ISA Project

For this project, you write Charm assembly code that corresponds to the following C code. Place your code in the file `isaproject.s`. Use `os.o` to accomplish the `printf` calls.

Project C Code

```
#include <stdio.h>

int sum_array(int *ia) {
    int s = 0;
    while (*ia != 0) {
        s += *ia;
        ia++;
    }
    return s;
}

int cmp_arrays(int *ia1, int *ia2) {
    int s1 = sum_array(ia1);
    int s2 = sum_array(ia2);
    printf("s1: %d, s2: %d\n", s1, s2);
    return s1 == s2 ? 0 : (s1 > s2 ? 1 : -1);
}

int numelems(int *ia) {
    int c = 0;
    while (*ia++ != 0)
        c++;
    return c;
}

void sort(int *ia) {
    int s = numelems(ia);
    for (int i = 0; i < s; i++)
        for (int j = 0; j < s-1-i; j++)
            if (ia[j] > ia[j+1]) {
                int t = ia[j];
                ia[j] = ia[j+1];
                ia[j+1] = t;
            }
}
```

```
int smallest(int *ia) {
    int s = numelems(ia);
    int sm = *ia;
    for (int *p = ia; p < ia+s; p++)
        if (*p < sm)
            sm = *p;
    return sm;
}


int factorial(int n) {
    if (n == 1)
        return 1;
    else
        return n * factorial(n-1);
}

static int sia[] = {50,43,100,-5,-10,50,0};
static int sib[] = {500,43,100,-5,-10,50,0};

int main() {
    int ia[] = {2,3,5,1,0};
    int cav = 0, n = 0, sm1 = 0, sm2 = 0;
    cav = cmp_arrays(sia, sib);
    printf("cmp_arrays(sia, sib): %d\n", cav);
    cav = cmp_arrays(sia, sia);
    printf("cmp_arrays(sia, sia): %d\n", cav);
    sib[0] = 4;
    cav = cmp_arrays(sia, sib);
    printf("cmp_arrays(sia, sib): %d\n", cav);
    cav = cmp_arrays(ia, sib);
    printf("cmp_arrays(ia, sia): %d\n", cav);
    sort(ia);
    n = numelems(ia);
    for (int i = 0; i < n; i++)
        printf("ia[%d]: %d\n", i, ia[i]);
    sort(sia);
    n = numelems(sia);
    for (int i = 0; i < n; i++)
        printf("sia[%d]: %d\n", i, sia[i]);
    sm1 = smallest(ia);
    sm2 = smallest(sia);
    printf("smallest(ia): %d\n", sm1);
    printf("smallest(sia): %d\n", sm2);
```

```
    if (sm1 != ia[0])
        printf("Something bad\n");
    else
        printf("Nice sort and smallest\n");
    if (sm2 != sia[0])
        printf("Something bad\n");
    else
        printf("Nice sort and smallest\n");
    n = factorial(4);
    printf("factorial(4) ia: %d\n", n);
    n = factorial(7);
    printf("factorial(7) ia: %d\n", n);
}
```

Project Assembly Template

The file `isaproject_template.s` is provided to you as your starting template[1]. You should copy `isaproject_template.s` to `isaproject.s`. You edit `isaproject.s` and submit your final version of it.

Project Guidance

1. **Edit/Assemble Guidance**: When editing `isaproject.c`, first invoke `chasm` to ensure your changes successfully assemble. If they do not successfully assemble, the errors are displayed in your terminal. Fix all assembly errors.
2. **Listing File Guidance**: The `-l` option on `chasm` can be used to generate a listing file, which shows the memory locations of your data and instructions. This file can be handy when examining the instruction stream in `chemun`.
    a. `% chasm -l isaproject.txt isaproject.s > isaproject.o`
3. **Understand Project C Code**: The first step is to fully understand the C code provided above. Study the code, create an `isa_project.c` file with the code, compile, link, and run the code on Linux to improve your understanding.
4. **Understand Template Charm Assembly**: Study the `isaproject.s` template file to understand what is provided as the starting point.
    a. Notice the program's memory layout, which is created with .data and .text directives. For example, the static data begins at address 0x100, the function `sum_array` begins at address 0x200, and the function `cmp_arrays` begins at address 0x300.
    b. Notice how main allocates a stack for its local variables and saving the link register.
    c. Notice how the other functions do not carve a stack. The functions `sum_array` and `numelems` are a leaf functions (does not call another function) and you

[1] I will post `isaproject_template.s` on Discord.

should be able to compute its answer with registers r0, r1, r2, and r3, which means you do not need to carve a stack for numelems. You must create a stack for the other functions.

d. Notice how the format strings are defined in the static variable section.
e. Notice how a format string is passed as a parameter in r1 to use the kernel service of `printf`. Notice how a `printf` format string may have up to two `%d` in it. Notice how the parameters matching the `%d`'s are passed to the kernel `printf` service in registers r2 (the first `%d`) and r3 (the second `%d`).
f. Notice how `main` creates a `for` loop.
g. Notice how `main` calls the functions. The returned values are stored in local variables. The template versions of functions return hard coded values. You must create code the computes and returns the correct values.
h. Notice how after running the program, you can dump the address in the stack pointer to observe the value on the stack.

5. **Run Template**: Assemble and execute the `isaproject.s` template file to improve your understanding. When running the `isaproject.s` template, you can first run it with a large step (s 300) to execute the entire program. Then you can run it using single steps to observe some features. Then you can run it with breakpoints. For example, set a breakpoint on one of the functions. Be sure to use the `-o` option with invoking `chemun`. You must have the file `os.o` in the current working directory.

```
% chasm os.s > os.o
% chasm isaproject.s   Ensure there are no assembly errors.
% chasm isaproject.s > isaproject.o
% chemun -o isaproject.o
```

6. **Project Incremental Development**: Incrementally add and test features to the `isaproject.s` template file. As you incrementally add features, first test them with the static`int sia` array. After your functions work with `sia`, you can add the local `int ia` array and retest your functions with a local array. The following is one approach to solving this problem incrementally.

a. Implement and test the `sum_array` leaf function, which sums the elements in a null terminated array. When processing a null terminated array, the algorithm iterates through the array elements until a 0 is encountered.
b. Implement and test the `cmp_array` function. You will have to create a stack for `cmp_array` to save the array parameters and the link register.
c. Implement and test the `numelems` function, which counts elements in a null-terminated array. Once you implement and test `numelems`, you can update the loop in main to use the returned count to print the entire `sia` array.

```
for (int i = 0; i < numelems(sia); i++)
     printf("ia[%d]: %d\n", i, sia[i]);
```

d. Implement and test the `smallest` function. Figure ISA-24 has one implementation of a function `largest`. You can mold this implementation into your solution for the `smallest` function.
e. Implement and test the `sort` function.

f. Implement and test the `factorial` function. `factorial` is a recursive function, which demonstrates the power of functions and allocating stack.
g. Modify `main` so that it matches the C code in the project.

Project Submission

Submit the following information.

1. **`isaproject_submission.docx`**[2] - This file contains the following questions. You place your answers to the questions in the `isa_submission.docx` file and submit the file. The last question requires you to write an answer. The others are simple yes / no (or check). You should elaborate on your yes / no answers where necessary. For example, maybe you think you implemented half of a function.

- ☐ I understand the concept of an instruction set architecture (ISA) and that an ISA defines how software communicates with hardware.
- ☐ I studied the template provided and I understand it.
- ☐ I ran the template provided with a large step value to improve my understanding.
- ☐ I ran the template provided with single steps and breakpoints to improve my understanding.
- ☐ I understand the program's memory layout.
- ☐ I understand how to create conditional statements in assembly.
- ☐ I understand how to create loops in assembly.
- ☐ I understand how the `blr` instruction works.
- ☐ I understand the register conventions for functions.
- ☐ I understand how to define a function.
- ☐ I understand how to allocate space for a function's stack, which includes space for local variables, space for saving registers r4 to r12 when a function uses them, and space for saving the link register.
- ☐ I understand that a leaf function that can compute its answer with registers r0, r1, r2, and r3 does not need a stack.
- ☐ I successfully implemented the `sum_array` function.
- ☐ I successfully implemented the `cmp_arrays` function.
- ☐ I successfully implemented the `numelems` function.
- ☐ I successfully implemented the `smallest` function.
- ☐ I successfully implemented the `sort` function.
- ☐ I successfully implemented the `factorial` function.
- ☐ I successfully implemented the `main` function.
- ☐ Describe your testing that proved to you your implementation is correct. This description should cover whatever portions you successfully implemented.

[2] I will post `isaproject_submission.docx` on Discord.

2. Your **`isaproject.s`** file.

3. Your work log.