Chapter 2

Introduction to myCPU

The myCPU is a fully modular **8-bit TTL/CMOS CPU** designed using only discrete logic elements implemented by the most common integrated circuits from the 74xx family over CMOS or TTL technologies. The myCPU instruction encode/decode is based on the Microprograming technique and the instruction execution flow is based on Microinstructions. The myCPU support debugging at level of an individual microinstruction. It was designed on printed circuit boards (PCBs) and it can built using TTL or CMOS technologies because the design is TTL compatible.

The feature of debugging at microinstruction level allows to observe the execution of individual microinstructions and step through each one, viewing the state of all logic components statically. Providing a detailed view of the instruction execution flow in real-time.

2.1 A development and learning Platform

The myCPU is designed as a learning platform to provide an educational experience in the workings of a CPU, including the logic elements involved in its basic architecture and how various digital logic elements interact in a synchronized way.

The myCPU, with its modular design and hardware-level accessibility, also serves as a development platform. With the ability to directly access the relationship between hardware and software, users have the opportunity to design their own modules and create their own instruction sets for program writing. This is made possible through the Instruction Decoder, a programmable EEProm-based decoder that can be configured with custom microinstructions to define a unique instruction set.

The myCPU architecture, including its modular design and the choice of components and their distribution, was created with the goal of providing a clear understanding of a CPU system and its workings to people of all levels, including beginners. The components can also be easily purchased from a variety of sources, including non-official online retailers and official stores.

👁 *You can see a more extensive explanation about the components in the Chapter: 4 - The Components, p.57.*

As a learning platform, the myCPU design may not be the most technically advanced approach for experienced engineers or advanced electronic hobbyists. While the design could potentially be improved for better performance, this was not a primary objective in its develop but the modularity, the open hardware and the accesibility of the design to beginners, students and less experienced people.

The next list shows some of the topics you can learn with this project:

- *Understanding the electronic relationship between software and hardware*
- *Understanding a basic CPU architecture, blocks, and logic devices involved*
- *Learn how a CPU execute an instruction and a whole program*
- *Learn how to implement a CPU instruction set and design new instructions*
- *Learn what a microinstruction is and how microinstructions are decoded*
- *Understanding the timings of a CPU, and the cycle of an instruction*
- And much more...

In this introduction, I will discuss some aspects of the myCPU that may seem obvious to more experienced individuals. However, as a learning platform, it is valuable to review these elements to gain a deeper understanding of the design from a perspective similar to that of real CPUs, rather than simply viewing it as an exercise in electronic assembly.

The extensive documentation provided may appear redundant or over-detailed to experienced individuals, but it is valuable for beginners or those with limited knowledge of digital electronics.

2.2 Open Architecture and modular design

The myCPU was designed with an open architecture concept in mind, focusing on modularity and ease of customization. This allows for scalability in size and complexity through the addition of additional modules or replacement of the provided ones with more advanced ones.

This design is made possible by the use of a base hardware layer composed by **chained BUS module boards**. The boards in the chain connect the data and control buses each other, and share them with the two modules plugged on both sides of the board. Each BUS module supplies power to the connected modules through its own voltage regulator and the 9V power source are passing along the chain. As a result, the power supply of a big composed circuit like that can be possible successfully without any issues, and you can increase the capabilities of your myCPU by adding more BUS module boards and additional modules. Due to the basic nature of the first release of myCPU, only 8 BUS module boards are needed to built a basic myCPU layout, although up to 10 BUS module boards will be provided in the myCPU kit. Is possible to build a chain of 8 BUS module boards in a vertical layout or 2 chains of 4 BUS module boards in a landscape layout to run the myCPU.

The modular and open architecture of the myCPU allows you to customize its functionality and behavior by designing your own modules and module boards. When designing custom modules, the only consideration to keep in mind, maintaining the compatibility with the myCPU design, is to properly place the control and data bus connectors according the mechanical dimensions specification of the BUS module board design.

👁 *You can check the mechanical dimensions of the BUS module board and module boards in the appendix: B.2 - Technical views, p.131.*

Moreover, the modular design allow you to customize the layout of your myCPU by choosing your desired module distribution. You can create two different types of layouts using either one chain of BUS module boards for a vertical layout or two chains connected by the myCPU layout connector for a landscape layout. However, not all module distributions may be possible due to the flat wire connections between certain modules forming a block. The position of a module will also depend on whether it has versions for both orientations: left and right. Regardless, you will find that the system is highly flexible.

2.3 Hardware architecture

The myCPU physical architecture consists of 3 hardware layers, plus an additional optional layer if a module expansion board is used.

1. **BUS layer** (Main board layer providing power supply)
2. **Module layer** (Modules)
3. **Module Expansion layer** (Optional)

4. **Auxiliary layer** (Output display or auxiliary modules)

The concept behind this hardware architecture is to separate the main BUS lines and power supply from the modules using a base **BUS layer**. This allow the buses to be shared among the module boards, through a base chain of BUS module boards. The logic elements forming the CPU are implemented as separate modules on a **Module layer** and connected to the buses through the BUS layer. The output display functionality is also isolated from the modules through an **Auxiliary layer**.

👁 *This feature enable the upgrade of the features of the myCPU by replacing existing modules with new ones without altering the existing BUS module base structure.*

The **Module layer** enable the possibility to design logic devices in a independent way and connect them to the buses through the BUS layer. Additionally allow the development of complex logic devices and segmenting them into smaller modular components such modules, and establishing connections between these modules using direct flat wires and IDC connectors.

The **Auxiliary layer** enable the possibility to design your own display or output modules because of the isolation of the device, handling the output, from the module itself. The myCPU kit includes three display modules: a 4-digit Hex or Decimal number display, an 8-bit LED binary display using 3mm LEDs, and an 8/16-bit binary display using LED bars for a clearer representation of a digital value.

The optional **Module Expansion layer** is used to design modules that due to their complexity and components density cannot fit into a single module board and you prefer have a compact module instead a wider one. You can see more information about expansion boards in the section: 10.5 - Expansion boards, p.90.

Chained BUS module board design

The BUS module board was designed to connect multiple BUS module boards together to form a **chained structure**, which support the myCPU layout formed by the modules. This is achieved by using two female connectors at the top and two male connectors at the bottom of the board connecting BUS module boards between them.

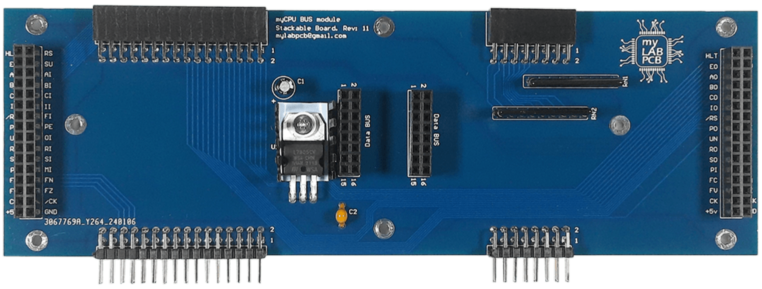
Figure 2.1: myCPU BUS module board

The myCPU supports different types of architectural layouts thanks to its modular nature. The most common are: **Vertical** and **Landscape** layouts but other layouts would be possible; In a landscape layout, 2 chains of BUS module boards are connected using the **Layout Connector** at the top to share the buses between both chains and the **Layout Terminator** at the bottom. In a vertical layout only a **Single Layout Connector** at the top is needed to provide the power source to the layout and IDC empty connectors such as terminators at the bottom to a clean isolation.

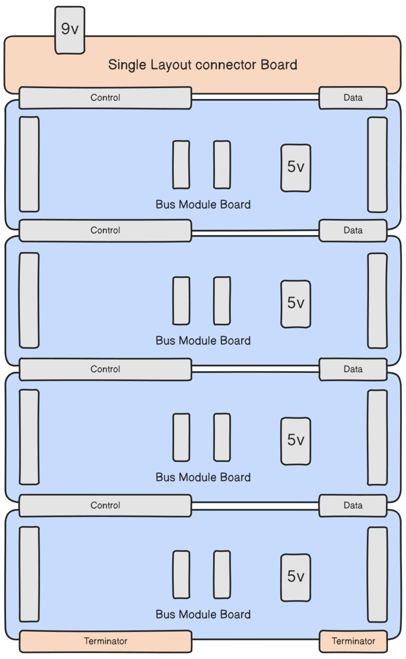

Figure 2.2: myCPU Layout vertical diagram

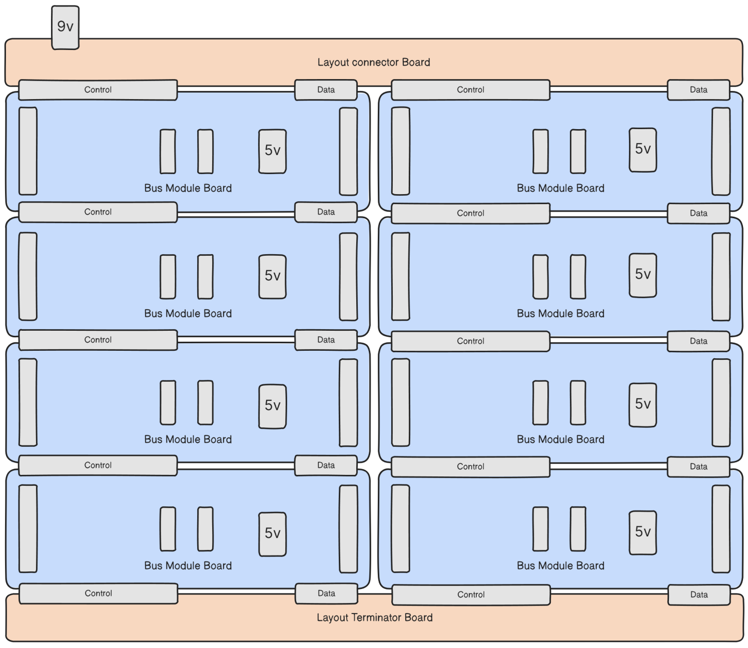

Figure 2.3: myCPU Layout landscape diagram°

The data and control buses, as well as the 9V power source, are propagated from one BUS module board to the next BUS module board in the chain using the top and bottom connectors of the board. And the buses are shared with the modules through a connectors pair located on both sides (left and right) of the BUS module board as you can see in the picture. For a more detailed explanation of the buses, see the section: 2.7 - The Buses, p.30.

Each Bus module board has its own +5V power supply using a 7805 voltage regulator, supplying power to the two modules plugged in to the board. The power source is supplied to the Bus module regulators through the chain connectors, and the power is supplied to the modules through the control bus connector.

This approach enable the use of a common 9V power source to provide a stable +5V power supply to the pair of modules plugged in to each Bus module board, without any voltage drop or electrical issues. The Layout Connector module board has the power source connector, a power switch and filter capacitors providing a clean +9v power supply to the BUS modules. You can find a full description of the BUS module in the chapter: 11 - The BUS module, p.91.

This design, which is based on providing an independent power supply for each pair of modules, makes it possible to increase the capabilities of the myCPU by adding more Bus module boards to the chain. This helps to avoid electrical issues related to the power supply needs of the ICs in a TTL compatibility power supply requirement that could arise from the size of the myCPU layout. This design supports the escalation with additional modules, even more complex modules at double board size.

2.4 Functional Architecture

The functional architecture describes the myCPU logic modules by their functionality, their response to specific control signals and the size of the data exchange between each module with the data bus. In addition describes the connections between modules belonging to a functional block. Figure bellow shows a diagram with a detailed view of the functional architecture.

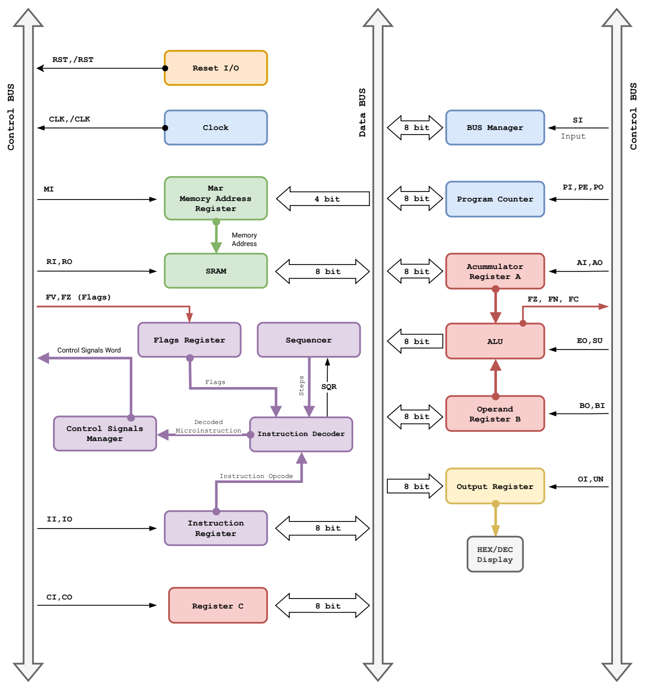

Figure 2.4: myCPU Architecture diagram

(Printable version can be found in the appendix.)

👁 *The myCPU design is based on 16 functional modules distributed over 15 physical modules.*

One of the most interesting features of the myCPU design, from a learning perspective, is that almost every logic element of the functional architecture corresponds to a physical module in the kit. Each module can be exchange separately with new modules from an upgrade, or custom modules with different characteristics, different digital functionality or simply exchanging the type or model of some of the ICs.

Functional blocks

In the myCPU design, logic modules sometimes are grouped as "**Functional Blocks**" because they working together to perform a specific task and could be connected by a dedicated connection, as in the case of the **ALU** module with the accumulator and operand, the **SRAM** module with the **MAR** module or the **Instruction Decoder** module with Sequencer, Flags Register, Instruction Register and CSM modules. The myCPU is based on **7 functional blocks**, plus one **Auxiliary block** that includes other modules to provide specific functionality or connectivity. Each block is composed by one or more functional modules and could be implemented using one or more physical modules. Bellow you can see a diagram view of the functional blocks architecture.

Additional tree view of the list of myCPU functional blocks can be found in the appendix: A - myCPU Diagrams, p.125 (Figure A.1).

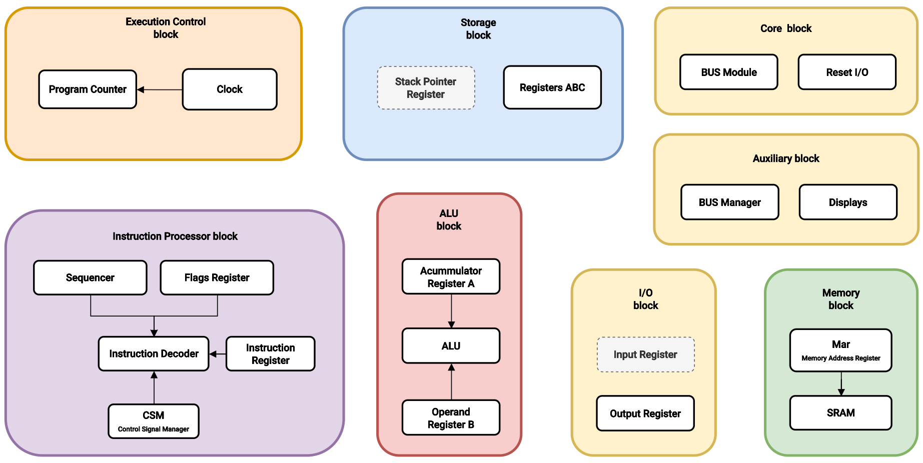

Figure 2.5: myCPU Functional Blocks diagram[1]

(Printable version can be found in the appendix.)

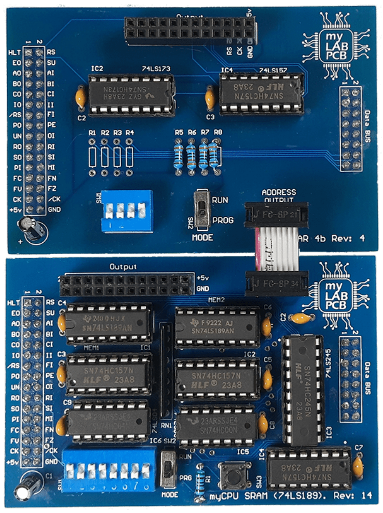

Figure 2.6: myCPU Memory block view

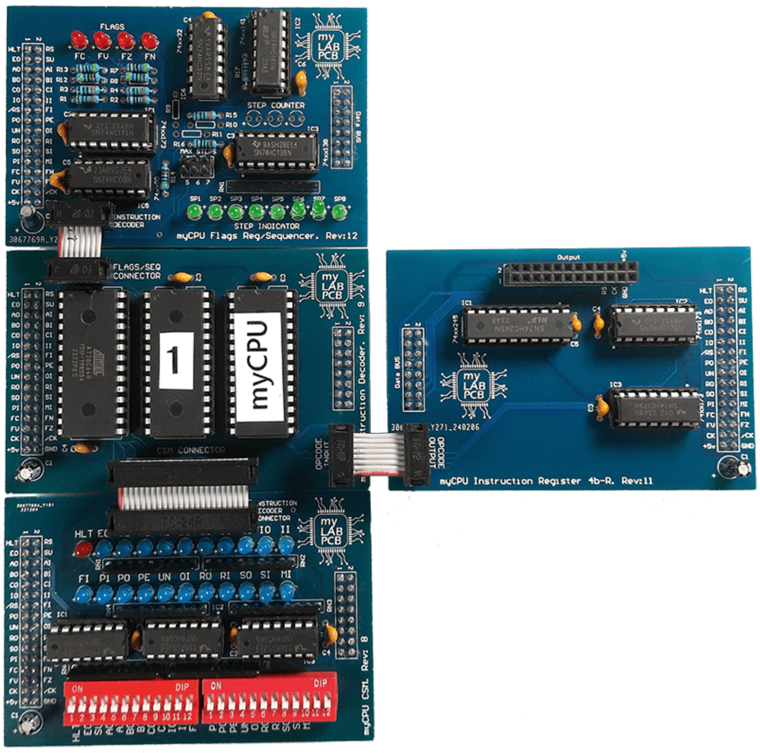

Figure 2.7: myCPU Instruction Processor block view

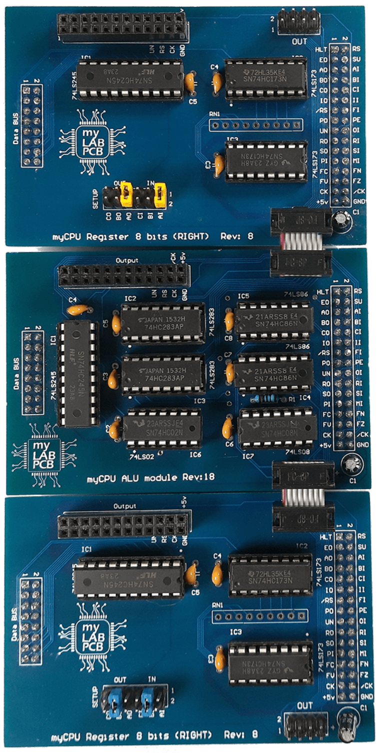

Figure 2.8: myCPU ALU block view

2.5 Modular architecture

The myCPU was designed with modularity in mind, allowing decoupling between functional logic. This allows for the design of modules that focus on specific logical functions and can be implemented as standalone physical modules. Modularity lets builders to create their own custom modules, control their complexity, and even personalize functional blocks by blending their own custom modules with those included in the myCPU kit.

This feature is very interesting, because you can redesign some of the modules included in the kit with your own new or modified version, while leave the others without any changes. An existing module could be redesign entirely, or just change one IC by another model of IC, for example changing the pair of 4-bit D-Type register 74xx173 IC of the Accumulator module by only one 8-bit D-Type register 74xx377 IC.

Modules have left-side or right-side compatibility, limiting their available positions

[1]Note that Stack Pointer and Input Register functional modules are not part of the architecture of the myCPU first release.

on the BUS module board. Right-Side modules cannot be plugged into the left side and Left-Side modules cannot be plugged into the right side. Not all modules have been designed for both-side compatibility, at the moment of write the book only the general-purpose register ABC board has versions for both sides.

Thanks to the modular architecture, you can build a highly personalized myCPU layout with a module distribution according to your preferences.

The myCPU design support customizable distributions of modules over the myCPU layout.

Module distribution layouts

According to the BUS module design, and thanks to its modular architecture, you can reorganize the positions of the myCPU modules to configure the myCPU layout. You will have the ability to choose from different physical layouts when plug in the modules. This feature enables you to customize the layout of your myCPU to suit your preferences. In my case, I prefer to place the clock module at the bottom of the layout, the BUS manager module at the top, and the memory block composed of the MAR and SRAM modules at the bottom as well, to make it easier to access the test switches.

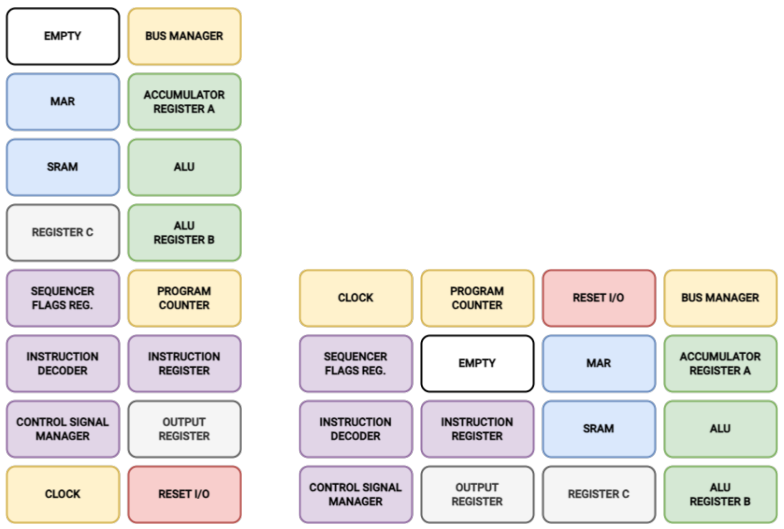

Figure 2.9: myCPU Layout examples

The myCPU design has certain layout limitations related to modules that are part of a functional block and are interconnected, or the compatible side version of the module.

Modules that are part of a functional block may need to be placed together due to the wiring between them.

In addition to customizing the distribution of the modules over the myCPU layout, you can choose the orientation of the layout: vertical or landscape. You can use a unique BUS module chain for the vertical layout or two BUS module chains for the landscape layout. The vertical layout need to be connected with **Single Layout Connector** to provide the power supply input. The landscape layout should have the chains connected by the **Layout Connector** at the top and the **Layout Terminator** at the bottom to keep connected both chains from to bottom, as well to provide the power supply input. You can See more info about it in the section: 26.1 - The Layout connector and Layout Terminator, p.123.

You will find some examples of myCPU layouts according to the functional block modules and module side version restrictions in the appendix: A - myCPU Diagrams, p.125 (figure A.4).

Functional and physical Modules

The myCPU architecture design include 16 functional modules in the final release, implemented on 15 physical module boards because the sequencer and the Flags register were designed on the same physical module board. Below you can find a list with all module boards will be provided in the myCPU kit.

Display Modules

The myCPU design include, in the final release, 3 display modules: two of them are LED based binary displays and the other is a binary to hexadecimal/decimal decoder 7 segment 4 digits display as you can see in the next table:

Additional tools and extra boards

The myCPU kit include one additional module for programming the EEProms AT28C64, used in the myCPU. Two auxiliary boards to connect and power the myCPU layout chains: a Single Layout Connector board for the vertical myCPU layout and a Layout Connector/Terminator board to support the landscape layout. And two extra board types for prototyping: A simple protoboard for general purpose and a specified protoboard with connectivity for a 245 and the output port added to the board. The protoboards have compatibility for left and right sides.

Table 2.1: myCPU List of physical modules boards

Num	Module	Description
1	`Reset I/O`	Reset module and I/O functions
2	`BUS Manager`	BUS display, testing and Input
3	`Clock`	Clock module
4	`CSM`	Control Signals Manager
5	`Program Counter`	8 bit counter
6	`Accumulator Register A`	8 bits register
7	`Operand Register B`	8 bits register
8	`ALU`	Simple full 8 bits adder with subtract
9	`Register C`	General purpose 8 bits register
10	`Output Register`	8 bits register
11	`MAR`	Memory Address 4 bits Register
12	`SRAM`	16 bytes SRAM memory
13	`Sequencer + Flags Register`	8 max steps sequencer and 4 flags register
14	`Instruction Register`	4+4 bits, instruction + argument
15	`Instruction Decoder`	24 bits microinstruction decoder

Table 2.2: myCPU List of display module boards

Num	Module	Description
1	`8 bits Hex/Dec Display`	8b binary to Hex/Dec decoder display
2	`8 bits LED display`	8b
3	`16 bits Bar LED display`	16b

2.6 Features of myCPU

In the next table you can find a brief list of myCPU digital features. Some of these features come from the original design of the Ben Eater's breadbooard computer and others are improvements over his design. It is highly recommended to understand the meaning of these features in relation to the design of the myCPU. In the following sections, I will briefly explain what each of those features means.

Table 2.3: myCPU List of extra boards

Num	Board
1	EEPROM programmer AT28C64B
2	Simple Protoboard
2	245 based Protoboard
3	Single Layout connector
4	Landscape Layout connector
4	Landscape Layout Terminator

Table 2.4: myCPU Features simplified list

Num	Feature
1	16-bit data BUS
2	32-lines control BUS (including non-control signals)
3	Microinstruction length up to 23 direct control signals
4	Instruction cycle with variable length up to 8 cycles
5	RISC instructions set
6	Von Neumann BUS architecture
7	Four state flags: FZ, FN, FC, *FV
8	Single clock cycle debugging support
9	Control Signals switches for testing
10	Microinstruction live testing and debugging support
11	Microprogramming instruction decoding
12	2's complement view support handled by signal control signal
13	Data BUS live testing

* FV is not supported in the final release.

2.7 The Buses

The myCPU uses only two buses to facilitate data sharing and connectivity between modules and functional blocks: the **Control Bus** and the **Data Bus**, following the **Von Neumann** Bus Architecture, where data and memory addresses are shared on the same bus. In the myCPU design is pointed as the Data Bus. Despite its basic design, it is important for students and learners to take note of this aspect of the myCPU design. Something like a dedicated **Address Bus** is only present in the connection between the MAR and SRAM modules.

The Data Bus supports data lengths up to 16 bits, which would be enough for future releases of the myCPU using the same BUS module board design or would be enough to design custom modules or modules prototypes supporting handle more than 8 bits.

With 16 bits, we can address up to 64Kb, which satisfies the requirements of a very basic CPU like myCPU, and more. Due to the 8-bit nature of the myCPU would be enough use just 8 bits for data exchange, However, this may result a limitation by the need of multiple fetch cycles to manage 16 bits data. The Management of 16 bits of data using an 8 bits data transfer requires at least two clock cycles for each exchange of data between registers and memory, so a 16 bits Data BUS is a very interesting feature for a learning platform like the myCPU and is a essential feature for future releases like the myCPU256 or the myCPU2K.

All of the myCPU control signals are transmitted through the **Control Bus**. The Control Bus has a capacity of 32 lines, via a 32-pin connector, although not all of these lines are used for control signals. The voltage regulators located in each bus module utilize the GND and +5v pins of the Control Bus to provide power to the modules. The clock and reset signals, as well as the status flags signals, also travel through the Control Bus.

The lines included on the control BUS are distributed as indicated in the list below:

Table 2.5: myCPU Control BUS lines distribution

Q	Lines
22	Microinstruction control signals
2	Clock signals: CLK, /CLK
4	Flag signals: FZ, FC, FN, FV, only the Zero, Carry out and Negative are covered in this release
1	Halt signal: HLT
2	Reset signals: RST, /RST
2	Power Supply lines: +5v and GND

* Exist an additional control signal **SQR** which is not part of the control BUS lines.

2.8 The Control Signals of myCPU

The myCPU support up to 23 control signals driving the logic devices of myCPU. 22 of the control signals are managed directly by a dedicated module: the Control Signals Manager (**CSM**), which sets the corresponding digital state for the control signals sent to each logical device in the myCPU. One of them, the SQR signal in internal an handled directly by the Instruction Decoder to control the sequencer cycle.

Knowing the default values of each control signal is essential for a proper understanding of the behavior of the modules. Control signals are the truly key in the interaction between the hardware (logic devices) and the software (instructions). Instructions are composed by microinstructions, and those microinstructions are binary strings composed by control signals acting directly on the electronics of the modules.

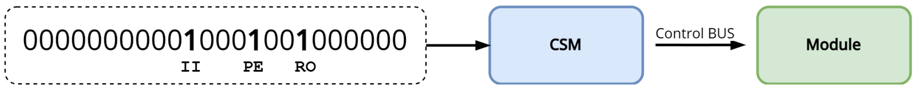

Figure 2.10: myCPU Microinstruction binary sample view

Designing your own custom modules also requires a deep understanding of the control signals, their default values and the actions they should produce on logic components.

The below table shows the full list of the myCPU control signals describing the action produced on the modules.

Table 2.6: myCPU Control Signals

Signal	Description
`CLK, /CLK`	Clock signals
`RST, /RST`	Reset signals, myCPU need both signals High and Low.
`HLT`	Halt signal, used by the clock module to stop execution.
`UN`	Enable display mode for 2's complement.
`AI,AO`	Register A Input/Output enable signals.
`BI,BO`	Register B Input/Output enable signals.
`CI,CO`	Register C Input/Output enable signals.
`II,IO`	Instruction register Input/Output enable signals.
`FI`	Flags register enable load.
`MI`	MAR input.
`RI,RO`	RAM memory Input/output enable signals.
`PE,PI,PO`	PC Count enable and Input/Output enable signals.
`SI,SO`	General purpose signals.
`OI`	Output register Input enable signal.
`EO,SU`	ALU Output and Subtract enable signals
`FC,FV,FZ,FN`	Carry out, Overflow, Zero, Negative Flags
`SQR*`	Sequencer Reset

* The **`SQR`** control signal is connected directly to sequencer instead travel trough control BUS.

Signed display mode support

The myCPU support a 2's complement display mode, its affects basically only to the mode on how the hex/dec decoder display shows binary values, not to the manipulation of a binary value itself. There's a control signal **UN** which enable the 2's complement display mode.

The binary to decimal display shows numbers as unsigned, with a range from **0 to 255**, when the **UN** control signal is LOW. When the **UN** control signal is HIGH, the display shows numbers as signed numbers using 2's complement representation, with a range from **-128 to 127**. This feature is in line with the corresponding feature of the Ben Eater's project.

You should watch the Ben Eater's video about Two's complement arithmetic: https://www.youtube.com/watch?v=4qH4unVtJkE

2.9 The Instruction Cycle

The myCPU instruction execution is performed through a Instruction cycle that consists of a fetch cycle followed by an execution cycle. The myCPU is a single fetch execution cycle, it means that uses only one fetch cycle to gets the instruction and the argument of the instruction. It is possible thanks to an Instruction Register with a 4+4 bits design, most significative 4 bits for the OpCode of the instruction and less significative 4 bits for the argument. So in a single 8 bits data exchange operation with memory retrieve the instruction and the argument, being the most simplified way to view and understand the execution cycle of a CPU in real time.

Due to the simplicity of the instruction set and program limitations, the myCPU execution cycle does not require any additional specialized phases. The length of the Instruction cycle reach to 8 states or steps. Each step of the instruction cycle requires one clock cycle to complete and executes a single microinstruction.

The myCPU support a microinstruction length up to 24 bits, involving 23 control signals and 22 of them shared through the Control BUS.

The myCPU is a basic approximation of an early CPU with limited capabilities compared to real CPUs. Unlike modern CPUs that can perform multiple instruction cycles simultaneously through pipelining, myCPU can only execute one instruction cycle per instruction. The design of myCPU supports a variable instruction cycle length through the ability to reset the sequencer at a specific step and initiate a new instruction cycle.

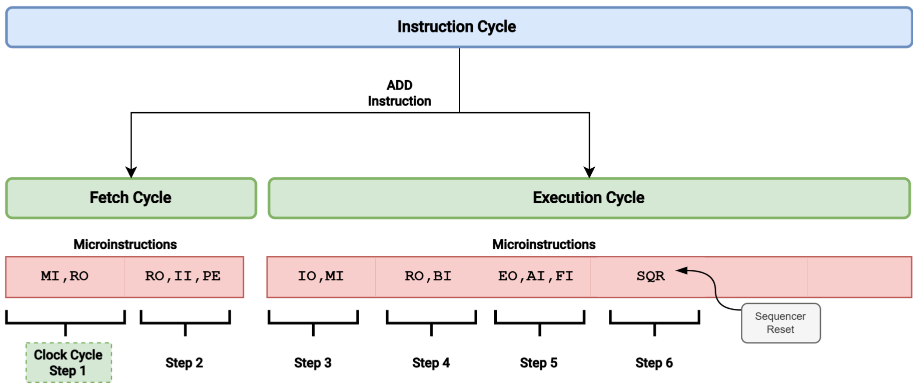

Figure 2.11: myCPU Instruction Cycle diagram

Due to the microinstruction based nature of the myCPU, the instruction architecture of myCPU could be described as **RISC (Reduced Instruction Set Computer)** instead of a CISC (Complex Instruction Set Computer). Each instruction can consume from 2 (the minimal instruction cycle length) up to 8 cycle steps, and probably, it could be upgraded up to 16 steps in next releases. Remember that in the myCPU design each step or state is equivalent to a single clock cycle.

2.10 The Flags

The myCPU design include up to 4 status flags, but only 3 of them are used in the first release. Mainly for the limitations of the ALU module, which is based on simple adders. myCPU uses only: The Carry Out flag **FC**, which correspond with the last carry bit of the adders cascade, the Zero flag **FZ** which is calculated using logic gates and the Negative flag **FN** which correspond to the most significant bit of the ALU value or sign bit in 2's complement value representation. The Overflow flag **FV** is not used, in the first release, although is supported by the flags register.

👁 *Only ALU related flags are supported in the first release.*

The corresponding flags signals are listed below:

Table 2.7: myCPU State Flags

Flag	Description
`FZ`	Zero flag
`FN`	Negative flag
`FC`	Carry out flag
`*FV`	Overflow flag

* Not used in the first release.

Flags are an essential part of the decoding process for the **Conditional Jump Instructions**. The flags combination define the status of the myCPU after each executed instruction and that flags state can enable the execution of a conditional jump instruction based on that flags state. In the myCPU first release, only conditional jump instructions for the state of 3 individual flags: FC, FC and FN, are encoded in the initial **ISA** (Instruction Set Architecture).

2.11 The Instruction processor

The execution process of an instruction in the myCPU design is based on **Microprogramming**.

In Microprogramming an instruction is divided into small pieces called **Microinstructions** which are executed sequentially. Microinstructions are binary strings of control signals exposed to CPU logic components through the CSM using the Control BUS, modifying the state of the logic components of a CPU and the set of microinstructions are called **Microcode**. The microcode is encoded into a ROM or into an EEProm memory providing a easy way to modify the behavior of an instruction or to design new instructions for the ISA (Instruction Set) of the CPU.

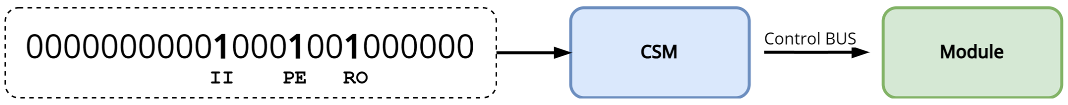

Figure 2.12: myCPU Microinstruction Flow

In a Microprogramming based CPU,like myCPU, the Instruction Processor is the logic block responsible of decoding an instruction, from the ROM or EEProm, and executing its associated microcode. The Instruction Processor involves several logic elements as you can see in the next list:

Table 2.8: myCPU Instruction Processor functional modules

Module	Action
`Instruction Register`	Keep the instruction OpCode
`Sequencer`	Manage the sequence of microinstruction execution
`Flags Register`	Keep the Flags state
`Instruction Decoder`	Decodes the microinstructions for an instruction
`CSM (Control Signals Manager)`	Manage binary strings of control signals

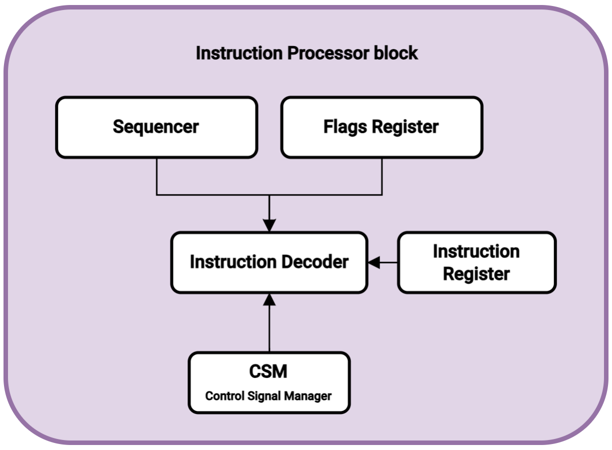

Figure 2.13: myCPU Instruction Processor block diagram

In microprogrammning, Each instruction is composed by a set of microinstructions executed in a sequence. The Sequencer manage the sequence of execution, each microinstruction is executed during a sequencer step synchronized with the fall state of the clock signal. Because of that, some control signals of the microinstruction cause immediate changes in the CPU state, while others will produce changes during the next rise state of the clock signal.

In addition, certain types of instructions like conditional jumps are dependent of the flags state and its microinstructions have to be encoded according to a specific state of flags

Therefore, Microprogramming is the process to encode the microinstruction corresponding to each step of the instruction cycle for an instruction OpCode, depending or not of a specific state of flags. Encoding the corresponding full set of microinstructions for every instruction of the ISA (instructon set).

In the myCPU the decoding of an instruction is based on a memory address for the memory in which the microcode is encoded, an AT28C64 EEProm memory for the myCPU. This address is a 13 bit address generated using the OpCode of the instruction in execution, the flags state stored in the flags register, the sequencer step and 2 bits for the memory unit selection. The bit order is saw in the picture bellow.

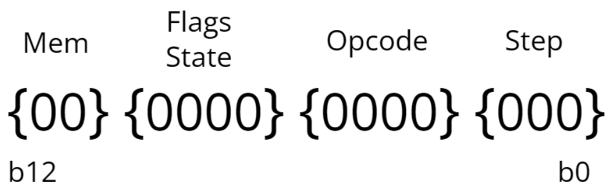

Figure 2.14: myCPU Microinstruction decoding address parts order

The **Sequencer** determine the current instruction step or state of the instruction cycle, in the myCPU the sequence is up to 8 steps, so is 3 bits length the address part for the step (multiplexed).

The **Instruction Register** holds the current **Opcode** of the instruction, in myCPU the length of for the opcode is 4 bits because only use the 4 most significative bits of the register for the opcode, the less significative 4 bits is for the argument.

The **Flags Register**, store the state of flags on the myCPU, in myCPU only 4 flags available so the length for this address part is 4 bits (not multiplexed).

So, the decoded microinstruction to execute will be determined by:

Table 2.9: myCPU Microinstruction encoding address parts

Order	Part	Length
1	`Sequencer step`	3 bits (up to 8 steps)
2	`Opcode`	4 bits (up to 16 opcodes)
3	`State flags`	4 bits (1 bit for each flag)
4	`Memory selection*`	2 bits (up to 3 memory modules)
	`Total` =	13 bits

* Up to 3 memories x 8 bits data length = 24 bits microinstruction length

The **Memory Selection** bits determine which memory unit will be used to encode or decode each byte part of a microinstruction. In the myCPU design, a microinstruction is composed of 24 bits or 3 bytes, which are handled by three memory units.

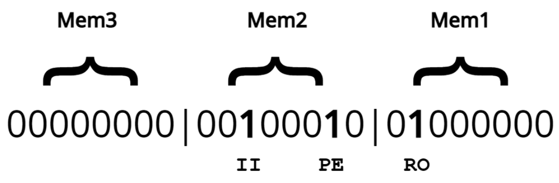

Figure 2.15: myCPU Microinstruction memory distribution

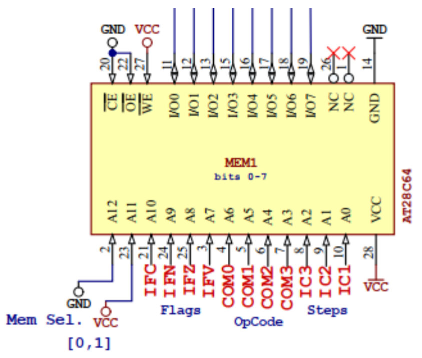

Figure 2.16: myCPU Microinstruction decoding address schematic

The decoded microinstruction is passed to the CSM (control signals manager), which is responsible to send the appropriate control signal values to the logic components of the myCPU.

2.12 Testing and Debugging the myCPU

The myCPU design has support for debugging by advance the execution flow step by step at a clock cycle level and viewing the state of the logic components of the CPU in real time. We can see what happens during the execution of a specific microinstruction, including the capability of testing microinstructions using the CMS module. myCPU has capabilities for input test values to check module behavior or include test values in the debug process of a microinstruction. In myCPU we could test and debug an entire instruction by testing and debugging each microinstruction that it is composed of. myCPU include a set of DIP switches for testing purposes.

Table 2.10: myCPU Test switches

Num	Description
1	`BUS Manager switches`
2	`Control Signals Manager (CSM) switches`

Testing modules

In myCPU we can test the electronic and logic behavior of modules by introducing testing values through the BUS Manager module and set specific control signals through the CSM module. This is possible thanks to the debug mode of the clock module that enables the myCPU running step by step.

The **BUS manager** has two DIP switches of 8 bits to put a 16 bits test value on the Data BUS by setting up a specific value on the switches and pressing the TEST button.

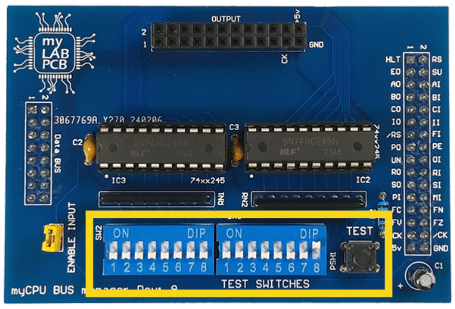

Figure 2.17: myCPU BUS Manager DIP switches

The **CSM module** has two DIP switches of 12 bits, to set a test value for each individual control signal. By using the CSM switches, is possible to test individual module actions, or to configure a real microinstruction by combining multiple switches at a time.

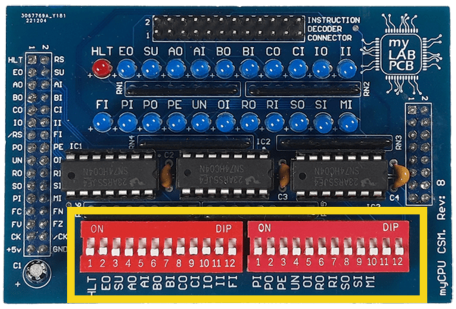
HLT EO SU AO AI BO BI CO CI IO II
FI PI PO PE UN OI RO RI SO SI MI
INSTRUCTION DECODER CONNECTOR
myCPU CSM

Figure 2.18: myCPU CSM DIP switches

👁 *Combining the BUS Manager and CSM switches you can reproduce a true runtime execution condition statically.*

Microinstruction debugging and testing

The myCPU Clock module support a debug mode that enable running the myCPU in a one clock step at a time using a push button. This debug mode facilitates the debugging by the view of the logic state of the modules statically in real time during the execution flow. This feature of view statically the state of the modules allows to check digital current values of logic elements using a digital logic tester.

We can build and test an entire microinstruction by removing the Instruction Decoder connector and using the CSM module test switches, checking what happens to the state of logic components at LOW and HIGH part of the clock cycle.

In myCPU, a microinstruction is executed when a control signals string is placed on the control BUS. This take place when the sequencer change to a new step and this happens during the fall state of the clock signal so the microinstruction is executed at the LOW state of the clock cycle. When we are in DEBUG mode we can transit to the HIGH state of the clock cycle by pushing the clock button.

When a microinstruction is placed on the control BUS, at the fall state of the clock signal, some changes happen modifying the state of some logic components of the myCPU, and the other related changes happen in the immediate rise state of the clock signal.

You can see more detailed explanation about the clock module in: 12 - The Clock module, p.95

2.13 Programming the myCPU

The myCPU design includes a unique SRAM memory module acting as CPU's internal memory and as program memory.

Programming the myCPU involves the **MAR** and **SRAM** memory modules, the MAR module store the address for the RAM memory in read and writing operations and is only 4 bits length. The SRAM module is a very basic discrete logic memory module with only capacity to 16 bytes. The MAR is connected to the SRAM module through an **Address Connector**, to setup the address of the SRAM memory module in both, RUN or PROG mode, because myCPU has not a dedicated Address BUS.

The myCPU Programming is performing by the introduction of a program into the program memory. This operation is realized manually using the DIP switches provided by the MAR and the SRAM memory modules. The MAR module includes in its design a switch to change the operation mode from "**RUN**" to "**TEST**" and a **4 bits DIP switch** to introduce a memory address. The address set at the MAR DIP switch will be exposed through the Address Connector whenever the MAR is in TEST mode. the

SRAM memory module includes also a switch to enable the "**PROG**" mode and a **8 bits DIP switch** to introduce the data to store into the memory whenever its mode switch is in the PROG mode.

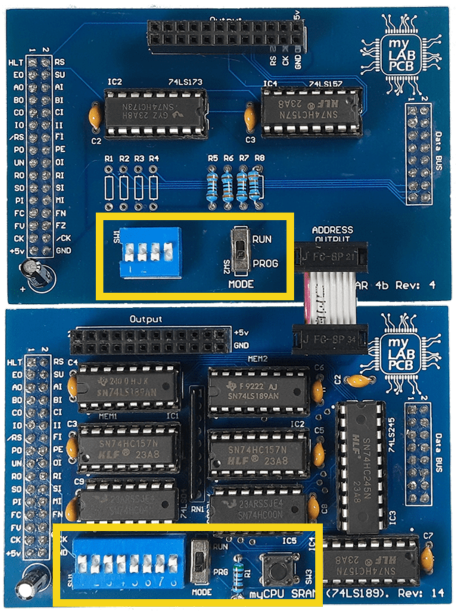

Figure 2.19: myCPU MAR and SRAM DIP switches

2.14 Limitations of myCPU

The myCPU design has some limitations due two reasons. The first reason is that the myCPU project is based on the Ben Eater's breadboard computer project and for taking advantage of his fantastic video lectures it was necessary to keep aligned the myCPU project with his project in some way. The second one is to provide a soft approach to a CPU design for beginners, inexperienced electronics hobbyists and students acting as an introductory project for a CPU building and programming.

👁 *The myCPU design try to keep aligned with the Ben Eater's breadboard computer project and take advantage of his great video lectures.*

Despite the myCPU limitations is a valuable learning and development platform and some of these limitations will be reduced or removed in the next releases. Some of these limitations are listed bellow:

Table 2.11: myCPU Limitations

Description
Basic ALU with support for adding and subtract using 2's complement
SRAM 16 bytes
Memory Address length 4 bit
4 bit instruction Opcode, supporting up to only 16 instructions
No input register or dedicated input modules
Only one output port with 8-bit support
Manual programming using switches

The **ALU** was designed using just 4-bit adders with the consequent limitation to adding operations and subtracting using the 2´s complement approach. But only the B operand in converted to 2's complement so the sustracting is limited to A + (-B) form.

The **SRAM** module is based on the **74xx189** which is a very limited SRAM chip of 16 x 4 bits including 2 of them. The limited available memory, to 16 bytes, prevents the possibility to write more then simple programs because are limited to 16 instructions. Due to the limitation of SRAM, the **Memory Address Register (MAR)** has a limited length to 4 bits address. Does not make sense make it bigger. The MAR and SRAM limit is a hard inconvenient but remember that myCPU is a learning platform and you can explore the basics of a CPU and a programmable computer using just 12 to 14 instructions including conditional jumps.

The **Instruction Register** has a limitation to 4 bits for the instruction opcode and 4 bits for the instruction argument. Limiting the possible maximum length of the ISA (Instruction Set) to 16 instructions and limit the argument to $(0F)_{16}$ or $(15)_{10}$ as maximum value.

In this first release, there is no input register module. However, it uses optionally the BUS Manager modules as input port with the "**SI**" control signal to expose the value at the BUS Manager DIP switches to the Data BUS through the **LDP** instruction.

👁 *keep in mind that the myCPU is not just a device executing a machine code program, its more than that, myCPU is a platform which let you to show what happens on a digital system, like a CPU, during the execution of a program.*

2.15 Electronic improvements of myCPU

The myCPU project, because of its PCB based design nature, includes several electronic improvements respect the original Ben Eater project because are difficult to implement on breadboards. Some of them are listed bellow:

Table 2.12: myCPU Electronic improvements

Description
+5V Power Supply of each pair of modules through individual voltage regulators
Separated displays from modules main board through output connectors
Limited resistors for LEDs and 7 segment displays
Pull-down resistors on IC inputs when needed
Pull-up resistors when needed
Pull-up/down unused logic inputs when needed
100nF Decoupling capacitors on all ICs
4-Layer PCBs
Independent PWR, GND layers separated from signal layers
A 10uF power filter capacitor in each module

2.16 The myCPU Roadmap

The main and final goal of the myCPU initiative is to desing a totally capable and functional CPU emulating a 6502 or a Z80. Including support up to 64K of SRAM using the open architecture and modular platform paradigms using only discrete logic. To reach this goal the project will navigate through a set of steps to evolve the myCPU from an early stage with the first release of myCPU16, following with the myCPU256 and myCPU2K, to the most advanced releases the myCPU+ 32/64K with a BASIC interpreter and a C compiler, close a tiny computer.

Since the finishing of this book, changes on the roadmap schedule could have been produce. Keep updated with the roadmap at the project site:
https://mycpu.mylabpcb.com/pages/en/mycpu/introduction#mycpu-roadmap

Below picture shows, graphically, the roadmap of the myCPU project:

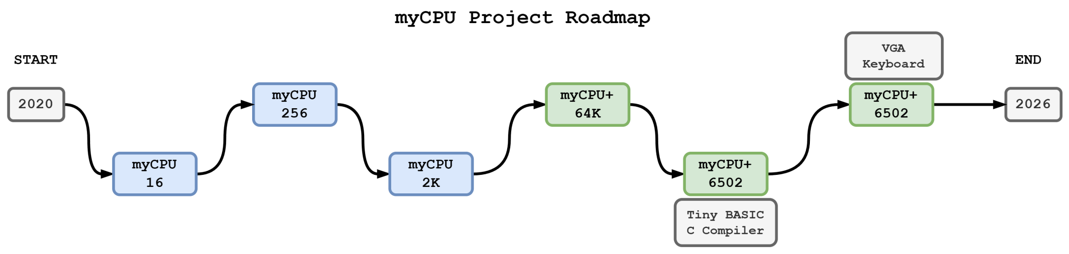

Figure 2.20: myCPU Roadmap

2.16.1 myCPU16

The first release of myCPU planned to the summer of 2024. Limited to only 16 bytes of memory enabling to execute very small programs up to 16 bytes length and capable only to support an instruction set up to 16 instructions.

2.16.2 myCPU256

This improved version of myCPU16 will includes several notable changes. Firstly, an 8 bits Memory Address Register (MAR) capable of addressing up to 256 bytes of SRAM. Secondly, the number of supported instructions will be increased to a total of 64, a good number to experimenting with the ISA. Additionally, this version will includes a new SRAM module up to 2K, although for this release only 256 bytes can be used due to the MAR address length limit. Lastly, an updated Instruction Decoder module, which can support the expanded instruction set and will be based on the **W27C010** EEprom memory.

Included features in the myCPU 256 are pointed below:

Table 2.13: myCPU256 Feature changes

Description
MAR 8 bits, capable to addressing 256 bytes of memory
SRAM module up to 2K, using only 256 bytes due to MAR limitation
6 bits for Opcodes supporting up to 64 instructions
Variable Instruction Execution Cycle up to 8 steps
EEProm Boot module to load programs, up to 256 bytes.
Instruction Register with support up to an 8 bit Opcode
Instruction Decoder with decoding support up to 14 bits based on W27C010
External SRAM programming using Arduino
Separated programming module

2.16.3 myCPU2K

This is a small evolution from the myCPU256. It will includes: a new 16 bits MAR module capable to addressing up to 2K bytes and more, a program counter up to 16 bits capable to count up to 2K and more. In this release the Flags Register and the Sequencer would be separated in 2 independent modules, to allow the design of a more advanced Sequencer which support up to 16 steps per execution cycle.

Included features in the myCPU2K are pointed below:

Table 2.14: myCPU2K Feature changes

Description
MAR 16 bits, capable to addressing up to 64K bytes of memory
SRAM module up to 2K, based on 6116 CMOS SRAM family
Program Counter 16 bits, up to 64K count
6 bits for Opcodes supporting up to 64 instructions
Variable Instruction Execution Cycle up to 16 steps
EEProm Boot module to load programs, up to 2K
Instruction Register with support up to an 8 bit Opcode
Instruction Decoder with decoding support up to 14 bits based on W27C010
External SRAM programming using Arduino
Separated programming module

2.16.4 myCPU+ 64K

This a very strong evolution regarding the myCPU releases: 16, 256 or 2K. One of the most radical advance is the change of the microprogramming paradigm, coming from a horizontal and static control signal model to a vertical paradigm with a dynamic control signals model. This model of microprogramming enable handle a large number and more complex logic devices with a reduced set of control signals.

Other advantages of the myCPU+ will be an advanced **ALU** module based on the **74xx181**, a special logic module for bit manipulation, shift and rotate bytes, an 8 bits stack pointer module, the support of more extended instruction decoding up to 17 bits which will get a capability to design and manage an ISA of 256 instructions, or a full SRAM module up to 32/64K based on a 62256 SRAM family

A full list of the feature changes of the myCPU+ is shown below:

Table 2.15: myCPU+ 64K Feature changes

Description
Vertical microprogramming supporting an hybrid control signals model
Advanced ALU based on 74181
ALU Shifter, Rotator and Bit manipulation module
MAR 16 bits, capable to addressing up to 64K bytes of memory
SRAM module up to 64K based on 62256 family
Program Counter 16 bits
8 bits for Opcodes supporting up to 256 instructions
Variable Instruction Cycle up to 16 steps
Up to 8 flags
8 bits input registers
Stack Pointer 8 or 16 bits
EEProm Boot module to load programs, up to 64K bytes, into SRAM at start
Instruction Register with support up to an 8 bit Opcode
Instruction Decoder with decoding support up to 17 bits based on WD27020
Support for Branches, Subroutines and Interrupts
Support RESUME for HLT states