


Made by Joshua Bernstein

*I pledge my honor that I have abided by the Stevens Honor System.*

# E.T.H.E.L Architecture CPU



**E:** Experimental  
**T:** Test for  
**H:** High-functioning  
**E:** Execution  
**L:** Language

## Getting Started with ETHEL Assembly

To start writing in ETHEL assembly, open any code editor, and create a file with the “.et” extension<sup>1</sup>. Then, copy the assembler program, which can be found here:

[https://drive.google.com/file/d/1-5Fp-pfCU7ret9Xn7l66jIS54K\\_JhwoV/view?usp=share\\_link](https://drive.google.com/file/d/1-5Fp-pfCU7ret9Xn7l66jIS54K_JhwoV/view?usp=share_link).

To actually assemble the program, you first need to install python onto your OS, and it can be found here: <https://www.python.org/downloads/>. After that, you need to go into a command terminal, and navigate to the directory that contains your .et file. Once there, type in python, followed by the name of the assembler program, space, and the name of the .et file (with the .et extension).

```
PS D:\Coding\382> python .\ETHELAssembler.py demo.et
PS D:\Coding\382> █
```

Once the assembler has completed, you will be left with 1 or 2 image files (depending on the content of your code). These files, named “Instructions” and “Data”, can be loaded into the instruction or data memory spots of Logisim using the “LOAD IMAGE” option that appears when right-clicking memory.

## ETHEL Language Basics

The ETHEL language is split into two groups, the .CODE segment and the .VAR segment. A functioning ETHEL program always have a .CODE segment, but .VAR is optional to include, and determines whether or not you want data in the data memory at the start of the program. .CODE contains the instructions carried out by the CPU, and is therefore, crucial to include.

---

<sup>1</sup> NOTE: It does not need this file extension, but it’s just fun

### *.CODE*

There are 11 basic instructions in ETHEL assembly, below they are listed with their equivalent operation in ARMv8 assembly:

ETHEL Version	ARMv8 Equivalent
FETCH	LDR
STAY	STR
UP	ADD
DOWN	SUB
JUMP	B
JUMPT	CBZ
JUMPS	BL
BACK	RET
DROPIT	MOV
DROPITN	MOV [With Immediate #s]
SPEAK	ADR

In ETHEL, you can comment entire lines or just part of lines using “//”. Lines or parts of lines with these notations will not be assembled and be completely ignored by the assembler program. Labels can be affixed to lines in the exact same format as ARMv8, with a string of characters followed by a colon.

```
No: UP E1, E2, E7//What
```

Moreover, it is important to note that all ETHEL instructions deal with only either registers or labels as their operands **EXCEPT for** `DROPITN`, which can take in an immediate number as its argument.

### *.DATA*

The `.DATA` segment of the code is incredibly simple, it is just a variable name followed by a value with an equals sign in between. These values can be up to 64 bits in size. To then acquire the address of this data, you use the `SPEAK` instruction, with arguments target register and variable name.

```
SPEAK E7, r
```

```
.VAR//Variable Segment
y = 765
w = 4
x = 2142
r = 928173
o = 9223
```

## ETHEL Architecture Description

ETHEL CPUs contain 8 general-purpose 64 bit registers. In ETHEL Assembly, they are referred to as E[X] where X is the register number from 0 to 7. The CPU can do two primary arithmetic functions, addition and subtraction. In ETHEL language, these actions are completed using the UP and DOWN instruction, respectively.

Furthermore, ETHEL is capable of both loading and storing from memory, as well as having a full implementation of branching, including procedure-related branching (using register E7 as the link register). It follows the sequential datapath model with no pipelining of any kind.

## ETHEL Instructions – In Depth

There are three primary classes of ETHEL instructions. First, there are instructions with three operands, all of which are required to be register names (Standard Type). Then, there are instructions with 2 operands, a destination (or source) register and an immediate number (Number Type). Finally, there is an instruction type which only takes in two operands, but is sized in memory for a third. This type is only used for memory instructions `FETCH` and `STAY` (Augmented Type).

### Standard - Type Instructions

	ALUp		ALUsrc	RegWrite	Reg2Loc	MemWrite	MemRead	MemToReg	LinkReg	UBr	CBR	Pbr	FILLER																dest/src			op1/base			op2		
Instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
UP dest, op1, op2	0	1		0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1														
DOWN dest, op1, op2	1	1		0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1														
PROFIT dest src	0	0		0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1														

### Number - Type Instructions

[illegible]

## Augmented - Type Instructions

	ALUop		ALUsrc		RegWrite	Reg2Loc	MemWrite	MemRead	MemToReg	LinkReg	UBr	CBr	Pbr	FILLER																op1/src			op1/src			op1		
Instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
FETCH dest. base	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1															
STAY src. base	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1															

Each instruction in ETHEL is 32 bits, or 4 bytes in length, following is a breakdown of why it needs to be this long:

1. **Bits 31 - 20:** opCode required to run all of the different instructions.
  - a. Each bit in the opCode corresponds to a different control signal in the ETHEL CPU.
2. **Bit 19:** Filler to keep the instruction length a power of 2.
3. **Bits 18-3:** Used in Number-Type instructions as the offset value.
  - a. ETHEL allows the user to input up to a 16-bit offset value.
4. **Bits 8-6/5-3:** Used in Standard and Augmented types for a register value.
  - a. It needs to be 3 bits in order to represent a number from 0-7 as there are 8 possible registers.
5. **Bits 2-0:** Used in all instruction types to denote a register

## The End!

You now know everything you need to start using ETHEL architecture and ETHEL assembly! Best of luck!