Computer Architecture I Mid-term Exam 1

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

E-Mail prefix: _____

Question	Points	Score
1	1	
2	16	
3	8	
4	12	
5	17	
6	14	
7	9	
8	15	
9	0	
Total:	92	

- This test contains 18 numbered pages, including the cover page, printed on both sides of the sheet.
- We will use GradeScope for grading, so only answers filled in the blank or in the brackets will be marked.
- Use the provided draft paper for calculations and then copy your answer to the exam paper.
- Please turn **off** all cell phones, smartwatches, and other mobile devices. Remove all hats and headphones. Put everything in your backpack. Place your backpacks, laptops and jackets out of reach.
- Unless told otherwise always assume a 32-bit machine.
- The total estimated time is 120 minutes.

- You have 120 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one cheat sheet (A4-sized, double-sided) of handwritten notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank or brackets, please fit your answer within the space provided.
- Do **NOT** start reading the questions/open the exam until we tell you so!

[1] 1. First Task: 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 18 times).

2. **MISC.**

[2] (a) True or False? On-chip cache has bigger capacity compared with the main memory because they are physically closer to the CPU. ( )

Solution: False.

[2] (b) Amdahl's Law: Assume you are given a program. If you are asked to parallelize its execution to achieve 5× speedup, what is the theoretical minimum fraction of the part in the program that could be parallelized?

Solution: 80% or 4/5.

[2] (c) Select the stage where the offset of a **`bge`** instruction is computed. ( )

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

Solution: B

[2] (d) In RV32I, how many data are loaded from the main memory respectively (excluding the sign-extension bits) by the instructions **`lw`** and **`lh`**? Select all that apply. ( )

A. 4 bits and 2 bits
B. 32 bits and 16 bits
C. 32 bytes and 16 bytes
D. 2 bytes and 1 byte
E. 4 bytes and 2 bytes
F. 4 bytes and 1 byte
G. None of the above

Solution: B and E. (0 for all the other cases)

[2] (e) Select below all the true statements. ( )

A. A Linux operating system cannot run on a RISC-V ISA-based computer.
B. RV32I assembly program cannot be executed on an ARM ISA-based machine directly.
C. A Linux executable file cannot run natively on the other operating systems.
D. A Python program can execute on any operating systems based on any ISAs given an appropriate Python interpreter.

Solution: B, C and D. (0.5 for not having A, 0.5 for having B/C/D each)

2 (f) True or False. Subtraction is performed on two 8-bit unsigned numbers: $(00001011)_2 - (01111111)_2$. This leads to an underflow. ( )

Solution: False.

2 (g) True or False. A 256-core CPU working at 1 GHz is faster than a single-core CPU working at 1.2 GHz running any programs. ( )

Solution: False.

2 (h) After running the following instructions, what is the value of **`x5`**? ( )

```
1 li t0,-5
2 li t1,5
3 sltu x5,t0,t1
```

A. 0.

B. 1.

C. **`0xFFFFFFFF`**.

D. Other values.

Solution: A.

3. **Number representation**

(a) The Meaning of Bits! Consider the following sequence of 16 bits: **`1000 0011 1110 0000`**. These bits can be interpreted in many different ways. (Tips: The powers of 2 from 2^0 to 2^{16} are as follows: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768 and 65536.)

[1] i. If we interpret these bits as a 16-bit unsigned binary integer, what is the decimal value represented by the bit sequence?

[1] ii. If we interpret these bits as a 16-bit sign-magnitude binary integer, what is the decimal value represented by the bit sequence?

[1] iii. If we interpret these bits as a 16-bit 2's complement integer, what is the decimal value represented by the bit sequence?

[2] iv. **`bfloat16`** format is used for efficient deep neural network computations. It represents a floating-point number in the same way as a single precision floating-point number except the number of bits used for the mantissa. **`bfloat16`** has 1 sign bit, 8 exponent bits and 7 mantissa bits. A hidden 1 is assumed. Then what is the decimal value represented by the bit sequence in **`bfloat16`**? Tips: the exponent bias is the same as a single precision floating point number since they all have 8-bit exponent. Write down the answer in the following form: $A \times 2^B$, where A and B are decimal numbers.

(b) Binary Arithmetic and Logical Operations. What will the following **`C`** code print?

[1] i. `printf("%d\n",0x00000011+0xFFFFFFF1);` _____

[1] ii. `printf("%d\n",0x00000011&0xFFFFFFF1);` _____

[1] iii. `printf("%d\n",0x00000011^0xFFFFFFF1);` _____

Solution:

a.
1. 33760
2. -992
3. -31776
4. -1.75×2^{-120}

b.
1. 2
2. 17
3. -32

4. C basics

```c
#include <stdio.h>

typedef struct {
  unsigned char read : 3;
  unsigned char write : 3;
  unsigned char execute : 3;
} user_permission;

typedef struct {
  user_permission owner;
  user_permission others;
} file_permission;

int main(void) {
  printf("Size of pointer: %u\n", sizeof(void *)); // 1
  printf("Size: %u\n", sizeof(user_permission)); // 2
  printf("Size: %u\n", sizeof(file_permission)); // 3
  file_permission file1;
  file1.owner = (user_permission){1, 1, 1};
  printf("Owner Read: %u\n", file1.owner.read); // 4
  printf("Others Write: %u\n", file1.others.write); // 5
  return 0;
}
```

2 (a) We compile the code with **`gcc -o main main.c -m32`**, and assume it compiles successfully. What will be the output of the first **`printf`**?

A. 1 C. 4 E. 32
B. 2 D. 8 F. 64

Solution: **C**: The size of a pointer is 4 bytes in a 32-bit system.

3 (b) The precise layout of a **`struct`** type is crucial to assemble and disassemble data packets and avoid memory waste. By **`unsigned char read : 3`**, we specify a structure field **`read`** as a bit field, which occupies exactly 3 bits instead of the size of the specified data type, so do bit fields **`write`** and **`execute`**. These consecutive bit fields are then packed into a larger storage unit in the structure. For simplicity, we assume that the size of the **`struct`** is decided by the minimum number of the specified data type (**`char`** in this case) that can fit all the bit fields (Tips: we consider that **`char`** type is 8-bit). Moreover, by convention, consecutive fields occupy consecutive bytes within the structure when bit field is not specified. What are the outputs of the second and third **`printf`** functions?

A. 1, 2
B. 2, 4
C. 3, 6
D. 3, 8
E. 3, 11
F. 3, 16
G. 9, 18
H. 9, 3

Solution: **B**: The minimum memory allocation for (`char`) variables is 1 byte. According to the assumption, `struct user_permission` occupies 2 bytes (2 `char`) to fit 3 3-bit numbers (in total 9 bits) since size of `char` is 1 byte by `C` standard. It implies that the size of `struct file_permission` is 4 bytes since it contains two `struct user_permission`.

3 (c) There is a bug in the code. Please identify and explain it.

Solution: The `others` field of `file1` is not initialized before use. It should be initialized before being accessed.

4 (d) The following code is compiled, and an executable file "main.out" is produced. Execute "./main.out Make CS110 great again!" in the terminal. Write down the content that will be printed.

```c
#include <stdio.h>
int main(int argc, const char *argv[]) {
   printf("argc = %d\n", argc);
   for (int ndx = 0; ndx != argc; ++ndx)
      printf("argv[%d] --> %s\n", ndx, argv[ndx]);
   return 0;
}
```

Solution:

```
argc = 5
argv[0] --> ./main.out
argv[1] --> Make
argv[2] --> CS110
argv[3] --> great
argv[4] --> again!
```

1 for `argc = 5`; 1 for `argv[0] --> ./main.out`; 0.5 for the others.

5. RISC-V

10

(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
```

Solution:

```
// 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

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)
```

4 (b) According to the RISC-V standard extension, we can compress some 32-bit RV32G instructions into 16-bit version (RVC instructions) for memory space-saving. In this question you only need to consider 2 RVC instructions below. The 2 instructions and their encoding format (from the MSB to the LSB) are shown in the table below. Note that for **`beq`** compressed instruction, only the last 3 bits of **`rs1`** is used as its **`rs1'`** field.

Instr.	funct4	rd/rs1	rs2	opcode
# of bit	4	5	5	2
add rd rd rs2	0b1001	dest≠0	src≠0	0b10

Instr.	funct3	imm.	rs1'	imm	opcode
# of bit	3	3	3	5	2
beq rs1 x0 offset	0b110	offset[8|4:3]	src[2:0]	offset[7:6|2:1|5]	0b10

```
1        main:
2          mul a0 a1 a3
3          add a0 a0 a2
4          addi a0 a0 10
5          beq a0 zero main
6          ...
```

Translate the branch instruction in line5 to 32-bit machine code (RV32I instruction) in **hexadecimal**.

Solution: `beq a0 zero main __0xFE050AE3_`

Now, to save memory, we compress the instructions in line3 and line5. Please write down their **compressed** machine code in **hexadecimal** (Tips: note that the compressed instructions occupy **2 bytes** instead of 4 bytes):

Solution: `add a0 a0 a2 ___0x9532_____`

`beq a0 zero main ___0xD97E_____`

Note that the instructions are now 16 bits, or 2 bytes. So when we calculate the immediate field of `beq`, we count the compressed instruction as 2, not 4.

3 (c) Translate the instructions below to machine code in **hexadecimal**.

Solution: `li a0 2048 ___0x00001537__;__0x80050513_`

`jal ra -16 _0xFF1FF0EF___`

6. **Calling conventions & memory management**

Assume a `struct` type defined as follows has the following layout, `c` at address 0 (or address x) and `next` at 4 (or address $x + 4$) because of alignment. The size of `struct node` is then 8-byte.

```
struct node {unsigned char c, struct node *next};
```

A function is defined as follows.

```
struct node * foo(char c){
  struct node *n;
  if (c < 0) return 0;
  n = malloc(sizeof(struct node));
  n->next = foo(c - 1);
  n->c = c;
  return n;
}
```

14 (a) Please complete the RV32I assembly implementing the `foo` function according to the comments and instructions below following **calling conventions**. Since we are calling other functions, assume local variable `c` is put in `s0` and `n` in `s1` so that we can still use these values after function call.

Solution:

```
foo:
  addi sp, sp, _-12_ ///prologue. Tips: the minimum stack
     space required for saving registers, disregard stack
     alignment requirements, i.e., the stack can be of any
     size (2 points, 1 for correct number (consistent with
     your next question's choice, e.g. if you select "EF" in
     the next question and fill -8 in this blank, you are
     correct!), 1 for positive/negative)

  sw __E,F and G__          // please select below all the
     registers that must be saved here in the stack before
     function call. (2 points, 0.25 for each option A-G)
```

A. `a0`.

B. `a1`.

C. `t0`.

D. `t1`.

E. **s0**.

F. **s1**.

G. **ra**.

H. **sp**.

```
  blt a0, x0, foo_true // if c<0, jump to foo_true to return
foo_false:
  mv s0, a0            //put c in s0 for further use

  li a0, __8__      //fill in the blank here to pass the
     parameter to the malloc function that to be called (1
     point)
  call malloc       //function call
  mv s1, a0         //put n in s1 for further use

  addi _a0_, _s0_, _-1_ //calculate c-1 and pass the
     parameter to foo function for recursive call (3 points)
  call foo           //recursive function call

  __sw__a0,_4(s1)_//write the return value into n->next (2
     points)

  __sb__s0,_0(s1)_____//write c into n->c (Tips: c is char
     type.) (2 points)
  mv a0, s1          // return n in a0
  j foo_exit         // Jump to epilogue for returning to the
     caller function of foo
foo_true:
  add a0, x0, x0    //return 0 if c<0
foo_exit:           //epilogue for returning
  lw                //load all the registers we have saved in
     prologue, i.e., all the registers you have chosen in
     the previous question. So we skip this.

  addi sp, sp, _12_//restore the stack pointer (2 points)
  ret               //return to the caller function
```

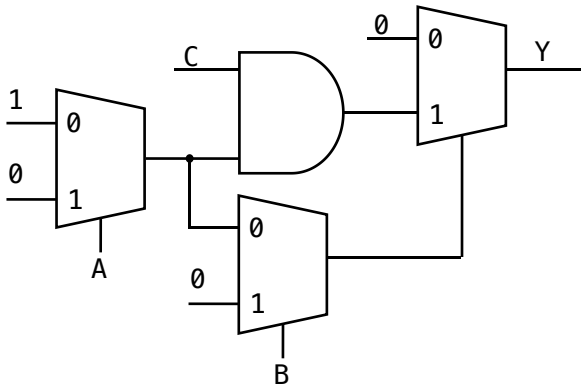
7. **Logic**

2 (a) **(Multiple Choice)** Which of the following statement(s) are(is) true about boolean algebra? ( )

A. $X + YZ = (X + Y)(X + Z)$
B. $(X + \overline{Y})X = X + X\overline{Y}$
C. $XY + X = X$
D. $\overline{XY} = \overline{X} + \overline{Y}$

Solution: ABCD. 2 for exactly the same with the answer. 0 for all the other cases.

4 (b) The following circuit is composed of several basic logic gates and 2-to-1 multiplexers. Please write down the truth table of the circuit below.



Solution: 0.5 mark for each line.

Truth Table

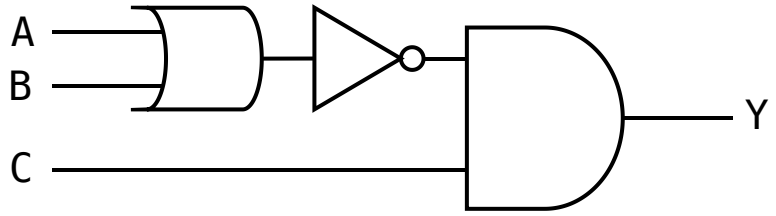
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

1 (c) Write down the logic expression that implements the truth table using sum of minterm.

Solution: $Y = \bar{A}\bar{B}C$. This is the only form of the logic expression using minterm. 0 mark for all the other cases.

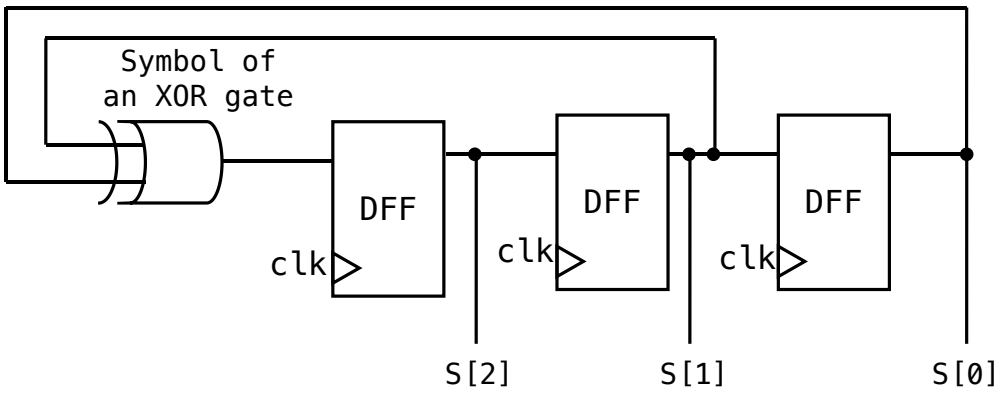
2 (d) Build a logic circuit that uses only 2-input **AND**, 2-input **OR** and **NOT** gates implementing the same logic above. Use as less logic gates as possible.

Solution: Optimal solution for this is shown below for full mark using the required logic gates. Direct implementation of $Y = \bar{A}\bar{B}C$ get 1 mark for no simplification. You also lose point(s) if not using the required gates. The other cases for 0.



8. **SDS**

4 (a) Below shows a synchronous circuit called linear feedback shift register (LFSR), consisting of one or more **XOR** gates and several DFFs. It has been widely used for generating pseudorandom numbers. It can be modeled by a finite state machine (FSM) like the other synchronous circuits, however, without any input signals. Given the current state S_{k-1}, fill in the truth table of its next state S_k. S is a 3-bit signal.



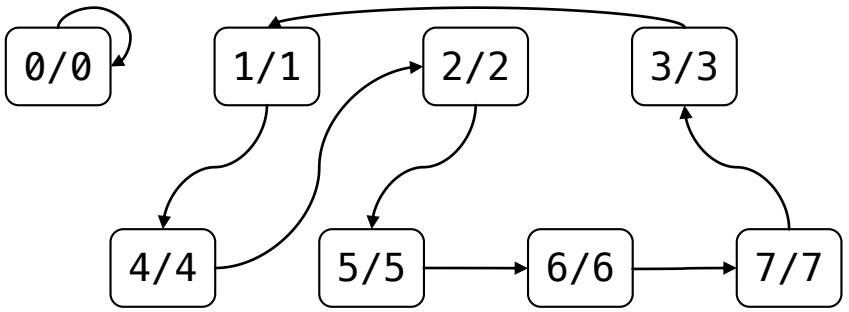
Solution: Each line for 0.5.

Truth Table

$S[2]_{k-1}$	$S[1]_{k-1}$	$S[0]_{k-1}$	$S[2]_k$	$S[1]_k$	$S[0]_k$
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	1	1

8 (b) For the above FSM, we use the unsigned number $(\mathrm{S}[2]\mathrm{S}[1]\mathrm{S}[0])_2$ to represent its state and output. Please complete the state transition diagram below. Tips: This FSM has no input, and we do not put the transition condition on the transition edges or lines. Also, we use "0/0" to denote that the FSM is currently at state 0 and its output is 0, respectively.

Solution: Each line for 1 mark.

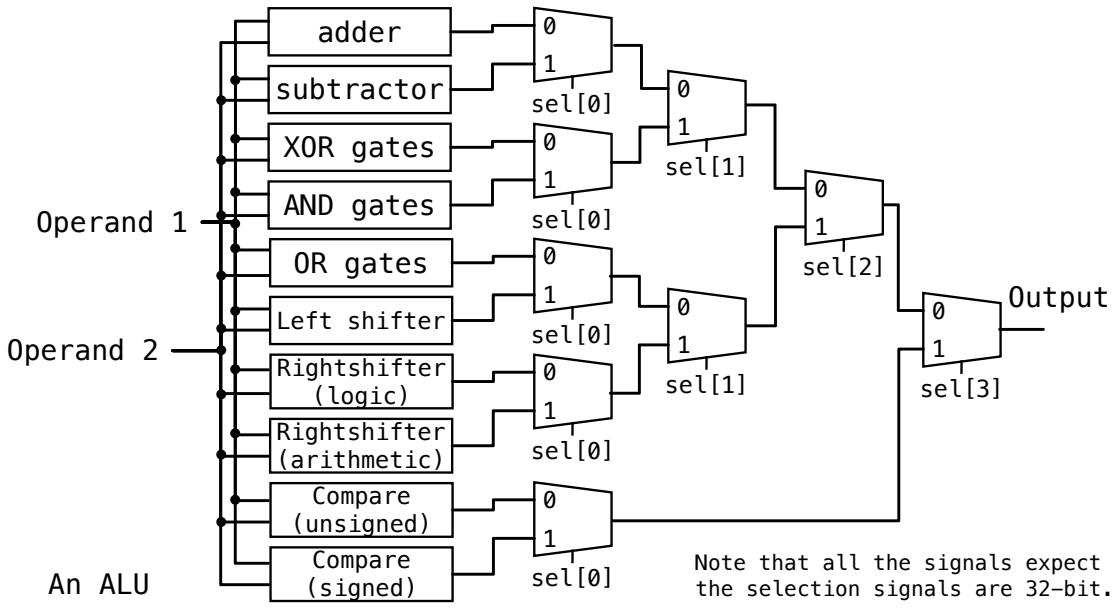


3 (c) The setup time of a DFF is 1 ns, the delay of an **`XOR`** gate is 2 ns, and the **`clk-to-q`** delay of the DFF is 1 ns. Compute the maximum frequency of this circuit. (We ignore the delay of the lines and ignore all the other non-ideal effects such as clock skews, etc.)

Solution: Critical path = XOR delay + DFF setup time + DFF clk-to-q delay = 4 ns (2 marks)

Max. frequency = 1/Critical path = 250 MHz or 0.25 GHz (1 mark, if you only have this result but not calculating critical path, you also get full marks.)

9. **Datapath** Below is a possible implementation of an ALU in a CPU that supports RV32I arithmetic and logic instructions. Assume that the rectangles are logic blocks that implement the corresponding functions described by the text. Please indicate the selection signals (**in binary**) of the multiplexer array so that the output of the corresponding logic block is selected when certain instructions are executed. Tips: An "X" can be used to represent that I do not care what this bit is. For example, "X100" means that it can either be "0100" or "1100".



Solution:

`addi x2, x2, -1`	`sel[3:0]=__0000__`
`sub x2, x2, x0`	`sel[3:0]=__0001__`
`sra x2, x2, x1`	`sel[3:0]=__0111__`
`sltu x2, x2, x0`	`sel[3:0]=__1xx0__`

It is fine if "x" is replaced with 0 or 1 for the `sltu` instruction.

Computer Architecture I Mid-term Exam 1

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

E-Mail ... @shanghaitech.edu.cn: _____

Question	Points	Score
1	1	
2	4	
3	14	
4	9	
5	19	
6	6	
7	10	
8	20	
9	17	
Total:	100	

- This test contains 16 numbered pages, including the cover page, printed on both sides of the sheet.
- We probably will use blackboard for grading, so only answers filled in at the obvious places will be used.
- Use the provided blank paper for calculations and then copy your answer here.
- Please turn **off** all cell phones, smartwatches, and other mobile devices. Remove all hats and headphones. Put everything in your backpack. Place your backpacks, laptops and jackets out of reach.
- Unless told otherwise always assume a 32-bit machine.
- The total estimated time is 120 minutes.

- You have 120 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use two A4 pages (front and back) of handwritten notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided.
- Do **NOT** start reading the questions/open the exam until we tell you so!

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 16 times).

Also, unless told otherwise, always assume a 32-bit machine, `int` type has 4 bytes or 32 bits, DFFs are triggered at the rising edge of the clock signal, and we use only `and, or and not` gates for combinational circuit design throughout the exam.

4 2. **Introduction [4 points]**

(1) Moore's Law: True (T) or False (F). According to Moore's Law, the number of transistors on a single chip will continue to double every two years forever. [2 points] ( )

(2) Amdahl's Law: Assume we have a task, 20% of which cannot be parallelized, what is the maximum acceleration factor can be achieved with unlimited parallelized computing resources? [2 points]

Solution: (1) F. Moore's Law is not a law, it is just a prediction. (2) 5.

$$F = \frac{1}{20\% + \frac{80\%}{P}} \tag{1}$$

When P goes to infinity, F is 5.

14 3. **Number representation [14 points]**

(1) Fill in the blanks below to show how the 32 bits can be interpreted in different ways. If a particular field has no solution, answer "N/A". [8 points]
bit pattern B = 1111 1111 0010 0100 0110 0000 0000 0000

(a) 4 sign-magnitude bytes (in decimal): _____

(b) 4 one's complement bytes (in decimal): _____

(c) 4 two's complement bytes (in decimal): _____

(d) Compute `B^(B<<5)` (in hexadecimal, ^ stands for XOR): _____

(2) Consider a base-9 number system (with nine valid digits of 0, 1, 2, 3, 4, 5, 6, 7, 8). [2 points]

(a) 188_{nine} in base 10: _____

(b) $2A5_{hex}$ in base 9: _____

(3) An **8-bit** floating-point number (FP8) has 1 sign bit, 4 exponent bits and 3 mantissa bits (with an un-shown hidden 1 for normal cases). The bias for the exponent is 7. Meanwhile, consider that we receive data with some unknown bits, and "x" is used to refer to the unknown bits. For example, if the received data are "0b0xx1", they could be "0b0001", "0b0011", "0b0101" or "0b0111". [4 points]

(a) We receive an FP8 number "0b0x1x0x1x". What is the **largest** number the sender could have sent? (in **hexadecimal**)

(b) Consider an FP8 number that is neither an NaN nor infinity nor 0, what is the smallest possible positive number the sender could have sent **as a power of 2**? Assume FP8 uses the same rule to represent a denormalized number as a single-precision floating-point number.

Solution:

(1) (a) -127, 36, 96, 0
(b) (-)0, 36, 96, 0
(c) -1, 36, 96, 0
(d) 0x1BA86000

(2) (a) 161_{ten}
(b) 832_{nine}

(3) (a) 0x77
(b) 2^{-9} ($2^{-6} \times 2^{-3}$)

4. **C basics [9 points]**

3 (a) What is the output of the following program segment? [3 points]

```
char c = 'a';
putchar(c);    /*Equivalent to print char c*/
putchar(F(c));
putchar(c);
```

Assume that the function F has been defined as follows:

```
char F(char c)
{
   c = 'f';
   return c;
}
```

Solution: afa

2 (b) Fill in the code to properly allocate memory (on the heap) for an `n` × `m` matrix `mat` of integers initialized to zeros. [2 points]

```
1 mat = (int **) calloc(n, sizeof(int *));
2 for (int i = 0; i < n; i++) {
3    mat[i] = _____
4 }
```

line 3: _____

Solution: (int *) calloc(m, sizeof(int));

4 (c) The following program performs an n-bit cyclic left-shift or bit rotation of an integer using a "while" loop. Specifically, the highest (leftmost) n bits are moved to the lowest bits, and the rest bits are left-shifted by n bits. For example, 1100 becomes 1001 after a 1-bit cyclic left-shift, and 0011 after a 2-bit cyclic left-shift. Please fill in C code to realize the function according to the comments. (**Hint**: In C language, $>>$ operator performs arithmetic right-shift or logical right-shift according to the type of the variables.)

```
1  int rotate_integer(int num, int n) {
2     int new_num = num;
3     int high_bit = 0;
4     while (___(1)__)
5     {
6        high_bit = __(2)__;/*Obtain the highest bit*/
7        new_num = __(3)__;/* Left-shift the number by 1*/
8        new_num = __(4)__;/* Put the highest bit to the lowest
            bit*/
9        num = new_num;
10       n--;
11    }
12    return new_num;
13 }
```

(1)_____
(2)_____
(3)_____
(4)_____

Solution:

```
1 n>0
2 high_bit = (unsigned int)(num&0x80000000)>>31 ;
3 new_num = new_num << 1;
4 new_num = new_num + high_bit;
```

Other reasonable solutions are also acceptable.

5. **RISC-V assembly [19 points]**

5 (a) Perform an R-type **signed** addition (add t2, t1, t0) and detect overflow. If an overflow occurs, t4 register is set to 1; otherwise, t4 is set to 0. Please use RV32I instructions (as less instructions as possible) to complete the below assembly code. (**Hint**: the sum should be less than one of the operands if and only if the other operand is negative. Feel free to use `t0~t6`. Also, please comment your code properly. Leave it blank if you do not use all the space below.)

```
add   t2, t1, t0

_____________________

_____________________

_____________________

_____________________

_____________________
addi  t3, t3, 1
beq   t4, t3, OVERFLOW
... ...
OVERFLOW: #some code to deal with overflow
```

Solution:

```
slti t5, t0, 0 #set t5 if t0<0
slt t4, t2, t1 #set t4 if t2<t1
xor t4, t4, t5 #if (t0>=0 and t2<t1) or (t0<0 and t2>=t1),
   overflow occurs and t4 is set to 1
addi t3, x0, 0 #initialize t3.
```

t1 and t2 are exchangable. t5 can be other tx.

8 (b) In this question, you will calculate the nth term of a Fibonacci sequence and then the factorial of it. The $(i+2)$th term of Fibonacci sequence is given by $f_{i+2} = f_{i+1} + f_i$. The initial values are given as $f_{-1} = 0$ and $f_0 = 1$, and the nth term refers to f_n. For example, if $n = 3$, you should obtain the result 6 (3!). [8 points] (**Hint**: The Fibonacci function first calculates the nth item and then calls "int factorial(int nth item)", which is a

recursive function. To simplify, we consider that multiplication can be implemented with just one "mul" instruction and you do not have to consider the higher/lower bits.)

```
1   main:
2      li    t0, 0
3      li    t1, 0        #the -1st term, f[-1]
4      li    t2, 1        #the 0th item, f[0]
5      li    a0, n        #input parameter n
6      li    a1, 0
7      beq   a0, t0, End  #exit when n is 0
8   Fibonacci_loop:
9      add   t3, t1, t2
10     add   t1, t2, x0
11     ___________________
12     add   a1, t3, x0
13     addi  t0, t0, 1
14     ___________________ #branch Fibonacci_loop when unfinished
15     jal   ra, Factorial
16  End:                  #print and exit, etc. stuff
17     ecall ...
18  Factorial:
19    addi  sp, sp, -8
20    sw    ra, 4(sp)
21    sw    a1, 0(sp)
22    addi  t4, a1, -1
23    bge   t4, x0, Factorial_loop
24    addi  a1, x0, 1     #the last recursive call
25    addi  sp, sp, 8
26    jalr  x0, ra, 0
27  Factorial_loop:
28    addi  a1, a1, -1
29    ___________________ #recursively call Factorial
30    addi  t5, a1, 0
31    lw    a1, 0(sp)
32    lw    ra, 4(sp)
33    addi  sp, sp, 8
34    ___________________ #perform multiplication
35    jalr  x0, ra, 0    #ret. to where last recursive call left
```

Fill in the missing code below.

line 11: _____

line 14: _____

line 29: _____

line 34: _____

Solution:

line 11: add t2, t3, x0 or mv t2, t3

line 14: blt or bne t0, a0, Fibonacci_loop

line 32: jal Factorial or jal ra, Factorial

line 37: mul a1, a1, t5

When n=3, fill in the stack space below with the stored value during the execution of the above code. The stack space grows downward. For return address, use the corresponding line number to represent the address of an instruction/label. (**Hint:** fill in all the stack space which has been used. Leave it blank if you do not use all the space below.)

Stack pointer to here when entrance

Solution: From top to down, the values are 16 (or 17), 3, 30, 2, 30, 1, 30, 0.

4 (c) Translate the instructions below to machine code written in **hexadecimal**. [4 points]

line 12: `add a1, t3, x0` _____

line 35: `jalr x0, ra, 0` _____

Solution:
line 12: 0x000E05B3

line 38: 0x00008067

2 (d) Calculate the target address in **hexadecimal** that `jalr` jumps to. [2 points]

```
lui x5, 0xFFFFF

jalr ra, x5, 0x123
```

The target address is _____

Solution:
0xFFFFF123

6. **Call convention/linker/loader/assembler [6 points]**

2 (a) Which register(s) is(are) used for returning values from a function in RISC-V calling convention?_____

A. a1
B. s0
C. ra
D. a0

Solution: A & D

2 (b) Which of the following statement(s) are(is) true?_____

A. Caller-saved registers includes registers used for storing local variables or passing function arguments, while callee-saved registers includes registers used for storing global variables.
B. Caller-saved registers are preserved across function calls, while callee-saved registers may be modified by the callee.
C. Caller-saved registers are saved by the caller before calling the callee if required, while callee-saved registers are saved by the callee.

Solution: C

2 (c) When are all the machine code bits decided for the following assembly instructions:

(1) `sub x2, x2, x3`
(2) `jal x1, malloc`

_____

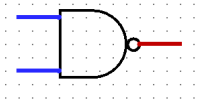
A. (1) & (2) after compilation
B. (1) after assembly, (2) after loading
C. (1) & (2) after assembly
D. (1) after assembly, (2) after linking

Solution: D

7. **Logic [10 points]**

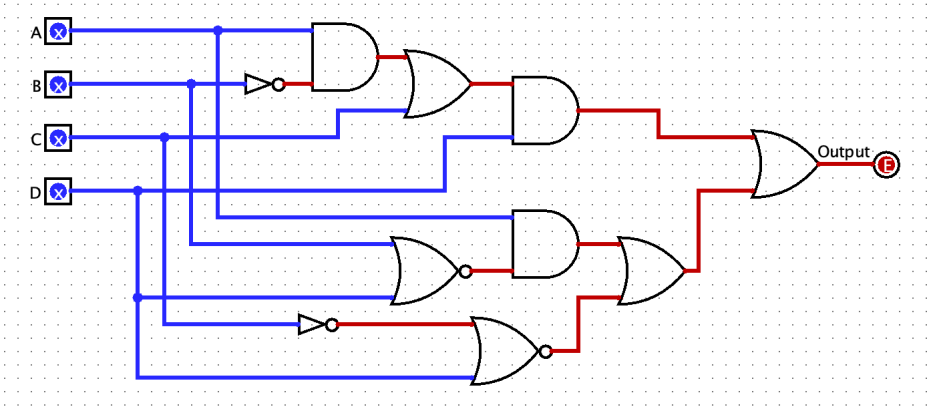
1 (a) What is the name of the following gate:_____

A. AND
B. OR
C. NAND
D. NOR



Solution: C

6 (b) Please write down the truth table of the circuit below and draw its Karnaugh map (A&B as a group; C&D as a group).



Solution: You can simplify the boolean expression first and obtain the truth table, which is faster.

$$(A\overline{B}+C)D+A(\overline{B+D})+\overline{\overline{C}+D}=A\overline{B}D+CD+A\overline{BD}+C\overline{D}=A\overline{B}+C$$

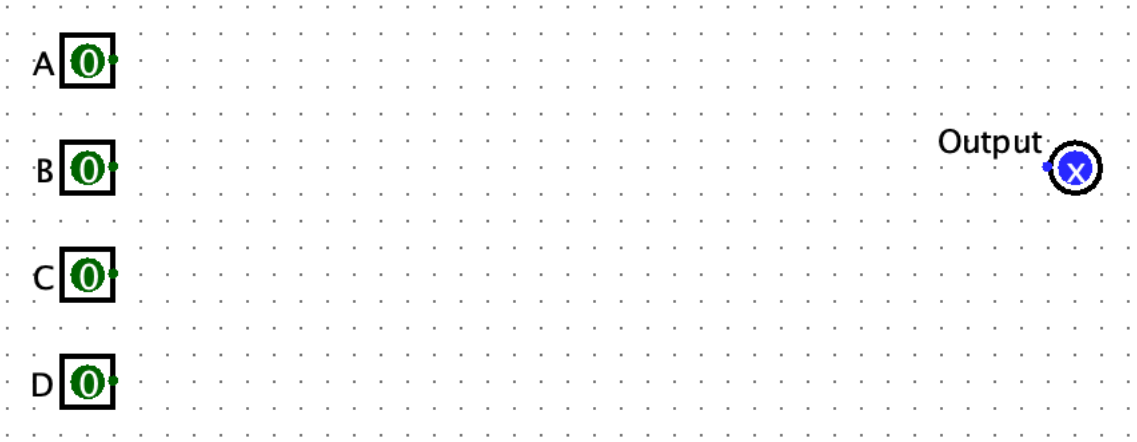
Truth table:

A	B	C	D	Output
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

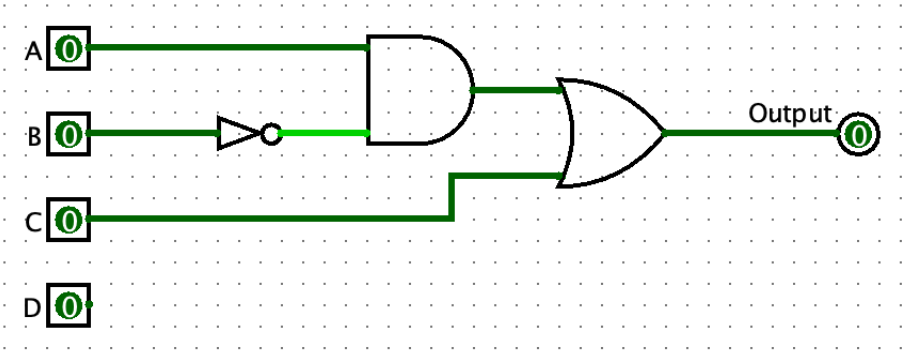
Karnaugh map:

CD \ AB	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	1	1	1

3 (c) Simplify the circuit using Karnaugh map and re-design the circuit using the least number of gates. You may use only `AND, OR and NOT` gates.



Solution: Optimal solution for this is



8. **Finite State Machine [20 points]**

1 (a) Below shows a state transition diagram of a finite state machine (FSM) with 4 states. What is the type of the FSM?

A. A Moore machine.

B. A Mealy machine.

Solution: B.

2 (b) Assume the input bit sequence to the FSM is `10011010`, the output is _____.

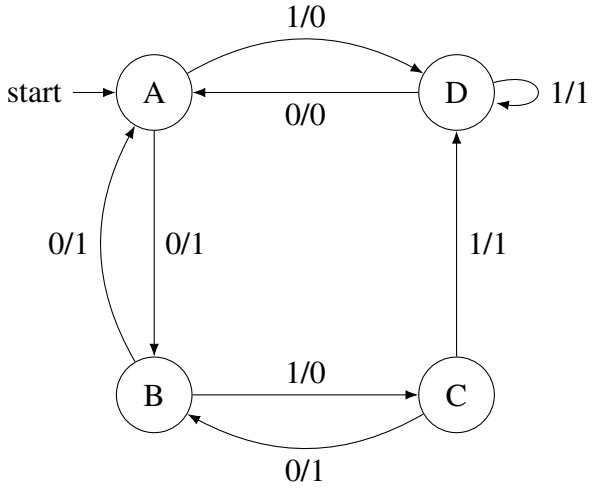
Solution: 00101000.

2 (c) Which state does the FSM arrive at last?_____

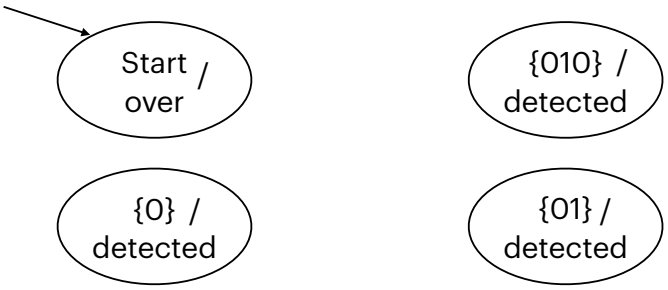
A. A

B. B
C. C
D. D

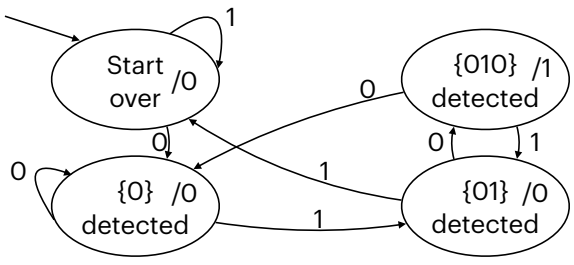
Solution: A.



4 (d) Build a **Moore** FSM model to detect "010" pattern in a bit sequence (use overlapping, i.e., the tail 0 of "010" can be considered as the head 0 for the next detection). The states are given below. Please complete the state transition diagram by adding the transitions, transition conditions and output for each state. [4 points]



Solution:



4 (e) Assign "00" (0) to represent state "Start over", "01" (1) to represent "{0} detected", "10" (2) to represent "{01} detected" and "11" (3) to represent "{010} detected". Write down

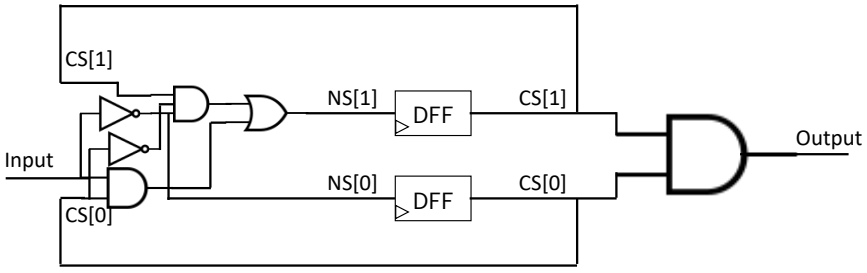
the truth table for the next-state and output logic. We use "CS" to represent current state and "NS" for next state.

Solution:

CS[1]	CS[0]	input	NS[1]	NS[0]	output
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	1	0	1

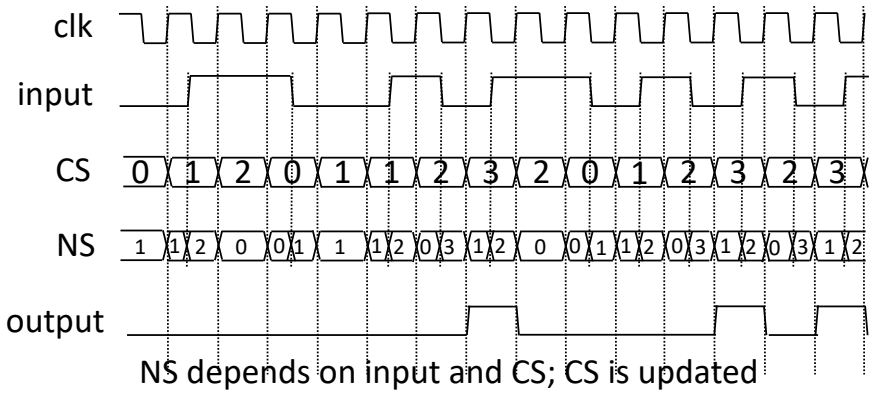
3 (f) Complete the circuit below for the "010" sequence detection task using the truth table you just wrote.

Solution:



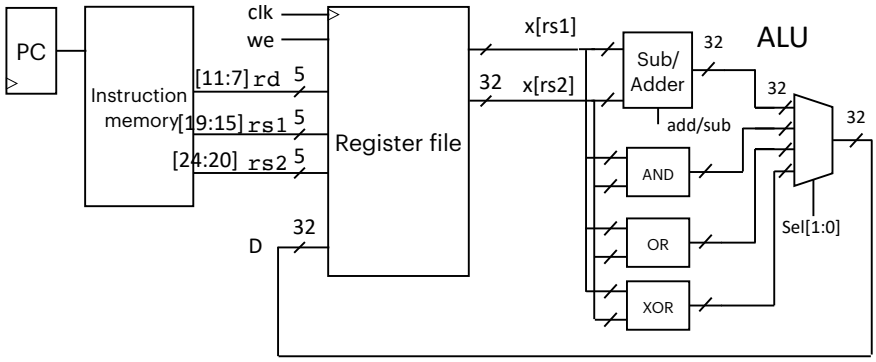
4 (g) Draw the timing diagram given the clock signal and input below. We ignore the non-ideal effects, and integers (use signal grouping) are used to represent the states.

Solution:



9. **RISC-V datapath [17 points]**

Below is the datapath we learned from class for some RISC-V R-type instructions:



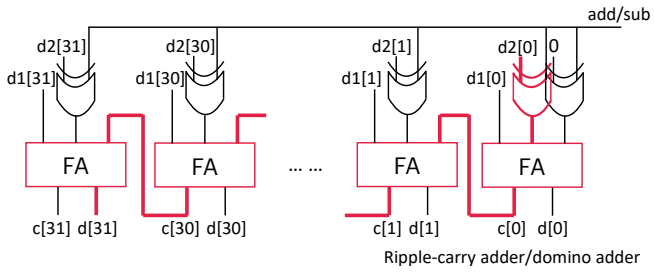
We will estimate the maximum clock frequency step by step. By "delay", we refer to **propagation delays**, unless stated otherwise. The delay of each element is shown in the table below.

Circuit	2-input AND	2-input OR	2-input XOR	DFF clk-to-Q
Delay (ps)	10	15	50	20
Circuit	2-to-1 multiplexer	5-32 decoder	1-bit full adder	3-input AND gate
Delay (ps)	15	100	30	15

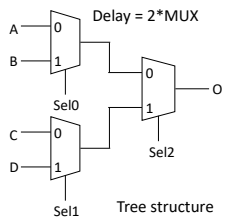
6 (a) **Delay of the ALU [6 points]**

The ALU consists of functional units such as the adder/subtracter (circuit shown below) and different types of logic gates, and a 4-to-1 multiplexer built from the 2-to-1 multiplexer using tree structure. Calculate the delay of the adder/subtracter and then the maximum delay of the ALU.

Solution:



add/sub delay $= XOR_{delay} + 32 * (\text{1-bit bull adder})_{delay} = 50 + 960 = 1010$ ps.

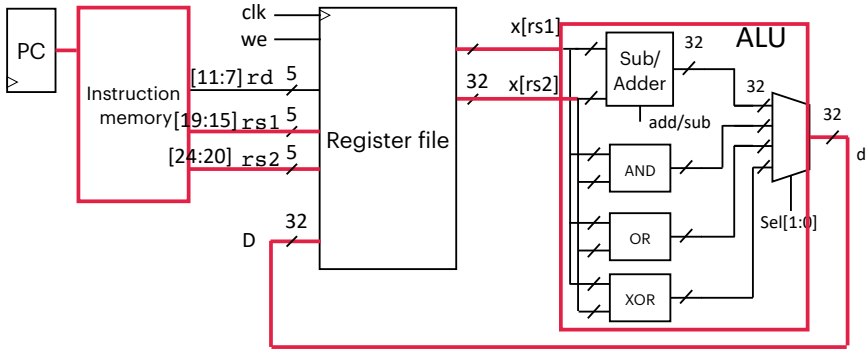


4-to-1 MUX delay = 2*2-to-1 MUX delay = 30 ps.

ALU max delay = 4-to-2 MUX delay + add/sub delay = 1040 ps.

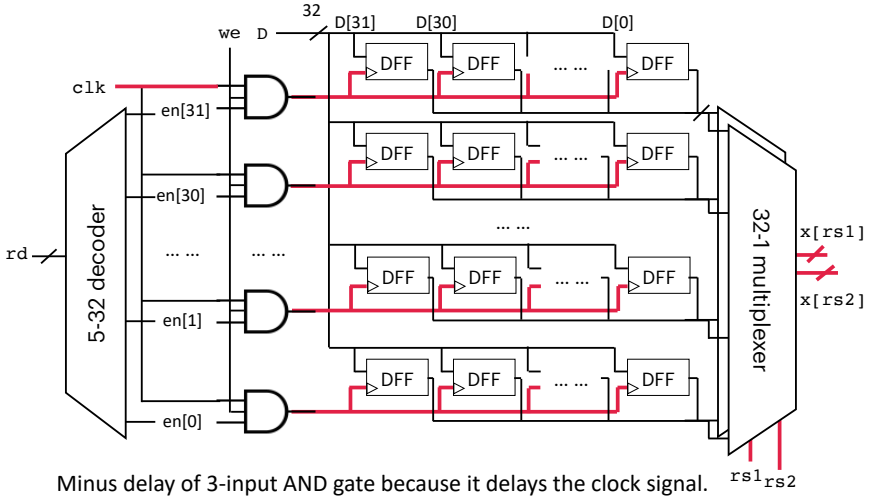
9 (b) **Delay of the datapath [9 points]**

The detailed circuit of the register file is shown below. Assume all the DFFs has a setup time of 50 ps. Indicate the elements of which the delay should be included to calculate the minimum clock cycle and the corresponding delay numbers. After that, calculate the maximum frequency of this datapath. Again, assume the 32-to-1 multiplexer is built from 2-to-1 multiplexer by the tree structure. Instruction memory delay is 80 ps.



Solution:

Critical path = clock-to-q (PC) + Imem delay + Reg. file delay + ALU delay + setup time (Reg. file).



Minus delay of 3-input AND gate because it delays the clock signal.

Reg. file delay in total = 32-1 MUX delay − 3-input AND delay
= 5 × 2-1 MUX delay − 3-input AND delay = 60 ps.

Minus the delay of the 3-input AND gate because it delays the clock signal, resulting in extra time for DFF setup in the register file.

Critical path = 20 + 80 + 60 + 1040 + 50 = 1250 ps.

Max frequency = 1/(Critical path delay) = 800 MHz.

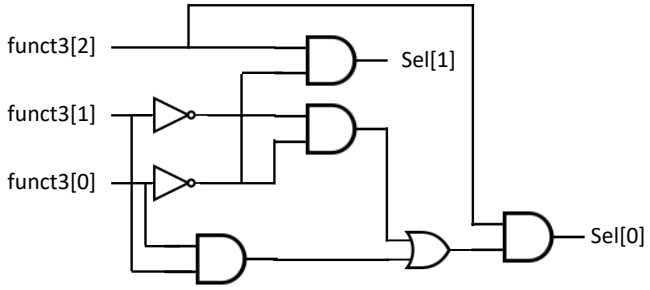
2 (c) Use the "funct3" field of the R-type instructions to generate the multiplexer select signal for the ALU. Assume the multiplexer selects add/sub, AND, OR and XOR results when the select signal is 00, 01, 10, 11, respectively.

Solution:

	funct3[2]	funct3[2]	funct3[2]	Sel[1]	Sel[0]
add/sub	0	0	0	0	0
AND	1	1	1	0	1
OR	1	1	0	1	0
XOR	1	0	0	1	1

$$Sel[1] = funct3[2] \cdot \overline{funct3[0]}$$

$$Sel[0] = funct3[2] \cdot (funct3[1] \cdot funct3[0] + \overline{funct3[1]} \cdot \overline{funct3[0]})$$



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.

```
1 quarter q1, q2, q3, c, d;
2 q1 = -0.25;
3 q2 = -4.0;
4 q3 = 0.125;
5 c = q1 + (q2 + q3);
6 d = (q1 + q2) + q3;
```

`q1` in binary _____

`q2` in binary _____

`c` in decimal _____

`d` in decimal _____

4. C Basics

5 (a) Memory of C

```
1 #include <stdlib.h>
2
3 int main() {
4     static int p = 5;
5
6     char *str = _____;
7     /* some other codes, and you can skip it. */
8     return 0;
9 }
```

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 $1 \to 2 \to 3 \to$ `NULL` to $3 \to 2 \to 1 \to$ `NULL`. (You may not need all of the lines)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Definition for singly-linked list. */
5 struct ListNode {
6     int val;
7     struct ListNode *next;
8 };
9
10 /* Given the head of a singly linked list, reverse
11     the list, and return the head of reversed list.*/
12 struct ListNode *reverse_list(struct ListNode *head) {
13     struct ListNode *prev = NULL;
14     struct ListNode *curr = head;
15     struct ListNode *next = head;
16     while (curr) {
17         next = next->next;
18
19         ____________________________
20
21         ____________________________
22
23         ____________________________
24
25         ____________________________
26
27         ____________________________
28
29         ____________________________
30     }
31     return prev;
32 }
```

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.

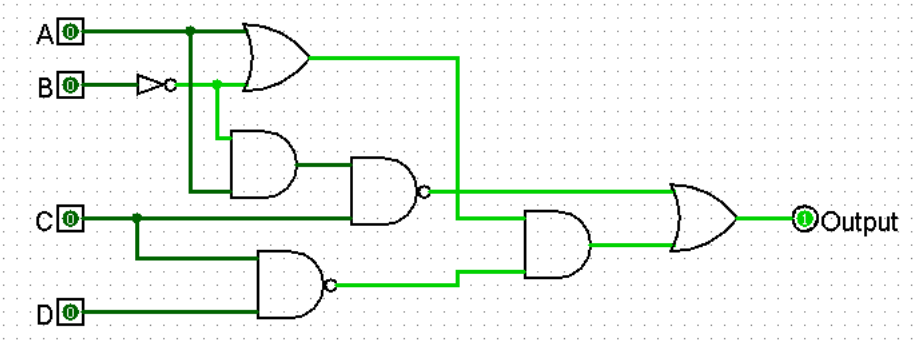
1. Pseudo instructions are not allowed in the output of compiler.
2. Statically-linked libraries are incorporated into the program during the load stage.
3. Dynamically-linked libraries are incorporated into the program during the link stage.
4. The interpreted program (like Python) runs way faster than 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

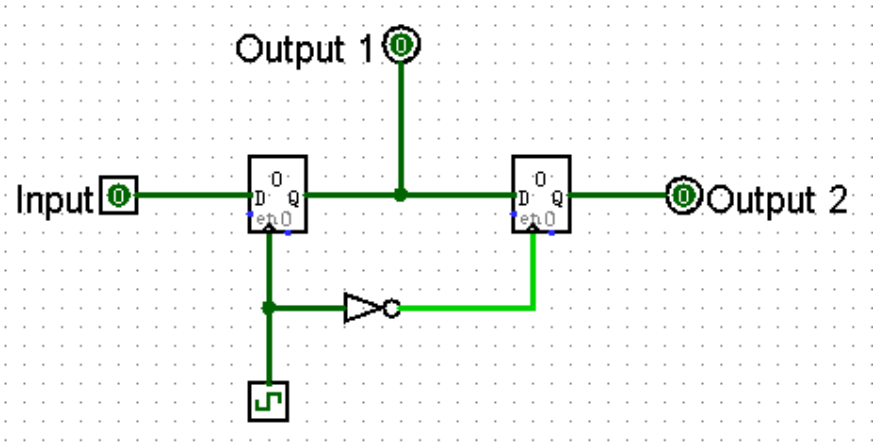
9. **Logic**

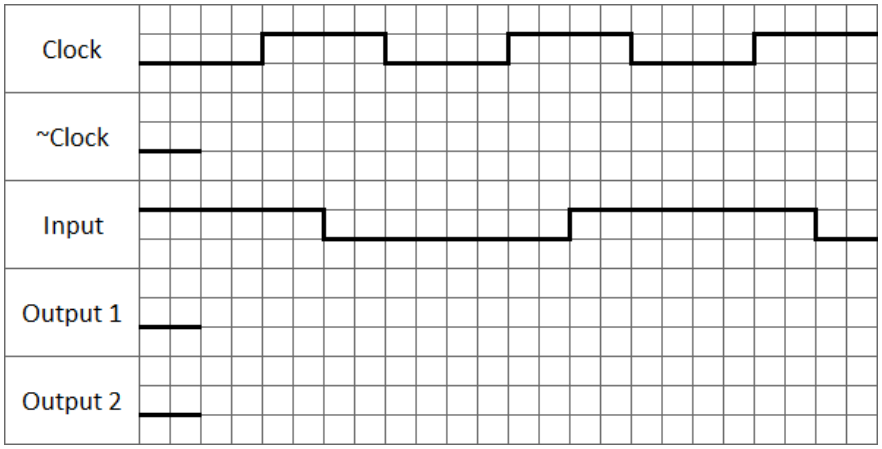
5 (a) The circuit shown below can be simplified. Please **write** down the boolean expression that exactly corresponds to the circuit shown (no simplification). Then simplify this prepossession step by step, applying one rule at a time. Then **draw** the circuit according to the simplified boolean expression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

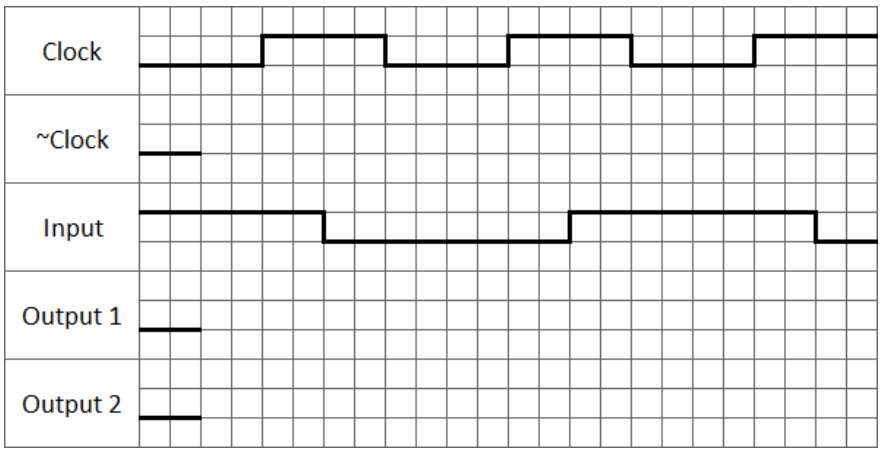


10. **SDS**

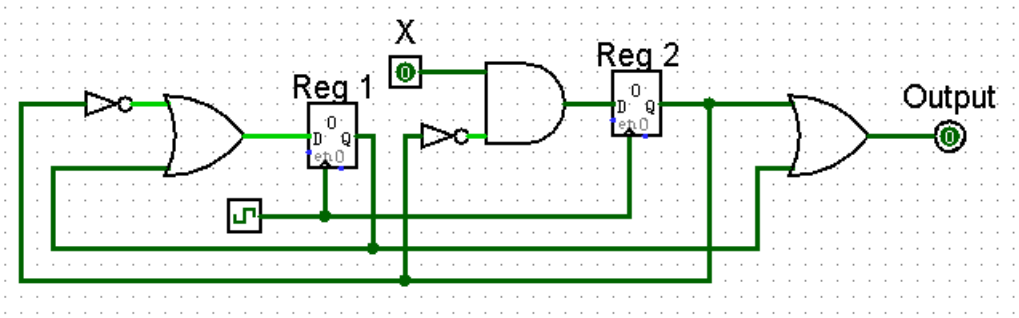
6 (a) Draw the Timing Diagram for the circuit below. NOT gates have a 2 ns propagation delay. For each register, the clk-to-q delay is 2 ns and setup time is 2 ns. The clock period is 8 ns, and each grid in the following diagram is a unit of 1 ns. The initial values of clock and output are given in the 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!







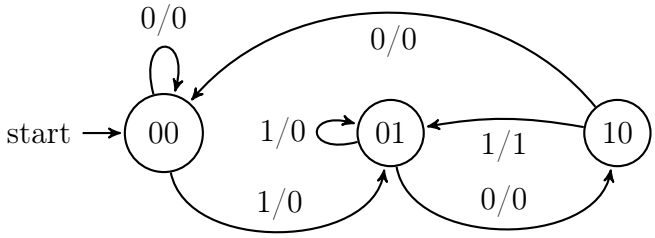
4 (b) Consider the following circuit. Assume the clock has a frequency of 50 MHz, all gates have a propagation delay of 6 ns, X changes 10 ns after the rising edge of clk, Reg1 and Reg2 have a clk-to-q delay of 1 ns.



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			

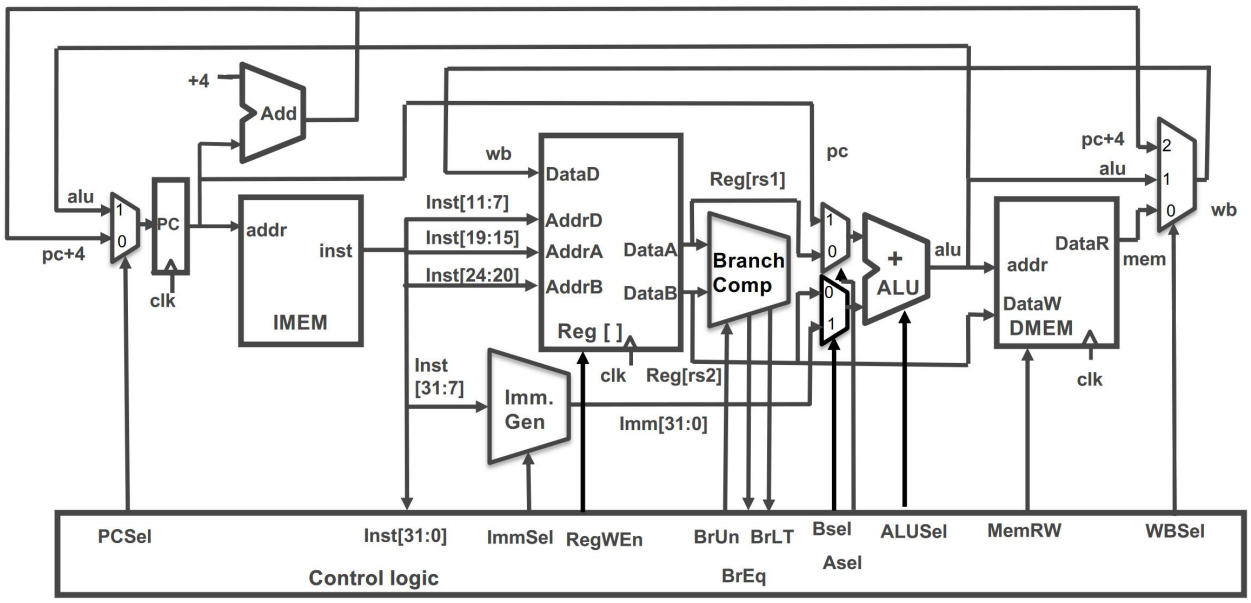
1 (b) What does the given FSM output with the input bit string '0100101010'?

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