Operationskoder

Följande operationskoder beskrivs i denna bilaga:

Absolute value

74/7c Request/Command

byte *index* Destination variable index. 0..31.
byte *source* Source type for operand. Only 0 and 2 allowed.
short *argument* Argument for operand.

Compute the absolute value of the value specified by *source* and *argument* and store the result in variable *index*. The absolute value of the largest negative number, -32768, is defined to be the largest positive number, 32767.

83/8b Reply

void

Reply indicates success.

Alive

10/18 Request

void

Check whether or not the RCX is alive. If the PC receives a reply to this request, it assumes the RCX is alive and the connection is good.

e7/ef Reply

void

Reply indicates that the RCX is alive.

Add to variable

24/2c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Add the value specified by *source* and *argument* to the value of variable *index*, and store the result in variable *index*.

d3/db Reply

void

Reply indicates success.

And variable

84/8c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Compute the logical AND of the value specified by *source* and *argument* and the value of variable *index*, and store the result in variable *index*.

73/7b Reply

void

Reply indicates success.

Branch always far

72/xx Command

ubyte *offset* Offset for computing branch target address.
ubyte *extension* Extension to offset.

Branch to the target address specified by *offset* and *extension*.

If bit 0x80 of *offset* is 0, the target address computed as:

address of *offset* + *offset* + 128 × *extension*.

Otherwise, the target address computed as:

address of *offset* - *offset* - 128 × *extension* + 128.

Branch always near

27/xx Command

ubyte *offset* Offset for computing branch target address.

Branch to the target address specified by *offset*.

If bit 0x80 of *offset* is 0, the target address computed as:

address of *offset* + *offset*.

Otherwise, the target address computed as:

address of *offset* - *offset* + 128.

Call subroutine

17/xx Command

byte *subroutine* Index of subroutine to call. 0..7.

Call the subroutine with index *subroutine*. If the subroutine is not defined, do nothing.

The RCX only supports one subroutine return address per task. If one subroutine calls another subroutine, execution of all tasks stops when the original subroutine returns.

Clear message

90/xx Command

void

Clear the message buffer by setting it to zero. The message buffer stores a single byte and allows for communication between multiple RCX units.

See also: **send message**, **set message**.

Clear sensor value

d1/d9 Request/Command

byte *sensor* Index of sensor to clear. 0..2.

Clear the counter associated with the specified sensor by setting it to a value of zero.

26/2e Reply

void

Reply indicates success.

Clear timer

a1/a9 Request/Command

byte *timer* Index of timer to clear. 0..3.

Clear the specified timer by setting it to a value of zero.

56/5e Reply

void

Reply indicates success.

Datalog next

62/6a Request/Command

byte *source* Source type for next datalog entry. Only 0, 1, 9, and 14 allowed.
byte *argument* Argument for next datalog entry.

Set the next datalog entry to the value specified by *source* and *argument*. If the datalog is full, leave it unmodified.

95/9d Reply

byte *errorcode* Return value.

A return value of 0 indicates success, while a return value of 1 indicates that the datalog was full.

Decrement loop counter far

92/xx Command

ushort *offset* Offset for computing branch target address.

Decrement the current loop counter, then, if the loop counter is less than zero, pop the loop counter stack and branch to the target address specified by *offset*.

The branch target address is computed as:

address of first byte of *offset* + *offset*.

Note that *offset* is unsigned. Backward branching is not allowed with this operation.

Decrement loop counter near

37/xx Command

ubyte *offset* Offset for computing branch target address.

Decrement the current loop counter, then, if the loop counter is less than zero, pop the loop counter stack and branch to the target address specified by *offset*.

The branch target address is computed as:

address of *offset* + *offset*.

Note that *offset* is unsigned. Backward branching is not allowed with this operation.

Delete all subroutines

70/78 Request/Command

void

Delete all subroutines belonging to the current program.

87/8f Reply

void

Reply indicates success.

Delete all tasks

40/48 Request/Command

void

Stop execution and delete all tasks belonging to the current program.

b7/bf Reply

void

Reply indicates success.

Delete firmware

65/6d Request/Command

byte *key*[5] Key. Must be {1,3,5,7,11}.

If *key* is valid, stop execution and delete the firmware. Otherwise, do nothing. The key prevents the firmware from being accidentally erased.

Before the firmware may be replaced, it must be deleted.

92/9a Reply

void

Reply indicates success.

Delete subroutine

c1/c9 Request/Command

byte *subroutine* Index of subroutine to delete. 0..7.

Delete the specified subroutine. If the subroutine is currently being executed, the related task is stopped.

36/3e Reply

void

Reply indicates success.

Delete task

61/69 Request/Command

byte *task* Index of task to delete. 0..9.

Delete the specified task. If the task is currently running, execution of all tasks stops.

96/9e Reply

void

Reply indicates success.

Divide variable

44/4c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Divide the value of variable *index* by the value specified by *source* and *argument*, and store the result in variable *index*. If the *source* and *argument* specify a denominator of zero, the variable *index* is left unchanged.

b3/bb Reply

void

Reply indicates success.

Get battery power

30/38 Request

void

Request the battery voltage from the RCX.

c7/cf Reply

ushort *millivolts* Battery voltage.

Reply indicates the current voltage of the RCX battery, in mV. A fresh set of alkaline batteries typically has a reading of around 9.3V.

Get memory map

20/28 Request

void

Request the memory map from the RCX.

d7/df Reply

ushort *map*[94] Memory map.

Reply contains the user program memory map of the RCX. The memory map is an array of 94 16-bit big endian addresses, organized as follows:

Index	Description
0-7	Starting addresses, program 0, subroutines 0-7
8-15	Starting addresses, program 1, subroutines 0-7
16-23	Starting addresses, program 2, subroutines 0-7
24-31	Starting addresses, program 3, subroutines 0-7
32-39	Starting addresses, program 4, subroutines 0-7
40-49	Starting addresses, program 0, tasks 0-9
50-59	Starting addresses, program 1, tasks 0-9
60-69	Starting addresses, program 2, tasks 0-9
70-79	Starting addresses, program 3, tasks 0-9
80-89	Starting addresses, program 4, tasks 0-9
90	First datalog address
91	Next datalog address
92	First free address
93	Last valid address

Get value

12/1a Request

byte *source* Source type for value. Sources 2 and 4 not allowed.
byte *argument* Argument for value.

Request the value specified by *source* and *argument*.

e5/ed Reply

short *value* Return value.

Reply contains the requested value.

Get versions

15/1d Request

byte *key*[5] Key. Must be {1,3,5,7,11}.

Request the ROM and firmware versions from the RCX. If *key* is not valid, no reply is sent.

e2/ea Reply

short *rom*[2] ROM version number.
short *firmware*[2] Firmware version number.

Reply contains the ROM and firmware version numbers. Each version number is composed of two big endian shorts; the first is the major version number and the second is the minor version number.

The ROM always has major and minor version numbers 3 and 1. The firmware shipped with the Robotics Invention System has major and minor version numbers 3 and 9. When no firmware is installed, both firmware version numbers are 0.

Multiply variable

54/5c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Multiply the value of variable *index* by the value specified by *source* and *argument*, and store the result in variable *index*.

a3/ab Reply

void

Reply indicates success.

Or variable

94/9c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Compute the logical OR of the value specified by *source* and *argument* and the value of variable *index*, and store the result in variable *index*.

63/6b Reply

void

Reply indicates success.

Play sound

51/59 Request/Command

byte *sound* Sound type. 0..5.

Play the specified sound.

There are six avaiable sound types:

Index	Description
0	Blip
1	Beep beep
2	Downward tones
3	Upward tones
4	Low buzz
5	Fast upward tones

a6/ae Reply

void

Reply indicates success.

Play tone

23/2b Request/Command

short *frequency* Tone frequency.
byte *duration* Tone duration.

Play the sound specified by *frequency* and *duration*. The tone frequency is measured in Hz, and the tone duration is measured in 1/100ths of a second.

d4/dc Reply

void

Reply indicates success.

Power off

60/68 Request/Command

void

Turn off the RCX. If the power down delay is zero, do nothing.

97/9f Reply

void

Reply indicates success.

Return from subroutine

f6/xx Command

void

Return from subroutine. This opcode is intended to be used in the middle of a subroutine, since a return from subroutine opcode is automatically added to every subroutine that is downloaded.

Send message

b2/xx Command

byte *source* Source type for message. Only 0 and 2 allowed.
byte *argument* Argument for message.

Send the value specified by *source* and *argument* to other RCX units. The value is sent by broadcasting a **set message** request over the infrared link.

Set datalog size

52/5a Request/Command

short *size* New datalog size.

Allocate and initialize a datalog with space for *size* data entries. A single extra entry is always allocated to hold the current size of the datalog, which is initialized to one since this size value is stored as the first entry in the datalog.

a5/ad Reply

byte *errorcode* Return value.

A return value of 0 indicates success, while a return value of 1 indicates that there is insufficient memory to allocate a datalog of the requested size.

Set display

33/3b Request/Command

byte *source* Source type for device. Only 0 and 2 allowed.
short *argument* Argument for device.

Display the input/output value associated with the device specified by *source* and *argument*. Valid devices are:

Index	Description
0	Watch
1	Sensor 1
2	Sensor 2
3	Sensor 3
4	Motor A
5	Motor B
6	Motor C

This operation controls the same functionality as the View button on the face of the RCX.

c4/cc Reply

void

Reply indicates success.

Set loop counter

82/xx Command

byte *source* Source type for counter value. Only 0, 2, and 4 allowed.
byte *argument* Argument for counter value.

Push the loop counter stack, then set the topmost loop counter to the value specified by *source* and *argument*. There are four loop counters. If more than four loops are nested, the loop counter stack is not pushed before the topmost value is set.

Set message

f7/xx Request/Command

byte *message* Message value.

Set the value of the message buffer to *message*. This is the only request with no matching reply.

Set motor direction

e1/e9 Request/Command

byte *code* Bit field to specify a direction and up to three motors.

Set the direction of the motors according to *code*. The bits of *code* have the following meanings:

Bit	Description
0x01	Modify direction of motor A
0x02	Modify direction of motor B
0x04	Modify direction of motor C
0x40	Flip the directions of the specified motors
0x80	Set the directions of the specified motors to forward

If both bit 0x40 and bit 0x80 are 0, the directions of the specified motors are set to reverse. If both bit 0x40 and bit 0x80 are 1, the directions of the specified motors are flipped.

16/1e Reply

void

Reply indicates success.

Set motor on/off

21/29 Request/Command

byte *code* Bit field to specify an on/off state and up to three motors.

Set the on/off state of the motors according to *code*. The bits of *code* have the following meanings:

Bit	Description
0x01	Modify on/off state of motor A
0x02	Modify on/off state of motor B
0x04	Modify on/off state of motor C
0x40	Turn off the specified motors
0x80	Turn on the specified motors

If both bit 0x40 and bit 0x80 are 0, the specified motors are set to float, which allows the shafts of the specified motors to spin freely. Setting both bit 0x40 and bit 0x80 to 1 turns on the specified motors.

d6/de Reply

void

Reply indicates success.

Set motor power

13/1b Request/Command

byte *motors* Bit field to specify up to three motors.
byte *source* Source type for power level. Only 0, 2, and 4 allowed.
byte *argument* Argument for power level.

Set the power level of the motors specified by *motors* to the value specified by *source* and *argument*. The bits of *motors* have the following meanings:

Bit	Description
0x01	Modify power level of motor A
0x02	Modify power level of motor B
0x04	Modify power level of motor C

Valid power levels are between 0 and 7, inclusive. Other power levels are ignored.

e4/ec Reply

void

Reply indicates success.

Set power down delay

b1/b9 Request/Command

byte *minutes* Power down delay.

Set the power down delay (0..99) to the specified value, which is measured in minutes. A power down delay of 0 instructs the RCX to remain on indefinitely and causes the **power off** opcode to be ignored.

46/4e Reply

void

Reply indicates success.

Set program number

91/99 Request/Command

byte *program* Program number. 0..4.

Stop execution and set the current program number to the specified value.

66/6e Reply

void

Reply indicates success.

Set sensor mode

42/4a Request/Command

byte *sensor* Index of sensor to modify. 0..2.
byte *code* Packed sensor slope and mode.

Set the slope and mode of sensor number *sensor* to the value specified by *mode*, and clear that sensor's value. The bits of *mode* are split into two portions. Bits 0-4 contain a slope value in 0..31, while bits 5-7 contain the mode, 0..7. The eight modes, which control the value returned by the sensor, are:

Mode	Name	Description
0	Raw	Value in 0..1023.
1	Boolean	Either 0 or 1.
2	Edge count	Number of boolean transitions.
3	Pulse count	Number of boolean transitions divided by two.
4	Percentage	Raw value scaled to 0..100.
5	Temperature in °C	1/10ths of a degree, -19.8..69.5.
6	Temperature in °F	1/10ths of a degree, -3.6..157.1.
7	Angle	1/16ths of a rotation, represented as a signed short.

The slope value controls 0/1 detection for the three boolean modes. A slope of 0 causes raw sensor values greater than 562 to cause a transition to 0 and raw sensor values less than 460 to cause a transition to 1. The hysteresis prevents bouncing between 0 and 1 near the transition point. A slope value in 1..31, inclusive, causes a transition to 0 or to 1 whenever the difference between consecutive raw sensor values exceeds the slope. Increases larger than the slope result in 0 transitions, while decreases larger than the slope result in 1 transitions. Note the inversions: high raw values correspond to a boolean 0, while low raw values correspond to a boolean 1.

b5/bd Reply

void

Reply indicates success.

Set sensor type

32/3a Request/Command

byte *sensor* Index of sensor to modify. 0..2.
byte *type* Sensor type. 0..4.

Set the type of sensor number *sensor* to *type*, and set the mode of that sensor to a default value. Valid types and their default modes are:

Type	Description	Default Mode
0	Raw	Raw
1	Touch	Boolean
2	Temperature	Temperature in °C
3	Light	Percentage
4	Rotation	Angle

c5/cd Reply

void

Reply indicates success.

Set time

22/2a Request/Command

byte *hours* Current hour. 0..23.
byte *minutes* Current minute. 0..59.

Set the current time to *hours* and *minutes*.

d5/dd Reply

void

Reply indicates success.

Set transmitter range

31/39 Request/Command

byte *range* Transmitter range. 0 or 1.

Set the transmitter range. 0 indicates short range, 1 indicates long range. Other values are ignored.

c6/ce Reply

void

Reply indicates success.

Set variable

14/1c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. All sources are allowed.
short *argument* Argument for second operand.

Set the value of variable *index* to the value specified by *source* and *argument*.

e3/eb Reply

void

Reply indicates success.

Sign variable

64/6c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Set the value of variable *index* to 0 if the value specified by *source* and *argument* is equal to 0, 1 if the value specified by *source* and *argument* is positive, or -1 if the value specified by *source* and *argument* is negative.

93/9b Reply

void

Reply indicates success.

Start firmware download

75/7d Request

short *address* Firmware entry address. Typically 0x8000.
short *checksum* Firmware checksum. Typically 0xc278.
byte *unknown* Unknown or unused value. Always 0.

Prepare the RCX for a firmware download starting at address 0x8000. The checksum is computed by taking the sum of all firmware bytes modulo 2^{16}. The specified address is used as the firmware entry point when the firmware is unlocked.

If firmware is already installed, this request is ignored and no response is sent.

82/8a Reply

byte *errorcode* Return value.

A return value of 0 indicates success, while any other return value indicates failure.

Start subroutine download

35/3d Request

byte *unknown* Unknown or unused value. Always 0.
short *subroutine* Subroutine index. 0..7.
short *length* Subroutine length.

Prepare the RCX for a download of subroutine number *subroutine* of the current program. Space for *length* bytes is allocated in the memory map by moving other data as needed.

c2/ca Reply

byte *errorcode* Return value.

A return value of 0 indicates success, a return value of 1 indicates that there is insufficient memory for a subroutine of the requested size, and a return value of 2 indicates that the subroutine index was invalid.

Start task

71/79 Request/Command

byte *task* Index of task to start. 0..9.

Start the specified task, or restart it if it is currently active. If the specified task is not defined, nothing happens.

86/8e Reply

void

Reply indicates success.

Start task download

25/2d Request

byte *unknown* Unknown or unused value. Always 0.
short *task* Task index. 0..9.
short *length* Task length.

Prepare the RCX for a download of task number *task* of the current program. Space for *length* bytes is allocated in the memory map by moving other data as needed.

d2/da Reply

byte *errorcode* Return value.

A return value of 0 indicates success, a return value of 1 indicates that there is insufficient memory for a task of the specified size, and a return value of 2 indicates that the task index was invalid.

Stop all tasks

50/58 Request/Command

void

Stops execution.

a7/af Reply

void

Reply indicates success.

Stop task

81/89 Request/Command

byte *task* Index of task to stop. 0..9.

Stops execution of the specified task.

76/7e Reply

void

Reply indicates success.

Subtract from variable

34/3c Request/Command

byte *index* Destination and first operand variable index. 0..31.
byte *source* Source type for second operand. Only 0 and 2 allowed.
short *argument* Argument for second operand.

Subtract the value specified by *source* and *argument* from the value of variable *index*, and store the result in variable *index*.

c3/cb Reply

void

Reply indicates success.

Test and branch far

95/xx Command

byte *opsrc1* Packed operator and source type for first value. Sources 4 and 8 not allowed.
byte *src2* Source type for second value. Sources 2, 4, and 8 not allowed.
short *arg1* Argument for first value.
byte *arg2* Argument for second value.
short *offset* Offset for computing branch target address.

Compare two values against one another and branch if the comparison is true. Bits 0-3 of *opsrc1* contain the source for the first value, while *arg1* contains the argument for the first value. The second value is specified by *src2* and *arg2*. The comparison operator, which is stored in bits 6-7 of *opsrc1*, has a value in 0..3. The meanings of these values are:

Value	Description
0	First value less than or equal to second value
1	First value greater than or equal to second value
2	First value not equal to second value
3	First value equal to second value

The branch target address is computed as:

address of first byte of *offset* + *offset*.

Note that *offset* is signed. Backward branching is allowed with this operation.

Test and branch near

85/xx Command

byte *opsrc1* Packed operator and source type for first value. Sources 4 and 8 not allowed.
byte *src2* Source type for second value. Sources 2, 4, and 8 not allowed.
short *arg1* Argument for first value.
byte *arg2* Argument for second value.
byte *offset* Offset for computing branch target address.

Compare two values against one another and branch if the comparison is true. Bits 0-3 of *opsrc1* contain the source for the first value, while *arg1* contains the argument for the first value. The second value is specified by *src2* and *arg2*. The comparison operator, which is stored in bits 6-7 of *opsrc1*, has a value in 0..3. The meanings of these values are:

Value	Description
0	First value less than or equal to second value
1	First value greater than or equal to second value
2	First value not equal to second value
3	First value equal to second value

The branch target address is computed as:

address of first byte of *offset* + *offset*.

Note that *offset* is signed. Backward branching is allowed with this operation.

Transfer data

45/4d Request

short *index* Sequence number of data block.
short *length* Length of data block.
byte *data*[*length*] Data block.
byte *checksum* Checksum of data block.

Download block number *index*, containing *length* data bytes, to the RCX. The bytes are stored as required by the most recent **start firmware download, start subroutine download**, or **start task download** operation.

Block sequence numbers start at 1 and increase by one with each successive block transferred. The special sequence number 0 indicates the last block of a transfer. Once this sequence number is received by the RCX, another start download operation is required before additional data blocks may be transferred.

The checksum is computed by taking the sum of all bytes in the data block modulo 256. It is only checked for packets that have a non-zero sequence number.

Note that, for task and subroutine downloads, the RCX does not check that the total number of bytes sent was equal to the amount of space allocated by the corresponding start download operation. If too few bytes are sent, a portion of the downloaded task or subroutine will contain invalid code; if too many bytes are sent, other tasks and subroutines may become corrupted.

b2/ba Reply

byte *errorcode* Return value.

A return value of 0 indicates success, a return value of 3 indicates a block checksum failure, a return value of 4 indicates a firmware checksum error, and a return value of 6 indicates an invalid or missing download start. If the block sequence number is out of order, no reply is sent.

When block checksum error number 3 is received, the PC may either retransmit the erroneous block or restart the download. A block with index 0 cannot be retransmitted. Moreover, a bug in the ROM requires that the PC increment its block sequence number when retransmitting firmware data in response to a block checksum error.

During a firmware download, duplicate transfer data requests are handled incorrectly. If the PC does not receive a transfer data reply, block sequence numbers might get out of sync.

Unlock firmware

a5/ad Request

byte *key*[5] Key. Must be {76,69,71,79,174} = {"LEGO®"}.

Activate the firmware after a successful download.

52/5a Reply

byte *data*[25] Return value. Always {"Just a bit off the block!"}.

Reply indicates success.

Upload datalog

a4/ac Request

short *first* Index of first entry to upload.
short *count* Number of entries to upload.

Upload *count* datalog entries, starting with entry number *first*.

The datalog entry with index 0 always contains the current size of the datalog, which is guaranteed to be at least one since the current size entry is considered to be part of the datalog. After the current size entry are *size* data entries, where *size* is specified with the **set datalog size** operation and is initially zero.

It is an error to read an entry outside the valid range of the datalog.

53/5b Reply

dlrec *data*[*length*] Requested datalog entries.

Reply contains the requested datalog entries, stored as an array of *length* dlrec duples, where *length* was specified as the *count* in the corresponding request. Each dlrec is organized as follows:

byte *type* Type of datalog entry.
short *value* Value of datalog entry.

The *type* of each dlrec specifies the appropriate interpretation of the corresponding *value*. Valid types are:

Type	Description
0xff	Current datalog size
0x00-0x1f	Variable value (source 0, variables 0..31)
0x20-0x23	Timer value (source 1, timers 0..3)
0x40-0x42	Sensor reading (source 9, sensors 0..2)
0x80	Clock reading (source 14)

If an error occurs while reading the datalog, *length* is set to zero.

Wait

43/xx Command

byte *source* Source type for delay. Only 0, 2, and 4 allowed.
short *argument* Argument for delay.

Wait for the delay specified by *source* and *argument*. The delay is in 1/100ths of a second.

Sources and Arguments

This section describes the available sources and arguments. Sources are like addressing modes. They specify where and how to get certain operand values. There are 16 sources available, of which 13 apply to the RCX:

Source	Name	Argument	Description
0	Variable	Variable index, 0..31.	Returns value of specified variable.
1	Timer	Timer index, 0..3.	Returns value of specified timer, in 1/100ths of a second.
2	Immediate	Immediate value.	Returns specified immediate value.
3	Motor State	Motor index, 0..2.	Returns state of specified motor. See below.
4	Random	Maximum value.	Returns random value, 0..max.
5	*Reserved*	*N/A*	*Cybermaster only.*
6	*Reserved*	*N/A*	*Cybermaster only.*
7	*Reserved*	*N/A*	*Cybermaster only.*
8	Current Program	Ignored.	Returns current program number.
9	Sensor Value	Sensor index, 0..2.	Returns value of specified sensor.
10	Sensor Type	Sensor index, 0..2.	Returns type of specified sensor.
11	Sensor Mode	Sensor index, 0..2.	Returns mode of specified sensor.
12	Raw Sensor Value	Sensor index, 0..2.	Returns raw value of specified sensor, 0..1023.
13	Boolean Sensor Value	Sensor index, 0..2.	Returns boolean value of specified sensor, 0..1.
14	Clock	Must be 0.	Returns minutes since power on.
15	Message	Must be 0.	Returns value of message buffer.

Motor state is encoded as a single byte. Bits 0-2 contain the motor power, 0..7. The remaining bits are used as follows:

Bit	Description	Notes
0x08	Forward flag	0 if forward, 1 if reverse.
0x40	Off flag	1 if off.
0x80	On flag	1 if on.

If both bit 0x40 and bit 0x80 are 0, the specified motor is set to float.