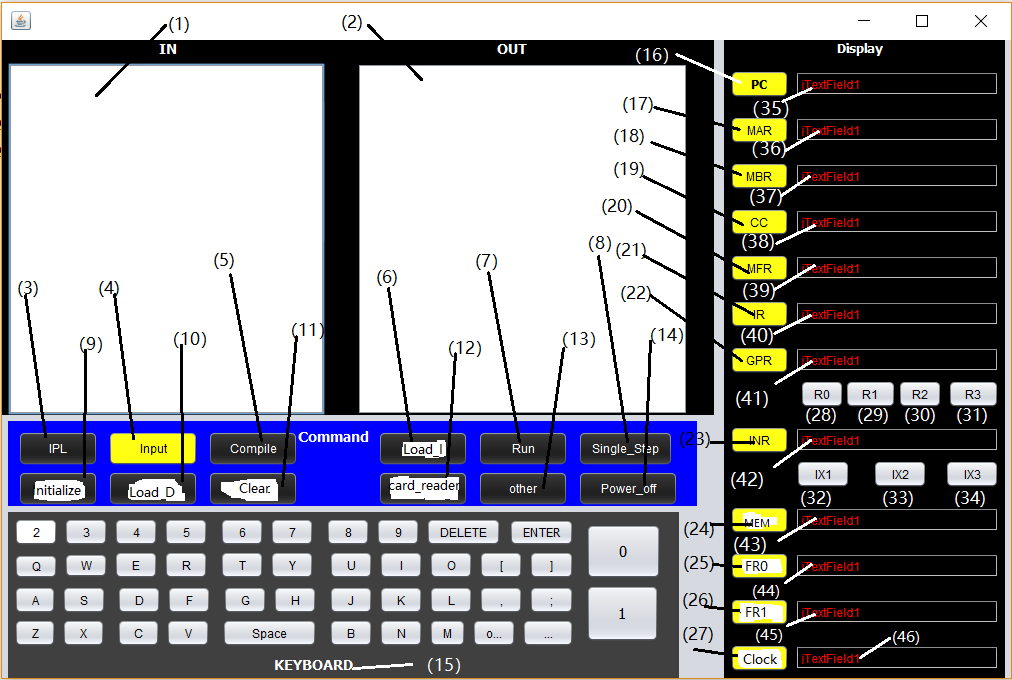
**User guide (Terminator I)**

1. **Features**

Diagram 1 （start page）



Diagram 2 （console）



**1.1 Textbox (console):**

**1.1.1 Input and output text area:**

**(1) Input text area:** Input instructions or data in this place.

**(2)** **output text area:** display the result of compiling the instructions or executing the instructions.

**1.1.2 Display area:**

**(35) PC text field:** Display the contents of Program Counter register or enter data into it.

**(36) MAR text field:** Display the contents of Memory Address Register or enter data into it.

**(37) MBR text field:** Display the contents of Memory Buffer Register or enter data into it.

**(38) CC text field:** Display the contents of Condition Code Register or enter data into it.

**(39) MFR text field:** Display the contents of Machine Fault Register or enter data into it.

**(40) IR text field:** Display the contents of Instruction Register or enter data into it.

**(41) GPR text field:** Display the contents of General Purpose Register or enter data into it.

**(42) INR text field:** Display the contents of Index Register or enter data into it.

**(43) MeM text field:** Enter memory address data, such that user can store data or instructions in the corresponding location of memory.

**(44) FR0 text field:** display the contents of FR0 register or input data into it.

**(45) FR1 text field:** display the contents of FR1 register or input data into it.

**(46) Clock text field:** display the number of cycles that the simulator takes to operate the specific program.

**1.2 Button:**

**1.2.1 Start page:**

**(0) power on button:** Start the Simulator

**1.2.2 Command area (console):**

**(3) IPL Button:** Load the boot program to memory and prepare for the Input instruction

**(4) Input Button:** control the permission of inputting instructions in (1) Input text area.

**(5) Compile Button:** Compile the instructions inputted in the (1) input text area and display the result in the (2) output text area.

**(6) Load instruction button:** Load the Instructions to the memory location, which correspond to the address value displayed in the (43)MeM text field.

**(7) Run button:** run the program （can’t be used now, you can just use single\_step button to execute your instructions.）

**(8) Single Step Button:** single step through the program

**(9) Initialize Button:** Initialize the Simulator

**(10) Load data button:** Load the data displayed in the (1) input text area to memory. The data must be represented as the format of 16-bit binary string

**(11) Clear button:** Clear the contents of (1) input text area and (2) output text area.

(12) **Card\_reader** button: Load a text file to the input text area.

(13) other Button

(14) Power off button : Turn off the simulator.

**1.2.3 Keyboard area (console):**

(**15) Most buttons** correspond to the character indicated by their text.

**“DELETE” button:** Delete the contents that you have inputted.

**“ENTER” button:** Move to a new input line.

**“space” button:** Input a blank character.

**1.2.4 Display area:**

**(16) PC button:** Control the permission of entering data into Program Counter register.

**(17) MAR button:** Control the permission of entering data into Memory Address register.

**(18) MBR button:** Control the permission of entering data into Memory Buffer register.

**(19) CC button:** Control the permission of entering data into Condition Code register.

**(20) MFR button:** Control the permission of entering data into Machine Fault register.

**(21) IR button:** Control the permission of entering data into Instruction register.

**(22) GPR button:** Control the permission of entering data into General Purpose register.

**(23) INR button**: Control the permission of entering data into Index register.

**(24) Memory button:** Control the permission of inputting memory address into (43)MeM text Field.

**(28) R0 button:** Indicate the general purpose register 0.

**(29) R1 button:** Indicate the general purpose register 1.

**(30) R2 button:** Indicate the general purpose register 2.

**(31) R3 button:** Indicate the general purpose register 3.

**(32) IX1 button:** Indicate the Index register 1.

**(33) IX2 button:** Indicate the Index register 2.

**(34) IX3 button:** Indicate the Index register 3.

**(25) FR0 button:** Indicate the float point register FR0.

**(26) FR1 button**: Indicate the float point register FR1.

**(27) Clock button:** Indicate the clock register. (Only appear in the Pipeline Simulator Version).

**2. Operations**

**2.1 Format of Input Data**

PC text field: Decimal value (<2048);

MAR text field: 16-digit numbers consist of “0” and “1”

MBR text field: 16-digit numbers consist of “0” and “1”

CC text field: 4-digit numbers consist of “0” and “1”

MFR text field: 4-digit numbers consist of “0” and “1”

IR text field: 16-digit numbers consist of “0” and “1”

GPR text field: 16-digit numbers consist of “0” and “1”

INR text field: 16-digit numbers consists of “0” and “1”

MeM text field: Decimal value. (<2048)

FR0 text field: decimal value.

FR1 text field: decimal value.

Clock text field: integer value.

**2.2 Enter data into XX (PC, MAR, MBR, CC, MFR, IR, FR0, FR1, Clock) register:**

Steps:

1. Click the XX button to make sure the color of XX button is green, then you can use the keyboard to input data into the PC text field.
2. Click the XX button again to make sure its color turn to yellow, such that the data in the XX text field has been stored in XX register.

**2.3 Enter data into GPR register:**

Steps:

1. Click the YY (R0, R1, R2, R3) button to make sure that the color of YY button is yellow. It means the data in the GPR text field will be stored in YY register.
2. Then click the GPR button to make sure the color of GPR button is green. Use the keyboard to input data into GPR text field.
3. Click the GPR button to make sure its color turn to yellow. such that the data in the GPR text field is stored in YY register.

**2.4 Enter data into INR register:**

Steps:

1. Click the ZZ (IX1, IX2, IX3) button to make sure that the color of ZZ button is yellow. It means the data in the INR text field will be stored in ZZ register.
2. Then click the INR button to make sure the color of INR button is green. Use the keyboard to input data (16-digit numbers consists of “0” and “1”) into INR text field.
3. Click the INR button to make sure its color turn to yellow. such that the data in the INR text field is stored in ZZ register.

**2.5 Enter Address data into Mem Text field:**

Steps:

1. Click the Mem button to make sure its color is green.
2. Use the keyboard to input decimal numbers in the Mem text field.
3. Then click the Mem button again to make sure its color turn into yellow.

**2.6 Load data into Memory:**

Steps:

1. Click the Input button to make sure its color is green. Then use the keyboard to input data into the Input text area. Each line of the data should consist of 16 numbers, which are “0” or “1”.
2. Click the Input button again to make sure its color turn into yellow. **Enter Address data into Mem Text field (2.5)**
3. Then Click the Load\_D button, such that the data in the Input text area are loaded into memory according to the memory address displayed in Mem text field.

**2.7 Load instructions into memory:**

Steps:

1. Click the Initialize button.
2. Click the Input button to make sure its color is green. Use the keyboard to input instructions specified by the instruction set into the Input text area. And each pair of instructions should be divided with “;” character. Click the Input button again to make sure its color turn into yellow.
3. Click compile button to compile the instructions in the input text area. The result of the binary format of the program will be displayed in output text area.
4. **Enter Address data into Mem Text field (2.5)**
5. Click the Load\_I button to load the instructions of binary format, which are displayed in the output text area, to the memory according to the address specified by the value displayed in the MeM text field.

**2.8 Run the instructions stored in memory:**

Steps:

1. **Enter the memory address, in which the instructions are stored, to the PC register (2.2) .**
2. Click Run or Single\_Step button, then simulator will run the program. And click “Display button”, the result will be displayed in the output text area.

**Attention:**

1. **Every time when you want to input some new data or instructions in the Input Text area, you must click the clear button to clear the contents of the Input text area.**
2. **Every time when you click some button and turn its color to green, you must click the button again when you finish inputting data in the text Field.**

**3. Appendix**

**3.1 Instruction Set:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OpCode8** | | **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  Memory(EA)  c(r) | |
| 03 | | LDA r, x, address[,I] | | Load Register with Address, r = 0..3  r  EA | |
| 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  Memory(EA) <- c(Xx) | |
| 000 | | HLT | | Stops the machine. | |
| 010 | JZ r, x, address[,I] | | Jump If Zero:  If c(r) = 0, then PC  EA  Else PC <- PC+1 | |
| 011 | JNE r, x, address[,I] | | Jump If Not Equal:  If c(r) != 0, then PC - EA  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  Else PC <- PC + 1 | |
| 013 | JMA x, address[,I] | | Unconditional Jump To Address  PC <- EA,  Note: r is ignored in this instruction | |
| 014 | JSR x, address[,I] | | Jump and Save Return Address:  R3  PC+1;  PC  EA  R0 should contain pointer to arguments  Argument list should end with –1 (all 1s) 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;  Else PC <- PC + 1 | |
| 017 | JGE r,x, address[,I] | | Jump Greater Than or Equal To:  If c(r) >= 0, then PC <- EA  Else PC <- PC + 1 | |
| 004 | AMR r, x, address[,I] | | Add Memory To Register, r = 0..3  r c(r) + c(EA) | |
| 005 | SMR r, x, address[,I] | | Subtract Memory From Register, r = 0..3  r c(r) – c(EA) | |
| 006 | AIR r, immed | | Add Immediate to Register, r = 0..3  r  c(r) + Immed  Note:  1. if Immed = 0, does nothing  2. if c(r) = 0, loads r with Immed  IX and I are ignored in this instruction | |
| 007 | SIR r, immed | | Subtract Immediate from Register, r = 0..3  r  c(r) - Immed  Note:  1. if Immed = 0, does nothing  2. if c(r) = 0, loads r1 with –(Immed)  IX and I are ignored in this instruction | |

|  |  |  |
| --- | --- | --- |
| 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) |
| OpCode | Instruction | Description |
| 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 | 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)  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)  c(EA), c(EA)+1 <- c(fr), if I-bit set |