HBC-2 Instruction Manual

Summary

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.

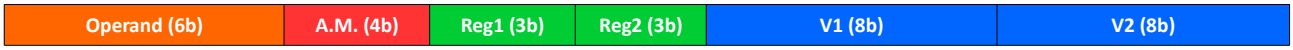
Specifications

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
I/O ports	256

Instruction format

and addressing modes

Instructions



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

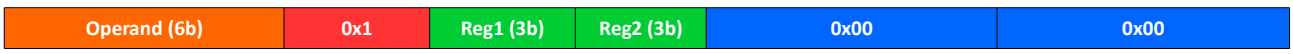
1. A 6 bits operand (64 instructions maximum, 47 implemented),
2. A 4 bits addressing mode (16 addressing modes maximum, 8 implemented),
3. Two 3 bits register selectors (8 registers, A, B, C, D, I, J, X, Y),
4. Two 8 bits user-defined values.

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



Reg / Imm8

Reg / Ram

RamReg / ImmReg

Reg16

Imm16

Imm8

RAM and I/O interfaces

The microprocessor uses two buses to access data from outside, 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 access 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 **first 8 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 trigerred by the IE pin (I/O Enable).

Control lines

RAM	R/W *(Read / Write)* + RE *(Ram Enable)*
I/O access	R/W *(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 an interrupt pin, **INT**, that can be trigerred by 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 **UP**, 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)**.

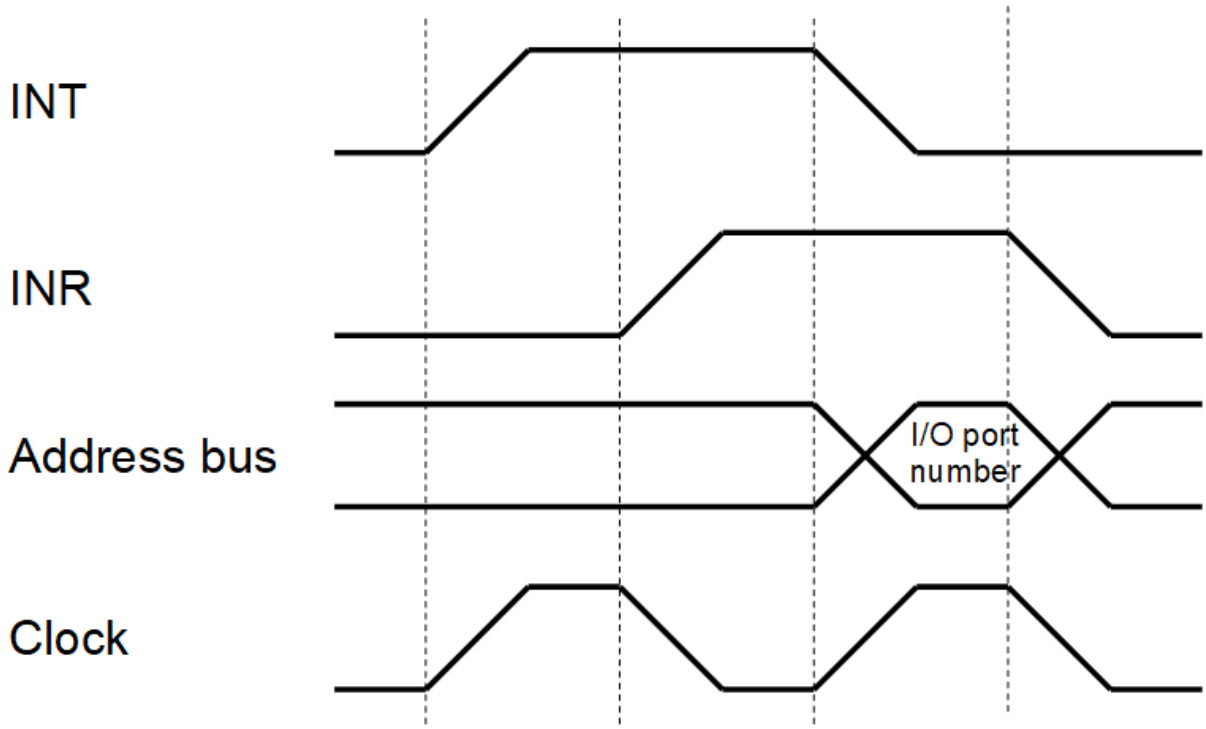
The interrupt vector table is a **512 bytes** memory block in the RAM, located from address **0x0100** to address **0x02FF**. Those address holds pointers that the CPU uses to set the **PC** (Program Counter register). As a reminder, RAM addresses are 2 bytes long, and there are 256 I/O ports that are 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** pin to **HIGH**,
2. If the interrupt flag is set to **HIGH**, the CPU finishes the current instruction and sets the **INR** pin to **HIGH** for one clock cycle,
3. The IOD chip then sets **INT** pin 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** pin 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** pin 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.

Time diagram :



IOD chip side

The CPU does not always have enough time to process an interrupt before the next one is trigerred. 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 trigerred 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** pin **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, and register I is not pushed on the stack (because the programmer can use that register to specify data related to the interrupt).

Instruction set

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

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

Abreviation	Operand	A.M.	Description	Flags
JMF	0x1A	*Reg16*	PC = $(R1 << R2) if Inferior flag	

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

Abreviation	Operand	A.M.	Description	Flags
SUB	0x2D	*Reg*	R1 = R1 - R2	**C N Z**
		Reg / Imm8	R1 = R1 - V1	
		Reg / Ram	R1 = R1 - $(Vx)	
SBB	0x2E	*Reg*	R1 = R1 - R2 - 1	
		Reg / Imm8	R1 = R1 - V1 - 1	
		Reg / Ram	R1 = R1 - $(Vx)	
XOR	0x2F	*Reg*	R1 = R1 ^ R2	**Z N**
		Reg / Imm8	R1 = R1 ^ V1	
		Reg / Ram	R1 = R1 ^ $(Vx)	