Translation between Assembly and Binary

2 min

Assembly was written as the first language above

Preview: Docs Loading link description

binary

to make it easier for humans to write functional programs. As such, there is almost a line for line equivalency between the two codes.

When we first translated the simple square()

Preview: Docs Loading link description

method

to Assembly in Exercise 3, our two lines of Python code expanded into 16 lines of Assembly. The

Preview: Docs Loading link description

code editor

has the Python and the translated Assembly in the window to the right.

When the Assembler translates the Assembly code into

Preview: Docs Loading link description

machine code

, each line will create a 32-bit MIPS instruction in accordance with the standards of the ISA.

The Assembly ADD function and the binary ADD function are structured the same way in the documentation in order to give developers better control over their programs.

Instructions

Check out the official MIPS32 Documentation and take a look at some of the Assembly statements. Come up with the binary output you would expect to see.

When you are finished, hit Run and see the binary output of the square() function and see how you did.

asm_to_binary.py

```
from translate import translate


# Assembly Code

#   square:

#     addiu $sp,$sp,8

#     sw $fp,4($sp)

#     move $fp,$sp

#     sw $4,8($fp)
#     lw $3,8($fp)
#     lw $2,8($fp)
#     nop
#     mult $3,$2
#     mflo $2
#     move $sp,$fp
#     lw $fp,4($sp)
#     addiu $sp,$sp,8
#     j $31
#     nop

translate()
```

translate.py

```
def translate():
   print("MIPS32 Binary Output")
   print()
   print("00100111101111010000000000001000")
   print("10101111101111100000000000000100")
   print("01000100000000001110111110000110")
   print("10101111110001000000000000001000")
   print("10001111110000110000000000001000")
   print("10001111110000100000000000001000")
   print("00000000000000000000000000000000")
   print("00000000011000100000000000011000")
   print("00000000000000000001000000010010")
   print("01000100000000001111011101000110")
   print("10001111101111100000000000000100")
   print("00100111101111010000000000001000")
   print("00001000000000000000000000011111")
   print("00000000000000000000000000000000")
```