Computer Architecture I Final

Chinese Name: ____________________

Pinyin Name: ____________________

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

Question	Points	Score
1	22	
2	14	
3	20	
4	10	
5	8	
6	10	
7	16	
Total:	100	

- This test contains 10 numbered pages, including the cover page. The back of each page is blank and can be used for scratch-work, but will not be looked at for grading.
- Put your pinyin name on the top of every page.
- 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 under your seat.
- You have 75 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use three A4 pages (front and back) of handwritten notes in addition to the provided green sheet.
- The estimated time needed for each of the topics is given in parenthesis. The total estimated time is 75 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.
- Answer all questions in English. Answers in Chinese get 50% of the score deducted.

1. SDS and Pipline (20 minutes)

   The following sub-questions will use the circuit shown in Figure 1 below.

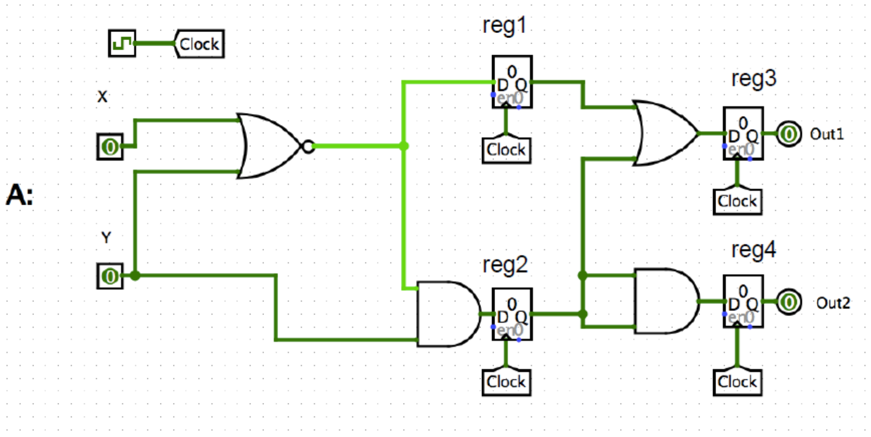

Figure 1

Circuit specs:

- 15ns clk-to-Q time
- Negligible hold time
- 20 ns logical gate propagation delay
- Gates with bubbles on the output have the same delay as ”normal” gates
- Assume that X and Y arrive at the positive edge of the clock

$\boxed{6}$ (a) Given Circuit A, with a clock period of 100ns, determine the maximum theoretical setup time we can have for this circuit to satisfy register-timing constraints (and function correctly). With this setup time, if signals X and Y are undefined until t = 0, when would Out1 be stable with the correct computed value?
Max setup time: 60ns = 100 - MAX(CriticalPath1 + CriticalPath2)
Out1 stable at t= 215ns = 2 clock cycles + CLK-to-Q

$\boxed{8}$ (b) Now, you need to pipeline the circuit to improve the clock period. **Draw a star on any wire in Figure 1 where you would place a pipelining register.** You may place up to 3 registers (but you may not need all 3). Your solution may introduce a cycle delay, but should not change the sequence of outputs after that initial delay (assuming the circuit gets a single uninterrupted stream of incoming X and Y values). Write a short explanation below:

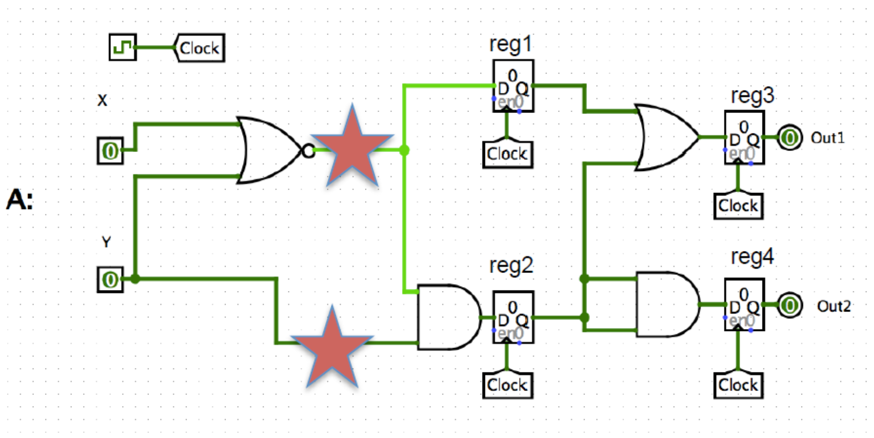

Figure 2

$\boxed{8}$ (c) Now, assume we have a Pipelined 5-stage MIPS CPU with the following specs:

- The CPU stalls on hazards, there is no forwarding
- Branch comparison happens in stage 2 and we **DO NOT** have a branch delay slot. (i.e. the branch "decision" is clocked into the PC at the end of stage 2).
- Both memory and registers **CAN** be written and read in the same clock cycle
- All Loads and Stores hit in the cache (ie. loads/stores take one cycle in the Mem stage)

Fill in the corresponding pipeline stages (F, D, E, M, W) at the appropriate times in the table below for the following 8 MIPS instructions assuming the above properties of your CPU. Suppose that any branches in the code are not taken for this specific instance. (You should use the back of the page for scratch work).

Instr\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
sll $v1 $v1 8	F	D	E	M	W												
xor $v0 $a1 $a2		F	D	E	M	W											
addu $a1 $s3 $t1			F	D	E	M	W										
andi $v0 $v0 1				F	D	D	E	M	W								
addu $t0 $a0 $t1					F	F	D	E	M	W							
lw $s0 0($t0)							F	D	D	D	E	M	W				
bne $s0 $0 end								F	F	F	D	D	D	E	M	W	
j taketwo											F	F	F	D	E	M	W

2. Control and Datapath (14 minutes)

Modify the following single cycle MIPS datapath diagram to accomodate a new instruction $swai$ (store word then auto-increment). The operation performs the regular sw operation, then post-increments the rs register by 1. Your modification may use simple adders, mux chips, wires, and new control signals. You may replace original labels where necessary. Recall the RTL for sw is: $Mem[R[rs] + SignExt[imm16]] = R[rt]$; $PC = PC + 4$, & that sw (and $swai$) has the following fields:

Opcode	Rs	Rt	Immediate

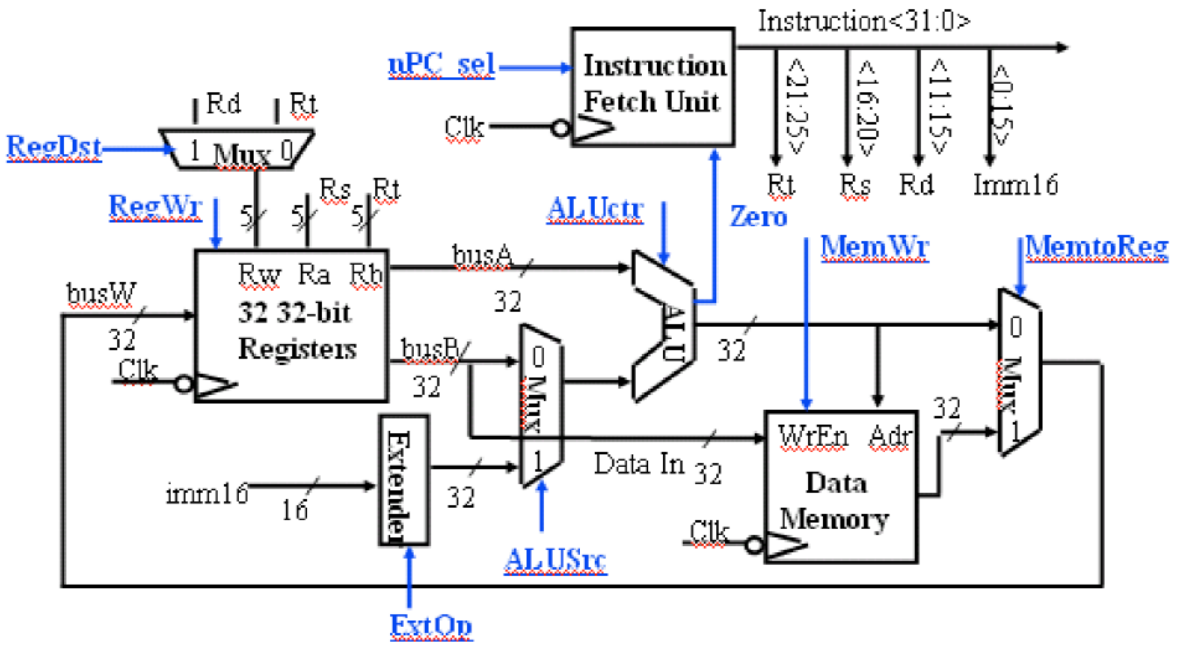

Figure 3: Datapath

$\boxed{10}$ (a) **Modify the picture above** and list your changes below. You may not need all lines. Write them in "pipeline stage order" (i.e., changes affecting IF first, MEM next, etc.)

1. Add a small adder with one input tied to 1, the other tied to busA, & output tied to the MemToReg mux.
2. Change the 2-input MemToReg mux to be a 3-input mux (adding output of adder)
3. Change the width of MemToReg from 1 to 2 (and the corresponding logic)
4. Change the RegDst-controlling 2-input Rd/Rt mux to be a 3-input Rd/Rt/Rs mux
5. Change the width of RegDst from 1 to 2 (and the corresponding logic)

$\boxed{4}$ (b) We also wish to do the same thing with lw, namely create lwai. Will this work? Please argue your point in one sentence.

NO because we can not write two registers at once

3. MIPS Mystery (14 minutes)

   We present `mystery`, a new helper routine for your C programming.
   In parts (a) and (b), you'll show us you know how to use `mystery` form C.
   In parts (c) and (d), you'll show us you understand its limitations.

```
mystery:  ori   $v0 $0 0x0
          beq   $a0 $0 done
          la    $t0 mystery
          lw    $t1 0($t0)
          addiu $t1 $t1 0x1
          sw    $t1 0($t0)
          lw    $v0 0($a0)
done:     jr    $ra
```

```
main() {
    char A[4], char4 = '4';
    int pi[] = {3, 4, 1, 5, 1};
    float float4 = 4;
      // part(a)
     // more code, ...
}
```

$\boxed{2}$ (a) If you are at "part (a)" in the C code, show a single call to mystery so that it returns 4.

`printf("Here is mystery returning four...%d\n", mystery(`

&pi[1] // or pi+1 `));`

$\boxed{6}$ (b) Complete the documentation of `mystery` for a fellow programmer. Use good abstraction–don't tell the user how it does what it does, just tell them what it does and how it's to be used.

"When called with a *non-NULL* argument, the subroutine `mystery`...

increments its internal counter and returns the data at arg's memory address "

"When called with *NULL* argument, the subroutine `mystery`...

returns its internal counter "

"Overall, `mystery` is a subroutine used to ...

record and report the number of memory reads "

$\boxed{2}$ (c) We'd like to know if there is a limit to the # of times `mystery` can be called with a *NULL* argument (so that it still does what you described in part b). If there is , state what the limit is and what happens if it's called one more time. If there isn't a limit, write N/A (not applicable) in both blanks.

"With a NULL argument, `mystery` may be called at most N/A times, Calling it once more...

N/A "

$\boxed{2}$ (d) We'd like to know if there is a limit to the # of times `mystery` can be called with a *non-NULL* argument (so that it still does what you described in part b). If there is, state what the limit is and what happens if it's called one more time. If there isn't a limit, write N/A (not applicable) in both blanks.
"With a non-NULL argument, `mystery` may be called at most $\underline{2^{16}-1}$ times, Calling it once more...

changes mystery so that it resets its counter to 0 but now puts the return value into v1 instead of v0, so you could not use mystery anymore since v0 would be garbage

$\boxed{4}$ (e) Translate the following assembly into machine code.

Instruction	Code
xor \$s0,\$s0,\$s0	

Solution:

Instruction	Code
xor \$s0,\$s0,\$s0	0x02108026

$\boxed{4}$ (f) What is the difference between 'jr', 'jal' and 'j' ?

Solution:
j:Jumps to the target offset
jal:Executes the next line, then jumps to the target offset, then changes the return value (ra) value so it jumps to the offset after the delay slot.
jr: Jumps to the offset stored in a register.

4. C Programming (10 minutes)

$\boxed{4}$ (a) **Write C Code in One Line**. Complete the following C function according to comments. You may assume that the architecture on which this code is run uses two's complement to represent signed integers. Zero points if you use `if/else` or `?/:`. Suggestion: *Use parentheses to explicitly denote the order of operations.*

```
/*
* Returns an unsigned integer whose first IDX low order bits are
* the same as that in LO , and whose remaining high order bits
* are the same as that of HI. Assume 0 <= IDX < sizeof ( int ).
*/
unsigned splice ( unsigned lo , unsigned hi , int idx ) {
  return (lo & (1 << idx) - 1) | (hi & -1 << idx);
}
```

6 (b) **Thread Level Paralielism**. Consider each of the following code segments and determine which of the following statements is true about the correctness/performance of the given code, and *a brief explanation of one or two sentences is a must*. Assume the default number of threads is greater than 1 and no thread will complete before another thread starts executing.

1. Sometimes incorrect
2. Always incorrect
3. Slower than serial
4. Faster than serial

(a) Set all elements in arr to 0.

```
int i;
#pragma omp parallel for
for (i = 0; i < len; i++) {
  arr[i] = 0;
}
```

Solution:
Faster than serial ? for directive actually automatically makes loop variable private, so this will work properly. Justification needed to mention that the for directive splits up the iterations of the loop into continuous chunks for each thread, so no data dependencies or false sharing.

Also accepted:
Sometimes incorrect ? variable i declared outside of parallel section, so each thread accesses the same variable (not true in practice). This can cause iterations of i to be skipped if consecutive i++ calls are made by different threads. Still possible to alternate storing and incrementing across all threads, so can still be correct sometimes.

(b) Set element i of arr to i.

```
#pragma omp parallel
for (int i = 0; i < len; i++) {
  arr[i] = i;
}
```

Solution:
Slower than serial ? there is no for directive, so every thread executes this loop in its entirety. 3 threads running 3 loops at the same time will actually execute in the same time as 1 thread running 1 loop, so credit for justification was only given if there was a mention of parallelization overhead or possible false sharing.

(c) Set arr to be an array of Fibonacci numbers.

```
arr[0] = 0;
arr[1] = 1;
#pragma omp parallel for
for (int i = 2; i < len; i++) {
  arr[i] = arr[i - 1] + arr[i - 2];
}
```

Solution:
Always incorrect ? Loop has data dependencies, so first calculation of all threads but the first one will depend on data from the previous thread. Because we said ?assume no thread will complete before another thread starts executing,? then this code will always be wrong from reading incorrect values.

Also accepted:
Sometimes incorrect ? This code will execute correctly for len ¡ 4. Otherwise, it executes incorrectly for the reasons listed above.

5. Cache (5 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).

```
// sizeof(uint8_t) = 1 byte (12Ish)
swapLeft (uint8_t *A, uint8_t *B, int n) {
 uint8_t tmp; // assume the compiler will use a register for tmp
 for (int i = 0; i < n; ++i)
```

```
    {
      tmp = A[i];
      A[i] = B[i];
      B[i] = tmp;
    }
  }
```

$\boxed{4}$ (a) If the cache is direct mapped and the best hit:miss ratio is "H:1", what is the block size in bytes?

(a) **(H+1)/2**

$\boxed{4}$ (b) Assuming a block size of 64 bytes, a cache size of 64KiB and a value for n of 65,536 and that we have just finished filling array B with values and no other processes are being executed. In the optimal case, how many cache misses must we have (i.e.: what is the lowest number of cache misses we can have)?

(b) **1536 explanation:**

Array A&B: size is 64KiB => we need (will load) 1024 cache lines per array. Each cache line will result in a miss. But B is already in the cache! In the best case A will start evicting cache lines 512 and following, while B will start with cache line 0. So:
1024 (number of cache lines needed for A) +1024 (number of cache lines needed for B) - 512 (cache lines already in cache of B - 512th and following will have been evicted for A) = 1536

6. Virtual Memory (8 minutes)
For the following question, assume that the machine in question has the following parameters:

- 64-bit virtual addresses
- 48 bit physical addresses
- 4KiB pages
- Fully associative TLB with 128 entries and LRU replacement
- Unlimited swap space

$\boxed{2}$ (a) How much physical memory can the machine support at most (in KiB, MiB, GiB or TiB)?

(a) 2^{48} **bytes = 256 TiB**

$\boxed{2}$ (b) What is the maximum number of pages that a process can use (use of power of two is ok)?

(b) 2^{52} **pages**

$\boxed{2}$ (c) How many bytes of main memory can be at maximum covered by the TLB (in KiB, MiB, GiB or TiB)?

(c) **128 * 4KiB = 512KiB**

$\boxed{4}$ (d) Given the program below, and assuming that the text and data segment are continuos and have a size of less than 2KiB, how much memory will the execution of this program need? Why?

```
#include <list>
int main(){
  std::list<int> li;
  li.push_back(3);
  return li.size();
}
```

(d) **<u>8KiB: 2 pages - one for text (and heap) and for stack (starts from the top)</u>.**

Also OK: 12KiB: 3 pages - one for text, one for stack, one for heap. Heap: list.push_back will create a new list item in the heap.

7. T/F Questions (Circle one. If the circling is unclear, you will receive no credit.) (4 minutes)

$\boxed{2}$ (a) Exceptions in early pipeline stages override exceptions in later stages for a given instruction. (True / False) A: True

$\boxed{2}$ (b) Exceptions are handled in the pipeline stage where they occur. (True / False) A: False

$\boxed{2}$ (c) CPUs need separate instructions to access I/O devices. (True / False) A: False

$\boxed{2}$ (d) Segmentation (base + bound) has fragmentation problems. (True / False) A: True

$\boxed{2}$ (e) RAID: Availability will be increased by increasing MTTF (True / False) A: False

$\boxed{2}$ (f) RAID: Availability will be increased by decreasing MTTR (True / False) A: True

$\boxed{2}$ (g) RAID: Availability will be increased by Redundant data copies (True / False) A: True

$\boxed{2}$ (h) RAID: Availability will be increased by using RAID 0 instead of RAID 1 (True / False) A: False