R-Type Instructions

3 min

R-Type instructions are the most common in MIPS and give us a good way of understanding how an ISA defines the process that a

Preview: Docs Central Processing Unit (CPU) is used to describe the electrical circuitry in a computer responsible for executing instruction sets derived from programs that have been converted into machine code. CPUs sit at the heart of the system of a computer and are connected to the motherboard. They also have caches which act as memory units capable of temporary storage and quick retrieval of files by the computer.

CPU

goes through when receiving data.

All R-type instructions have an op of '000000' which signifies to the

Preview: Docs A processor is the electric circuitry found in computer hardware which is responsible for executing instruction sets derived from programs that have been converted into machine code.

processor

to look at the func bits to determine which process to execute.

The three references to registers, (rs, rt, rd) directly call them by number. There are 32 registers in a MIPS system and the five bits will produce 32 numbers (0 as 00000 to 31 as 11111). The first two (rs & rt) are the operands and the last (`rd) is where to store the result.

The shift amount is used to shift the number by the amount in the shift bits, for our purposes this will always be 00000, meaning no shift. The last six bits are the function to be performed on the operands.

Let's take a look at this example instruction:

000000 00101 10010 00110 00000 100000
op rs rt rd shamt func

Copy to Clipboard

Now, let's break the instruction down:

- op -> 000000 signifies an R-Type instruction
- rs -> 00101 gets contents in Register 5
- rt -> 10010 gets contents in Register 18
- rd -> 00110 sets the result in Register 6
- shamt -> 00000 means there is no shift
- func -> 100000 function signature for adding

Our processor will add the contents of Register 5 (16) and Register 18 (103) and store that result (119) into Register 6. In the example, these numbers are shown as integers, but in the registers they will be stored in

Preview: Docs Binary is a number system of 1s and 0s which serves as a textual representation method for giving instructions to a computers CPU. In computing and telecommunications, binary codes are used for various methods of encoding data such as turning the character strings of source code, into bit strings or instruction sets for the CPU to execute.

binary

. Notice that the first register is always 0 and is a protected register, this is common in most ISAs.

Instructions

The two tables in the code editor represent the data stored in the registers before and after the ADD instruction is passed to the CPU. You will notice that the original numbers being added still remain in their location in the register. They will stay there until they are overwritten or power is turned off to the CPU. Registers are volatile memory.

script.py

```
# Registers (32 total on CPU, 0-indexed, register 0 is constant 0)

# Before addition

# +========+========+========+========+========+========+========+========+

# | 0: 0   | 1:     | 2:     | 3:     | 4:     | 5: 16  | 6:     | 7:     |

# +--------+--------+--------+--------+--------+--------+--------+--------+

# | 8:     | 9:     |10:     |11:     |12:     |13:     |14:     |15:     |

# +--------+--------+--------+--------+--------+--------+--------+--------+

# |16:     |17:     |18: 103 |19:     |20:     |21:     |22:     |23:     |

# +--------+--------+--------+--------+--------+--------+--------+--------+

# |24:     |25:     |26:     |27:     |28:     |29:     |30:     |31:     |

# +========+========+========+========+========+========+========+========+


# Instruction sent from software (machine code) to hardware (CPU):

00000000101100100011000000100000


# Registers (32 total on CPU, 0-indexed, register 0 is constant 0)

# After addition

# +========+========+========+========+========+========+========+========+

# | 0: 0   | 1:     | 2:     | 3:     | 4:     | 5: 16  | 6: 119 | 7:     |

# +--------+--------+--------+--------+--------+--------+--------+--------+

# | 8:     | 9:     |10:     |11:     |12:     |13:     |14:     |15:     |

# +--------+--------+--------+--------+--------+--------+--------+--------+
```

|16: |17: |18: 103 |19: |20: |21: |22: |23: |

+--------+--------+--------+--------+--------+--------+--------+--------+

|24: |25: |26: |27: |28: |29: |30: |31: |

+========+========+========+========+========+========+========+========+