**Assembler code manual**

For the One Square Inch TTL CPU

https://hackaday.io/project/161251-1-square-inch-ttl-cpu

R.J.H. 20181020

This is a von Neumann architecture that can access 64Kbytes of memory. The memory is organized as bytes. When 16-bit words are stored, the LSB is at an even address and the MSB is at the following uneven address.

I/O locations should be memory mapped.

The CPU relies on microcode to execute its instructions. This document assumes that the microcode version of oct 19th, 2018 is in the processor. The microcode itself is not discussed here.

**GENERAL INSTRUCTION INFORMATION**

**REGISTERS**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Size** | **Address** |
| A | Accumulator | 16 bits | 0x0004 |
| PC | Program counter | 16 bits | 0x0002 |
| SP | Stack pointer | 8 bits | 0x0000 |
|  | Internal use | 8 bits | 0x0001 |
|  | Internal use | 16 bits | 0x0006 |

- The registers are stored in the RAM at the indicated address

**RESERVED MEMORY LOCATIONS**

RAM locations 0x0000-0x00FF can be accessed with Zero-page addressing

RAM locations 0x0100 – 0x1FF contain the stack.

RAM locations 0x0200 – 0x2FF are used for the compare instruction.

After reset, the processor starts at address 0x8012 so it is useful to have ROM locations there. The following locations in ROM must contain fixed values:

|  |  |  |
| --- | --- | --- |
| **Address** | **Value** |  |
| 0x8000 | 0x10 |  |
| 0x8001 | 0x80 |  |
| 0x8002 | 0x00 |  |
| 0x8003 | 0x20 |  |
| 0x8100-0x81FF | 0x02-0x01 | Table of increment-by-two values |
| 0x8200-0x82FF | 0xFE-0xFD | Table of decrement-by-two values |

**ADDRESSING MODES**

There are four addressing modes

* Immediate
* Direct addressing (Zero page mode)
* Indirect
* Implied

**MOVE INSTRUCTIONS**

**LDW** Load Word

|  |  |  |  |
| --- | --- | --- | --- |
| **LDW** | **1st byte** | **2nd byte** | **example** |
| Zero page | 0x08 | Z page location | LDW A,var |
| Indirect | 0x60 | Z page pointer | LDW A,(var) |

**LDB** Load Byte

|  |  |  |  |
| --- | --- | --- | --- |
| **LDB** | **1st byte** | **2nd byte** | **example** |
| Immediate | 0x30 | Immediate byte | LDB AL,#0x78 |
| Immediate | 0x2C | Immediate byte | LDB AH,#0x78 |
| Indirect | 0x40 | Z page pointer | LDB A,(var) |

Immediate loading of the lower half of A will clear the upper half.

**STW** Store Word

|  |  |  |  |
| --- | --- | --- | --- |
| **STW** | **1st byte** | **2nd byte** | **example** |
| Zero page | 0x38 | Z page location | STW var,A |
| Indirect | 0x70 | Z page pointer | STW (var),A |

**STB** Store Byte

|  |  |  |  |
| --- | --- | --- | --- |
| **STB** | **1st byte** | **2nd byte** | **example** |
| Indirect | 0x50 | Z page pointer | STB (var),A |

**PUSH** Push

|  |  |  |  |
| --- | --- | --- | --- |
| **PUSH** | **1st byte** | **2nd byte** | **example** |
| Zero page | 0xC0 | Z page location | PUSH var |

Pushes the 16-bit contents of the Z-page location to stack. Before use, the stack must be initialized to an even value.

**POP** Pop

|  |  |  |  |
| --- | --- | --- | --- |
| **POP** | **1st byte** | **2nd byte** | **example** |
| Indirect | 0xD0 | Z page location | POP var |

**ARITHMETIC INSTRUCTIONS**

**INCD** Increment Double

|  |  |  |  |
| --- | --- | --- | --- |
| **INCD** | **1st byte** | **2nd byte** | **example** |
| Implied | 0x19 | 0x81 | INCD A |

Adds 2 to the contents of A. It does only effect the lower byte of A. The instruction works with a table at address 0x8100-0x81FF in ROM. The second byte of the instruction is the MSB of the table location. By using another second byte, the instruction can work with another table.

The used table at 0x8100 is also used by the microcode to increment the PC and the stack pointer.

**DECD** Decrement Double

|  |  |  |  |
| --- | --- | --- | --- |
| **DECD** | **1st byte** | **2nd byte** | **Example** |
| Implied | 0x19 | 0x82 | DECD A |

Subtracts 2 from the contents of A. It does only effect the lower byte of A.

The used table at 0x8200 is also used by the microcode to decrement the stack pointer.

**CMPB** Compare Byte

|  |  |  |  |
| --- | --- | --- | --- |
| **CMPB** | **1st byte** | **2nd byte** | **example** |
| Zero page | 0xE0 | Z page location | CMPB A,var |

Compares the lowest byte of A with the byte in the zero page location.The result of the compare is put in A:

* A = 0x00 if the bytes were equal
* A = 0x80 if the bytes were not equal

**PROGRAM FLOW INSTRUCTIONS**

All instructions have a length of two bytes (except CALL, that is four bytes). Before fetching an instruction, the PC is incremented. Only the lowest byte of the PC is incremented. Jumping to another 256-byte page can be done with the PAGE instruction. CALL and RET can also cross page boundaries. It is also possible to load the PC (on zero page location 2) with a 16-bit value representing the jump address.

**BR** Branch (Jump to same-page location)

|  |  |  |  |
| --- | --- | --- | --- |
| **BR** | **1st byte** | **2nd byte** | **example** |
| Implied | 0x9C | New PCL value | BR label |

**BRP** Branch if positive

|  |  |  |  |
| --- | --- | --- | --- |
| **BR** | **1st byte** | **2nd byte** | **example** |
| Implied | 0xA0 | New PCL value | BRP label |

The BRP jump is only done if bit7 of the A register is 0

**BRM** Branch if minus

|  |  |  |  |
| --- | --- | --- | --- |
| **BR** | **1st byte** | **2nd byte** | **example** |
| Implied | 0x90 | New PCL value | BRM label |

The BRM jump is only done if bit7 of the A register is 1

**BEQ** Branch if equal

|  |  |  |  |
| --- | --- | --- | --- |
| **BR** | **1st byte** | **2nd byte** | **example** |
| Implied | 0xA0 | New PCL value | BEQ label |

The BEQ is convenient to use after a CMPB instruction. The jump is done when both bytes were equal.

**BNE** Branch if not equal

|  |  |  |  |
| --- | --- | --- | --- |
| **BR** | **1st byte** | **2nd byte** | **example** |
| Implied | 0x90 | New PCL value | BNE label |

The BNE is convenient to use after a CMPB instruction. The jump is done when both bytes were not equal.

**PAGE** Jump to another page

|  |  |  |  |
| --- | --- | --- | --- |
| **PAGE** | **1st byte** | **2nd byte** | **example** |
| Implied | 0xC8 | New PCH value | PAGE 0x84 |

The PAGE instruction also sets LSB of the PC to 0. In the current version this has the effect that next executed instruction is at 0x8402 because the PC is incremented before an instruction is fetched.

**CALL** Call subroutine

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CALL** | **1st byte** | **2nd byte** | **3rd byte** | **4th byte** | **example** |
| Immediate | 0xB0 | unused | New PCL value | New PCH value | CALL subr\_name |

The 16-bit return address is pushed on the stack, and the program continues at the subroutine address. Before use, the stack must be initialized to an even value.

**RET** Return from subroutine

|  |  |  |  |
| --- | --- | --- | --- |
| **RET** | **1st byte** | **2nd byte** | **example** |
| Implied | 0xD0 | 0x02 | RET |

The program counter is popped from the stack.

**ASSEMBLER DIRECTIVES**

**DB** Define byte

The specified byte is placed in the code.

**DW** Define word

The specified word is placed in the code

**DS** Define storage

The specified number of bytes is reserved in the code

**EQU** Define label

The label at the beginning of the line gets the value that is specified after “EQU”. Note that there must be a “:”directly after the label. It is used to define constant values.

**ORG** Define origin

The assembly continues at the specified address

**TABLE** Define a 256-byte table

This is used with the values INCD and DECD. This directive can be used to generate the required tables at address 0x8100 and 0x8200

Note that the assembler will automatically align to even locations, except for a DB directive.

--- end ---