EE 4720 Homework 2 Due: 21 February 2001

Problem 1: Translate the following C program to DLX assembly, use the minimum number of comparison instructions. Pay attention to data type sizes. The line labels are provided for convenience, please use them in the assembly language version.

```
  extern int r1, r2, r3, r10, r11;
  extern int *r20, *r21;
  /* For DLX: sizeof(int) = sizeof(int*) = 4   */
  /* For IA-64: sizeof(int) = sizeof(int*) = 8   */

  if( r1 < 3 )
    {
    LINE1:
      if( r2 == r3 )
        {
        LINE11: r10 = *r20++;
        }
      else
        {
        LINE10: r10 = 4720;
        }
    LINE1E:
      r11 = r11 + r10;
    }
  else
    {
    LINE0:
      r21 = r21 + 7;
      if( r2 == r3 )
        {
        LINE01: r10 = *r21++;
        }
      else
        {
        LINE00: r10 = 7700;
        }
    }
 DONE:
```

Problem 2: Translate the C program from the previous problem into IA-64 assembly using predicated instructions. (You're not expected to know it at this point.) IA-64 is described in the IA-64 Application Developer's Architecture Guide, available at
`http://developer.intel.com/design/ia64/downloads/adag.pdf`.

For this problem one can ignore alot of IA-64's features. Here is what you will need to know: IA-64 has 64 1-bit predicate registers, `p0` to `p63`, which are written by `cmp` (compare) and other instructions. Predicates can be specified for most instructions, including `cmp`. See 11.2.2 for a description of how to use IA-64 predicates.

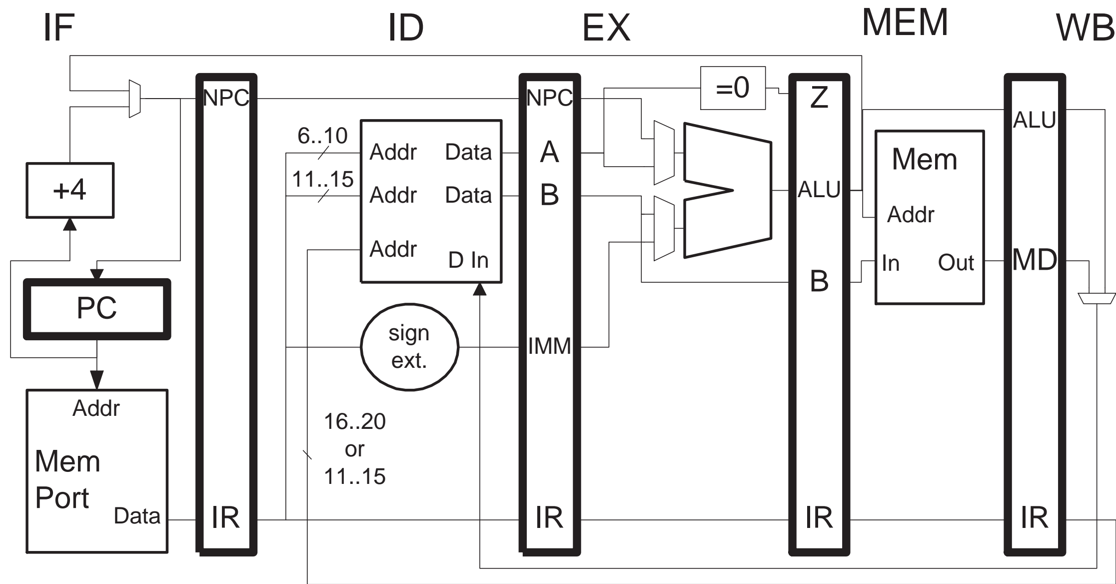
To solve the problem look at the following sections: 11.2.2 (predicate description) and Chapter 7 (for instruction descriptions). The following instructions will be needed: `cmp` (compare, look at the normal [none] and `unc` comparison types), `ld1`, `ld2`,... (loads), and `add`.

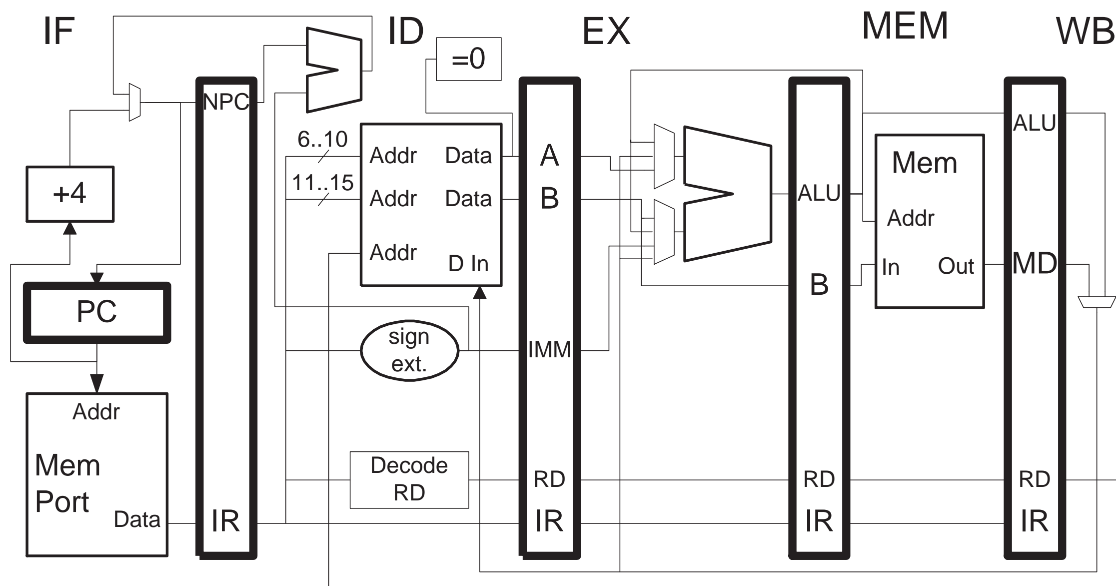
To save time, ignore instruction stops (;;) and consider only normal loads. (Post-increment like loads are considered normal here.)

- Use general-purpose registers r0-r31 and predicate registers p1-p63 in your solution. (There are 128 general-purpose registers, but those above r31 must be allocated.)
- **Do not** use branches (or any other CTI).
- Ignore stops. (These will be covered later.)
- Use the minimum number of `cmp` instructions. (Three is possible.)
- Do not assign a value to a register unless it's needed.
- Make use of post-increment loads.
- Pay attention to data type sizes.

Problem 3: Show a pipeline execution diagram of the code below on each implementation. (There should be a total of two diagrams.) The branch is always taken, show the diagram until the second execution of the first instruction reaches WB. If a bypass path is not shown, it's not there.

```
LOOP:
 addi r2, r2, #4
 lw   r1, 0(r2)
 add  r3, r3, r1
 slt  r4, r2, r5
 beqz r4, LOOP
 xor  r5, r4, r1
```





Problem 4: For each implementation from the problem above, determine the CPI for a large number of iterations.

Problem 5: For the second pipeline execution diagram above, show the location(s) of *the latest value of* `r1` and `r2` at the beginning of each cycle on the diagram below. For `r1` box the appropriate cycle numbers and draw an arrow to the locations. For `r2` circle the cycle numbers and draw an arrow to the locations. In the diagram below this has been completed for cycles zero and two, assuming `addi` is in IF at cycle zero. The arrows should only point to register values that are valid at the indicated cycles. Note: A valid value can be in more than one location at once.