[CEG3420 Lab 3 Report]

|  |  |  |
| --- | --- | --- |
| GAO, Mingyuan |  | 1155107738 |

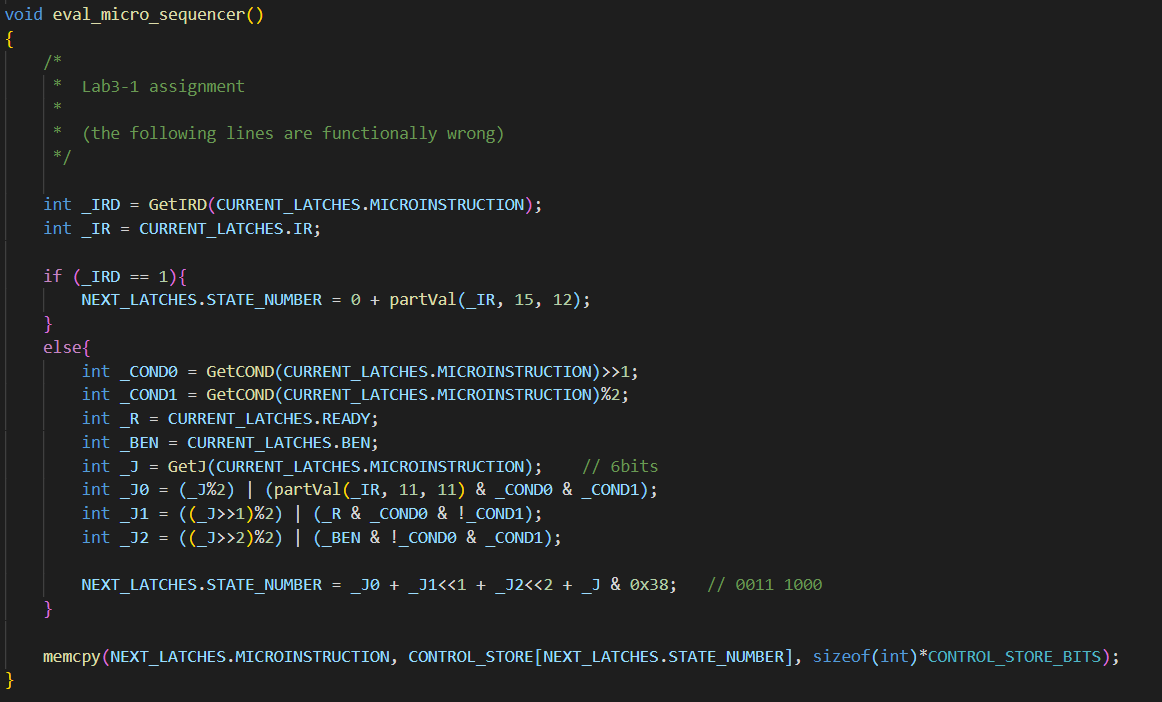
# Task 1

**Describe the outline of how to implement the function “eval\_micro\_sequencer()” here:**

Get IRD using GetIRD() to determine which input should be taken. If it is 1, we directly set the NEXT STATE\_NUMBER as IR[15:12].

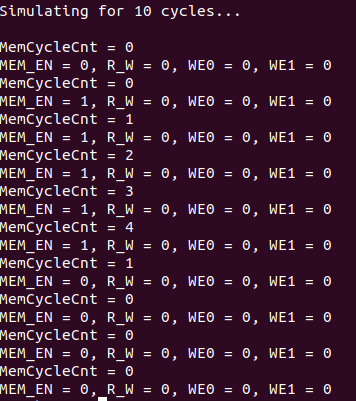
Otherwise, get all values useful including COND, R, BEN, J. According to the architecture shown in lab3-1 notes, get the low 3 bits of J using logic gates. And J&0x38 (0011 1000) to clear the low 3 bits of original J, so that we could substitute those 3 bits to our output of logic gates. Finally set NEXT\_LATCHES.STATE\_NUMBER as new J.

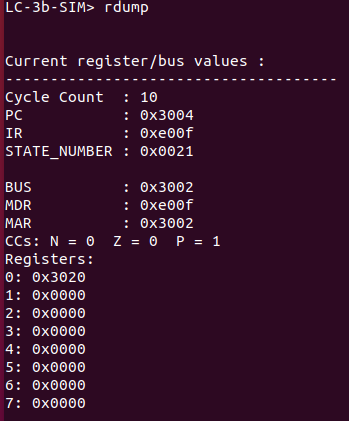
Set NEXT\_LATCHES.MICROINSTRUCTION using the control store of the corresponding state.



**Paste your experimental results of** **“run 10” here by running “toupper.cod”.**

**(You need paste the information of instructions and registers)**





# Task 2

**Describe the outline of how to implement the function eval\_bus\_drivers() here:**

**Input of GateMARMUX:**

**The input from IR is IR[7:0] is through LSHF1 (left shift 1 bit) & ZEXT(zero extended). And the input from adder is provided, so we use blockMARMUX and pass those two possible inputs together with the control bit to get the required value.**

**Input of GatePC**

**Directly fetch the current PC.**

**Input of GateALU**

**One possible input of ALUMUX is from SR1, which can be got from reg file.**

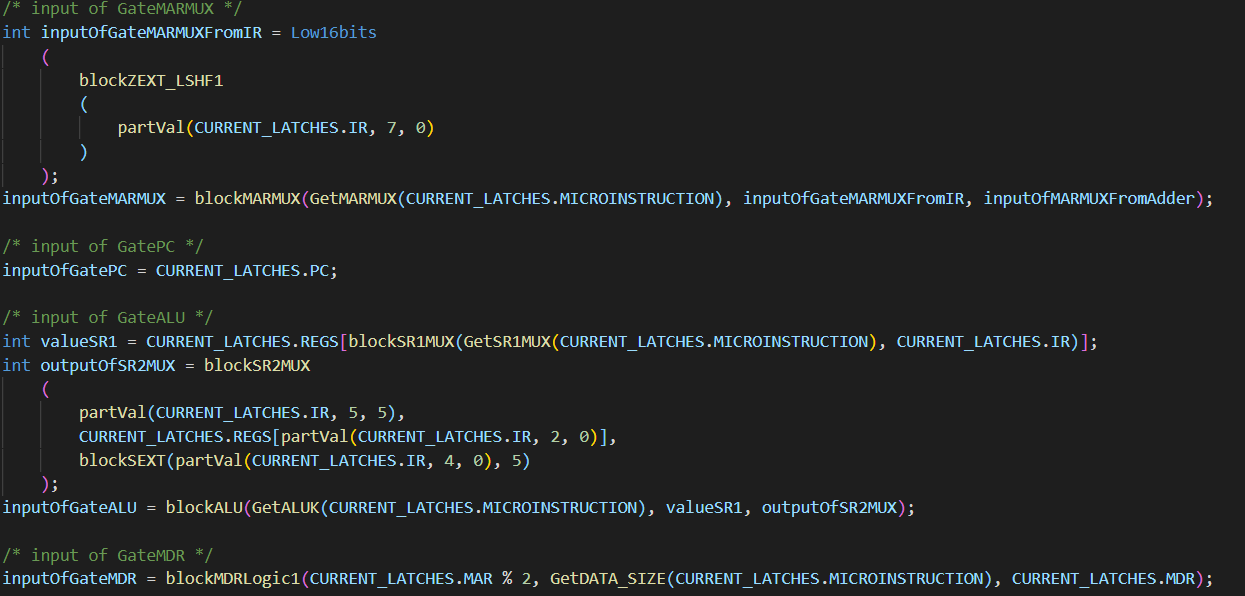
**Another one input of ALUMUX is the output of SR2MUX, which receives two inputs: SEXT(IR[4:0]) and SR2 from reg file. Also, pass a control bit (bit[5]) to SR2MUX.**

**The control bit of ALUMUX can be gotten by calling GetALUK()**

**With those two inputs and a control bit, ALUMUX will output the required value.**

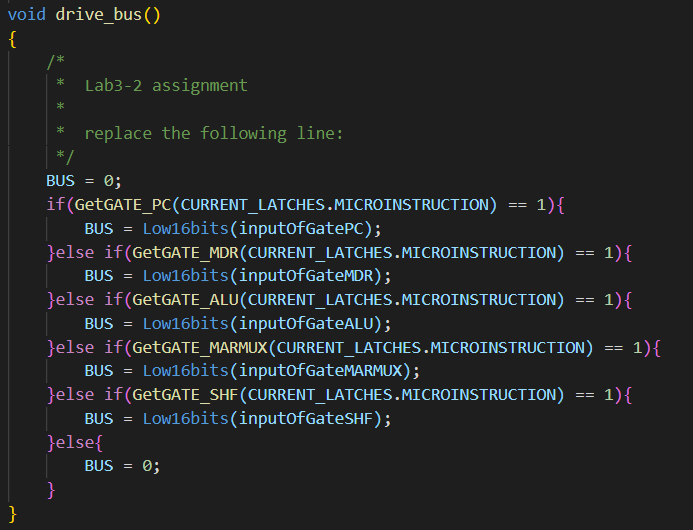
**Input of GateMDR**

**Two control value, DATA\_SIZE determines byte base or word base. If byte base, MAR[0] determines which part of MDR should be passed. We directly pass those two control bits into blockMDRLogic1, together with the input data MDR. The block will do all the things for us.**



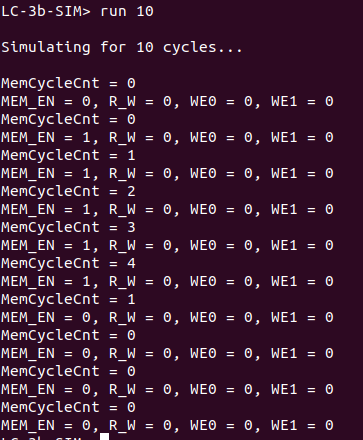
**Describe the outline of how to implement function “drive\_bus()” here:**

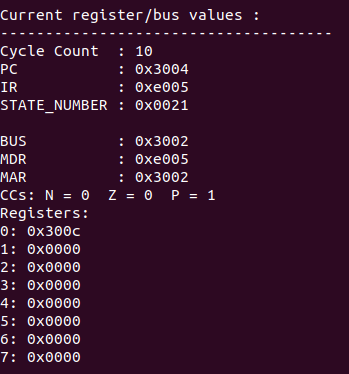
**Check whether there is a control bit sent by any of those five tristate drivers. It there is any, BUS will store the value of input of that gate, which is calculated in eval\_bus\_drivers(). If no, just set BUS to 0.**



**Paste your experimental results of “run 10” here by running “count10.cod”.**

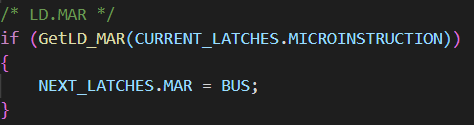
**(You need paste the information of instructions and registers)**





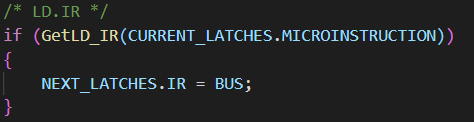
# Task 3

**Describe how you implement the “LD.MAR” in the function “latch\_datapath\_value()”:**



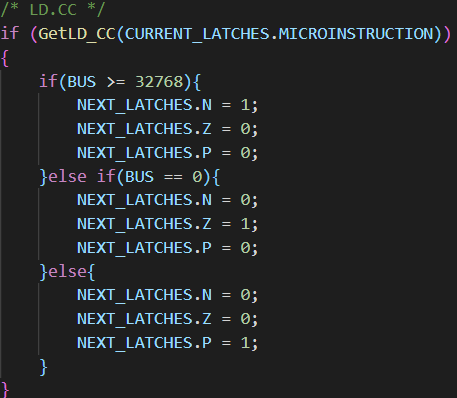
Read from bus directly.

**Describe how you implement the “LD.IR” in the function “latch\_datapath\_value()”:**



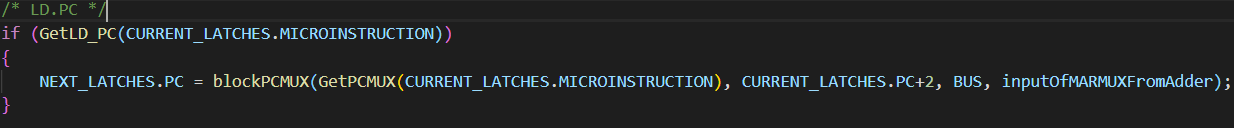
Read from bus directly.

**Describe how you implement the “LD.CC” in the function “latch\_datapath\_value()”:**



**Read the value from bus, set cc according to whether this value is negative, 0 or positive. Use (BUS>=32678) to represent for a negative value.**

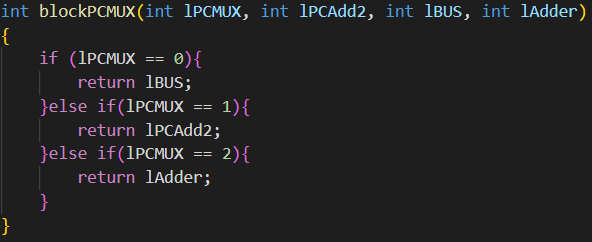
**Describe how you implement the “LD.PC” in the function “latch\_datapath\_value()”:**



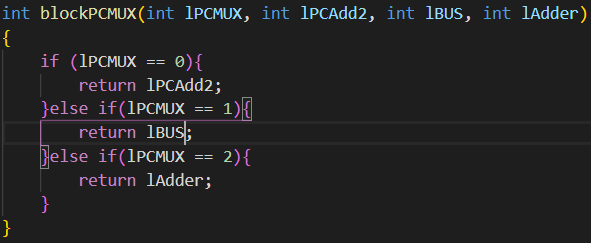
Pass PCMUX, PC+2, BUS and the output of the adder in datapath (equivalent to one of the input of MARMUX, which we have set in eval\_bus\_drivers() ) to blockPCMUX, and set next PC as the output of PCMUX.

**Describe the influence of the different SID in this task and how you resolve it.**

**The end of my SID is 8. The following is my implementation.**



The original one should be



I only need to swap the return value of two conditions, lPCMUX=0 and 1.

**Paste your experimental results of “run 10” here by running “toupper.cod” using corresponding ucode3 file (** **ucode3-even for students with even ending SID, ucode3-odd for students with odd ending SID) .**

**(You need paste the information of instructions and registers)**

