LabCPU

A Simple Stack-based LabView CPU

Mathias Hörtnagl

mathias.hoertnagl@gmail.com

Contents

1 Introduction

LabCPU is a simple stack machine which executes assembly code without compilation immediatly. It supports 26 instructions and offers a special commands to set and read the pixels of a virtual display. The source code layout is very limited in order to keep the file processing simple.

2 Recognized Instructions

LabCPU supports a pretty limited number of case-insensitive instructions. Some of them are accompanied by a single argument (`<num>`, `<addr, ><label>`, `<0|1>`). To ease the definition of the instructions, let <x> be the top, <y> be the second element on the stack and let <z> be the new top element after executing a command.

2.1 Stack Operations

`PSH <num>`

Push a number `<num>` onto the stack. The number can be either represented as a decimal or hexadecimal one. The prefix '0x' indicates a hexadecimal number.

```
PSH 2011
PSH 0x7DB
```

In each case the top stack element is 2011.

`POP`

Pop top element off the stack.

`CPY`

Copy top stack element and push it onto the stack.

`SWP`

Swap top two stack elements.

`CTM <addr>`

Copy top stack element to <addr> in RAM.

`CFM`

Copy element at <addr> in RAM to top of stack.

2.2 Logic Operations

`AND`

$<z> = <x> \wedge <y>$

`OOR`

$<z> = <x> \vee <y>$

`NOT`

$<z> = \neg <x>$

LabCPU does not support logical shift operations.

2.3 Arithmetic

`ADD`

$<z> = <x> + <y>$

`SUB`

$<z> = <x> - <y>$

`MUL`

$<z> = <x> \times <y>$

`DIV`

$<z> = <x> \div <y>$

2.4 Jumps and Branches

We can set a label by starting a line with a '.' immediatly followed by a unique label name. The '.' must be the first character starting a label line. Everything after that '.' will be taken as a label name. If we jump to a label then we can't place anything after a label but whitespaces.

```
...
.label-name
...
;Don't comment the JMP line!
JMP label-name
```

We can jump back to the next line followed by the label line. The jump and brach operations take as an argument the name of the label – without the '.' – we want to jump to.

`JMP <label>`

Unconditionally jump to position <label> + 1.

`BEQ <label>`

<x> = <y>, jump to <label> + 1.

`BLE <label>`

<x> $\leq$ <y>, jump to <label> + 1.

`BGE <label>`
: $<x> \geq <y>$, jump to $<\text{label}> + 1$.

`BLT <label>`
: $<x> < <y>$, jump to $<\text{label}> + 1$.

`BGT <label>`
: $<x> > <y>$, jump to $<\text{label}> + 1$.

`BNE <label>`
: $<x> \neq <y>$, jump to $<\text{label}> + 1$.

`CAL <label>`
: Unconditionally jump to position $<\text{label}> + 1$ and put the current address $+ 1$ onto the stack.

`RET`
: Unconditionally jump to the position stored as top stack element.

```
...
CAL sub ;Save return address and jump
;Next instruction to be executed after return
...
.sub
;Subprogramm
RET ;Return to address stored on top of stack
...
```

2.5 Miscellaneous

`WDP <0|1>`
: Set display pixel at row $<x>$ and column $<y>$ either $<1>$ or $<0>$.

`RDP`
: Read display pixel at row $<x>$ and column $<y>$.

`SLP <num>`
: Sleep for $<\text{num}>$ milliseconds. Every other operation sleeps for 1 ms.

`HLT`
: Halts the CPU and must be the last command in a source file, because the symbol search routine stopps with the first occurence of a `HLT`. We recommend a jump to this last symbol after the program finished execution.

```
...
;Program done, jump to HLT
```

```
JMP hlt
...
.hlt
HLT
```

3 Source File Guidelines

Comments do not need a special start character. However we recommend a ';' to avoid coplications with comments starting with an operator mnemonic, since these would be executed. Don't comment Jump and Branch operations since their label would not only consist of the original label name but the comment as well. The same holds for label lines.

```
;Example Programm Description
HLT
```

4 Examples