Computer Architecture I Final Exam

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

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

Question	Points	Score
1	1	
2	6	
3	11	
4	5	
5	9	
6	6	
7	12	
8	7	
9	23	
10	4	
11	8	
12	6	
Total:	98	

- This test contains 21 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.
- Unless told otherwise always assume a 32bit 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 21 times).

2. •Number Representation

2 (a) Convert hexadecimal 0x3A7 into

Decimal (base 10) _____

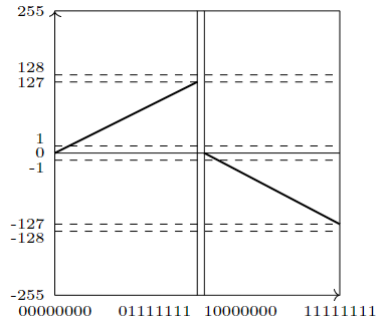
Octal (base 8) _____

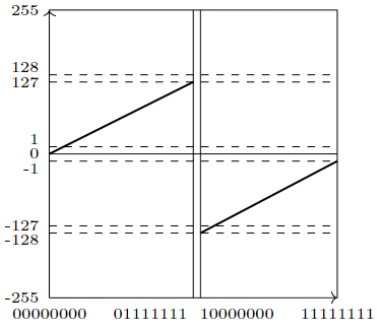
Solution:
935
1647

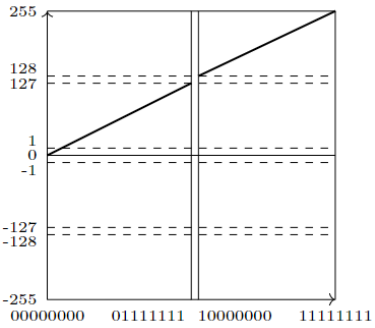
2 (b) Assume the horizontal axis represents a bit string incrementing from 00000000 to 11111111, the vertical axis represents a decimal number. Each diagram below shows a different number representation. Please write down the letter (A, B, C or D) for each number representation:

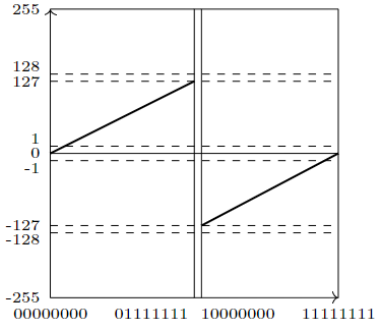
Unsigned _____ Sign Magnitude _____

One's complement _____ Two's complement _____

A


B


C


D


Solution:
CADB

2 (c) Suppose we are using single precision IEEE 754 floating point. Complete the following conversions.

0xC26B0000 $\Rightarrow$ _____

19.5 $\Rightarrow$ 0x _____

Solution:
-58.75
0x419C0000

3. **Various Questions**

2 (a) Assume the execution latency of the longest-latency instruction in a 4-wide superscalar, out-of-order machine implementing one algorithm is 1000 cycles.

How large should the instruction window be such that the decode of instructions does not stall in the presence of this longest-latency instruction?

Solution:
4000 instructions.
More detail: 1000 cycles/stall $\times$ 4 instructions to buffer every cycle $\rightarrow$ Need a 4000-instruction-entry window.

2 (b) On a 32-bit Linux system, the largest memory one process can have is 4 GiB. Can we allocate 4 GiB on stack? Why or why not?

Solution: No, since there will be no space for other parts (code, heap).

For the next part, numbers are represented in 2's complement and stored in little endian. Except for decimals, please keep the leading zeros for full representation of the specified data length. All answers should adhere to the required format indicated by the subscripts.

4 (c) Consider the following numbers are stored in an `unsigned char`, figure out the conversions and fill the blanks.

$(255)_{10}$ = (_____)$_2$ $(DF)_{16}$ = (_____)$_{10}$

$(11101010)_2$ = (_____)$_{16}$ $(2333)_4$ = (_____)$_{16}$

Solution:
$(255)_{10} = (11111111)_2$ $(DF)_{16} = (223)_{10}$

$(11101010)_2 = (EA)_{16}$ $(2333)_4 = (BF)_{16}$

3 (d) Suppose we have this assignment `uint32_t a = 0xFFFFFEDC;` in our C code, and all code are compiled using gcc with `-m32` option. Calculate the value of the following expressions.

`(int64_t) a`	$(\_\_\_\_\_)_{16}$
`*((int8_t*) &a)`	$(\_\_\_\_\_)_{10}$
`sizeof (((void*) &a) + 1)`	$(\_\_\_\_\_)_{10}$

Solution:

`(int64_t) a`	$(00000000FFFFFEDC)_{16}$
`*((int8_t*) &a)`	$(-36)_{10}$
`sizeof (((void*) &a) + 1)`	$(4)_{10}$

5 4. **CALL** See the following steps in software development:

1. Produce an executable binary from a collection of C source files.
2. Checks if there are any syntax errors in the C source fils.
3. Validate all memory accesses are legal.
4. Relocation, i.e. assigning load addresses for position-dependent code and data of a program and adjusting the code and data to reflect the assigned addresses.
5. Runtime symbol resolution.
6. Resolve `#include` directives.
7. Translate symbolic register name to integer register name.

Assuming we are using a RISC-V system running on Ubuntu 18.04,

(a) Compiler will be involved in _____

(b) Assembler will be involved in _____

(c) Linker will be involved in _____

(d) Loader will be involved in _____

Solution: Compiler: 1. 2. 6.

Assembler: 1. 7.

Linker: 1. 4.

Loader: 5.

Each missing/wrong assignment will deduct 0.5 pts.

We will however not deduct points beyond the max points you can get from this question.

5. **OpenMP and Flynn Taxonomy**

3 (a) Considering the following piece of C code:

```
int SIZE = 100;
int main(){
   int A[SIZE];
   for(int i=0; i<SIZE; i++)
      A[i] = i;
}
```

We want to parallel the for-loop using OpenMP. Given the following three stageties, circle the correct statement.

(i)
```
#pragma omp parallel for
for (int j=0; j<SIZE; j++){
   *A = j;
   A++;
}
```

a) Always correct but slower than original
b) Always correct but speed depends on caching scheme comparing to original
c) Always correct and faster than original
d) Sometimes correct.
e) Always incorrect.

(ii)
```
#pragma omp parallel
{
   for (int j=omp_get_thread_num();
         j<SIZE;
         j+=omp_get_num_threads()){
      A[j] = j;
   }
}
```

a) Always correct but slower than original
b) Always correct but speed depends on caching scheme comparing to original
c) Always correct and faster than original
d) Sometimes correct.
e) Always incorrect.

(iii)
```
#pragma omp parallel
{
   for (int j=0; j<SIZE; j++){
      A[j] = j;
   }
}
```

a) Always correct but slower than original

b) Always correct but speed depends on caching scheme comparing to original
c) Always correct and faster than original
d) Sometimes correct.
e) Always incorrect.

Solution: (i)d (ii)b (iii)a

6 (b) We want to use SIMD to count the non-zero elements of array. Assume n to be multiple of 2, assume double use 64-bit . You may find the following intrinsics helpful:

1. `void _mm_storeu_pd(double *a, __m128d b)` store b at memory a
2. `__m128d _mm_set1_pd(double val)` set packed 64-bit doubles of two val
3. `__m128d _mm_cmpneq_pd (__m128d a, __m128d b)` compare packed 64-bit doubles respectively, get 64-bit all 0 if equal, 64-bit all 1 if not equal
4. `__m128d _mm_add_pd (__m128d a, __m128d b)` add packed 64-bit doubles
5. `__m128d _mm_and_pd (__m128d a, __m128d b)` return bitwise AND of packed 64-bit doubles

```
1  int count_non_zero(double arr[], int n){
2     int nonzero = 0;
3     double ans[2];
4     __m128d ans_vec = _mm_setzero_pd();
5     __m128d zeros = _mm_setzero_pd();
6     __m128d ones = _mm_set1_pd(1.0);
7     __m128d mask,data;
8     for(int i=0; i<n; i+=2){
9        data = _mm_loadu_pd((__m128d *) (arr+i));
10
11       mask = ____________________________________
12
13       mask = _mm_and_pd(mask,ones);
14
15       ____________________________________________
16
17       ____________________________________________
18    }
19
20    ____________________________________________________
21    nonzero = ans[0]+ans[1];
22    return nonzero;
23 }
```

Solution: By lines:

```
1 (1)_mm_cmpneq_pd(zeros, data);
```

```
(2)ans_vec = _mm_add_pd(ans_vec, mask);
(3)leave empty
(4)_mm_storeu_pd(ans,ans_vec);
```

6. **RISC-V pipelining**

Data hazard Look for data hazards in the code below.

Instruction	C1	C2	C3	C4	C5	C6	C7	C8
1. addi s0, s0, 1	IF	ID	EX	MEM	WB			
2. addi t0, t0, 4		IF	ID	EX	MEM	WB		
3. lw t1, 0(t0)			IF	ID	EX	MEM	WB	
4. add t2, t1, x0				IF	ID	EX	MEM	WB

There are two data hazards in the code. fill the instruction number or reg name below:

2 (a) the first hazard is between instructions __ and __ , from register __

(a) **2 3 t0**

2 (b) the second hazard is between instructions __ and __ , from register __

(b) **3 and 4, from t1**

2 (c) one of them cannot be solved with forwarding, we can fix this by inserting a nop (no-operation) between instructions __ and __ .

(c) **3 and 4**

7. **RISC-V Questions**

The following shows the C code for "exponentiation by squaring" algorithm that computes a^n, which is a quick algorithm with time complexity $O(\log n)$:

```
int ans = 1;
while(n)
{
    if(n & 1)       // if n ends with 1:
        ans *= a;   // ans is multiplied by a
    a *= a;         // a <- square of a
    n >>= 1;        // get next bit of n
}
printf("%d", ans);
```

It's equivalent RISC-V code is as below: (Some lines are blurred.)

```
      # Blurred line 1
loop: beqz t0, end
      andi t3, t0, 1
      # This line is blurred, but we know its machine code: 0x000E0463
      # Blurred line 2
calc: mul t1, t1, t1
```

```
7         # Blurred line 3
8         j loop
9   end:  li a0, 1
10        mv a1, t2
11        # Blurred line 4
```

For this question, assume $a > 0, n > 0$, no *mulh* or *mulhu* is needed, and no overflow happens.

2 (a) Translate between assembly code and machine code. Fill in the following table:

Assembly code	Machine code
	0x000E0463
~~mul t1, t1, t1~~	

Solution:

Assembly code	Machine code
beq t3(or: x28), zero(or: x0), calc(or: 8)	0x000E0463
~~mul t1, t1, t1~~	~~0x02630333~~

You're given a task to fill back the blurred part of the code.

4 (b) First, by your indication from the C code and the parts that were not blurred, match between registers and their correspondent C expressions. Choose from those 1 to 4:

1. a 2. n 3. ans 4. n & 1

t0: _____ **t1**: _____ **t2**: _____ **t3**: _____

Solution: **t0**: n **t1**: a **t2**: ans **t3**: a & 1 Answer: 2 1 3 4

6 (c) Then you can fill back the blurred part. Write the code below:

Blurred line **1**: _____

Blurred line **2**: _____

Blurred line **3**: _____

Blurred line **4**: _____

Solution: 1. li t2, 1 (or: addi t2, x0/zero, 1) 2. mul t2, t2, t1 (or: mul t2, t1, t2)
3. srli t0, t0, 1 (or: srai t0, t0, 1) 4. ecall

8. **SDS and FSM**

[2] (a) Use A and B as input and C as output, extract the boolean expression of this truth table.

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

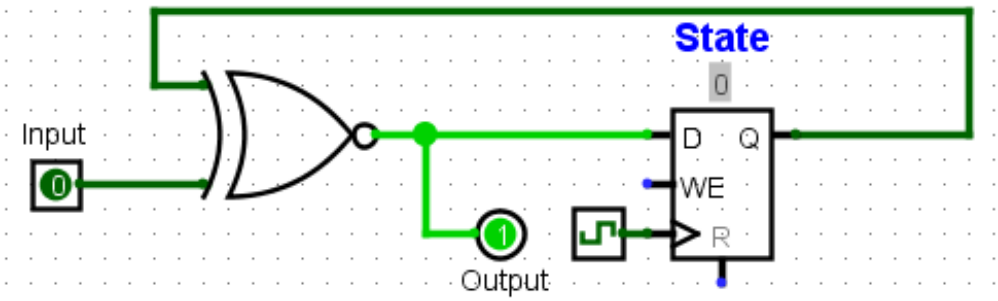
Expression:

_____

Solution:

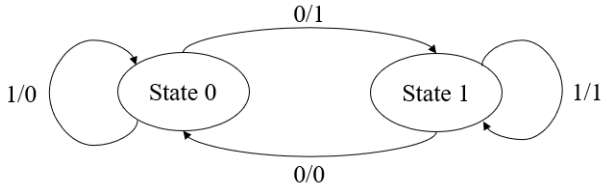
$$C = AB + \bar{A}\bar{B}$$

[3] (b) The truth table above is an XNOR gate. The register can store only one bit. Draw the FSM state diagram in the space below. The initial state is shown in the graph.



Draw the diagram below:

Solution:



2 (c) For the circuit above, assume:

- XNOR gate delay is 30 ps
- setup time is 40 ps
- Clk-to-Q time is 100 ps

If we run this circuit on a 4GHz microchip, and the input arrives t_{hold} after the clock triggers. What is the maximum allowable hold time?

Solution:

maximum $t_{hold} = t_{clk-to-q} + t_{XNOR} = 130ps$

9. **Cache**

5 (a) Consider a write-allocate, write-back, direct-mapped cache with 32 byte blocks and 2^{19} bytes of data. Assume a byte-addressed machine with 32-bit addresses.

1. What is the width of each field of following address bit assignment?

TAG	Set index	Block offset

Solution: 13 bits 14 bits 5 bits

2. What is the total number of bits (data ***AND*** cache management) that comprise the cache? You should show the process of your calculation. You may use size prefixes - use the correct ones according to the RISC-V green sheet!

Solution:
blocks: 2^{14}
data bits per block: 256
management bits per block: 13 + 1 (valid) +1 (dirty)
$2^{14} * (256 + 15)$bits = $2^{14} * 271 = 1024 * 16 * 271 = 4336$ Ki bit

8 (b) Assume we have a single-level, 1 KiB direct-mapped L1 cache with 16-byte blocks. We have 4 GiB of memory. An integer is 4 bytes. The array is block-aligned.

1. Calculate the number of tag, index, and offset bits in the L1 cache.

TAG	Set index	Block offset

Solution: 22 bits 6 bits 4 bits
Offset: $\log_2 \text{blocksize} = \log_2 16 = 4$
Index: $\log_2 \text{cachesize}/\text{blocksize} = \log_2 1\text{KiB}/16 = \log_2 64 = 6$
Tag: First find total address bits $\log_2 4\text{GiB} = \log_2 4 * 2^{30} = \log_2 2^{32} = 32$
Then $32 - \text{Index} - \text{Offset} = 32 - 6 - 4 = 22$

2. Given the following C source code, what is the hit rate? Assume C processes expressions left-to-right.

```
#define LEN 2048

int ARRAY[LEN];
int main() {
   for (int i = 0; i < LEN - 256; i+=256) {
     ARRAY[i] = ARRAY[i] + ARRAY[i+1] + ARRAY[i+256];
     ARRAY[i] += 10;
   }
}
```

Hit rate : _____

Solution: 50%
Every iteration it's
ARRAY[i] read MISS
ARRAY[i+1] read HIT
ARRAY[i+256] read MISS
ARRAY[i] write MISS (conflict! - same cache line as i+256!)
ARRAY[i] read HIT
ARRAY[i] write HIT
3 MISSES, 3 HITS. 50% hit rate.

3. You decide to add an L2 cache to your system! You shrink your L1 cache, so it now takes 3 cycles to access L1. Since you have a larger L2 cache, it takes 50 cycles to access L2. The L1 cache has a hit rate of 25% while the L2 cache has a hit rate of 90%. It takes 500 cycles to access physical memory. What is the average memory access time in cycles? You should show the process of your calculation.

Solution: AMAT = 3 + 3/4 * (50 + 1/10 * 500) = 78

4 (c) We are given the task of counting the number of even and odd numbers in an array, A, which only holds integers greater than 0. Using a single thread is too slow, so we have decided to parallelize it with the following code:

```
#include <stdio.h>
#include "omp.h"
void count_eo (int *A, int size, int threads) {
   int result[2] = {0, 0};
   int i,j;

   omp_set_num_threads(threads);

   #pragma omp parallel for
   for (j=0; j<size; j++)
      result[A[j] % 2] += 1;

   printf("Even: %d\n", result[0]);
   printf("Odd: %d\n", result[1]);
}
```

As we increase the number of threads running this code:

1. Will it print the correct values for Even and Odd? Explain your answer.

Solution: No, there may be a data race.

2. Can there be false sharing if the cache block size is 8 bytes? _____

Solution: Yes.

3. Can there be false sharing if the cache block size is 4 bytes? _____

Solution: No.

6 (d) Consider an array of following `location` structs:

```
typedef struct {
    ... // some undefined number of other struct members
    int visited;
    int danger;
} location;

location locs[NUM_LOCS];
```

Here's a piece of code that counts the number of places we've visited. Assume this gets executed somewhere in the middle of our program, that `count` is held in a register, and the size of the array is 4KiB.

```
for (int i = 0; i < NUM_LOCS; i++)
    if (locs[i].visited) count++;
```

1. In the best case, what could the number of bytes written to main memory be?

   _____

2. What is the greatest possible number of bytes written to main memory?

   _____

Solution: 0B 4KiB

Now consider if we store the `visited` and `danger` information in individual arrays instead:

```
int visited[NUM_LOCS];
int danger[NUM_LOCS];
```

3. This way, the cache can exploit better _____ for the above task.

4. We can expect a _____ (higher or lower) miss rate because of the change in the number of _____ (type of cache miss) misses.

Solution:

3. spatial locality
4. lower compulsory

10. **Virtual Memory**

[2] (a) **(Multiple choice)** Which of the following things are the Paging capable of while segmenting (base and bound) does not? Circle the letter of the your choice(s).

A. location independent programming
B. run programs larger than DRAM
C. protection and privacy
D. no (external) memory fragmentation

Solution: B, D

[2] (b) **(Multiple choice)** TLB (Translation Look-aside Buffer) is a cache for page table entries used to speed up the address translation. Because a TLB contains only information of the VPN (virtual page number) and its corresponding PPN (physical page number), when must we flush it to keep it consistent with the current page table?

A. After a program allocated some new memory, new page table entries are added to its page table.
B. Context switch. (i.e. after the current program is paused to let another program to be executed)
C. After written to a memory page which is not in DRAM. (i.e. the first access to that memory page)
D. After a page currently in DRAM is written back to disk because free page is required and no unused page is left.

Solution: B, D

11. **OS, I/O and DMA**

[2] (a) We know PIO (programmed I/O) is not ideal in some cases. Which of the following statements are the weakness(es) of PIO? Circle the correct answer(s).

A. CPU has to execute all transfers, but it could be doing other work.
B. We need to consider how to arbitrate between hardware engine/ device and CPU.
C. We have a high energy cost of using beefy general-purpose CPU where simpler hardware would suffice.
D. Device speeds may not align well with CPU speeds.

Solution: ACD.

[3] (b) **DMA** needs a new hardware, what is its name? Then, briefly explain the main difference between PIO and DMA.

Solution: New hardware: the DMA Engine
Difference:
PIO: CPU has sole control of main memory
DMA: Allows I/O devices to directly read/write main memory

3 (c) Recall the 4 parts of Disk Access Time mentioned in class, now assume we have a disk which spins 7,200 times per minute (7,200 Revolutions Per Minute). And its average seek time is 4ms, data transfer rate is 20MB/s. Also, the controller overhead is ignored in this problem. What is the average access time of reading a block sector with size 4KB from this disk?

Solution:

$$T_{rotation} = \frac{60}{7200} * \frac{1}{2} = 4.17 * 10^{-3}s = 4.17ms$$

$$T_{transfer} = \frac{4KB}{20MB/s} = 0.2ms$$

$$T_{access} = T_{rotation} + T_{transfer} + T_{seek} = (4.17 + 0.2 + 4)ms = 8.37ms$$

12. **FPGA, IOs and Dependability, Warehouse-Scale Computing**

3 (a) Please choose the correct descriptions from below:

A. `Embedded System are designed to perform predefined functions on certain platforms.`

B. `Embedded Systems usually provide limited functions and have fewer components than Computer Systems.`

C. `Embedded Systems perform functions fast and thus consume much power.`

D. `Compared with Processors, FPGAs have a higher performance but poorer flexibility.`

E. `ASIC can not be changed once designed, but it is suitable for high-volume mass production.`

Solution: A, B, E

3 (b) True or False Questions, please circle the correct answer. **Notice: NO selection will be treated as a wrong choice.**

T / F: If every machine of one data center employs hard disk drives configured in RAID 1, there is no need to maintain machine-level replicas.

T / F: Static web servers use request-level Parallelism while MapReduce is Data-level Parallelism.

T / F: It can be said that RAID 10 does not fulfill fault tolerance nor redundancy.

T / F: Flash memory can be used as a disk cache to a magnetic hard disk.

T / F: Inefficient load balancing in a warehouse-scale computer may lead to higher energy consumptions.

T / F: MapReduce is well capable of processing data at large scale, so it is suitable to compute the large-scale Fibonacci series.

Solution: F T F T T F

No question here!