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
```

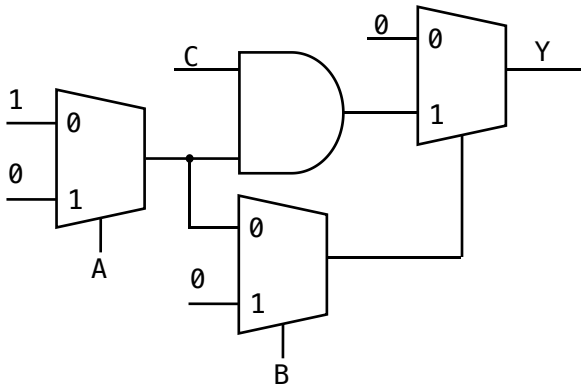
7. **Logic**

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

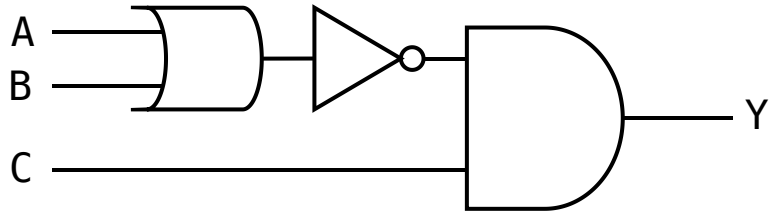
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

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.

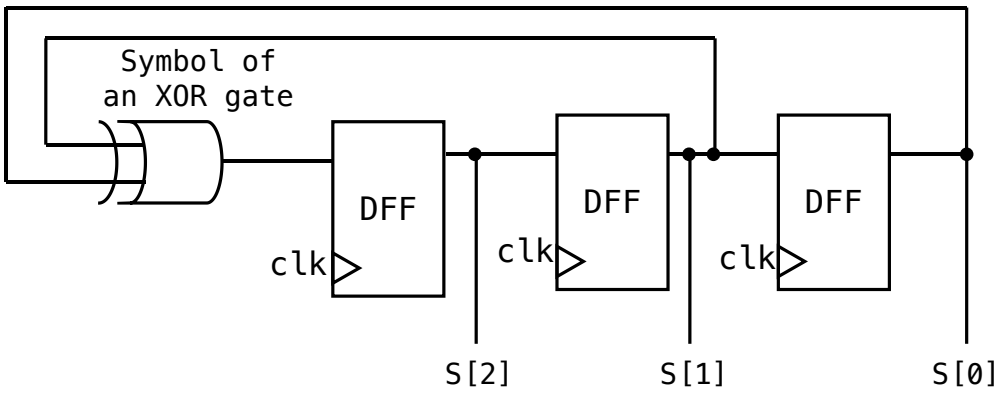
2 (d) Build a logic circuit that uses only 2-input **AND**, 2-input **OR** and **NOT** gates implementing the same logic above. Use as less logic gates as possible.

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.



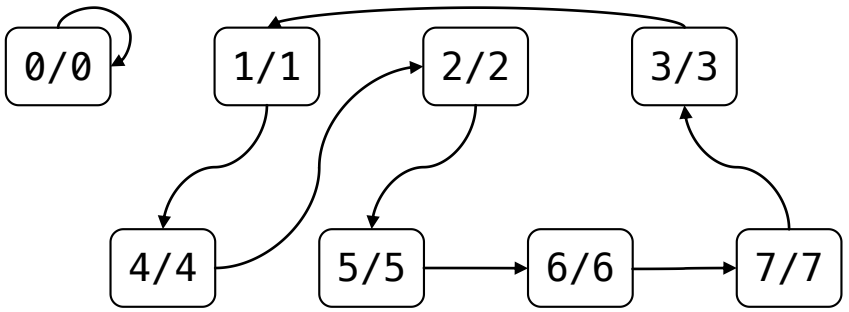
Solution: Each line for 0.5.

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.

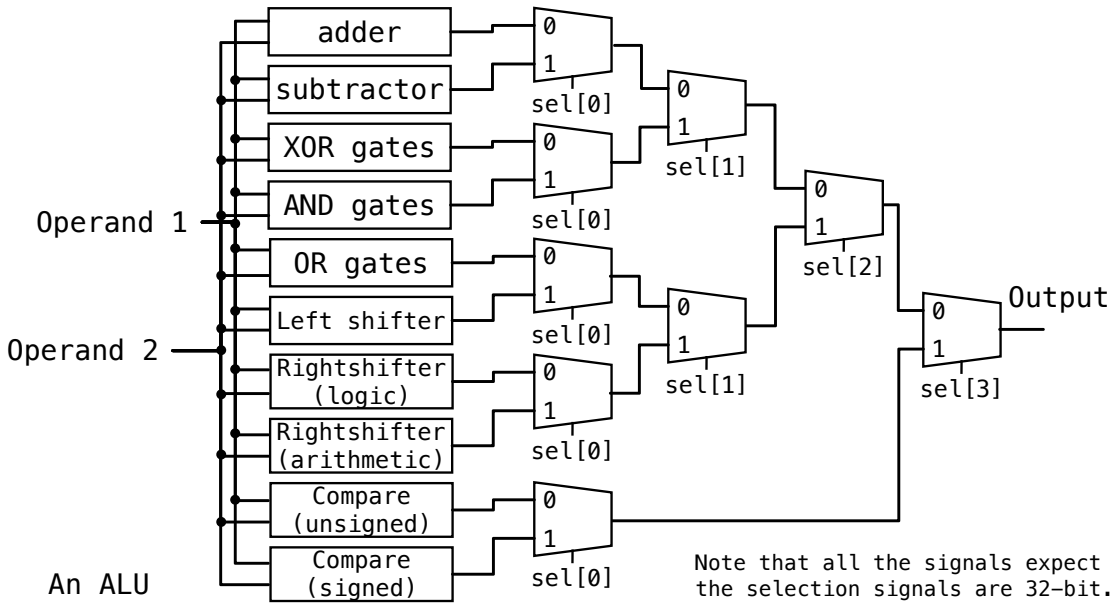


3 (c) The setup time of a DFF is 1 ns, the delay of an **`XOR`** gate is 2 ns, and the **`clk-to-q`** delay of the DFF is 1 ns. Compute the maximum frequency of this circuit. (We ignore the delay of the lines and ignore all the other non-ideal effects such as clock skews, etc.)

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.