MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS

ELECTRICAL AND ELECTRONICS ENGINEERING PROGRAM

EEE 446 Computer Architecture

Lab Module 4/5

MULTI-CYCLE CPU DESIGN w/ SPLIT INSTRUCTION AND DATA MEMORY

Name, Surname	Doğukan Fikri Arat
Student ID	2079648
Name, Surname	Saad Yousaf
Student ID	2246536

Objective

In this lab module, we were responsible for designing our complete CPU by using split Data and Instruction memories, all controlled and synchronized using a hardwired control unit. Our main aim was to obtain functionality of all our ISA instructions while minimizing the CPI. A front panel was also implemented to start and stop the CPU execution and choose between a free running and manual clock for debugging. We made modifications to our datapath to accommodate the few changes we made and fix some errors we encountered while testing the datapath in the previous module. The complete updated datapath is included in Appendix B. We tested all individual instructions to verify their functionality and the results have been reported in this report.

High Level CPU Design

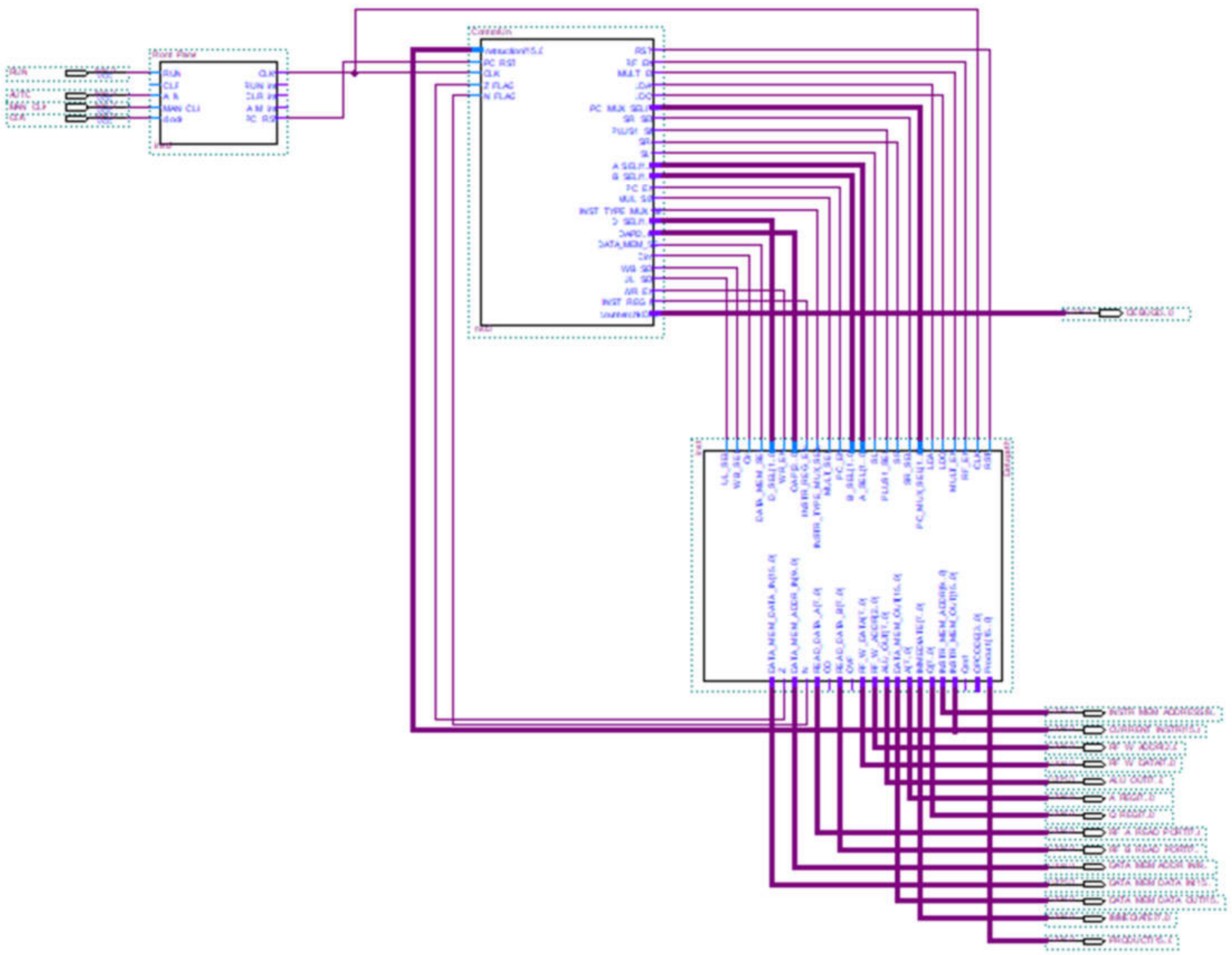


Figure 1: Complete CPU high-level design with front panel (Note: Hi-Res picture in Appendix A)

MULTI-8 ISA complete list of instructions

Instruction	Type	Operands	Description	Operation	Opcode	OAP
ADD	R	Rd, Rs, Rt, Cin	Add two registers and a carry (Cin)	Rd <= Rs + Rt + Cin	0	0
ADDI	I	Rd, Rs, C	Add a register and a constant (C)	Rd <= Rs + C	1	X
SUB	R	Rd, Rs, Rt, Cin	Subtract two registers and a carry (Cin)	Rd <= Rs - Rt - Cin	0	1
AND	R	Rd, Rs, Rt	Logical AND two registers	Rd <= Rs ^ Rt	0	4
ANDI	I	Rd, Rs, C	Logical AND two registers and a constant (C)	Rd <= Rs ^ C	2	X
OR	R	Rd, Rs, Rt	Logical OR two registers	Rd <= Rs ^{v} Rt	0	3
DMADDR	J	C	Set most significant 2-bits of Data Memory Address to least significant 2-bits of constant (C)	DM_address[8:9] <= C [1:0]	3	X
XOR	R	Rd, Rs, Rt	Logical XOR two registers	Rd <= Rs $\oplus$ Rt	0	6
SLT	R	Rd, Rs, Rt	If Rs<Rt, Rd=1, else Rd=0.	Rs-Rt, Rd <= N (zero flag)	0	5
MUL	R	Rd, Rs, Rt	Multiply two registers	[Rd,Rd+1] <= Rs * Rt	0	2
DIV	R	Rd, Rs, Rt	Divide two registers	Rs / Rt, [Rd,Rd+1] <= [Q, A]	0	7
SLL	I	Rd, Rs, C	Logical shift left Rs and save into Rd with constant amount (C)	Rd <= Rs << C	4	X
SRL	I	Rd, Rs, C	Logical shift right Rs and save into Rd with constant amount (C)	Rd <= Rs >> C	5	X
SRA	I	Rd, Rs, C	Arithmetic shift right Rs and save into Rd with constant amount (C)	Rd <= Rs >> C	6	X
LW	I	Rd, Rs, C	Load 16-bit word from data memory	[Rd,Rd+1] <= MEM [Rs + C]	7	X
LWU	I	Rd, Rs, C	Load upper byte from data memory location	[Rd, -] <= MEM [Rs + C]	8	X
LWL	I	Rd, Rs, C	Load lower byte from data memory location	[-, Rd] <= MEM [Rs + C]	9	X
SW	I	Rd, Rs, C	Store 16-bit word to data memory	MEM [Rs + C] <= [Rd,Rd+1]	10	X
SWU	I	Rd, Rs, C	Store 8-bit data to upper byte of data memory location	MEM [Rs + C] <= [Rd, -]	11	X

Table 1: List of Instructions

Instruction	Type	Operands	Description	Operation	Opcode	OAP
SWL	I	Rd, Rs, C	Store 8-bit data to lower byte of data memory location	MEM [Rs + C] <= [-,Rd]	12	X
BREQ	I	Rd, Rs, D	Branch if Rd equals Rs	if Rs-Rd=0, i.e. Z=1, PC <= PC + D	13	X
BRNE	I	Rd, Rs, D	Branch if Rd not equal to Rs	if Rs-Rd!=0, i.e. Z=0, PC <= PC + D	14	X
JUMP	J	D	Jump to address location (C)	PC <= PC+C	15	X
*** D is the branch Destination Address, calculated by counting the number of addresses we want to branch less 1. *C is the jump destination address**						

Table 1: List of Instructions

MULTI-8 Instruction formats:

R-TYPE				
4-bit	3-bit	3-bit	3-bit	3-bit
OPCODE (Op)	DESTINATION (Rd)	SOURCE_1 (Rs)	SOURCE_2 (Rt)	OAP
I-TYPE				
4-bit	3-bit	3-bit	6-bit	
OPCODE (Op)	DESTINATION (Rd)	SOURCE_1 (Rs)	CONSTANT (C)	
J-TYPE				
4-bit	12-bit			
OPCODE (Op)	CONSTANT (C)			

Table 2: Different instruction types supported by MULTI-8 ISA

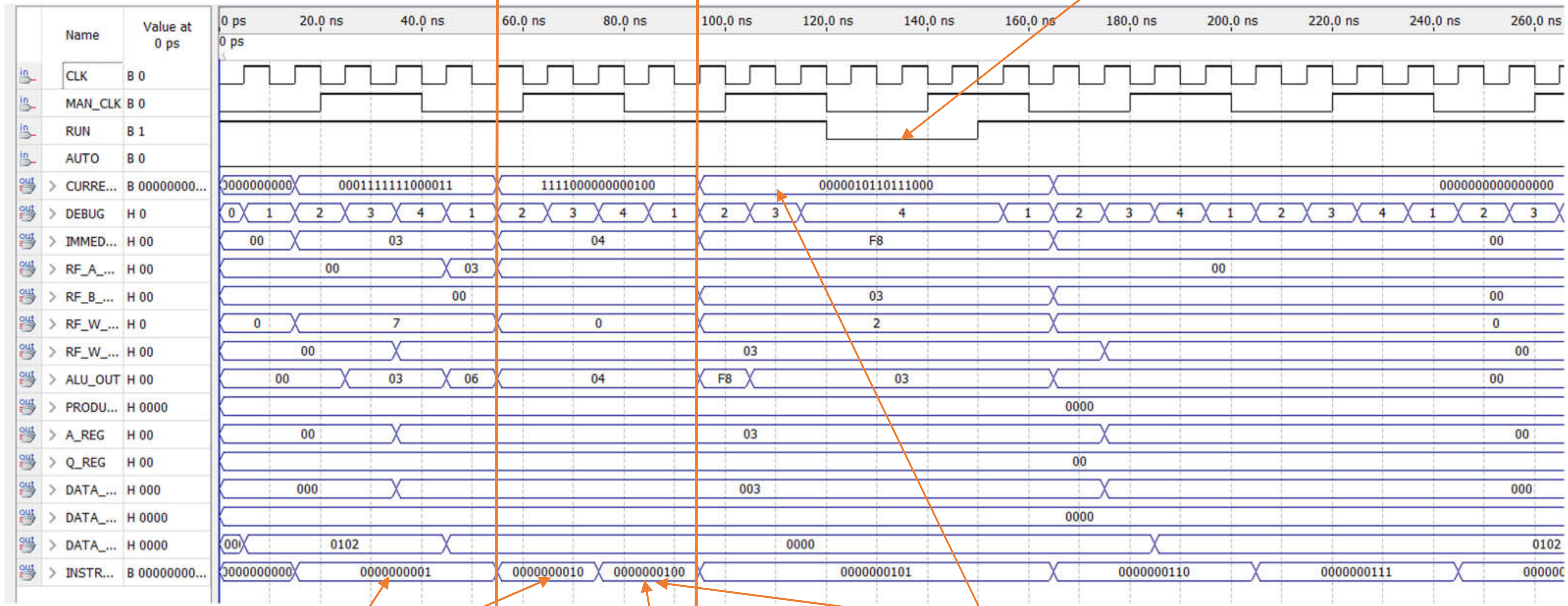
MULTI-8 control unit transitions

- J-type Instructions

Example: Jump instruction: **J 100 (11110000000100)** (i.e. jump to instruction memory address 100)

C

RUN deserted to show functionality of front panel. Upon reassertion it resumes from last state



Name
Value at 0 ps
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
180.0 ns
200.0 ns
220.0 ns
240.0 ns
260.0 ns
CLK B 0
MAN_CLK B 0
RUN B 1
AUTO B 0
CURRE... B 00000000... 0000000000 0001111111000011 1111000000000100 0000010110111000 0000000000000000
DEBUG H 0 0 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3
IMMED... H 00 00 03 04 F8 00
RF_A_... H 00 00 03 00
RF_B_... H 00 00 03 00
RF_W_... H 0 0 7 0 2 0
RF_W_... H 00 00 03 00
ALU_OUT H 00 00 03 06 04 F8 03 00
PRODU... H 0000 0000
A_REG H 00 00 03 00
Q_REG H 00 00
DATA_... H 000 000 003 000
DATA_... H 0000 0000
DATA_... H 0000 00 0102 0000 0102
INSTR... B 00000000... 0000000000 0000000001 0000000010 0000000100 0000000101 0000000110 0000000111 000000

The next instruction at the jump target address 0000000100 fetched

Fetches the jump instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle

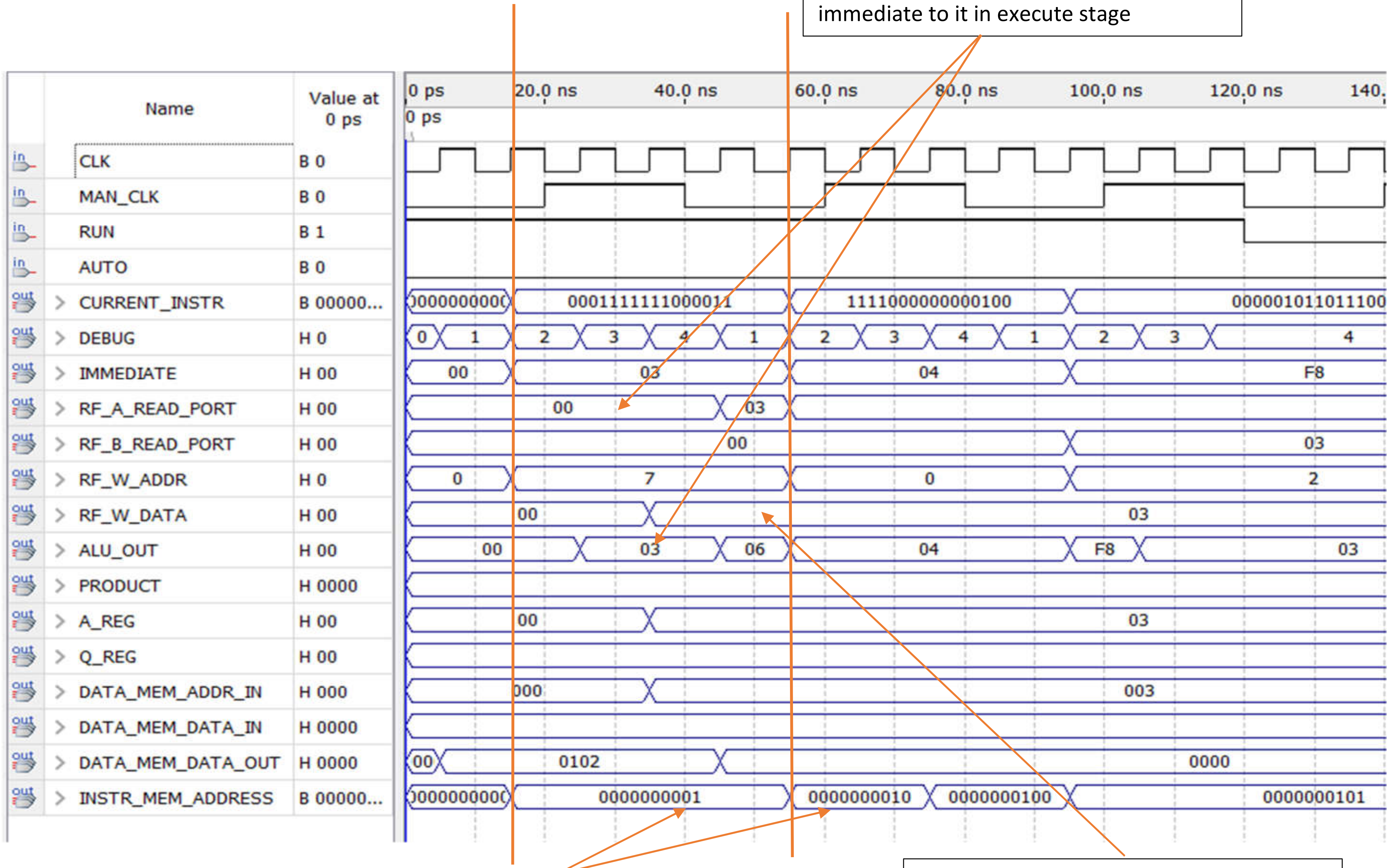
The PC is now fed the jump target address and once execution is done the next instruction at the jump target is fetched

Figure 2: J type instruction state transitions

- I-type Instructions

Example: Add immediate instruction: **ADDI R7,R7,2 (0001111111000010) (R7=0 initially)** (Add 2 to register 7 and store to register 7)

Reads from source register and adds immediate to it in execute stage



Fetches the ADDI instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle

The result is then written back to destination register

Figure 3: I type instruction state transitions (Immediate)

Example: Conditional Branch: **BREQ R7,R7,1 (1101111111000001) (R7=1 initially)** (compare two registers, if equal, load PC with PC plus immediate)

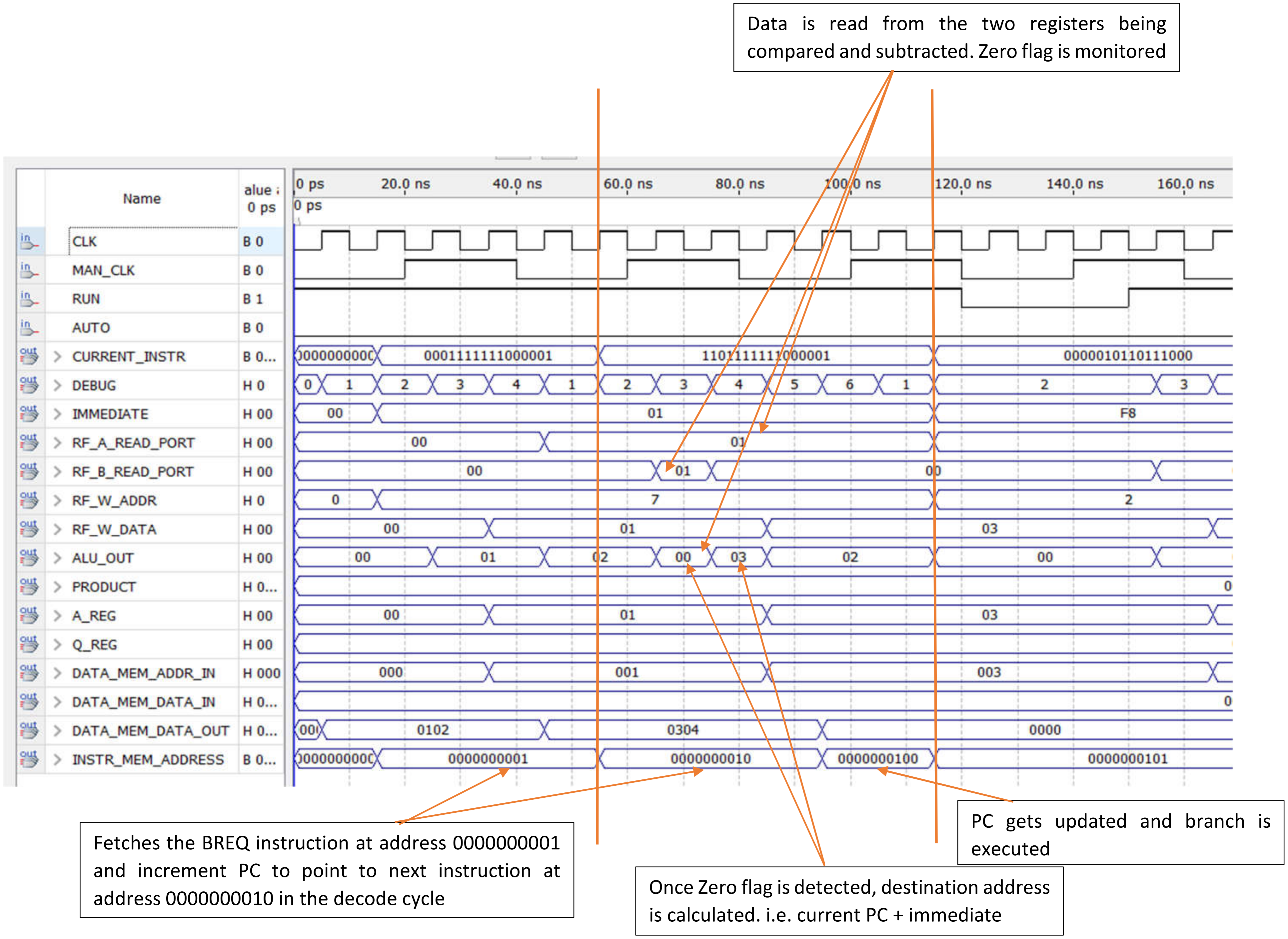

Data is read from the two registers being compared and subtracted. Zero flag is monitored
Name
Value
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
CLK
MAN_CLK
RUN
AUTO
CURRENT_INSTR
DEBUG
IMMEDIATE
RF_A_READ_PORT
RF_B_READ_PORT
RF_W_ADDR
RF_W_DATA
ALU_OUT
PRODUCT
A_REG
Q_REG
DATA_MEM_ADDR_IN
DATA_MEM_DATA_IN
DATA_MEM_DATA_OUT
INSTR_MEM_ADDRESS
0001111111000001
1101111111000001
0000010110111000
0000000001
0000000010
0000000100
0000000101
Fetches the BREQ instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle
Once Zero flag is detected, destination address is calculated. i.e. current PC + immediate
PC gets updated and branch is executed

Figure 4: I type instruction state transitions (Conditional Branch)

- R-type Instructions

Example: Add instruction: **ADD R2,R6,R7 (0000010110111000) (R7=1 and R6=R2=0 initially)** (Add R6 to R7 and store to R2)

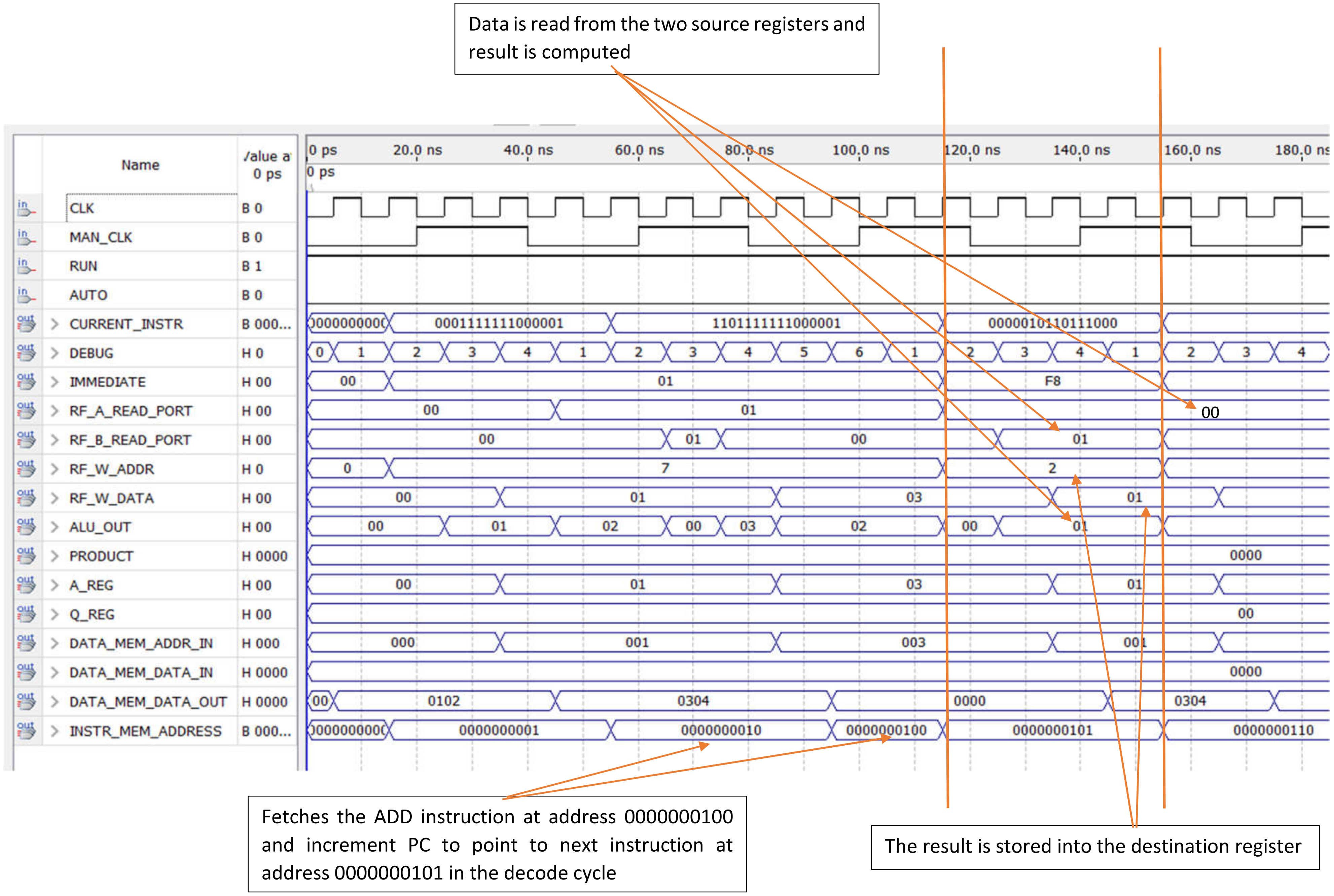


Figure 5: R type instruction state transitions

Sample code sequence simulation

```
ADDI $S0, $S0, 12 # $S0 = 12

ADDI $S1, $S1, 2 # $S1 = 2

SUB $S2, $S0, $S1 # $S2 = 10

SW $S2, $S1, 0 # MEM[2] = 000A

LW $S3, $S1, 0 # $S3,$S4 = MEM[2]

BREQ $S3, $S2, 16 # PC = 16

JMP 3 # PC = 3 (@ PC = 16)
```

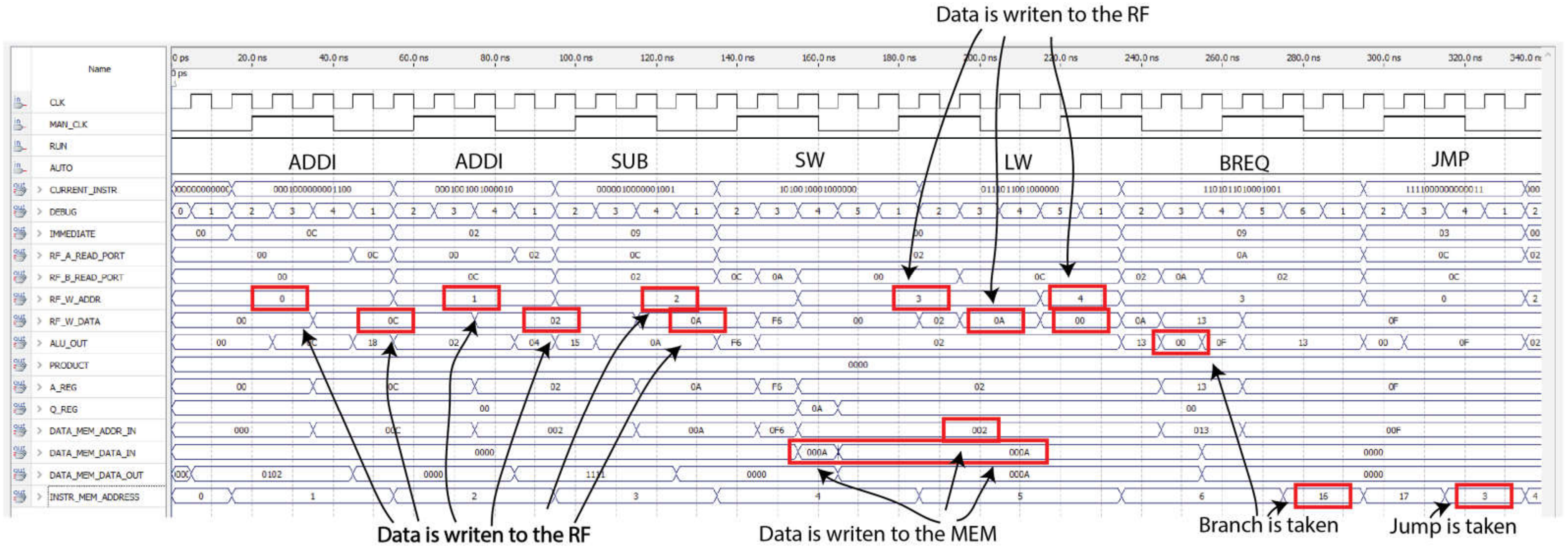


Figure 6: Code sequence simulation

New added blocks

- Front panel

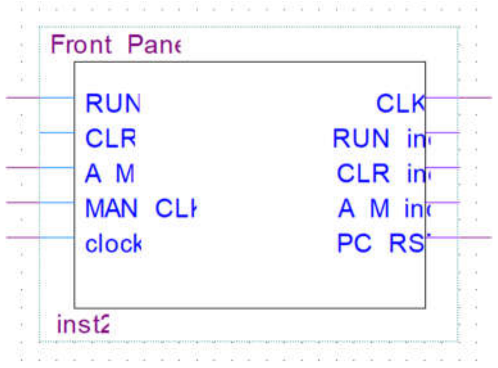


Figure 7: Block diagram of front panel

- Control Unit

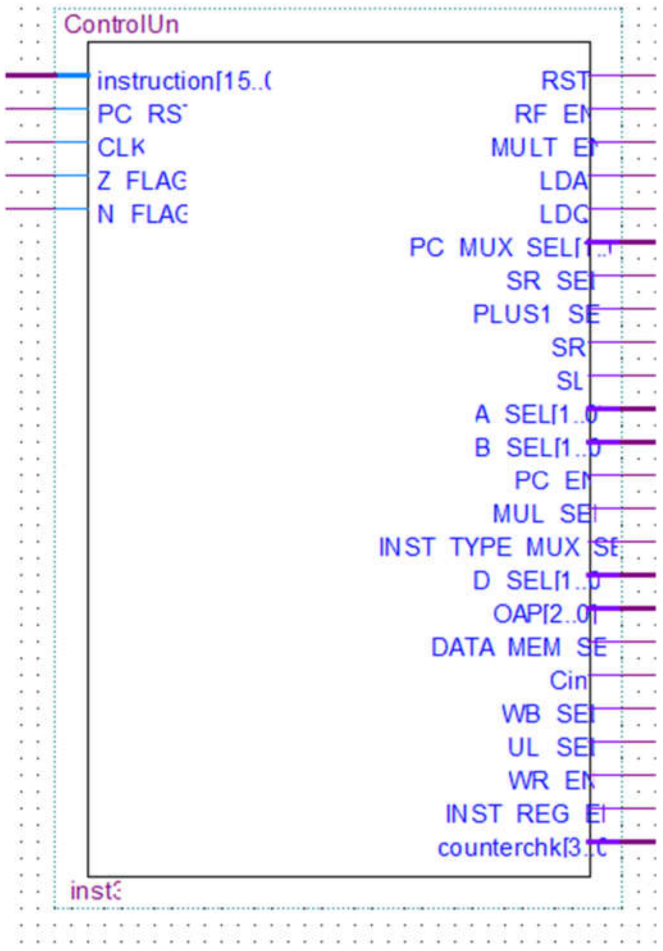


Figure 8: Block diagram of control unit

Verilog code for front panel and control unit

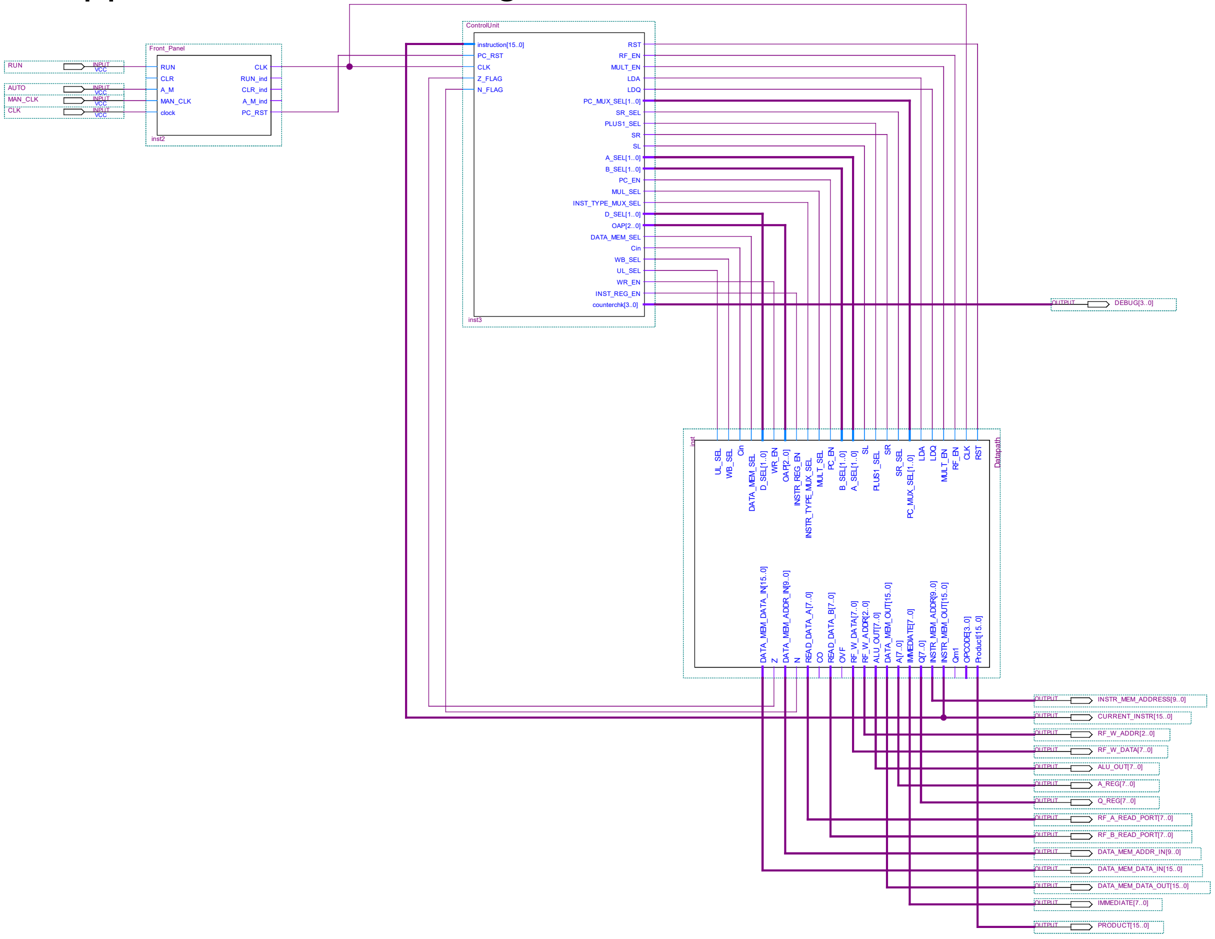
Front panel

```
module Front_Panel(RUN, CLR, A_M, MAN_CLK, CLK, clock, RUN_ind, CLR_ind, A_M_ind, PC_RST);

output reg CLK;

output reg RUN_ind, CLR_ind, A_M_ind, PC_RST;


input RUN, CLR, A_M, MAN_CLK, clock;


always @*

begin

        if(RUN)

        begin

                RUN_ind <= 1'b1;

                if(A_M==0)

                        begin

                        CLK <= clock;

                        A_M_ind <= 1'b1;

                        end

                if(A_M==1)

                        begin

                         CLK <= MAN_CLK;

                        A_M_ind <= 1'b0;

                        end

        end


        if(RUN==0)

        begin

        RUN_ind <= 1'b0;

        end
        if(CLR==1)
        begin
        CLR_ind <= 1'b1;
        PC_RST <= 1'b1;
        end

        if(CLR==0)
        begin
        CLR_ind <= 1'b0;
        PC_RST <= 1'b0;
        end
end
endmodule
```

Control Unit

Note: The code for the control unit is too long to add to the document. It can be provided as a softcopy upon request.

Appendix A : CPU Design



Appendix B: Datapath