HW 1 Solutions

1a)

96 bits are needed. 32 bits for each operand address.

Instruction format: [{95:64}, {63:32} {31:0}]
a b c

Note: other formats instruction formats with different ordering of a b and c also acceptable.

1b)

Based on class discussion, Subleq a 2-address ISA since one of the operands is also used for result. However, it does have 3 addresses, so it is arguably a 3-address ISA. Both solutions are acceptable.

Subleq is a memory-memory ISA

1c)

Assume Mem[A] = vA; Mem[B] = vB before code executes.

The pseudo-code below explains what the subleq instructions do.

```
Mem[A] = vA-vB
Mem[B] = vB
If (vA<=vB)
        Mem[C] = vA-vB
Else
        Mem[C] = vB
Mem[D] = 0
```

With the daa given in the question: vA = 7; vB=-9; vC=4; vD=3 the final values are:

Mem[A] = 16
Mem[B] = -9
Mem[C] = -9
Mem[D] = 0

1d)

There are several possible solutions to this problem. Shown below is just one.

```
L0: subleq M1 M1 L1 //M1=0
L1: subleq M2 M2 L2 //M2=0
L2: subleq M2 M0 L3 //M2 = -N
L3: subleq M0 C1 L5 //Decrement M0 by 1
L4: subleq M2 M0 L3 // M2 = M2 - decremented N
L5: subleq M1 M2  L6 // Set result in M1
L6: exit
```

2a)

```
//Original instruction
add rs, rt, rd //R[rd] <- R[rs] + R[rt]
```

Note that the add instruction generates a trap on over-flow. Ideally, your implementation should also trap on over-flow with the same values stored in registers rs, rt and rd as the original instruction.

```
//Pseudo-code
maxval = 32'b1
If (rs==maxval)
        If (rt==maxval)
                Rx = 32'b1
                addi rx, rd, 1
        Else
                subu $0, rt, rx
                Sub rs, rx, rd
Else
        Subu $0, rs, rx
        Sub rt, rx, rd

//Solution
L0: Addiu $0, rx, 16'b1
L1: bneq  rs, rx, L8
        L2: bneq rt, rx,  L5
                L3: addi rx, rd, 1
                L4: beq $0, $0, end
        L5: subu $0, rt, rx5
        L6: sub rs, rx, rd
        L7: beq $0, $0, end
```

L8: subu $0, rs, rx
L9: sub rt, rx, rd

2b)

//Original
and rs, rt, rd

//x.y = (x' NOR y')

//Solution
nor rs, rs, rs
nor rt, rt, rt
nor rs, rt, rd
nor rs, rs, rs
nor rt, rt, rt

2c)

//Original
lw rs, rt, immediate // M[rt] = M[rs + SignExtimmediate]

//Solution assuming Little Endian (re-do for Big Endian)
lhu rs, rx, immediate
sll rx, rx, 16
lhu rs, rt, immediate+2
or rt, rx, rt

2d)

//Original
j jmpAddress //Go to address {PC+4[31:28], jmpAddress, 2'b0}

//Re-implementation; note that the PC at which j jmpAddress is known to you and can be used
//as an immediate
// imm1 = {PC+4[31:28], jmpAddress[25:14]}
// imm2 = {jmpAddress[13:0],00}

lui rx, imm1 //R[rx] = {imm1, 16'b0}
ori rx, rx, imm2 //R[rx] = {imm1,imm2} (i.e., the target address for the j instruction
jr rx

3a)
Examples of 3-address instructions: [5 Points]

Format 2 ADD/SUB instructions

ADD Rd, Rs, Rn
SUB Rd, Rs, Rn
(See 5.2. in manual)0

Examples of 2-address instructions [5 Points]

Format 4 instructions

AND Rd, Rs
ORR Rd, Rs
… others
(See 5.4 in manual)

3b)

Absolute addressing: (load from an address provided as an immediate field

A Format 6 PC-relative load (LDR Rd, [PC, #Imm]) or a Format 11 SP-relative load/store can be considered an example, although it loads from PC+imm or SP+imm instead of just imm. Format 12 load/stores also work because they are a combination of Format 6 and Format 11.
Finally Format 13 load/stores use an offset from the stack pointer so that's good too.

Register Indirect: (load from address stored in register)

A format 9 (or format 10) load/store with the immediate offset set to zero is an example. Again, this is not a perfect match because the it adds an offset, but as long as the offset is zero it acts like a register indirect.

Displaced/Based (load/store from register address + plus imm offset)

A format 9 or Format 10 load/store is a clear example.

Indexed (base and offsets picked from registers)

A Format 7 load/store is a clear example.

Memory Indirect

No examples.

Auto-increment

A Format 15 multiple load/store is a clear example. It does almost exactly what we said auto-increment does (actually a little more); so hard to argue that this mode does not exist.