Assembly Code Format

8 min

The generated Assembly from the last exercise follows much of the same semantics as the

Preview: Docs Machine code (also known as machine language or native code) is a low level programming language in the form of hexadecimal or binary instructions that execute computer programs on the computers CPU.

machine code

we learned in the ISA lesson. This is because Assembly language and

Preview: Docs Loading link description

binary

code have almost a direct translation between their outputs.

Assembly was created as a mnemonic language to make machine code easier to read and write, one instruction translating to one instruction. In fact, most ISAs will have both the binary code and Assembly language breakdown on the same page when talking about specific instructions.

Just like in binary, Assembly begins with an opcode. Luckily for us, the opcodes are much more readable than a bunch of 0's and 1's. Let's take a look at the multiply function:

Assembly:

MULT $3, $2

to Clipboard

Binary:

00000000111001100000000000011000

to Clipboard

It's easy to see how much easier it is to write a program in Assembly than to write it in binary. In most Assembly instructions, what follows the opcodes are the memory locations to be operated on. These memory locations are referred to as operands. Generally, these are direct register addresses but can also be memory references to values stored in other types of memory such as the

Preview: Docs Loading link description

cache

or

Preview: Docs Loading link description

RAM

.

In MIPS, direct register addresses begin with the $ symbol, so in our MULT example, we are multiplying the value stored in register $2 by the value in register $3. You can learn more about MULT and other MIPS functions in the official MIPS32 Documentation.

Instructions

1. Checkpoint 1 Passed

1.

Create a new variable, answer_1, and set it equal to the term used for the first part of an Assembly instruction.

Your answer should be all lowercase.

Hint

The first part of an Assembly instruction is the same as the first part of a binary instruction.

2. Checkpoint 2 Passed

2.

Create another variable, answer_2, and set it equal to the symbol used in MIPs to address registers.

Hint

We use this symbol when addressing a register directly by its number.

3. Checkpoint 3 Passed

3.

Create a final variable, answer_3, and set it equal to the part of the Assembly instruction that gets operated on.

Hint

This is the part of the instruction where specific locations are designated to be operated on, they have a specific name.

code_format.py

```
# Write your answers here:

answer_1 = "opcode"

answer_2 = "$"

answer_3 = "operand"
```