Wawrzynek, Weaver
Fall 2021

CS 61C

Midterm

PRINT your name: _____ _____
(first) (last)

PRINT your student ID: _____

Read the following honor code and sign your name.

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam and a corresponding notch on Nick's Stanley Fubar demolition tool.

SIGN your name: _____

You have 110 minutes. There are 6 questions of varying credit (100 points total).

For questions with **circular bubbles**, you may select only one choice.

◯ Unselected option (completely unfilled)

● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

■ You can select

■ multiple squares (completely filled).

Anything you write that you ~~cross out~~ will not be graded.

If an answer requires hex input, make sure you only use capitalized letters! For example, `0xDEADBEEF` instead of `0xdeadbeef`. Please include hex (`0x`) or binary (`0b`) prefixes in your answers. For all other bases, do not add the suffix or prefixes.

This page is intentionally left blank.

Q1 *Potpourri* **(10 points)**

Q1.1 (1.25 points) True or False: The compiler resolves define statements.

◯ True ◯ False

Q1.2 (1.25 points) True or False: The assembler is the step with the highest computational complexity among CALL.

◯ True ◯ False

Q1.3 (1.25 points) True or False: The assembler produces an executable.

◯ True ◯ False

Q1.4 (1.25 points) True or False: In the loader, the program is placed in memory in preparation of running the code.

◯ True ◯ False

Q1.5 (1.25 points) Convert `0xDA71` to a 16-bit binary value, including the prefix.

Q1.6 (1.25 points) Convert `0x85` to decimal, assuming the data was stored as an unsigned one-byte integer.

Q1.7 (1.25 points) Convert `0x85` to decimal, assuming the data was stored as a 2's complement one-byte integer.

Q1.8 (1.25 points) Convert `0x85` to decimal, assuming the data was stored as a sign-magnitude one-byte integer.

Q2 *Now, Where Did I Put Those Strings?* **(10 points)**

Consider the following code:

```
char *foo() {
    char *str1 = "Hello World";
    char str2[] = "Hello World";
    char *str3 = malloc(sizeof(char) * X);
    strcpy(str3, "Hello World");
    // INSERT CODE FROM PARTS 5-7
}
```

The `char *strcpy(char *dest, char *src)` copies the string pointed to by `src`, including the terminating null byte (`'\0'`), to the buffer pointed to by `dest`. The strings may not overlap, and the destination string `dest` must be large enough to receive the copy.

Q2.1 (1 point) Where is `*str1` located in memory?

◯ code ◯ static ◯ heap ◯ stack

Q2.2 (1 point) Where is `*str2` located in memory?

◯ code ◯ static ◯ heap ◯ stack

Q2.3 (1 point) Where is `*str3` located in memory?

◯ code ◯ static ◯ heap ◯ stack

Q2.4 (1 point) What is the minimum value of X needed for the code to have well-defined behavior?

The code from the previous page has been copied here:

```
char *foo() {
  char *str1 = "Hello World";
  char str2[] = "Hello World";
  char *str3 = malloc(sizeof(char) * X);
  strcpy(str3, "Hello World");
  // INSERT CODE FROM PARTS 5-7
}
```

Which of the following lines can be inserted into the function at the given line, with well-defined behavior? Select all that apply.

Q2.5 (1 point) Returning the string.

- ☐ `return str1;`
- ☐ `return str3;`
- ☐ `return str2;`
- ☐ None of the above

Q2.6 (1 point) Modifying the string.

- ☐ `str1[0] = 'J';`
- ☐ `str3[0] = 'J';`
- ☐ `str2[0] = 'J';`
- ☐ None of the above

Q2.7 (1 point) Freeing the string.

- ☐ `free(str1);`
- ☐ `free(str3);`
- ☐ `free(str2);`
- ☐ None of the above

Q2.8 (1 point) Printing the string.

- ☐ `printf("%s\n", str1);`
- ☐ `printf("%s\n", str3);`
- ☐ `printf("%s\n", str2);`
- ☐ None of the above

Q2.9 (2 points) If this code was run on a little-endian system, what would `((uint32_t*) str1)[2]` evaluate to? Express your answer in hexadecimal, with the necessary prefix. Note that `uint32_t` refers to an unsigned 32-bit integer.

Q3 *I C a Scheme* **(20 points)**

Consider the following C code:

```
union ExtraStuff {
    char a[5];
    uint16_t b;
    int c;
    double d;
};
```

```
typedef struct ConsCell {
    void *car;
    void *cdr;
    union ExtraStuff extra;
} cons;
```

Consider the following function: `cons *map(cons *c, (void *)(*f)(void *));`

`map` takes a pointer to a `cons` struct `c` and a function pointer `f`.

If the `cons` struct pointer is `NULL`, `map` returns `NULL`. Otherwise, it does the following:

1. Allocate a new `cons` struct. `ret` is a pointer to this new struct.
2. Set the contents of the `extra` union in `ret` to be all zeros.
3. Set the `car` field in `ret` to the result of calling `f` on the `car` pointer in `c`.
4. Set `cdr` field in `ret` to the result of calling `map` recursively on the `cdr` pointer in `c`.

Q3.1 (18 points) Complete the following code by filling in the blanks. This code should compile without errors or warnings. Each blank is worth 2 points.

```
cons *map(cons *c, (void *) (*f) (void *)) {
    cons *ret;

    if ( _____ ) return _____;

    ret = malloc(_____);

    _____extra_____ = 0;

    _____car = _____;

    _____cdr = _____;
    return ret;
}
```

Q3.2 (2 points) On a 32-bit architecture, what is `sizeof(cons)`?

Q4 *To Float or Not to Float* **(20 points)**

Consider a floating point system that has 16 bits with 7 bits of exponent and an exponent bias of -63, which otherwise follows all conventions of IEEE-754 floating point numbers (including denorms, NaNs, etc.). In this question, we will compare this system to an unsigned 16-bit integer system.

Q4.1 (4 points) What is the value of floating point number `0xC2C0` in decimal?

Q4.2 (1 point) Which representation has more representable numbers? Count +0, -0, $+\infty$, and $-\infty$ as 4 different representable numbers.

◯ The floating point number

◯ The unsigned 16-bit integer

◯ Both systems can represent the same number of values

Q4.3 (3 points) How many more numbers can be represented? Write 0 if both systems can represent the same number of values.

Q4.4 (4 points) Out of all numbers representable by this floating point system, what is the largest number that can also be represented as an unsigned 16-bit integer?

Q4.5 (4 points) What is the smallest positive number representable by this floating point system that isn't representable by the unsigned 16-bit integer?

Q4.6 (4 points) What is the smallest positive number representable by the unsigned 16-bit integer that isn't representable by this floating point system?

Q5 *A RISC-y Program* **(20 points)**

In addition to storing the `ra` register and the `s` registers, the stack can also store local variables. You have access to the following labels defined externally:

- `Password`: a pointer to a statically-stored string `"secretpass"`
- `Get20chars`: A function defined as follows:
  - Input: `a0` is a pointer to a buffer
  - Effect: Reads characters from stdin, and fills the buffer pointed to by `a0` with the read data, null-terminating the string. Your code may assume that the input is at most 19 characters, not including the null-terminator.
  - Output: None

The function `verifypassword` is defined as follows:

- Input: No register input; however, the function receives a string input from stdin.
- Output: a0 returns 1 if the input from stdin is exactly `"secretpass"`, and 0 otherwise.

Q5.1 (12 points) Complete the function `verifypassword`. Each line contains exactly one instruction or pseudoinstruction.

```
 1: verifypassword:
 2:     addi sp, sp, -24 # Make space for a 20-byte buffer

 3:     sw _______________ 20(sp)

 4:     ___________________________________________

 5:     jal ra Get20chars

 6:     _____________ t0 Password

 7:     _____________ t1 sp

 8: Loop:

 9:     _____________ t2 0(t0)

10:     _____________ t3 0(t1)

11:     ___________________________________________

12:     ___________________________________________

13:     addi t0 t0 _____________

14:     addi t1 t1 _____________

15:     ___________________________________________

16: Pass:

17:     ___________________________________________

18:     ___________________________________________

19: Fail:

20:     ___________________________________________

21: End:

22:     ___________________________________________

23:     ___________________________________________

24:     ___________________________________________
```

Q5.2 (4 points) Translate `addi sp, sp, -24` to its machine-language hexadecimal representation, with the appropriate prefix.

Q5.3 (4 points) Assume that `verifypassword` is located at `0x00001000`, and `Get20chars` is located at `0x00000f00`, and that line 4 is exactly one instruction (not a pseudoinstruction). Translate the line `jal ra Get20chars` to its machine-language hexadecimal representation, with the appropriate prefix.

Q6 *Testception* **(20 points)**

Recall the following information from the previous question:

You have access to the following labels defined externally:

- `Password`: a pointer to a statically-stored string "secretpass"
- `Get20chars`: A function defined as follows:
  - Input: a0 is a pointer to a buffer
  - Effect: Reads characters from stdin, and fills the buffer pointed to by a0 with the read data, null-terminating the string. Your code may assume that the input is at most 19 characters, not including the null-terminator.
  - Output: None

The function `verifypassword` is defined as follows:

- Input: No register input; however, the function receives a string input from stdin.
- Output: a0 returns 1 if the input from stdin is exactly "secretpass", and 0 otherwise.

Propose a suite of 4-6 tests you would use to verify that an implementation of this function works properly. Your test suite does not need to be comprehensive, but each test should test something different. We will only count your best 4 tests when grading.

Each test should consist of a list of inputs, expected outputs, and a one-sentence justification for why this test is useful. A useful test without proper justification will not receive credit.

Some cases you can test include:

- A generic case that returns true
- A generic case that returns false
- Other calling convention checks
- Edge cases

`rand0` through `rand11` is a constant set of 12 randomly generated numbers, which you can use in your tests.

A valid example is provided below; you may not reuse the example.

	Test 0
Input(s)	`a0-a7 = rand0-rand7` `stdin = "secretpass"`
Output(s)	`a0 = 1`
Justification	Check for using unset `a` registers.

	Test 1
Input(s)	
Output(s)	
Justification	

	Test 2
Input(s)	
Output(s)	
Justification	

	Test 3
Input(s)	
Output(s)	
Justification	

	Test 4
Input(s)	
Output(s)	
Justification	

	Test 5
Input(s)	
Output(s)	
Justification	

	Test 6
Input(s)	
Output(s)	
Justification	