Computer Architecture I Mid-term Exam 1

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

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

Question	Points	Score
1	1	
2	4	
3	14	
4	9	
5	19	
6	6	
7	10	
8	20	
9	17	
Total:	100	

- This test contains 16 numbered pages, including the cover page, printed on both sides of the sheet.
- We probably will use blackboard 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 32-bit 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 16 times).

Also, unless told otherwise, always assume a 32-bit machine, `int` type has 4 bytes or 32 bits, DFFs are triggered at the rising edge of the clock signal, and we use only `and, or and not` gates for combinational circuit design throughout the exam.

4 2. **Introduction [4 points]**

(1) Moore's Law: True (T) or False (F). According to Moore's Law, the number of transistors on a single chip will continue to double every two years forever. [2 points] ( )

(2) Amdahl's Law: Assume we have a task, 20% of which cannot be parallelized, what is the maximum acceleration factor can be achieved with unlimited parallelized computing resources? [2 points]

Solution: (1) F. Moore's Law is not a law, it is just a prediction. (2) 5.

$$F = \frac{1}{20\% + \frac{80\%}{P}} \tag{1}$$

When P goes to infinity, F is 5.

14 3. **Number representation [14 points]**

(1) Fill in the blanks below to show how the 32 bits can be interpreted in different ways. If a particular field has no solution, answer "N/A". [8 points]
bit pattern B = 1111 1111 0010 0100 0110 0000 0000 0000

(a) 4 sign-magnitude bytes (in decimal): _____

(b) 4 one's complement bytes (in decimal): _____

(c) 4 two's complement bytes (in decimal): _____

(d) Compute `B^(B<<5)` (in hexadecimal, ^ stands for XOR): _____

(2) Consider a base-9 number system (with nine valid digits of 0, 1, 2, 3, 4, 5, 6, 7, 8). [2 points]

(a) 188_{nine} in base 10: _____

(b) $2A5_{hex}$ in base 9: _____

(3) An **8-bit** floating-point number (FP8) has 1 sign bit, 4 exponent bits and 3 mantissa bits (with an un-shown hidden 1 for normal cases). The bias for the exponent is 7. Meanwhile, consider that we receive data with some unknown bits, and "x" is used to refer to the unknown bits. For example, if the received data are "0b0xx1", they could be "0b0001", "0b0011", "0b0101" or "0b0111". [4 points]

(a) We receive an FP8 number "0b0x1x0x1x". What is the **largest** number the sender could have sent? (in **hexadecimal**)

(b) Consider an FP8 number that is neither an NaN nor infinity nor 0, what is the smallest possible positive number the sender could have sent **as a power of 2**? Assume FP8 uses the same rule to represent a denormalized number as a single-precision floating-point number.

Solution:

(1) (a) -127, 36, 96, 0
(b) (-)0, 36, 96, 0
(c) -1, 36, 96, 0
(d) 0x1BA86000

(2) (a) 161_{ten}
(b) 832_{nine}

(3) (a) 0x77
(b) 2^{-9} ($2^{-6} \times 2^{-3}$)

4. **C basics [9 points]**

3 (a) What is the output of the following program segment? [3 points]

```
char c = 'a';
putchar(c);    /*Equivalent to print char c*/
putchar(F(c));
putchar(c);
```

Assume that the function F has been defined as follows:

```
char F(char c)
{
   c = 'f';
   return c;
}
```

Solution: afa

2 (b) Fill in the code to properly allocate memory (on the heap) for an `n` × `m` matrix `mat` of integers initialized to zeros. [2 points]

```
1 mat = (int **) calloc(n, sizeof(int *));
2 for (int i = 0; i < n; i++) {
3    mat[i] = _____
4 }
```

line 3: _____

Solution: (int *) calloc(m, sizeof(int));

4 (c) The following program performs an n-bit cyclic left-shift or bit rotation of an integer using a "while" loop. Specifically, the highest (leftmost) n bits are moved to the lowest bits, and the rest bits are left-shifted by n bits. For example, 1100 becomes 1001 after a 1-bit cyclic left-shift, and 0011 after a 2-bit cyclic left-shift. Please fill in C code to realize the function according to the comments. (**Hint**: In C language, $>>$ operator performs arithmetic right-shift or logical right-shift according to the type of the variables.)

```
1  int rotate_integer(int num, int n) {
2     int new_num = num;
3     int high_bit = 0;
4     while (___(1)__)
5     {
6        high_bit = __(2)__;/*Obtain the highest bit*/
7        new_num = __(3)__;/* Left-shift the number by 1*/
8        new_num = __(4)__;/* Put the highest bit to the lowest
            bit*/
9        num = new_num;
10       n--;
11    }
12    return new_num;
13 }
```

(1)_____
(2)_____
(3)_____
(4)_____

Solution:

```
1 n>0
2 high_bit = (unsigned int)(num&0x80000000)>>31 ;
3 new_num = new_num << 1;
4 new_num = new_num + high_bit;
```

Other reasonable solutions are also acceptable.

5. **RISC-V assembly [19 points]**

5 (a) Perform an R-type **signed** addition (add t2, t1, t0) and detect overflow. If an overflow occurs, t4 register is set to 1; otherwise, t4 is set to 0. Please use RV32I instructions (as less instructions as possible) to complete the below assembly code. (**Hint**: the sum should be less than one of the operands if and only if the other operand is negative. Feel free to use `t0~t6`. Also, please comment your code properly. Leave it blank if you do not use all the space below.)

```
add   t2, t1, t0

_____________________

_____________________

_____________________

_____________________

_____________________
addi  t3, t3, 1
beq   t4, t3, OVERFLOW
... ...
OVERFLOW: #some code to deal with overflow
```

Solution:

```
slti t5, t0, 0 #set t5 if t0<0
slt t4, t2, t1 #set t4 if t2<t1
xor t4, t4, t5 #if (t0>=0 and t2<t1) or (t0<0 and t2>=t1),
   overflow occurs and t4 is set to 1
addi t3, x0, 0 #initialize t3.
```

t1 and t2 are exchangable. t5 can be other tx.

8 (b) In this question, you will calculate the nth term of a Fibonacci sequence and then the factorial of it. The $(i+2)$th term of Fibonacci sequence is given by $f_{i+2} = f_{i+1} + f_i$. The initial values are given as $f_{-1} = 0$ and $f_0 = 1$, and the nth term refers to f_n. For example, if $n = 3$, you should obtain the result 6 (3!). [8 points] (**Hint**: The Fibonacci function first calculates the nth item and then calls "int factorial(int nth item)", which is a

recursive function. To simplify, we consider that multiplication can be implemented with just one "mul" instruction and you do not have to consider the higher/lower bits.)

```
1   main:
2      li    t0, 0
3      li    t1, 0        #the -1st term, f[-1]
4      li    t2, 1        #the 0th item, f[0]
5      li    a0, n        #input parameter n
6      li    a1, 0
7      beq   a0, t0, End  #exit when n is 0
8   Fibonacci_loop:
9      add   t3, t1, t2
10     add   t1, t2, x0
11     ___________________
12     add   a1, t3, x0
13     addi  t0, t0, 1
14     ___________________ #branch Fibonacci_loop when unfinished
15     jal   ra, Factorial
16  End:                  #print and exit, etc. stuff
17     ecall ...
18  Factorial:
19    addi  sp, sp, -8
20    sw    ra, 4(sp)
21    sw    a1, 0(sp)
22    addi  t4, a1, -1
23    bge   t4, x0, Factorial_loop
24    addi  a1, x0, 1     #the last recursive call
25    addi  sp, sp, 8
26    jalr  x0, ra, 0
27  Factorial_loop:
28    addi  a1, a1, -1
29    ___________________ #recursively call Factorial
30    addi  t5, a1, 0
31    lw    a1, 0(sp)
32    lw    ra, 4(sp)
33    addi  sp, sp, 8
34    ___________________ #perform multiplication
35    jalr  x0, ra, 0    #ret. to where last recursive call left
```

Fill in the missing code below.

line 11: _____

line 14: _____

line 29: _____

line 34: _____

Solution:

line 11: add t2, t3, x0 or mv t2, t3

line 14: blt or bne t0, a0, Fibonacci_loop

line 32: jal Factorial or jal ra, Factorial

line 37: mul a1, a1, t5

When n=3, fill in the stack space below with the stored value during the execution of the above code. The stack space grows downward. For return address, use the corresponding line number to represent the address of an instruction/label. (**Hint:** fill in all the stack space which has been used. Leave it blank if you do not use all the space below.)

Stack pointer to here when entrance

Solution: From top to down, the values are 16 (or 17), 3, 30, 2, 30, 1, 30, 0.

4 (c) Translate the instructions below to machine code written in **hexadecimal**. [4 points]

line 12: `add a1, t3, x0` _____

line 35: `jalr x0, ra, 0` _____

Solution:
line 12: 0x000E05B3

line 38: 0x00008067

2 (d) Calculate the target address in **hexadecimal** that `jalr` jumps to. [2 points]

```
lui x5, 0xFFFFF

jalr ra, x5, 0x123
```

The target address is _____

Solution:
0xFFFFF123

6. **Call convention/linker/loader/assembler [6 points]**

2 (a) Which register(s) is(are) used for returning values from a function in RISC-V calling convention?_____

A. a1
B. s0
C. ra
D. a0

Solution: A & D

2 (b) Which of the following statement(s) are(is) true?_____

A. Caller-saved registers includes registers used for storing local variables or passing function arguments, while callee-saved registers includes registers used for storing global variables.
B. Caller-saved registers are preserved across function calls, while callee-saved registers may be modified by the callee.
C. Caller-saved registers are saved by the caller before calling the callee if required, while callee-saved registers are saved by the callee.

Solution: C

2 (c) When are all the machine code bits decided for the following assembly instructions:

(1) `sub x2, x2, x3`
(2) `jal x1, malloc`

_____

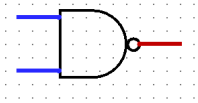
A. (1) & (2) after compilation
B. (1) after assembly, (2) after loading
C. (1) & (2) after assembly
D. (1) after assembly, (2) after linking

Solution: D

7. **Logic [10 points]**

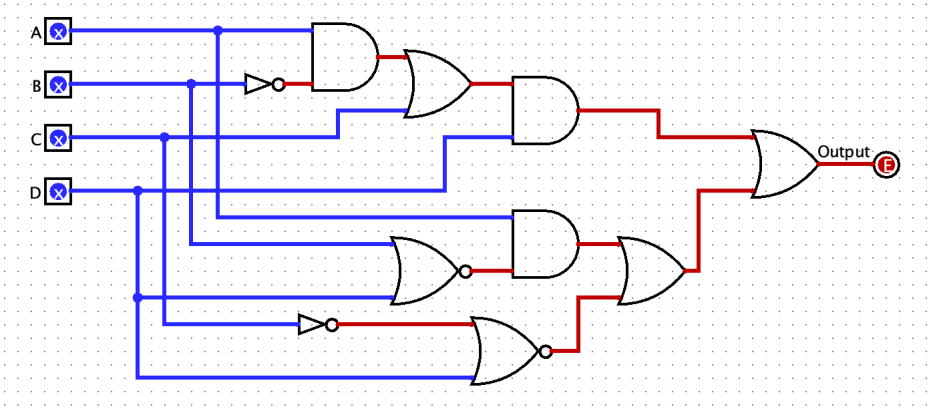
1 (a) What is the name of the following gate:_____

A. AND
B. OR
C. NAND
D. NOR



Solution: C

6 (b) Please write down the truth table of the circuit below and draw its Karnaugh map (A&B as a group; C&D as a group).



Solution: You can simplify the boolean expression first and obtain the truth table, which is faster.

$$(A\overline{B}+C)D+A(\overline{B+D})+\overline{\overline{C}+D}=A\overline{B}D+CD+A\overline{BD}+C\overline{D}=A\overline{B}+C$$

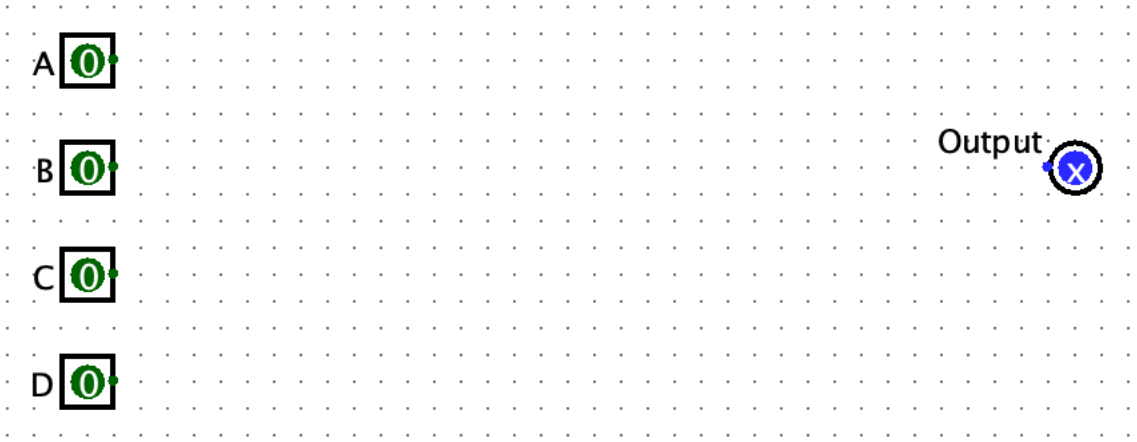
Truth table:

A	B	C	D	Output
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

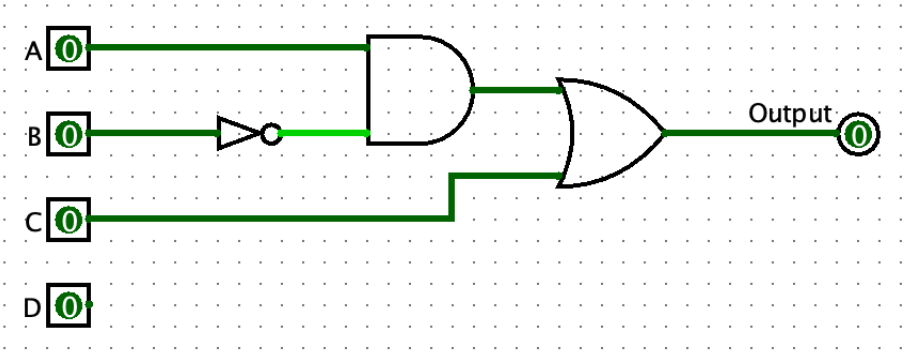
Karnaugh map:

CD \ AB	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	1	1	1

3 (c) Simplify the circuit using Karnaugh map and re-design the circuit using the least number of gates. You may use only `AND, OR and NOT` gates.



Solution: Optimal solution for this is



8. **Finite State Machine [20 points]**

1 (a) Below shows a state transition diagram of a finite state machine (FSM) with 4 states. What is the type of the FSM?

A. A Moore machine.

B. A Mealy machine.

Solution: B.

2 (b) Assume the input bit sequence to the FSM is `10011010`, the output is _____.

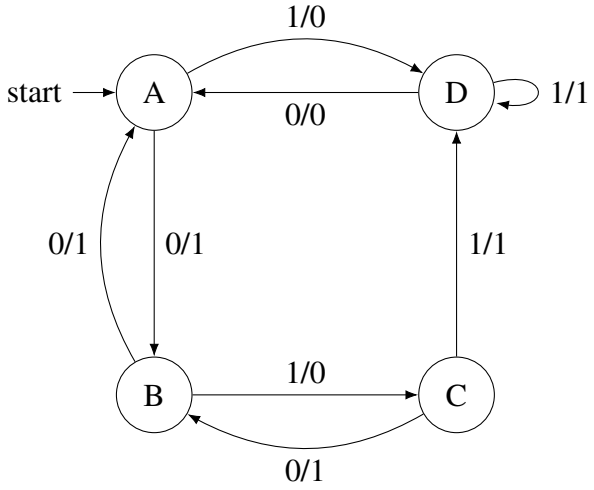
Solution: 00101000.

2 (c) Which state does the FSM arrive at last?_____

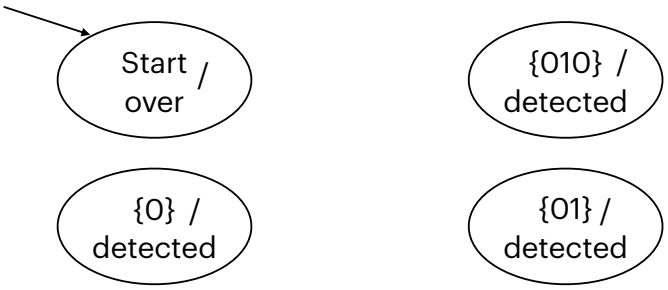
A. A

B. B
C. C
D. D

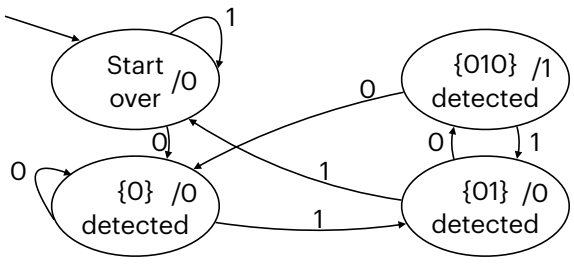
Solution: A.



4 (d) Build a **Moore** FSM model to detect "010" pattern in a bit sequence (use overlapping, i.e., the tail 0 of "010" can be considered as the head 0 for the next detection). The states are given below. Please complete the state transition diagram by adding the transitions, transition conditions and output for each state. [4 points]



Solution:



4 (e) Assign "00" (0) to represent state "Start over", "01" (1) to represent "{0} detected", "10" (2) to represent "{01} detected" and "11" (3) to represent "{010} detected". Write down

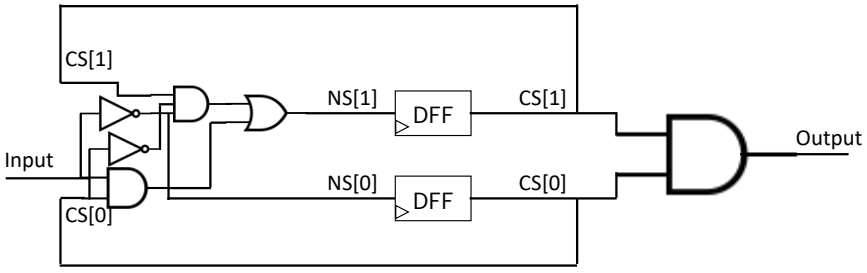
the truth table for the next-state and output logic. We use "CS" to represent current state and "NS" for next state.

Solution:

CS[1]	CS[0]	input	NS[1]	NS[0]	output
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	1	0	1

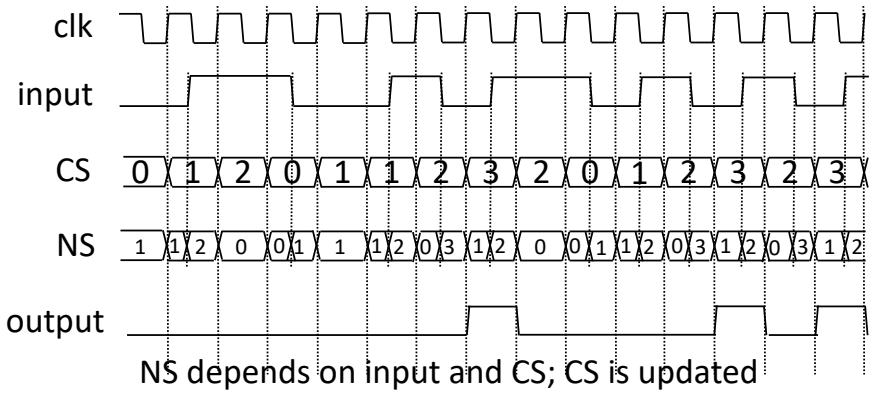
3 (f) Complete the circuit below for the "010" sequence detection task using the truth table you just wrote.

Solution:



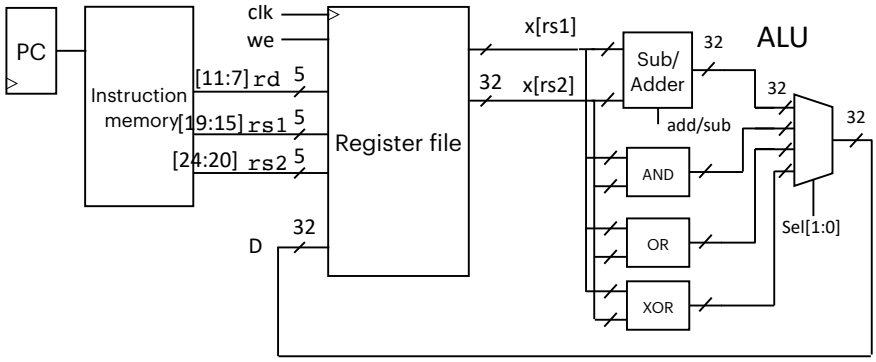
4 (g) Draw the timing diagram given the clock signal and input below. We ignore the non-ideal effects, and integers (use signal grouping) are used to represent the states.

Solution:



9. **RISC-V datapath [17 points]**

Below is the datapath we learned from class for some RISC-V R-type instructions:



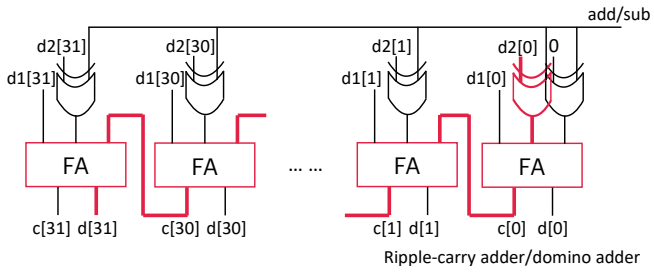
We will estimate the maximum clock frequency step by step. By "delay", we refer to **propagation delays**, unless stated otherwise. The delay of each element is shown in the table below.

Circuit	2-input AND	2-input OR	2-input XOR	DFF clk-to-Q
Delay (ps)	10	15	50	20
Circuit	2-to-1 multiplexer	5-32 decoder	1-bit full adder	3-input AND gate
Delay (ps)	15	100	30	15

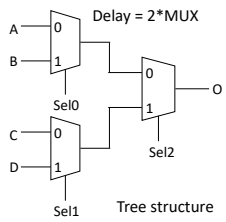
6 (a) **Delay of the ALU [6 points]**

The ALU consists of functional units such as the adder/subtracter (circuit shown below) and different types of logic gates, and a 4-to-1 multiplexer built from the 2-to-1 multiplexer using tree structure. Calculate the delay of the adder/subtracter and then the maximum delay of the ALU.

Solution:



add/sub delay $= XOR_{delay} + 32 * (\text{1-bit bull adder})_{delay} = 50 + 960 = 1010$ ps.

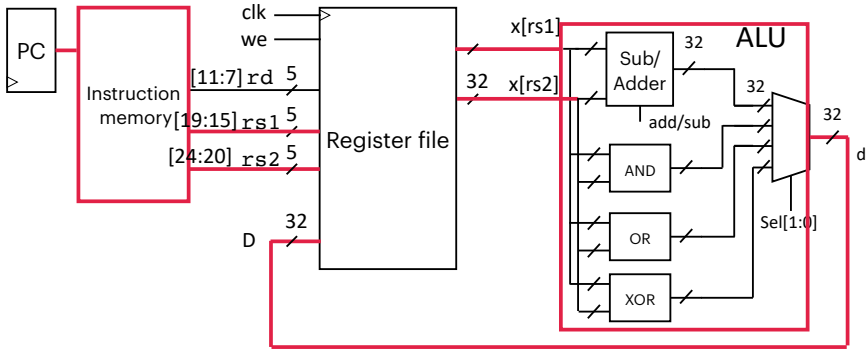


4-to-1 MUX delay = 2*2-to-1 MUX delay = 30 ps.

ALU max delay = 4-to-2 MUX delay + add/sub delay = 1040 ps.

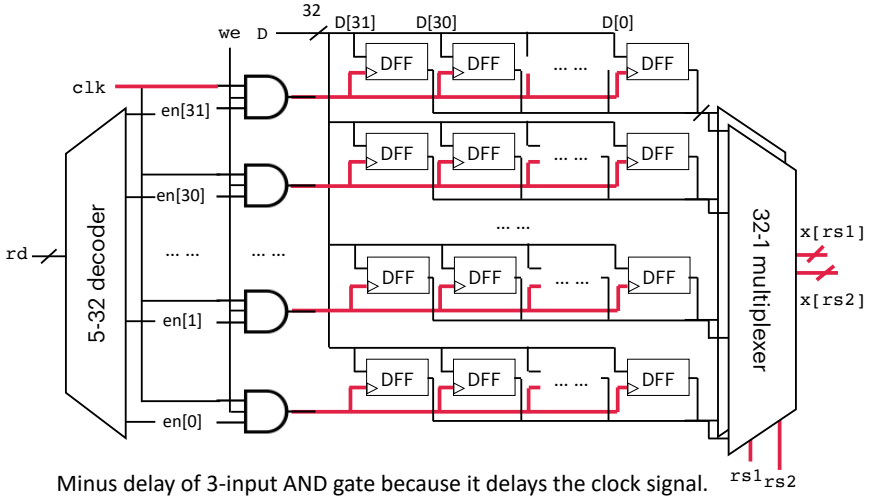
9 (b) **Delay of the datapath [9 points]**

The detailed circuit of the register file is shown below. Assume all the DFFs has a setup time of 50 ps. Indicate the elements of which the delay should be included to calculate the minimum clock cycle and the corresponding delay numbers. After that, calculate the maximum frequency of this datapath. Again, assume the 32-to-1 multiplexer is built from 2-to-1 multiplexer by the tree structure. Instruction memory delay is 80 ps.



Solution:

Critical path = clock-to-q (PC) + Imem delay + Reg. file delay + ALU delay + setup time (Reg. file).



Minus delay of 3-input AND gate because it delays the clock signal.

Reg. file delay in total = 32-1 MUX delay − 3-input AND delay
= 5 × 2-1 MUX delay − 3-input AND delay = 60 ps.

Minus the delay of the 3-input AND gate because it delays the clock signal, resulting in extra time for DFF setup in the register file.

Critical path = 20 + 80 + 60 + 1040 + 50 = 1250 ps.

Max frequency = 1/(Critical path delay) = 800 MHz.

2 (c) Use the "funct3" field of the R-type instructions to generate the multiplexer select signal for the ALU. Assume the multiplexer selects add/sub, AND, OR and XOR results when the select signal is 00, 01, 10, 11, respectively.

Solution:

	funct3[2]	funct3[2]	funct3[2]	Sel[1]	Sel[0]
add/sub	0	0	0	0	0
AND	1	1	1	0	1
OR	1	1	0	1	0
XOR	1	0	0	1	1

$$Sel[1] = funct3[2] \cdot \overline{funct3[0]}$$

$$Sel[0] = funct3[2] \cdot (funct3[1] \cdot funct3[0] + \overline{funct3[1]} \cdot \overline{funct3[0]})$$