Computer Architecture I Mid-Term

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

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

Question	Points	Score
1	1	
2	14	
3	10	
4	28	
5	11	
6	11	
7	20	
8	5	
Total:	100	

- 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.
- The total exam 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 one A4 page (front and back) of handwritten notes in addition to the provided green card.
- 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 20 times).

2. **MISC [14 points]**

[2] (a) (_____) **[Multi-choice]** Which rounding mode(s) can lead to the following behavior: 2.5 $\rightarrow$ 2, -2.5 $\rightarrow$ -2?

A. Round-to-nearest-ties-to-even
B. Round-to-$+\infty$
C. Round-to-$-\infty$
D. Round-towards-0
E. Round-to-nearest-ties-to-max-magnitude
F. None of the above

Solution: A,D

[2] (b) (_____) **[Multi-choice]** Select all the ISAs.

A. x86/amd64
B. ARMv8
C. Ubuntu
D. CentOS
E. None of the above

Solution: A,B

[2] (c) (_____) **[Multi-choice]** Given the code:

```
1  int *p = (int *)malloc(10 * sizeof(int));
2  p++;
3  free(p);
```

Which of the following statement(s) will happen?

A. Normally free the memory from the second element onward, with no error in the program.
B. Only free the first element, and the program continues to run.
C. Cause a runtime error or undefined behavior.
D. The compiler will report an error and stop the translation.
E. None of the above.

Solution: C

[2] (d) (_____) **[Multi-choice]** Choose all the input combination(s) that make(s) the circuit output, **out**, evaluate to '1' (high voltage).

.

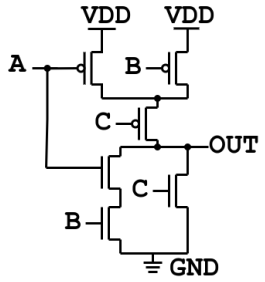
A. `A = 1, B = 0, C = 0`

B. `A = 0, B = 1, C = 0`

C. `A = 0, B = 0, C = 0`

D. `A = 0, B = 1, C = 1`

E. None of the above



Solution: ABC

1 (e) (_____) **[True or False]** Both `jal` and `jalr` are J-type instructions.

Solution: F

1 (f) (_____) **[True or False]** Interpreting the 8-bit binary `0b11111111` as unsigned, two's complement, and sign-magnitude integers yields at least two identical values.

Solution: F

1 (g) (_____) **[True or False]** RV32I has no instruction that can store a 8-byte value to memory in a single instruction.

Solution: T

1 (h) (_____) **[True or False]** The loader is responsible for loading the executable from the operating system to the main memory.

Solution: F

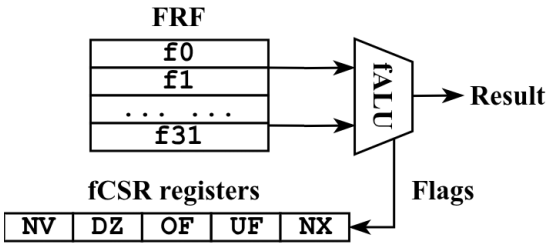
1 (i) (_____) **[True or False]** A computer can be modeled as a finite-state machine.

Solution: T

1 (j) (_____) **[True or False]** In a pipelined CPU, the execution time of each instruction is reduced, so the performance is improved.

Solution: F

3. **Floating-Point Extension [10 points]**
RISC-V is well-known for its extensibility. For example, F-extension can be added so that floating-point operations are realized by using F-instructions. To support the execution of the F-instructions, the hardware must also be extended.
As shown in the diagram below, 32 registers are organized into a "floating-point register file" (FRF) to hold 32 single precision floating-point numbers (32-bit), and the "fALU" is responsible for executing floating-point arithmetic operations.
However, floating-point arithmetic may produce exceptional results, such as overflow, underflow, etc. Therefore, "Flags" are required to mark these exceptions, and these flags are also generated by the "fALU" module. The 5 "Flags" are then stored in the 5 corresponding bits of "fCSR registers" (floating Control and Status Registers).



The 5 bits in the "fCSR" indicates:

1. **`NV`**: invalid operation, set to 1 when the result is **`NaN`**;
2. **`DZ`**: divide-by-zero, set to 1 when a dividsion by zero is attempted;
3. **`OF`**: overflow, set to 1 when the result overflows the single-precision format;
4. **`UF`**: underflow, set to 1 when the result underflows the single-precision format;
5. **`NX`**: inexact, set to 1 when the result is rounded, or when any of the above exceptions is raised;

otherwise, the flags are **RESET TO 0**.

5 (a) Suppose F-instruction **`fadd.s frd,frs1,frs2`** adds the two floating-point numbers held in the FRF **`frs1`** and **`frs2`**; the result is then stored in **`frd`** register. Assume **`f1=0x3FC00000`** and **`f2=0xFFC00000`**, and the F-instruction **`fadd.s f3,f1,f2`** is executed. Please fill in the values of the flags in the table below.

NV	DZ	OF	UF	NX

Solution:

NV	DZ	OF	UF	NX
1	0	0	0	1

5 (b) Suppose F-instruction **`fmul.s frd,frs1,frs2`** multiplies the two floating-point numbers held in the FRF **`frs1`** and **`frs2`**; the result is then stored in **`frd`** register. Assume **`f1=0x3FC00001`**, and the F-instruction **`fmul.s f2,f1,f1`** is executed. Please fill in the values of the flags in the table below.

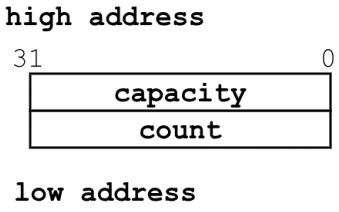
NV	DZ	OF	UF	NX

Solution:

NV	DZ	OF	UF	NX
0	0	0	0	1

4. Calling Convention & RISC-V & Pipeline [28 points]

Assume a **RV32I** machine following the **standard RISC-V calling convention**. Unless otherwise stated, **`int`** and pointers are both **32 bits**, the stack grows toward **lower addresses**, **`struct Batch`** has **no extra padding** between fields, and its layout in memory is shown on the right.



The following code models a simple telemetry (data collection) batching system. A **`Batch`** stores the number of events currently buffered in the active batch as **`count`**. If adding **`x`** new events would cause the batch to exceed its **`capacity`**, the active batch is first submitted by calling **`batch_close`**. After submission, the batch is emptied, and the new events are then added to the now-empty batch.

Assume that:

- **`b`** is always a valid pointer;
- **`capacity > 0`**;
- **`0 <= count <= capacity`**;
- **`0 < x <= capacity`**.

The helper function **`void submit_batch(int n)`** is provided elsewhere; it submits the current batch containing **`n`** buffered events.

```
1  struct Batch {
2     int count;
3     int capacity;
4  };
5
6  void batch_close(struct Batch *b) {
7     submit_batch(b->count);
8     b->count = 0;
9  }
10
11 int batch_add(struct Batch *b, int x) {
12    if (x > b->capacity - b->count) {
13       batch_close(b);
14    }
15    b->count = b->count + x;
16    return 0;
17 }
```

5 (a) Fill in the blanks about the standard RISC-V calling convention.

1. In the function call **`batch_add(b,3)`**, the argument **`b`** is passed in register _____, the argument **`3`** is passed in register _____, the return value is placed in register _____, and the return address register is _____.

2. Suppose a function makes a nested function call and needs the old value of each register below to survive that call. For each register, write **`caller`** if the caller is responsible for saving it, or **`callee`** if the callee must preserve it:

`s0` : _____ **`s1`** : _____ **`a0`** : _____

`a1` : _____ **`t0`** : _____ **`ra`** : _____

Solution:

1. **`a0`, `a1`, `a0`, `ra`**
2. **`callee`, `callee`, `caller`, `caller`, `caller`, `caller`**

3 (b) Complete the following assembly implementation of **`batch_add`**. Assume that **`batch_close`** assembly is provided elsewhere and correctly follows the standard RISC-V calling convention. Each blank has a **unique best answer** under the given code and the standard calling convention. Assume the stack pointer must remain **16-byte aligned** at function-call boundaries. Do not allocate extra unused stack space.

```
1  batch_add:
2     addi sp, sp, -______
3     sw  ra, 12(sp)
4     sw  s0, 8(sp)
5     sw  s1, 4(sp)
6
7     mv  s0, a0          # keep b across the call
8     mv  s1, a1          # keep x across the call
9
10    lw  t0, ______(s0)   # b->capacity
11    lw  t1, ______(s0)   # b->count
12    sub t0, t0, t1      # remaining = capacity - count
13    bge t0, s1, .Ladd
14    mv  a0, s0
15    call batch_close    # call function batch_close
16
17 .Ladd:
18    lw  t0, ______(s0)   # b->count
19    add t0, t0, s1
20    sw  t0, ______(s0)
21    li  a0, ______
22
23    lw  s1, 4(sp)
24    lw  s0, 8(sp)
25    lw  ra, 12(sp)
26    addi sp, sp, ______
27    _______             # return
```

Solution:

```
1  batch_add:
2     addi sp, sp, -16
3     sw  ra, 12(sp)
4     sw  s0, 8(sp)
5     sw  s1, 4(sp)
6
7     mv  s0, a0          # keep b across the call
8     mv  s1, a1          # keep x across the call
9
10    lw  t0, 4(s0)       # b->capacity
11    lw  t1, 0(s0)       # b->count
12    sub t0, t0, t1      # remaining = capacity - count
13    bge t0, s1, .Ladd
14    mv  a0, s0
15    call ra, batch_close
16
17 .Ladd:
18    lw  t0, 0(s0)       # b->count
19    add t0, t0, s1
20    sw  t0, 0(s0)
21    li  a0, 0
22
23    lw  s1, 4(sp)
24    lw  s0, 8(sp)
25    lw  ra, 12(sp)
26    addi sp, sp, 16
27    ret
```

6 (c) For this part, use the completed assembly from the previous question.

During the time that **`batch_add(b,3)`** has been called, a snapshot is taken just **BEFORE the instruction in line 20** begins execution in **`batch_add`**, and **`pc`** should be ready to fetch the instruction in line 20. For this problem, interpret the snapshot using **ISA-level sequential semantics**, i.e., all earlier instructions have already completed. Each instruction occupies **`4 bytes`**.

Additional assumptions:

- **`batch_add`** starts at address **`0x00400100`**
- before entering **`batch_add`**, the registers are:
  - **`sp = 0x8000`**
  - **`ra = 0x00400080`**
  - **`a0 = 0x1000`**
  - **`a1 = 3`**
  - **`s0 = 0x11111111`**

  - **`s1 = 0x22222222`**
- the struct **`b`** is stored at address **`0x1000`**
- **`b->count = 2`**; **`b->capacity = 8`**

Fill in the register states at the snapshot point. **Write all the values in hexadecimal with a `0x` prefix.**

Item	Value
`pc`	
`sp`	
`ra`	
`t0`	
`s0`	
`s1`	

Solution:

Item	Value
`pc`	`0x00400138*`
`sp`	`0x00007ff0`
`ra`	`0x00400080`
`t0`	`0x00000005`
`s0`	`0x00001000`
`s1`	`0x00000003`

*`pc` would be `0x0040013C` should `call` instruction be assembled into 2 instructions. This should be a more reasonable solution.

10 (d) Identify all the **`read-after-write (RAW)`** data hazards from the code in question (b). Write down the instruction pairs by their line numbers and the registers that causes the data hazard.
[Fill in the blanks] Assume a 5-stage classic pipelined CPU, and the register file supports write in the first half of the clock cycle and read of the updated value in the second half.

A. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

B. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

C. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

D. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

E. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

F. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

G. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

H. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

I. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

J. There is a RAW data hazard on register _____ between Instruction in line _____ and Instruction in line _____.

(There might be more space than you need, just leave them blank.)

Solution: A. There is a RAW data hazard on register `sp` between Instruction in line `2` and Instruction in line `3` .
B. There is a RAW data hazard on register `sp` between Instruction in line `2` and Instruction in line `4` .
C. There is a RAW data hazard on register `s0` between Instruction in line `7` and Instruction in line `10` .
D. There is a RAW data hazard on register `t0` between Instruction in line `10` and Instruction in line `12` .
E. There is a RAW data hazard on register `t1` between Instruction in line `11` and Instruction in line `12` .
F. There is a RAW data hazard on register `t0` between Instruction in line `12` and Instruction in line `13` .
G. There is a RAW data hazard on register `t0` between Instruction in line `18` and

Instruction in line `19` .
H. There is a RAW data hazard on register `t0` between Instruction in line `18` and Instruction in line `20` .
I. There is a RAW data hazard on register `t0` between Instruction in line `19` and Instruction in line `20` .
J. There is a RAW data hazard on register `[You should left this line blank]` between Instruction in line `[You should left this line blank]` and Instruction in line `[You should left this line blank]`.

2 (e) **[Multiple choices]** Among the data hazards above, _____ cannot be solved by forwarding itself only. Select your answer for this question using options A, B, C, ... from the previous question (d).

Solution: You should choose
E. There is a RAW data hazard on register `t1` between Instruction in line `11` and Instruction in line `12` .
G. There is a RAW data hazard on register `t0` between Instruction in line `18` and Instruction in line `19` .
whatever the letter is.

2 (f) Translate the `bge` instruction in **line 13** to machine code in hexidecimal:

_____

Solution: `0x0092D663` or strictly `0x0092D863` (`call` should be assembled into 2 instructions)

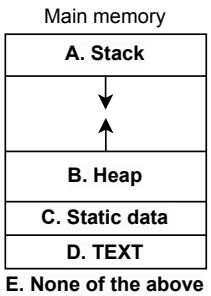
5. CALL [11 points]

Consider the following 32-bit C program. **`sqrt`**, **`printf`**, **`calloc`** and **`free`** come from the C standard libraries, and are **dynamically linked**. The functions **`add`** and **`max`** are **statically linked**. The compilation of the program **does not record debugging information**. Please answer the following questions based on the code.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAG(x, y) (sqrt((x)*(x) + (y)*(y)))

int add(int x,int y)
{return x+y;}

int max(int x,int y)
{return x>y?x:y;}

int main(void) {
   double x=3.0,y=4.0;
   double mag = MAG(x,y);
   printf("The magnitude is %f\n",mag);
   int *p = (int *)calloc(2,sizeof(int));
   int c = add(*p,*(p+1));
   printf("The sum is %d\n",c);
   free(p);
   return 0;
}
```

6 (a) **[Single choice]** Where are they located when the excutable file of function **`main`** is running?

1. Pre-processing macro **`MAG`** (_____)
2. The executable file itself. (_____)
3. The instructions of function **`add`** (_____)
4. The string literal **`"The sum is %d\n"`** (_____)
5. The pointer **`p`**. (_____)
6. The integer variable **`c`**. (_____)



Solution: E, E, D, C, A, A

4 (b) **[Fill in the blanks]** At which stage can the final machine code be determined for the following instructions:

i. The function call to `add(*p,*(p+1))`
ii. The branch label of `x>y?x:y;`
iii. The standard math library function call to `sqrt((x)*(x)+ (y)*(y))`
iv. The function call return `return 0;`

A. After preprocessing / compilation
B. After assembly
C. After static linking
D. After dynamic linking / loading

i	ii	iii	iv

Solution: C, B, D, B

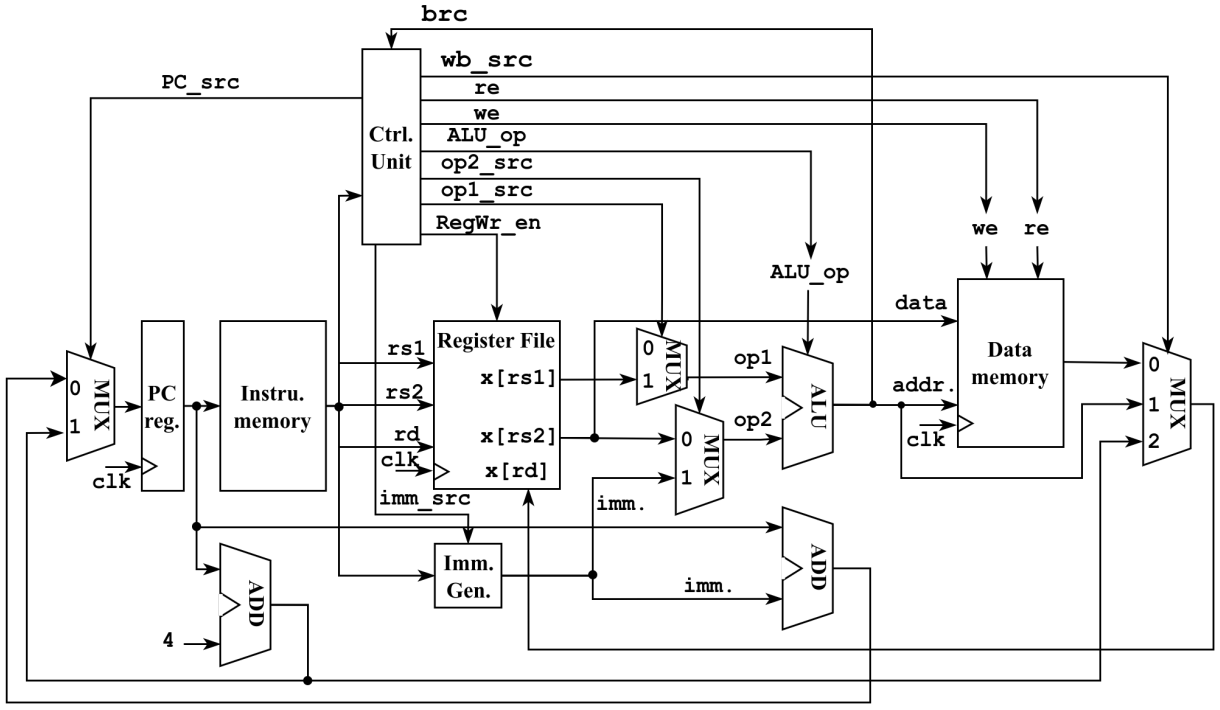
1 (c) i. (_____) **[Single-Choice]** Which of the following most likely appears in `main.o` before the final linking?

A. The final absolute address of `calloc`
B. Relocation information for an unresolved external reference to `printf`
C. The full machine code of the C standard library
D. The runtime stack frame of `main`

Solution: B

6. **Datapath [11 points]**

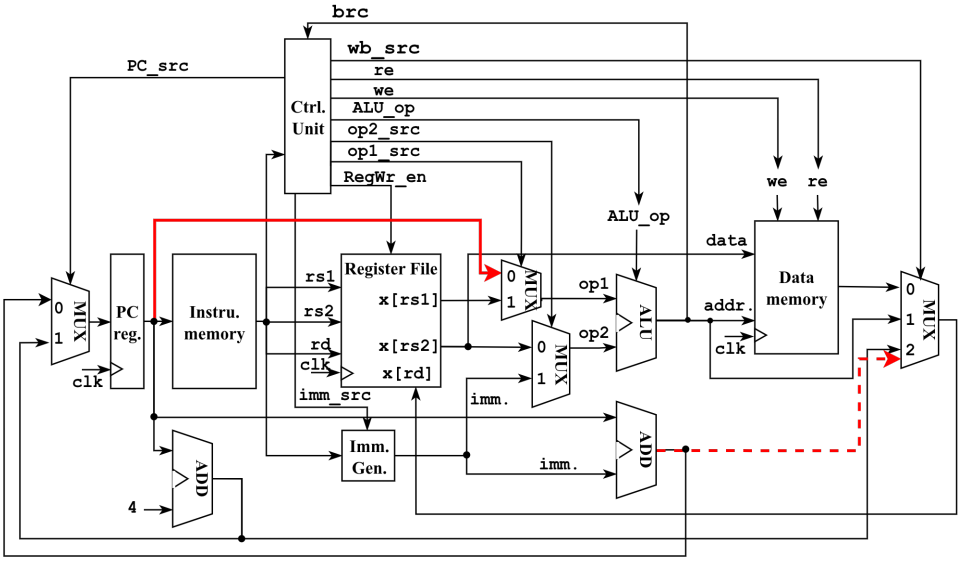
In this part, we provide a microarchitecture that supports a subset of RV32I. Analyze the diagram and answering the following questions.



Note that in this figure, "**we**" stands for "write enable", "**re**" stands for "read enable", and "**RegWr_en**" stands for "register file write enable". **brc** evaluates to 1 when the branch condition is true, and vice versa.

7 (a) **Add one wire** in the provided datapath diagram to enable execution of the instruction **auipc**. Complete the table below with all the control signal values for **auipc**. Use **X** to indicate "don't-care" or unknown signals.

Solution: The solid red line is the optimal solution, while the dotted line is also acceptable. The control signal solution below corresponds to the optimal solution.



PC_src	brc	wb_src	re	we
ALU_op	**op2_src**	**op1_src**	**RegWr_en**	**imm_src**

Solution:

PC_src	brc	wb_src	re	we
1	X	1	0/X	0
ALU_op	**op2_src**	**op1_src**	**RegWr_en**	**imm_src**
add	1	0	1	U_type

For **ALU_op**, you can choose among **add**, **sub**, **and**, **or**, **xor**, **shift** (for any instructions that requires a shift, e.g., **sll**, **slli**, etc.), or **compare** (for any instructions that requires a compare, e.g., **slt**, **beq**, etc.)

For **imm_src**, you can choose among I-type, S-type, SB-type, U-tpye or UJ-type.

4 (b) Write down all the instruction(s) in RV32I for which the **wb_src** signal must be **2**.

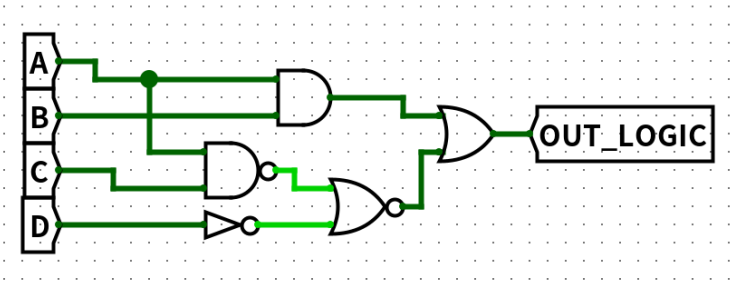
Solution: **jal** **jalr**

7. **Digital Circuits [20 points]**

[1] (a) (_____) **[Fill in the blank]** What is the name of the following logic gate?

Solution: NAND

[10] (b) Please write down the truth table of the circuit below, and write down the simplified boolean expression that uses the least total number of **2-input** and/or gates and not gates.



A	B	C	D	OUT_LOGIC

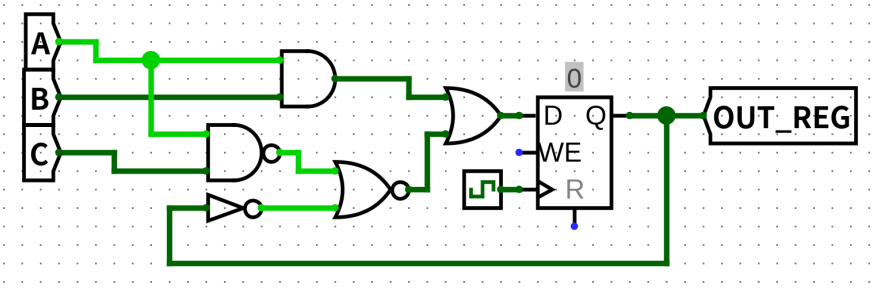
OUT_LOGIC = _____

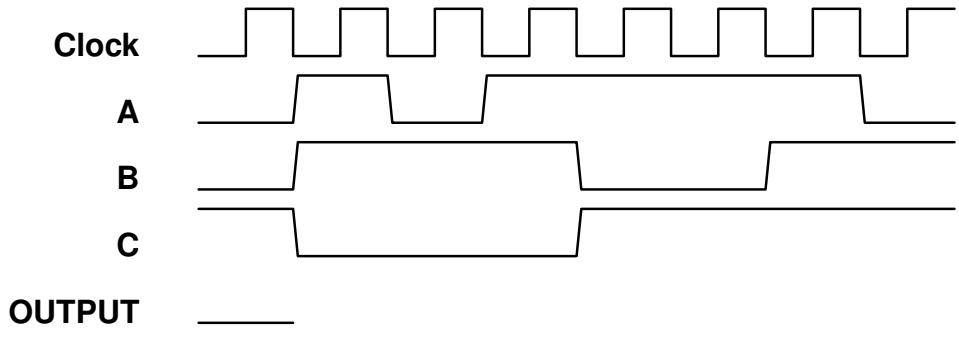
Solution:

A	B	C	D	Y
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
1	1	0	0	1
1	0	1	0	0
1	0	0	1	0
0	1	1	0	0
0	1	0	1	0
0	0	1	1	0
1	1	1	0	1
1	1	0	1	1
1	0	1	1	1
0	1	1	1	0
1	1	1	1	1

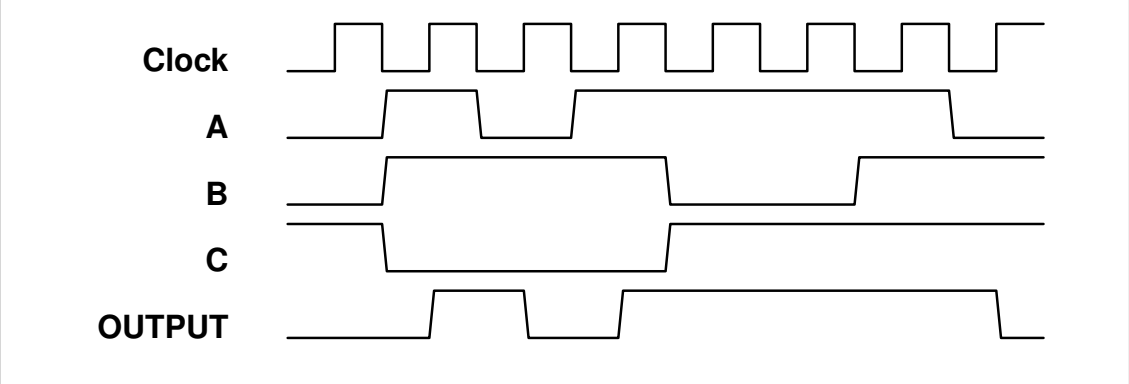
OUTPUT = A(B+CD)

4 (c) Now a register is added and the circuit is modified. Please complete the timing diagram according to the clock signal and input provided. Ignore the non-ideal effects.





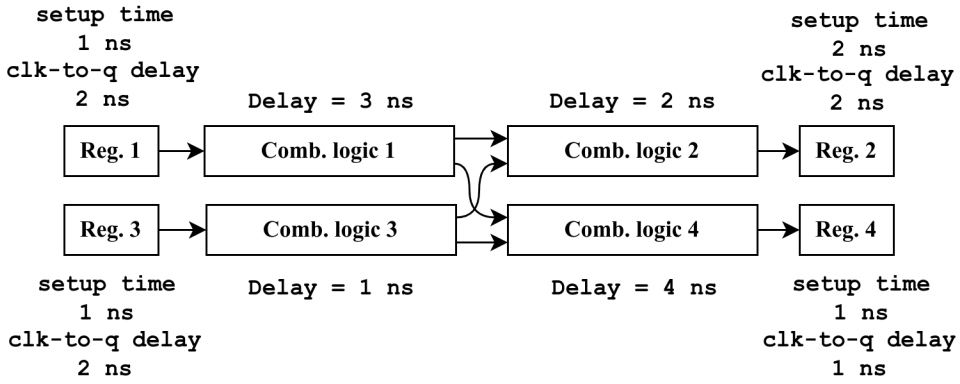
Solution:



1 (d) (_____) **[Fill in the blank]** When the previous digital circuit in question (c) is modeled as a finite state machine (FSM), what is the type of this FSM?

Solution: Moore machine

2 (e) **Timing analysis.** The following circuit consists of 4 combinational logic blocks and 4 registers. The delays of the combinational blocks and timing parameters of the registers are given in the circuit. Compute the maximum frequency of the circuit.



Solution: Longest path: **Reg.1 clk-to-q delay + Comb. logic 1 delay + Comb. logic 4 delay + Reg.4 setup time**
= 2 + 3 + 4 + 1 = 10 ns
Max. frequency **= 1/10 ns = 100 MHz or 0.1 GHz**

2 (f) **Pipeline** can improve the frequency of a digital circuit. Assume pipeline registers are allow only between the combinational blocks in the circuit above in question (e). Compute the maximum frequency of the pipelined circuit. Assume the setup time of all the pipeline registers are **1 ns**, and the **clk-to-q** delay of them are **3 ns**.

Solution: Longest path after pipelining: **Reg. 1 clk-to-q delay + Comb. logic 1 delay + Pipeline Reg.4 setup time**

= 2 + 3 + 3 = 8 ns
Max. frequency **= 1/8 ns = 125 MHz or 0.125 GHz**

8. **Performance analaysis [5 points]** The CPU time for a computer follows the golden formula

$$\text{CPU time} = \text{Instruction count} \times \text{Average CPI} \times \text{Clock period} \quad (1)$$

Two processors, P1 and P2, run the same program, which consists of 1×10^9 instructions.
Clock frequency of P1 is $f_1 = 2.5$ GHz, and average CPI is 1.0. P1 is not pipelined and has no hazards.
P2 is pipelined, and the clock frequency is $f_2 = 4$ GHz. The ideal case (considering no hazards) CPI is 1.0. But in practice, load-use data hazards present in the program, and we assume they are the only type of hazard that exists. For such hazards, processor P2 inserts 1 stall cycle to resolve them. Pipeline fill and drain cycles at the beginning and end of the program are neglected.
Statistical data shows that among all instructions executed in this program: 40% of are load instructions. 50% of these load instructions have their result used by the immediately following instruction, i.e., cause a load-use hazard.

5 (a) Compute the CPU time of this program on the two processors.

Solution: P1: CPU time = $0.4\text{ns} \times 1 \times 10^9 = 0.4\text{s}$
P2: CPU time = $0.25\text{ns} \times 1 \times 10^9 + 0.25\text{ns} \times 1 \times 10^9 \times 50\% \times 40\% = 0.3\text{s}$