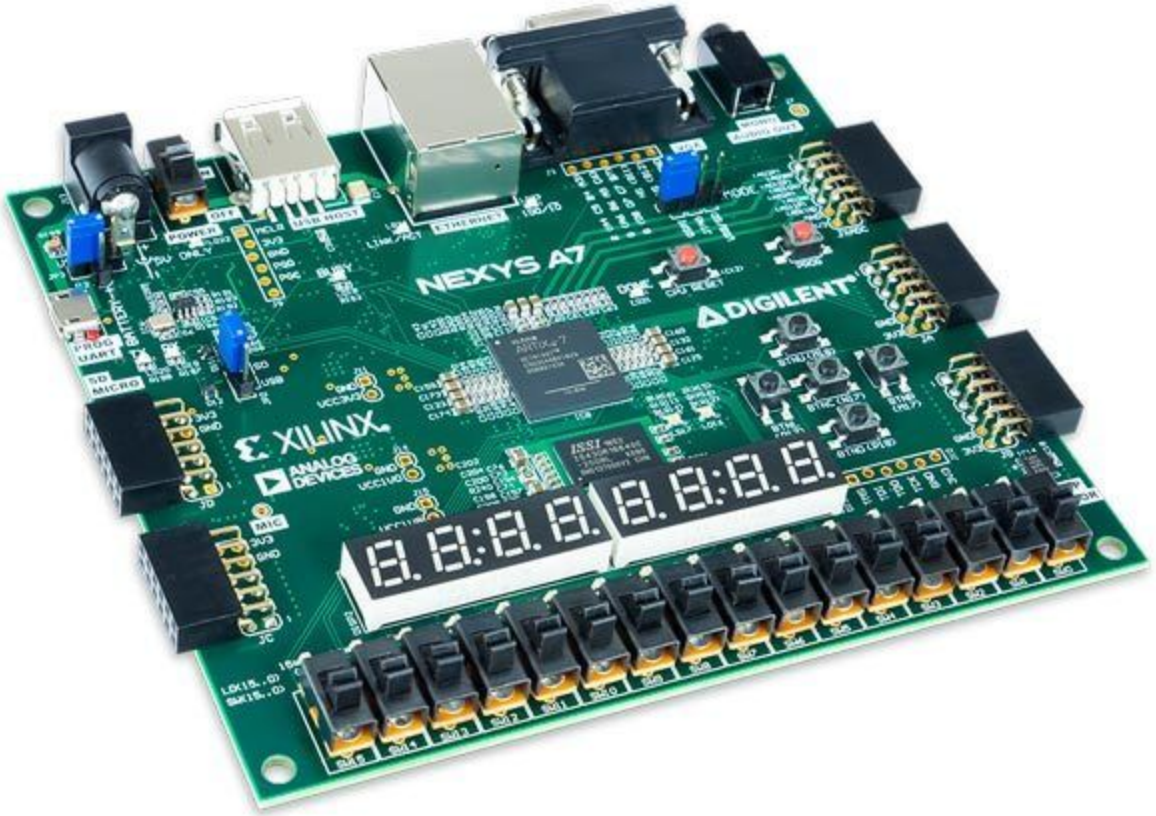
NTP-MICROPROCESSOR



Execution Block

Input and Output:

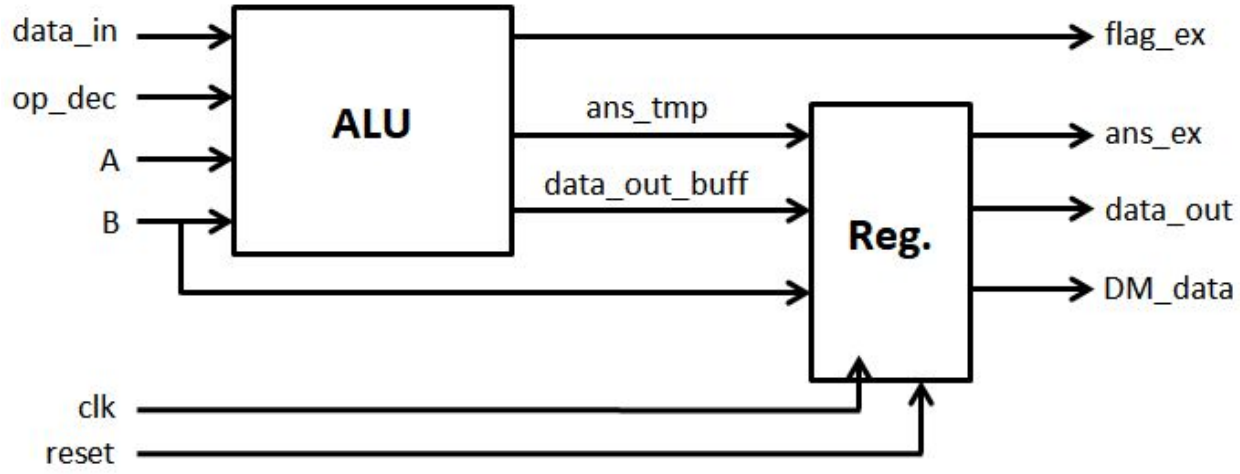
Signals to Execution Block	Input variable Name	No. of Bits	Explanation
Inputs	A	8	First operand for ALU operations
	B	8	Second operand for ALU operations
	data_in	8	Data from external input port
	op_dec	5	Opcode
	clk	1	Clock for execution block
	reset	1	Reset signal to clear register of execution block
Outputs	ans_ex	8	Execution block output
	DM_data	8	DM_data = B (Data for Data Memory block)
	data_out	8	Data to external output port
	flag_ex	4	Status bits :- flag_ex[0] = carry; flag_ex[1] = zero; flag_ex[2] = overflow; flag_ex[3] = parity;

Note -:

1) Above outputs should be generated on a positive edge of clock with respect to 'clk', except 'flag_ex'.

2) 'reset' is negative level triggered and clock synchronous.

Block Diagram:



Opcodes and Operations:

Sr. No.	Instruction	Opcode	Operation in Execution Block	Flags Affected
1	ADD	00000	ans_ex = A+B	Parity, Overflow, Zero, Carry
2	SUB	00001	ans_ex = A-B	Parity, Overflow, Zero, Carry
3	MOV	00010	ans_ex = B	Parity, Zero (Reset other flags)
4	AND	00100	ans_ex = A&B	Parity, Zero (Reset other flags)
5	OR	00101	ans_ex = A|B	Parity, Zero (Reset other flags)
6	XOR	00110	ans_ex = A^B	Parity, Zero (Reset other flags)
7	NOT	00111	ans_ex = ~B	Parity, Zero (Reset other flags)
8	ADI	01000	ans_ex = A+B	Parity, Overflow, Zero, Carry
9	SBI	01001	ans_ex = A-B	Parity, Overflow, Zero, Carry
10	MVI	01010	ans_ex = B	Parity, Zero (Reset other flags)
11	ANI	01100	ans_ex = A&B	Parity, Zero (Reset other flags)
12	ORI	01101	ans_ex = A|B	Parity, Zero (Reset other flags)
13	XRI	01110	ans_ex = A^B	Parity, Zero (Reset other flags)
14	NTI	01111	ans_ex = ~B	Parity, Zero (Reset other flags)
15	RET	10000	Hold previous 'ans_ex'	Reset all flags
16	HLT	10001	Hold previous 'ans_ex'	Reset all flags
17	LD	10100	ans_ex = A	Reset all flags
18	ST	10101	ans_ex = A	Reset all flags
19	IN	10110	ans_ex=data_in	Parity, Zero (Reset other flags)
20	OUT	10111	Hold previous 'ans_ex' data_out = A	Reset all flags
21	JMP	11000	Hold previous 'ans_ex'	Reset all flags
22	LS	11001	ans_ex = A<<B	Parity, Zero (Reset other flags)
23	RS	11010	ans_ex = A>>B	Parity, Zero (Reset other flags)
24	RSA	11011	ans_ex = A>>>B	Parity, Zero (Reset other flags)
25	JC	11100	Hold previous 'ans_ex'	Reset all flags
26	JNC	11101	Hold previous 'ans_ex'	Reset all flags
27	JZ	11110	Hold previous 'ans_ex'	Reset all flags
28	JNZ	11111	Hold previous 'ans_ex'	Reset all flags

Note -:

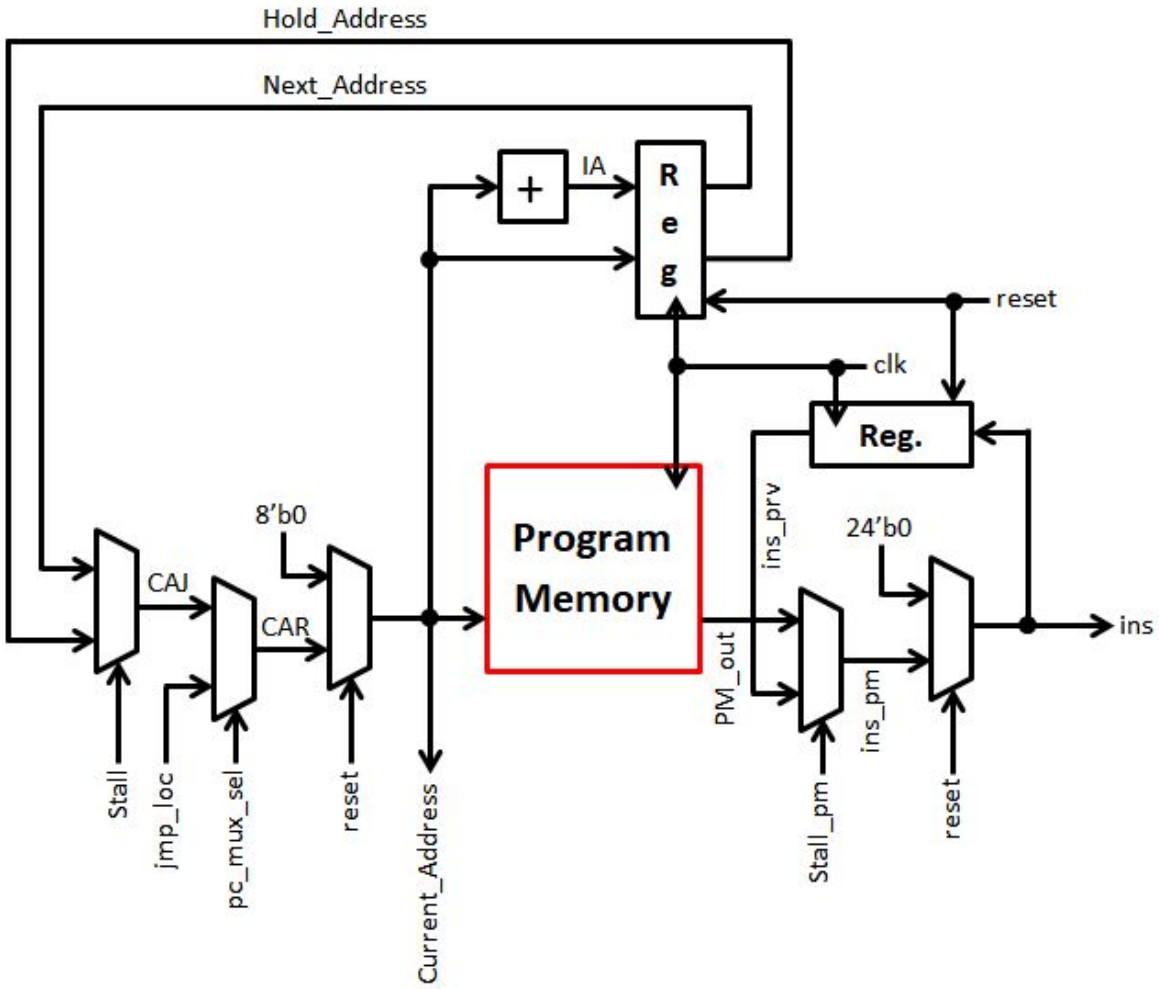
1) Arithmetic operations are on 2's complimented numbers.
2) Use 'reset' signal to clear junk data from the registers.
3) data_out signal is only affected by OUT instruction. For other instructions, its value will be unchanged.

Program Counter (PC) & Instruction Memory (IM) Block

Input and Output:

Signals to PC_IM Block	Input variable Name	No. of Bits	Explanation
Inputs	jmp_loc	8	Address of jump location
	pc_mux_sel	1	Selection bit for jmp_loc or pc_out
	Stall	1	Selection bit for address control mux
	Stall_pm	1	Selection bit for instruction control mux
	reset	1	Reset signal for register, address and instruction
	clk	1	Input clock signal
Outputs	ins	24	Instruction
	Current_Address	8	Present address value for IM

Block Diagram:



Block Diagram Description:

- The objective of this block is to generate instructions for decode, execution and data memory blocks, and also take care of different controls.
- There are 3 different 2 x 1 multiplexer having selection line namely ‘Stall’, ‘pc_mux_sel’ and ‘Stall_pm’ respectively. ‘Stall’ generated from ‘stall control’ block (i.e. one of the module of the processor) which is used to select assign either Hold_Address or Next_Address as CAJ. ‘pc_mux_sel’ decides whether the current address will pass to program memory or address provided by ‘jump control’ block. A mux with ‘Stall_pm’ selection bit decides whether to pass actual instruction (PM_out) or pass previous instruction (ins_prv) to ‘ins_pm’. Another two multiplexers with ‘reset’ selection bit are used to reset ‘Current_Address’ and ‘ins’ values.
- Processor needs to generate new instruction at every positive edge of clock and ‘program counter’ will provide the address of the new instruction. Program counter is combination of ‘increment’ (+) and ‘register’ block. ‘Increment’ block increments address and register will pass that value to ‘Next_Address’ on every positive edge of the clock and when there is reset it will clear the output of ‘register’.
- ‘reset’ is a negative level triggered and clock synchronized.
- Another important block is program memory which has been designed using Xilinx IP core generator discussed separately in the next section.

Instruction Memory Block Specifications:

- In order to work with Memory Elements of Xilinx FPGA Devices, We need to be generated this memory block from IP core generator of Xilinx ISE
- Path: Device of your project (Right Click) -> new source -> IP (Core generator and Architecture Wizard) ->IP Catalog -> Memory and Storage elements -> RAM’s and ROM’s -> Block memory generator
- Select single port ROM
- Data Read width -: 24 bits
- Data Read depth -: 256 locations
- Memory Initialization -> Load Init file -> Import .coe file (*Read Point 2 on Guidelines for COE file generation and importing the same)
- Inputs -: [7:0] Current_Address, clk
- Output -: [23:0] PM_out

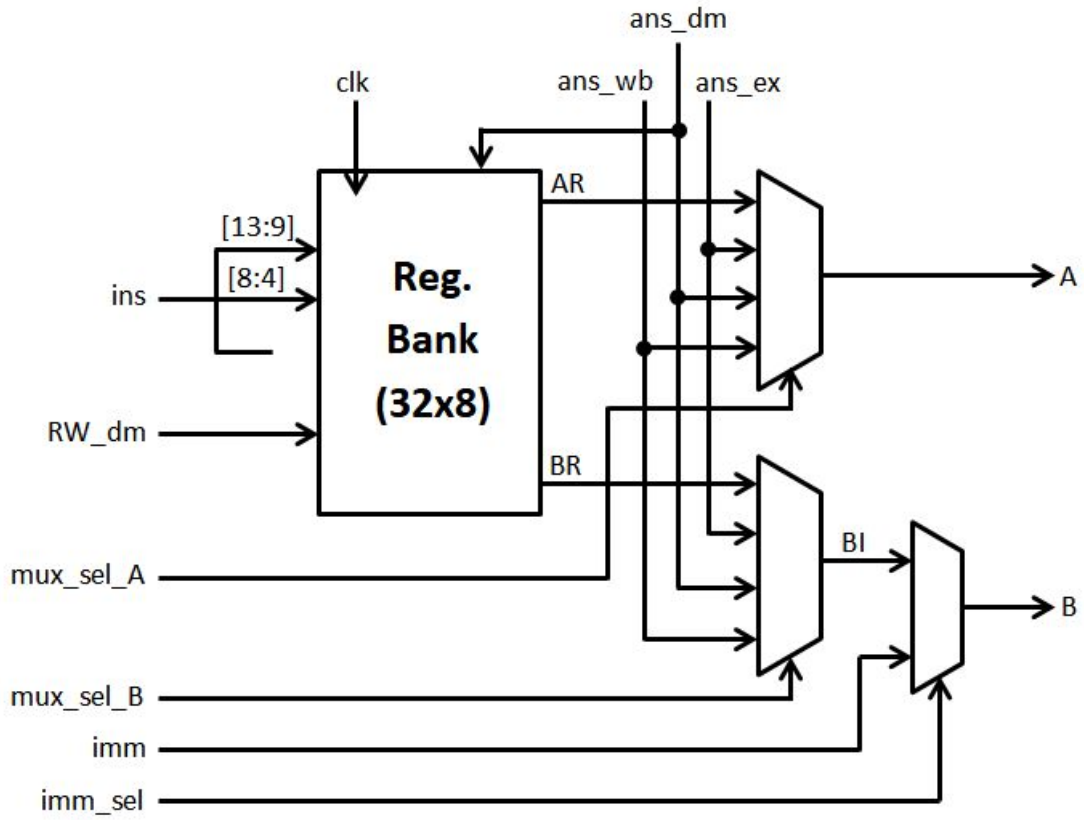
Generate IP core generator model from which you need to copy module for instantiation given by .veo file

Register Bank

Input and Output:

Signals to RB Block	Input variable Name	No. of Bits	Explanation
Inputs	ins	24	Instruction from Program Memory
	ans_ex	8	Answer from Execution block
	ans_dm	8	Answer from Data Memory block
	ans_wb	8	Answer from Writeback block
	imm	8	Reset signal for register, address and instruction
	RW_dm	5	Write address
	mux_sel_A	2	Mux selection bit for A
	mux_sel_B	2	Mux selection bit for BI
	imm_sel	1	Immediate number select
	clk	1	Input clock signal
Outputs	A	8	First operand for Execution block
	B	8	Second operand for Execution block

Block Diagram:



Block Diagram Description:
This block generates operands (A and B) for Execution block based on the instruction generated from Instruction Memory block. Instruction bits 13 to 9 are used as first operand (A) address and 8 to 4 are used as second operand (B) address.

If applied instruction is register type then both the operands fetched from register bank, else one of the operand (A) fetched from register bank and other operands (B) taken as immediate number (imm). This selection is controlled by 2x1 Mux.

Data forwarding operations performed by both 4x1 Mux based on the values applied to its selection line (mux_sel_A and mux_sel_B).

Answer from Data Memory block (ans_dm) is written in register bank as per the address indicated by RW_dm.

mux_sel_A	A
00	AR
01	ans_ex
10	ans_dm
11	ans_wb

mux_sel_B	BI
00	BR
01	ans_ex
10	ans_dm
11	ans_wb

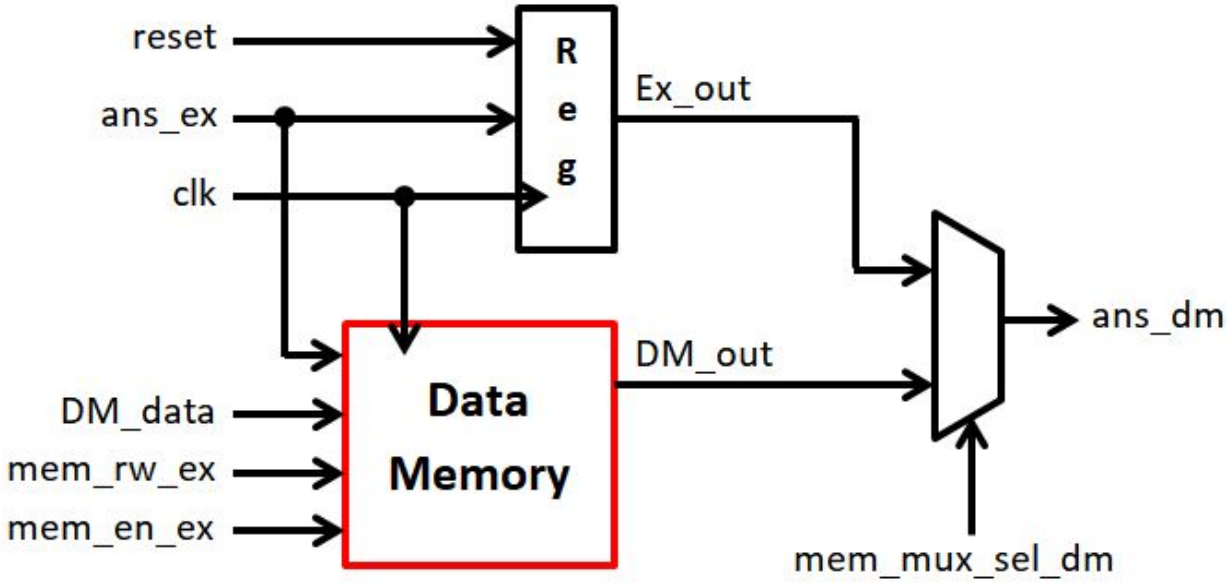
imm_sel	B
0	BI
1	imm

Data Memory Block

Input and Output:

Signals to DM Block	Input variable Name	No. of Bits	Explanation
Inputs	ans_ex	8	Answer from Execution block (used as address)
	DM_data	8	Input data for Data Memory
	mem_rw_ex	1	Memory read/write signal
	mem_en_ex	1	Memory enable signal
	mem_mux_sel_dm	1	Mux selection bit for ans_dm
	reset	1	Reset signal for register
	clk	1	Input clock signal
Outputs	ans_dm	8	First operand for Execution block

Block Diagram:



Block Diagram Description:

Data Memory is designed using IP core. It's a 256x8 RAM with R/W and enable signal. Data read or write in Data Memory is possible only if 'mem_en_ex' signal is 1. If 'mem_rw_ex' is 0 data read operation will perform else data write. Data written in the Data Memory should be provided to 'DM_data'; and 'ans_ex' is used as an address.

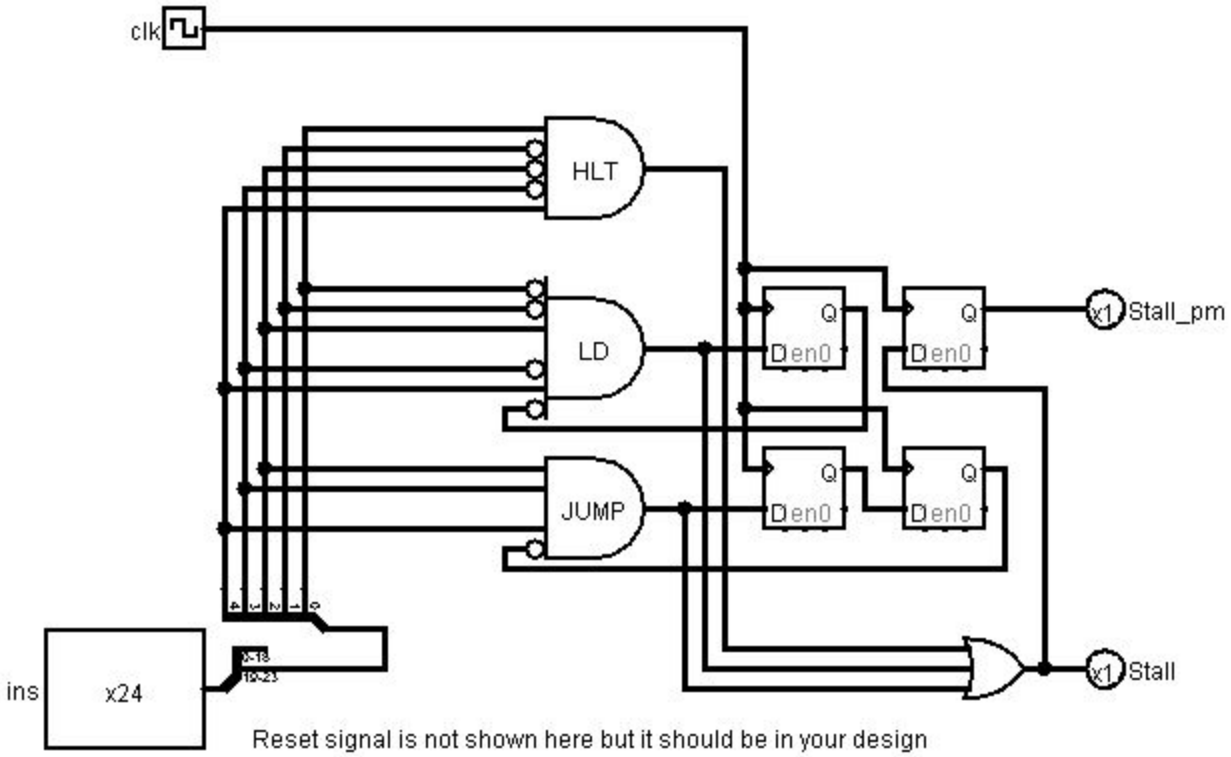
2x1 Mux is used to select between execution block answer (Ex_out) or Data Memory's output (DM_out). When 'mem_mux_sel_dm' is 1, it will select 'DM_out' as an 'ans_dm' else it will select 'Ex_out' as an 'ans_dm'.

Stall Control Block

Input and Output:

Signals to SC Block	Input variable Name	No. of Bits	Explanation
Inputs	ins	24	Instruction from Program Memory
	clk	1	Input clock signal
	reset	1	Reset for FFs
Outputs	Stall	1	Stall signal for PC_IM block
	Stall_pm	1	Delayed stall signal for PC_IM block

Circuit Diagram:



Stall Control Block Description:

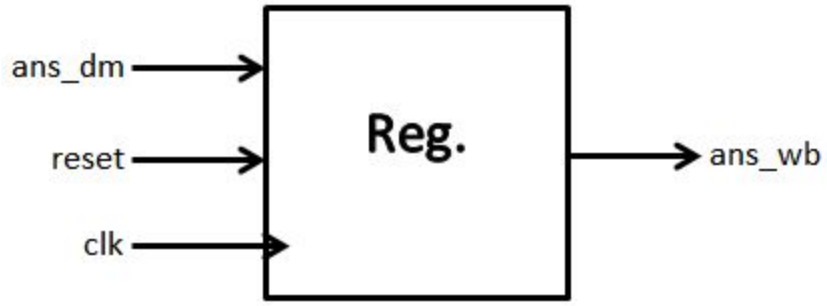
Some instructions like 'Load' and 'Conditional Jump' are creating control hazard in pipeline processor. To overcome this problem, we need to stall the pipeline for one or two clock cycles. Stall Control block decides when to stall a pipeline. Also, this block will stall the pipeline forever when 'HLT' instruction will execute.

Write Back Block

Input and Output:

Signals to WB Block	Input variable Name	No. of Bits	Explanation
Inputs	ans_dm	8	Output of Data Memory block
	clk	1	Input clock signal
	reset	1	Reset signal for register
Output	ans_wb	8	Output of Write Back block

Block Diagram:



Write Back Block Description:

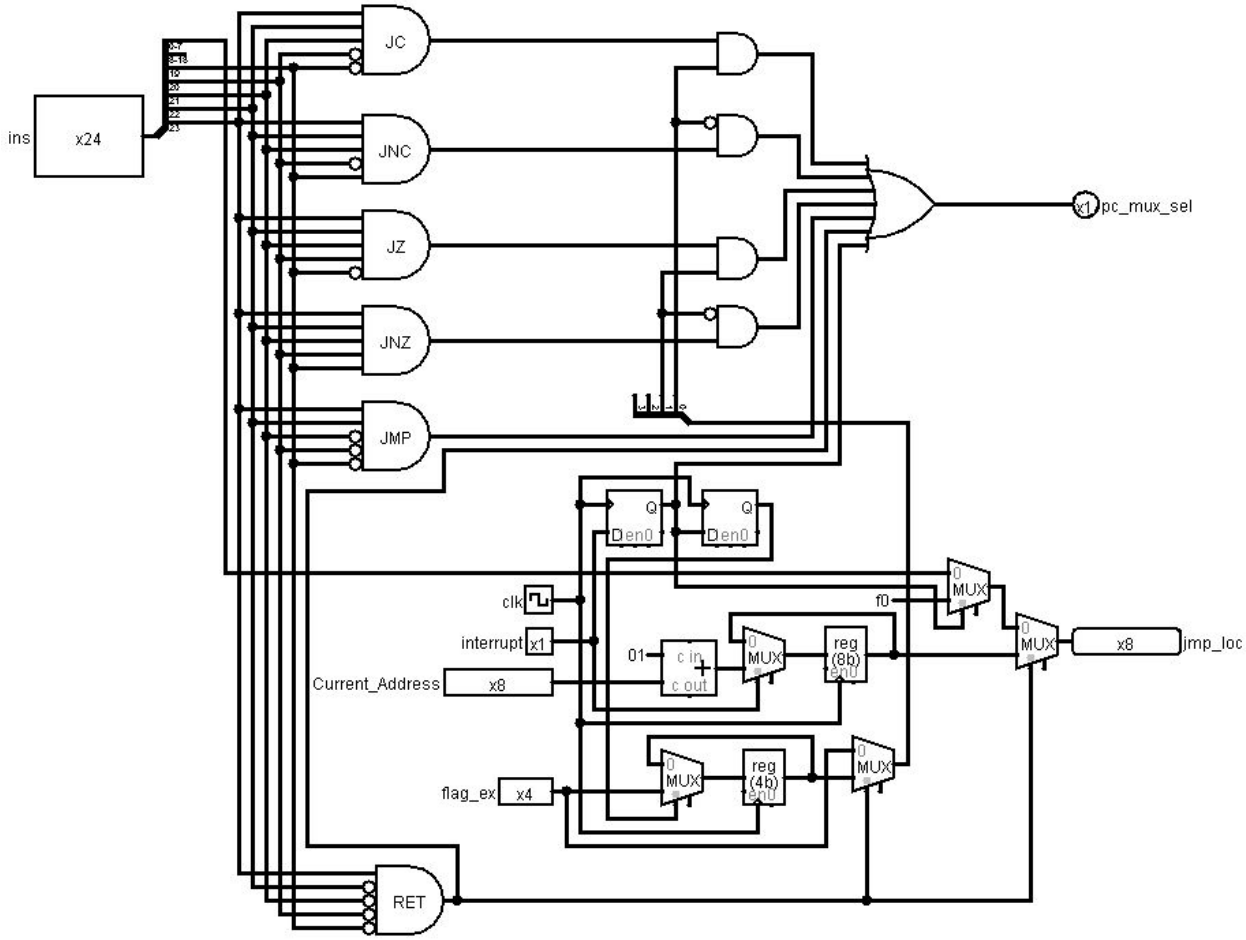
Write Back block is used to delay ‘ans_dm’ for one clock cycle. This block helps to resolve read after write hazard in the IDOF (Instruction Decode and Operand Fetch) stage.

Jump Control (JC) Block

Input and Output:

Signals to JC Block	Input variable Name	No. of Bits	Explanation
Inputs	ins	24	Instruction from Program Memory
	Current_Address	8	Address of current instruction
	flag_ex	4	Flag from execution block
	interrupt	1	External interrupt signal (Non maskable)
	clk	1	Input clock signal
	reset	1	Reset for registers/FFs
Outputs	jmp_loc	8	Jump address for PC_IM block
	pc_mux_sel	1	Mux selection bit for PC_IM block

Circuit Diagram:



Description:

Jump control block generates a jump location address (jmp_loc) and mux control signal (pc_mux_sel) for the program memory block, based on jump conditions and value of interrupt. For conditional jumps (JC, JNC, JZ and JNZ), if condition is true then ‘pc_mux_sel’ value will be ‘1’, else we check for unconditional jump (JMP), interrupt and return (RET) instructions, if any of these instruction/signal is true then also ‘pc_mux_sel’ value will be ‘1’, for other cases ‘pc_mux_sel’ value will be ‘0’.

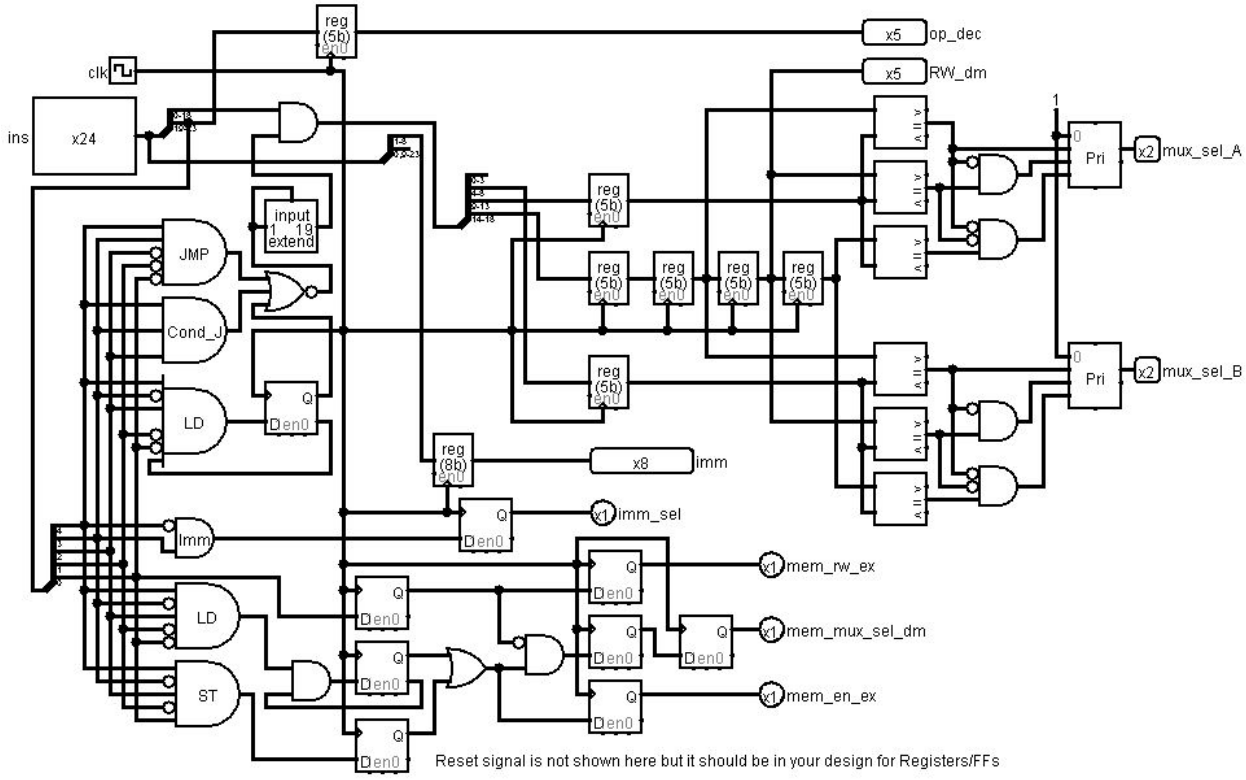
When interrupt signal value is ‘1’, we stores ‘current_address’ and ‘flag_ex’ in the system and program will jump to interrupt subroutine location (8’hF0). When interrupt subroutine is over then we provide RET instruction to continue our main program, for that we need to reproduce ‘current_address’ and ‘flag_ex’ which was previously stored in the system.

Dependency Check Block

Input and Output:

Signals to DC Block	Input variable Name	No. of Bits	Explanation
Inputs	ins	24	Instruction from Program Memory
	clk	1	Input clock signal
	reset	1	Reset for registers/FFs
Outputs	imm	8	Immediate number
	RW_dm	5	Write address
	op_dec	5	Opcode
	mux_sel_A	2	Selection bit for Mux A
	mux_sel_B	2	Selection bit for Mux B
	imm_sel	1	Immediate number select
	mem_en_ex	1	Memory enable signal
	mem_rw_ex	1	Memory read/write signal
	mem_mux_sel_dm	1	Memory mux select bit

Equivalent Circuit level diagram of Dependency check block:



Description:

Dependency check block handles data hazard in the pipeline processor. It can check the dependency of current instruction with previous three instructions and enables data forwarding in the pipeline by generating ‘mux_sel_A’ and ‘mux_sel_B’ signals. This block also generates memory control and immediate control signals. All outputs of this block are generated at the positive edge of the clock.