(Problem 3)

The instruction iaddl would merge the computations for irmovl and addl. There are five stages to the instruction:

(Fetch Stage) The instruction code (C), function code (0), register codes and immediate value V are all fetched from memory. Only one register is needed, so rA = F (no register) and rB designates the register used. The four-byte word valC corresponds with V and valP predicts a 6 byte increase for the PC.

(Decode Stage) The register codes are decoded to specify the register used. For example, if %eax is to be used, then rB = 0 so valB will specify that %eax should be read. Since rA is not needed rA = F so valA will specify “no register”.

(Execute Stage) Now that valC holds the proper bit sequence for the V and valB specifies the register to be used, the ALU can perform an add operation on the contents of the register and the value for V. The result appears at valE and condition codes are set according to the result.

(Write Back) The result of the add performed by the ALU is stored in the destination register, which is just the same register used as an input (rB). Therefore, in the next clock cycle rB will contain the result.

(PC Update) The program counter is incremented by 6 bytes for the next instruction.

Here are the computations using the notation from our textbook:

```
# Fetch
icode:ifun <– M1[PC] # icode = C, ifun = 0
rA:rB <– M1[PC+1] # rA = F, rB = 0
valC <– M1[PC+1] # valC = V
valP <– PC + 6 # predict program counter to increment by 6 bytes (next
instruction)

# Decode
valA <– R[rA] # valA = “no register”
valB <– R[rB] # valB = designated register for computation

# Execute
valE <– valB + valC # sum V and register eax
Set CC # set the condition codes

# Write back
R[rB] <– valE # store result in %eax

# PC Update
PC <– valP # program counter incremented by 6 bytes for the next instruction
```