Back-end:

Cache: the cache uses FIFO algorithm to store the lasted data being used. When the next instruction is executed, the machine will first search whether the data in the EA was stored in the cache. If yes, the data will be gotten from the cache instead of the memory.

In this part of the project, we implement all instructions except those in Part 4.

Transfer Instructions:

JZ: if the content in the required GPR is zero, the EA will be given to PC. Otherwise, the number of PC will plus one.

JNE: if the content in the required GPR is not zero, the EA will be given to PC. Otherwise, the number of PC will plus one.

JCC: if the GPR number corresponds to the cc position with a value of 1, the EA will be given to PC. Otherwise, the number of PC will plus one.

JMA: the EA will be given to PC.

JSR: PC plus one and put that result into GRP3, put EA into PC.

RFS: put the address in the instruction into GPR0 and put the content in GPR3 into PC.

SOB: the content in the required PC minus one. If the result is larger than zero, put EA into PC. Otherwise, PC plus one.

JGE: if the content in required GPR is large or equal to zero, put EA into PC. Otherwise, PC plus one.

Arithmetic and Logical Instructions:

AMR: the content in required GPR plus the content in EA, and put the result into the required GPR.

SMR: the content in required GPR minus the content in EA, and put the result into the required GPR.

AIR: the content in required GPR plus the address in the instruction, and put the result into the required GPR.

SIR: the content in required GPR minus the address in the instruction, and put the result into the required GPR.

MLT: multiply the content in two required GPR, put the high order bits of the result into the first GPR and the low order bits into the first GPR plus one.

DVD: Divide the content in two required GPR, using the first GRP divides the second one. Put the quotient into the first GPR and the remainder into the first GPR plus one.

TRR: if the content of the two required GPR are equal, set the fourth bit of CC to 1, otherwise set it to 0.

AND: logical And of the content of the two required GPR, and put the result into the result into the first GPR.

ORR: logical Or of the content of the two required GPR, and put the result into the result into the first GPR.

NOT: logical Not of the content of the two required GPR, and put the result into the result into the first GPR.

Shift/Rotate Operations

SRC: if the L/R =1, shift the content if the required GPR left and append 0 at the end. Otherwise, shift right and append 0 at the head.

RRC: when the A/L is 1. If the L/R =1, rotate the content if the required GPR left. If the L/R is 0

when the A/L is 1, rotate right.

I/O Operations

IN: pop out a window which allows user to input an integer. After finish inputting, the machine will get the first bit of the integer and store it, and then get the second bit until the integer is loaded all in.

OUT: the system will output the aimed integer into the scroll window.

CHK: the machine check whether the input window still has character which is not being stored. If yes, keep store the next bit of the integer and the system will not pop out the input windows anymore. Otherwise, the machine will allow to pop out the input windows to let the user input another integer.

Program1:

When the user load the Program1.txt file, the system will store all the instructions in the file into the memory and the PC will jump to 256. When running the program, the user can input 21 integers. During each input cycle, the system will first print out the words: "PLZ INPUT" into the GUI and then pop out a window to let user input integer. After finishing input all integers, the system will out put the closest number in inverted order into the GUI. For example, if the user input 1 – 20 and 21, the system will print 02.

However, our program may view the last input integer as the closest number.

Our team uses a list named memory to simulate the computer memory. The index of it is MAR and the value corresponding to each MAR is MBR. All the content in the memory will be initiated as 0000000000000000.

Clicking SS button, the program will get the value of PC, and use it as the address to find the instruction in that address from the memory. Then store the first six binary values into Opcode, put seventh and eighth binary values into GPR, put the nineth and tenth binary values into IXR, put the Eleventh binary value into I, put the last five binary value into Address. After that, computing EA using I, Address and IXR and analyzing the Opcode to do LDR, STR, LDA, LDX, or STX.

Clicking RUN button, the program will go through all instructions until meet the condition of HALT.

GUI:

We used 12 label to simulate and display the binary in it. They are the four GPR, three IXR, PC, MAR, MBR, IR and MFR. Using 16 button displayed on the bottom of the window to let the user enter binary and load it in the GPR IXR registers, PC, MAR or MBR for future use. Put Store, St+, Load, Init, SS, and Run button on the right side