18-240: Structure and Design of Digital Systems

Electrical & Computer ENGINEERING

HW8 [4 problems, 64 points]

Covers lectures L16 – L18 *Due: 20 November 2023*

Homework sets are due at 5:00PM on the due date. Upload your answers, to Gradescope by then. No late homework will be accepted. Remember, we let you drop two homework assignments over the semester.

Important: You will use **`handin240`** to submit the assembly language files for this homework, as you have for most other homeworks this semester. However, **`handin240`** does not run these files through the assembler and does not emit error messages should it fail to assemble properly. You should be using **`as240`** (and **`sim240`**) on your own to test your work. If your file fails to assemble correctly, you will still get a zero, even though **`handin240`** does not report the error.

Discussions about homework in small groups are encouraged — think of this as giving hints, not solutions, to each other. However, homework must be written up individually (no copying is allowed). If you discussed your homework solutions with someone else, either as the giver or receiver of information, your write-up must explicitly identify the individuals and the manner information was shared.

You must show details of your work. There is no credit for just writing down an answer.

Non-Drill Problems [64 points]

There are no drill problems this week.

1. [18 points, Lecture 17] Complete the RISC240 Assembler and Simulator Tutorial, and read the RISC240 Reference Manual. Both are available on Canvas under "Supplemental Handouts → RISC240".

   In Step 3 of the tutorial, when you are filling out the table, you are asked for **`MAR`** and **`MDR`** values. Those are registers that you will learn about soon. Nevertheless, copy the values from the simulator to the table.

   From the tutorial, you will need to turn in the following:

   - [6 points] Your hand-assembled binary memory listing from Step 1. It should be obvious from your submission that you have done the work to assemble this by hand.

     ▷ **Submit your hand-assembled binary memory listing in your PDF.**

   - [6 points] The table from Step 4, along with a screenshot of the simulator after execution of the **`BRZ done`** instruction (the first time it is encountered).

     ▷ **Submit the table and the screenshot in your PDF.**

   - [6 points] Three assembly files for programs that you generated in Step 5. Make sure they include a comment at the beginning of the file, describing what they do.

     ▷ **Submit the three assembly files as three individual files, named `hw8prob1a.asm`, `hw8prob1b.asm`, and `hw8prob1c.asm`.**

2. [8 points, Lecture 18] Take a look at the datapath diagram from the end of Lecture 18's slides. Answer the following questions about that diagram.

   (a) The decoder in the upper left, as you know, converts the binary value **dest[2:0]** into a one-hot signal. Where does the **dest[2:0]** signal come from? Hint: It is shown in green, which has a particular meaning on this diagram.

   (b) What is the *purpose* of the decoder in this datapath?

   (c) If I wanted to add two values together, each of which is in the register file, and put the result in the Memory Address Register, what values would I have to provide for each of the control points?

   (d) What does it indicate that the **condCodes** are colored purple?

3. [20 points, Lecture 17] Consider this RISC240 assembly language program:

```
        .ORG $1234
A       .DW  $1
        .DW  $5
        .DW  $FFFE
        .DW  $7FFF
        .DW  $1234
        .DW  $0
B       .DW  $12

        .ORG $1000
        LI r3, A
        MV r1, r0
        LI r4, B

C       LW r6, r3, $0
        SLT r7, r6, r1
        BRNZ D
        MV  r1, r6
D       SLT r7, r3, r4
        BRZ E
        ADDI r3, r3, $1
        BRA C
E       STOP
```

   Ouch! Whoever wrote this would be a horrible lab partner – no comments and single character, nonsense labels.

Answer the following questions about this code:

(a) [3 points] What is the general task that this snippet is performing? *(Hint: this is a fairly common mathematical task.)*

(b) [2 points] For each of the eight registers, list what the registers are being used for in this program. If a register is not used, write "not used".

(c) [3 points] What is the address of the first **`SLT`** instruction?

(d) [4 points] How many bytes of memory does the code for the program take up? Don't include the **`.DW`** pseudo-operations. Show your work.

(e) [1 point] When the program finishes executing, where is the answer stored?

(f) [4 points] When the program is executed, using the data values provided, what value is the answer? There is another value that one might suspect would be the answer (but, isn't). Do you see it? Explain exactly why the program doesn't choose this value as the answer.

(g) [3 points] The program, as written, actually has a bug – one single character is wrong. Find and fix the bug. Explain why this is a bug.

4. [18 points, Lecture 17] Write a RISC240 assembly program to generate a magic square of (somewhat) arbitrary size.

   A magic square is a square matrix of consecutive integers, $1 \ldots N^2$, arranged in such that the sums of the values in any row, column or diagonal are the same.

   Since antiquity, algorithms for the creation of magic squares have been known. A very common variant, which works for all squares with an odd-sized side, is described in Knuth's *The Art of Computer Programming, Vol 1* on page 163. Simply put, start in the square below the middle and place a '1' in it. Then, move diagonally one place, down and to the right. Place a '2' there. Continue the process. All movements are, of course, mod the size of the square, so if you move off the bottom, then simply wrap around to the top. If you ever are to place a number in an occupied square, instead drop down two places from the last spot you placed a number in. You should continue for the numbers $1 \ldots N^2$ (where N is the size of the side of the square).

   Your program will follow the above algorithm, placing the results in a memory array which represents the two-dimensional square. Values are placed in memory, starting at a base address, in a row-major fashion (thus, the upper left cell is in the first spot of the array, followed by those in the top row, from left to right. Remember that the memory is word-aligned, so a cell at row R and column C (both counted from zero) is placed in memory a the address $BASE + 2 * (R * N + C)$.

   Your program will be given N, $BASE$ and N^2 (the latter to avoid making you do arbitrary multiplication. For this problem, N will always be 9, thus avoiding some more arbitrary multiplication (but, substituting a shift and an add).

   I have written a python program to illustrate the details of the algorithm. My program doesn't use mod operations, instead testing if the row or column counter is greater than N and subtracting N if it is. Find my program on Canvas as **`hw8prob4.py`**.

   Start your program at location **`$1000`**. Make sure you test your program using sim240!

   Here is some starter code for you:

```
; Compute a 9-sided magic square, using RISC240 assembly language

        .ORG $FF0   ; Input Data
SIDE    .DW  $9     ; Size of the Magic Square
SQUARE  .DW  $51    ; SIDE * SIDE (yep, doing the multiplication myself)
BASE    .EQU $2000  ; Base address of destination array

        .ORG $1000  ; Code segment
        ; Your code starts here
```

   ▷ **Submit your code in a file named `hw8prob4.asm`.**