4 A very simple microprocessor

(This chapter has been written cooperation with Dr. Mahfuz Aziz, Senior Lecturer at the School of Electrical and Information Engineering, University of South Australia)

This chapter gives an introduction to microprocessor architecture. The goal of the project is to build a 4-bit processor at logic level and then simulate step-by-step its internal structure.

1. Introduction

The Very-Simple-Microprocessor (VSM) is an updated version of the very popular SAP (Simple-As-Possible) computer architecture proposed by Albert P. Malvino [1] in 1993 in his famous book "Digital Computer Electronics". The VSM computer introduces the basic concepts of microprocessor architecture in the simplest possible way. The VSM is very primitive, but already quite complex, as shown in Figure 4-1.

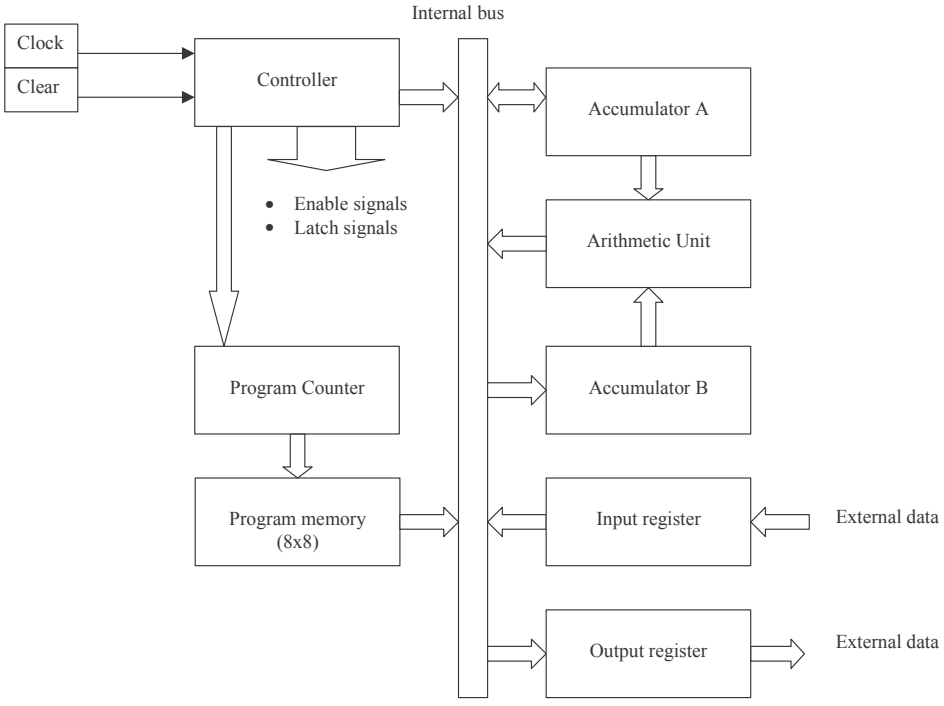


Figure 4-1: VSM basic architecture

The function of each block is described in Table 4-1.

Block	Block	Size
Program Counter	The program counter counts from 0000 to 1111. It monitors the address of the active instruction. Initially, the program counter is set to 0000, so the microprocessor starts with the instruction at the first memory location.	4 bits
Program Memory	The program memory stores the program. Each program line has an 8bit format: the four most significant bits represent the instruction itself, and the four least significant bits represent the data attached to the instruction, if necessary.	8x8 bits
Accumulator A	The accumulator is a 4-bit register. It is used to store one of the operands for an arithmetic operation. It also stores the intermediate results computed by the microprocessor. Upon request (EnableA), the accumulator result is placed on the internal bus.	4 bits
Accumulator B	The accumulator B is also a 4-bit register. It is used to store the second operand for an arithmetic operation. For addition this operand is added to accumulator A and for substraction accumulator A is subtracted from this operand.	4 bits
Arithmetic Unit	The Arithmetic Unit performs the operation S=A+B (Addition) Or S=B+~A+1 (Substraction)	4 bits
Input Register	The Input Register gives the opportunity to transfer data from the outside world to the microprocessor.	4 bits
Output Register	The Output Register transfers the contents of the internal bus to the outside world. Usually, this instruction is executed at the end of the program to display the final result. The output register stores the output data on the falling edge of the clock. The output register is usually connected to a circuit which transfers or displays the result to the user.	4 bits

Table 4-1: The main blocks of the VSM architecture

The operation of the VSM is based on a bus called “internal bus” (IB). Each block shown in Figure 4-2 may take control of the bus using a specific enable signal. For example, accumulator A uses an enable signal called *EnableA*. When *EnableA* is high, the content of accumulator A is placed on the internal bus.

All the enable signals used in the VSM are shown in Figure 4-2. Table 4-2 summarizes their functions. The control of these enable signals is provided by the *MicroInstruction* block, which plays a fundamental role in the operation of the microprocessor.

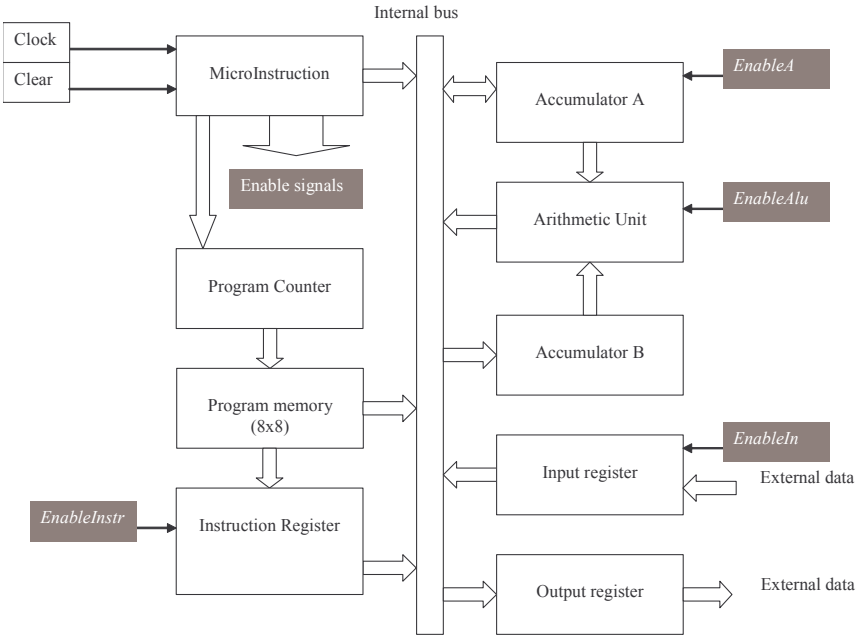


Figure 4-2: The controller generates "Enable" signals that allow one block to take the control of the bus

Enable Signal	Description
EnableA	Authorizes A to take control of the bus.
EnableAlu	Places the result of the arithmetic operation (ADD or SUB) on the bus
EnableInstr	Places the data part of the instruction (four least significant bits) on the bus.
EnableIn	Transfers the contents of the external input to the internal bus.

Table 4-2: Four blocks may take the control of the internal bus, thanks to "Enable" signals

2. Instructions

Each instruction of the VSM is 8-bits long. However, only the four most significant bits represent the instruction itself. Therefore, only 16 different instructions are possible.

No Operation (NOP=0000)

The No Operation instruction has no effect. It does not modify the content of any register. However, this instruction is very interesting to understand how the basic clock controls work.

Addition (ADD=0001)

The content of accumulator A is added to the data given with the instruction as a parameter, and the result updates the accumulator A. The addition is performed on four bits. The carry is ignored. For example, considering that A=2, the instruction "ADD 3" corresponds to A=A+3, that is A=2+3. The final value of A is 5.

Subtraction (SUB=0010)

The content of accumulator A is subtracted from the data given as a parameter, and the result updates the accumulator A. The subtraction is performed on four bits. The carry is ignored.

Get Input (In=0100)

The content of the input port is transferred to accumulator A.

Give Output (OUT=0011)

The content of accumulator A is stored on the output port. The output port is a 4-bit register that memorizes the output value and makes it available to external devices until its content is refreshed by a new "Give Output" instruction.

Load Accumulator A (LDA=0101)

This instruction loads the accumulator A with the value given as a parameter. For example, the instruction LDA 9 transfers the value 9 (1001 in binary format) to accumulator A.

3. Program Memory

The program memory contains up to 8 bytes, where we store the instructions to be executed. Each instruction is 8-bits long. As shown in Figure 4-3 each instruction is split into two parts: the four most significant bits represent the instruction code, while the four least significant bits represent the data. The program given in Table 4-3 loads accumulator A with the value "2", then adds "1", and places the result in the output register.

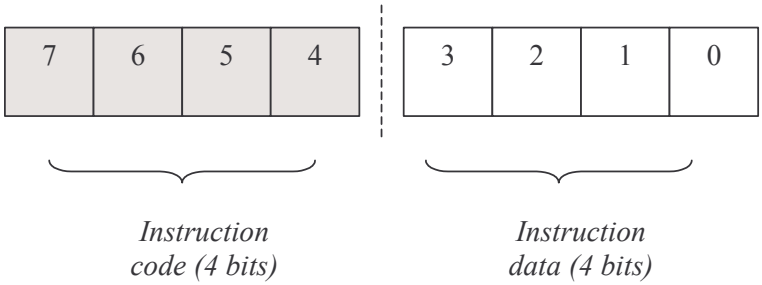


Figure 4-3: Each instruction is split into 4-bit microinstruction code and 4-bit data fields

Mnemonic	**OpCode (binary)**	**OpCode (hexa)**
LDA 2	0101 | 0010	0x52

ADD 1	0001 | 0001	0x11
OUT	0011 | 0000	0x30
NOP	0000 | 0000	0x00

Table 4-3: A sample program for adding two 4-bit numbers

Figure 4-4 the shows the memory symbol along with the corresponding schematic diagram depicting the contents of all the eight memory locations. . The memory has 8 registers, each register having 8 elementary memory cells. You can change the contents of the memory by clicking on the desired memory cells. When you save the schematic diagram, you also save the memory contents. The memory symbol may be found in the basic symbol palette in DSCH.

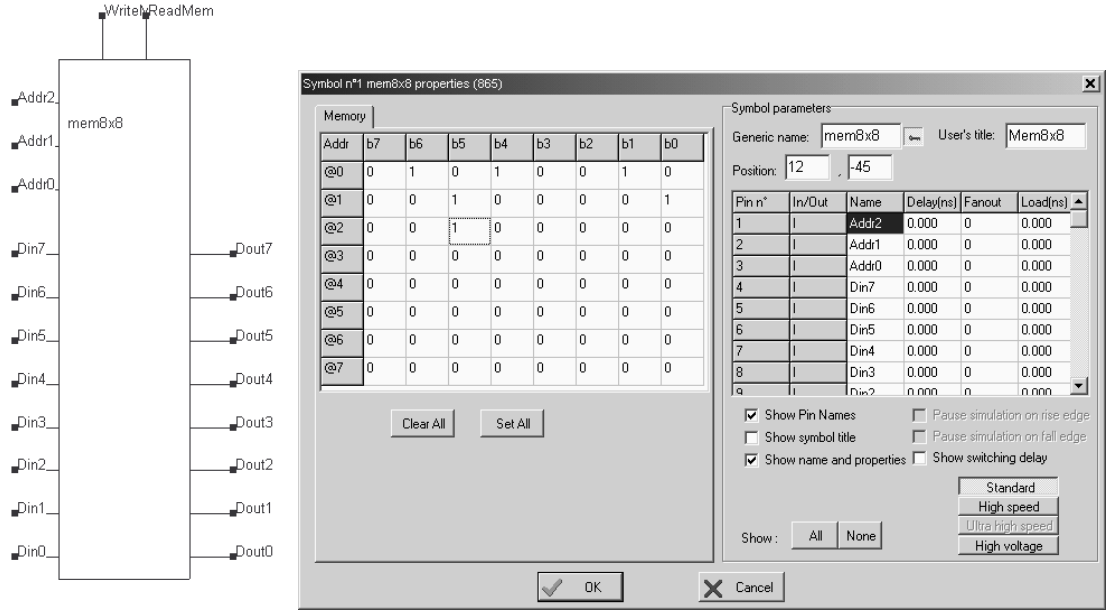


Figure 4-4: Storing the program in the memory (VSM-mem8x8macro.sch)

4. Executing the instructions

Introducing the micro-instructions

Each VSM instruction is executed as a sequence of four internal micro-operations also called *microinstructions*. Therefore the period of execution of each instruction can be divided into four time phases (T_1-T_4), each for one microinstruction, as shown in Figure 4-5. The reader should note the distinction between the microprocessor instruction itself such as "LDA 2" and the four internal microinstructions needed to complete the "LDA 2" instruction, called phase 1, 2, 3 and 4. The first two phases are called the *fetch sequence*. The corresponding microinstructions are independent of the user's instruction. The last two phases are called the *execute sequence*. Table 4-4 summarizes the microinstructions.

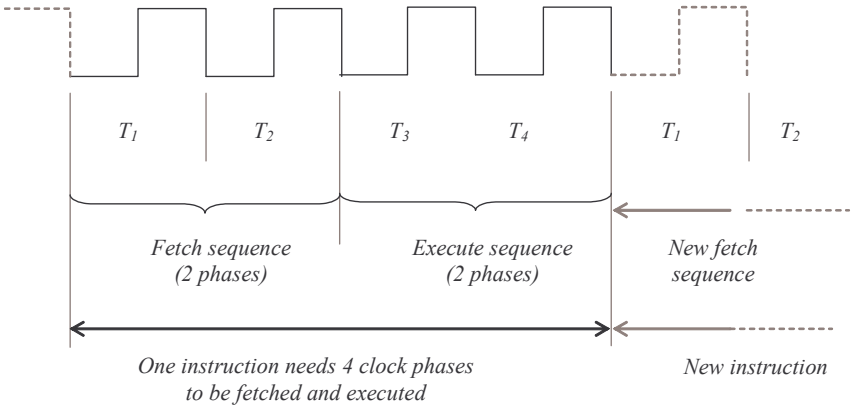


Figure 4-5: The execution of one VSM instruction involves the execution of four microinstructions in four separate time phases

Phase	Name	Description
Phase 1	Address state	The content of the desired memory location is loaded into the instruction register.
Phase 2	Increment state	The program counter address is incremented. The instruction register provides the microinstruction decoder with the instruction.
Phase 3	Execute step 1	Depending on the instruction, the microprocessor performs the first step of the execution phase.
Phase 4	Execute step 2	The microprocessor performs the second step of the execution phase

Table 4-4: The execution of one instruction is based on four time phases

No Operation (NOP=0000)

The control flow for the 'No Operation' instruction is shown in Figure 4-6. The Fetch sequence corresponds to access to the memory (ReadMem=1), and the loading of the corresponding instruction (LoadInstr=1) during phase 1. During phase 2, the stored instruction is sent to the microinstruction controller (EnableInstr=1), while the counter is incremented (ProgCount=1). As the 'No Operation' instruction does not affect any internal register, the execution phases (Phase 3 and phase 4) do not correspond to any specific activity.

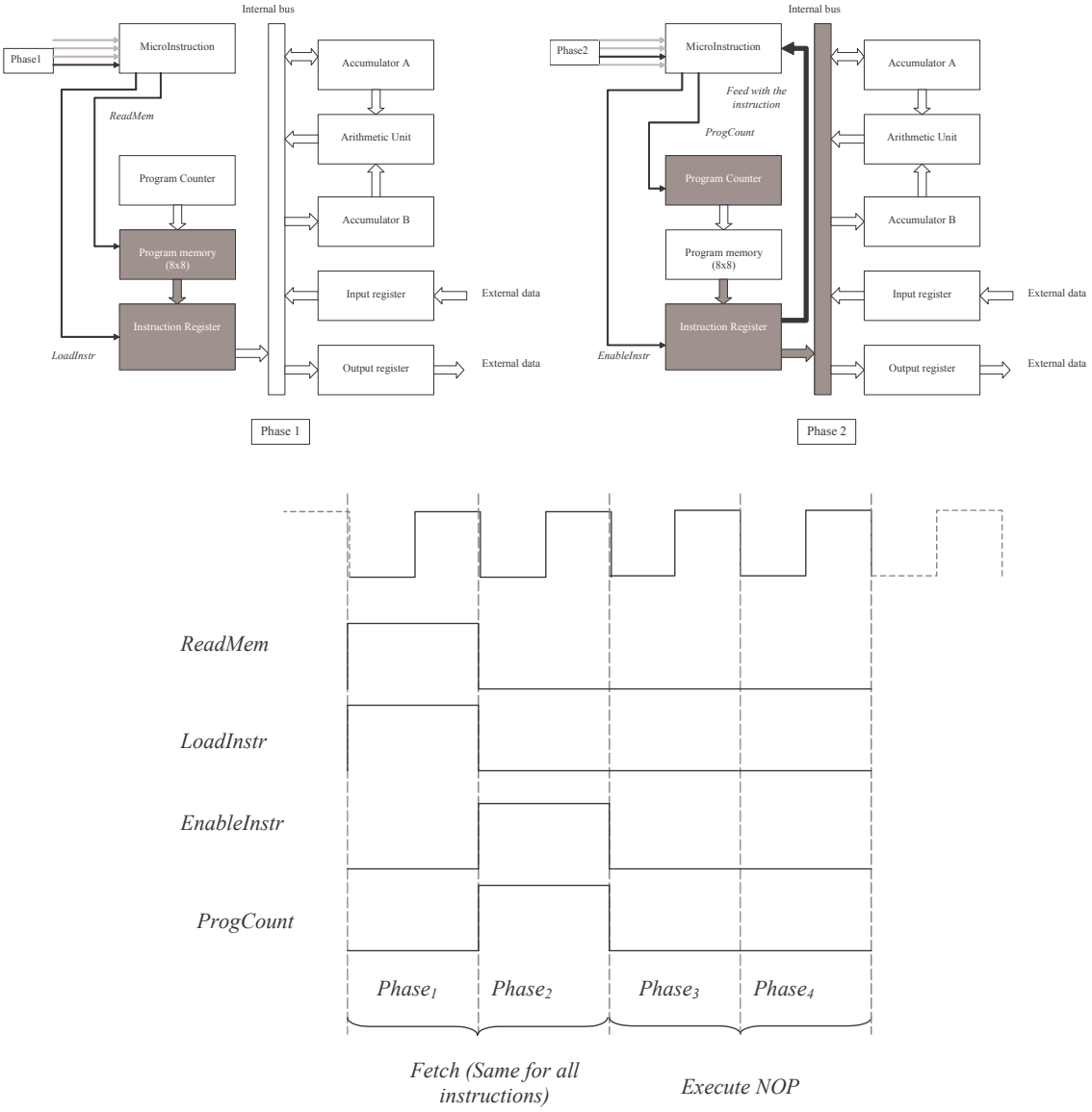


Figure 4-6: Execution of the microinstructions corresponding to the NOP instruction

Addition (ADD=0001)

Addition is performed between the content of accumulator A and the 4-bit data given as a parameter of the ADD instruction. Consequently, the addition is executed by storing the data in accumulator B (Phase 3), then asking the arithmetic unit to produce the addition between accumulator A and accumulator B (Phase 4), and finally by transferring the result back to accumulator A on the rising edge of the clock during phase 4, as illustrated in Figure 4-7.

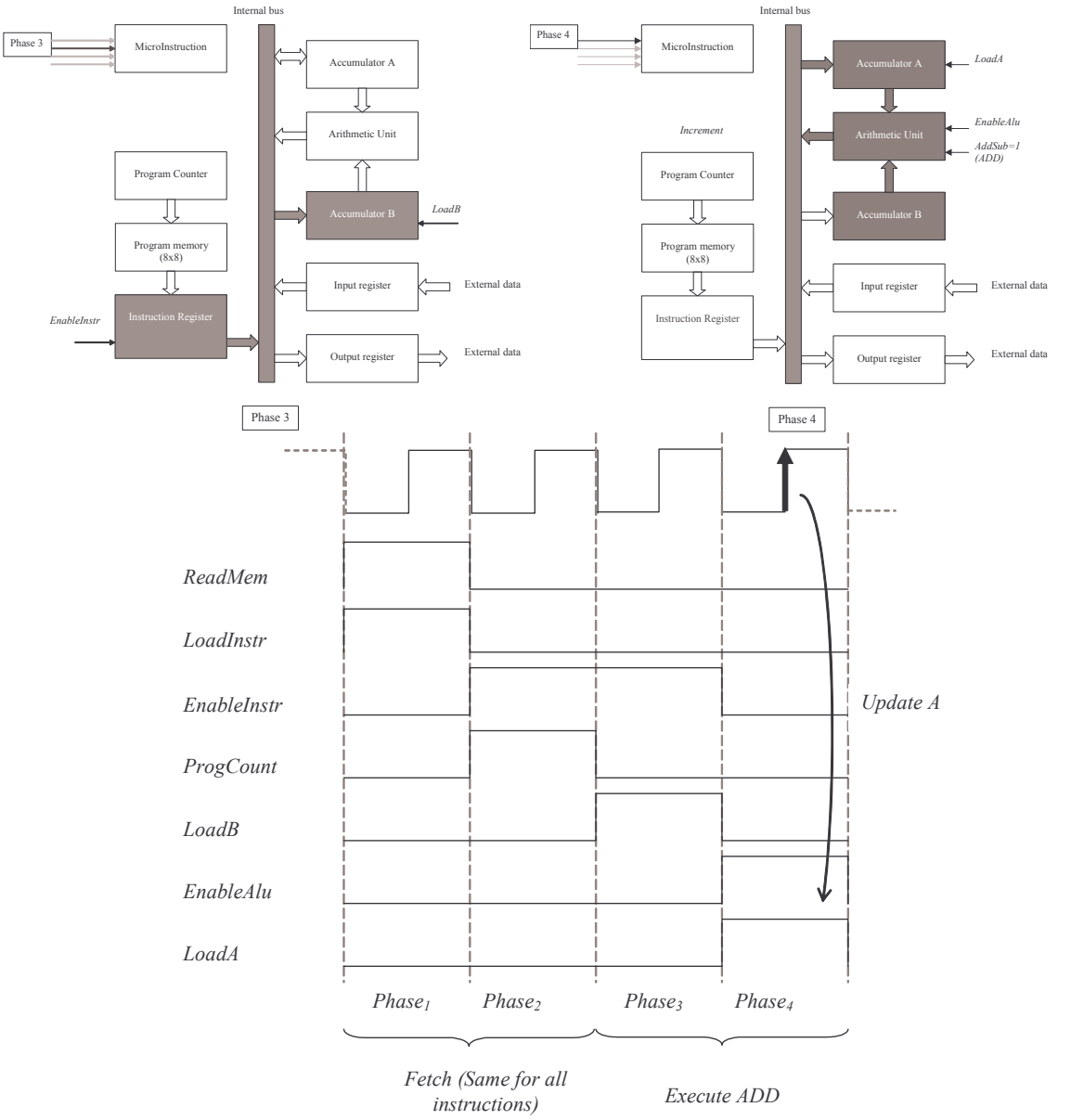


Figure 4-7: Execution of the microinstructions corresponding to the ADD instruction

Subtraction (SUB=0010)

The execution phase of the subtraction instruction is identical to that of the addition instruction. The only difference is that the "AddSub" signal is set to 0, which means "Subtract".

Get Input (In=0100)

The content of the input port is transferred to accumulator A during phase 3 (Figure 4-8). There is nothing to do in phase 4, when all registers remain inactive.

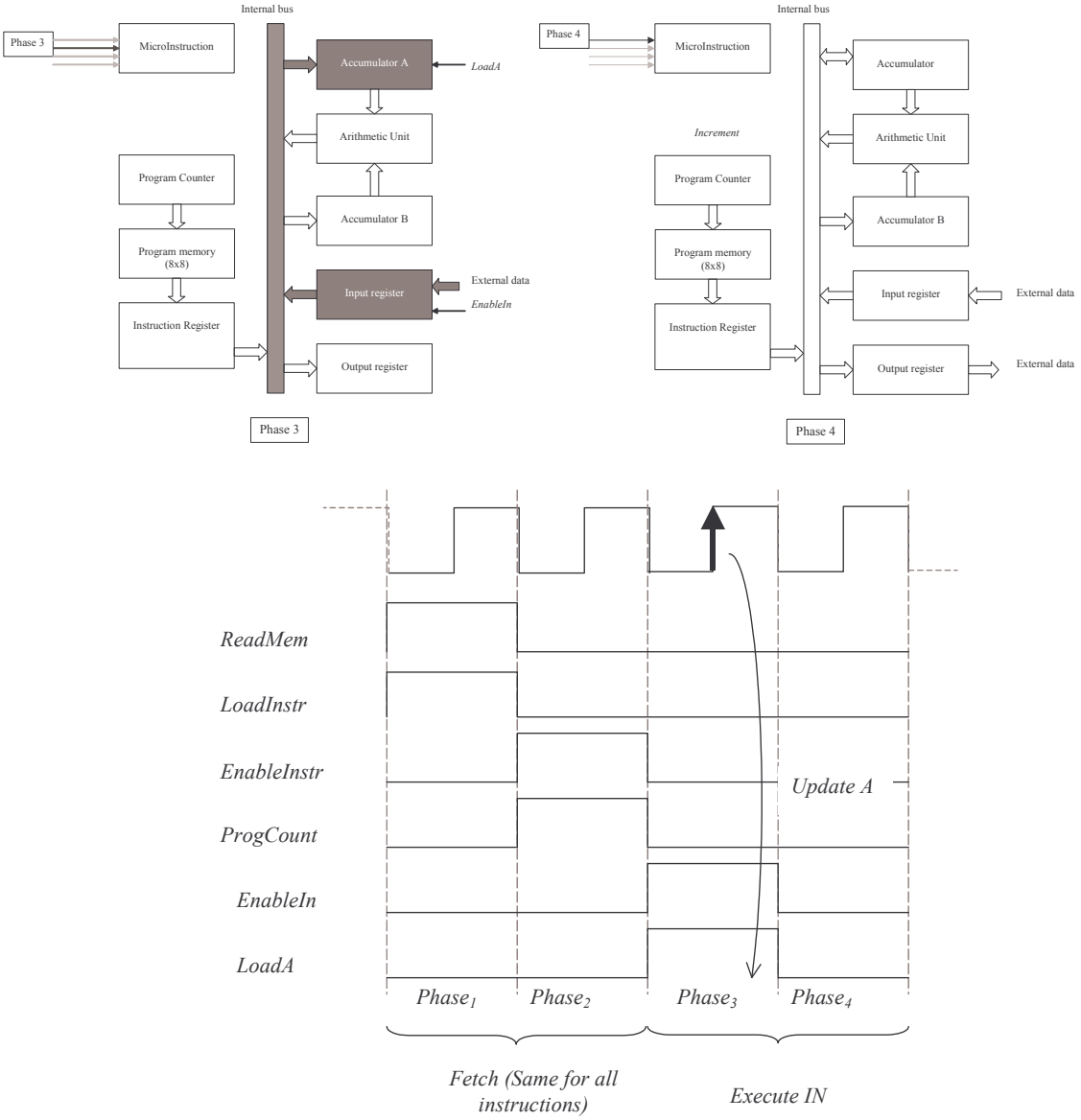


Figure 4-8: Execution of the microinstructions corresponding to the IN instruction

Give Output (OUT=0011)

The content of accumulator A is transferred to the output port via the internal bus during phase 3. The output port memorizes the accumulator value and makes it available to external devices thanks to its four registers. The processor is inactive during phase 4.

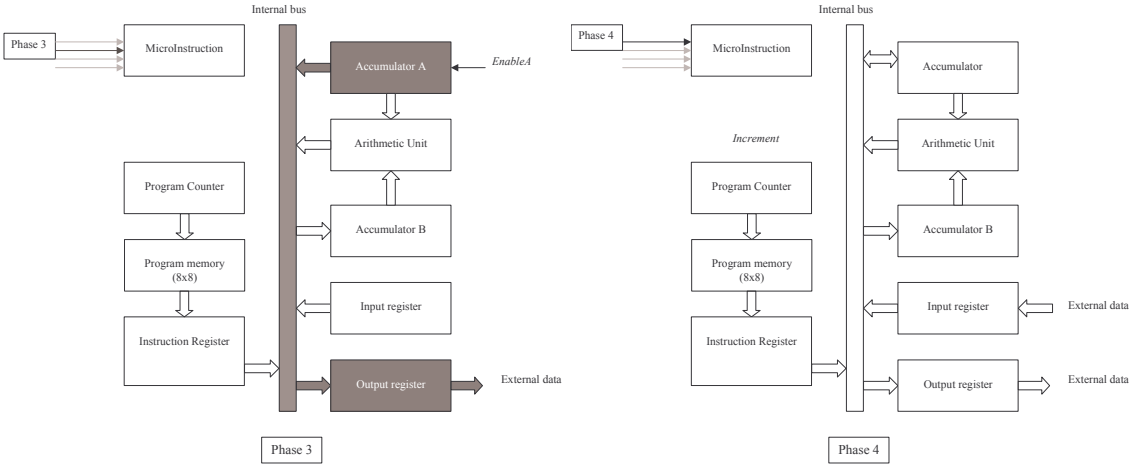
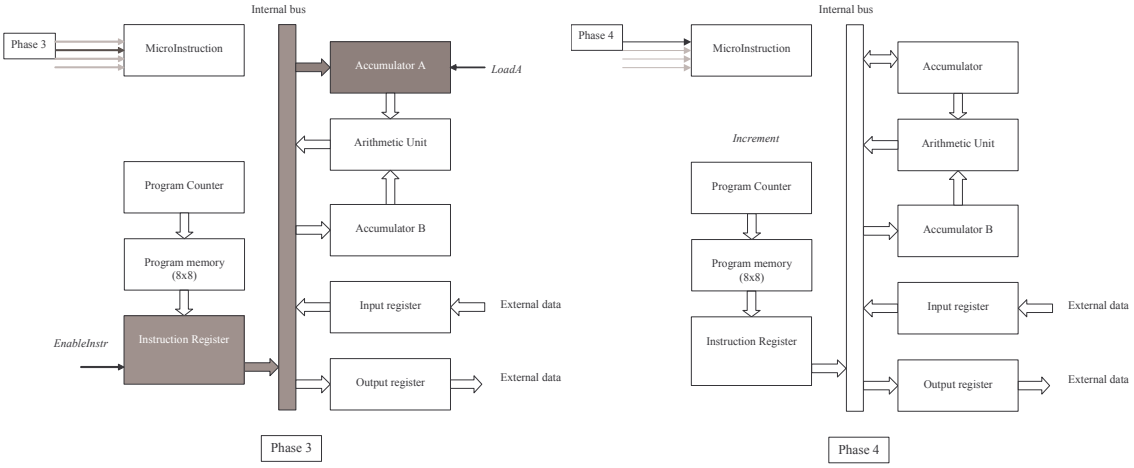


Figure 4-9: Execution of the microinstructions corresponding to the OUT instruction

Load Instruction (LDA=0101)

The load instruction transfers the 4-bit data given as a parameter of the LDA instruction to accumulator A. For example, the instruction “LDA 9” transfers the value 9 (1001 in binary format) to accumulator A. In Figure 4-10, the four least significant bits of the instruction register are placed on the internal bus and then transferred to accumulator A. As a result, the updated value of A is 1001. There is no activity during phase 4.



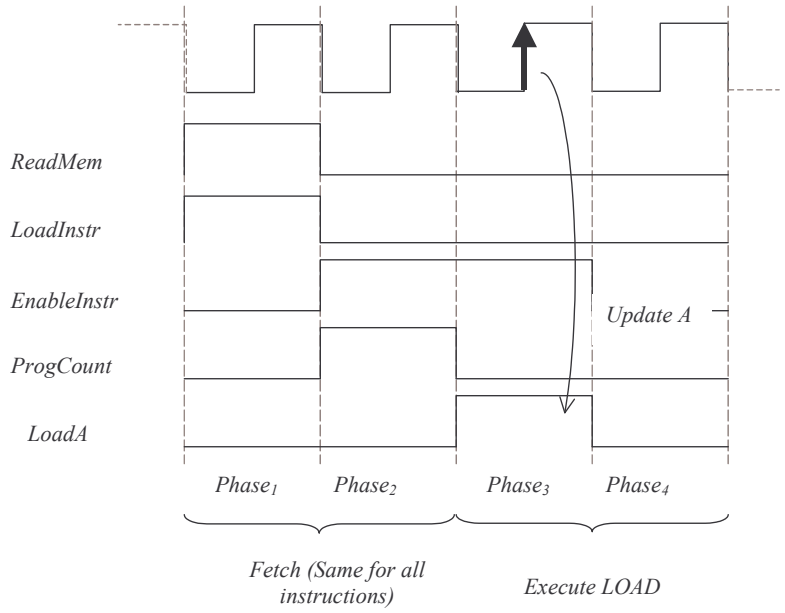


Figure 4-10: The microinstruction during phase 3 executes the load operation. During phase 4, the processor is inactive.

5. Basic Block design

The structure of each sub-block of the microprocessor is presented in detail here.

Accumulator A

The accumulator is composed of four edge-sensitive D flip-flops as shown in Figure 4-11. The register output is available through AluA0..AluA3 for the ADD and SUB operations. The content of A is transferred to the internal bus when “EnableA” is asserted. We use tri-state inverters to facilitate access to the internal bus. The “latchA” signal authorizes the transfer of input data (here, a keyboard) to accumulator A at the falling edge of the main clock.

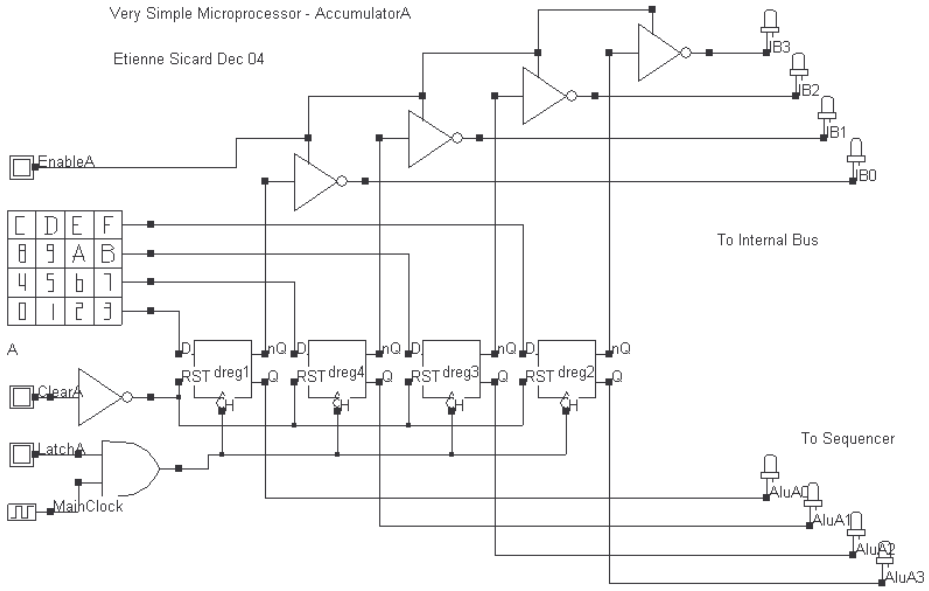


Figure 4-11: Structure of the accumulator A showing its connections to the internal bus and the arithmetic unit (Vsm-AccumulatorA.sch)

Accumulator B

Like accumulator A, the accumulator B is composed of four edge-sensitive D flip-flops as shown in Figure 4-12. The register output is available through AluB0..AluB3 for the ADD and SUB operations. The “latchB” signal authorizes the transfer of input data (here, a keyboard) to accumulator B at the falling edge of the main clock.

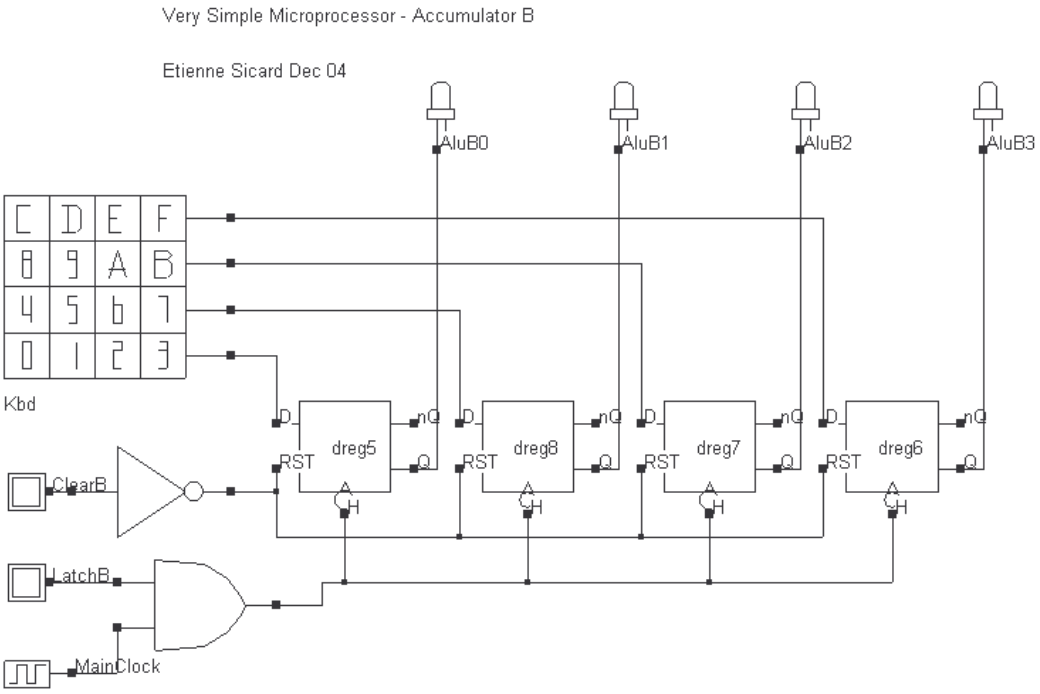


Figure 4-12: Structure of the accumulator B showing its connections to the arithmetic unit (Vsm-AccumulatorB).sch

Add/subtract Block

The addition is based on the full-adder sub-circuit that has been described in Chapter 7 of the book "Basic CMOS cell design" by the same authors [2]. The full-adder consists of a set of XOR gates for generating the "SUM" output and a complex gate for generating the "Carry" output, as shown in Figure 4-13.

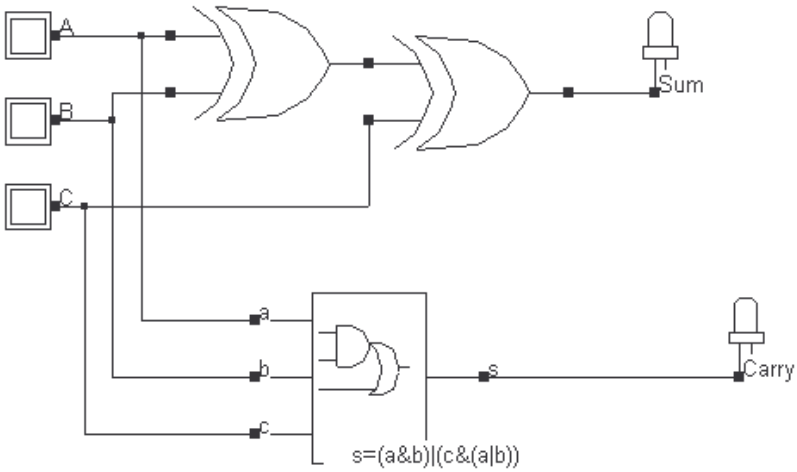


Figure 4-13: The internal structure of the full-adder (Vsm-fullAdder.sch)

Adding two 4-bit numbers requires four cascaded full-adders as illustrated in Figure 4-14. The carry signal propagates from the lower stage to the upper stage in order to perform the complete add operation.

To subtract two numbers (B-A in this case) using the same full-adders, we need to build two supplementary things:

- A circuit that produces the 1's complement of A
- A small circuit that sets the initial carry to 1.

One approach consists of using multiplexer circuits, which may be found in the symbol palette, advanced symbol menu, sub-menu "Switches". When "Sel" equals to 0, the input i0 is transferred to the output, otherwise, i1 is transferred to the output. Consequently, "AddSub=0" corresponds to the transfer of A to the adder chain (Add operation), while "AddSub=1" corresponds to the transfer of ~A to the adder chain (Subtract operation).

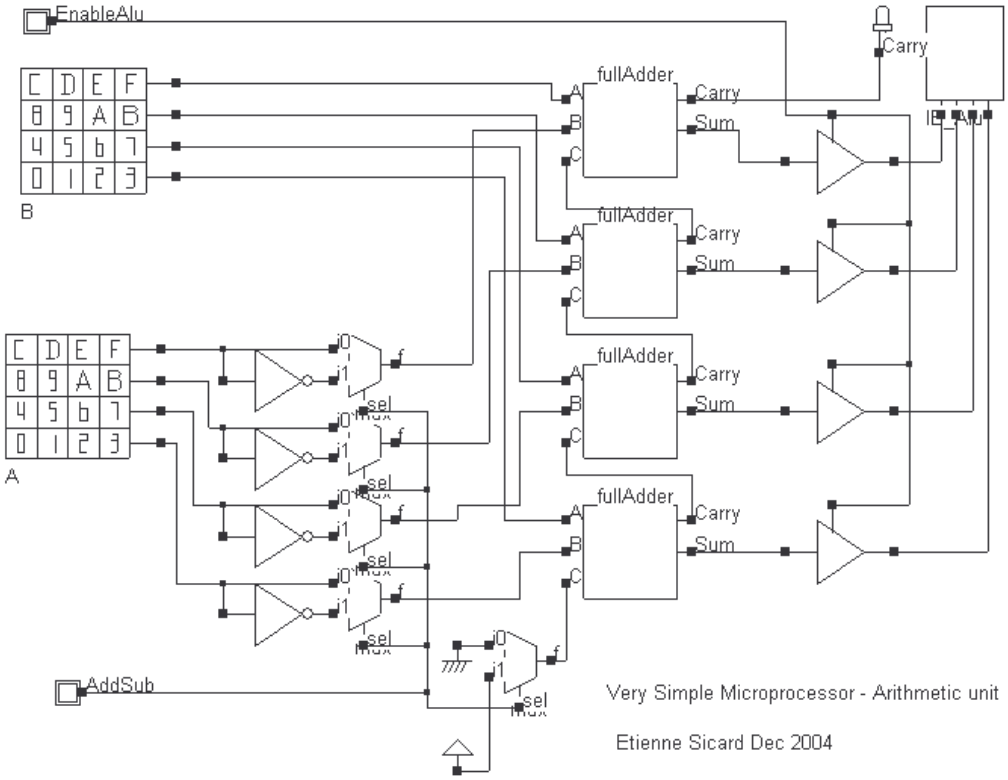


Figure 4-14: Structure of the arithmetic unit which performs the ADD and SUB operations (Vsm-ArithmeticUnit.sch)

At this point, it sounds very interesting to connect the accumulators and the arithmetic unit in order to perform manually what the microprocessor will later do with its internal sequencer. The circuit made of the accumulators A, B and the arithmetic unit is shown in Figure 4-15. The two keyboards serves as inputs A and B, the displays are placed on the output bus and the arithmetic unit's connections to the internal bus.

Trying to operate this simple circuit would be a very interesting introduction to the microprocessor's operation. Below is the set of actions we need to perform sequentially in order to add two numbers:

- De-active the main Reset. Initially the Reset pin is set to 0 (Default value at the start), which corresponds to an active Reset. Both registers A and B are cleared (A=0, B=0). Nothing will work until you set the button "~MainReset" to 1.
- Load the desired value on A. Click on a digit on the lower keyboard named "A", for example 3. Click "LatchA" and wait at least one complete cycle of the main clock. The accumulator A stores 3 at the falling edge of the clock.
- Load the desired value on B. Click on a digit on the upper keyboard named "B", for example 2. Click "LatchB" and wait at least one complete cycle of the main clock. The accumulator B stores 2 at the falling edge of the clock. The arithmetic unit computes the sum A+B as "AddSub" is set by default to 0, which corresponds to the ADD instruction. However the result is not displayed as "EnableAlu" is 0.
- Set "EnableAlu" to 1 to display the result "5", as shown in Figure 4-15.

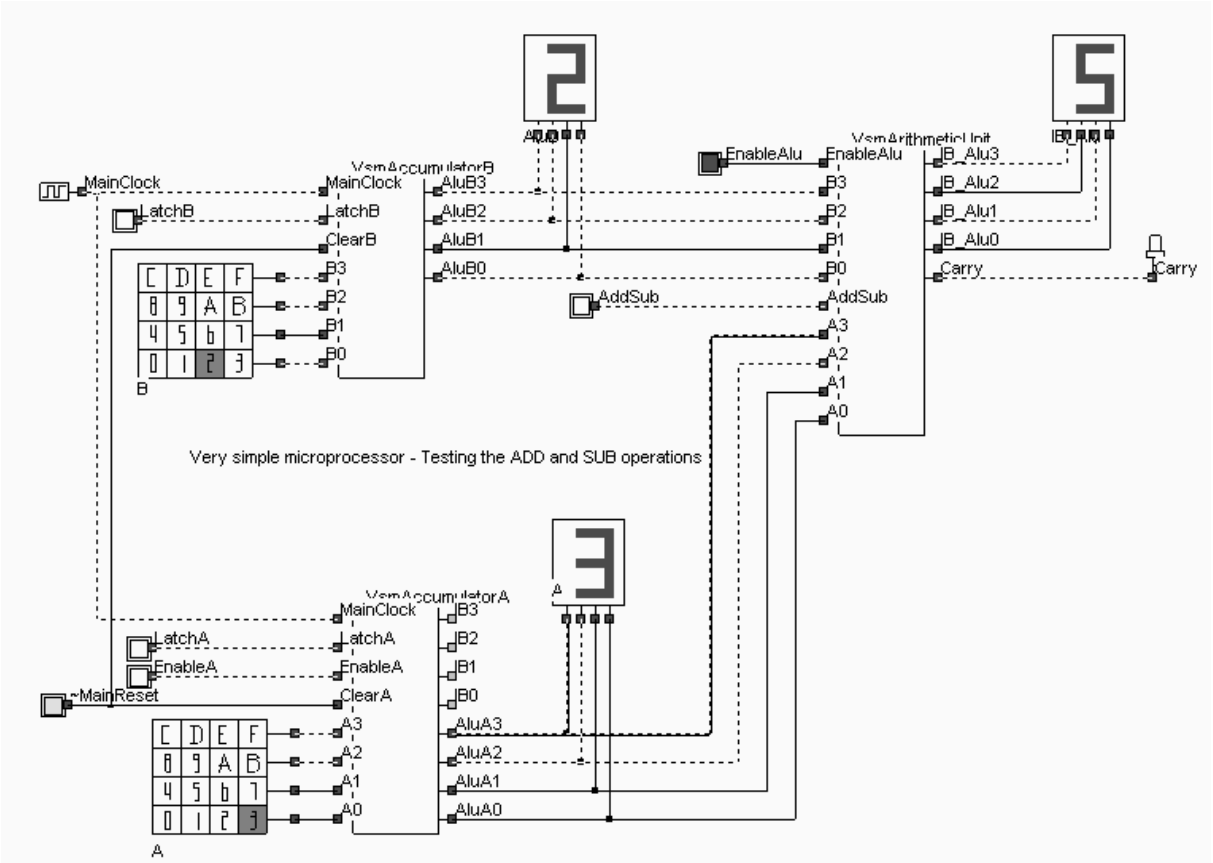


Figure 4-15: The connection between accumulators A, B and the arithmetic unit to test the ADD and SUB instructions (Vsm-RegARegBAlu.SCH)

The input register

The input register is a simple set of 3-state buffers as shown in Figure 4-16. There is no need for D-registers as the input will be directly transferred to accumulator A.

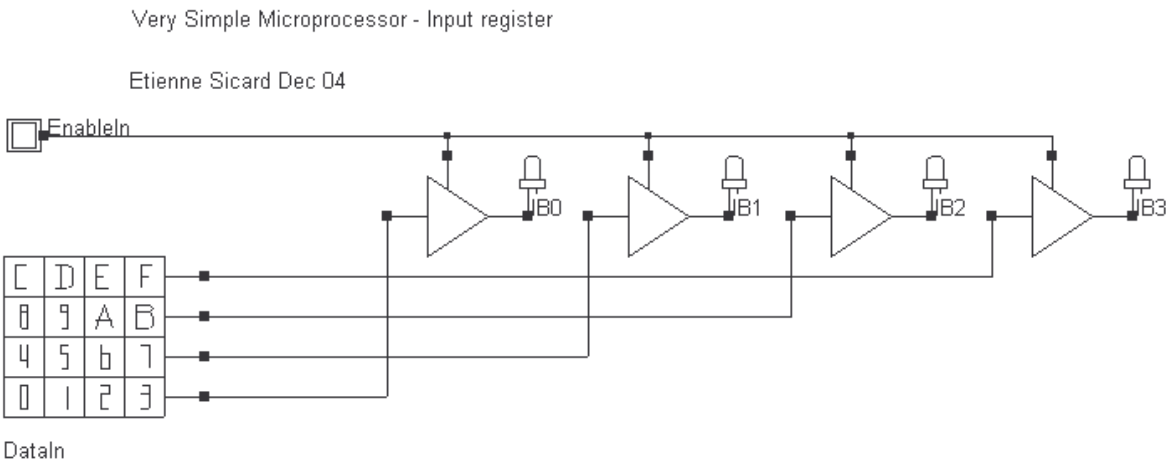


Figure 4-16: The input register (Vsm-InRegister.SCH)

The output Register

The output register is composed of D-register cells as shown in Figure 4-17. On the positive edge of the clock, the data is saved in the registers. It is very important that the data is stored on the positive edge of the clock during phase 3, and not on the negative edge. The later would give rise to synchronization conflicts. Therefore, a NAND gate is used to make the circuit sensitive to the rising edge of the main clock, as shown in Figure 4-18.

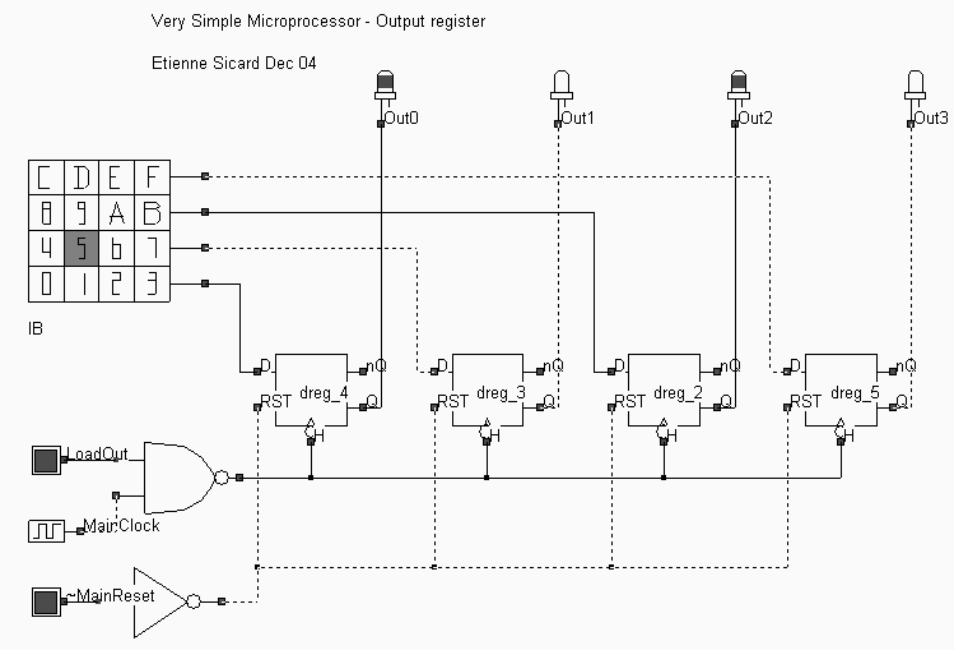


Figure 4-17: Internal structure of the output register (Vsm-OutRegister.SCH)

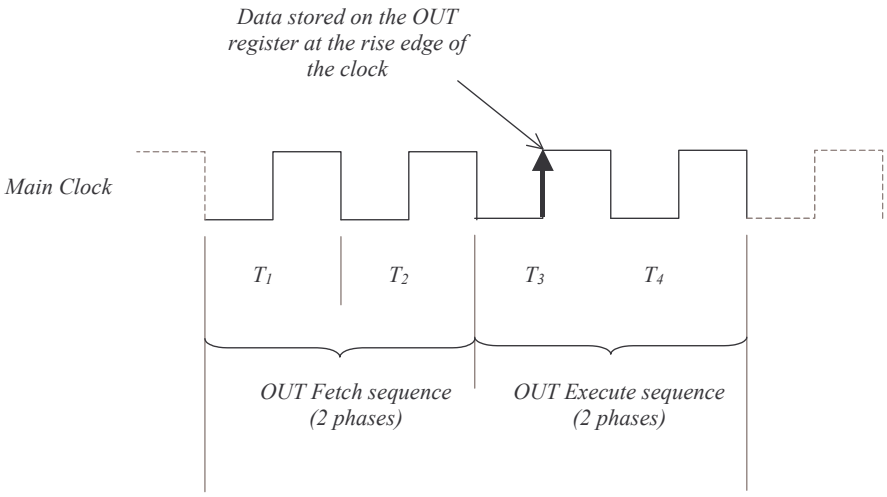


Figure 4-18: The output register must store the data at the rising edge of clock in phase 3

A manual microprocessor

In this paragraph, we propose to build a "manually-controlled" microprocessor which consists of accumulators A, B, and the input and the output registers. The goal of the simulation reported in Figure 4-19 is to transfer the input information (DataIn) to the output port (DataOut). To perform this transfer, we need to enable the input port (EnableIn=1) and then enable the output port (EnableOut=1). At the next rising edge of the main clock, the contents of the input keyboard (5 in this case) will appear on the display connected to the output register. Several other transfers may be performed:

- Input register to accumulator A
- Input register to accumulator B
- Result of the addition of A and B to the output port

The arrow symbol (Symbol menu "Advanced", symbol "Arrow") is used to ease electrical connections for the clock and reset signals. In the example shown in Figure 19, connections are made automatically among all arrows having the same name. Double click the "Clk" arrow symbol in Figure 4-19 to access the arrow name which identifies the electrical net. In the example shown in Figure 4-20, we built two different electrical connections, one called "Clk" and the other called "Rst". Notice that the electrical node names are not case sensitive.

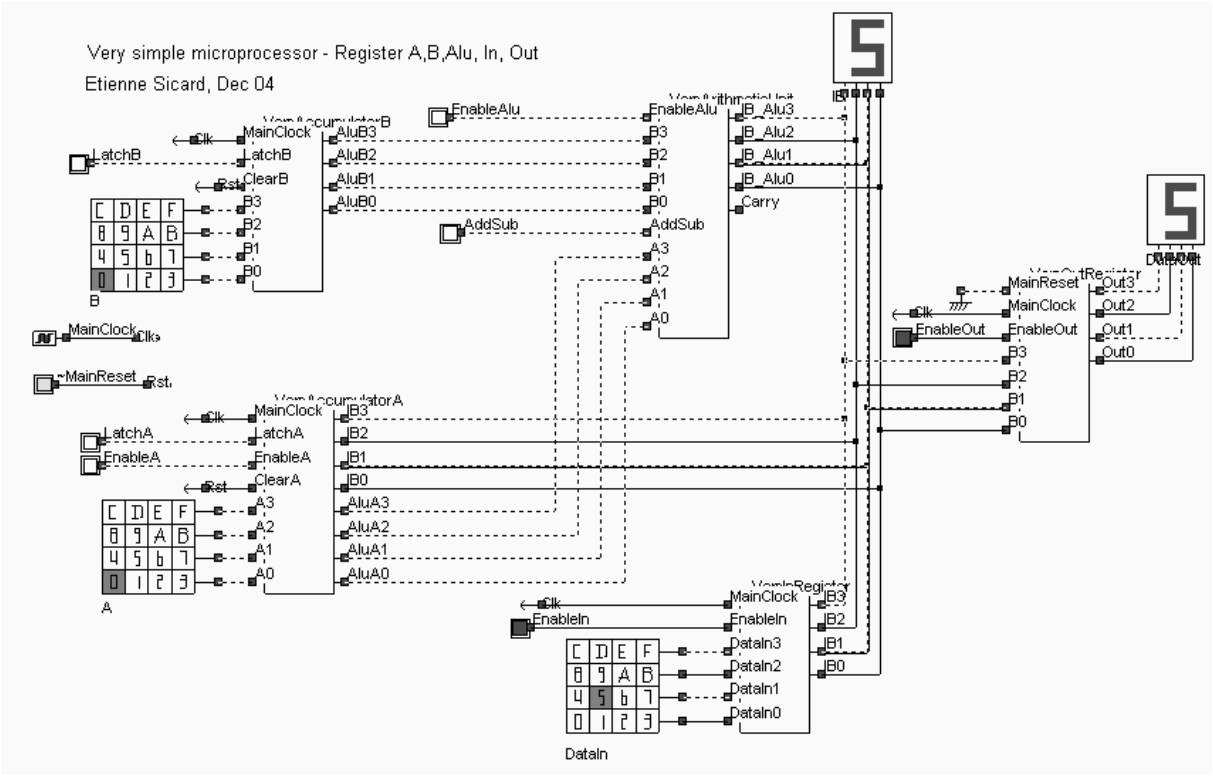


Figure 4-19: A manually-controlled microprocessor (Vsm-RegARegBAluInOut.SCH)

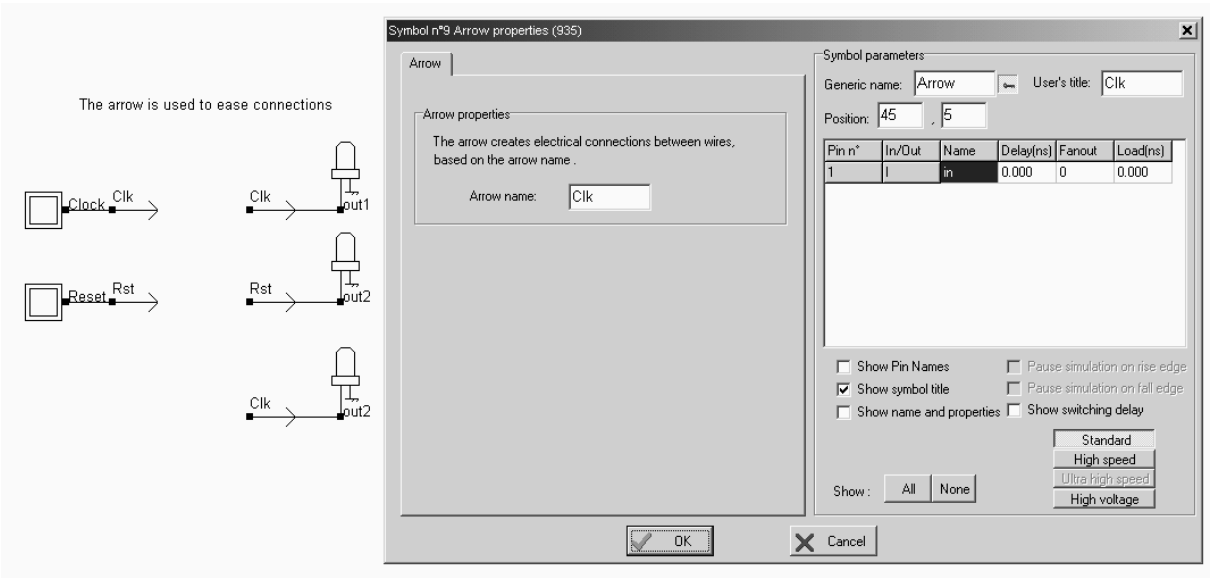


Figure 4-20: Building arrow connections to ease the electrical wiring of the main signals (Vsm_arrow.SCH)

The Phase Generator

In order to transform the previous 'manual' microprocessor into a fully programmable microprocessor, we need to build several circuits to generate the appropriate control signals. First, the phase counter must produce the four phase signals Phase0 to Phase3 at the negative edge of the clock. The counter must be reset by an active low

"Clear" signal. The design of the phase counter is based on edge-sensitive latches and XOR gates as shown in Figure 4-21.

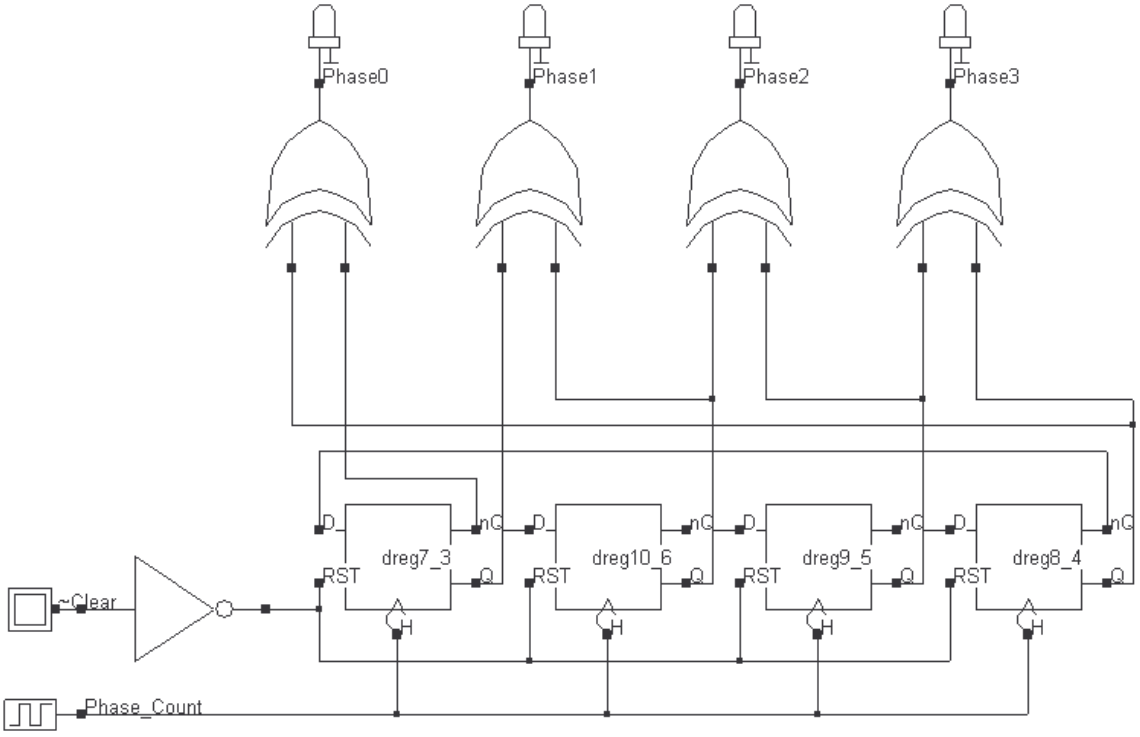


Figure 4-21: The phase counter structure (Vsm-RingCounter4.sch)

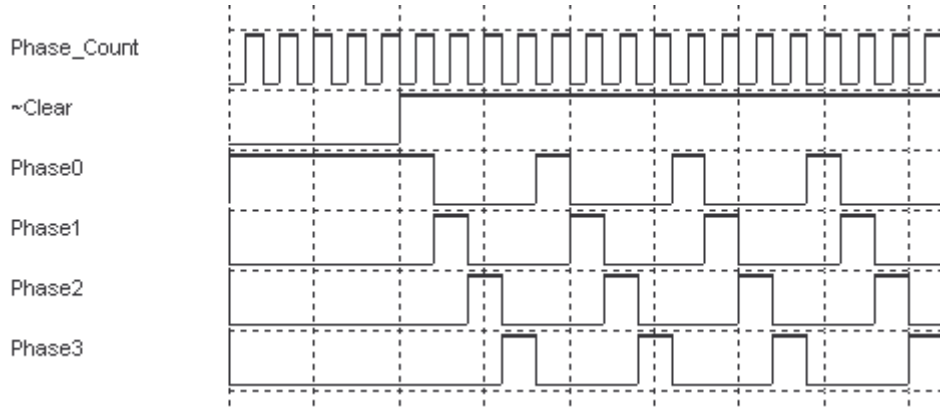


Figure 4-22: Simulation of the phase counter (Vsm-RingCounter4.sch)

When the "Clear" signal becomes inactive (logic high) the phases appear sequentially.

Program Counter 0 to 15

The program counter plays a very important role in the microprocessor as it supplies the main program memory with the address of the active instruction (Figure 4-23). At the start, the program counter is 0. At the end of each instruction the program counter is incremented in order to select the next instruction.

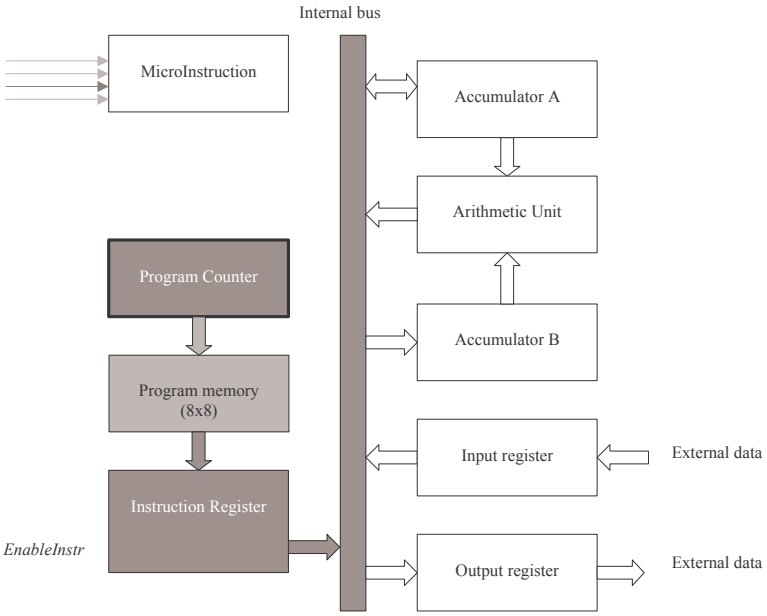


Figure 4-23: The program counter supplies the program memory with the address of the active instruction

One simple way to build a 0-to-15 counter is to use a cascaded chain of edge-sensitive D flip-flops, as shown in Figure 4-24. The circuit is very simple, but works asynchronously. This means that due to propagation delays between stages, some intermediate results appear on the display for a very short period of time. These glitches have no impact on the microprocessor operation as the counter is incremented during phase 2 of the microinstruction sequence, and is only exploited during phase 1 of the next instruction to load the instruction register.

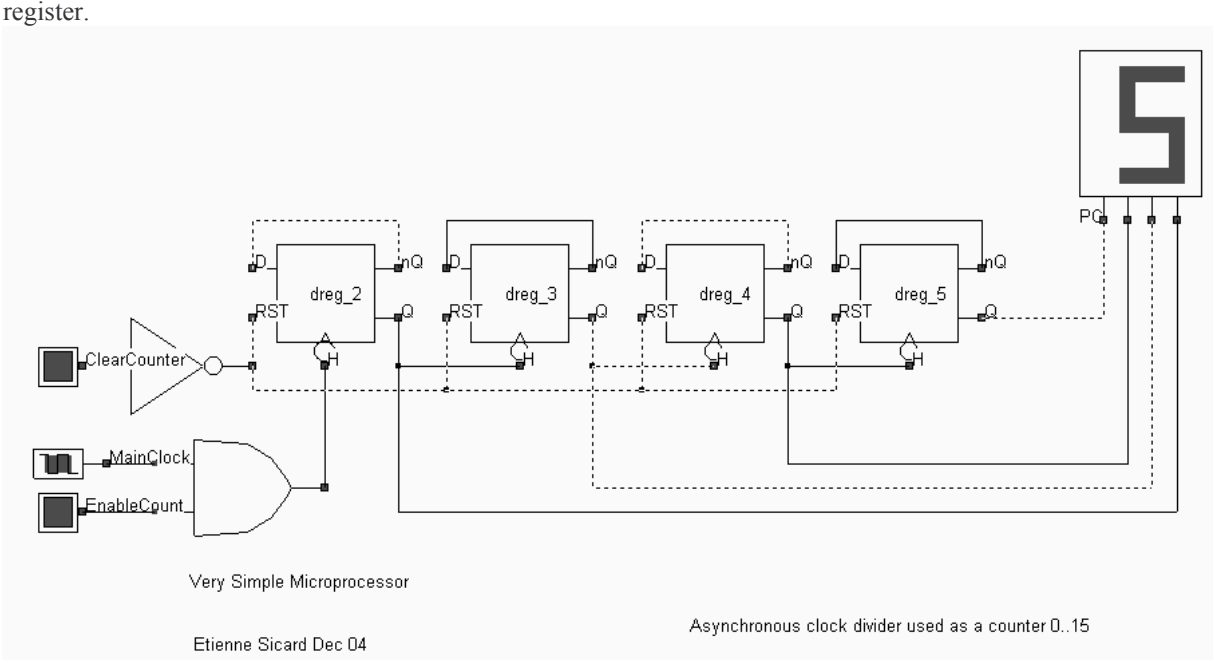


Figure 4-24: The program counter at work. Counting is enabled only during phase 2, at the falling edge of the main clock (Vsm-Counter16.SCH)

The Instruction Register

The instruction register stores the instruction being executed. The 8-bit information is split into two parts: the most significant bits correspond to the instruction code, while the least significant bits are the data. The instruction code is stored in the four D-registers situated at the bottom of Figure 25, in order to be available for the microinstruction decoder. The data is stored in four separate D-register cells and can be made available on the internal bus. The instruction register keeps a copy of the current instruction and releases the main memory, which can be accessed later for both read or write operation.

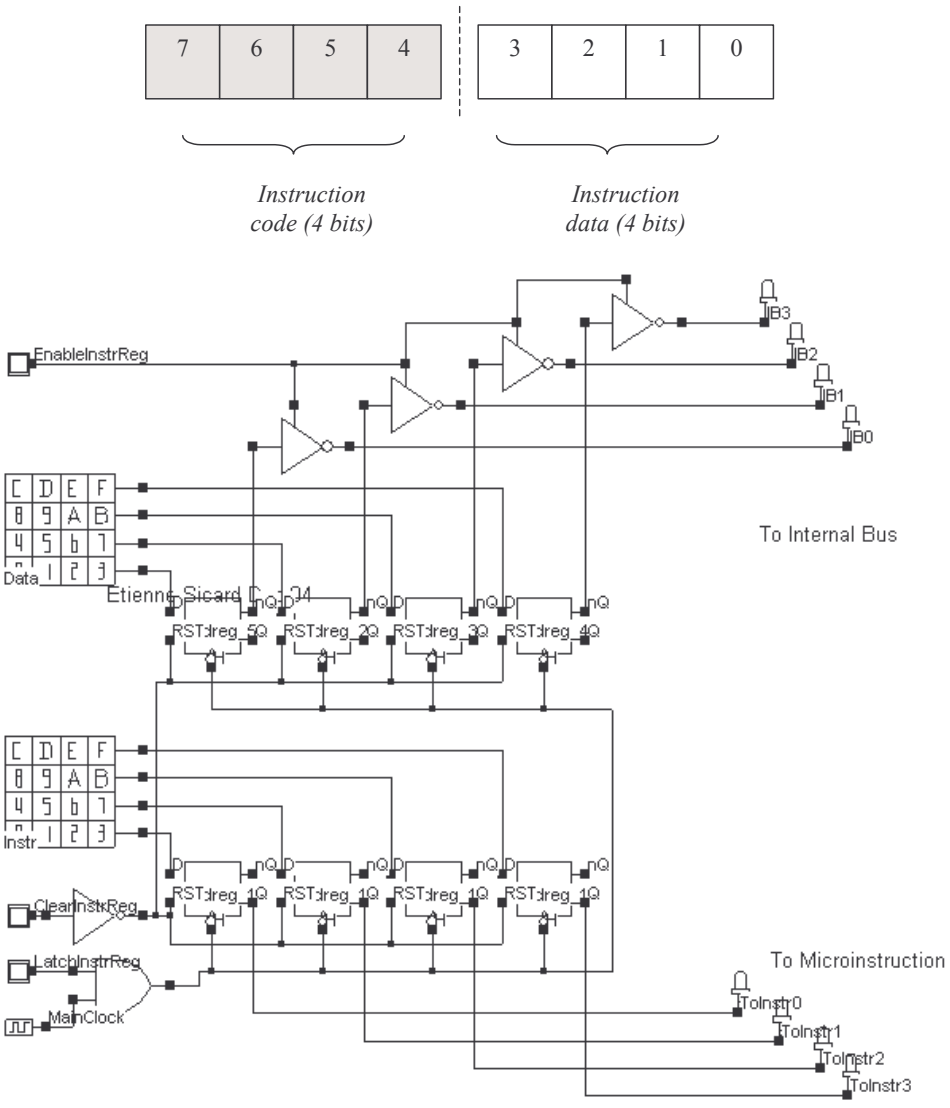


Figure 4-25: The instruction register stores the contents of the memory and separates the code part (lower registers) from the data part (upper registers) (Vsm-InstructionReg.SCH)

The MicroInstruction Controller

The microinstruction controller is the 'heart' of the microprocessor. It generates the most important signals for controlling the operation of the processor, for example the 'Enable' and 'latch' signals. The design of the microinstruction controller is shown in Figure 4-26. The input to the microinstruction controller is the instruction code from the instruction register plus the phase information from the phase counter. The 4-input AND gates serve as instruction decoders. For example, the instruction 0000 turns on the upper AND gate, which corresponds to the NOP instruction. Notice that pahes0 and phase1 are not connected to the instruction decoder. This is because the first two phases are not dependent on the instruction itself. Then, depending on the type of instruction, the desired control signals are set to 1 if active, or kept at 0 to be inactive.

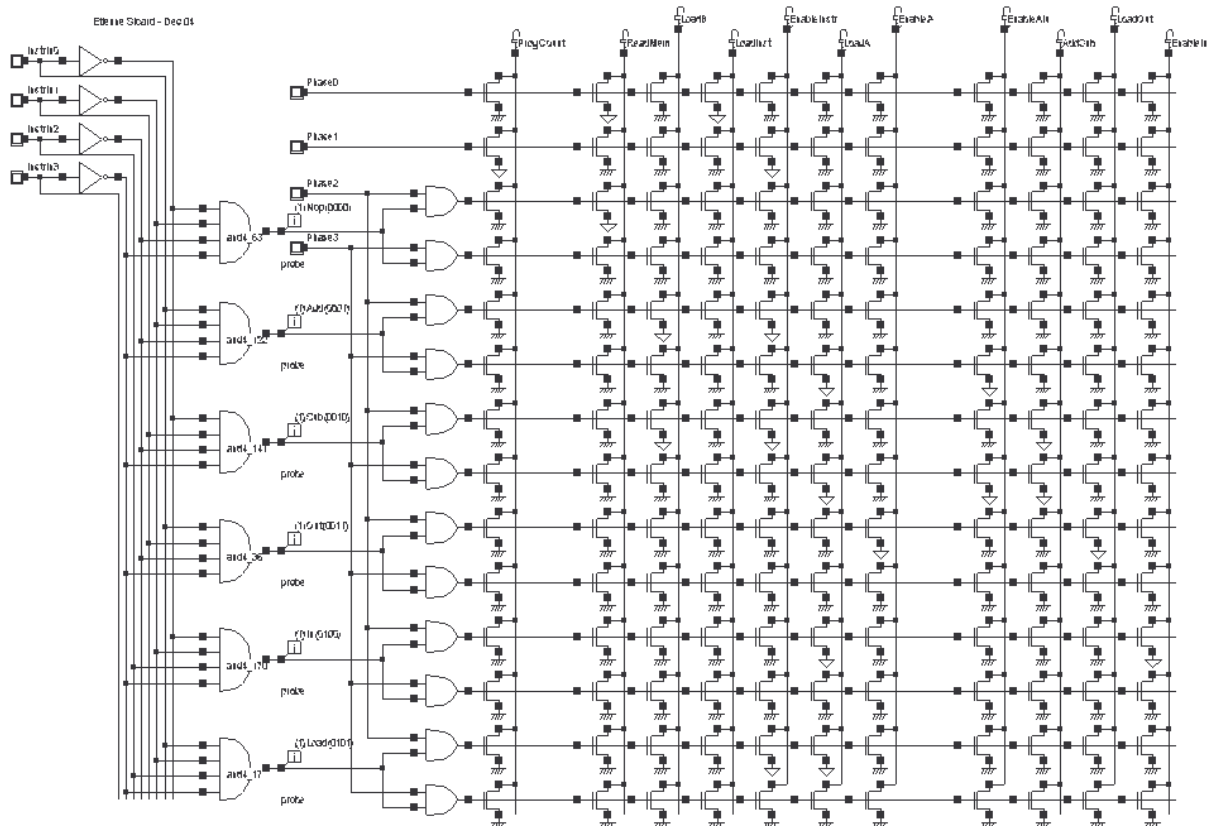


Figure 4-26: The control signals activated by the microinstruction controller during the first two time phases are same for all instructions and depend on the instruction code during the two last phases

The Complete Microprocessor

It is time now to connect all the sub-circuits together and test the entire microprocessor. Each of these sub-circuits has been embedded into a symbol where only the input and output pins appear. The complete circuit is shown in Figure 4-27. We should keep in mind that this is only a "very simple" and very low complexity microprocessor. Before starting the simulation, we must load the program into the memory. The program shown in Table 4-5 has been written into the microprocessor's memory.

Mnemonic	OpCode (binary)	OpCode (hexa)
LDA 1	0101 | 0001	0x51
ADD 2	0001 | 0010	0x12
OUT	0011 | 0000	0x30

Table 4-5: The code stored into the program memory

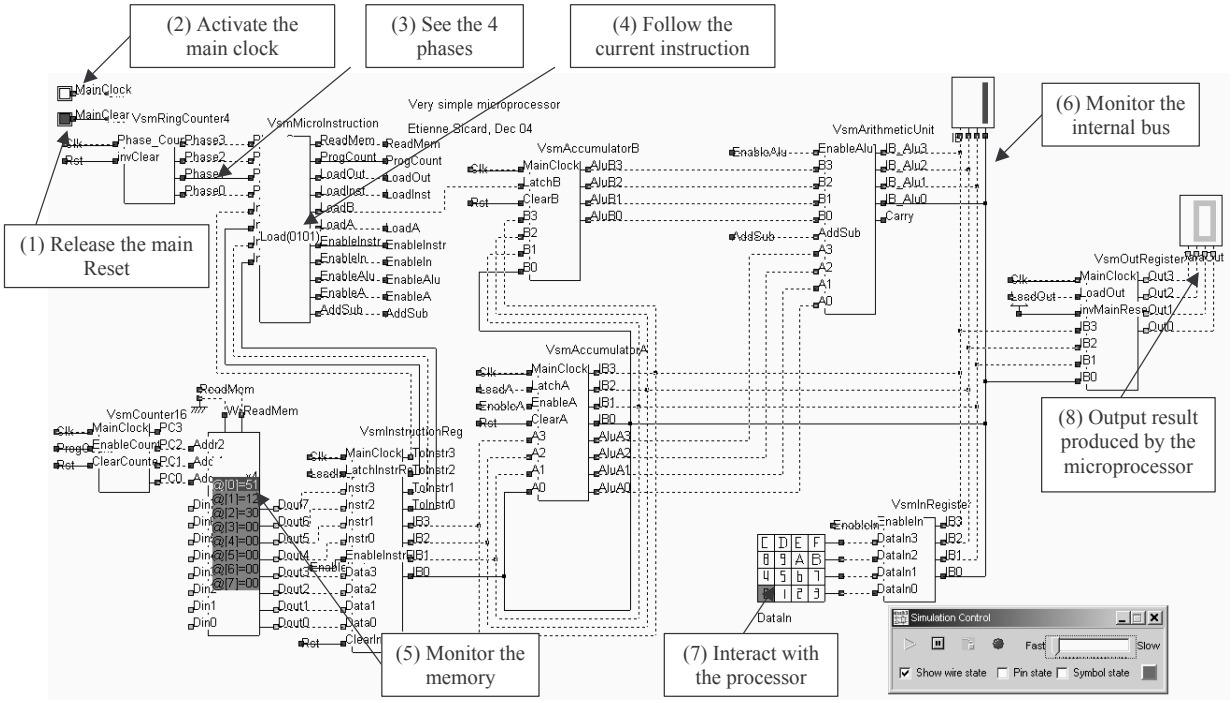


Figure 4-27: The microprocessor circuit ready for simulation (Vsm-Microprocessor.SCH)

Once simulation has started, there are several things to do in order to run the code:

- De-active the reset (1)
- Click on the main clock (2)
- At each active edge of the clock, observe the phase counter shifting from phase0 to phase1, phase2, phase3 and back to phase0 (3).
- Starting in phase 2, the instruction is loaded into the microinstruction controller. The active instruction appears as shown in (4), which corresponds here to “Load (0101)”.
- You can monitor the memory contents and the active memory location (5).
- Also worth monitoring is the internal bus (6).
- If required by the program, you can enter data through the keyboard named DataIn (7)
- If the “OUT” instruction is running the result should appear on the output display (8).

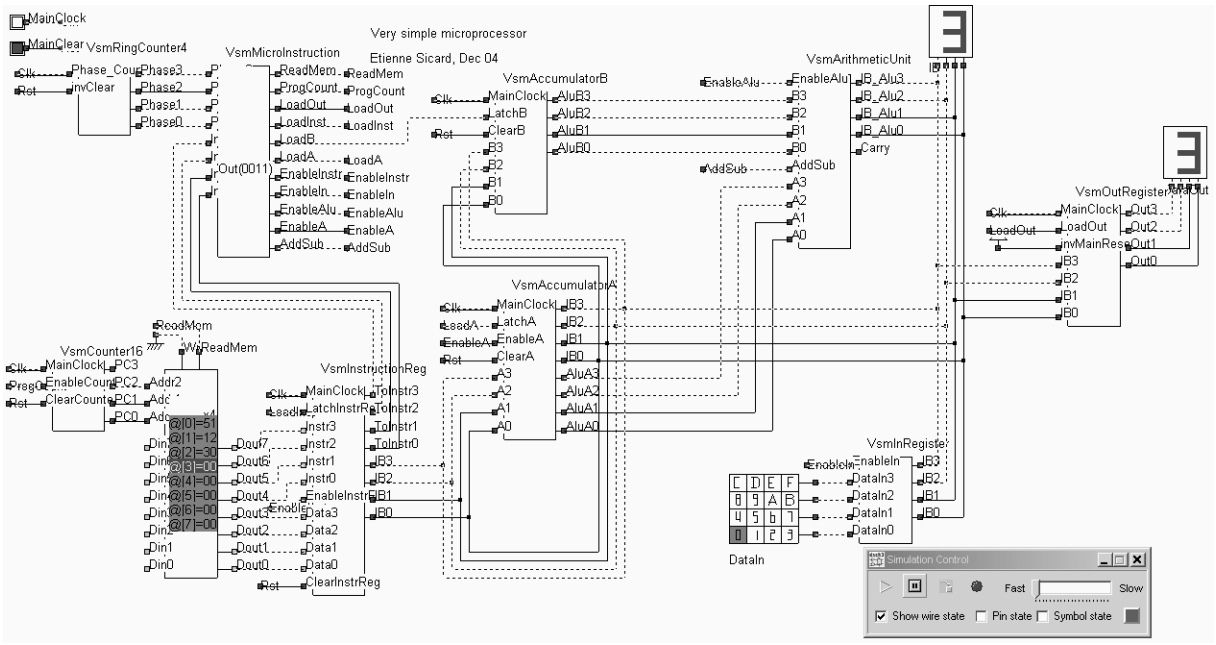


Figure 4-28: Final result of the addition of 1 and 2 using the program proposed in Table 6 (Vsm-Microprocessor.SCH)

Memory Move

One important feature NOT handled by the very simple microprocessor is the memory move (MOVE). This instruction transfers the contents of a memory location to accumulator A or vice versa. Why didn't we build this functionality into the first version of our processor? This is because the structure of the memory control and access must be deeply modified and would require a significant amount of supplementary hardware.

Assuming that the MOVE operation transfers the contents of one memory location to A, we need to perform the following sequence of operations: during phase 3, we need to have access to a *new* memory location, whose address is not the one currently stored in the program counter. This means that a new type of access must be provided in the processor from the internal bus to the memory, without altering the contents of the instruction register. The differences between the two structures are displayed in Figure 4-29.

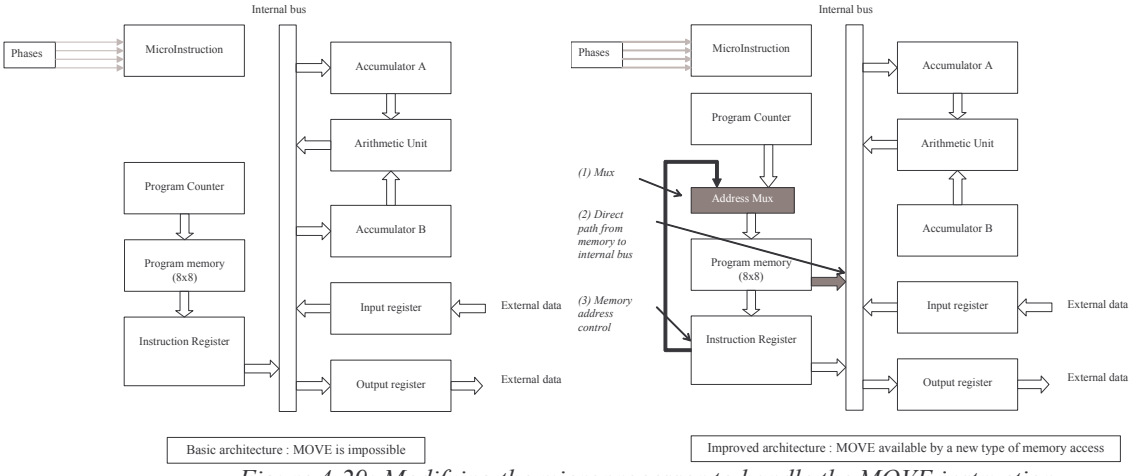


Figure 4-29: Modifying the microprocessor to handle the MOVE instruction

In practice, the MOVE instruction can be incorporated by adding the following:

- a direct path from memory to the internal bus (with its appropriate Enable control),
- a 4-bit address bus from the instruction register to the memory, and
- a multiplexer for selecting a memory address either from the Program Counter or from the Instruction Register.

Physical Implementation

Description of the design flow

The VSM processor has been described and simulated at logic level using DSCH, and saved under the name **vsm-microprocessor.SCH**. It can be converted automatically into layout using MICROWIND. The design flow is detailed in Figure 4-30. First we create a VERILOG description of the VSM processor using the command **File →Make Verilog File**. The resulting text file **vsm-microprocessor.TXT** contains a VERILOG description of the processor. This file can be compiled in MICROWIND using the command **Compile →Compile Verilog File** in order to automatically generate the layout of the processor.

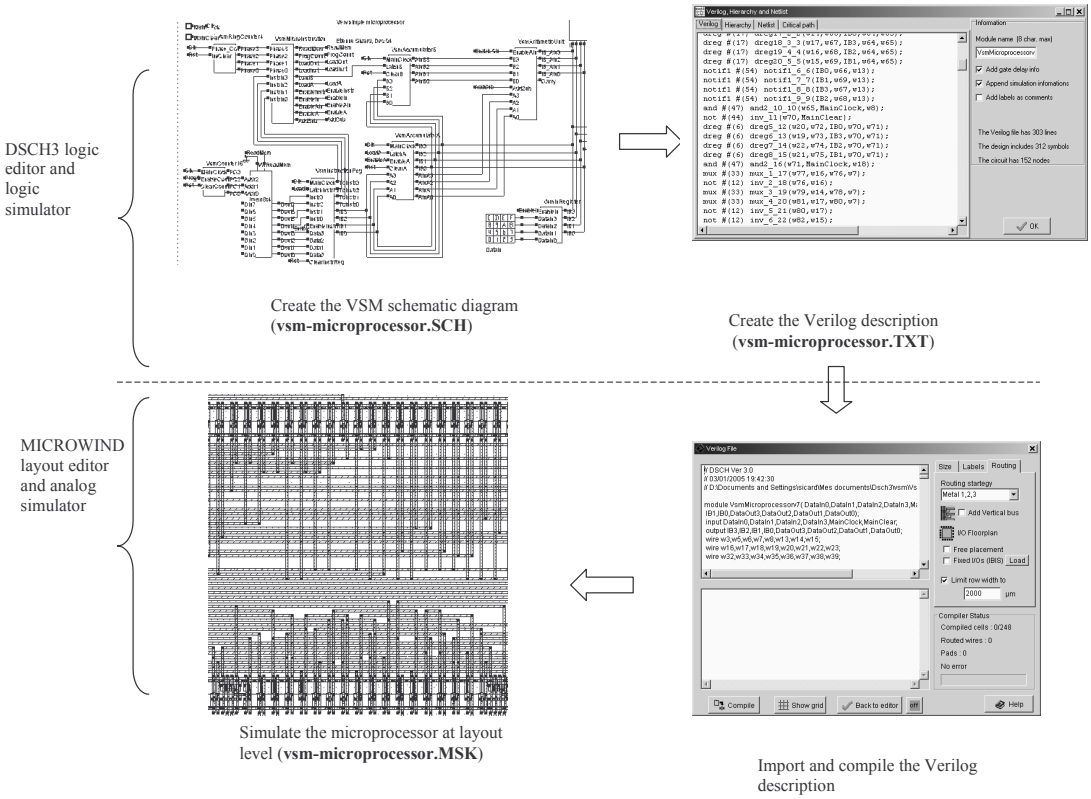


Figure 4-30: Automatically generating the layout of the VSM processor from the logic circuit

VERILOG translation

In its basic version, the microprocessor includes 312 primitives. This relatively small number of devices is due to the fact that the memory symbol is ignored during the translation to Verilog. This is because the memory macro-cell used in the microprocessor design is not a real memory as it does not contain any real memory element such as flip-flops. The warning generated by DSCH during the Verilog translation is shown in Figure 4-31. A partial view of the Verilog description of the VSM (**vsm-microprocessor.TXT**) is shown in Figure 4-32.

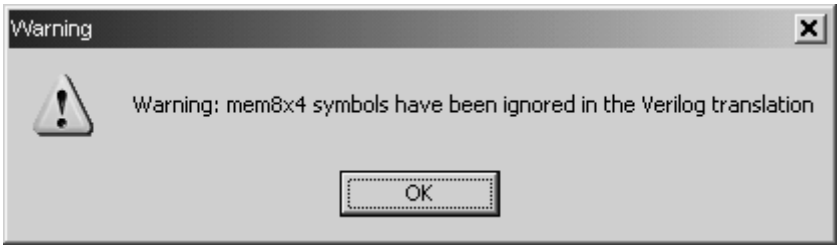


Figure 4-31: Warning concerning the memory macro that has not been translated into a standard VERILOG description

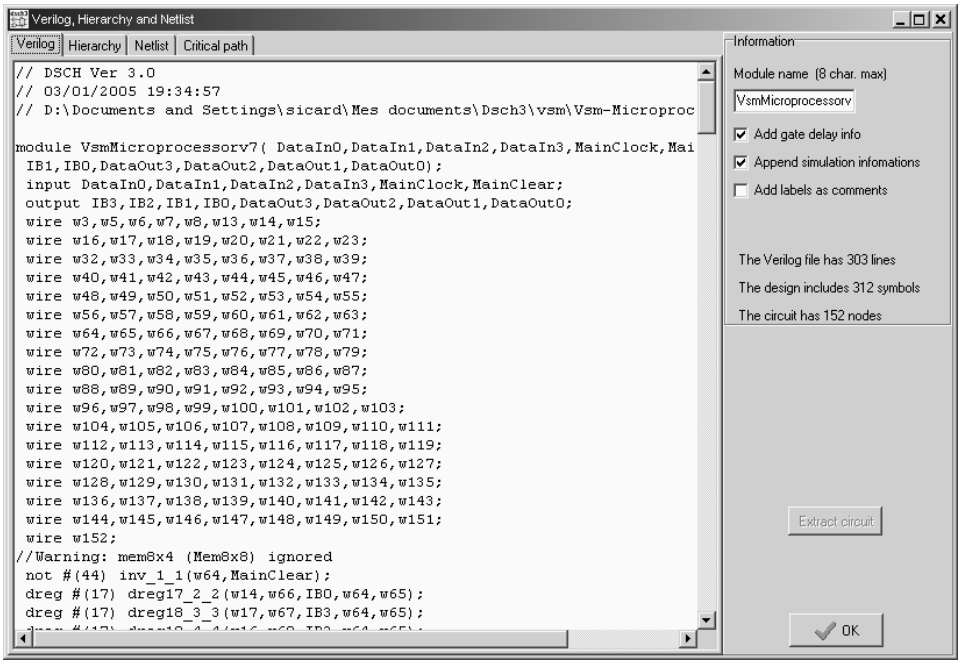


Figure 4-32: A partial view of the VERILOG description of the 4-bit microprocessor (vsm-microprocessor.TXT)

Creating the layout of the complete microprocessor

To generate a complete layout of the microprocessor, we need to design a cell-based 8x8 bit memory that works exactly as the memory macro-cell. This can be done by constructing an array of 8x8 register cells based on very simple ring inverters as shown in Figure 4-33. Data can be written to the memory cell via nmos N1 when the 'Write' control is high. Data is read from the cell when the 'Read' control is high.

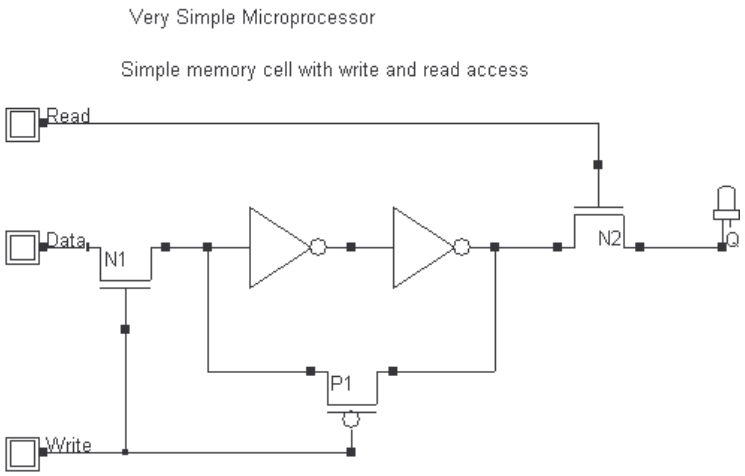


Figure 4-33: Design of a very simple memory cell based on two ring inverters (Vsm-memorycell.sch)

The design of an 8x8-bit memory array is shown in Fig. 4-34. There are eight memory cells in each row for storing the 8-bits of an instruction. At any one time only one memory location (one row) can be accessed by asserting one of the signals MemLocn0-MemLocn7. These 8 signals are generated by the 3-to-8 decoder shown in Fig. 4-35 using the 3-bit address information (Addr2-Addr0). In Figure 4-34, either the Read or the Write signal is asserted for a Read or a Write operation. The complete 8x8 memory including the address decoder is shown in Fig. 4-36.

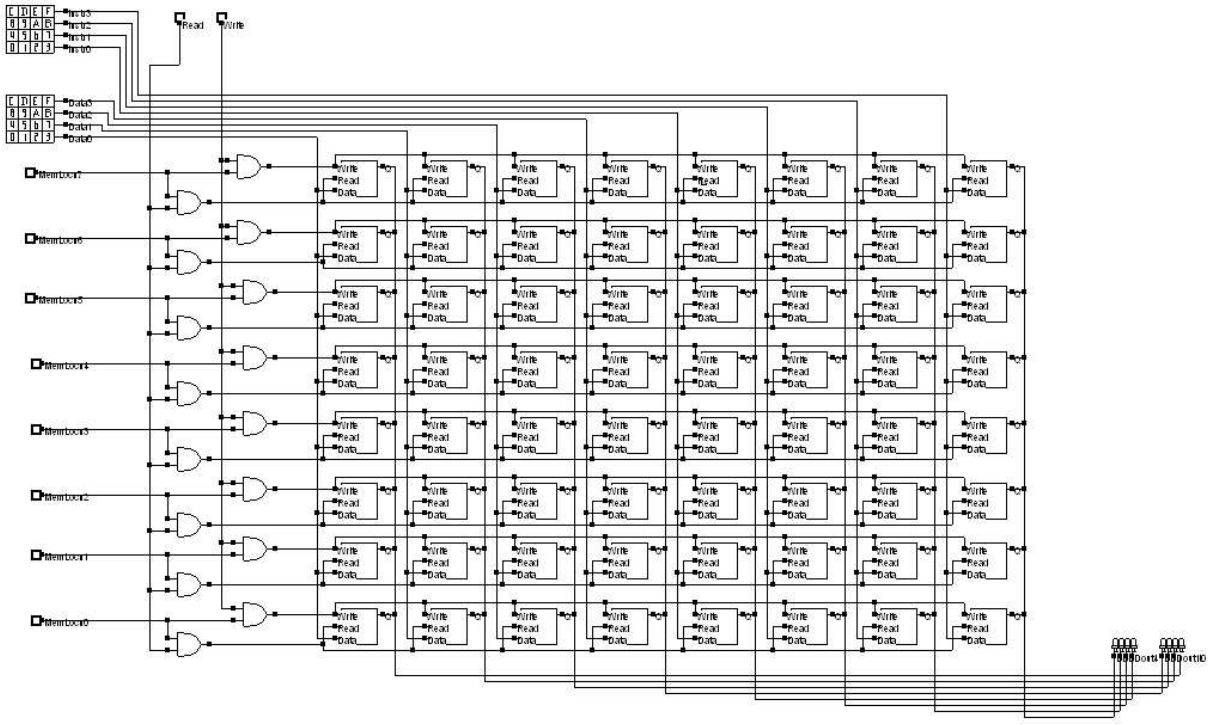


Figure 4-34: An 8x8-bit memory array (Vsm-Mem8x8Array.sch)

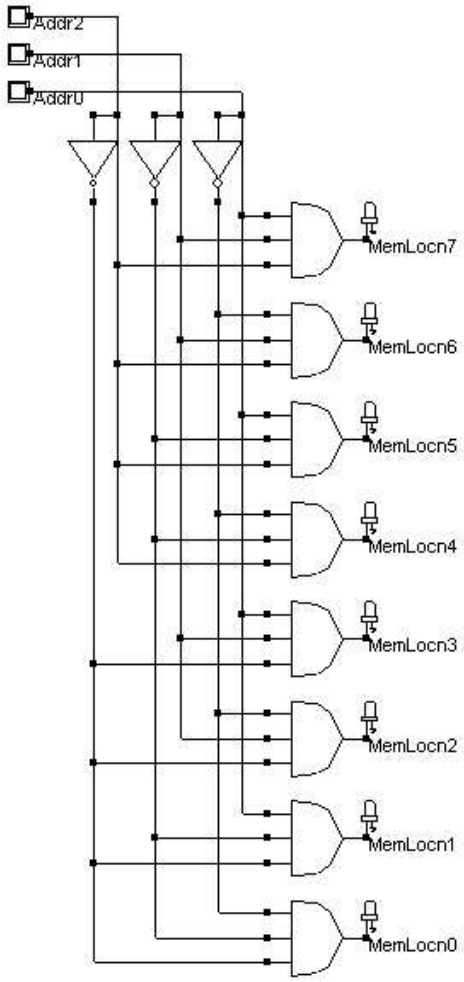


Figure 4-35: A 3-to-8 decoder for memory addressing (Vsm-3to8Decoder.sch)

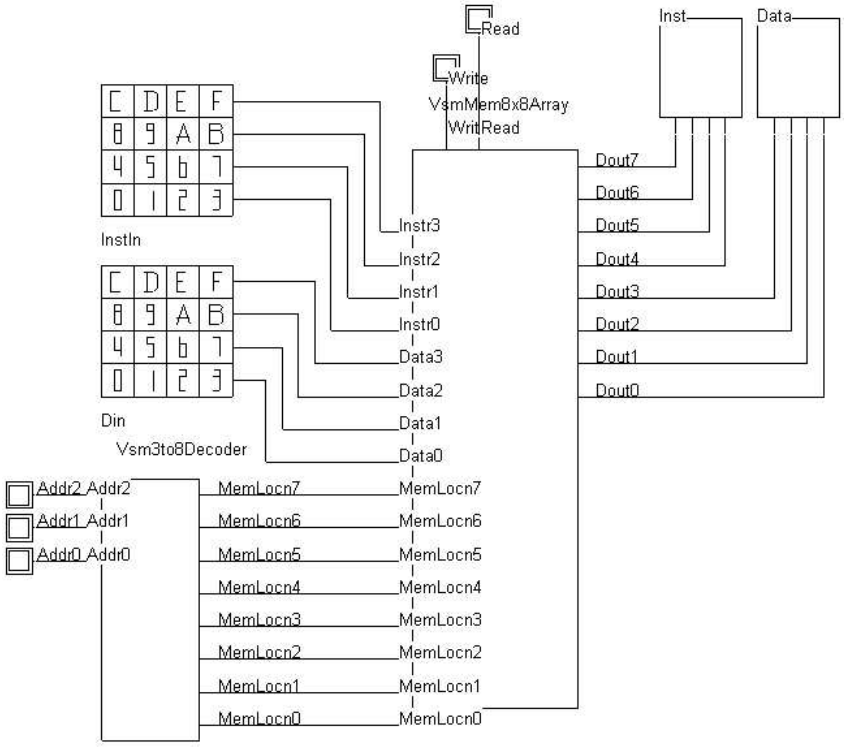


Figure 4-36: The complete 8x8 memory including address decoder (Vsm-Mem8x8.sch)

In order to generate a layout of the microprocessor we replace the memory macro (Vsm-Mem8x8Macro.sch) used in the microprocessor of Figure 4-27 with the real 8x8 memory block presented in Figure 4-36. The new microprocessor containing this real memory block is shown in Fig. 4-37. Note that the 3-bit address information can be supplied to the memory (VsmMem8x8) either from the top keypad titled Addr or from the program counter using a set of three multiplexers controlled by the WriteMem signal. During memory write operation (WriteMem is high) the address comes from the top keypad. Therefore the user is able to specify the memory addresses where to store instructions. When the processor executes instructions it reads the instructions from memory one by one according to the addresses supplied from the program counter (WriteMem is low).

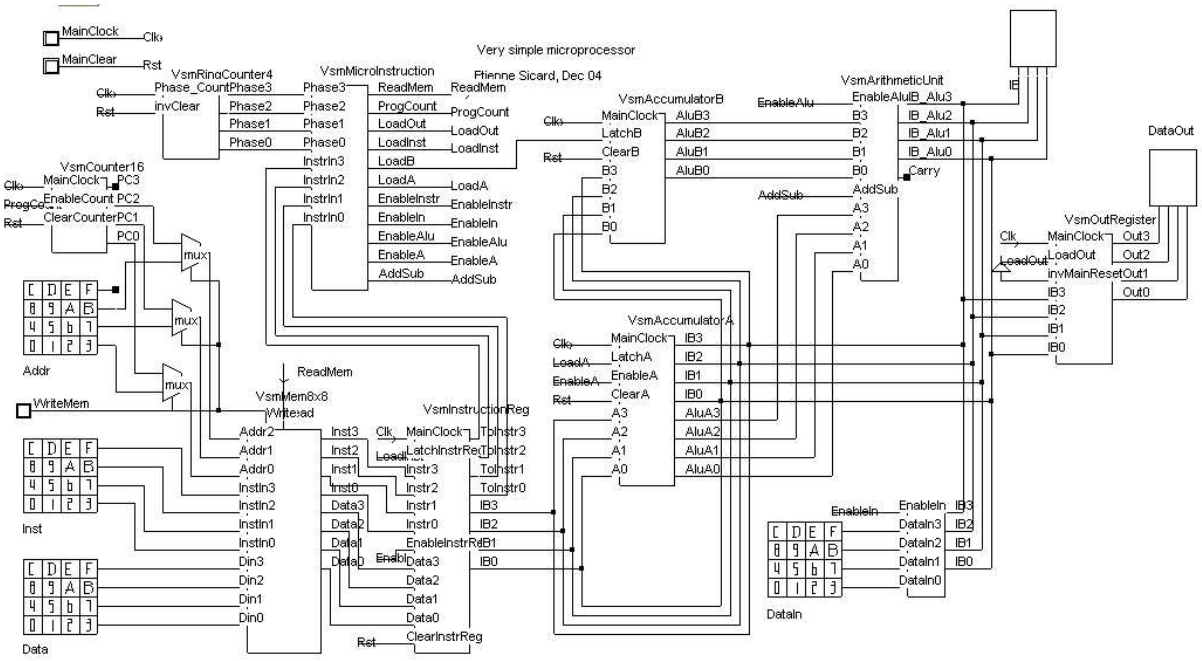


Figure4- 37: Complete microprocessor containing real memory (Vsm-ProcessorRealMem.SCH)

Follow the steps below for entering a program into the memory and then simulating the operation of the processor with the loaded program.

Program entry

Enter the program given in Table 4-6 into the processor memory as follows:

- Start simulation in Dsch3.
- The processor should be disabled by default. In any case it can be disabled by making sure **MainClear** is active (low).
- Assert the Memory Write signal by clicking the **WriteMem** button (high).
- Enter address (0) using the top keypad titled **Addr**. The first memory location is now selected.
- Enter the first instruction using the two bottom keypads titled **Inst** and **Data**.
- Change addresses sequentially and enter the corresponding instructions.
- No clocking is necessary for the program entry operation.
- When all instructions are entered into the memory, click the **WriteMem** button in order to disable the memory write operation.

Program execution

- Enable the processor by deactivating the **MainClear** (high).
- Cycle through various phases of processor operation by repeatedly clicking on the **MainClock** button until all instructions are executed by the processor.
- At each active edge of the clock observe the phase counter shifting from *phase0* to *phase1*, then *phase2*, then *phase3*, and back to *phase0* for the next instruction.
- You can observe the intermediate results in the top display (attached to the Arithmetic Unit) as each instruction is read and executed by the processor.
- When the **OUT** instruction is executed the final result appears on the output display (attached to the Output Register).

Figure 4-38 shows the simulation results from execution of the program given in Table 4-6.

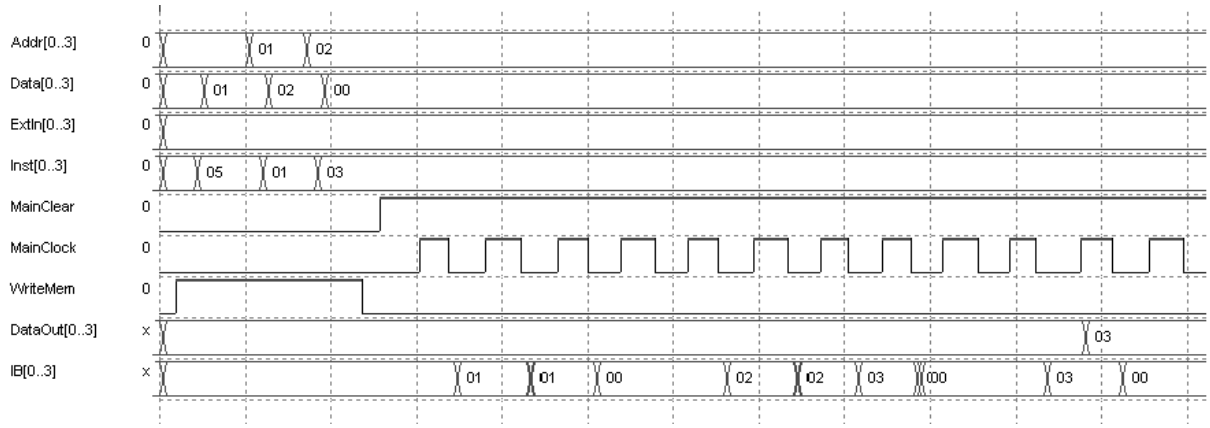


Figure 4-38: Simulation results for addition of 1 and 2 by the processor using the program given in Table 4-6

More details about the implementation of the VSM microprocessor may be found on the web-site of Microwind [3], and concern the interfacing of the microprocessor to the external world.

6. Conclusion

In this chapter, the design of a very simple 4-bit microprocessor has been presented. The basic processor implements 5 instructions. This gives the foundations for building more complex processors with extended instruction set, more sophisticated exchanges between the main memory and the accumulators, more powerful arithmetic unit, in order to build a more attractive microprocessor.

References

[1] A. P. Malvino, J. A. Brown “Digital computer electronics”, Third Edition, Glenco-Macmillan, ISBN 0-02-800594-5, 1992, USA

[2] E. Sicard, S. Ben Dhia “Basics of CMOS Cell design”, Tata McGraw Hill, 2005, IBSN 0-07-059933-5

[3] The Microwind web site is www.microwind.org

Exercises

1.1 Modify the microprocessor in order to handle the MOVE operation from the memory to the accumulator A, according to the recommendations of Figure 4-29. The instruction code can be 0110, with the op-code MOVE.

1.2 List the necessary hardware and supplementary control signals in order to perform the STORE operation from accumulator A to a desired memory location.

1.3 Modify the arithmetic unit in order to perform the Shift Right one bit (SHR, coded 1000) and Shift Left one bit (SHL, coded 1001) operations. What new input do you need to add? In order to reduce the number of ALU controls, how can you handle the ADD, SUB, SHR and SHL signals?

1.4 Modify the microinstruction controller to handle the ADD, SUB, SHR and SHL operation.

1.5 Test the new microprocessor with these enhanced functions