

# HBC-2 Instruction Manual

## Summary

- I. Generalities
- II. Instruction format and addressing modes
- III. RAM and I/O interfaces
- IV. Interrupts
- V. Instruction set

## Generalities

The HomeBrew Computer 2 (or HBC-2) is a home made 8 bit CPU designed to understand the basics of computer science via a simplified central processing unit.

No chip implementing this CPU design exists to this day, but an FPGA implementation is possible, given enough work.

### Specifications

Data bus width	8 bits
Address bus width	16 bits
RAM maximum capacity	64 kB
Clock frequency	1-16 MHz
Instruction size	32 bits
Instruction set size	48
Architecture	Von Neumann / RISC
Stack size	256 bytes (no under/overflow flag)
I/O ports	256

# Instruction format and addressing modes

## Instructions

Operand (6b)	A.M. (4b)	Reg1 (3b)	Reg2 (3b)	V1 (8b)	V2 (8b)
--------------	-----------	-----------	-----------	---------	---------

**Instructions are 32 bits (or 4 bytes) long, divided in 4 sections :**

1. A 6 bits operand (64 instructions maximum, 48 implemented),
2. A 4 bits addressing mode (16 addressing modes maximum, 8 implemented),
3. Two 3 bits register selectors (8 registers, A, B, C, D, I, J, X, Y),
4. Two 8 bits user-defined values.

## Addressing modes (A.M.)

Many instructions can work in multiple addressing modes. The instruction set described at the end of this manual gives information about which addressing modes you can use for each instruction.

Reg	0x1
Reg / Imm8	0x2
Reg / Ram	0x3
RamReg / ImmReg	0x4
Reg16	0x5
Imm16	0x6
Imm8	0x7

### Reg

Two 8-bit values retrieved from registers 1 and 2.

Operand (6b)	0x1	Reg1 (3b)	Reg2 (3b)	0x00	0x00
--------------	-----	-----------	-----------	------	------

### Reg / Imm8

One 8-bit value retrieved from register 1 and one user-defined 8-bit value.

Operand (6b)	0x2	Reg1 (3b)	0x0	V1 (8b)	0x00
--------------	-----	-----------	-----	---------	------

### Reg / Ram

One user-defined 16-bit value and one 8-bit value retrieved from register 1.

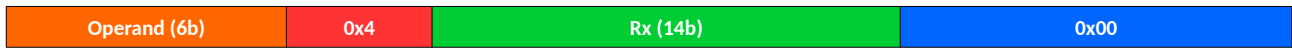
Operand (6b)	0x3	Reg1 (3b)	0x0	Vx (16b)
--------------	-----	-----------	-----	----------

## RamReg / ImmReg

Register 1 and 2 concatenated as a 16-bit address to retrieve one 8-bit value from memory.

Register 1 is the most significant byte (MSB).

The 8-bit value in register 3 is used directly.



## Reg16

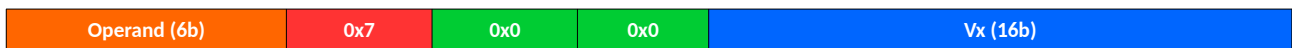
Register 1 and 2 are concatenated in a 16-bit value used directly.

Register 1 is the most significant byte (MSB).



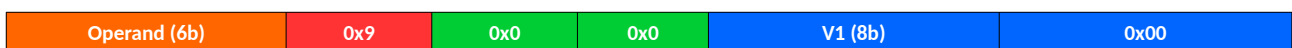
## Imm16

One user-defined 16-bit value.



## Imm8

One user-defined 8-bit value.



## RAM and I/O interfaces

The microprocessor uses two buses to access data from external devices, one for **data**, and one for the **address**. Those buses are used both to access RAM and I/O devices. Accessing I/O devices is done via the I/O Driver chip, **IOD**, which is specifically design for this CPU, and handles both I/O accesses and I/O interrupts.

Data bus width	8 bits
Address bus width (RAM)	16 bits (64 kB)
Address bus width (I/O)	8 bits (256 ports)

This means that when accessing **I/O ports**, the CPU only uses the **8 least significant bits** of the address bus. Whatever is on the 8 other bits is left unused. The port number is read by the IOD chip, which is triggered by the IE pin (I/O Enable).

### Control lines

RAM	RW (Read / Write) + RE (Ram Enable)
I/O access	RW (Read / Write) + IE (I/O Enable)

To access data from **RAM**, the CPU sets the R/W pin accordingly, and sets the **RE pin** to **HIGH**.

To access data from **I/O**, the CPU sets the R/W pin accordingly, and sets the **IE pin** to **HIGH**.

*When the RE pin is set to HIGH, the IE pin is always set to LOW, and vice versa.*

## Interrupts

To handle interrupts, the microprocessor implements two control lines, **INT** (pending interrupt) and **INR** (interrupt ready), that can be triggered by the CPU and the IOD chip.

### CPU side

The CPU is not necessarily ready to handle interrupts. An **interrupt flag** is implemented in the flag register. When this flag is **set**, the CPU is capable of handling interrupts. Two instructions control this behaviour : **STI** (Set Interrupt flag), and **CLI** (Clear Interrupt flag).

When the CPU has received an interrupt, and is ready to manage it, it retrieves the I/O port number via the IOD chip, and then jumps to a memory location defined by the programmer in the **interrupt vector table (IVT)**.

The interrupt vector table is a **512 bytes** memory block in the RAM, located from address **0x0100** to address **0x02FF**. Those address holds pointers that the CPU uses to set the **PC** (Program Counter register). As a reminder, RAM addresses are 2 bytes long, and there are 256 I/O ports that are numbered from 0x00 to 0xFF.

I/O port number	Address in the IVT
0x00	0x0100 << 0x0101
0x01	0x0102 << 0x0103
[...]	[...]
0xFE	0x02FC << 0x02FD
0xFF	0x02FE << 0x02FF

**Procedure :**

1. The IOD chip sets the **INT** control line to **HIGH**,
2. If the interrupt flag is set to **HIGH**, the CPU finishes the current instruction and sets the **INR** control line to **HIGH** for one clock cycle,
3. The IOD chip then sets **INT** control line to **LOW**, and sends the I/O port number on the **address bus** for **one clock cycle**, as well as the data from the device on the **data bus**,
4. The CPU then reads the I/O port number on the address bus, pushes the **I register**, pushes the **PC register** to the **stack**, stores the I/O data in the **I register**, sets the **interrupt flag** to **LOW**, sets the **INR** control line to **LOW**, and then jumps to the memory location pointed by the **interrupt vector table**.
5. On the next clock cycle, if the interrupts queue is not empty, the IOD chip set the **INT** control line back to **HIGH** until the CPU is ready to manage it (step 2).
6. When the CPU has finished processing the interrupt (by hitting the **IRET** instruction), it pops the PC register out of the stack to get back to work on the code that was interrupted, and pops the **I register**. If **INT** pin is **HIGH**, it manages the next interrupt immediately.

## IOD chip side

The CPU does not always have enough time to process an interrupt before the next one is triggered. Therefore, the IOD chip implements a 256 interrupts queue via two 256 bytes stacks, one for I/O port numbers and one for data sent by devices.

If an interrupt is triggered but the previous one was not processed by the CPU yet, then the I/O port number is stacked in the queue, and the IOD chip will keep the **INT** control line **HIGH** until the CPU is ready to manage it.

*If the queue is full, any new interrupt will be discarded.*



## Software interrupts

The CPU implements the **INT** instruction, that triggers a software interrupt on the I/O port number defined by the programmer in the **V1** value of the instruction.

This kind of interrupt does not end up in a dialog between the CPU and IOD chip. Also, they have no priority on hardware interrupts.

***Register I** will be pushed onto the stack, and the CPU will handle this interrupt just a regular hardware interrupt. Therefore, instruction **INT** sets the data bus with the content of register I.*

## Instruction set

Abbreviation	Operand	A.M.	Description	Flags
<b>NOP</b>	0x00		No operation	
<b>ADC</b>	0x01	<i>Reg</i>	$R1 = R1 + R2 + 1$	<b>C N Z</b>
		<i>Reg / Imm8</i>	$R1 = R1 + V1 + 1$	
		<i>Reg / Ram</i>	$R1 = R1 + \$(Vx) + 1$	
<b>ADD</b>	0x02	<i>Reg</i>	$R1 = R1 + R2$	
		<i>Reg / Imm8</i>	$R1 = R1 + V1$	
		<i>Reg / Ram</i>	$R1 = R1 + \$(Vx)$	
<b>AND</b>	0x03	<i>Reg</i>	$R1 = R1 \& R2$	<b>Z N</b>
		<i>Reg / Imm8</i>	$R1 = R1 \& V1$	
		<i>Reg / Ram</i>	$R1 = R1 \& \$(Vx)$	
<b>CAL</b>	0x04	<i>Reg16</i>	$PC = R1 \ll R2$	
		<i>Imm16</i>	$PC = Vx$	
<b>CLC</b>	0x05		Clears the Carry flag	<b>C</b>
<b>CLE</b>	0x06		Clears the Equal flag	<b>E</b>
<b>CLI</b>	0x07		Clears the Interrupt flag	<b>I</b>
<b>CLN</b>	0x08		Clears the Negative flag	<b>N</b>
<b>CLS</b>	0x09		Clears the Superior flag	<b>S</b>
<b>CLZ</b>	0x0A		Clears the Zero flag	<b>Z</b>
<b>CLF</b>	0x0B		Clears the Inferior flag	<b>F</b>
<b>CMP</b>	0x0C	<i>Reg</i>	$FR = R1 ? R2$	<b>S E F</b>
		<i>Reg / Imm8</i>	$FR = R1 ? V1$	
		<i>RamReg / ImmReg</i>	$FR = R3 ? \$(R1 \ll R2)$	

## HBC-2 Instruction Manual

Abbreviation	Operand	A.M.	Description	Flags
<b>DEC</b>	0x0D	<i>Reg</i>	$R1 = R1 - 1$	<b>C N Z</b>
		<i>Reg16</i>	$\$(R1 \ll R2) = \$(R1 \ll R2) - 1$	
		<i>Imm16</i>	$\$(Vx) = \$(Vx) - 1$	
<b>HLT</b>	0x0E		Halts the CPU until next interrupt	<b>H I</b>
<b>IN</b>	0x0F	<i>Reg</i>	Data bus = R1 Address bus = R2 IOD Receive pin = HIGH	
<b>OUT</b>	0x10		Data bus = R2 Address bus = R1 IOD Send pin = HIGH	
<b>INC</b>	0x11	<i>Reg</i>	$R1 = R1 + 1$	<b>C N Z</b>
		<i>Reg16</i>	$\$(R1 \ll R2) = \$(R1 \ll R2) + 1$	
		<i>Imm16</i>	$\$(Vx) = \$(Vx) + 1$	
<b>INT</b>	0x12	<i>Imm8</i>	Software interrupt with port number V1	
<b>IRT</b>	0x13		PC_MSB = $\$(STK)$ , STK = STK - 1 PC_LSB = $\$(STK)$ , STK = STK - 1 Sets Interrupt flag	<b>I</b>
<b>JMC</b>	0x14	<i>Reg16</i>	PC = $\$(R1 \ll R2)$ if Carry flag	
		<i>Imm16</i>	PC = $\$(Vx)$ if Carry flag	
<b>JME</b>	0x15	<i>Reg16</i>	PC = $\$(R1 \ll R2)$ if Equality flag	
		<i>Imm16</i>	PC = $\$(Vx)$ if Equality flag	
<b>JMN</b>	0x16	<i>Reg16</i>	PC = $\$(R1 \ll R2)$ if Negative flag	
		<i>Imm16</i>	PC = $\$(Vx)$ if Negative flag	
<b>JMP</b>	0x17	<i>Reg16</i>	PC = $\$(R1 \ll R2)$	
		<i>Imm16</i>	PC = $\$(Vx)$	
<b>JMS</b>	0x18	<i>Reg16</i>	PC = $\$(R1 \ll R2)$ if Superior flag	
		<i>Imm16</i>	PC = $\$(Vx)$ if Superior flag	
<b>JMZ</b>	0x19	<i>Reg16</i>	PC = $\$(R1 \ll R2)$ if Zero flag	
		<i>Imm16</i>	PC = $\$(Vx)$ if Zero flag	

## HBC-2 Instruction Manual

Abbreviation	Operand	A.M.	Description	Flags
<b>JMF</b>	0x1A	<i>Reg16</i>	PC = \$(R1 << R2) if Inferior flag	
		<i>Imm16</i>	PC = \$(Vx) if Inferior flag	
<b>STR</b>	0x1B	<i>RamReg / ImmReg</i>	\$(R1 << R2) = R3	
		<i>Reg / Ram</i>	\$(Vx) = R1	
<b>LOD</b>	0x1C	<i>RamReg / ImmReg</i>	R3 = \$(R1 << R2)	
		<i>Reg / Ram</i>	R1 = \$(Vx)	
<b>MOV</b>	0x1D	<i>Reg</i>	R1 = R2	
		<i>Reg / Imm8</i>	R1 = V1	
<b>NOT</b>	0x1E	<i>Reg</i>	R1 = ! R1	
		<i>Imm16</i>	\$(Vx) = ! \$(Vx)	
<b>OR</b>	0x1F	<i>Reg</i>	R1 = R1    R2	<b>Z N</b>
		<i>Reg / Imm8</i>	R1 = R1    V1	
		<i>Reg / Ram</i>	R1 = R1    \$(Vx)	
<b>POP</b>	0x20	<i>Reg</i>	R1 = \$(STK) STK = STK - 1	
<b>PSH</b>	0x21		STK = STK + 1 \$(STK) = R1	
<b>RET</b>	0x22		PC_MSB = \$(STK), STK = STK - 1 PC_LSB = \$(STK), STK = STK - 1	
<b>SHL</b>	0x23	<i>Reg</i>	Left shift on R1	<b>C Z N</b>
<b>ASR</b>	0x24		Arithmetical shift on R1	
<b>SHR</b>	0x25		Right shift on R1	<b>Z N</b>
<b>STC</b>	0x26		Sets the Carry flag	<b>C</b>
<b>STE</b>	0x27		Sets the Equal flag	<b>E</b>
<b>STI</b>	0x28		Sets the Interrupt flag	<b>I</b>
<b>STN</b>	0x29		Sets the Negative flag	<b>N</b>
<b>STS</b>	0x2A		Sets the Superior flag	<b>S</b>
<b>STZ</b>	0x2B		Sets the Zero flag	<b>Z</b>
<b>STF</b>	0x2C		Sets the Inferior flag	<b>F</b>

## HBC-2 Instruction Manual

Abbreviation	Operand	A.M.	Description	Flags
<b>SUB</b>	0x2D	<i>Reg</i>	$R1 = R1 - R2$	<b>C N Z</b>
		<i>Reg / Imm8</i>	$R1 = R1 - V1$	
		<i>Reg / Ram</i>	$R1 = R1 - \$ (Vx)$	
<b>SBB</b>	0x2E	<i>Reg</i>	$R1 = R1 - R2 - 1$	
		<i>Reg / Imm8</i>	$R1 = R1 - V1 - 1$	
		<i>Reg / Ram</i>	$R1 = R1 - \$ (Vx)$	
<b>XOR</b>	0x2F	<i>Reg</i>	$R1 = R1 \wedge R2$	<b>Z N</b>
		<i>Reg / Imm8</i>	$R1 = R1 \wedge V1$	
		<i>Reg / Ram</i>	$R1 = R1 \wedge \$ (Vx)$	