# T-33 User Manual

## Content Page

# Introduction

---

**The T-33 is an experimental processor design which was created for personal, educational purposes and is not intended for any form of real-world application. Feel free to use the designs anywhere and in any way you like as long as you do not claim it as your own work.**

**The T-33 architecture is aimed at a simplistic representation of a microprocessor for ease of understanding and use and thus is missing some critical components. It takes data and instructions as parallel inputs. Instructions are sent straight to the components themselves for decoding/enabling of certain functions. Data is written directly to the accumulator. The processor contains six registers in total, three of which are dedicated for storage and three that are used for other purposes. The registers connected to the ALU (Accumulator and Separated register) can interact with each other through the 'SWP' function and the storage registers cannot interact with the system unless enabled. The 'SR' register is not directly writeable. The processor is not meant to have an output but it is implemented through the 'READ' function for use with debugging and presentation of results.**

**There are however flaws with the design that should be taken into consideration:**

- **Lack of Signed bit (Two's complement). This means that the total possible numbers that can be represented are from 0-255 and only positive integers can be output of a subtraction.**
- **No 'right-shift' as a specified ALU function. The disadvantage of this would be the inability of programming division algorithms.**
- **No bus to connect to/address peripherals**

# Component Overview

---

**The following components are present in the T-33:**

### ACC (Accumulator):

**Register that accumulates data. Receives input from 'Data In', the main registers (1, 2 and 3) and 'SWP'. The ACC outputs data to the ALU and the SWP. The ACC will maintain its data until erased using the 'Clear ACC' input (Refer to table 1.0) or swapped using the 'SWP' function. If any additional data is met with the data already contained in the ACC it performs a logical OR function and saves it to the ACC.**

### SR (Separated Register):

**The register that is only accessible via the 'SWP' function. Takes SWP as an input and outputs to the ALU and SWP.**

### INS Reg (Instruction Register):

**The register that holds the instruction.**

### Registers 1, 2 and 3:

**General purpose registers used for holding unsigned binary integers. Their I/O is controlled via instruction which is then decoded to multiplex/enable the preferred register. Takes an enable-only input from the ALU and outputs enable-only to the ACC. When a register is read, the data is also sent to the raw output.**

### SWP (Swapper):

**The part of the processor that swaps the states of the ACC and SR registers. The SWP module contains two hidden registers which store the data after every clock in inverse positions. When the SWP is triggered, the ALU registers are cleared, then rewritten with the SWP's contents. This is the only possible way to communicate with the SR register.**

### ALU (Arithmetic Logic Unit):

**The combinational logic circuit that provides access to binary arithmetic functions. The ALU has three possible functions and one post-process (Listed Consecutively): add, subtract, logical OR and binary shift left. The three functions are decoded whereas the post-process has its own bit as an input. The ALU receives data from the ACC and SR and outputs to the main registers (1, 2 and 3).**

# Instruction and Data Input Table

---

**Bits are based on the inputs found in the simulation.**

## DATA INPUT:

Table 1.0

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | Clear ACC |
|---|---|---|---|---|---|---|---|---|
| Bit 0 (Far left) | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 | Bit 7 | Bit 8 (Far right) |

**INPUT**

**V**

## INSTRUCTION INPUT:

Table 1.1

| SWP | ALU Function | Register Read | Register Write |
|---|---|---|---|
| Bit 0 (Far left) | Bits 1, 2 and 3 | Bits 4 and 5 | Bits 6 and 7 (Far Right) |

**INPUT**

**V**

# Command List

Table 2.0

| Command | Equivalent Microcode (Also refer to Table 1.1) | Description |
|---|---|---|
| SWP | 10000000 | Switches the data stored in the ACC and SR. When referenced, the data stored in the ACC will be written to the SR and the data originally stored in the SR will be written to the ACC. (ACC = X, SR = Y > SWP > ACC = Y, SR = X) |
| ADD | 00000000 | ACC + SR. At default, the ALU is set to add. Numbers that can be calculated are: 0-255 |
| SUB | 00100000 | ACC - SR. When referenced, ALU subtracts the value of the SR from the ACC. Only works with positive integers. Numbers that can be calculated are: 0-255 |
| OR | 01000000 | ACC OR SR. Performs a logical OR function on the ACC and SR. |
| SHFT | 00010000 | Binary shift-left. Moves all bits one place to the left with no wrap-around/rotation. (Ie. * 2) |
| READ | 00000100 - 1<br>00001000 - 2<br>00001100 - 3 | Enables the read of a specified register (1, 2 or 3). READ writes the value of the selected register to the ACC and also sends it to the output. When the T-33 is halted during a READ instruction, the number stored in the register is displayed on the output LEDs. |
| WRITE | 00000001 - 1<br>00000010 - 2<br>00000011 - 3 | Enables the writing of a specified register (1, 2 or 3). WRITE will replace the value of the selected register with the result of the ALU. |
| NULL | 00000000 | Will perform no operation. Not paired with any other command. |

# Command Rules and Syntax

---

## Rules and Recommendations:

- **Never add a number over 255 or subtract to less than 0. This will cause an overflow/underflow and thus provide an incorrect calculation.**
- **Do not use SWP with any other command or input data**
- **01100000 is not a specified command for the ALU and will do nothing.**
- **ALU functions should be followed up with WRITE. If not, nothing will happen.**

## Syntax:

**WRITE must always come after an ALU function (ADD, SUB, OR, SHFT).**
**(Eg. ADD WRITE2).**
**WRITE can be paired with READ for a somewhat equivalent to 'MOV' in assembly where it will copy the value of one storage register to another storage register.**
**SHFT can be combined with any base arithmetic function (ADD, SUB, OR)**
**(Eg. SUB SHFT WRITE1)**
**SWP is standalone command because it will cause errors when paired with other commands and input data.**

**Examples of acceptable syntaxes:**
**<Arithmetic> <READ#> <WRITE#>**
**Or**
**<Data> <Arithmetic> <WRITE#>**
**Or**
**<SWP>**
**Or**
**<NULL>**

## Decoding instructions:

**Take note that an instruction decoder is non-existent, these commands are used to make it easier to plan out code, therefore you must decode your own instructions. Put simply, this involves logically ORing the microcode of your commands.**
**Example:**
**SUB SHFT WRITE1 = 00100000 00010000 00000001**
**Therefore your microcode for this line will be:**
**00100000 OR 00010000 OR 00000001 = 00110001**

# Using the Circuit

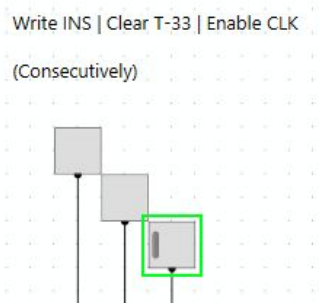
---

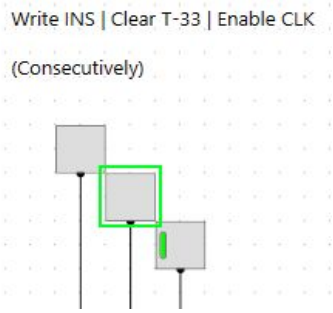
**The T-33 is programmed through Logic Circuit, a graphical digital logic editor found at: [http://www.logiccircuit.org/](http://www.logiccircuit.org/)**

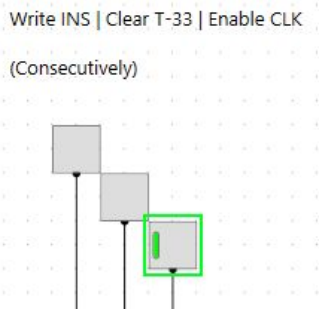
## Clearing and resetting everything (Can be skipped initially).

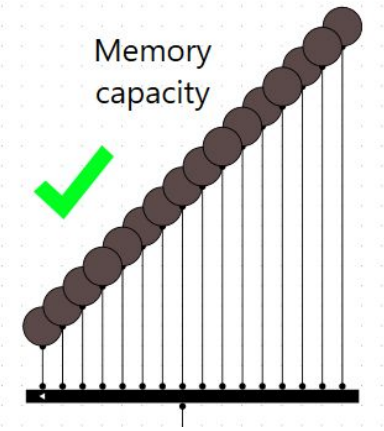
1. **Enable the clock for at least 16 cycles by clicking the 'Enable CLK' button.**
2. **Press the 'Clear T-33' button.**
3. **Disable the clock by clicking the 'Enable CLK' button.**

**Given that none of the LEDs on 'Memory Capacity' are showing, effective simulation can be achieved.**

1. 


2. 


3. 

# Using the Circuit

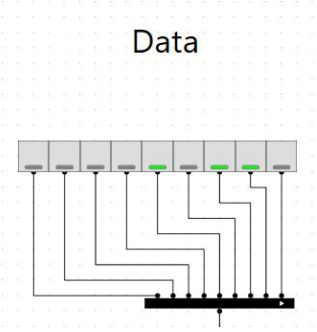
---

## Writing instructions and data:

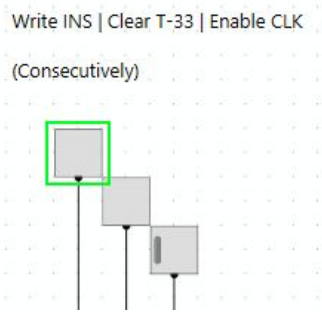
1. **Select the desired data or instruction to input by pressing the associated buttons.**
2. **Press the 'Write INS' button.**
3. **Deselect the inputs and repeat.**

**After completing step two, one light should now be showing on the 'Memory Capacity'**

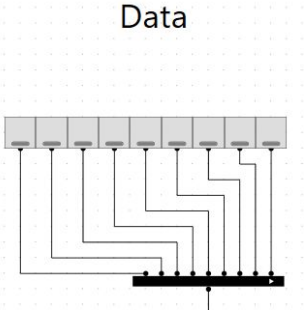
**1.**

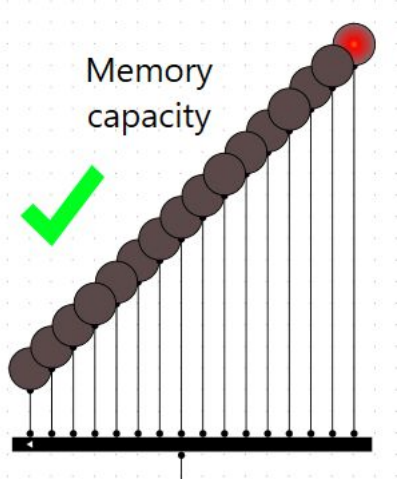



**2.**




**3.**



# Using the Circuit

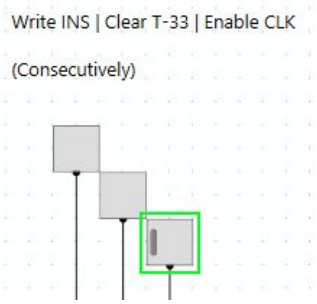
---

## Running the Simulation:

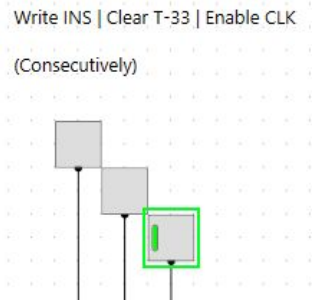
1. **Once the desired input has been saved, press the 'Enable CLK' button.**
2. **Once complete, press the 'Enable CLK' button again to halt the circuit.**

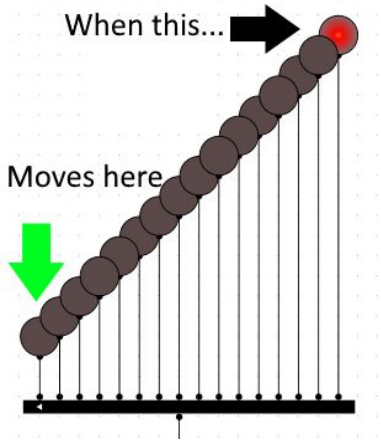
**Notice the memory capacity decrementing. When a capacity status LED reaches the final LED, the code is sent to the processor.**

**1.**




**2.**




## Tips:

**When reading a register and intending to see the value from the output, halt the machine state by pressing the 'Enable CLK' button just as it enters the T-33 at the last capacity LED.**

# Examples

---

## 11 * 3:

| Data: | Instruction: | Instruction microcode: |
|---|---|---|
| 00001011 0 | NULL | 00000000 |
| 00000000 0 | ADD WRITE1 | 00000001 |
| 00000000 0 | ADD SHFT WRITE2 | 00010010 |
| 00000000 1 | NULL | 00000000 |
| 00000000 0 | READ2 | 00001000 |
| 00000000 0 | SWP | 10000000 |
| 00000000 0 | READ1 | 00000100 |
| 00000000 0 | ADD WRITE1 | 00000001 |
| 00000000 0 | READ1 | 00000100 |

**Write the value of 11 to ACC > Add the ALU registers and write to storage register 1 > Add the ALU registers, shift the result and write to storage register 2 > Clear ACC > Copy the value of storage register 2 into the ACC > Swap the ALU registers > Copy the value of storage register 1 into the ACC > Add the ALU registers and write to storage register 1 > Display result ($33_{10}$ or $00100001_2$)**

## 2 * 17 - 29:

| Data: | Instruction: | Instruction microcode: |
|---|---|---|
| 00010001 0 | ADD SHFT WRT1 | 00010001 |
| 00000000 1 | NULL | 00000000 |
| 00011101 0 | NULL | 00000000 |
| 00000000 0 | SWP | 10000000 |
| 00000000 0 | SUB READ1 WRT2 | 00100110 |
| 00000000 0 | READ2 | 00001000 |

**Input $17_{10}$, shift and write it to register 1 > Clear ACC > Input $29_{10}$ > Swap the ALU registers > Read the value of register 1, subtract the ALU registers and write result to register 2 > Display result**

# Examples

---

## $3 * 2^4$:

| Data: | Instruction: | Instruction microcode: |
|---|---|---|
| 00000011 0 | ADD SHFT WRITE1 | 00010001 |
| 00000000 1 | NULL | 00000000 |
| 00000000 0 | ADD SHFT READ1 WRITE1 | 00010101 |
| 00000000 1 | NULL | 00000000 |
| 00000000 0 | ADD SHFT READ1 WRITE1 | 00010101 |
| 00000000 1 | NULL | 00000000 |
| 00000000 0 | ADD SHFT READ1 WRITE1 | 00010101 |
| 00000000 0 | READ1 | 00000100 |

**Input $3_{10}$, shift and write to register 1 > Clear ACC > Read 1, shift it and write it back to register 1 (repeat 3 times) > Display result**