_____ **UC Berkeley CS61C** _____

Your Name (first last) **Fall 2018 Final Exam** *SID*

_____ _____ _____

← Name of person on left (or aisle) *TA name* *Name of person on right (or aisle) →*

Fill in the correct circles & squares completely…like this: ● (select ONE), and ■(select ALL that apply)

The questions that strictly clobber the Midterm are the first three, labeled “M1” through “M3”.
The remaining questions cover the material after the Midterm, “F1” through “F4”.

Question	M1	M2 (2 pages)	M3	Ms	F1	F2	F3 (2 pages)	F4	Fs	Total
Minutes	20	20	20	**60**	30	30	30	30	**120**	**180**
Points	8	8	8	**24**	20	20	20	21	**81**	**105**

M1) *Cache, money y'all…* (8 points, ½ points each, 20 minutes)

A mystery, byte-addressed cache has Tag:Index:Offset (T:I:O) = 9:4:3. For the computer,

a) What is the *virtual address space*? (select ONE) ◯8-bit ◯16-bit ◯32-bit ◯64-bit ◯Not enough info!
b) What is the *physical address space*? (select ONE) ◯8-bit ◯16-bit ◯32-bit ◯64-bit ◯Not enough info!

Different caches can have the same T:I:O! Let's explore that in parts (b) and (c) below.

c) How could we ***maximize*** cache size, while preserving T:I:O=9:4:3? (select ONE per row)

Associativity	◯ 2-way set	◯ Direct Mapped	◯ 8-way set	◯ 4-way set
Block Size	◯ 4 bytes	◯ 8 bytes	◯ 12 bytes	◯ 3 bytes
# of Blocks	◯ 8 blocks	◯ 16 blocks	◯ 4 blocks	◯ 128 blocks

d) How could we ***minimize*** cache size, while preserving T:I:O=9:4:3? (select ONE per row)

Associativity	◯ 2-way set	◯ Direct Mapped	◯ 8-way set	◯ 4-way set
Block Size	◯ 4 bytes	◯ 8 bytes	◯ 12 bytes	◯ 3 bytes
# of Blocks	◯ 8 blocks	◯ 16 blocks	◯ 32 blocks	◯ 64 blocks

e) Now we're working with a write-back, 1024B direct-mapped cache that has 8B blocks. We're interested in seeing if we can lower our AMAT if our memory access pattern is iterating through an array with a fixed stride. The majority of our misses are conflict misses, and we have an inconsequential amount of compulsory and capacity misses. For each of the following modifications, mark how it changes each component of AMAT (Hit Time, Miss Penalty, Miss Rate) and the overall Hardware Complexity.

Modification	Hit Time	Miss Penalty	Miss Rate	Hardware Complexity
Change block size from 8B to 16B, *but keep the cache size the same*	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect
Change to 2-way Associativity (same cache & block size)	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect	◯ Increase ◯ Decrease ◯ No effect

M2) *Floating down the C...* [this is a 2-page question] (8 points = 1,1,2,1,1,1,1, 20 minutes) SID_____

Consider an 8-bit "minifloat" S**E**EEMMMM (1 sign bit, 3 exponent bits, 4 mantissa bits). All other properties of IEEE754 apply (bias, denormalized numbers, ∞, NaNs, etc). The bias is -3.

a) How many minifloats are there in the range [1, 4)? (i.e., 1 ≤ **f** < 4) _____

b) What is the number line distance between 1 and the smallest minifloat bigger than 1? _____

c) Write **times2** in one line using integer operations. Assume the leftmost "E" bit of **f** (bolded above) is 0.
minifloat times2(minifloat f) { return f * 2.0; }

```
times2: ______ a0, a0, ______ ## Assume f is in the lowest byte of the register

        ret
```

M2) *Floating down the C…, continued...* (8 points = 1,1,2,1,1,1,1,1, 20 minutes)

Consider the following code (and truncated ASCII table; as an example of how to read it, "G" is `0b0100 0111`):

```
uint8_t mystery (uint8_t *A) {
   return *A ? (*A & mystery(A+1)) : 0xFF;
}
```

d) Where is **A** stored? (*not* what it points to, ***A**)
◯code ◯static ◯heap ◯stack

e) What is **(char)mystery("GROW")**? _____

f) A two-character string is passed into **mystery** that makes it return the **uint8_t** value **0** (not the character "**0**"). The first character is "M", the second character is a number from 1-9. Which?
◯1 ◯2 ◯3
◯4 ◯5 ◯6
◯7 ◯8 ◯9

$b_7 b_6 b_5$ → / $b_4 b_3 b_2 b_1$ ↓	000	001	010	011	100	101	110	111
0000	0	0	@	P	`	p		
0001	1	A	Q	a	q			
0010	2	B	R	b	r			
0011	3	C	S	c	s			
0100	4	D	T	d	t			
0101	5	E	U	e	u			
0110	6	F	V	f	v			
0111	7	G	W	g	w			
1000	8	H	X	h	x			
1001	9	I	Y	i	y			
1010	:	J	Z	j	z			
1011	;	K	[	k	{			
1100	<	L	\	l	|			
1101	=	M	]	m	}			
1110	>	N	^	n	~			
1111	?	O	_	o				

---

g) Incrementing or decrementing a variable is common: e.g., **sum += amount**, so we decide to make a new RISC-V instruction (based on I-format instructions), and reuse the unused bits from **rs1** to give to the immediate (**rd** will be read *and* written, don't touch `funct3` or `opcode`). Using *only* that operation, what's the most **amount** that **sum** could now increase by (approx)? (select ONE for each column)

◯1 ◯2 ◯4 ◯8 ◯16 ◯32 ◯64 ◯128 ◯256 ◯512	◯<blank> ◯kilo ◯mega ◯giga ◯tera ◯peta ◯exa ◯kibi ◯mebi ◯gibi ◯tebi ◯pebi ◯exbi

M3) *Just one more thing…RISC-V self-modifying code!* (8 points, 20 minutes) SID_____

(this is meant to be a fairly hard problem, we recommend saving it until the end of the exam…)

Your Phase I date was too late, so you can't get into the course you want. You need to hack CalCentral's server to enroll yourself! You find the following program running on the CalCentral server:

```
.data ### Starts at 0x100, strings are packed tight (not word-aligned)
    benign: .asciiz "\dev/null"
    evil:   .asciiz "/bin/sh"

.text ### Starts at 0x0
      addi t0 x0 0x100       ### Load the address of the string "\dev/null"
      addi t2 x0 '/'         ### Load the correct character. The ASCII of '/' is 47_10.
      jal ra, change_reg
      sb t2 0(t0)            ### Fix the backslash "\dev/null" → "/dev/null"
      addi a0 x0 0x100
      jal ra, os
```

The subroutine `change_reg` allows a user to arbitrarily set the value of any registers they choose when the function is executed (similar to the debugger on Venus). **`os(char *a0)`** runs the command at **`a0`**. *Select as few registers as necessary, set to particular values* to **MAKE THE RISC-V CODE MODIFY ITSELF** so the **`os`** function runs "**`/bin/sh`**" to hack into the CalCentral database. **Please note: even though `change_reg` can arbitrarily change any register it STILL follows the RISC-V calling convention. You CANNOT assume that the registers are initialized to zero on the launch of the program. Also, the assembler is NOT optimized.** Hint: Think about *where* the change needs to happen, then *what* it should be.

Reg	Value to set it to (in HEX without leading zeros)
▢ a0	0x
▢ a1	0x
▢ a2	0x
▢ s0	0x
▢ s1	0x
▢ s2	0x
▢ t0	0x
▢ t1	0x
▢ t2	0x
▢	Not Possible

F1) *VM...* (20 points = 12+4+4, 30 minutes)

a) What are the steps that occur for a memory read (when a **page fault** is encountered)? You may assume there's room in memory for a new page, and we're using LRU replacement. Assume there's no data cache. Mark the order of the required actions, there's at most one choice per #, and e*very row/col should have a #.*

Below, ◯⇒ Select ONE, ▢⇒ Select ALL that apply	1	2	3	4	5	6	7	8	9	10
Request data using the (◯**physical** ◯**virtual**) address	●	○	○	○	○	○	○	○	○	○
Access Page Table with ◯**PPN** ◯**VPN** ◯**Offset**	○	○	○	○	○	○	○	○	○	○
Access TLB with ◯**PPN** ◯**VPN** ◯**Offset**	○	○	○	○	○	○	○	○	○	○
Adjust LRU bits in ▢**TLB** ▢**Page Table** ▢**Memory**	○	○	○	○	○	○	○	○	○	○
Split physical address into (◯**PPN+Offset** ◯**VPN+Offset** ◯**PPN+VPN**) fields	○	○	○	○	○	○	○	○	○	○
Split virtual address into (◯**PPN+Offset** ◯**VPN+Offset** ◯**PPN+VPN**) fields	○	○	○	○	○	○	○	○	○	○
Request new page from OS/Memory manager	○	○	○	○	○	○	○	○	○	○
Update Page Table with new ▢**PPN** ▢**VPN** ▢**Offset**	○	○	○	○	○	○	○	○	○	○
Update TLB with new ▢**PPN** ▢**VPN** ▢**Offset**	○	○	○	○	○	○	○	○	○	○
Return the data *(this is the last thing we do)*	○	○	○	○	○	○	○	○	○	●

b) Mark the following questions as either **True** or **False**:

◯**True** ◯**False**	If we have a TLB which contains a number of entries equal to `MEMORY_SIZE` / `PAGE_SIZE`, every TLB miss will also be a page fault.
◯**True** ◯**False**	If we change our TLB to direct-mapped, we're likely to see fewer TLB misses.
◯**True** ◯**False**	Every TLB miss is equally expensive in terms of the amount of time it takes for us to resolve our virtual address to a physical address.
◯**True** ◯**False**	A virtual address will always resolve to the same physical address.

c) Consider a VM system on a RISC-V 32-bit machine with 2^{20} page table rows, no TLB, and limited physical RAM, choose ONE of the following code snippets that **would always have the most page faults per memory access** by touching elements of a page-aligned **`uint8_t`** array `A`, and choose ONE value of `STRIDE` (choose the minimum possible value that accomplishes it). Both `A_SIZE` and `STRIDE` are powers of 2, and `A_SIZE > STRIDE`. `random(N)` returns a random integer from 0 to N.

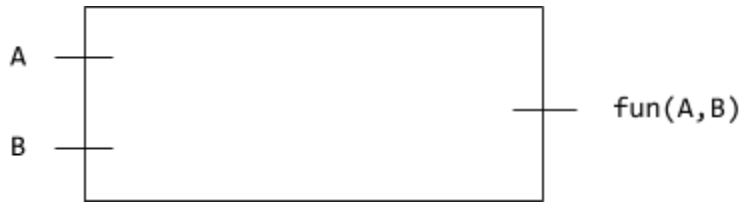
```
◯ for (i = 0;      i < STRIDE; i++)       { A[random(A_SIZE)] = random(255); }
◯ for (i = 0;      i < A_SIZE; i++)       { A[random(STRIDE)] = random(255); }
◯ for (i = 0;      i < A_SIZE; i++)       { A[i] = random(STRIDE); }
◯ for (i = 0;      i < STRIDE; i++)       { A[i] = random(255); }
◯ for (i = 0;      i < A_SIZE; i+=STRIDE) { A[i] = random(255); }
◯ for (i = STRIDE; i < A_SIZE; i++)       { A[i] = A[i-STRIDE];  A[i-STRIDE] = A[i]; }
```

STRIDE: ◯2^0 ◯2^1 ◯2^2 ◯2^3 ◯2^4 ◯2^5 ◯2^6 ◯2^7 ◯2^8 ◯2^9 ◯2^{10} ◯2^{11} ◯2^{12}
◯2^{13} ◯2^{14} ◯2^{15} ◯2^{16} ◯2^{17} ◯2^{18} ◯2^{19} ◯2^{20} ◯2^{21} ◯2^{22} ◯2^{23} ◯2^{24}
◯2^{25} ◯2^{26} ◯2^{27} ◯2^{28} ◯2^{29} ◯2^{30} ◯2^{31} ◯2^{32}

F2) *SDS* (20 points = 8+5+7, 30 minutes) SID_____

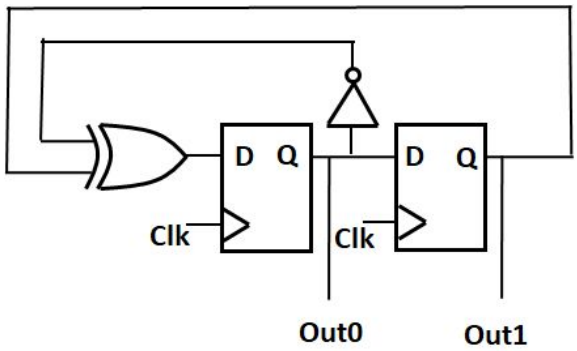
a) Transform the **fun** function below into the *fewest Boolean gates* that implement the same logic. You may use AND, OR, XOR and NOT gates. *Hint: start with the truth table.*

```
bool fun(bool A, bool B) { return (A == B) ? true : B; }
```



---

b) The logic implementation of a state machine is shown in the figure below. How many states does this state machine have? (Assume that it always starts from Out0=0, Out1=0)



Number of states = ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9

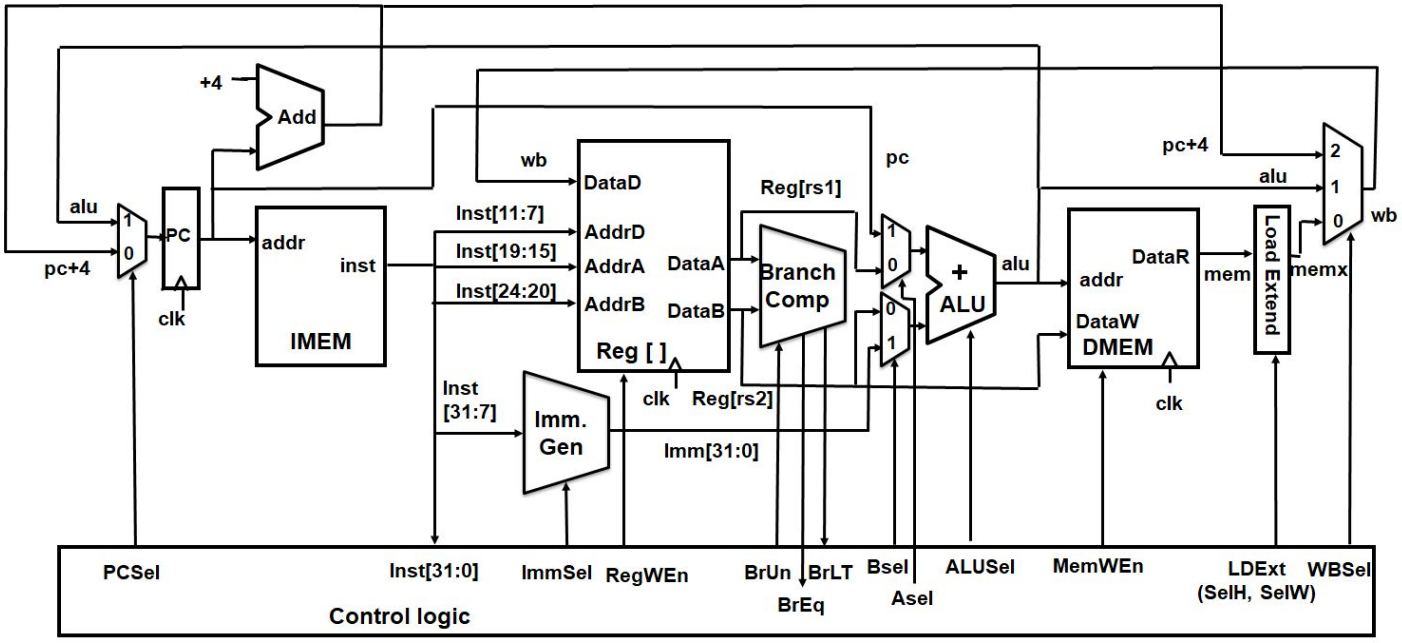
---

c) In the figure above, flip-flop clk-to-q delay is **40ps**, setup time is **30ps** and hold time is **30ps**. XOR delay is **20ps** and the inverter delay is **10ps**. What is the *maximum frequency* (F_{max}) of operation?

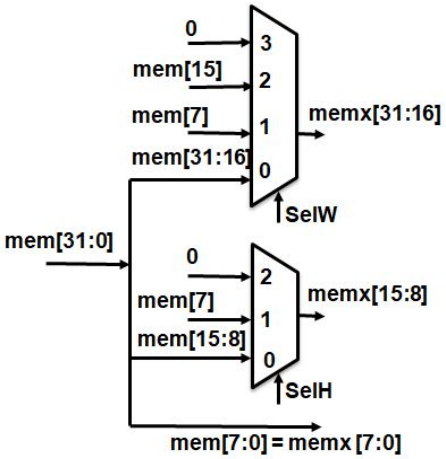
F_{max} = _____GHz .

F3) *Datapathology* [this is a 2-page question] (20 points = 4+10+6, 30 minutes)

The datapath below implements the RV32I instruction set. We'd like to implement sign extension for loaded data, but our loaded data can come in different sizes (recall: **`lb`**, **`lh`**, **`lw`**) and different intended signs (**`lbu`** vs. **`lb`** and **`lhu`** vs **`lh`**). Each load instruction will retrieve the data from the memory and "right-aligns" the LSB of the byte or the half-word with the LSB of the word to form **`mem[31:0]`**.



a) To correctly load the data into the registers, we've created two control signals **SelH** and **SelW** that perform sign extension of **`mem[31:0]`** to **`memx[31:0]`** (see below). **SelH** controls the half-word sign extension, while **SelW** controls sign extension in the two most significant bytes. What are the Boolean logic expressions for the four (0, 1, 2, 3) **SelW** cases in terms of Inst[14:12] bits to handle these five instructions (**`lb`**, **`lh`**, **`lw`**, **`lbu`** and **`lhu`**)? **SelH** has been done for you. In writing your answers, use the shorthands "I14" for Inst[14], "I13" for Inst[13] and "I12" for Inst[12]. You don't have to reduce the Boolean expressions to simplest form. (Hint: green card!)



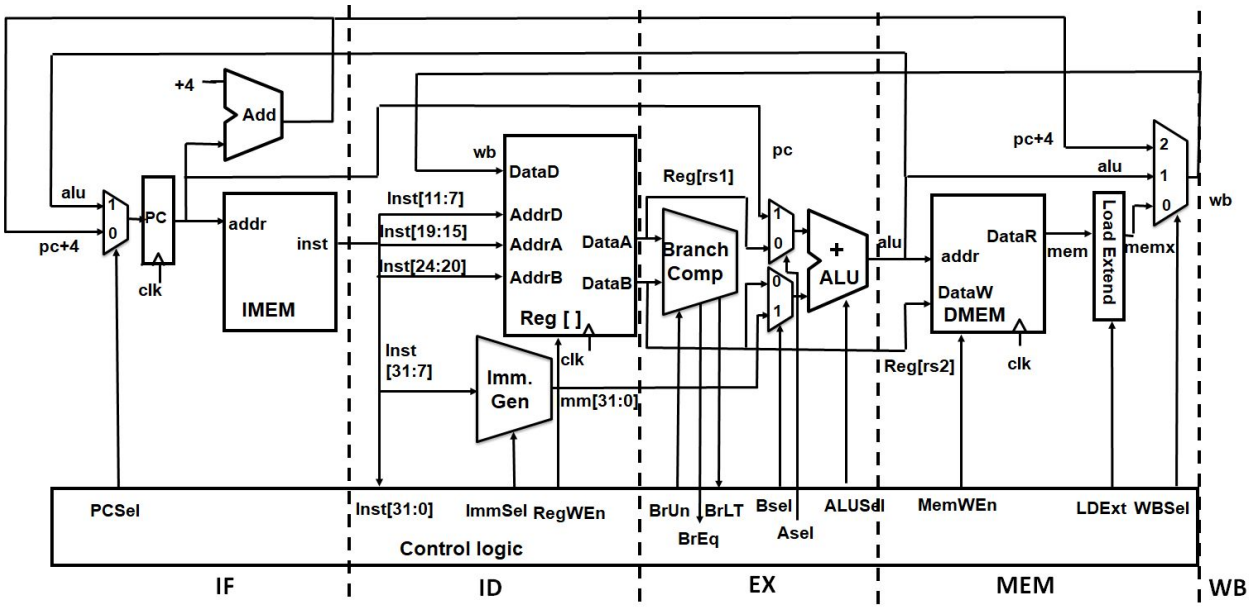
SelW=3	
SelW=2	
SelW=1	
SelW=0	

SelH=2	I14 • ~I13 • ~I12
SelH=1	~I14 • ~I13 • ~I12
SelH=0	I13 + I12

(Single-bit values **`mem[7]`** and **`mem[15]`** are wired to 8 or 16 outputs)

F3) *Datapathology, continued* (20 points = 4+10+6, 30 minutes) SID_____

(this is the same diagram as on the previous page, with five stages of execution annotated)



b) In the RISC-V datapath above, mark what is used by a `jal` instruction. *(See green card for its effect...)*

Select one per row	**PCSel Mux:**	◯ "pc + 4" branch	◯ "alu" branch	◯ * (don't care)	
	ASel Mux:	◯ "pc" branch	◯ Reg[rs1] branch	◯ * (don't care)	
	BSel Mux:	◯ "imm" branch	◯ Reg[rs2] branch	◯ * (don't care)	
	WBSel Mux:	◯ "pc + 4" branch	◯ "alu" branch	◯"mem" branch	◯ * (don't care)
Select all that apply	**Datapath units:**	▢ Branch Comp	▢ Imm. Gen	▢ Load Extend	
	RegFile:	▢ Read Reg[rs1]	▢ Read Reg[rs2]	▢ Write Reg[rd]	

c) If we convert the above datapath to a 5-stage pipeline with **no forwarding**, what types of hazards (S=structural, D=data, C=control) exist **after** each line in the following code; how many `nops` must we add? (Assume a register can be written and read in the same cycle, and that the Branch Comp is in the EX stage.)

```
start: lw    t0, 0(a0)      Hazard (circle one): S D C None   # of nop _____
       beq   t0, 0, end     Hazard (circle one): S D C None   # of nop _____
       addi  t0, t0, 2      Hazard (circle one): S D C None   # of nop _____
       sw    t0, 0(a0)      Hazard (circle one): S D C None   # of nop _____
end:
```

F4) *What's that smell?! Oh, it's Potpourri...* (20 points=2 each, 30 minutes)

a) We build a small Internet-of-things device to measure dog body temperature and send it to a receiver. It will only send the following temperatures: {100.0, 100.1, 100.2, 100.4, 100.8, 101.6, 103.2, 106.4}, and any time the temperature is not those exact values, it'll send whatever value is the closest one. What encoding/decoding *scheme* would you use for these numbers and how many total *bits* would you need?
Scheme: ◯Unsigned fixed point ◯Bias fixed point ◯2s complement fixed point ◯Other
Bits: ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16

b) ◯True ◯False A `0/0` ALU operation would cause an *interrupt*, dealt with by the trap handler.

c) ◯True ◯False *DMA (Direct Memory Access)* is a form of Programmed I/O the CPU handles.

d) ◯True ◯False A *shared-based network* is another kind of parallelism; multiple nodes can talk to each other at the same time, "sharing" the network.

e) ◯First-fit ◯Next-fit ◯Best-fit would make sense for *allocating blocks on a Flash (SSD) drive*.

f) ◯Control ◯Datapath ◯Memory ◯Input ◯Output causes the most headaches with multi-core.

g) ◯True ◯False Introducing *locks* in C (e.g., code below) cures the race condition bug with threads.

```
while (lock != 0) ; // spin-wait until the variable lock is released
// lock == 0 now (unlocked)
lock = 1; // set lock (locked)
// access shared resource ...
lock = 0; // release lock (unlocked)
```

h) The code below was written to sum the numbers from **1** to **N** (always a positive number). Select ONE.

◯ It works, but it has to be run on a machine with 16 *physical* cores
◯ It works, but it has to be run on a machine with 16 *logical* cores
◯ It works, but only when **N** is bigger than the number of *physical* cores
◯ It works, but only when **N** is bigger than the number of *logical* cores
◯ It has a race condition bug
◯ It has a deadlock bug
◯ It always works
	`int sumup(int N) {`
	`  int THREADS = 16, TOTAL = 0, sum[THREADS];`
	`  omp_set_THREADS(THREADS);`
	`  for (int i=0;i<THREADS;i++) sum[i]=0;`
	`  #pragma omp parallel`
	`  {`
	`  int id = omp_get_thread_num();`
	`  for (int i=id;i<=N;i+=THREADS) sum[id] += i;`
	`  TOTAL+=sum[id];`
	`  }`
	`  return TOTAL;}`

i) This: `main() { for(uint8_t i = 9; i >= 0; --i) printf("%u", i); }` causes… (select ONE)
◯ A compile error because the **`printf`** statement needs to be on a different line
◯ A compile error because the **`printf`** statement needs to be surrounded by curly brackets **`{ }`**
◯ An infinite loop
◯ Nothing to print out because there's no trailing **`\n`** (so the output doesn't get flushed)
◯ The numbers to print out like this: **`987654321`**
◯ The numbers to print out like this: **`876543210`**
◯ The numbers to print out like this: **`9876543210`**