IIT Bombay
Department of Electrical Engineering

EE224 Course Project Report

Implementation of i281 CPU

Submitted by:

1. Pramod Chandra Pakanati — Roll No: 24B1269
2. Sai Vishal Reddy Malireddy — Roll No: 24B1293
3. Ganesh Preetham Vulise — Roll No: 24B1243

Guided by:
Prof. Sachin B. Patkar
Professor, Dept. of EE

November 23, 2025

Contents

Implementation of a Single-Cycle CPU

0.1 File Structure Description

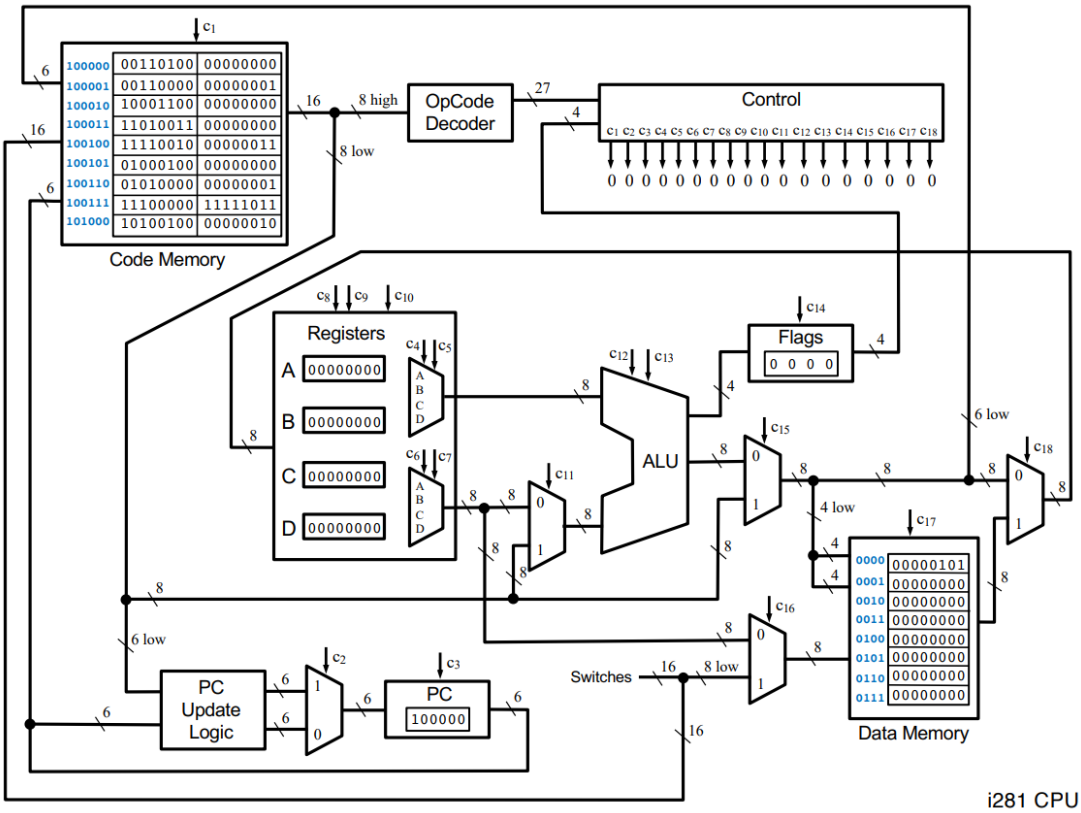

Figure 1: i281 CPU

This is the final realization of the i281 processor. It is organized into different verilog modules and a top-level module, CPU, connects them all. Each module corresponds to the functional blocks in the CPU architecture diagram.
The following subsections describe each major module present in the file structure.

0.1.1 OpCode Decoder

The OpCode Decoder takes the first 8 bits of the code memory instruction as input and decides what action should be performed by the CPU out of 23 actions such as ADD, LOADI, JUMP etc. This outputs a 27-bit string in which the first 23 bits are one hot encoded and last 4 are bits (mentioned as X1, X0, Y1, Y0) are used further in the control unit as required. We can see few commands are aliased and use same one hot string because they require same set of control bits.

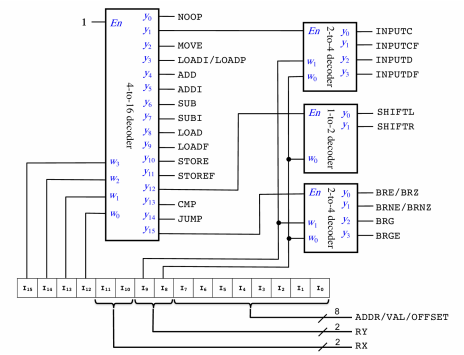

Figure 2: OpCode Decoder

0.1.2 Control

The control unit takes the 27-bit string and creates 18 control bits, which are assigned to different units in the CPU to instruct whether they should perform the task or not. The units will only allow changing the values stored when the WRITE_ENABLE signal of that particular unit is HIGH. Control bits also guide the ALU on what function needs to be done. We also consider the value of flags to fix the controls for Branch commands. The commands for this particular CPU are coded according to the table shown here.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}
	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_EN	REGISTERS_P_PORT0_SELECT1	REGISTERS_P_PORT0_SELECT0	REGISTERS_P_PORT1_SELECT1	REGISTERS_P_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESULT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADF			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

computed using the flags register

B1= ZF
B2= ~ZF
B3= AND(~ZF, XNOR(NF, OF))
B4= XNOR(NF, OF)

Zero Flag (ZF)
Negative Flag (NF)
Overflow Flag (OF)

Figure 3: OpCode Decoder

0.1.3 Code Memory

The Code Memory consists of 64 16-bit instructions of which first 8 bits tell us about what command should be executed and next 8 bits give the information about either address, offset or a value to be stored. These 16 bit instructions are stored in specific 6 bit addresses.

0.1.4 Data Memory

This is the 16-Byte memory block, where the unsorted numbers are given. By executing the commands in the code memory one by one, the numbers in the 4-bit addresses gets sorted one by one and by end of all commands, all the given numbers will be sorted. The data memory accepts the inputs during run time as well.

0.1.5 ALU

The ALU or Arithmetic Logic Unit performs the arithmetic and logical operations on the 8-bit inputs from the register file. It takes the control inputs c12 and c13 and implements the operations, Shift left, Shift right, Add and Subtract or Compare, based on them. This module internally uses adder_sub module for arithmetic operations, shifter module for shifting operations and flag_calculator module for computing zero and negative flags. The ALU flags register contains a total of four flags, carry, overflow, negative and zero.

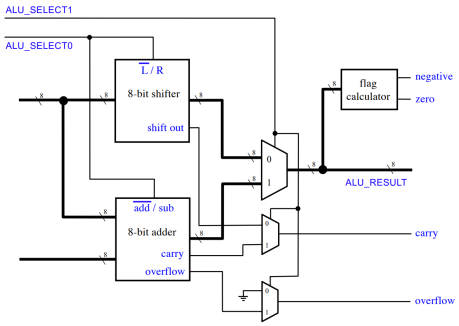

Figure 4: ALU

0.1.6 PC Update Logic

The PC or Program Counter update logic handles the sequential execution and jumps of the instructions from code memory. It takes the control input c2 for deciding on jump with an offset or just going to the next instruction. It passes through the PC register before updating the code memory.

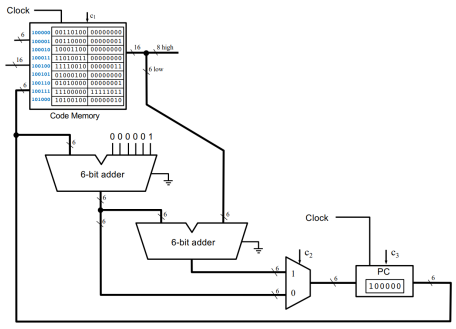

Figure 5: PC update logic

0.2 Simulation Waveforms

The following are the simulation waveforms on Modelsim for the execution of the bubble sort algorithm on the CPU.

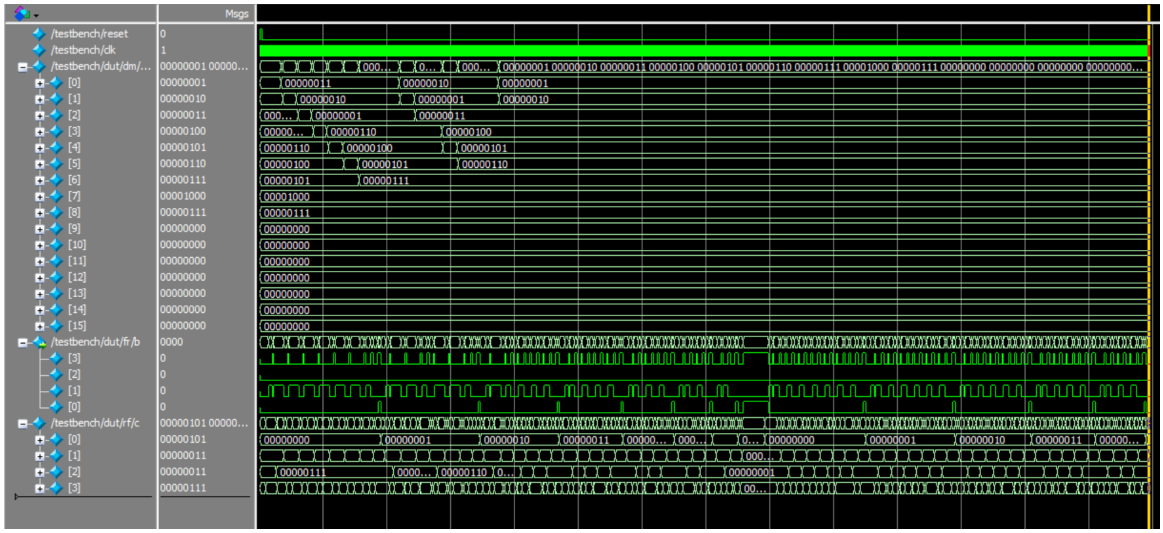

Figure 6: Full waveform

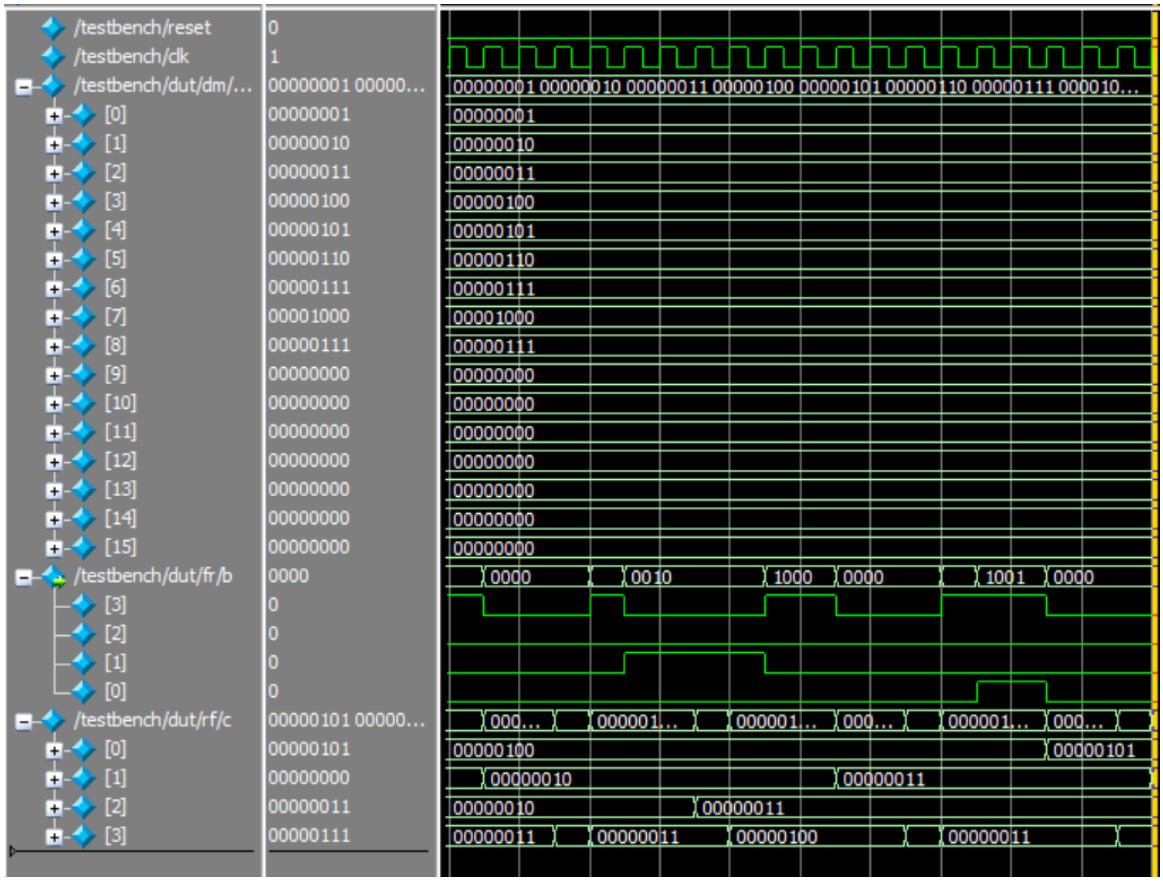

Figure 7: Waveform indicating final values

Implementation of a Multi-Cycle CPU

0.3 File Structure Description

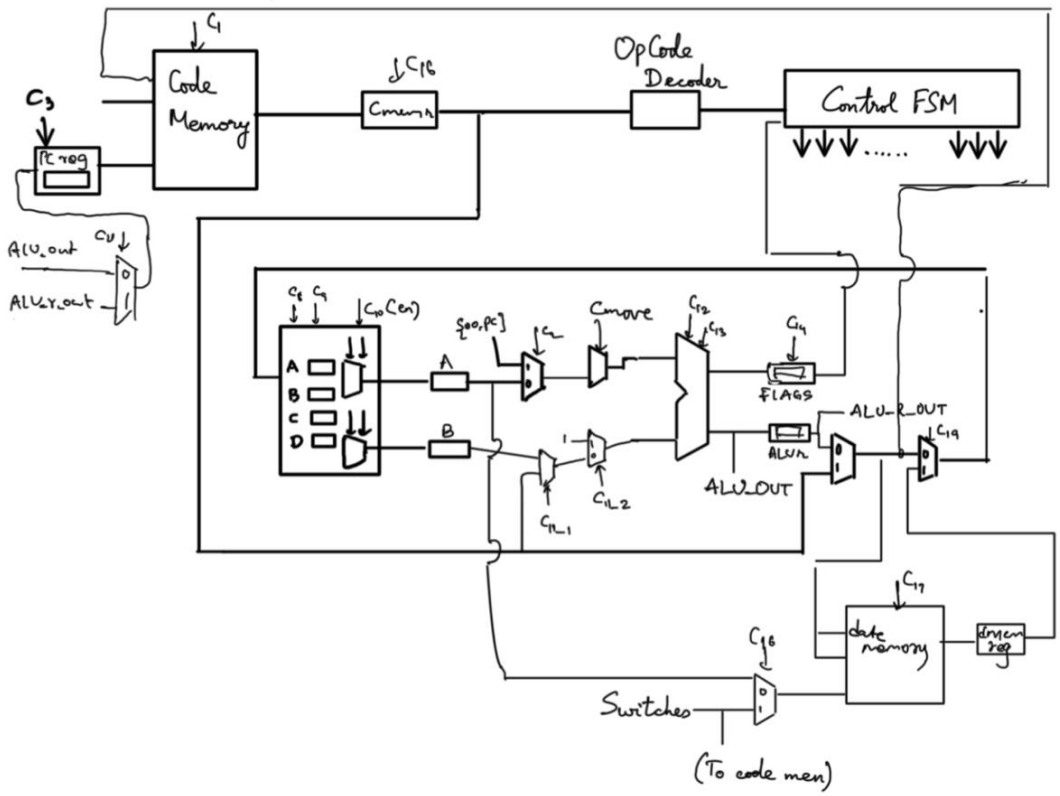

Figure 8: Multi-cycle CPU

The whole structure of the CPU can be seen from the code.

0.3.1 Datapath Modification

We have added new registers after code memory, data memory, ALU output and register file. It is to ensure that the corresponding values from the previous clock cycles can be used in the present cycles. We removed PC update logic block and designed the control such that the ALU does the update over successive clock cycles.

0.3.2 Control Modification

We designed a five state fsm, which has fetch, decode, cycle 1, cycle 2 and cycle 3 as its states.

Some instructions such as add and sub need 4 cycles to complete execution and jump type statements need 3 cycles to execute.