Discussion 4 RISC-V

- caodq@shanghaitech.edu.cn

Agenda

1. Big Endian vs. Little Endian
2. Label and Assembler Directives
3. Enviroment calls
4. RISC-V Practices
5. Venus Q&A

Big Endian vs. Little Endian

Big Endian vs. Little Endian

Big-endian and little-endian from Jonathan Swift's *Gulliver's Travels*

- **The order in which BYTES are stored in memory**
- **Bits always stored as usual. (E.g., 0xC2=0b 1100 0010)**

Consider the number 1025 as we normally write it:

BYTE3	BYTE2	BYTE1	BYTE0
00000000	00000000	00000100	00000001

Big Endian

ADDR3	ADDR2	ADDR1	ADDR0
BYTE0	BYTE1	BYTE2	BYTE3
00000001	00000100	00000000	00000000

Little Endian

ADDR3	ADDR2	ADDR1	ADDR0
BYTE3	BYTE2	BYTE1	BYTE0
00000000	00000000	00000100	00000001

(E.g., 1025 = 0x401 = 0b 0100 0000 0001)

ADDR3	ADDR2	ADDR1	ADDR0
BYTE3	BYTE2	BYTE1	BYTE0
00000000	00000000	00000100	00000001

- Hexadecimal number:
  0xFD34AB88 ($4,248,087,432_{ten}$) =>
  - Byte 0: 0x88 (136_{ten})
  - Byte 1: 0xAB (171_{ten})
  - Byte 2: 0x34 (52_{ten})
  - Byte 3: 0xFD (253_{ten})

Address:	64 address of word (e.g. int)			
Address:	67	66	65	64
Data:	0xFD	0x34	0xAB	0x88

- Little Endian: Starts with the little end of a word:
  - It starts with the smallest (least significant) Byte

Little Endian
Least significant byte in a word
(numbers are addresses) ↓

...	...	...	...
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

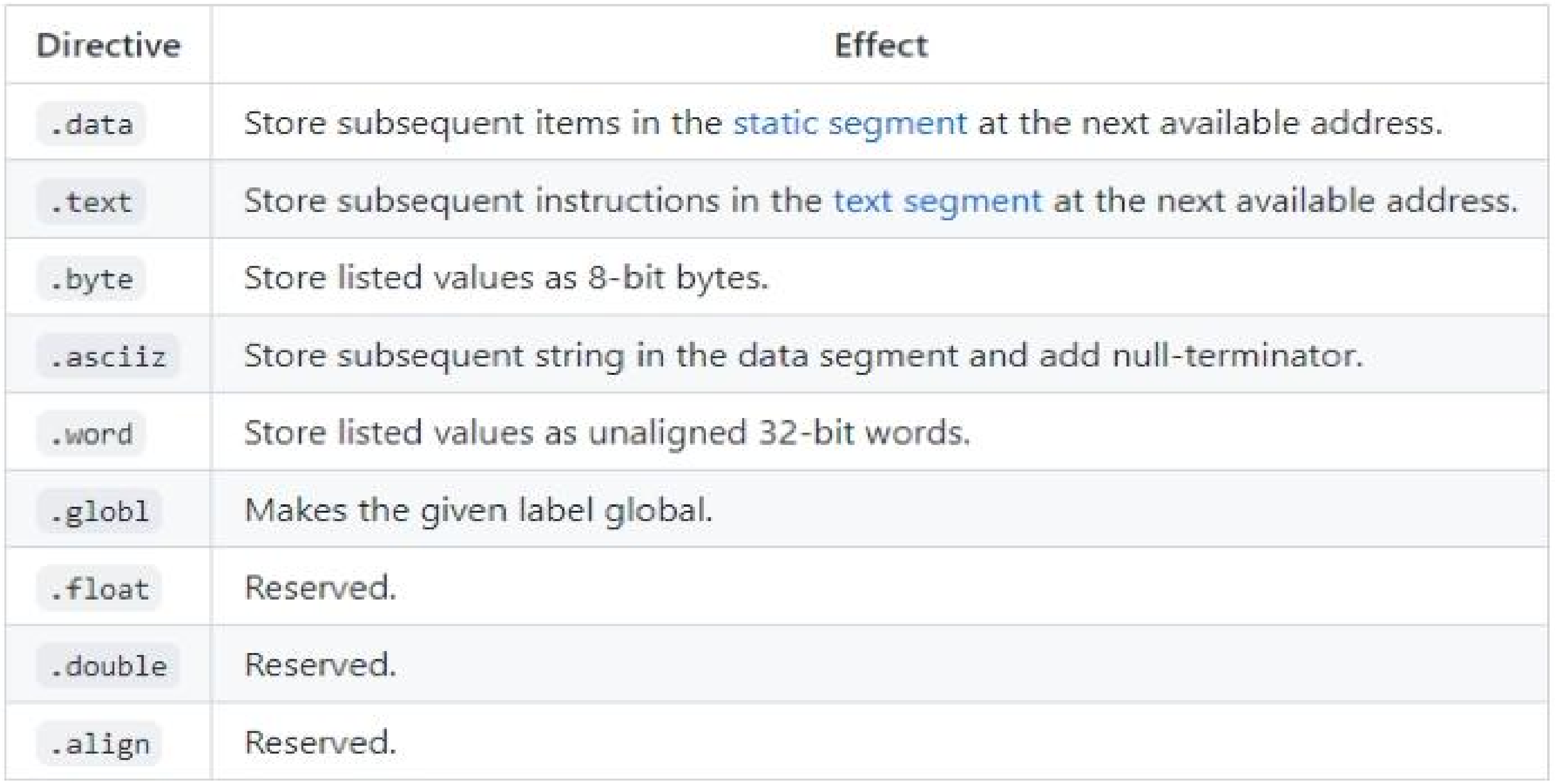
bit: 31 24 23 16 15 8 7 0

立志成才报国裕民

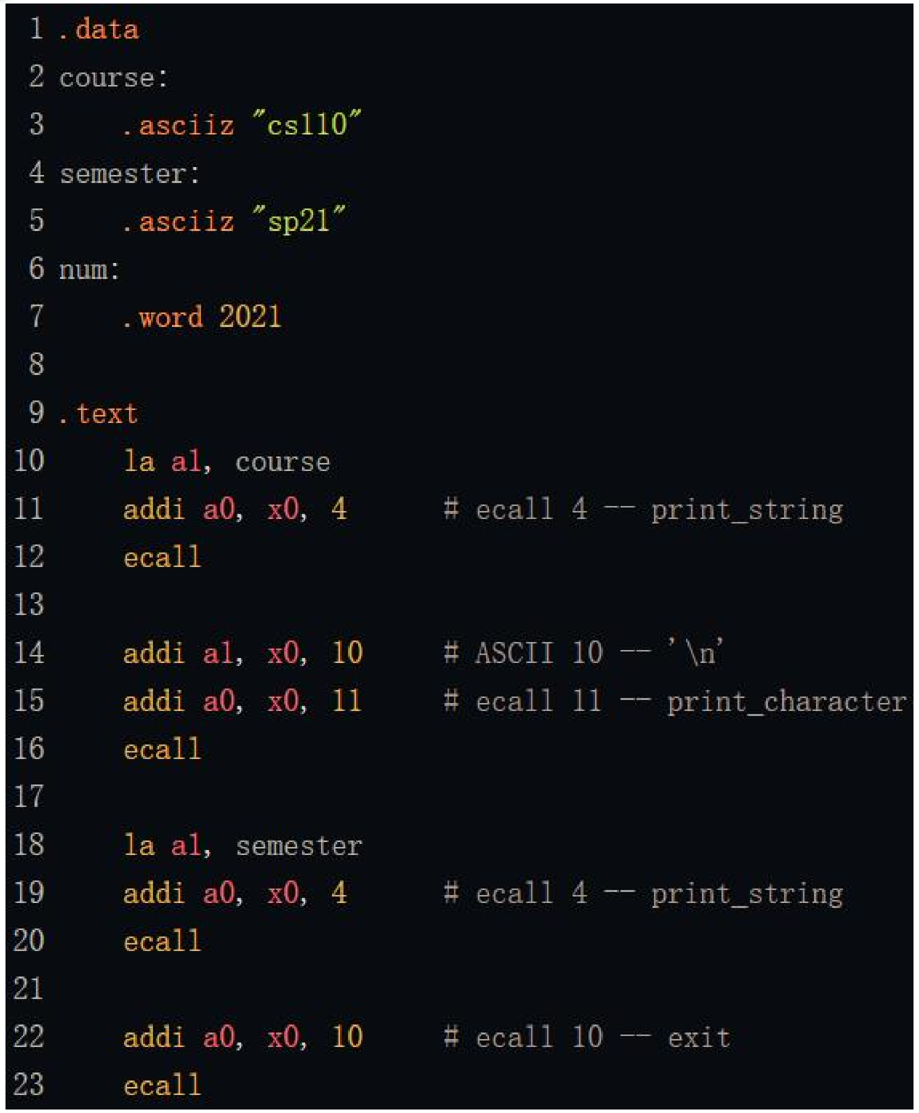
Label and Assembler Directives

Label : Hold the address of data or instructions.
(Will be placed by the actual address during assembly or link.)

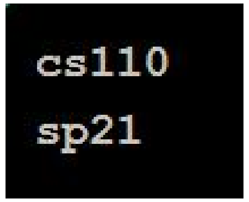
Some directives :



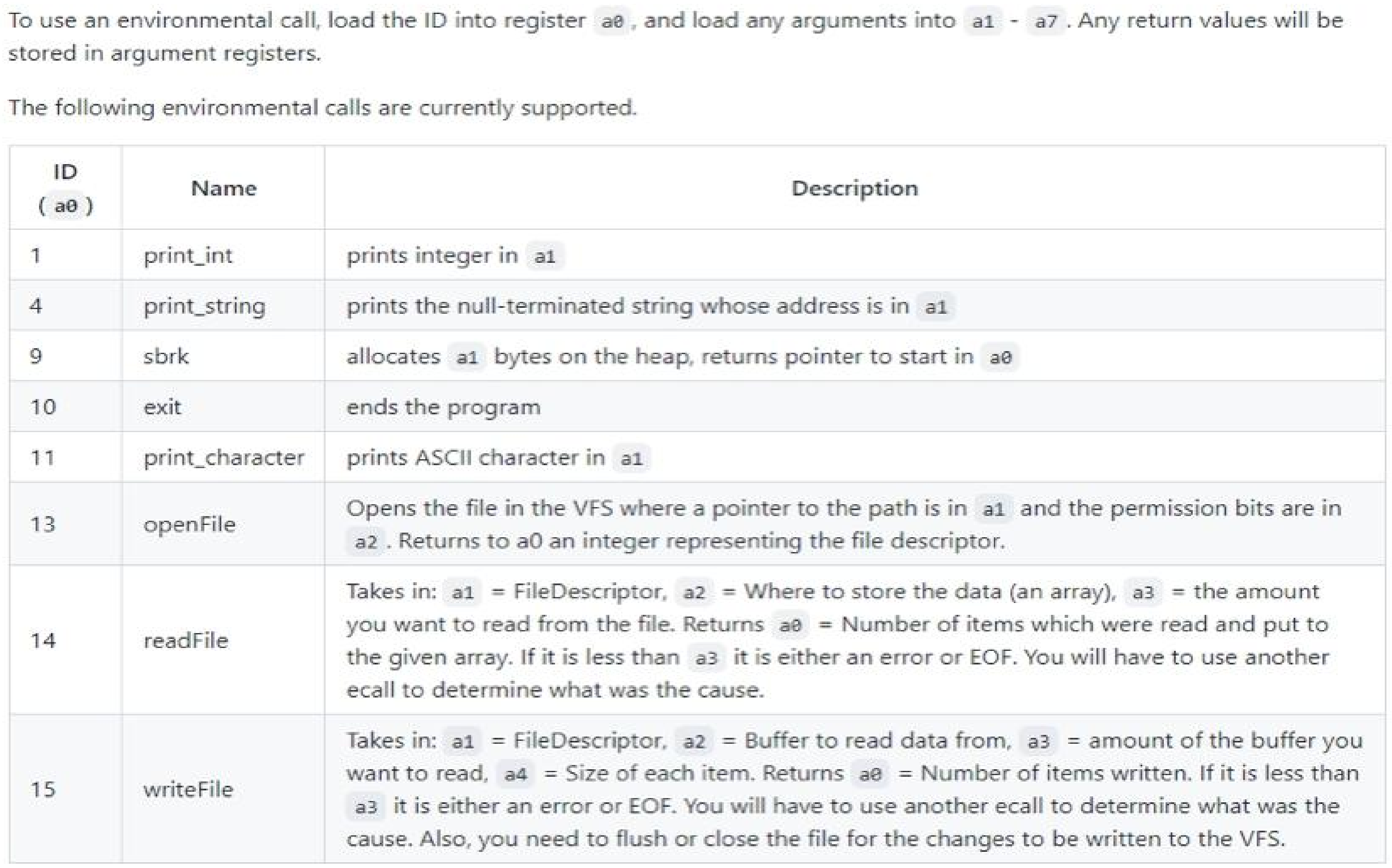
Directive	Effect
`.data`	Store subsequent items in the static segment at the next available address.
`.text`	Store subsequent instructions in the text segment at the next available address.
`.byte`	Store listed values as 8-bit bytes.
`.asciiz`	Store subsequent string in the data segment and add null-terminator.
`.word`	Store listed values as unaligned 32-bit words.
`.globl`	Makes the given label global.
`.float`	Reserved.
`.double`	Reserved.
`.align`	Reserved.


```
.data
course:
    .asciiz "cs110"
semester:
    .asciiz "sp21"
num:
    .word 2021

.text
    la a1, course
    addi a0, x0, 4      # ecall 4 -- print_string
    ecall

    addi a1, x0, 10     # ASCII 10 -- '\n'
    addi a0, x0, 11     # ecall 11 -- print_character
    ecall

    la a1, semester
    addi a0, x0, 4      # ecall 4 -- print_string
    ecall

    addi a0, x0, 10     # ecall 10 -- exit
    ecall
```

Output :


```
cs110
sp21
```

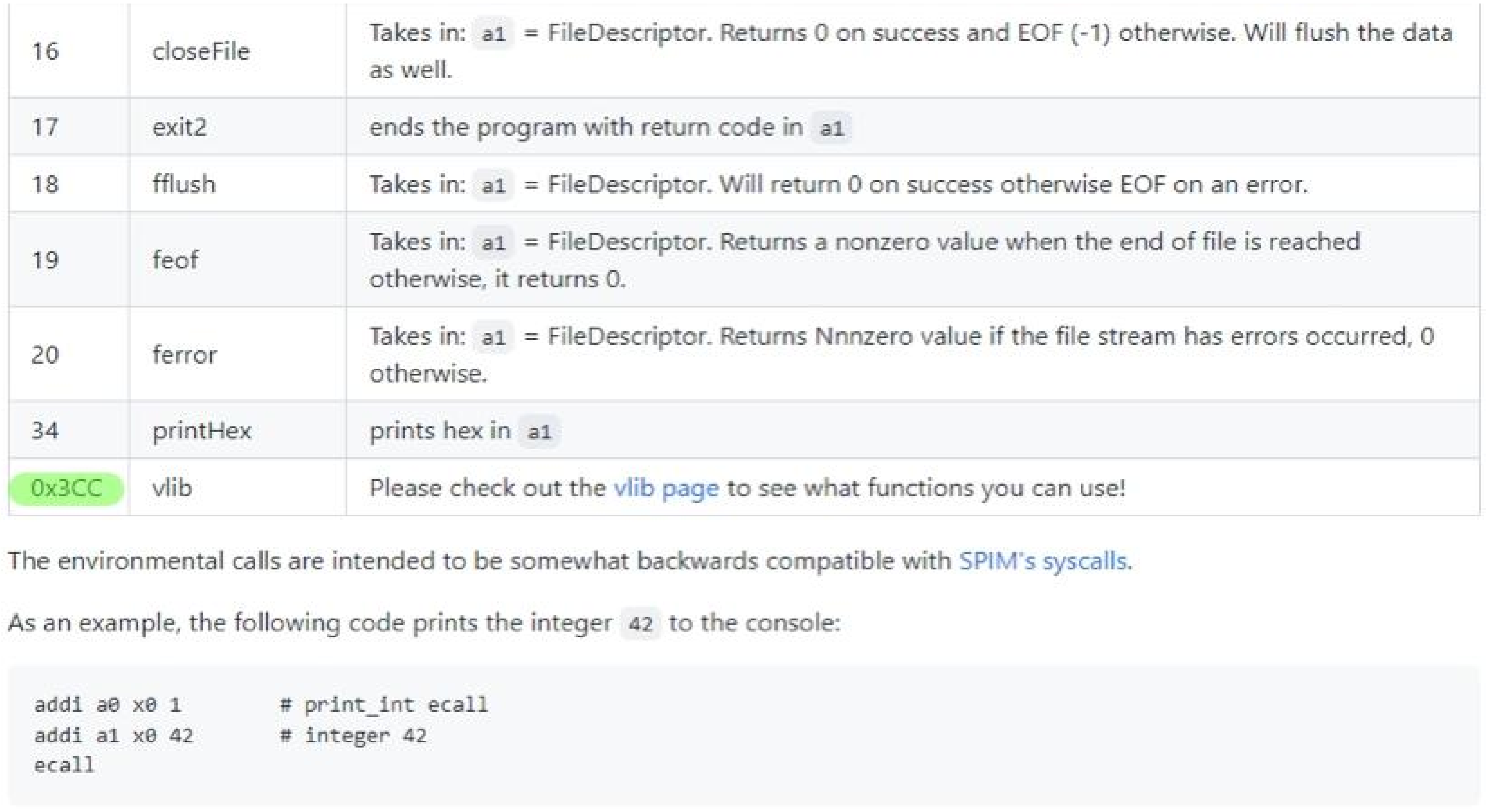
Ecall



To use an environmental call, load the ID into register `a0`, and load any arguments into `a1` - `a7`. Any return values will be stored in argument registers.

The following environmental calls are currently supported.

ID (`a0`)	Name	Description
1	print_int	prints integer in `a1`
4	print_string	prints the null-terminated string whose address is in `a1`
9	sbrk	allocates `a1` bytes on the heap, returns pointer to start in `a0`
10	exit	ends the program
11	print_character	prints ASCII character in `a1`
13	openFile	Opens the file in the VFS where a pointer to the path is in `a1` and the permission bits are in `a2`. Returns to a0 an integer representing the file descriptor.
14	readFile	Takes in: `a1` = FileDescriptor, `a2` = Where to store the data (an array), `a3` = the amount you want to read from the file. Returns `a0` = Number of items which were read and put to the given array. If it is less than `a3` it is either an error or EOF. You will have to use another ecall to determine what was the cause.
15	writeFile	Takes in: `a1` = FileDescriptor, `a2` = Buffer to read data from, `a3` = amount of the buffer you want to read, `a4` = Size of each item. Returns `a0` = Number of items written. If it is less than `a3` it is either an error or EOF. You will have to use another ecall to determine what was the cause. Also, you need to flush or close the file for the changes to be written to the VFS.

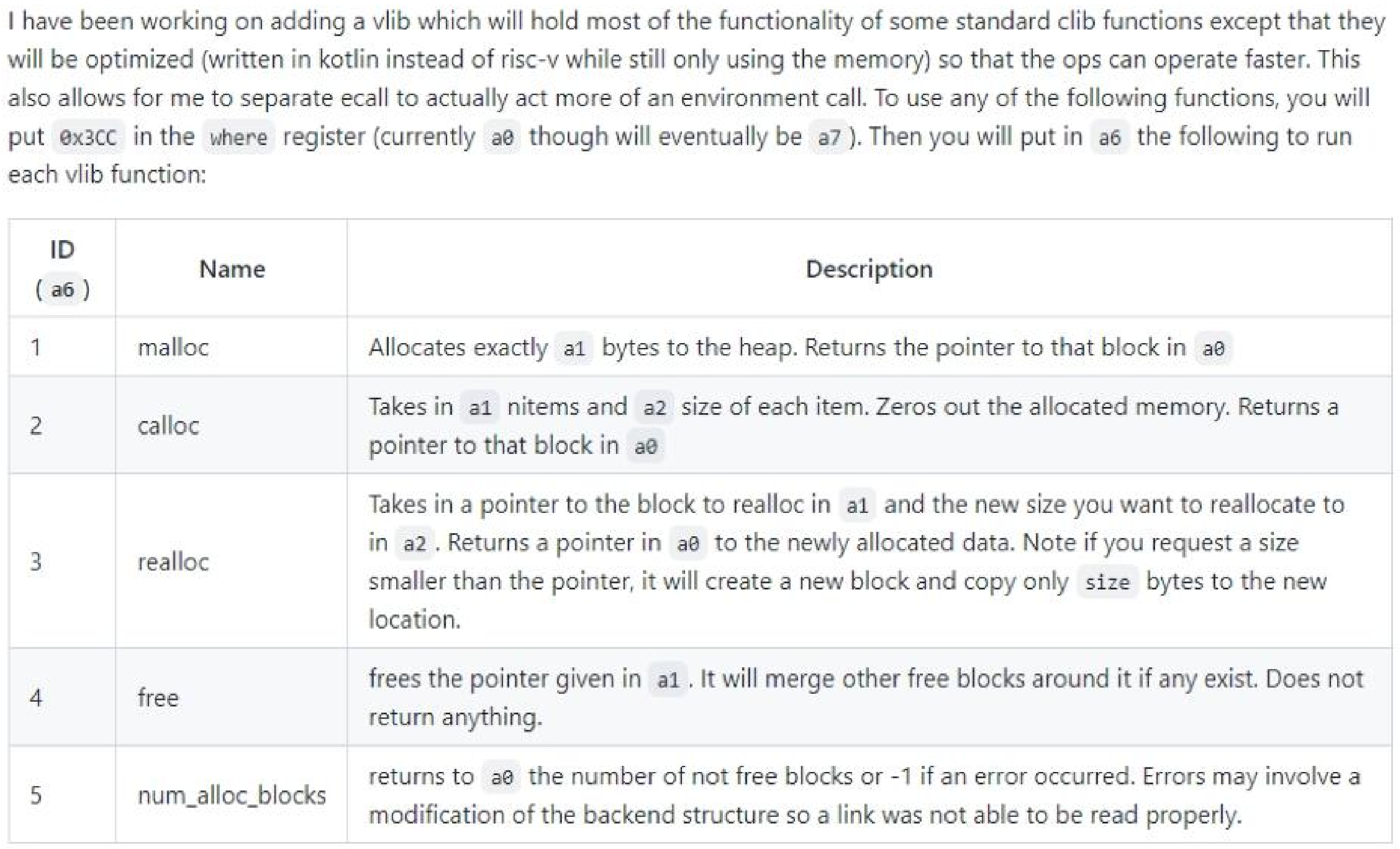


16	closeFile	Takes in: `a1` = FileDescriptor. Returns 0 on success and EOF (-1) otherwise. Will flush the data as well.
17	exit2	ends the program with return code in `a1`
18	fflush	Takes in: `a1` = FileDescriptor. Will return 0 on success otherwise EOF on an error.
19	feof	Takes in: `a1` = FileDescriptor. Returns a nonzero value when the end of file is reached otherwise, it returns 0.
20	ferror	Takes in: `a1` = FileDescriptor. Returns Nnnzero value if the file stream has errors occurred, 0 otherwise.
34	printHex	prints hex in `a1`
0x3CC	vlib	Please check out the vlib page to see what functions you can use!

The environmental calls are intended to be somewhat backwards compatible with SPIM's syscalls.

As an example, the following code prints the integer `42` to the console:

```
addi a0 x0 1        # print_int ecall
addi a1 x0 42       # integer 42
ecall
```

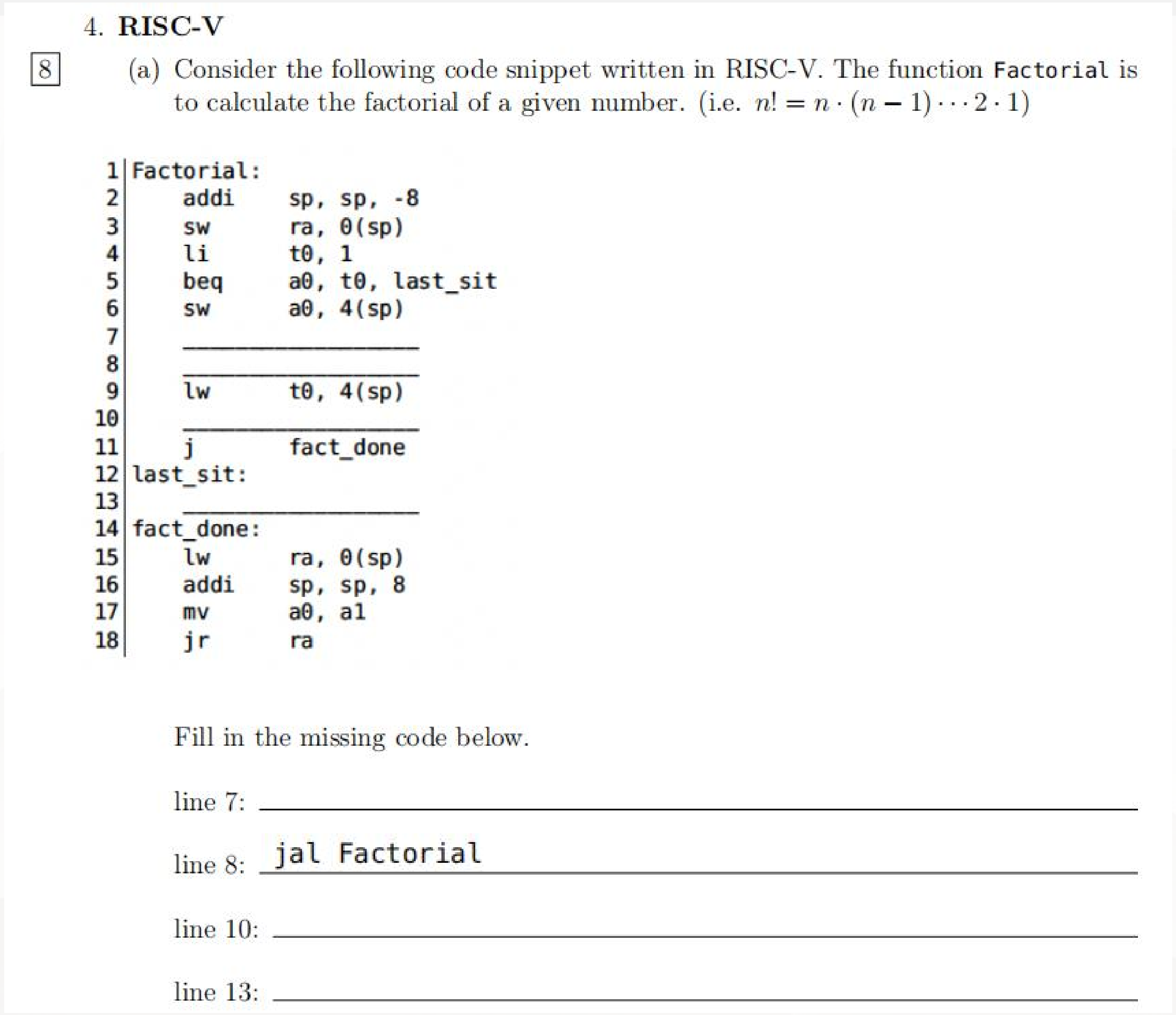


I have been working on adding a vlib which will hold most of the functionality of some standard clib functions except that they will be optimized (written in kotlin instead of risc-v while still only using the memory) so that the ops can operate faster. This also allows for me to separate ecall to actually act more of an environment call. To use any of the following functions, you will put `0x3CC` in the `where` register (currently `a0` though will eventually be `a7`). Then you will put in `a6` the following to run each vlib function:

ID (`a6`)	Name	Description
1	malloc	Allocates exactly `a1` bytes to the heap. Returns the pointer to that block in `a0`
2	calloc	Takes in `a1` nitems and `a2` size of each item. Zeros out the allocated memory. Returns a pointer to that block in `a0`
3	realloc	Takes in a pointer to the block to realloc in `a1` and the new size you want to reallocate to in `a2`. Returns a pointer in `a0` to the newly allocated data. Note if you request a size smaller than the pointer, it will create a new block and copy only `size` bytes to the new location.
4	free	frees the pointer given in `a1`. It will merge other free blocks around it if any exist. Does not return anything.
5	num_alloc_blocks	returns to `a0` the number of not free blocks or -1 if an error occurred. Errors may involve a modification of the backend structure so a link was not able to be read properly.

RISC-V Practices

上海科技大学 ShanghaiTech University



4. **RISC-V**

$\boxed{8}$ (a) Consider the following code snippet written in RISC-V. The function `Factorial` is to calculate the factorial of a given number. (i.e. $n! = n \cdot (n-1) \cdots 2 \cdot 1$)

```
 1 Factorial:
 2     addi    sp, sp, -8
 3     sw      ra, 0(sp)
 4     li      t0, 1
 5     beq     a0, t0, last_sit
 6     sw      a0, 4(sp)
 7     __________________
 8     __________________
 9     lw      t0, 4(sp)
10     __________________
11     j       fact_done
12 last_sit:
13     __________________
14 fact_done:
15     lw      ra, 0(sp)
16     addi    sp, sp, 8
17     mv      a0, a1
18     jr      ra
```

Fill in the missing code below.

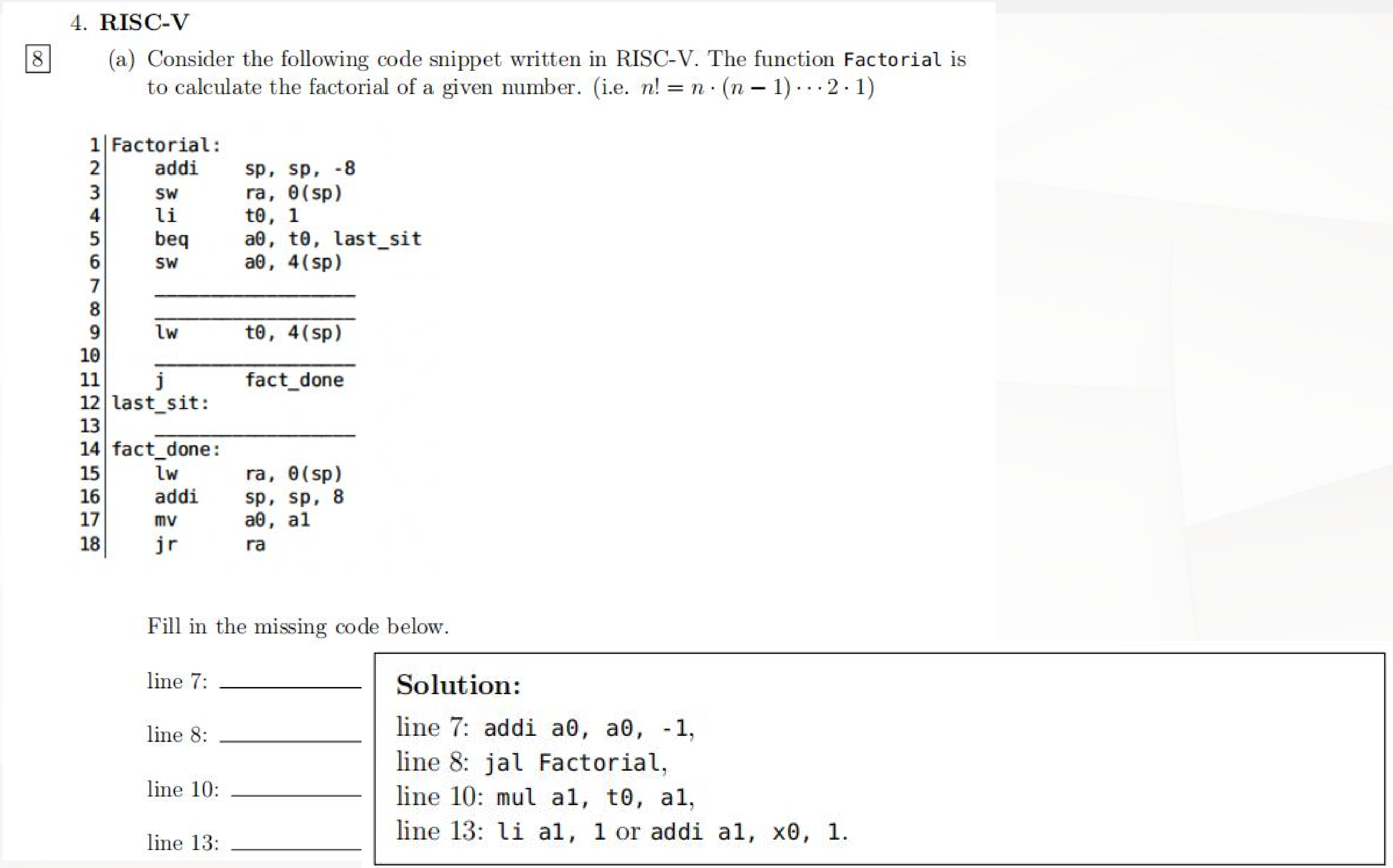
line 7: ____________________

line 8: jal Factorial

line 10: ____________________

line 13: ____________________

RISC-V Practices



4. **RISC-V**

$\boxed{8}$ (a) Consider the following code snippet written in RISC-V. The function `Factorial` is to calculate the factorial of a given number. (i.e. $n! = n \cdot (n-1) \cdots 2 \cdot 1$)

```
 1 Factorial:
 2     addi    sp, sp, -8
 3     sw      ra, 0(sp)
 4     li      t0, 1
 5     beq     a0, t0, last_sit
 6     sw      a0, 4(sp)
 7     __________________
 8     __________________
 9     lw      t0, 4(sp)
10     __________________
11     j       fact_done
12 last_sit:
13     __________________
14 fact_done:
15     lw      ra, 0(sp)
16     addi    sp, sp, 8
17     mv      a0, a1
18     jr      ra
```

Fill in the missing code below.

line 7: ____________

line 8: ____________

line 10: ____________

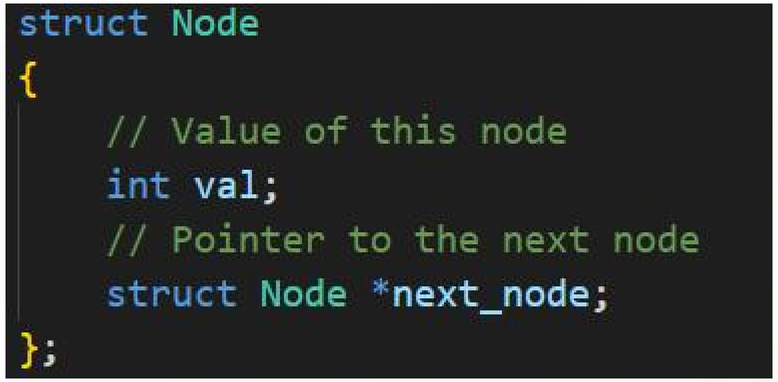
line 13: ____________

Solution:

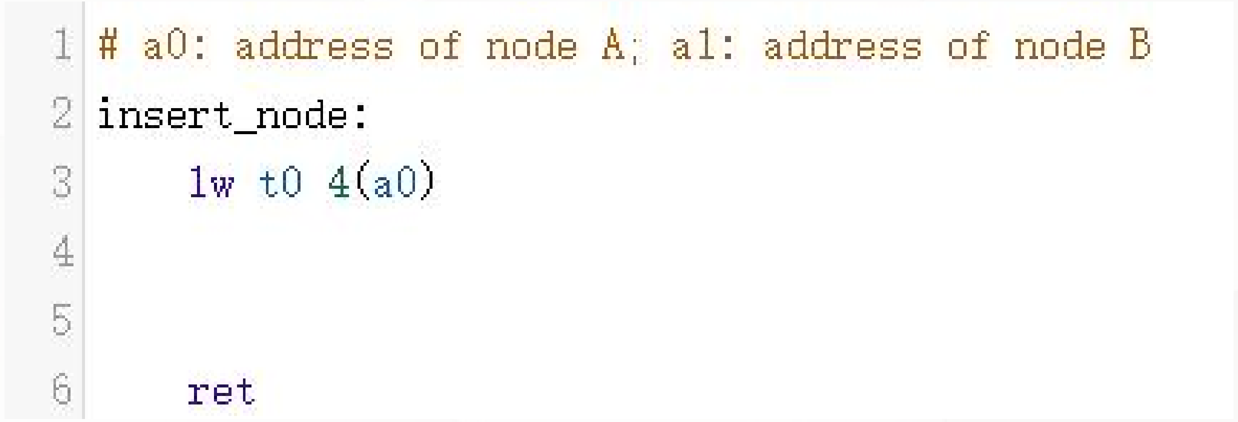
line 7: `addi a0, a0, -1`,
line 8: `jal Factorial`,
line 10: `mul a1, t0, a1`,
line 13: `li a1, 1` or `addi a1, x0, 1`.

RISC-V Practices

Singly-linked list is a common and useful data structure. In this problem, you are going to implement a linked list operation in RISC-V assembly. Assume the assembly is for **a 32-bit machine**. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a singly-linked list is defined as a **struct** type as follows.



```c
struct Node
{
    // Value of this node
    int val;
    // Pointer to the next node
    struct Node *next_node;
};
```



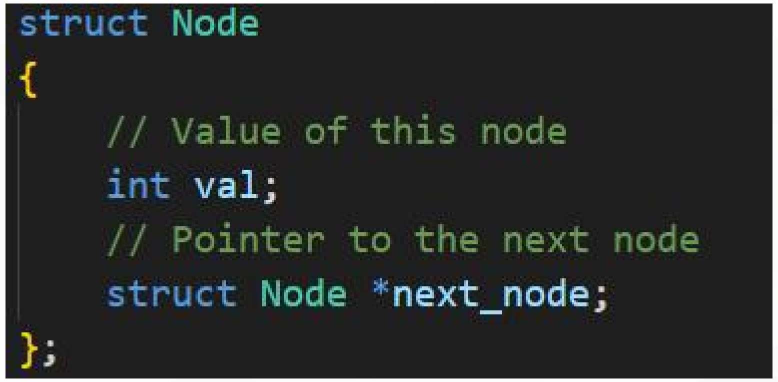
```
# a0: address of node A; a1: address of node B
insert_node:
    lw t0 4(a0)


    ret
```

Then we define the function:

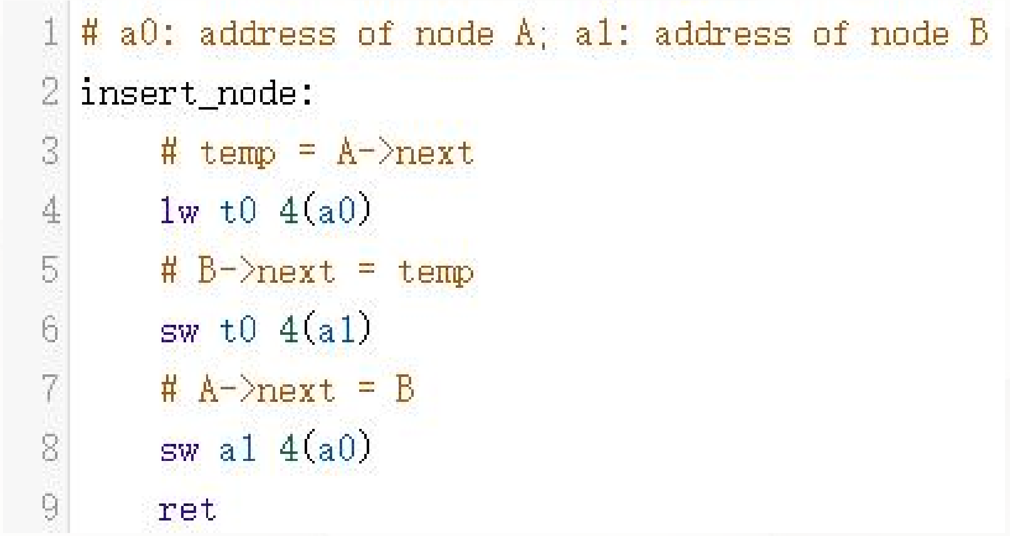
• insert node : Given a pointer to node A and a pointer to node B, this function will insert node B into the linked list, making node B the next node of node A. Node A is already in the list and assume that it is not the last node (tail) of the list.

RISC-V Practices

Singly-linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for **a 32-bit machine**. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a singly-linked list is defined as a **struct** type as follows.



```
struct Node
{
    // Value of this node
    int val;
    // Pointer to the next node
    struct Node *next_node;
};
```



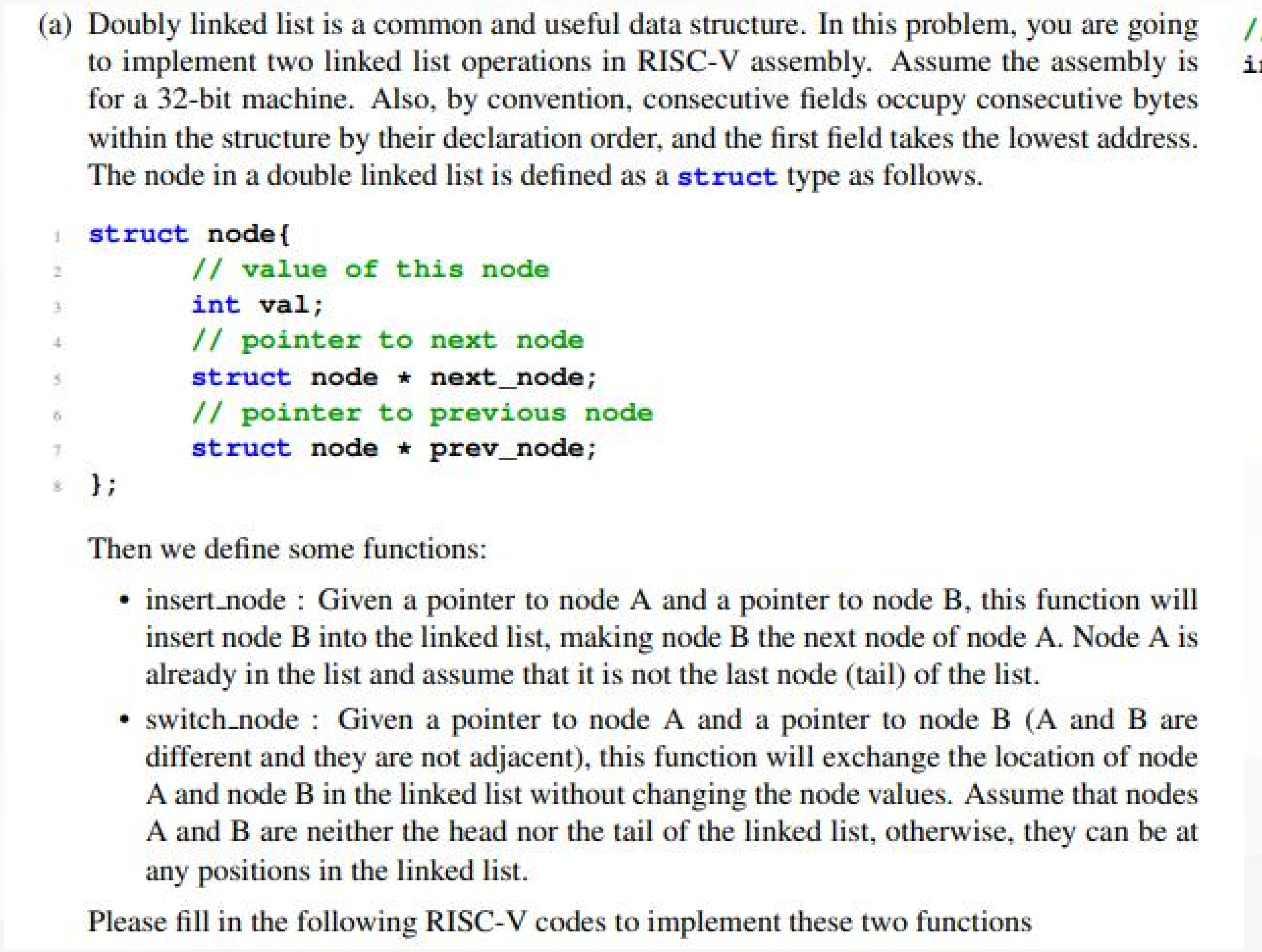
```
# a0: address of node A; a1: address of node B
insert_node:
    # temp = A->next
    lw t0 4(a0)
    # B->next = temp
    sw t0 4(a1)
    # A->next = B
    sw a1 4(a0)
    ret
```

Then we define some functions:

• insert node : Given a pointer to node A and a pointer to node B, this function will insert node B into the linked list, making node B the next node of node A. Node A is already in the list and assume that it is not the last node (tail) of the list.

RISC-V Practices

上海科技大学 ShanghaiTech University



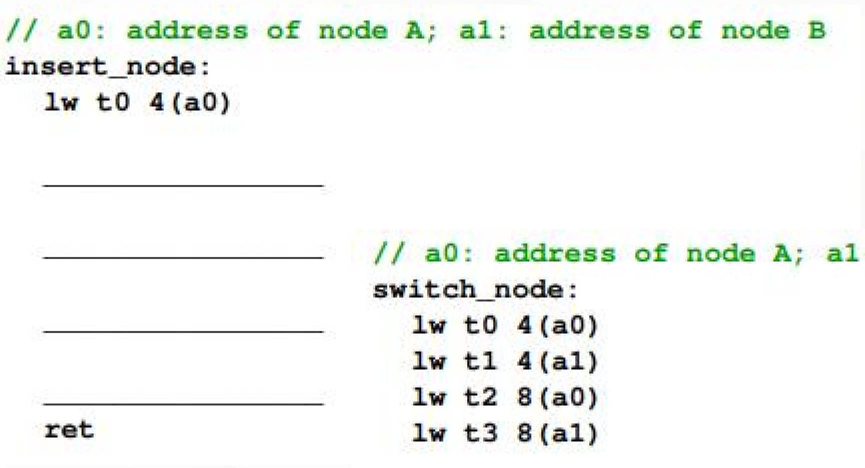
(a) Doubly linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for a 32-bit machine. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a double linked list is defined as a `struct` type as follows.

```
struct node{
        // value of this node
        int val;
        // pointer to next node
        struct node * next_node;
        // pointer to previous node
        struct node * prev_node;
};
```

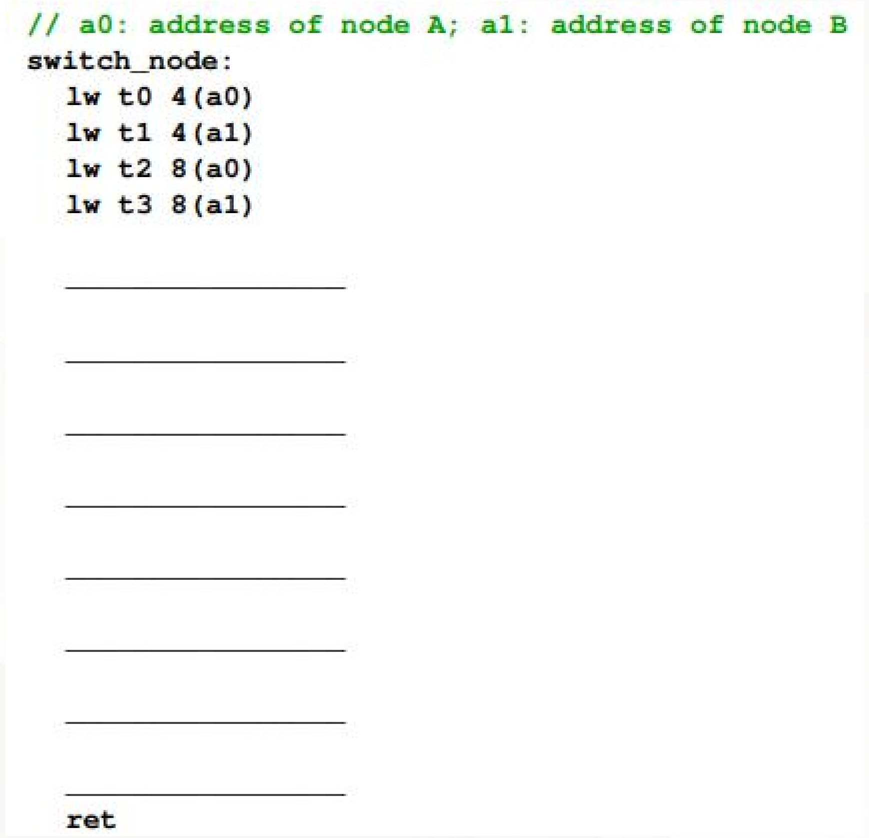
Then we define some functions:

- insert_node : Given a pointer to node A and a pointer to node B, this function will insert node B into the linked list, making node B the next node of node A. Node A is already in the list and assume that it is not the last node (tail) of the list.
- switch_node : Given a pointer to node A and a pointer to node B (A and B are different and they are not adjacent), this function will exchange the location of node A and node B in the linked list without changing the node values. Assume that nodes A and B are neither the head nor the tail of the linked list, otherwise, they can be at any positions in the linked list.

Please fill in the following RISC-V codes to implement these two functions

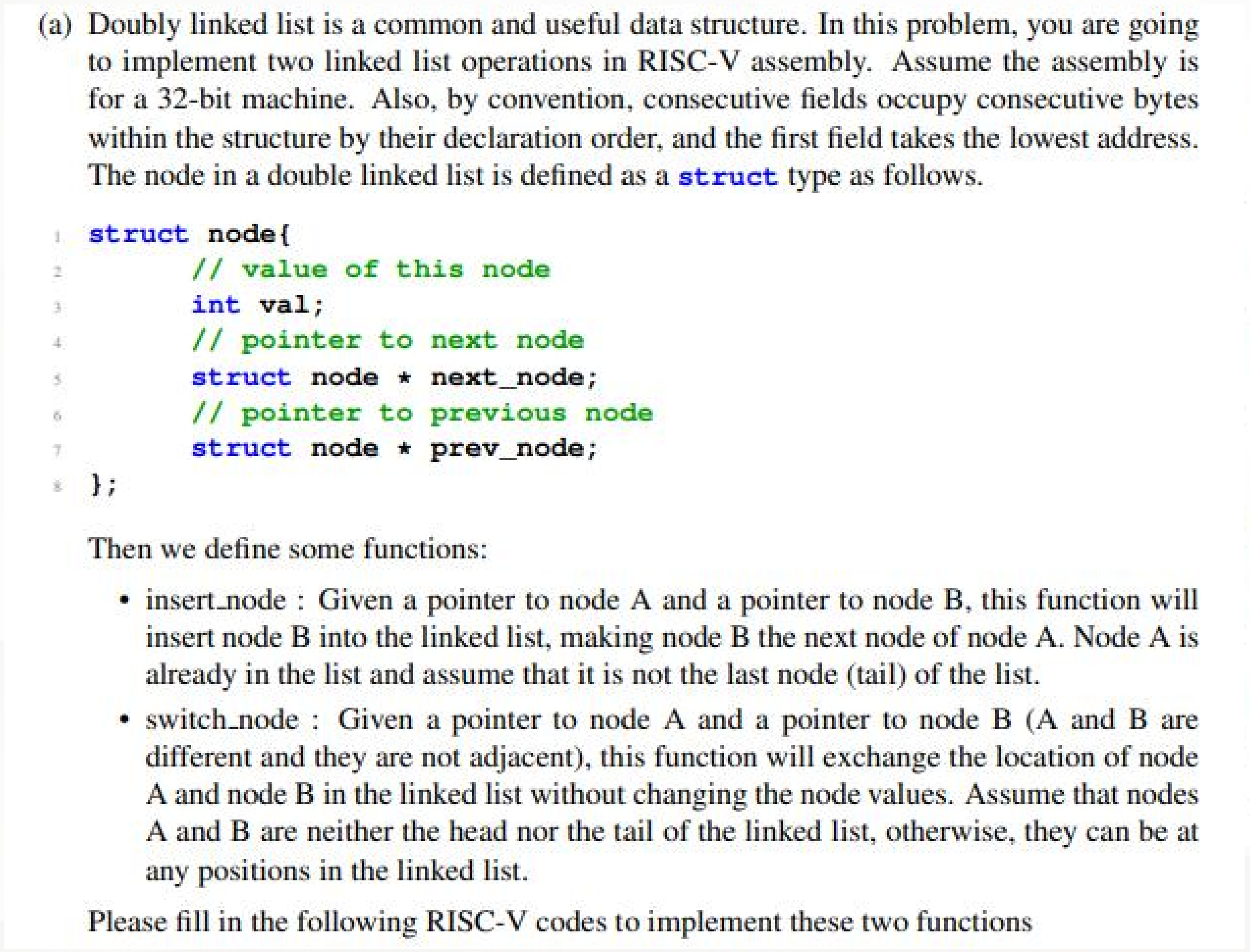


```
// a0: address of node A; a1: address of node B
insert_node:
  lw t0 4(a0)

  ________________

  ________________

  ________________

  ________________
  ret
```



```
// a0: address of node A; a1: address of node B
switch_node:
  lw t0 4(a0)
  lw t1 4(a1)
  lw t2 8(a0)
  lw t3 8(a1)

  ________________

  ________________

  ________________

  ________________

  ________________

  ________________

  ________________

  ________________
  ret
```

RISC-V Practices



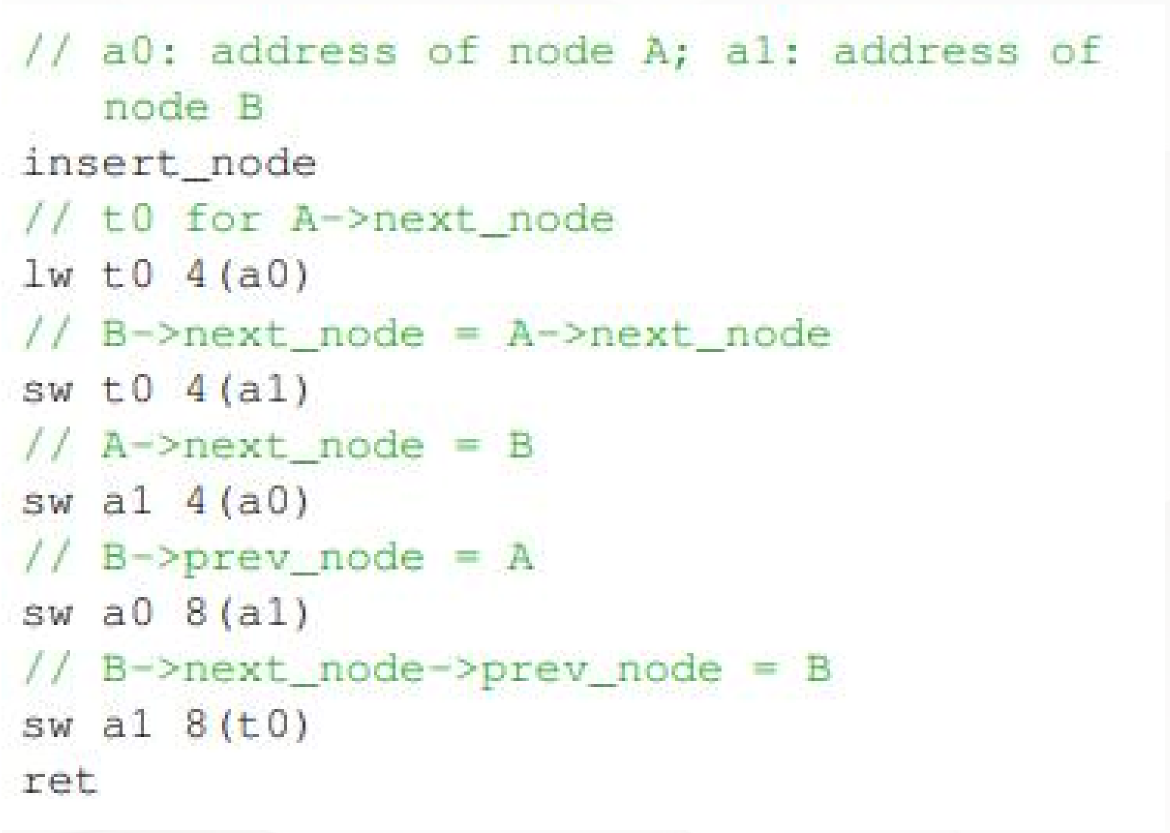
(a) Doubly linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for a 32-bit machine. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a double linked list is defined as a `struct` type as follows.

```c
struct node{
        // value of this node
        int val;
        // pointer to next node
        struct node * next_node;
        // pointer to previous node
        struct node * prev_node;
};
```

Then we define some functions:

- insert_node : Given a pointer to node A and a pointer to node B, this function will insert node B into the linked list, making node B the next node of node A. Node A is already in the list and assume that it is not the last node (tail) of the list.
- switch_node : Given a pointer to node A and a pointer to node B (A and B are different and they are not adjacent), this function will exchange the location of node A and node B in the linked list without changing the node values. Assume that nodes A and B are neither the head nor the tail of the linked list, otherwise, they can be at any positions in the linked list.

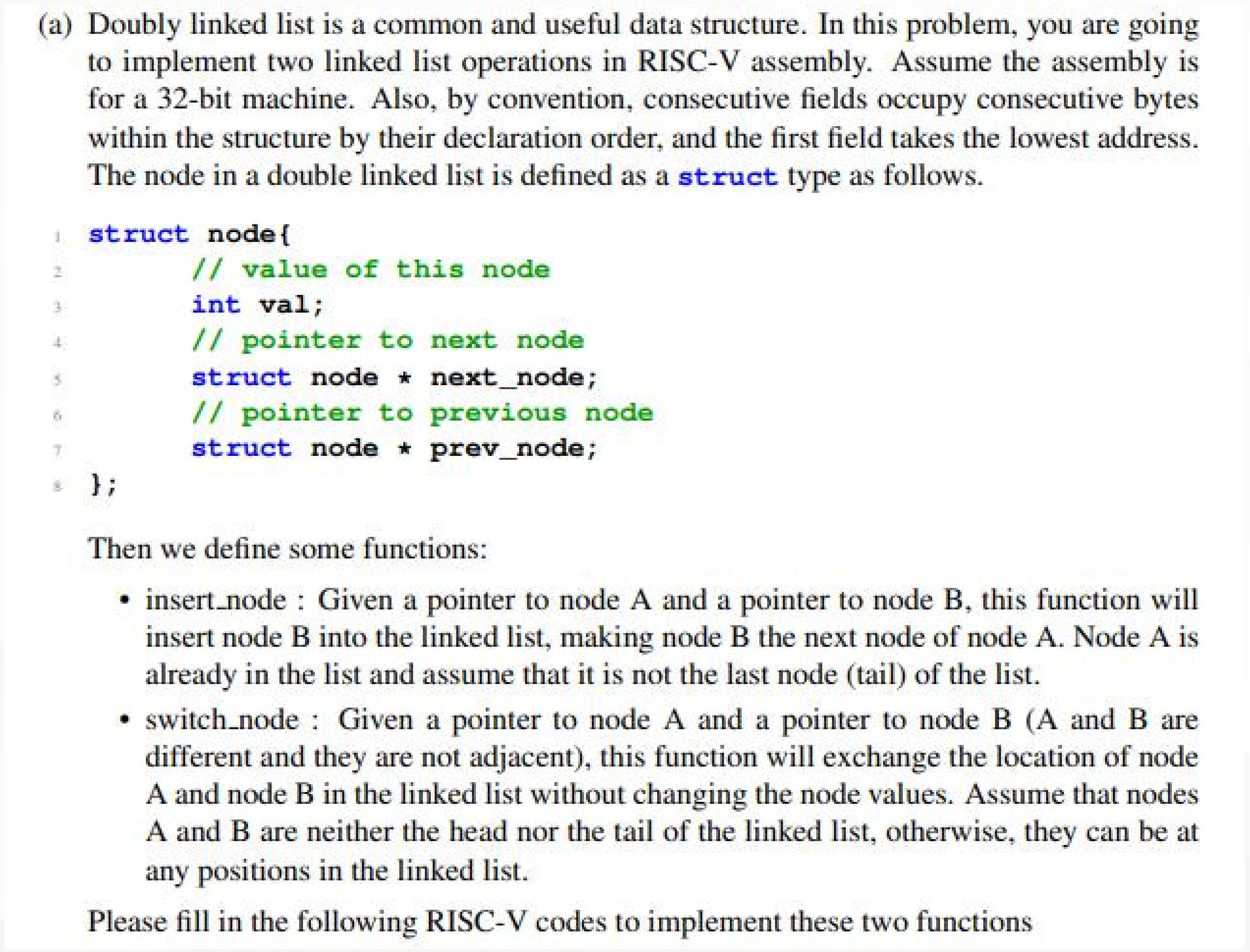
Please fill in the following RISC-V codes to implement these two functions



```
// a0: address of node A; a1: address of
   node B
insert_node
// t0 for A->next_node
lw t0 4(a0)
// B->next_node = A->next_node
sw t0 4(a1)
// A->next_node = B
sw a1 4(a0)
// B->prev_node = A
sw a0 8(a1)
// B->next_node->prev_node = B
sw a1 8(t0)
ret
```

RISC-V Practices

上海科技大学 ShanghaiTech University



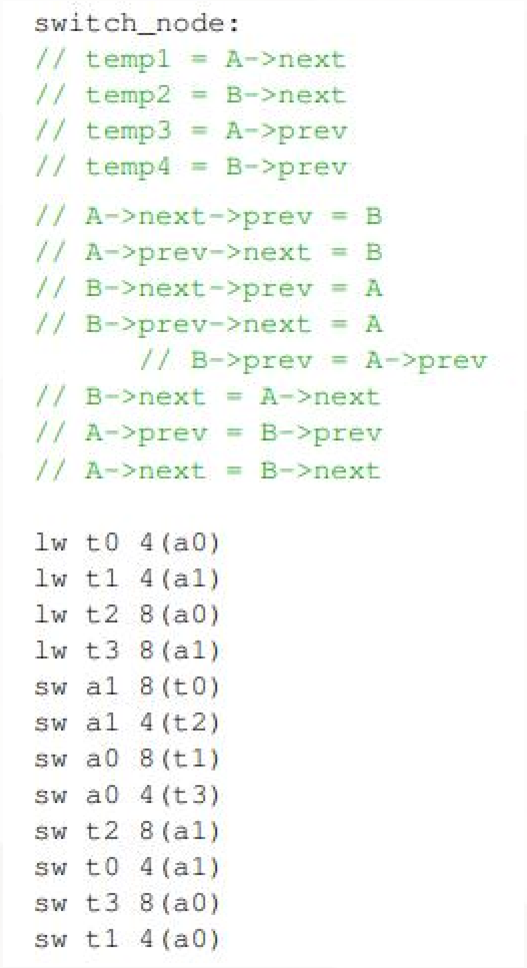
(a) Doubly linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for a 32-bit machine. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a double linked list is defined as a `struct` type as follows.

```c
struct node{
        // value of this node
        int val;
        // pointer to next node
        struct node * next_node;
        // pointer to previous node
        struct node * prev_node;
};
```

Then we define some functions:

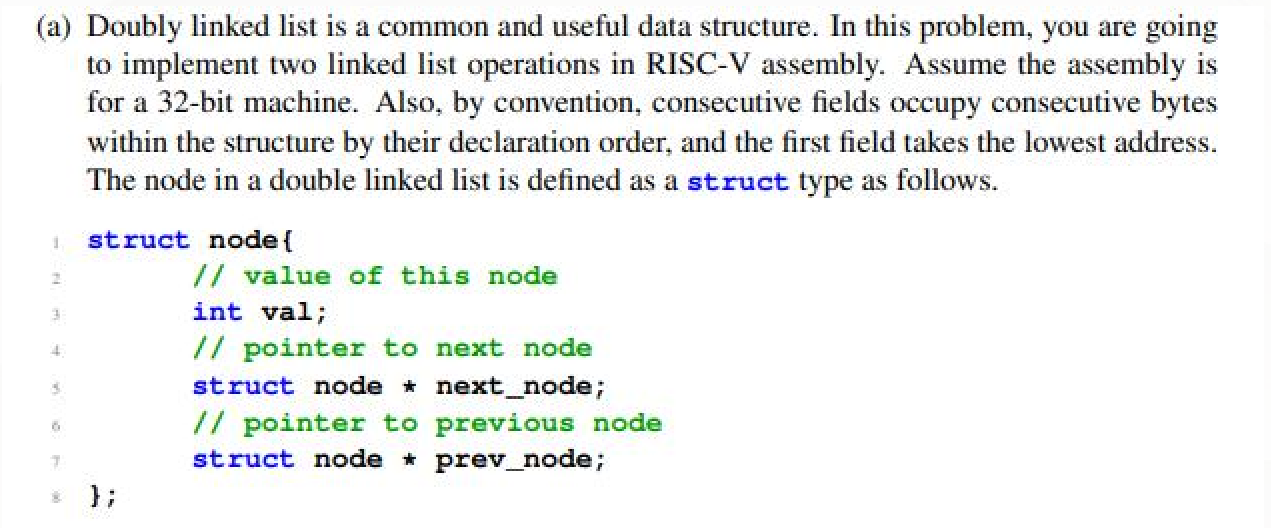
- insert_node : Given a pointer to node A and a pointer to node B, this function will insert node B into the linked list, making node B the next node of node A. Node A is already in the list and assume that it is not the last node (tail) of the list.
- switch_node : Given a pointer to node A and a pointer to node B (A and B are different and they are not adjacent), this function will exchange the location of node A and node B in the linked list without changing the node values. Assume that nodes A and B are neither the head nor the tail of the linked list, otherwise, they can be at any positions in the linked list.

Please fill in the following RISC-V codes to implement these two functions



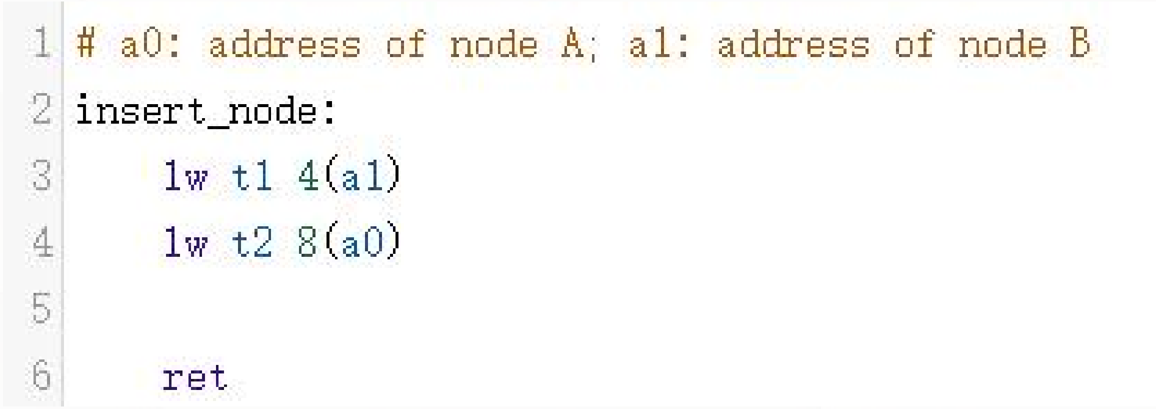
```
switch_node:
// temp1 = A->next
// temp2 = B->next
// temp3 = A->prev
// temp4 = B->prev

// A->next->prev = B
// A->prev->next = B
// B->next->prev = A
// B->prev->next = A
      // B->prev = A->prev
// B->next = A->next
// A->prev = B->prev
// A->next = B->next

lw t0 4(a0)
lw t1 4(a1)
lw t2 8(a0)
lw t3 8(a1)
sw a1 8(t0)
sw a1 4(t2)
sw a0 8(t1)
sw a0 4(t3)
sw t2 8(a1)
sw t0 4(a1)
sw t3 8(a0)
sw t1 4(a0)
```

RISC-V Practices



(a) Doubly linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for a 32-bit machine. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a double linked list is defined as a `struct` type as follows.

```c
struct node{
        // value of this node
        int val;
        // pointer to next node
        struct node * next_node;
        // pointer to previous node
        struct node * prev_node;
};
```



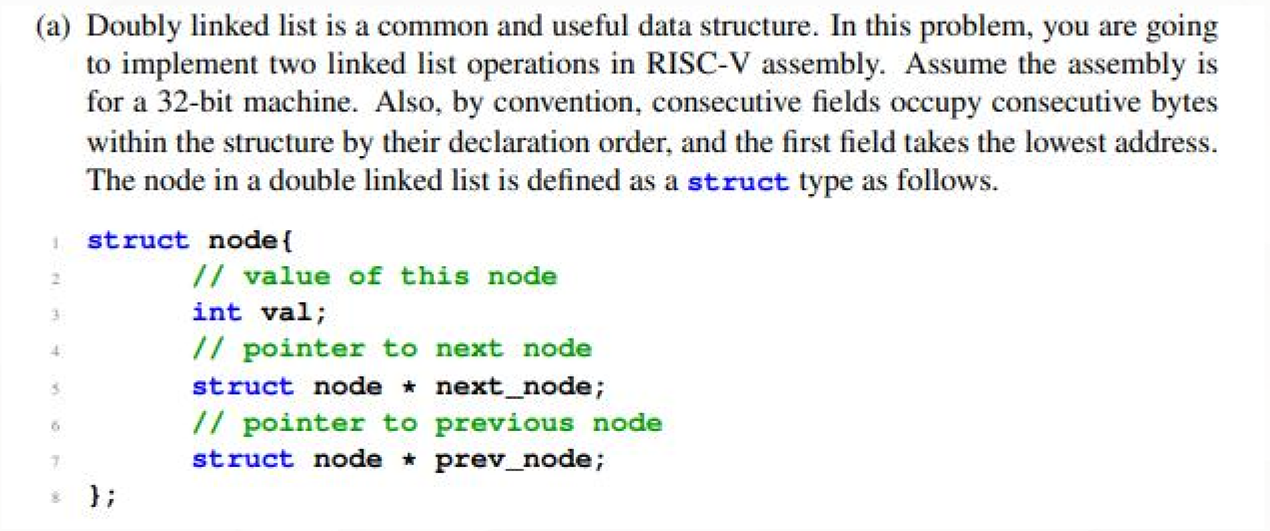
```
# a0: address of node A; a1: address of node B
insert_node:
    lw t1 4(a1)
    lw t2 8(a0)

    ret
```

Consider another senario for switch node:

• switch node : Given a pointer to node A and a pointer to node B (**A and B are**
different and they are adjacent, the next node of A is B), this function will exchange the location of node
A and node B in the linked list without changing the node values. Assume that nodes
A and B are neither the head nor the tail of the linked list, otherwise, they can be at
any positions in the linked list.

RISC-V Practices

上海科技大学 ShanghaiTech University

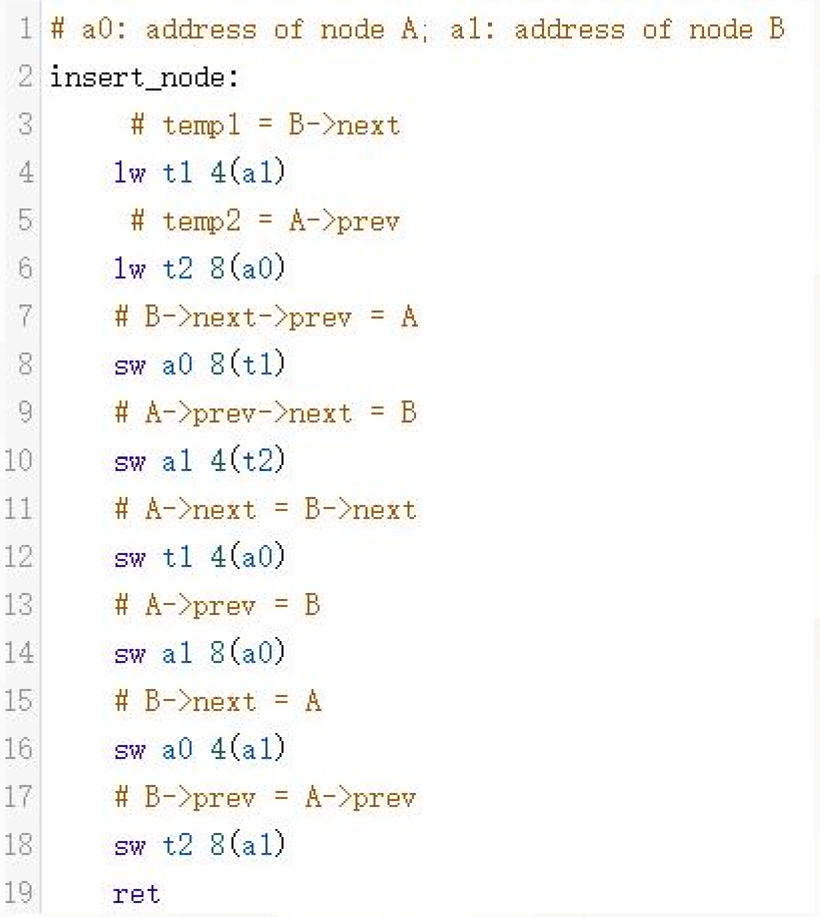


(a) Doubly linked list is a common and useful data structure. In this problem, you are going to implement two linked list operations in RISC-V assembly. Assume the assembly is for a 32-bit machine. Also, by convention, consecutive fields occupy consecutive bytes within the structure by their declaration order, and the first field takes the lowest address. The node in a double linked list is defined as a struct type as follows.

```c
struct node{
        // value of this node
        int val;
        // pointer to next node
        struct node * next_node;
        // pointer to previous node
        struct node * prev_node;
};
```

Consider another senario for switch node:

• switch node : Given a pointer to node A and a pointer to node B (**A and B are**
different and they are adjacent, the next node of A is B), this function will exchange the location of node
A and node B in the linked list without changing the node values. Assume that nodes
A and B are neither the head nor the tail of the linked list, otherwise, they can be at
any positions in the linked list.



```
# a0: address of node A; a1: address of node B
insert_node:
     # temp1 = B->next
    lw t1 4(a1)
     # temp2 = A->prev
    lw t2 8(a0)
    # B->next->prev = A
    sw a0 8(t1)
    # A->prev->next = B
    sw a1 4(t2)
    # A->next = B->next
    sw t1 4(a0)
    # A->prev = B
    sw a1 8(a0)
    # B->next = A
    sw a0 4(a1)
    # B->prev = A->prev
    sw t2 8(a1)
    ret
```

Venus Q&A

Thank you for attending the discussion!

Wish you good luck doing homework/projects/exams!