DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

CHITTAGONG UNIVERSITY OF ENGINEERING AND TECHNOLOGY

CHATTOGRAM – 4349, BANGLADESH

Experiment No. 2

Schematic Modeling and Implementation of Modified SAP Architecture in Logisim – 2

PRECAUTIONS:

- Students must carefully read the lab manual before coming to lab to avoid any inconveniences.
- Students must carry a flash drive to transfer files and lab manuals.
- Use of mobile phone in lab is strictly prohibited and punishable offence.
- Experiment files must be uploaded to GitHub including home tasks.

OBJECTIVES:

- To familiarize with the BUS in a microprocessor system.
- To familiarize with addressing and RAM.
- To familiarize with program execution cycle.

THEORY:

The instruction cycle (also known as the fetch–decode–execute cycle, or simply the fetch-execute cycle) is the cycle that the central processing unit (CPU) follows from boot-up until the computer has shut down in order to process instructions. It is composed of three main stages: the fetch stage, the decode stage, and the execute stage. In simpler CPUs, the instruction cycle is executed sequentially, each instruction being processed before the next one is started. In most modern CPUs, the instruction cycles are instead executed concurrently, and often in parallel, through an instruction pipeline: the next instruction starts being processed before the previous instruction has finished, which is possible because the cycle is broken up into separate steps. This entire process can be visualized by figure 1 [1]. Here PC stands for Program Counter, MAR stands for Memory Address Register, MDR stands for Memory Data Register, CIR stands for Current Instruction Register, CU stands for Control Unit and ALU stands for Arithmetic Logic Unit.

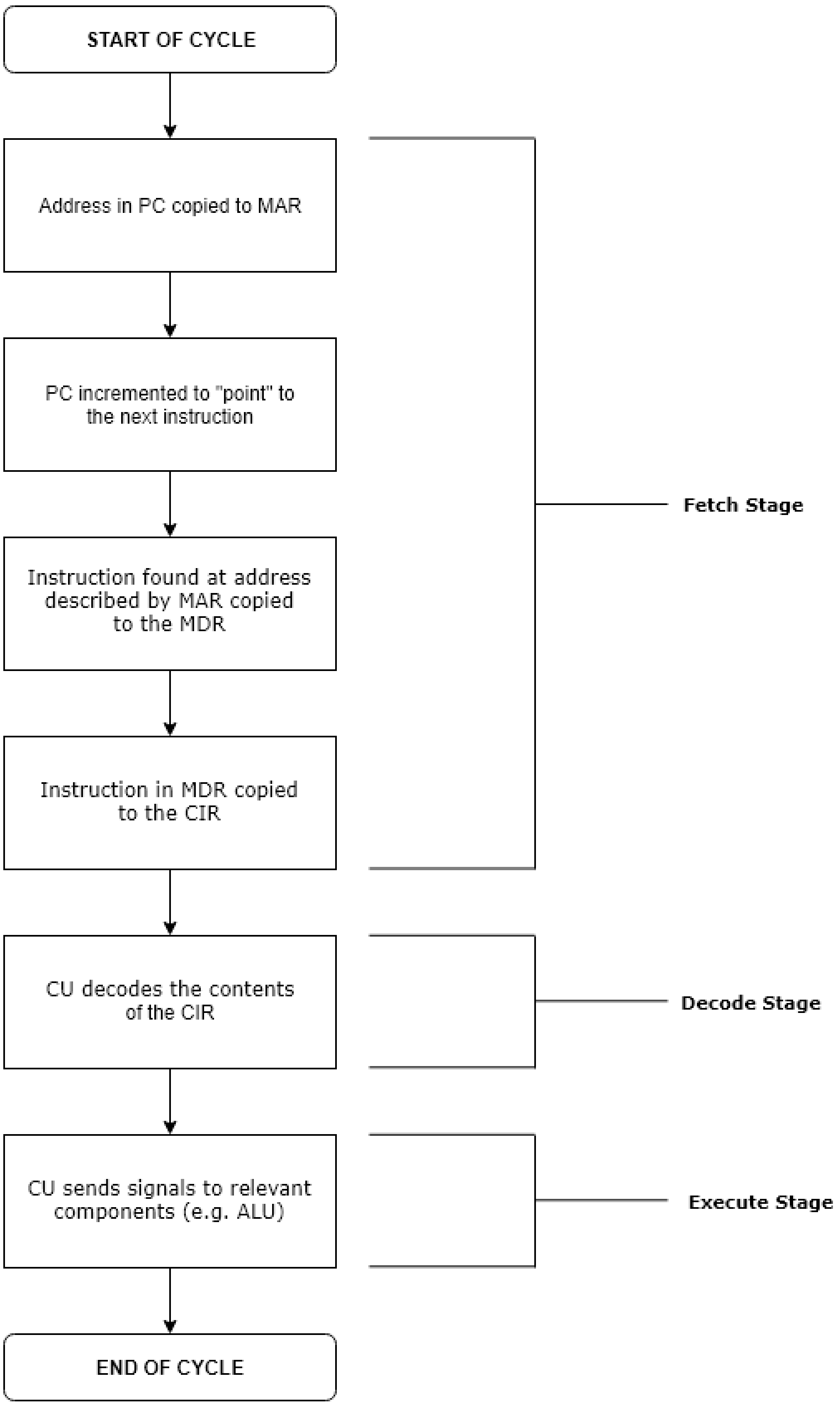
START OF CYCLE
Address in PC copied to MAR
PC incremented to "point" to the next instruction
Instruction found at address described by MAR copied to the MDR
Instruction in MDR copied to the CIR
Fetch Stage
CU decodes the contents of the CIR
Decode Stage
CU sends signals to relevant components (e.g. ALU)
Execute Stage
END OF CYCLE

Figure 1: Individual stages in a fetch-decode-execution cycle [1].

The program counter (PC) is a special register that holds the memory address of the next instruction to be executed. During the fetch stage, the address stored in the PC is copied into the memory address register (MAR) and then the PC is incremented in order to "point" to the memory address of the next instruction to be executed. The CPU then takes the instruction at the memory address described by the MAR and copies it into the memory data register (MDR). The MDR also acts as a two-way register that holds data fetched from memory or data waiting to be stored in memory (it is also known as the memory buffer register (MBR) because of this). Eventually, the instruction in the MDR is copied into the current instruction register (CIR) which acts as a temporary holding ground for the instruction that has just been fetched from memory.

During the decode stage, the control unit (CU) will decode the instruction in the CIR. The CU then sends signals to other components within the CPU, such as the arithmetic logic unit (ALU) and the floating-point unit (FPU). The ALU performs arithmetic operations such as addition and subtraction and also multiplication via repeated addition and division via repeated subtraction. It also performs logic operations such as AND, OR, NOT, and binary shifts as well. The FPU is reserved for performing floating-point operations.

Summary of stages:

Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

- Fetch stage: The next instruction is fetched from the memory address that is currently stored in the program counter and stored into the instruction register. At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
- Decode stage: During this stage, the encoded instruction presented in the instruction register is interpreted by the decoder.
  Read the effective address: In the case of a memory instruction (direct or indirect), the execution phase will be during the next clock pulse. If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (clock pulse: T3). If the instruction is direct, nothing is done during this clock pulse. If this is an I/O instruction or a register instruction, the operation is performed during the clock pulse.
- Execute stage: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant functional units of the CPU to perform the actions required by the instruction, such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU. The result generated by the operation is stored in the main memory or sent to an output device. Based on the feedback from the ALU, the PC may be updated to a different address from which the next instruction will be fetched.
- Repeat cycle

In addition, on most processors interrupts can occur. This will cause the CPU to jump to an interrupt service routine, execute that and then return. In some cases, an instruction can be interrupted in the middle, the instruction will have no effect, but will be re-executed after return from the interrupt [2].

<u>DESIGN PROCESS:</u>

Decoder (Optional):

A decoder is a combinational circuit that has n number of select bits and a maximum of 2^n number of output lines. Only one of the outputs of the decoder will be active at any given time for a combination of input bits. In our case, we would be using a 4 to 16 decoder, meaning, there are 4 selection lines and 16 output lines. These 16 output lines will be used to access the 16 different addresses in our RAM, which we will design after this step. The schematic of the decoder can be seen in figure 2.

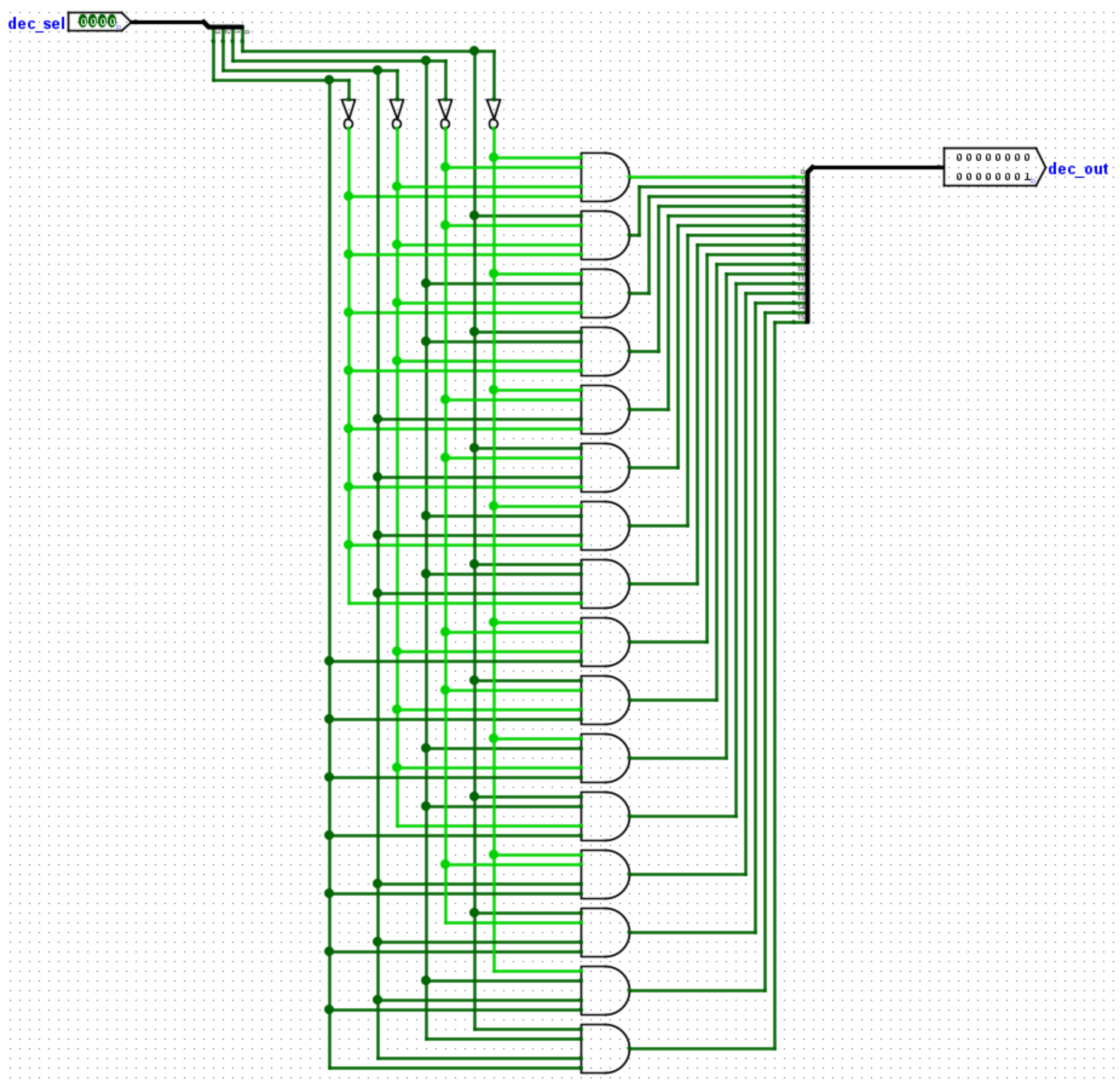

Figure 2: Schematic of 4 to 16 decoder.

Now we will create one of the memory cells of our SRAM. This can be created by creating a new subcircuit and using the following schematic, incorporating our previously designed general purpose register.

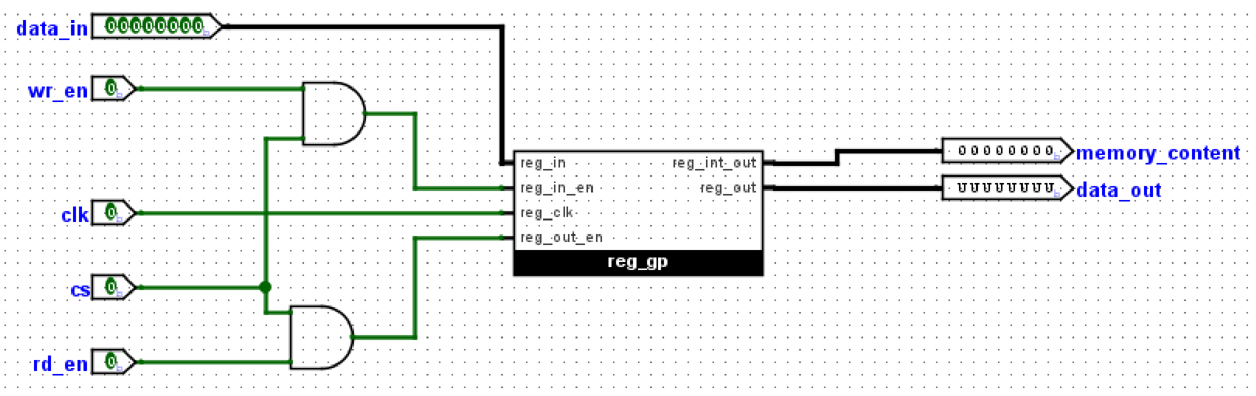

Figure 3: Schematic of a single SRAM cell.

Here, the "wr_en" pin can be used to enable writing to the cell and "rd_en" can be used to enable reading. Both these pins are enabled using the "cs" or Chip Select pin. This memory cell can now be used to build our RAM.

Random Access Memory (RAM):

RAM is a bank of memory available to the processor to be used while executing a program. Any program that needs to be executed, is stored in RAM first. The design of the RAM can be seen from figure 4 and 5.

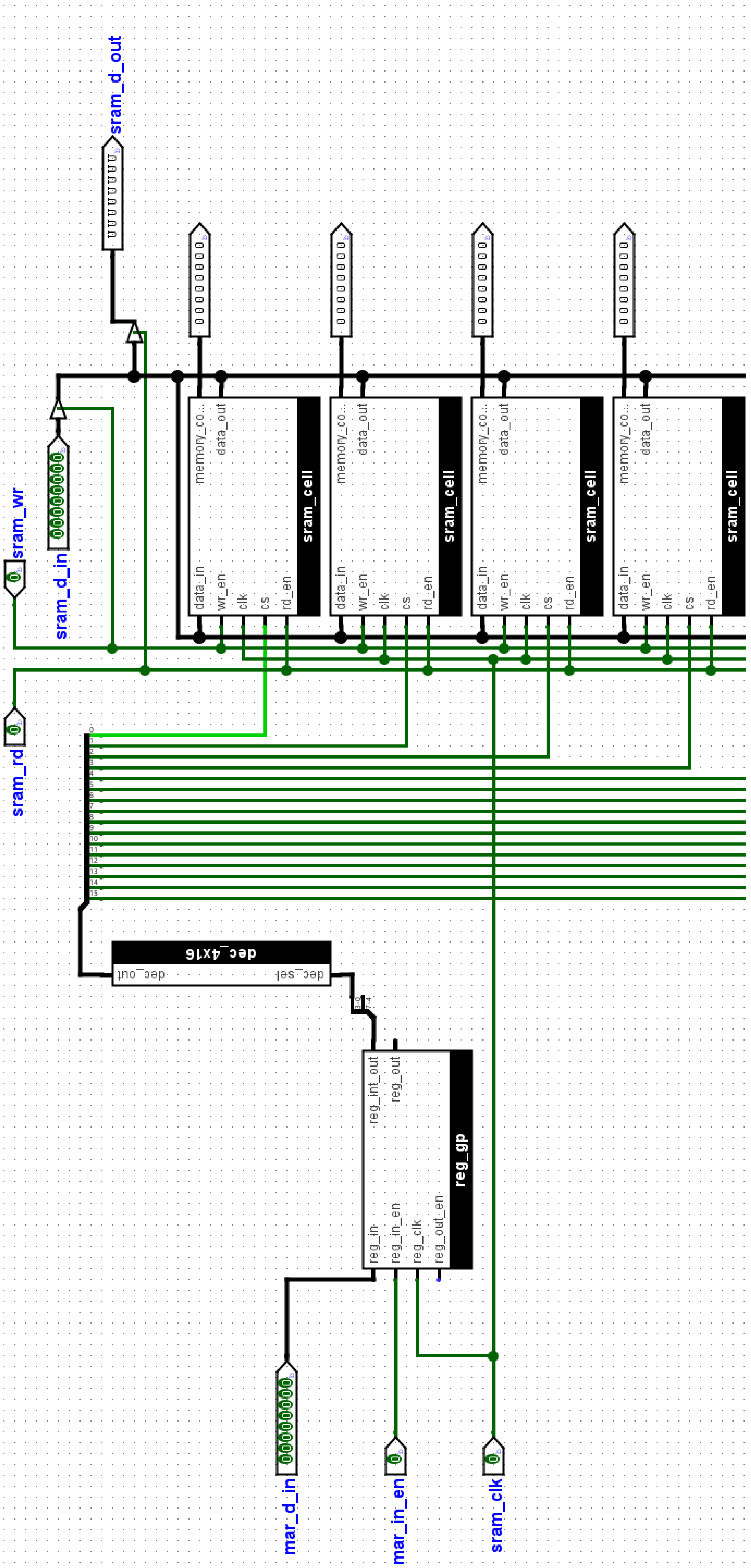

Figure 4: Schematic of RAM part-I.

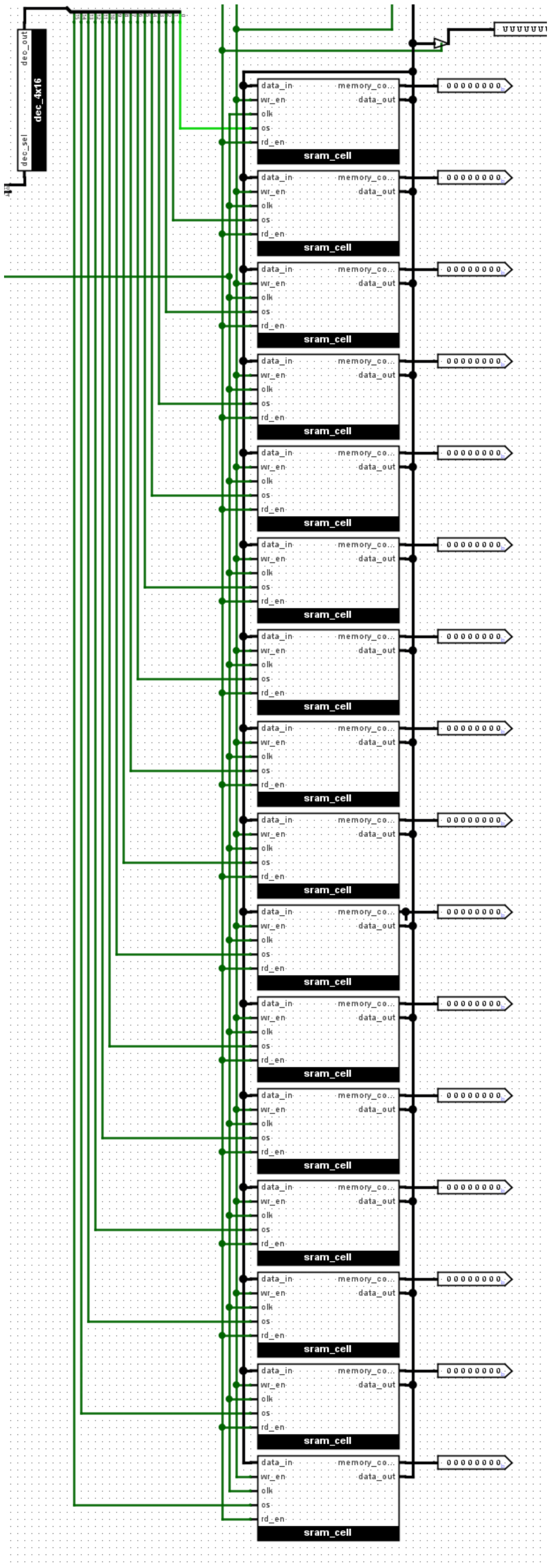

Figure 5: Schematic of RAM part-II.

Instruction Register:

The instruction register holds the instruction currently to be executed. This separates the BUS information into two 4-bit parts. The higher 4 bits are the Opcode and the lower 4 bits are the operands or addresses. The schematic of the instruction register can be seen in figure 6.

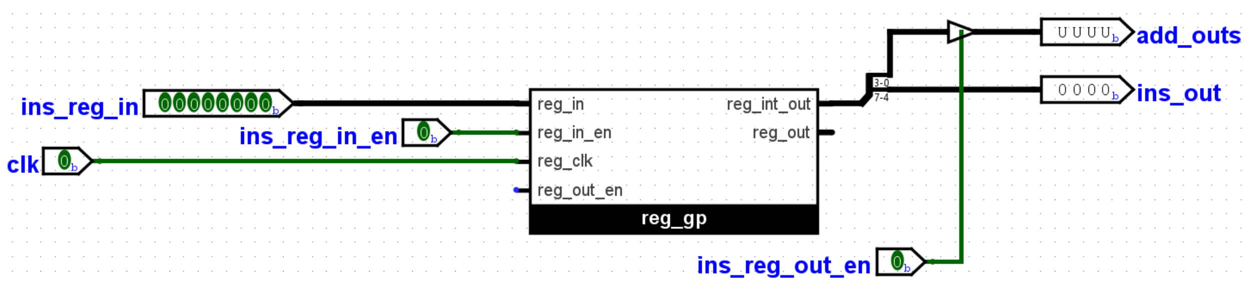

Figure 6: Schematic of IR.

Now we have all the necessary parts for building our very own processor. At this stage we assemble all parts of the processor as shown in the figure 7. Various parts of the processor are now connected through the help of the central BUS. The control pins of all the subcircuits are tunneled to a unified location for the ease of manipulations. A debug control and debug data pin is also added to the bus in order to program the RAM. At this stage we would need one final component, the control sequencer or control unit. This control unit controls the control BUS to manipulate the various control pins to ensure proper operation. This is out of the scope for our lab and we shall run the processor manually to understand how it works.

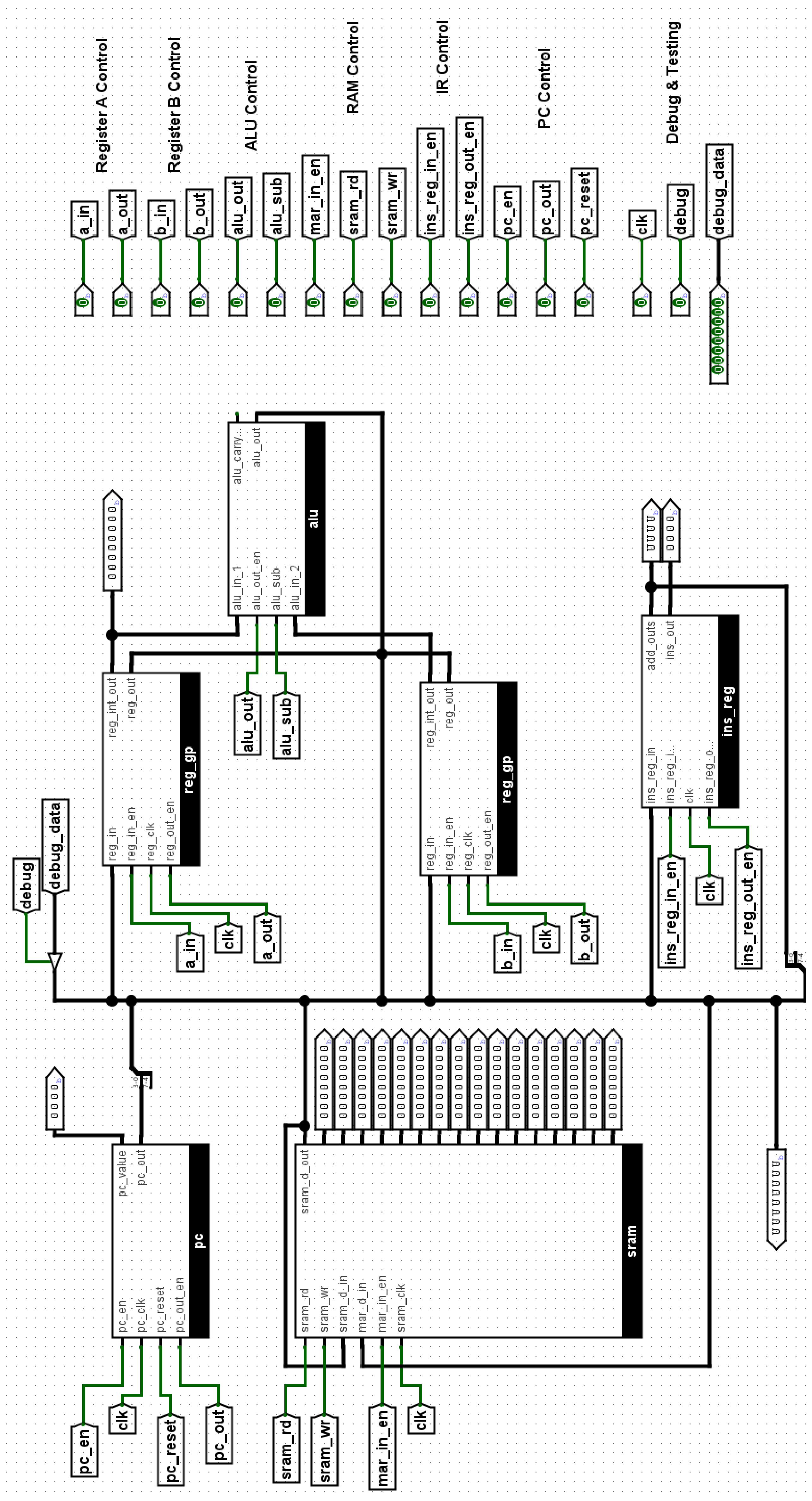

Figure 7: Final schematic layout.

After the design is ready, its time to run our first program. To run our first program, we must program our RAM first. As the processor is designed by us, the instruction set it uses must be designed by us as well.

Let us denote the instruction: 0001 1010 as LDA 10 or load the register A with memory contents of location 10. This instruction has two parts. The Opcode: 0001 meaning load A and the operand 1010 meaning 10. We need to load this into the first location on the RAM and some value which would be moved to register A, in memory location 10.

Programming Steps:

- Turn on *debug* pin.
- Pulse the *pc_reset* pin.
- Set *debug_data* to 0000 0000.
- Toggle *mar_in_en* and give a clock pulse.
- Turn off *mar_in_en.*
- Set *debug_data* to 0001 1010.
- Toggle *sram_wr* and give a clock pulse.
- Turn off *sram_wr* and observe the data saved in address 0000.
- Now set *debug_data* to 0000 1010.
- Toggle *mar_in_en* and give a clock pulse.
- Turn off *mar_in_en.*
- Set *debug_data* to 0000 0111.
- Toggle *sram_wr* and give a clock pulse.
- Turn off *sram_wr* and observe the data saved in address 1010.
- Turn off *debug* pin.

Now that programming is done, let us run our program.

Fetch:

Fetch stage has 3 T-states, all followed by a clock pulse. Before proceeding, reset program counter to 0000.

- T1: Toggle *pc_out* & *mar_in_en* and give a clock pulse. Address of next instruction to be executed is sent to MAR from PC. Turn off the control pins.
- T2: Toggle *sram_rd* & *ins_reg_in_en* and give a clock pulse. Check the IR loaded with the Opcode and operand. Turn off the control pins.
- T3: Toggle *pc_en* and give a clock pulse. This increments the counters value to be 0001 or location of the next line of code. Turn of the control pin.

Decode:

We are acting as the decoder in our first program. Decode stage has no T-states and in pure combinational logic.

Execute:

As the loaded instruction is load A or LDA, it also has 3 T-states as follows:

- T1: Toggle *ins_reg_out_en* & *mar_in_en* and give a clock pulse. This saves the address to be fetched into the MAR. Turn off the control pins.
- T2: Toggle *sram_rd* & *a_in* and give a clock pulse. This reads the memory contents of address 1010 which is 0111 and saves it in register A. Turn off the control pins.
- T3: This state is unused by LDA.

Using this strategy, another 15 more instructions can be made up for this processor and the entire thing would be called the instruction set. Control Unit is implemented to automate this manual control switching. Design of the sequencer can also be learned from the reference [3].

<u>HOME TASK:</u>

Objective:

Your task is to write a simple machine code program for your SAP – 1 architecture that performs an addition operation. Specifically, you will load two pre-defined values into registers A and B, add them, and then store the sum in a designated memory location.

1. **Choose Your Values:**
   - Select two 8-bit values to add. First is the last 5 digits of your ID (i.e. 08010), second being the current year (i.e. 2025). Convert these to binary first.
   - Decide on a memory address where you will store these initial values.
   - Decide on a separate memory address where the final sum will be stored.
2. **Write the Machine Code Program:**
   - Choose your own opcodes and write the machine code to carry out the steps.
   - Load the first value to Register A (i.e. LDA 13).
   - Load the second value to Register B (i.e. LDB 14).
   - Store the added value from the ALU to your chosen memory location (i.e. STA 15).
   - Halt the program (i.e. HLT).
3. **Load and Execute Your Code:**
   - Use the *debug* pin and load your instructions to memory, starting from address 0000.
   - Load your two chosen values into their respective memory locations.
   - Reset the Program Counter (PC) to 0000.
   - Run the simulation step-by-step, observe the data flow and register contents.
   - At the HLT stage, simply do nothing in the execute phase.

Properly explain your thought process and rational behind your approach along with how the code would be executed, showing how and when each control pin is turned on/off.

Rubrics:

Criteria	Below Average (1)	Average (2)	Good (3)
Formatting	The report is generally well-formatted, including all required sections, but may have minor inconsistencies or omissions in presentation.	The report is well-formatted, logically structured with clear objectives, detailed discussions, and properly cited references, presenting a professional appearance.	
Program Execution and Home Task	Only the final outcomes were presented without sufficient explanation of the program's logic, execution steps, or how the home task was achieved.	The outcomes were shown with some explanation, but the home task implementation contains errors, is incomplete, or lacks clear documentation.	The outcomes of both the manual lab tasks and the home task are thoroughly documented, correctly executed, and clearly explained, demonstrating a solid understanding.
Writing	Writing is unclear, contains significant grammatical errors, is not informative, fails to address design decisions, and/or shows high plagiarism.	Writing is generally clear and original, with some grammatical errors. Design decisions are briefly explored, and plagiarism is minimal.	Writing is clear, concise, original, and informative. Design decisions are adequately explored, and the report is free from plagiarism.
Diagram	Diagrams are present and generally clear, but may lack complete labeling, optimal resolution, or could be better integrated with the text.	Diagrams are clear, high-quality, well-labeled, and effectively support the explanation of the circuit and program logic.	

References

[1] Wikipedia, "Instruction Cycle," [Online]. Available: https://en.wikipedia.org/wiki/Instruction_cycle.

[2] FutureLearn, "Fetch-Decode-Execute Cycle," [Online]. Available: https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49284.

[3] B. Eater, "Building an 8-bit breadboard computer!," [Online]. Available: https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU.