|  |
| --- |
|  |
| T-16 USER MANUAL |
| with language specification |

|  |
| --- |
| Written by Peter Pretorius  210543717 |
|  |

Overview

The T-machine is a virtual machine with a 16bit address bus and 64K of memory. It is a Von Neumann machine (i.e. program and data both stored in memory) with a limited instruction set.

**Registers:**

Accumulator – 16bit general purpose register

Index register – 16bit general purpose register, designed specifically for tracking indices

**Flags:**

Z - set when an operation gives a zero result

P – set when an operation gives a non-negative result

**Stack pointer:**

A single 16 bit stack pointer is available for stack operations

Language specification

T language makes use of a number of directives and operations. The full specification is as follows:

**Table of Directives**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LABEL | MNEMONIC | ADDRESS | ACTION | ADDITIONAL INFORMATION |
| Not used | *BEG* | Not used | Mark the beginning of the code | Specifies where the data segment ends and the code segment begins. |
| Not used | *END* | Not used | Mark the end of the code | Must be the final entry in any T program. |
| Not used | *ORG* | Address | Specify ‘address’ in memory where the code following is to be loaded | Must be the very first entry in any T program. The address must be in hex. |
| Optional | *DC* | Value | Set aside one (optionally labelled) byte, set initially to the value chosen | Must be declared in the data segment. |
| Optional | *DS* | Length | Set aside ‘length’ bytes, with optional label associated with the first byte | Must be declared in the data segment. |
| Identifier | *EQU* | Value | Set ‘identifier’ as a synonym for the given value | Must be declared in the data segment. |

Apart from END, all other directives must appear in the data segment of the program (Before the code segment). The end of the data segment is marked by BEG. Any directives in the incorrect place will cause compilation errors.

Table of operations

|  |  |  |
| --- | --- | --- |
| MNEMONIC | OPCODE(HEX) | FUNCTION |
| *NOP* | 0000 | No operation |
| *CLA* | 0001 | Clear accumulator |
| *CLX* | 0002 | Clear index register |
| *INC\** | 0003 | Increment accumulator by one |
| *DEC\** | 0004 | Decrement accumulator by one |
| *INX\** | 0005 | Increment index register by one |
| *DEX* | 0006 | Decrement index register by one |
| *TAX* | 0007 | Transfer accumulator to index register |
| *INI\** | 0008 | Load accumulator with integer typed at keyboard |
| *INA\** | 0009 | Load accumulator with ASCII value typed at keyboard |
| *OTI* | 000A | Print accumulator (interpreted as integer) to screen |
| *OTA* | 000B | Print accumulator (interpreted as ASCII value) to screen1 |
| *PSH* | 000C | Push accumulator onto stack |
| *POP\** | 000D | Pop stack into accumulator |
| *RET* | 000E | Return from subroutine (return address popped from stack) |
| *HLT* | 000F | Halt program execution |
| *LDA B\** | 0010 | Load accumulator directly with contents of location whose address is given as B |
| *LDX B\** | 0011 | Load accumulator with contents of location whose address is given as B, indexed by contents of X register2 |
| *LDI B\** | 0012 | Load accumulator with the immediate value B |
| *STA B* | 0013 | Store accumulator on the location whose address is given as B |
| *STX B* | 0014 | Store accumulator on the location whose address is given as B, indexed by contents of X register2 |
| *ADD B\** | 0015 | Add to accumulator the contents of location whose address is given as B |
| *ADX B\** | 0016 | Add to accumulator the contents of location whose address is given as B, indexed by contents of X register2 |
| *ADI B\** | 0017 | Add the immediate value B to the accumulator |
| *SUB B\** | 0018 | Subtract from accumulator the contents of location whose address is given as B |
| *SBX B\** | 0019 | Subtract from accumulator the contents of location whose address is given as B, indexed by contents of X register2 |
| *SBI B\** | 001A | Subtract the immediate value B from the accumulator |
| *CMP B\** | 001B | Compare the accumulator with the contents of the location whose address is given as B3 |
| *CPX B\** | 001C | Compare the accumulator with the contents of the location whose address is given as B, indexed by contents of X register2, 3 |
| *CPI B\** | 001D | Compare the accumulator directly with the value B3 |
| *LSP B* | 001E | Load stack pointer with contents of location whose address is given as B |
| *LSI B* | 001F | Load stack pointer immediately with the value B |
| *BRN B* | 0020 | Branch to the address given as B |
| *BZE B* | 0021 | Branch to the address given as B if the Z flag condition is set |
| *BNZ B* | 0022 | Branch to the address given as B if the Z flag condition is unset |
| *BPZ B* | 0023 | Branch to the address given as B if the P flag condition is set |
| *BNG B* | 0024 | Branch to the address given as B if the P flag condition is unset |
| *JSR B* | 0025 | Call subroutine whose address is given as B, pushing return address onto the stack |

1. If accumulator is greater than maximum ASCII value, a newline will be forced (on machine with reduced character sets, for example)
2. That is, an address computed as the value of B+X
3. Comparison is done by virtual subtraction of operand from accumulator and setting flags

Operations marked with an asterisk affect the condition flags

Labels may be used to mark sections of code and are used in the declaration of subroutines. The rules for labels are that they may not contain any ASCII characters except the underscore (\_) and cannot contain more than 8 significant characters. Labels must end with a colon (:).

tIDE



tIDE is an IDE for t-language written by Peter Pretorius. It integrates the assembler, disassembler and VM into one all-inclusive execution platform. The editor supports basic syntax highlighting, error highlighting and verbose error output for easier debugging.