

This exam consists of 8 substantive questions. You have 3 hours – budget your time wisely. Assume the armlab/gcc217 environment unless otherwise stated in a problem.

Do all of your work on these pages. You may use the provided blank spaces for scratch space, however this exam is preprocessed by computer, so for your final answers to be scored you must write them inside the designated spaces and fill in selected circles and boxes completely (\bigcirc and \bigcirc , not \checkmark or \checkmark). Please make text answers dark and neat.

Name: NetID: Precept: P01 - MW 1:30 P04 - TTh 12:30 P08 TTh 3:30 Xiaoyan Li Tolulope Oshinowo Yang Duan P02 - MW 3:30 P06 - TTh 1:30 P09 TTh 7:30 Xiaoyan Li Indu Panigrahi Andrew Sheinberg P03 - TTh 12:30 P07 - TTh 1:30 Lana Glisic Ryan Oet

This is a closed-book, closed-note exam, except you are allowed one two-sided study sheet. Please place items that you will not need out of view in your bag or under your working space at this time. Electronic devices such as cell phones, laptops, smartwatches except to check the time, etc. may not be used during this exam.

This examination is administered under the Princeton University Honor Code. Students should sit one seat apart from each other and refrain from talking to other students during the exam. All suspected violations of the Honor Code must be reported to honor@princeton.edu.

In the box below, copy **and** sign the Honor Code pledge before turning in your exam: *"I pledge my honor that I have not violated the Honor Code during this examination."*

Х

Please don't make the course staff's life harder: make sure you have filled out your name, NetID (i.e., armlab login – not PUID, not email alias), precept and the Honor Code pledge text on the front page. Sign your name once you have finished the exam.

Question 1: A 2nd chance at midterm topics? Accepto. 5 points

For each of the code snippets below, indicate whether the equality operation always evaluates to true (1), always evaluates to false (\emptyset), or depends on the system because the behavior is not guaranteed by the C90 standard.

		TRUE	FALSE	DEPENDS
a.	sizeof(signed long) == sizeof(unsigned long)	\bigcirc	\bigcirc	\bigcirc
b.	sizeof(size_t) == sizeof(unsigned long)	\bigcirc	\bigcirc	\bigcirc
C.	sizeof(sizeof(char)) == sizeof(char)	\bigcirc	\bigcirc	\bigcirc
d.	char ac[] = "muratarum"; ac[9] == (ac[8] - *ac)	\bigcirc	\bigcirc	\bigcirc
e.	int ai[] = {0, 4}; int *pi = &ai[1]; (pi - ai) == 4;	\bigcirc	\bigcirc	\bigcirc

Question 2: Four ballots per day stages per build 5 points

Which stage of the build process (**P**reprocessor, **C**ompiler, **A**ssembler, or **L**inker) is responsible for each of the following operations:

		Р	С	A	L
a.	Combines object files' contents	\bigcirc	\bigcirc	\bigcirc	\bigcirc
b.	Resolves references to external library implementations	\bigcirc	\bigcirc	\bigcirc	\bigcirc
C.	Optimizes to use callee-saved registers	\bigcirc	\bigcirc	\bigcirc	\bigcirc
d.	Handles macro replacement, e.g., E0F (aka substitution or expansion)	\bigcirc	\bigcirc	\bigcirc	\bigcirc
e.	Checks for unterminated compound statements (aka blocks)	\bigcirc	\bigcirc	\bigcirc	\bigcirc

Consider the following program:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
char fmt[] = "%lu\n";
size_t fun(char *x) {
   static size_t i;
   return ++i;
}
int main(int argc, char **argv) {
  char *pc;
   size_t ulResult = 0;
   if(argc < 2)
      return EXIT_FAILURE;
  pc = argv[1];
  while(*pc)
      ulResult = fun(pc++);
  printf(fmt, ulResult);
   return 0;
}
```

a. Fill in the memory section and size on armlab for each variable:



b. Replace the loop in main with a single statement that calls a function from the C string library, such that the same result would be printed in the subsequent printf: Consider the following C function, whose identifier naming leaves much to be desired:

```
struct Node {
    int payload;
    struct Node *next;
};
struct Node *mystery(struct Node *arg) {
    struct Node *var;
    if(!arg) return NULL;
    var = mystery(arg->next);
    if(!var) return arg;
    arg->next->next = arg;
    arg->next = NULL;
    return var;
}
```

a. Determine what the function does, then for each specified identifier give a more semantically meaningful name to replace its current vague identifier:



b. Describe three cases that would be good function boundary tests for mystery, in less than 10 words each.



Question 5: Leo XIV has none of the long name issues 7 points

\$./a.out What is your name? Christopher Moretti Thank you, Christopher Mor tti. The answer to life. the universe. ??? (!)

(Note: this is just the number that's actually printed when you run the code. It's not an attempt to Easter egg a phone number or anything like that. Please don't try to call it. Doing so almost certainly won't give you the answer to life, the universe, and everything.)

The answer to life, the universe, and everything is 6911092

Lecture 22 gave an example of a buffer overrun, in which an int variable was corrupted with character input that overran the buffer. The slide above shows the result, where printing the overwritten variable results in four consecutive chars stored in memory – the characters: 't', 't', 'i', '0' – being interpreted as the int value 0×00697474 .

a. This result relies on many non-portable factors, e.g., the stack memory layout of the function and that the size of four chars is also the precise size of an int. List two more armlab properties that are not guaranteed by the C standard that caused the bytes {'t', 't', 'i', '\0'} to be interpreted as 0x00697474.

b. I asked ChatGPT to convert 0×00697474 to decimal and it got the wrong answer (6900092, instead of the number on the slide). Prove yourself more reliable than ChatGPT by converting the <u>sum</u> of the hex addition $0 \times C05 + 0 \times 217$ to decimal:



You may refer to this abbreviated ARM assembly language reference for Q6 – Q8.

Instruction(s)	Description			
{add,sub,lsl,sdiv} dst, src1, src2	dst = src1 {+, -, <<, / } src2			
{beq,bne} label	Go to labe1 if comparison was {"equal", "not equal"}			
{b,bl} label	{Unconditionally go to , Call function at} labe1			
cmp first, second	Compare first with second, setting bits in PSTATE			
ldr dst, [src]	Load 4 or 8 bytes pointed to by src into dst			
ldrb dst, [src]	Load 1 byte pointed to by src into dst			
str src, [dst]	Store 4 or 8 bytes in src to memory pointed to by dst			
mov dst, src	Copy contents of register src to register dst			
ret	Return to address pointed to by x30			
R0 - R7 and R0 (w or x)	Used for arguments to and return value from functions			
R0 - R7 and R9 - R15 (w or x)	Caller-saved scratch registers			

Consider the following function, modified from the Euclid program you saw in precept:

1			- I	(/]Temp =]Abe1 %]Abe2
1	.equ LABS2, 8		<u>/</u>	// liemp = lADSI % lADSZ
2	.equ LABS1, 16	2	8	// rem = (dividend -
3	.equ LTEMP, 24	2	9	<pre>// (quotient * divisor))</pre>
4	.equ L2, 32	3	0	ldr x0, [sp, LABS1]
5	.equ L1, 40	3	1	ldr x1, [sp, LABS2]
6		3	2	sdiv x2, x0, x1
7	gcd:	3	3	mul x3, x2, x1
8	sub sp, sp, 48	3	4	sub x4, x0, x3
9	str x0, [sp, L1]	3	5	str x4, [sp, LTEMP]
10	str x1, [sp, L2]	3	6	
11		3	7	// lAbs1 = lAbs2
12	// lAbs1 = labs(l1)	3	8	ldr x0, [sp, LABS2]
13	ldr x0, [sp, L1]	3	9	str x0, [sp, LABS1]
14	bl labs	4	0	
15	str x0, [sp, LABS1]	4	1	// lAbs2 = lTemp
16		4	2	ldr x0, [sp, LTEMP]
17	// lAbs2 = labs(l2)	4	3	str x0, [sp, LABS2]
18	ldr x0, [sp, L2]	4	4	
19	bl labs	4	5	// goto gcdLoop
20	str x0, [sp, LABS2]	4	6	b gcdLoop
21		4	7	
22	gcdLoop:	4	8	loopEnd:
23	// if (lAbs2 == 0) goto loopEnd	4	9	ldr x0. [sp. LABS1]
24	ldr x0. [sp. LABS2]	5	0	add sp. sp. 48
25	cmp x0.0	5	1	ret
26	beg loopEnd	ľ	·	
20				

a. I asked ChatGPT what the next instruction after the ret on line 51 is executed will be, and it answered that this function returns to the next instruction after the b1 gcd in its caller. That's not the case here – there's a bug! Why does this function not return to its caller, and to where does it return instead? Be specific.

 Imagine unconditional branch (b) instructions were no longer a part of ARM assembly language. Replace line 46 with no more than 2 instructions that would result in the same behavior as the existing b gcdLoop instruction.

Question 7: Does ChatGPT need more Scrutatoribus? 14 points

ChatGPT infamously had trouble correctly counting the number of occurrences of a letter in some words, as seen on the image on this page. In a running theme, you can do better – this time, in assembly!

The C function Str_count, shown below on the left, returns the number of instances of its char parameter c within its string parameter pcStr before the first nullbyte.

Translate this function into assembly language using the 14 instructions below – you must use each letter exactly once. We have given you the assembly language function structure in the box to the right of the C code.



<pre>#include <stddef.h></stddef.h></pre>
<pre>#include <assert.h></assert.h></pre>
<pre>size_t Str_count(const char* pcStr, char c) {</pre>
<pre>size_t ulCount = 0;</pre>
assert(pcStr != NULL);
while(*pcStr != '\0') {
if(*pcStr == c)
ulCount++;
pcStr++;
}
return ulCount;
}

.section .text					
.global Str_count					
Str_count:					
sub sp, sp, 16					
str xzr, [sp]					
str x30, [sp,8]					
<pre>// Your instructions go here</pre>					
ldr x30, [sp, 8]					
add sp, sp, 16					
ret					

These are the 14 instructions you will use to fill the body of Str_count. Fill in one letter on each blank below, in the order in which they should appear in the function.

A	Done:	Н	bne NoCount
В	Loop:	Ι	cmp w2, w1
С	NoCount:	J	cmp w2, wzr
D	add x0, x0, 1	Κ	ldr x0, [sp]
E	add x3, x3, 1	L	ldr x3, [sp]
F	b Loop	М	ldrb w2, [x0]
G	beq Done	Ν	str x3, [sp]

🗗 Сору

🕏 Edit

I asked ChatGPT to solve the Fall 2022 final exam "Encore" problem, which required generating the machine code for the instruction adr x1, label2. In that problem, label2's address was 0x217217 and the address of the adr instruction itself was 0x214127. The correct answer was 0x10018781.

Here was the response from ChatGPT:

The adr instruction encodes the difference between the label address and the current PC as a signed 19-bit immediate in multiples of 4 (because ARM instructions are 4 bytes wide).

Step 1: Compute the byte offset:

csharp 0x217217 - 0x214127 = 0x2AF0 = 10992 (decimal)

Step 2: Divide by 4 to get the immediate field:

yaml D Copy 2 Edit

This value (2748) is placed in bits [23:5] of the instruction.

Step 3: Register x1 is encoded in bits [4:0] as 00001₂ (or just 1 decimal).

Step 4: The top 8 bits of an adr instruction are always 0b00010000 \rightarrow 0x10.

Putting it all together:

- Top bits: 0x10 << 24 = 0x1000000
- Immediate field: 2748 << 5 = 0x35A00
- Destination register: x1 = 0x1

Final instruction:

Your questions and a reference sheet for adr appear on the next page.

Page 8 of 10

a. ChatGPT made major errors in (1) the initial paragraph, (2) Step 1, (3) Step 4 bold header, and (4) a bullet point in Step 4. Find and circle each of these four errors in the ChatGPT response on the previous page – be precise as to which portion constitutes the error.

For (3), the error is **not** the use of 0b... as a shorthand for a binary number. This notation isn't valid C90 but is valid in some languages and eminently reasonable.

Note that each error is identified only once, even if it cascades through the answer. For example, if ChatGPT claimed 1 + 1 = 3, you would circle that claim, not subsequent uses of the erroneous 3 later in the calculation.

Hint: there's also a minor error in Step 2: dividing by 4 gets the immediate hi field

b. One last time, prove yourself better than ChatGPT. In the box below, give the (correct!) hex machine code for the instruction adr x26, label3 where label3's address is 0x4217A7 and the address of the adr instruction itself is 0x402170. (ChatGPT says the answer to this question is 0x073F2E9A ... do you trust it?)



As a reminder, the relative offset calculation is the label address minus the adr instruction's address – ChatGPT *did* manage to get that order right in its response on the previous page.

Extra omnes! I hope you take with you from COS 217 many useful skills and a good deal of pride for having persevered through it, even if perhaps you'd like to lock the memories of the midterm exam, late night debugging sessions, etc. away *cum clave* and throw away the key.

(The space below is intentionally left blank. You may use it for scratch work, but any answers given below will not be graded.)

- Q0: And I, [name], so promise, pledge and swear.
- Q1: I accept; surrounded by walls.
- Q3: To the City and the World.
- Q4: By what name do you wish to be called?
- Q7: Scrutineers vote checkers.
- Q8 (taking some liberties): We have, again, "Encore" This page: White smoke!; Everyone out!; with a key.