Computer Architecture I Final (Solution)

Chinese Name: _____

Pinyin Name: _____

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

Question	Points	Score
1	1	
2	44	
3	18	
4	18	
5	15	
6	20	
7	19	
8	15	
Total:	150	

- This test contains 18 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 (one of those can be printed).
- The estimated time needed for each of the 7 topics is given in parenthesis - it is 36 minutes for question 1 (Various Questions) and about 15 minutes for each of the 6 others. The total estimated time is 120 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 14 times).

2. **Various Questions** (36 minutes)

[5] (a) This subquestion involves T / F questions. Incorrect answers on T / F questions are penalized with negative credit. Circle the correct answer.

T / F : ECC provides protection from disk failures.
T / F : A single parity bit allows us to detect any bit errors we have, but we need Hamming ECC (or something similar) to correct them.
T / F : All RAID configurations improve reliability.
T / F : All RAID configurations improve performance.
T / F : MapReduce is intended to run on a single, multi-core machine.
T / F : Exceptions in early pipeline stages override exceptions in later stages for a given instruction.
T / F : Exceptions are handled in the pipeline stage where they occur.
T / F : PUE does not measure the power efficiency of Warehouse Scale Computer servers.
T / F : Amdahl's law is restricting the speed increase predicted by Moore's Law.
T / F : We can implement the atomic test-and-set operation in MIPS using lw and sw.

Solution: FFFTFTFTFF

[3] (b) Consider attaching a hard disk to a CPU. Should we use polling or interrupts to deal with the hard disk? Should we use a DMA engine? Explain.

We should use DMA + interrupts. Since we transfer data in large chunks (pages), it makes sense to give control to another program when we get a page fault and let the DMA engine do the page transfer from disk to memory. We then receive an interrupt when the transfer is completed.

[3] (c) What does AMAT stand for and how is it defined?

Solution: Average Memory Access Time = $t_{hit} + rate_{hit} * t_{miss}$

[6] (d) For a single, running process, where in memory are the `heap`, `stack`, `code` and `static data` located? For each memory types, write if they can (typically) change in size and if so, how do they grow.

Solution: Stack: at the top of the address space, growing downwards (towards lower addresses)
Heap: in between, growing: allocated at free space

Data: at the bottom above the text, does not change size
Text: at the very bottom, does not change size (unless you load a dll during runtime)

2 (e) Do different processes have (typically) access to each other's memory (Yes/ No). What is the name of the technique that is responsible for this?

(e) **No, Virtual Memory**

3 (f) What does TLB stand for and what does it do?

(f) **Translation Lookaside Buffer; Cache for the Virtual Memory Page Tables**

3 (g) What are the advantages and disadvantages between using a static library and a dynamically linked library?

Solution: DLL advantages:
program requires less disk space (if dll is shared between programs)
execution of 2 programs requires less memory - also slight performance gain due to better caching
a replaced (de-bugged) library automatically improves the program that uses this library.
DLL disadvantage:

At loading or runtime there is overhead to (dynamically) link
having the executable is not enough - you need (the correct major version) of the library, too - complex

(g) _____

6 (h) Briefly explain what happens between switching on a computer and it being fully started up (4 keywords and very brief explanations for each).

Solution: BIOS: find storage device (HDD, flash) and load first sector (bootloader)
Bootloader: load OS kernel from file system of the HDD into memory
Operating System boot: Initialize devices, drivers, etc.; start init
Init: launch applications: fork process

(h) _____

2 (i) Which of the following can increase the availability?

a. Increasing MTTF
b. Decreasing MTTF
c. Increasing MTTR
d. Decreasing MTTR
e. Redundant data copies

(i) **a, d, e**

2 (j) Explain very briefly why RAID1 is the most expensive form of RAID.

(j) **Mirrored dist $->$ 100 % overhead**

2 (k) Represent the following decimal number using 2's complement:

-68(dec) = 0x _____ = 0b _____

Solution: -68(dec) = 0xFFFFFFBC = 0b 1111 1111 1111 1111 1111 1111 1011 1100

2 (l) Using IEEE 754 representation, what decimal number is encoded?

0xC2C00000 = _____

Solution: 0xC2C00000 = -96(or $1.5*2^6$)

1 (m) We represent an unsigned integer. What decimal number is encoded? You may use the size prefixes for memory (MIPS Green Sheet).

0xC0000000 = _____

Solution: 0xC0000000 = 3G (3 * 1024 * 1024 * 1024)

4 (n) Consider the truth table below. On the right side, first extract the "Sum of Products" from the table. Afterwards simplify it.

A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Solution: SoP: $X = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + A\bar{B}\bar{C}D + AB\bar{C}D$
$= \bar{C}\left(\bar{D} + \bar{A}B\bar{D}\right) = \bar{C}D + \bar{A}B\bar{C}\bar{D}$
Even better: $= \bar{C}D + \bar{A}B\bar{C}$

3. **C Programming** (15 minutes)

7 (a) The following function should allocate space for a new string, copy the string from the passed argument into the new string, and convert every lower-case character in the new string into an upper-case character. Fill in the blanks and the body of the for() loop. You may **not** declare any new variable.

```
char* upcase(char* str){
  char* p;
  char* result;

  result = (char*) malloc(strlen(str)+1);
  strcpy(result, str);
  for(p = result; *p != '\0'; p++){
    if(*p >= 'a' && *p <='z')
      *p +='A' - 'a';
  }
  return result;
}
```

3 (b) Please explain what a dangling pointer is:

Solution: A pointer initially holding valid address, but later the held address is released or freed. Then such a pointer is called as dangling pointer.

3 (c) Please correct the following program by writing the correct version on the right side:

```
swap( int* p1, int* p2 )
{
   int  *p;
  *p = *p1;
  *p1 = *p2;
  *p2 = *p;
}
```

Solution:

```
void swap( int* p1, int* p2 )
{
```

```
int p;
p =*p1;
 *p1 =*p2;
 *p2 = p;
}
```

1 (d) The system program that combines separately compiled modules of a program into a form suitable for execution is called:
A. Assembler
B. Loader
C. Linker
D. None of the Above

(d) **C**

1 (e) Which flag would you put in a compilation command (e.g. gcc) to include debugging information?
A. -o
B. d
C. g
D. debug

(e) **C**

1 (f) At the end of the compiling stage, the symbol table contains the ___ of each symbol.
A. relative address
B. absolute address
C. the stack segment beginning address
D. the global segment beginning address

(f) **A**

1 (g) beq and bne instructions produce ____ and they _____.
A. PC-relative addressing, never relocate
B. PC-relative addressing, always relocate
C. Absolute addressing, never relocate
D. Absolute addressing, always relocate

(g) **A**

1 (h) j and jal instructions add symbols and ___ to ___.
A. instruction addresses, the symbol table
B. symbol addresses, the symbol table
C. instruction addresses, the relocation table
D. symbol addresses, the relocation table

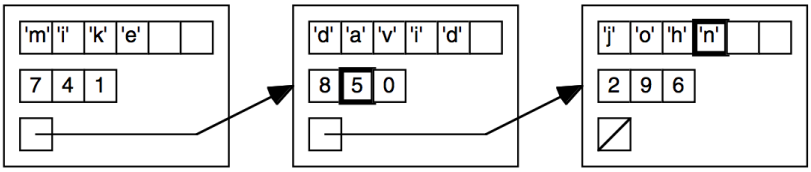
(h) **C**

4. **MIPS & C programming** (12 minutes)

Consider a list with nodes defined in C as follows:

```
struct ListNode {
   char name[6];
   int code[3];
   struct ListNode* next;
};
```

The diagram below, not drawn to scale, gives an example of such a list.



5 (a) Assume that register $a0 contains a pointer to the first node of the list. Write a MIPS assembly language function called `getty1` that returns (in $v0) with the second integer in the second node in the list (with the list pictured above, this will load a 5 into $v0). You don't need to do any error checking (assume node is existing). Adhere to the standard MIPS programming conventions but be as brief as possible!

```
getty1:
     lw $t0, 20($a0) # // 20 because the int is memory aligned
        to 4 byes - so 2 padding bytes will be used!
     lw $v0, 12($t0)
     jr $ra
```

6 (b) Again assume that register $a0 contains a pointer to the first node of the list. Write a MIPS assembly language function called `getty2` that returns (in $v1) the fourth character in the third node in the list (with the list pictured above, this will load 'n' into $v1). Again, you don't need to do any error checking (assume node is existing and string long enough). Adhere to the standard MIPS programming conventions but be as brief as possible!

```
getty2:
     lw $t0, 20($a0) # // second node in $t0
     lw $t0, 20($t0) # // third node in $t0
     lb $v1, 3($t0)  # // load the 3rd character
     jr $ra
```

7 (c) Read and understand the following MIPS assembly code. Then translate this function into C code.Your answers should be as concise as possible. The parameter x in register $a0 is an unsigned int.

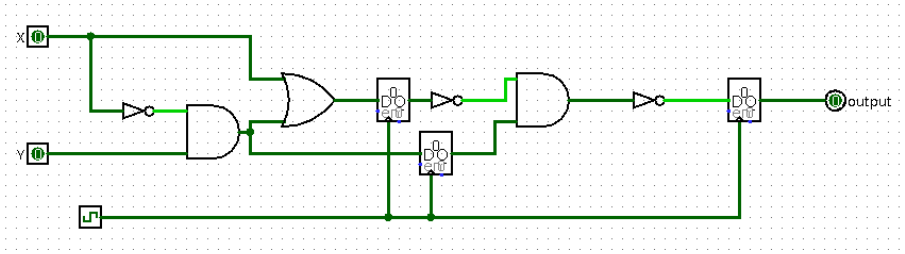
```
#####################################################################
## BitCount
##  $a0 = x, $v0 = return value
#####################################################################
BitCount:
  addi $sp, $sp, -8
  sw $ra, 4($sp)
  sw $s0, 0($sp)
  add  $v0, $0, $0
  beq  $a0, $0, end
  andi $s0, $a0, 1
  srl  $a0, $a0, 1
  jal  BitCount
  add  $v0, $v0, $s0
end:
  lw   $ra, 4($sp)
  lw $s0, 0($sp)
  addi $sp, $sp, 8
  jr $ra
```

Solution:

```
int BitCount(unsigned x) {
     int bit=0;
     if (x == 0) return 0;
     bit = x & 0x1;
     return bit + BitCount(x >> 1);
}
```

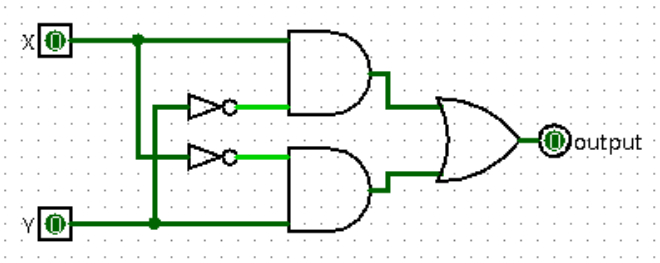
5. **SDS** (12 minutes)

[5] (a) Consider the following circuit, the delay is 50 ns for the NOT gate and 40 ns for the rest of the logic gates. The register has 30 ns setup time, 30 ns clk-to-q delay and 10 ns hold time. Please calculate the maximum clock frequency for this circuit.



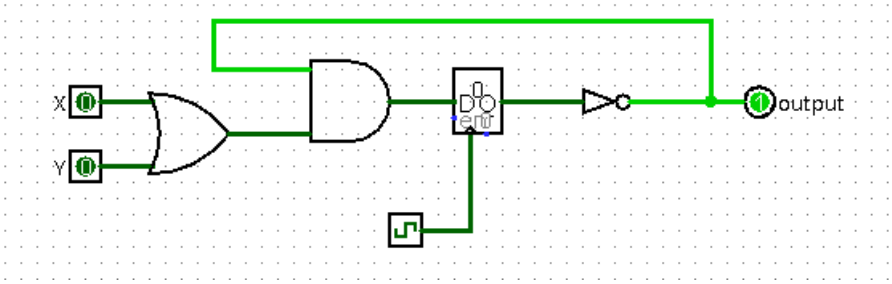
(a) **5GHz**

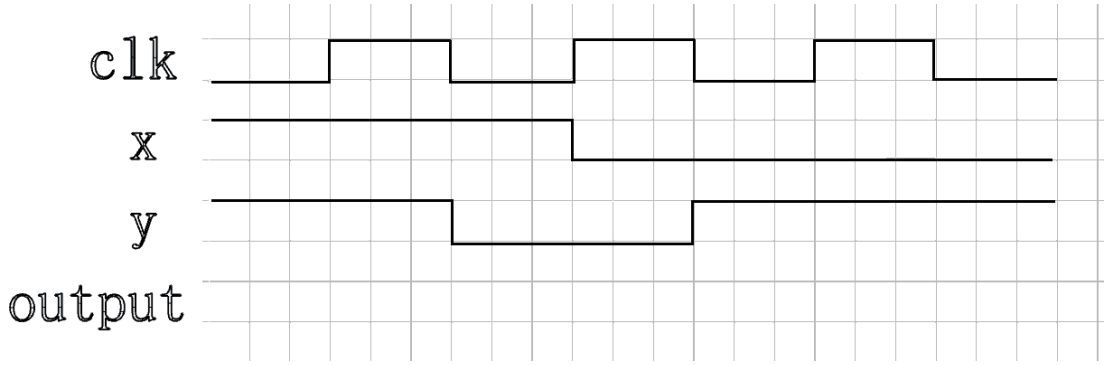
[4] (b) Write the C expression of the following circuit (e.g. $output = (X\&Y)$).

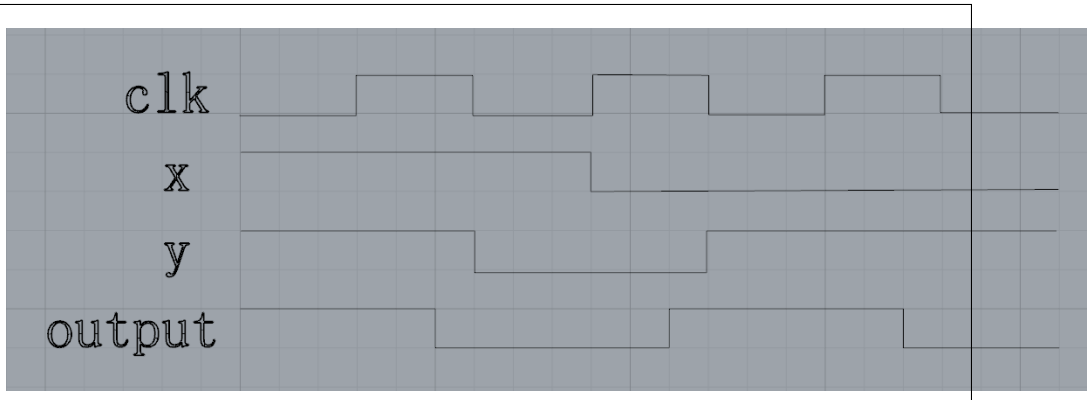


(b) $(X\&!Y||!X\&Y)$ **or** $X\hat{}Y$ **(xor)**

[6] (c) Draw the Timing Diagram for the circuit below. The delay for any logic is 10 ns, the setup time and hold time can be ignored. The clock-to-q delay for a register is also 10 ns. Each clock cycle is 30 ns (updated to 60 ns on the whiteboard), each grid in the following diagram is a unit of 10 ns (output is initially high).







Solution:

Things only happen after the rising clock edge plus clock to Q delay plus the NOT circuit.

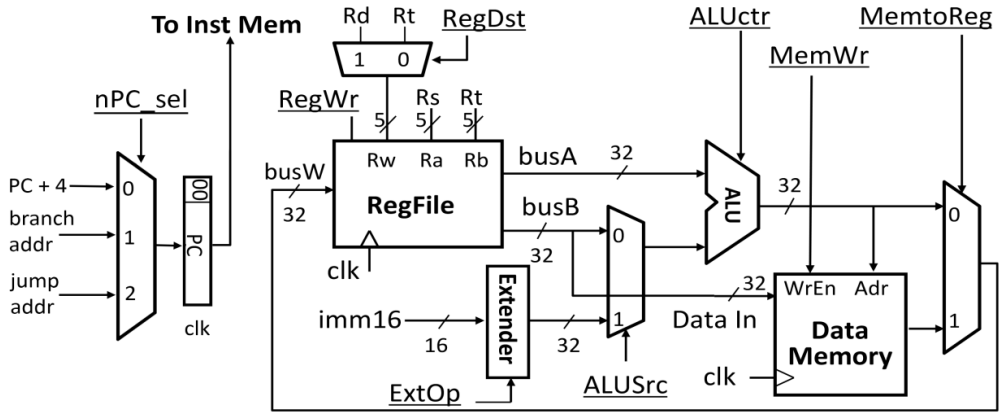
6. **MIPS CPU & Performance Analysis** (15 minutes)

Consider the following new instruction: *jals $rt $rs imm*. The instruction stores *PC + 4* in register *$rt*. At the same time, it sets the *PC* to the value in register *$rs* offset by the sign-extended *imm* value.

3 (a) Write the register transfer language (RTL) corresponding to *jals*

```
R[rt] <- PC + 4;
PC <- R[rs] + SignExtImm
```

5 (b) Change as little as possible in the 1-stage datapath below to support **jals**. In case of ties, pick the set of changes that maximizes the number of control signals that can be set to "don't care". Draw your changes directly in the diagram **or** describe your changes below. You may only add multiplexers, wires, splitters, tunnels, adders, and add or modify control signals.



Describe your changes below:

Solution: 1. Add a mux to busW and add a new control signal "RegSrc" to control the mux. Connect the existing busW input and PC + 4 to the new mux. 2. Connect

the ALU output to the mux controlled by nPC_sel under port 3 (nPC_sel itself doesn't need to be changed since it's already a 2-bit signal)

4 (c) We now want to set the control lines appropriately. List what each signal should be, either by an intuitive name or 0,1,"don't care", etc.. Include any new control signals you added.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	RegSrc		
0	1	3	Sign	Imm(1)	Add	0	x	ALUout		

(d) *The following questions are base on the CPU we learned in class. First let's recall clocking methodology.*

- The input signal to each state element must stabilize before each rising edge.
- Critical path: Longest delay path between state elements in the circuit.
- $t_{clk} \geqslant t_{clk-to-q} + t_{CL} + t_{setup}$, where t_{CL} is the critical path in the combinational logic.
- If we place registers in the critical path, we can shorten the period by reducing the amount of logic between registers.

The delays of circuit elements 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	250	200	150	20

1 (e) What instruction exercises the critical path?

(e) **Load Word (lw)**

3 (f) What are the minimum clock cycle, t_{clk}? Provide the formula/ details for partial credit.

Solution: $t_{clk} \geqslant t_{clk-to-q} + t_{IMEMread} + t_{RFread} + t_{ALU} + t_{DMEMread} + 2 \cdot t_{mux} + t_{RFsetup} = 950ps$
1. MUX for ALUSrc, 2. MUX for MemtoReg

(f)

2 (g) Why is a single cycle CPU inefficient?

Solution: Not all instructions exercise the critical path.
It is not parallelized. Each component can be active concurrently.
Slow.

(g) _____

2 (h) How can you improve its performance?

Solution: Pipeling: Put pipeline registers between two datapath stages. $->$ reduce the clock time.

(h) _____

7. **Parallel Programming** (15 minutes)

Read the following C code for matrix addition.

```
void matadd (float * A, float * B, float * C, int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            A[j*n+i] = B[j*n+i] + C[j*n+i];
        }
    }
}
```

Assume no compiler optimizations are applied and big values for `n`. We can easily see that this piece of code is not optimal. As a computer science student, it is your job to optimize this code to improve efficiency. Answer the following questions:

2 (a) There is a very obvious error in one line that is slowing down the program a lot. What is this error? Explain why it is slowing down the program so much.

(a) **The index of the array is stepping by n instead of 1. So we get lots of cache misses.**

4 (b) Restructure the code using loop unrolling. Also correct the error you identified in the above task. Use 4 statements per iteration. You may assume that n is dividable by 4.

```
void matadd (float * A, float * B, float * C, int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j+=4){
            int indx = j+n*i;
            A[indx] = B[indx] + C[indx]; indx++;
            A[indx] = B[indx] + C[indx]; indx++;
            A[indx] = B[indx] + C[indx]; indx++;
            A[indx] = B[indx] + C[indx];
        }
    }
}
```

4 (c) Optimize this code further with OpenMP (no explicit loop unrolling in this task!).

```
void matadd (float * A, float * B, float * C, int n){
    #pragma omp for
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n;){
            A[j+n*i] = B[j+n*i] + C[j+n*i];
        }
    }
}
```

Don't put the pragma on the inner loop (false sharing: threads trash each other's cache lines!)

5 (d) Use SSE intrinsics to utilize SIMD instructions on modern processors. (Do not use OpenMP here.) Hint: You might need the following intrinsics:

```
__m128 _mm_add_ps (__m128 a, __m128 b)
__m128 _mm_load_ps (float const* mem_addr)
void _mm_store_ps (float* mem_addr, __m128 a)

void matadd (float * A, float * B, float * C, int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n;){
            __m128 tmp1 =  _mm_load_ps(B+j+n*i);
            __m128 tmp2 =  _mm_load_ps(C+j+n*i);
            tmp1 = _mm_add_ps(tmp1, tmp2);
            _mm_store_ps(A+j+n*i, tmp1);
        }
    }
}
```

2 (e) Amdal's Law: Suppose that we've developed a new method that can run 3 times faster in 60% of the instructions. What is the overall speedup?

Solution: S = 1/(0.4+0.6/3) = 5/3

2 (f) Amdal's Law: Suppose that we can run n times faster in 60% of the instructions. What is the maximum overall speedup that we can achieve?

Solution: S = 1/0.4 = 2.5

8. **Memory Access** (15 minutes)

Consider a 32-bit physical memory space and a 32 KiB 4-way associative cache with LRU replacement. You are told that the hit rate of loop two is 9/16.

```
int ARRAY_SIZE = 32 * 1024;
int arr[ARRAY_SIZE]; // *arr is aligned to a cache block

/* loop one */
for (int i = 0; i < ARRAY_SIZE; i += 4) arr[i] = i;
/* loop two */
for (int i = ARRAY_SIZE - 4; i >= 0; i -= 4) arr[i+1] = arr[i]-1;
```

4 (a) Fill the number of bits in the tag, index and offset fields in the figure below.

Tag	Index	Offset

Solution: 19,8,5

(b) **It's Not My Fault**

Consider the following OpenMP snippet:

```
int values[size];
#pragma omp parallel
{
     int i = omp_get_thread_num();
     int n = omp_get_num_threads();
     for(int j = i * (size / n); j < (i + 1) * (size / n); j++){
          values[j] = j;
     }
}
```

All cores share the same physical memory and we are running 2 threads. This is the sole process running. Each page is 1 KiB, and you have 2 pages of physical memory. The code snippet above starts at virtual address 0x400, and the values array starts at 0x800. The size, n, i, and j variables are all stored in registers. The functions omp_get_thread_num and omp_get_num_threads are stored in virtual addresses 0x440 to 0x480. The replacement policy for the page table is Least Recently Used.

At the start of the pragma omp parallel call the page table looks as follows:

Virtual Page Number	Valid	Dirty	Physical Page Number
0	0	0	0
1	1	0	0
2	1	1	1
3	0	0	1

3 (c) How many page faults will occur if size=0x080?

(c) **0**

Solution: Traversing 0x200 (4 bytes per int!) of memory (addresses 0x800 to 0xa00), which all sit in virtual page 2, which is already Valid according to the table. Instruction accesses are also all in a Valid page.

5 (d) What is the minimum number of page faults that will occur if size=0x200? And what is

the maximum?

(d) **1 ; 512**

Solution: In the (very unlikely) case that first the complete first thread is running and then the complete 2nd thread is running we will have exactly 1 page fault as we switch from page table 2 to 3.

The array has 512 entries, so each thread is working on 256 entries, which use exactly 1K memory. The array starts exactly at page table 2, so the first thread is only working on table 2 and the 2nd thread only working on table 3. Every instruction is also using a page table (instruction load), but the instruction table is then always the most recently used so it is not the one that is being replaced but the page table to the data.

In the other extreme the 2nd thread runs first, has a page fault, then the 1st thread runs, has a page fault, and so on, 512 times (0x200 means the array has 512 entries).

3 (e) How could you reduce the maximum page faults for part (b) (updated to (d))?
Choose the valid options amongst the following (- 1 pt for every incorrect answer):
1. increase virtual address space
2. decrease number of threads
3. increase number of threads
4. use SIMD instructions
5. change page table replacement policy to Random
6. increase physical address space
7. add a 4-entry TLB

(e) **2,4,6**