Computer Architecture I Final

Chinese Name: ______________________

Pinyin Name: ______________________

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

Question	Points	Score
1	1	
2	30	
3	10	
4	9	
5	4	
6	16	
7	13	
8	13	
9	14	
Total:	110	

- This test contains 20 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 three A4 pages (front and back) of handwritten notes in addition to the provided green sheet.
- The estimated time needed for each of the 9 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.

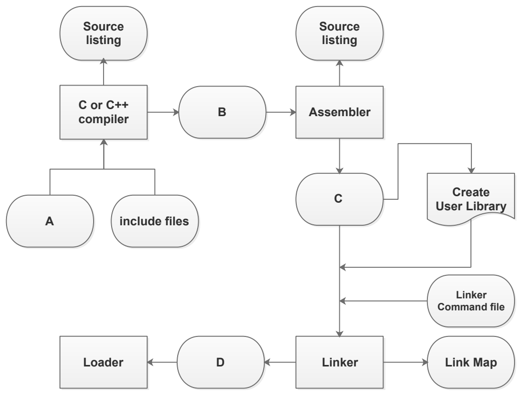
1 1. First Task (worth one point): Fill in you name
Fill in your name and email on the front page and your ShanghaiTech email on top of every page (without @shanghaitech.edu.cn) (so write your email in total 20 times).

2. **Miscellaneous Questions** (30 points, 30 minutes)

$\boxed{2}$ (a) **CALL** This is one program's life period (before killing). You must know CALL. Select which file is described in the table below (fill in A, B, C or D).

File	File location: A, B, C or D
1. absolutable object module:	
2. c/c++ source file:	
3. assembly source file:	
4. relocatable object module:	

Solution: 1.: D; 2.: A; 3: B; 4.: C;



Floating Point Numbers

$\boxed{2}$ (b) Convert the following single precision IEEE 754 floating point number to a decimal number:
1 1000 0011 0101 1010 0000 0000 0000 000

(b) -21.625

$\boxed{2}$ (c) Convert the following decimal number to single precision IEEE 754 floating point format (please leave a space for every four bits as the above problem):
9.25

(c) **0 1000 0010 0010 1000 0000 0000 0000 000**

Cache and virtual memory paging

In operating systems, there is a concept called `Demand Paging`, an algorithm in virtual memory paging. The paging process may come out of page faults, so we also have some thing like AMAT called EAT (Effective Access Time). The formula to calculate this is $\texttt{EAT} = (1-p) * \texttt{AMAT} + p * \texttt{PAGE FAULT TIME}$. Assume our computer paging system has the fault rate $p = 0.01\%$ and page fault time = $15ms$. Also consider a multilevel cache architecture.

- $L1 hits in 1 cycle (local miss rate 25%)
- $L2 hits in 10 cycles (local miss rate 40%)
- $L3 hits in 50 cycles (global miss rate 6%)
- Memory hits 100 cycles
- the cpu is 1MHZ

Please calculate the AMAT and EAT of our computer. Show progress, otherwise you will get 0 points !

3 (d) AMAT (in ns) :

1 (e) EAT (formula with values, no end result needed) :

Solution: AMAT (cycles) = 1 + 25% * (10 + 40% * (50)) + 6% * 100 cycles
AMAT (ns) = 14.5 * 10 ns ;
EAT = 0.0001*(AMAT + 15ms)

Meltdown

4 (f) Shortly explain (on a high level - we are mainly looking for the correct keywords) how meltdown works.

Solution: Keywords: Cache, timing, speculative execution, memory paging, OS pages, data of another process (4 of those keywords are enough).
We want to read from OS pages that hold data of other processes. Those are in our Virtual Memory space in order for the OS to process them quickly. But it is forbidden for our process to read them. If we try to read them the OS will check (with an ”if”) if we are allowed to read them and thus return with a page fault. But speculative execution will nevertheless cause those pages to be read (but the process doesn’t see the result).

We use this to take the to-be-read value and use it as an offset in an array. We later check all the array if it is in the cache (using timing) - if so we know (which range of) value the variable that we were not supposed to read had.

DMA, Dependency and RAID

$\boxed{2}$ (g) Please briefly explain what **DMA** is in 2 sentences.

Solution: Allows I/O devices to directly r/w main memory, requires hardware support: DMA engine

$\boxed{2}$ (h) During the DMA procedure, which of them may raise an interrupt when handling the **incoming data**? **circle** them out.

A. CPU
B. I/O Device
C. DMA engine
D. ALU

Solution: BC

2 (i) Which of the following will decrease the availability? **Circle** them.

A. decrease MTTF
B. increase MTTF
C. decrease MTBF
D. increase MTBF
E. decrease MTTR
F. increase MTTR

Solution: AF

4 (j) True or False Questions, please **circle** the correct answer.
T / F: RAID 1 will result in slower read but it has very high availability
T / F: RAID 1 is the most expensive architecture among RAID 1,3,4,5
T / F: RAID 3 uses hamming ECC to check and correct the data
T / F: RAID 4 is still slow on small writes

Solution: FTFT

Warehouse Scale Computing

1 (k) In the course, we discussed two parallelism strategies in WSC. Name them.

____

Solution: Request-level, Data-level

2 (l) Let's answer some questions from the class. (Circle T or F)

1. T / F: Idle servers consume almost no power.
2. T / F: Disks will fail once in 20 years, so failure is not a problem of WSC.
3. T / F: The search requests of the same keyword from different users are dependent.
4. T / F: More than half of the power of WSCs goes into cooling.
5. T / F: WSCs contain many copies of data.

Solution: F F F F T

3 (m) Spark. Let's review the Lab 11. Exercise 1, word count, is a famous application for map reduce. We simply provide the code to students. Let's continue this problem then. Let's reimplement the word count show me your result.

```
    text_RDD = sc.textFile("filename")#the file is our simple
        and elegant course Webpage :)

    counts_RDD = text_RDD.________.__________.__________
    print(counts_RDD.take(3))
```

Fill your answer in the blank space. Explain each function below:

Solution:

```
            flatMap(lambda x: x.split(" ")).map(lambda x:
               (x, 1)).reduceByKey(lambda x,y: x+y)
```

Split word, build (key, value) pair, count word with same key.

3. **C Memory Management** Read the following code and answer the questions (10 points, 10 minutes)

```
#include <stdio.h>
#include <stdlib.h>

int global = 0;

void func(const char * x) {
      char arr[16];
      int i = 0;
      while((*x) != '\0'){
            /* toupper(): capitalize a single char. e.g. a -> A */
            arr[i] = toupper((*x));
            x = x + 1;
            i = i + 1;
      }
      arr[i] = '\0';
      printf("%s\n", arr);
}
int main(int argc, const char * argv[]) {
      char* str = "HelloWorld";
      char str2[100] = "cs110";
      func(argv[1]);
      return 0;
}
```

$\boxed{4}$ (a) In what part of memory are each of the following values stored?

$*str$: ______________

$str2[0]$: ______________

x : ______________

$global$: ______________

Solution:

- static
- stack
- stack
- static

$\boxed{1}$ (b) If the input is HelloWorld,CS110, what's the expected output?

(b) **HELLOWORLD,CS110**

$\boxed{2}$ (c) If the input is HelloWorld, what's the expected output?

(c) **HELLOWORLD**

$\boxed{3}$ (d) What kind of input could crash the program?
Hint: Think about how the code get executed in assembly code (like MIPS).

Solution: It's buffer overflow attack. An input easily can smash the program if it's long enough. e.g HelloWorld,DDDDDDDDDDDDDD.
Reason: When program get compiled into assembly, function need to push $ra to stack right after being called and read from it before exiting. A long enough buffer overflow can overwrite the $ra, and lead program into an invalid PC after execution.

4. **FPGA** (9 points, 9 minutes)

$\boxed{1}$ (a) What does FPGA stand for: (Full name of FPGA)

____

Solution: Field Programmable Gate Array

3 (b) List at least 3 examples of FPGA application/devices that used in real life:

____

____

Solution:

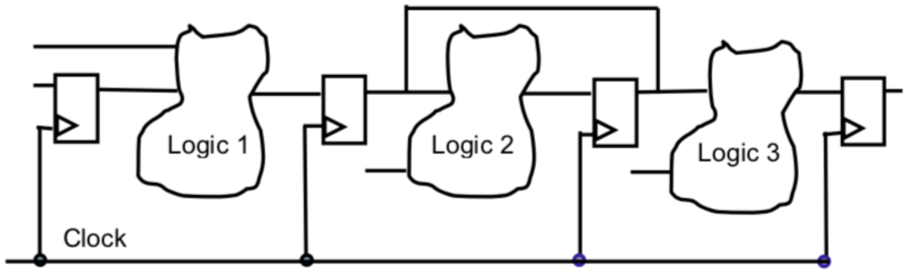
- Communication devices
  Wired and wireless routers and switches
- Automotive applications
  Braking systems, traction control, airbag release systems, and cruise-control applications
- Aerospace applications
  Flight-control systems, engine controllers, auto-pilots and passenger in-flight entertainment systems
- Defense systems
  Radar systems, fighter aircraft flight-control systems, radio systems, and missile guidance systems

2 (c) Which component(s) do not belong to a simplified FPGA Architecture:

(a) Functional Block
(b) Storage Block
(c) I/O Block
(d) Routing Network
(e) Cache

Solution: b,e

3 (d) Assume the longest delays for the combinational logic networks Logic 1, Logic 2 and Logic 3 are 4ns, 3ns and 4ns, respectively. Calculate the maximum clock frequency for the whole circuit in Figure:



Solution: $\frac{1}{4*10^{-9}} = 0.25$ GHz $= 250$ MHz

5. **Interrupt** (4 points, 4 minutes)

\boxed{1} (a) While the CPU is executing a program, an interrupt exists when it
   (a) Follows the next instruction in the program
   (b) Jumps to instruction in other registers
   (c) Breaks the normal sequence of execution of instructions
   (d) Stops executing the program

Solution: c

\boxed{3} (b) Polling and interrupts are two ways used by operating systems to check whether I/O is done. Briefly describe how each of them works, and discuss which one is better.

Solution: Polling: CPU periodically queries device to determine if they need attention
Interrupts: Each device signals to CPU that it wants to be serviced
The latter one is better since it utilizes CPU better

6. **MIPS** (16 points, 16 minutes)

\boxed{6} (a) Please **translate** the following instruction from MIPS to hex value and vice versa. Besides, specify which **type** of instructions these are. Finally, **explain** what would the two instructions do?

```
0x0200f809
ori   $a3, $t6, 1024
```

Translation of Line 1: ______________________________, instruction type: _____

Translation of Line 2: ______________________________, instruction type: _____

Explanation of Line 1:
(Hint: this instruction is a combination of two instructions that you are familiar with)

____

Explanation of Line 2:

____

Solution:

```
        # 1.5pt for translation. 0.5pt for type.
    jalr  $s0          # 0x0200f809 R type.
    # Or jalr $ra, $s0
    # 1pt. 0.5pt for type.
    ori   $a3, $t6, 1024 # 0x35c70400 I type.
```

Explain 1(1.5pt): Jump to the address in register $t0 and load return address in $ra
Explain 2(1pt): Bit wise or between register $t6 and const 1024, save the result in register $a3

OOP

You probably have wondered: can assembly language successfully handle the complexity when it comes to Objected Oriented Programming (OOP)? The answer is: yes, it can.

However, how to implement it using assembly can be an interesting problem. One possible way is to use pointers for each object (even if it is an int). In COOL (a programming language), each object has the following memory layout:

Offset	Content
-4	GC indicator. (Don't worry about this)
0	Tag. (Don't worry about this either)
4	Size of this object.
8	Pointer to a table where the address of it's functions are saved.
12	Attribute_0
16	Attribute_1
...	Attribute_n

For example, suppose we have a Object with it's pointer saved in $s0, accessing it's attribute_1 would require us to do something like:

```
    lw $t0, 16($s0)
```

$\boxed{3}$ (a) 1. In this memory layout scenario, how would you translate the following:

```
a = b + c
```

Suppose a, b and c has **Int** type and their pointers are placed in $t0, $t1, $t2. You can think of **Int** as a class with it's value put in Attribute_0. Fill in the blanks:

```
# get data from b, save it in $t1

________________________________
# get data from c, save it in $t2

________________________________
# add them and save it in $s0

________________________________
# save the result back to the first attribute of a.

________________________________
```

$\boxed{4}$ (b) 2. Suppose you have an Object with many attributes, how would you access it's n-th attribute? Suppose n is an Int typed object with it's address saved in $s0 and that Object's address is stored in $s2.

Please following the comments below and fill in codes.

```
# get n from the pointer.

________________________________
# calculate memory offset without using multiplication.

________________________________

________________________________
# add offset with original pointer.

________________________________
# access n-th attribute
# save the pointer of the attribute in $s1

________________________________
```

3 (c) 3. We can think of an array as a new class with many attributes of the same class. For example, you can think Int[5] as the following:

```
class IntArray5 {
   Int int0;
   Int int1;
   Int int2;
   Int int3;
   Int int4;
}
```

Explain how in C nasty buffer overflow problems can happen and describe in English how would you prevent overflow if you have an array defined like this.

Solution:

```
# Code 1.
  lw $t1, 12($t1)            # 0.5pt.*
  lw $t2, 12($t2)            # 0.5pt. This is the same as
     the one above.*
  addu $s0, $t1, $t2         # 1pt.
  sw $s0, 12($t0)                  # 1pt.
```

```
# Code 2.
     lw $t0, 12,($s0)              # 0.5pt, this is the
        same as the one above.*
  sll $t0, $t0, 2            # 1pt.
  addi  $t0, $t0, 12         # 1pt.
  addu  $t0, $t0, $s2        # 1pt.
  lw $s1, 0($t0)                   # 1pt.
  # Or add $s1, $0, $t0 because it's not clear what is a
     pointer.
```

Because C does not, neither can it, check array's size before access since that information is never stored, C relies on unreliable programmers to do that checking.(2pt) However, in this case, we have the size of the array stored at offset 4 and we can check the offset with size before access.(1pt)

* 3 instructions worth 0.5pt each, but since they are ultimately the same, they get at most 1pt even if they got them all correct.

7. **MIPS pipelining** (13 points, 13 minutes)

The delays of circuit elements of a datapath are given as follows:

Element	Register clk-to-q	Register Setup	MUX	ALU	Mem Read	Mem Write	RegFile Read	RegFile Setup
Parameter	$t_{clk-to-q}$	t_{setup}	t_{mux}	t_{ALU}	$t_{MEMread}$	$t_{MEMwrite}$	t_{RFread}	$t_{RFsetup}$
Delay(ps)	30	20	25	200	175	125	150	20

Answer the following questions.

$\boxed{2}$ (a) What was the clock time and frequency of a single cycle CPU ?

(a) **800ps 1.25GHz**

$\boxed{2}$ (b) What is the clock time and frequency of a pipelined CPU?

(b) **250ps 4GHz**

$\boxed{2}$ (c) What is the speed-up? Why is it less than five?

(c) ______

Solution: 3.2 This is because pipeline stages are not balanced evenly and there is overhead from pipeline registers

We are using a 5 stage MIPS pipelined datapath with separate I$ and D$ that can read and write to registers in a single cycle. Assume no other optimizations (no forwarding, no branch prediction, etc.). The default behavior is to stall when necessary. Branch checking is done during the Execute stage. On the next page is the code we want to analyze:

```
// sets *value = (*value) * 2^pow using shifting instructions
// int multMemPow2(int *value, unsigned int pow);
// $a1 = 1 in the following when pow = 1
   multMemPow2:
1     lw $v0, 0($a0) # load value
2  loop: beq $a1, $0, exit # exit condition
3     sll $v0, $v0, 1 # multiply by 2
4     addi $a1, $a1, -1 # decrement counter
5     sw $v0, 0($a0) # store result
6     j loop
7  exit: jr $ra
```

$\boxed{4}$ (d) How many clock cycles does it take to execute multMemPow2 when $pow = 1$? Fill the table below with 5-stages of the above code. (You may not need all of the columns.)

(d) **18**

Cycle #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
lw	F	D	X	M	W																	
beq																						
sll																						
addi																						
sw																						
j																						
beq																						
jr																						

$\boxed{1}$ (e) How many FEWER cycles are taken for the addition of Forwarding optimization when $pow = 1$ compare to no Forwarding? Why ?

(e) **1 , Only eliminates 35 data hazard stall.**

$\boxed{2}$ (f) How many FEWER cycles are taken for the addition of) always taken Branch Prediction optimization when $pow = 10$ with no Prediction? Why?

(f) **20 , 10 right predict and 1 wrong, so total 20 FEWER**

8. **Parallel Computing** (13 points, 13 minutes)

In this problem, we will be parallelizing ways to compute the outer product. The outer product of two vectors is defined below. In this problem, our input vectors (x and y) will both be of length n.

$$uv^T = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \\ u_3v_1 & u_3v_2 & u_3v_3 \\ u_4v_1 & u_4v_2 & u_4v_3 \end{bmatrix}$$

$\boxed{5}$ (a) We want to parallelize the following code with OpenMP, but it is currently done incorrectly. #pragma omp parallel specifies a code block that is restricted to access by only one thread at a time. When this omp critical pragma is used, a thread waits at the beginning of a critical section until no other thread in the team is executing a critical section having the same name.

```
void outer_product(float *dst, float *x, float *y, size_t n)
    {
    #pragma omp parallel
    for (size_t i = 0; i < n; i += 1) {
        for (size_t j = 0; j < n; j += 1) {
            #pragma omp critical
            dst[i * n + j] = x[i] * y[j];
        }
    }
}
```

You may only add or remove #pragma omp statements. What changes do we need to make the code run both quickly and correctly?

8

(b) Now use SSE instrinsics to optimize `outer_product()`. Assume n is a multiple of 4. You may find the following useful:

1. `_mm_loadu_ps(__m128 *src)` loads the next four floats of `src` into the vector
2. `_mm_load1_ps(float *f)` loads `float f` into each slot of the vector
3. `_mm_storeu_ps(__m128 *dst, __m128 val)` stores `val` at memory `dst`
4. `_mm_mul_ps(__m128 a, __m128 b)` multiplies two vectors

```
void outer_product(float *dst, float *x, float *y, size_t n)
   {
   for (size_t i = 0; i < n; i += 1) {
      for (size_t j = 0; j < n; j += 4) {
         __m128 a = _mm_load1_ps(&x[i]);


         ________________________________________________


         ________________________________________________


         ________________________________________________
      }
   }
}
```

Solution:

a)

Change the `#pragma omp parallel` to `#pragma omp parallel for`

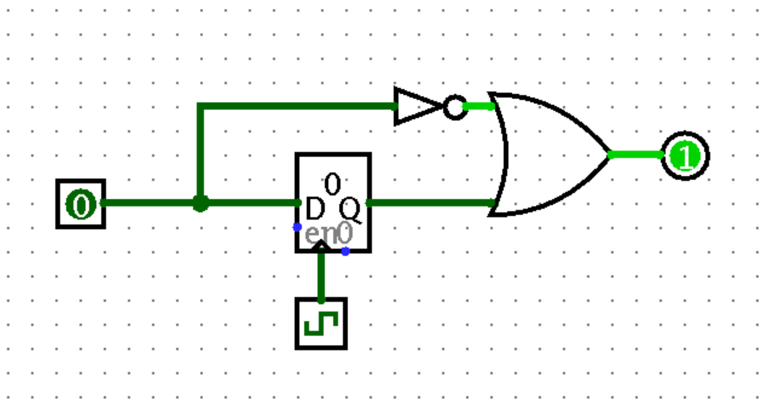
Remove the `#pragma omp critical`

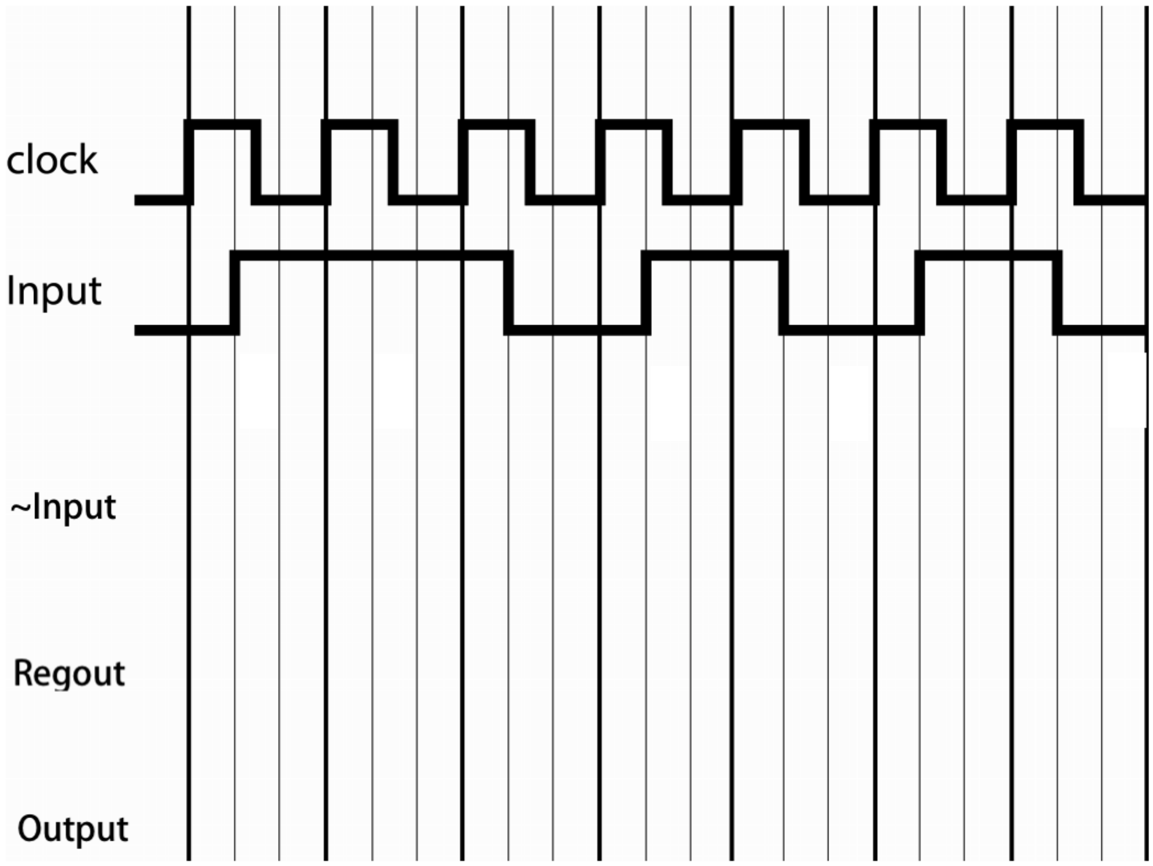
b)

```
      __m128 b = _mm_loadu_ps(&y[j]);
      __m128 products = _mm_mul_ps(a, b);
      __mm_storeu_ps(&dst[i * n + j], products);
```

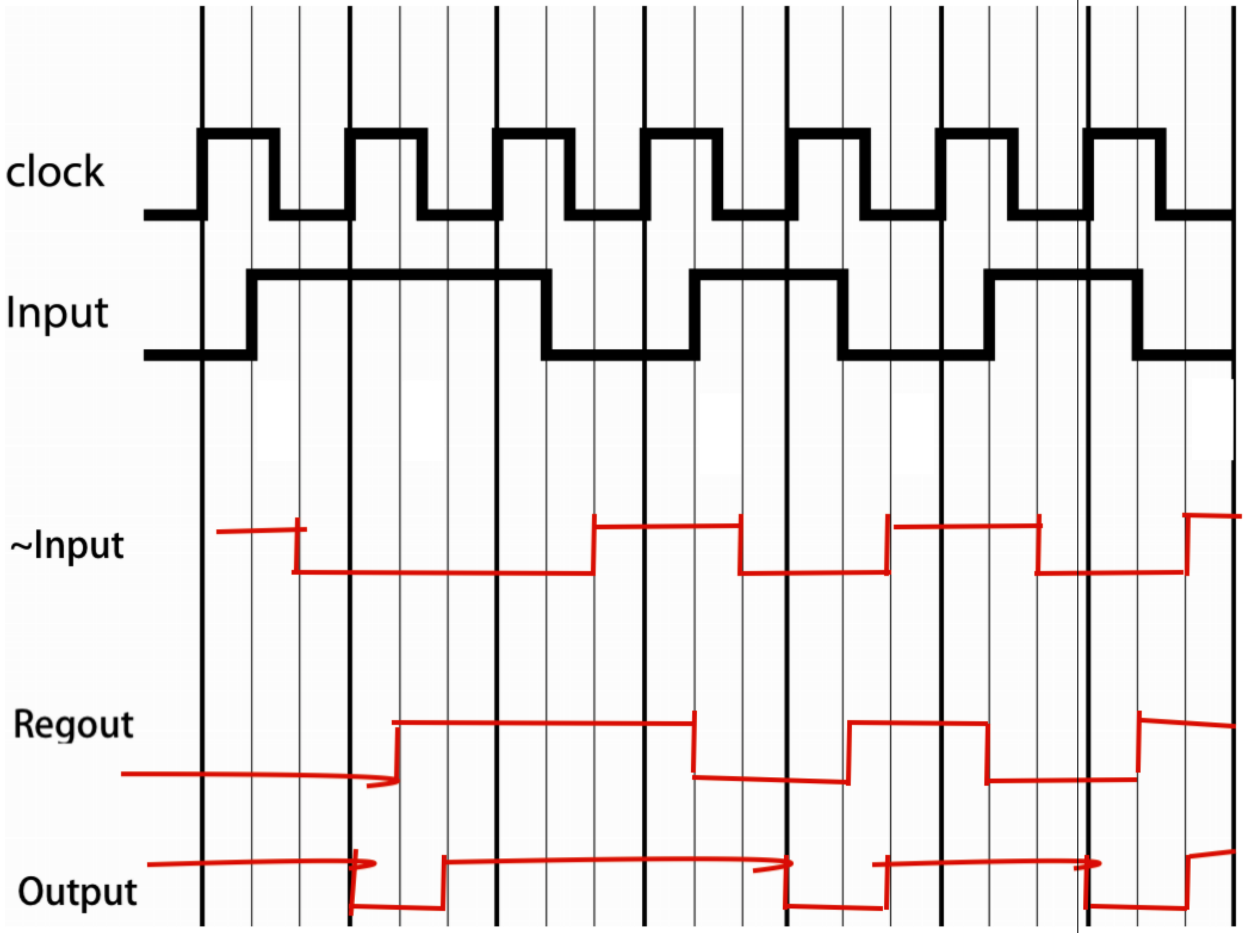
9. **SDS, FSM** (14 points, 14 minutes)

$\boxed{6}$ (a) Assume that the clock period is 15ns, the input comes 5ns after each positive clock edge, the register has no hold and setup times but has a clock-to-Q delay of 5ns, and logic gates have a 5ns delay. Please complete **wave form**, each column is 5ns, if the wave form of some timestamp is unknown, you can freely assign it.

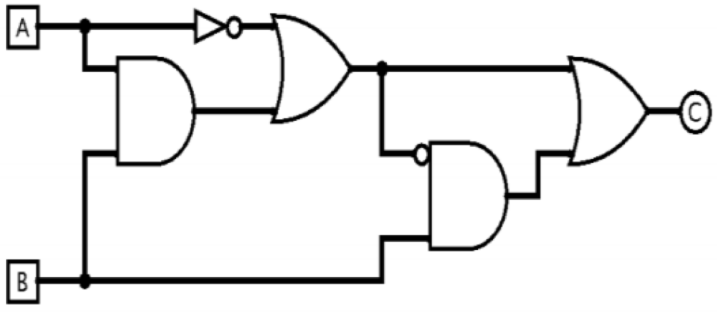




Solution:



$\boxed{3}$ (b) Give the simplest Boolean expression for the following circuit in terms of A and B.



(b) ________ **$\sim$ A + B** ________

$\boxed{5}$ (c) **FSM**

Jeff and Onion are playing basketball one on one on the ShanghaiTech basketball court.

- They both have 0 points at the beginning.
- Each round they both shoot once. We check the score after each round, so the order is not important. If they make a shot (put the ball in the basket), they get 1 point, otherwise they got 0 points.
- The first player who gets 2 points (after the round is completed) wins, if they get 2:2 after one round, they start a new game from 0:0.
- If after one round, Jeff wins (Jeff:Onion is 2:1 or 2:0), output 01_2, and the game restarts from 0:0 and Onion will buy a bottle of water for Jeff. If Onion wins, output 10_2, and then game restarts from 0:0. On all the other cases output 00_2
- Design and draw this FSM diagram using the format we require in previews exam and hw.

Specify what your states and input represent.

State:

(c) ________________________________

Input:

(c) ________________________________

Draw your FSM:

State: Jeff : Onion

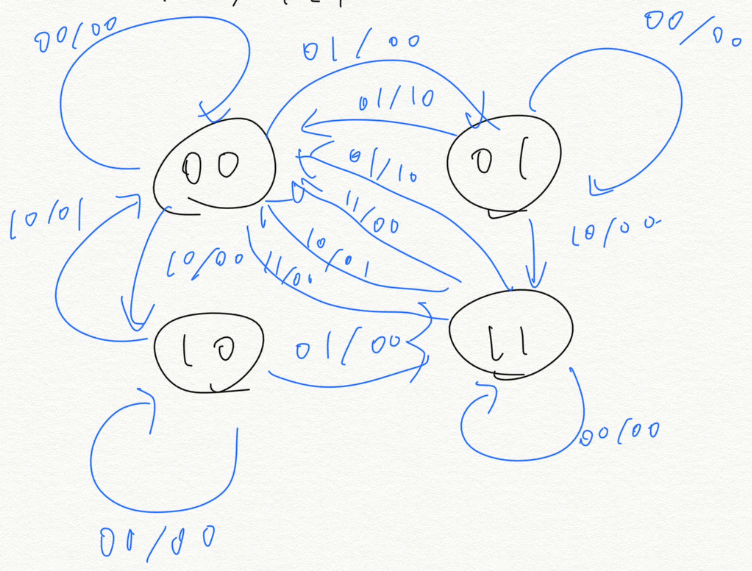
00 → 0:0 01 → 0:1 10 → 1:0

11 → 1:1

Input : Jeff : Onion each round

00 → 0:0 01 → 0:1 10 → 1:0

1:1 → 1:1



Solution:

No question here - do not fill.