

# Chapter 2 Exercise Solutions Guide

## Principles of Computer Systems

Abolfazl Naderi  
avn@disroot.org

November 22, 2025

# 1 Question 1: Endianness

## 1.1 Problem Data

- Register x7 contains: 0xabcd0ef01
- Stored at memory address: 0x10000000

## 1.2 a) Big-Endian at Address 0x10000003

In big-endian format, the most significant byte (MSB) is stored at the lowest address.

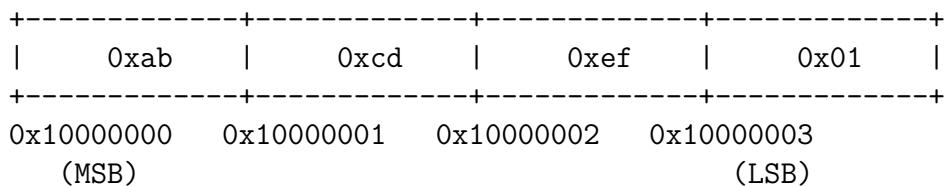
**Memory Layout:**

Address	Byte Value	Description
0x10000000	0xab	MSB (bits [31:24])
0x10000001	0xcd	bits [23:16]
0x10000002	0xef	bits [15:8]
0x10000003	0x01	LSB (bits [7:0])

**Visual Representation:**

Value: 0xabcd0ef01

Big-Endian Storage:



**Answer:** 0x01

### 1.3 b) Little-Endian at Address 0x10000003

In little-endian format, the least significant byte (LSB) is stored at the lowest address.

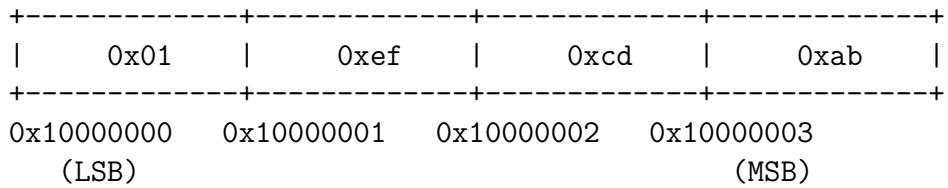
**Memory Layout:**

Address	Byte Value	Description
0x10000000	0x01	LSB (bits [7:0])
0x10000001	0xef	bits [15:8]
0x10000002	0xcd	bits [23:16]
0x10000003	0xab	MSB (bits [31:24])

**Visual Representation:**

Value: 0xabcdef01

**Little-Endian Storage:**



**Answer:** 0xab

## 2 Question 2: Instruction Encoding

### 2.1 Problem Data

Instruction: `sw x15, 40(x19)`

This is an S-type instruction in RISC-V.

### 2.2 S-Type Format

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

### 2.3 Field Values

- **opcode:** 0100011 (store instruction)
- **funct3:** 010 (word, 32-bit)
- **rs1:** x19 = 10011
- **rs2:** x15 = 01111
- **immediate:** 40 = 000000101000
  - imm[11:5] = 0000001
  - imm[4:0] = 01000

### 2.4 Binary Representation

0000001 | 01111 | 10011 | 010 | 01000 | 0100011

### 2.5 Hexadecimal Representation

Grouping into 4-bit chunks:

0000 0010 1111 1001 1010 0100 0010 0011

**Machine Code:** 0x027CA423

### 3 Question 3: Bit Manipulation Operations

#### 3.1 Problem Data

- $x5 = 0x0000ABCD$
- $x6 = 0x12345678$

#### 3.2 a) `slli x7, x6, 1`

Shift left logical immediate: multiply by 2.

**Calculation:**

$$x6 = 0x12345678$$

$$x7 = x6 << 1 = 0x2468ACF0$$

**Answer:**  $x7 = 0x2468ACF0$

#### 3.3 b) Two Instruction Sequence

Instructions:

```
1 slli x7, x5, 4
2 or   x7, x7, x6
```

**Step 1:** `slli x7, x5, 4`

$$x7 = 0x0000ABCD << 4 = 0x000ABCD0$$

**Step 2:** `or x7, x7, x6`

$$x7 = 0x000ABCD0 | 0x12345678$$

Binary OR:

0000 0000 0000 1010 1011 1100 1101 0000
0001 0010 0011 0100 0101 0110 0111 1000
<hr/>
0001 0010 0011 1110 1111 1110 1111 1000

**Answer:**  $x7 = 0x123EFEF8$

### 3.4 c) Two Instruction Sequence

Instructions:

```
1 srl x7, x5, 3  
2 and x7, x7, 0xFEF
```

**Step 1: srl x7, x5, 3**

$$x7 = 0x0000ABCD \gg 3 = 0x0000157A$$

Binary: 0xABCD = 1010 1011 1100 1101

After right shift by 3: 0001 0101 0111 1010 = 0x157A

**Step 2: and x7, x7, 0xFEF**

$$x7 = 0x0000157A \& 0x00000FEF$$

Binary AND:

$$\begin{array}{r} 0001\ 0101\ 0111\ 1010 \\ \& 0000\ 1111\ 1110\ 1111 \\ \hline 0000\ 0101\ 0110\ 1010 \end{array}$$

**Answer: x7 = 0x0000056A**

## 4 Question 4: Bit Field Replacement

### 4.1 Problem

Write a RISC-V program with minimum instructions to replace bits [14:10] of register x6 with bits [28:24] of register x5, keeping all other bits of x6 unchanged.

### 4.2 Solution

Strategy:

1. Extract bits [28:24] from x5
2. Shift them to position [14:10]
3. Clear bits [14:10] in x6
4. OR the results together

**Important Note:** The mask 0xFFFF83FF is too large for an immediate field in I-type instructions (which only support 12-bit immediates). We must use `lui` and `addi` to load it.

### 4.3 RISC-V Code (5 instructions)

```
1 srl x7, x5, 24          # Extract bits [28:24]
2 and x7, x7, 0x1F        # Mask to keep only 5 bits
3 slli x7, x7, 10         # Shift to position [14:10]
4 lui x8, 0xFFFF8          # Load 0xFFFF8000 into x8
5 addi x8, x8, 0x3FF      # x8 = 0xFFFF83FF (mask)
6 and x6, x6, x8          # Clear bits [14:10] in x6
7 or x6, x6, x7           # Insert new bits
```

#### Explanation:

The mask 0xFFFF83FF in binary is:

1111 1111 1111 1111 1000 0011 1111 1111

Breaking it down:

- `lui x8, 0xFFFF8` loads 0xFFFF8000
- `addi x8, x8, 0x3FF` adds 0x3FF, giving 0xFFFF83FF
- This clears bits [14:10] = positions 10, 11, 12, 13, 14

## 5 Question 5: Array Operations

### 5.1 Problem Data

- Array C base address is in register x20
- Elements are 32-bit integers (4 bytes each)
- Result stored in A (assume register x5)

### 5.2 a) $A = C[2] \ll 3$

Calculation:

- Offset for  $C[2] = 2 \times 4 = 8$  bytes

Memory Layout:

Element	Address
C[0]	x20 + 0
C[1]	x20 + 4
C[2]	x20 + 8
C[3]	x20 + 12

RISC-V Code:

```
1 lw x5, 8(x20)          # Load C[2] into x5
2 slli x5, x5, 3         # Shift left by 3 (multiply by 8)
```

### 5.3 b) $A = C[2] \ll n$

Assume n is in register x6 (user input).

RISC-V Code:

```
1 lw x5, 8(x20)          # Load C[2] into x5
2 sll x5, x5, x6         # Shift left by n
```

## 6 Question 6: Even/Odd Number Sorting

### 6.1 Problem

Read two integers. If both have the same parity (both even or both odd), print in ascending order. Otherwise, print the odd number first, then the even number.

### 6.2 RISC-V Assembly Code

```
1      # Read first number
2      li a7, 5                      # syscall: read integer
3      ecall
4      mv x5, a0                     # Store in x5
5
6      # Read second number
7      li a7, 5
8      ecall
9      mv x6, a0                     # Store in x6
10
11     # Check parity of x5
12     andi x7, x5, 1                # x7 = 1 if x5 is odd, 0 if even
13
14     # Check parity of x6
15     andi x8, x6, 1                # x8 = 1 if x6 is odd, 0 if even
16
17     # Compare parities
18     xor x9, x7, x8                # x9 = 0 if same parity
19
20     beqz x9, same_parity        # Branch if same parity
21
22 different_parity:
23     # One is odd, one is even - print odd first
24     bnez x7, print_x5_x6        # If x5 is odd
25
26     # x6 is odd, x5 is even
27     mv a0, x6
28     li a7, 1
29     ecall
30     li a0, '\n'
31     li a7, 11
32     ecall
33     mv a0, x5
34     li a7, 1
35     ecall
36     j end
37
38 print_x5_x6:
39     # x5 is odd, x6 is even
40     mv a0, x5
41     li a7, 1
42     ecall
```

```

43     li a0, '\n'
44     li a7, 11
45     ecall
46     mv a0, x6
47     li a7, 1
48     ecall
49     j end
50
51 same_parity:
52     # Print in ascending order
53     blt x5, x6, print_x5_first
54
55     # x6 <= x5
56     mv a0, x6
57     li a7, 1
58     ecall
59     li a0, '\n'
60     li a7, 11
61     ecall
62     mv a0, x5
63     li a7, 1
64     ecall
65     j end
66
67 print_x5_first:
68     mv a0, x5
69     li a7, 1
70     ecall
71     li a0, '\n'
72     li a7, 11
73     ecall
74     mv a0, x6
75     li a7, 1
76     ecall
77
78 end:
79     li a7, 10          # syscall: exit
80     ecall

```

## 7 Question 7: Digit Count Classification

### 7.1 Problem

Read  $n$  integers and count how many are 1-digit, 2-digit, and 3-or-more-digit numbers.

### 7.2 RISC-V Assembly Code

```
1      # Read count
2      li a7, 5
3      ecall
4      mv x5, a0                      # x5 = n (loop counter)
5
6      # Initialize counters
7      li x10, 0                      # 1-digit count
8      li x11, 0                      # 2-digit count
9      li x12, 0                      # 3+ digit count
10
11     loop:
12     beqz x5, print_results # Exit loop when counter is 0
13
14     # Read number
15     li a7, 5
16     ecall
17     mv x6, a0
18
19     # Get absolute value
20     bgez x6, classify
21     sub x6, x0, x6                # Negate if negative
22
23     classify:
24     li x7, 10
25     blt x6, x7, one_digit      # If x6 < 10
26
27     li x7, 100
28     blt x6, x7, two_digits    # If x6 < 100
29
30     # Three or more digits
31     addi x12, x12, 1
32     addi x5, x5, -1            # Decrement counter
33     j loop
34
35     one_digit:
36     addi x10, x10, 1
37     addi x5, x5, -1            # Decrement counter
38     j loop
39
40     two_digits:
41     addi x11, x11, 1
42     addi x5, x5, -1            # Decrement counter
43     j loop
```

```

44
45 print_results:
46     # Print 1-digit count
47     mv a0, x10
48     li a7, 1
49     ecall
50     li a0, '\n'
51     li a7, 11
52     ecall
53
54     # Print 2-digit count
55     mv a0, x11
56     li a7, 1
57     ecall
58     li a0, '\n'
59     li a7, 11
60     ecall
61
62     # Print 3+ digit count
63     mv a0, x12
64     li a7, 1
65     ecall
66
67     # Exit
68     li a7, 10
69     ecall

```

### Optimization Notes:

- Removed the `next` label for efficiency
- Each classification path decrements the counter and jumps directly to `loop`
- Reduced total jumps from 2 per iteration to 1 per iteration

## 8 Question 8: Date-Time Storage in 32-bit Register

### 8.1 Problem

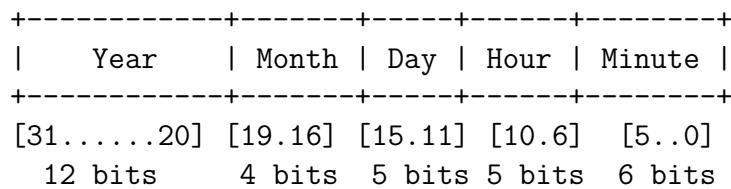
Store date and time (year, month, day, hour, minute) in a single 32-bit register.

### 8.2 Bit Field Allocation

Field	Bits	Size	Range
Minute	[5:0]	6 bits	0-63 (need 0-59)
Hour	[10:6]	5 bits	0-31 (need 0-23)
Day	[15:11]	5 bits	0-31 (need 1-31)
Month	[19:16]	4 bits	0-15 (need 1-12)
Year	[31:20]	12 bits	0-4095

**Visual Representation:**

32-bit Register Layout:



### 8.3 a) Increment One Minute Function

**Function Logic:**

1. Increment minute
2. If minute = 60, set to 0 and increment hour
3. If hour = 24, set to 0 and increment day
4. If day exceeds month's days, set to 1 and increment month
5. If month = 13, set to 1 and increment year

## RISC-V Code:

```
1 increment_minute:
2     # Extract minute [5:0]
3     andi t0, a0, 0x3F
4     addi t0, t0, 1
5
6     li t1, 60
7     blt t0, t1, update_minute
8
9     # Minute overflow
10    li t0, 0
11    srlt t1, a0, 6
12    andi t1, t1, 0x1F      # Extract hour
13    addi t1, t1, 1
14
15    li t2, 24
16    blt t1, t2, update_time
17
18    # Hour overflow
19    li t1, 0
20    srlt t2, a0, 11
21    andi t2, t2, 0x1F      # Extract day
22    addi t2, t2, 1
23
24    # Check day overflow (simplified: assume 31 days)
25    li t3, 32
26    blt t2, t3, update_time
27
28    # Day overflow
29    li t2, 1
30    srlt t3, a0, 16
31    andi t3, t3, 0xF      # Extract month
32    addi t3, t3, 1
33
34    li t4, 13
35    blt t3, t4, update_date
36
37    # Month overflow
38    li t3, 1
39    srlt t4, a0, 20      # Extract year
40    addi t4, t4, 1
41    slli t4, t4, 20
42    slli t3, t3, 16
43    or a0, t4, t3
44    slli t2, t2, 11
45    or a0, a0, t2
46    slli t1, t1, 6
47    or a0, a0, t1
48    or a0, a0, t0
49    ret
50
```

```
51 update_date:
52     li t5, 0xFFFF0000
53     and a0, a0, t5          # Clear date/time
54     slli t3, t3, 16
55     or a0, a0, t3
56
57 update_time:
58     li t5, 0xFFFF0000
59     and a0, a0, t5
60     slli t2, t2, 11
61     or a0, a0, t2
62     slli t1, t1, 6
63     or a0, a0, t1
64
65 update_minute:
66     li t5, 0xFFFFFFF0
67     and a0, a0, t5
68     or a0, a0, t0
69     ret
```

## 8.4 b) Display Date-Time Function

Display format: YYYY/MM/DD, HH:MM

RISC-V Code:

```
1  display_datetime:
2      mv s0, a0                      # Save packed date-time
3
4      # Extract and print year [31:20]
5      srl a0, s0, 20
6      li a7, 1                        # Print integer
7      ecall
8
9      li a0, '/'
10     li a7, 11                      # Print character
11     ecall
12
13     # Extract and print month [19:16]
14     srl a0, s0, 16
15     andi a0, a0, 0xF
16     li a7, 1
17     ecall
18
19     li a0, '/'
20     li a7, 11
21     ecall
22
23     # Extract and print day [15:11]
24     srl a0, s0, 11
25     andi a0, a0, 0x1F
26     li a7, 1
27     ecall
28
29     # Print comma and space
30     li a0, ','
31     li a7, 11
32     ecall
33     li a0, ' '
34     ecall
35
36     # Extract and print hour [10:6]
37     srl a0, s0, 6
38     andi a0, a0, 0x1F
39     li a7, 1
40     ecall
41
42     li a0, ':'
43     li a7, 11
44     ecall
45
46     # Extract and print minute [5:0]
47     mv a0, s0
```

```
48      andi a0, a0, 0x3F
49      li a7, 1
50      ecall
51
52      ret
```