LSU EE 4720 **Homework 4** **Due: 10 March 2017**

To help in solving this problem it might be useful to study the solutions to the following problems: Spring 2014 Homework 3 (ARM A32 instructions, ARM-like scaling instructions). Fall 2010 Homework 3 (shift unit in MIPS).

See the course references page for a link to the ARM v8 ISA, which will be needed to solve the problems below.

Problem 1: In most RISC ISAs register number 0 is not a true register, its value as a source is always zero, and it can be harmlessly used as a destination (for example, for use as a `nop` instruction).

The ARM A64 instruction set (not to be confused with A32 [arm] or T32 [thumb]) takes a different approach to the zero register.

(*a*) What register number is the zero register in A64?

(*b*) Let z denote the answer to the previous part, meaning that `rz` can denote the zero register. In an ISA like MIPS, the general purpose register (GPR) file only needs enough storage for 31 registers, since the register zero location can be hardwired to zeros. But in ARM A64 the GPR file needs 32 storage locations because register number z is the zero register for some instructions, but an ordinary register for others. In ARM notation `ZR` denotes the zero register, in this problem `rz` indicates the register number of the zero register, which, depending on the instruction can refer to the zero register or to an ordinary register.

Show two instructions that can read `rz` and one that can write `rz`, for these instructions `rz` is an ordinary register (at least for certain operand fields).

Show a one-destination-register, two-source-register instruction for which `rz` is the zero register for all operand fields.

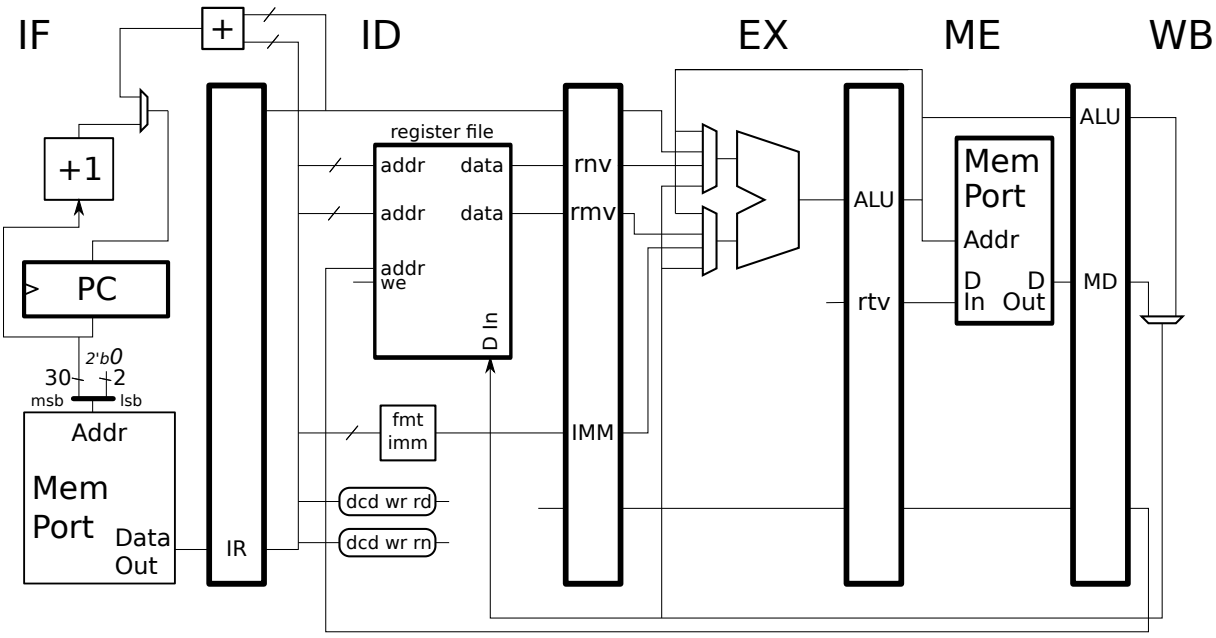
There's another problem on the next page.

Problem 2: The ARM A64 code below computes the sum of an array of 64-bit integers. The load instruction uses *post-index addressing*, the behavior of this instruction is shown in the comments. (The `@` is the comment character.)

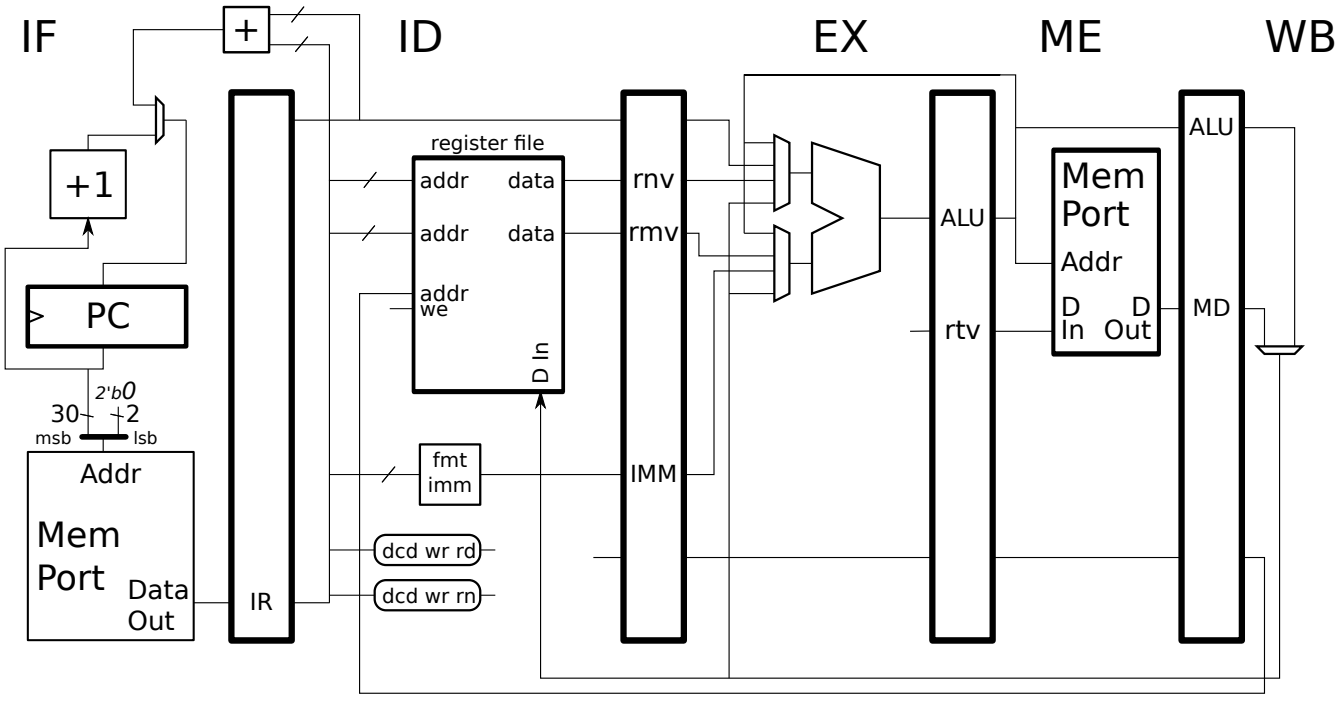
```
LOOP: @ ARM A64
 ldr x1, [x2], #8   @ x1 = Mem[x2];  x2 = x2 + 8
 cmp x2, x4
 add x3, x3, x1
 bne LOOP
```

(*a*) Appearing below is a pipeline based on our MIPS implementation. Add datapath elements so that it can execute the `ldr` with pre-index, post-index, and immediate addressing. Don't make changes that will break other instructions. Note that the register file has a write-enable input, which is necessary because there is no full time register that acts as register zero.

An Inkscape SVG version of the implementation can be found at `https://www.ece.lsu.edu/ee4720/2017/hw04-armskel.svg`.



- Show the bits used to connect to the address inputs of register file.
- Show the second write port on the register file needed for the updated address.
- Use fixed bit positions for the destination registers.
- Use the given decode logic to determine a write enable signal for each dest.
- The changes must work well with pipelining.
- As always, avoid excessively costly solutions.
- **Do not** add hardware for the branch or compare.


IF
ID
EX
ME
WB
+
+1
PC
2'b0
30
2
msb
lsb
Addr
Mem
Port
Data
Out
IR
register file
addr
data
addr
data
addr
we
D In
fmt
imm
dcd wr rd
dcd wr rn
rnv
rmv
IMM
ALU
rtv
Mem
Port
Addr
D
In
D
Out
ALU
MD