\langle ADDR \rangle$ 例: MEM 0x200 显示内存地址 $\langle ADDR \rangle$ 中的值, 注意 $\langle ADDR \rangle$ 必须是十六进制表示。 STACK $\langle NUMID \rangle$ 例: STACK 0 显示系统栈顶第 $\langle NUMID \rangle$ 个元素的值, 下标从0开始, 注意 $\langle NUMID \rangle$ 必须是十进制表示。 将表达式 $\langle EXP \rangle$ 指定的值设为 $\langle value \rangle$ 。
clear	格式: clear 清除当前所有代码和状态。
help	格式: help $\langle CMD \rangle$ 显示 $\langle CMD \rangle$ 指令的使用说明。 当不加参数时, 显示所有指令组。 当 $\langle CMD \rangle$ 为指令组时, 显示该指令组内的所有指令简介。 当 $\langle CMD \rangle$ 为某个指令时, 显示该指令的使用方法。
quit	格式: quit 退出Y86 Assembly IDE。

## 4 注意事项

- 载入.yo文件时，不支持任何label操作。因为载入时直接读取指令码，所以无法对应label。
- 直接输入回车[Enter]时，会执行上一条指令。
- 载入文件时，会自动清除之前的所有代码与状态。
- 支持#和/\*\*/两种注释格式，暂不支持分行注释。