Student Name Chinese: ____________________

Student Name Pinyin: ____________________

Student Email: ____________________ **(Shanghaitech email w/o "@shanghaitech.edu.cn")**

Student Id: ____________________

School: ____________________

Year of Entrance: ____________________

ShanghaiTech University Midterm I Examination Cover Sheet

April 6 2021

Academic Year : ___2020___to___2021__ Term: __Spring______

Course-offering School: ____SIST___________

Instructor: Sören Schwertfeger & Chundong Wang

Course Name: Computer Architecture I

Course Number: CS110

Exam Instructions for Students:

1. All examination rules must be strictly observed throughout the entire test, and any form of cheating is prohibited.

2. Other than allowable materials, students taking closed-book tests must place their books, notes, tablets and any other electronic devices in places designated by the examiners.

3. Students taking open-book tests may use allowable materials authorized by the examiners. They must complete the exam independently without discussion with each other or exchange of materials.

For Marker's Use:

Section	1	2	3	4	5	6	7	8	9	10	11	12	Total
Max	1	3	11	11	10	12	7	16	5	10	7	7	100
Marks													
Recheck													

1 1. **First Task (worth one point): Fill in you name**

Fill in your name and email on the front page and your ShanghaiTech email on top of every page (without @shanghaitech.edu.cn) (so write your email in total 14 times).

2. **Various Questions**

3 (a) Name the 6 Great Ideas in Computer Architecture as taught in the lectures.

____

____

____

____

____

____

3. **Number Representation**

3 (a) Given the number `0x811F00FA`. It can be interpreted as:

a binary number: ______________________________

four unsigned bytes: ______________________________

four two's complement bytes: ______________________________

4 (b) A **quarter** is a single byte split into the following fields (1 sign, 3 exponent, 4 mantissa): `SEEEMMMM`. It has all the properties of **IEEE 754** (including denormal numbers, NaNs and $\pm\infty$) just with different ranges, precision and representations. For a **quarter**, the bias of the exponent is 3, and the implicit exponent for denormal numbers are -2.

What is the largest number smaller than ∞?

In binary ______________

In decimal ______________

Which negative denormal number is closest to 0?

In binary ______________

In decimal ______________

4 (c) What is the value of `q1`, `q2`, `c`, `d`?

Hint Rounding mode: round toward even/0.

```
quarter q1, q2, q3, c, d;
q1 = -0.25;
q2 = -4.0;
q3 = 0.125;
c = q1 + (q2 + q3);
d = (q1 + q2) + q3;
```

`q1` in binary ____________________

`q2` in binary ____________________

`c` in decimal ____________________

`d` in decimal ____________________

4. C Basics

5 (a) Memory of C

```
#include <stdlib.h>

int main() {
    static int p = 5;

    char *str = ______________________;
    /* some other codes, and you can skip it. */
    return 0;
}
```

1. You need to allocate a string `str` containing `p` characters. Write the code above (please use `malloc`).
2. Fill in the correct memory section based on what the given C expressions evaluate to.

`&p` ____________________

`&str` ____________________

`str` ____________________

3 (b) Catch bugs!

1. When you want to debug with GDB, what flag you will put in your compilation?

   ____

2. Write down some essential commands in GDB. Example: Start your program: `run/r`

   Set break point: ______________________________

   Show next line(stepping into function calls): ______________________________

3 (c) C programming: Reverse singly linked list. For example, convert 1 → 2 → 3 → `NULL` to 3 → 2 → 1 → `NULL`. (You may not need all of the lines)

```
#include <stdio.h>
#include <stdlib.h>

/* Definition for singly-linked list. */
struct ListNode {
    int val;
    struct ListNode *next;
};

/* Given the head of a singly linked list, reverse
    the list, and return the head of reversed list.*/
struct ListNode *reverse_list(struct ListNode *head) {
    struct ListNode *prev = NULL;
    struct ListNode *curr = head;
    struct ListNode *next = head;
    while (curr) {
        next = next->next;

        ____________________________

        ____________________________

        ____________________________

        ____________________________

        ____________________________

        ____________________________
    }
    return prev;
}
```

5. **Byte-Swap Operation**

Assuming we are in a **32bit**, **little endian** system. *Little Dragon* receives a 4-byte integer num, he wants to swap the value of num's i^{th} byte and j^{th} byte ($i, j \in \{0, 1, 2, 3\}$, $i \neq j$) to get a new number!

3 (a) **Idea I**: *Little Dragon* wants to directly retrieve the i^{th} and j^{th} byte of num, then swap them.

First of all, define a MACRO to get the i^{th} byte of num. Read the following C code, then help *Little Dragon* to fill in the blank lines (Line 4 and 10) so the output should be `0x34`. When defining the MACRO, use `&`, `|`, `^`, `~`, `>>`, `<<` operators **only**. Remember to write a meaningful MACRO such that *Little Dragon* can reuse it again (directly return `0x34` is not allowed)!

```
 1 #include <stdio.h>
 2 #include <stdint.h>
 3
 4 #define GET_BYTE(num, ind) _________________________________
 5
 6 int main(){
 7     int number, index;
 8     int8_t byte;
 9     number = 0x12345678;
10
11     index = ___________; /* index is one of {0, 1, 2, 3} */
12     byte = GET_BYTE(number, index);
13     printf("%#x\n", byte); /* should print 0x34 */
14     return 0;
15 }
```

Write your answer above.

4 (b) **Idea II**: An alternative way to fetch the i^{th} byte is *Union*. *Little Dragon* wrote the following code, but he is a little confused about the concept of *little endian* and *big endian*. Help him answer the questions below!

```
 1 #include <stdio.h>
 2 #include <stdint.h>
 3
 4 /* Tip on union: data type that stores its members
 5    in the same memory location */
 6 typedef union {
 7     struct {
 8         uint8_t byte0;
 9         uint8_t byte1;
10         uint8_t byte2;
11         uint8_t byte3;
12     } bytes;
13     int all_bits;
14 } MyInt;
15
16 int main() {
17     MyInt intA;
18     intA.all_bits = 0x12345678;
19     printf("%#x, %#x\n", intA.bytes.byte1, intA.bytes.byte3);
20     return 0;
21 }
```

What is the expected output (in hexadecimal format) of Line 19:

- if the system is **little endian**?

___

- if the system is **big endian**?

___

3 (c) **Idea III**: *Little Dragon* is fasczinated in playing with bitwise operations. He wrote the following function in C.

```
1 void byte_xor(int num, int a, int b) {
2     char *ret_val = (char *) &num;
3
4     ret_val[b] ?? ret_val[a];
5     ret_val[a] ?? ret_val[b];
6     ret_val[b] ?? ret_val[a];
7
8     printf("%#x\n", num);
9 }
```

What operators are expected to substitute the `??` in Line 4, 5, and 6, such that the result of `byte_xor(0x1133CCFF, 1, 3)` will be `0xCC3311FF`?

A. `&=`, `&=`, `&=` B. `&=`, `^=`, `^=` C. `|=`, `^=`, `~=` D. `^=`, `^=`, `^=`

6. **RISC-V programming**

In this question, you are asked to implement a simple recursive function in RISC-V. The function takes a decimal number as input, then outputs it's octal representation encoded as decimal digits. For example, if the input to this function is 100, then the output would be 144.

The recursive function implemented in C is given below:

```c
int find_octal(unsigned int decimal) {
    if (decimal == 0) {
        return 0;
    } else {
        return decimal % 8 + 10 * find_octal(decimal / 8);
    }
}
```

A skeleton of RISC-V code is given below.

DO NOT fill in them immediately. Do some warm-ups first!

```
find_octal:
    addi    sp, sp, -8
    sw      ra, 4(sp)
    sw      s0, 0(sp)

    beq     a0, x0, _____

    _____________________   # set s0 to something

    _____________________   # set a0 to something

    jal     ra, _________   # recursive call

    _____________________
    mul     a0, t0, a0

    _____________________   # a0 = ???
postamble:

    _____________________   # Restore ra

    _____________________   # restore ...

    _____________________   # restore ...
end:
    jr      ra
```

2 (a) Translate the following RISC-V instructions into machine code.

```
sw      ra, 4(sp)      __________________________

andi    s0, a0, 7      __________________________
```

2 (b) What is one pseudo instruction in the RISC-V code above? How can you change it into one base instruction?

Pseudo instruction: ________________________

After your change: ________________________

8 (c) Fill in the missing code above.

7. RISC-V Basic

5 (a) Write a function in RISC-V code to return 0 if the input 32-bit float is an infinite value, else a non-zero value. The input and output will be stored in `a0`, as usual. Do not use pseduo instructions!

```
is_not_infinity:




    ret  #  <=  Return instruction
```

2 (b) True or False.

1. Let `a0` point to the start of an array `x`. `lw s0, 4(a0)` will always load `x[1]` into `s0`.
2. After calling a function and having that function return, the t registers may have been changed during the execution of the function, while a registers cannot.

1	2

8. **CALL**

Answer the following questions with regard to the following C program.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc == (1 + 1)) {
        printf("Hello, %s.\n", argv[1]);
    } else {
        printf("Goodbye.\n");
    }

    return 0;
}
```

8 (a) Select which stage of CALL is responsible for the following actions. Please fill you answer (A, B, C or D) in the table below.

A. Compiler B. Assembler C. Linker D. Loader

1. Removes all pseudo instructions.
2. Provide the address to the string `"Goodbye.\n"`.
3. Remove most duplicate instructions in the program in order to optimize the program.
4. Put arguments in the address of `argv` so that the program could read from it.
5. Incorporating dynamic libraries so that the program could call `printf` in the C standard library.
6. Creates the symbol table so that we can know the address to the function `main` in future stages.
7. The parser is used to determine the operator precedence in `argc == (1 + 1)`.
8. Determine the jump address the `if` statement is jumping to.

1	2	3	4	5	6	7	8

8 (b) True or False. Please fill your answer (T or F) in the table below.

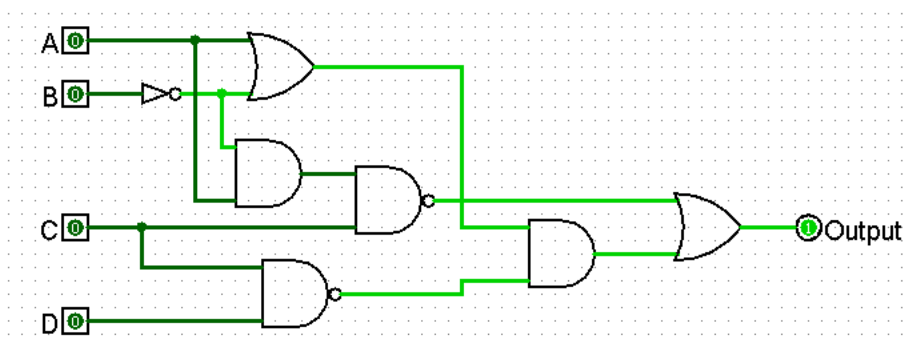
1. Pseudo instructions are not allowed in the output of compiler.
2. Statically-linked libraries are incorporated into the program during the load stage.
3. Dynamically-linked libraries are incorporated into the program during the link stage.
4. The interpreted program (like Python) runs way faster than a compiled one (like C) in most cases.
5. The assembler takes two passes over the code to resolve PC-Relative target addresses.

6. Copying arguments passed to the program onto the stack is done during the linking stage.
7. Assembler can always provide the correct immediate value when translating all `la` instructions.
8. Compiling stage is the one most often responsible for code optimization.

1	2	3	4	5	6	7	8

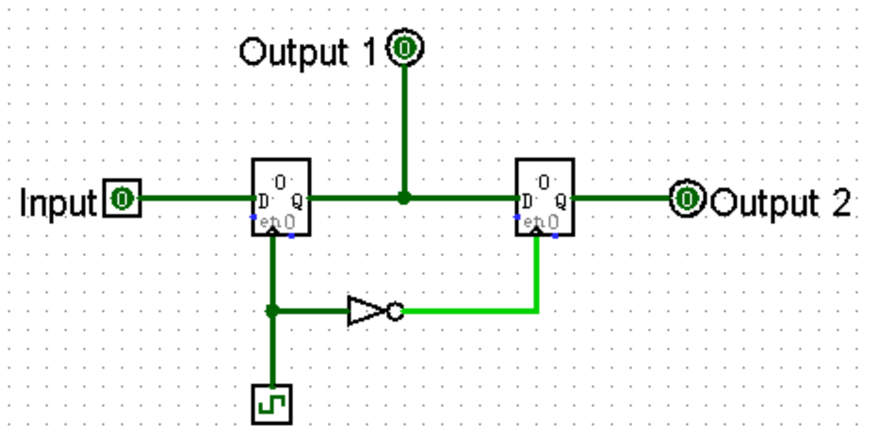
9. **Logic**

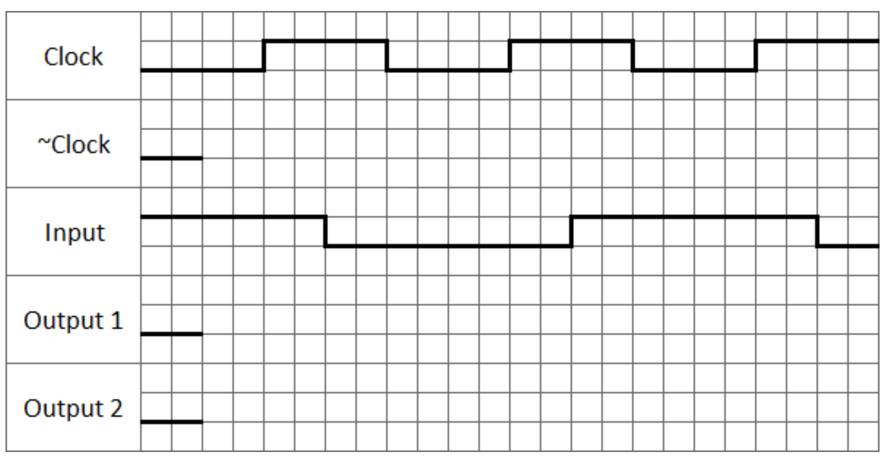
5 (a) The circuit shown below can be simplified. Please **write** down the boolean expression that exactly corresponds to the circuit shown (no simplification). Then simplify this prepossession step by step, applying one rule at a time. Then **draw** the circuit according to the simplified boolean expression using the minimum number of **one-** or **two-input** logic gates.

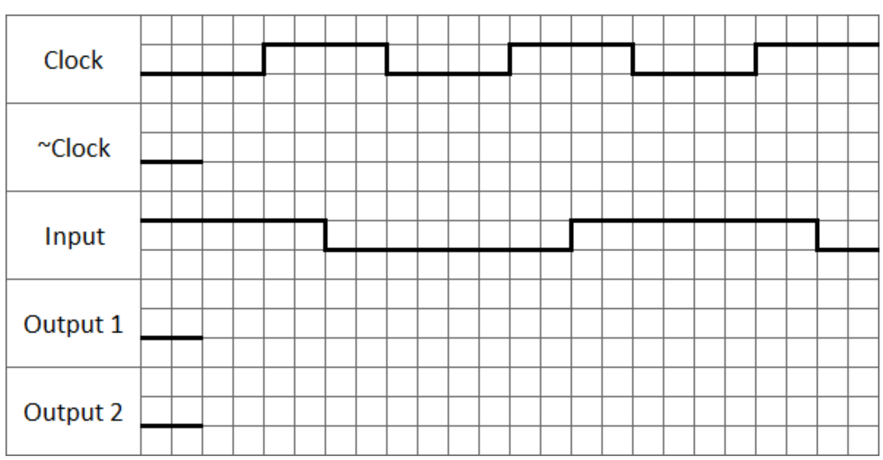


10. **SDS**

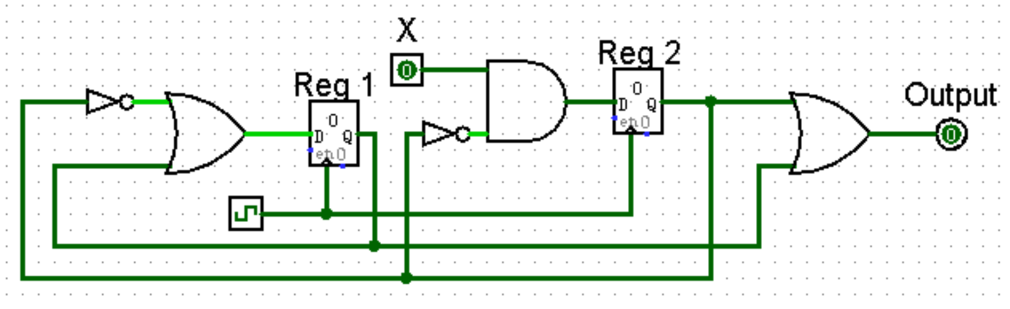
[6] (a) Draw the Timing Diagram for the circuit below. NOT gates have a 2 ns propagation delay. For each register, the clk-to-q delay is 2 ns and setup time is 2 ns. The clock period is 8 ns, and each grid in the following diagram is a unit of 1 ns. The initial values of clock and output are given in the diagram. Use any of the two empty graphs to put in your answer (so you can re-do it). Clearly **mark your final answer** if you use more than one graph!







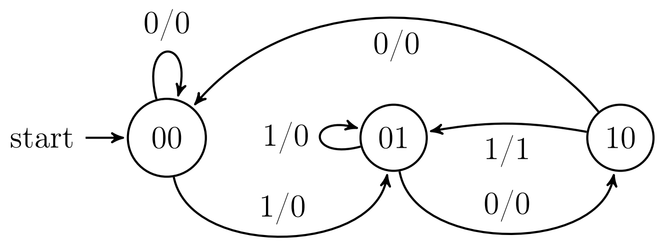
4 (b) Consider the following circuit. Assume the clock has a frequency of 50 MHz, all gates have a propagation delay of 6 ns, X changes 10 ns after the rising edge of clk, Reg1 and Reg2 have a clk-to-q delay of 1 ns.



What is the **longest possible setup time** such that there are no setup time violations?

What is the **longest possible hold time** such that there are no hold time violations?

11. **FSM**



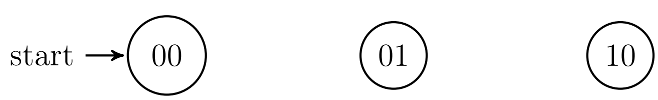
3 (a) Fill in the truth table for the FSM above.

state bit1	state bit0	input	next state bit1	next state bit0	output
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			

1 (b) What does the given FSM output with the input bit string '0100101010'?

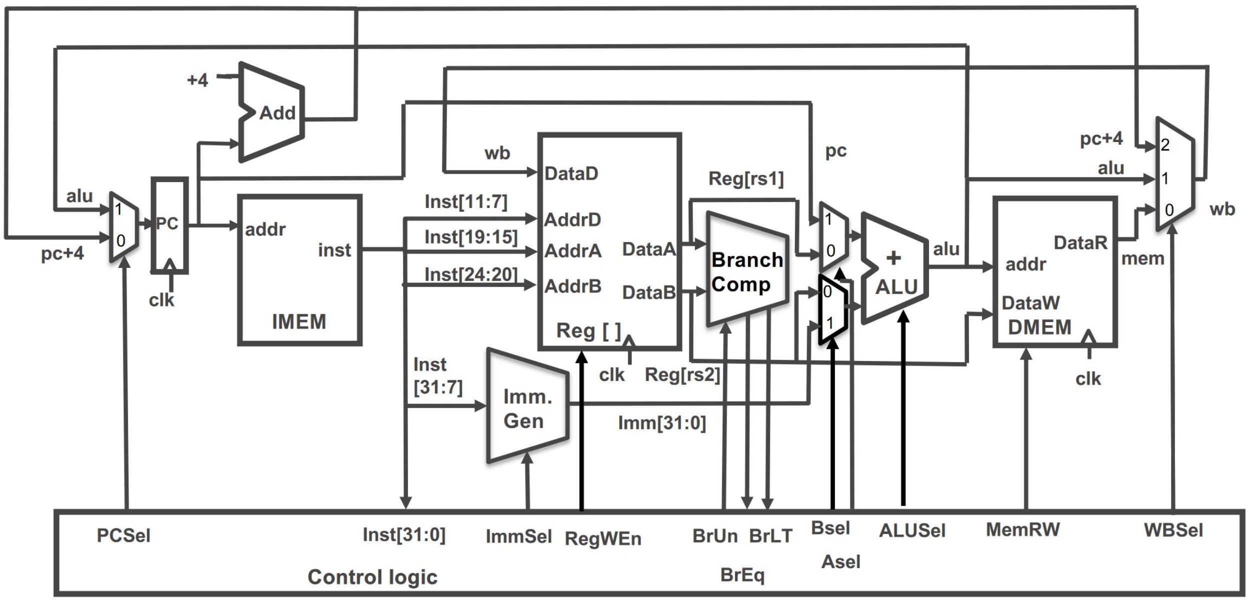
1 (c) What does the given FSM implement (Describe when the FSM will output 1)?

2 (d) Draw a FSM that outputs 1 when it receives two or more successive '0'.



12. **RISC-V Datapath**

Here is the datapath we learnt from class:



2 (a) Assume our single-cycle CPU works in 1Ghz, fill in the two blanks.

Stage	IF		EXE	MEM	WB
Time Cost(ps)	200		350	170	130

2 (b) Which of following instructions involves all stages of execution?

A. addi B. jalr C. lw D. auipc

3 (c) Assume t3 = 0x8fffffff, t4 = 0x0fffffff. Write down control signals for **blt t3, t4, label**. Please use * to indicate that what this signal is does not matter.

PCSel	ImmSel	RegWEn	BrUn	BrEq	BrLT	ASel	BSel	ALUSel	MemRW	WBSel