Basic Specifications of UNCG-RISC

Memory size: 128 MB expandable to 4 GB; byte addressing;
Registers: 64 integer registers R0 to R63, R0 is always zero, each register is 64 bits;
64 floating point registers, F0 to F63, each register is 64 bits;
32-bit PC, 32-bit IR, other internal registers as needed;

Use the 5-stage (integer) CPU architecture from [HP]. You may need to modify or extend the architecture slightly to accommodate all integer operations (including the jump/branch group).

Design the instruction format (machine code) for the following assembly language. Try to maximize the number of bits for displacement (d) and immediate operand (#p).

INSTRUCTION SET:
All instructions are 32-bits;

Load, store

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB Rn, d(Rm)</td>
<td>Regs[Rn] &lt;- Mem [d + Regs[Rm]] (One byte is loaded)</td>
</tr>
<tr>
<td>LH Rn, d(Rm)</td>
<td>Regs[Rn] &lt;- Mem [d + Regs[Rm]] (Half word is loaded)</td>
</tr>
<tr>
<td>LW Rn, d(Rm)</td>
<td>Regs[Rn] &lt;- Mem [d + Regs[Rm]] (One word is loaded)</td>
</tr>
<tr>
<td>LD Rn, d(Rm)</td>
<td>Regs[Rn] &lt;- Mem [d + Regs[Rm]] (Two words are loaded)</td>
</tr>
<tr>
<td>SB Rn, d(Rm)</td>
<td>Mem [d + Regs[Rm]] &lt;- Regs[Rn] (One byte is stored)</td>
</tr>
<tr>
<td>SH Rn, d(Rm)</td>
<td>Mem [d + Regs[Rm]] &lt;- Regs[Rn] (Half word is stored)</td>
</tr>
<tr>
<td>SW Rn, d(Rm)</td>
<td>Mem [d + Regs[Rm]] &lt;- Regs[Rn] (One word is stored)</td>
</tr>
<tr>
<td>SD Rn, d(Rm)</td>
<td>Mem [d + Regs[Rm]] &lt;- Regs[Rn] (Two words are stored)</td>
</tr>
<tr>
<td>LI Rn, #p</td>
<td>Regs[Rn] &lt;- p (sign-extended to 64 bit)</td>
</tr>
<tr>
<td>L.S Fn, d(Rm)</td>
<td>Single precision floating point load</td>
</tr>
<tr>
<td>L.D Fn, d(Rm)</td>
<td>Double precision floating point load</td>
</tr>
<tr>
<td>S.S Fn, d(Rm)</td>
<td>Single precision floating point store</td>
</tr>
<tr>
<td>S.D Fn, d(Rm)</td>
<td>Double precision floating point store</td>
</tr>
</tbody>
</table>

NOP
A machine instruction consisting of 32 zeros is the NOP.
Register-Register, register-immediate, and shift

ADD  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] + Regs[Rp]
SUB  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] - Regs[Rp]
MUL  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] * Regs[Rp]
DIV  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] \ Regs[Rp]
AND  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] & Regs[Rp]
OR   Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] | Regs[Rp]
XOR  Rn, Rm, Rp  Regs[Rn] <- Regs[Rm] xor Regs[Rp]
ADDI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] + p
SUBI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] - p
MULI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] * p
DIVI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] \ p
ANDI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] & p
ORI  Rn, Rm, #p  Regs[Rn] <- Regs[Rm] | p
XORI Rn, Rm, #p  Regs[Rn] <- Regs[Rm] xor p
SLLI Rn, Rm, #q  Regs[Rn] <- Regs[Rm] logical shift left q places
SRLI Rn, Rm, #q  Regs[Rn] <- Regs[Rm] logical shift right q places
SLAI Rn, Rm, #q  Regs[Rn] <- Regs[Rm] arithmetic shift left q places
SRAI Rn, Rm, #q  Regs[Rn] <- Regs[Rm] arithmetic shift right q places

Floating Point

ADD.S Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] + Regs[Fp] (single precision)
SUB.S Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] - Regs[Fp] (single precision)
MUL.S Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] * Regs[Fp] (single precision)
DIV.S Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] / Regs[Fp] (single precision)
ADD.D Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] + Regs[Fp] (double precision)
SUB.D Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] - Regs[Fp] (double precision)
MUL.D Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] * Regs[Fp] (double precision)
DIV.D Fn, Fm, Fp  Regs[Fn] <- Regs[Fm] / Regs[Fp] (double precision)
MOV.S Fn, Fm  Regs[Fn] <- Regs[Fm] (single precision)
MOV.D Fn, Fm  Regs[Fn] <- Regs[Fm] (double precision)

Jump and Branch

J #p  PC <- PC+4+p
JR Rn  PC <- Regs[Rn]
JAL #p  Regs[R31] <- PC+4; PC <- PC+4+p
JALR Rn  Regs[R31] <- PC+4; PC <- Regs[Rn]
BEQZ Rn, #p  if (Regs[Rn] == 0) then PC <- PC+4+p
BNEZ Rn, #p  if (Regs[Rn] != 0) then PC <- PC+4+p
BEQ Rn, Rm, #p  if (Regs[Rn] == Regs[Rm]) then PC <- PC+4+p
BNE Rn, Rm, #p  if (Regs[Rn] != Regs[Rm]) then PC <- PC+4+p