CprE 2810 HW13
ELECTRICAL AND COMPUTER ENGINEERING
IOWA STATE UNIVERSITY

i281 CPU architecture, assembly language, and register machines

Finish by Dec. 12, 2025

P1. (10 points) Type the program from page 114 in the assigned reading on register machines into RodRego: https://rodrego.it.tufts.edu/
Then, run the program for two different input conditions (different values stored in the registers). Provide several screenshots of the execution of your program.

P2. (10 points): (a) Explain the meaning of this register machine program.
(b) If the initial values of the registers are R1=3, R2=5, R3=4, and R4=2, what will be their final values?

STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	2	2	4
2.	Inc	1	3	
3.	Inc	3	1	
4.	End			

P3. (10 points) Consider a register machine with four registers R0, R1, R2, and R3. Let the initial values be: R0 = a, R1 = b, R2 = c, R3=0. The values a, b, and c are 3 random integers (positive or zero). Write the instructions (in the table format shown in P2) for a register machine program that computes 2(a+b) + 3c and stores the result in R3. Write a short comment for each line/block of your program. The program does not need to preserve the contents of registers R0, R1, and R2, as long as the result in R3 is correct.

P4. (20 points) ALU Circuit.

A. Draw the high-level circuit diagram for the ALU of the i281 CPU together with the Flags register. Label all inputs, outputs, and pins.
B. Explain how the output of the CMP instruction is computed. Does it set the flags?
C. Explain in detail how the Zero flag is computed.
D. Explain in detail how the ALU is used during the instruction ADDI B, 1. Where are the inputs coming from? Where is the result stored? Are the flags set?

P5. (20 points) Program Counter Circuit.

A. Draw the high-level diagram for the program counter circuit of the i281 CPU. Include the adders, the mux, and the register. Label all inputs, outputs, and pins.
B. Explain the role of the C_2 and C_3 control signals.
C. Explain how the new CMEM address is computed during a BRGE instruction.
D. Explain what changes will have to be made to the overall i281 architecture if the Program Counter circuit is implemented with an actual up-counter with parallel load capability instead of the current implementation that uses a register. Draw the high-level diagram for the new circuit.

CprE 2810 HW13
ELECTRICAL AND COMPUTER ENGINEERING
IOWA STATE UNIVERSITY

i281 CPU architecture, assembly language, and register machines

Finish by Dec. 12, 2025

P6. (10 points) The i281 assembler is written in Java. It takes an assembly program, which is stored in a plain text file with *.asm extension, and maps it to machine code. Download the sample assembly programs and the assembler from this link: https://www.ece.iastate.edu/~alexs/classes/i281_CPU/i281_CPU_Software.zip

Read the README.txt for instructions. Pick one of the sample programs and compile it (pick a file that is not in the instructions). To get the 10 points, attach one or more screenshots of the text that the assembler prints on the screen. Your ISU NetID should be visible in the screenshots.

P7. (10 points) Write an assembly program (in i281assembly language) that loads the variables X=3, Y=2, and Z=2 from data memory into registers A, B, and C, respectively. Then, computes 2(X + Y) – 3Z and stores the result in register D. Finally, it overwrites the value of Z in the data memory with the contents of register D.

P8. (10 points) Convert your assembly program from P7 into i281 machine code. You can do this by hand or with the java assembler. To get the 10 points, however, you must use the i281 simulator, which works in a web browser (for best results use Firefox): https://www.ece.iastate.edu/~alexs/classes/i281_simulator/index.html

Hint: Click on the "Load" button to try some of the stored examples. You may find the ones in the Arithmetic section particularly useful. Click the green button labeled "Go to CPU" and then "Step" multiple times to trace the execution of the program. You can also try the "Load New File" button to upload your own assembly file (in plain text format) into the simulator.

Provide a screenshot of your program loaded into the simulator with the syntax highlighting turned on. Your image should look like this one: