LSU EE 4720 **Homework 1** **Due: 17 February 2012**

Problem 1: To save space in a program an array is designed to hold four-bit unsigned integers instead of the usual 32-bit integers (it is known in advance that their values are $\in [0, 15]$). Because this 4-bit data size is less than the smallest MIPS integer size, 8-bits, even a load byte instruction will fetch two array elements. Code to read such an array and a test routine appear on the next page, along with a stub for code to write the array. The routine `compact_array_read` is used to read an element of this array and `compact_array_write` is the start (mostly comments) of a routine to write an element.

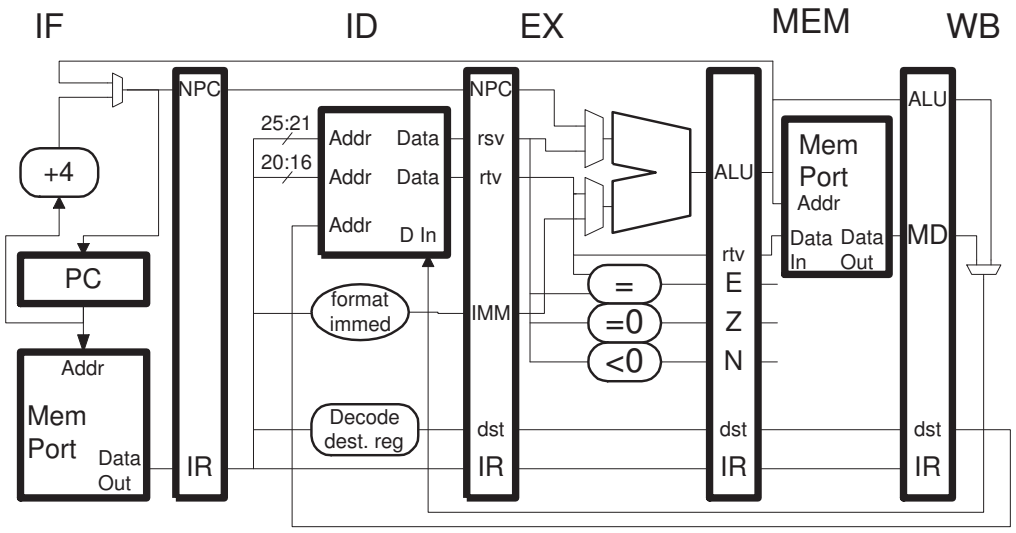
(*a*) Add comments to `compact_array_read` appropriate for an experienced programmer. The comments should describe how instructions achieve the goal of reading from the array. The comments **should not** explain what the instruction itself does, something an experienced program already knows. See the test code for examples of good comments.

(*b*) Complete the routine `compact_array_write`, so that it writes data into the array. See the comments for details.

```
################################################################################
##
## Test Code
##
        .data
a:      # Array of values to test. Each byte hold two 4-bit elements.
        .byte 0x12, 0x34, 0x56

msg:    # Message format string (similar to printf).
        .asciiz "Value of array element a[%/s0/d] is 0x%/s3/x\n"

        .text
        .globl __start
__start:
        addi $s2, $0, 4    # Last index in array a.
        addi $s0, $0, 0    # Initialize loop index.
LOOP:
        la $a0, a                # First argument, address of array.
        jal compact_array_read
        addi $a1, $s0, 0         # Second argument, index of element to read.
        la $a0, msg              # Format string for test routine's msg.
        addi $s3, $v0, 0         # Move return value (array element) ...
        addi $v0, $0, 11         # ... out of $v0 and replace with 11 ...
        syscall                  # ... which is the printf syscall code.
        bne $s0, $s2 LOOP
        addi $s0, $s0, 1         # Good Comment: Advance index to next
                                 #               element of test array.
                                 # Bad Comment: Add 1 to contents of $s0.

        li $v0, 10               # Syscall code for exit.
        syscall

################################################################################
##
## compact_array_read
##
compact_array_read:
        ## Register Usage
        #
        # CALL VALUES
        #   $a0:  Address of first element of array.
        #   $a1:  Index of element to read.
        #
        # RETURN VALUE
        #   $v0:  Array element that has been read.
        #
        # Element size: 4 bits.
        # Element format: unsigned integer.

        srl $t0, $a1, 1
        add $t1, $a0, $t0
        andi $t3, $a1, 1
        bne $t3, $0 SKIP
        lb $t2, 0($t1)
        jr $ra
        srl $v0, $t2, 4
SKIP:   jr $ra
        andi $v0, $t2, 0xf
```

```
################################################################################
#
# compact_array_write
#

compact_array_write:
        ## Register Usage
        #
        # CALL VALUES
        #   $a0:  Address of first element of array.
        #   $a1:  Index of element to write.
        #   $a2:  Value to write.
        #
        # RETURN VALUE
        #   None.
        #
        # Element size: 4 bits.
        # Element format: unsigned integer.
```

Problem 2: The MIPS code below executes on the illustrated implementation. The loop iterates for many cycles. The register file bypasses data from the write ports to the read port in the same cycle.




```
LOOP:
 srl r4, r3, 2
 sw r4, 0(r3)
 bne r3, r2 LOOP
 addi r3, r3, 4
```

(*a*) Show a pipeline execution diagram for the code above on the illustrated implementation for enough iterations to determine CPI.