/ Messages (https://groups.io/g/TekScopes/messages?msgnum=194319)
/ MC6800 disassembler available for Ghidra

Single

Mute This Topic (https://groups.io/g/TekScopes/ft/91988908/194319?csrf=5513314409256117711&mute=1&p=%2C%2C%2C20%2C0%2C0%2C0)

# MC6800 disassembler available for Ghidra

Siggi

Jun 25 (https://groups.io/g/TekScopes/message/194319)

Hey y'all,

TL;DR: you can now disassemble and successfully reverse MC6800 with Ghidra by downloading this ( https://github.com/sigurasg/ghidra/tree/6800-specs-add/Ghidra/Processors/MC6800 (https://github.com/sigurasg/ghidra/tree/6800-specs-add/Ghidra/Processors/MC6800)) language spec.

Some time before Christmas I'd set about reverse engineering the ROMs in the 2465 as a project. I started by downloading Ghidra, and running the 6805 disassembler on the ROMs. This didn't work at all, as it turns out the 6805 is a VERY different architecture and instruction set than the MC6800 (6802/6808 et al).
After discovering this, I set about writing a processor "language spec" for the MC6800 for Ghidra, which was aided a fair bit by the existence of a MC6809 processor spec. The results from using my spec were quite disappointing, so I shifted to writing a MAME emulator for the 2465 instead.

Last week I was contacted by someone who's been trying to use my MC6800 spec to reverse some pinball machines, complaining about how I was handling the JMP/JSR instructions. While looking at the patch they sent me, I realized that there was a big gaffe in my language spec. Instead of handling the operands to JSR/JMP as effective addresses, I'd been dereferencing them.
After fixing this gaffe, Ghidra works like magic for me. If you've ever reverse-engineered firmware before, but haven't used Ghidra (or IDA Pro) before, you'll be amazed - guaranteed. The big trick those tools have is "decompilation" where the assembly code is summarized into C-like code, which makes it almost humane to read.

As a case in point, here's the "decompiled" reset routine from the 2465 after spending about 15 minutes to set up a memory map with suitable definitions for the scope's IO registers and massaging the RAM test portion of it for better results.
From this code it's reasonably easy to see what's initialized on RESET, how the RAM is tested and so on.

--- cut here ---
/* WARNING: Switch with 1 destination removed at 0xff37 */
/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

```
void RESET_SERVICE(void)
{
  bool bVar1;
  byte bits_to_set;
  byte bits_expected;
  undefined uVar2;
  byte bit_count;
  byte ros_2_data;
  byte *ram_ptr;
  byte *ram_ptr2;
  byte *ram_ptr3;
  bool last_bit;

  bit_count = 0x38;
  write_volatile_1(IORegion_0800.io.PORT2_CLK[0],0x21);
  do {
    read_volatile(IORegion_0800.io.fine[0].LED_CLK);
    read_volatile(IORegion_0800.io.fine[0].DISP_SEQ_CLK);
    bit_count = bit_count - 1;
  } while (bit_count != 0);
  ros_2_data = 0xbf;
  do {
    write_volatile_1(IORegion_0800.io.ROS_2_CLK[0],ros_2_data);
    ros_2_data = ros_2_data >> 1;
  } while (ros_2_data != 0);
  read_volatile(IORegion_0800.io.ROS_1_CLK[0]);
  read_volatile(IORegion_0800.io.ROS_1_CLK[0]);
  do {
    SMALL_RAMTOP = 0;
    _DAT_07fe = CONCAT11(DAT_07fe,0xff);
    ram_ptr = (byte *)&IORegion_0800;
    do {
      ram_ptr = ram_ptr + -1;
      *ram_ptr = 0;
    } while (ram_ptr != (byte *)0x0);
    ram_ptr2 = (byte *)&IORegion_0800;
    do {
      ram_ptr2 = ram_ptr2 + -1;
      if (*ram_ptr2 != 0) {
RAM_BAD:
                    /* WARNING: Subroutine does not return */
        KERNEL_TEST_FAILURE(0);
      }
      bits_to_set = 0x80;
      do {
        *ram_ptr2 = bits_to_set;
        if (bits_to_set != *ram_ptr2) goto RAM_BAD;
        last_bit = (bool)(bits_to_set & 1);
        bits_to_set = bits_to_set & 0x80 | (char)bits_to_set >> 1;
      } while (last_bit == false);
    } while (ram_ptr2 != (byte *)0x0);
    ram_ptr3 = (byte *)&IORegion_0800;
    do {
      ram_ptr3 = ram_ptr3 + -1;
      bits_expected = 0xff;
```

```
      do {
        if (bits_expected != *ram_ptr3) goto RAM_BAD;
        *ram_ptr3 = *ram_ptr3 >> 1;
        bVar1 = (bool)(bits_expected & 1);
        bits_expected = bits_expected >> 1;
      } while (bVar1);
    } while (ram_ptr3 != (byte *)0x0);
    _DAT_07fe = 0xfca8;
    uVar2 = FUN_f35f();
    if (DAT_01e2 != '\0') {
                    /* WARNING: Subroutine does not return */
      KERNEL_TEST_FAILURE(uVar2);
    }
    if ((byte)~DAT_1004 == DAT_1005) {
      _DAT_07fe = 0xfcbf;
      FUN_f35f();
      if (DAT_01e2 != '\0') {
        if (DAT_01e2 != -0x7d) {
LAB_fcd9:
                    /* WARNING: Subroutine does not return */
          KERNEL_TEST_FAILURE(0xf1);
        }
        DAT_01e2 = '\0';
        _DAT_07fe = 0xfcd1;
        FUN_f39c();
        if (DAT_0157 != _DAT_1007) goto LAB_fcd9;
      }
      _DAT_07fe = 0xfce1;
      uVar2 = FUN_100f();
      if (DAT_01e2 != '\0') {
                    /* WARNING: Subroutine does not return */
        KERNEL_TEST_FAILURE(uVar2);
      }
      DAT_7fff = 0x40;
      if (DAT_400f == '@') {
        DAT_7f87 = 1;
        _DAT_07fe = 0xfcfd;
        FUN_ff39();
      }
      DAT_7fff = 0x80;
      if (DAT_400f == -0x80) {
        DAT_7f94 = DAT_7f90 | 4;
        DAT_7f96 = 2;
        _DAT_07fe = 0xfd1e;
        FUN_ff39();
      }
    }
    _DAT_07fe = 0xfd23;
    FUN_c570();
    _DAT_07fe = 0xfd26;
    FUN_c8f3();
    _DAT_07fe = 0xfd2b;
    FUN_c570();
    _DAT_07fe = 0xfd30;
    FUN_c570();
```

```
    DAT_0090 = 0xbf;
  } while( true );
}
```

--- cut here ---

I'm trying to get this spec merged into the Ghidra distribution, but alas without luck so far.

Siggi

Reply Like More

(https://groups.io/g/TekScopes/message/194318) #194319 (https://groups.io/g/TekScopes/message/194320)