



This exam consists of 10 questions (85 points). You have 180 minutes: 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 (● and ■, not ✓ or ✕). Please make text answers dark and neat.

Name: NetID:

Precept:

- | | | |
|--|---|--|
| <input type="radio"/> P01 / P02 - MW
Xiaoyan Li | <input type="radio"/> P10 - TTh 12:30
Dwaha Daud | <input type="radio"/> P07 TTh 2:30
Nanqinqin Li |
| <input type="radio"/> P03 - TTh 12:30
Donna Gabai | <input type="radio"/> P05 - TTh 1:30
Donna Gabai | <input type="radio"/> P08 TTh 3:30
Indu Panigrahi |
| <input type="radio"/> P04 - TTh 12:30
Guðni Nathan Gunnarsson | <input type="radio"/> P06 - TTh 1:30
Austin Li | <input type="radio"/> P09 TTh 7:30
Gongqi Huang |

This is a closed-book, closed-note exam, except you are allowed one double-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."

X _____

Question 1: Artillery

8 points

Since 2022, Princeton Stadium boasts a new Daktronics video board just beyond its south endzone. The video board is a 19' high × 49' wide two-dimensional array of pixels.

Each pixel consists of:

- components for 3 primary colors (red, green, and blue – each represented as an integer in the range 0–255, inclusive)
- references to up to four neighbors (left, right, above, and below), and
- its own x and y coordinates (indices) within the array of pixels

We want to define an interface and implementation for streaming to Princeton Stadium's main video board. To do so, let's assume that there will only ever be one such gigantic video board in the stadium.

- a. Would this video board module be better represented as an abstract object (AO) or an abstract data type (ADT)? Write your answer in the box below:

- b. Would a pixel module be better represented as an abstract object (AO) or an abstract data type (ADT)? Write your answer in the box below:

- c. In the box below, give plausible definitions (type and name) for pixel's state variables, as they would appear as `static` file-scope variables (if you put in b. that it's an AO) or fields of a pixel struct (if you put in b. that it's an ADT).

Question 2: Campanile

12 points

This program is intended to print out each of its command line arguments (including the executable name itself) in order, **each on its own line**. Unfortunately, the ten lines from the core of its implementation have become jumbled, and the macro definitions of `PPC_ZERO` and `PC_ZERO` have been lost:

```
#include <stdio.h>
int main(int argc, char *argv[]) {

A   char **ppc = argv;
B   while(*ppc != PPC_ZERO) {
C   } /* end while(*ppc ...) { */
D   char *pc = *ppc;
E   while(*pc != PC_ZERO) {
F   } /* end while(*pc ...) { */
G   ppc++;
H   pc++;
I   putchar('\n');
J   putchar(*pc);

   return 0;
}
```

a. Which is the correct ordering of lines A through J?

b. Which are the most correct literal values for `PPC_ZERO` and `PC_ZERO`?

- `PPC_ZERO: NULL` and `PC_ZERO: NULL`
- `PPC_ZERO: NULL` and `PC_ZERO: '\0'`
- `PPC_ZERO: '\0'` and `PC_ZERO: NULL`
- `PPC_ZERO: '\0'` and `PC_ZERO: '\0'`

Question 3: Seventh Solfège Syllable

8 points

Consider the following partial implementation of a “round” song as a **circular queue** of music notes. Queues are “First In, First Out”. The questions appear on page 5.

```
enum note { DO, RE, MI, FA, SOL, LA, TI };

struct node {
    enum note en;
    struct node *next;
};

struct queue {
    struct node *tail;
    struct node *head;
    size_t size;
};

struct queue *Queue_new() {
    return calloc(1, sizeof(struct queue));
}

void Queue_free(struct queue *psQ) {
    struct node *curr;
    assert(psQ != NULL);
    curr = psQ->head;
    while(curr != NULL) {
        psQ->head = curr->next;
        free(curr);
        if( /* to be completed in part a. */ )
            break;
        curr = psQ->head;
    }
    free(psQ);
}

void Queue_append(struct queue *psQ, enum note enNote) {
    struct node *new;
    assert(psQ != NULL);
    new = calloc(1, sizeof(struct node));
    if(new == NULL)
        return;
    if(psQ->head == NULL)
        psQ->head = new;
    else
        /* to be completed in part b. */
        psQ->tail = new;
    psQ->size++;
    new->next = psQ->head;
    new->en = enNote;
}
```

- a. In the box below, complete the missing conditional in the `Queue_free` function, so that the function returns after freeing all allocated dynamic memory.

```
if(                                     )
```

- b. In the box below, complete the missing `else` clause in the `Queue_append` function, so that the function appends the new node to the tail end of the queue. This should be a single C assignment.

- c. In the box below, write a single C statement that would establish `Q_T` as a type alias for the `struct queue * opaque` pointer type.

The exam continues on page 6. The remainder of this page may be used for scratch work, however any answers given on this page below this text will not be graded.

Question 4: *Hedera helix*

16 points

Consider the following incomplete scaffolding for a computational biology program:

```
enum base {A, C, G, T, U};

struct pair {
    enum base b1;
    enum base b2;
};

struct pair wcf1;
static struct pair wcf2 = {A, T};

void DNA() {
    wcf1.b1 = G;
    wcf1.b2 = C;
    /* other code will follow */
}

void mRNA() {
    struct pair mRNApairs[3] = { {G, C}, {U, A}, {A, T} };
    /* other code will follow */
}

void tRNA() {
    static struct pair tRNApairs[2] = { {G, C}, {A, U} };
    /* other code will follow */
}
```

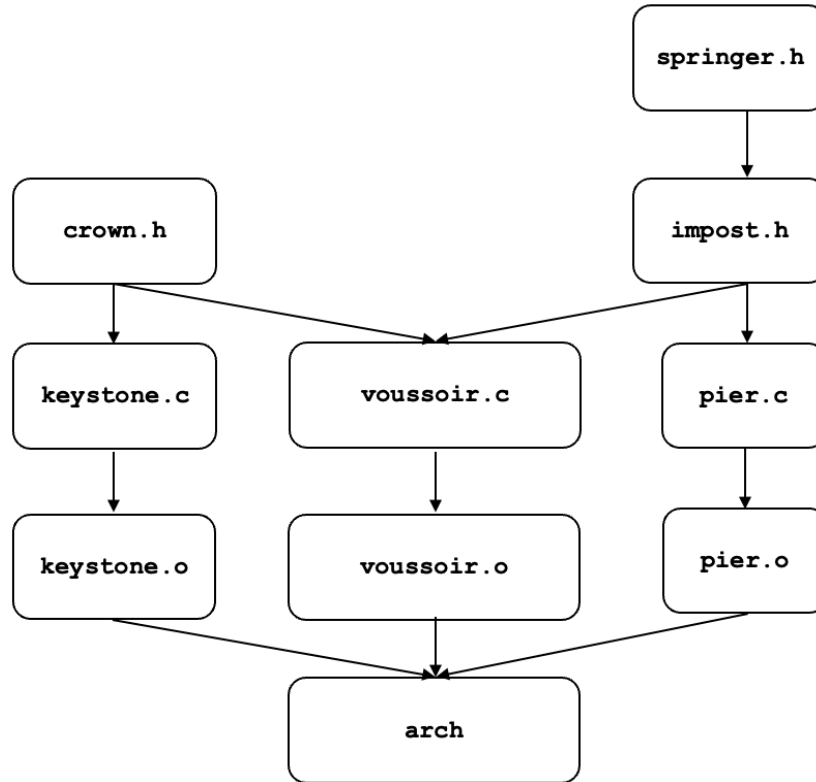
Complete the table below to indicate the scope, linkage, and duration of each variable and the section of memory in which it resides. For scope, write either “FILE” or “BLOCK”; for linkage, write either “INTERNAL” or “EXTERNAL”; and for duration, write either “PROCESS” or “TEMPORARY”.

	SCOPE	LINKAGE	DURATION	SECTION
wcf1				
wcf2				
mRNApairs				
tRNApairs				

Question 5: Abbey Arcade

10 points

Imagine a proper Makefile that supports partial builds and produces an executable named `arch` according to the dependency graph shown below. (Arrows from header files indicate `#includes`, e.g., `pier.c #includes impost.h`. Arrows from other files indicate the progression of the build process, e.g., `pier.o` is built out of `pier.c`.)



Note that each of the seven architectural terms in the source files' names begins with a unique letter, so you may choose to use that letter instead of the full word (e.g., `i.h` instead of `impost.h` or `p.o` instead of `pier.o`) as you answer the questions below.

- a. In the box below, write the **list of dependencies** for the target `voussoir.o`

- b. In the box below, write the (`gcc217`) **command** for building the target `arch`

You may refer to this abbreviated ARM assembly language reference for Q6 – Q9.

Instruction(s)	Description
{add,sub,lsl} dst, src1, src2	dst = src1 {+, -, <<} src2
{beq,bne} label	Go to label if comparison was {"equal", "not equal"}
{b,bl} label	{Unconditionally go to , Call function at} label
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

Question 6: Founding Document

4 points

These symbolic constants have been defined in an ARM assembly language program:

```
.equ PSTRUCT, 8
.equ FIELD, 16
.equ VAR, 16
```

Later on in the program, this series of instructions appear:

```
// REPLACE THIS COMMENT
ldr x0, [sp, PSTRUCT]
add x0, x0, FIELD
mov x1, 217
ldr x0, [x0, x1, lsl 3]
str x0, [sp, VAR]
```

In the box below, write an appropriate line of C – using variable names similar to the .equs defined above – that could replace the comment on the first line in the previous box in order to explain those 5 instructions.

Question 7: Pre-Revolution

11 points

Consider the following function that returns the length of a string's prefix that contains only a specific character. For example, `Str_prefixLen("CClub", 'C')` will return 2.

```
#include <stddef.h>
size_t Str_prefixLen(const char *s, char c) {
    if(*s != c)
        return 0;
    return 1 + Str_prefixLen(s+1, c);
}
```

In the box below, write the function in ARM assembly language, with these restrictions:

1. the algorithm should be faithful to the C code (i.e., it should still be recursive)
2. the stack should be used only for x30 (i.e., not local variables and parameters)
3. Scratch registers should be used for local variables, parameters, and any temporary values required for your computations.

```
.section ".text"
.global Str_prefixLen
Str_prefixLen:
```

```
ret
```

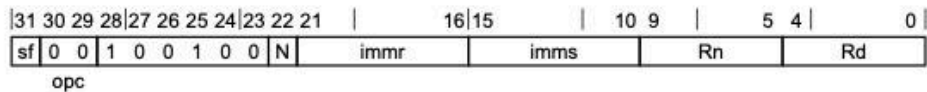
Question 8: Veranda

10 points

Consider the following two patterns for ARM assembly language and instructions, which will be needed to complete parts a. through e. of this question on page 11.

C6.2.11 AND (immediate)

Bitwise AND (immediate) performs a bitwise AND of a register value and an immediate value, and writes the result to the destination register.



32-bit variant

Applies when $sf == 0$ && $N == 0$.

AND <Wd|WSP>, <Wn>, #<imm>

64-bit variant

Applies when $sf == 1$.

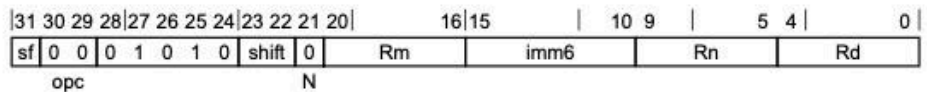
AND <Xd|SP>, <Xn>, #<imm>

<imm> For the 32-bit variant: is the bitmask immediate, encoded in "imms:immr".

For the 64-bit variant: is the bitmask immediate, encoded in "N:imms:immr".

C6.2.12 AND (shifted register)

Bitwise AND (shifted register) performs a bitwise AND of a register value and an optionally-shifted register value, and writes the result to the destination register.



32-bit variant

Applies when $sf == 0$.

AND <Wd>, <Wn>, <Wm>{, <shift> #<amount>}

64-bit variant

Applies when $sf == 1$.

AND <Xd>, <Xn>, <Xm>{, <shift> #<amount>}

<shift> Is the optional shift to be applied to the final source, defaulting to LSL and encoded in the "shift" field. It can have the following values:

LSL	when shift = 00
LSR	when shift = 01
ASR	when shift = 10
ROR	when shift = 11

<amount> For the 32-bit variant: is the shift amount, in the range 0 to 31, defaulting to 0 and encoded in the "imm6" field.

For the 64-bit variant: is the shift amount, in the range 0 to 63, defaulting to 0 and encoded in the "imm6" field.

In the box beside each machine language instruction encoding below, write the number of the corresponding assembly language instruction from the list on the right, or NONE if it does not match any of the instructions in the list. Each option, including NONE, will be used exactly once.

Warning: the N, immr, and imms fields in the immediate operand version of the instruction are inscrutable. But don't despair! You do **not** need to produce or interpret these fields' values in order to solve this problem: the other fields give enough information to do the matching below.

a. 0xf27f0020	<input type="text"/>	1. and w0, w1, w2
b. 0x121f0020	<input type="text"/>	2. and x0, x1, x2
c. 0x8a020020	<input type="text"/>	3. and w0, w1, #2
d. 0x0a020020	<input type="text"/>	4. and x0, x1, #2
e. 0x927f0020	<input type="text"/>	

The exam continues on page 12. The remainder of this page may be used for scratch work, however any answers given on this page below this text will not be graded.

Question 9: Special Regalia

3 points

In one of your first ARM precept handouts, a key was given for interpreting the allowable registers used as register operands in ARM assembly language instructions:

Wn	4 byte general register, or WZR
Wn WSP	4 byte general register, or WSP
Xn	8 byte general register, or XZR
Xn SP	8 byte general register, or SP

For many instructions, their register operands may be SP but not ZR; while other instructions have the opposite restriction: their register operands may be ZR but not SP. Violating this will result in an assembler error, for example:

```
zrspzr.s:1: Error: operand 2 must be an integer register
-- `sub xzr, sp, xzr'
```

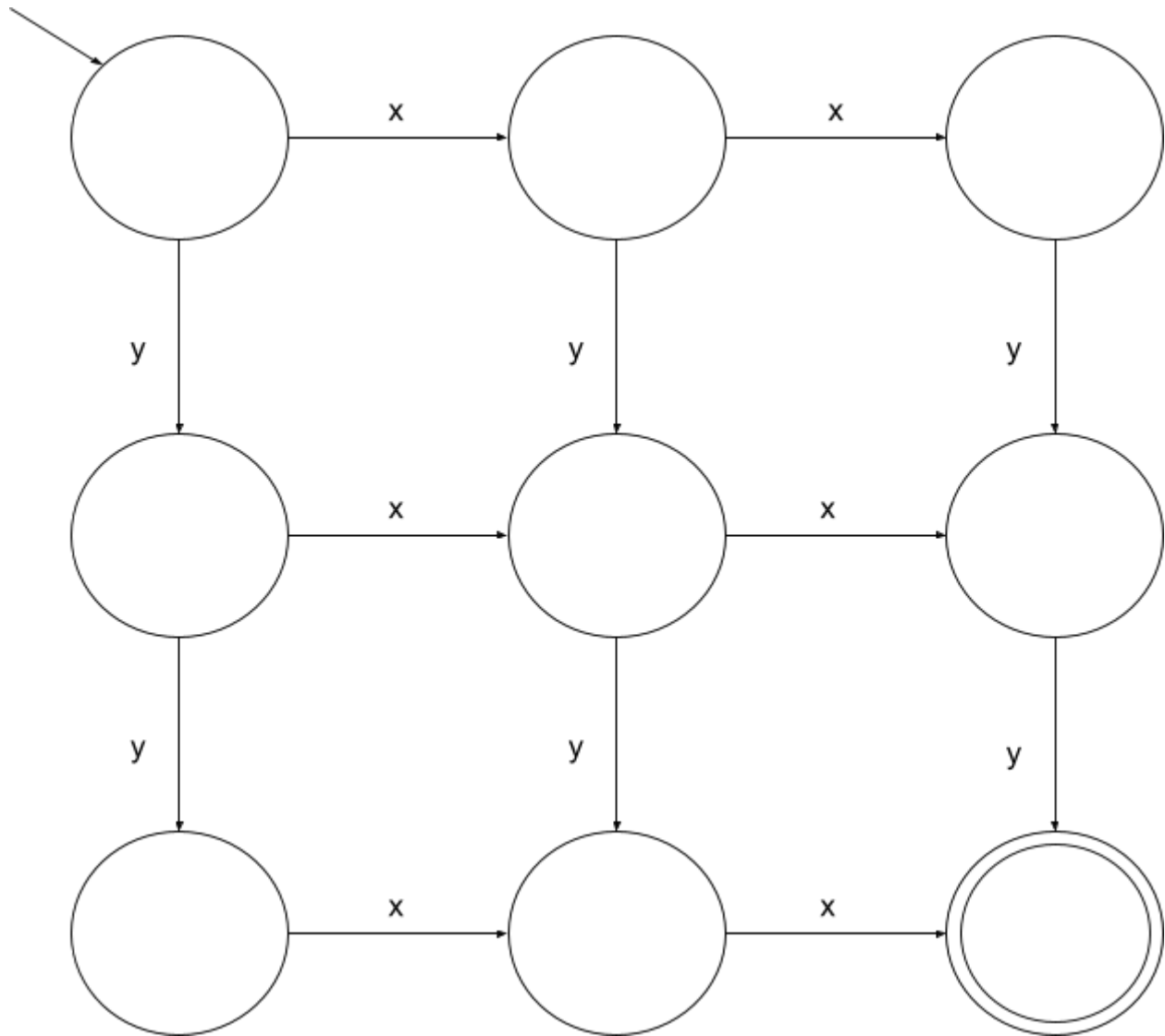
In the box below, explain in 1 sentence why SP and ZR must be mutually exclusive in these instructions' operands. (Hint: consider their machine language representation.)

Question 10: Tetragon

3 points

Consider this DFA, which handles strings consisting of only the characters **x** and **y**. The top left state is the start state. The bottom right state is the only accepting state.

Any possible state transition that is not shown with an edge should be assumed to be a self-loop (i.e., remain in the same state). E.g., when the bottom middle state reads **y**, it stays there. Similarly, once we are in the accept state, we will stay there forever.



In the box below, give a short English description of the set of strings this DFA accepts.

Question 10 was the last question. This page is only frivolity.

For 0 points (so please don't think about this if your time would be better spent reviewing one of the actual questions!), but significant puzzle-solving respect:

This exam had 10 questions, with each title referencing one member of a group of 11. If the exam had one more question, to complete the set, what might its title have been?

The space below is intentionally left blank. It may be used for scratch work, however any answers given on this page will not be graded.