BCI Assembly Language

Contents

Commands, Small Arguments and Big Arguments

A command in BCI Assembly is a word starting with an alphabetic character (`a..zA..Z`) following by a sequence of alphanumeric characters (`a..zA..Z0..9`). This word will be converted to a 10bit opcode.

Embedded in the 16bits of a word there is also a 6bit small argument. If a command has no small argument these bits will be zeroed. In the assembly the command will be only one word, for example:

```
cli
```

If the command has a small argument, the 6 bit will be filled with the small argument. In the assembly the small argument is separated by one whitespace, for example:

```
inc r0
```

Any other arguments are stored in further words and have thus a width of 16bits. They are separated by commas (`,`) from both the first and any other arguments. It is recommended to only add one more argument.

Example for one big argument:

```
ldi r0, 0xdead
```

It might be useful to have more arguments for other applications, like double precision floating points. Example (not implemented):

```
lddfi r0, 0xdead, 0xbeef
; load double precision floating point
; to r0 and r1
```

Built-In Commands

`ldi <sa>, <ba>`

Load the value `<ba>` into register `<sa>`.

`ld <sa>, <ba>`

Load the value of the memory cell at `<ba>` into register `<sa>`.

`st <sa>, <ba>`

Store the value of register `<sa>` into the memory cell at `<ba>`.

`inc <sa>`

Increment the value of register `<sa>`.

`dec <sa>`

Decrement the value of register `<sa>`.

`add|sub|mul|div <sa>, <ba>`

`<sa> = <sa> +|-|*|/ <ba>` where `<sa>` and `<ba>` are registers. Write the overflow into the status register.

`gt|ge|lt|le|eq <sa>`

Check if the value of register `<sa>` is `>|>=|<|<=|==` to `0`. Set the status register to `1` if it evaluates true, else to `0`.

`not`

If the status register is `0` set it to `1`, else set it to `0`.

`jmp <sa>`

Set the program counter to the value of register `<sa>`.

`call <sa>`

Push the current program counter on the stack and set the program counter to the value of register `<sa>`.

`ret`

Pop the previously pushed program counter from the stack.

`stop`

Write `1` into the shutdown register. This will cause the interpreter to halt.

`cl`

Write `0` into the status register.

`cjmp <sa>`

If there not a `0` in the status register, `jmp <sa>`, else continue execution.

`ccall <sa>`

Like `cjmp <sa>` but with `call` instead.

Comments

Comments start with a `;` at the beginning of the line and end at the end of the line.

Marks

Marks represent a special location of the assembly code. The assembler keeps track of those marks and they can be used as immediate input.

A mark is defined by a single word, starting with an alphabetic character (`a..zA...Z`) containing alphanumeric characters and underscores (`a..zA..Z0..9_`) followed by a colon (`:`) and a newline character.

Example:

```
ldi r0, this_is_a_mark
ldi r1, 0xfefe
ldi r2, 0xefef

this_is_a_mark:
add r2, r1
; this will result in an infinite loop.
jmp r0
```

Direct Input

The core instruction set contains the `ldi` command that can be used to load data into a register directly.

The first (big) argument of this command is always a 16bit word. The assembler can automatically generate the correct value if the argument is provided in the following ways:

Marks

The assembler inserts the absolute offset of the Mark.

A decimal value

The assembler inserts the value (i.e. `ldi r0, 12`).

A hexadecimal value

If the argument starts with `0x` the assembler will interpret the argument as hexadecimal.

A binary value

If the argument starts with `0b` the assembler will interpret the value as binary.

A character

If the argument is either a single character surrounded by two `'` characters or any unicode escape sequence surrounded by `'` characters the assembler will insert the integer representation.