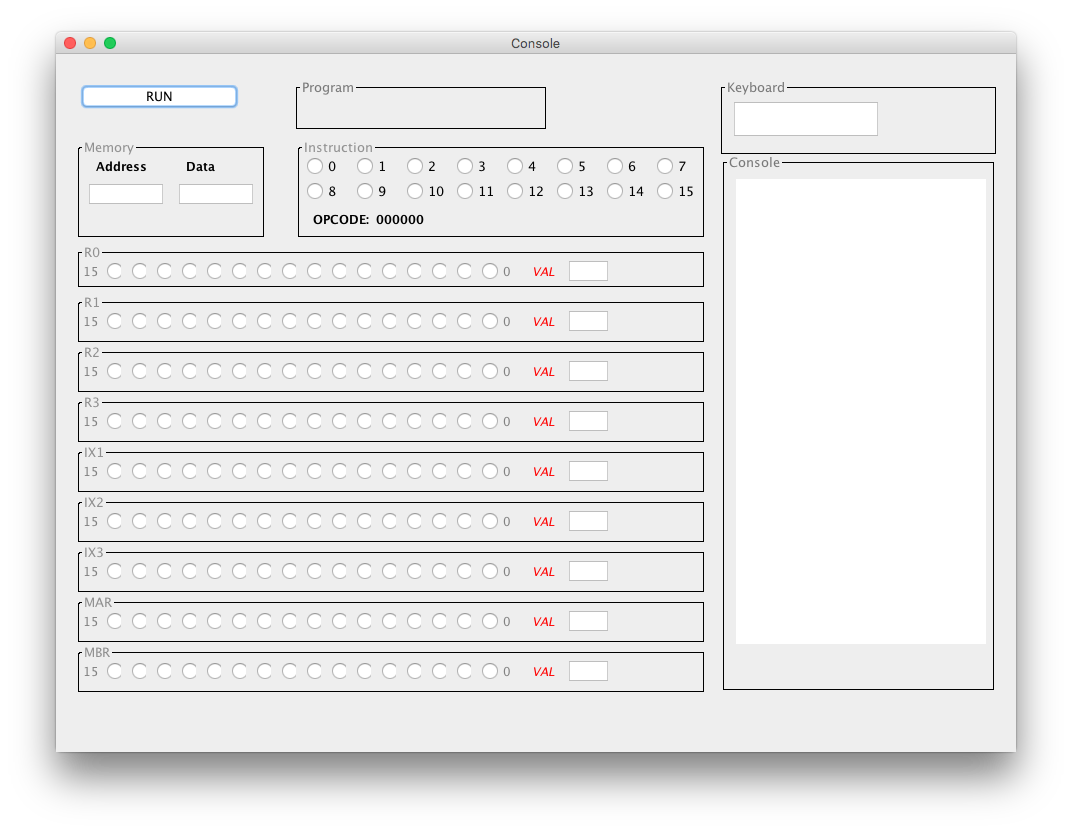
**Computer System Architecture Project – Phase IV**

**Machine Simulation Usage Manual**

**By Priyanka More, Phani Gadepalli, Akshay Thombre, Siddarth Gandhi**

* **Console UI:**

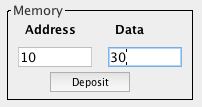


The console UI can be used to:

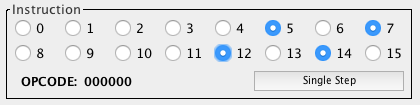
1. Run/halt the simulator.
2. Deposit data into a specific memory location using the Memory panel.
3. Execute a single instruction using the Instruction panel.
4. Deposit data into or display data from the General Purpose Registers, Index Registers or the Memory.
5. Run a given program saved in a txt file “program1.txt” in the same directory as the jar file.

* **Working of the LDR instruction:**

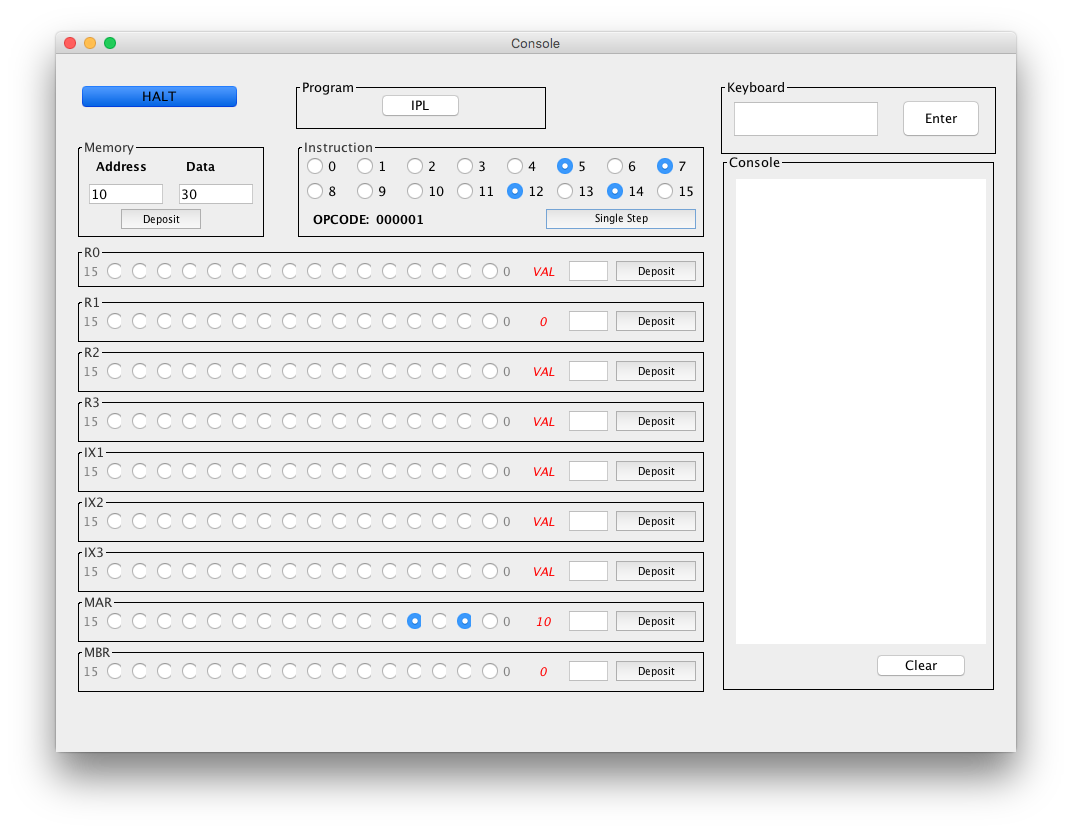
1. Use the memory panel to enter the address and the data that is to be stored in that address location.



1. Select the radio buttons in the instruction panel as follows:
   1. 0-5 for Opcode
   2. 6 and 7 to select General Purpose Register (0-3)
   3. 8 and 9 to select Index Register (1-3)
   4. 10 for Indirect access
   5. 11-15 for memory address



1. Click on the “Single Step” button to execute the instruction.
2. After the instruction is executed, the register and memory values will be updated.



* 1. The selected GPR, register 1 now holds the value “30” which was deposited into the memory earlier.
  2. The MAR shows the memory address where the data is stored.
  3. The MBR shows the data that was loaded into the selected GPR.
* **Depositing Data in a Register:**

/Users/Siddarth/Desktop/Screen Shot 2016-02-06 at 6.27.21 PM.png

1. Enter a value to deposit in the text field.

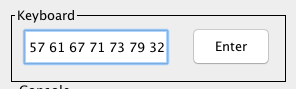
/Users/Siddarth/Desktop/Screen Shot 2016-02-06 at 6.27.45 PM.png

1. Click on deposit to deposit the value in the selected register.

/Users/Siddarth/Desktop/Screen Shot 2016-02-06 at 6.19.43 PM.png

* **Program 1 Execution:**

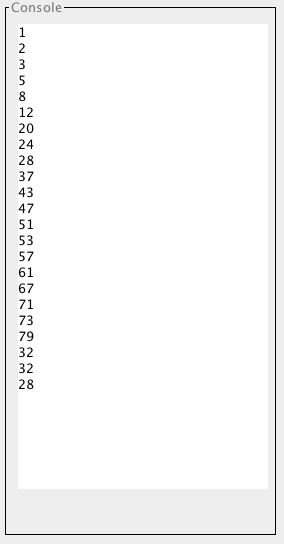
1. Launch and run the simulator.
2. In the keyboard field, enter 20 integers divided by spaces followed by an integer which you want to search in the first given 20 integers.



1. For example, the 20 integers are “1, 2, 3, 5, …., 67, 71, 73, 79” and the value to be searched is “32”.
2. Click on the Program 1 button to execute the instructions from the program1.txt file.

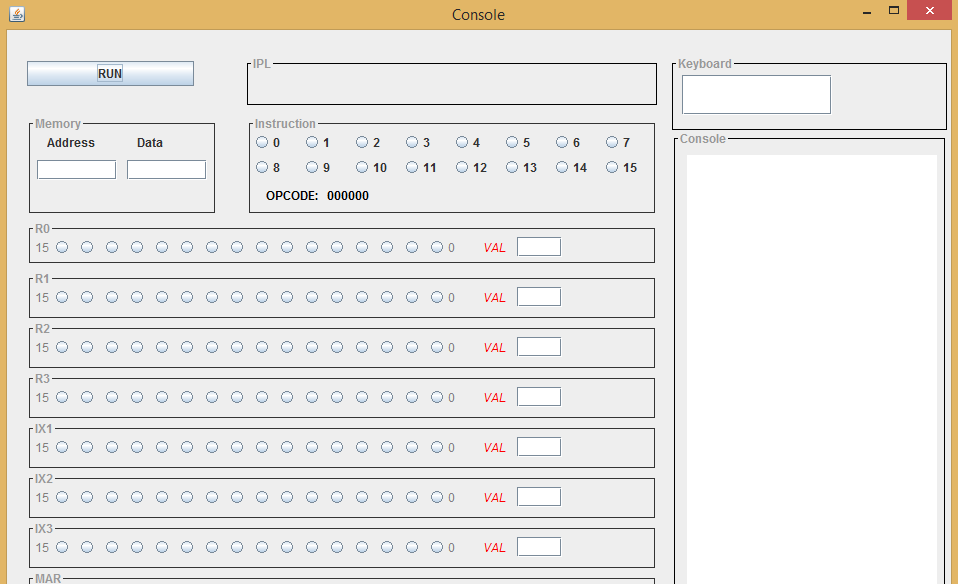


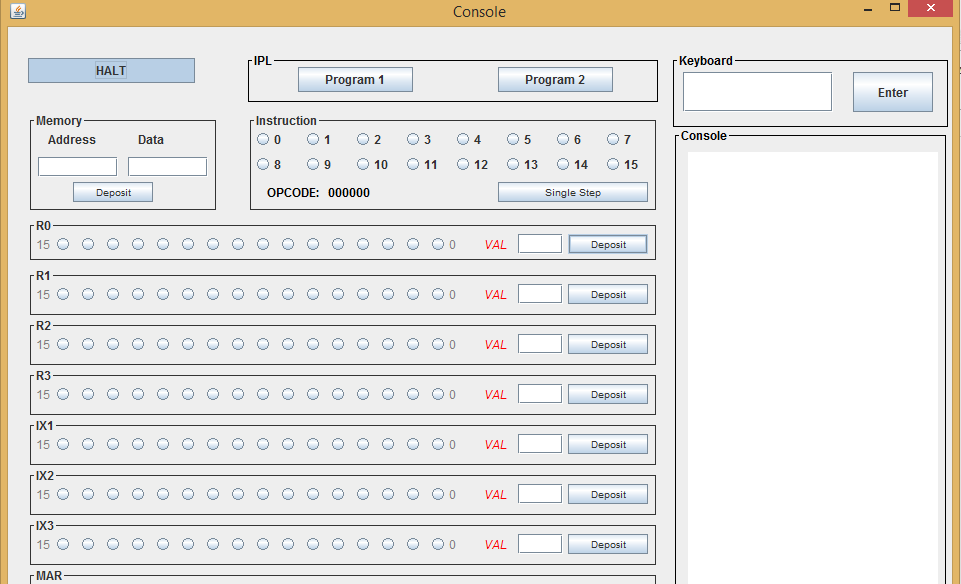
1. The simulator will execute the program and show the entered 20 integers followed by the search result, if found in the simulator console.



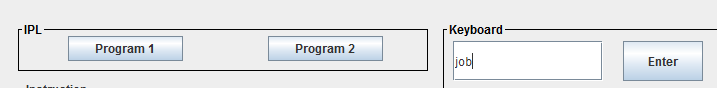
* **Program 2 Execution**

1. Launch and run the simulator.

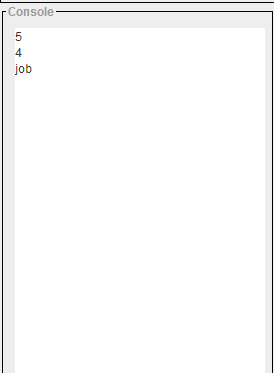




1. In the keyword to be searched in keyboard field, enter any of the following keywords-‘wife’, ‘son’, ‘dad’, ‘mom’, ‘job’.

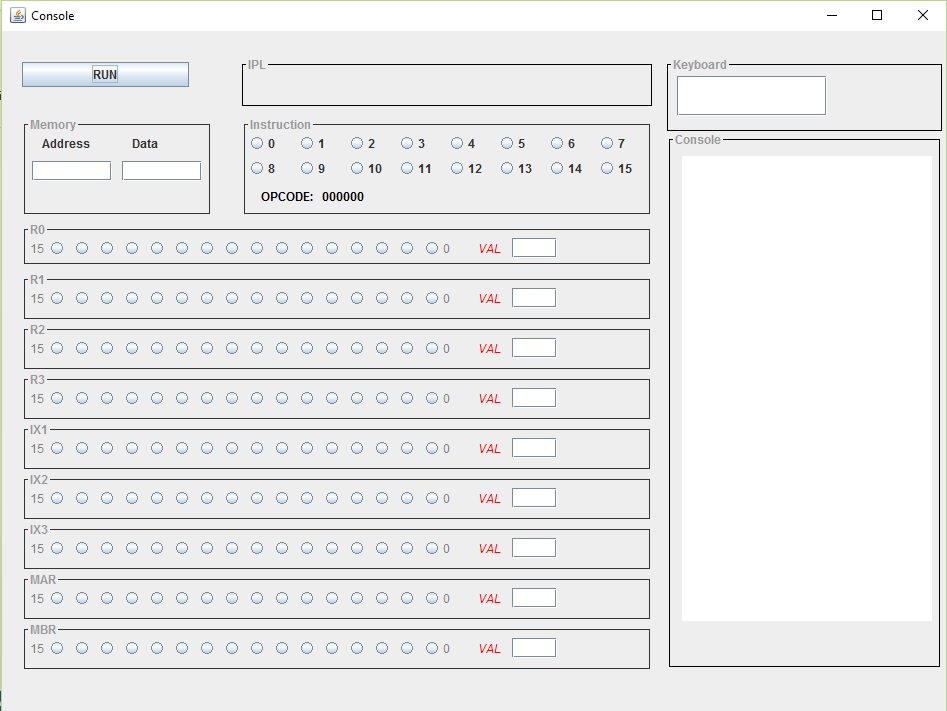


1. Click on the Program2 button to execute the instructions from the program2.txt file.
2. The simulator will execute the program2 and show the number of sentence followed by the position of keyword in sentence, and the actual search keyword entered.



* **Program 3 execution**

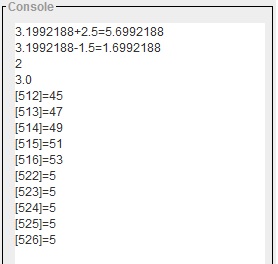
1. Launch and run the simulator.



1. Click on the Program3 button to execute the instructions from the program3.txt file.



1. Program 3 performs Floating point addition/subtraction, Float<->Int(fixed) conversion and Vector integer addition/subtraction.
   1. Float add/sub: Adds 3.2 in register fr0 to 2.5 from mem location. Subs 2.5 in mem location with 3.2 in fr0.
   2. Converts 2.5f to integer(fixed) point and 3 (fixed) to float.
   3. Vector add/sub: Adds between locations [512]..[516] with [517]..[521] and result stored in [512]..[516]. Subs between locations [522]..[526] and [527]..[531] and stores in [522]..[526]. (These locations are automatically filled from Simulator application for convenience.)
2. The simulator will execute the program3 and the floating point and vector operations carried out in the program and output displayed in the console from simulator. (not using IN/OUT instructions).



* **Other Instructions That Can Be Executed Using the Console:**

The above example shows how to execute the LDR instruction on the console. Similarly, the other instructions that can be executed are:

|  |  |  |
| --- | --- | --- |
| **OpCode** | **Instruction** | **Description** |
| 01 | LDR r, x, address[,I] | Load Register From Memory, r = 0-3  r  c(EA)  r <- c(c(EA)), if I bit set |
| 02 | STR r, x, address[,I] | Store Register To Memory, r = 0-3  EA  c(r)  c(c(EA)) <- c(r), if I-bit set |
| 03 | LDA r, x, address[,I] | Load Register with Address, r = 0-3  r  EA  r  c(EA), if I bit set |
| 41 | LDX x, address[,I] | Load Index Register from Memory, x = 1-3  Xx <- c(EA) |
| 42 | STX x, address[,I] | Store Index Register to Memory. X = 1-3  EA <- c(Xx)  C(EA) <- c(Xx), if I-bit set |
| 04 | AMR r, x, address[,I] | Add Memory To Register, r = 0-3  r c(r) + c(EA) |
| 05 | SMR r, x, address[,I] | Subtract Memory From Register, r = 0-3  r c(r) – c(EA) |
| 06 | AIR r, immediate | Add Immediate to Register, r = 0-3  r c(r) + Immediate  Note:  1. if Immediate = 0, does nothing  2. if c(r) = 0, loads r with Immediate  IX and I are ignored in this instruction |
| 07 | SIR r, immediate | Subtract Immediate from Register, r = 0-3  r c(r) - Immediate  Note:  1. if Immediate = 0, does nothing  2. if c(r) = 0, loads r1 with –(Immediate)  IX and I are ignored in this instruction |
| 010 | JZ r, x, address[,I] | Jump If Zero:  If c(r) = 0, then PC  EA or c(EA), if I bit set;  Else PC <- PC+1 |
| 011 | JNE r, x, address[,I] | Jump If Not Equal:  If c(r) != 0, then PC - EA or c(EA) , if I bit set;  Else PC <- PC + 1 |
| 012 | JCC cc, x, address[,I] | Jump If Condition Code  cc replaces r for this instruction  cc takes values 0, 1, 2, 3 as above and specifies the bit in the Condition Code Register to check;  If cc bit = 1, PC  EA or c(EA), if I bit set;  Else PC <- PC + 1 |
| 013 | JMA x, address[,I] | Unconditional Jump To Address  PC <- EA, if I bit not set; PC  c(EA), if I bit set  Note: r is ignored in this instruction |
| 014 | JSR x, address[,I] | Jump and Save Return Address:  R3  PC+1;  PC  EA or PC <- c(EA), if I bit set  R0 should contain pointer to arguments  Argument list should end with –17777 value |
| 015 | RFS Immed | Return From Subroutine w/ return code as Immed portion (optional) stored in the instruction’s address field.  R0  Immed; PC  c(R3)  IX, I fields are ignored. |
| 016 | SOB r, x, address[,I] | Subtract One and Branch. R = 0..3  r  c(r) – 1  If c(r) > 0, PC <- EA; but PC  c(EA), if I bit set;  Else PC <- PC + 1 |
| 017 | JGE r,x, address[,I] | Jump Greater Than or Equal To:  If c(r) >= 0, then PC <- EA or c(EA) , if I bit set;  Else PC <- PC + 1 |
| 020 | MLT rx,ry | Multiply Register by Register  rx, rx+1 <- c(rx) \* c(ry)  rx must be 0 or 2  ry must be 0 or 2  rx contains the high order bits, rx+1 contains the low order bits of the result  Set OVERFLOW flag, if overflow |
| 021 | DVD rx,ry | Divide Register by Register  rx, rx+1 <- c(rx)/ c(ry)  rx must be 0 or 2  rx contains the quotient; rx+1 contains the remainder  ry must be 0 or 2  If c(ry) = 0, set cc(3) to 1 (set DIVZERO flag) |
| 022 | TRR rx, ry | Test the Equality of Register and Register  If c(rx) = c(ry), set cc(4)  1; else, cc(4)  0 |
| 023 | AND rx, ry | Logical And of Register and Register  c(rx)  c(rx) AND c(ry) |
| 024 | ORR rx, ry | Logical Or of Register and Register  c(rx)  c(rx) OR c(ry) |
| 025 | NOT rx | Logical Not of Register To Register  C(rx)  NOT c(rx) |
| 031 | SRC r, count, L/R, A/L | Shift Register by Count  c(r) is shifted left (L/R =1) or right (L/R = 0) either logically (A/L = 1) or arithmetically (A/L = 0)  XX, XXX are ignored  Count = 0…15  If Count = 0, no shift occurs |
| 032 | RRC r, count, L/R, A/L | Rotate Register by Count  c(r) is rotated left (L/R = 1) or right (L/R =0) either logically (A/L =1)  XX, XXX is ignored  Count = 0…15  If Count = 0, no rotate occurs |
| 061 | IN r, devid | Input Character To Register from Device, r = 0..3 |
| 062 | OUT r, devid | Output Character to Device from Register, r = 0..3 |
| 063 | CHK r, devid | Check Device Status to Register, r = 0..3  c(r) <- device status |
| 000 | HLT | Stops the machine. |
| 036->067  (Opcode clashes with opcode for VSUB) | TRAP code | Traps to memory address 0, which contains the address of a table in memory. Stores the PC+1 in memory location 2. The table can have a maximum of 16 entries representing 16 routines for user-specified instructions stored elsewhere in memory. Trap code contains an index into the table, e.g. it takes values 0 – 15. When a TRAP instruction is executed, it goes to the routine whose address is in memory location 0, executes those instructions, and returns to the instruction stored in memory location 2. The PC+1 of the TRAP instruction is stored in memory location 2. |
| 033 | FADD fr, x, address[,I] | Floating Add Memory To Register  c(fr)  c(fr) + c(EA)  c(fr)  c(fr) + c(c(EA)), if I bit set  fr must be 0 or 1.  OVERFLOW may be set |
| 034 | FSUB fr, x, address[,I] | Floating Subtract Memory From Register  c(fr)  c(fr) - c(EA)  c(fr)  c(fr) - c(c(EA)), if I bit set  fr must be 0 or 1  UNDERFLOW may be set |
| 035 | VADD fr, x, address[,I] | Vector Add  fr contains the length of the vectors  c(EA) or c(c(EA)), if I bit set, is address of first vector  c(EA+1) or c(c(EA+1)), if I bit set, is address of the second vector  Let V1 be vector at address; Let V2 be vector at address+1  Then, V1[i] = V1[i]+ V2[i], i = 1, c(fr). |
| 036 | VSUB fr, x, address[,I] | Vector Subtract  fr contains the length of the vectors  c(EA) or c(c(EA)), if I bit set is address of first vector  c(EA+1) or c(c(EA+1)), if I bit set is address of the second vector  Let V1 be vector at address; Let V2 be vector at address+1  Then, V1[i] = V1[i] - V2[i], i = 1, c(fr). |
| 037 | CNVRT r, x, address[,I] | Convert to Fixed/FloatingPoint:  If F = 0, convert c(EA) to a fixed point number and store in r.  If F = 1, convert c(EA) to a floating point number and store in FR0.  The r register contains the value of F before the instruction is executed. |
| 50 | LDFR fr, x, address [,i] | Load Floating Register From Memory, fr = 0..1  fr  c(EA), c(EA+1) \*\*This is confusing. So only implemented one load. c(EA+1) isn't quite clear.  fr <- c(c(EA), c(EA)+1), if I bit set |
| 51 | STFR fr, x, address [,i] | Store Floating Register To Memory, fr = 0..1  EA, EA+1  c(fr) \*\* only implemented one store. EA+1 spec isn't quite clear.  c(EA), c(EA)+1 <- c(fr), if I-bit set |

* **Assumptions:**

1. Vector ADD/SUB – Add and subtract Integer vectors. It is not clear in the specification about the types to add or subtract.

2. LDFR/STFR: Specification isn't clear for the second element, and so ignoring it for now.

* **Limitations:**

1. Bit width validations are not done. Values only up to 16 bit can be added.
2. In/out instructions read/write integers for simplicity or can be made to read/write characters by setting a flag.
3. Floating point OVERFLOW/UNDERFLOW/NAN/INFINTY are not implemented/supported.
4. Extensive testing not done!