ME 333 Homework 3

Marshall Johnson

January 25, 2022

Chapter 3 Exercises

1) **Convert the following virtual addresses to physical addresses, and indicate whether the address is cacheable or not, and whether it resides in RAM, flash, SFRs, or boot flash.**

   a. **0x80000020** — 0x00000020 — Cacheable — RAM
   b. **0xA0000020** — 0x00000020 — Not Cacheable — RAM
   c. **0xBF800001** — 0x0F800001 — Not Cacheable — SFRs
   d. **0x9FC00111** — 0x1FC00111 — Cacheable — Boot Flash
   e. **0x9D001000** — 0x1D001000 — Cacheable — Program Flash

3) **Refer to the Memory Organization section of the Data Sheet and Figure 2.1.**

   a. **Referring to the Data Sheet, indicate which bits, 0-31, can be used as input/outputs for each of Ports B through G. For the PIC32MX795F512H in Figure 2.1, indicate which pin corresponds to bit 0 of port E (this is referred to as RE0).**

      - Port B: 0-15
      - Port C: 12-15
      - Port D: 0-11
      - Port E: 0-7
      - Port F: 0-1; 3-5
      - Port G: 2-3; 6-9
      - Bit 0 of Port E = Pin 60

b. **The SFR INTCON refers to "interrupt control." Which bits, 0-31, of this SFR are unimplemented? Of the bits that are implemented, give the numbers of the bits and their names.**

- Unimplemented: 5-7; 11; 13-15; 17-31
- Implemented:
  - 0-4: INT0EP/INT1EP/INT2EP/INT3EP/INT4EP
  - 8-10: TPC<2:0>
  - 12: MVEC
  - 16: SS0

7) **The processor.o file linked with your simplePIC project is much larger than your final .hex file. Explain how that is possible.**

The processor.o file contains vitrual memory addresses for the PIC32's SFRs. This is a lot of information since multiple VAs can map to the same PA. After the linker takes object files and outputs a large .elf file, the final step is to create a stripped down .hex file, which is significantly smaller.

8) **The building of a typical PIC32 program makes use of a number of files in the XC32 compiler distribution. Let us look at a few of them.**

a. **Look at the assembly startup code pic32-libs/libpic32/startup/crt0.S. Although we are not studying assembly code, the comments help you understand what the startup code does. Based on the comments, you can see that this code clears the RAM addresses where uninitialized global variables are stored, for example. Find and list the line(s) of code that call the user's main function when the C runtime startup completes.**
Lines 522-526

b. **Using the command xc32-nm -n processor.o, give the names and addresses of the five SFRs with the highest addresses.**

- bf88cb4c A C2FIFOCI31INV
- bfc02ff0 A DEVCFG3
- bfc02ff4 A DEVCFG2
- bfc02ff8 A DEVCFG1
- bfc02ffc A DEVCFG0

c. **Open the file p32mx795f512h.h and go to the declaration of the SFR SPI2STAT and its associated bit feld data type __SPI2STATbits_t. How many bit fields are defined? What are their names and sizes? Do these coincide with the Data Sheet?**

10 bit fields defined (not including the empty fields):

Name:Size

- SPIRBF:1
- SPITBF:1
- SPITBE:1
- SPIRBE:1
- SPIROV:1
- SRMT:1
- SPITUR:1
- SPIBUSY:1
- TXBUFELM:5
- RXBUFELM:5

These coincide with the Data Sheet.

9) **Give three C commands, using TRISDSET, TRISDCLR, and TRISDINV, that set bits 2 and 3 of TRISD to 1, clear bits 1 and 5, and flip bits 0 and 4.**

```
TRISDSET = 0b1100;

TRISDCLR = 0b100010;

TRISDINV = 0b10001;
```