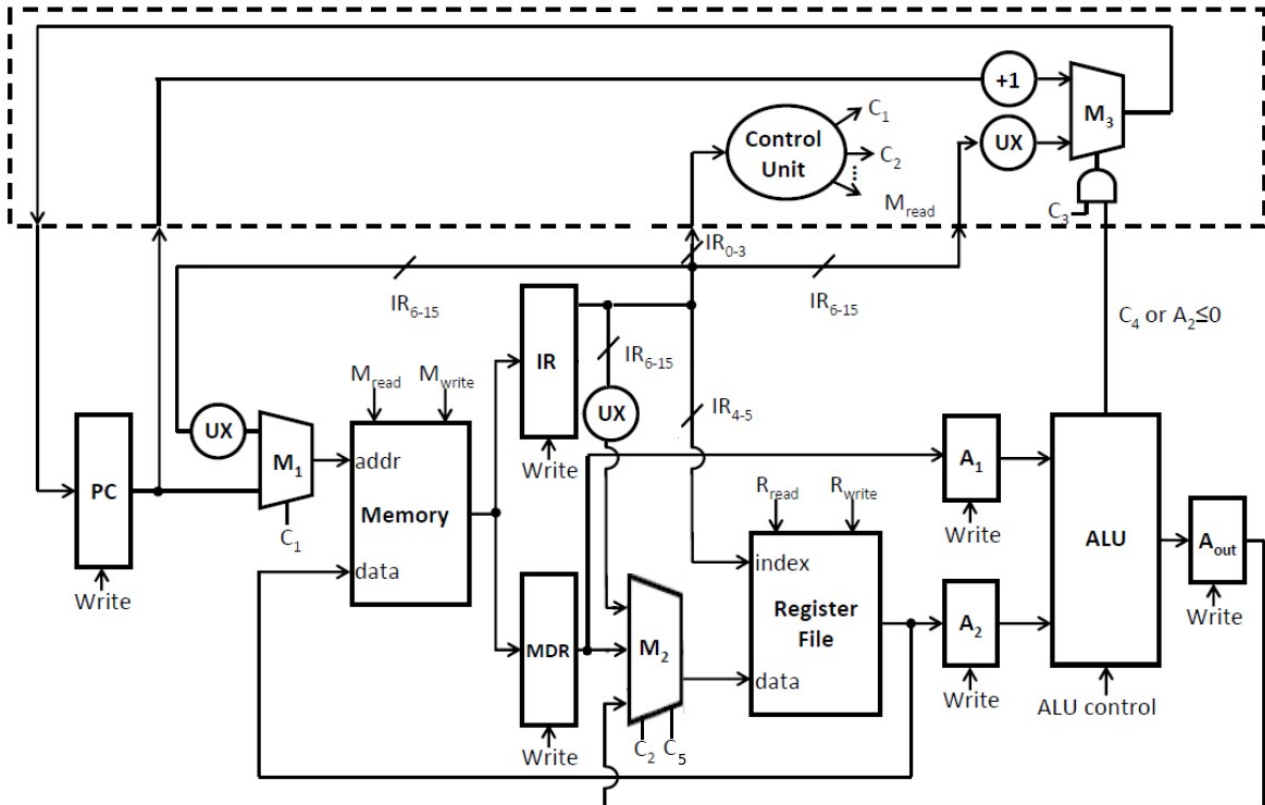


## Architecture Coursework

1i) To access and use a constant stored in bit 6 to 15 of the instruction, assuming that the constant is unsigned, this can be done by adding  $IR_{6-15}$ , after having being increased to 16 bits using a UX unit, to the inputs to  $M_2$  and adding another control signal  $C_5$ , also to  $M_2$ , using the two control signals  $C_2$  and  $C_5$  to select which inputs to use for the Register File data input. These changes are displayed in **figure 1**. Therefore to use the **LOADI** instruction we would first read the instruction into Memory and we would set  $C_2 = 0$  and  $C_5 = 1$  and  $R_{write}$ , to read the 16 constant into the Register File.



**figure 1**

The outputs from  $M_2$  would be based on the  $C_2$  and  $C_5$  would be the following:

$C_2$	$C_5$	$M_2$ Output
0	0	MDR output
0	1	IR <sub>6-15</sub>
1	0	A <sub>out</sub> output
1	1	<i>unused</i>

ii) The registers and control signals that would be used in the `LOADI` instruction are:

Step 1:

*PCwrite*: set PCwrite so that the instruction can be written into the PC

$C_I$ : set to 1 so that the address stored in the PC is used as the Memory address input

$C_3$ : set to 0 so that the PC value is incremented

$M_{read}$ : set  $M_{read}$  so that an instruction can be read from memory a written into the IR

$IR_{write}$ : set  $IR_{write}$ , so that the instruction can be written into the IR

*IR*: the instruction from memory, that is to be used containing the constant, is loaded into the IR so that it may be used

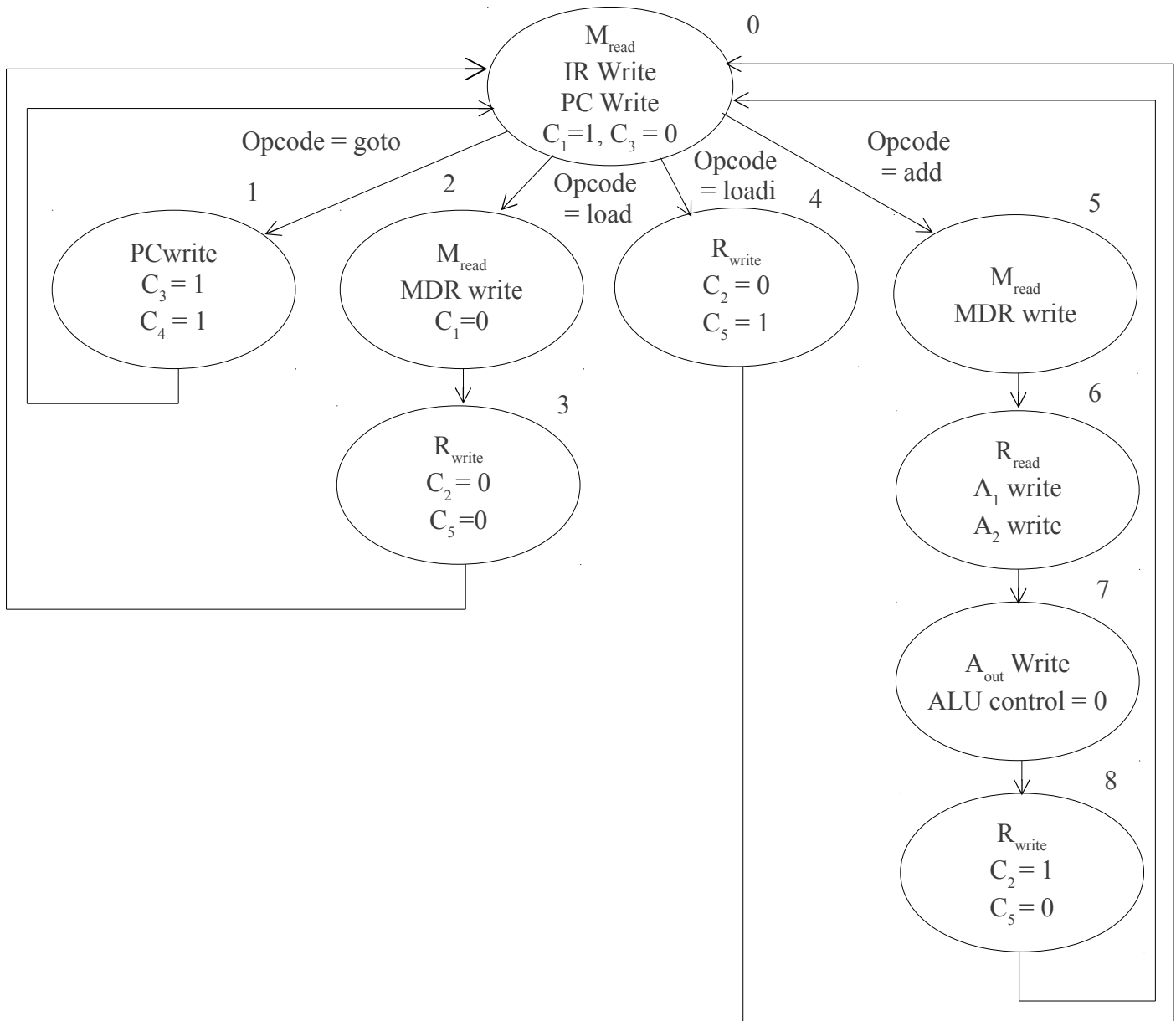
Step 2:

$R_{write}$ : set so that the constant can be written into the Register File

$C_2$ : set to 0 so that the constant from the instruction is selected using the multiplexer and used as the data input in the Register File.

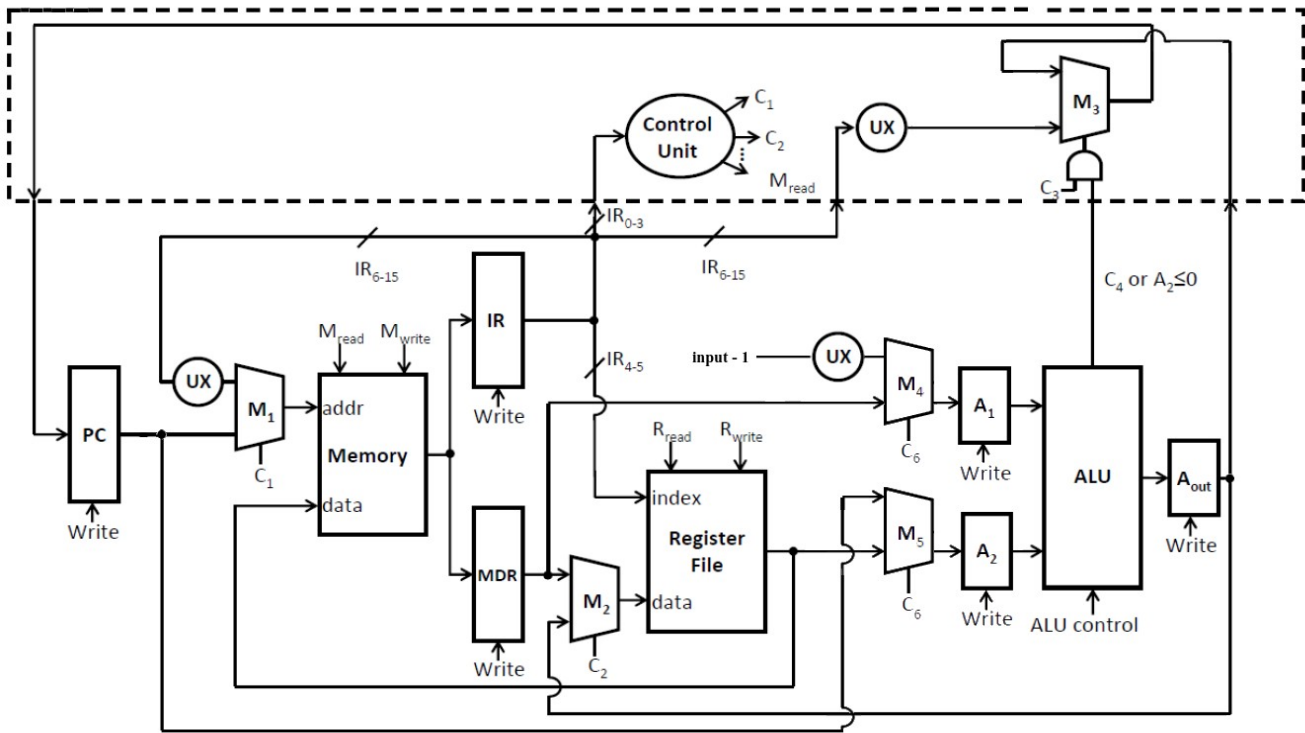
$C_5$ : set to 1, and used in the same way as  $C_2$

iii) the new state diagram, after these alterations, would involve adding a new branch to incorporate the LOADI instruction opcode as well as altering any branches which involve  $M_2$ . This is shown in **figure 2**.



**figure 2**

2i) The “+1” unit can be removed by adding 2 new multiplexers, say  $M_4$  and  $M_5$ ,  $M_4$  would have inputs of an input of 1, which has been put converted to a 16 representation of 1 using a UX unit, and the MDR output.  $M_4$  would then lead into the  $A_1$  register.  $M_5$  would have the inputs of the current value stored in the PC and, also, the output from the register file, its output wire will lead into  $A_2$ . The two multiplexers will share a control signal, say  $C_6$ , which will select whether you want to use the PC value and 1 or the MDR and register file outputs, we can then set the ALU to add the PC and 1, by setting the ALU control to 0, and output this to  $A_{out}$ . The output from  $A_{out}$  will then lead into  $M_3$ , and  $M_2$  as it originally did, and we can use  $C_3$  and PC write to write the new, incremented value, into the PC. The following changes to the original TOY1 architecture are displayed in **figure 3**.



**figure 3**

2ii) The registers and control signals, now, used in the load instruction are:

Step 1:

*PC*: used to store the address of the next instruction

*C<sub>1</sub>*: set to 1 so that the address from the PC is used as the input address in the memory unit to load the instruction from a specified location in memory

*M<sub>read</sub>*: Allow an instruction to be read from memory and written into the IR

*IR<sub>write</sub>*: Allow the instruction to be written to the IR

*IR*: Store the current instruction which has been written from Memory

Step 2:

*M<sub>read</sub>*: set so that the data can be read from the specified address in memory to load the data needed

*MDR<sub>write</sub>*: set so that data may be written to the MDR

*MDR*: store the data that is loaded, and needed, by the instruction

*C<sub>1</sub>*: set to 0 so that the address is read from current instruction that contains the address of the data needed for the instruction

Step 3:

*R<sub>write</sub>*: set so that data, and index, may be written to the Register File

*C<sub>2</sub>*: set to 0 so that the data is loaded from the MDR

Step 4:

*A<sub>1</sub>write*: set so that 1 may be written into the A<sub>1</sub> register

*A<sub>1</sub>*: contains the value of 1

*A<sub>2</sub>write*: set so that current PC address may be written into A<sub>2</sub>

*A<sub>2</sub>*: contains the current address in the PC

*C<sub>6</sub>*: set to 0 so that 1 is written into A<sub>1</sub> and the current PC address into A<sub>2</sub>

Step 5

*A<sub>out</sub>write*: Set so that a value may be written to A<sub>out</sub>

*A<sub>out</sub>*: Holds the incremented value of the PC, i.e. the next address

*ALU control*: set to 0 so that it adds the current PC address and 1

Step 6:

*PCwrite*: set so that the new address may be written to the PC

*C<sub>3</sub>*: set to 0 so that the incremented PC value is written to the PC