Basic-10

BASIC to IC10 MIPS Compiler

for Stationeers

User Manual & Language Reference

Version 1.9.1

December 2025

Table of Contents

1. Introduction

What is Basic-10?

Basic-10 is a BASIC to IC10 compiler designed for the game Stationeers. It allows you to write programs in a beginner-friendly BASIC dialect, which are then compiled to IC10 MIPS assembly code that runs on the game's programmable Integrated Circuit (IC) chips.

Why Use Basic-10?

- Write readable, maintainable code instead of low-level assembly
- Automatic register allocation - no manual register management
- Built-in functions for math, timing, and device operations
- Real-time syntax checking and error highlighting
- Integrated simulator for testing without the game
- One-click deployment to Stationeers scripts folder

IC10

IC10 is the assembly language used by Integrated Circuits in Stationeers. Each IC chip has:

- 16 general-purpose registers (r0-r15)
- 6 device connection pins (d0-d5)
- A 512-value stack for temporary storage
- A maximum of 128 lines of code

Basic-10 handles all the complexity of register allocation and generates optimized IC10 code that fits within these constraints.

2. Getting Started

Your First Program

Here is a simple program that blinks a light on and off:

```
# My first IC10 program - Blink a light
ALIAS light d0    # Connect a light to pin d0

main:
    light.On = 1  # Turn light on
    SLEEP 1       # Wait 1 second
    light.On = 0  # Turn light off
    SLEEP 1       # Wait 1 second
    GOTO main     # Repeat forever
END
```

Understanding the Code

- Lines starting with # are comments (ignored by compiler)
- ALIAS creates a friendly name for device pins (d0-d5)
- main: is a label - a named location in your code
- SLEEP pauses execution for the specified seconds
- GOTO jumps to a label
- END marks the end of your program

The

Almost every IC10 program uses a main loop that runs continuously. This is essential because IC chips need to constantly monitor sensors and update device states.

```
main:
    # Read sensors and make decisions
    VAR temp = sensor.Temperature
    IF temp > 300 THEN
        heater.On = 0
    ENDIF

    YIELD         # IMPORTANT: Let game process
    GOTO main     # Loop back
END
```

IMPORTANT: Always include YIELD or SLEEP in your loops! Without it, your program will freeze the game due to infinite loop detection.

Connecting Devices

IC chips have 6 device pins (d0-d5). Connect devices in-game using cables, then reference them in your code with ALIAS:

```
ALIAS sensor d0      # Gas sensor on pin 0
ALIAS heater d1      # Wall heater on pin 1
```

```
ALIAS display d2     # LED display on pin 2

# Now use the friendly names
VAR temp = sensor.Temperature
heater.On = 1
display.Setting = 42
```

3. Language Reference

3.1 Variables & Constants

Variable Declaration

```
VAR temperature = 0     # Declare with initial value
VAR count               # Declare (defaults to 0)
LET x = 5               # Alternative syntax
x = 42                  # Reassign value
```

Constants

```
CONST MAX_TEMP = 373.15    # Named constant (cannot change)
DEFINE TARGET_PRESSURE 101  # IC10-style define (no = sign)
```

Arrays

```
DIM values(10)         # Declare array of 10 elements
values(0) = 100        # Set first element
VAR x = values(0)      # Read element
```

3.2 Operators

Arithmetic Operators

Operator	Description	Example	IC10
`a + b`	Addition	`x = 5 + 3`	add
`a - b`	Subtraction	`x = 10 - 4`	sub
`a * b`	Multiplication	`x = 3 * 4`	mul
`a / b`	Division	`x = 10 / 2`	div
`a MOD b`	Modulo	`x = 10 MOD 3`	mod
`a ^ b`	Power	`x = 2 ^ 3`	exp+log
`-a`	Negation	`x = -value`	sub

Compound Assignment Operators (v1.9.0)

Operator	Equivalent	Description	IC10
`x += n`	`x = x + n`	Add and assign	add
`x -= n`	`x = x - n`	Subtract and assign	sub
`x *= n`	`x = x * n`	Multiply and assign	mul
`x /= n`	`x = x / n`	Divide and assign	div

Increment/Decrement Operators (v1.9.0)

Operator	Description	Example
`++x`	Prefix increment (returns new)	`y = ++x  # x=11, y=11`
`x++`	Postfix increment (returns old)	`y = x++  # x=11, y=10`
`--x`	Prefix decrement (returns new)	`y = --x  # x=9, y=9`
`x--`	Postfix decrement (returns old)	`y = x--  # x=9, y=10`

Comparison Operators

Operator	Description	Example
`=` or `==`	Equal to	`IF x = 5 THEN`
`<>` or `!=`	Not equal to	`IF x <> 0 THEN`
`<`	Less than	`IF temp < 300 THEN`
`>`	Greater than	`IF pressure > 100 THEN`
`<=`	Less than or equal	`IF charge <= 0.2 THEN`
`>=`	Greater than or equal	`IF ratio >= 0.21 THEN`

Logical Operators

Operator	Description	Example
`a AND b`	Logical AND	`IF a > 0 AND b > 0`
`a OR b`	Logical OR	`IF error OR warning`
`NOT a`	Logical NOT	`IF NOT active THEN`

Bitwise Operators

Operator	Description	IC10
`a & b or BAND(a,b)`	`Bitwise AND`	`and`
`a | b or BOR(a,b)`	`Bitwise OR`	`or`
`a ^ b or BXOR(a,b)`	`Bitwise XOR`	`xor`
`~a or BNOT(a)`	`Bitwise NOT`	`nor`
`a << n or SHL(a,n)`	`Shift left`	`sll`
`a >> n or SHR(a,n)`	`Shift right`	`srl`

Bit shift example:

```
VAR a = 1
VAR b = a << 4    # b = 16 (shift left 4 bits)
VAR c = 16 >> 2   # c = 4 (shift right 2 bits)
VAR d = 5 ^ 3     # d = 6 (XOR: 101 ^ 011 = 110)
```

3.3 Control Flow

IF...THEN...ELSE...ENDIF

```
IF temperature > 300 THEN
    heater.On = 0
ELSEIF temperature < 290 THEN
    heater.On = 1
ELSE
    # Temperature is acceptable
ENDIF
```

SELECT CASE

```
SELECT CASE mode
    CASE 0
        # Handle mode 0
    CASE 1
        # Handle mode 1
    DEFAULT
        # Default handling
END SELECT
```

Labels and GOTO

```
main:
    # Main program code
    YIELD
    GOTO main      # Jump back to main

errorHandler:
    # Handle errors
    GOTO main
```

3.4 Loops

WHILE...WEND

```
WHILE temperature > 300
    heater.On = 0
    YIELD
WEND
```

FOR...NEXT

```
FOR i = 1 TO 10
    display.Setting = i
    YIELD
NEXT i

FOR j = 10 TO 0 STEP -1
    # Countdown
NEXT j
```

DO...LOOP

```
DO
    pump.On = 1
    YIELD
LOOP UNTIL pressure > 100
```

Loop Control: BREAK and CONTINUE

```
# BREAK - exit loop immediately
WHILE 1
    IF done THEN BREAK
    YIELD
WEND

# CONTINUE - skip to next iteration
FOR i = 1 TO 10
    IF i MOD 2 = 0 THEN CONTINUE
    # Only processes odd numbers
NEXT i
```

3.5 Subroutines

GOSUB and RETURN

```
main:
    GOSUB ReadSensors
    GOSUB UpdateOutputs
    YIELD
    GOTO main

ReadSensors:
    temp = sensor.Temperature
```

```
    pressure = sensor.Pressure
    RETURN

UpdateOutputs:
    display.Setting = temp
    RETURN
```

SUB and FUNCTION

```
SUB UpdateDisplay
    display.Setting = temp
END SUB

FUNCTION Clamp(val, minVal, maxVal)
    IF val < minVal THEN RETURN minVal
    IF val > maxVal THEN RETURN maxVal
    RETURN val
END FUNCTION

# Usage
CALL UpdateDisplay
VAR safe = Clamp(input, 0, 100)
```

4. Device Operations

Device Pins

Each IC chip has 6 device pins (d0-d5) plus a self-reference (db). Use ALIAS to give devices friendly names:

```
ALIAS sensor d0      # Device on pin 0
ALIAS pump d1        # Device on pin 1
ALIAS chip THIS      # The IC chip itself (db)
```

Reading Device Properties

```
VAR temp = sensor.Temperature
VAR pressure = sensor.Pressure
VAR isOn = heater.On
VAR charge = battery.Charge
```

Writing Device Properties

```
heater.On = 1           # Turn on
pump.Setting = 100      # Set target
display.Setting = temp  # Show value
door.Open = 0           # Close door
```

Slot Operations

Many devices have slots (inventories) you can access:

```
VAR hash = device.Slot(0).OccupantHash
VAR qty = device.Slot(0).Quantity
VAR occupied = device.Slot(0).Occupied
```

Named Device References

Bypass the 6-pin limit by referencing devices by their prefab name. This uses batch operations to find devices on the network:

```
DEVICE sensor "StructureGasSensor"
DEVICE furnace "StructureFurnace"

# Use like regular aliases
VAR temp = sensor.Temperature
furnace.On = 1
```

Batch Operations

Read from or write to ALL devices of a type at once:

```
DEFINE SENSOR_HASH -1234567890

# Batch read modes: 0=Average, 1=Sum, 2=Min, 3=Max
VAR avgTemp = BATCHREAD(SENSOR_HASH, Temperature, 0)
```

```
VAR totalPower = BATCHREAD(BATTERY_HASH, PowerGeneration, 1)

# Write to all devices
BATCHWRITE(LIGHT_HASH, On, 1)  # Turn on all lights
```

5. Built-in Functions

Math Functions

Function	Description	Example
ABS(x)	Absolute value	ABS(-5) = 5
SQRT(x)	Square root	SQRT(16) = 4
MIN(a,b)	Minimum value	MIN(5, 3) = 3
MAX(a,b)	Maximum value	MAX(5, 3) = 5
CEIL(x)	Round up	CEIL(3.2) = 4
FLOOR(x)	Round down	FLOOR(3.8) = 3
ROUND(x)	Round nearest	ROUND(3.5) = 4
TRUNC(x)	Truncate	TRUNC(3.9) = 3
SGN(x)	Sign (-1,0,1)	SGN(-5) = -1
RND()	Random 0-1	RND() = 0.xxx

Trigonometry (angles in radians)

Function	Description	Example
SIN(x)	Sine	SIN(0) = 0
COS(x)	Cosine	COS(0) = 1
TAN(x)	Tangent	TAN(0) = 0
ASIN(x)	Arc sine	ASIN(1) = 1.57
ACOS(x)	Arc cosine	ACOS(0) = 1.57
ATAN(x)	Arc tangent	ATAN(1) = 0.785
ATAN2(y,x)	2-arg arctangent	ATAN2(1, 1) = 0.785

Exponential & Logarithmic

Function	Description	Example
EXP(x)	e raised to x	EXP(1) = 2.718
LOG(x)	Natural logarithm	LOG(2.718) = 1

Control Functions

Function	Description	Example
YIELD	Pause 1 game tick	YIELD
SLEEP n	Pause n seconds	SLEEP 0.5
WAIT(n)	Same as SLEEP	WAIT(1)
END	Stop execution	IF error THEN END

Stack Operations

```
PUSH value      # Push to stack
POP variable    # Pop from stack
PEEK variable   # Read top without removing
```

6. IC10 MIPS Reference

This reference shows the IC10 assembly instructions that your BASIC code compiles to. Understanding these helps with debugging and optimization.

Registers

- r0-r15: General purpose registers (16 total)
- sp: Stack pointer
- ra: Return address (for subroutines)
- d0-d5: Device references
- db: IC housing device (self)

Mat

Instruction	Meaning	Description
add r0 r1 r2	r0 = r1 + r2	Addition
sub r0 r1 r2	r0 = r1 - r2	Subtraction
mul r0 r1 r2	r0 = r1 * r2	Multiplication
div r0 r1 r2	r0 = r1 / r2	Division
mod r0 r1 r2	r0 = r1 % r2	Modulo
sqrt r0 r1	r0 = sqrt(r1)	Square root
abs r0 r1	r0 = |r1|	Absolute value
round r0 r1	r0 = round(r1)	Round
floor r0 r1	r0 = floor(r1)	Round down
ceil r0 r1	r0 = ceil(r1)	Round up
min r0 r1 r2	r0 = min(r1,r2)	Minimum
max r0 r1 r2	r0 = max(r1,r2)	Maximum

Logic & Bitwise

Instruction	Meaning	Description
and r0 r1 r2	r0 = r1 & r2	Bitwise AND
or r0 r1 r2	r0 = r1 | r2	Bitwise OR
xor r0 r1 r2	r0 = r1 ^ r2	Bitwise XOR
nor r0 r1 r2	r0 = ~(r1|r2)	NOR
sll r0 r1 r2	r0 = r1 << r2	Shift left
srl r0 r1 r2	r0 = r1 >> r2	Shift right
sra r0 r1 r2	r0 = r1 >>> r2	Arithmetic shift

Comparison (Set Instructions)

Instruction	Meaning	Description
`slt r0 r1 r2`	`r0 = (r1 < r2)`	Set if less than
`sgt r0 r1 r2`	`r0 = (r1 > r2)`	Set if greater
`sle r0 r1 r2`	`r0 = (r1 <= r2)`	Set if less/equal
`sge r0 r1 r2`	`r0 = (r1 >= r2)`	Set if greater/equal
`seq r0 r1 r2`	`r0 = (r1 == r2)`	Set if equal
`sne r0 r1 r2`	`r0 = (r1 != r2)`	Set if not equal
`seqz r0 r1`	`r0 = (r1 == 0)`	Set if zero
`snez r0 r1`	`r0 = (r1 != 0)`	Set if not zero

Branching & Jumps

Instruction	Meaning	Description
`j label`	`goto label`	Unconditional jump
`jal label`	`call label`	Jump and link
`jr r0`	`goto r0`	Jump to register
`beq r0 r1 lbl`	`if r0==r1 goto`	Branch if equal
`bne r0 r1 lbl`	`if r0!=r1 goto`	Branch if not equal
`blt r0 r1 lbl`	`if r0<r1 goto`	Branch if less
`bgt r0 r1 lbl`	`if r0>r1 goto`	Branch if greater
`beqz r0 lbl`	`if r0==0 goto`	Branch if zero
`bnez r0 lbl`	`if r0!=0 goto`	Branch if not zero

Device Operations

Instruction	Meaning	Description
`l r0 d0 Prop`	`r0 = d0.Prop`	Load from device
`s d0 Prop r0`	`d0.Prop = r0`	Store to device
`ls r0 d0 s Prop`	`r0=d0.Slot(s).P`	Load slot prop
`lb r0 h Prop m`	`batch read`	Load batch
`sb h Prop r0`	`batch write`	Store batch

Special Instructions

Instruction	Meaning	Description
`move r0 r1`	`r0 = r1`	Copy value
`yield`	`pause 1 tick`	Yield execution
`sleep r0`	`pause r0 sec`	Sleep for time
`push r0`	`stack.push(r0)`	Push to stack
`pop r0`	`r0=stack.pop()`	Pop from stack
`hcf`	`halt`	Halt and catch fire

7. Example Programs

Thermostat with Hysteresis

Maintains temperature with dead-band to prevent rapid cycling:

```
ALIAS sensor d0
ALIAS heater d1

CONST TARGET = 293    # 20C in Kelvin
CONST TOLERANCE = 2

main:
    VAR temp = sensor.Temperature

    IF temp < TARGET - TOLERANCE THEN
        heater.On = 1
    ELSEIF temp > TARGET + TOLERANCE THEN
        heater.On = 0
    ENDIF

    YIELD
    GOTO main
END
```

Solar Panel Tracker

Automatically positions solar panels to track the sun:

```
ALIAS panel d0

main:
    VAR angle = panel.SolarAngle
    panel.Horizontal = angle
    panel.Vertical = 60

    YIELD
    GOTO main
END
```

Battery Monitor with Backup

Activates generator when battery is low:

```
ALIAS battery d0
ALIAS generator d1
ALIAS display d2

CONST LOW = 0.20
CONST HIGH = 0.90
```

```
main:
    VAR charge = battery.Charge

    IF charge < LOW THEN
        generator.On = 1
    ELSEIF charge > HIGH THEN
        generator.On = 0
    ENDIF

    display.Setting = charge * 100
    YIELD
    GOTO main
END
```

Counter with Compound Assignment

Demonstrates v1.9.0 compound operators:

```
ALIAS display d0
ALIAS button d1

VAR count = 0
VAR lastBtn = 0

main:
    VAR btn = button.Setting

    IF btn = 1 AND lastBtn = 0 THEN
        count += 1    # Compound assignment
    ENDIF

    lastBtn = btn
    display.Setting = count

    YIELD
    GOTO main
END
```

Bit Flags for Status Display

Uses bit shifts for compact status:

```
ALIAS sensor d0
ALIAS display d1

VAR status = 0

main:
    status = 0

    IF sensor.Power > 0 THEN
        status = status | (1 << 0)
    ENDIF
    IF sensor.Temperature > 250 THEN
        status = status | (1 << 1)
    ENDIF
    IF sensor.Pressure > 80 THEN
        status = status | (1 << 2)
    ENDIF

    display.Setting = status
    YIELD
    GOTO main
END
```

8. Tips & Best Practices

Always Use YIELD

Every loop must contain YIELD or SLEEP. Without it, the game will detect an infinite loop and halt your program.

```
# BAD - will crash
WHILE 1
    # no yield!
WEND

# GOOD
WHILE 1
    YIELD
WEND
```

Cache Device Reads

Each device read takes time. Read once and reuse:

```
# BAD - reads 3 times
IF sensor.Temperature > 100 THEN
ELSEIF sensor.Temperature > 50 THEN
ENDIF

# GOOD - reads once
VAR temp = sensor.Temperature
IF temp > 100 THEN
ELSEIF temp > 50 THEN
ENDIF
```

Use Hysteresis

Prevent rapid on/off switching by adding a dead-band around your target value:

```
CONST TARGET = 100
CONST TOLERANCE = 5

IF value < TARGET - TOLERANCE THEN
    device.On = 1
ELSEIF value > TARGET + TOLERANCE THEN
    device.On = 0
ENDIF
```

Use Constants for Magic Numbers

```
# BAD
IF temp > 373.15 THEN

# GOOD
CONST BOILING_POINT = 373.15
```

```
IF temp > BOILING_POINT THEN
```

Bit Shifts for Efficiency

Use bit shifts for power-of-2 multiplication/division:

```
x = x << 1    # Same as x * 2
x = x >> 2    # Same as x / 4

# Set/clear/check flags
flags = flags | (1 << n)      # Set bit n
flags = flags & ~(1 << n)     # Clear bit n
isSet = (flags >> n) & 1      # Check bit n
```

Appendix A: Common Device Properties

Universal Properties

Property	Type	R/W	Description
On	0/1	R/W	Power state
Setting	Number	R/W	Target/display value
Mode	Integer	R/W	Operating mode
Lock	0/1	R/W	Lock state
Error	0/1	R	Error state
Power	Watts	R	Power consumption
PrefabHash	Integer	R	Device type hash

Atmosphere Properties

Property	Type	R/W	Description
Temperature	Kelvin	R	Gas temperature
Pressure	kPa	R	Total pressure
RatioOxygen	0-1	R	O2 ratio
RatioCarbonDioxide	0-1	R	CO2 ratio
RatioNitrogen	0-1	R	N2 ratio
RatioVolatiles	0-1	R	H2 ratio
RatioWater	0-1	R	Steam ratio
TotalMoles	Moles	R	Total gas quantity

Power Properties

Property	Type	R/W	Description
Charge	0-1	R	Battery charge ratio
PowerGeneration	Watts	R	Power output
PowerRequired	Watts	R	Power demand
SolarAngle	Degrees	R	Sun angle
Horizontal	Degrees	R/W	Panel horizontal
Vertical	Degrees	R/W	Panel vertical

Appendix B: Color Constants

Built-in color constants for lights and displays:

Name	Value	RGB Hex	Usage
Blue	0	#0000FF	light.Color = Blue
Gray	1	#808080	light.Color = Gray
Green	2	#00FF00	light.Color = Green
Orange	3	#FFA500	light.Color = Orange
Red	4	#FF0000	light.Color = Red
Yellow	5	#FFFF00	light.Color = Yellow
White	6	#FFFFFF	light.Color = White
Black	7	#000000	light.Color = Black
Brown	8	#8B4513	light.Color = Brown
Khaki	9	#F0E68C	light.Color = Khaki
Pink	10	#FFC0CB	light.Color = Pink
Purple	11	#800080	light.Color = Purple

Custom RGB Colors

For custom colors, use decimal RGB values:

```
# RGB to decimal: R*65536 + G*256 + B
DEFINE RED 16711680       # FF0000
DEFINE GREEN 65280        # 00FF00
DEFINE BLUE 255           # 0000FF
DEFINE YELLOW 16776960    # FFFF00
DEFINE CYAN 65535         # 00FFFF
DEFINE MAGENTA 16711935   # FF00FF
DEFINE WHITE 16777215     # FFFFFF

light.Color = RED
```

Slot Type Constants

For slot operations:

Name	Value	Description
Import	0	Input slot (also: Input)
Export	1	Output slot (also: Output)
Content	2	Content/storage slot
Fuel	3	Fuel slot