DEPARTMENT OF COMPUTER SCIENCE

COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 341: Computer Organization
WS 7: Procedures

1 (a) Which registers do you put values into for a function to get them from (i.e. parameters/arguments)?

(b) What two actions does JAL perform?

(c) In which register should you save the return address to when calling a function?

(d) Which registers does a function put values in to return them to the caller?

(e) jalr zero, ra, 0 jumps to the address in ra. What should ra hold?

2 List which registers are saved by callers and callees. Which are not saved at all

3 What is a leaf procedure? What registers should it use?

4 Arrays in C and RISC-V may seem different, but under the hood they operate the same. Consider the following C code:

```
int array[]={0,1,2,3,4,5,9,7,8,6};
printf("%d",array[6]);
```

array[3] is what programmers call "syntactic sugar", it makes the higher level program easier to read. However, when the compiler sees this, it might replace it with this:

```
printf("%d\n",*(array+6));
```

Why? An array is actually a pointer. The above line of code takes the pointer to the first element of the array, adds the size of 6 integers to it, then dereferences the pointer to get the value at that memory location. You might be asking how C knows to add the size of 6 integers to the base address. The answer is because C is (mostly) typesafe. The compiler notices that the array is of type integer, and adjusts the amount added to the array accordingly. This can be done manually, but a couple "hacks" are needed. We need to remove C's type safety. This is accomplished by casting the int pointer to a void pointer. This removes any notion of the compiler knowing what size the underlying data type is.

Translate the first segment of C code presented in this problem, shown again below for convenience, to RISC-V. Don't worry about formatting the string, just print an integer.

```
int array[]={0,1,2,3,4,5,9,7,8,6};
printf("%d",array[6]);
```

5 Write this recursive C function as a recursive RISC-V function (assume no overflow):

```
unsigned int sumseries(unsigned int start, unsigned int end) {
    if (start >= end) return end;
    return start + sumseries(start+1,end);
}
```