Cellular Automata Research Platform
Instruction Set Architecture

2015-06-14

Contents

1 Introduction

This document is a complete specification of the instruction set for the Cellular Automata Research Platform. It documents all effects and possible side effects of every instruction.

Unless otherwise stated, an instruction completes in one cycle. However, keep in mind that multi-word instructions require multiple cycles to send over PCI Express.

When a bit vector is broken into multiple words, the least significant part is always listed first.

Instructions

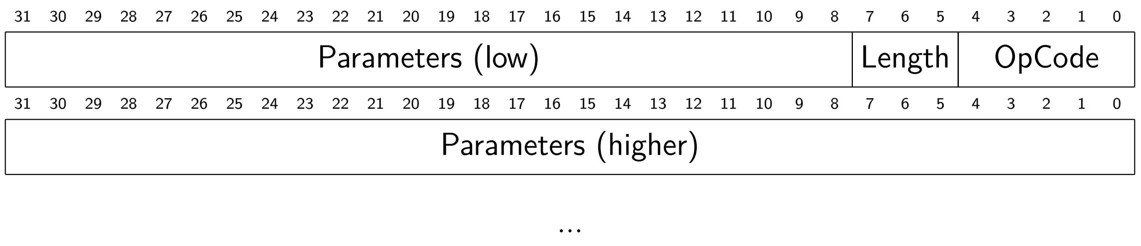
Each instruction is 256 bits and consists of a 5-bit operation code, a 3-bit length field and up to 248 bits of parameters.

The operation code specifies what kind of instruction it is, and how the parameters should be parsed.

The length field is used to improve communication speed by only transmitting the necessary parts of an instruction; It is zero-extended back to 256 bits by the fetch module. The field directly specifies the number of words after the first that are sent.

The parameters are of different types and lengths for each instruction. Please see the individual instruction pages.

Instruction Format



Rules

Rules consists of conditions for each cell in the neighborhood and a result that will be applied to the cell if the conditions match.

Each condition contains a type, a state and a bit for each that marks if it should be checked. The result format is identical except for that the check bits are exchanged with change bits that mark which parts of the cell should change if all conditions match.

In the formats below, [type bits] is assumed to be 5 and [states bits] 1 for the purpose of having everything nicely align to bytes.

Rule Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Condition X–								Condition X+								Condition Self								Result							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Condition Z–								Condition Z+								Condition Y–								Condition Y+							

Condition Format

7	6	5	4	3	2	1	0
Type					Check T	State	Check S

Result Format

7	6	5	4	3	2	1	0
Type					Change T	State	Change S

Notes

For a rule to be counted as a hit, all conditions must match and at least one change bit must be set.

Conditions for Z are ignored when [matrix depth] is 1.

LUTs

The indexing for the look-up tables is (Z–,Z+,Y–,Y+,X–,X+,Self). For each of these indexes, the next cell state is specified. The least significant index is written first (to the right).

In the format below, [state bits] is assumed to be 1 since it is the only value currently supported. This allows the entries for (Y–,Y+,X–,X+,Self) to fit exactly within one word.

LUT Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Y–,Y+,X–,X+,Self) when (Z–,Z+) is 00																															
(Y–,Y+,X–,X+,Self) when (Z–,Z+) is 01																															
(Y–,Y+,X–,X+,Self) when (Z–,Z+) is 10																															
(Y–,Y+,X–,X+,Self) when (Z–,Z+) is 11																															

Notes

The Z parts are ignored when [matrix depth] is 1.

2 General Instructions

This section covers instructions that are not used directly or do not fit into any of the following sections.

No Operation

31–8	7–5	4–0
▨	0 0 0	0 0 0 0 0

Format

nop()

Purpose

To do nothing for one cycle.

Description

Nothing is done for one cycle.

Read Information

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								0	0	0	0	0	0	0	1

Format

read_information()

Purpose

To retrieve information about the system.

Description

The following words are put into the Send Buffer.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
[matrix depth]								[matrix height]								[matrix width]								[matrix wrap]							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
[counter bits]								[counter amount]								[type bits]								[state bits]							

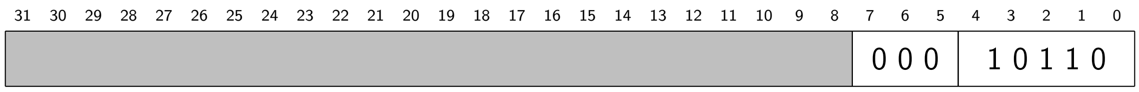
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
[rule amount]																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
[fitness parameters]																[fitness words]								[fitness id]							

Notes

This instruction takes 4 cycles.

Read Fitness



Format

read_fitness()

Purpose

To retrieve a fitness value.

Description

[fitness words] words are transfered from the Fitness Buffer to the Send Buffer.

Notes

This instruction takes [fitness words] cycles.

Swap Cell Storage

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
▨	0 0 0	1 0 1 0 0

Format

swap_cell_storage()

Purpose

To swap the contents of the two brams within the cell storage.

Description

Cell BRAM A and Cell BRAM B are remapped so that the contents appear to have been swapped.

Reset Buffers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								0	0	0	1	0	1	0	1

Format

reset_buffers()

Purpose

To clear the Rule Vector, Live Count and Fitness Buffers.

Description

The read and write pointers of the circular FIFO buffers are set to 0. This makes them appear to be empty.

Notes

If the Fitness module is processing data, the contents of the Live Count and Fitness Buffers may become undefined.

If the Fitness buffer is full, this incruction should be called an additional time after any pending data from Fitness has been transfered.

3 Development Instructions

This section covers all instructions affecting the development module. This includes writing rules, setting active rules, running development and reading data for which rules have triggered.

Read Rule Vectors

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
N	▨	0 0 0	0 0 0 1 0

Format

read_rule_vectors(N)

Purpose

To retrieve N rule vectors.

Description

N rule vectors are placed into the Send Buffer. Each consists of [rule amount] bits, where the first bit (rule zero) is always 1. The Send Buffer is word-aligned after each rule vector by padding with 0.

Example

Assume a system with [rule amount] set to 48, where rules 13 and 47 have triggered. read_rule_vectors(1) will put the following words into the Send Buffer.

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 1

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Notes

This instruction takes [words per rule vector] * N cycles.

When there are no rule vectors available and less than N have been sent, this instruction waits.

Read Rule Numbers

31–8	7–5	4–0
	0 0 0	0 0 0 1 1

Format

read_rule_numbers()

Purpose

To retrieve the last rule that triggered for each cell during the previous development step.

Description

Rule numbers for the entire matrix is put into the Send Buffer. Each consists of log_2[rule amount] bits, sent in raster order (first X, then Y, then Z). A value of 0 means that no rules triggered. The Send Buffer is word-aligned after each row by padding with 0. If a rule number would be split across two words, it is instead aligned to the next word.

Example

Assume a system with [matrix depth] set to 1, [matrix height] set to 2, [matrix width] set to 3 and [rule amount] set to 256. If rule 2 triggered for all cells in the fist row and rule 8 for all in the second, read_rule_numbers() will put the following words into the Send Buffer.

31–24	23–16	15–8	7–0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0	0 0 0 0 0 0 1 0	0 0 0 0 0 0 1 0

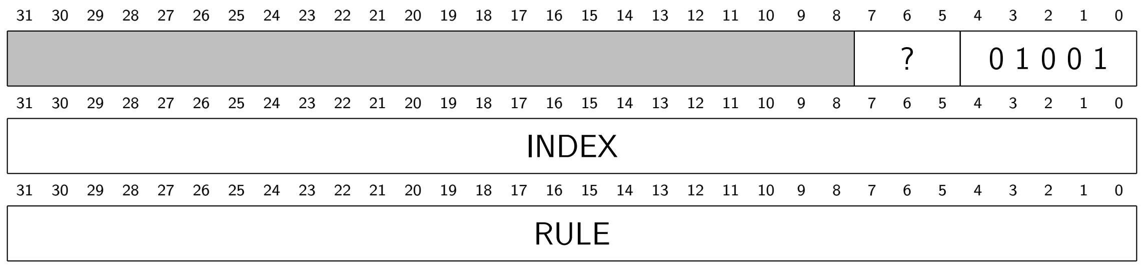
31–24	23–16	15–8	7–0
0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0	0 0 0 0 1 0 0 0	0 0 0 0 1 0 0 0

Notes

The execution time (T) for this instruction depends on [matrix depth] (M_Z), [matrix height] (M_Y), [matrix width] (M_X) and [rule amount] (R_A).

$$T = M_Z M_Y \left\lceil \frac{M_X}{\max\left(\left\lfloor \frac{32}{\lceil \log_2 R_A \rceil} \right\rfloor, M_X\right)} \right\rceil + 1$$

Write Rule



Format

write_rule(RULE, INDEX)

Purpose

To write a development rule.

Description

RULE is written to Rule BRAM at address INDEX. The length of RULE varies depending on [matrix depth], [type bits] and [state bits]. It is sent as one continuous piece spanning multiple words. The instruction length field is adjusted accordingly.

Notes

INDEX is cropped to the number of bits in [rule amount].

Set Active Rules

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N (low)																▨	▨	▨	▨	▨	▨	▨	▨	0	0	1	0	1	0	1	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	N (high)															

Format

set_rules_active(N)

Purpose

To set the number of rules that are currently active, so others can be skipped to reduce development time.

Description

Rules 1 to N is set to active (rule 0 is reserved). If N is 0, no rules will be set to active.

Notes

N is cropped to the number of bits in [rule amount]. If this is 16 or less, the second word can be discarded (and instruction length field set to 0).

Develop

31–8	7–5	4–0
	0 0 0	1 0 0 0 0

Format

develop()

Purpose

To perform development on all cells.

Description

The cells in Cell BRAM A are fetched and tested against all active rules. If a rule matches a cell, the state and/or type of the cell is changed based on the rule. Rules of higher index override those of lower index. The developed cells are then stored in Cell BRAM B.

The lastly matched rule of each cell is stored in Rule Number BRAM, and a list of all rules with a match is stored to the Rule Vector Buffer.

Notes

An overridden rule will be listed as having a match, but all its effects are discarded.

The execution time (T) for this instruction depends on [matrix depth] (M_Z), [matrix height] (M_Y), [rules active] (R_A) and [rules tested in parallel] (R_{TIP}).

$$T_{3D} = M_Z M_Y \max\left(\frac{R_A + 1}{R_{TIP}}, 7\right) + 6$$

$$T_{2D} = M_Y \max\left(\frac{R_A + 1}{R_{TIP}}, 5\right) + 4$$

4 Cell Storage Instructions

This section covers all instructions for writing and reading states and types to/from the cell storage.

Read One State

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
Z	Y	X	0 0 0	0 0 0 0 4

Format

read_state(Z, Y, X)

Purpose

To retrieve the state of the cell at (Z, Y, X).

Description

The state of cell (Z, Y, X) is put into the Send Buffer. The Send Buffer is then word-aligned by padding with 0. Accessing cells outside the matrix dimensions yields undefined states.

Read All States

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	0	0	0	0	5

Format

read_states()

Purpose

To retrieve the state of all cells.

Description

The states of all cells are put into the Send Buffer in raster order (first X, then Y, then Z). The Send Buffer is word-aligned after each row by padding with 0. If a state would be split across two words, it is instead aligned to the next word.

Notes

The execution time (T) for this instruction depends on [matrix depth] (M_Z), [matrix height] (M_Y), [matrix width] (M_X) and [state bits] (B_S).

$$T = M_Z M_Y \left\lceil \frac{M_X}{\max\left(\left\lfloor \frac{32}{B_S} \right\rfloor, M_X\right)} \right\rceil + 1$$

Read One Type

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z								Y								X								0	0	0	0	0	0	0	6

Format

read_type(Z, Y, X)

Purpose

To retrieve the type of the cell at (Z, Y, X).

Description

The type of cell (Z, Y, X) is put into the Send Buffer. The Send Buffer is then word-aligned by padding with 0. Accessing cells outside the matrix dimensions yields undefined types.

Read All Types

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
▨	0 0 0	0 0 0 0 7

Format

read_types()

Purpose

To retrieve the types of all cells.

Description

The types of all cells are put into the Send Buffer in raster order (first X, then Y, then Z). The Send Buffer is word-aligned after each row by padding with 0. If a type would be split across two words, it is instead aligned to the next word.

Notes

The execution time (T) for this instruction depends on [matrix depth] (M_Z), [matrix height] (M_Y), [matrix width] (M_X) and [type bits] (B_T).

$$T = M_Z M_Y \left\lceil \frac{M_X}{\max\left(\left\lfloor \frac{32}{B_T} \right\rfloor, M_X\right)} \right\rceil + 1$$

Write One State

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z								Y								X								0 0 1			0 1 1 0 1				

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STATE																															

Format

write_state(Z, Y, X, STATE)

Purpose

To write one state.

Description

State (Z, Y, X) in Cell BRAM A is set to STATE.

Notes

Each coordinate is cropped to the bits in its matrix dimension.

STATE is cropped to [state bits].

If row (Z, Y) is outside the defined matrix, data will still be written but only accessible by read instructions.

If X is outside the defined matrix, nothing will happen.

Write Row of States

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z								Y								X								?			0	1	1	0	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STATES																															

Format

write_states(Z, Y, X, STATES)

Purpose

To write one row (or as many can fit an instruction) of states.

Description

STATES is a list of states in little-endian order. It is either [matrix width] or as many can fit 224 bits in length. Each state is [state bits] long.

The states are written to Cell BRAM A at row (Z, Y). They are offset so the first state is written to position X within the row. States offset to [matrix width] or more are discarded.

The length of STATES varies depending on [matrix width] and [state bits]. It is sent as one continuous piece spanning multiple words. The instruction length field is adjusted accordingly.

Notes

Each coordinate is cropped to the bits in its matrix dimension.

If row (Z, Y) is outside the defined matrix, data will still be written but only accessible by read instructions.

Write One Type

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z								Y								X								0	0	1	0	1	1	0	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TYPE																															

Format

write_types(Z, Y, X, TYPE)

Purpose

To write one state.

Description

Type (Z, Y, X) in Cell BRAM A is set to TYPE.

Notes

Each coordinate is cropped to the bits in its matrix dimension.

TYPE is cropped to [type bits].

If row (Z, Y) is outside the defined matrix, data will still be written but only accessible by read instructions.

If X is outside the defined matrix, nothing will happen.

Write Row of Types

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z								Y								X								?			0	1	1	1	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TYPES																															

Format

write_types(Z, Y, X, TYPES)

Purpose

To write one row (or as many can fit an instruction) of types.

Description

TYPES is a list of types in little-endian order. It is either [matrix width] or as many can fit 224 bits in length. Each type is [type bits] long.

The types are written to Cell BRAM A at row (Z, Y). They are offset so the first type is written to position X within the row. Types offset to [matrix width] or more are discarded.

The length of TYPES varies depending on [matrix width] and [type bits]. It is sent as one continuous piece spanning multiple words. The instruction length field is adjusted accordingly.

Notes

Each coordinate is cropped to the bits in its matrix dimension.

If row (Z, Y) is outside the defined matrix, data will still be written but only accessible by read instructions.

Fill Cells

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
TYPE	STATE	0 0 0	0 1 0 1 0

Format

fill_cells(STATE, TYPE)

Purpose

To set the state and type of all cells.

Description

STATE and TYPE is written to each cell in Cell BRAM A.

Notes

STATE is cropped to [state bits].

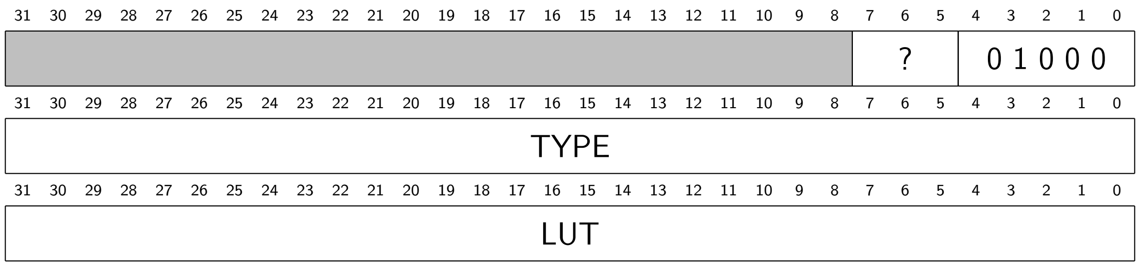
TYPE is cropped to [type bits].

This instruction takes [matrix depth] * [matrix height] cycles.

5 Cellular Automaton Instructions

This section covers all instructions affecting the Cellular Automaton. This includes writing look-up tables, configuring the CA, running the CA, and reading back the new states.

Write LUT



Format

write_lut(LUT, TYPE)

Purpose

To write a type to lookup table conversion entry.

Description

LUT is written to LUT BRAM at address TYPE. The length of LUT varies depending on [matrix depth]. It is sent as one continuous piece spanning multiple words. The instruction length field is adjusted accordingly.

Notes

TYPE is cropped to [type bits].

Configure

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
	0 0 0	1 0 0 1 0

Format

config()

Purpose

To configure the sblock matrix.

Description

The cells in Cell BRAM B are fetched along with the LUTs corresponding to each of their types. The LUTs and states are then written to the sblocks.

Notes

The execution time (T) for this instruction depends on [matrix depth] (M_Z), [matrix height] (M_Y) and [lut configuration bits] (LUT_{CB}).

$$T_{3D} = M_Z M_Y \frac{128}{LUT_{CB}} + 2$$

$$T_{2D} = M_Y \frac{32}{LUT_{CB}} + 2$$

Readback

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								0	0	0	1	0	0	1	1

Format

readback()

Purpose

To read back cell states from the sblock matrix.

Description

The states of all sblocks are written to Cell BRAM B. Types in Cell BRAM B are preserved.

Notes

This instruction takes [matrix depth] * [matrix height] cycles.

Step

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5	4 3 2 1 0
STEPS	▨	0 0 0	1 0 0 0 1

Format

step(STEPS)

Purpose

To perform updates of the sblock matrix.

Description

The sblock matrix is updated STEPS times. After each step, the number of live cells (state equals 1) are counted and stored in the Live Count buffer.

6 Control Flow Instructions

This section covers all instructions that are related to the program memory. This includes those for storing, starting and exiting programs, in addition to control flow within the programs.

Break

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																								0	0	0	1	1	0	0	0

Format

break_out()

Purpose

To break out of a running program and restore control to the host.

Description

The Fetch module exits [read from memory] mode and enters [read from communication] mode.

Notes

This has no effect if the Fetch module is already in [read from communication] mode.

Store

31–16	15–8	7–5	4–0
ADDRESS	▨	0 0 0	1 1 0 1 0

Format

store(ADDRESS)

Purpose

To begin storage of a program to internal memory.

Description

The Fetch module exits [read from communication] mode and enters [save to memory] mode. The next instruction will be saved at address ADDRESS, and then each address thereafter.

Notes

This will be saved as a nop if the Fetch module is already in [save to memory] mode.

ADDRESS is cropped to [program counter bits].

End

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	1	1	0	1	1

Format

end()

Purpose

To end storage of a program to internal memory.

Description

The Fetch module exits [save to memory] mode and enters [read from communication] mode.

Notes

This will be parsed as a nop if the Fetch module is already in [read from communication] mode.

Jump

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRESS																▨	▨	▨	▨	▨	▨	▨	▨	0	0	0	1	1	1	0	0

Format

jump(ADDRESS)

Purpose

To begin execution of or jump within a program stored to internal memory.

Description

If the Fetch module is not in [read from memory] mode, it exits [read from communication] mode and enters [read from memory] mode. The program counter is then updated so the next instruction is the one at address ADDRESS.

Notes

ADDRESS is cropped to [program counter bits].

Jump Equal

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRESS																COUNTER								0	0	1	1	1	1	0	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VALUE																															

Format

jump_equal(ADDRESS, COUNTER, VALUE)

Purpose

To begin execution of or jump within a program stored to internal memory if a counter matches a value.

Description

If counter COUNTER is equal to VALUE, this instructions is exactly like jump(ADDRESS). Otherwise, it is discarded.

Notes

Accessing counter [counter amount] or higher yields undefined behavior.

ADDRESS is cropped to [program counter bits].

VALUE is cropped to [counter bits].

Increment Counter

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	COUNTER								0	0	0	1	1	1	1	0

Format

counter_increment(COUNTER)

Purpose

To increment a counter.

Description

Counter COUNTER is incremented by 1. If counter COUNTER is at maximum, it instead becomes 0.

Notes

Accessing counter [counter amount] or higher yields undefined behavior.

Reset Counter

31–16	15–8	7–5	4–0
▨	COUNTER	0 0 0	1 1 1 1 1

Format

counter_reset(COUNTER)

Purpose

To reset a counter.

Description

Counter COUNTER is set to 0.

Notes

Accessing counter [counter amount] or higher yields undefined behavior.