**Module 4: Introduction to RISC-V ISA Study Guide**

1. Master different types of instructions, including R-, I-, S-, U-, B-, J-type instructions.
2. How to initialize a register? (Refer to Problem 2 in Module 4 assignment 1)
   1. 12-bit constant
      1. s0 = 29

Answer: addi s0, zero, 29

* + 1. s1 = -214

Answer: addi s1, zero, -214

* 1. 32-bit constant
     1. s2 = 0xEEEEEFAB

Answer: lui s2, 0xEEEEF

addi s2, s2, -85

* + 1. s3 = 0xEDCBA123

Answer: lui s3, 0xEDCBA

addi s3, s3, 0x123

1. How to program an IF-ELSE-ENDIF/IF-ENDIF structure in RSIC-V? (Refer to Problem 1 in Module 4 assignment 1)

|  |  |  |
| --- | --- | --- |
| **IF: Branch instruction to goto ELSE**   |  | | --- | | **Then operations** |   **Then:**  **J ENDIF**  **ELSE:**   |  | | --- | | **Else operations** |   **ENDIF:** |

**Example:**

**if (g <= h)**

**g = (g + h)/128;**

**else**

**h = (g – h) \* 12;**

Assume variables **g** and **h** are in registers **t0** and **t1, respectively.**

|  |  |  |
| --- | --- | --- |
| **IF: bgt t0, t1, ELSE**  **Then:**   |  | | --- | | **add t0, t0, t1**  **srai t0, t0, 7** |     **J ENDIF**  **ELSE:**     |  | | --- | | **sub t2, t0, t1**  **slli t3, t2, 3**  **slli t4, t2, 2**  **add t1, t3, t4** |   **ENDIF:** |

1. How to program a loop in RISC-V? (Refer to Problem 1 of Module 4 assignment II)

**For Loop:**

|  |  |  |
| --- | --- | --- |
| **Initialize the loop control variable**  **For: Branch instruction to break the loop by going to Exit**   |  | | --- | | **Iteration operations** |  |  | | --- | | **Update loop control variable** |   **J For**  **Exit:** |

Example:

**for (i = 0; i < size; i = i + 1)**

**temp[i] = temp[i] \* 128;**

Assume: **a0** holds the base address of **temp** and **a1** holds the size of the **temp** array

|  |  |  |
| --- | --- | --- |
| **add t0,zero, zero**  **For: bge t0, a1, Exit**   |  | | --- | | **slli t1, t0, 2**  **add t1, a0, t1**  **lw t2, 0(t1)**  **slli t2, t2, 7**  **sw t2, 0(t1)** |  |  | | --- | | **addi t0, t0, 1** |     **J For**  **Exit:** |

**While Loop:**

|  |  |  |
| --- | --- | --- |
| **Initialize the loop control variable**  **While: Branch instruction to break the loop by going to Exit**   |  | | --- | | **Iteration operations** |  |  | | --- | | **Update loop control variable** |   **J While**  **Exit:** |

Example:

**i = 0;**

**while (i < size){**

**temp[i] = temp[i] \* 128;**

**i = i + 1;**

**}**

|  |  |  |
| --- | --- | --- |
| **add t0,zero, zero**  **While: bge t0, a1, Exit**   |  | | --- | | **slli t1, t0, 2**  **add t1, a0, t1**  **lw t2, 0(t1)**  **slli t2, t2, 7**  **sw t2, 0(t1)** |  |  | | --- | | **addi t0, t0, 1** |     **J While**  **Exit:** |

**Do-While Loop:**

|  |  |  |
| --- | --- | --- |
| **Initialize the loop control variable**  **Do:**   |  | | --- | | **Iteration operations** |  |  | | --- | | **Update loop control variable** |   **While: Branch instruction to stay in the loop by going to Do** |

Example:

**i = 0;**

**do {**

**temp[i] = temp[i] \* 128;**

**i = i + 1;**

**} while (i < size)**

|  |  |  |
| --- | --- | --- |
| **add t0,zero, zero**  **Do:**   |  | | --- | | **slli t1, t0, 2**  **add t1, a0, t1**  **lw t2, 0(t1)**  **slli t2, t2, 7**  **sw t2, 0(t1)** |  |  | | --- | | **addi t0, t0, 1** |     **While: blt t0, a1, Do**  **Exit:** |

1. How to read/write data from memory?

lb rd, imm(rs1): read 1 byte from memory at address rs1 + sign\_extend(imm). Then, sign extend this one byte of data to 32 bits and store in register rd.

lbu rd, imm(rs1): read 1 byte from memory at address rs1 + sign\_extend(imm). Then, zero extend this one byte of data to 32 bits and store in register rd.

lh rd, imm(rs1): read 2 byte from memory at address rs1 + sign\_extend(imm). Then, sign extend these two bytes of data to 32 bits and store in register rd.

lhu rd, imm(rs1): read 2 byte from memory at address rs1 + sign\_extend(imm). Then, zero extend these two bytes of data to 32 bits and store in register rd.

lw rd, imm(rs1): read 4 byte from memory at address rs1 + sign\_extend(imm) and store in register rd.

sb rs2, imm(rs1): write 1 byte rs2[7…0] to memory at address rs1 + sign\_extend(imm).

sh rs2, imm(rs1): write 2 byte rs2[15…0] to memory at address rs1 + sign\_extend(imm).

sw rs2, imm(rs1): write 4 byte rs2 to memory at address rs1 + sign\_extend(imm).

1. How to write RISC-V code with multiple if conditions? (Refer to Problem 4 Module 4 Assignment 1)

# If ( cond1 and cond2 and cond3 and cond4)

# operations

if cond1 fail, goto endif

if cond2 fail, goto endif

if cond3 fail, goto endif

if cond4 fail, goto endif

Do:

Do operations come here

Endif:

# If ( cond1 or cond2 or cond3 or cond4)

# operations

if cond1 true, goto Do

if cond2 true, goto Do

if cond3 true, goto Do

if cond4 fail, goto Endif

Do:

Do operations come here

Endif:

1. How to develop function calls in RISC-V? (Refer to Problems 2 and 3 in Module 4 Assignment II)

FunctionA

1. Backup some registers values if needed. For example, register t0 is needed after function call.
2. Initialize passing arguments
3. Call FunctionB with **jal ra, FunctionB**

FunctionB:

1. Create stack and backup some registers or local variables if needed.
2. FunctionB operations
3. Prepare return values if any
4. Restore register values and feel stack resources if needed
5. Return back to the caller function with **jr ra**

How many bytes are needed for stack of FunctionB?

It depends on how the body of FunctionB. So, complete step 5) first, then continues to step 4), 6), 7) and 8).

**RISC-V RV32I Instructions Subset**

**R-type Instructions**

ALU performs the required operations on two source registers (rs1 and rs2) and stores ALU result back to the destination register rd.

**I-type ALU Instructions**

ALU performs the required operations on the source register rs1 and immediate constant, and stores the ALU result back to the destination register rd.



**Data Transfer**

**Memory Read (I-type Instructions)**

Read memory at address (**Reg[rs1] + offset**) and store to the destination register rd.

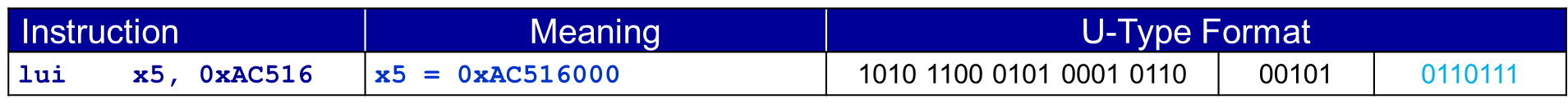
**Memory Write (S-type Instructions)**

Write register content (**Reg[rs2]**) to memory at address (**Reg[rs1] + offset**).



**U-Type Instructions**

**Lui:** load upper immediate instruction which load 20-bit immediate constant from the instruction and store to the upper 20 bits of the destination register **rd**.



**Branch Instructions (B-type and J-type)**

Conditional branch instructions: B-type (if branch condition is true, then go to label; otherwise continue to the next instruction after the branch instruction)

Unconditional branch instructions: J-type (used for function calls and returns, these two instructions are used as a pair.

Calendar

Description automatically generated