# The Instruction Set Architecture of a Simple Computer (YZU-ISA)

Version 2

Rung-Bin Lin

# YZU-ISA

- **YZU uses memory-mapped I/O addressing.**
  - **The memory address space employs 8 bits to denote an address.**
  - **Each word has two bytes.**
  - **The memory address 254 is used for input from a keyboard and 255 is used for output to a monitor.**
- **Each instruction takes 16 bits.**
  - **Opcode takes 4 bits. It has 16 instructions.**
- **There are 16 general registers. Each takes 16 bits.**
  - **These registers are denoted by R0, R1, R2, …, R15.**
  - **R0 always stores a value 0.**

# Instruction Format

| Opcode | Operands | | |
|---|---|---|---|
| **$d_1$ (4 bits)** | $d_2$ (4 bits) | $d_3$ (4 bits) | $d_4$ (4 bits) |

# LOAD

| | | | | |
|---|---|---|---|---|
| LOAD | $R_D$ | $M_S$ | 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 in $R_D$. |

**Example:**

```
LOAD R1 123     // Read the data at address 123 of the memory and store it into R1
LOAD R4 254     // Read the data from a keyboard and store it in R4
```

# STORE

| | | | | |
|---|---|---|---|---|
| STORE | $M_D$ | $R_S$ | STORE $M_D$ $R_S$ | Store the content of $R_S$ into memory at address $M_D$. However, if $M_D$=255, print a number stored in $R_S$ on the monitor. |

**Example:**

```
STORE 123 R1      // Store the data in R1 to address 123 of the memory
STORE 255  R4     // Take the data from R4 and display it on a monitor
```

# LOADI

| | | | | |
|---|---|---|---|---|
| LOADI | $R_D$ | $R_S$ | LOADI $R_D$ $R_S$ | Read data from the memory address specified in $R_S$ and store it in $R_D$. This instruction should not be used to read data from a keyboard. |

**Example:**

**LOADI R2 R1** **// Read data from the memory address specified in R1 and store it into R2.**
**// For example, if R1 now stores a number 123, then the computer will read the**
**// data at address 123 of the memory and store it in R2**

# STOREI

| | | | | |
|---|---|---|---|---|
| STOREI | $R_D$ | $R_S$ | 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. |

**Example:**

**STOREI R2 R1 // Take the data in R1 and write it into the memory address specified in R2.**
**// For example, if R2 now stores a number 123, then the computer will take the**
**// data in R1 and store it at address 123 of the memory.**

# ADD

| | | | | | |
|---|---|---|---|---|---|
| ADD | $R_D$ | $R_{S1}$ | $R_{S2}$ | ADD $R_D$ $R_{S1}$ $R_{S2}$ | $R_D = R_{S1} + R_{S2}$ |

**Example:**

**ADD R3 R2 R1 // ADD the data in R1 and R2 and then put the result into R3**

# ADDI

| | | | | |
|---|---|---|---|---|
| ADDI | $R_D$ | n | ADDI $R_D$ n | $R_D = R_D + n$ where $-128 \leq n \leq 127$ |

**Example:**

**ADDI R3 -20** // ADD -20 to the data in R3 and then store the result back to R3

**ADDI R4 200** // An illegal instruction

# SUB

| | | | | | |
|---|---|---|---|---|---|
| SUB | $R_D$ | $R_{S1}$ | $R_{S2}$ | SUB $R_D$ $R_{S1}$ $R_{S2}$ | $R_D = R_{S1} - R_{S2}$ |

**Example:**

**SUB R3 R2 R1 // Subtract the data in R1 by the data in R2 and then put the result into R3**

# AND

| | | | | | |
|---|---|---|---|---|---|
| AND | $R_D$ | $R_{S1}$ | $R_{S2}$ | AND $R_D$ $R_{S1}$ $R_{S2}$ | $R_D = R_{S1}$ & $R_{S2}$ (bit-wise AND) |

**Example:**

**AND R3 R2 R1 // Do bit-wise AND on the data in R1 and R2 and then put the result into R3**

# OR

| | | | | | |
|---|---|---|---|---|---|
| OR | $R_D$ | $R_{S1}$ | $R_{S2}$ | OR $R_D$ $R_{S1}$ $R_{S2}$ | $R_D = R_{S1} \mid R_{S2}$ (bit-wise OR) |

**Example:**

**OR R3 R2 R1 // Do bit-wise OR on the data in R1 and R2 and then put the result into R3**

# XOR

| | | | | | |
|---|---|---|---|---|---|
| XOR | $R_D$ | $R_{S1}$ | $R_{S2}$ | XOR $R_D$ $R_{S1}$ $R_{S2}$ | $R_D = R_{S1}$ ^ $R_{S2}$ (bit-wise exclusive OR) |

**Example:**

**XOR R3 R2 R1 // Do bit-wise XOR on the data in R1 and R2 and then put the result into R3**

# ROTATE

| | | | | | |
|---|---|---|---|---|---|
| ROTATE | R | m | dir | ROTATE R m 1<br>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 $0 \leq \mathrm{m} \leq 15$. |

**Example:**

**ROTATE R3 8 0 // Rotate the bits in R3 to the right by 8 places**
**ROTATE R3 7 1 // Rotate the bits in R3 to the left by 7 places**

# JUMPGE

| | | | | |
|---|---|---|---|---|
| JUMPEQ | R | n | JUMPEQ R n | Jump to the instruction at address PC + n if the content of R is equal to 0 (i.e., the content of $R_0$) where $-128 \le n \le 127$. |

**Example:**

**JUMPEQ R3 8** **// Jump to the instruction at address PC + 8 if the content of R3 is 0.**

**JUMPEQ R3 -8** **// Jump to the instruction at address PC - 8 if the content of R3 is 0.**

# JUMPGE

| | | | | |
|---|---|---|---|---|
| JUMPGE | 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 $-128 \le n \le 127$. |

**Example:**

```
JUMPGE R3 8     // Jump to the instruction at address PC + 8 if the content of R3 is
                // greater than or equal to 0
JUMPGE R3 -8    // Jump to the instruction at address PC - 8 if the content of R3 is
                // greater than or equal to 0
```

# JUMPGT

| | | | | |
|---|---|---|---|---|
| JUMPGT | R | n | JUMPEQ 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 $-128 \leq n \leq 127$. |

**Example:**

```
JUMPGT R3 8     // Jump to the instruction at address PC + 8 if the content of R3 is
                // greater than 0
JUMPGT R3 -8    // Jump to the instruction at address PC - 8 if the content of R3 is
                // greater than 0
```

# JUMPNE

| | | | | |
|---|---|---|---|---|
| JUMPNE | 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 $-128 \leq n \leq 127$. |

**Example:**

**JUMPNE R3 8** // Jump to the instruction at address PC + 8 if the content of R3 is not equal to 0

**JUMPNE R3 -8** // Jump to the instruction at address PC - 8 if the content of R3 is not equal to 0

# HALT

- **Stop execution of a program. It can appear many times at any places**

# END

- **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.**

# Program : Example 1

- **This program adds K integers read from the keyboard. Each line contains only one instruction. A program should contain only instructions and comments. A comment should be placed after a //. The last line should be END. Note that your program should not include Inst #.**

| 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** |

# Program: Example 2

- **This program reads two numbers from keyboard. Print out the larger one. If the larger one is 0, the program terminates. Otherwise, repeat reading two numbers.**
- **This program uses an address label such as START and STOP to represent the address of an instruction. Note that we can also place the label START and LOAD R1 254 on the same line.**

```
START:                          // Define an address label that associates 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 (that is R3 > 0),  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
```