Computer Architecture I Mid-Term I

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

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

Question	Points	Score
1	1	
2	18	
3	9	
4	5	
5	5	
6	4	
7	10	
8	9	
9	12	
10	20	
11	7	
Total:	100	

- This test contains 16 numbered pages, including the cover page, printed on both sides of the sheet.
- We will use gradescope 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.
- The total estimated time is 105 minutes.

- You have 105 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one A4 page (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!
- Unless otherwise stated, always assume a 32 bit machine for this exam.

[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).

2. **Various Questions**

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

Solution:
1. Abstraction (Layers of Representation/Interpretation)
2. Moores Law (Designing through trends)
3. Principle of Locality (Memory Hierarchy)
4. Parallelism
5. Performance Measurement and Improvement
6. Dependability via Redundancy

[2] (b) Provide the complete names of the following abbreviations:

ISA: _____

RISC: _____

Solution:
ISA: Instruction Set Architecture
RISC: Reduced Instruction Set Computing / Computers

[2] (c) There is an opposite philosophy of architectural design other than RISC. One of the instruction sets that follows such philosophy is the famous *Intel x86*. Pick its correct name:

_____ a. AISC b. CISA c. CISC d. RISA

Give its full name: _____

Solution: c
Complex Instruction Set Computing / Computers

[2] (d) Let's play with `gcc`! It does the correct thing only if you provide the correct options. Write the option you need for the following common scenarios:

1. Enable debugging mode, add debug information: -_____
2. Turn **off** all the optimizations: -_____
3. Be strict! All minor warnings should be treated as errors: -_____
4. Set the *C* language standard to ANSI C (C89): -_____

Solution:
1. `-g` / `-ggdb` / `-g`*`level`* with *level* ≥ 1

2. `-O0` (Attention: "Hypen" "Oh" "Zero")
3. `-Werror` 4. `-std=c89`

4 (e) We can classify computer languages into two categories: *Compiling* and *Interpreting*. (Forget JAVA-like languages which mix up these two). Which scheme does the following languages take? Write exactly "compiling" or "interpreting".

C/C++: _____ Python: _____

Based on your understanding, are the following statements true (T) or false (F)? Circle your answer.

T / F It is often harder to decompile (reverse-engineer) a compiling language software than an interpreting language software.

T / F For the same algorithm, a compiled implementation always runs faster.

T / F We do **not** need any environment dependency to support interpreting language execution on a brand-new machine.

Solution: compiling interpreting T F F

5 (f) Given below is a piece of *C* code. What location will the five expressions on the right give, when the execution reaches point `Break!`? Write "stack", "heap", "static" or "text" for each of them. (For example, write "heap" to indicate that the expression evaluates to an address within the memory section of heap.)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  static const int year = 2019;
4
5  int main (void) {
6     char name[] = "Jose";
7     char *game = "The Elder Scrolls";
8     int *ver = malloc(sizeof(int));
9     *ver = 5;
10    /* Break! */
11    printf("Until %d, %s's favourite game is
         %s %d.\n", year, name, game, *ver);
12    return 0;
13 }
```

Expressions:

`&year` _____

`name` _____

`game` _____

`ver` _____

`&ver` _____

Solution:
`&year`: static
`name`: stack

`game`: static
`ver`: heap
`&ver`: stack

3. **Number Representation**

4 (a) Fill in the blanks with a letter (a, b, c, d) to match each expression in the left column, with an equivalent expression from the right column, ^ is XOR:

1. (x >> 16) << 16 = _____	a. 0
2. x ^ x ^ x = _____	b. ~x
3. x ^ 1 = _____	c. x
4. x & ~x = _____	d. x & 0xFFFF0000

Solution: 1. `d` 2. `c` 3. `b` 4. `a`

3 (b) Show how the binary string 0b1001 0101 can be interpreted and displayed as the following types:

1. Hexadecimal: `0x`_____
2. Unsigned Decimal: _____
3. Two's Complement Decimal: _____

Solution: 1. `0x95` 2. `149` 3. `-107`

2 (c) Consider we have a base 32 number, with each number position represented by the numerals 0 through 9 plus the letters A (10), B (11), C(12), D(13), E (14), F (15), G (16), H (17), I (18), J (19), K (20), L (21), M (22), N (23), O (24), P (25), Q (26), R (27), S (28), T (29), U (30), and V(31).

Convert VAN_{32} to:

1. Binary: _____

2. Hexadecimal: `0x`_____

Solution: 1. `111 1101 0101 0111` 2. `0x7d57`

4. C types and variables

Note: All following code is complied with "-m32 -std=c89" and executed on a little-endian machine.

[2] (a) What is the output of the following program:

```
#include <stdio.h>
#include <stdint.h>
int main () {
   int8_t a;
   uint8_t b;
   a = 255;
   b = 255;
   a += 255;
   b += 255;
   printf ("%d %u\n", a, b);
   return 0;
}
```

Solution: -2 254

[3] (b) Suppose we have a union:

```
union Foo {
   uint8_t a;
   int16_t b;
   uint32_t c;
   char d[4];
}bar;
```

If we copied string `"ABC"` into `bar.d` by `strcpy(bar.d, "ABC")`, what is the value of the following variables? Write the value in hexadecimal format (use two's complement format for signed integers).

1. bar.a: `0x`_____
2. bar.b: `0x`_____
3. bar.c: `0x`_____

Solution: 1. `0x41` 2. `0x4241` 3. `0x434241`

5. See what you can C

[5] (a) This section involves T / F questions. Incorrect answers on T / F questions are penalized with negative credit. Circle the correct answer.

T / F: `bool` cannot be directly used under C89 standard.
T / F: Increment operator (++) should be used with cautious, since it may reduce the readability of the code sometimes.
T / F: A constant variable is different from a macro.
T / F: Empty pointers (`NULL`) cannot be directly dereferenced.
T / F: Conditional branches (`if ... else ...`) cannot interchange with conditional compilation (`#if ... #else ... #endif`) for most of the time.
T / F: All header files from C standard library can be included multiple times.
T / F: The address of a function is sometimes larger than that of a variable.
T / F: Different members of a union can have different sizes.
T / F: `const int *` means this pointer can be changed, but the value it points to cannot be changed.
T / F: `int * const` means this pointer cannot be changed, but the value it points to can be changed.

Solution: TTTTT TFTTT

6. **Playing fire with C pointers**

2 (a) There are 4 kinds of pointers in C: wild pointer, void pointer, null pointer and dangling pointer. Which kind(s) of pointer(s), when used, will **definitely** cause memory issues?

Solution: Wild and dangling pointers. (One point for each, if you write null pointer, you won't receive or lose any point.)

2 (b) For the following code, specify which kind (wild, null or dangling) do these void pointers belong to. If the void pointer does not belong to any of the three kinds, write down "void".

```
1  void * foobar () {
2     int x = 5;
3     return &x;
4  }
5  int main () {
6     void *a;
7     void *b = NULL;
8     void *c = foobar;
9     void *d = foobar ();
10    return 0;
11 }
```

1. a: _____ pointer 2. b: _____ pointer
3. c: _____ pointer 4. d: _____ pointer

Solution: a: `wild` b: `null` c: `void` d: `dangling`

7. **King of C Macro**

[1] (a) Suppose we want to declare an array whose length can be easily modified afterwards,

```
const int MAX_LEN = 10;
int a[MAX_LEN];
```

However, this is not allowed under C89 standard. How to fix it using C macro?

Solution:
```
#define MAX_LEN 10
```

[3] (b) Suppose we have an macro:

```
#define MUL(a, b) a * b
```

What is the result of `MUL(1 + 2, 3 + 4)`? Did we get the result we want? If not, how to fix this macro?

Solution: 11. No.
```
#define MUL(a, b) ((a) * (b))
```

[1] (c) Based on (b), write a macro `MAX(a, b)` that returns the maximum value of the two. You **should** use ternary operator: `cond ? x : y` returns `x` when `cond` is true, `y` otherwise.

Solution: (One possible version, other versions are accepted if correct)
```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

[3] (d) Based on the macro you wrote in (c), if `a = 10`, `b = 15`, what is the value of `a` after the execution of `MAX(a++, b)`? What if `a = 15`, `b = 10`? Why?

Solution: 11 and 17, respectively.
Because `a++` only executed once if `a <= b` (depends on your answer in (c)), twice otherwise.

[2] (e) Suppose we have an array for `struct command`, we can initialize the array in the following way (`quit_command` and `help_command` are just variables):

```
struct command commands[] = {
    { "quit", quit_command },
    { "help", help_command },
    ...
};
```

Now we want to use a single macro `COMMAND(NAME)` to help us initialize the array, which should look like:

```
struct command commands[] = {
    COMMAND(quit),
    COMMAND(help),
    ...
};
```

Finish the implementation of the macro. You can use "#", but ternary operator and `if-else` statement are **not** allowed.

Solution: Note: 1pt for `#NAME`, 1pt for `##`. There is no quotes around `NAME`.

```
#define COMMAND(NAME) { #NAME, NAME ## _command }
```

8. DIY session

Note: In this session, you are required to write you own code into the blanks. All code should adhere to the C89 standard, and there can be only one instruction per line (e.g. each line cannot have more than one semicolon, comma operator is not allowed, etc.) Your code should not produce any warning as well.

3 (a) Implement a function that swaps two integers' value.

```
/* Swap two integers. */
void mySwapInt (_____ a, _____ b) {

    ____________________________________

    ____________________________________

    ____________________________________
}
```

Solution: Reference program:

```
void mySwapInt (int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

3 (b) Implement a function resembles `string.h`'s `memcpy`. No function from the C standard library is allowed.

```
/* Copies the values of size bytes from the location pointed to
   by src directly to the memory block pointed to by dst. */
void myMemcpy (void *dst_, const void *src_, size_t size) {
    unsigned char *dst = dst_;

    ____________________ *src = src_;
```

```
7
8       _____________________________________
9
10      _____________________________________
11
12      _____________________________________
13
14      _____________________________________
15  }
```

Solution: Reference program:

```
void myMemcpy (void *dst_, const void *src_, size_t size) {
   unsigned char *dst = dst_;
   const unsigned char *src = src_;
   while (size-- > 0)
       *dst++ = *src++;
}
```

3 (c) Implement a generic swap function that can swap variables of any type, as long as the size is given. You can **only** use `malloc`, `free` and `myMemcpy` from (b). No memory leaks are allowed.

```
1  /* Swap two variables with any type. */
2  void mySwap (_____ a, _____ b, size_t size) {
3
4       _____________________________________
5
6       _____________________________________
7
8       _____________________________________
9
10      _____________________________________
11
12      _____________________________________
13 }
```

Solution: Reference program:

```
void mySwap (void *a, void *b, size_t size) {
   void *t = malloc (size);
   myMemcpy (t, a, size);
   myMemcpy (a, b, size);
   myMemcpy (b, t, size);
   free (t);
}
```

9. RISC-V Programming

2 (a) Getting familiar with a programming usually starts with printing "Hello world!" in the console. We will jump into RISC-V part by doing so. Please fill in the blanks of the following piece of code.

```
.data

_____________________________________
.text

_____________________________________

_____________________________________

_____________________________________
```

Solution:

```
.data
   str:  .string "Hello world!"
.text
   li a0, 4
   la a1, str
   ecall
```

5 (b) In the C programming language, you may count the length of a string via *strlen* given in the lectures. Please fill in the following blanks to implement this function in RISC-V. Note that you are not allowed to write lines of code more than the number of blank lines given, otherwise some but not all of the points will be deducted.

```
   addi  s1, x0, 1
   jal strlen
   add a0, x0, s1
   ecall
strlen:

   _____________________________________

   _____________________________________

   _____________________________________
   la s1, str
loop:

   _____________________________________

   _____________________________________
```

```
18
19      _____________________________________
20
21      _____________________________________
22
23      _____________________________________
24
25      _____________________________________
26  epilogue:
27
28      _____________________________________
29
30      _____________________________________
31      jr ra
```

Solution:

```
1      addi  s1, x0, 1
2      jal strlen
3      add a0, x0, s1
4      ecall
5   strlen:
6      addi  sp, sp, -4
7      sw s1, 0(sp)
8      mv a1, x0
9      la s1, str
10  loop:
11     lbu t0, 0(s1)
12     beq t0, x0, epilogue
13     addi  a1, a1, 1
14     addi  s1, s1, 1
15     j  loop
16  epilogue:
17     lw s1, 0(sp)
18     addi  sp, sp, 4
19     jr ra
```

5 (c) For ease of use, RISC-V provides a variety of pseudo instructions, most of them should be translated to TAL instructions during assembly.

1. Identify whether the following instructions are pseudo instructions. If it is, write the expansion form of this instruction.

T / F `jal ra, L1 #1`

T / F `ble t1, t2, L2 #2`

T / F `j L3 #3`

T / F `jr t3 #4`

T / F `li t0, 0x12345678 #5`

Solution: F T T T T

```
bge t2, t1, L2 #2
jal x0, L3 #3
jalr x0, t3, 0 #4
lui t0, 74565 #5
addi t0, t0, 1656
```

10. **RISC-V Questions**

Notice. In this part, you can write at most **ONE** line of code in each space when we ask you to write down codes, but you do not have to use all of the spaces. **If you write more than one line of code in one space, that answer will be voided.** ("one line of code" means one semicolon in C or one instruction in RISC-V)

4 (a) This section involves T / F questions. Incorrect answers on T / F questions are penalized with negative credit, the section's credit will not be less than 0 point (e.g. one incorrect and three correct, you will get 2 points). Circle the correct answer.

T / F: The following instruction will be modified in linker.

```
1           lui $t0, $t0, 0x8000
```

T / F: The instruction 0x00a98863 is a "SB-Format" instruction.
T / F: The instruction "jump register" will not change program counter.
T / F: $AUIPC$ used for absolute addressing.

Solution: F T F F

Look at the following RISC-V code.

```
1  mystery:
2     add a0, ra, zero  a0: R[31:0]
3     la  t0, mystery
4     addi t2, zero, -1 t2:
5                                ____________________________
6     slli t2, t2, 28   t2:
7                                ____________________________
8     xori t2, t2, -1   t2:
9                                ____________________________
10    and ra, t2, a0    ra:
11                               ____________________________
12    srai ra, ra, 6    ra:
13                               ____________________________
14    lui t2, 0x800     t2:
15                               ____________________________
16    or  ra, t2, ra    ra:
17                               ____________________________
18    sw  ra, 4(t0)
19    jr  a0
```

4 (b) Please use the formats like following examples to write the annotated registers values in binary.
e.g.1 a0: R[31:0] *e.g.2 zero: 0 × 32*

1. R[15:0] - gives the less significant half of R.
2. 0×13 - gives 13 consecutive 0s.
3. 1×12 - gives 12 consecutive 1s.
4. $0 \times 28 \mid 1 \mid 0 \times 3$ - gives 0b1000 (8).

Solution:

$$1 \times 32$$
$$1 \times 4 | 0 \times 28$$
$$0 \times 4 | 1 \times 28$$
$$0 \times 4 | R[27:0]$$
$$0 \times 10 | R[27:6]$$
$$0 \times 8 | 1 | 0 \times 23$$
$$0 \times 8 | 1 | 0 | R[27:6]$$

1 (c) There is an instruction "beq t1, t2, mystery". When will the goal address be determined?

Solution: Assembly.

1 (d) Towards end of the above code "jr a0". When will the goal address be determined?

Solution: Run time.

Look at the following RISC-V code.

```
num_of_ones:
   addi sp, sp, -12 # t1
   sw  s1, 0(sp)
   sw  s2, 4(sp)
   sw  s3, 8(sp)   # t2
   0x00a004b3      # t3
   ________________________# 1
   ________________________# 2
loop:
   ________________________# 3
   addi s3, s3, -1
   ________________________# 4
   ________________________# 5
   0xfe0288e3      # t4
   addi s2, s2, 1
   ________________________# 6
exit:
   ________________________# 7
   lw  s1, 0(sp)
   lw  s2, 4(sp)
   lw  s3, 8(sp)
   addi sp, sp, 12
   jr  ra
```

4 (e) Translate instructions #t1, #t2 to machine code written in hexadecimal! Translate #t3, #t4 to RISC-V instructions!

addi sp, sp, -12 # t1: _____

sw s3, 8(sp) # t2: _____

0x00a004b3 # t3: _____

0xfe0288e3 # t4: _____

Solution:

```
0xff410113      # t1
0x01312423      # t2
add s1, zero, a0 # t3
beq t0, zero, loop # t4
```

4 (f) Finish the function "num_of_ones()", which returns the number of 1s in the given binary parameter. Every line can only have at most one instruction, but you do not have to use all of the lines.

Solution:

```
mv  s2, zero    # 1 {add s2, zero, zero / li s2, 0}
li  s3, 32      # 2 {addi s3, zero, 32}
beq s3, zero, exit # 3
andi t0, s1, 1 # 4
srli s1, s1, 1 # 5
j   loop        # 6 {jal zero, loop / jalr zero, loop}
mv  a0, s2      # 7 {add a0, zero, s2}
```

2 (g) The assembly code above can be used as a function. How would you call the function and which registers would you need to fill with proper values before calling the function?

Solution: "jal num_of_ones", a0 set to the the target number.

11. **CALL Questions**

7 (a) For each step, list the task that step is responsible for. Some task might belong to multiple steps.

1. translate a source file written in a higher level language to assembly code
2. turn branch labels into addresses
3. copy the .text and .data segments from disk into memory
4. for each unresolved reference to a symbol, find the definition of the symbol
5. jump to the main() function to start the program
6. translate pseudo instructions to real instructions
7. optimize the code

Compile: _____

Assemble: _____

Link: _____

Load: _____

Solution:
Compile: 1 7
Assemble: 2 6
Link: 4
Load: 3 4 5