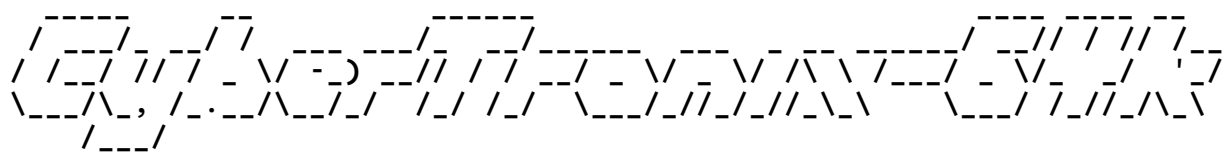
CYBERTRONIX



10 June, 1977

TABLE OF CONTENTS

Address
A 16-bit number corresponding to a specific place in memory.

Memory
All 64k of memory addressable by the CPU.

Register Memory
Any memory within the first 4k of memory addressable by the CPU.

Immediate
A specific number, not referring to a memory location.

Bit
The smallest unit of information which can be represented. Either 0 or 1.

Byte
A group of 8 contiguous bits occupying a single memory location.

Word
A group of 2 contiguous bytes, or 16 contiguous bits. The smallest addressable unit.

Instruction
A single operation that the computer can do.

Base Instruction

Program
A sequence of instructions which, as a group, allow the computer to accomplish a task.

Stack
A block of memory for pushing and popping data. By convention, from 300h to 1000h.

Instruction Pointer
The register address which holds the address of the current instruction. Always at memory address 0h.

Stack Pointer
The register address which holds the current address of the top of the stack. By convention, always at memory address 1h.

Base Pointer
A register address which holds the functions original stack pointer.

Scratch Constant
A register which is used as scratch for when one needs to do arithmetic.

0bNNNN
NNNN represents a number in binary format

0dNNNN
NNNN represents a number in decimal format

0oNNNN
NNNN represents a number in octal format

0xNNNN
NNNN represents a number in hexadecimal format

This manual has been written to help the reader program the CYBERTRONIX CyberTronix64k microcomputer in assembler language. Accordingly, this manual assumes that the reader has a good understanding of logic, but may be completely unfamiliar with programming concepts.

For those readers who do understand programming concepts, several features of the CYBERTRONIX Cyber-Tronix64k microcomputer are described below. They include:

- 16-bit CPU on a single chip
- 16 instructions, for simplicity and optimization purposes
- Direct addressing for 4096 words of memory

There are two ways in which programs for the 64k may be assembled: either via the resident assembler or the cross assembler. The resident assembler is a program which runs on the 64k, and may be loaded. The cross assembler runs on any computer having a B compiler whose word size is 16 bits or greater, and generates programs which run on the 64k.

The experienced programmer should note that the assembly language has a macro capability which allows users to tailor the assembly language to individual need.

This section provides the programmer with a functional overview of the 64k. Information is presented in this section at a level that provides a programmer with necessary background in order to write efficient programs.

To the programmer, the computer is represented as consisting of the following parts:

- The first 4k of memory which provide one means for addressing memory. These are similar to the “registers” of other architectures, and therefore are called “register memory”. Peripherals are also mapped here.
- The rest of memory, which may hold program instructions or data.
- The instruction pointer, at memory address 0x0.
- The stack pointer, at memory address 0x1, which enables parts of memory to be used as a stack. At bootup, the stack pointer is initialized to 0x300, and goes up towards 0x1000.
- Input/Output, which is the interface between a programmer and the outside world. Located between 0x100 and 0x300. Two very useful memory locations in the default layout is 0x200 and 0x201, which are used as a character by character output and input, respectively.

LAYOUT OF THE FIRST 4K

The 64k has a very different architecture from many of the CPUs you may be used to; unlike many architectures, the 64k is designed to rely on memory first. The first 4k of memory, or 0x1000, is special. The first 0x100 is designed to be used as normal register memory. 0x0 is the instruction pointer, which is, as written above, initialized to 0x1000. There is no way to change where the instruction pointer is; unlike any other pseudo-register, the IP is hard-coded into the CPU. By convention, the stack pointer is next, at 0x1; at bootup, it is initialized to 0x300. The next is the base pointer, which is at 0x2, and also initialized to 0x300; and finally, assembler scratch, “scN”, the four registers from 0x3 to 0x7, which are not initialized, and intended to be used for pseudo-ops. After these, according to convention, 0x10 to 0x40 is “callee save” register memory, known as “rNN” in the assembly, which will not change from a function call. From 0x40 to 0x100, is “scratch” register memory, or “sNN” in the assembler, meaning a call can change them (also known as “caller save”). From 0x100 to 0x300 are peripherals; the only addresses used in the default configuration are 0x200 and 0x201

From 0x300 to 0x1000 is the stack, which grows towards high addresses.

AFTER THE FIRST 4k

Program code is often copied from a ROM disk attached to the CPU, into address 0x1000 on startup, where the CPU then starts executing.

There are 16 base instructions used by the 64k. Each starts with the same header; a 4-bit opcode, and then a 12-bit operand. From there depends on the specific opcode. These base instructions are identified by the two-letter op name. There are also many pseudo-instructions, made by putting base instructions together.

A note on layout: [xxx] is a single word. “RM” stands for “Register Memory”, or 12 bits of memory address, while “MEM” stands for a full 16-bit memory address. “IMM” stands for a literal 16-bit word. The explanation uses a pseudo-C language in order to show what the instruction does.

A note on overlong shifts: the behavior is as if the number had been masked with 0x10; in other words, it wraps.

BASE INSTRUCTIONS:

```
mi RM,IMM
      *RM = IMM;

      [0x0|RM][IMM]

mv RM,MEM
      *RM = *MEM;

      [0x1|RM][MEM]

md RM,MEM
      *RM = **MEM;

      [0x2|RM][MEM]

ld RM,MEM
      **RM = *MEM;

      [0x3|RM][MEM]

st RM,MEM
      **MEM = *RM;

      [0x4|RM][MEM]

ad RM,MEM
      *RM += *MEM;

      [0x5|RM][MEM]

sb RM,MEM
      *RM -= *MEM;

      [0x6|RM][MEM]
```

```
nd RM,MEM
      *RM &= *MEM;

      [0x7|RM][MEM]

or RM,MEM
      *RM |= *MEM;

      [0x8|RM][MEM]

xr RM,MEM
      *RM ^= *MEM;

      [0x9|RM][MEM]

sr RM,MEM
      (unsigned)
      *RM >>= *MEM;

      [0xA|RM][MEM]

sl RM,MEM
      *RM <<= *MEM;

      [0xB|RM][MEM]

sa RM,MEM
      (signed)
      *RM >>= *MEM;

      [0xC|RM][MEM]

jg RM,MEM,LABEL
      if (*RM > *MEM) goto LABEL;

      [0xD|RM][MEM][IMM]

jl RM,MEM,LABEL
      if (*RM < *MEM) goto LABEL;

      [0xE|RM][MEM][IMM]

jq RM,MEM,LABEL
      if (*RM == *MEM) goto LABEL;

      [0xF|RM][MEM][IMM]

hf
      Halt and catch fire

      [0xF|0x0][0x0][*ip]
```

PSEUDO-INSTRUCTIONS:

```
ji LABEL
      goto LABEL;

      mi ip, LABEL

jm MEM
      goto *MEM;

      mv ip, MEM

inc RM
      *RM += 1;

      mi sc0, 1
      ad RM, sc00

dec RM
      *RM -= 1;

      mi sc0, 1
      sb RM, sc0

neg RM
      *RM = -*RM;

      mv sc0, RM
      mi RM, 0
      sb RM, sc0

adi RM, IMM
      *RM += IMM;

      mi sc0, IMM
      ad RM, sc0

sbi RM, IMM
      *RM -= IMM;

      mi sc0, IMM
      sb RM, sc0

push MEM
      push(*MEM);

      mi sc0, 1
      ad sp, sc0
      ld sp, MEM
```

```
pop RM
      *RM = pop();

      md RM, sp
      mi sc0, 1
      sb sp, sc0

call LABEL
      LABEL();

      mi sc0, 1
      ad sp, sc0
      mi sc0, $ + 6
      ld sp, sc0
      ji LABEL

ret
      return;

      md sc1, sp
      mi sc0, 1
      sb sp, sc0
      jm sc1
---
```

All “sNN” registers are used as arguments. Any additional arguments beyond 96 will be pushed onto the stack. The return address will be pushed onto the stack after any arguments.

Pointers and integers will be passed as normal, in the argument registers. Any structural types which are 4 words or smaller will be passed in the registers, split out into however many are required. Anything bigger than 4 words will be passed as a pointer.

Returns of pointers and integers will be in 0x40. Any structural types with size less than 4 words will be split among 0x40, 0x41, 0x42, and 0x43 as necessary. Anything with a size greater than 4, the calling function must allocate enough space to store the return argument, and an implicit pointer to this space will be passed as the first parameter.

All register memory from 0x40 to 0x100, may be changed inside the function. All registers from 0x10 to 0x40 must be saved and restored if they are changed.

sp and bp must be restored on function return.