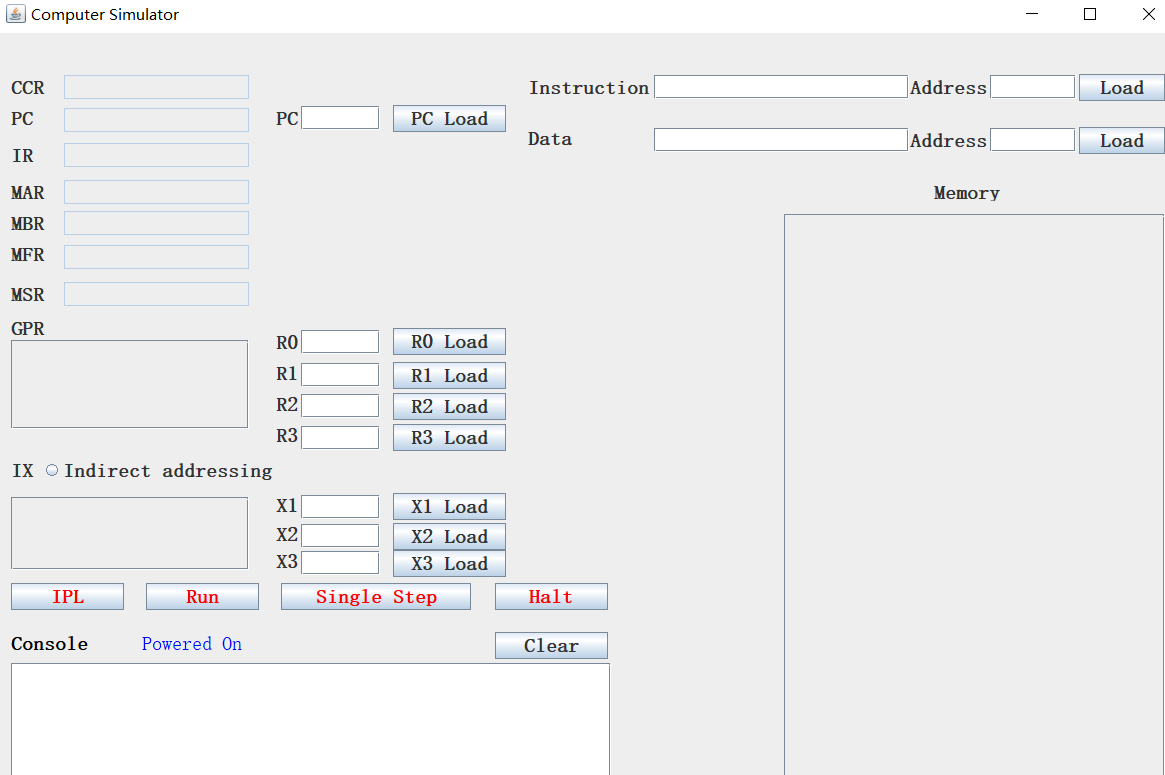
Short description of the layout of the computer simulator

1. **Start up the application**

When start up the simulator that this computer is powered on, but it does not load any instructions or data into the memory.



1. **load bootstrap into memory**

Then when you click “IPL”, each instruction from the bootstrap file will be encoded to binary number and then moved to the memory begin with the location 8. Meanwhile, the PC will be set to 8, pointing to the first address of the boot program.

1. **Reserved memory**

Memory location 0 – 7 is reserved, they are:

0 Reserved for the Trap instruction for Part III.

1 Reserved for a machine fault (see below).

2 Store PC for Trap

3 Not Used

4 Store PC for Machine Fault

5 Not Used

1. **Execute instructions**

Click “Run”, the simulator will run all instructions from the PC pointing location and stop at the location with empty instruction.

For boot program, PC returns to the first address of the boot program, which is 8, and display the cpu is idle.

For other program, PC will return to the next address of the previous instruction.

1. **The data flow while executing instructions**

One instruction is fetched and executed in the following order:

PC->MAR->Memory->MBR->IR->InstructionDecoder->SingalController->Different operations, such as copy data from memory to registers.

1. **MFR**

MFR records the machine fault code when there is machine fault happens. In our simulator, now we defined the following machine fault code:

/\*

\* ID Fault

\* 0 Illegal Memory Address to Reserved Locations

\* 1 Illegal TRAP code

\* 2 Illegal Operation Code

\* 3 Illegal Memory Address beyond 2048 (memory installed)

\* 4 Illegal Access Memory address. No allocation

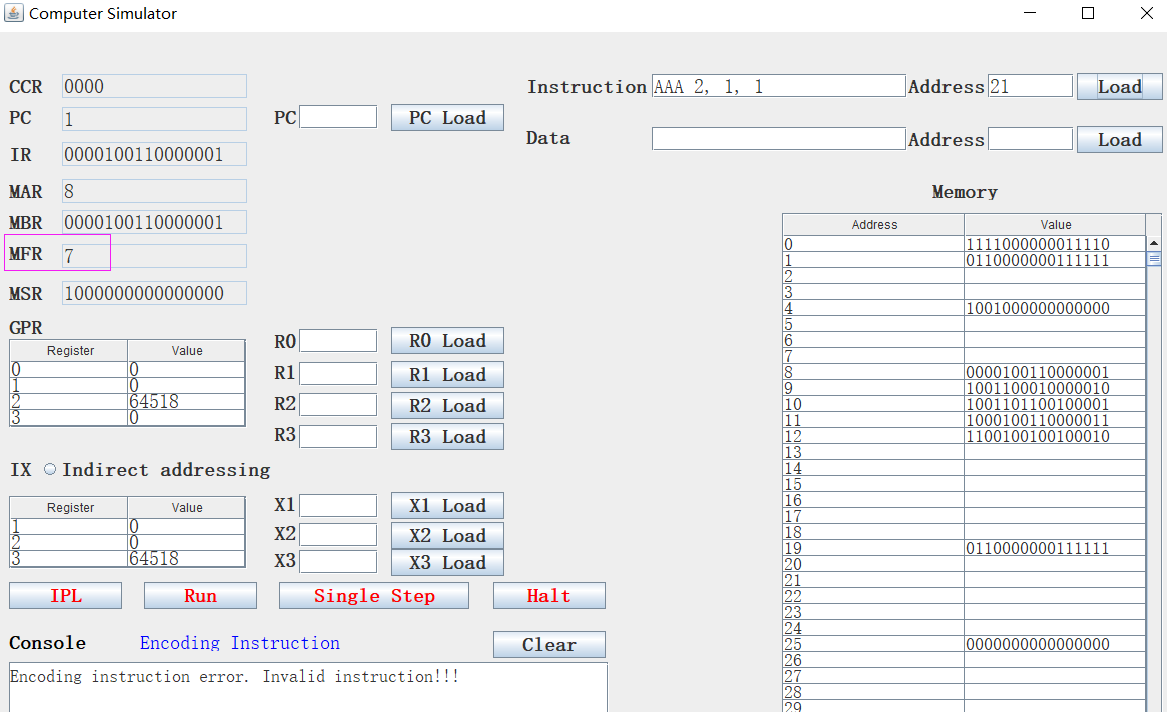
\* 5 Illegal IX index (1-3)

\* 6 Illegal GPR index (0-3)

\* 7 Encode instruction error

\*/

We will append more later. For example, when users try to load an invalid instruction to memory, the console printer will display “Encoding instruction error. Invalid instruction, meanwhile, the MFR will changed to 7. See the following screen short:



1. **CCR**

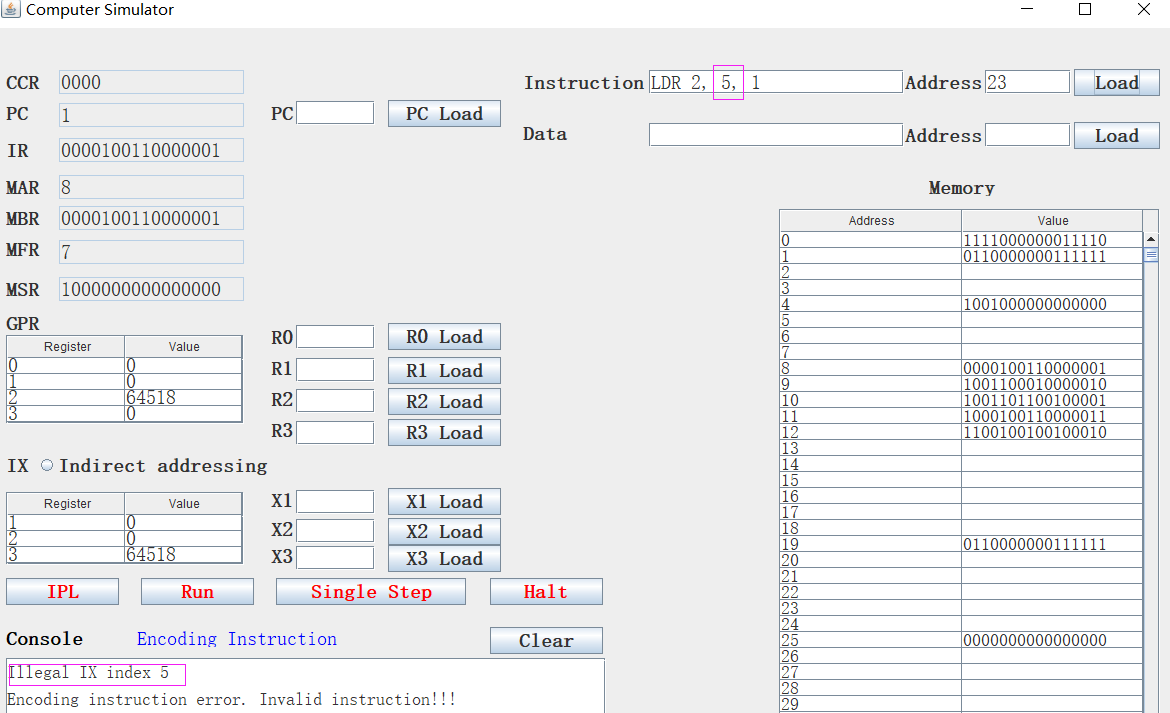
We haven’t implemented the logic of CCR, we will do it later.

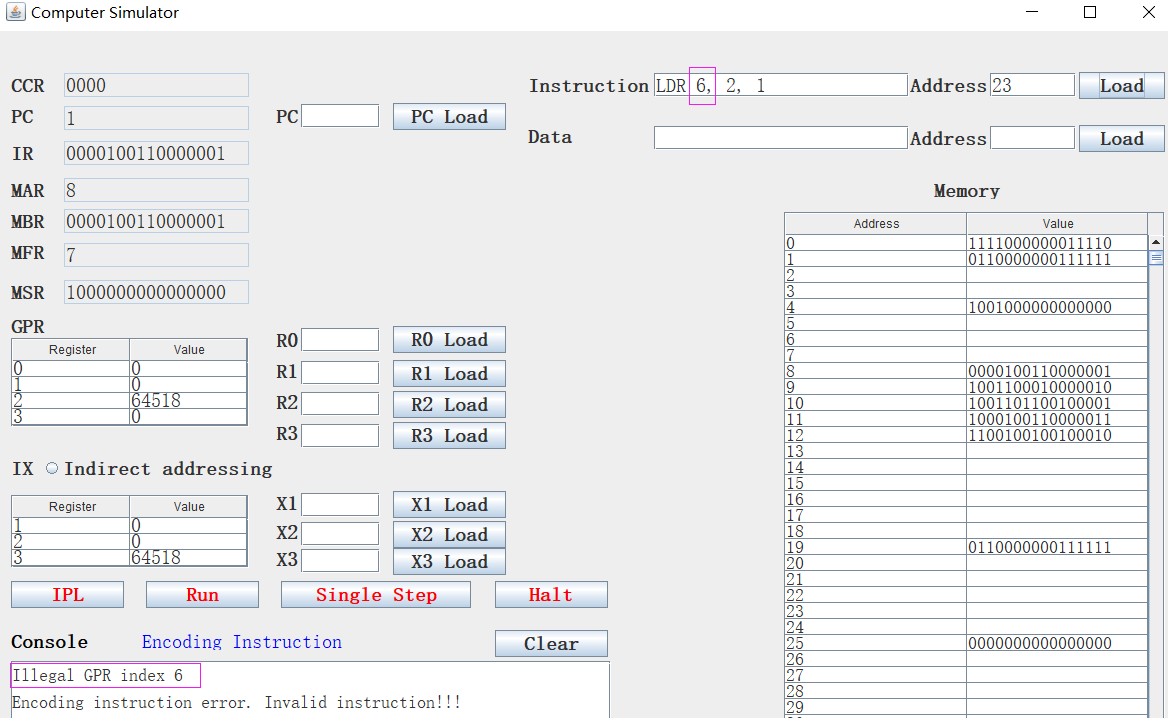
1. **MSR**

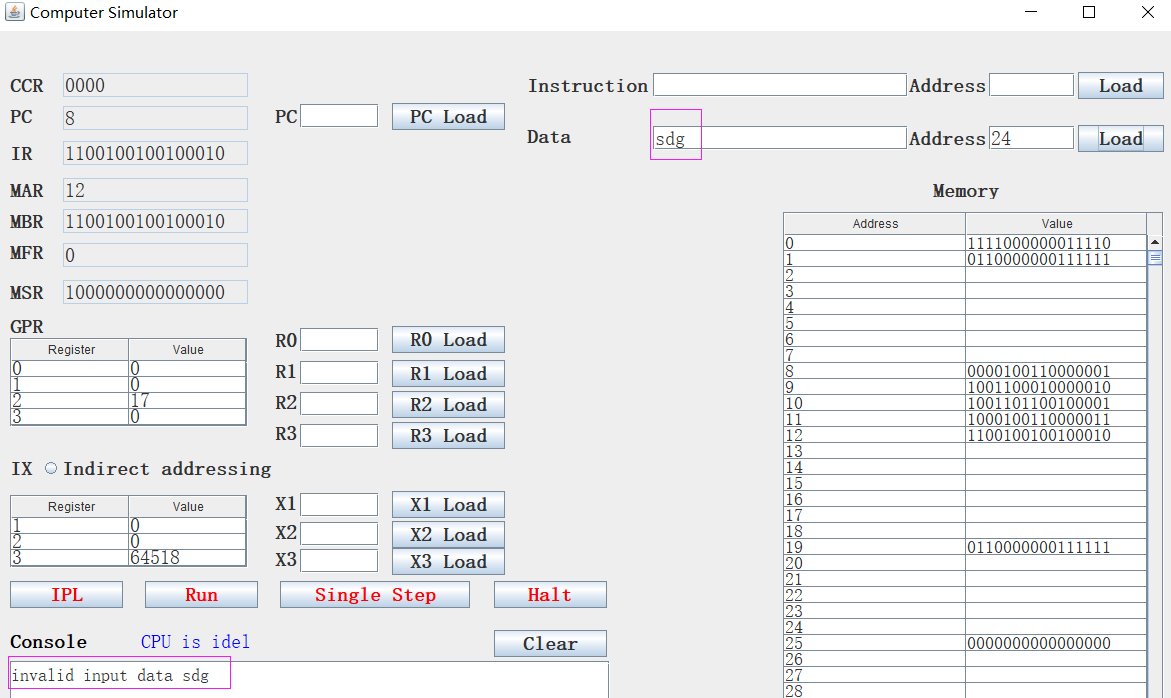
We haven’t implemented the logic of MSR. We will do it later. Now, we only set it to 1, meaning nothing.

1. **Fault-tolerant**

Our program performs various fault-tolerant processing, such as:







Also includes the invalid input number for registers.

1. **Indirect addressing and direct addressing**

We add a radio button to let users choose which addressing mode he wants. For example:

LDR 2, 1, 17 X1 = 20, M[37] = 30, M[30] = 100, M[100] = 111

If select direct addressing, EA = c(X1) + 17 = 20 + 17 = 37. r(2) <- c(EA) = M(37)= 30

If select no direct addressing, EA = c(c(X1) + 17) = M(37)= 30. r(2) <-c(c(EA)) = M[100] = 111