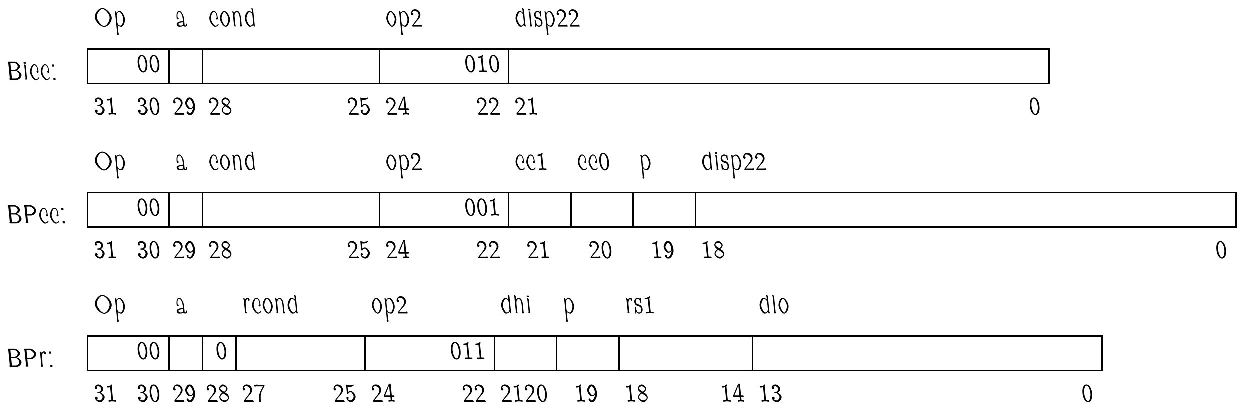
LSU EE 4720 **Homework 3** Solution **Due: 21 February 2013**

Problem 1: As described in class, SPARC v7 integer branch instructions use a 22-bit immediate field for the displacement. Branches are typically used in loops and if/else constructs, and so the ±2097152 instruction range might be more than is needed. So did the computer engineers at Sun Microsystems (now part of Oracle). Look up the v7 integer branch instruction in the SPARC Joint Programming Specification (JPS1), linked to the course references page (look for JPS1). You'll find SPARC v7 integer branch under Instruction Definitions in the Deprecated Instructions section. Then look up the replacement integer branch instructions (not in the deprecated section).

(*a*) Sketch (or cut-and-paste, take a picture with your cell phone, etc.) the format of the three instructions (one old, two new).

The formats appear below. The `dhi` and `dlo` in the last format refer to the 16-bit displacement, split across two fields.



(*b*) Describe how `BPr` is different than the original v7 integer branch instruction, and point out two benefits.

The `BPr` instruction can test the value of a register (the one specified by the `rs1` field), avoiding the need for an instruction like `subcc` to set the condition code register. It also provides, in field `p`, a hint to the hardware on whether the branch will be taken. The hardware may choose to ignore the hint (relying on its own branch predictor). If the hardware uses the hint and it is wrong performance will be slightly lower than if the hint were right.

Grading Note: Many incorrectly answered that with `BPcc` and `Bicc` the condition setting instruction would have to immediately precede the branch. That's not true, the condition setting instruction, such as `subcc`, can be placed far away from the branch.

(*c*) Describe how `BPcc` is different than the original v7 integer branch instruction. This instruction shares one benefit with `BPr`, but it has lost 2 bits of displacement in order to accommodate 64-bit register values. (The other third lost bit has nothing to do with 64-bit register values). Explain.

The shared benefit is the prediction hint, using a bit in the `p` field. The two "lost" bits are for the `cc1` and `cc0` fields. These specify which condition code register to use. There are two integer condition code registers, `icc` and `xcc`. The `icc` is set based on the low 32 bits of a result (and so ignore the upper 32 bits), while the `xcc` is set based on the full 64 bits. The `icc` register is need for code compiled for SPARC v7, which used 32-bit registers. An arithmetic operation might overflow a 32-bit register, and SPARC v7 code would expect that overflow. If run on v9, the operation would not cause an overflow. The overflow (V) bit on the `xcc` register would not be set, but the V bit on the `icc` would be set. So, when developing a 64-bit version of the SPARC ISA, Sun engineers added those `cc1` and `cc0` bits to maintain compatibility. (They could have used just one bit, but two bits were used so that `BPcc` would be similar to the floating-point version, `FBPfcc`, which does need two bits because their are 4 FP condition code registers.)

Problem 2: For the following assignment familiarize yourself with the VAX ISA by looking in the VAX-11 Architecture Reference Manual (linked to the course references page). In particular, see Section 2.6 for a summary of the instruction format, and Chapter 3 for details on the operand specifiers used in the instruction formats. For examples, look at some past homework assignments in this course: `http://www.ece.lsu.edu/ee4720/2010f/hw04_sol.pdf`, `http://www.ece.lsu.edu/ee4720/2007f/hw03_sol.pdf`, and `http://www.ece.lsu.edu/ee4720/2002/hw02_sol.pdf`.

The VAX format is simple, it consists of a one- or two-byte opcode followed by some number of *operand specifiers* and any additional fields they may use. The operand specifiers are 8 bits, and are followed by a possible extension and immediates. (See Section 2.6 and Chapter 3 of the VAX-11 Architecture Reference Manual.)

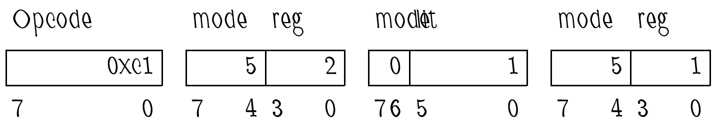
(*a*) The VAX operand specifier is 8 bits, it includes a 4-bit mode field, and for literal addressing, a 6-bit literal field. (A *literal* in VAX is a small immediate.) Explain how it's possible to fit a 4-bit mode field and a 6-bit literal field into 8 bits.

The two least-significant bits of the mode field is used for part of the literal. Therefore literal mode actually uses four mode values, 0 through 3.

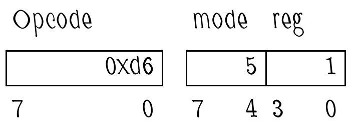
(*b*) Find the best VAX replacement for each of the two MIPS instructions below and show their encoding. The two VAX instructions will be different.

The VAX instructions appear right after the corresponding MIPS instruction, followed by the VAX instruction coding. For the first MIPS instruction a 3-operand VAX add was used. But for the second, a VAX increment (`INCL`) instruction was used, which requires only a single operand, the register number. The `INCL` instruction is one byte shorter than an `ADDL2` which needs a one-byte specifier for the immediate value, 1.

```
addi r1, r2, 1      # MIPS
ADDL3 r2, S^#1, r1    # VAX (destination register is last)
```



```
addi r1, r1, 1      # MIPS
INCL r1             # VAX
```

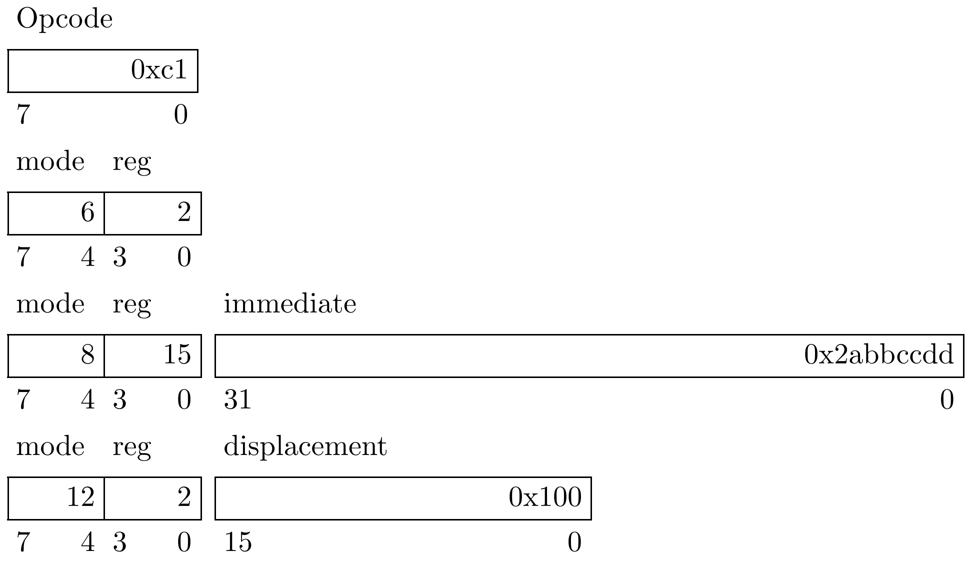


(*c*) Find a VAX instruction to replace the following sequence of MIPS instructions, and show its encoding.

```
lw r1, 0(r2)
lui r3, 0x2abb
ori r3, r3, 0xccdd
add r1, r1, r3
sw r1, 0x100(r2)
```

Solution appears below. Note that the load performed by the first MIPS `lw` instruction uses VAX register deferred mode, rather than displacement mode, since the displacement is zero. The VAX encoding shown below on several lines, with one line for the opcode and one line for each operand.

```
# SOLUTION
ADDL3 (r2), I^#0x2abbccdd, W^0x100(r2)   # Destination is last.
```



(*d*) Compare the size of the VAX instruction from the problem above to the size of the MIPS instructions.

There are five MIPS instructions, for a total size of 20 bytes. The single VAX instruction is 10 bytes, half the size.