18-240: Structure and Design of Digital Systems

Electrical & Computer ENGINEERING

HW9 [5 problems, 64 points]

Covers lectures L19 – L21 *Due: 4 December 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.

Drill Problems [28 points]

1. [12 points, Lecture 19] Identify the assembly language instructions from the following microinstructions. Be sure to show your work.

(a)

aluOp	aluSrcA	aluSrcB	dest	$\overline{\text{ldCC}}$	$\overline{\text{re}}$	$\overline{\text{we}}$
4'b0010	2'b01	2'bxx	3'b011	1'b1	1'b1	1'b1
4'b0100	2'b01	2'bxx	3'b001	1'b1	1'b0	1'b1
4'b0010	2'b10	2'bxx	3'b100	1'b1	1'b1	1'b1
4'bxxxx	2'bxx	2'bxx	3'b111	1'b1	1'b1	1'b1
4'b1001	2'b00	2'b00	3'b000	1'b0	1'b1	1'b1

(b)

aluOp	aluSrcA	aluSrcB	dest	$\overline{\text{ldCC}}$	$\overline{\text{re}}$	$\overline{\text{we}}$
F_A	MUX_PC	MUX_UNDEF	DEST_MAR	NO_LOAD	NO_RD	NO_WR
F_A_PLUS_2	MUX_PC	2'bxx	DEST_PC	NO_LOAD	MEM_RD	NO_WR
F_A	MUX_MDR	2'bxx	DEST_IR	NO_LOAD	NO_RD	NO_WR
4'bxxxx	2'bxx	2'bxx	DEST_NONE	NO_LOAD	NO_RD	NO_WR
F_A	MUX_PC	2'bxx	DEST_MAR	NO_LOAD	NO_RD	NO_WR
F_A_PLUS_2	MUX_PC	2'bxx	DEST_PC	NO_LOAD	MEM_RD	NO_WR
F_A_ASHR	MUX_REG	MUX_MDR	DEST_REG	LOAD_CC	NO_RD	NO_WR

(c) After the **`DECODE`** state:

aluOp	aluSrcA	aluSrcB	dest	$\overline{\text{ldCC}}$	$\overline{\text{re}}$	$\overline{\text{we}}$
F_A	MUX_PC	2'bxx	DEST_MAR	NO_LOAD	NO_RD	NO_WR
F_A_PLUS_2	MUX_PC	2'bxx	DEST_PC	NO_LOAD	MEM_RD	NO_WR
F_A_MINUS_B	MUX_REG	MUX_MDR	DEST_NONE	LOAD_CC	NO_RD	NO_WR
F_A_LT_B	MUX_REG	MUX_MDR	DEST_REG	NO_LOAD	NO_RD	NO_WR

2. [16 points, Lecture 20] Lecture 20 showed pictures of the execution steps involved in several different RISC240 instructions. Use **`sim240`** and its microstep operations to verify those steps for the following instructions. In each case, you will need to write some assembly code, assemble it and run the simulator. Submit a screenshot of the simulator showing each step and write a short description describing where you see the effects of each step in the screenshot. Make sure your description includes the actual assembly language instruction being executed.

   (a) The **`SW`** instruction from slides 10-14.

   (b) The **`ADDI`** instruction from slides 16-19.

   (c) The **`BRN`** instruction from slides 22-25. Show a situation where the branch is not taken.

   (d) The **`BRN`** instruction from slides 22-25. Show a situation where the branch is taken.

Non-Drill Problems [36 points]

3. [12 points, Lecture 21] In HW7, you designed a hardware thread to determine if a string in memory represented a palindrome or not. For this problem, you will write an assembly program to do the same job and then compare performance with the solution from HW7.

   Part A: [4 points]

   Write the assembly language program to detect palindromes. Start your code at address **`$1000`**. The following code defines the algorithm parameters and can be considered the input for your program.

```
; Determine if a string represents a palindrome or not

        .ORG $FF0   ; Input Data
STRADDR .DW  STR   ; String address, must be even
STRLEN  .DW  $B     ; Size of the string, in words.  Guaranteed not
                    ; to wrap beyond the top of memory
IS_PAL  .DW $0      ; After completion, 1 if yes, 0 if no

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

        ; At any random spot in memory, there is a string
STR     .DW $0001
        .DW $0002
        .DW $0000
        .DW $1000
        .DW $FFFE
        .DW $EFFF
        .DW $FFFE
        .DW $1000
        .DW $0000
        .DW $0002
        .DW $0001
        .DW $CAFE
        .DW $DEAD
        .DW $BEEF
```

   Your program will check in a "word-for-word" basis, as all objects in RISC240 are 16-bit words. If the check works out, store a **`$0001`** at the location **`IS_PAL`**. Otherwise, store a **`$0000`**.

   ▷ **Submit your code as a file named `hw9prob3.asm`.**

   Part B: [7 points]

Compare the performance between your code and the solution to HW7 problem 1.

Start by setting up the comparison. What do you consider a situation worth measuring? Do you need a "best-case" and "worst-case" scenario? How about "average-case"? Why?

Next, do the measurements / analysis. For each scenario you describe, figure out how many clock cycles the hardware thread would take and how many clock cycles the RISC240 would take. Are those clock cycles equivalent (i.e. is a clock cycle on the hardware thread faster than a clock cycle on RISC240)? If not, is there a way to compare your numbers?

▷ **Submit your work in your PDF.**

Part C: [1 point]

Draw a conclusion from your analysis. State clearly if one or the other implementations is better. Or, state a class of data for which one implementation is better.

▷ **Submit your work in your PDF.**

4. [12 points, Lectures 19 and 20] Write the microinstruction for the following RISC240 instructions using the register transfer notation (i.e., **`MAR`** ← **`PC`** style). Make sure to indicate which microinstructions belong in the same state. Assume the **`DECODE`** state has just completed.

   (a) **`OR r3, r6, r2`**

   (b) **`BRN $2000`**

   (c) **`SLLI r3, r3, $5`**

   (d) **`SW r5, r2, $240`**

5. [12 points, Lecture 20] In the RISC240 ISA, memory accesses are *aligned*, meaning that the address bus only contains even numbers.

   (a) What happens if you try to read from an odd memory address, say with an operation like **`LW r1, r0, $0001`**?

   (b) This seems complicated. Why would an ISA want to have aligned memory accesses?

   (c) Describe how memory alignment would work for an ISA that has larger memory values, say 32-bit words.