Whitepaper

for

THEO 32

8-BIT CPU

2024 V1.0

Mathijs Klaver

Introduction

Welcome to the whitepaper of the THEO 32 CPU. This whitepaper contains all the provisional functions and workings of this CPU.

The goal of this project is to learn about the limitations of CPU made of transistors that can be seen by the human eye (SOT32 *2,5x1mm*), as well as getting a deeper understanding of computer logic and programming on the hardware level.

Expectations of this project

There are a few expectations that have been set for this CPU to keep a minimum. One of these expectations is that the CPU should be able to process simple games like pong live. Bonus expectations are being able to run a custom operating system with integrated MS BASIC support.

Limitations for this project

To keep this project to become unrealistic, a maximum of 3000 transistors was set. This number of transistors is comparable to the Intel 4004 from *1971,* the first CPU to ever be produced. While more transistors mean that the processor can process more data faster, the size of the board needed to contain these extra transistor increases. More transistors also mean more assembly and because this will not be done manually. This will add considerable cost to the CPU.

Audience

This documentation is targeted to individuals with a basic understanding of computer architecture. Not every detail is covered in this documentation.

Disclaimer

It's important to note that the THEO 32 CPU is a product of individual design and may not represent the most efficient solution for all components. Therefore, it should be treated as an experimental design and not used in real-life applications. This CPU is also not compatible with existing code because of its custom architecture / instruction set.

(THEO 32 is completely opensource and anyone is free to take information from this documentation for their own purposes)

Features of the THEO 32

- 8-bit Data Bus
- 16-bit Address Bus to provide 65536 available addresses
- 8-bit ALU, Accumulator, AR Register, Instruction Register
- 15-bit Access to RAM, ROM and the bootloader to provide 32768 available bytes per storage unit.
- Direct compatibility with the AY-3-8910 sound IC
- 5 possible device extensions
- 16-bit Program Counter
- 51 Custom Operation Codes (OpCodes)
- 5v Operating Voltage

Extra for THEO 32

To be able to write code for this CPU efficiently a new compiler called THASM had to be written. This compiler compiles a custom programming language into a binary file that can be uploaded to one of the EEPROMS on the CPU. THASM is available on Github.

Basic Concepts

Computer Architecture:

Computer Architecture is the design and organization of hardware components within the CPU. It defines how these components interact to process data and execute instructions.

Registers:

Registers are small storage units located within the CPU. They store temporary data during the processing of a instruction. They service in several places including the ALU (Arithmetic Logic Unit) and communications between the CPU and RAM.

Memory:

Memory refers to the storage space where data and instructions are stored for processing by the CPU. There are two types of memory.

- **RAM** (Random Access Memory) witch stores data temporarily and is very quick. Contains data witch the CPU is currently working on.
- **ROM** (Read Only Memory) witch stores data forever, it stores data like games or programs, but is slower than RAM. ROM can't be written to.
- **EEPROM** (Electrically Erasable Programmable Read Only Memory), it functions the same as ROM, but using electrical signals the CPU can write to the ROM to change its data. This makes things like game progression storage possible.

Instruction Set:

The Instruction Set defines a set of instruction that a CPU can execute with their opcodes. It serves as the interface between software and hardware making it possible to make or run programs to be executed on the CPU.

Control Unit (CU)

The Control Unit is a big part of the CPU. It controls all the other components of the CPU based on the instruction its executing, the clock and Micro Steps (Smaller steps to complete an instruction). This is created using a complex combination of logic gates. The CU is the only component that can change information / pins on the Control Bus.

Arithmetic Logic Unit (ALU)

The ALU does all the calculations in the CPU, as well as do comparison or logical operations on binary numbers.

Architecture Details

Early during the design phase, a few definitive choices about the architecture had to be made. Here are the definitive choices about the architecture:

- **8-Bit System**, After careful consideration the choice was made to not go higher or lower than 8-bit. While this CPU is for educational purposes and thus does not need to be able to perform certain tasks. Anything less than 8 bit will be to slow to perform more complex tasks. Anything higher will be faster, but also larger and will use more transistors and thus going over the limit of 3000 transistors. 8-bit was the sweet spot between cost and performance.
- **CMOS**, The CMOS type system is used in the THEO 32. This is because other logic types like NMOS which was commonly used for the first CPUs, would draw too much current at this scale and would become unfeasible.
- **SOT32 2N7002 & AO3401**, these transistors are used in THEO 32, NMOS and PMOS respectively. These transistors both fulfil the minimum requirements for this CPU while being relatively cheap. The size of the transistors is also a perfect balance between size, assembly cost and transistor density. The area per transistor is roughly 4x3,5mm.
- **No X,Y registers**, Microprocessors like the MOS 6502 used extra registers X,Y to store data temporarily. While this makes operations faster, every register takes a lot of transistors and the faster speeds were not worth the extra cost. This choice doesn’t impact the capabilities.
- **Limited ALU**, The ALU present has limited functionality. It can only do adding, subtracting and a few logical operations. This is because of the large amount of transistors a multiplication or division module takes. This was not worth the extra speed and efficiency.

Architecture Overview

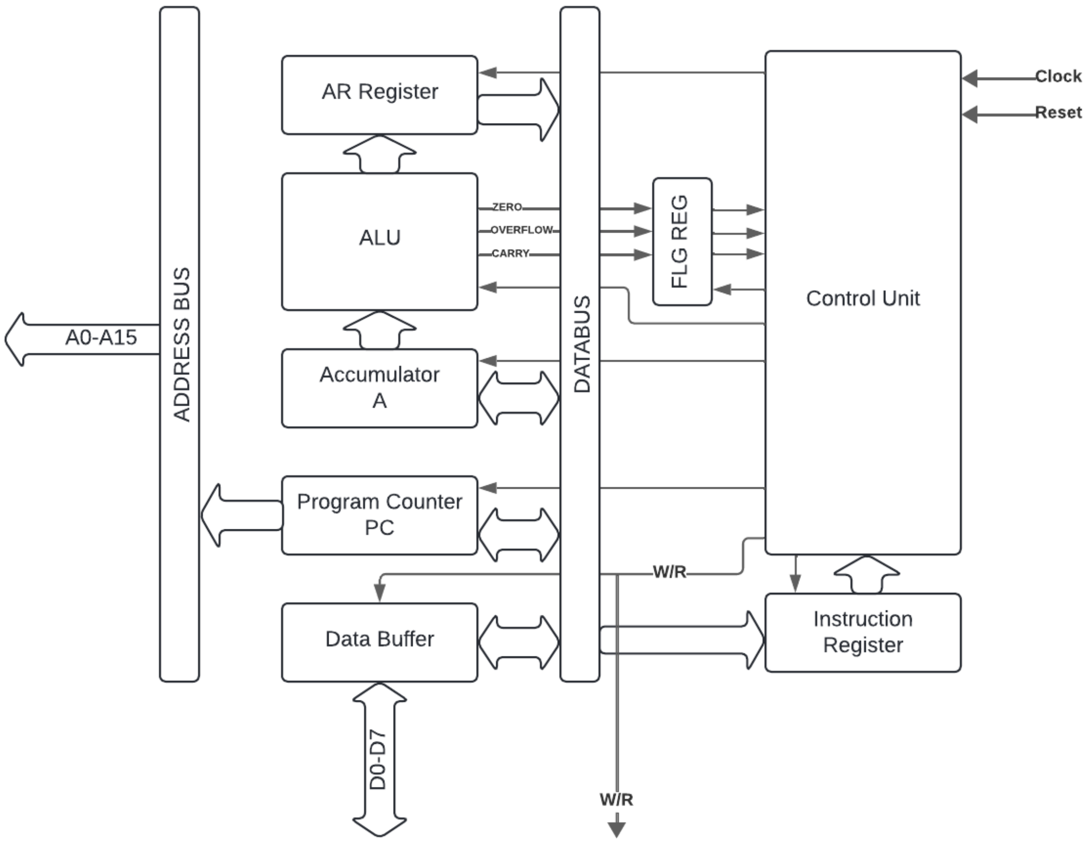

FIGURE 1: THEO 32 (SIMPLIFIED) ARCHITECTURE DIAGRAM

Control Unit (CU)

The Control Unit is provides timing for each instruction cycle that is executed. Based on the instruction the control unit gets from the Instruction Register, it turns control lines on and off to move data through the CPU. Every component is dependent on the CU for instructions on what to do.

Instruction Register (IR)

The Operation Code (OpCode) portion of the instruction is loaded into the instruction register. The Control Unit has direct access to this data and can decode this to it knows what instruction to perform.

Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations are done within the ALU. The ALU has a limited functionality and can only do: Add, Subtract, Shift left, Shift right, Comparison (Greater, Less or Equal), AND, OR and finally XOR operations. The result of Comparison is stored in the Flags Register (FLG) which is read by the Control Unit for conditional jumps. The result of a arithmetic or logic operation automatically gets stored in the Accumulator, this move from registers can't be influenced by software.

Arithmetic Register(AR)

The result of a Arithmetic or Logical operation get stored in the Arithmetic Register so the CU can use the next Micro Instructions to put the value into the Accumulator.

Program Counter (PC)

The Program Counter is a 16 bit register that is used for keeping track of the current address. The PC can Increment to fetch the next instruction or argument of the instruction or load a value from the data bus into the PC to jump to another location in RAM/ROM or the Bootloader.

Flag Register (FLG)

The Flag Register stores 3 bits called: ZERO, OVERFLOW and CARRY. These Flags serve 2 purposes

Normal Arithmetic Operation: *(CARRY) Add, (CARRY) Subtract*:

- **ZERO:** 0 Means that the result of the operation was not zero ***(1+1 ≠ 0)***, 1 Means that the result of the operation was zero ***(1-1 = 0)***.
- **OVERFLOW:** 0 Means that the result of the operation was not less than 0, ***(23-20 =3   3>0)***, 1 Means that the result of the operation was less than 0, causing the value to start from 255 and count down ***(100-200 = -100 | -100 < 0 | 255-100 = 155)***.
- **CARRY:** 0 Means that the result of the operation was not over 255, ***(1+1 =2 | 2<255)***, 1 Means that the result of the operation was over 255, causing the value to start from 0 ***(128+200 = 328 | 328 > 255)***.

Comparison Operation CMP (V):

- **ZERO:** 0 Means that the value in the Accumulator and V are not equal, 1 Means that the value in the Accumulator and V are equal
- **OVERFLOW:** 0 Means that value in the Accumulator is not less than V, 1 Means that value in the Accumulator is less than V.
- **CARRY:** 0 Means that value in the Accumulator is not greater than V, 1 Means that value in the Accumulator is greater than V.

With these flags go the 3 instruction JME, JMO and JMC (*Jump if equal, Jump if overflow and Jump if carry* respectively). These instructions can be used to create if-statements in software.

Address Bus

The Address Bus is a 16 bit bus to access 65536 possible addresses. Only Half of these addresses are actually used for accessing RAM/RAM and the Bootloader making it 32768 addresses available per storage unit, making this a CPU with a maximal of 32KB (The 32KB is not shared between RAM, ROM and the Bootloader, but instead all 3 can be 32KB). The other half is not spend on storage, but instead on external devices for sound, video and interfaces (VIA or ACIA). See Address Map.

Data Bus

The Data Bus is a 8-bit bus to transfer data through the CPU, communicate to RAM/RAM or the bootloader and to communicate to external devices.

Operation Tables

	Mnemonic	Description
1	NOP*	No OPeration
2	CNU*	CoNtinUe to boot and start reading from RAM
3	BRK*	BReaK the program, halt the CPU
4	INC*	INCrement the Accumulator
5	DEC*	DECrement the Accumulator
6	ADD	ADD a value to the Accumulator
7	SUB	SUBtract a value to the Accumulator
8	ADC	ADd a value to the Accumulator with Carry
9	SUC	SUbtract a value to the Accumulator with Carry
10	ROL*	bitwise ROll Left Accumulator
11	ROR*	bitwise ROll Right Accumulator
12	AND	AND a value with the Accumulator
13	OR	OR a value with the Accumulator
14	XOR	XOR a value with the Accumulator
15	CMP	CoMPare a value with the Accumulator
16	STA	STore the value inside the Accumulator into RAM/ROM
17	LDA	LoaD a value into the Accumulator
19	!LDA(Boot)	LoaD a value from the bootloader into the Accumulator
20	JMP	JuMP to new location
21	JME	JuMP to new location if Zero flag is set
22	JMC	JuMP to new location if Carry flag is set
23	JMO	JuMP to new location if Overflow flag is set

FIGUR 2: INSTRUCTION SET TABLE

Notes fig. 2:

! = This operation runs in/from the Bootloader.
** = This operation takes no arguments. Standalone instruction.*

	LOWER NIBBLE								
	0	1	2	3	4	5	6	7	
0	NOP*	INC*	ADD, $A	ROL*	AND, $A	CMP, $A	STA, $A		**RAM**
1	CNU*	DEC*	ADD,#V	ROR*	AND, #V	CMP,#V		JMP,$A	
2	BRK*		SUB, $A		OR, $A		LDA, $A		
3			SUBB,#V		OR,#V		LDA,#V	JME,$A	
4			ADC, $A		XOR, $A				
5			ADC,#V		XOR,#V			JMC,$A	
6			SUC, $A				!LDA, $A		
7			SUC,#V				!LDA,#V	JMO,$A	
8			ADD, $A		AND, $A	CMP, $A	STA, $A		**ROM**
9			ADD,#V		AND,#V	CMP,#V			
A			SUBB, $A		OR, $A		LDA, $A		
B			SUBB,#V		OR,#V		LDA,#V		
C			ADC, $A		XOR, $A				
D			ADC,#V		XOR,#V				
E			SUC, $A						
F			SUC,#V						

FIGUR 3: INSTRUCTION SET TABLE

Notes fig. 2:

$A = Expects 2 arguments for the address in RAM/ROM. ***Big-endian***
#V = Expects 1 argument containing the value to do the operation with.
** = This operation takes no arguments. Standalone instruction.*
! = This operation runs in/from the Bootloader, overrides the RAM/ROM section.

Addressing

The Mode Bit is the MSb (Most significant bit) of the address and influences the access mode. When **MSb = 0**, *the CPU accesses* <u>*normally*</u> *RAM/ROM or the Boot-loader*, but when **MSb = 1**, *the CPU accesses* <u>*external*</u> *devices. T*his type of system is not efficient at all and wastes a lot of address space, but this reduces the hardware required to add external devices like sound, video and interface integrated circuits. The CPU can use normal read and write instructions to read or write data to these addresses. Some external devices may use the remaining 'Control lines' to set the mode.

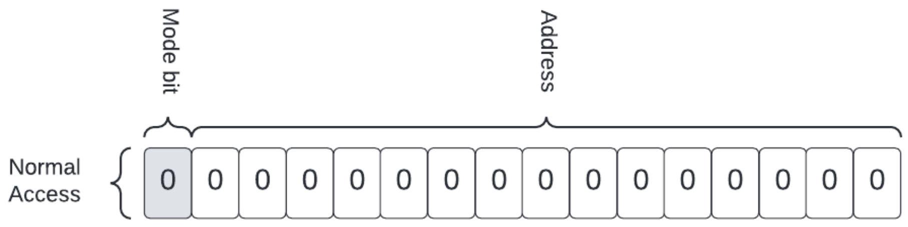

FIGURE 4: NORMAL ACCESS

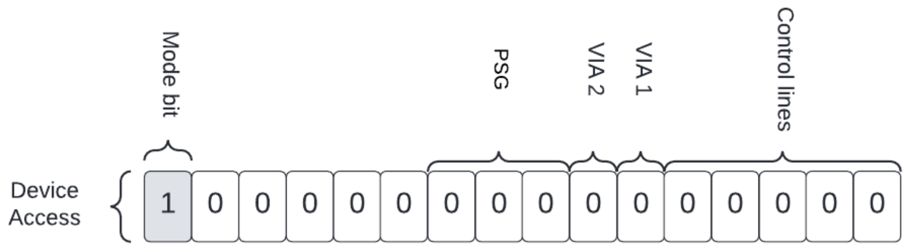

FIGURE 5: DEVICE ACCESS

Address Map

Address	**Label**
$0000-$7FFF	RAM/ROM/BOOTLOADER
$8000 -$801F	Control lines
$8020	VIA 1 IC **(MOS 6522) I/O**
$8040	VIA 2 IC **(MOS 6522) I/O**
$8080 - $8380	**(PSG AY-3-8910) Sound**
$8400	External **CE** (Chip Enable) 1
$8800	External **CE** (Chip Enable) 2
$9000	External **CE** (Chip Enable) 3
$A000	External **CE** (Chip Enable) 4
$C000	External **CE** (Chip Enable) 5

FIGURE 6: TABLE CONTAINING ADDRESS MAP

Integration THEO 32

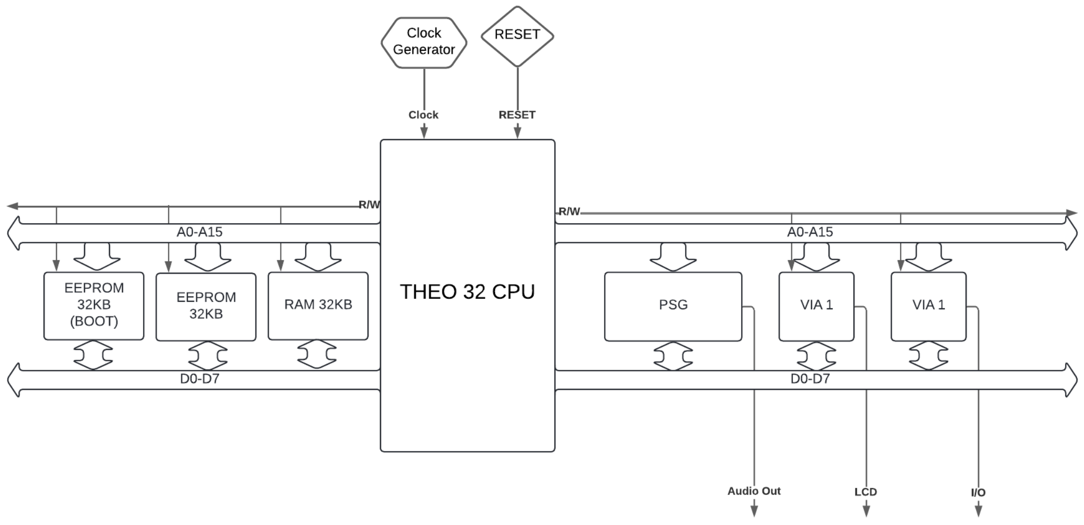

FIGURE 7: INTEGRATION OF THEO32

The THEO 32 CPU is designed to be connected to a few other components. Here are a collection of the components with their description:

Name	Description
EEPROM 32KB (Boot)	Bootloader -> The code that runs first on start, reads program and puts program in ram for execute
EEPROM 32KB	Storage -> The chip that contains a program or information that gets loaded into Ram to execute
RAM 32KB	Memory - > Contains all temporary information like variables, stack/stack pointer and the program that the CPU has to execute
SI-THEO	Sound Interface -> Produce sound / music.
VIA 1	Versatile interface 1 -> General purpose I/O
VIA 2	Versatile interface 2 -> General purpose I/O

With these base components, already complex calculations or controlling / interpreting data is possible. External devices can be connected for outputs like Video or communications via serial.

External Devices and ICs

External Devices should be connected to the THEO 32 by its dedicated EDC (External Device Connector) ports. These EDC consist of 2types: B-EDC & I-EDC.

Maximal power rating for these ports is yet to be determined. The estimated maximal power usage for the 5V lines on B-EDC is 300mA (0.3A) which is equivalent to around 1.5 Watt per power line.

Name	**Pins**
B-EDC	A0-A15, D0-D7, RE,WE,5V,5V,0V,GND
I-EDC	IO 0 - IO 32

All ICs on the CPU board that require addressing can't change address. See Addressing section of this document figure 6 for information about the addresses of certain components.

I/O Interfaces

The THEO 32 has 2 onboard sockets for **MOS 6522 VIA** (Versatile Interface Adapter) chips (VIA 1 & VIA 2). These chips both have two 8-bit channels for input or output. With these chips the THEO 32 is able to communicate with character displays, joysticks, interrupt control or a simple serial communication.

Information about the addresses of these VIAs can be found in the Addressing section of this document, figure 6.

On the CPU Board there is room for a 16x2 LCD Character display. This LCD displays port is directly connected to the outputs of the VIA 1

Sound Interface

The SI-THEO (Sound Interface THEO) is an integrated system on the CPU to generate sound using a PSG.

The PSG (Programmable Sound Generator) is a 16-register based sound generator that can produce a wide variety of complex sounds. This IC is integrated on the CPU and is therefore directly accessible with software.

The PSG used is the **AY-3-8910** from General Instrument released in *1978*. This device is connected to addresses $8080 - $8380. (See figure 8). In the table (See figure 9) the addresses with their corresponding functions are put, as well as an example.

(CONTINUES ON NEXT PAGE)

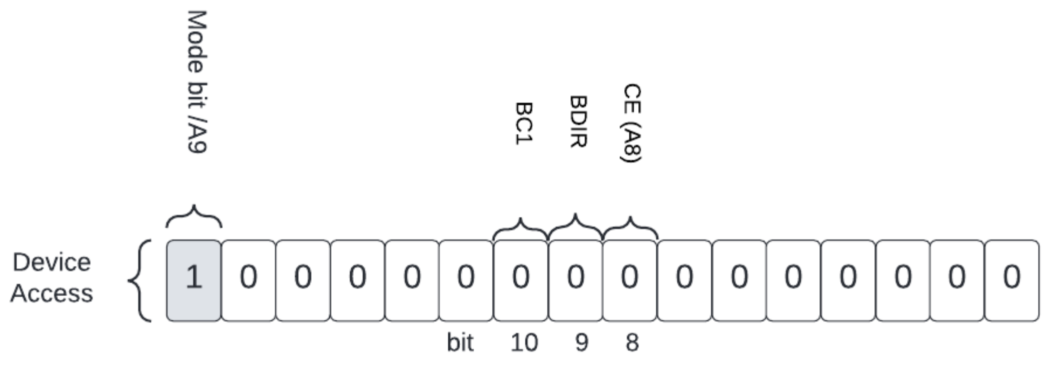

FIGURE 8: PSG ACCESS

PSG Function Name	Address	Example
INACTIVE	0x8080	NOP (NOTHING)
READ FROM PSG	0x8280	LDA, 0x8280
WRITE TO PSG	0x8180	STA, 0x8180
LATCH ADDRESS	0x8380	STA, 0x8380

FIGURE 9: PSG ADDRESS TABLE

Video Interface

(The Video Interface *is not integrated on the CPU board*)

The VI-THEO (Video Interface THEO) is a extension for the THEO 32 to add video capabilities to the CPU. This Video Interface is controlled by a VDP (Video Display Processor).

The VDP used is the **TMS9918** from Texas Instruments from *1979*. The VDP has several features including a 256x192 resolution, 15 unique colours, composite video output and 32 planes each containing a single sprite and more.

Source Reference

The Western Design Center Inc. (2009). W65C22S VIA. https://www.westerndesigncenter.com/wdc/documentation/w65c22s.pdf

General Instruments. AY-3-8910. https://map.grauw.nl/resources/sound/generalinstrument_ay-3-8910.pdf

Alliance Memory Inc. 32Kx8 BIT CMOS SRAM (AS6C62256). https://nl.mouser.com/datasheet/2/12/AS6C62256_23_March_2016_rev1_2-1288423.pdf

ATMEL. 32Kx8 BIT EEPROM (AT28C256). https://nl.mouser.com/datasheet/2/268/doc0006-1108095.pdf

Additional Resources

Github Repository: https://github.com/MathijssYDev/THEO-32