

CSED311 Lab3: Single Cycle CPU

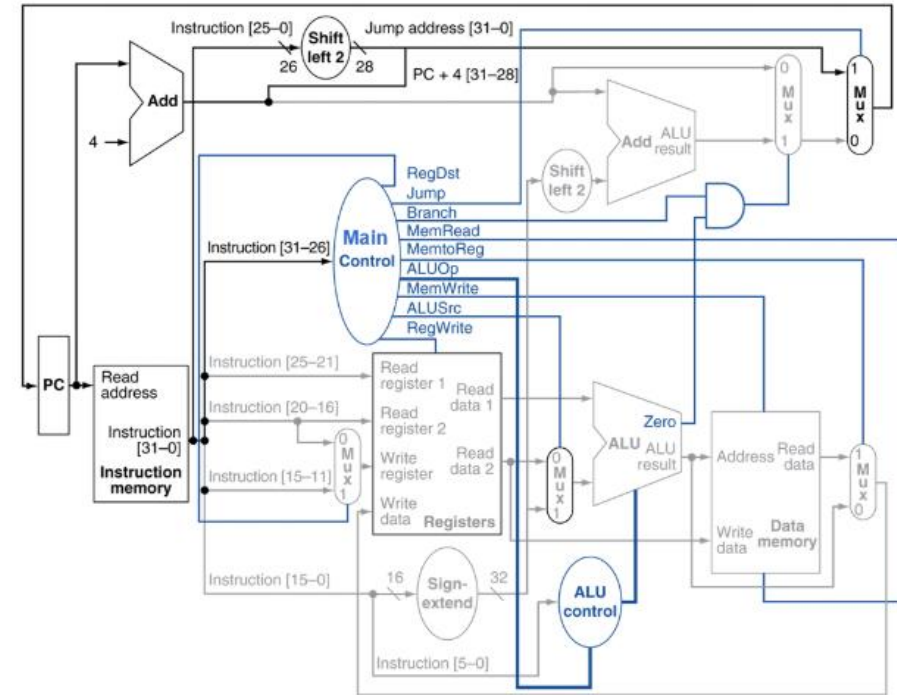
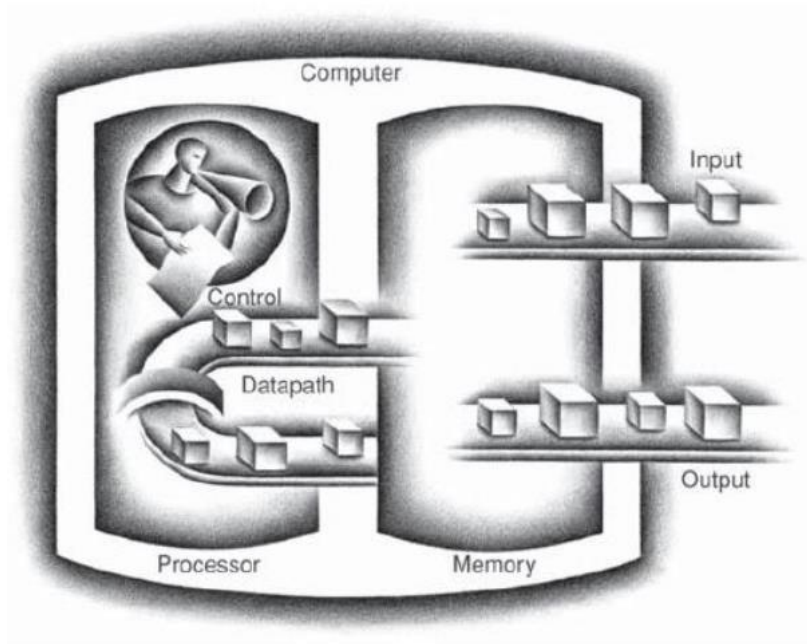
Jaejun Ha

dreamline91@postech.ac.kr

Contents

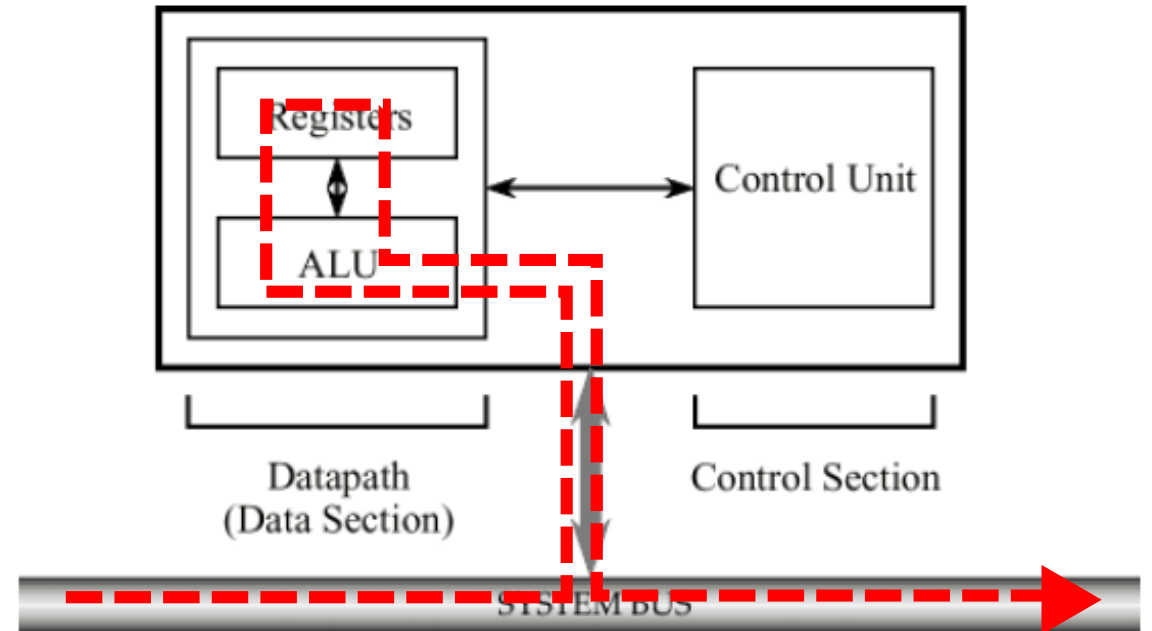
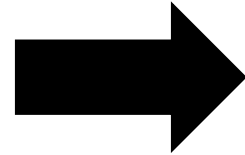
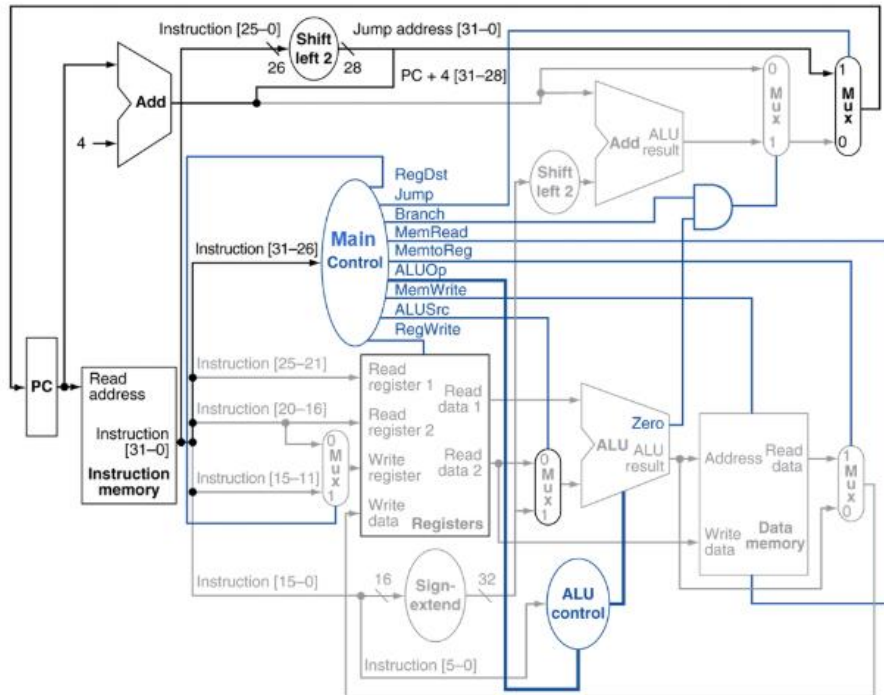
- CPU structure
 - Datapath
 - Control Unit
- TSC CPU
- Assignment
- Tips
- Lab2 Demo

CPU structure



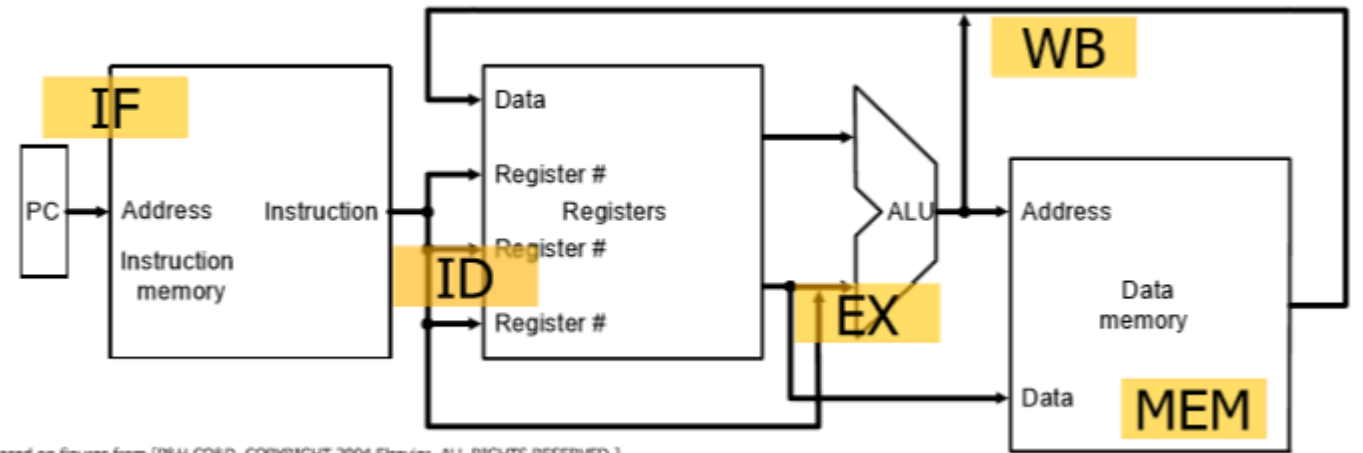
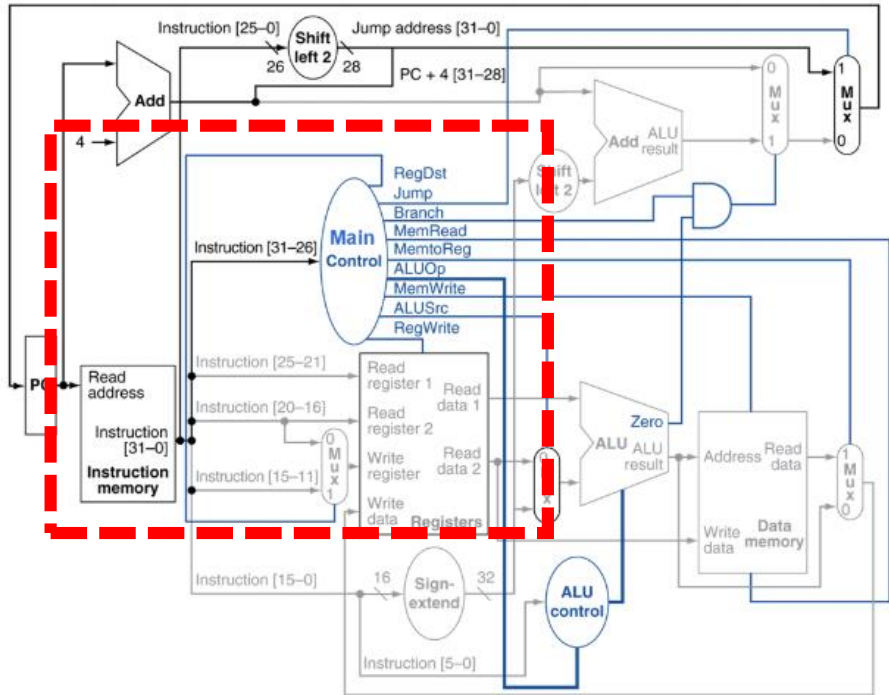
- CPU consists of two components
 - Datapath
 - Control Unit

Datapath



- Datapath:
 - Units in the path of data
 - SYSTEM BUS → ALU → Register → ALU → SYSTEM BUS
 - Instruction fetch, ALU, Register, Memory

Control Unit

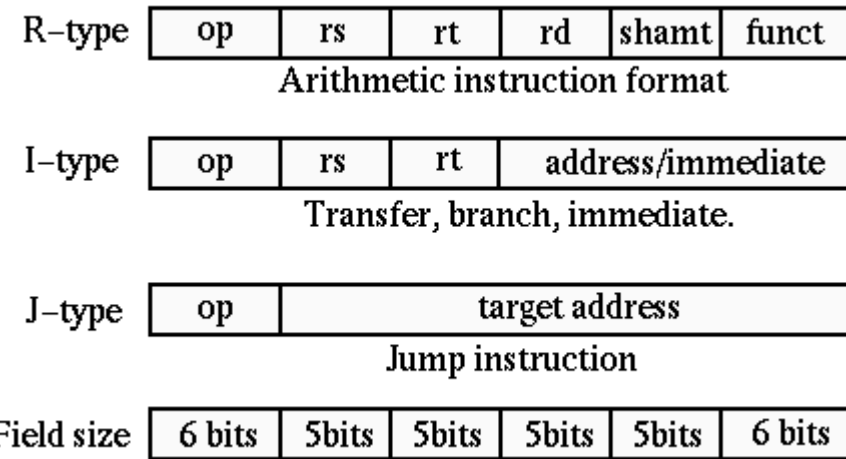


**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

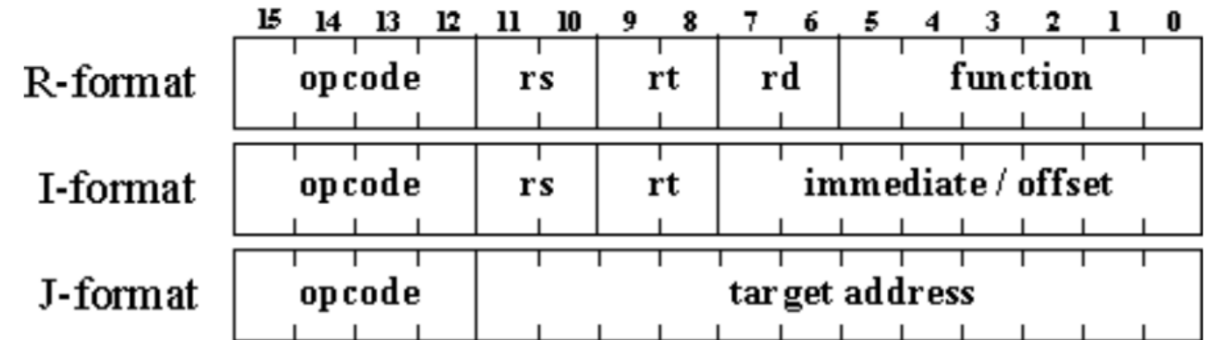
- Control Unit:
 - It decodes instruction
 - It generates control signals which are used in datapath

TSC CPU

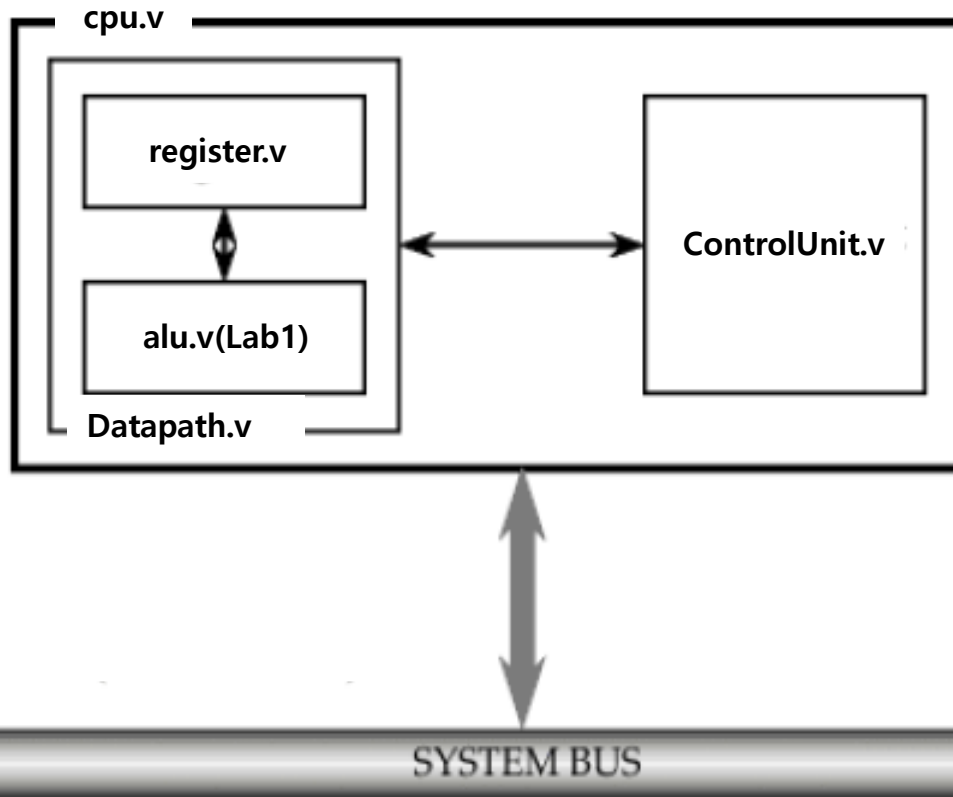
- MIPS CPU vs TSC CPU



- Has less bits
- more simple structure



Assignment



- Implement Single Cycle CPU
 - Single-cycle CPU
 - Datapath
 - for 16-bit CPU (4 registers)
 - Control unit
 - to generate control signals used in datapath
 - Instructions
 - refer to opcodes.v

Tips

- CPU module port

output	readM	"read" signal to memory
output	writeM	"write" signal to memory
output	[`WORD_SIZE-1:0] address	target memory address
input	[`WORD_SIZE-1:0] data	data for reading or writing
input	ackOutput	signal from memory ("data is written")
input	inputReady	signal from memory ("data is ready for reading")
input	reset_n	reset CPU
input	clk	clock signal

Tips

- Testbench

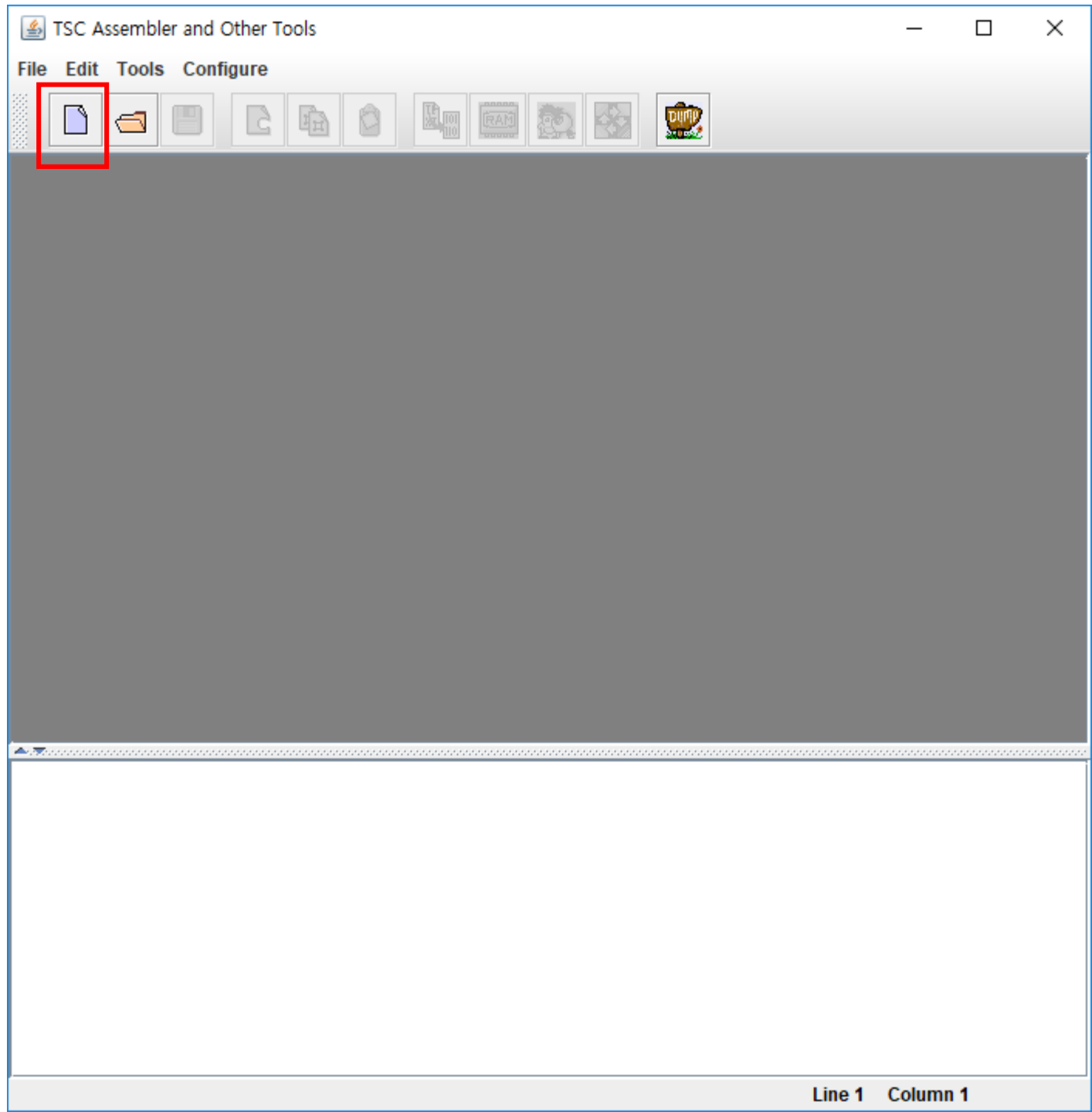
```
assign data = (readM || inputReady) ? loadedData : `WORD_SIZE'bz;
always begin
  loadedData = `WORD_SIZE'bz;
  #`PERIOD1;
  forever begin
    wait (readM == 1 || writeM == 1);
    if (readM == 1) begin
      #`READ_DELAY;
      loadedData = memory[address];
      inputReady = 1;
      #(`STABLE_TIME);
      inputReady = 0;
      loadedData = `WORD_SIZE'bz;
    end else if (writeM == 1) begin
      memory[address] = data;
      #`WRITE_DELAY;
      ackOutput = 1;
      #(`STABLE_TIME);
      ackOutput = 0;
    end
  end
end // of forever loop
end // of always block for memory read
```

Tips

- TSC assembler
 - `tsc_assem > assem > runit.bat`
 - It needs Java
 - It can make binary code

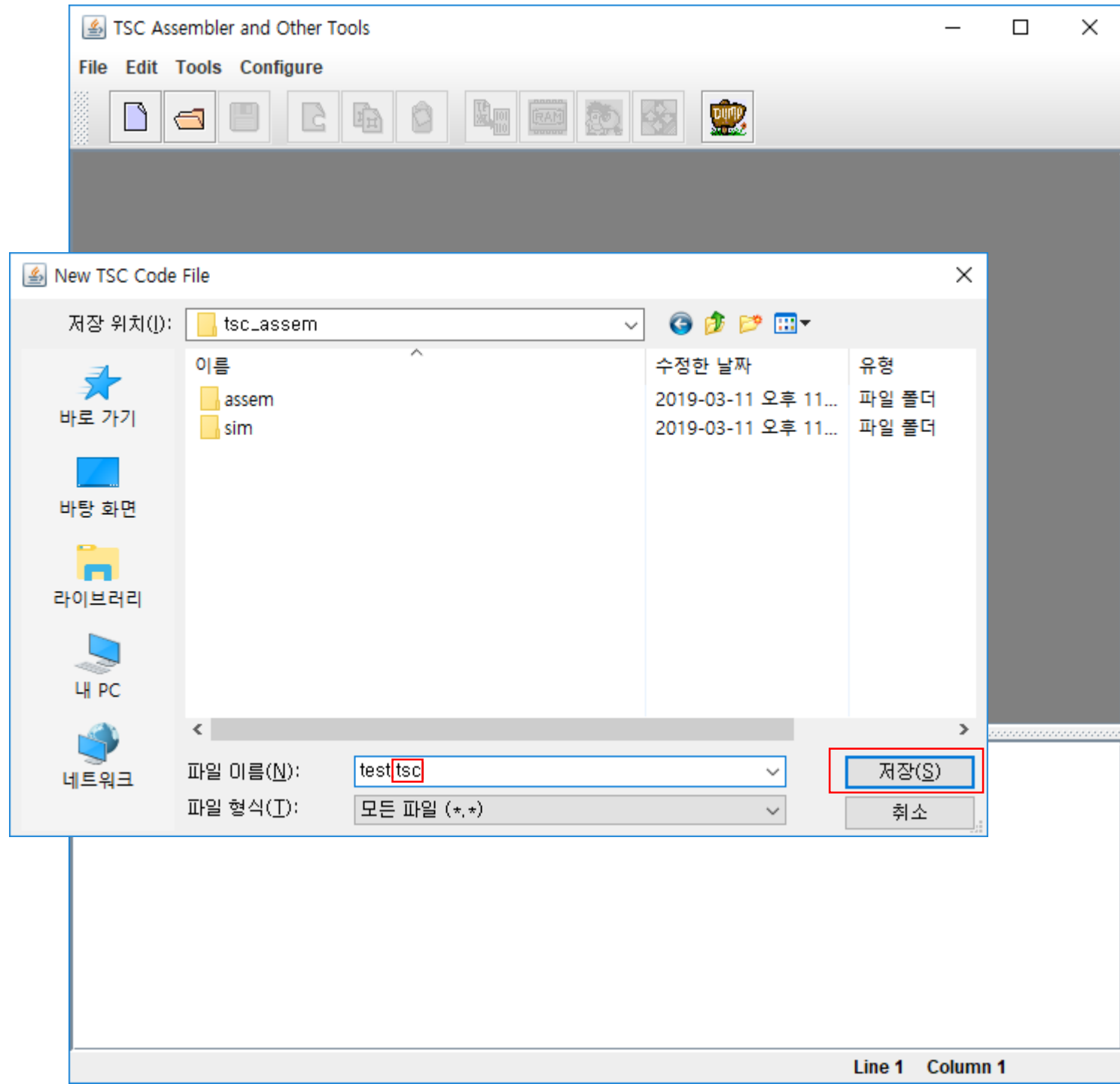
Tips

- TSC assembler



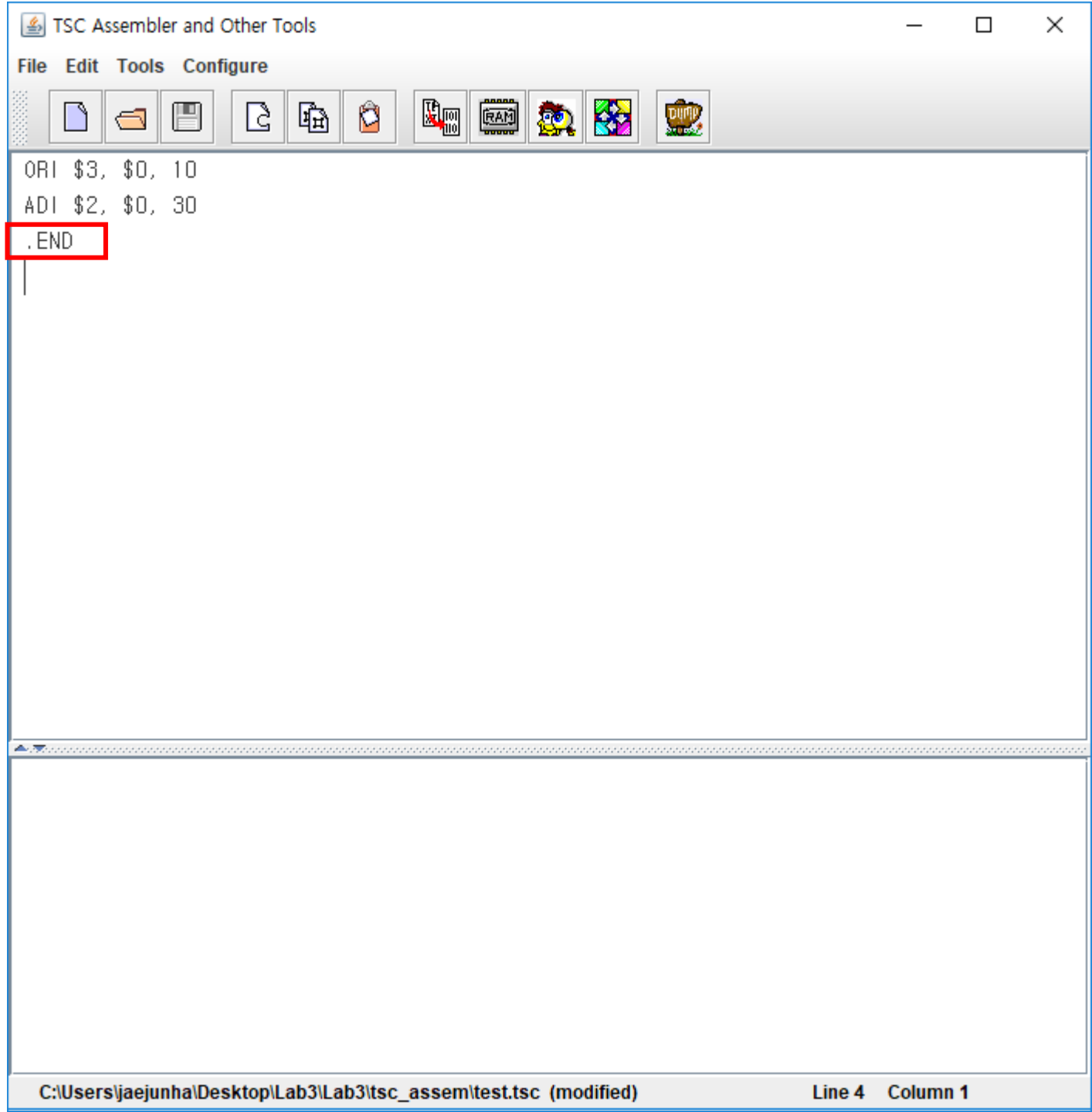
Tips

- TSC assembler



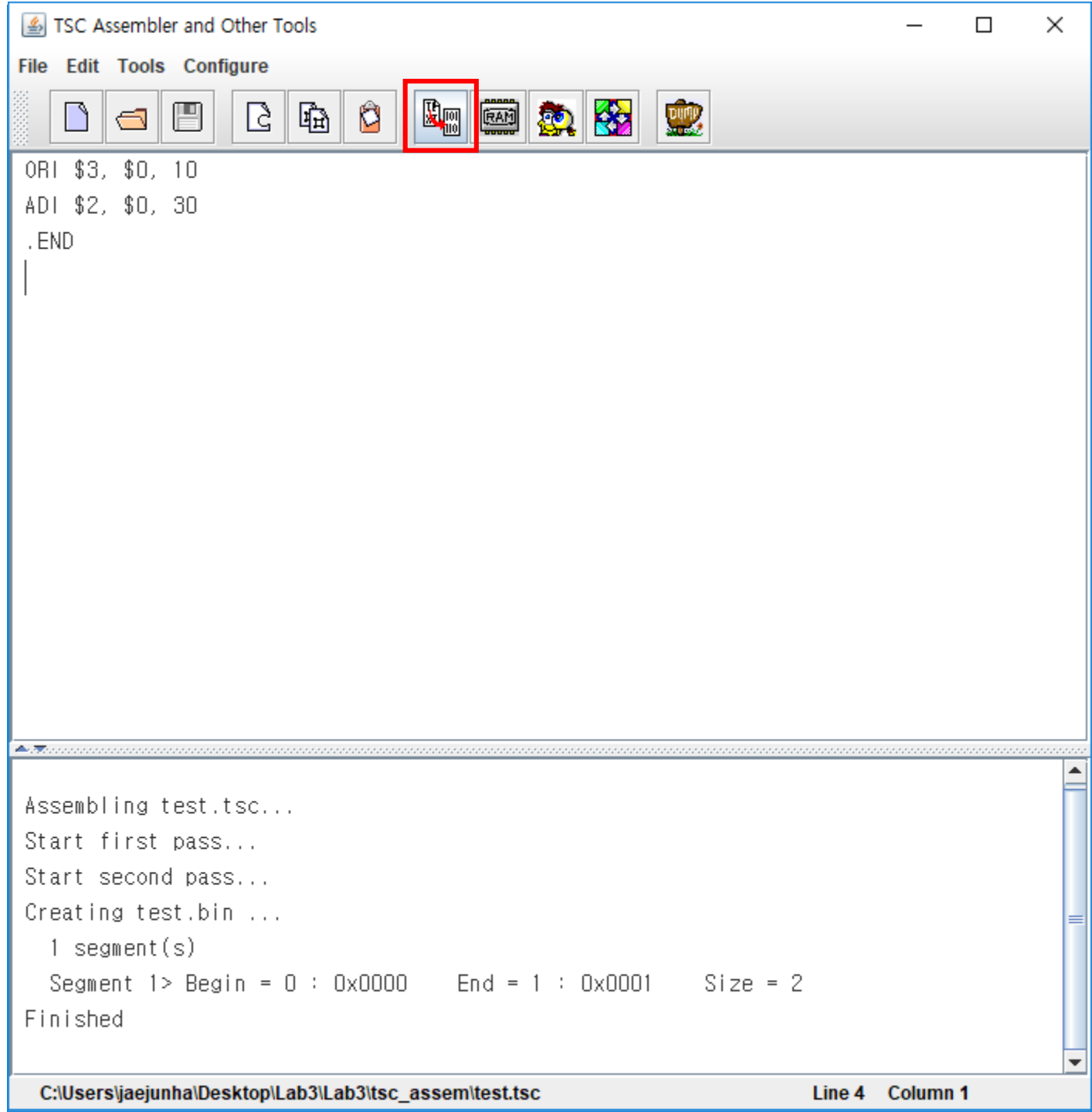
Tips

- TSC assembler



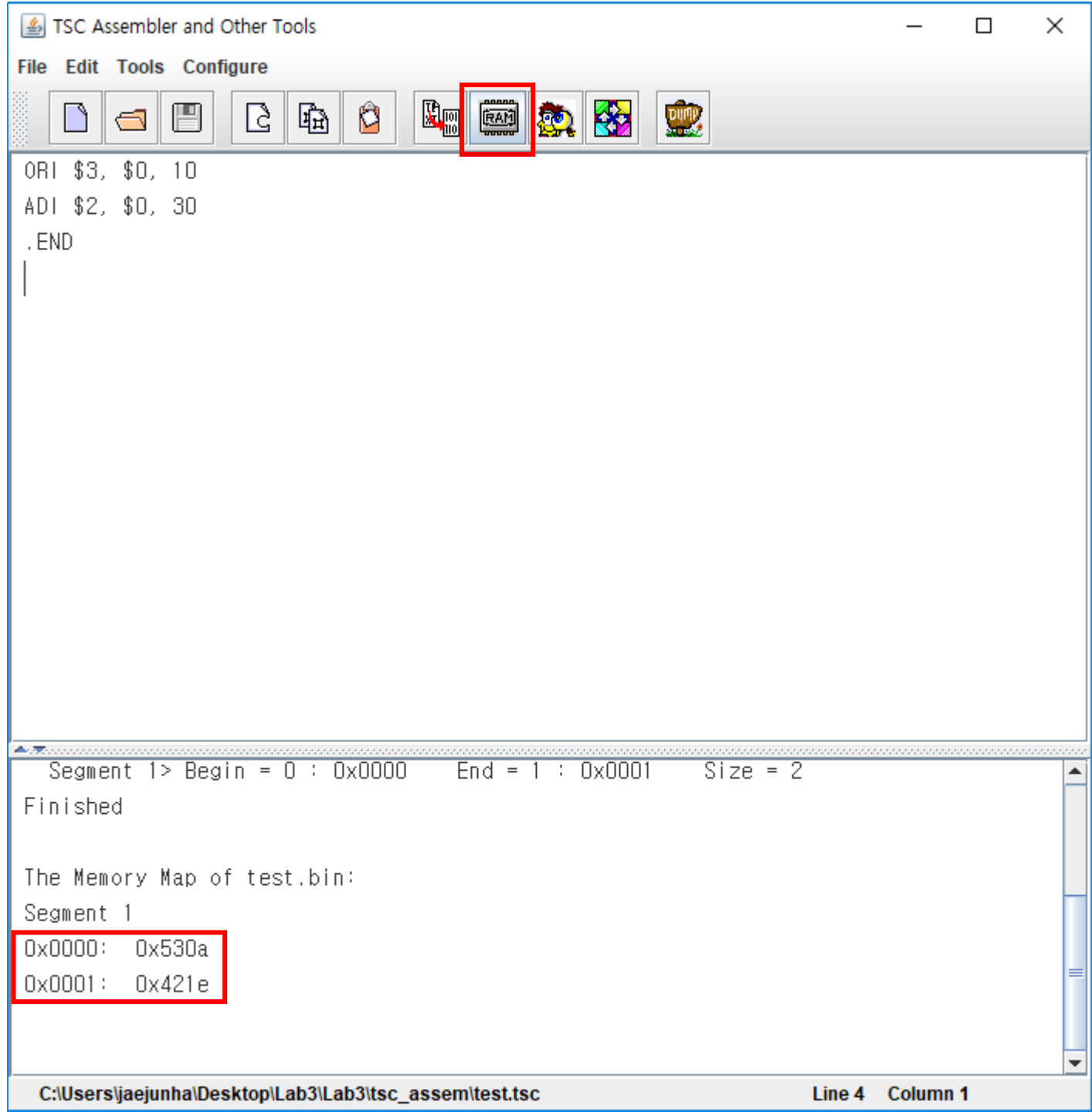
Tips

- TSC assembler



Tips

- TSC assembler

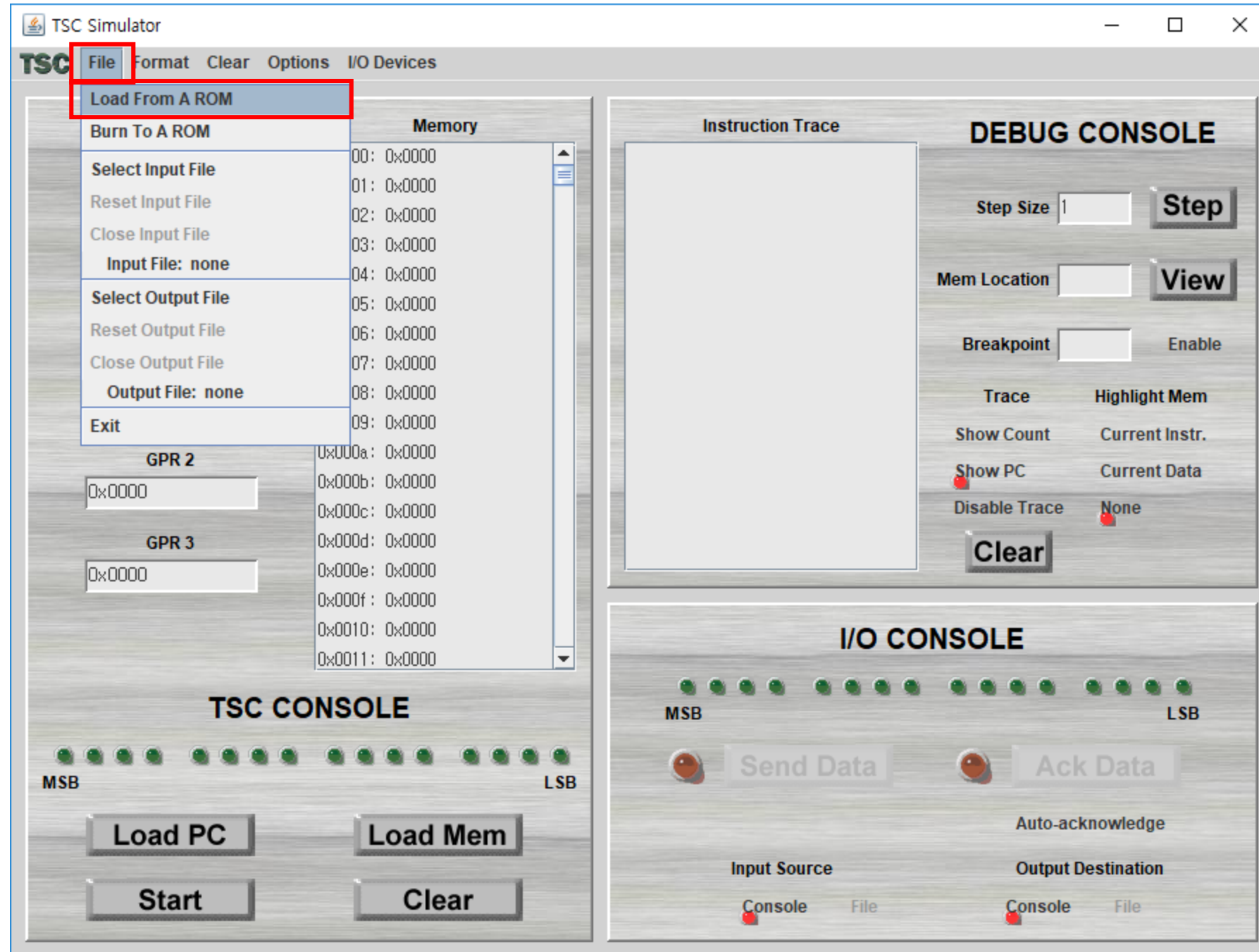


Tips

- TSC simulator
 - `tsc_assem > sim > runit.bat`
 - It needs Java
 - It can check value stored in memory / register

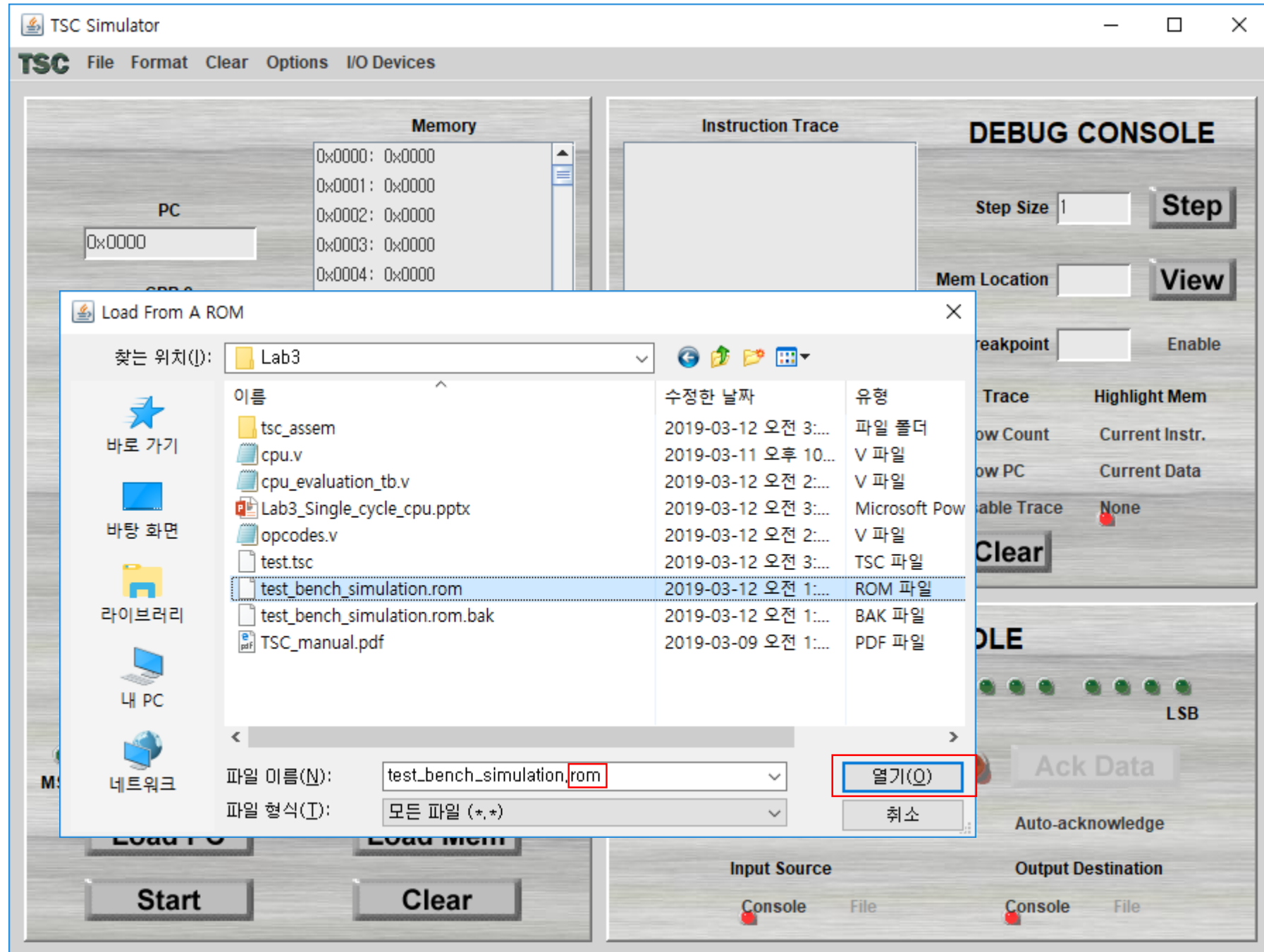
Tips

- TSC simulator



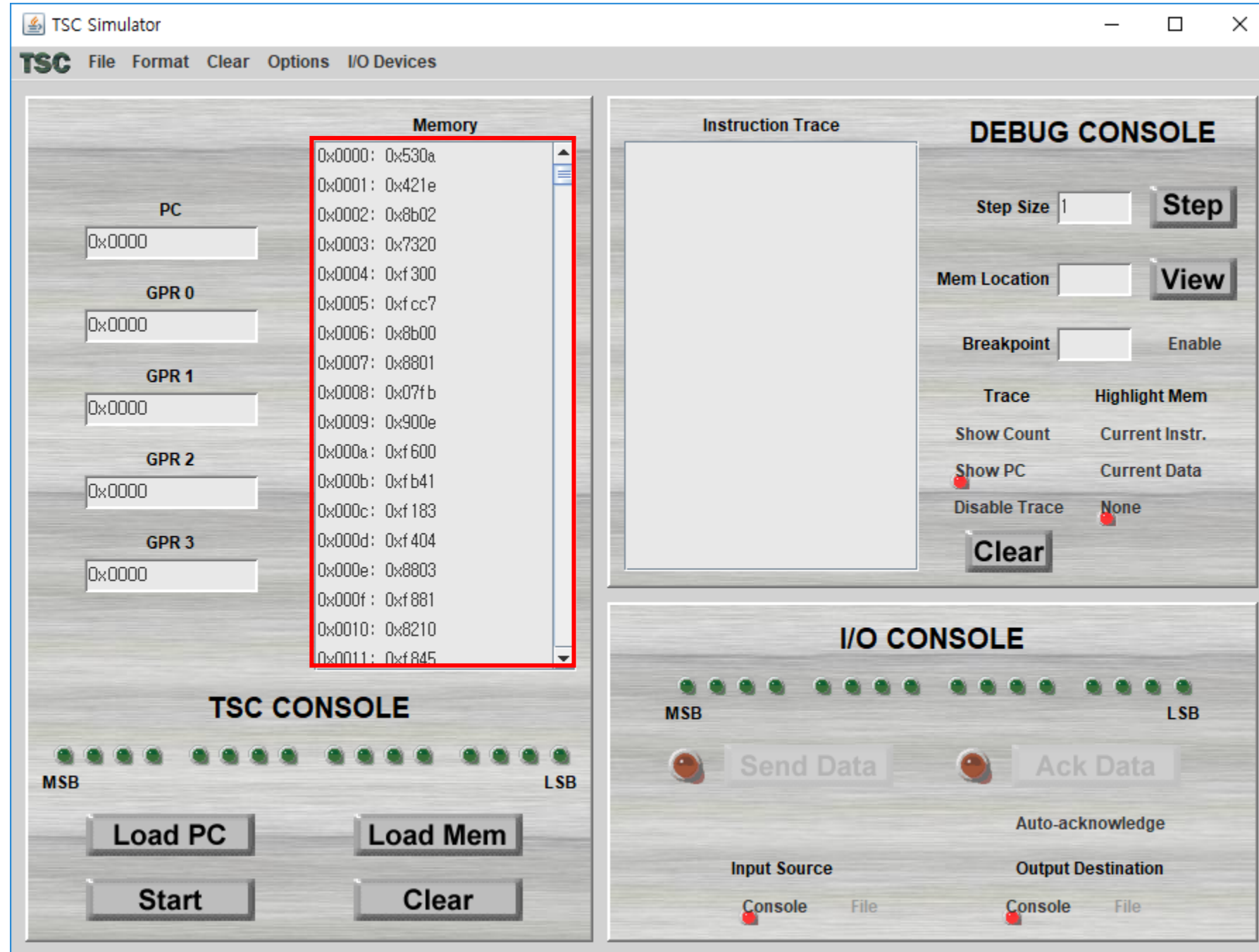
Tips

- TSC simulator



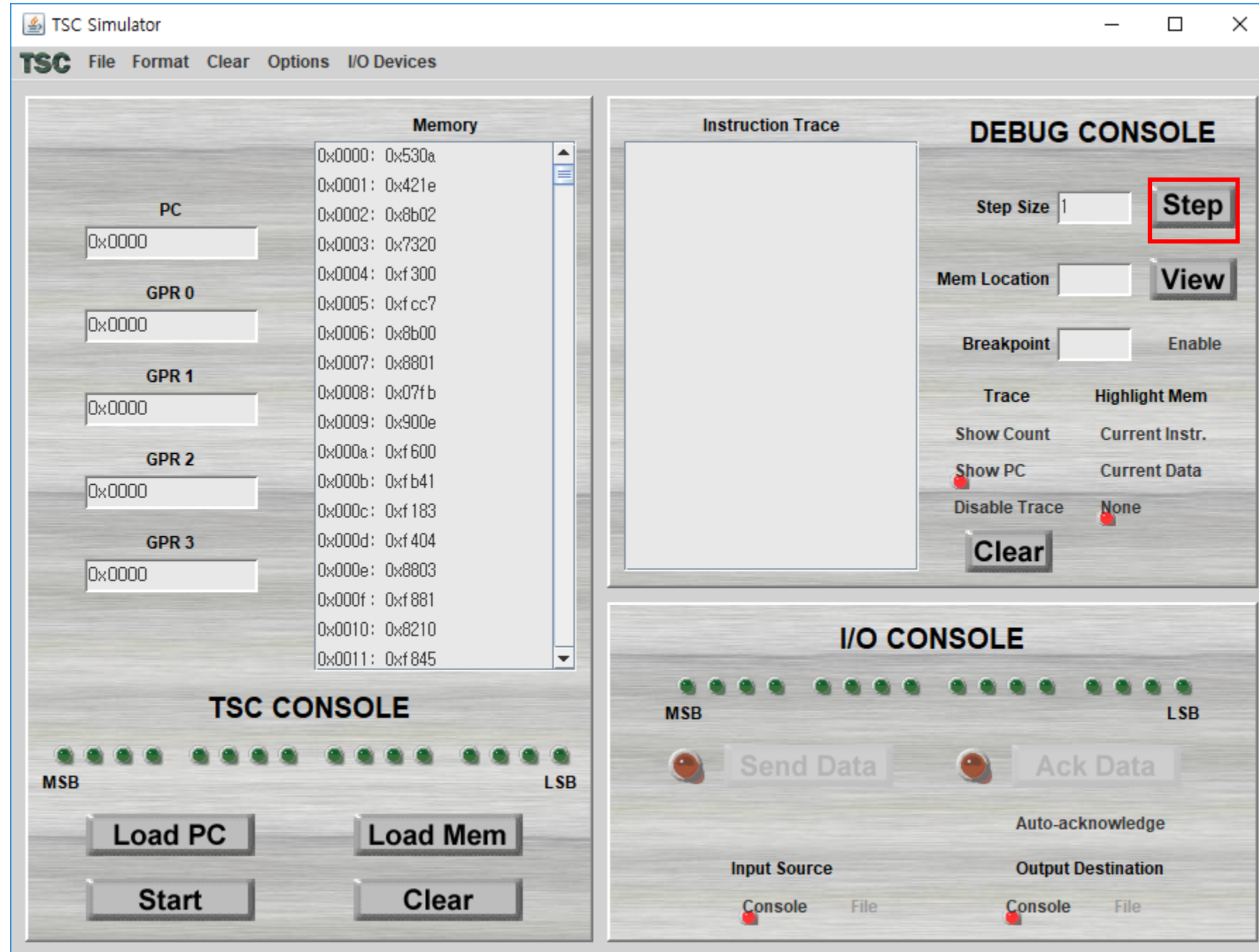
Tips

- TSC simulator



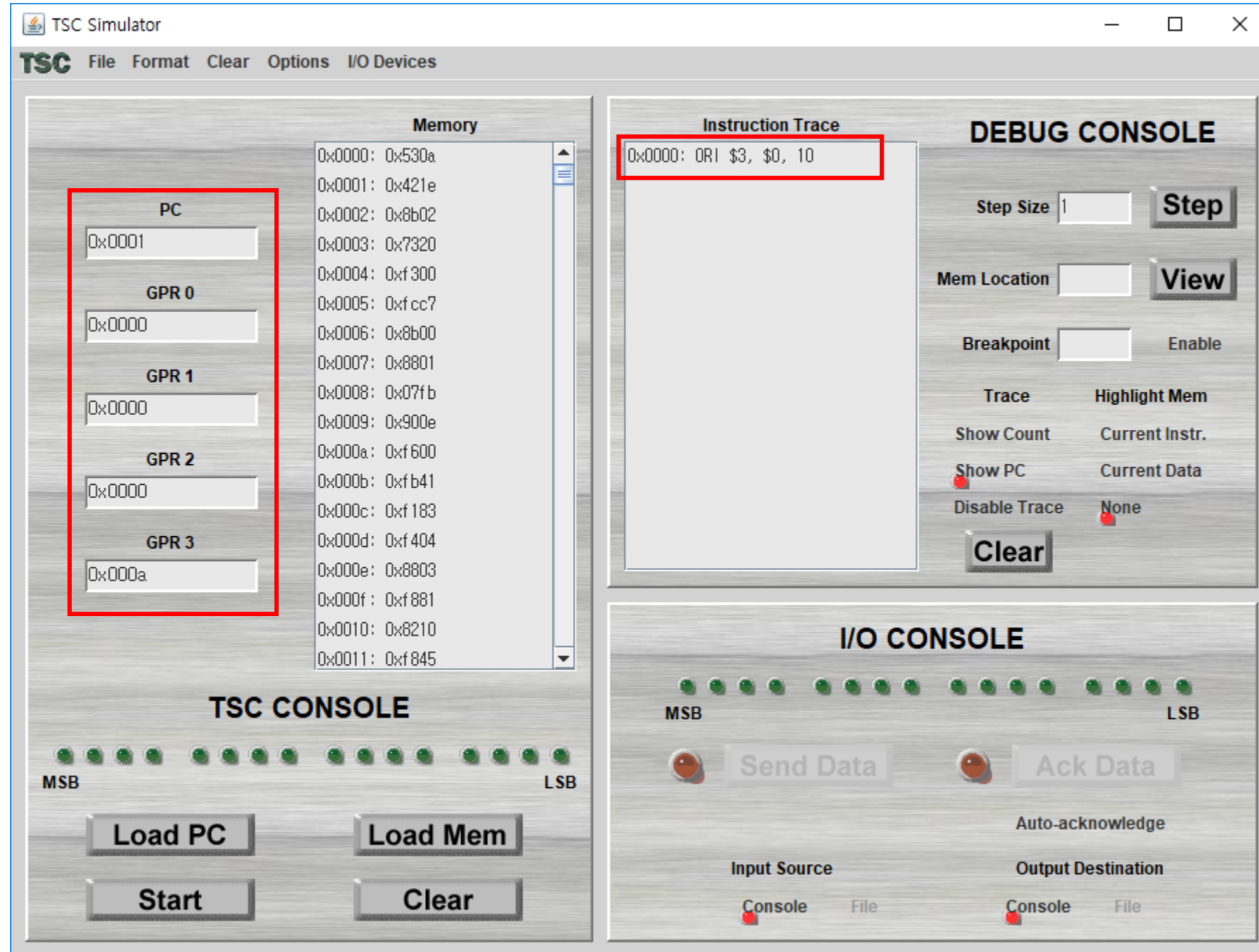
Tips

- TSC simulator



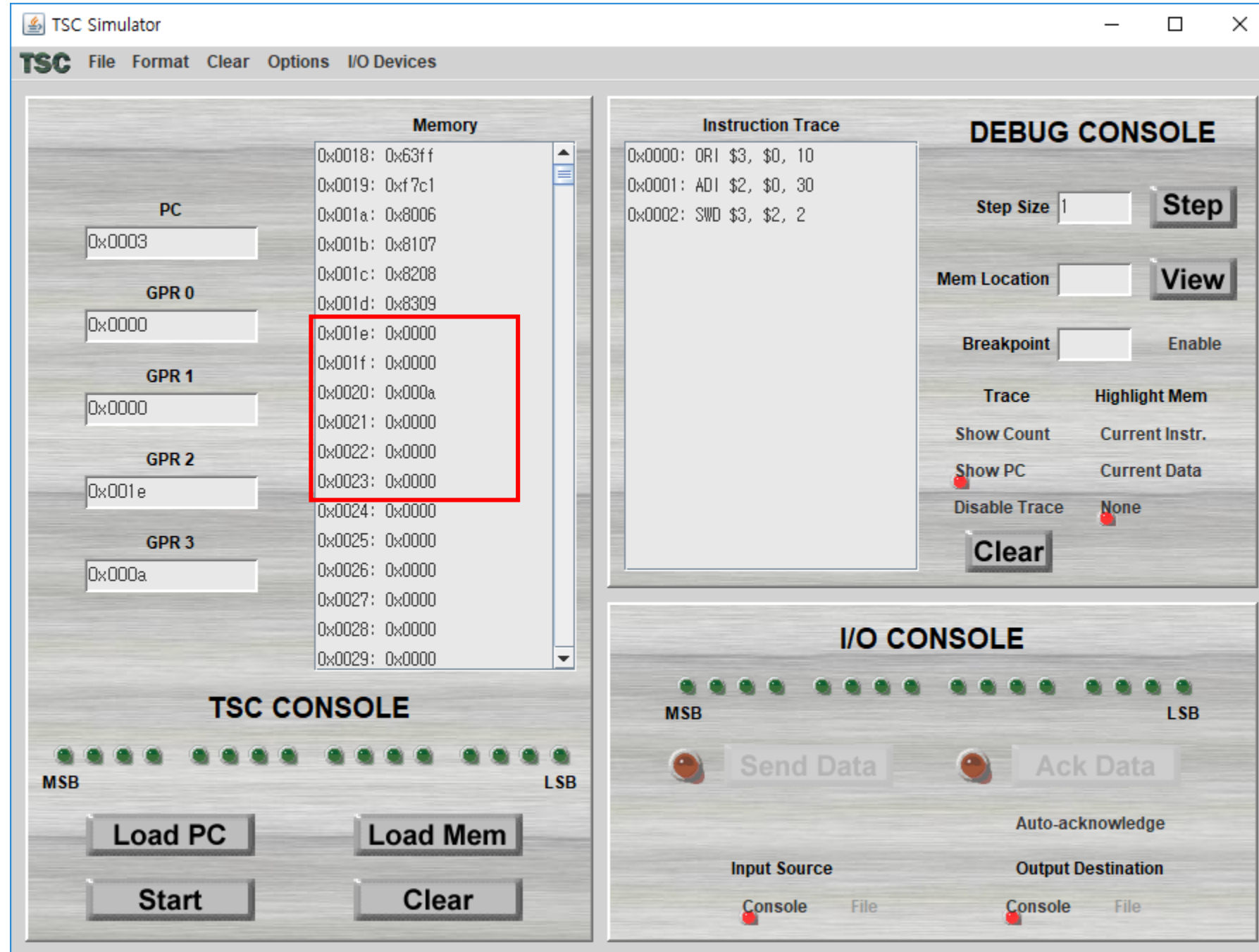
Tips

- TSC simulator



Tips

- TSC simulator



Tips

- TSC_manual.pdf

The screenshot shows a PDF viewer window titled 'TSC_manual.pdf'. The search bar contains the text '웹 주소 검색 또는 입력'. The main content area displays MIPS assembly code examples and the description of the BNE instruction.

```
o SWD $1, $1, 120      ;M[$1 + 120] <-- $1
o SWD $1, $3, -2       ;M[$3 - 2] <-- $1
o SWD $1, $3, 0        ;M[$3] <-- $1
```

BNE

1. Assembler Format
 - o BNE \$rs, \$rt, offset
2. Description
 - o A branch target address is computed from the sum of the address of the instruction after the branch instruction and the 8-bit, sign-extended offset. The contents of \$rs and \$rt are compared. If they are not equal, then the target address is written into the PC and program execution continues with the instruction at the target address. Otherwise, program execution continues with the instruction following the branch.
3. Operation
 - o If \$rs != \$rt then \$pc <-- \$pc + { (offset7)8 ## offset7..0 }
4. Examples
 - o BNE \$0, \$2, 6 ;If \$0 != \$2 then \$pc <-- \$pc + 7

Lab2 Demo