Leg System Documentation

Amaziner01

2021-05-26

Contents

Chapter 1

Assembly

Leg Chip is an imaginary machine made for the sake of learning, and to hopefully help others to understand how computers work using graphics and a custom oversimplified x86 Assembly subset. It was made using TypeScript, HTML5 and CSS.

1.1 Getting Started

When getting started of something, it is important to get a "Hello World" working first. In this machine it would be defining a string with the **ds** directive (define string) and using the **print** instruction to display it on the Teletype as shown in the code snippet bellow:

```
mov r0, string
print
halt

string: ds "Hello World!\n"
```

Chapter 2

Virtual Machine Specifications

The main objective of this project was that it would hopefully be ismple enough so peoplle won't need much knowledge about computers to fully understand it. This machine consists of 17 registers, 2 flags, and 2 stacks.

You have 16 general purpose registers and the program counter. Only those general purpose registers, identified as r0, r1, ...r15, are accesible from code.

There are only two flags on the system, the Negative Flag and the Zero Flag. Those are used to do comparisons in code using the **cmp** instruction.

You also have two stacks, the general stack and the call stack. I decided to put function calls in a different stack so there would be no conflicts when pushing and popping from the general stack. There limit of elements you can push on the stack depends on your computer (it is not strictly limited).

Also, the memory of the virtual machine depends on the capabilities of your computer, so you have some freedom when it comes to creating programs for it.

Chapter 3

Assembly

Mnemonics:

- RR - Register (r0 - r15)
- NNNN - Literal number value (0 - 65535)

3.1 Processor

3.1.1 Halt

Finishes execution of the program.

```
Example: halt
```

3.1.2 Nop

Does nothing, just takes 1 cycle.

```
Example: nop
```

3.2 Register

3.2.1 Mov

Sets register value as the specified number.

```
Example: mov RR, NNNN
```

3.3 Stack

3.3.1 Push

Puts value of specified register in the top of the stack, so it can be retrieved later.

```
Example: push RR
```

3.3.2 Pop

Retrieves value from the top of the stack and writes it in the specified register, removing it afterwards from the stack. If the stack is empty and the Pop instruction is called, it is going to return the value 0.

```
Example: pop RR
```

3.3.3 Print

Uses the value in register zero as a pointer to a string and print every character in the Teletype until it finds a zero in memory.

```
Example: print
```

3.3.4 PrintR

Prints the value of the register in the Teletype.

```
Example: pop RR
```

3.4 Arithmetic

3.4.1 Add

Add the values of the two specified registers and puts it into register zero (r0).

```
Example: add RR, RR
```

3.4.2 Sub

Subtracts the values of the two specified registers and puts it into registerzero (r0).

```
Example: sub RR, RR
```

3.4.3 Mul

Multiplies the values of the two specified registers and puts it into register zero (r0).

```
Example: mul RR, RR
```

3.4.4 Div

Divides the values of the two specified registers and puts it into register zero (r0).

```
Example: div RR, RR
```

3.4.5 Mod

Gets the module of the two specified registers and puts it into register zero (r0).

```
Example: mod RR, RR
```

3.5 Logic

3.5.1 Jmp

Sets the program counterto the number or the label name (if the label exists in the program) specified.

```
Example: jmp NNNN
```

3.5.2 Cmp

Compare the values of the two specified registers. In essence it is a substraction that is not saved in the register zero. If the result is negative sets the Negative Flag to on, and if it is zero, it sets the Zero Flag to on.

```
Example: cmp RR, RR
```

3.6 Subroutines

Chapter 4

Example Code

4.0.1 Hello World!

```
mov r0, string
print
halt

string: ds "Hello world!\n"
```

4.0.2 Print all characters

```
mov r0, byte

loop:
inc r1
str r1, string
print
jmp loop

byte: ds "a"
```

4.0.3 Power of two numbers

```
jmp start

data1: ds " to the power of "
data2: ds " is "

start:

; You can change the following values
mov r1, 8 ; Base
mov r2, 3 ; Exponent

push r1
pop r0
printr
mov r0, data1
print
push r2
pop r0
printr
mov r0, data2
print

call power
printr
mov r0, endl
print
halt

power:
push r1
pop r0
mov r3, 1

powerloop:
mul r0, r1
dec r2
cmp r2, r3
jg powerloop
ret

endl: ds "\n"
```

Chapter 5

Useful Tricks

Chapter 6

Simulator