# Design Notes

November edition

## Group Member

Cheng Yi:

Designed for the CPU class, instructions execution, cache, memory argument, code text of program 2

Mochen Xia:

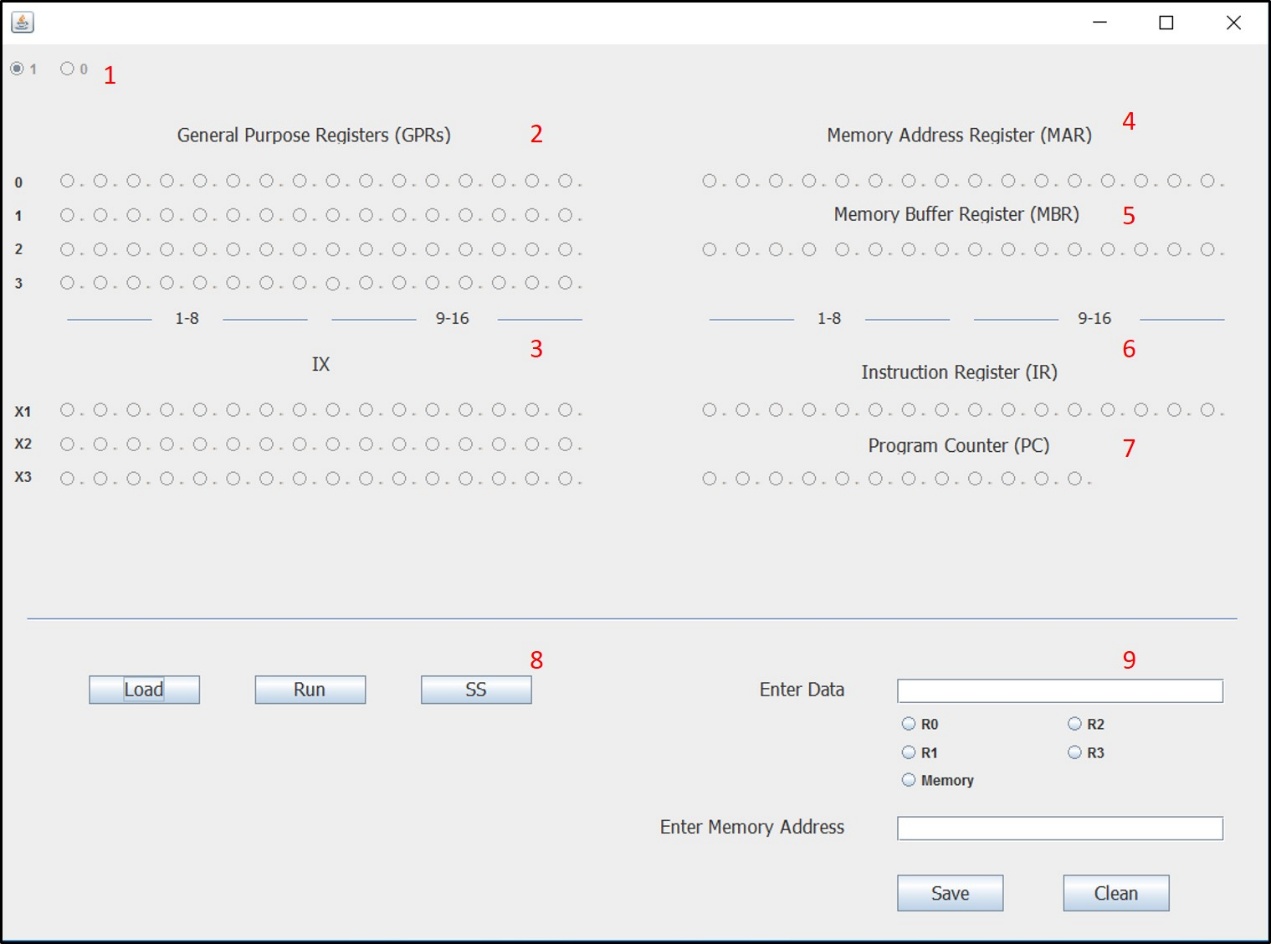
Added button listeners on the GUI that called functions from CPU class, instruction execution, program 1

Baosheng Feng:

Designed for the GUI, instruction execution, program 1

## Part 1

### UI Part Design



The above picture is the UI of simulation project.

**1**: The examples of “1” and “0” in the UI.

In this UI design, each circle indicates 1 bit in GPRs, IX, MAR, MBR, IR, and PC.

The circle with a black spot presents “1” in binary system. The hollow circle presents “0” in binary system.

**2**: General Purpose Registers (GPRs) R0-R3. It is a quickly accessible location available to a digital processor’s central processing unit. From left to right is number 1 bit to number 16 bit.

**3**: Index Register X1, X2, and X3. It contains a base address that supports base register addressing of memory. From left to right is number 1 bit to number 16 bit.

**4**: Memory Address Register (MAR). It holds the address of the word to be fetched from memory. From left to right is number 1 bit to number 16 bit.

**5**: Memory Buffer Register (MBR). It holds the word just fetched from or the word to be last stored into memory. From left to right is number 1 bit to number 16 bit.

**6**: Instruction Register (IR). It holds the instruction to be executed. From left to right is number 1 bit to number 16 bit.

**7**: Program Counter (PC). It is the address of next instruction to be executed. From left to right is number 1 bit to number 12 bit.

**8**: Button Load, Run, and SS

Load: This button is used to load an instruction txt file in the src folder, and PC will show user the value when load is done.

Run: This button is used to execute the instruction file. The file will be executed to the end if user use this button.

SS: This button is used to execute the instruction step by step. When user clicks this button, it will run one line in the instruction file.

**9**: Button Save and Clean. Data text field and Memory Address text field.

Save: This button is used to save and execute what users have put in. There are five choices for users to choose where to store the data: R0, R1, R2, R3 or Memory. Data will be read from the data text field. If users want to save data into memory, they must enter the address of the memory in memory address text field.

Clean: This button is used to set all button in UI to “false” for next step.

Data text field: This field asks users to enter 16-digit binary code for data.

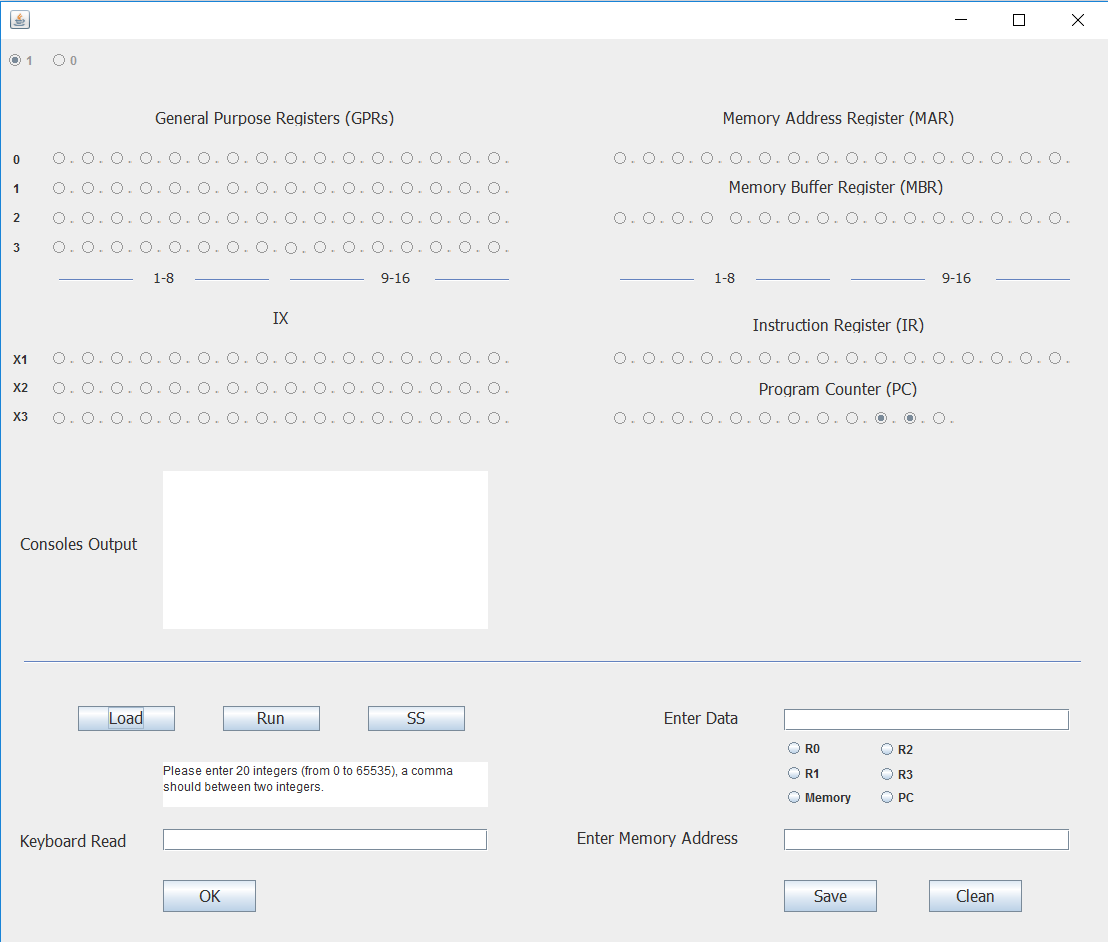
Memory Address text field: This field asks users to enter an integer that among 6 and 2048.

### CPU Part Design

The CPU class is the class that will handle all the back end operation of the cpu. All the register are represented using short variables and the memory are represented by a short array. For every instruction, the user is going to run the executeNext() function which will execute the instruction at the address the program counter (pc) is pointing to. The cpu will then parse the instruction and do a switch case based on the op code (first 6 digit) of the instruction. After the instruction function is executed, the pc will be incremented by 1 and be ready for the next instruction to be called.

## Part 2

### UI Part Design



For the part II of the project, we have done some adjustments to the UI design:

1. add PC button. Address can be put into PC by this button.

2. add Consoles output window. It is the printer of test program 1.

3. add keyboard read area. 20 integers can be input into computer by this area.

### CPU Part Design

#### Instructions:

Realizing the below opcodes.

**No.10 opcode:**

This instruction, jump if zero, in our program is jz(short r, short x, short address, short indirect).

r means the number of general purpose registers. x means the number of index registers. address is the address information in the instruction. Indirect means the indirect addressing. We use x, address and indirect to find the effective address (EA).

In this instruction, if the contents of register r(c(r)) is zero, then pc=EA or c(EA), else pc=pc+1.

**No.11 opcode:**

This instruction, jump if not equal, in our program is jne(short r, short x, short address, short indirect).

If c(r)!=0, pc=EA or c(EA), else pc=pc+1.

**No.12 opcode:**

This instruction, jump if condition code, in our program is jcc(short x, short address, short indirect).

cc means condition code. If cc =1 which is underflow, pc=EA or c(EA), else pc=pc+1.

**No.13 opcode:**

This instruction, unconditional jump to address, in our program is jma(short x, short address, short indirect).

pc=EA or c(EA)

**No.14 opcode:**

This instruction, jump and save return address, in our program is jsr(short x, short address, short indirect).

R3=pc+1, pc=EA or c(EA), r0=pointer to arguments

**No.15 opcode:**

This instruction, return from subroutine, in our program is rfs(short immed).

R0=immed, pc=c(R3).

**No.16 opcode:**

This instruction, subtract one and branch, in our program is sob(short r, short x, short address, short indirect).

R=c(r)-1, if c(r)>0 pc=EA or c(EA), else pc=pc+1.

**No.17 opcode:**

This instruction, jump greater than or equal, in our program is jge(short r, short x, short address, short indirect).

If c(r)>=0 pc=EA or c(EA), else pc=pc+1.

**No.20 opcode:**

This instruction, multiply two registers together, in our program is mlt(short rx, short ry)

Rx, ry must be 0 or 2, then we put the high order bits into rx and low order into rx+1.

**No.21 opcode:**

This instruction, divide two registers, in our program is dvd(short rx, short ry)

Rx, ry must be 0 or 2, when c(ry) = 0, DIVZERO flag will be set, then we put quotient into rx and remainder into rx+1.

**No.22 opcode:**

This instrument, test if two registers are equal, in our program is trr(short rx, short ry)

EQUALORNOT flag will be set to 0 (false) or 1 (true).

**No.23 opcode:**

This instrument, logical AND of two registers, in our program is and(short rx, short ry)

The result will be stored into rx.

**No.24 opcode:**

This instrument, logical ORR of two registers, in our program is orr(short rx, short ry)

The result will be stored into rx.

**No.25 opcode:**

This instrument, logical NOT of a register, in our program is not(short rx)

The result will be stored into rx.

**No.31 opcode:**

This instrument, will shift register by count, in our program is src(short ri, short count, short lr, short al)

Register will be shifted left (lr = 1) or right (lr = 0) either logically (al = 1) or arithmetically (al = 0).

**No. 32 opcode:**

This instrument, will rotate the register by count, in our program is rrc(short r, short count, short lr, short al)

Register will be rotated left (lr = 1) or right (lr = 0) by count.

**No. 61 opcode:**

This instrument, will input character to register from device, in our program is in(short r, short devid)

The devid defines which kind of devices.

**No. 62 opcode:**

This instrument, will output character to device from register, in our program is out(short r, short devid)

**No. 63 opcode:**

This instrument, will check device status to register, in our program is chk(short r, short devid)

#### Cache Design

We set a simple cache between memory and the rest of the processor that use ADT sort queue to store the addresses and content of data in memory. In each cycle, before search memory, we will first search the address in cache. If we cannot find the address, we will search the memory, and put a 16 bits word into cache. If we find the address in cache, we will use the content stored in cache.

#### Test Program 1

The codes are shown as follows.

501,AIR R2 20

6,1500

7,500

502,LDX X2 X0 7

503,LDX X1 X0 6

504,IN R0 0

505,OUT R0 1

506,STR R0 X1 0 0

507,STX X1 X0 0 10

508,LDR R1 X0 0 10

509,AIR R1 1

510,STR R1 X0 0 10

511,LDX X1 X0 0 10

512,SOB R2 X2 0 4

513,IN R0 0

514,STR R0 X1 0 0

515,OUT R0 1

516,LDA R0 X1 0 00

517,STR R0 X0 0 10

518,LDX X3 X0 0 10

519,LDX X1 X0 0 6

520,STX X2 X0 0 10

521,LDR R1 X0 0 10

522,AIR R1 26

523,STR R1 X0 0 10

524,LDX X2 X0 0 10

525,AIR R2 20

526,LDR R1 X1 0 0

527,SMR R1 X3 0 0

528,JGE R1 X2 0 6

529,STR R1 X0 0 10

530,SMR R1 X0 0 10

531,SMR R1 X0 0 10

532,STR R1 X1 0 21

533,STX X1 X0 0 10

534,LDR R1 X0 0 10

535,AIR R1 1

536,STR R1 X0 0 10

537,LDX X1 X0 0 10

538,SOB R2

539,STX X2 X0 0 10

540,LDR R1 X0 0 10

541,AIR R1 24

542,STR R1 X0 0 10

543,LDX X2 X0 0 10

544,STX X1 X0 0 10

546,LDR R1 X0 0 10

547,AIR R1 1

548,STR R1 X0 0 10

549,LDX X1 X0 0 10

545,LDX X3 x0 0 10

550,AIR R2 20

551,LDR R1 X1 0 0

552,SMR R1 X3 0 0

553,STX X1 x0 0 10

554,JGE R1 X2 0 5

555,LDX X3 X0 0 10

556,LDR R1 X0 0 10

557,AIR R1 1

558,STR R1 X0 0 10

559,LDX X1 X0 0 10

560,SOB R2 X2 0 0

561,LDA R1 X3 0 0

562,SIR R1 21

563,STR R1 X0 0 10

564,LDX X1 X0 0 10

565,LDR R1 X1 0 0

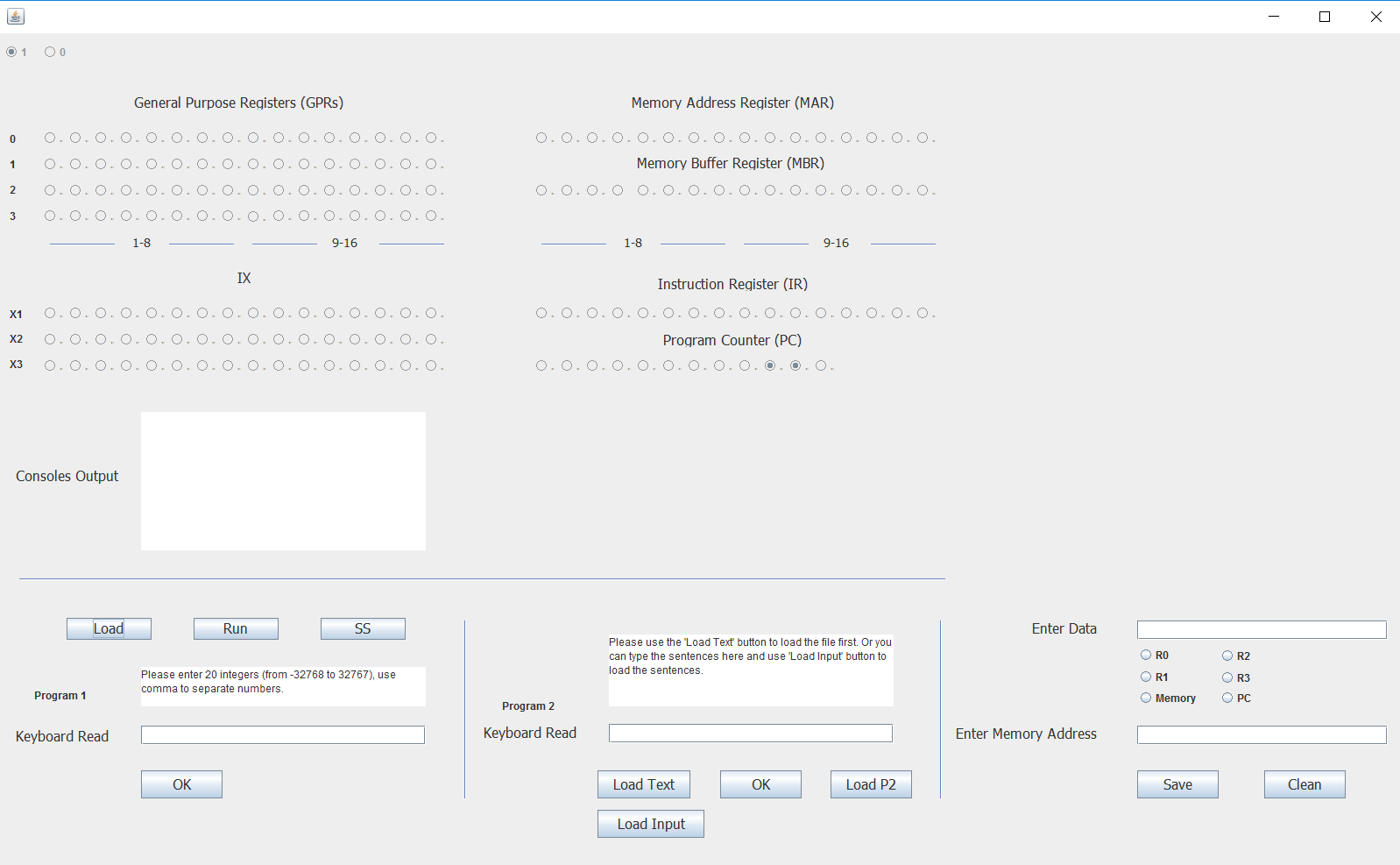
566,OUT R1 1

1401,1500

1403,1521

## Part 3

### UI Design Part



For the requirement of program 2 test, some designs of the UI have been adjusted. The positions of program 1 and part 1 input were changed. And, there is a new area for the input of program two test.

In the new area, there are one input text field and four button.

**Keyboard Read**: Keyboard input area.

**Load Text**: This button is used to load the text file which already contains six sentences.

**Load Input**: You can type six sentences in the keyboard read area. Then, use ‘Load Input’ button to load these sentences into memory.

**OK**: After you load the text file or load the typed sentences, you can type the word which is used to search in the keyboard read area. Then, use ‘OK’ button to load this word into memory.

**Load P2**: This button will load the program 2 which is machine code and run this program.

The results will output in the Consoles Output window.

#### Test Program 2

The codes are shown as follows.