Leonardo Villalobos

Prof. Jose Coria

CSS 3DE 9406

7 December 2021

Architecture

CLobos ISA is built with an architecture that has an 8K memory with word length of 13 bits, where 5 bits are used for the opcode, 6 bits are used for the address, and 2 bits are used for the registers. Below is a visual representation of how an instruction would be broken up.

|  |  |  |
| --- | --- | --- |
| 0b00000 | 0b000000 | 0b00 |
| Opcode | Address | Register |

Certain instructions require more than one argument, so the address field is partitioned into 3 separate parts, each representing a space for a register. The address is partitioned as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0b00000 | 0b00 | 0b00 | 0b00 | 0b00 |
| Opcode | Register | Register | Register | Register |

Registers

I am using 4 registers; therefore 2 bits are required to represent all registers. Registers are defined as:

* \_\_REG\_1: 00
* \_\_REG\_2: 01
* \_\_REG\_3: 10
* \_\_REG\_4: 11

Opcodes

I am using an enumeration to enumerate my opcodes using an unsigned integer. Each opcode and its usage are listed below.

Stop (0b00000)

Required by every program, this instruction terminates the program.

*Assembly*:

0b 00000 000000 00

There are no arguments, so the trailing bits after the opcode should be padded with 0s.

Sample usage:

Clear all the registers and then terminate the program (Does not display anything nor change much since registers are already 0 be default).

1. TidyUp
2. Stop

In <dest.> (0b00001)

Grab input from the keyboard and store it in the destination register specified.

*Assembly*:

0b 00001 XX0000 00

This instruction takes in one argument. The Xs in the assembly above represent the binary value of the destination register.

Sample usage:

Request user input and store the value (an unsigned integer) into \_\_REG\_1 (register 1). Value stored is displayed onto the output stream and the program is terminated.

1. In \_\_REG\_1
2. Out \_\_REG\_1
3. Stop

Out <src.> (0b00010)

Display the value stored in the source register to the output stream.

*Assembly*:

0b 00010 XX0000 00

This instruction takes in one argument. The Xs in the assembly above represent the binary value of the source register.

Sample usage:

Similar to the sample from the previous instruction, this sample program requests user input and stores the value (an unsigned integer) into \_\_REG\_2 (register 2). The value stored is displayed onto the output stream and then the program is terminated.

1. In \_\_REG\_2
2. Out \_\_REG\_2
3. Stop

Incr <src.> <amt.> (0b00011)

Increment the value stored at the source register by the literal amount specified.

*Assembly*:

0b 00011 XXXXXX YY

This instruction takes in two arguments. The Xs in the assembly above represent the binary value of the literal amount, while the Ys represent the source register.

Sample usage:

Request user input and store a value into \_\_REG\_3 (register 3). The value is incremented by 7 and displayed to the output stream.

1. In \_\_REG\_3
2. Incr \_\_REG\_3 7
3. Out \_\_REG\_3
4. Stop

Add <lhs> <rhs> <dest.> (0b00100)

Add the value stored in the left-hand side (lhs) register to the value stored in the right-hand side (rhs) register and store the sum in the destination register.

*Assembly*:

0b 00100 XXYYZZ 00

This instruction takes in three arguments. The Xs in the assembly above represent the left-hand side register, the Ys represent the right-hand side register, and the Zs represent the destination register. The last two bits are padded with 0s size they are not used.

Sample usage:

Request user input twice, storing the first value into \_\_REG\_1 and the second value into \_\_REG\_2. The values are summed up and stored into \_\_REG\_3. Finally, the value at register 3 (the sum) is displayed to the screen.

1. In \_\_REG\_1
2. In \_\_REG\_2
3. Add \_\_REG\_1 \_\_REG\_2 \_\_REG\_3
4. Out \_\_REG\_3
5. Stop

Sub <lhs> <rhs> <dest.> (0b00101)

Subtract the value stored in the left-hand side (lhs) register to the value stored in the right-hand side (rhs) register and store the difference in the destination register.

*Assembly*:

0b 00101 XXYYZZ 00

This instruction takes in three arguments. The Xs in the assembly above represent the left-hand side register, the Ys represent the right-hand side register, and the Zs represent the destination register. The last two bits are padded with 0s size they are not used.

Sample usage:

Request user input twice, storing the first value into \_\_REG\_1 and the second value into \_\_REG\_2. The difference is calculated and stored into \_\_REG\_3. Finally, the value at register 3 is displayed to the screen.

1. In \_\_REG\_1
2. In \_\_REG\_2
3. Sub \_\_REG\_1 \_\_REG\_2 \_\_REG\_3
4. Out \_\_REG\_3
5. Stop

Mul <lhs> <rhs> <dest.> (0b00110)

Multiply the value stored in the left-hand side (lhs) register to the value stored in the right-hand side (rhs) register and store the product in the destination register.

*Assembly*:

0b 00110 XXYYZZ 00

This instruction takes in three arguments. The Xs in the assembly above represent the left-hand side register, the Ys represent the right-hand side register, and the Zs represent the destination register. The last two bits are padded with 0s size they are not used.

Sample usage:

Request user input twice, storing the first value into \_\_REG\_1 and the second value into \_\_REG\_2. The product is calculated and stored into \_\_REG\_3. Finally, the value at register 3 is displayed to the screen.

1. In \_\_REG\_1
2. In \_\_REG\_2
3. Mul \_\_REG\_1 \_\_REG\_2 \_\_REG\_3
4. Out \_\_REG\_3
5. Stop

List <amt.> <dest.> (0b00111)

Define an array with the size specified by <amt.>. This argument can be a reference to a register or a literal unsigned integer type. Prefix a register with @ to get the value at the register. The destination argument specifies at which register to store the array.

Example: Using a reference to a register to define the size of an array

In the example below, the size of the array depends on the amount specified by the user. The array is defined and stored into \_\_REG\_2.

1. In \_\_REG\_1
2. List @\_\_REG\_1 \_\_REG\_2

Example: Using a literal to define the size of an array

In the example below, the size of the array is defined by the programmer using a literal. The array is defined to have a size of 34 and it is stored into \_\_REG\_4.

1. List 35 \_\_REG\_4

*Assembly*:

0b 00111 XXXXXX YY

This instruction takes in two arguments. The Xs in the assembly above represent the size of the list (an unsigned integer) and the Ys represent the destination register.

Sample usage:

Request user input and store the value at \_\_REG\_1. The value at \_\_REG\_1 determines the size of the array being defined at \_\_REG\_3. This array is then initialized, and the program terminates.

1. In \_\_REG\_1
2. List @\_\_REG\_1 \_\_REG\_3
3. ListInit \_\_REG\_3
4. Stop

ListInit <src.> (0b01000)

Initializes the array specified by the <src.> by requesting user input for each element in the array.

*Assembly*:

0b 01000 XX0000 00

This instruction takes in one argument. The Xs in the assembly above represent the register at which an array is stored. Each bit after is padded with a 0.

Sample usage:

Request user input and store the value at \_\_REG\_1. The value at \_\_REG\_1 determines the size of the array being defined at \_\_REG\_3. This array is then initialized, and the program terminates.

1. In \_\_REG\_1
2. List @\_\_REG\_1 \_\_REG\_3
3. ListInit \_\_REG\_3
4. Stop

ListSum <src.> <dest.> (0b01001)

Calculates the sum for each element in the source register and stores the result in the destination register.

*Assembly*:

0b 01001 XXYY00 00

This instruction takes in two arguments. The Xs in the assembly above represent the register at which an array is stored, and the Ys represent the destination register. Each bit after is padded with a 0.

Sample usage:

Request user input and store the value into \_\_REG\_1. This value is used to specify the size of an array, being stored at \_\_REG\_3. The array at \_\_REG\_3 is initialized, the sum is calculated, and finally the program terminates.

1. In \_\_REG\_1
2. List @\_\_REG\_1 \_\_REG\_3
3. ListInit \_\_REG\_3
4. ListSum \_\_REG\_3 \_\_REG\_2
5. Stop

TidyUp (0b01010)

Clears the data from all four registers (resets each value to 0).

*Assembly*:

0b 01010 000000 00

This instruction takes in no arguments. All bits after the opcode are padded with 0s.

Sample usage:

Input two values and store one into \_\_REG\_1 and another into \_\_REG\_2. Next, calculate the sum of \_\_REG\_1 and \_\_REG\_2, storing the result into \_\_REG\_3. Now multiply \_\_REG\_3 and \_\_REG\_1, storing their result into \_\_REG\_4. Display the value at \_\_REG\_4. Now, the registers are cleared. Display \_\_REG\_4 again. The program is then terminated.

1. In \_\_REG\_1
2. In \_\_REG\_2
3. Add \_\_REG\_1 \_\_REG\_2 \_\_REG\_3
4. Mul \_\_REG\_3 \_\_REG\_1 \_\_REG\_4
5. Out \_\_REG\_4
6. TidyUp
7. Out \_\_REG\_4
8. Stop