CMSC 23000 **Operating Systems** **Project 1**
Autumn 2006 **October 4**

RCX memory management

Due: Monday October 16 at 10pm

1 Introduction

This project is the first part of the RCX kernel project. The goals of this project are two:

1. Successfully compile, load, and run code using the RCX simulator.
2. Implement a memory management library for your kernel.

2 The project

The main task of this project is to implement the following functions:

`void kmem_init ();`
Initialize the memory subsystem.

`void *malloc (size_t sz);`
Allocate `sz` bytes of memory aligned on a word (2-byte) boundary. If there is insufficient memory available, then return `0`.

`void free (void *p);`
Free the memory object referenced by `p`, which should have been allocated by a `malloc`. This function should accept null pointers.

Programs are loaded at address `0x8000`; you can use the global symbol `end` to determine the end of the code and statically allocated data (*i.e.*, the expression `&end` is the address of the first word following your program in RAM). Handout 3 describes the memory layout of the RCX processor in more detail.

Your memory subsystem will need to keep track of free blocks of memory and be able to coalesce adjacent free blocks. We recommend using a first-fit strategy for allocating memory, but you could also try a best-fit approach. Remember that memory is tight on the RCX and that you cannot afford too much space overhead.

Make sure that you have committed your final version by 10pm on Monday, October 16. Using Doxygen, generate the documentation for your code. Make sure that it includes your name! The project documentation is due in class on Tuesday, October 17.

3 Compiling

Since the RCX uses an H8/3292 microcontroller for its processor, we need a *cross-compiler* to generate code for it. We have built such a toolchain, which you can find in

`/stage/cmsc23000/bin`

on the CS department's Linux machines.[1] We will seed your gforge repository with a directory tree that includes makefiles for compiling your kernel code and test programs. The result of the build process will be an ASCII S-record file, which will have a "`.srec`" filename suffix. The build process also produces a file with a "`.map`" suffix, which the simulator uses for line information. To test a program, you give the name of the S-record file as a command-line argument. For example, to compile and run the test program `test01.c` in the `user` directory of the project tree, do the following:

```
% cd user
% make
% /stage/cmsc23000/bin/rcx-sim test01.srec
```

The simulator will load the program into RAM starting at address `0x8000` and initialize the `pc` and `sp`. You can run the program by using the simulator's **Run** button and you can single-step execution using the **Step** button (there are also buttons to step by 10 and 100 instructions). The **Reset** button will reset the `pc` and `sp` to their initial values, but note that it does not reset the machine state!

4 Grading

Your project will be graded on both correctness (70%) and programming style (30%). The documentation is evaluated as part of the style component of your grade. Failure to document your code will result in no credit for the style portion of your grade.

5 Document history

2006-10-04 Original document.

[1] We hope to have a version of these tools available on the Macs in the MacLab too.