Computer Architecture I Final Exam

Chinese Name: ______________________

Pinyin Name: ______________________

Student ID: ______________________

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

Question:	1	2	3	4	5	6	7	8	9
Points:	1	4	6	9	9	4	4	2	6
Score:									

Question:	10	11	12	13	14	15	16	17	Total
Points:	8	9	8	4	6	6	9	5	100
Score:									

- This test contains 31 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.
- Unless told otherwise always assume a 32bit 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 31 times).

4 2. **Memory of C**

Read the following code, then fill the blanks.

```c
#include <stdlib.h>
int main() {
    char char_a = 'c';
    char *str_a = "str a";
    char str_b[] = "str b";
    char *str_c = malloc(10);
    return 0;
}
```

Each of the following expressions represent a memory location. Fill the correct memory section (static, stack, heap) they belong to.

`char_a` ____________________

`str_a[0]` ____________________

`str_b[0]` ____________________

`str_c[0]` ____________________

Solution: stack; static; stack; heap.

3. RISC-V programming

In this question, you are required to understand the given RISC-V code `func` and answer some questions about it. Assume there is no cache or memory protection.

```
func:
    addi    a0, a0, 0
    la      t0, func
    lh      t1, 2(t0)
    addi    t1, t1, 16
    sh      t1, 2(t0)
    ret
```

$\boxed{1}$ (a) Translate the following RISC-V instruction into machine code (in hexadecimal).

`addi    a0, a0, 0` ____________________

Solution: `0x00050513`

$\boxed{3}$ (b) Think about what this function does and answer the following questions:

What does `func(0)` output when it is called **the first time**? ______________

What does `func(0)` output when it is called **the second time**? ______________

What does `func(1)` output when it is called **the third time**? ______________

Hint: The answer from the last question may help once you translate it in binary!

Solution: 0, 1, 3

$\boxed{2}$ (c) If we change `lh t1, 2(t0)` into `lb t1, 3(t0)` and `sh t1, 2(t0)` into `sb t1, 3(t0)`, what does `func(0)` output when it is called the third time?

____

Solution: 512

4. **Number representation**

3 (a) What is `0xB0E` in decimal?

____

What is 339_{10} in hexadecimal?

____

What is the decimal equivalent of the 6-bit two's complement number `0b101011`?

____

Solution: `0xFADEDABE`; 2830; `0x153`; -21.

2 (b) Write T or F in each cell.

	unsigned	sign & magnitude	2's complement
Can represent positive numbers			
Can represent negative numbers			
Has more than one representation for 0			
Uses the same addition process as unsigned			

Solution:

unsigned	sign & magnitude	2's complement
T	T	T
F	T	T
F	T	F
T	F	T

4 (c) Consider a new 5-bit floating point format *STAR* `SEEEF`, which contains sign, exponent, significant (fraction) and follows the same rules as the 32-bit IEEE 754

standard (denorms, biased exponent, non-numeric values, etc.), but allocates its bits differently.

number of NaNs (decimal): ____________________

exponent bias (decimal): ____________________

smallest positive denorm (in forms of *STAR*): ____________________

negative infinity (in forms of *STAR*): ____________________

Solution: 2; 3; `0x1`; `0x1E`.

$\boxed{9}$ 5. **CALL**

Multiple Choice. For each question select which option correctly answers the question. The options are,

A. Compiler; B. Assembler; C. Linker; D. Loader.

Fill your answer (A, B, C or D) in the table below, each question has **only one** correct answer.

1. Its output includes pseudo-instructions.
2. Prepares the virtual address space for the static section.
3. Fills in the final value for the immediate in jump instructions. Note that the label we are jumping to exists in a different file than the jump instruction.
4. Uses Lexers to process the input into tokens.
5. Copies instructions and data from executable file into the address space.
6. Incorporates statically-linked libraries.
7. Be responsible for loop unrolling.
8. Produces executable file containing text and data (plus header).
9. Prepares the relocation table.

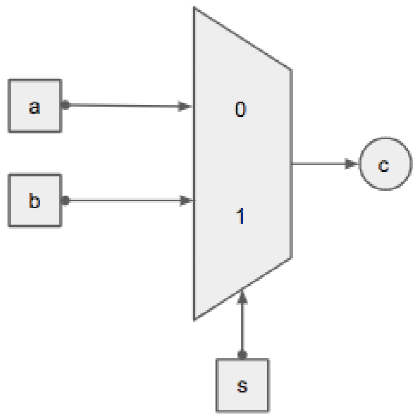
1	2	3	4	5	6	7	8	9

Solution: A; C; C; A; D; C; A; C; B.

6. **1-bit-wide Mux**

$\boxed{2}$ (a) 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. Please fill in the truth table presented. If one element is wrong, you will lose all points.

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



Solution:

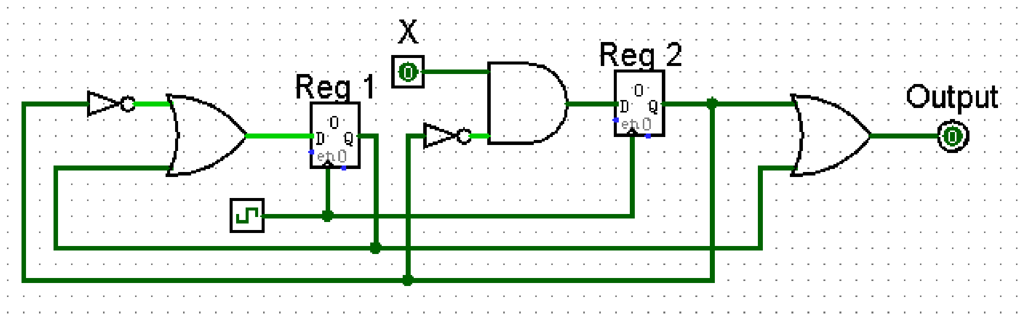
s	a	b	c
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

$\boxed{2}$ (b) According to the truth table, please write down two boolean algebra expressions on c with respect to a, b and s, one is directly from the truth table and no simplifications, the other is the most simplified one.

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

7. **Synchronous Digital Systems**

Consider the following circuit. Assume the clock has a frequency of 50 MHz, all gates have a propagation delay of 5 ns, X changes 9 ns after the rising edge of clk, Reg1 and Reg2 have a clk-to-q delay of 2 ns.



$\boxed{2}$ (a) What is the **longest possible setup time** such that there are no setup time violations? Please show your calculation steps, only an answer will get no point.

Solution:

The clock period is $\frac{1}{50\times10^6}s = 20\ ns$.

Reg1 longest possible setup time: the path is *the output of Reg2 → NOT → OR*, with a delay of 2 ns + 5 ns + 5 ns = 12 ns. So 20 - 12 = 8 ns.

Reg2 longest possible setup time: the path is *X changes → AND*, with a delay of 9 ns + 5 ns = 14 ns. So 20 - 14 = 6 ns.

So longest setup time: min(8 ns, 6 ns) = 6 ns.

$\boxed{2}$ (b) What is the **longest possible hold time** such that there are no hold time violations? Please show your calculation steps, only an answer will get no point.

Solution:

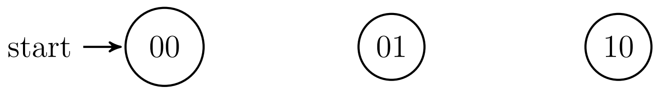
Reg1 longest possible hold time: the path is *the output of Reg1* $\rightarrow OR$, with a delay of 2 ns + 5 ns = 7 ns.

Reg2 longest possible hold time: the path is *the output of Reg2* $\rightarrow NOT \rightarrow AND$, with a delay of 2 ns + 5 ns + 5ns = 12 ns.

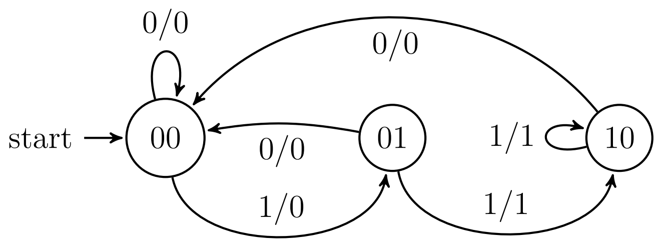
So longest hold time: min(7 ns, 12 ns) = 7 ns.

$\boxed{2}$ 8. **Finite State Machine**

Draw a FSM that outputs 1 when it receives two or more successive '1'.

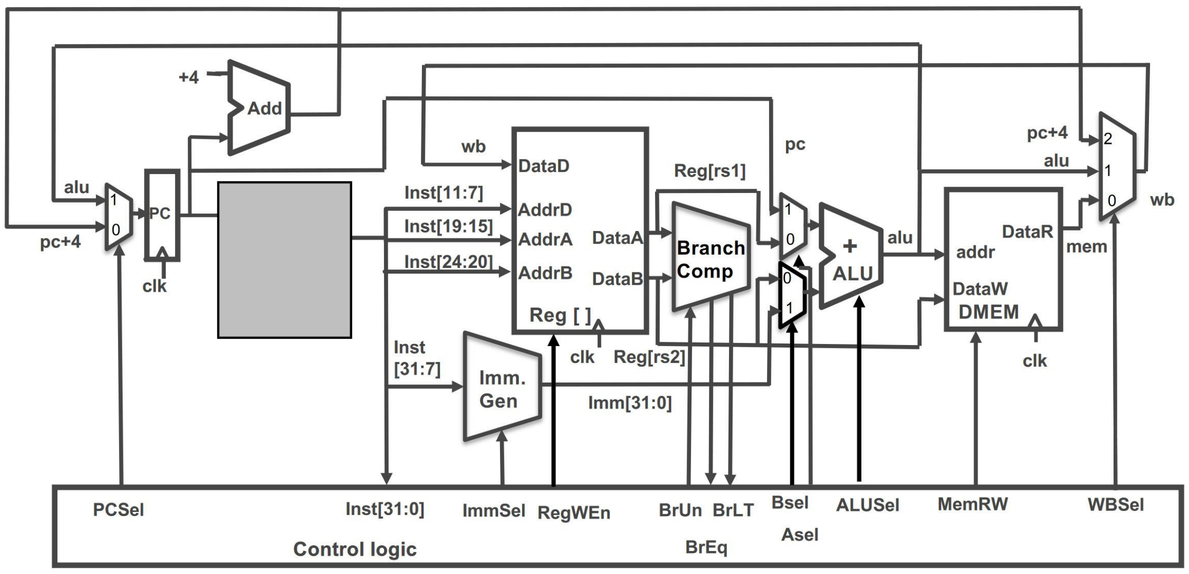


Solution:



9. **RISC-V Datapath**

Here is the datapath we learnt from class:



4 (a) Write down control signals for `jal 64`. Please use * to indicate that what this signal is does not matter.

PCSel	ImmSel	RegWEn	BrUn	BrEq	BrLT
ASel	BSel	ALUSel	MemRW	WBSel	-
					-

Solution:

PCSel	ImmSel	RegWEn	BrUn	BrEq	BrLT
ALU	J	1	*	*	*
ASel	BSel	ALUSel	MemRW	WBSel	-
1	1	Add	Read	2	-

1 (b) Which register saves the return address of the instruction above?

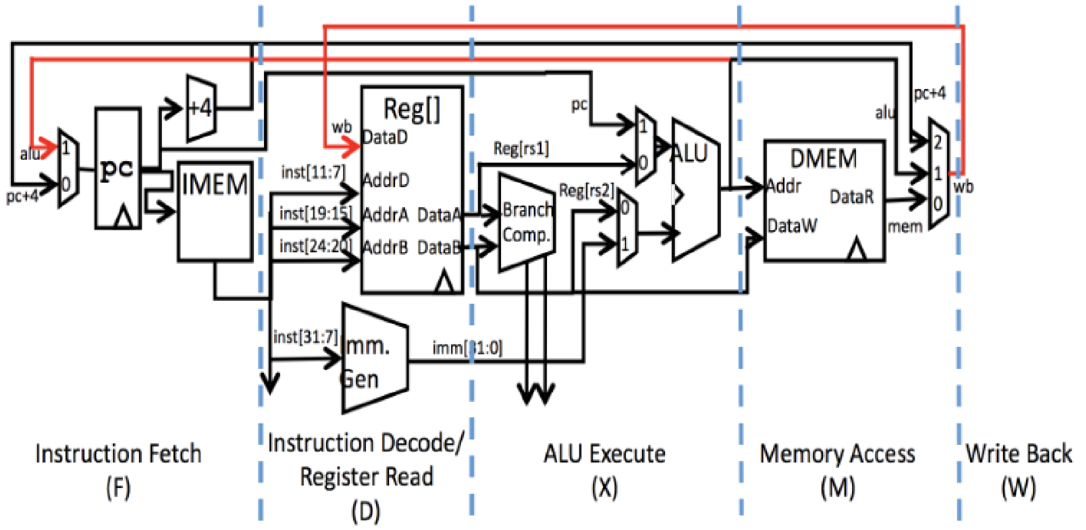
Solution: `x1` or `ra`.

1 (c) The datapath above is missing some details in the grayed out box. Describe the function of this missing part.

Solution: Output the instruction corresponding to the address given by PC.

10. **RISC-V Pipeline and Hazard**

Consider a typical 5-stage (Fetch, Decode, EXecute, Memory, WriteBack) pipeline. Assume pipeline registers exist where the dotted lines are. For questions that require calculation, you should show your calculation process. Only giving a solution will receive no point. Provide your answer in fraction if it is indivisible.



Register clk-to-q	30 ps	**Branch comp.**	75 ps	**Memory write**	200 ps
Register setup	20 ps	**ALU**	200 ps	**RegFile read**	150 ps
Mux	50 ps	**Memory read**	250 ps	**RegFile setup**	20 ps
Immediate Gen.	50 ps				

$\boxed{2}$ (a) With the delays provided above for each of the datapath components, what would be the fastest possible clock time for a single cycle datapath?

Solution:

$$\begin{aligned} t_{clk} &= t_{PCclk2q} + t_{IMEMread} + t_{RFread} + t_{mux} \\ &\quad + t_{ALU} + t_{DMEMread} + t_{mux} + t_{RFsetup} \\ &= 30 + 250 + 150 + 50 + 200 + 250 + 50 + 20 \\ &= 1000ps \end{aligned}$$

1 GHz.

$\boxed{2}$ (b) What is the fastest possible clock time for a pipelined datapath?

Solution:

$$IF : t_{PCclk2q} + t_{IMEMread} + t_{Regsetup} = 30 + 250 + 20 = 300ps$$
$$ID : t_{Regclk2q} + t_{RFread} + t_{Regsetup} = 30 + 150 + 20 = 200ps$$
$$EX : t_{Regclk2q} + t_{mux} + t_{ALU} + t_{Regsetup} + t_{mux} = 350ps$$
$$MEM : t_{Regclk2q} + t_{DMEMread} + t_{mux} + t_{Regsetup} = 350ps$$
$$WB : t_{Regclk2q} + t_{RFsetup} = 30 + 20 = 50ps$$

$\frac{1000}{350}$ GHz

$\boxed{2}$ (c) What is the speedup from the single cycle datapath to the pipelined datapath? Under what situation can we achieve perfect speedup of 5?

Solution:

$\frac{1000}{350}$;

1. No hazards,
2. No pipeline register delay,
3. All pipeline stages have equal delay.

$\boxed{2}$ (d) When do **structural hazards** occur? How can we resolve them?

Solution:

Structural hazards occur when more than one instruction needs to use the same datapath resource at the same time. Structural hazards can always be resolved by adding more hardware.

11. **Cache**

Consider a 16 way set associative cache with two word blocks, 16 sets and a 4 GiB physical byte addressed address space. (4 bytes for a word)

Use the cache above, consider the following function when it is invoked with parameters `a = 0x10000` and `b = 0x20000`. The cache is empty before the function is invoked.

```c
void sequence(int *a, int *b) {
    int i;
    /* PART C */
    for (i = 0; i < 16; i++) {
        b[i] = 2;
        a[i] = 4;
    }
    /* PART D */
    for (i = 16; i < 272; i++) {
        b[i] = b[i - 8] + a[i - 8];
        a[i] = b[i - 16] + a[i - 16];
    }
}
```

$\boxed{3}$ (a) Calculate the bit width of tag, index, and offset bits. Fill your answer in the table below.

TAG	Set Index	Block Offset

Solution: 25; 4; 3.

$\boxed{2}$ (b) What is the cache hit rate of the **PART C** loop? (Correct answer can get full points)

Solution: 1/2.

$\boxed{2}$ (c) What is the cache hit rate of the **PART D** loop when running sequentially after **PART C** loop? (Correct answer can get full points)

Solution: 5/6.

$\boxed{2}$ (d) Assume that some sequence was run on a computer with an L1 and L2 cache. Say that the L1 cache has an access time of 10ns, the L2 cache has an access time of 20ns, main memory has an access time of 50ns, the L1 cache has an 80% hit rate, and that the total AMAT for running sequence is 16 ns. What is the local hit rate for the L2 cache? Please write your answer as a decimal.

Solution:

10 + 0.2 * (20 + L2_miss_rate * 50) = 16

L2_miss_rate = 0.2

L2_local_hit_rate = 0.8

12. **OpenMP Integration**

We try to accelerate the calculation of π under the assistance of **OpenMP**. Read the following code and answer the questions.

```
#include<omp.h>
double calculate_pi(int num_steps) {
    double result = 0, local = 0;
    double step = 1.0 / ((double) num_steps);
    int i;
    #pragma omp parallel
    {
        #pragma omp for
        for (i = 0; i < num_steps; i++) {
            double x = (i + 0.5) * step;
            local += 4.0 * step / (1.0 + x * x);
        }
        result += local;
    }
    return result;
}
```

$\boxed{4}$ (a) Identify the data sharing attributes of the following variables with shared or private.

`result` ______________ `i` ______________

`local` ______________ `x` ______________

Solution: shared; shared; private; private.

$\boxed{2}$ (b) This code has two bugs. Identify each of them with a short statement.

____

____

Solution:

Variable `local` should be private;

There is a data race when adding `local` to `result`.

$\boxed{2}$ (c) Propose a fix towards bugs you found in (b) with only one line of modification. Indicate the line number and your changes. (Required: use reduction)

Solution:

Line 6. `#pragma omp parallel private(local) reduction(+:result)`

13. Threading

Read the following code and two impossible outputs, answer questions.

```c
#include <omp.h>
#include <stdio.h>
int main() {
    omp_set_num_threads(2);
    #pragma omp parallel
    {
        int i;
        #pragma omp for
        for (i = 0; i < 4; i++) {
            printf("tid %d: i is %d", omp_get_thread_num(), i);
        }
        printf("tid %d: finished", omp_get_thread_num());
    }
    return 0;
}
```

Impossible Output 1:

```
tid 0: i is 0
tid 1: i is 2
tid 0: i is 1
tid 0: finished
tid 1: i is 3
tid 1: finished
```

Impossible Output 2:

```
tid 0: i is 0
tid 1: i is 1
tid 1: i is 3
tid 0: i is 2
tid 1: finished
tid 0: finished
```

$\boxed{2}$ (a) Explain why Impossible Output 1 is not possible.

____

____

Solution:

There is an **implicit barrier** synchronization at the end of OpenMP for loop.

$\boxed{2}$ (b) Explain why Impossible Output 2 is not possible in default. Also state the key concern of why the design of OpenMP for loop does not want this output to happen in default.

____

Solution:

OpenMP for loop divides index regions **sequentially per thread** in default. For potential spatial locality and cache optimization.

14. **Virtual Memory**

$\boxed{4}$ (a) **True or False.** Fill your answer (T or F) in the table below.

1. In a bare system without virtual memory, a program can modify any part of the memory.
2. Both base and bound memory system and paged memory system can run programs with larger memory than DRAM.
3. The TLB should be flushed after a context switch or when any new memory is allocated.
4. Increasing page table size can always lead to reduction in page faults.

1	2	3	4

Solution: T; F; F; F.

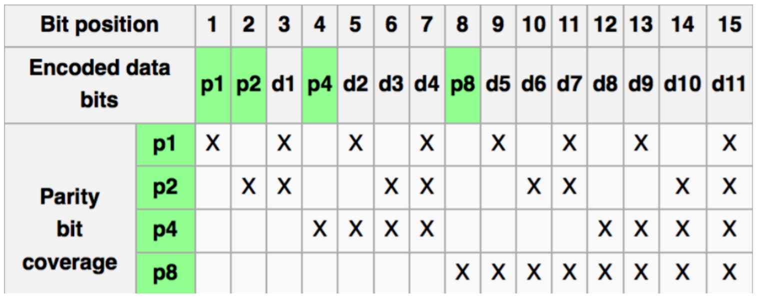
$\boxed{2}$ (b) Consider a machine with a physical memory of 4 GB, a page size of 4 KB, and a page table entry size of 8 bytes. How many levels of page tables would be required to map a 48-bit virtual address space if every page table fits into a single page? And how many bits are there for each level of page table index in virtual address?

Solution:

Since each PTE has 8 bytes and each page has 4 KB, there are 2^9 entries in each page, which means that a one-page page table points to 2^9 pages. Also, we can derive that the virtual address offset has 12 bits, so the VPN has 48 - 12 = 36 bits. Each page table has 2^9 entries so we need 4 level page tables to map 36 bits VPN and the index for each level has 9 bits.

15. **Dependability**

Hamming ECC (Error Correction Code) is a method of both detecting and correcting errors in data. The general rule of Hamming ECC can be shown visually.



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

For the rest of the tasks, we only consider **even** parity.

$\boxed{2}$ (a) Given a code word 101011_2, the following table shows *bit position* and its corresponding *data bit.*

bit position	1	2	3	4	5	6
data bit	1	1	0	1	0	1

How many bits do we need to add to this code word to enable single error correction?

____

Then, fill the encoded code word in the following table and **circle** your parity bits on its corresponding **bit position**. (Leave the unused data bit blank)

bit position	1	2	3	4	5	6	7	8	9	10	11	12
encoded data bit												

Solution:

4;

bit position	1	**2**	3	**4**	5	6	7	**8**	9	10	11	12
encoded data bit	1	1	1	0	1	0	1	1	0	1		

$\boxed{2}$ (b) Given a *Single Error Correction* Hamming Code: 1011000000_2. The following table shows *bit position* and its corresponding *data bit.* One of the data bit is flipped during the transmission. **Circle** all the error parity bits on its corresponding **bit position**.

bit position	1	2	3	4	5	6	7	8	9	10
encoded data bit	0	0	0	0	0	0	1	1	0	1

Then write down the **corrected decoded** code word in the following table. (Leave the unused data bit blank)

bit position	1	2	3	4	5	6	7	8
encoded data bit								

Solution:

bit position	**1**	2	3	**4**	5	6	7	8	9	10
encoded data bit	0	0	0	0	0	0	1	1	0	1

bit position	1	2	3	4	5	6	7	8
decoded data bit	0	1	0	1	0	1		

$\boxed{2}$ (c) **Multiple Choice.** Fill your answer in the table below. There may be **more than one** correct answer.

A. RAID 1 is good for no overhead.

B. RAID 3 employs the strategy of byte-level striping with single parity disk, which is inefficient to detect errors.

C. RAID 4 works well for small reads.

D. None of the above.

Answer

Solution: B, C.

16. **Various Questions**

$\boxed{7}$ (a) **True or False.** Fill your answer (T or F) in the table below.

1. One of the main tasks of the OS is to provide easy to use services to user program, like file systems and network stack.
2. You would like to use polling when doing IO when the data comes in very infrequently in order to achieve higher performance.
3. Very long instruction word (VLIW) processors rely heavily on compilers to produce correct answer and achieve high performance.
4. RAID0 splits data across multiple disks to provide higher throughput and reliability.
5. Advantages of FPGA include its higher performance compared to the conventional CPUs and higher flexibility compared to the ASICs.
6. Modern Internet uses hierarchical layers of protocols to allow data to be sent through many heterogeneous (异构的) components (e.g., Wi-Fi and ethernet cable).
7. One can only create no more than 4 threads with pthread or OpenMP in a program when the system has only 4 CPU cores, or the program will crash.

1	2	3	4	5	6	7

Solution: T; F; T; F; T; T; F

$\boxed{2}$ (b) **Multiple Choice.** Which of the following statements is true for interrupt? Fill your answer in the table below. There may be **more than one** correct answer.

A. Interrupt has higher latency than polling;
B. Interrupt requires a CPU to run all the time to receive a signal from a device;
C. Handling an interrupt requires interrupt handler, which adds overhead to the whole process;
D. Interrupt is good for devices that is always sending data to the CPU.

Answer

Solution: A, C.

$\boxed{5}$ 17. **SIMD**

In this question, you will implement a vectorized min function. The goal is to find the minimum element in an array of n signed 16-bit integers. You will need to compute partial minima that are stored in a vector register and finally reduce it down to a single element. You may find the following intrinsics are helpful.

1. `__m128i _mm_set1_epi16 (short a)`

   Description: Broadcast 16-bit integer a to all all elements of dst.

   Operation:
   ```
   FOR j := 0 to 7
       i := j*16
       dst[i+15:i] := a[15:0]
   ENDFOR
   ```

2. `__m128i _mm_loadu_si128 (__m128i const* mem_addr)`

   Description: Load 128-bits of integer data from memory into dst.

   Operation:
   ```
   dst[127:0] := MEM[mem_addr+127:mem_addr]
   ```

3. `void _mm_storeu_si128 (__m128i* mem_addr, __m128i a)`

   Description: Store 128-bits of integer data from a into memory.

   Operation:
   ```
   MEM[mem_addr+127:mem_addr] := a[127:0]
   ```

4. `__m128i _mm_min_epi16 (__m128i a, __m128i b)`

   Description: Compare packed signed 16-bit integers in a and b, and store packed minimum values in dst.

   Operation:
   ```
   FOR j := 0 to 7
       i := j*16
       dst[i+15:i] := MIN(a[i+15:i], b[i+15:i])
   ENDFOR
   ```

5. `__m128i _mm_alignr_epi8 (__m128i a, __m128i b, int imm8)`

   Description: Concatenate 16-byte blocks in a and b into a 32-byte temporary result, shift the result right by imm8 bytes, and store the low 16 bytes in dst.

   Operation:
   ```
   tmp[255:0] := ((a[127:0] << 128)[255:0] OR b[127:0]) >> (imm8*8)
   dst[127:0] := tmp[127:0]
   ```

Please fill in the blanks.

```
#define MAX_VALUE ((1 << 15) - 1)
/* n for a’s size, n is multiple of 8 */
int16_t fast_min(size_t n, int16_t *a) {
    /* Init elements to minimum value */
    size_t i;
    __m128i min_vec = _mm_set1_epi16(MAX_VALUE);
    for (i = 0; i < n / 8 * 8; i += 8) {

        ____________________________________________________________________;

        ____________________________________________________________________;

        ____________________________________________________________________;
    }
    /* Reduction step */
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(__________, __________, ____));
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(__________, __________, ____));
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(__________, __________, ____));
    /* Get the min value so far */

    ________________________________________________________________________;

    ________________________________________________________________________;

    ________________________________________________________________________;
}
```

Solution:

```
#define MAX_VALUE ((1 << 15) - 1)
/* n for a’s size, n is multiple of 8 */
int16_t fast_min(size_t n, int16_t *a) {
    /* Init elements to minimum value */
    size_t i;
    __m128i min_vec = _mm_set1_epi16(MAX_VALUE);
    for (i = 0; i < n / 8 * 8; i += 8) {
        __m128i temp_vec = _mm_loadu_si128((__m128i *)(a + i));
        min_vec = _mm_min_epi16(min_vec, temp_vec);
    }
    /* Reduction step */
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(min_vec, min_vec, 8));
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(min_vec, min_vec, 12));
    min_vec = _mm_min_epi16(min_vec,
                            _mm_alignr_epi8(min_vec, min_vec, 14));
    /* Get the min value so far */
    int16_t result[8];
    _mm_storeu_si128((__m128i *)result, min_vec);
    return result[7];
}
```