Computer Architecture I Mid-Term II

Chinese Name: _____

Pinyin Name: _____

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

Question	Points	Score
1	1	
2	15	
3	13	
4	10	
5	10	
6	18	
7	20	
8	13	
Total:	100	

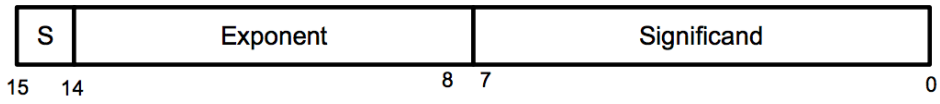
- This test contains 17 numbered pages, including the cover page, printed on both sides of the sheet.
- We will use gradescope for grading, so only answers filled in at the obvious places will be used.
- Use the provided blank paper for calculations and then copy your answer here.
- Please turn **off** 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.

- 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.
- The estimated time needed for each of the 7 topics is given in parenthesis - roughly about 1 minute per point. The total estimated time is 110 minutes.
- 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 17 times).

2. **Floating Point Numbers** (15 points, 15 minutes)

We want to develop a new half-precision floating-point standard for 16-bit machines. The basic structure is as follows:



Here are the design choices:

- 1 bit for sign
- 7 bits for the exponent
- 8 bits for the significand
- Everything else follows the IEEE standard 754 for floating point, except bias.

2 (a) Following the choices of the bias of the IEEE 754, what should the bias for this 16 bit float be?

2 (b) Convert the decimal number -10.625 to our 16 bit floating point number. Write your answer in hexadecimal.

2 (c) What is the smallest non-infinite **even** positive integer that is **NOT** representable?

3 (d) What is the smallest positive denormalized number?

3 (e) How many non-positive real numbers can be represented?

3 (f) Convert the floating point number 0xCA60 to a decimal number.

Solution: a) 63
b)
-1010.101
sign: 1
exponent:3+63=66
significand: 01010100
0xc254

b)
$2^{10} + 2$

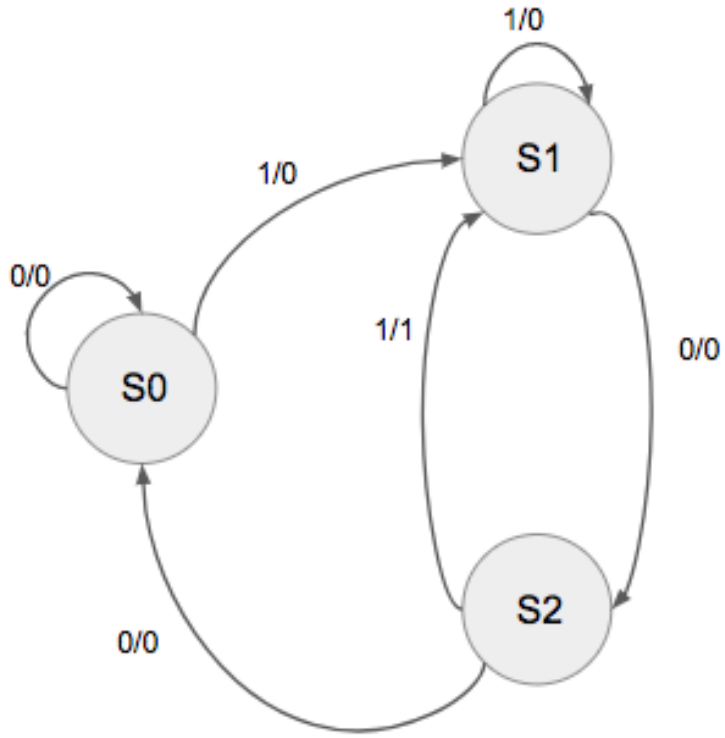
c) 2^{-70} or 0x4001

d) $2^{15} - 2^{8}$

e) $-1.375 * 2^{11}$

3. **FSM and SDS** (13 points, 15 minutes)

Here is a predefined finite state machine with 3 states in total:



Jeff is flipping coins infinitely, and such FSM is created to help him detect certain **consecutive** outcome pattern. When the outcome is H(head), the FSM gets input 1. When the outcome is T(tail), the FSM gets input 0. And when certain pattern occurs, the FSM output 1. (So he may

wait on pattern 001 which means a sequential outcome: TTH)

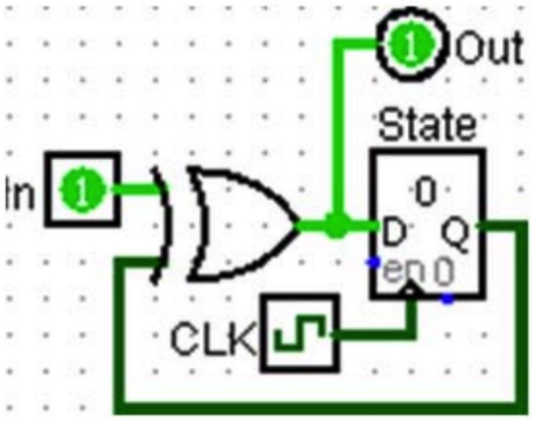
3 (a) Breifly explain which 3-bit pattern makes this FSM output 1 at the first time. (start at empty state S0):

Solution: Pattern: 101 or HTH

2 (b) Now the FSM starts at $S0$, and read binary input from left to right.
How many times does this FSM output 1 given the input 1001101010011(start at empty state S0):

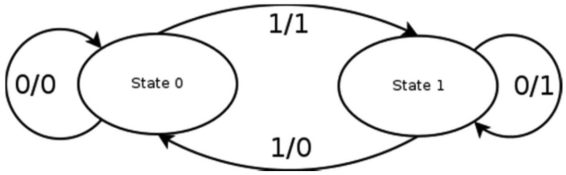
Solution: 2 times. See the bits: 10101.

4 (c) Draw the FSM state diagram based on the following figure (assume the initial state shown) in the space below:



Requirements:
Specify the input/output on each transition edge. Input must be on the left of Output.
Each transition edge represent exactly 1 (input/output) combination.
Assume the CLK will keep ticking.

Solution:



4 (d) Let $t_{\text{setup}} = t_{\text{hold}} = 40$ ps and $t_{\text{XOR}} = 30$ ps. If we run the above ciucuit on a 5-GHz processor and the input arrives t_{hold} after the clock triggers, what are the maximun and minimum $t_{\text{clk-to-q}}$ for the register to ensure proper functionality?

Min: _____ Max: _____

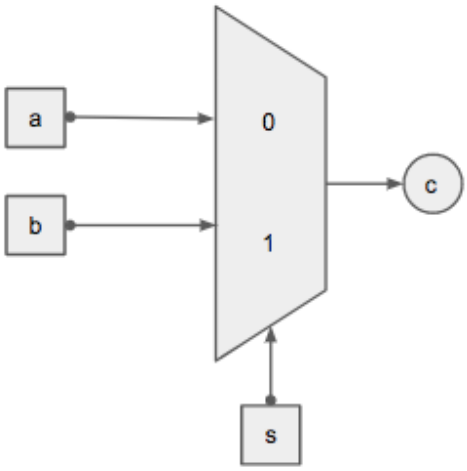
Solution: $t_{\text{hold}} \leq t_{\text{clk-to-q}} + t_{\text{XOR}} \leq T - t_{\text{setup}}$, because the input must be stable after the clk ticks and the Q propagate back to D
Min: 10ps, Max:130ps

4. **Logic** (10 points, 10 minutes)

Hopefully you still remember multiplexers. Please see the simple 1-bit-wide mux below. On the left side, **a** and **b** are input pins, **s** serves as select bit and **c** is output pin.

3 (a) Please fill in the truth table presented.

s	a	b	c
0	1	1	
	1	0	
	0	1	
	0	0	
1	1	1	
	1	0	
	0	1	
	0	0	



Solution:

s	a	b	c
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

3 (b) According to the truth table, please write down the boolean algebra expression on c with respect to a, b and s (directly from the truth table, no simplifications).

Solution: $c = \bar{s}ab + \bar{s}a\bar{b} + sab + s\bar{a}b$.

4 (c) Start from your boolean algebra expression above and simplify it (write down the intermediate steps!).

Solution: $c = \bar{s}a + sb$

5. **MIPS pipeline** (10 points, 10 minutes)

Compare two pipeline implementation options A and B with 4 and 7 stages, respectively.

2 (a) The logic delays of the pipeline stages are as follows:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	250 ps	180 ps	400 ps	200 ps	-	-	-
Option B	200 ps	150 ps	250 ps	250 ps	200 ps	150 ps	180 ps

What are the maximum clock rates for the two implementations?

Option A fs_max = _____ (include the unit with your result)

Option B fs_max = _____ (include the unit with your result)

6 (b) The table below states the operation of each pipeline stage:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	IF/ID	EXE	MEM	WB	-	-	-
Option B	IF	ID	EXE-1	EXE-2	EXE-3	MEM	WB

Compared to the MIPS CPU, option A merges IF and ID in a single stage, while option B splits EXE over three pipeline stages. Registers and memory are written to in the first half of the cycle and read during the second half of the cycle (same as MIPS) but there are no forwarding paths. How many instructions are executed after an **add** and a **lw** instruction, respectively, before the new register values can be used by the next instruction?

	# nops after add	# nops after lw
Option A	_____	_____
Option B	_____	_____

2 (c) Calculate the number of instructions executed per second for each implementation for the specified clock rate (these are not necessarily the correct results for part a) and CPI.

	clock rate	CPI	Instructions per second
Option A	3 GHz	1.5	_____
Option B	5 GHz	2.0	_____

Solution: a) Option A fs_max = _____1/0.4ns = 2.5GHz_____ (include the unit with your result)
Option B fs_max = _____1/0.25ns = 4GHz_____ (include the unit with your result)

b)
Option A (both columns): 4
Option A (both columns): 2

c) 1/CPI * 1/f = 1/CPI * Cycles / sec

	f_s	CPI	Instructions per second
Option A	3 GHz	1.5	$(1/1.5)(1/((1/3)*10^{-9}))= 2*10^9$
Option B	5 GHz	2.0	$(1/2.0)(1/((1/5)*10^{-9})) = 2.5*10^9$

6. **Pipeline Hazards** (18 points, 20 minutes)

   The goal of the problem is to increase the execution speed of the code below by eliminating as many stalls and useless operations (**nop**s in the branch delay slots) as possible. The code runs on a 5-stage pipelined MIPS CPU with forwarding with the characteristics discussed in lectures.
   Note: the branch delay slot is not hidden, i.e. **nop**s in the code below are always executed regardless of the branch decision.

   The following table shows the code you'll be working with:

Table 1: Pipeline Hazard

loop:	sltiu $t0, $s0, 100	loop:	_____
	beq $t0, $0, exit		_____
	nop		_____
	lw $t0, 0($s1)		_____
	addiu $t0, $t0, 1		_____
	lw $t1, 0($s2)		_____
	addu $t0, $t0, $t1		_____
	sw $t0, 0($s2)		_____
	addiu $s0, $s0, 1		_____
	addiu $s1, $s1, 4		_____
	addiu $s2, $s2, 4		_____
	beq $0, $0, loop		_____
	nop		_____
exit:		exit:	_____

4 (a) Indicate stalls in the code above (Table 1) with arrows right to the instructions after which the stall occurs. The following is not necessarily correct and only used as an example:

loop:	sltiu $t0, $s0, 100
→	beq $t0, $0, exit
	nop

3 (b) How many cycles does it take to execute the entire code sequence above (Table 1), including stalls and **nop**s?

$$N_1 = \_\_\_\_\_ cycles$$

3 (c) Assuming all stalls and nops can be eliminated (also for the code above, Table 1), by how many cycles does the execution time **decrease**?

$$N_2 = \_\_\_\_\_ cycles$$

8 (d) Rewrite the code in Table 1, eliminating as many stalls and nops as possible. The improved code must store the same results to memory, but register values may differ between the two versions when exit is reached. Fill in the right side of Table 1.

Solution:

b)

19 or 20 (depends on if you observed the stall between sltiu and beq)

c) N2 = 4 or 5 (depends on if you observed the stall between sltiu and beq) cycles

a) and d)

Original Code Sequence

loop:	sltiu $t0, $s0, 100
	beq $t0, $0, exit
	nop
	lw $t0, 0($s1)
➔	addiu $t0, $t0, 1
	lw $t1, 0($s2)
➔	addu $t0, $t0, $t1
	sw $t0, 0($s2)
	addiu $s0, $s0, 1
	addiu $s1, $s1, 4
	addiu $s2, $s2, 4
	beq $0, $0, loop
	nop
exit:	

Improved Code Sequence

loop:	sltiu $t0, $s0, 100
	beq $t0, $0, exit
	addiu $s0, $s0, 1
	lw $t0, 0($s1)
	lw $t1, 0($s2)
	addiu $t0, $t0, 1
	addu $t0, $t0, $t1
	sw $t0, 0($s2)
	addiu $s1, $s1, 4
	beq $0, $0, loop
	addiu $s2, $s2, 4
exit:	

7. **Caches !** (20 points, 25 minutes)
This C code runs on a 32-bit MIPS machine with 4 GiB of memory and a single L1 cache. Vectors A and B live in different places of memory, are of equal size (n is a power of 2 and a [natural number] multiple of the cache size), block aligned. The size of the cache is C, a power of 2 (and always bigger than the block size, obviously)

```
1   // sizeof(uint8_t) = 1
2   Swap(uint8_t *A, uint8_t *B, int n) {
3     uint8_t tmp;
4     for (int i = 0; i < n; i++) {
5
6       tmp = A[i]; //_____________________
7
8       A[i] = B[i]; //_____________________
9
10      B[i] = tmp; //_____________________
11
12                  //_____________________
13    }
```

```
14          }
```

Consider **Swap** code for (a) and (b). And for all hit:miss ratio, you should write in format "XXX:YYY", where "XXX" and "YYY" can be hit number and miss number during executing. You can simplify them, they may contain symbols.

3 (a) If the cache is direct mapped and the best (hit more) hit:miss ratio is H:1, what is the block size in bytes?

Solution: (H+1)/2

3 (b) What is the worst hit:miss rate?

Solution: 0:4n (0:any none-zero is ok)

4 (c) Fill the code in the comment lines so that it does the same thing as **Swap** inner for loop. but **improves** the (b) hit:miss rate(hit more). You may not need all the blanks.

Solution:

```
1       tmpA = A[i];
2       tmpB = B[i];
3       B[i] = tmpA;
4       A[i] = tmpB;
```

4 (d) If the block size is a bytes, what is the worst hit:miss ratio for your improved **Swap** ?

Solution: 2a-1: 2a+1

6 (e) We next change the cache to be 2-way set-associative, and lets go back to just considering **`Swap`** with out improvement. What is the worst hit:miss rate (hit fewest) for **`Swap`** with the following replacement policies? The cache size is `C` (bytes), the block size is `a` (bytes), LRU = Least Recently Used, MRU = Most Recently Used.

1. LRU and an empty cache

2. MRU and a full cache

Solution: 1. `2a-1 : 1` 2. `0 : 4n` (`0`: any-non-zero is ok)

8. Parallel Computation (13 points; 15 min)

[5] (a) Name all elements of the Flynn Taxonomy and provide a short description for each element and name an example if you can.

Solution: Single Instruction Single Data (SISD): one instruction after the other; example: MIPS CPU
Single Instruction Multiple Data (SIMD): one instruction can work on more than one data elements; example: SSE or MMX, AVX
Multiple Instruction Single Data (MISD): multiple (different) instruction working with the same data; no example
Multiple Instruction Multiple Data (MIMD): Different instructions at the same time, with different data; example: multi-core CPUs

[1] (b) Provide the formula for Amdahl's Law.

(b) **1 / (1 - F) + F / S)**

Solution: Initial: 1/ ((1 - 0.9 + 0.9 / 36) = 1 / (0.1 + 0.025) = 8
16384 cores: 1 / (1 - 0.9 + 0.9 / 16384) approximately 1 / (0.1 + 0) = 10
Improvement: 10/8 = 1.25 == 25% speedup only

[3] (c) What is loop unrolling? What are the advantages of it (name at least 3)?

Solution: Take a for loop in C and explicitly write a certain number of those loop iterations in the body of the loop. Advantages:
less loop overhead; it can avoid data hazards; SIMD instructions can be used.

[2] (d) What are SSE compiler intrinsics? What enable they us to do?

Solution: Those are C instructions that are translated directly to SSE assembler instructions, enabling us to put SSE instructions in C code.

2 (e) What does the following instruction do exactly? Please also write C-pseudocode for this instruction!

```
mulpd %xmm0, %xmm1
```

Solution: Multiply xmm0 with xmm1 and save the result in xmm1. xmm0 and xmm1 hold both 2 double floating point numbers. C example:
xmm1[0] = xmm0[0] * xmm1 [0];
xmm1[1] = xmm0[1] * xmm1 [1];