CS2300 PROJECT FINAL SUBMISSION

REG

REG is a 8-bit Register constructed using 8 1-bit D Flip-Flop with a bi-directional I/O line. The 8 1-bit D flip-flops are synchronous and connected to a single clock.

When the REG_IN is high, the tristate buffer is enabled and the input is stored in the D Flip-Flops on the rising edge of the clock.

When the REG_OUT is high, the tristate buffers are enabled and the output from the D flip-flops goes onto the I/O line.

REG_IN and REG_OUT cannot be high at the same time,*********

Gajendra ALU

Gajendra ALU in this assignment supports 8-bit Subtraction and Addition (can also support others with some modifications)

When the SUB is low, the input B of the 8-bit Adder is the XOR of 0 and B and Cin is 0, And the output OUT gives out the sum of the two binary inputs from REG_A and REG_B, the multiplexer ******

When the SUB is high, the input B of the 8-bit Adder is the XOR of 11111111 and B (giving out B') and Cin is 1, And the output OUT gives out

****** of the two binary inputs from REG_A and REG_B, the multiplexer ******

ALU_OUT is used to control the flow of the output to the Display.

CF is high and NCF is low when the C_out of the 8 bit adder is 1 and vice-versa.

ZF (Zero flag) turns high when the OUTPUT of Gajendra ALU is 0, and vice-versa.

NZF is the complement of ZF.

STATUS_REG

The status Register takes input from the Gajendra ALU with the most significant bit IN_3 = CF, IN_2 = ZF , IN_1 = NCF , IN_0= NZF and stores the input in the 4-bit D flip-flop on the rising edge of the clock .

MEMORY ADDRESS REGISTER

MAR, or “Memory Address Register,” is part of a computer’s central processing unit (CPU). CPUs possess memory addresses of data that must be fetched and stored from or to the main memory. The memory address register is essential for data coordination and transfers between the CPU and memory.

In our design the REG_MAR is feeding the bit data into the ROM for 8 bit input it takes the INPUT 8 bit data and separate the 4-bit MSB and 4-bit LSB

LSB = Address, the LSB is feeded into the ROM as discussed above.

INPUT = Takes 8 bit input

CLK = Normal synchronized CLK

PROGRAM_COUNTER

The program counter keeps track of the address of the instruction that is to be fetched from memory and executed next.

LOAD : loads the counter with the value available in the bus (with the 4 least significant bits).

OUT : The PC’s current count value will be output to the bus (4 least significant bits, the 4 most significant bits are ignored).

INC : increments the value in the counter by 1 given a clock pulse.

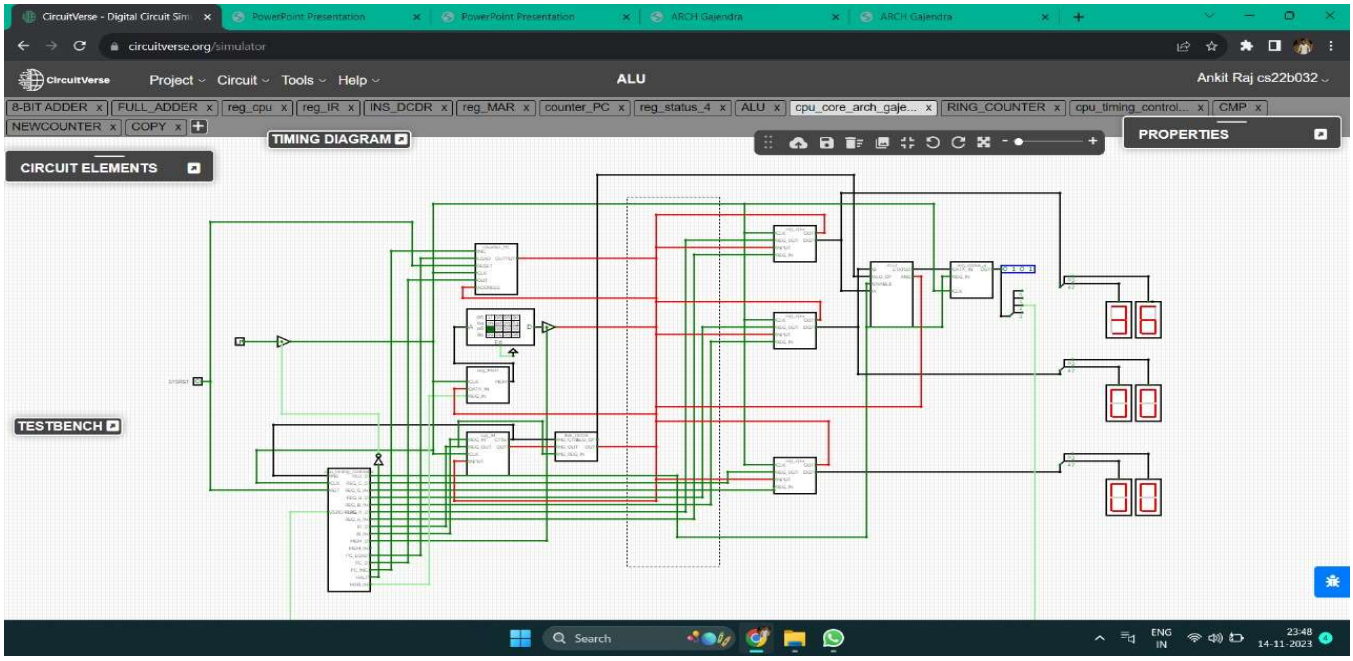
RESET : resets the PC count value to 0.

INSTRUCTION_REGISTER

The construction of the Instruction Register is very similar to that of Normal Register. It uses the 8 D-flip flop for the processing of the data, 8 corresponding Tri-states are used for the same and again merged with the use of 8bit (1 1 1 1 1 1 1 1) splitter to to get the OUTPUT

Further the 4 bit MSB and 4 bit LSB are separated using a 4 bit splitter

(1 1 1 1) the 4 bit MSB is used as the CTRL to fetch the Instruction from the Memory.



SECTION 2:

INST.	Syntx	OP code	Operands	Operation	PC
NOP	NOP	0000 XXXX	NIL	NIL	PC <- PC + 1
LDA	LDA AD	0001 AAAA	0 <= AD <= 15	A <- *AD	PC <- PC + 1
STA	STA AD	0010 AAAA	0 <= AD <= 15	AD <- A	PC <- PC + 1
ADD	ADD AD	0011 AAAA	0 <= AD <= 15	A <- *AD + A	PC <- PC + 1
SUB	SUB	0100 AAAA	0 <= AD <= 15	A <- A -	PC <- PC + 1

	AD			*AD	
LDI	LDI DT	0101 DDDD	0 <= DT <= 15	A <- DT	PC <- PC + 1
OUT	OUT	0110 XXXX	NIL	OUT <- A	PC <- PC + 1
JMP	JMP AD	0111 AAAA	0 <= AD <= 15	NIL	PC <- AD
JNZ	JNZ AD	1000 AAAA	0 <= AD <= 15	NIL	PC <- AD (If F == 1)
SWAP	SWAP	1001 XXXX	NIL	C <- A followed A <- B followed B <- C	PC <- PC + 1
MOVEAB	MOVAB	1010 XXXX	NIL	B <- A	PC <- PC + 1
MOVEAC	MOVAC	1011 XXXX	NIL	C <- A	PC <- PC + 1
MOVEBA	MOVBA	1100 XXXX	NIL	A <- B	PC <- PC + 1
MOVEBC	MOVBC	1101 XXXX	NIL	C <- B	PC <- PC + 1
MOVECB	MOVCB	1110 XXXX	NIL	B <- C	PC <- PC + 1
HALT	HALT	1111 XXXX	NIL	NIL	PC <- 0000

SECTION 4:

NOP:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

LDA:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<IR_out | 1<<MAR_in

T3:1<<MEM_out | 1<<REGA_in

ADD:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<IR_out | 1<<MAR_in

T3:1<<MEM_out | 1<<REGB_in

T4:1<<ALU_out | 1<<REGA_in

SUB:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<MAR_in | 1<<INS_OUT

T3:1<<DM_out | 1<<REGB_in

T4:1<<REGA_IN<<ALU_OUT

LDI:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<INS_OUT | 1<<REGA_IN

OUT:

T0:1<<PC_OUT | 1<<MAR_IN

T1::1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGA_OUT | 1<< OUT_IN

JMP:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<LOAD | 1<<INS_OUT

JNZ:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<LOAD | 1<<INS_OUT (IF FLAG == 1)

SWAP:

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGA_OUT | 1<<REGC_IN

T3:1<<REGA_IN | 1<<REGB_OUT

T4:1<<REGB_IN | 1<<REGC_OUT

MOV AB

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGA_OUT | 1<<REGB_IN

MOV AC

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGA_OUT | 1<<REGC_IN

MOV BA

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGB_OUT | 1<<REGA_IN

MOV BC

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGB_OUT | 1<<REGC_IN

MOV CB

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<REGC_OUT | 1<<REGB_IN

HALT

T0:1<<PC_OUT | 1<<MAR_IN

T1:1<<PC_inc | 1<<MEM_out | 1<<IR_in

T2:1<<RST