1 LABORATION 3 – FINITE STATE MACHINE

In the laboration, we build a finite state machine (FSM) that will be used to fetch and decode instructions. The FSM will then later be combined with an ALU and register file to create a microprocessor FSMD. For the purposes of this course, we will call this instruction FSM. In the next laboration, we will combine the design with instruction and data memories to create a complete system, see Figure 3.1.

The lab only requires simulation in Questasim or similar.

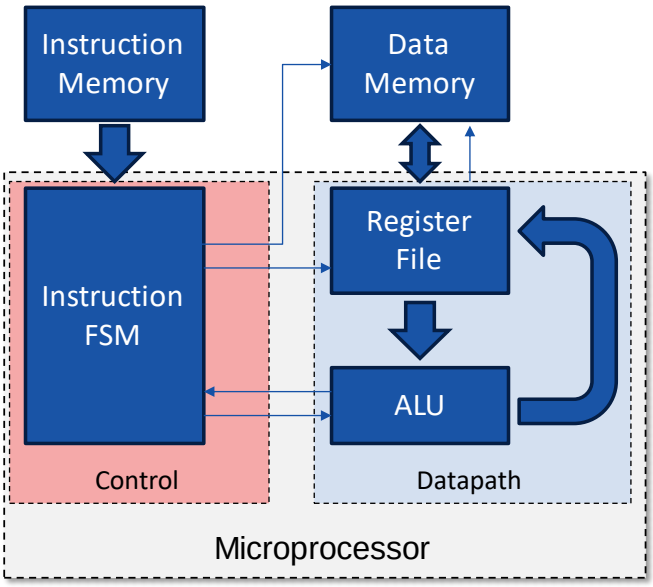

Figure 3.1. Microprocessor coupled with memories

1.1 TASKS

1) Design an FSM based on the following specifications. (Only the FSM)

2) Design a testbench that will test the correct functionality of the FSM.

3) Simulate and verify the functionality of your design.

1.2 FSM SPECIFICATIONS

1) Your microprocessor will have a 3-cycle instruction FSM, as seen in Figure 3.2. A more detailed description of each stage of the FSM with examples for each instruction can be seen in sections 1.5 and 1.6.

C1) Fetch

C2) Decode

C3) Execute and update PC

C1: Fetch - Send the Address to the instruction memory
C2: Decode – receive the instruction and decode it. Register the control signals.
C4: Execute – The Unit operate to complete the necessary calculations and processes

Figure 3.2. Example of the instruction pipeline

2) The FSM should also include a special register to store the Program Counter (PC). The program counter is used to fetch the proper instruction from the instruction memory.

3) The FSM will send control signals to the Register File (RF), arithmetic and logic unit (ALU), data and instruction memory.

3) Your design should be parametric. **Read the following instructions carefully**.

4) The FSM should stop iterating between the stages when the PC overflows. Make sure that you handle any overflow in the PC.

- When an overflow is about to happen in your PC, the FSM should stall, and no other instruction should be loaded.
- The FSM should remain in that state until it is reset, and a warning for overflow signal should be asserted.

1.2.1 INSTRUCTIONS

The FSM should be able to decode the following instructions and send the appropriate control signals out to the units.

Instr.	Inst. Code	OPA	OPB	Function	PC	Type
ADD	0000	RA	RB	RA = RA+RB	PC+1	ALU OP
SUB	0001	RA	RB	RA = RA-RB	PC+1	ALU OP
AND	0010	RA	RB	RA = RA AND RB	PC+1	ALU OP
OR	0011	RA	RB	RA = RA OR RB	PC+1	ALU OP
XOR	0100	RA	RB	RA = RA XOR RB	PC+1	ALU OP
NOT	0101	RA	RB	RA = NOT RB	PC+1	ALU OP
MOV	0110	RA	RB	RA = RB	PC+1	ALU OP
NOP	0111	Null	Null	Null	PC+1	
Load	1000	RA	RB	RA = mem(RB)	PC+1	LD/ST
Store	1001	RA	RB	Mem(RB) = RA	PC+1	LD/ST
Load Imed.	1010	RA	DATA	RA = Sign Ext. DATA	PC+1	LI
BRN_Z	1011	Offset		Branch if zero	PC+offset or PC+1	BRN
BRN_N	1100	Offset		Branch if neg.	PC+offset or PC+1	BRN
BRN_O	1101	Offset		Branch if ovf.	PC+offset or PC+1	BRN
BRN	1110	Offset		Branch always	PC+offset	BRN

The instructions are divided into 3 fields:

1. Instruction code
2. Operant A

3. Operand B

For the design of the instruction FSM, you should use the following parameters.

1. The instruction code, that is 4-bits.
2. Operand A (OPA) and operand B (OPB), the size of each operand field should be parametric with parameter M and have the same size as the address of the register file.
3. Consider that each register word is N-bits.
4. The PC counter should also be parametric with parameter P, and the instruction memory address should also be P-bit in width.

In the table, we specify the function of the instruction as in the following example:

ADD: RA = RA + RB → On the right-hand side, RA and RB are register addresses that are used to read data from the RF. On the left-hand side, RA is the register address where the result of the operations (in this example, '+') will be written back to the RF.

In the case of a branch instruction, the two fields are combined to generate the offset that will be used to update the program counter (PC). When the branch instruction is executed, you should check the appropriate flag (overflow – BRN_O, negative – BRN_N, zero – BRN_Z) and update the PC with PC±offset if the respective flag is '1', otherwise update the PC to PC+1. The offset should be treated as a signed magnitude representation. In the signed magnitude, the first bit is treated as the sign bit (1 means -, 0 means +) while the rest are treated as unsigned. That means that when the offset is required to update the PC, you should use the following logic:

```
{overflow, PC_next} = (offset[MSB]) ? PC – offset[MSB-1:0]: PC + offset[MSB-1:0];
```

!!! Note that together with the next value of the PC, we also check for potential overflow!!!

1.2.2 INPUTS/OUTPUTS

Your FSM design should have the following inputs and outputs:

1.2.2.1 GENERAL SIGNALS

1) 1-bit clock signal.

2) 1-bit rst_n signal.

1.2.2.2 SIGNALS TO AND FROM THE REGISTER FILE (RF)

The following signals will control the RF.

1.2.2.2.1 FOR EACH READ PORT

1) Read address (parametric M-bit) - out

2) Destination select signal (1 bit) - out

1.2.2.2.2 FOR THE WRITE PORT

1) Write the address (parametric M-bit) - out

2) Write enable (1 bit) - out

3) Source select signal (2 bits) - out

4) Immediate value (parametric N-bit) - out

Note: When the immediate output is not required, set it to zero. The immediate value will be used only in the Load immediate instruction, and it will be assigned the DATA value (operant B).

1.2.2.3 SIGNALS TO AND FROM THE ALU

1) OP signal (3-bits) - out

2) Flag synch reset (1-bit) - out

3) ONZ Flags (3-bits) – in

1.2.2.4 SIGNALS TO AND FROM THE INSTRUCTION MEMORY

1) Instruction input (4+2*M bit) -in

2) Enable read instruction (1-bit) - out

3) Instruction Read Address (P-bits) – out

1.2.2.5 SIGNALS TO AND FROM THE DATA MEMORY

1) Read Enable (1-bits) - out

2) Write Enable (1-bit) - out

1.2.2.6 WARNING SIGNALS

1) 1-bit warning that the FSM has been stalled due to PC overflow.

1.2.3 PC UPDATE

The program counter should be updated within your FSM. Create a separate process that will update and register the program counter.

When the PC is about to overflow, the FSM should return and stay in the IDLE state forever.

1.2.4 FSM RESET

The FSM should reset using a negative reset signal. When reset, the FSM should be in an IDLE state, and the PC should be equal to zero.

1.3 TESTBECH SPECIFICATION

Your testbench should test the functionality of the FSM.

1. Your testbench should test all possible cases for instructions codes.
2. Generate your instructions using a constraint random class
    a. Make sure that you only generate valid instruction codes
    b. Ensure you do not generate a branch instruction that its jump will exceed the current PC. (Hint: use inline constraint; you can read the PC from the FSM by using reading DUT.PC in your testbench module, where DUT is the name of your FSM instance)
3. Use assertions to verify that the control signals are asserted correctly.
4. Test that your Program Counter (PC) is updated correctly according to the instructions.
5. Make sure that the overflow in the PC is handled.

1.4 DELIVERABLES:

1) All your files describing the design of the instruction FSM

2) All your files that implement your testbench

1.5 ASM CHART OF THE INSTRUCTION FSM

Here, we present an ASM chart that describes the behaviour of the instruction FSM, see Figure 3.3.

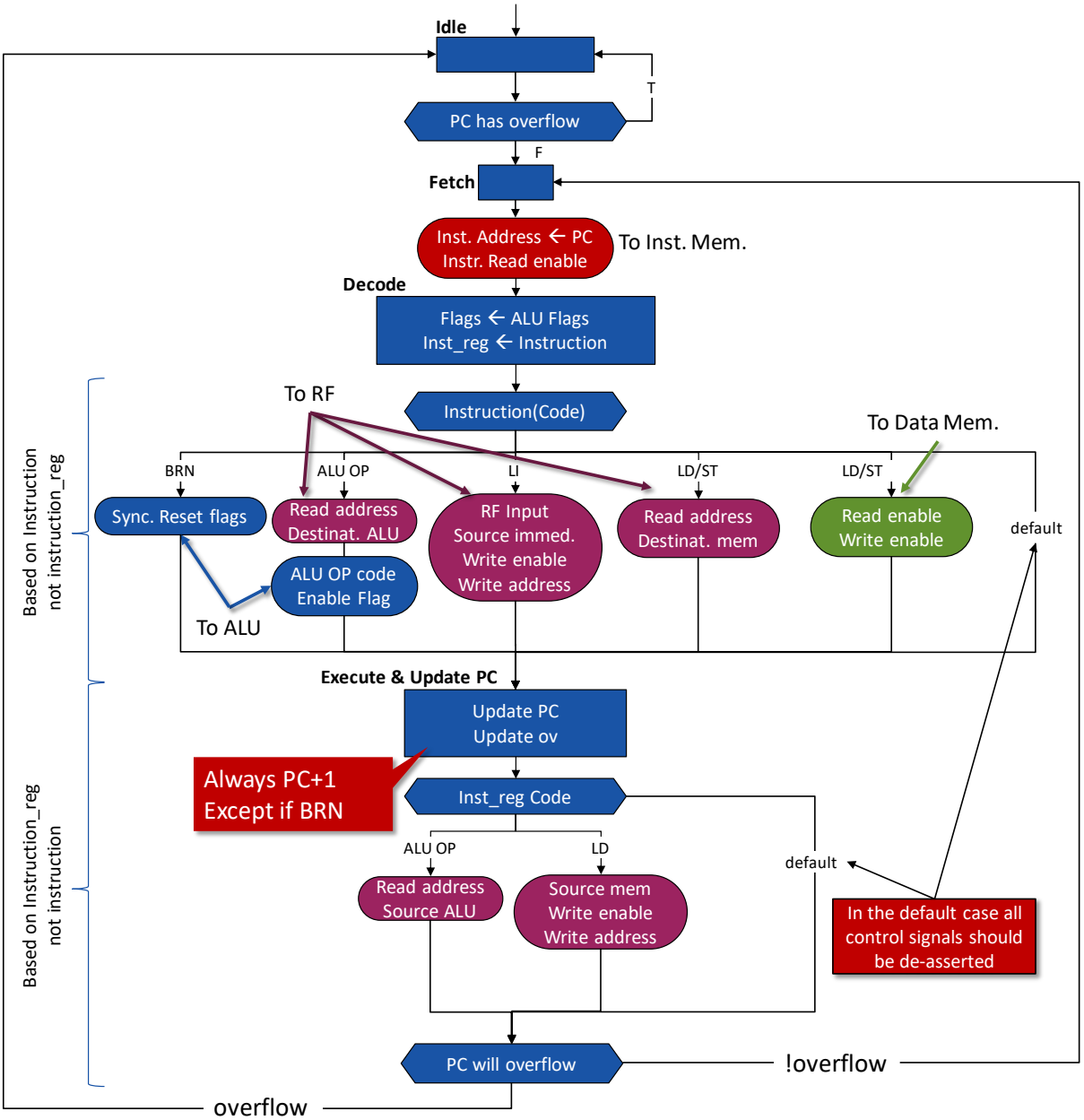

Figure 3.3 Instruction ASM chart

1.6 EXAMPLES OF THE INSTRUCTIONS STEPS

Here, we list some examples of the instruction and how the instruction pipeline and your FSM should operate. Each instruction takes 3 cycles to complete, namely C1, C2, and C3, as seen in Figure 3.1.

1.6.1 LOAD & STORE

C1: Send the address to the instruction memory based on the PC

C2: Decode the received instruction and send out the control signals. The register should send out the memory address. The FSM should send out the read/write and select signals. If it is a Store instruction, the RF should also send out the data.

C3: The op is completed. If it is a load instruction, the RF should get the write address and enable from the instruction FSM where the incoming data should be stored. Update the PC

1.6.2 LOAD IMMEDIATE

C1: Send the address to the instruction memory based on the PC

C2: Decode the received instruction and send out the control signals. The instruction FSM should send to the register the data value (sign extended) and the proper address where the value should be stored along with the appropriate select signals.

C3: The op is completed. Update the PC

1.6.3 BRANCH

C1: Send the address to the instruction memory based on the PC

C2: Decode the received instruction and send out the control signals. Read the appropriate flags from the ALU flag registers and decide how the PC should be updated. Reset the ALU flag register (synchronous reset).

C3: The op is completed. Update the PC according to the value of the ALU flag register.

1.6.4 ALU INSTRUCTION

C1: Send the address to the instruction memory based on the PC

C2: Decode the received instruction and send out the control signals. The instruction FSM should send the appropriate signals to the RF (addresses, rd and select signals) and the ALU. The output of the ALU and the resulting flags should be registered. The enable signal of the Flag register should also be asserted in this step.

C3: The op is completed. The instruction FSM should send out the appropriate control and address signals to the RF so that the data are written back to the RF. Update the PC and de-assert the enable signal of the ALU flag register.