Computer Architecture I Mid-Term

Chinese Name: ____________________

Pinyin Name: ____________________

Student ID: ____________________

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

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	1	12	18	8	5	12	8	8	18	10	100
Score:											

- This test contains 12 numbered pages, including the cover page, printed on both sides of the sheet.
- Unless told otherwise always assume a 32bit machine.
- The total estimated time is 90 minutes.
- You have 90 minutes to complete this exam, 10 min for submission. The exam is open book.
- Under no circumstances are you allowed to communicate to anybody except the CS110 professors or TAs.
- 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.
- Write your answers on blank A4 pages. Include the correct header on every page (see Question 1 below). At the end scan/ photo all pages into a pdf (there are plenty of apps for that). Submit in time - we will NOT accept late submissions!
- Each question from Q2 to Q10 will need to be started on a new A4 sheet - so you'll need at least 9 A4 sheets!

1 1. **Submitted pages format**
Correctly write the header for ALL the pages that you submit. That includes: Your name in Chinese, your ShanghaiTech Email address (without '@shanghaitech.edu.cn'), the string "CS110 Midterm", this Exam number (see on the top left of the following pages), the page number (your own numbering). Also follow the rule that every new major question (e.g. Question 2, Question 3, etc.) is written on a new A4 paper. Submit correctly on gradescope or (only if gradescope fails) via email to soerensch@shanghaitech.edu.cn AND wangchd@shanghaitech.edu.cn .

2. **CA & Number Representations** All floating point numbers follow IEEE 754 standard in this question.

$\boxed{1}$ (a) Name the 6 Great Ideas in Computer Architecture as taught in the lectures.

$\boxed{3}$ (b) Consider this 8-bit binary pattern `0b10101010`, please write down the number in **decimal** form if we are using the following representations:

Unsigned binary ____________________

Two's complement binary ____________________

One's complement binary ____________________

$\boxed{2}$ (c) For a 9-bit value, sign-magnitude integer, what are the largest **AND** smallest value you can represent in decimal?

____

$\boxed{2}$ (d) What is the smallest positive integer that **cannot** be represented by a 32-bit floating point number?

____

$\boxed{2}$ (e) **(Multiple Choice)** A 32-bit floating point number consists of:

A. 1 sign bit, 8 exponent bits and 23 significant bits.
B. 1 sign bit, 7 exponent bits and 24 significant bits.
C. 1 sign bit, 6 exponent bits and 25 significant bits.
D. 1 sign bit, 5 exponent bits and 26 significant bits.

$\boxed{2}$ (f) **(Multiple Choice)** Which of the following choices about floating point number is true?

A. Given that double precision number has 11 bits in its exponent part, then the bias should be -1024.
B. Floating number addition is associative.
C. The distance between floating point numbers remains constant as the absolute value of the numbers increase.
D. Max representable integer in floating point format is larger than the max integer in *unsigned int* type (both in 32 bit).

3. **C Basics**

$\boxed{3}$ (a) Suppose we have following macro:

```
#define DIFSQUARE(a, b) a * a - b * b
```

Write the result of `DIFSQUARE(3 + 1, 1 + 2)`. You may find you did not get the result you want. What result did you expect? Fix the definition of `DIFSQUARE(a, b)`.

$\boxed{2}$ (b) Consider the following code.

```
#include <stdio.h>
#define INT_NAME_VAL(name) ________________
int main() {
    int a = 1, b = 2, c = 3;
    INT_NAME_VAL(a);
    INT_NAME_VAL(b);
    INT_NAME_VAL(c);
    return 0;
}
```

Complete the definition of `INT_NAME_VAL(name)` using `printf` function, format specifiers (`%d, %u, %x, %s, %c...`) and `#`. So that the output is

```
a = 1
b = 2
c = 3
```

$\boxed{2}$ (c) What is the output of the following program.

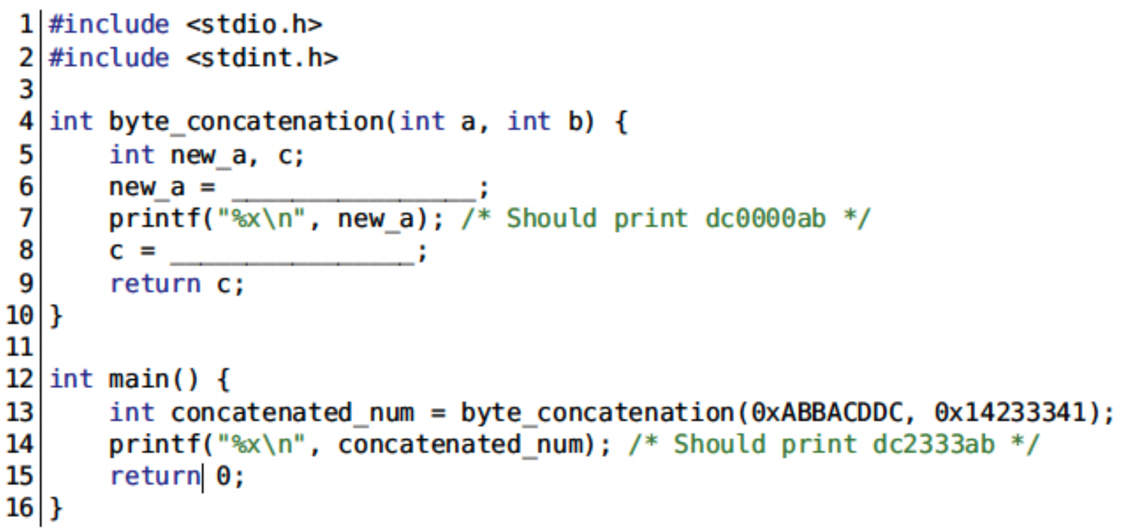
```
#include <stdio.h>

int some_bit_op(int x, int y, int z) {
    return ((x >> 1) | (y)) ^ ((y) & (z << 1));
}

int main() {
    printf("%d\n", some_bit_op(10, 14, 5));
    return 0;
}
```

$\boxed{4}$ (d) Consider the following program.

We want to concatenate two 32bit integer to become one new integer. For the new integer, we want its 1st byte to be the 4th byte of number **a**, its 2nd and 3rd byte to be the 2nd and 3rd byte of number **b**, and its 4th byte to be the 1st byte of number **a**.



```
 1 #include <stdio.h>
 2 #include <stdint.h>
 3
 4 int byte_concatenation(int a, int b) {
 5     int new_a, c;
 6     new_a = ________________;
 7     printf("%x\n", new_a); /* Should print dc0000ab */
 8     c = ________________;
 9     return c;
10 }
11
12 int main() {
13     int concatenated_num = byte_concatenation(0xABBACDDC, 0x14233341);
14     printf("%x\n", concatenated_num); /* Should print dc2333ab */
15     return 0;
16 }
```

1. **(Multiple Choice)** At line 6, we want ***new_a*** to get the 1st and 4th byte of **a** and exchange their position, choose the right expression that can correctly print the result.

   A. `(a >> 24) & (a << 24 & 0xFF)` B. `(a >> 24 & 0xFF) | (a << 24)`

   C. `(a >> 24) | (a << 24 & 0xFF)` D. `(a >> 24 & 0xFF) & (a << 24)`

2. At line 8, Write an expression that can correctly print the result using only bit operators including &, |, ^, ~, >>, <<. (You should not get 0xdc2333ab directly.)

7 (e) Memory in C

```c
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdint.h>
typedef struct _data {
   char      str[8];
   uint64_t  num;
   union {
      short     y;
      char      z[4];
      uint64_t  w;
   } InnerData;
} Data;

int main(){
   char* info = malloc(sizeof(char) * 8);
   Data testdata;
   sprintf(info, "%s", "Size: ");
   printf("%s%s%lu\n", "Data", info, sizeof(testdata));
   printf("%s%s%lu\n", "InnerData", info, sizeof(testdata.InnerData));
   return 0;
}
```

1. Fill in the correct memory section based on what the given C expressions evaluate to.

   `testdata` ____________________

   `info` ____________________

   `&info` ____________________

2. What is the expected output of the following code?

   ____

   ____

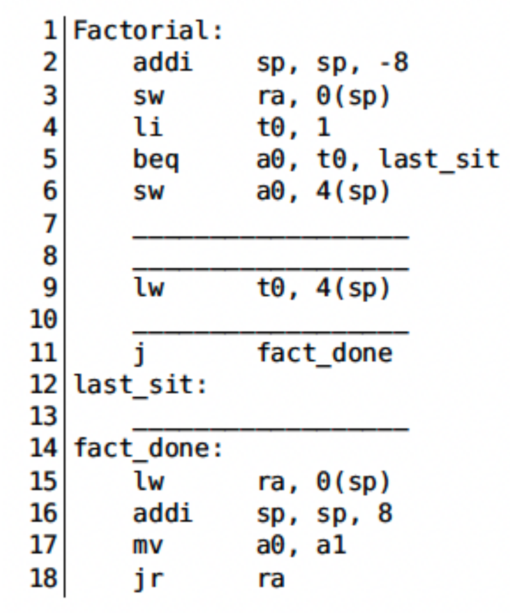
3. Does this simple program encounters memory problems? Point it out if there are such problem(s), or answer "No problem" if not.

   ____

   ____

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: ______________________________

$\boxed{5}$ 5. **Logic**

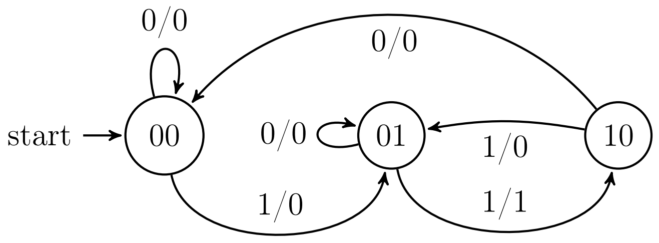
Draw the **fully** simplified (with **fewest** number of primitive gates) circuit for the expression below. Write down the **steps** for simplification and draw the **final** circuit. You may use the following primitive gates: AND, OR, XOR and NOT.

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

____

____

6. **FSM**

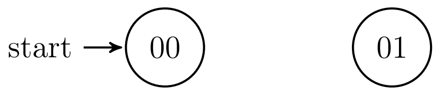


$\boxed{6}$ (a) Copy and fill in the truth table for the FSM above.

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

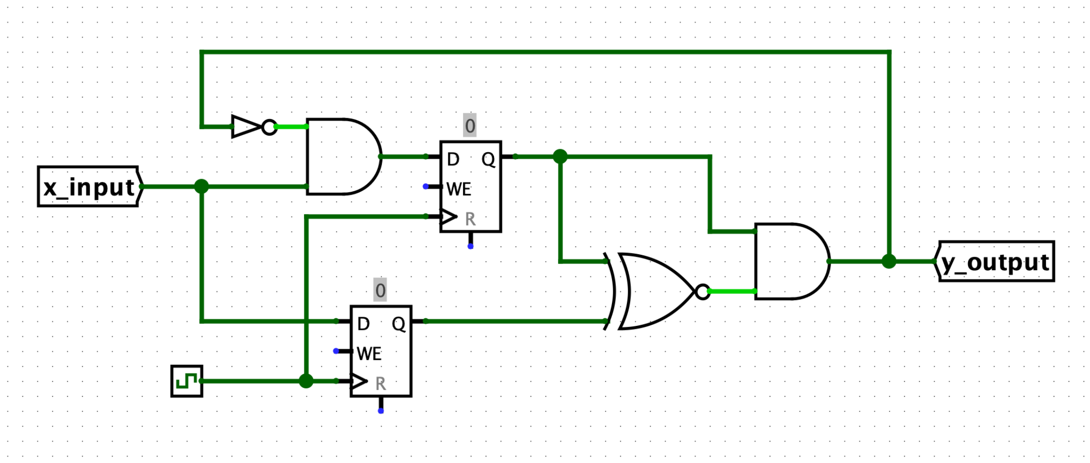
$\boxed{2}$ (b) What does the given FSM output with the input bit string '1111011010'?

$\boxed{4}$ (c) Draw a FSM that shares the same function with the above given FSM using only two states (Output the same as the given FSM for any input bit string).



7. **SDS**

We will be analyzing the following circuit, given the information below.



- For all registers, **Clk-to-q** delay time 4ns, **Set up** time 3ns.
- Gates Delay: `AND` gate 5ns, `NOT` gate 3ns, `XOR` gate 7ns, `NXOR` gate 10ns.
- `x_input` is directly attached to a source that changes its value (i.e. 1 to 0, or 0 to 1) 15ns after the rising edge of the clock.
- `y_output` is directly attached to a register.
- All wires are ideal, i.e. they have zero delay.

2 (a) What is the maximum possible hold time for registers to function properly in this circuit?

3 (b) What is the minimum possible clock period of this circuit?

3 (c) Now we assume the clock frequency is 20MHz (which would be a proper frequency if you analyzed the circuit correctly) and the first rising edge of the clock is at 0ns. x_input is initialized to 0 at 0ns. Try to figure out at what time will y_output become 1?

8. **Pipeline**

$\boxed{3}$ (a) **(Multiple Choice)** Choose the correct statement(s) for a five stage pipelined processor. There may be more than one correct answer.

A. Pipeline rate is limited by the slowest pipeline stage.

B. Using pipeline is an effective method to reduce the latency of a single task.

C. When the two instructions `add t0, t0, t1` and `add t2, t0, t1` are executed consecutively, there is no hazard.

D. When instructions `add t1, t0, t0`, `addi t2, t0, 5`, and `addi t4, t1, 5` are executed consecutively, the hazard can be solved by forwarding.

$\boxed{3}$ (b) Consider the following instructions and complete the pipeline diagram for the first iteration. Register writes happen before register reads in the same clock cycle, branch comparison is done during the register read stage, and forwarding is implemented. Assume the branch is not taken in the first iteration.

```
loop:
    lw      t0, 0(a0)
    beq     t0, 0, end
    addi    t0, t0, 10
    sw      t0, 0, 0(a0)
end:
```

Note: we use F for fetch, D for decode, X for execute, M for memory, W for write back.

lw	F	D	X	M	W								
beq													
addi													
sw													

$\boxed{2}$ (c) Each stage in pipeline and the registers in pipeline has following timing:

F	D	X	M	W	Clock-to-Q	Hold time	Setup
175ps	100ps	100ps	200ps	100ps	30ps	20ps	30ps

for the 5 stage pipeline, compute the minimum clock period, and show the calculation process.

____

____

9. **Cache**

   Notice: We assume a 32-bit machine by default.

5 (a) **(True or False)** Please fill your answer (T or F) in the table below.

Notice: You will get 1 point for a correct answer, -1 point for a wrong answer, and 0 point for not answering. The total point will not be less than 0.

1. Cache replacement policy is used for choosing which **cache line** should be evicted.
2. One way to decrease compulsory miss is enlarging the block size.
3. For a cache with fixed block size and length of the **tag** field, increasing or decreasing associativity will not influence the hit time.
4. Cache blocking can make transposition faster because it decreases compulsory miss.
5. For a 1-level, LRU cache, cache blocking will greatly speed up matrix transposition when the cache size is larger than the matrix size.

1	2	3	4	5

4 (b) Suppose the cache has the following settings:

Cache levels	1
Block size	16 bytes
Number of sets	4
Cache size	128 bytes
Block replacement policy	LRU

1. Is this cache direct-mapped, set-associative, or fully-associative? If it is associative, also write down its associativity.

   ____

   ____

2. Suppose the memory addresses are 32-bit long, what is the length of the tag field?

   ____

   ____

$\boxed{9}$ (c) Suppose the following code is running on a system with the above cache, where `sizeof(int) == 4`.

```
 1 #define array_size   64
 2 #define repeat_times 1
 3 #define step_size    2
 4 
 5 int main() {
 6     int array[array_size] = { };
 7 
 8     for (int r = 0; r < repeat_times; r++) {
 9         for (int i = 0; i < array_size; i += step_size) {
10             array[i] = array[i] + 2333;
11         }
12     }
13 
14     return 0;
15 }
```

1. What is the total number of accesses to the cache?

2. What is the hit rate?

3. Which type(s) of miss occur(s)?

4. Suppose `repeat_times` **goes to infinity** (only for this question), what number will the hit rate converge to?

5. If `repeat_times` **is changed to 2** (only for this question), try to swap two lines of the above code to maximize the hit rate without disturbing the results. Which two lines will you choose and what is the maximized hit rate?

6. **For the modified code in the previous question**, suppose again `repeat_times` **goes to infinity** (only for this question), what number will the hit rate converge to?

10. **Threads and OpenMP**

Consider the following code snippet written in C. The function `abs_sum` is to calculate the sum of absolute values in a given square matrix (i.e., $\sum_{i=1}^{n}\sum_{j=1}^{n}|mat[i][j]|$).

```
 1 #include <omp.h>
 2 #include <math.h>
 3 #include <stdlib.h>
 4 double abs_sum(double** mat, int n){
 5     double ret = 0.0;
 6     #pragma omp parallel for
 7     for (int i = 0; i < n; i++){
 8         for (int j = 0; j < n; j++)
 9             ret += abs(mat[i][j]);
10     }
11     return ret;
12 }
```

$\boxed{3}$ (a) This code doesn't always give a correct result. Please give your explanation.

$\boxed{4}$ (b) Someone suggests to add a line `#pragma omp atomic` between line 8 and 9 to fix this issue. Does it work? Is it a efficient implementation? If so, give your explanation. If not, try to give a more efficient implementation.

$\boxed{3}$ (c) Assume we have fixed this bug now. If you modify line 9 to `ret += abs(mat[j][i])`, you will observe a significantly performance degradation. Try to explain it.