Programming in Infernal

v0.666

2018–06-29

Contents

1 Overview

Infernal is a simple assembly-like scripting language. Its design is based on the accumulator-oriented instruction sets of many microcontrollers.

To this end, most of its instructions operate on a single special-purpose variable (the so-called "working register", `WREG`), and all instructions require either one or zero operands. This operand is always a register, a label, or an unsigned 8-bit value.

The language is designed for controlling peripheral devices, and does not currently support signed numbers.

1.1 Definitions

register A location for storing data. Infernal provides 16 registers for general use and 5 special registers.

instruction A binary code which instructs a badge to do something useful.

instruction queue The ordered list of instructions which forms an Infernal program. The last instruction in the instruction queue must be the `END` instruction.

command An instruction that will be executed immediately rather than being added to the instruction queue.

mnemonic A handy way to refer to an instruction or register instead of using its binary code.

label A numbered marker placed within the instruction queue using the `LBL` instruction. Branching instructions require labels as operands.

1.2 Manual Conventions

- The mnemonic for a register refers to the address of that register. For instance, `WREG` refers to the address 0x10.
- The mnemonic for a register is prepended by an asterisk when referring to the contents of that register. For instance, *`WREG` refers to whatever value is currenty held by the working register.
- The mnemonic for each register or instruction is written in uppercase `typewriter` font. If an instruction requires an operand, the type of the operand is specified by a lowercase letter following the mnemonic:
  - *`l`* is a label (from `0x00` to `0x0F`)
  - *`r`* is a register (general- or special-purpose)
  - *`v`* is an unsigned 8-bit value (from `0x00` to `0xFF`)

- Complete Infernal scripts are shown in `typewriter` font in whole paragraphs. The mnemonics in these scripts can be converted to their binary representations and entered into the badge.

1.3 Syntax and Procedure

The general form of a statement in Infernal is [operator] followed by [operand]. When writing Infernal code, it is helpful to follow the following format:

```
[mnemonic] [operand] // [comment]
```

where each statement is written as a separate line in a plaintext file. A complete Infernal script may thus be fed into an Infernal translation program, which will output the equivalent sequence of binary codes for use with a badge.

2 Registers

Infernal provides 16 general-purpose registers, `R0-R15`. Additionally, 5 special-purpose registers are accessible. Each register contains an unsigned 8-bit value.

Many instructions operate implicitly on one or more of the special-purpose registers. If an instruction requires a register as an operand, any of the following registers may be used.

Mnemonic	Code	Name	Purpose
`WREG`	`0xFC`	**Working Register**	**Contains the result of most arithmetic and logic operations**
`COMP`	`0xFD`	**Comparison Register**	**Contains the value against which `*WREG` is compared during branching instructions**
`WHEEL`	`0xFE`	**Wheel Register**	**Contains user input value**
`DISP`	`0xFF`	**Display Register**	**Value stored here will be immediately displayed to user**
`LINK`	`0xFB`	**Link Register**	**Contains most recent subroutine return location**
`R0-R15`	`0x00-0x0F`	**General Purpose Registers**	

3 Instructions

3.1 Commands

Commands are executed immediately rather than being added to the instruction queue.

Mnemonic	Code	Function
`RUN`	`0xF5`	**Executes the program in the instruction queue.**
`SVEEP`	`0xF4`	**Saves current script to EEPROM (TODO footnote)**
`LDEEP`	`0xF3`	**Loads script from EEPROM into instruction queue (TODO footnote)**
`DELEEP`	`0xEE`	**Saves current script to EEPROM**
`NUKE 'Y'`	`0xDD`	**Resets badge to factory settings (TODO footnote)**
`MENU`	`0xFF`	**Exits Scripting Mode; returns badge to menu Mode**

TODO Footnotes:

1 Using `SVEEP` copies the current instruction queue to EEPROM so that it can be loaded again after the badge is reset. This will overwrite any script currently saved in EEPROM.

2 Using `LDEEP` copies the script currently saved in EEPROM to the instruction queue. This will overwrite any script currently contained in the instruction queue. If there is no script stored in EEPROM, the `LDEEP` command has the effect of clearing the instruction queue.

3 CAUTION: the `NUKE` command will destroy all puzzle progress and any saved Infernal script. When the `NUKE` command is entered, it must be confirmed with the confirmation character 'Y' (`0x59`). Any other value will cancel the command.

3.2 Move Instructions

Each of the following instructions loads an unsigned 8-bit value into a register. The value to be loaded can be an explicit value (*v*), or come from another register (**r* or **WREG*).

Mnemonic	Code	Function
MOVVW *v*	0x01	*v* → WREG
MOVRW *r*	0x02	*r → WREG
MOVWR *r*	0x03	*WREG → *r*
MOVVC *v*	0x04	*v* → COMP
MOVRC *r*	0x05	**r* → COMP
MOVWC	0xFA	*WREG → COMP

3.3 Flow Control Instructions

Mnemonic	Code	Function
LBL *l*	0x06	**Creates label *l* in place**
BR *l*	0x07	**Unconditional branch to label *l***
BEQ *l*	0x08	**Branch to *l* if *WREG == *COMP**
BNE *l*	0x09	**Branch to *l* if *WREG != *COMP**
BGT *l*	0x0A	**Branch to *l* if *WREG > *COMP**
BLT *l*	0x0B	**Branch to *l* if *WREG < *COMP**
CALL *l*	0x0C	**Call subroutine at label *l*** (TODO footnote)
RETURN	0xF7	**Return from subroutine through LINK register**

3.4 Arithmetic Instructions

Mnemonic	Code	Function
CLR	0xFD	0x00 → WREG
CLRR *r*	0x0F	0x00 → *r*
INC	0xFC	*WREG + 1 → WREG
INCR *r*	0x0D	**r* + 1 → *r*
DEC	0xFB	*WREG - 1 → WREG
DECR *r*	0x0E	**r* - 1 → *r*
ADDVW *v*	0x10	*v* + *WREG → WREG
ADDRW *r*	0x11	**r* + *WREG → WREG
SUBVW *v*	0x14	|*v* - *WREG| → WREG (TODO footnote)
SUBRW *r*	0x15	|**r* - *WREG| → WREG (TODO footnote)

Note: SUBVW and SUBRW place the absolute value of the result of the subtraction operation into WREG.

3.5 Logic Instructions

Mnemonic	Code	Function
ANDVW *v*	0x19	*v* **AND** *WREG → WREG
ANDRW *r*	0x1A	*****r* **AND** *WREG → WREG
ORVW *v*	0x1B	*v* **OR** *WREG → WREG
ORRW *r*	0x1C	*****r* **OR** *WREG → WREG
XORVW *v*	0x1D	*v* **XOR** *WREG → WREG
XORRW *r*	0x1E	*****r* **XOR** *WREG → WREG
BSL *v*	0x12	**Shift WREG left by** *v* **bits**
BSR *v*	0x13	**Shift WREG right by** *v* **bits**

3.6 Peripheral Control and Miscellaneous Instructions

Mnemonic	Code	Function
NOP	0x00	**No operation**
WINP	0xEF	**Halt until value in WHEEL is changed**
DUS *v*	0xEE	**Halt for** *v* **microseconds**
DMS *v*	0xED	**Halt for** *v* **milliseconds**
DS *v*	0xEC	**Halt for** *v* **seconds**
END	0xF9	**Must be the final instruction in the instruction queue.**

4 Examples

4.1 Counting

The following script counts from 0 to 255 and displays each number on the badge in binary format for one second.

```
1 CLR             // Set WREG to 0
2 MOVVC   0xFF    // Set COMP to 255
3 LBL     0x01    // Create label 1
4 MOVWR   DISP    // Copy WREG value to DISP
5 INC             // Increment WREG by 1
6 DS      0x01    // Delay for one second
7 BNE     0x01    // If the values in WREG and COMP are not equal,
                     jump to label 1
8 END
```

This script uses the COMP register to set a final value for WREG. It then increments the value in WREG until it matches the value in COMP. Each value that WREG takes on is displayed on the badge by copying it to the DISP register.