Pseudoinstructions in the MiniMIPS assembly language

We have learned many basic instructions for MiniMIPS in our lectures, such as *add*, *and*, *jal*, etc. It is, on the other hand, sometimes easier and more natural to use some alternative formulations. They are collectively called *pseudoinstructions*. The MiniMIPS assembler is responsible for translating them into the basic instructions.

For instance, we can use the following instruction:

```
nor $s0, $s0, $zero
```

to invert each bit in register $s0. But it would be natural to understand and write the operation as

```
not $s0
```

Toward this end, we thus define not to be a pseudointruction in MiniMIPS. It is treated as an ordinary instruction by the MiniMIPS assembler and is replaced by the equivalent nor instruction as above, before the assembler converts it into its binary format.

Some pseudoinstructions can be replaced by more than one instruction and may involve some intermediate values. Recall that in the register file, there is a register called $at. Usually, MiniMIPS assembler uses this register for intermediate values. Look at the following pseudointrouction abs, which places the absolute value of the content of a source register into a destination register.

```
abs $t0, $s0          # put |($s0)| into $t0
```

The MiniMIPS assembler might translate the pseudoinstruction into the following sequence of four basic instructions.

```
add $t0, $s0, $zero     # copy the operand x into $t0
slt $at, $t0, $zero     # is x negative?
beq $at, $zero, +4      # if not, skip the next instruction
sub $t0, $zero, $s0     # the result is 0 – x
```

The following table lists those frequently-used pseudoinstructions provided by MiniMIPS.

Pseudoinstruction	Usage	Meaning
Move	move regd, regs	regd ← (regs)
Load address	la regd, address	load computer address, not content
Load immediate	li regd, anyimm	regd ← immediate value
Absolute value	abs regd, regs	regd ← |(regs)|
Negate	neg regd, regs	regd ← - (regs)
Multiply (into register)	mul regd,reg1,reg2	regd ← (reg1) × (reg2)
Divide (into register)	div regd,reg1,reg2	regd ← (reg1) ÷ (reg2)
Remainder	rem regd,reg1,reg2	regd ← (reg1) mode (reg2)
Set greater than	sgt regd,reg1,reg2	regd ← if (reg1) > (reg2) then 1 else 0
Set less or equal	sle regd,reg1,reg2	regd ← if (reg1) ≤ (reg2) then 1 else 0
Set greater or equal	sge regd,reg1,reg2	regd ← if (reg1) ≥ (reg2) then 1 else 0
Rotate left	rol regd,reg1,reg2	regd ← (reg1) left rotated by (reg2)
Rotate right	ror regd,reg1,reg2	regd ← right left rotated by (reg2)
NOT	not reg	reg ← (reg)'
Load doubleword	ld regd,address	load regd and the next register
Store doubleword	sd regd,address	store regd and the next register
Branch less than	blt reg1,reg2,L	if (reg1) < (reg2) then goto L
Branch greater than	bgt reg1,reg2,L	if (reg1) > (reg2) then goto L
Branch less or equal	ble reg1,reg2,L	if (reg1) ≤ (reg2) then goto L
Branch greater or equal	bge reg1,reg2,L	if (reg1) ≥ (reg2) then goto L