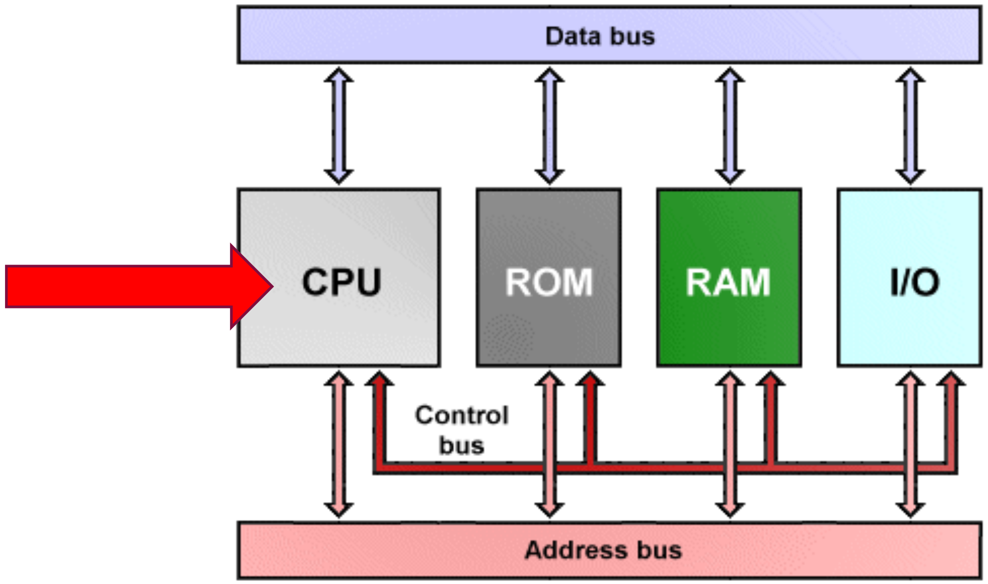
MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

THE CPU

- **What is the CPU?**

  - The **Central Processing Unit** is the brain of the computer.
    - It performs multiple logical and arithmetic operations on instructions and data.
    - It controls the behavior of the different system components.



THE CPU - 2

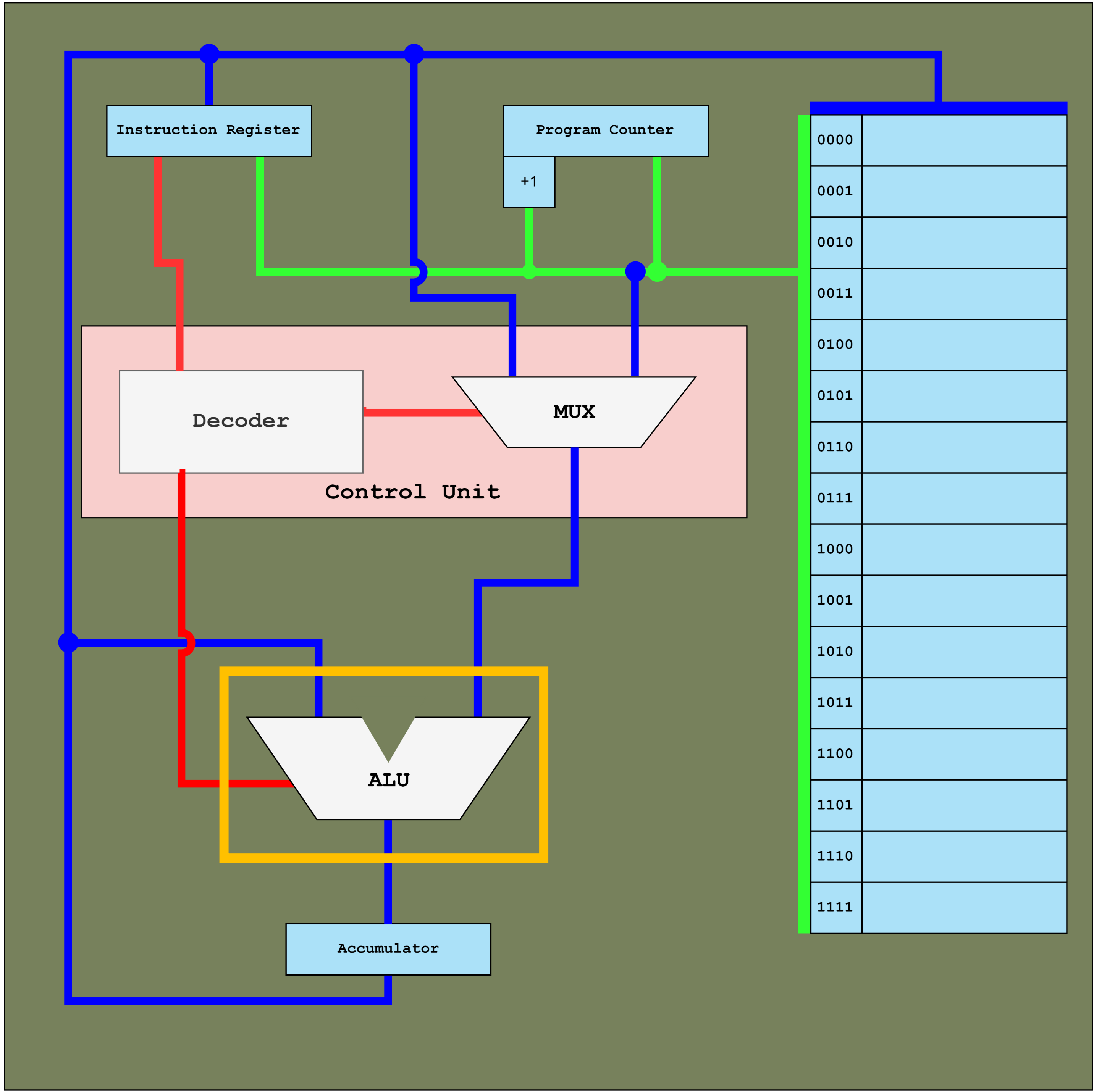
- **What is the CPU made of?**

  The CPU is made out of several components that work together.

The Arithmetic/Logic Unit (ALU):

Is the number processor component of the CPU**. It performs logical and arithmetic operations on operands and addresses.** It is made out of different complex circuits such as **adders, subtractors, comparators, and other logic gates that perform different operations.**

The ALU **receives data from the memory using the data bus (blue)** and performs operations received from the control unit.



THE CPU - 2

- **What is the CPU made of?**

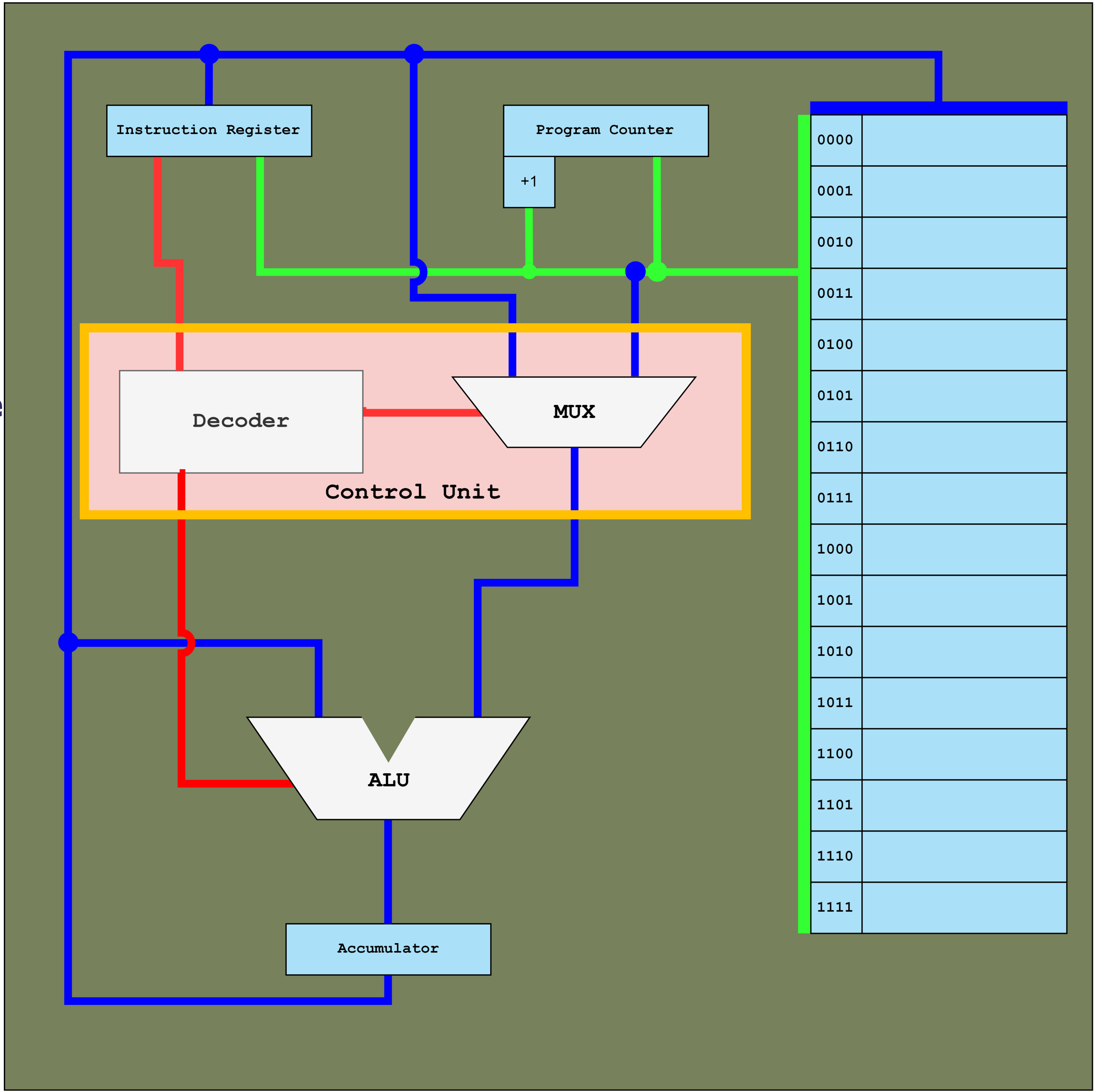
  The CPU is made out of several components that work together.

The Control Unit (CU):

The control unit does what the name suggests, **it controls the different components of the CPU**. The control unit is responsible for **decoding the instruction into commands that are passed to the different components** using the control bus **(red).**

The control unit itself is made of different components. It is made out of decoders, multiplexers, and some cases encoders.

The control unit is also responsible of passing data to the ALU for performing operations on them.



THE CPU - 2

- **What is the CPU made of?**

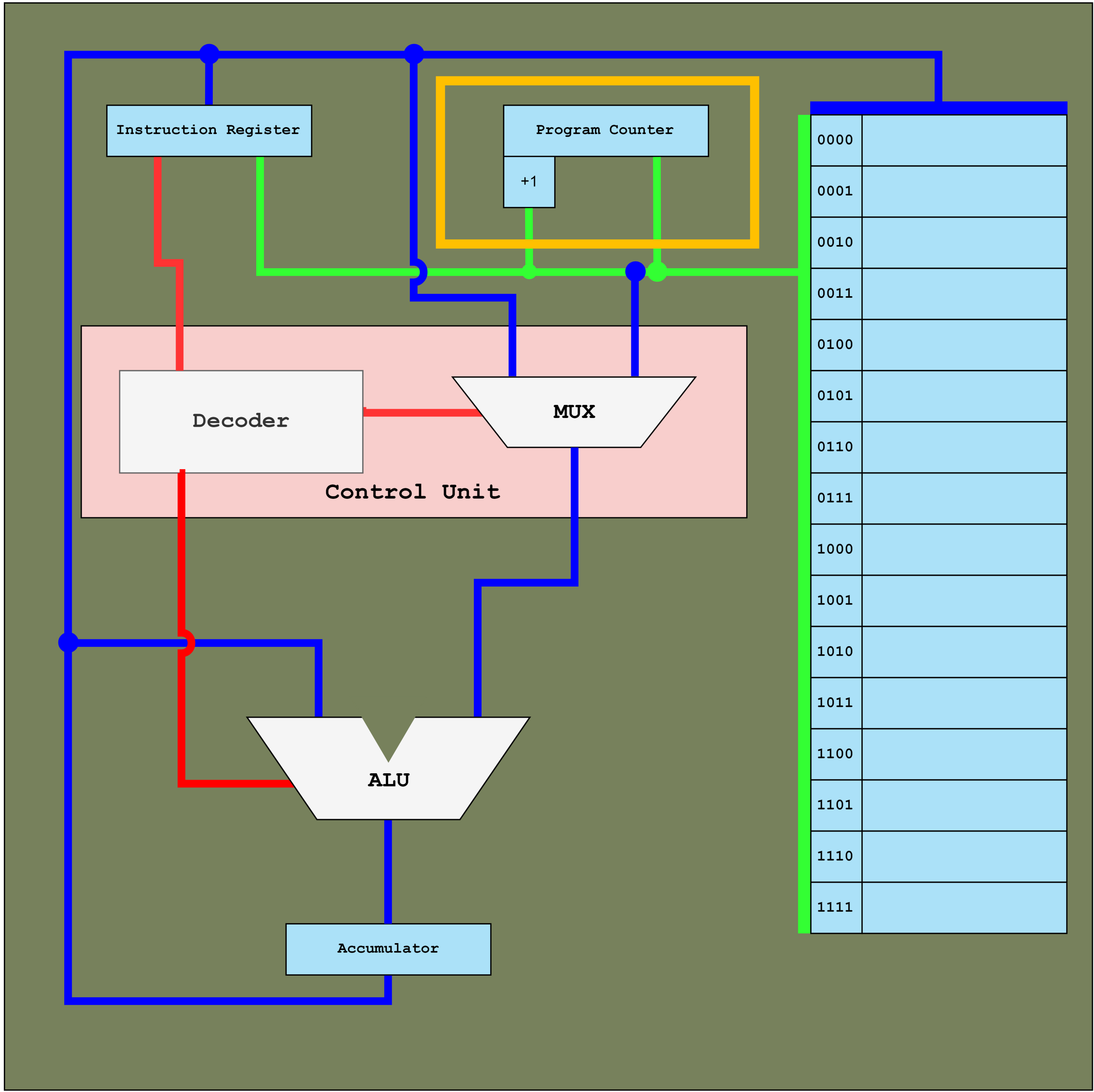
  The CPU is made out of several components that work together.

The Program Counter Register (PC):

The program counter **keeps track of the instruction address that is to be executed next.** It is used to specify which memory location holds the instruction that the CPU must execute next.

The program counter is **incremented after each instruction is finished executing** allowing the next instruction to be fetched from memory.

The program counter **can also be loaded with a number allowing execution jumps**



THE CPU - 2

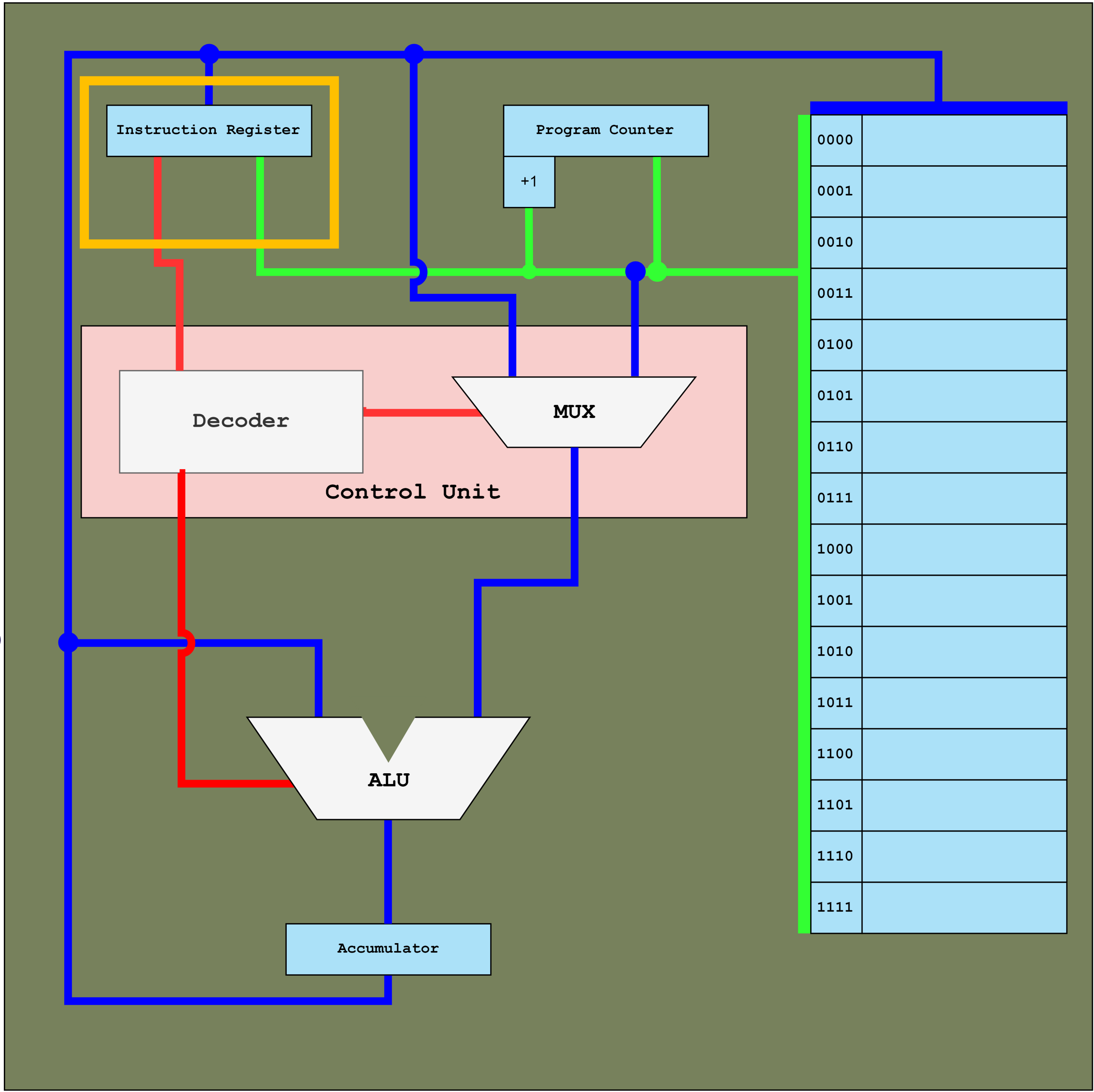
- **What is the CPU made of?**

  The CPU is made out of several components that work together.

The Instruction Register (IR):

The instruction register **holds the current instruction to be executed.** Once the PC is set, the instruction register asks the memory to load the instruction at the location specified by the PC. It also **signals the PC to increment its value** once the instruction is executed

The memory in turn, **loads the instruction to the address bus** (**green**) and the instruction register holds to the data. It then **sends this instruction to the control unit when the control unit asks for it.**



THE CPU - 2

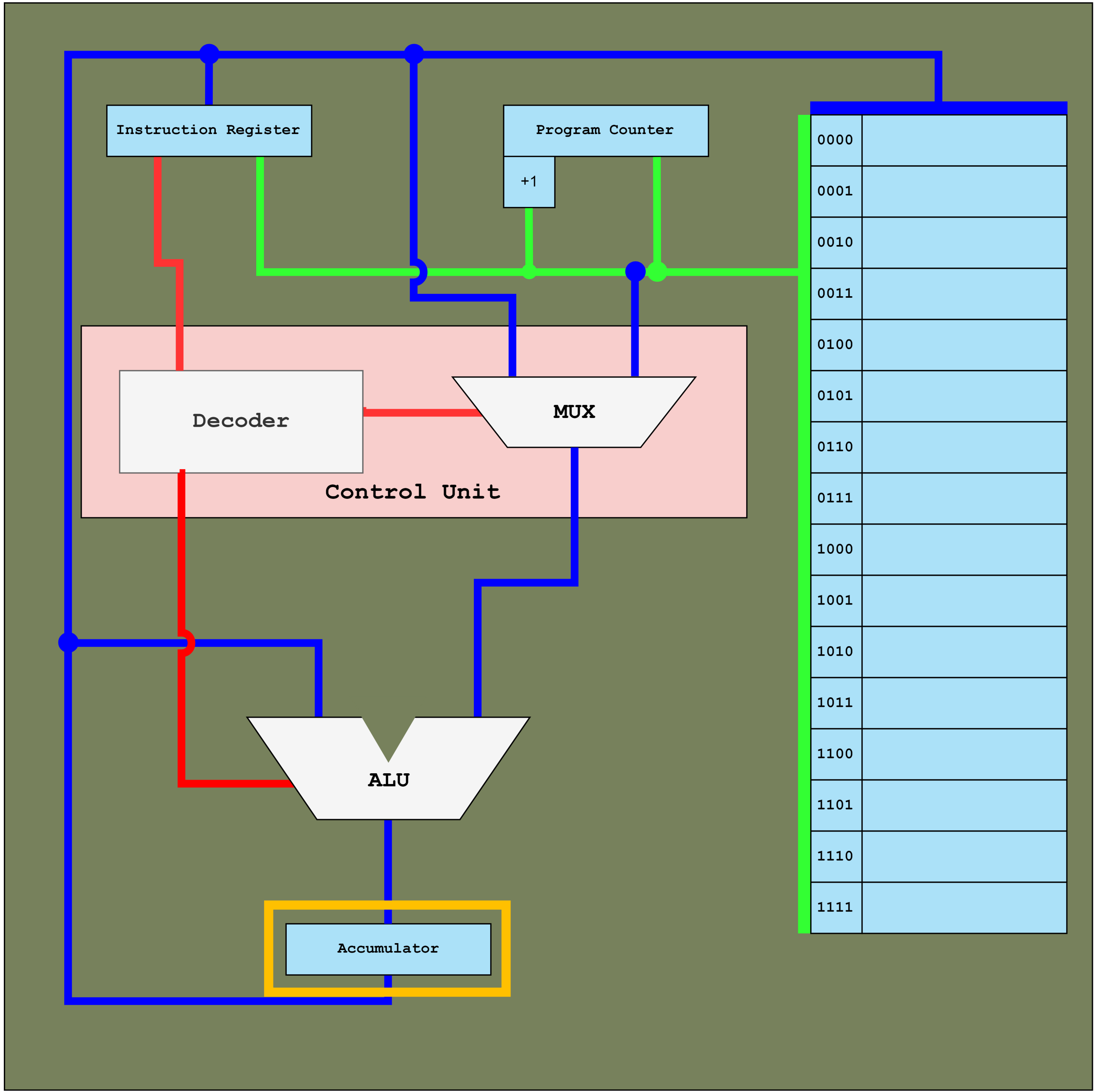
- **What is the CPU made of?**

  The CPU is made out of several components that work together.

The Accumulator (Accu):

The accumulator **is a designated register to hold the results from the ALU operation.** The Accumulator can also be a collection of registers – known as **the register file**.

The Accumulator is connected to the ALU both **as input and output using the data bus. This allows data stored in the accumulator to be used by the ALU again in the next instruction execution.** It is also connected to **the memory allowing storing the data inside it to the memory when needed**.



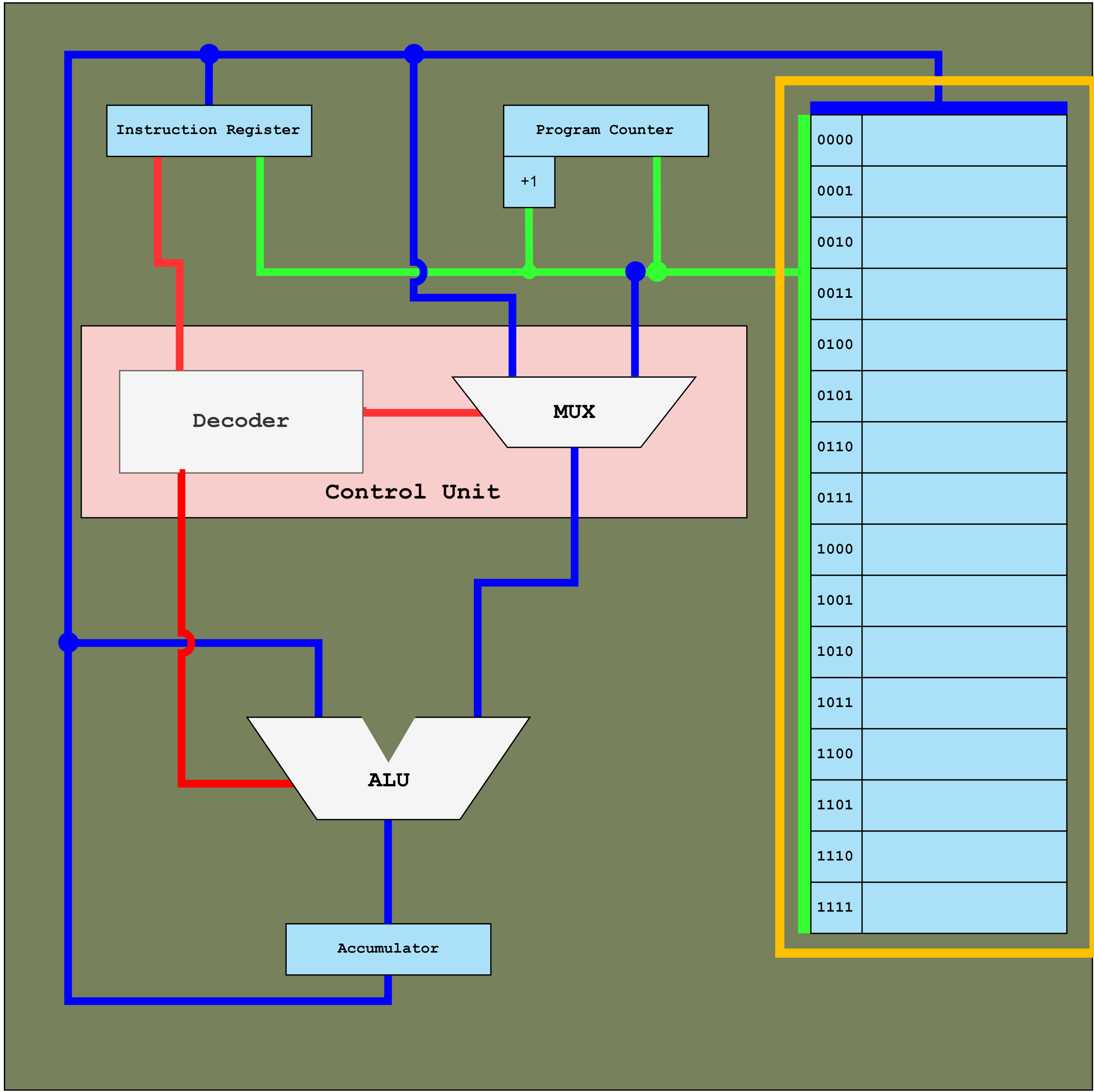
THE CPU - 2

- **What is the CPU made of?**

  The CPU is made out of several components that work together.

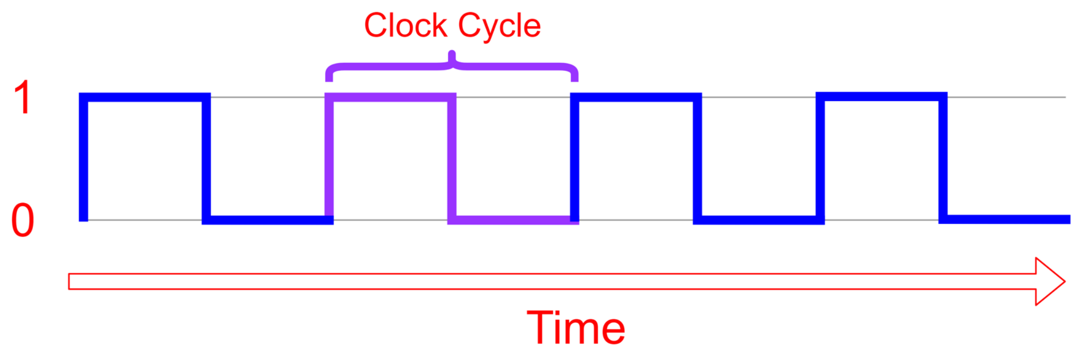
The memory (mem):

It does what the name suggests, it remembers stuff. **It is a storage location for data to be processed by the CPU**. It holds addresses and immediate values. The size the memory and its type are essential factors in the CPU's performance. For almost every operation that the CPU completes, the memory is accessed, whether it was for loading data from memory or to store data to memory.



THE CPU - 2

- **The Clock:**
  - The CPU clock is what determines **when** the CPU does what.
  - The clock is a an **oscillator that produces a symmetrical square wave digital signal that indicates time (cycles) and is used to synchronizing the components of the CPU.**
  - The clock frequency is measured in Gigahertz (GHz).
  - A clock cycle is a single period of clock signal. The **time the square signal takes between reaching the rising edge of the clock (1) and then reach the falling edge of the clock (0).**
  - A 1GHz CPU, will produce 1 billion clock cycles per second.



THE CPU - 3

- **How does the CPU work:**

  - The CPU **interprets instructions and executes them based on the design of its internal components.**

  - It is designed on predefined **Instruction Set Architecture (ISA).**
    - The ISA defines how the CPU internal components behave.
    - The ISA defines how instructions are formatted and what bits in the instruction mean what!

THE CPU - 3

- **What is a CPU instruction:**
  - The **instruction** is a series of **bits that describe what <u>operations</u> (op code) must be executed on what <u>operands</u> (address/immediate) and the <u>type of those operands</u> (id).

- The **instruction** is made out of different parts, most notably:
  - The **Op Code:**
    - This parts defines what operation must be performed by the CPU
  - The **operand:**
    - This part contains the **memory address** or the **immediate value** that the operation will be performed on.
  - The **data identifier:**
    - This part identifies the **data type** of the operand.
  - The **registers**:
    - This part specifies what registers the CPU will use to execute the operation.

THE DUMMY CPU - EXAMPLE

- For this course, we will use an 8-bit dummy CPU that we will use in the following classes.
- 8-bit means that the instruction is 8-bits long. Eg: 00100101
  - Lets start with the **op code**:
    - In this example we will use **3 bits** to specify the **operation** that the instruction must execute. The red part is the op code **001**00101
      - 3 bits allows us to have $2^3 = 8$ combinations of bits: 000,001,010,011, …
      - This means that with using 3 bits to specify the op code, we can have 8 operations for the instruction.
      - The CPU needs to move stuff around, which means that we need instructions telling the CPU where to put and get things. A load and store instructions are useful here.

Op code	operation	Function
001	LD	Load to accum
010	ST	Store to memory

THE DUMMY CPU - EXAMPLE

- Now that we have operation to move things around, lets do some basic arithmetic operations such as Add and Subtract.

Op code	operation	Function
001	LD	Load to accum
010	ST	Store to memory
011	ADD	Add value to value in accum
100	SUB	Subtract value from value in accum

- Now that we have these operations, we need some flow control, these operations allows the CPU to make decisions without the help of the control unit. Namely, the CPU needs to know if two values are equal (EQ) and then use that to jump (JP) to new instruction.

Op code	operation	Function
101	EQ	Checks in value is equal to value in accum
110	JP	Set value in PC to value

THE DUMMY CPU - EXAMPLE

- Now that we have some good operations and functions, we need to halt execution (HE) the CPU. Without a direct instruction telling the CPU to stop, it will go forever and ever and ever without stopping (This only means that the PC counter will keep incrementing until it overflows)

Op code	operation	Function
001	LD	Load to accum
010	ST	Store to memory
011	ADD	Add value to value in accum
100	SUB	Subtract value from value in accum
101	EQ	Checks in value is equal to value in accum
110	JP	Set value in PC to value
111	HE	Halt Execution

- Now that we have the op code we need, lets see what other things we can do.

THE DUMMY CPU - EXAMPLE

- The next thing we need to consider is telling the CPU what data to operate on. We need to specify where the data is found in the instruction.
- Since we used the first three bits for the ops code, we will use the last 4 bits for the **operand**. The green part is the **operand** 0010**0101**
- How many values can we store in 4 bits?

THE DUMMY CPU - EXAMPLE

- The next thing we need to consider is telling the CPU what data to operate on. We need to specify where the data is found in the instruction.
- Since we used the first three bits for the ops code, we will use the last 4 bits for the **operand**. The green part is the **operand** 0010**0101**
- How many values can we store in 4 bits?
  - We can store $2^4 = 16$ combinations of bits. What is the largest value we can store in 4 bits?

THE DUMMY CPU - EXAMPLE

- The next thing we need to consider is telling the CPU what data to operate on. We need to specify where the data is found in the instruction.
- Since we used the first three bits for the ops code, we will use the last 4 bits for the **operand**. The green part is the **operand** 0010**0101**
- How many values can we store in 4 bits?
  - We can store $\mathbf{2^4 = 16}$ combinations of bits.
- What is the largest value we can store in 4 bits?

THE DUMMY CPU - EXAMPLE

- The next thing we need to consider is telling the CPU what data to operate on. We need to specify where the data is found in the instruction.
- Since we used the first three bits for the ops code, we will use the last 4 bits for the **operand**. The green part is the **operand** 0010**0101**
- How many values can we store in 4 bits?
  - We can store $\mathbf{2^4 = 16}$ combinations of bits.
- What is the largest value we can store in 4 bits?
  - $\mathbf{1111_2}$ which equals $\mathbf{15_{10}}$ is the largest value.
- The number of bits we decide to reserve for the operand also **dictates the size of the memory**. Since the **operand will be used to refer to memory locations** (addresses), the addresses need to be within the operands range.
- In this example, there will be two types of operands, addresses and immediate values (integers)

THE DUMMY CPU - EXAMPLE

- Now that we know where the operands can be found in the instruction, we also need to tell the CPU the type of operand it is performing operation on. This has huge effect on CPU accuracy.
- We will use the last bit to specify operand type or also called id. Id is in yellow 00100101
- The id type has two possible combinations (1 and 0).
  - Some operations, require an **immediate value** to be performed, such as LD, ADD, SUB, EQ. But other operations such as ST and JP will require an address. So for this example, we will be using the id bit to specify what type of data we will be operating on.
    - 0 will be reserved for address
    - 1 will be reserved for immediate values

THE DUMMY CPU - EXAMPLE

- Now that we specified what the instruction look like, we will end us with the following table.

Op code	operation	Function	Use	binary
001	LD	Load to accum	LD # 5	00110101
010	ST	Store to memory	ST $ 2	01000010
011	ADD	Add value to value in accum	ADD # 10	01111010
100	SUB	Subtract value from value in accum	SUB # 4	10010100
101	EQ	Checks in value is equal to value in accum, if true, skip next instruction	EQ # 5	10110101
110	JP	Set value in PC to value	JP $ 8	11001000
111	HE	Halt Execution	HE	11100000

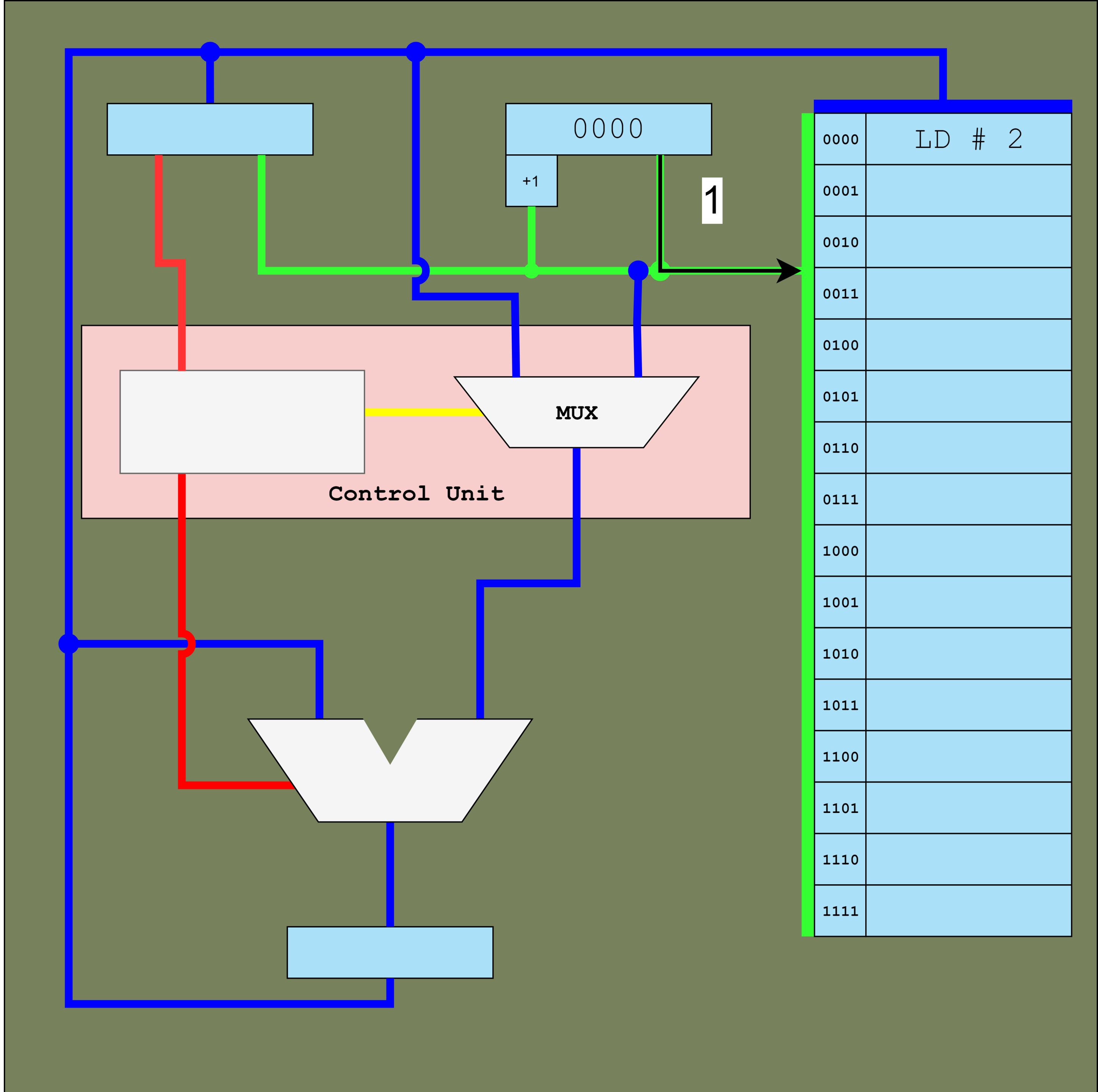
- # donates id 1, meaning that the # symbol is for immediate values
- $ donates id 0, meaning that the $ symbol is for addresses

THE DUMMY CPU - EXAMPLE

- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**

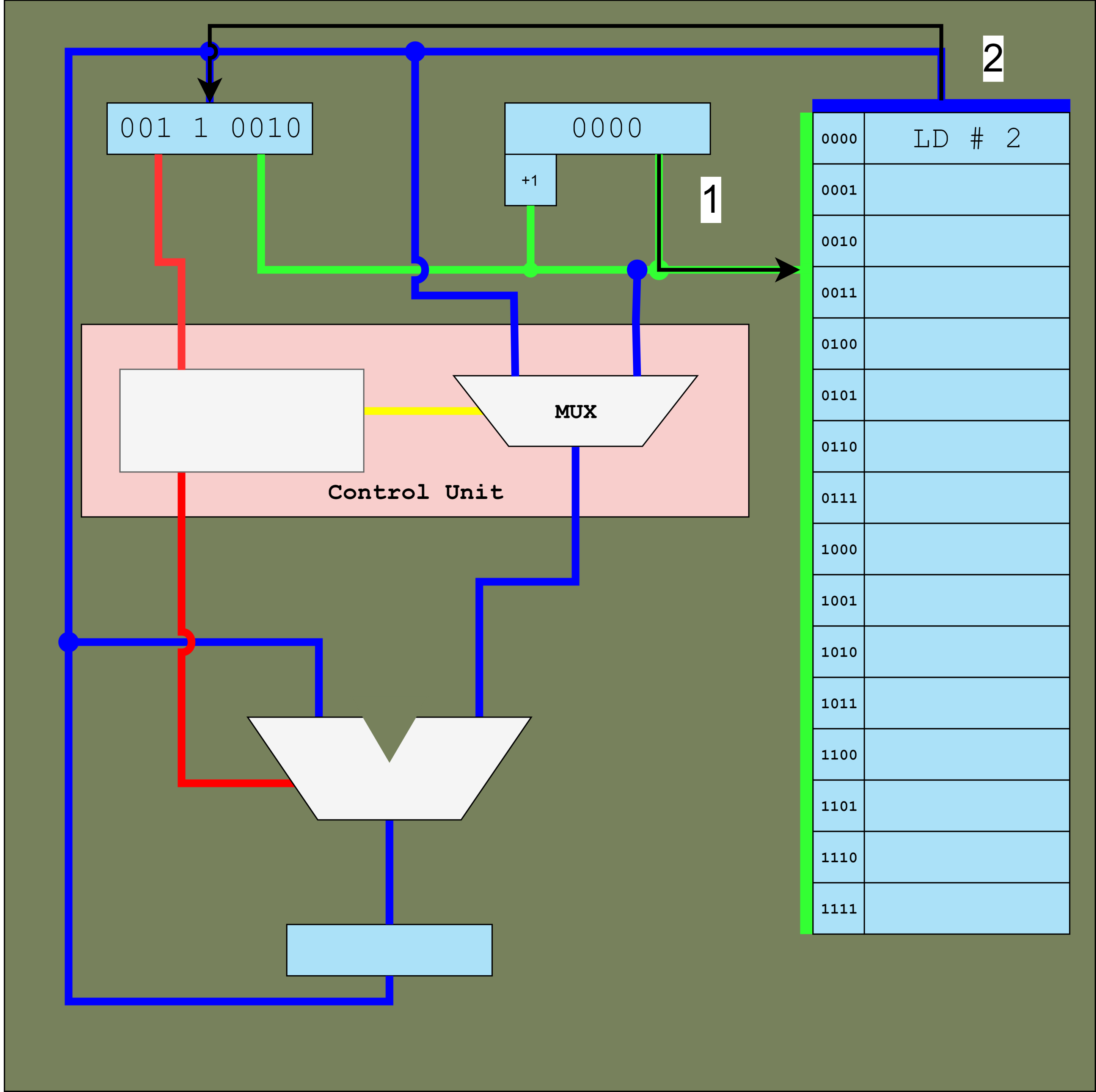
THE DUMMY CPU - EXAMPLE

- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**
- **Step 1** : the PC requests the instruction from the memory.



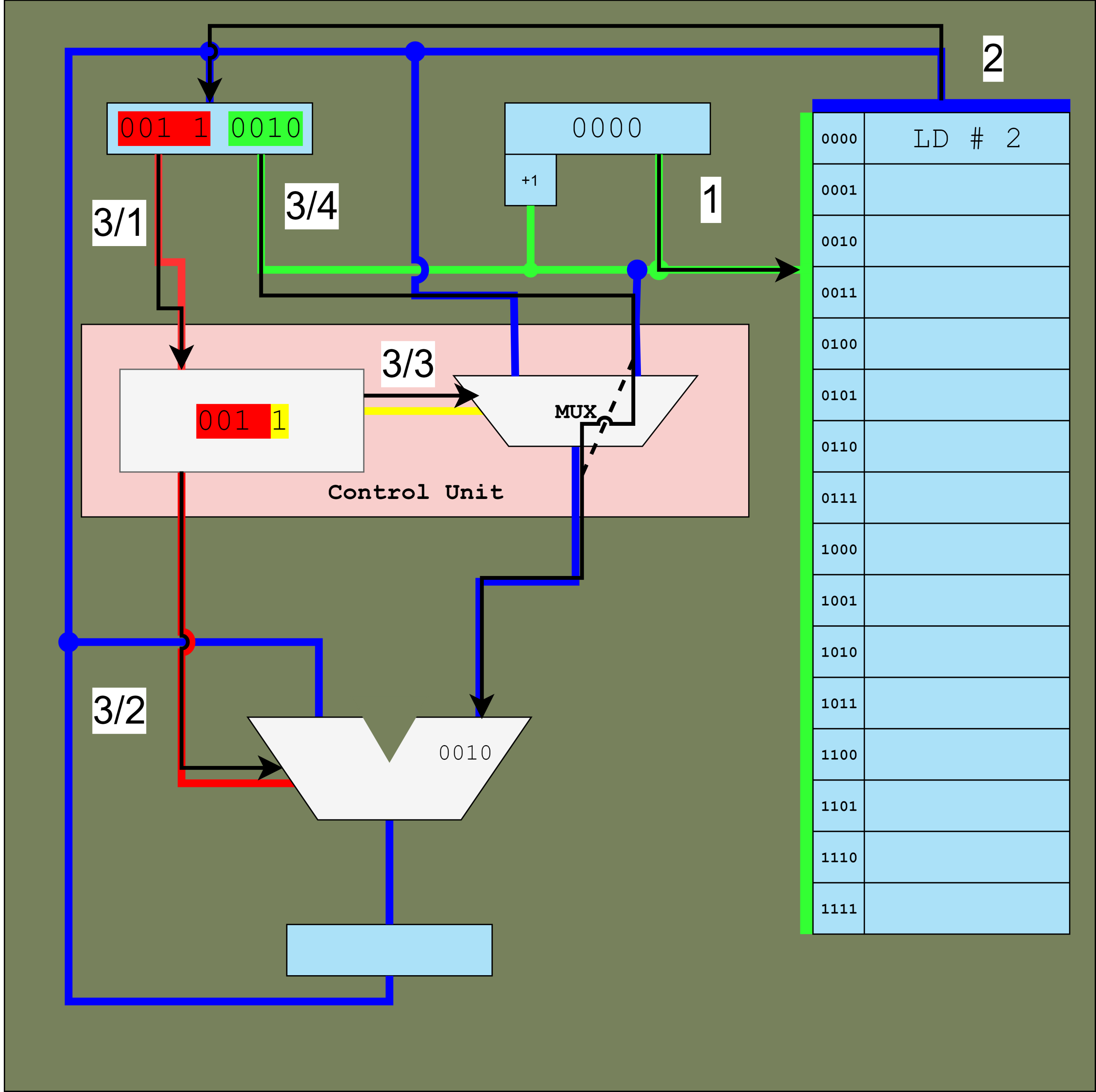
THE DUMMY CPU - EXAMPLE

- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**
- **Step 1** : the PC requests the instruction from the memory.
- **Step 2** : the memory responds by sending the instruction to the IR where it is held before it is executed.
- **Those two steps are called <u>Instruction Fetch cycle (IF).</u>**



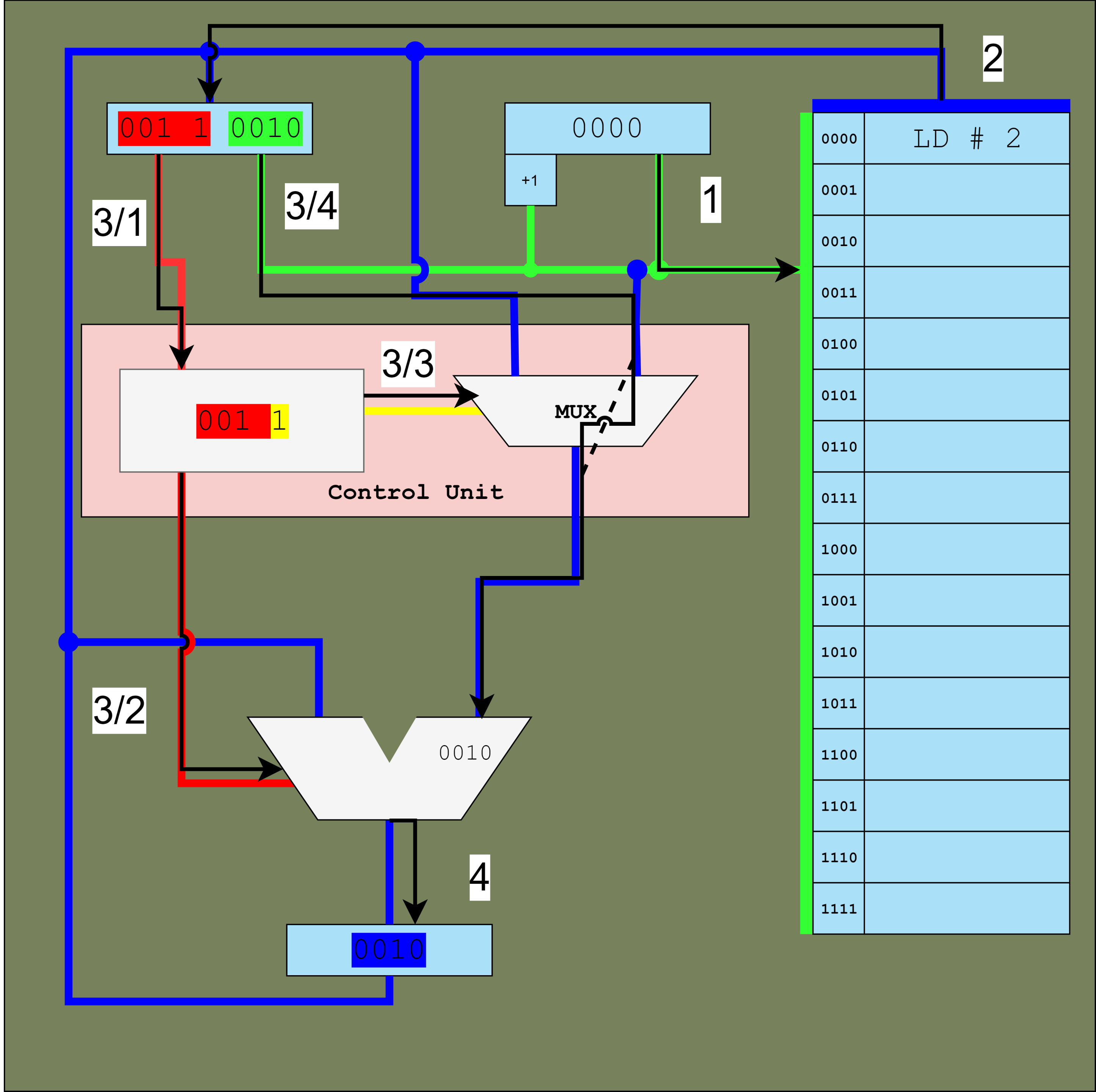
THE DUMMY CPU - EXAMPLE

- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**
- **Step 3** : several things happen:
  - 3/1: The control unit receives the instruction from the IR.
  - 3/2-3: The control unit decoder, sends commands to the ALU preparing for execution and the multiplexer.
  - 3/4: Since the information needed is in the IR, the control unit signals the IR to send it to the ALU for addition.
- **This step is called <u>Instruction Decode (ID).</u>**



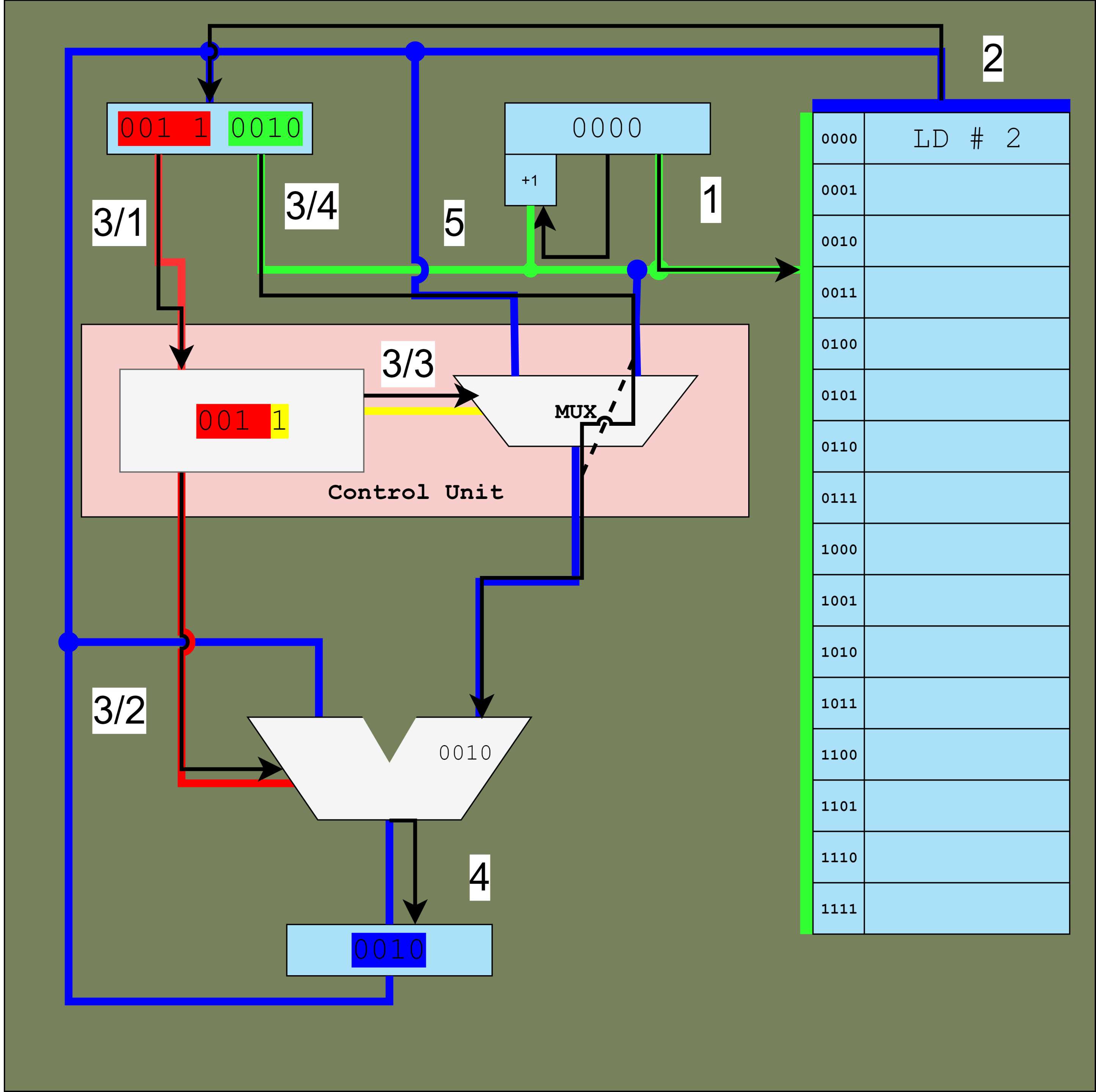
THE DUMMY CPU - EXAMPLE

- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**
- **Step 4** : The ALU executes the operation according to the command received from the control unit. In this case it is a value load to the accumulator.
- **This step is called <u>Instruction Execution (EXE).</u>**



THE DUMMY CPU - EXAMPLE

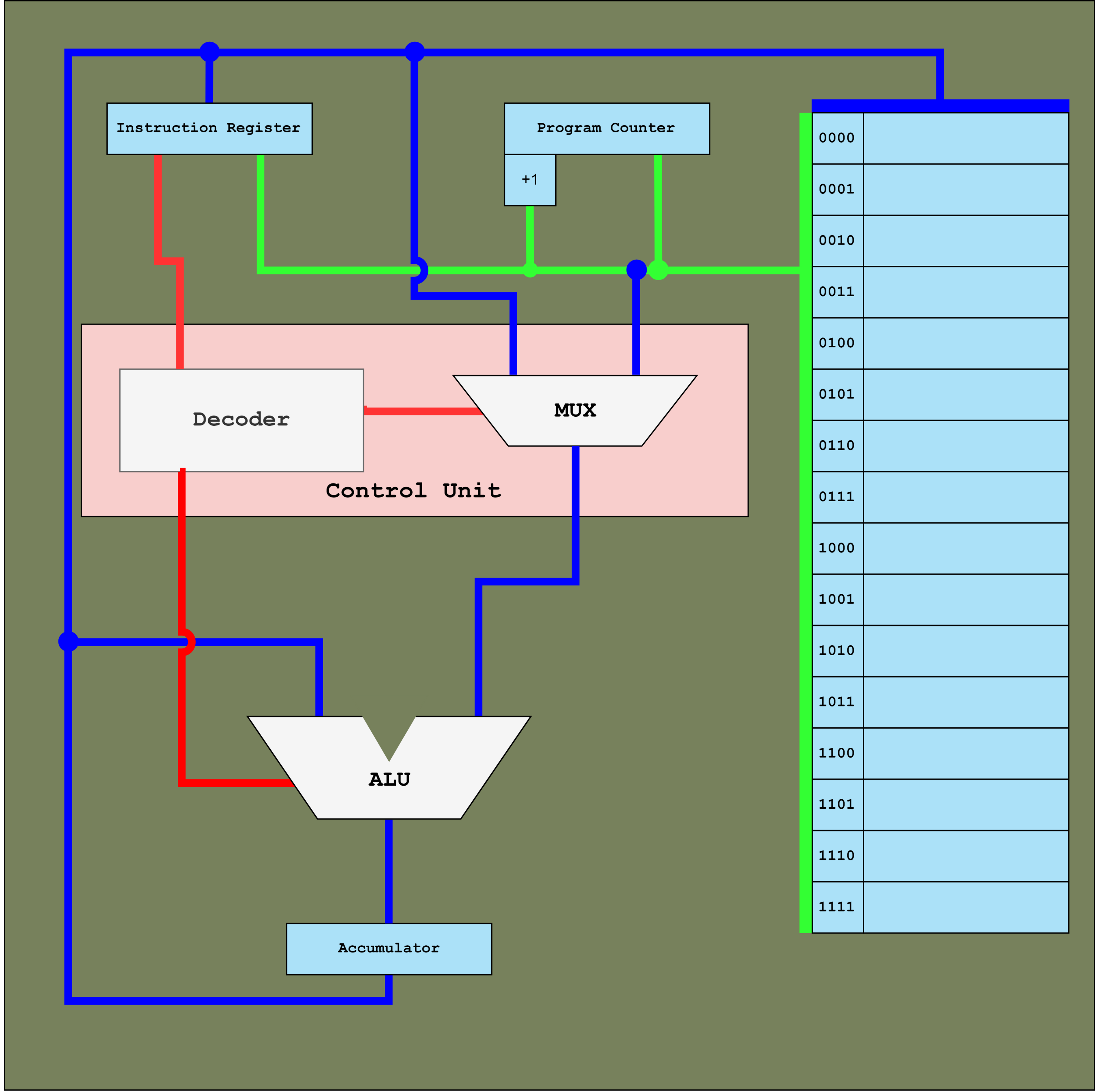
- Now that we have the instruction set, lets see how the CPU would interoperate that. Lets start with the following instruction : **LD # 2**
- **Step 5** : The IR signals the PC to increment the value in order to fetch the next instruction.



THE DUMMY CPU - EXAMPLE

- **Class exercise:**
  - Execute the following program using the 8-bit instruction set of the dummy CPU:

0	LD # 0
1	ADD #1
2	EQ #3
3	JP $1
4	ST $6
5	HE



EXERCISE

- We load the program into the memory.

0	LD # 0
1	ADD #1
2	EQ #3
3	JP $1
4	ST $6
5	HE

PC	IR	Accum	Mem location	Memory
			0000	LD # 0
			0001	ADD # 1
			0010	EQ # 3
			0011	JP $ 1
			0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- We load the program into the memory.

0	LD # 0
1	ADD #1
2	EQ #3
3	JP $1
4	ST $6
5	HE

- We start by setting the PC to the first memory location and loading that memory location into the IR.
- Since the instruction is a load instruction, the value 0000 in the instruction will be put into the Accum.

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
			0001	ADD # 1
			0010	EQ # 3
			0011	JP $ 1
			0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is then incremented the value in the PC points to location 0001. We fetch that instruction into the IR.
- Since the instruction is addition, the value in the accum is added to the value in the instruction. 0 + 1 = 1

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
			0010	EQ # 3
			0011	JP $ 1
			0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is then incremented the value in the PC points to location 0001. We fetch that instruction into the IR.
- Since the instruction is addition, the value in the accum is added to the value in the instruction. 0 + 1 = 1
- The PC is incremented.
- Since this is an equality check, we compare the accum to the value in the instruction. Value in accum (0001) does not equal 3 (0011). Thus, we continue to the next instruction. Value in accum does not change

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
			0011	JP $ 1
			0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is then incremented the value in the PC points to location 0001. We fetch that instruction into the IR.
- Since the instruction is addition, the value in the accum is added to the value in the instruction. 0 + 1 = 1
- The PC is incremented.
- Since this is an equality check, we compare the accum to the value in the instruction. Value in accum (0001) does not equal 3 (0011). Thus, we continue to the next instruction. Value in accum does not change
- The PC is incremented.
- jump instruction resets the PC to the value in the instruction which is 0001. value in accum does not change

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
			0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is set to the value in the jump instruction. And the instruction is loaded again.
- This is another addition, but now it is 1 + 1 = 2

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
			0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is set to the value in the jump instruction. And the instruction is loaded again.
- This is another addition, but now it is 1 + 1 = 2
- Check equality again. Since it is not equal we go to the next instruction

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
			0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is set to the value in the jump instruction. And the instruction is loaded again.
- This is another addition, but now it is 1 + 1 = 2
- Check equality again. Since it is not equal we go to the next instruction
- We have another jump instruction, we reset the PC counter to match the value in the jump instruction.

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	
			0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is set to the value in the jump instruction. And the instruction is loaded again.
- This is another addition, but now it is 1 + 1 = 2. We update accum
- Check equality again. Since it is not equal we go to the next instruction
- We have another jump instruction, we reset the PC counter to match the value in the jump instruction.
- Another addition, 1 + 2 = 3 and update accum

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	
0001	ADD # 1	0011	0111	
			1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The PC is set to the value in the jump instruction. And the instruction is loaded again.
- This is another addition, but now it is 1 + 1 = 2. We update accum
- Check equality again. Since it is not equal we go to the next instruction
- We have another jump instruction, we reset the PC counter to match the value in the jump instruction.
- Another addition, 1 + 2 = 3 and update accum
- Now, the equality is true. The value in accum is equal to value in equality instruction. We update the PC counter to skip the next instruction.

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	
0001	ADD # 1	0011	0111	
0010	EQ # 3	0011	1000	
			1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The next instruction is a store instruction. We store the value in the accum into memory location in the instruction. In this case, We store value 0011 into location 0110.

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	0011
0001	ADD # 1	0011	0111	
0010	EQ # 3	0011	1000	
0100	ST $ 6	0011	1001	
			1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

- The next instruction is a store instruction. We store the value in the accum into memory location in the instruction. In this case, We store value 0011 into location 0110.
- The PC is incremented.
- The instruction is halt execution. The program has finished.

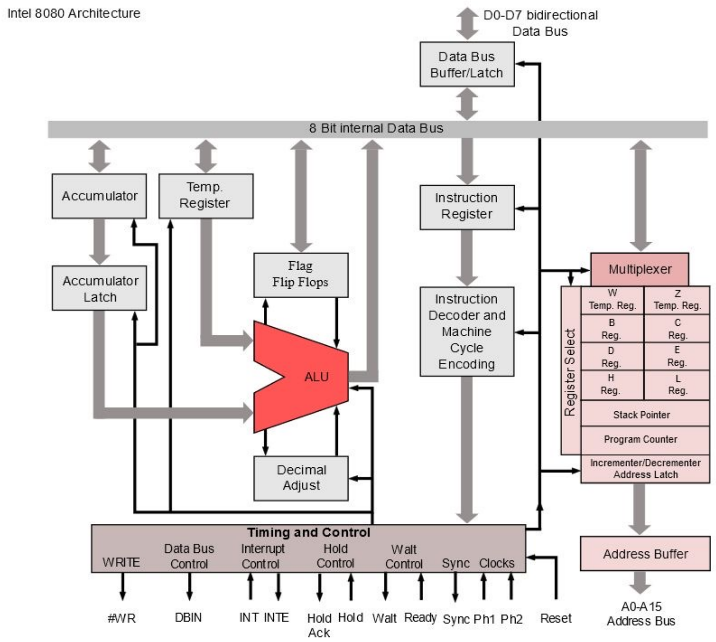
PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	0011
0001	ADD # 1	0011	0111	
0010	EQ # 3	0011	1000	
0100	ST $ 6	0011	1001	
0101	HE	0011	1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXERCISE

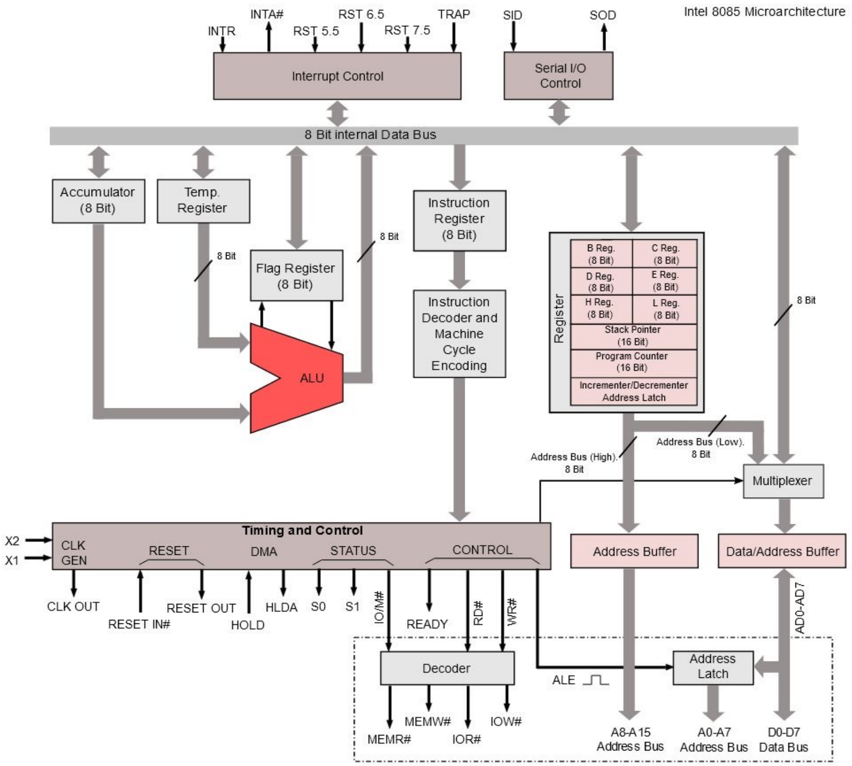
- The next instruction is a store instruction. We store the value in the accum into memory location in the instruction. In this case, We store value 0011 into location 0110.
- The PC is incremented.
- The instruction is halt execution. The program has finished.
  - **What does this program do?**
  - **What is it similar to?**

PC	IR	Accum	Mem location	Memory
0000	LD # 0	0000	0000	LD # 0
0001	ADD # 1	0001	0001	ADD # 1
0010	EQ # 3	0001	0010	EQ # 3
0011	JP $ 1	0001	0011	JP $ 1
0001	ADD # 1	0010	0100	ST $ 6
0010	EQ # 3	0010	0101	HE
0011	JP $ 1	0010	0110	0011
0001	ADD # 1	0011	0111	
0010	EQ # 3	0011	1000	
0100	ST $ 6	0011	1001	
0101	HE	0011	1010	
			1011	
			1100	
			1101	
			1110	
			1111	

EXAMPLES: INTEL 8080 / 8085



i8080



i8085

i8086/88

source: Wikipedia.org

HOME TASK

- **Home Task for the coming two weeks: (find details in Moodle HOME TASK 1)**
  - Check Moodle for the task description and file needed to complete the task
- **Lab assignment 1 can also be found on Moodle.**