RISC16 Simulator Debugging

February 23, 2016

1 Bugs

☑ In the sequential version, writting instructions in the program memory is not enough. It must first be saved in ROM, then imported back and only then can be executed.

→ You just need to press the “Assembly" button, you moron.

☑ If there are multiple comment symbols after an instruction, the whole line is ignored all together.

→ e.g. When importing `addi 1,1,0//double//comment`, the line does not appear in the program memory.

→ The problem probably comes from the comment parsing when importing the file in `MemProg::fileopen()`.

☑ When coding a label in the program memory, the said label is not recognized as an address and generates errors. The code needs to be exported and imported back again so that the label is set in the address column correctly.

→ We can only code using the ASM column, the others are read-only. However, the label needs to be in the Address column. Hence, when we input something like `loop:  beq 2,0,end`, the label `loop` stays in the same column as the opcode even though it should be considered as an address. To work around that, we can export and import back the code; the labels are then moved away from the ASM column to the Address column.

→ When using the `Assembly` function (through the dedicated button), the labels should be parsed and moved to the Address column.

→ The problem comes from `MemProg::getIns`, the function parsing the ASM instruction upon assembly. It supposes that the first token of the assembly line is always an instruction, otherwise it throws an “error : bad instruction" at the face of the poor user. We should probably first check if there is no label before the instruction.

→ Checking first for the label was a good idea. Now the label is correctly put in the address column upon assembly. However, when parsing the rest of the

code, if a label is used on a line and has not been defined before that line, it throws a warning fucking everything up ("Unknown label or format error"). Assembling twice, first to parse the label and second for the rest, does not help.

→ The label parsing needs to be a full step on its own. See the new `getLabel` loop in `MemProg`.

☑ When using a label as the signed immediate operand of `addi`, the RiSC stores the relative address of the label, not its absolute. Hence, if we want to jump with `jalr` to the address taken from a register filled with an `addi` operating with a label, it won't go to the expected place.

→ All the jumps where considered relative. However, jumps using `beq` are relative, but jumps using `jalr` are absolute. It's just a matter of selecting the good jump type uppon arguments parsing.

☑ The shifting in `lui` is wrong.

→ The program uses a rotation instead of a shift.

→ For some reason, it rotates to the right instead of the left.

→ It also conflicted with `movi`

☐ When there are too many arguments, the editor does not complain.

☐ If there is the name of an instruction in a label, it crashes.

☐ In the ASM simulator, if there are instructions after the `halt`, it keeps going instead of stopping.

☑ Jump after overflow does not work.

→ In the ASM simulator, when an overflow occurs (in an ADD operation, for example), if a label is used to point to the exception routine, it does not jump to the right place. Instead of converting the label to a relative jump, it takes the address of the label and does `PC + 1 + label_address`.

→ It was only a matter of label translation.

☑ The ASM LUI is so fucked up.

→ The shifting seems to be right, but the operand is completely different to what is expected.

→ `lui 1,0x200` should yield `r1=0x8000`.

→ The operand was indeed shifted on two different places.

2 Performance improvements

☐ When loading a ROM, the file is opened twice in a row.

→ See `Memprog::fileopen(String path)`.