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 $IR_{6\text{-}15}$, 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.

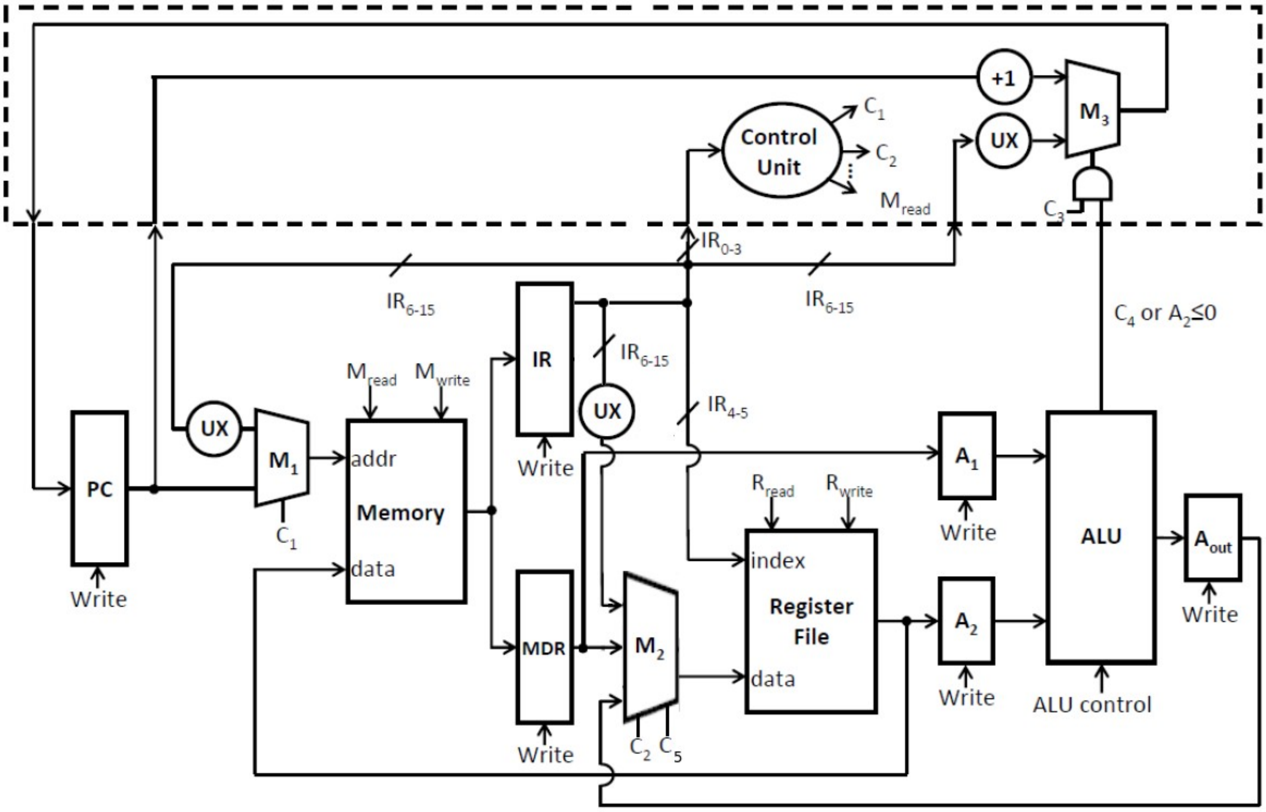

figure 1

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	$IR_{6\text{-}15}$
1	0	A_{out} output
1	*1*	*unused*

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_1: 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_{wite}: 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**.

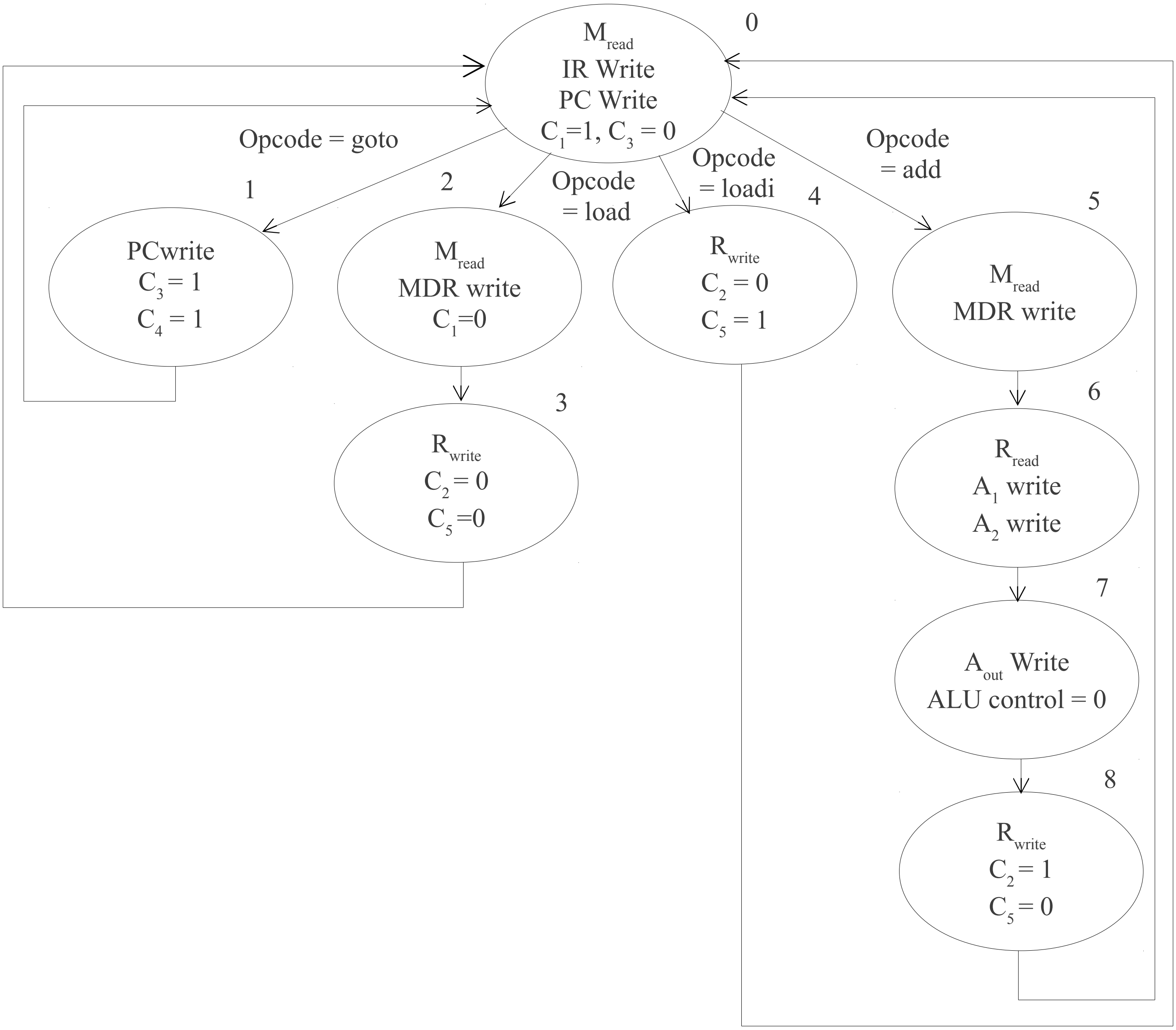

figure 2

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, it's 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 the 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 a the new, incremented value, into the PC. The following changes to the original TOY1 architecture are displayed in **figure 3**.

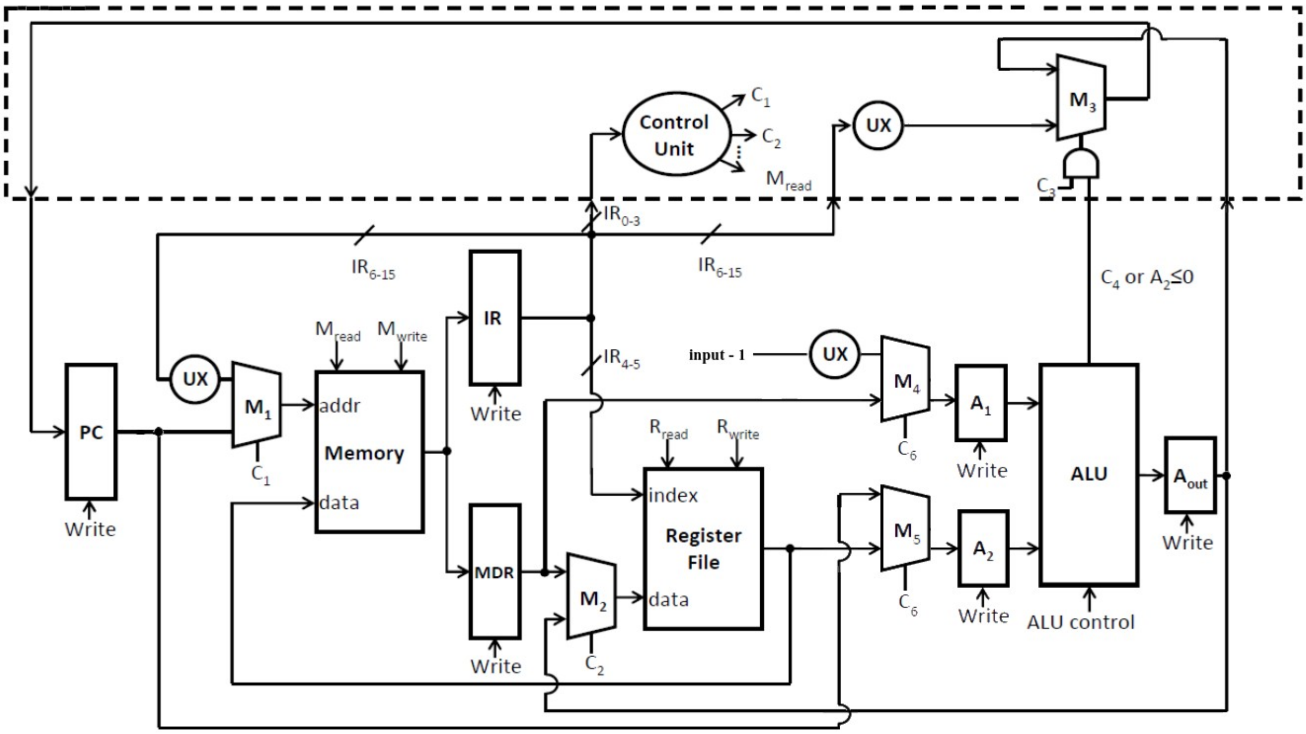

figure 3

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_1: 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

Mread: Allow and instruction to be read from memory and written into the IR

IRwrite: Allow the instruction to be written to the IR

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

Step 2:

Mread: set so that the data can be read from the specified address in memory to load the data needed

MDRwrite: set so that data may be written to the MDR

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

C_1: 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:

Rwrite: set so that data, and index, may be written to the Register File

C_2: set to 0 so that the data is loaded from the MDR

Step 4:

A_1write: set so that 1 may be written into the A_1 register

A_1: contains the value of 1

A_2write: set so that current PC address may be written into A_2

A_2: contains the current address in the PC

C_6: set to 0 so that 1 is written into A_1 and the current PC address into A_2

Step 5

A_{out}write: Set so that a value may be written to A_{out}

A_{out}: 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_3: set to 0 so that the incremented PC value is written to the PC