CS311 - COMPUTER SYSTEMS, I

Homework #6 - Due: Monday, November 12, at the start of class

Emphasis: CPU Internals, Control, Pipelining

1. Referring to the RTL for the simulated MIPS implementation that was handout out in class, list the microoperations that would be performed in executing the following program, assuming that `$4` initially contains 2 and `$31` initially contains `00002000`. For each microoperation, specify the cycle on which it would be done. (The first instruction and the start of the second have been done as an example - note that "variables" in the RTL have been replaced with actual values - do likewise). (Hint: the entire answer will be quite a bit less than 50 cycles!)

```
Memory address Contents    Assembly language equivalent
(memory address/contents hex)

00001000       00001025          or    $2, $0, $0
                           loop:
00001004       00441020          add   $2, $2, $4
00001008       2084ffff          addi  $4, $4, -1
0000100c       1480fffd          bne   $4, $0, loop
00001010       03e02008          jr    $31
```

   Answer begins:

```
Cycle 0:       IR ← 00001025, PC ← 00001004
Cycle 1:       ALUInputA ← 00000000, ALUInputB ← 00000000
Cycle 2:       ALUOutput ← 00000000
Cycle 3:       $2 ← 00000000
Cycle 0:       IR ← 00441020, PC ← 00001008
```

   (You do the rest through all cycles for executing the `jr` instruction)

2. (Adapted from Patterson and Hennessy). Referring to the diagram of a simulated MIPS CPU implementation that was handed out in class, consider what would happen if one of the control signals were "stuck" at some value - i.e., due to a hardware fault, the signal always had some particular value, regardless of what its value should be.

   For each of the following cases, indicate which group(s) of instructions in the MIPS ISA (if any) would still execute corrrectly (i.e. they either don't use the relevant portion of the datapaths at all, or they require this control signal to be the value that it's stuck at anyway.) Groups of instructions to consider: R-format; I-format immediate; I-format load; I-format store; I-format branch; j; jal. Briefly indicate why you reached the conclusion you did.

   Example: Control word bit used to control what gets written into a register in the register set stuck at 0, rest OK:

   RType, I-format immediate, jal - OK; correct value for these.
   I-format load - would fail: can't store result.
   I-format store, I-format branch, j - OK: don't write to registers

   a. Control word bit used to control what gets written into a register stuck at 1, rest OK.

   b. Control word bit used to control the first of the two sources to the ALU stuck at 0, rest OK.

   c. Control word bit used to control what serves as the memory address stuck at 0, rest OK.

   d. Control word bit used to control what serves as the memory address stuck at 1, rest OK.

3. Suppose a certain instruction takes 5 clock cycles to execute.

   a. How long would it take to execute 100 of these instructions on the non-pipelined CPU with a 1 GHz clock?

   b. How long would it take to execute 100 of these instructions on a pipelined CPU with a 1 GHz clock?

4. Repeat part (b) above for the case that one step in the instruction's execution requires 2 ns, while the remaining four steps can each be done in 1 ns. (Assume the long step cannot be broken into two steps)

5. Consider the following C++ code fragment, and a possible translation into MIPS assembly language, which was done by someone who was unaware of the fact that MIPS branch instructions take effect after a one instruction delay (delayed branch), and that (on the particular MIPS implementation in use) data loaded by a MIPS load instruction cannot be used on the very next instruction (delayed load). (Assume that variables s, a, and b reside in memory.)

```
s = a + b  + 1;
if (s == 0)
      a = 0;
else
      b = 0;
```

   Assembly language translation:

```
      lw    $2, a
      lw    $3, b
      add   $2, $2, $3
      addi  $2, $2, 1
      sw    $2, s
      bne   $2, $0, else
      sw    $0, a
      b     endif
else:
      sw    $0, b
endif:
```

   a. Insert necessary nops to make the code correctly translate the C++ fragment, without rearranging or altering any instructions. (I.e. do what a minimal MIPS assembler might do.)

   b. Eliminate as many nops as possible by rearrangment of the code - still preserving the intended meaning. (I.e. do what a really "smart" MIPS assembler or compiler does.) **Note**: If you are clever, you can create a version that requires no nops!

6. Consider the following program fragment from a MIPS-like RISC that uses delayed branch:

```
i1:   Some  non-branch  instruction
i2:   b     i5          # Unconditional branch to i5
i3:   beq   $2, $0, i7  # Branch to i7 if $2 = zero
i4:   Some  non-branch  instruction
i5:   Some  non-branch  instruction
i6:   Some  non-branch  instruction
i7:   Some  non-branch  instruction
```

   a. Which instructions will be executed, and in what order, if $2 is not zero at i3?

   b. Repeat, but assume that $2 is zero at i3.