Richard Homan
Dr. Joonwan Kim
Microcomputer Design
17 April 2024

# Microcomputer Design: My Final Product

## I. Description of Project

The Microcomputer Design class offers students the opportunity to design a programmable single board computer. This computer must:

1. Be able to communicate with a personal computer via a serial port or similar
2. Have an 8-bit data bus or wider
3. At least 64K RAM
4. Have capability of 64K address space
5. Run a monitor program stored on-board

The computer's monitor program must:

1. Examine the contents of a specified memory location
2. Change the contents of any RAM location
3. Be able to load a machine coded program into memory
4. Be able to execute a loaded machine coded program from memory

Students must design the computer circuit, as well as research and implement parts compatible with the requirements. Students must also keep track of their time spent and the total monetary cost of the project. The instructor is informed of student's progress through weekly progress reports, which outline what was done in that week, and what the student is planning to do the next week. Once the board has been completed, the student must present the product and write a report.

## II. Hardware Description

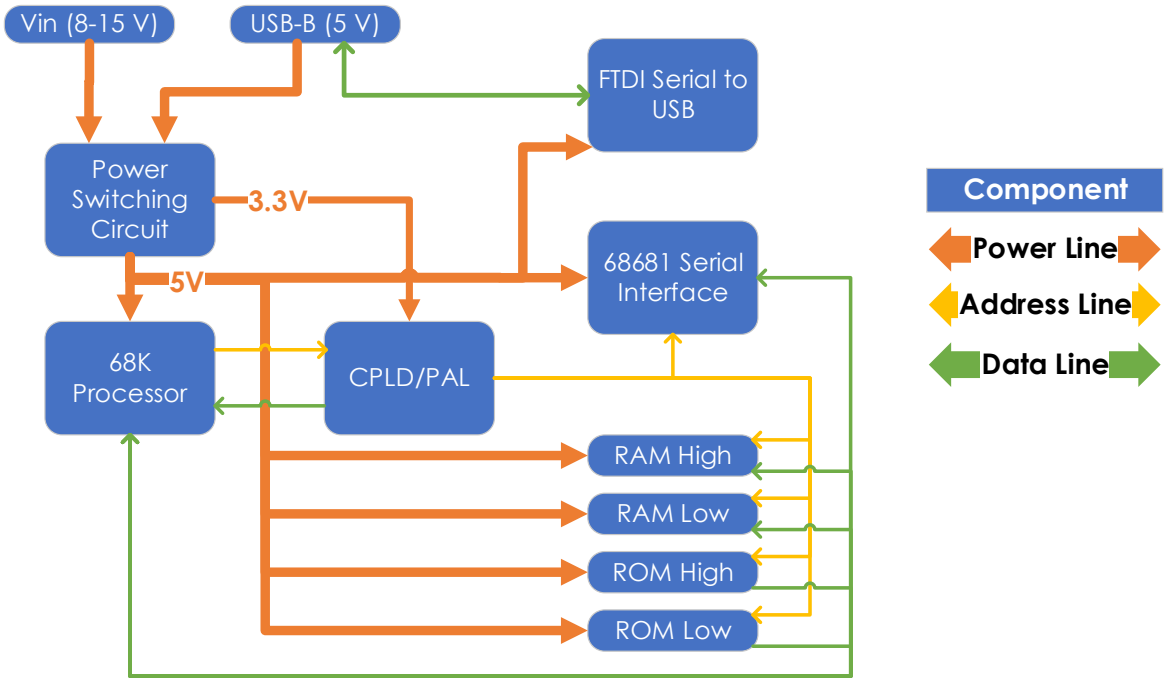
### 1. General Introduction

The MCD project, while time intensive also gives the student many freedoms in design. I took advantage of these freedoms to direct the product to my liking. I created 3 design rules for my project: the design should be compact, debuggable, and as professional as possible. To try and fit these requirements as best I could, I spent lots of time during the early weeks planning hardware design.

To make my design as professional as possible, I tried to eliminate as many unknowns as unknowns can lead to "jank" fixes in hardware. I made sure the I understood the hardware well before attempting to create a schematic. At one point, I made a manually controlled SRAM circuit to tinker with to better understand how it worked. To make my design as debuggable as possible, I made sure to include test points on control lines as well as use as many through-hole components as possible.

The exposed metal makes probing much easier. Finally, to make the design compact, I spent a great deal of time manually routing PCB traces and trying different combinations of orientations of chips to find the ideal routing. Once the PCB was complete, I had the board fabricated in Germany for a higher cost, but it was worth it for the professional and robust finished PCB.

2. Hardware Block Diagram

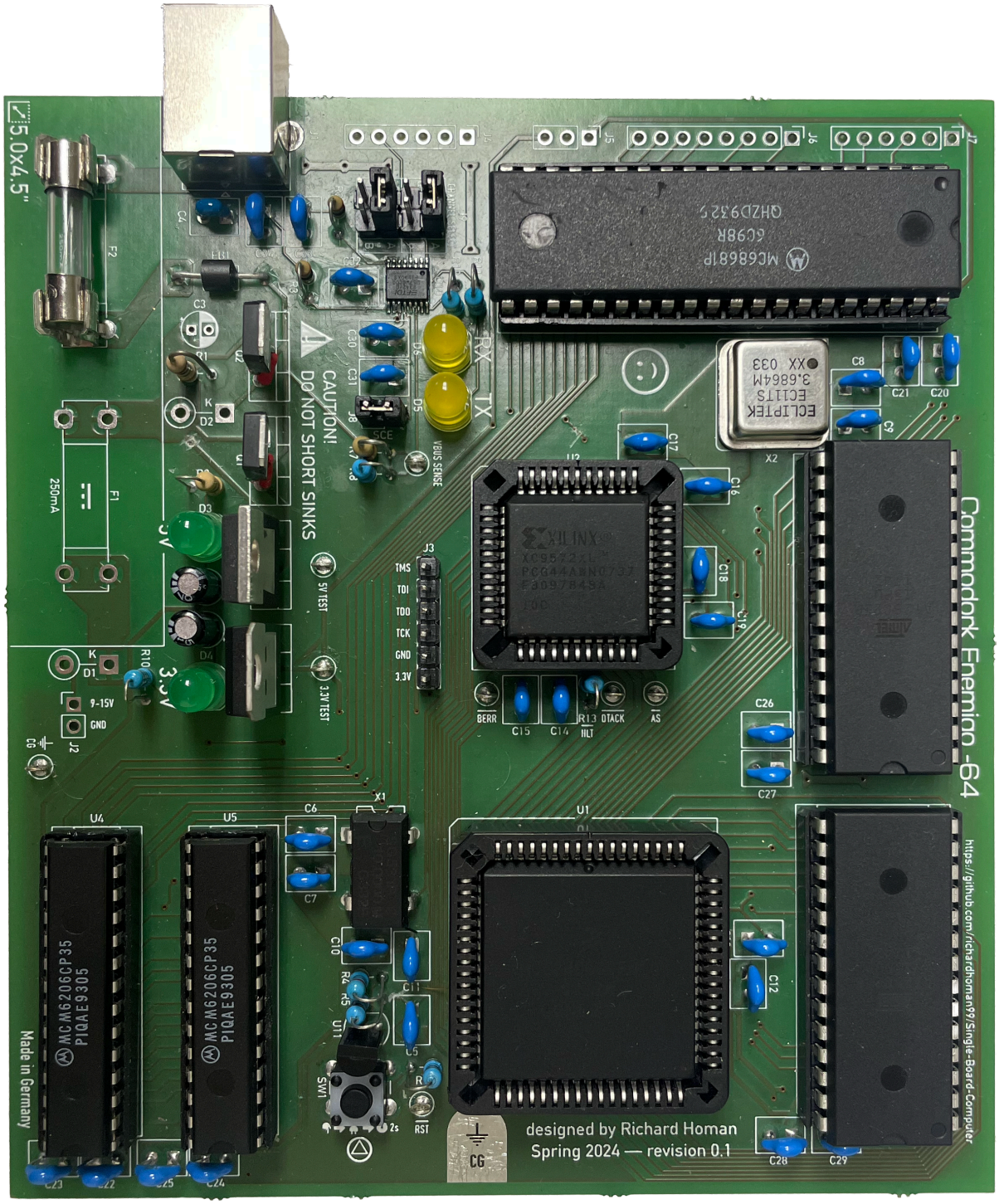



3. Device and Chip specifications

| | |
|---|---|
| MPU | Motorola 68000 |
| MPU Clock Speed | 10 MHz |
| RAM | 2x Motorola MCM6206CP35 32KB |
| ROM | 2x Atmel AT28C256 32KB |
| Interface | Motorola 68681 & FTDI FT230X |
| Connectivity | USB type B |
| Power Consumption (Average) | 1 W |

One of the main contributors to my parts selection was parts availability. The M68k MPU was selected because it was provided, and the RAM, ROM, DUART, and MPU Clock items were found in the lab.

4. Hardware Layout




5. Schematics

(See Appendix A)

6. Parts List

| Component | Item | Price | Count | Subtotal |
|---|---|---|---|---|
| **Processor** | MC68HC000FN10 | $0.00 | 1 | $0.00 |
| **CPLD** | XC9572XL | $0.00 | 1 | $0.00 |
| **RAM** | MCM6206CP35 | $0.00 | 2 | $0.00 |
| **ROM** | AT28C256 | $0.00 | 2 | $0.00 |
| **Serial Peripheral** | MC68681 | $0.00 | 1 | $0.00 |
| **USB to Serial Interface** | FT230XS-R | $2.26 | 3 | $6.78 |
| **MPU Oscillator** | SG531P 10.0000M | $0.00 | 2 | $0.00 |
| **DUART Oscillator** | EC11TS 3.6864M | $0.00 | 2 | $0.00 |
| **Supervisor** | MC34064P | $0.00 | 1 | $0.00 |
| **Reset Button** | 6mm Tact Button | $0.00 | 1 | $0.00 |
| **Regulator 5** | MC7805ACT | $0.00 | 2 | $0.00 |
| **Regulator 3.3** | LD1117AV33 | $0.79 | 2 | $1.58 |
| **Switching Mosfets** | AOI21357 | $0.72 | 3 | $2.16 |
| **USB Port Socket** | USB-B-S-RA | $0.49 | 2 | $0.98 |
| **Processor Socket** | PLCC68 | $2.94 | 2 | $5.88 |
| **CPLD Socket** | PLC44 (4x11) | $2.36 | 2 | $4.72 |
| **Fuse Holder** | 507-FC-203BRIGHTTIN-ND | $0.10 | 8 | $0.80 |
| **Ferrite Bead** | | $0.00 | 1 | $0.00 |
| **Jumpers** | QPC02SXGN-RC | $0.10 | 6 | $0.60 |
| **Test Points** | Keystone 5021 | $0.25 | 8 | $2.00 |
| **Diode** | SB160 | $0.24 | 3 | $0.72 |
| **Fuse 250** | 5ST 250-R | $0.63 | 3 | $1.89 |
| **Fuse 315** | 5ST 315-R | $0.63 | 3 | $1.89 |
| **Fuse 400** | 5ST 400-R | $0.63 | 3 | $1.89 |
| **Capacitor 47p** | BC1009CT-ND | $0.27 | 4 | $1.08 |
| **Capacitor .1µ** | 399-4264-ND | $0.24 | 20 | $4.80 |
| **Capacitor 1µ** | 399-C410C105K3R5TA-ND | $0.35 | 16 | $5.60 |
| **Capacitor 10µ** | 445-181284-1-ND | $0.52 | 5 | $2.60 |
| **Resistor 27** | CF14JT27R0CT-ND | $0.10 | 4 | $0.40 |
| **Resistor 5.6k** | CF14JT5K60CT-ND | $0.10 | 3 | $0.30 |

| Component | Item | Price | Count | Subtotal |
|---|---|---|---|---|
| **Resistor 1M** | CF14JT1M00CT-ND | $0.10 | 2 | $0.20 |
| **Serial USB Cable** | TTL-234X-5V | $20.25 | 0.33 | $6.68 |
| | | | | **$53.55** |
| | | | **Final Parts Cost (shipping & tax)** | **$63.04** |

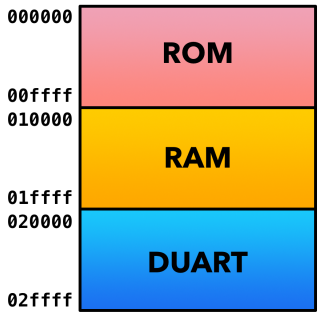
Note: Items with a price of $0 are parts that were either given or found in the lab.

## III. Software Description

### 1. General Introduction

I wanted my monitor program to be easily readable, modifiable, reliable, and user friendly. I am very familiar with the C language, and so I researched and found GCC to be a reliable C cross compiler for M68k. To make the program reliable and user-friendly, I chose to create a command-line style interface which can intuitively handle the common input-output style of shell commands. The monitor program is capable of running 7 commands: help, dump registers, dump memory, change register, change memory, load S-Record executable, and run loaded executable.

### 2. Memory Mapping




### 3. Detailed Description

The core code is split up into 7 modules: the entry point, the command lookup, the working commands, the universal prompt, the serial manager, and the S-Record parser.

*a. Entry Point*

Upon resetting the computer, the program counter is moved to the entry point. On first run, initialization code is run for the serial and working command modules. The module then goes into an infinite loop which is known as the monitor prompt. This activates the prompt module to wait and get user input. Once the user has given input and pressed the return key, the module activates the command lookup module to get the function pointer to the appropriate working command. Once the

working command is fetched, it is executed. An appropriate error message is printed if the working command exited with an error. The monitor prompt repeats.

*b. Command Lookup*

The command lookup module has two jobs: to split the single line of user input into an argument vector, and to fetch the appropriate command for given user input. To split arguments, the module simply converts all delimiting spaces to the null character, and then generates an array of string pointers which point to positions in the input buffer. To determine the proper command function pointer, the first argument in the argument list (the command name) is hashed. The hashed value is then used as an index to an array with the function pointers of the working commands.

*c. Working Commands*

The working commands module is really a set of the 7 possible commands: help, dump registers, dump memory, change register, change memory, load S-Record executable, and run loaded executable.

*d. Universal Prompt*

The universal prompt module is a subroutine which handles displaying a prompt and getting user input. This module handles displaying characters, delete/backspace events, and prevents buffer overflow. This module is used by the entry point and load S-Record working command modules.

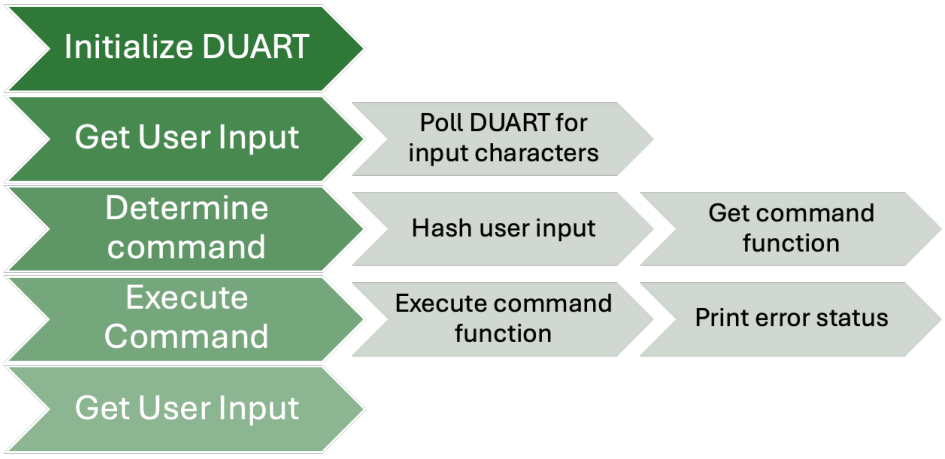
*e. Serial Manager*

The serial manager module handles serial initialization, receiving and sending characters through the serial interface, and sending strings through the serial interface. This module contains two sections: a section for the hardware, and a section for the simulator (EASy68K) as these targets have different methods of interface.

*f. S-Record Parser*

The S-Record parser module takes in an S-Record file line-by-line and parses it. The module has functionality that checks the checksum and verifies that the program doesn't run out-of-bounds.

4. Flow Chart



5. Source Code Listing
   (See Appendix B)

## IV. Troubleshooting

1. No 5V on 5V supply

   The first roadblock in board development after soldering the PCB was figuring out why voltage on the 5V supply line was in spec unloaded, but dipped drastically when load was applied (ROM, MPU installed). The issue was relatively straight forward to identify where the problem was occurring, as the problem was simply electrical. The board's power switching circuit uses a pull-down resistor from the external power node to gate 2 MOSFETs which switch between power sources (USB bus power and external power input). Connected directly to the external power node was the input voltage to the 5V regulator. During design, I did not anticipate back flow from the output of the 5V regulator to the input. This back flow switched the MOSFETs into the active region which limited the 5V supply line as the 5V supply line is connected to the USB bus supply line directly via those MOSFETs.

   To fix this problem, I opened the trace that led from the external power input node to the input on the 5 volt regulator. In the future, this design flaw could be fixed by simply adding a diode between the external power input net and the 5V regulator input net.

2. Supervisor Asserting Reset

   Almost directly after fixing the 5V supply voltage issue, the MPU would still not start. It was determined that the RESET line was being asserted. The supervisor IC I chose would assert if the voltage on its input pin was less than 4.6 volts. The voltage divider I had calculated to properly translate a 4.75V supply voltage (USB supply minimum voltage specification) to 4.6 volt supervisor input voltage was too low, causing the supervisor to always reset. After some more calculations, I chose a new combination of resistors which improved the input signal, but still did not work. After some trial and error with a few more sets of resistors, a working combination was determined to be 220 Ω for R4 and 10 kΩ for R5.

3. Stack Misalignment

   Once the MPU was running, it would begin running in its normal startup sequence, then assert HLT after a few bus cycles. This seemingly unreasonable error took many hours and about a week to determine and fix. My first approach to solving the issue was to single step the MPU. This was done with modified Address Router code and a button signal on a then unused input on the DUART socket. After verifying each access to data, I was given no leads. I then checked the assembly to the machine code, which checked out. I didn't make the connection between the stack and the faulty instruction

for many hours of contemplation. It was late on a Sunday when I realized that the last buss access before HALT was an attempt to access the stack after a `link` instruction generated by the compiler. This RAM access prompted me to check my linker script.

The problem stemmed from writing the linker script the previous month. When writing the script, I mistakingly set my stack to `0xffff`, which is a byte aligned address that also existed in ROM, not RAM. It had just so happened that my instructor was discussing stack alignment in my Computer Architecture class, and I made the realization. Recognizing the stack misalignment error was exactly the type of bug I was expecting to find, given that the result was repeatable on multiple MPUs.

After fixing the alignment, the MPU completed a `nop` program with ease. However, when I wrote a program that used stack variables and then attempted to access the variable, the value would not change. This lead me to the possibility that maybe the code is trying to write to ROM, which it was. I then changed my RAM section to start at 0x010000 and my stack to start at 0x020000.

4. Laptop Not Recognizing Serial Interface

   At the end of a lab session, I attempted to change a setting on the FTDI serial converter, but the device was no longer recognized by either of my laptops. Because the device was working before, I reviewed the devices datasheet. With some careful reading, I noticed that in every example of the chip being used, the RESET line was only ever pulled up to 3.3V. My design had it connected to the 5V board RESET signal. I figured that this had burned out internal silicon on the chip, so I devised a solution.

   The RESET and 3.3V output lines of the serial converter are located directly next to each other. I would open the 5V board reset line that connected to the serial converter, then strategically solder bridge the RESET and 3.3V output lines on it. This solution worked, and no other problems with the serial converter were encountered after this point.

5. Intermittent Access Errors On Function Calls

   DUART testing was successful, so I compiled my monitor program for machine hardware. The program would run for intermittent amounts of time before encountering an access error. To try to narrow down the issue, I began commenting out modules of the program until I found the source of the error. This action helped only intermittently. The program would sometimes make a full run, other times not. I had noticed that one of the RAM chips didn't seem fully installed, even when applying extreme amounts of force. Applying pressure to a part of this chip seemed to help the program complete. I hypothesized that during the soldering process, some of the solder had traveled into the socket, preventing the ship from fully seating. I reflowed all the solder joints on the RAM, which allowed the RAM chip to fully seat, and the monitor program would run a complete pass without fail, but other bugs were still present.

6. Garbage Output When Printing Strings From String Pointer Array

   This problem occurred when the program would try to print a string from a pointer stored in an array (the help command). Instead of printing the string, garbage data would be printed. The code worked on the simulator, just not on hardware. To debug this, I went first to the assembly, which showed no issues. Then I went to the disassembly, which pointed me in the right direction. With the way my linker is set up, the `data` section is not loaded into RAM on program startup. The array that was storing the string pointers wasn't declared as constant, and therefore was being stored in the `data` section, not the `rodata` section. As such, garbage data was used from uninitialized RAM.

7. Data Bit 1 Intermittent Garbage Data

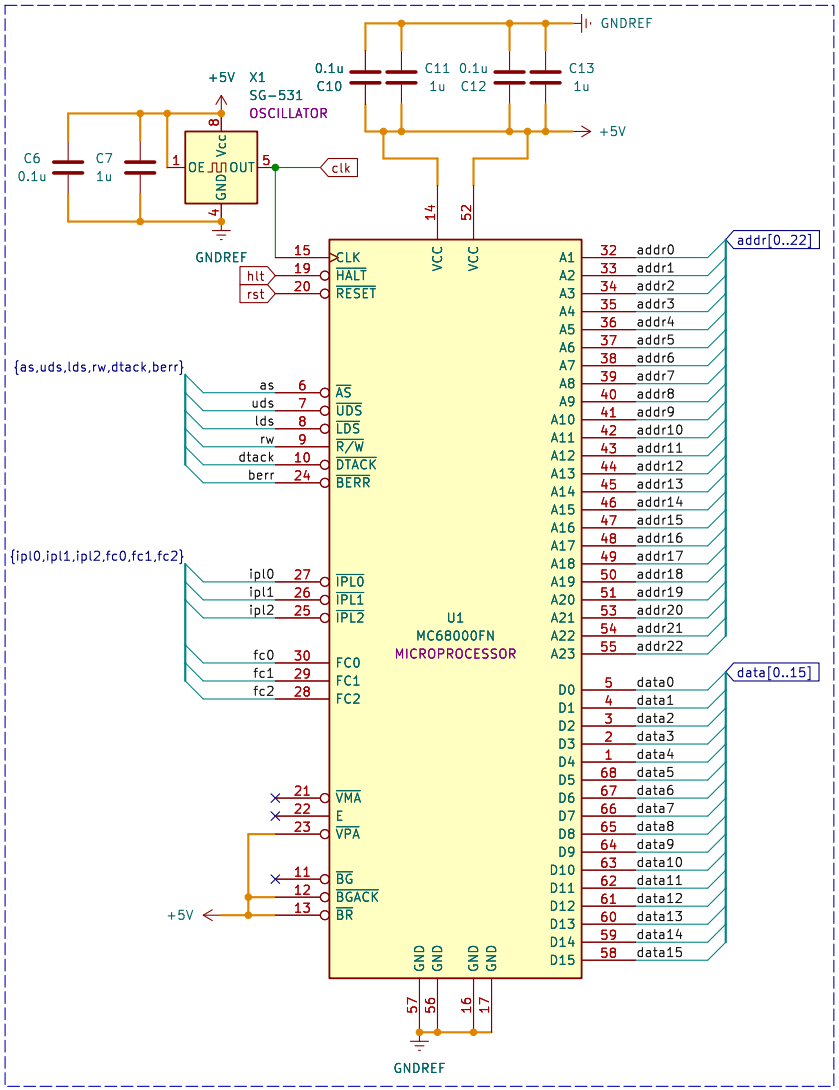
   At this point in time, the monitor program was working well, except for the occasional buss errors. This problem got worse during testing to the point were the program would assert an access error almost as soon as it started. The determining factor, however, was when the program would print its welcome string, but some of the characters were off by small amounts. For example, an 's' appeared as a 'q'. In fact, they were off by a single bit: bit 1. I decided to reflow the ROM, DUART, MPU, and CPLD pins as solder reflowing is not difficult nor too time consuming. This solution fixed the final problem with the computer.

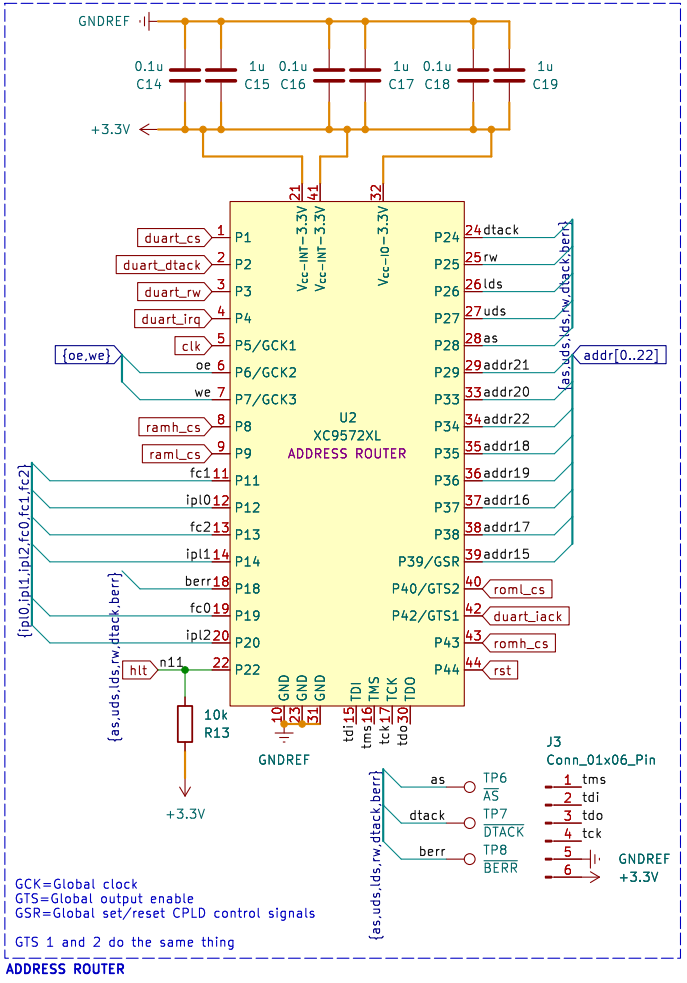
## V. Suggestions

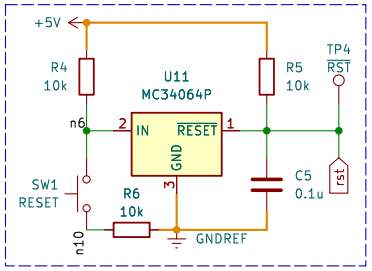
Future students: spend the majority of your time in the beginning of the semester when the workload is light! Do lots of research ahead of time and plan out your board. Tinker with some of the components to increase your confidence in working with them. Set yourself up for success by designing alternatives for risky design decisions. If you can, try to use through hole components. This makes soldering and debugging much easier, and shouldn't be too much more expensive than using surface mount. Don't be afraid to go above and beyond. This is your project so take pride in it. The more you take pride in it, the more likely you are to make a reliable, working product.

VI. Appendices

# A. Schematics



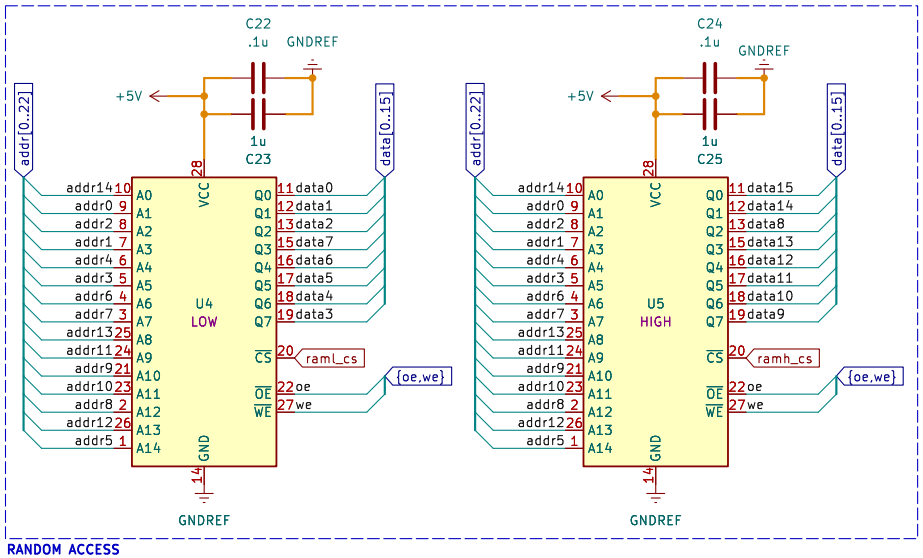




Richard Homan
LeTourneau University

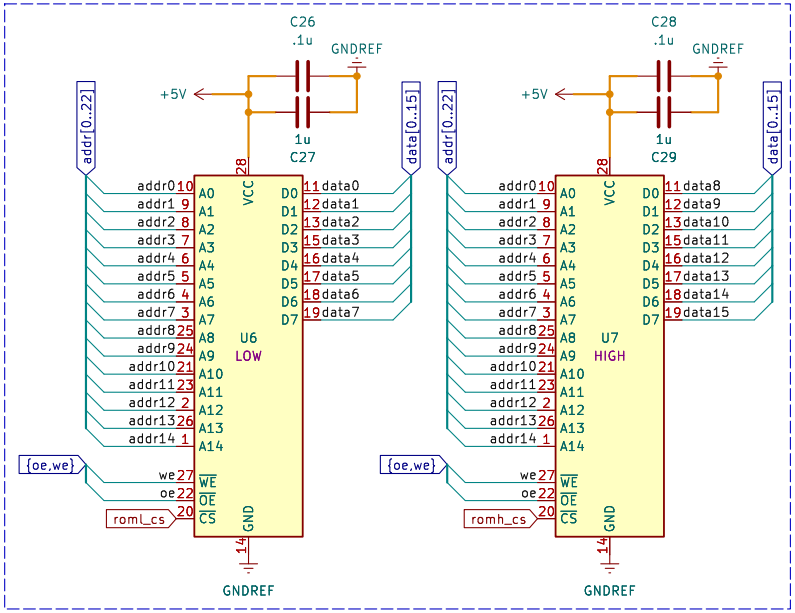
Sheet: /
File: SBC Circuit.kicad_sch

**Title: MICROPROCESSOR & ADDRESS ROUTER**

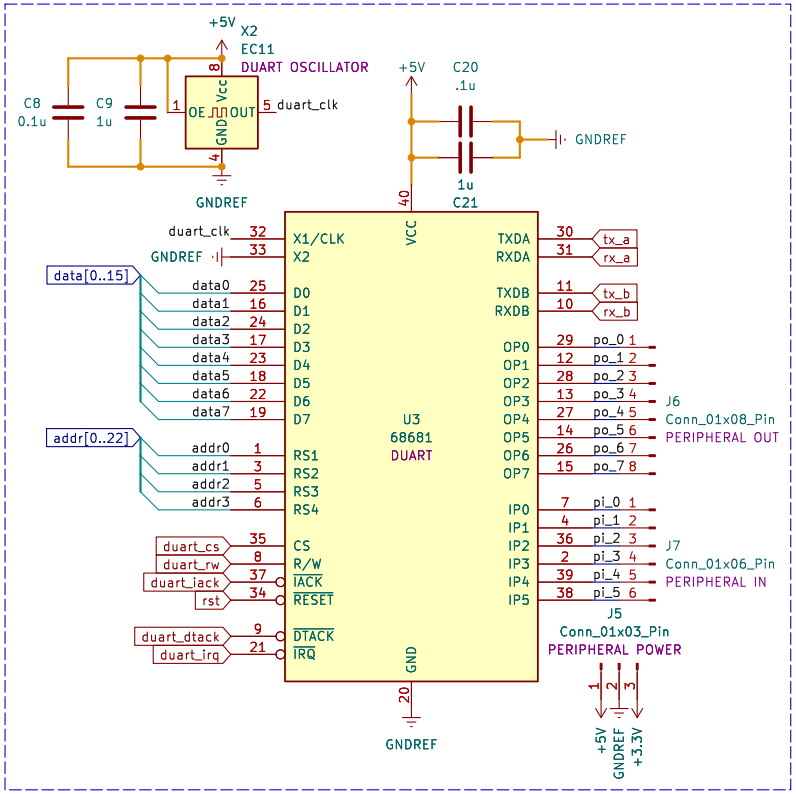
| Size: A4 | Date: 2024-02-08 | Rev: 0.10 |
|---|---|---|
| KiCad E.D.A. kicad 7.0.10 | | Id: 1/3 |




RANDOM ACCESS




READ ONLY

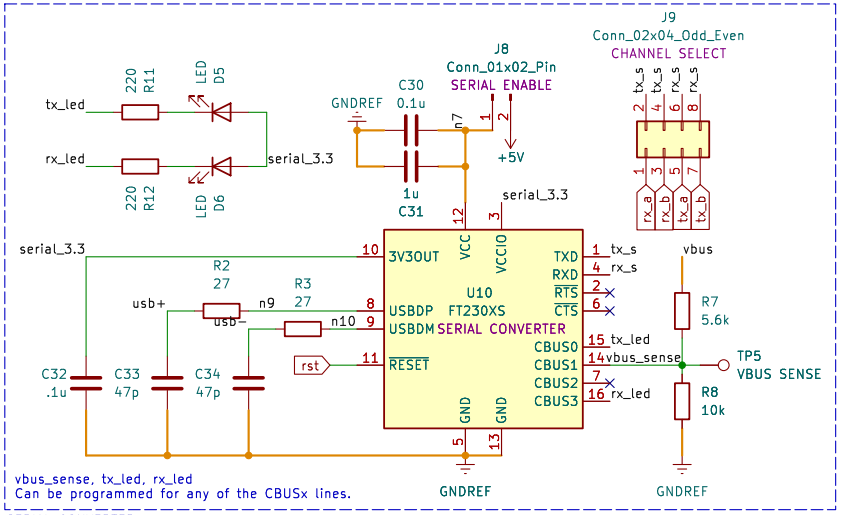



DUART

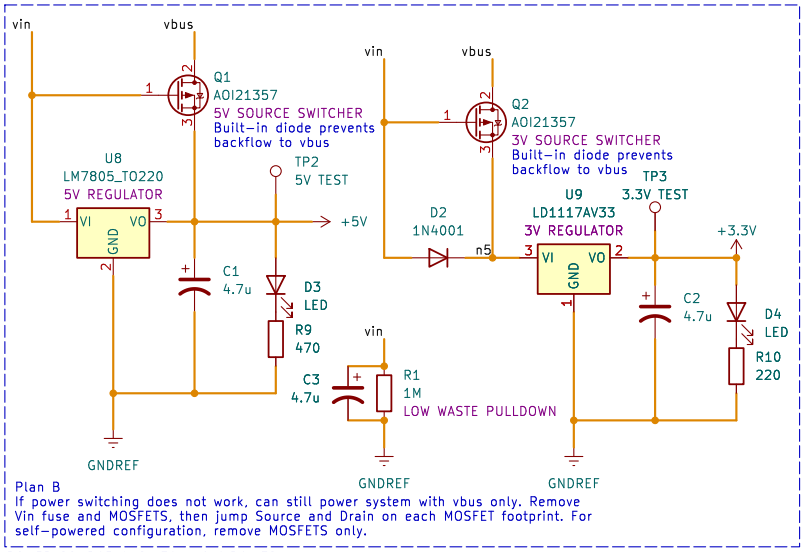
Richard Homan
**LeTourneau University**
Sheet: /Memory+Peripheral/
File: Memory+Peripheral.kicad_sch

***Title: MEMORY & PERIPHERAL***

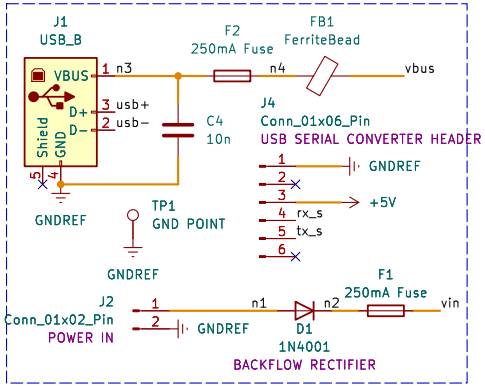
| | | |
|---|---|---|
| Size: A4 | Date: 2024-02-08 | **Rev: 0.10** |
| KiCad E.D.A. kicad 7.0.10 | | Id: 2/3 |




SERIAL CONVERTER




POWER SOURCE & SWITCHING CIRCUITS




EXTERNAL INTERFACE

Richard Homan
LeTourneau University

Sheet: /Power+IO/
File: Power+IO.kicad_sch

**Title: POWER & IO**

| Size: A4 | Date: 2024-02-08 | Rev: 0.10 |
|---|---|---|
| KiCad E.D.A. kicad 7.0.10 | | Id: 3/3 |

# B. Source Code

```c
/*
 * command.h
 * Richard Homan
 * 02/17/2024
 * Command determination for Enemigo Monitor System
 */

#include "err.h"

#ifndef COMMAND_H
#define COMMAND_H

typedef int (*command_func)(const char **argv, int argc) ;

int ems_cr(const char **argv, int argc);
int ems_cm(const char **argv, int argc);
int ems_dr(const char **argv, int argc);
int ems_dm(const char **argv, int argc);
int ems_l (const char **argv, int argc);
int ems_r (const char **argv, int argc);
int ems_h (const char **argv, int argc);

int invalid_command(const char **argv, int argc);

int split_args(char *in, int len, char **argv, int *argc, int argm);
command_func gcommand(const char *commname);

#endif
```

```
/*
 * def_68681.c
 * Richard Homan
 * 02/11/2024
 * Defintions of the 68681 DUART as connected on Commodork Enemigo -64
 */

/*
 * Important info:
 * DUART RS1-4 lines are connected to M68K A1-A4 respectively. DUART D0-D7 lines
 * are connected to M68K D0-D7 respectively. This means that we need to have
 * each register address spaced in multiples of two and have each on odd
 * addresses.
 */

#ifndef DEF_68681_H
#define DEF_68681_H

#define DUART_PTR 0x20000 // location of DUART

// offsets
#define MR1A_OFF (0x0 * 2) + 1
#define MR2A_OFF MR1A_OFF
#define  SRA_OFF (0x1 * 2) + 1
#define CSRA_OFF SRA_OFF
#define  CRA_OFF (0x2 * 2) + 1
#define  RBA_OFF (0x3 * 2) + 1
#define  TBA_OFF RBA_OFF
#define IPCR_OFF (0x4 * 2) + 1
#define  ACR_OFF IPCR_OFF
#define  ISR_OFF (0x5 * 2) + 1
#define  IMR_OFF ISR_OFF
#define  CUR_OFF (0x6 * 2) + 1
#define CTUR_OFF CUR_OFF
#define  CLR_OFF (0x7 * 2) + 1
#define CTLR_OFF CLR_OFF
#define MR1B_OFF (0x8 * 2) + 1
#define MR2B_OFF MR1B_OFF
#define  SRB_OFF (0x9 * 2) + 1
#define CSRB_OFF SRB_OFF
#define  CRB_OFF (0xa * 2) + 1
#define  RBB_OFF (0xb * 2) + 1
#define  TBB_OFF RBB_OFF
#define  IVR_OFF (0xc * 2) + 1
#define  IPR_OFF (0xd * 2) + 1
#define OPCR_OFF IPR_OFF
#define OPRS_OFF (0xe * 2) + 1
#define OPRC_OFF (0xf * 2) + 1

// pointer definitions
#define MR1A_PTR (ubyte * const)(DUART_PTR + MR1A_OFF)
#define MR2A_PTR (ubyte * const)(DUART_PTR + MR2A_OFF)
#define SRA_PTR  (ubyte * const)(DUART_PTR +  SRA_OFF)
#define CSRA_PTR (ubyte * const)(DUART_PTR + CSRA_OFF)
#define CRA_PTR  (ubyte * const)(DUART_PTR +  CRA_OFF)
#define RBA_PTR  (ubyte * const)(DUART_PTR +  RBA_OFF)
#define TBA_PTR  (ubyte * const)(DUART_PTR +  TBA_OFF)
#define IPCR_PTR (ubyte * const)(DUART_PTR + IPCR_OFF)
#define ACR_PTR  (ubyte * const)(DUART_PTR +  ACR_OFF)
#define ISR_PTR  (ubyte * const)(DUART_PTR +  ISR_OFF)
#define IMR_PTR  (ubyte * const)(DUART_PTR +  IMR_OFF)
#define CUR_PTR  (ubyte * const)(DUART_PTR +  CUR_OFF)
#define CTUR_PTR (ubyte * const)(DUART_PTR + CTUR_OFF)
#define CLR_PTR  (ubyte * const)(DUART_PTR +  CLR_OFF)
#define CTLR_PTR (ubyte * const)(DUART_PTR + CTLR_OFF)
#define MR1B_PTR (ubyte * const)(DUART_PTR + MR1B_OFF)
#define MR2B_PTR (ubyte * const)(DUART_PTR + MR1B_OFF)
#define SRB_PTR  (ubyte * const)(DUART_PTR +  SRB_OFF)
#define CSRB_PTR (ubyte * const)(DUART_PTR + CSRB_OFF)
#define CRB_PTR  (ubyte * const)(DUART_PTR +  CRB_OFF)
#define RBB_PTR  (ubyte * const)(DUART_PTR +  RBB_OFF)
#define TBB_PTR  (ubyte * const)(DUART_PTR +  TBB_OFF)
#define IVR_PTR  (ubyte * const)(DUART_PTR +  IVR_OFF)
#define IPR_PTR  (ubyte * const)(DUART_PTR +  IPR_OFF)
#define OPCR_PTR (ubyte * const)(DUART_PTR + OPCR_OFF)
#define OPRS_PTR (ubyte * const)(DUART_PTR + OPRS_OFF)
#define OPRC_PTR (ubyte * const)(DUART_PTR + OPRC_OFF)

#endif
```

```c
/*
 * ems_cm.c
 * Richard Homan
 * 02/17/2024
 * Change memory logic for Enemigo Monitor System
 */

#include "err.h"
#include "types.h"
#include "convert.h"

// argv[1] = address, argv[2] = value
int ems_cm(const char **argv, int argc)
{
          ulword addr;
          byte b, val;
          register int i, r;

          if (argv[1][6] != '\0' || // address not len 6
                    argv[2][2] != '\0')   // value not len 2
                    return EMS_BAD_ARG;

          addr = 0;
          val = 0;
          for (i = 0; i < 6; i+=2)
          {
                    r = ahtob(&(argv[1][i]), &b);
                    if (r != 0)
                              return r;
                    addr = (addr << 8) | b;
          }

          r = ahtob(argv[2], &b);
          if (r != 0)
                    return r;
          val = b;

          *(byte *)addr = val;

          return 0;
}
```

```c
/*
 * ems_cr.c
 * Richard Homan
 * 02/17/2024
 * Change register logic for Enemigo Monitor System
 */

#include "err.h"
#include "types.h"
#include "convert.h"
#include "serial.h"

// argv[1] = register num, argv[2] = value
int ems_cr(const char **argv, int argc)
{
          register int i, r;
          byte b;
          lword val;

          if (argv[1][2] != '\0' || // arg 1 isn't len 2
                    argv[2][8] != '\0')   // arg 2 isn't len 8
                    return EMS_BAD_ARG;

          val = 0;
          for (i = 0; i < 8; i+=2)
          {
                    r = ahtob(&(argv[2][i]), &b);
                    if (r != 0)
                              return r;
                    val = (val << 8) | (ubyte)b;
          }

          if (argv[1][0] == 'd')
          {
                    switch(argv[1][1])
                    {
                    case '0':
                              __asm__ ("move.l    %0,%/d0\n" :: "m" (val) : "cc");
                              break;
                    case '1':
                              __asm__ ("move.l    %0,%/d1\n" :: "m" (val) : "cc");
                              break;
                    case'2':
                              __asm__ ("move.l    %0,%/d2\n" :: "m" (val) : "cc");
                              break;
                    case'3':
                              __asm__ ("move.l    %0,%/d3\n" :: "m" (val) : "cc");
                              break;
                    case '4':
                              __asm__ ("move.l    %0,%/d4\n" :: "m" (val) : "cc");
                              break;
                    case '5':
                              __asm__ ("move.l    %0,%/d5\n" :: "m" (val) : "cc");
                              break;
                    case '6':
                              __asm__ ("move.l    %0,%/d6\n" :: "m" (val) : "cc");
                              break;
                    case '7':
                              __asm__ ("move.l    %0,%/d7\n" :: "m" (val) : "cc");
                              break;
                    default:
                              return -1;
                    }
          }
          else if (argv[1][0] == 'a')
          {
                    switch(argv[1][1])
                    {
                    case '0':
                              __asm__ ("move.l    %0,%/a0\n" :: "m" (val) : "cc");
                              break;
                    case '1':
                              __asm__ ("move.l    %0,%/a1\n" :: "m" (val) : "cc");
                              break;
                    case'2':
                              __asm__ ("move.l    %0,%/a2\n" :: "m" (val) : "cc");
                              break;
                    case'3':
                              __asm__ ("move.l    %0,%/a3\n" :: "m" (val) : "cc");
                              break;
                    case '4':
                              __asm__ ("move.l    %0,%/a4\n" :: "m" (val) : "cc");
                              break;
                    case '5':
                              __asm__ ("move.l    %0,%/a5\n" :: "m" (val) : "cc");
                              break;
                    default:
                              return -1;
                    }
          }
          else
                    return EMS_BAD_ARG;
```

```
        return 0;
}
```

```c
/*
 * ems_dm.c
 * Richard Homan
 * 02/17/2024
 * Dump memory logic for Enemigo Monitor System
 */

#include "err.h"
#include "types.h"
#include "convert.h"
#include "strings.h"
#include "serial.h"

// argv[1] = address, argv[2] = n lines of 16 bytes
int ems_dm(const char **argv, int argc)
{
	char header[9];
	char sbyte[4];
	ulword addr;
	byte b, len;
	register int i, j, r;

	if (argv[1][6] != '\0' ||
		argv[2][2] != '\0')
		return EMS_BAD_ARG;


	addr = 0;
	for (i = 0; i < 6; i+=2)
	{
		r = ahtob(&(argv[1][i]), &b);
		if (r != 0)
			return r;
		addr = (addr << 8) | b;
	}

	r = ahtob(argv[2], &len);
	if (r != 0)
		return r;

	header[6] = ':';
	header[7] = ' ';
	header[8] = '\0';
	sbyte[2] = ' ';
	sbyte[3] = '\0';

	addr -= addr % 16; // align to 0x10
	for (i = 0; i < len; i++)
	{
		btoah((addr >> 16) & 0xff, &(header[0]));
		btoah((addr >>  8) & 0xff, &(header[2]));
		btoah((addr >>  0) & 0xff, &(header[4]));
		serial_puts(header, ARR_LEN(header));

		for (j = 0; j < 16; j++)
		{
			btoah(*(byte *)((ulword)addr), &(sbyte[0]));
			serial_puts(sbyte, ARR_LEN(sbyte));
			addr++;
		}

		serial_puts(nl_str, NL_STR_LEN);
	}

	return 0;
}
```

```c
/*
 * ems_dr.c
 * Richard Homan
 * 02/17/2024
 * Dump registers logic for Enemigo Monitor System
 */

#include "types.h"
#include "convert.h"
#include "strings.h"
#include "serial.h"

const char d0_str[] = "d0";
const char d1_str[] = "d1";
const char d2_str[] = "d2";
const char d3_str[] = "d3";
const char d4_str[] = "d4";
const char d5_str[] = "d5";
const char d6_str[] = "d6";
const char d7_str[] = "d7";
const char a0_str[] = "a0";
const char a1_str[] = "a1";
const char a2_str[] = "a2";
const char a3_str[] = "a3";
const char a4_str[] = "a4";
const char a5_str[] = "a5";
const char fp_str[] = "fp";
const char sp_str[] = "sp";

const char * const reg_str[] =
{
          d0_str,   a0_str,
          d1_str,   a1_str,
          d2_str,   a2_str,
          d3_str,   a3_str,
          d4_str,   a4_str,
          d5_str,   a5_str,
          d6_str,   fp_str,
          d7_str,   sp_str
};

#define GET_REG(r, o) __asm__ ("move.l  %/"#r",%0\n" :"=m" (o) :: "cc");

// no arguments
int ems_dr(const char **argv, int argc)
{
          lword v[16];
          char out[26];
          register int i, j;

          GET_REG(d0, v[ 0]); GET_REG(a0, v[ 1]);
          GET_REG(d1, v[ 2]); GET_REG(a1, v[ 3]);
          GET_REG(d2, v[ 4]); GET_REG(a2, v[ 5]);
          GET_REG(d3, v[ 6]); GET_REG(a3, v[ 7]);
          GET_REG(d4, v[ 8]); GET_REG(a4, v[ 9]);
          GET_REG(d5, v[10]); GET_REG(a5, v[11]);
          GET_REG(d6, v[12]); GET_REG(fp, v[13]);
          GET_REG(d7, v[14]); GET_REG(sp, v[15]);

          out[2] = ':';
          out[3] = ' ';
          out[12] = ' ';
          out[15] = ':';
          out[16] = ' ';
          out[25] = '\0';
          for (i = 0; i < 16; )
          {
                    // set reg names to out string
                    out[0] = reg_str[i][0];
                    out[1] = reg_str[i][1];
                    out[13] = reg_str[i+1][0];
                    out[14] = reg_str[i+1][1];

                    for (j = 6; j >= 0; j-=2)
                    {
                              btoah(v[i] & 0xff, &(out[j+4]));
                              v[i] = v[i] >> 8;
                    }
                    i++;
                    for (j = 6; j >= 0; j-=2)
                    {
                              btoah(v[i] & 0xff, &(out[j+17]));
                              v[i] = v[i] >> 8;
                    }
                    i++;

                    serial_puts(out, ARR_LEN(out));
                    serial_puts(nl_str, NL_STR_LEN);
          }

          return 0;
}
```

```c
/*
 * ems_h.c
 * Richard Homan
 * 02/17/2024
 * Help menu logic for Enemigo Monitor System
 */

#include "types.h"
#include "strings.h"
#include "serial.h"

const char help_str0[] = "cr reg val : change register";
const char help_str1[] = "cm loc val : change memory";
const char help_str2[] = "dr         : dump registers";
const char help_str3[] = "dm loc len : dump memory";
const char help_str4[] = "l          : load srecord executable";
const char help_str5[] = "r          : run srecord executable";
const char help_str6[] = "h          : display this message";

const char * const help_str[] =
{
          help_str0,
          help_str1,
          help_str2,
          help_str3,
          help_str4,
          help_str5,
          help_str6
};

const int help_str_len[] =
{
          ARR_LEN(help_str0),
          ARR_LEN(help_str1),
          ARR_LEN(help_str2),
          ARR_LEN(help_str3),
          ARR_LEN(help_str4),
          ARR_LEN(help_str5),
          ARR_LEN(help_str6)
};

// no arguments
int ems_h(const char **argv, int argc)
{
          register int i;
          for (i = 0; i < ARR_LEN(help_str); i++)
          {
                    serial_puts((const char *)help_str[i], (ubyte)help_str_len[i]);
                    serial_puts(nl_str, NL_STR_LEN);
          }

          return 0;
}
```

```c
/*
 * ems_l.c
 * Richard Homan
 * 02/17/2024
 * Load record logic for Enemigo Monitor System
 */

#include "err.h"
#include "convert.h"
#include "strings.h"
#include "exec.h"
#include "lsrec.h"
#include "serial.h"
#include "prompt.h"
// #include "debug_prog.h"

#define INBUF_SIZE 64
static char inbuf[INBUF_SIZE];

// no arguments
int ems_l(const char **argv, int argc)
{
	int r, inbuf_len;

	lsrec_init();

	has_exec = 0;
	exec_entry_ptr = 0x0;
	// keep accepting input until entry addr is given
	while (exec_entry_ptr == 0x0)
	{
		// no prompt string, cancel on ^C
		inbuf_len = user_prompt(inbuf, INBUF_SIZE, 0x0, 0, '\x03');
		if (inbuf_len == 0)
			return 0;

		inbuf_len--;
		r = lsrec_in(inbuf, inbuf_len, &(exec_entry_ptr));

		if (r != 0)
			return r;
	}

	has_exec = 1;
	// print out entry address, reuse inbuf as output
	btoah(((lword)exec_entry_ptr >>  0) & 0xff, &(inbuf[6]));
	btoah(((lword)exec_entry_ptr >>  8) & 0xff, &(inbuf[4]));
	btoah(((lword)exec_entry_ptr >> 16) & 0xff, &(inbuf[2]));
	btoah(((lword)exec_entry_ptr >> 24) & 0xff, &(inbuf[0]));
	inbuf[8] = '\0';
	serial_puts(inbuf, 8);
	serial_puts(nl_str, NL_STR_LEN);


	// for (int i = 0; i < ARR_LEN(debug_prog_l); i++)
	// {
	//	r = lsrec_in(debug_prog[i], debug_prog_l[i], &(exec_entry_ptr));
	//	if (r != 0)
	//	{
	//		serial_puts(err_str, ERR_STR_LEN);
	//		btoah(r, inbuf);
	//		inbuf[2] = '\0';
	//		serial_puts(inbuf, 2);
	//		serial_puts(nl_str, NL_STR_LEN);
	//		return r;
	//	}

	//	if (exec_entry_ptr != 0x0)
	//	{
	//		has_exec = 1;
	//		return 0; // found entry, exit out of loader
	//	}
	// }

	return 0;
}
```

```c
/*
 * ems_r.c
 * Richard Homan
 * 02/17/2024
 * Run executable logic for Enemigo Monitor System
 */

#include "err.h"
#include "types.h"
#include "convert.h"
#include "strings.h"
#include "exec.h"
#include "serial.h"

const char exit_code_str[] = "exit status: ";

// no arguments
int ems_r(const char **argv, int argc)
{
	char n[9];
	register int i, r;

	if (!has_exec)
		return EMS_NO_EXEC;

	r = exec_entry_ptr(); // run program
	for (i = 6; i >= 0; i-=2)
	{
		btoah(r & 0xff, &(n[i]));
		r = r >> 8;
	}
	n[9] = '\0';
	serial_puts(exit_code_str, ARR_LEN(exit_code_str));
	serial_puts(n, 8);
	serial_puts(nl_str, NL_STR_LEN);

	return 0;
}
```

```c
/*
 * exec.h
 * Richard Homan
 * 02/18/2024
 * Loaded executable shared entrypoint for Enemigo Monitor System
 */

#ifndef EXEC_H
#define EXEC_H

#include "types.h"

extern byte has_exec;
extern int (*exec_entry_ptr)(void);

#endif
```

```
/*
 * isv.s
 * Richard Homan
 * 02/11/2024
 * m68k interrupt vector definitions
 */

          .macro SLOT,n,prefix=,suffix=
          .long __\prefix\n\suffix
          .endm

          .macro ISR n
          SLOT \n,interrupt,
          .endm

          .macro TRAP n
          SLOT \n,trap,
          .endm

          .macro FP n
          SLOT \n,fp_,
          .endm

          .macro UNIMP_OPCODE n
          SLOT \n,unimplemented_,_opcode
          .endm

          .macro BREAKPOINT_DEBUG n
          SLOT \n,,_breakpoint_debug_interrupt
          .endm

          .section  .isv,"a"
          .globl              __interrupt_vector
__interrupt_vector:
          .long __stack                                                 /* 0 */
          .long __reset                                                 /* 1 */
          .long __access_error                              /* 2 */
*         .long __address_error                             /* 3 */
*         .long __illegal_instruction             /* 4 */
*         .long __divide_by_zero                            /* 5 */
*         ISR 6
*         ISR 7
*         .long __privilege_violation             /* 8 */
*         .long __trace                                             /* 9 */
*         UNIMP_OPCODE line_a                               /* 10 */
*         UNIMP_OPCODE line_f                               /* 11 */
*         BREAKPOINT_DEBUG non_pc                           /* 12 */
*         BREAKPOINT_DEBUG pc                               /* 13 */
*         .long __format_error                              /* 14 */
*         .irp N,15,16,17,18,19,20,21,22,23
*         ISR \N                                                                  /* [15,24) */
*         .endr
*         .long __spurious_interrupt              /* 24 */
*         .irp N,25,26,27,28,29,30,31
*         ISR \N                                                                  /* [25,32) */
*         .endr
*         .irp N,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
*         TRAP \N                                                                 /* [32,48) */
*         .endr
*         FP branch_unordered                               /* 48 */
*         FP inexact_result                                 /* 49 */
*         FP divide_by_zero                                 /* 50 */
*         FP underflow                                              /* 51 */
*         FP operand_error                                  /* 52 */
*         FP overflow                                               /* 53 */
*         FP input_not_a_number                             /* 54 */
*         FP input_denormalized_number  /* 55 */
*         .irp N,56,57,58,59,60
*         ISR \N                                                                  /* [56,61) */
*         .endr
*         .long __unsupported_instruction         /* 61 */
*         .irp N,62,63
*         ISR \N                                                                  /* [62,64) */
*         .endr
*         .irp N,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79
*         ISR \N                                                                  /* [64,80) */
*         .endr
*         .irp N,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95
*         ISR \N                                                                  /* [80,96) */
*         .endr
*         .irp N,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111
*         ISR \N                                                                  /* [96,112) */
*         .endr
*         .irp N,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127
*         ISR \N                                                                  /* [112,128) */
*         .endr
*         .irp N,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143
*         ISR \N                                                                  /* [128,144) */
*         .endr
*         .irp N,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159
*         ISR \N                                                                  /* [144,160) */
*         .endr
```

```
*         .irp N,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175
*         ISR \N                                                              /* [160,176) */
*         .endr
*         .irp N,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191
*         ISR \N                                                              /* [176,192) */
*         .endr
*         .irp N,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207
*         ISR \N                                                              /* [192,208) */
*         .endr
*         .irp N,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223
*         ISR \N                                                              /* [208,224) */
*         .endr
*         .irp N,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239
*         ISR \N                                                              /* [224,240) */
*         .endr
*         .irp N,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255
*         ISR \N                                                              /* [240,256) */
*         .endr
*
```

```c
/*
 * lsrec.c
 * Richard Homan
 * 02/12/2024
 * Parser and loader of srec executables for Enemigo Monitor System
 */

#include "lsrec.h"
#include "serial.h"
#include "convert.h"

static uword drec_count;
static ubyte has_entry;
static ulword dbegin_addr;
static ulword dend_addr;
static lword err;

int srec_gtype(const char *, ubyte *); // get type
int srec_gcount(const char *, ubyte *); // get count
int srec_gaddr(const char *, ulword *); // get address
int srec_gdatab(const char *, ubyte, byte *); // gets one byte from data section

void lsrec_init()
{
          drec_count = 0;
          dbegin_addr = LSREC_MAX_ADDR;
          dend_addr = LSREC_MIN_ADDR;
          has_entry = 0;
          err = 0;
}

int lsrec_in(const char *recstr, ubyte len, int (**entry_func)(void))
{
          register int i, r;
          ubyte type, count;
          byte calc_chksum, chksum;
          ulword addr;

          err = 0;
          if (len < 10) // absolute minimum length is 10
          {
                    err = LSREC_ERR_NOT_SREC_FORMAT;
                    return err;
          }

          if (has_entry)  // entry should be the last value given, if we get any more
                          // records, give an error
          {
                    err = LSREC_ERR_DATA_GIVEN_AFTER_ENTRY;
                    return err;
          }

          // get type value
          r = srec_gtype(recstr, &type);
          if (r < 0)
          {
                    err = r;
                    return r;
          }

          switch (type) // don't support S1, S3, S4, S5, S7, S9
          {
          case 0:
                    err = LSREC_SUCCESS;
                    return LSREC_SUCCESS;
          case 1:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S1;
                    break;
          case 3:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S3;
                    break;
          case 4:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S4;
                    break;
          case 5:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S5;
                    break;
          case 7:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S7;
                    break;
          case 9:
                    err = LSREC_ERR_UNSUPPORTED_TYPE_S9;
                    break;
          default:
                    break;
          }
          if (err != 0)
                    return err;

          // get count value
          r = srec_gcount(recstr, &count);
          if (r < 0)
```

```
{
          err = r;
          return r;
}

if (count * 2 != len - 4) // check that actual length matches said count
{
          err = LSREC_ERR_LENGTH_NONEQUAL;
          return err;
}

// get address value
r = srec_gaddr(recstr, &addr);
if (r != 0)
{
          err = r;
          return r;
}

if (type == 6) // verify that we got the number of S2 records specified
                                  // in S6
{
          if (count != 3) // S6 record count should only be 4
          {
                    err = LSREC_ERR_INVALID_FORMAT;
                    return err;
          }

          if (addr != drec_count) // value is contained in address
          {
                    err = LSREC_ERR_RECORD_COUNT_NONEQUAL;
                    return err;
          }

          drec_count = 0;
          return LSREC_SUCCESS;
}
if (type == 8) // entry is specified
{
          if (count != 4) // S8 record count should only be 4
          {
                    err = LSREC_ERR_INVALID_FORMAT;
                    return err;
          }

          if (addr < dbegin_addr || addr > dend_addr)
          {
                    err = LSREC_ERR_ENTRY_OUT_OF_BOUNDS;
                    return err;
          }

          // cast to ulword then to func pointer
          *entry_func = (int (*)(void))addr;
          return LSREC_SUCCESS;
}

drec_count++; // if we got to this point, then we have a data record
 // verify that the address isn't out of bounds
if (addr < LSREC_MIN_ADDR || addr > LSREC_MAX_ADDR)
{
          err = LSREC_ERR_ADDRESS_OUT_OF_BOUNDS;
          return err;
}
// subtract out address and checksum and verify that memory doesn't go out
// of bounds
if (addr + count - 2 - 1 > LSREC_MAX_ADDR)
{
          err = LSREC_ERR_LENGTH_EXTENDS_OUT_OF_BOUNDS;
          return err;
}
// update min and max given values
if (addr < dbegin_addr)
          dbegin_addr = addr;
else if (addr > dend_addr)
          dend_addr = addr;

// initialize calculated checksum
calc_chksum = count +
                                  ((addr >> 16) & 0xff) + // add addr high
                                  ((addr >>  8) & 0xff) + // add addr semi
                                  ((addr >>  0) & 0xff); // add addr low
for (i = 0; i < count-2-1; i++)
{
          byte *baddr;
          byte b;

          r = srec_gdatab(recstr, i, &b);
          if (r != 0)
          {
                    err = r;
                    return r;
          }
```

```
                        calc_chksum += b;

                        baddr = (byte *)(addr + (ulword)i);
                        *baddr = b;
                }

                calc_chksum = 0xff - (calc_chksum & 0xff); // do final calculation
                r = srec_gdatab(recstr, count-4, &chksum);
                if (r != 0)
                {
                        err = r;
                        return r;
                }

                return LSREC_SUCCESS;
}

int srec_gtype(const char *recstr, ubyte *ret)
{
                if ((recstr[0] != 'S') ||
                        !(recstr[1] >= '0' && recstr[1] <= '9'))
                {
                        err = LSREC_ERR_NOT_SREC_FORMAT;
                        return err;
                }

                *ret = (byte)(recstr[1] - '0');
                return 0;
}

int srec_gcount(const char *recstr, ubyte *ret)
{
                byte b;
                int r;

                r = ahtob(&(recstr[2]), &b);
                if (r != 0)
                        return r;

                *ret = (ubyte)b;
                return 0;
}

int srec_gaddr(const char *recstr, ulword *ret)
{
                byte b;
                ulword addr;
                register int i, r;

                addr = 0;
                for (i = 4; i < 10; i+=2)
                {
                        r = ahtob(&(recstr[i]), &b);
                        if (r != 0)
                                return r;
                        addr = (addr << 8) | (ubyte)b;
                }

                *ret = addr;
                return 0;
}

int srec_gdatab(const char *recstr, ubyte byte_num, byte *ret)
{
                const ubyte b_idx = 2+2+6+(byte_num*2);
                byte b;
                register int r;

                r = ahtob(&(recstr[b_idx]), &b);
                if (r != 0)
                        return r;

                *ret = b;
                return 0;
}
```

```c
/*
 * lsrec.h
 * Richard Homan
 * 02/12/2024
 * Parser and loader of srec executables for Enemigo Monitor System
 */

#include "types.h"

#ifndef LSREC_H
#define LSREC_H

#define LSREC_MIN_ADDR        0x10000 // start of ram
#define LSREC_MAX_ADDR        0x20000 // start of duart

#define LSREC_SUCCESS 0
#define LSREC_ERR_NOT_SREC_FORMAT -101 // below minimum length, or doesn't start with 'S'
#define LSREC_ERR_UNSUPPORTED_TYPE_S1 -102
#define LSREC_ERR_UNSUPPORTED_TYPE_S3 -103
#define LSREC_ERR_UNSUPPORTED_TYPE_S4 -104
#define LSREC_ERR_UNSUPPORTED_TYPE_S5 -105
#define LSREC_ERR_UNSUPPORTED_TYPE_S7 -106
#define LSREC_ERR_UNSUPPORTED_TYPE_S9 -107
#define LSREC_ERR_ADDRESS_OUT_OF_BOUNDS -108 // valid address from 0x10000-0x20000
#define LSREC_ERR_LENGTH_NONEQUAL -109 // given srec len does not equal actual len
#define LSREC_ERR_LENGTH_EXTENDS_OUT_OF_BOUNDS -110
#define LSREC_ERR_INVALID_CHECKSUM -111 // checksum invalid
#define LSREC_ERR_RECORD_COUNT_NONEQUAL -112 // the count of S1, S2, and S3 records given in S5 is incongruent to recieved
#define LSREC_ERR_ENTRY_OUT_OF_BOUNDS -113 // entry point specified in S7, S8, or S9 is not defined in the address of the data
#define LSREC_ERR_DATA_GIVEN_AFTER_ENTRY -114 // S1, S2, S3 after S7, S8, S9
#define LSREC_ERR_INVALID_FORMAT -115 // if S5 len is not 3
#define LSREC_ERR_INVALID_HEX -116

// must be called before calling lsrec_in
void lsrec_init();
// line is null terminated, returns the entry function when it gets it
// recieving the entry function means the end of the srec
int lsrec_in(const char *recstr, ubyte len, int (**entry_func)(void));

#endif
```

```c
/*
 * main.c
 * Richard Homan
 * 02/11/2024
 * Entry point for Enemigo Monitor System
 */

#include "types.h"
#include "convert.h"
#include "strings.h"
#include "command.h"
#include "exec.h"
#include "serial.h"
#include "prompt.h"

#define INBUF_SIZE 16
#define ARG_MAX 3

const char parse_err_str[]  = "error parsing arguments";
const char inv_comm_str[]   = "unknown command";
const char bad_arg_str[]    = "bad argument";
const char noexecload_str[] = "no executable loaded";

const char pmt_str[] = "ems% ";

static char inbuf[INBUF_SIZE];
static char *argv[ARG_MAX];

byte has_exec;
int (*exec_entry_ptr)(void);

int main(void)
{
          int r;
          int inbuf_len;
          int argc;
          command_func command;

          has_exec = 0;
          exec_entry_ptr = 0x0;
          serial_init();

          serial_puts(welcome_str, WELCOME_STR_LEN);

mpmt:
          inbuf_len = user_prompt(inbuf, INBUF_SIZE, pmt_str, ARR_LEN(pmt_str), 0);

          r = split_args(inbuf, inbuf_len, argv, &argc, ARG_MAX);
          if (r != 0) // inform about error and return to prompt
          {
                    serial_puts(parse_err_str, ARR_LEN(parse_err_str));
                    serial_puts(nl_str, NL_STR_LEN);
                    goto mpmt;
          }

          if (inbuf_len > 0)
          {
                    command = gcommand(argv[0]);
                    // serial_puts(argv[0], INBUF_SIZE);
                    // goto mpmt;

                    r = command((const char **)argv, argc);
          }
          else
                    r = EMS_INV_COMM;

          switch (r)
          {
          case 0:
                    break;
          case EMS_INV_COMM:
                    serial_puts(inv_comm_str, ARR_LEN(inv_comm_str));
                    break;
          case EMS_BAD_ARG:
                    serial_puts(bad_arg_str, ARR_LEN(bad_arg_str));
                    break;
          case EMS_NO_EXEC:
                    serial_puts(noexecload_str, ARR_LEN(noexecload_str));
                    break;
          default:
                    serial_puts(err_str, ERR_STR_LEN);
                    btoah((byte)(r & 0xff), &(inbuf[0]));
                    serial_puts(inbuf, 2);
                    break;
          }
          if (r != 0)
                    serial_puts(nl_str, NL_STR_LEN);

          goto mpmt; // never exit

          // we should never get to this point, but if we do, loop
          while (1)
```

```
                __asm__ __volatile__ ("nop");
        return 0;
}

__attribute__ ((interrupt_handler))
void __access_error(void)
{
        serial_putc('~');

        while (1) // infinite loop trap
                __asm__ __volatile__ ("nop");
}
```

```c
/*
 * command.c
 * Richard Homan
 * 02/17/2024
 * Command determination for Enemigo Monitor System
 */

#include "command.h"
#include "types.h"
// #include "convert.h"
// #include "serial.h"

// 16 bins
// 10 = cr
// 15 = cm
// 9 = dr
// 14 = dm
// 3 = l
// 13 = r
// 7 = h
const command_func command_vec[16] =
{
          ems_cm,   // 00
          ems_dm,   // 01
          ems_r,    // 02
          invalid_command,    // 03
          invalid_command,    // 04
          ems_cr,   // 05
          ems_dr,   // 06
          invalid_command,    // 07
          ems_h,    // 08
          invalid_command,    // 09
          invalid_command,    // 10
          invalid_command,    // 11
          ems_l,    // 12
          invalid_command,    // 13
          invalid_command,    // 14
          invalid_command,    // 15
};

int invalid_command(const char **argv, int argc) { return EMS_INV_COMM; }

// kind of cheats. since (for this application) we don't care what happens
// to in, this function delimits the input by ' ' and then replaces the spaces
// with '\0'. this way we don't have to copy the strings and can use in
// as the storage for argv
int split_args(char *in, int len, char **argv, int *argc, int argm)
{
          register int i;
          register int next; // flag to indicate that next char is a new arg
          char c;

          *argc = 0;
          next = 1;
          for (i = 0; i < len; i++)
          {
                    if (*argc > argm) // past arg max, don't parse anymore
                              break;
                    c = in[i];
                    if (c == '\0') // found end early, break out
                              break;
                    if (c == ' ')
                    {
                              in[i] = '\0';
                              next = 1;
                    }
                    else if (next)
                    {
                              argv[*argc] = &(in[i]); // set next argument
                              (*argc)++;
                              next = 0;
                    }
          }
          // make sure that last arg is null terminated
          in[len-1] = '\0';

          return 0;
}

// performs the hashing function on the command name to get the command function
// pointer
command_func gcommand(const char *commname)
{
          char c;
          ubyte index;
          int i;
          command_func command;

          command = &invalid_command;
          c = commname[0];
          index = 0;
          i = 0;
```

```
        while (c != '\0')
        {
                index += c;
                i++;
                c = commname[i];
        }

        index = index % 16;

        // char debug_buf[5];
        // btoah(index, debug_buf);
        // debug_buf[2] = '\r';
        // debug_buf[3] = '\n';
        // debug_buf[4] = '\0';
        // serial_puts(debug_buf, 5);

        command = command_vec[index];

        return command;
}
```

```c
/*
 * mem.c
 * Richard Homan
 * 02/15/2024
 * Batch memory functions for Enemigo Monitor System
 */

#include "mem.h"

lword
strncpy (char *__restrict dst0,
          const char *__restrict src0,
          lword count)
{
          char *dscan;
          const char *sscan;
          lword r = count;

          dscan = dst0;
          sscan = src0;
          while (count > 0)
          {
                    --count;
                    if ((*dscan++ = *sscan++) == '\0')
                              break;
          }
          r -= count;
          while (count-- > 0)
                    *dscan++ = '\0';

          return r;
}
```

```c
/*
 * mem.h
 * Richard Homan
 * 02/15/2024
 * Batch memory functions for Enemigo Monitor System
 */

#include "types.h"

#ifndef MEM_H
#define MEM_H

#define STRLEN(s) (sizeof(s)/sizeof(s[0]))

// returns new length of string
lword strncpy(char *dest, const char *src, lword n);

#endif
```

```c
/*
 * prompt.c
 * Richard Homan
 * 04/03/2024
 * String input prompt routine for Enemigo Monitor System
 */

#include "strings.h"
#include "serial.h"
#include "convert.h"

int user_prompt(char *inbuf,         int inbuf_size,
                const char *pmt_str, int pmt_str_len,
                char escc)
{
          int inbuf_len;
          char inchar;

          inbuf_len = 0;
          serial_puts(pmt_str, pmt_str_len);
#ifdef SIM
          serial_puts(cur_str, CUR_STR_LEN);
#endif

waitc:
          while (!serial_isc()) // wait for next character
                    __asm__ __volatile__ ("nop");

          inchar = serial_getc();
          if (inchar == escc)
          {
                    serial_puts(nl_str, NL_STR_LEN);
                    return 0;
          }
          if (inchar == '\r') // on return key
          {
                    inbuf[inbuf_len++] = ' ';
                    inbuf[inbuf_len]   = '\0'; // add null terminator
          #ifndef SIM
                    serial_puts(nl_str, NL_STR_LEN);
          #endif
                    inbuf[inbuf_len] = '\0'; // add null terminator
                    return inbuf_len;
          }
          if (inchar == '\x7f' || inchar == '\b') // on delete or backspace key
          {
                    if (inbuf_len > 0)
                    {
                              inbuf_len--;
                              serial_puts(del_str, DEL_STR_LEN);
                    }
          }
          else // any other input
          {
                    if (inbuf_len < inbuf_size-1) // buffer overflow protection
                    {                             // sub 1 because we add '\0' at end
                              inbuf[inbuf_len] = inchar;
                              inbuf_len++;
                    #ifndef SIM
                              serial_putc(inchar);
                              // btoah(inchar, &(inbuf[0]));
                              // inbuf[2] = '\0';
                              // serial_puts(inbuf, 2);
                              // inbuf_len = 0;
                    #endif
                    }
                    else // prevent overflow
                    {
                    #ifdef SIM
                              serial_putc(del_str, DEL_STR_LEN); // eat char off terminal
                    #endif
                    }
          }
#ifdef SIM
          serial_puts(cur_str, CUR_STR_LEN);
#endif
          goto waitc;
}
```

```c
/*
 * prompt.h
 * Richard Homan
 * 04/03/2024
 * String input prompt routine for Enemigo Monitor System
 */

#ifndef PROMPT_H
#define PROMPT_H

// inbuf: buffer to put users string into
// size : size of input buffer
// escc : optional escape character ('\0' or 0 for none)
// ret  : the length of the buffer written or an error code
int user_prompt(char *inbuf,         int inbuf_size,
                const char *pmt_str, int pmt_str_len,
                char escc);

#endif
```

```c
/*
 * serial.c
 * Richard Homan
 * 02/11/2024
 * Implementations for serial control
 */

#include "serial.h"

#ifndef SIM
#include "def_68681.h"

#define MR1A_VAL (ubyte)0x13
#define MR2A_VAL (ubyte)0x07
// #define MR2A_VAL (ubyte)0x47 // enable automatic echo
#define CSRA_VAL (ubyte)0xbb

#define CRA_VAL0 (ubyte)0x30 // reset transmitter
#define CRA_VAL1 (ubyte)0x20 // reset reciever
#define CRA_VAL2 (ubyte)0x01 // enable reciever
#define CRA_VAL3 (ubyte)0x05 // enable transmitter and reciever

#define ACR_VAL  (ubyte)0x00 // clock set select 1

inline
void serial_init()
{
          *CRA_PTR = CRA_VAL0;
          *CRA_PTR = CRA_VAL1;
          *CRA_PTR = CRA_VAL2;
          *ACR_PTR = ACR_VAL;
          *CSRA_PTR = CSRA_VAL;
          *MR1A_PTR = MR1A_VAL;
          *MR2A_PTR = MR2A_VAL;
          *CRA_PTR = CRA_VAL3;
          return;
}

void serial_puts(const char *in, int len)
{
          register int i;
          char c;

          for (i = 0; i < len; i++)
          {
                    c = in[i];
                    if (c == '\0') break;

                    serial_putc(c);
          }

          return;
}

inline
void serial_putc(char c)
{
          while ((*SRA_PTR & 0x4) == 0) ;
          *TBA_PTR = (ubyte)c;
          return;
}

inline
byte serial_isc()
{
          return (*SRA_PTR & 0x1);
}

inline
char serial_getc()
{
          return *(char *)RBA_PTR;
}

#else
void serial_init()
{
          return;
}

// trap 15, d0=1, a1=string, d1=string len
void serial_puts(const char *in, int len)
{
          __asm__ __volatile__ ("move.l %/a1,-(%/sp)\n" // save registers
                                "move.l %/d1,-(%/sp)\n"
                                "move.l %/d0,-(%/sp)\n"
                                "move.l %0,%/a1\n" // address of string to a1
                                "move.l %1,%/d1\n" // len to d1
                                "move.b #1,%/d0\n" // task 1
                                "trap           #15\n"   // trap 15: SIM tasks
                                "move.l (%/sp)+,%/d0\n" // restore registers
                                "move.l (%/sp)+,%/d1\n"
```

```
                                "move.l (%/sp)+,%/a1\n"
                               :: "rm" (in), "rm" (len) : "cc", "memory");
          return;
}

// trap 15, d0=6, d1=c
void serial_putc(char c)
{
          __asm__ __volatile__ ("move.l %/d1,-(%/sp)\n" // save registers
                                "move.l %/d0,-(%/sp)\n"
                                "move.b %0,%/d1\n" // load c
                                "move.b #6,%/d0\n" // task 6
                                "trap             #15\n"   // trap 15: SIM tasks
                                "move.l (%/sp)+,%/d0\n" // restore registers
                                "move.l (%/sp)+,%/d1\n"
                               :: "rm" (c) : "cc", "memory");
          return;
}

// trap 15, d0=7, d1.b = return
byte serial_isc()
{
          byte r;
          __asm__ __volatile__ ("move.l %/d1,-(%/sp)\n" // save registers
                                "move.l %/d0,-(%/sp)\n"
                                "move.b #7,%/d0\n" // task 7
                                "trap             #15\n"   // trap 15: SIM tasks
                                "move.b %/d1,%0\n" // get return value
                                "move.l (%/sp)+,%/d0\n" // restore registers
                                "move.l (%/sp)+,%/d1\n"
                                : "=m" (r) :: "cc", "memory");
          return r;
}

// trap 15, d0=5, d1.b = return
char serial_getc()
{
          char r;
          __asm__ __volatile__ ("move.l %/d1,-(%/sp)\n" // save registers
                                "move.l %/d0,-(%/sp)\n"
                                "move.b #5,%/d0\n" // task 5
                                "trap             #15\n"   // trap 15: SIM tasks
                                "move.b %/d1,%0\n" // get return value
                                "move.l (%/sp)+,%/d0\n" // restore registers
                                "move.l (%/sp)+,%/d1\n"
                               : "=m" (r) :: "cc", "memory");
          return r;
}

#endif
```

```c
/*
 * serial.h
 * Richard Homan
 * 02/11/2024
 * Declarations for serial control. Use sim make target for EASy68K simulator
 *        target.
 */

#include "types.h"

#ifndef SERIAL_H
#define SERIAL_H

void serial_init();
void serial_puts(const char *, int);
void serial_putc(char);
byte serial_isc();
char serial_getc();

#endif
```

```c
/*
 * strings.c
 * Richard Homan
 * 04/03/2024
 * Common strings for Enemigo Monitor System
 */

#include "strings.h"

const char welcome_str[] =
"Enemigo Monitor System\r\n\
r" REVISION " @ " __TIME__ " " __DATE__ "\r\n\
Designed by Richard Homan\r\n";
const int WELCOME_STR_LEN = ARR_LEN(welcome_str);

const char nl_str[]   = "\r\n";
const int  NL_STR_LEN = ARR_LEN(nl_str);

const char err_str[]   = "error ";
const int  ERR_STR_LEN = ARR_LEN(err_str);

#ifdef SIM
const char cur_str[]   = "_\b";
const int  CUR_STR_LEN = ARR_LEN(cur_str);

const char del_str[]   = " \b \b\b";
const int  DEL_STR_LEN = ARR_LEN(del_str);

#else

const char del_str[]   = "\b \b";
const int  DEL_STR_LEN = ARR_LEN(del_str);

#endif
```

```c
/*
 * strings.h
 * Richard Homan
 * 02/18/2024
 * Common strings for Enemigo Monitor System
 */

#ifndef STRINGS_H
#define STRINGS_H

#define REVISION "0.02"

#define ARR_LEN(a) sizeof(a)/sizeof(a[0])

extern const char welcome_str[];
extern const int  WELCOME_STR_LEN;

extern const char nl_str[];
extern const int  NL_STR_LEN;

extern const char err_str[];
extern const int  ERR_STR_LEN;

extern const char del_str[];
extern const int  DEL_STR_LEN;

#ifdef SIM
extern const char cur_str[];
extern const int  CUR_STR_LEN;
#endif

#endif
```

```c
/*
 * types.h
 * Richard Homan
 * 02/11/2024
 * Refer to units as byte, word, and longword for m68k language
 */

#ifndef TYPES_H
#define TYPES_H

typedef char byte;
typedef short word;
typedef int lword;

typedef unsigned char ubyte;
typedef unsigned short uword;
typedef unsigned int ulword;

#endif
```

```
/*
 * ems.ld
 * Richard Homan
 * 02/11/2024
 * Linker script for Enemigo Monitor System
 */

ENTRY(main)

MEMORY
{
          rom       (rx)      :         ORIGIN = 0x00000, LENGTH = 0x10000
          ram       (rwx)     :         ORIGIN = 0x10000, LENGTH = 0x10000
}

/* reset vector points to main */
PROVIDE(__reset = main);
/* stack begins at bottom of ram */
PROVIDE(__stack = 0x10000 + 0x10000);

/*
 * text & data in ROM
 * stack in RAM
 */
SECTIONS
{
          .text :
          {
                    *(.isv);
                    FILL(0xFF)
                    . = 0x400; /* incase isv is not defined, seek */
                    *(.text);
          }> rom

          .rodata :
          {
                    . = ALIGN(0x10); /* easy to read alignment */
                    *(.rodata);
          }> rom

          .data :
          {
                    . = ALIGN(0x10); /* easy to read alignment */
                    *(.data);
          }> ram AT> rom

          /* There shouldn't exist a bss section anyway but just in case */
          .bss (NOLOAD) :
          {
                    . = ALIGN(0x10); /* easy to read alignment */
                    *(.bss);
          }> ram
}
```

```
#makefile
PROG_NAME=ems

PROG_CSRCS = isr.c strings.c prompt.c convert.c serial.c command.c main.c \
             lsrec.c ems_cr.c ems_cm.c ems_dr.c ems_dm.c ems_l.c ems_r.c ems_h.c
PROG_SSRCS= isv.s
PROG_COBJS= $(patsubst %.c,%.o,$(PROG_CSRCS))
PROG_SOBJS= $(patsubst %.s,%.o,$(PROG_SSRCS))
PROG_LDSRPT= ems.ld

SIM_CSRCS = strings.c prompt.c convert.c serial.c command.c main.c \
            lsrec.c ems_cr.c ems_cm.c ems_dr.c ems_dm.c ems_l.c ems_r.c ems_h.c
SIM_SSRCS=
SIM_COBJS= $(patsubst %.c,%.sim.o,$(SIM_CSRCS))
SIM_SOBJS= $(patsubst %.s,%.sim.o,$(SIM_SSRCS))
SIM_LDSRPT= ems-sim.ld

CC=m68k-elf-gcc
CCFLAGS=-Wall -nostdlib -nodefaultlibs -m68000
AS=m68k-elf-as
ASFLAGS=
LD=m68k-elf-ld
#LD_OPTS=--defsym=_start=main -Ttext=0x2000 -Tdata=0x3000 -Tbss=0x4000 --section-start=.rodata=0x5000
LDFLAGS=
OBJCOPY=m68k-elf-objcopy
OBJCOPYFLAGS=-I elf32-m68k -O srec --srec-len=16

all: stamp_gen $(PROG_COBJS) $(PROG_SOBJS)
          $(LD) $(LDFLAGS) -T ${PROG_LDSRPT} $(PROG_COBJS) $(PROG_SOBJS) -o $(PROG_NAME).out
          $(OBJCOPY) $(OBJCOPYFLAGS) $(PROG_NAME).out $(PROG_NAME).srec

sim: CCFLAGS += -DSIM
sim: stamp_gen $(SIM_COBJS) $(SIM_SOBJS)
          $(LD) $(LDFLAGS) -T ${SIM_LDSRPT} $(SIM_COBJS) $(SIM_SOBJS) -o $(PROG_NAME).sim.out
          $(OBJCOPY) $(OBJCOPYFLAGS) $(PROG_NAME).sim.out $(PROG_NAME).sim.S68

$(PROG_COBJS): %.o: %.c
          $(CC) $(CCFLAGS) -c $< -o $@

$(PROG_SOBJS): %.o: %.s
          $(AS) $(ASFLAGS) -o $@ $<

$(SIM_COBJS): %.sim.o: %.c
          $(CC) $(CCFLAGS) -c $< -o $@

$(SIM_SOBJS): %.sim.o: %.s
          $(AS) $(ASFLAGS) -o $@ $<

.phony stamp_gen:
          touch strings.c

# generate assembly source for c file
%.s: %.c
          $(CC) $(CCFLAGS) -S $*.c -o $*.c.s

clean:
          rm -rf *.o *.out *.S68 *.srec *.c.s
```