The instruction set architecture of a simple computer

(YZU-ISA-V2)

(Memory-mapped I/O)

Mnemonic Code	Opcode	Operands			Assembly Code	Description	Note
	d_1 (4 bits)	**d_2** (4 bits)	**d_3** (4 bits)	**d_4** (4 bits)			
HALT	0				HALT	Terminate program execution	
LOAD	1	R_D	M_S		LOAD R_D M_S	Load the content at memory address M_S to R_D. However, if M_S=254, read a number from keyboard and store it into R_D.	
STORE	2	M_D		R_S	STORE M_D R_S	Store the content of R_S into memory address M_D. However, if M_D=255, print a number stored in R_S on the monitor.	
LOADI	3	R_D	R_S		LOADI R_D R_S	Read data from the memory address specified in R_S and store it into R_D. This instruction should not be used to read data from a keyboard.	
STOREI	4	R_D	R_S		STOREI R_D R_S	Write the data stored in R_S into the memory address specified in R_D. This instruction should not be used to display data on a monitor.	
ADD	5	R_D	R_{S1}	R_{S2}	ADD R_D R_{S1} R_{S2}	$R_D = R_{S1} + R_{S2}$	
ADDI	6	R_D	n		ADDI R_D n	$R_D = R_D + n$ where $-128 \leq n \leq 127$	
SUB	7	R_D	R_{S1}	R_{S2}	SUB R_D R_{S1} R_{S2}	$R_D = R_{S1} - R_{S2}$	
AND	8	R_D	R_{S1}	R_{S2}	AND R_D R_{S1} R_{S2}	$R_D = R_{S1}$ & R_{S2} (bit-wise AND)	
OR	9	R_D	R_{S1}	R_{S2}	OR R_D R_{S1} R_{S2}	$R_D = R_{S1}$ | R_{S2} (bit-wise OR)	
XOR	A	R_D	R_{S1}	R_{S2}	XOR R_D R_{S1} R_{S2}	$R_D = R_{S1} \oplus R_{S2}$ (bit-wise exclusive OR)	

XNOR	B	R_D	R_{S1}	R_{S2}	XNOR R_D R_{S1} R_{S2}	$R_D = R_{S1} \wedge R_{S2}$ (bit-wise exclusive NOR)	
ROTATE	C	R	m	dir	ROTATE R m 1 ROTATE R m 0	Rotate R to the right by m places if dir == 0, otherwise, rotate R to the left by m places where $0 \leq m \leq 15$.	
JUMPGE	D	R	n		JUMPGE R n	Jump to the instruction at address **PC + n** if the content of R is greater than or equal to 0 (i.e., the content of R_0) where -128≤n≤127.	
JUMPGT	E	R	n		JUMPGT R n	Jump to the instruction at address **PC + n** if the content of R is greater than 0 (i.e., the content of R_0) where -128 ≤ n ≤ 127.	
JUMPNE	F	R	n		JUMPNE R n	Jump to the instruction at address **PC + n** if the content of R is not equal to 0 (i.e., the content of R_0) where $-128 \leq n \leq 127$.	
END		This mnemonic code is placed at the last line of a program to indicate the end of a program. This is not an instruction. So, there is no opcode for it.					

-
- This is a memory-mapped I/O computer. The memory address 254 is used for input and 255 is used for output.
- This ISA is revised on October 22, 2018. INC and DEC are removed because they can be replaced by "ADDI R 1" and "ADDI R -1", respectively. "MOVE R_D R_{S1}" is also removed because its task can be done by "ADD R_D R_{S1} R_0". Therefore, "JUMPGT R n", JUMPGE R n", and "SUB R_D R_{S1} R_{S2}" are added. NOT is also replaced by XNOR because it can be done using XNOR R_D R_S R_0.
- R_0 always holds a value of 0. So R_0 cannot be changed.
- PC holds the address of the instruction being executed.
- The two instructions STOREI and LOADI are very important. They allow us to implement an array data structure easily. Without them, array implementation is nearly impossible.

R_S, R_{S1}, R_{S2}: Address of source registers
R_D: Address of destination register
M_S: Address of source memory location
M_D: Address of destination location
d_1, d_2, d_3, d_4: First, second, third, and fourth field of an instruction

An example of a simple program that adds K integers read from the keyboard is shown below. Each line contains only one instruction. A program should contain only instructions, i.e., nothing else. The last line should be END. END is not an instruction. It is simply used here to inform the Assembler that the program comes to an end here.

Inst # Instruction

```
1   LOAD R1 254     // Input the number K from keyboard into R1. K is the number of integers to be added.
2   ADD R2 R0 R0    // Initialize R2 to 0. R2 will be used to store the sum of K integers
3   JUMPNE R1 3     // Check whether R1, i.e., K is reduced to zero. If it is reduced to zero, the program
                    // continues executing Inst #4. If not, go to Inst #6.
4   STORE 255 R2    // Output the sum of K integers.
5   HALT            // Stop the program execution
6   LOAD R3 254     // Input an integer and store it into R3
7   ADD R2 R2 R3        // Do R2=R2+R3. That is, add R3 to the sum.
8   ADDI R1 -1          // Decrease R1 by 1. The number of integers yet to be processed is reduced by 1.
9   JUMPNE R1 -3        // If R1 is not reduced to zero, go to Inst #6. That is PC+n = 9 + (-3)=6.
                        // Otherwise, continue executing Inst #10.
10  STORE 255 R2        // Output the sum.
11  HALT                // Stop program execution
12  END
```

In the above code, “LOAD R1 254” reads a number from keyboard and stores it into R1. Similarly, “STORE 255 R2” display the content of R2 on a monitor.

Below is another example. This example is very important. It shows you how to use “labels” for jump. This is very convenient since you need not count the number of lines to skip for a jump.

```
// This program reads two numbers. Print out the one which is larger.
// If the larger one is 0, the program terminates. Otherwise, repeat reading two numbers.
//
// The following program uses an address label to represent the address of an instruction.
// The address associate with the address label is the address of the first instruction following
// the label. The first instruction that follows an address label may be in the same line as the address
//    label or in different line. For example, START is an address label. it represents the address
// of LOAD R1 254 located at the next line. STOP is also an address label which represents
// the address of STORE 255 R2 in the same line. An address label can be used
//    inside in a JUMPNE instruction to serve as the jump address. For example,
// JUMPNE R3 STOP means if R3 is not zero, jump to the address of STORE 255 R2.
// You can add comments to explain or annotate your code as it can be done in C++.
// Use a pair of "/*" and "*/" to add comments that need more than one line. Or you can use "//"
//    to write comment on a line.
```

```
START:                  // Define an address label that associate with LOAD R1 254.
    LOAD R1 254         // Read the first number from keyboard and place it in R1
    LOAD R2 254             // Read the second number
    SUB   R3 R2 R1          //R3 = R2 – R1
    JUMPGT R3 STOP          // If R1 < R2,    jump to STOP and print out R2
    STORE 255 R1            // Because R1>=R2, print out R1
    JUMPNE R1 START         // Check R1=0? If not equal, jump to START
    HALT                // Because R1=0 and R1 >=R2, program terminates
STOP: STORE 255 R2      // print out R2 because R2 > R1
    JUMPNE R2 START     // Check R2=0? if not equal, jump to START
    HALT            // Because R2=0 and R2>R1, program terminates
    END
```

We can employ an instruction set simulator for YZU-ISA to check whether a program is correctly written. This simulator is called YZU-ISA simulator for short. We can interactively develop a program by keying in instructions one by one using a keyboard or just provide a program from a file. Below is the scenario of running the programs of the first example (filename: PG1.txt) and the second example (filename: PG2.txt) using YZU-ISA simulator. You can try many other capabilities provided by this simulator. For example, we can print out the content of registers and the machine code, set the initial value of PC, etc.

```
Welcome to the Instruction Set Simulator for YZU-ISA-V2.
This instruction set simulator is copyrighted by Rung-Bin Lin, October 30, 2018.

Print out the register content? (Y for yes): N
Print out the machine code? (Y for yes): N
Specify the initial value of program counter: 10
PC is set to a value = 10
Load program from file (Y for yes): Y
Enter file name of the program: PG1.txt
 Continue to execute the code? (Y for continuing): Y
IN: 5
IN: 1
IN: 2
IN: 3
IN: 4
IN: 5
OUT: 15
*** Program terminated ***

Run another program? (Y for yes): Y
Load program from file (Y for yes): Y
Enter file name of the program: PG2.txt
 Continue to execute the code? (Y for continuing): Y
IN: 10
IN: 2
OUT: 10
IN: 13
IN: 10
OUT: 13
IN: -1
IN: 0
OUT: 0
*** Program terminated ***
```

Below is the scenario of interactively developing the program in the first example, i.e., keying in one instruction by one instruction from a keyboard. To run the simulator in this mode, we should not select to load a program from a file. Note that we cannot use labels for denoting instruction addresses with this approach.

```
Welcome to the Instruction Set Simulator for YZU-ISA-V2.
This instruction set simulator is copyrighted by Rung-Bin Lin, October 30, 2018.

Print out the register content? (Y for yes): Y
Print out the machine code? (Y for yes): N
Specify the initial value of program counter: 12
PC is set to a value = 12
Load program from file (Y for yes): N
*** You will enter program code using keyboard. ***
**** Key in your program below. The last line should be a END.
LOAD R1 254
LOAD R2 254
SUB R3 R2 R1
JUMPGT R3 4
STORE 255 R1
JUMPNE R1 -5
HALT
STORE 255 R2
JUMPNE R2 -8
HALT
END
 Continue to execute the code? (Y for continuing): Y
IN: 10
(PC= 12)   LOAD    R1  254
  R0          0
  R1         10
  R2          0
  R3          0
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0

IN: 20
(PC= 13)   LOAD    R2  254
  R0          0
  R1         10
  R2         20
  R3          0
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0
```

```
(PC= 14)   SUB     R3  R2  R1
  R0          0
  R1         10
  R2         20
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0

(PC= 15)   JUMPGT  R3  4
  R0          0
  R1         10
  R2         20
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0

OUT: 20
(PC= 19)   STORE   255  R2
  R0          0
  R1         10
  R2         20
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0
```

→

```
(PC= 20)   JUMPNE  R2  -8
  R0          0
  R1         10
  R2         20
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0

IN: -2
(PC= 12)   LOAD    R1  254
  R0          0
  R1         -2
  R2         20
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0

IN: 0
(PC= 13)   LOAD    R2  254
  R0          0
  R1         -2
  R2          0
  R3         10
  R4          0
  R5          0
  R6          0
  R7          0
  R8          0
  R9          0
  R10         0
  R11         0
  R12         0
  R13         0
  R14         0
  R15         0
```

```
(PC= 14)   SUB     R3  R2  R1
  R0           0
  R1          -2
  R2           0
  R3           2
  R4           0
  R5           0
  R6           0
  R7           0
  R8           0
  R9           0
  R10          0
  R11          0
  R12          0
  R13          0
  R14          0
  R15          0

(PC= 15)   JUMPGT  R3  4
  R0           0
  R1          -2
  R2           0
  R3           2
  R4           0
  R5           0
  R6           0
  R7           0
  R8           0
  R9           0
  R10          0
  R11          0
  R12          0
  R13          0
  R14          0
  R15          0

OUT: 0
(PC= 19)   STORE   255  R2
  R0           0
  R1          -2
  R2           0
  R3           2
  R4           0
  R5           0
  R6           0
  R7           0
  R8           0
  R9           0
  R10          0
  R11          0
  R12          0
  R13          0
  R14          0
  R15          0
```

→

```
(PC= 20)   JUMPNE  R2  -8
  R0           0
  R1          -2
  R2           0
  R3           2
  R4           0
  R5           0
  R6           0
  R7           0
  R8           0
  R9           0
  R10          0
  R11          0
  R12          0
  R13          0
  R14          0
  R15          0

(PC= 21)   HALT
  R0           0
  R1          -2
  R2           0
  R3           2
  R4           0
  R5           0
  R6           0
  R7           0
  R8           0
  R9           0
  R10          0
  R11          0
  R12          0
  R13          0
  R14          0
  R15          0

*** Program terminated ***

Run another program? (Y for yes):
```