LSU EE 4720 Homework 5 Due: ~~11~~13 April 2018

This assignment consists of questions on the ARM A64 (AAarch64) ISA. (Not to be confused with ARM A32, which might be called classic ARM. Older information sources that refer to ARM are probably referring to A32, which is not relevant to this assignment.)

A description of the ARM ISA is linked to the course references page, at `http://www.ece.lsu.edu/ee4720/reference.html`. Feel free to seek out introductory material as a suppliment.

ARM A64 was used in EE 4720 Spring 2017 Homework 4 and Spring 2017 Midterm Exam Problems 2 and 3. It may be useful to see those assignments for code samples, but the questions themselves are different.

Appearing on the next page is a simple C routine, `lookup`, that returns a constant from a list. The routine appears to have been written with the expectation that its call argument, `i`, would be either 0, 1, or 2. Following the C code is ARM A64 code for `lookup` as compiled by gcc version 8.

Use the course reference materials and external sources to understand the ARM code below. The course references page has a link to the ARM ISA manual which should be sufficient to answer questions in this assignment. Feel free to seek out introductory material on ARM A64 (AArch64) assembly language, but after doing so use the ARM Architecture Reference Manual to answer questions in this assignment.

Full-length versions of the code on the next page, along with other code examples can be found at `http://www.ece.lsu.edu/ee4720/2018/hw05.c.html` and `http://www.ece.lsu.edu/ee4720/2018/hw05-arm.s.html`. These include the `pi` program and a simple copy program that was a part of the decompress program used in Homeworks 1, 2, and 3.

Code on next page, problems on following pages.

```
int lookup(int i)
{
  int c[] = { 0x12345678, 0x1234, 0x1234000 };
  return c[i];
 }


@ ARM A64 Assembly Code. C code appears in comments.
lookup:
@@
@
@    CALL VALUE
@     w0: The value of i (from the C routine above).
@
@    RETURN VALUE
@     w0: The value of c[i].
@
@    Note: The size of int here is 4 bytes.

@   const int c[] = { 0x12345678, 0x1234, 0x1234000 };

        adrp    x1, .LC1

        mov     w2, 0x4000

        ldr     d0, [x1, #:lo12:.LC1]

        movk    w2, 0x123, lsl 16

        str     w2, [sp, 8]

        str     d0, [sp]

@    return c[i];

        ldr     w0, [sp, w0, sxtw 2]

        ret


        .rodata.cst8,"aM",@progbits,8
.LC1:
        .word   0x12345678
        .word   0x1234
```

Problems start on next page.

Problem 1: The ARM code above uses three kinds of register names, those starting with `d`, `w`, and `x`.

(*a*) Explain the difference between each.

(*b*) MIPS has general-purpose registers and four sets of co-processor registers. Indicate the name of the register set for each of the three types of ARM registers above. Hint: two are part of the same set.

Problem 2: The `mov` moves constant `0x4000` into register `w2`. Actually, `mov` is a pseudo instruction.

(*a*) What are pseudo instructions called in ARM?

(*b*) What is the real instruction that the assembler will use in this particular case?

(*c*) Show the encoding for this use of `mov`. Be sure to show how `w2` and `0x4000` fit into the fields.

Problem 3: MIPS-I does not have an instruction like `adrp`.

(*a*) Describe what the `adrp` instruction does in general.

(*b*) Explain what it is doing in the code above. (It might be easier to look at the documentation for `adr` first.)

(*c*) Show MIPS code that writes the same value to its destination as `adrp`. Do not use MIPS pseudo instructions other than `la`. Assume that MIPS integer registers are 64 bits.

Problem 4: The `movk` instruction is sort of an improved version of `lui`.

(*a*) Describe what the `movk` instruction does in general.

(*b*) Explain why a single MIPS `lui` instruction could not do what the `movk` is doing in the code above.

Problem 5: Add comments to the ARM code above that explain what the code is doing, rather than what the individual instructions do.

Problem 6: The `lookup` routine was compiled using `gcc` at optimization level 3, the highest. Nevertheless, the code appears more complicated than it need to be. Explain what about the code is excessively complicated and how it could be simplified.